# Table of Contents

1. WSO2 API Manager Documentation ................................................................. 6
   1.1 Introduction ............................................................................................... 6
      1.1.1 Overview ......................................................................................... 7
      1.1.2 About this Release ....................................................................... 7
   1.2 Quick Start Guide .................................................................................. 8
   1.3 Key Concepts .......................................................................................... 41
   1.4 Tutorials ................................................................................................ 59
      1.4.1 Getting Started ............................................................................. 60
      1.4.2 API Publishing ............................................................................ 86
         1.4.2.1 Add API Documentation ................................................... 87
            1.4.2.1.1 Add API Documentation In-line, using a URL or a File 87
            1.4.2.1.2 Add Apache Solr-Based Indexing ...................... 91
      1.4.2.2 Manage the API Lifecycle ....................................................... 93
      1.4.2.3 Publish to Multiple External API Stores .................................. 110
      1.4.2.4 Publish through Multiple API Gateways .............................. 116
      1.4.2.5 Block Subscription to an API ............................................. 120
      1.4.2.6 Enforce Throttling and Resource Access Policies ............. 126
      1.4.2.7 Change the Default Mediation Flow of API Requests .......... 131
      1.4.2.8 Map the Parameters of your Backend URLs with the API Publisher URLs .................................................... 143
      1.4.2.9 Convert a JSON Message to SOAP and SOAP to JSON .... 152
      1.4.2.10 Invoke an API using a SOAP Client .................................. 163
      1.4.2.11 Create and Publish an API from a Swagger Definition .... 168
      1.4.2.12 Create a WebSocket API ..................................................... 176
      1.4.2.13 Create a Prototyped API with an Inline Script ................ 184
      1.4.2.14 Pass a Custom Authorization Token to the Backend ....... 188
      1.4.2.15 Create and Publish a SOAP API ...................................... 197
      1.4.2.16 Disable Message Chunking .............................................. 203
      1.4.2.17 Enable API Indexing on Remote Publisher and Store Nodes 203
      1.4.2.18 Remove Specific Request Headers From Response .......... 204
      1.4.2.19 Scope Management with OAuth Scopes ............................ 205
      1.4.2.20 Generate REST API from SOAP Backend ...................... 206
   1.4.3 Developer Portal ............................................................................... 213
      1.4.3.1 Include Additional Headers in the API Console .................. 214
      1.4.3.2 Log in to the API Store using Social Media ....................... 219
      1.4.3.3 Test an API using a Testing Tool ...................................... 226
      1.4.3.4 Use the Community Features ............................................ 237
      1.4.3.5 Write a Client Application Using the SDK ..................... 241
      1.4.3.6 Obtaining User Profile Information with OpenID Connect 246
      1.4.3.7 Cleaning Up Partially Created Keys .................................. 248
      1.4.3.8 Add Custom Attributes to Applications .......................... 248
   1.5 Using WSO2 API Manager ....................................................................... 250
      1.5.1 Installation Guide ......................................................................... 251
         1.5.1.1 Installation Prerequisites ................................................... 251
         1.5.1.2 Installing the Product ....................................................... 253
            1.5.1.2.1 Installing on Linux or OS X ................................... 253
            1.5.1.2.2 Installing on Solaris ............................................... 254
            1.5.1.2.3 Installing on Windows ............................................ 255
            1.5.1.2.4 Installing as a Linux Service ................................ 257
            1.5.1.2.5 Installing as a Windows Service .............................. 259
      1.5.1.3 Running the Product .................................................................. 264
      1.5.1.4 Basic Health Checks .................................................................. 268
      1.5.2 WSO2 API Manager Tooling ......................................................... 269
         1.5.2.1 Installing the API Manager Tooling Plug-In ....................... 269
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.3 Product Administration</td>
<td>273</td>
</tr>
<tr>
<td>1.5.3.1 Upgrading from the Previous Release</td>
<td>281</td>
</tr>
<tr>
<td>1.5.3.1.1 Upgrading from the Previous when WSO2 IS is the Key Manager</td>
<td>314</td>
</tr>
<tr>
<td>1.5.3.2 Deploying WSO2 API Manager</td>
<td>316</td>
</tr>
<tr>
<td>1.5.3.2.1 Deployment Patterns</td>
<td>316</td>
</tr>
<tr>
<td>1.5.3.2.2 Deploying API Manager using Single Node Instances</td>
<td>324</td>
</tr>
<tr>
<td>1.5.3.2.3 Configuring the keyStore and client trustStore for a Selected Pattern</td>
<td>332</td>
</tr>
<tr>
<td>1.5.3.2.4 Distributed Deployment of API Manager</td>
<td>333</td>
</tr>
<tr>
<td>1.5.3.2.5 Configuring the Proxy Server and the Load Balancer</td>
<td>360</td>
</tr>
<tr>
<td>1.5.3.2.6 Installing and Configuring the Databases</td>
<td>370</td>
</tr>
<tr>
<td>1.5.3.2.7 Configuring rsync for Deployment Synchronization</td>
<td>377</td>
</tr>
<tr>
<td>1.5.3.2.8 Configuring WSO2 Identity Server as a Key Manager</td>
<td>378</td>
</tr>
<tr>
<td>1.5.3.2.9 Configuring Admin App Event Publishing for Traffic Manager HA Setup</td>
<td>388</td>
</tr>
<tr>
<td>1.5.3.3 Changing the Default API-M Databases</td>
<td>392</td>
</tr>
<tr>
<td>1.5.3.4 Administration - API-M Analytics</td>
<td>404</td>
</tr>
<tr>
<td>1.5.3.4.1 Re-indexing Existing Data</td>
<td>404</td>
</tr>
<tr>
<td>1.5.3.4.2 Adding Internationalization and Localization</td>
<td>404</td>
</tr>
<tr>
<td>1.5.3.4.3 Adding Links to Navigate Between the Store and Publisher</td>
<td>405</td>
</tr>
<tr>
<td>1.5.3.4.4 Changing the Default Transport</td>
<td>409</td>
</tr>
<tr>
<td>1.5.3.4.5 Configuring API Monetization Category Labels</td>
<td>409</td>
</tr>
<tr>
<td>1.5.3.4.6 Configuring Caching</td>
<td>409</td>
</tr>
<tr>
<td>1.5.3.4.7 Configuring Keystores in WSO2 API Manager</td>
<td>409</td>
</tr>
<tr>
<td>1.5.3.4.8 Configuring Multiple Tenants</td>
<td>409</td>
</tr>
<tr>
<td>1.5.3.4.8.1 Managing Tenants</td>
<td>425</td>
</tr>
<tr>
<td>1.5.3.4.9 Configuring Single Sign-on</td>
<td>425</td>
</tr>
<tr>
<td>1.5.3.4.9.1 Configuring Single Sign-on with SAML2</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.9.2 Configuring Single Sign-on with OpenID Connect</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.10 Configuring Transports</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.11 Configuring User Stores</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.11.1 Realm Configuration</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.11.2 Changing the RDBMS</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.11.3 Configuring Primary User Stores</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.12 Customizing the API Store</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.13 Directing the Root Context to the API Store</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.14 Enabling Access Control Support for API Publisher</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.15 Enabling CORS for APIs</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.16 Enabling Notifications</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.17 Generating SDKs</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.18 Logging</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.18.1 Application Logs</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.18.2 Setting Up Logging</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.18.3 System Logs</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.18.4 Monitoring Access Logs</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.19 Maintaining Separate Production and Sandbox Gateways</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.20 Managing Users and Roles</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.20.1 Adding User Roles</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.20.2 Adding Users</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.21 Message Tracing</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.22 Migrating the APIs and Applications to a Different Environment</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.23 Performance Tuning and Testing Results</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.23.1 Tuning Performance</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.23.2 WSO2 API-M Performance and Capacity Planning</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.24 Prevent API Suspension</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.25 Removing Unused Tokens from the Database</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.26 Revoke OAuth2 Application</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.27 Sharing Applications Between Multiple Groups</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.28 Transforming API Message Payload</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.29 Whitelisting Host Names for API Store</td>
<td>426</td>
</tr>
<tr>
<td>1.5.3.4.30 Working with Access Tokens</td>
<td>426</td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2015-2018
## 1.5.4.31 Working with Databases

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.4.31.1 Managing Datasources</td>
<td>607</td>
</tr>
<tr>
<td>1.5.4.32 Changing the Hostname</td>
<td>615</td>
</tr>
<tr>
<td>1.5.4.33 Whitelisting Hostnames for API Store</td>
<td>615</td>
</tr>
</tbody>
</table>

## 1.5.5 Working with the API Microgateway

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.5.1 Overview of the Microgateway</td>
<td>616</td>
</tr>
<tr>
<td>1.5.5.2 Microgateway Quick Start</td>
<td>618</td>
</tr>
<tr>
<td>1.5.5.2.1 API Microgateway Walkthrough</td>
<td>623</td>
</tr>
<tr>
<td>1.5.5.3 Grouping APIs with Labels</td>
<td>624</td>
</tr>
<tr>
<td>1.5.5.4 Overriding Endpoint Information</td>
<td>628</td>
</tr>
<tr>
<td>1.5.5.5 Importing Certificates to the API Microgateway Truststore</td>
<td>629</td>
</tr>
<tr>
<td>1.5.5.6 Analytics for Microgateway</td>
<td>629</td>
</tr>
<tr>
<td>1.5.5.7 Deploying the API Microgateway in Docker</td>
<td>632</td>
</tr>
<tr>
<td>1.5.5.8 Deploying the API Microgateway in Kubernetes</td>
<td>635</td>
</tr>
<tr>
<td>1.5.5.9 Working with the Microgateway in a Windows Environment</td>
<td>638</td>
</tr>
<tr>
<td>1.5.5.10 Troubleshooting the Microgateway</td>
<td>650</td>
</tr>
</tbody>
</table>

## 1.5.6 Extending the API Manager

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.6.1 Managing Workflow Extensions</td>
<td>653</td>
</tr>
<tr>
<td>1.5.6.1.1 Adding an Application Creation Workflow</td>
<td>654</td>
</tr>
<tr>
<td>1.5.6.1.2 Adding an Application Registration Workflow</td>
<td>658</td>
</tr>
<tr>
<td>1.5.6.1.3 Adding an API Subscription Workflow</td>
<td>668</td>
</tr>
<tr>
<td>1.5.6.1.4 Adding a User Signup Workflow</td>
<td>673</td>
</tr>
<tr>
<td>1.5.6.1.5 Invoking the API Manager from the BPEL Engine</td>
<td>677</td>
</tr>
<tr>
<td>1.5.6.1.6 Customizing a Workflow Extension</td>
<td>678</td>
</tr>
<tr>
<td>1.5.6.1.7 Configuring Workflows for Tenants</td>
<td>685</td>
</tr>
<tr>
<td>1.5.6.1.8 Configuring Workflows in a Cluster</td>
<td>693</td>
</tr>
<tr>
<td>1.5.6.1.9 Changing the Default User Role in Workflows</td>
<td>696</td>
</tr>
<tr>
<td>1.5.6.1.10 Cleaning Up Workflow Tasks</td>
<td>696</td>
</tr>
<tr>
<td>1.5.6.1.11 Adding an API State Change Workflow</td>
<td>697</td>
</tr>
<tr>
<td>1.5.6.2 Writing Custom Handlers</td>
<td>703</td>
</tr>
<tr>
<td>1.5.6.3 Adding Mediation Extensions</td>
<td>706</td>
</tr>
<tr>
<td>1.5.6.4 Writing Custom Grant Types</td>
<td>711</td>
</tr>
<tr>
<td>1.5.6.5 Extending Key Validation</td>
<td>711</td>
</tr>
<tr>
<td>1.5.6.6 Extending the Key Manager Interface</td>
<td>712</td>
</tr>
<tr>
<td>1.5.6.7 Adding a New API Store Theme</td>
<td>713</td>
</tr>
<tr>
<td>1.5.6.8 Extending Scope Validation</td>
<td>715</td>
</tr>
<tr>
<td>1.5.6.9 Extending the API Life Cycle</td>
<td>717</td>
</tr>
<tr>
<td>1.5.6.10 Customize the API Store and Gateway URLs for Tenants</td>
<td>723</td>
</tr>
<tr>
<td>1.5.6.11 Editing API Templates</td>
<td>728</td>
</tr>
<tr>
<td>1.5.6.12 Customizing Login Pages for API Store and API Publisher</td>
<td>728</td>
</tr>
<tr>
<td>1.5.6.13 API Gateways with Dedicated Backends</td>
<td>731</td>
</tr>
<tr>
<td>1.5.6.14 Securing OAuth Token with HMAC Validation</td>
<td>734</td>
</tr>
<tr>
<td>1.5.6.15 Customizing User SignUp in API Store</td>
<td>735</td>
</tr>
</tbody>
</table>

## 1.5.7 Working with Security

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.7.1 Passing Enduser Attributes to the Backend Using JWT</td>
<td>741</td>
</tr>
<tr>
<td>1.5.7.2 Dynamic SSL Certificate Installation</td>
<td>744</td>
</tr>
<tr>
<td>1.5.7.3 Maintaining Logins and Passwords</td>
<td>747</td>
</tr>
<tr>
<td>1.5.7.4 Saving Access Tokens in Separate Tables</td>
<td>750</td>
</tr>
<tr>
<td>1.5.7.5 Configuring WSO2 Identity Server as the Key Manager</td>
<td>752</td>
</tr>
<tr>
<td>1.5.7.6 Configuring a Third-Party Key Manager</td>
<td>753</td>
</tr>
<tr>
<td>1.5.7.7 Enabling Role-Based Access Control Using XACML</td>
<td>754</td>
</tr>
<tr>
<td>1.5.7.8 Encrypting OAuth Keys</td>
<td>760</td>
</tr>
<tr>
<td>1.5.7.9 Provisioning Out-of-Band OAuth Clients</td>
<td>768</td>
</tr>
<tr>
<td>1.5.7.10 Basic Auth</td>
<td>772</td>
</tr>
<tr>
<td>1.5.7.11 Digest Auth</td>
<td>774</td>
</tr>
<tr>
<td>1.5.7.12 Working with Encrypted Passwords</td>
<td>775</td>
</tr>
<tr>
<td>1.5.7.12.1 Encrypting Secure Endpoint Passwords</td>
<td>777</td>
</tr>
<tr>
<td>1.5.7.13 Mutual SSL between API Gateway and Backend</td>
<td>778</td>
</tr>
<tr>
<td>1.5.7.14 Securing APIs</td>
<td>780</td>
</tr>
<tr>
<td>1.5.7.14.1 Multi Factor Authentication (MFA) for Publisher and Developer Portals</td>
<td>788</td>
</tr>
<tr>
<td>1.5.7.15 Gateway Threat Protectors for API Manager</td>
<td>788</td>
</tr>
<tr>
<td>1.5.7.15.1 Regular Expression Threat Protection for API Gateway</td>
<td>789</td>
</tr>
<tr>
<td>1.5.7.15.2 JSON Threat Protection for API Gateway</td>
<td>792</td>
</tr>
<tr>
<td>1.5.7.15.3 XML Threat Protection for API Gateway</td>
<td>794</td>
</tr>
</tbody>
</table>
1.5.8 Working with Throttling .......................... 811
  1.5.8.1 Introducing Throttling Use-Cases .............. 812
  1.5.8.2 Setting Maximum Backend Throughput Limits .... 815
  1.5.8.3 Setting Throttling Limits ..................... 816
  1.5.8.4 Adding New Throttling Policies ................ 822
  1.5.8.5 Managing Throttling .......................... 830
  1.5.8.6 Enforcing Throttling to an API ................. 837
  1.5.8.7 Engaging a new Throttling Policy at Runtime .... 842
  1.5.8.8 Engaging Multiple Throttling Policies to a Single API 844
1.5.9 Working with Endpoints ......................... 846
1.5.10 Working with Analytics ......................... 851
  1.5.10.1 Configuring APIM Analytics ................. 851
  1.5.10.2 Analyzing APIM Statistics .................. 855
    1.5.10.2.1 Introducing the WSO2 API Manager Statistics Model 856
    1.5.10.2.2 Viewing API Statistics ................ 869
    1.5.10.2.3 Using Geolocation Based Statistics ....... 888
  1.5.10.3 Managing Alerts ............................ 892
    1.5.10.3.1 Alert Types .......................... 892
    1.5.10.3.2 Configuring Alerts ..................... 894
    1.5.10.3.3 Subscribing for Alerts ................. 904
    1.5.10.3.4 Viewing Alerts ....................... 906
  1.5.10.4 Integrating with Google Analytics ........... 908
  1.5.10.5 Monitoring with WSO2 Carbon Metrics ....... 911
    1.5.10.5.1 Enabling Metrics and Storage Types ........ 911
    1.5.10.5.2 Configuring Metrics Properties .......... 914
    1.5.10.5.3 Using JVM Metrics ..................... 917
  1.5.10.6 Purging Analytics Data ..................... 919
  1.5.10.7 Default Ports of WSO2 API-M Analytics ....... 921
  1.5.10.8 Enabling API-M Analytics Features in WSO2 Stream Processor 921
1.5.11 Reference Guide ................................ 922
  1.5.11.1 WSO2 API Manager Best Practices ............. 922
  1.5.11.2 Product Profiles ........................... 923
  1.5.11.3 Default Product Ports ....................... 925
  1.5.11.4 Changing the Default Ports with Offset ....... 930
  1.5.11.5 Error Handling ................................ 931
  1.5.11.6 Message Flow in the API Manager Gateway ...... 937
  1.5.11.7 Updating WSO2 API Manager .................. 937
  1.5.11.8 Accessing API Manager by Multiple Devices Simultaneously 938
1.5.12 Developer Guide ................................. 938
  1.5.12.1 Working with the Source Code ................. 939
  1.5.12.2 Java Documentation .......................... 939
  1.5.12.3 WSO2 Admin Services ........................ 939
  1.5.12.4 Product APIs ................................ 943
    1.5.12.4.1 RESTful APIs .......................... 943
    1.5.12.4.2 Token API ............................ 944
    1.5.12.4.3 Deprecated APIs ....................... 975
  1.5.12.5 Working with Audit Logs ..................... 992
  1.5.12.6 Enabling Authentication Session Persistence ... 994
  1.5.12.7 Using the Registry REST API .................. 994
1.6 FAQ .................................................... 995

Copyright © WSO2 Inc. 2015-2018
Get started with WSO2 API Manager

If you are new to using WSO2 API Manager, follow the steps below to get started:

- **Get familiar with WSO2 API Manager**
  Understand the basics of the API Manager, including the business cases it solves, its features, and its architecture.

- **Quick Start Guide**
  Download, install, and get up and running quickly.

Deep dive into WSO2 API Manager

<table>
<thead>
<tr>
<th>Tutorials</th>
<th>Using WSO2 API Manager</th>
<th>Admin Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product APIs</td>
<td>Working with Analytics</td>
<td>Reference Guide</td>
</tr>
</tbody>
</table>
Introduction

The topics in this section introduce WSO2 API Manager Server, including the business cases it solves, its features, and architecture.

- Overview
- About this Release

Overview

As an organization implements SOA, it can benefit by exposing core processes, data and services as APIs to the public. External parties can mash up these APIs in innovative ways to build new solutions. A business can increase its growth potential and partnership advancements by facilitating developments that are powered by its APIs in a simple, decentralized manner.

However, leveraging APIs in a collaborative way introduces new challenges in exercising control, establishing trust, security and regulation. As a result, proper API management is crucial.

WSO2 API Manager overcomes these challenges with a set of features for API creation, publication, lifecycle management, versioning, monetization, governance, security etc. using proven WSO2 products such as WSO2 Enterprise Integrator 6.4.0, WSO2 Identity Server 5.7.0, and WSO2 Governance Registry. In addition, it is also powered by the WSO2 Stream Processor 4.3.0 and is immediately ready for massively scalable deployments.

WSO2 API Manager is highly performant, fully open source, and is released under Apache Software License Version 2.0, one of the most business-friendly licenses available today. It provides Web interfaces for development teams to deploy and monitor APIs, and for consumers to subscribe to, discover and consume APIs through a user-friendly storefront. The API Manager also provides complete API governance and shares the same metadata repository as WSO2 Governance Registry. If your setup requires to govern more than APIs, we recommend you to use WSO2 API manager for API governance and WSO2 Governance Registry for the other artifacts. The default communication protocol of the Key Manager is Thrift.

The WSO2 API Manager is an on-going project with continuous improvements and enhancements introduced with each new release to address new business challenges and customer expectations. WSO2 invites users, developers and enthusiasts to get involved or get the assistance of our development teams at many different levels through online forums, mailing lists and support options.

About this Release

WSO2 API Manager is a platform for creating, managing, consuming and monitoring APIs. It employs proven SOA best practices to solve a wide range of API management challenges such as API provisioning, API governance, API security and API monitoring. It combines some of the most powerful and mature components of the WSO2's state-of-the-art Carbon platform to deliver a smooth and end-to-end API management experience while catering to both API publisher and API consumer requirements.

WSO2 API Manager comprises of several modules.

- API Publisher: Define new APIs and manage them.
- API Store: Browse published APIs and subscribe to them.
- API Gateway: The underlying API runtime based on WSO2 Enterprise Integrator (WSO2 EI) 6.4.0
- API Key Manager: Performs key generation and key validation functionalities.
- API Traffic Manager: Performs rate limiting of API requests

For more information on WSO2 API Manager, see the overview in the WSO2 2.6.0 documentation and go to the product page on WSO2 API Manager on the WSO2 website.

What is new in this release

The WSO2 API Manager version 2.6.0 is the successor of version 2.5.0. It contains the following new features and enhancements:

- WSO2 API-M Analytics based on WSO2 Stream Processor. From WSO2 API-M 2.6.0 onwards, WSO2 API-M Analytics is now powered by WSO2 Stream Processor 4.3.0 instead of WSO2 Data Analytics Server.
- Support to download the Microgateway analytics report.
- Support for alert threshold configurations via the WSO2 API Manager Publisher and the store.

What has changed in this release

Removed features and functionalities

- Log Analyzer Analyzing Logs with the Log Analyzer is removed from this release.

Deprecated features and functionalities

This release includes the following features/functionalities that were deprecated, which may be removed in a future release. To view previously deprecated features, see the About this Release page of the previous version.

- Gateway Manager profile Prior to WSO2 API Manager 2.1.0, the Publisher and Gateway were required to be on two different cluster domains. However, from WSO2 API
Manager 2.1.0 onwards, clustering is no longer a necessity because the Publisher can play the role of the Gateway Manager. Therefore, the Gateway Manager profile (-Dprofile=gateway-manager) has now been deprecated as it is redundant.

- **The -DSetup option**
  When proper Database Administrative (DBA) practices are followed, the systems (except analytics products) are not granted DDL (Data Definition) rights on the schema. Therefore, maintaining the -DSetup option is redundant and typically unusable. As a result, WSO2 has deprecated the -DSetup option. Note that the proper practice is for the DBA to run the DDL statements manually so that the DBA can examine and optimize any DDL statement (if necessary) based on the DBA best practices that are in place within the organization.

**WUM updates**

This section lists out the features that were updated or introduced newly to WSO2 API-M 2.6.0 via WUM updates.

<table>
<thead>
<tr>
<th>Updated or newly introduced feature</th>
<th>The date of the update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working with Observability</td>
<td>November 24, 2018</td>
</tr>
</tbody>
</table>

**Compatible WSO2 product versions**

WSO2 API-M 2.6.0 is based on WSO2 Carbon 4.4.35 and is expected to be compatible with any of the WSO2 products that are based on any Carbon 4.4.x version, except when using WSO2 Identity Server as a Key Manager, you need to specifically use WSO2 Identity Server 5.7.0 when working with WSO2 API-M 2.6.0. If you get any compatibility issues, please contact team WSO2. For information on the third-party software required with API-M 2.6.0, see Installation Prerequisites. For more information on the products in each Carbon platform release, see the Release Matrix.

**Fixed issues**

For a list of fixed issues, refer to the following links.

- WSO2 API Manager 2.6.0 - Fixed Issues in the product-apim GitHub repository.
- WSO2 API Manager 2.6.0 - Fixed Issues in the carbon-apim GitHub repository.

**Known issues**

For a list of known issues, refer to the following links.

- WSO2 API Manager 2.6.0 - Known Issues in the product-apim GitHub repository.
- WSO2 API Manager 2.6.0 - Known Issues in the analytics-apim GitHub repository.
Quick Start Guide

WSO2 API Manager is a complete solution for designing and publishing APIs, creating and managing a developer community, and for securing and routing API traffic in a scalable manner. It leverages proven components from the WSO2 platform to secure, integrate and manage APIs. In addition, it integrates with the WSO2 analytics platform and provides out of the box reports and alerts, giving you instant insights into the APIs behavior.

Let’s go through the use cases of the API Manager:

- Invoking your first API
- Understanding the API Manager concepts
- Deep diving into the API Manager
  - Creating users and roles
  - Creating an API from scratch
  - Adding API documentation
  - Adding interactive documentation
  - Versioning the API
  - Associating Scopes to API Resources
  - Publishing the API
  - Subscribing to the API
  - Invoking the API
  - Monitoring APIs and viewing statistics

Invoking your first API

Follow the steps in this section to quickly deploy a sample API, publish it, subscribe to it, and invoke it.

1. Open the API Publisher (https://<hostname>:9443/publisher) and sign in with admin/admin credentials.
2. Exit from API creation tutorial by clicking the close icon(X) on top right corner.
3. Click the Deploy Sample API button. It deploys a sample API called PizzaShackAPI into the API Manager.

This Deploy Sample API option is available only when there are no APIs in API Publisher. If you have already created a API, this option will not be available.
4. Click **PizzaShackAPI** to open it.

5. Go to the **Lifecycle** tab and note that the **State** is **PUBLISHED**. The API is already published to the API Store.
6. Sign in to the API Store (https://<hostname>:9443/store) with the admin/admin credentials and click on the PizzaShackAPI API.
7. Select the default application and an available tier, and click **Subscribe**.
8. When the subscription is successful, click View Subscriptions on the information message that appears. Click the Production Keys tab and click Generate Keys to generate an access token to invoke the API.

You have now successfully subscribed to an API. Let's invoke the API using the integrated Swagger-based API Console.

9. Click the APIs menu again and click the PizzaShackAPI to open it. When the API opens, click its API Console tab.
Expand the GET method (which retrieves the menu) and click **Try it out**.

```json
{
  "list": [],
  "price": "string",
  "description": "string",
  "name": "string",
  "image": "string"
}
```

**Response Content Type**: application/json

**Response Messages**
- **304** Not Modified. Empty body because the client has already the latest version of the requested resource.
- **406** Not Acceptable. The requested media type is not supported

Note the response for the API invocation. It returns the list of menu items.
You have deployed a sample API, published it to the API Store, subscribed to it, and invoked the API using our integrated API Console.

**Understanding the API Manager concepts**

Before we look into the API management activities in detail, let's take a look at the basic API management concepts.

Components

The API Manager consists of the following components:

- **API Publisher**: Enables API providers to publish APIs, share documentation, provision API keys and gather feedback on features, quality and usage. You access the Web interface via https://<Server Host>:9443/publisher.
- **API Store (Developer Portal)**: Enables API consumers to self register, discover and subscribe to APIs, evaluate them and interact with API Publishers. You access the Web interface via https://<Server Host>:9443/store.
- **API Gateway**: Secures, protects, manages, and scales API calls. It is a simple API proxy that intercepts API requests and applies policies such as throttling and security checks. It is also instrumental in gathering API usage statistics. The Web interface can be accessed via https://<Server Host>:9443/carbon.
- **Key Manager**: Handles all security and key-related operations. The API Gateway connects with the Key Manager to check the validity of subscriptions, OAuth tokens, and API invocations. The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway.
- **Traffic Manager**: Helps users to regulate API traffic, make APIs and applications available to consumers at different service levels and secures APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time.
- **WSO2 API Manager Analytics**: Provides a host of statistical graphs and an alerting mechanism on predetermined events.
**Users and roles**

The API manager offers three distinct community roles that are applicable to most enterprises:

- **Creator**: A creator is a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions, how it is exposed by the Gateway, etc.) and uses the API publisher to provision APIs into the API Store. The creator uses the API Store to consult ratings and feedback provided by API users. Creators can add APIs to the store but cannot manage their life cycle (e.g., make them visible to the outside world).
- **Publisher**: A publisher manages a set of APIs across the enterprise or business unit and controls the API life cycle and monetization aspects.
- **Consumer**: A consumer uses the API Store to discover APIs, see the documentation and forums, and rate/comment on the APIs. Consumers subscribe to APIs to obtain API keys.

**API lifecycle**

An API is the published interface, while the service is the implementation running in the backend. APIs have their own lifecycles that are independent of the backend services they rely on. This lifecycle is exposed in the API Publisher and is managed by the publisher role.

The following stages are available in the default API life cycle:

- **CREATED**: API metadata is added to the API Store, but it is not visible to subscribers yet, nor deployed to the API Gateway.
- **PROTOTYPED**: The API is deployed and published in the API Store as a prototype. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. Users can try out a prototyped API without subscribing to it.
- **PUBLISHED**: The API is visible in the API Store and available for subscription.
- **DEPRECATED**: The API is still deployed in the API Gateway (i.e., available at runtime to existing users) but not visible to subscribers. You can deprecate an API automatically when a new version of it is published.
- **RETIRED**: The API is unpublished from the API Gateway and deleted from the Store.
- **BLOCKED**: Access to the API is temporarily blocked. Runtime calls are blocked, and the API is not shown in the API Store anymore.
Applications

An application is primarily used to decouple the consumer from the APIs. It allows you to do the following:

- Generate and use a single key for multiple APIs.
- Subscribe multiple times to a single API with different SLA levels.

You create an application to subscribe to an API. The API Manager comes with a default application, and you can also create as many applications as you like.

Throttling tiers

Throttling tiers are associated with an API at subscription time and can be defined at an API-level, resource-level, subscription-level and application-level (per token). They define the throttling limits enforced by the API Gateway, e.g., 10 TPS (transactions per second). The final throttle limit granted to a given user on a given API is ultimately defined by the consolidated output of all throttling tiers together. The API Manager comes with three predefined tiers for each level and a special tier called Unlimited, which you can disable by editing the `<ThrottlingConfigurations>` element of the `<API-M_HOME>/repository/conf/api-manager.xml` file.

In API Manager 2.0.0 onwards, Advanced Throttling is enabled by default with following configuration in `<API-M_HOME>/repository/conf/api-manager.xml`.

```
<ThrrottlingConfigurations>
    <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
    ........
    <ThrottlingConfigurations>
```

If you are disabling Advanced Throttling in any case by setting the value of `<EnableAdvanceThrottling>` false, Advanced Throttling is disabled and basic Throttling mechanism is enabled thereafter. In such a scenario, if you want to disable the Unlimited Throttling tier of basic Throttling configurations, you need to disable it under `<TierManagement>` by setting `<EnableUnlimitedTier>` to false.

```
<TierManagement>
    <EnableUnlimitedTier>true</EnableUnlimitedTier>
</TierManagement>
```

Predefined Subscription Tiers.

<table>
<thead>
<tr>
<th>Throttling Tier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlimited</td>
<td>Allows unlimited requests</td>
</tr>
<tr>
<td>Gold</td>
<td>Allows 5000 requests per minute</td>
</tr>
<tr>
<td>Silver</td>
<td>Allows 2000 requests per minute</td>
</tr>
<tr>
<td>Bronze</td>
<td>Allows 1000 requests per minute</td>
</tr>
</tbody>
</table>

API keys

The API Manager supports two scenarios for authentication:

- An access token is used to identify and authenticate a whole application.
- An access token is used to identify the final user of an application (for example, the final user of a mobile application that is deployed on many devices).

Application access token: Application access tokens are generated by the API consumer and must be passed in the incoming API requests. The API Manager uses the OAuth2 standard to provide key management. An API key is a simple string that you pass with an HTTP header (e.g., "Authorization: Bearer NtBqXcKe1u0H1a1fQ0Q06fo61X4a,") and it works equally well for SOAP and REST calls.

Application access tokens are generated at the application level and valid for all APIs that you associate to the application. These tokens have a fixed expiration time, which is set to 60 minutes by default. You can change this to a longer time, even for several weeks. Consumers can regenerate the access token directly from the API Store. To change the default expiration time which is 60 minutes by default, you open the `<API-M_HOME>/repository/conf/identity/identity.xml` file and change the value of the element `<AccessTokenDefaultValidityPeriod>`. If you set a negative value, the token never expires. Changes to this value are applied only to the new applications that you create.
**Application user access token**: You generate access tokens on demand using the Token API. In case a token expires, you use the Token API to refresh it.

The Token API takes the following parameters to generate the access token:

- **Grant Type**
- **Username**
- **Password**
- **Scope**

To generate a new access token, you issue a Token API call with the above parameters where `grant_type=password`. The Token API then returns two tokens: an access token and a refresh token. The access token is saved in a session on the client side (the application itself does not need to manage users and passwords). On the API Gateway side, the access token is validated for each API call. When the token expires, you refresh the token by issuing a token API call with the above parameters where `grant_type=refresh_token` and passing the refresh token as a parameter.

**API resources**

An API is made up of one or more resources. Each resource handles a particular type of request and is analogous to a method (function) in a larger API. API resources accept the following optional attributes:

- **verbs**: Specifies the HTTP verbs a particular resource accepts. Allowed values are GET, POST, PUT, DELETE, PATCH, HEAD, and OPTIONS. You can give multiple values at once.
- **url-mapping**: A URL mapping defined as per the servlet specification (extension mappings, path mappings, and exact mappings).
- **Throttling tiers**: Limits the number of hits to a resource during a given period of time.
- **Auth-Type**: Specifies the Resource level authentication along the HTTP verbs. Auth-type can be None, Application, Application User, or Application & Application User.
  - None: Can access the particular API resource without any access tokens.
  - Application: An application access token is required to access the API resource.
  - Application User: A user access token is required to access the API resource.
  - Application & Application User: An application access token together with a user access token is required to access the API resource.

---

**Deep diving into the API Manager**

Let's take a look at the typical API management activities in detail:

- Creating users and roles
- Creating an API from scratch
- Adding API documentation
- Adding interactive documentation
- Versioning the API
- Associating Scopes to API Resources
- Publishing the API
- Subscribing to the API
- Invoking the API
- Monitoring APIs and viewing statistics

**Creating users and roles**

In [users and roles](https://<hostname>:9443/carbon), we introduced a set of users who are commonly found in many enterprises. Let's see how you can sign in to the Management Console as an admin and create these roles.

2. Click Add in the **Users and Roles** section under the Main menu.
The creator, publisher, and subscriber roles are available by default as shown below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>Assign Users View Users</td>
</tr>
<tr>
<td>Internal/creator</td>
<td>Rename Permissions Assign Users View Users</td>
</tr>
<tr>
<td>Internal/everyone</td>
<td>Permissions</td>
</tr>
<tr>
<td>Internal/publisher</td>
<td>Rename Permissions Assign Users View Users</td>
</tr>
<tr>
<td>Internal/subscriber</td>
<td>Rename Permissions Assign Users View Users</td>
</tr>
</tbody>
</table>

3. Click Add New Role.

4. Give the role name as creator and click Next.

5. A list of permissions opens. Select the following and click Finish.
   - All Permissions > Admin Permissions > Configure > Governance and all underlying permissions
   - All Permissions > Admin Permissions > Login
   - All Permissions > Admin Permissions > Manage > API > Create
   - All Permissions > Admin Permissions > Manage > Resources > Govern and all underlying permissions
6. Similarly, create the publisher role with the following permissions.
   - All Permissions > Admin Permissions > Login
   - All Permissions > Admin Permissions > Manage > API > Publish

7. Note that the API Manager comes with the subscriber role available by default. It has the following permissions:
   - All Permissions > Admin Permissions > Login
   - All Permissions > Admin Permissions > Manage > API > Subscribe

8. The roles you added (creator and publisher) are now displayed under Roles.
For more details Add roles and permission assign to roles, see Adding User Roles.

9. Click Add in the Users and Roles section under the Main menu.

10. Click Add New User.

11. Give the username/password and click Next. For example, let's create a new user by the name apipublisher.
Select the role you want to assign to the user (e.g., publisher) and click Finish.

Similarly, create a new user by the name apicreator and assign the creator role.

Creating an API from scratch

Let's create an API from scratch.

1. Sign in to the API Publisher (https://<hostname>:9443/publisher) as apicreator.
2. In the API menu, click Add New API.
3. Select the option to design a new API and click **Start Creating**.

   Let's get started!

Add New API

- **I Have an Existing API**
  Use an existing APIs endpoint or the API Swagger definition to create an API.

- **I Have a SOAP Endpoint**
  Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

- **Design a New REST API**
  Design and prototype a new REST API.

Start Creating

- **Design a New Websocket API**
  Design and prototype a new Websocket API.

4. Give the information in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>PhoneVerification</td>
</tr>
<tr>
<td>Context</td>
<td>/phoneverify</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Access Control</td>
<td>All</td>
</tr>
<tr>
<td>Visibility on Store</td>
<td>Public</td>
</tr>
<tr>
<td>API Definition</td>
<td>• URL pattern: CheckPhoneNumber</td>
</tr>
<tr>
<td></td>
<td>Note that this URL Pattern is the name of one of the resources that we are going to invoke from the backend service.</td>
</tr>
<tr>
<td></td>
<td>• Request types: GET, POST</td>
</tr>
</tbody>
</table>

Click **Add** and then click **Next: Implement >** to move on to the next page.
4. Select the **Managed API** option.

PhoneVerification: /phoneverify/1.0.0

5. For instructions on how to Implement Prototyped API, see [Create a Prototyped API with an Inline Script](#).

Give the following information and click **Next:Manage** once you are done.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP/REST Endpoint</td>
</tr>
<tr>
<td>Production endpoint</td>
<td>Endpoint is <a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a>. To verify the URL, click the <strong>Test</strong> button next to it. In this example, we use a phone validation service exposed by the Cdyne services provider. This service has SOAP and REST interfaces. <strong>This sample service has two operations:</strong> <a href="#">CheckPhoneNumber</a> and <a href="#">CheckPhoneNumbers</a>.</td>
</tr>
<tr>
<td>Sandbox endpoint</td>
<td>Endpoint is <a href="http://ws.cdyne.com/phoneverify/phoneverify.asmx">http://ws.cdyne.com/phoneverify/phoneverify.asmx</a>. To verify the URL, click the <strong>Test</strong> button next to it.</td>
</tr>
</tbody>
</table>

**Endpoint Type** is selected to specify whether the endpoint is based on a URI template or based on a URI template or an address specified in the "To" header.
Provide the following information in the Manage tab. Leave default values for the rest of the parameters in the UI.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscription Tiers</td>
<td>&lt;Select all available tiers&gt;</td>
<td>The API can be available for subscription at different levels of service. They allow you to limit the number of successful hits to an API during a given period of time.</td>
</tr>
</tbody>
</table>
Once you are done, click **Save**.

**Adding API documentation**

1. In the **APIs** menu, click the thumbnail of the API to open it.
2. Click on the API's **Docs** tab and click **Add New Document**.

3. The document options appear. Note that you can create documentation inline, via a URL, or as a file. For inline documentation, you can edit the content directly from the API publisher interface. You get several documents types:
   - How To
   - Samples and SDK
   - Public forum / Support forum (external link only)
   - API message formats
   - Other

4. Create a 'How To' named **PhoneVerification**, specifying in-line content as the source and optionally entering a summary. When you have finished, click **Add Document**.
5. Once the document is added, click **Edit Content** to open an embedded editor.

6. Enter your API's documentation and click **Save and Close**.
Adding interactive documentation

WSO2 API Manager has an integrated Swagger UI, which is part of the Swagger project.

Swagger is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe services in the same way that interfaces describe lower-level programming code.

The Swagger UI is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generates documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation and more discoverability. The Swagger UI has YAML code, and its UI facilitates easier code indentation, provides keyword highlighting, and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger UI and it has provides the facility to download your API definition as YAML or JSON file. Go to the Swagger 2.0 specification for more information.

1. Open the API Publisher (https://<hostname>:9443/publisher) and sign in as apicreator.
2. Click the Edit icon for the PhoneVerification API. This opens the API in its edit mode.
3. Click the **Edit Source** button under the **API Definition** section.

4. The API definition as a YAML code opens in a separate page. Expand its GET method, add the following parameters and click **Apply Changes**.
parameters:
- in: query
  name: PhoneNumber
  description: Give the phone number to be validated
  type: string
  required: true
- in: query
  name: LicenseKey
  description: Give the license key as 0 for testing purpose
  type: string
  required: true
5. Back in the API Publisher, note that the changes you did appear in the API Console's UI. You can add more parameters and edit the summary/descriptions using the API Publisher UI as well. Once done, click **Save**.

![API Definition](image)

**Versioning the API**

Let's create a new version of this API.

1. Sign in to the API Publisher as `apicreator` if you are not logged in already.
2. Click the `PhoneVerification` API to open it and then click **Create New Version**.
2. Give a new version number (e.g., 2.0.0) and click **Done**.

3. **PhoneVerification - 1.0.0**

Create New Version

- **New Version**: 2.0.0
- **Make this the Default Version**: unchecked

**Info!**

No default version defined for the current API

4. Note that the new version of the API is created in the API Publisher.

**Associating Scopes to API Resources**

Different API resources can be associated with different user roles. For an example, consider the following resources and the operations:
In order to map a scope to an API resource, the following should be done:

a. API creator should first create scopes, by clicking on Add Scopes in the Manage tab.

![Add Scopes](image)

b. Fill the scope related information in the dialog that pops up. Note that Scope Key and Roles are the most important attributes. Click on Add Scope button on the right hand side bottom.

![Define Scope](image)

The roles added here are validated against the user store to check if they exist. However, this can be overridden so that the roles are not checked in the user store. For thus purpose, set the Java system property `disableRoleValidationAtScopeCreation=true` at the server startup:

i. Open `<API-M_HOME>/bin/wso2server.(sh|bat)` file.
ii. Add `-DdisableRoleValidationAtScopeCreation=true` at the end of the file.
iii. Restart the server.

c. Now the scope with key 'item_view' is added with roles manager and agent. To associate this scope with the get operation on the /time resource, click on the +Scope sign on the right hand side of the resource.

d. From the drop down menu that appears, select the scope name and click on the tick sign to the right. Now the scope will be associated with the GET operation on the resource /item.

See Scope Management with OAuth Scopes for an in depth example.

**Publishing the API**

1. Sign in to the API Publisher as the `api_publisher` user that you created earlier in this guide, and click the `PhoneVerification API`'s version 2.0.0.
2. The API opens. Go to its Lifecycle tab and click Publish.

PhoneVerification - 2.0.0

The check boxes mean the following:
- **Require re-subscription when publish the API**: Invalidates current user subscriptions, forcing users to subscribe again.
- **Deprecate old versions after publish the API**: If selected, any prior versions of the API that are published will be set to the DEPRECATED state automatically.

3. Go to the API Store (https://<hostname>:9443/store) using your browser and note that the PhoneVerification 2.0.0 API is visible under the APIs menu.
Subscribing to the API

1. Go to the API Store (https://<hostname>:9443/store) and create an account using the Sign-up link.

Users who sign-up through the API Store are assigned the subscriber role by default. Therefore, you do not need to specify the role through the management console to be able to subscribe to an API.

2. Fill the details in the Sign Up form appears and click Sign Up.

Users who registered with the API Store Signup can be view by login to the Management Console (https://localhost:9443/carbon) and accessing Users and Roles > Users > List.
After signing up, sign in to the API Store and click the PhoneVerification 2.0.0 API that you published earlier.

3. Note that you can now see the subscription options. Select the default application and the Bronze tier. Click Subscribe.

5. Once the subscription is successful, click View Subscriptions in the information message that appears to review your subscriptions.
6. Click the **Production Keys** tab of the application and then click **Generate Keys** to generate an access token that you use later to invoke the API. If you have already generated keys before, click **Re-generate**.

![Production Keys Tab](image)

**Tip**: You can set a token validity period in the given text box. By default, it is set to one hour. If you set a minus value (e.g., -1), the token will never expire.

You are now successfully subscribed to an API. Let's invoke it.

**Invoking the API**

1. Click the **APIs** menu in the API Store and then click on the API that you want to invoke. When the API opens, go to its **API Console** tab.
2. Expand the GET method of the resource `CheckPhoneNumber`. Note the parameters that you added when creating the interactive documentation now appear with their descriptions so that as a subscriber, you know how to invoke this API.

3. Give sample values for the `PhoneNumber` and `LicenseKey` and click Try it out to invoke the API.
3. Note the response for the API invocation. Since we used a valid phone number in this example, the response is valid.

Response Body

```xml
<?xml version="1.0" encoding="utf-8"?>
  <Company>Call Free</Company>
  <Valid>true</Valid>
  <Use_assigned_to_a_code_holder_for_normal_use.</Use>
  <State>VF</State>
  <RC />
  <CN />
  <OriginalNumber>18006785432</OriginalNumber>
  <CleanNumber>8006785432</CleanNumber>
  <SwitchName />
  <SwitchType />
  <Country>United States</Country>
</PhoneReturn>
```

Response Code

200

Response Headers

```json
{
  "pragma": "no-cache",
  "content-type": "text/xml; charset=utf-8",
  "cache-control": "no-cache",
  "expire": "-1"
}
```

You have invoked an API using the API Console.

**Troubleshooting**

When using the API Console, the web browser sends an HTTPS request to the Gateway. If the certificate in the Gateway is not CA signed, the browser will not accept it. Therefore, you may get the following error.

```
ERROR - SourceHandler I/O error: Received fatal alert: unknown_ca
javax.net.ssl.SSLException: Received fatal alert: unknown_ca
```

As a workaround, you can access the Gateway URL on a new browser tab and trust the certificate from the browser.

**Monitoring APIs and viewing statistics**
Both the API publisher and store provide several statistical dashboards.

**API Publisher statistics**

**Statistics**

**APIs**
- Published APIs Over Time
- API Usage
- API Response Times
- API Last Access Times
- Usage by Resource Path
- Usage by Destination
- API Usage Comparison
- API Throttled Requests
- Faulty Invocations
- API Latency Time
- API Usage Across Geo Locations
- API Usage Across User Agent

**Applications**
- App Throttled Requests
- Applications Created Over Time

**Subscriptions**
- API Subscriptions
- Developer Signups Over Time
- Subscriptions Created Over Time

**API Store statistics**

![WSO2 API Store Analytics](WSO2_API_Manager_WSOS2.png)
The steps below explain how to configure WSO2 API Manager Analytics with the API Manager. The statistics in these dashboards are based on data from WSO2 APIM Analytics which is similar to WSO2 Stream Processor (WSO2 SP).

Let's do the configurations first.

Before you begin,

1. Download the WSO2 APIM Analytics distribution by clicking ANALYTICS in the WSO2 API Management page. It is best to download and extract it to the same directory to which you downloaded WSO2 API Manager.
2. If you have the API Manager server running, stop the server.
3. If you are running on Windows, download the snappy-java_1.1.1.7.jar from here and copy the JAR file to the <ANALYTICS_HOME>\repository\components\lib directory.

1. To enable Analytics, open the <API-M_HOME>/repository/conf/api-manager.xml file and set the Enabled property under <Analytics> to true as shown below. Save this change.

```
<Enabled>true</Enabled>
```

2. Start the WSO2 APIM Analytics server, and then start the WSO2 API Manager server.

To avoid connection errors during API Manager startup, start WSO2 APIM Analytics before WSO2 API Manager.

- a. On Windows: wso2server.bat --run
- b. On Linux/Mac OS: sh wso2server.sh

By default, WSO2 API Manager has a port offset of 0 (no port offset) and WSO2 API Manager Analytics has an offset of 1.

3. Invoke several APIs to generate some statistical data and wait a few seconds.
4. Connect to the API Publisher as a creator and click one of the statistical dashboards available in the Statistics menu. For example,

**Overall API Subscriptions (Across All Versions)**

![Pie chart showing API subscriptions]

The Statistics menu is available for API creators and shows statistics of all APIs. Additionally, API creators can also see the following:

- Statistics of the APIs created by them by selecting the My APIs option in the drop down menu above each table or graph.
- The subscriptions of each API by clicking Manage Subscriptions.
- The alerts that can be configured for their APIs by clicking Manage Alert Types.

This concludes the API Manager quick start. You have set up the API Manager and gone through the basic use cases of the product. For more advanced use cases, see the Tutorials, Using WSO2 API Manager and Admin Guide of the API Manager documentation.
Key Concepts

Let's take a look at some concepts and terminology that you need to know in order to follow the use cases.

**API Manager components**

A component is made up of one or more OSGi bundles. A bundle is the modularization unit in OSGi, similar to a JAR file in Java. The component-based architecture of all WSO2 products gives developers flexibility to remove or add features with minimum dependencies.

The API Manager consists of the following high-level components as illustrated in the diagram and listed below:

**API Publisher**

API development is usually done by someone who understands the technical aspects of the API, interfaces, documentation, versions etc., while API management is typically carried out by someone who understands the business aspects of the APIs. In most business environments, API development is a responsibility that is distinct from API publication and management.

WSO2 API Manager provides a simple Web interface called WSO2 API Publisher for API development and management. It is a structured GUI designed for API creators to develop, document, scale and version APIs, while also facilitating more API management-related tasks such as publishing API, monetization, analyzing statistics, and promoting.

The API Publisher URL is https://<YourHostName>:9443/publisher and it is accessible on HTTPS only. The default credentials are admin/admin.
The diagram below shows the common lifecycle activities of an API developer/manager:

API Store (Developer Portal)

The API Store Web application provides a collaborative interface for API publishers to host and advertise their APIs and for API consumers to self register, discover, evaluate, subscribe to and use secured, protected, authenticated APIs.

The API Store URL is https://<YourHostName>:9443/store and it is accessible on HTTPS only. The default credentials are admin/admin.

The diagram below shows common API consumer lifecycle activities:

API Gateway

A runtime, backend component (an API proxy) developed using WSO2 ESB. API Gateway secures, protects, manages, and scales API calls. It intercepts API requests, applies policies such as throttling and security using handlers, and manages API statistics. Upon validation of a policy, the Gateway passes web service calls to the actual backend. If the service call is a token request, the Gateway passes it directly to the Key Manager.

When WSO2 API Manager is running, you can access the Gateway using the following URL: https://localhost:9443/carbon. You integrate a monitoring and
analytics component to the API Manager by configuring WSO2 API Manager Analytics. This component provides reports, statistics and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and, carry out detailed analysis of the logs.

Although the API Gateway contains ESB features, it is recommended not to use it for ESB-specific tasks. Use it only for Gateway functionality related to API invocations. For example, if you want to call external services like SAP, use a separate ESB cluster for that purpose.

Key Manager

Manages all clients, security and access token-related operations. The Gateway connects with the Key Manager to check the validity of OAuth tokens, subscriptions and API invocations. When a subscriber creates an application and generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth client and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database.

The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway. All tokens used for validation are based on the OAuth 2.0.0 protocol. Secure authorization of APIs is provided by the OAuth 2.0 standard for key management. The API Gateway supports API authentication with OAuth 2.0, and enables IT organizations to enforce rate limits and throttling policies.

The Key Manager properly decouples the operations for creating OAuth applications and validating access tokens so that you can even plug in a third party-authorization server for key validations.

You can avoid making the Gateway connect with the Key Manager every time it receives an API invocation call, by enabling API Gateway caching. When caching is not enabled, a verification call happens every time the Gateway receives an API invocation call. For this verification, the Gateway passes an access token, the API, and API version to the Key Manager. Communication between the API Gateway and the Key Manager happens in either of the following ways:

- Through a Web service call
- Through a Thrift call (Thrift is the default communication protocol and is much faster than SOAP over HTTP)

If your setup has a cluster of multiple Key Manager nodes that are fronted by a load balancer that does not support Thrift, change the key management protocol from Thrift to WSClient using the `<KeyValidatorClientType>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file. Thrift uses TCP load balancing.

In a typical production environment, you can configure one of the following setups:

- Configure a WSO2 API Manager instance as the Key Manager in a separate server. See [Product Profiles](#).
- Configure an instance of WSO2 Identity Server as the Key Manager. See [Configuring WSO2 Identity Server as the Key Manager](#).
- Configure a third-party authorization server for key validations and an API Manager instance for the rest of the key management operations. See [Configuring a Third-Party Key Manager](#).

Traffic Manager
The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests. For more information, see Working with Throttling.

Analytics

Additionally, monitoring and analytics are provided by the analytics component, WSO2 API Manager Analytics. This component provides a host of statistical graphs and an alerting mechanism on pre-determined events. For more information, see Working with Analytics.

Handlers

When an API is created, a file with its synapse configuration is added to the API Gateway. You can find it in the `<API-M_HOME>/repository/deployer` folder. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration. You find the default handlers in any API's Synapse definition.

For a detailed description of handlers and how to write a custom handler, see Writing Custom Handlers.
Users and roles

The API Manager offers four distinct community roles that are applicable to most enterprises:

- **Admin**: the API management provider who hosts and manages the API Gateway. S/he is responsible for creating user roles in the system, assigning them roles, managing databases, security etc. The Admin role is available by default with the credentials admin/admin.
- **Creator**: a creator is a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions etc.) and uses the API publisher to provision APIs into the API store. The creator uses the API Store to consult ratings and feedback provided by API users. Creator can add APIs to the store but cannot manage their lifecycle.
- **Publisher**: a publisher manages a set of APIs across the enterprise or business unit and controls the API lifecycle, subscriptions and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.
- **Subscriber**: a subscriber uses the API store to discover APIs, read the documentation and forums, rate/comment on the APIs, subscribes to APIs, obtain access tokens and invoke the APIs.

Tip: See Managing Users and Roles for more information.

API lifecycle

An API is the published interface, while the service is the implementation running in the backend. APIs have their own lifecycles that are independent to the backend services they rely on. This lifecycle is exposed in the API publisher web interface and is managed by the API publisher role.

The following stages are available in the default API lifecycle:

- **CREATED**: API metadata is added to the API Store, but it is not deployed in the API Gateway and therefore, is not visible to subscribers in the API Store.
- **PROTOTYPED**: The API is deployed and published in the API Store as a prototype. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. Users can invoke the API without a subscription.
Applications

An application is a logical collection of APIs. An application is primarily used to decouple the consumer from the APIs. It allows you to:

- Generate and use a single key for multiple APIs
- Subscribe multiple times to a single API with different tiers/Service Level Agreement (SLA) levels

You subscribe to APIs through an application. Applications are available at different SLA levels and have application-level throttling tiers engaged in them. A throttling tier determines the maximum number of calls you can make to an API during a given period of time.

The API Manager comes with a pre-created default application, which allows unlimited access by default. You can also create your own applications.

Callback URL

A callback URL is a URL that sends a callback to a specific server or program soon after your application request is sent. The callback URL is specified when generating or re-generating production or sandbox keys for an application. You can either provide a single application callback URL or a RegEx pattern as the callback URL. You use a RegEx pattern as the callback URL when you need to specify multiple callback URLs for an application.

For example, consider a situation where you have two (2) service providers that need to use the same application that have the following callback URLs:

- https://mytestapp.com/callback
- https://testapp:8000/callback

In this instance, your callback URL should have the following RegEx pattern:


You can configure any RegEx pattern to match the callback URLs that you need to register with the application. However, it is mandatory to have the prefix `regexp=` before the pattern.

Access tokens

An access token is a simple string that is passed as an HTTP header of a request. For example, "Authorization: Bearer NkBqKxKoKE1uOH1afQ0DFrfo6IX4a." Access tokens authenticate API users and applications, and ensure better security (e.g., prevent certain types of DoS attacks. Note that DoS attacks made to the key manager with random access tokens can not be prevented. DoS attacks with the same fake access token can affect the Gateway as well). If a token that is passed with a request is invalid, the request is discarded at the first stage of processing. Access tokens work equally well for SOAP and REST calls.

For more information on different types of access tokens and how to generate them, see Working with Access Tokens.
**Throttling tiers**

Throttling allows you to limit the number of successful hits to an API during a given period of time, typically in cases such as the following:

- To protect your APIs from common types of security attacks such as certain types of Denial of Service (DoS) attacks
- To regulate traffic according to infrastructure availability
- To make an API, application, or a resource available to a consumer at different levels of service, usually for monetization purposes

![Throttling tiers diagram](image)

You can define throttling at the API, application, and resource levels. The final request limit granted to a given user on a given API is ultimately defined by the consolidated output of all throttling tiers together. For more information about throttling, see [Setting Throttling Limits](#).

---

**Multi tenanted API management**

A tenant in WSO2 API Manager is a separate business level entity, such as an organizational unit or a department. Multi tenancy enables such organizational units/departments to share the same API Manager deployment and the respective resources, but function individually with an isolated view of the deployment. A tenant does not need to be aware of the other tenants in the system as by design the Multitenancy creates an isolated space for each tenant, although they are sharing the same deployment.

See the section [Configuring Multiple Tenants](#) for information on how to create tenants.

In WSO2 API Manager deployment, API Visibility and Subscription Availability are the two main applications of tenancy.

**API visibility**

API Manager allows users to control API visibility and subscription availability. API visibility can be one of the following options:

- Public
- Restricted by roles
- Visible to my domain

**Subscription availability**

Subscription availability has three options. Those options are as follows:

- Available to current Tenant Only
- Available to All the Tenants
- Available to Specific Tenants
See the next section on API Visibility and Subscription for more details on each category.

Additionally, it's possible to configure a secondary userstore per tenant as well. For more information, see Configuring Secondary Userstores.

**API visibility and subscription Availability**

**API visibility**

Visibility settings prevent certain user roles from viewing and modifying APIs created by another user role.

- **Public**: The API is visible to all users who are registered and anonymous (who use APIs without login to the store, for example testing and demonstration), and can be advertised in multiple stores (central and non-WSO2 stores).
- **Restricted by Roles**: The API is visible to its tenant domain and only to the user roles that you specify. You should provide the roles separated by commas in the UI or as a cURL parameter when creating or editing the API.
- **Visible to my domain**: The API is visible to all users who are registered to the API's tenant domain. This option is available only in a multi-tenanted environment. It's not applicable when there is only one active tenant (super tenant) in the system.

Given below is how visibility levels work for users in different roles:

- The API Creator and Publisher roles can see all APIs in their tenant store even if you restrict access to them. This is because those roles have permission to view and edit all APIs in the API Publisher, and therefore, does not have to be restricted in the Store.
- Anonymous users can only see APIs that have the visibility set as Public.
- Registered users can see:
  - public APIs of all tenant domains.
  - all APIs in the registered user's tenant domain as long as the API is not restricted to a role that the user is assigned to.

By default, an API has public visibility. You can set the API visibility in the Design tab of the API Publisher at the time the API is created or updated.

When using the REST API directly, these visibility options are available as public, private and restricted.

<table>
<thead>
<tr>
<th>API visibility level specified in the UI</th>
<th>API visibility level specified in the REST API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>public i.e. &quot;visibility&quot;: &quot;PUBLIC&quot;</td>
</tr>
<tr>
<td>Visible to my domain</td>
<td>private i.e. &quot;visibility&quot;: &quot;PRIVATE&quot;</td>
</tr>
<tr>
<td>Restricted by roles</td>
<td>restricted i.e. &quot;visibility&quot;: &quot;RESTRICTED&quot;, visibleRoles: [&quot;role1&quot;,&quot;role2&quot;,&quot;role3&quot;]</td>
</tr>
</tbody>
</table>
Subscription availability

The subscription availability option has three values as follows. You can set subscription availability to an API through the API Publisher's Manage tab.

- **Available to current tenant only**: only users in the current organization/tenant domain can subscribe to the API.
- **Available to all the tenants**: users of all organizations/tenant domains can subscribe to the API.
- **Available to specific tenants**: users of the organizations/tenant domains that you specify, as well as the current tenant domain, can subscribe to the API.

Subscription is only available to the current tenant in the following instances:

- When there is only one tenant in your system.
- Even if there are multiple tenants in your system, when you select **Visible to my domain** or **Restricted by roles** as the API's visibility in the previous step.

The diagram below depicts the relationship between the API's visibility and subscription availability:

Refer the article [Multi Tenant API Management with WSO2 API Manager](#) for examples and real world usage of the above concepts.

**API documentation visibility**
By default, any document associated with an API has the same visibility level of the API. That is, if the API is public, its documentation is also visible to all users (registered and anonymous). To enable other visibility levels to the documentation, go to the `<API-M_HOME>/repository/conf/api-manager.xml` file, uncomment and set the following element to true:

```xml
<APIPublisher>
    ....
    <EnableAPIDocVisibilityLevels>true</EnableAPIDocVisibilityLevels>
</APIPublisher>
```

Then, log in to the API Publisher, click the `Docs` tab of an API, and click `Add New Document` to see a new drop-down list added to select visibility from:

You set visibility in the following ways:

- **Same as API visibility**: Visible to the same user roles who can see the API. For example, if the API's visibility is public, its documentation is visible to all users.
- **Visible to my domain**: Visible to all registered users in the API's tenant domain.
- **Private**: Visible only to the users who have permission to log in to the API Publisher web interface and create and/or publish APIs to the API Store.

API resources

An API is made up of one or more resources, each of which handles a particular type of request. A resource has a set of methods that operate on it. The methods is analogous to a method or a function, and a resource is analogous to an object instance or a class in an object-oriented programming language. There are a few standard methods defined for a resource (corresponding to the standard HTTP GET, POST, PUT and DELETE methods.)

The diagram below shows a resource by the name `CheckPhoneNumber` added with four HTTP methods.
When you add resources to an API, you define a URL pattern and HTTP methods. A resource can also have a list of OAuth scopes.

<table>
<thead>
<tr>
<th>URL Pattern</th>
<th>A URL pattern can be one of the following types:</th>
</tr>
</thead>
</table>
| GET /CheckPhoneNumber | - As a url-mapping. E.g., /state/town/*  
- As a uri-template. E.g., /{state}/{town} |

The terms url-mapping and uri-template come from synapse configuration language. When an API is published in the API Publisher, a corresponding XML definition is created in the API Gateway. This XML file has a dedicated section for defining resources. See examples below:

```
<resource methods="POST GET"
url-mapping="/state/town/*">
<resource methods="POST GET"
uri-template="/\{state\}/\{town\}">
```

url-mapping performs a one-to-one mapping with the request URL, whereas the uri-template performs a pattern matching.

Parametrizing the URL allows the API Manager to map the incoming requests to the defined resource templates based on the message content and request URI. Once a uri-template is matched, the parameters in the template are populated appropriately. As per the above example, a request made to `http://gatwa_host:gateway_port/api/v1/texas/houston` sets the value of state to texas and the value of town to houston. You can use these parameters within the synapse configuration for various purposes and gain access to these property values through the `uri.var.province` and `uri.var.district` properties. For more information on how to use these properties, see Introduction to REST API and the HTTP Endpoint of the WSO2 ESB documentation.


Once a request is accepted by a resource, it will be mediated through an in-sequence. Any response from the backend is handled through the out-sequence. Fault sequences are used to mediate errors that might occur in either sequence. The diagram below shows the flow of a request sent to an API resource.
The default in-sequence, out-sequence and fault sequences are generated when the API is published.

**HTTP methods**

HTTP methods specify the desired action to be performed on an API's resource. You can select multiple methods from **GET, POST, PUT, DELETE, PATCH, HEAD and OPTIONS**. A method has attributes such as an OAuth scope, authentication type, response content type, parameters etc. as the diagram below shows:

The main attributes of a method are described below:
OAuth scopes

You can define a list of OAuth scopes to an API's resource and assign one of them to each HTTP method.

Authentication type

The authentication type can be one of the following:

- **None**: No authentication is applied and the API Gateway skips the authentication process.
- **Application**: Authentication is done by the application. The resource accepts application access tokens.
- **Application User**: Authentication is done by the application user. The resource accepts user access tokens.
- **Application and Application User**: Both application and application user level authentication is applied. Note that if you select this option in the UI, it appears as Any in the API Manager's internal data storage and data representation, and Any will appear in the response messages as well.

*Note* that for the resources that have HTTP verbs (GET, POST etc.) requiring authentication (i.e., Auth Type is not NONE), set None as the Auth type of OPTIONS. This is to support CORS (Cross Origin Resource Sharing) between the API Store and Gateway. (The above screenshot shows this).

The auth type is cached in the API Manager for better performance. If you change the auth type through the UI, it takes about 15 minutes to refresh the cache. During that time, the server returns the old auth type from the cache. If you want the changes to be reflected immediately, please restart the server after changing the auth type.

Response content type

You can use this attribute to document in what type the backend sends the response back to the API Manager. Note that this attribute doesn't do any message type conversion, but used simply as a way of letting the user know what type the response will be.

Parameters

Parameters of an HTTP method are analogous to arguments of a function in an object-oriented programming language. A resource's parameters are cached in the resource cache at the API Gateway.

### Cross-origin resource sharing

**Cross-origin resource sharing (CORS)** is a mechanism that allows restricted resources (e.g., fonts, JavaScript) of a Web page to be requested from another domain outside the domain from which the resource originated.

The Swagger API Console that is integrated in the API Manager runs as a JavaScript client in the API Store and makes calls from the Store to the API Gateway. Therefore, if you have the API Store and Gateway running on different ports, enable CORS between them.

The CORS configuration is in the `<APIM_HOME>/repository/conf/api-manager.xml` file. Given below is a sample code:

```xml
<CORSConfiguration>
  <Enabled>true</Enabled>
  <Access-Control-Allow-Methods>GET,PUT,POST,DELETE,PATCH,OPTIONS</Access-Control-Allow-Methods>
  <Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction</Access-Control-Allow-Headers>
  <Access-Control-Allow-Credentials>false</Access-Control-Allow-Credentials>
</CORSConfiguration>
```

The elements are described below:

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
<th>Sample values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access-Control-Allow-Origin</td>
<td>Determines whether a resource can be shared with the resource of a given origin. The API Gateway validates the origin request header value against the list of origins defined under the Access Control Allow Origins configuration(this can be All Allow Origins or a specific value like localhost). If the host is in the allowed origin list, it will be set as the Access-Control-Allow-Origin response header in the response.</td>
<td>All Allow Origins(*), localhost</td>
</tr>
</tbody>
</table>
Access-Control-Allow-Headers

Determines, as part of the response to a preflight request (a request that checks to see if the CORS protocol is understood), which header field names can be used during the actual request. The gateway will set the header values defined under Access Control Allow Headers configurations.

`authorization, Access-Control-Allow-Origin, Content-type, SOAPAction`

Access-Control-Allow-Methods

This header specifies the method(s) allowed when accessing the resource in response to a preflight request. Required methods can be defined under the Access Control Allow Method configuration.

`GET, PUT, POST, DELETE, PATCH, OPTIONS`

Access-Control-Allow-Credentials

Determines whether or not the response to the request can be exposed to the page. It can be exposed when the header value is true. The header value can be set to true/false by enabling/disabling the Access Control Allow Credentials configuration.

`true, false`

If you try to invoke an API with inline endpoints, you add the CORS Handler in the `<handlers>` section of the API's configuration as follows. You can find the API's configuration in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder. Change your code according to the sample given here.

```xml
<handlers>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler"/>
</handlers>
```

OAuth scopes

Scopes enable fine-grained access control to API resources based on user roles. You define scopes to an API's resources. When a user invokes the API, his/her OAuth 2 bearer token cannot grant access to any API resource beyond its associated scopes.

OAuth provides a method for clients to access a protected resource on behalf of a resource owner. OAuth 2 bearer token is a security token that any party in possession of it can use the token for authentication. Refer [OAuth 2.0 Specification of Bearer Token Usage](#) for more information.

How scopes work

To illustrate the functionality of scopes, assume you have the following scopes attached to resources of an API:

```
Scope: news_read
Roles: employee and manager roles

Scope: news_write
Roles: manager role only
```

API

Resource: /read (GET)

Resource: /write (POST)

Assume there are two users Tom and John. Tom is assigned the employee role. John is assigned both employee and manager roles.

Tom requests a token through the Token API as `grant_type=password&username=tom&password=xxxx&scope=news_read news_write`. However, as Tom is not in the manager role, he will only be granted a token bearing the news_read scope. The response from the Token API will be similar to the following:
Next, John requests a token as grant_type=password&username=john&password=john123&scope=news_read news_write. As John has both roles assigned, the token will bear both the requested scopes and the response will be similar to the following:

```
"scope":"news_read news_write", "token_type":"bearer", "expires_in":3299,
"refresh_token":"4ca244f321bd555bd3d55df39315",
"access_token":"42a377a0101877d1d9e29c5f30857e"
```

This means that Tom can only access the GET operation of the API while John can access both as he is assigned to both the employee and manager roles. If Tom tries to access the POST operation, there will be an HTTP 403 Forbidden error as follows:

```
<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
  <ams:code>900910</ams:code>
  <ams:message>The access token does not allow you to access the requested resource</ams:message>
  <ams:description>Access failure for API: /orgnews, version: 1.0.0 with key: eb51eff0b4d85cdalebl312c5b6a3b8</ams:description>
</ams:fault>
```

**Applying a scope**

You apply scopes to an API resource at the time the API is created or modified. In the API Publisher, click the API > Add menu (to add a new API) or the Edit link next to an existing API. Then, navigate to the Manage tab and scroll down to see the Add Scopes button under Resources.

On the screen that appears, enter a scope key, scope name and optionally, allowed roles and a description. Click Add Scope.
Scope Key | A unique key for identifying the scope. Typically, it is prefixed by part of the API's name for uniqueness, but is not necessarily reader-friendly.

Scope Name | A human-readable name for the scope. It typically says what the scope does.

Roles | The user role(s) that are allowed to obtain a token against this scope. E.g., manager, employee.

Note that the role name is case sensitive in the DBMSs that are case sensitive, such as PostgreSQL.

When the role you specify is in a secondary user store, you have to give the role as <userstore name>/<role name>.

To apply the scope, you add the scope to a resource, save and publish the API.

Tip: When you generate access tokens for applications with APIs protected by scope/s in the API Store, a Scopes drop down list is displayed in the Production Keys tab of the application, where you can select the scope/s after the token is generated.

Generating Access Tokens
The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://<hostname>:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://<hostname>:8243/token
```

Generate a Test Access Token

Access Token

```
............................
```

Above token has a validity period of 3600 seconds. And the token has (am_application_scope,default) scopes.

Scopes

```
Read News
```

Validity period

```
3600 Seconds.
```

Re-generate
Scope whitelisting

A scope is not always used for controlling access to a resource. You can also use it to simply mark an access token. There are scopes that cannot be associated to roles (e.g., openid, device_). Such scopes do not have to have roles associated with them. Skipping role validation for scopes is called scope whitelisting.

If you do not want a role validation for a scope in an API's request, add the scope under the `<OAuthConfigurations>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file and restart the server. It will be whitelisted. For example,

```xml
<ScopeWhitelist>
  <Scope>^device_.*</Scope>
  <Scope>some_random_scope</Scope>
</ScopeWhitelist>
```

Next, invoke the Token API to get a token for the scope that you just whitelisted. For example,

```
curl -k -d "grant_type=password&username=admin&password=admin&scope=some_random_scope" -H "Authorization: Basic WmRFUFBvZmZwYVFnR25ScG5iZldtcUtSS3IwYTpsaG5ocEVYUVCMDN3T1FReWpiZTJwDBzc1Vh" -H "Content-Type: application/x-www-form-urlencoded" https://10.100.0.3:8243/token
```

Note that the issued token has the scope you requested. You get the token without any role validation as the scope is whitelisted.

API templates

An API template is its XML representation, which is saved in `<APIM_HOME>/repository/resources/api_templates/velocity_template.xml` file. This file comes with the API Manager by default. You can edit this default template to change the synapse configuration of all APIs that are created.

If you are using a distributed API Manager setup (i.e., Publisher, Store, Gateway and Key Manager components are running on separate JVMs), edit the template in the Publisher node.

Endpoints

An endpoint is a specific destination for a message such as an address, WSDL, a failover group, a load-balance group etc.

WSO2 API Manager has support for a range of different endpoint types, allowing the API Gateway to connect with advanced types of backends. It supports HTTP endpoints, URL endpoints (also termed as address endpoint), WSDL endpoints, Failover endpoints, Load-balanced endpoints. For more information about endpoints, see Working with Endpoints.

Sequences

The API Manager has a default mediation flow that is executed in each API invocation. There are 3 default sequences engaged as in, out and fault which perform following.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>In-sequence is the first place that will be mediated through, once a request is dispatched to a resource of an API. At the end of the in-sequence the request can be forwarded to a back-end application for further processing.</td>
</tr>
<tr>
<td>out</td>
<td>Any responses coming from the back-end system are mediated through the out-sequence of the resource of the API.</td>
</tr>
</tbody>
</table>
Fault sequence is there to handle any errors that may occur while mediating a message through a resource. When the sequence or the proxy service encounters an error during mediation or while forwarding a message, the message that triggered the error is delegated to the specified fault sequence. Using the available mediators it is possible to log the erroneous message, forward it to a special error-tracking service, and send a SOAP fault back to the client indicating the error. We need to configure the fault sequence with the correct error handling instead of simply dropping messages. For more information, see Error Handling.

Caching

For information on configuring caching response messages and caching API calls at the Gateway and Key Manager server, see Configuring Caching.
Tutorials

This section covers the following usecases of the product:
- Getting Started
- API Publishing
- Developer Portal

Getting Started

How do I...
- Create and Publish an API
- Subscribe to an API
- Invoke an API using the Integrated API Console
- Edit and Consume APIs using Swagger Tools

Create and Publish an API

API creation is the process of linking an existing backend API implementation to the API Publisher so that you can manage and monitor the API's lifecycle, documentation, security, community, and subscriptions. Alternatively, you can provide the API implementation in-line in the API Publisher itself.

Click the following topics for a description of the concepts that you need to know when creating an API:
- API visibility
- Resources
- Endpoints
- Throttling tiers
- Sequences
- Response caching

1. Sign in to the WSO2 API Publisher.
   https://<hostname>:9443/publisher (e.g., https://localhost:9443/publisher). Use admin as the username and password.

2. Close the interactive tutorial that starts automatically if you are a first-time user, and click ADD NEW API.

You can go back to the interactive tutorial at a later stage by clicking API Walkthrough on the top right corner.
3. Click **Design a New REST API** and click **Start Creating**.

Let's get started!

Add New API

- **I Have an Existing API**
  Use an existing API's endpoint or the API Swagger definition to create an API.

- **I Have a SOAP Endpoint**
  Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

- **Design a New REST API**
  Design and prototype a new REST API.

Show More Options

Start Creating

4. Give the information in the table below and click **Add** to add the resource.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>PhoneVerification</td>
</tr>
<tr>
<td>Context</td>
<td>/phoneverify</td>
</tr>
</tbody>
</table>

The API context is used by the Gateway to identify the API. Therefore, the API context must be unique. This context is the API's root context when invoking the API through the Gateway.

You can define the API's version as a parameter of its context by adding the `{version}` into the context. For example, `{version}/phoneverify`. The API Manager assigns the actual version of the API to the `{version}` parameter internally. For example, https://localhost:8243/1.0.0/phoneverify. Note that the version appears before the context, allowing you to group your APIs based on the versions.

<table>
<thead>
<tr>
<th>Version</th>
<th>1.0.0</th>
</tr>
</thead>
</table>
4. Access Control

Visibility on Store: Public

Tags: phone, checkNumbers

Tags can be used to filter out APIs matching certain search criteria. It is recommended that you add tags that explain the functionality and purpose of the API as subscribers can search for APIs based on the tags.

Resources

<table>
<thead>
<tr>
<th>URL pattern</th>
<th>Request types</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckPhoneNumber</td>
<td>GET, POST</td>
</tr>
</tbody>
</table>

The selection of the HTTP method should match the actual backend resource. For example, if the actual backend contains the GET method to retrieve details of a phone number, that resource should match a GET resource type with a proper context.
For more information on URL patterns, see API Resources.

5. After you add the resource, click its GET method to expand it. Update the value for Produces as application/xml and the value for Consumes as application/json.

6. Next, add the following parameters. You use these parameters to invoke the API using the integrated API Console, which is explained in later tutorials.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Parameter Type</th>
<th>Data Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneNumber</td>
<td>Give the phone number to be validated</td>
<td>query</td>
<td>string</td>
<td>True</td>
</tr>
<tr>
<td>LicenseKey</td>
<td>Give the license key as 0 for testing purpose</td>
<td>query</td>
<td>string</td>
<td>True</td>
</tr>
</tbody>
</table>

In the resource definition, you define the MIME types. Consumes refers to the MIME type of request accepted by the backend service and Produces refers to the MIME type of response produced by the backend service that you define as the endpoint of the API.

HTTP Post

By design, the HTTP POST method specifies that the web server accepts data enclosed within the body of the request. Therefore, when adding a POST method, API Manager adds the payload parameter to the POST method by default.
7. Once done, click Next: Implement >. Alternatively, click Save to save all the changes made to the API. You can come back later to edit it further by selecting the API and clicking Edit. For details about the states of the API, see Manage the API Lifecycle.

The following parameter types can be defined according to the resource parameters you add.

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Contains the fields added as part of the invocation URL that holds the data to be used to call the backend service.</td>
</tr>
<tr>
<td>header</td>
<td>Contains the case-sensitive names followed by a colon (:) and then by its value that carries additional information with the request which defines the operating parameters of the transaction.</td>
</tr>
<tr>
<td>formData</td>
<td>Contains a properly list of attribute names and values that are included in the body of the message.</td>
</tr>
<tr>
<td>body</td>
<td>An arbitrary amount of data of any type sent with a POST message</td>
</tr>
</tbody>
</table>

You can use the following data type categories, supported by swagger:

- primitive (input/output)
- containers (as arrays/sets) (input/output)
- complex (as models) (input/output)
- void (output)
- file (input)

8. Click the Managed API option.

PhoneVerification: /phoneverify/1.0.0

9. The Implement tab opens. Enter the information in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP/REST endpoint</td>
</tr>
</tbody>
</table>

**Load balanced and fail over endpoints**

The load balanced and failover endpoint types are not selected in this example. For details about these endpoint types, see Working with Endpoints and ESB Endpoints.

Production endpoint

This sample service has two operations: CheckPhoneNumber and CheckPhoneNumbers. Let's use CheckPhoneNumber here.

http://ws.cdyne.com/phoneverify/phoneverify.asmx

To verify the URL, click the Test button next to it (this is the actual endpoint where the API implementation can be found).

Sandbox endpoint

This sample service has two operations: CheckPhoneNumber and CheckPhoneNumbers. Let's use CheckPhoneNumber here.

http://ws.cdyne.com/phoneverify/phoneverify.asmx

To verify the URL, click the Test button next to it.

For more information on Endpoints, see Working with Endpoints.
For additional information, see Enabling CORS for APIs and Adding Mediation Extensions. For details on adding and managing certificates, see Dynamic SSL Certificate Installation.

You can deploy your API as a Prototyped API in the Implement tab. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. You can implement it inline or by specifying an endpoint.

You can invoke the API without a subscription after publishing the API to the Store. For more information, see Deploy and Test as a Prototype.
10. Click **Next: Manage >** and enter the information in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transports</td>
<td>HTTP and HTTPS</td>
<td>The transport protocol on which the API is exposed. Both HTTP and HTTPS transports are selected by default. If you want to limit API availability to only one transport (e.g., HTTPS), clear the checkbox of the other transport.</td>
</tr>
<tr>
<td>Subscription</td>
<td>Select all</td>
<td>The API can be available at different levels of service. They allow you to limit the number of successful hits to an API during a given period.</td>
</tr>
</tbody>
</table>

**Make Default Version**

**Make this the Default Version** checkbox ensures that the API is available in the Gateway without a version specified in the production and sandbox URLs. This option allows you to create a new version of an API and set it as the default version. Then, you can invoke the same resources in the client applications without changing the API gateway URL. This allows you to create new versions of an API with changes, while at the same time allowing existing client applications to be invoked without the client having to change the URLs.

You can only **try out HTTPS based APIs via the API Console** because the API Store runs on HTTPS.
For more information on maximum backend throughput and advanced throttling policies, see Working with Throttling.

11. Click Save & Publish. This publishes the API that you just created to the API Store so that subscribers can use it.

You have created an API.

Related Tutorials

- Create and Publish an API from a Swagger Definition
- Create a Prototyped API with an Inline Script
- Create a WebSocket API
- Create and Publish a SOAP API

Subscribe to an API

You subscribe to a published API before using it in your applications. Subscription enables you to receive access tokens and be authenticated to invoke the API.

The examples here use the PhoneVerification REST API, which is created in the section Create and Publish an API.

1. Sign in to the WSO2 API Store (https://<hostname>:9443/store) and click on an API (e.g., PhoneVerification 1.0.0) to open it.

In a multi-tenanted WSO2 API Manager setup, you can access any tenant's store using the URL http://<hostname>:[port]/store?tenant=<tenant_name>.

2. Note the subscription options for the REST API.

3. Click the Applications menu and click Add Application to create a new application.
4. Enter the name as TestApp and select the per token quota as 50PerMin for the application and click Add.

5. Click APIs and click the PhoneVerification API to view the API's subscription options.

6. Select the application that you just created, a tier, and click Subscribe.

7. Click the View Subscriptions button when prompted.
   The Subscriptions tab opens.

8. Click the Production Keys tab.

   If you have a supported callback URL that sends a callback to a specific server or program soon after your application request is sent, you can specify it under the Callback URL field in the Production Keys tab.
9. Click **Generate Keys** to create an application access token. You can use this token to invoke all APIs that you subscribe to using the same application.

You can set a token validity period in the **Access token validity period** text box. By default, it is set to one hour. If you set a minus value (e.g., -1), the token never expires.

By default, the Client Credentials grant type is used to generate the access token. Make sure the Client Credentials grant type is selected when generating keys from the UI. For more information on how to generate supported grant types of WSO2 API Manager, see **Token API**.

**Access Tokens with specific Scopes**

Access tokens can be generated for specific scopes. A scope acts as a limiting factor on what API resources can be accessed using a token. To generate an access token corresponding to a scope, use the drop down menu under **Scopes** and select the required scope parameter.

If you are using WSO2 Identity Server 5.6.0 as the Key Manager for your API Manager deployment, generating keys will result in the creation of a **Service Provider** in the Identity Server.

10. Install **cURL** if it is not already installed.

   cURL comes by default in some operating systems. You can also use a REST client instead.
a. Open the command line and execute the following cURL command:

```
curl -k -H "Authorization: Bearer <access_token>" -v 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
```

In the above example, the placeholders mentioned in the cURL command format are replaced as follows:

- **<access token>**: Give the test token generated in step 8. Click **Applications**, click on the respective application, which in this case is TestApp, click **Production Key**, and click **copy button** to copy the access token.

- **<API URL>**: Click the respective API, in this case, PhoneVerification - 1.0.0. When the API's **Overview** tab appears in the API Store, copy the production URL and append the required query params to it. For example, `https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0`

  The above example is made up as follows:

  `<API-endpoint>/<resource>?<query-parameter-1>=<query-parameter-1-value>&<query-parameter-2>=<query-parameter-2-value>`
Note the result `<Valid>true</Valid>` that appears in the command line.

```xml
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmni:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmni="http://ws.cdyne.com/PhoneVerify/query">
  <Company>Toll Free</Company>
  <Valid>true</Valid>
  <Use>Assigned to a code holder for normal use.</Use>
  <State>TF</State>
  <RC />
  <OCN />
  <OriginalNumber>18006785432</OriginalNumber>
  <CleanNumber>8006785432</CleanNumber>
  <SwitchName />
  <SwitchType />
  <Country>United States</Country>
  <CLLI />
  <PrefixType>Landline</PrefixType>
  <LATA />
  <sms>Landline</sms>
  <Email />
  <AssignDate />
  <TelecomCity />
  <TelecomCounty />
  <TelecomState>TF</TelecomState>
  <TelecomZip>
  <TimeZone />
  <Lat />
  <Long />
  <Wireless>false</Wireless>
</PhoneReturn>
```

**Troubleshooting**

If you get an error that states "Invalid Credentials", carryout the following steps to overcome the error. This error is a result of the access token expiring. The default validity period of the access token is 1 hour.

i. You can update the token validity period in the **Access token validity period** text box so that the access token is valid for a longer period, or you can even set a minus value (e.g., -1) so that the token never expires.

ii. Re-generate the access token.

Click **Applications**, click the respective application (i.e., TestApp), click **Production Keys**, and click **Re-generate**. Thereafter, use the new access token when running the cURL commands.

11. Similarly, invoke the POST method using the following cURL command:

```
curl -k -H "Authorization :Bearer <access token>" --data "PhoneNumber=<phone_number>&LicenseKey=<license_key>" <api_url>
```
You have subscribed to an API and invoked it.

To unsubscribe from an API, click the Applications menu and click View next to the application used for the subscription. Go to the Subscriptions tab, locate the API, and click the Unsubscribe link associated with it.

If you unsubscribe from an API and then resubscribe with a different tier, it takes approximately 15 minutes for the tier change to be reflected. This is because the older tier remains in the cache until it is refreshed periodically by the system.

Invoke an API using the Integrated API Console

WSO2 API Manager (WSO2 API-M) has an integrated Swagger UI, which is part of the Swagger project.
Swagger is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interacts with the service with a minimal amount of implementation logic. Swagger helps describe a service in the same way that interfaces describe lower-level programming code.

The Swagger UI is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation and more discoverability. The Swagger UI has JSON code and its UI facilitates easier code indentation, keyword highlighting, and shows syntax errors on the fly. You can add resource parameters, summaries, and descriptions to your APIs using the Swagger UI.

For more information also, see the Swagger 2.0 specification.

Let's see how to use the API Console in the Store to invoke an API.

See the following topics for a description of the concepts that you need to know when invoking an API:

- Applications
- Throttling
- Access tokens
- Cross-origin resource sharing - This is needed only if you have the API Store and Gateway in different ports or you want to invoke an API with inline endpoints.

You can only try out HTTPS based APIs via the API Console because the API Store runs on HTTPS.

The examples here use the PhoneVerification REST API, which was created in Create and Publish an API.

1. Sign in to the WSO2 API Store and click an API (e.g., PhoneVerification).
   https://<hostname>:9443/store
2. Subscribe to the API (e.g., PhoneVerification 1.0.0) using the default application and an available tier.

3. On the Applications menu, open the default application you used to subscribe to the API. Click the Production Keys tab and click Generate keys to generate a production key.
3. On the **APIs** menu, select the API that you want to invoke. When the API opens, go to its **API Console** tab.

4. On the **APIs** menu, select the API that you want to invoke. When the API opens, go to its **API Console** tab.

---

Production and Sandbox Tokens

To generate keys for the Sandbox endpoint, go to the **Sandbox Keys** tab. For more details, see [Maintaining Separate Production and Sandbox Gateways](#).
If you have subscribed to an application, the retrieved access token value appears automatically as the **Authorization Bearer Token**.

5. Expand the GET method and click Try it out. Provide the required parameters and click **Execute**. For example,

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PhoneNumber</strong></td>
<td>E.g., 18006785432</td>
</tr>
<tr>
<td><strong>LicenseKey</strong></td>
<td>Give 0 for testing purpose</td>
</tr>
<tr>
<td><strong>Authorization</strong></td>
<td>The API console is automatically populated by the access token that you generated in step 3 after subscribing to the API. The token is prefixed by the string &quot;Bearer&quot; as per the OAuth bearer token profile. <strong>OAuth security</strong> is enforced on all published APIs. If the application key is invalid, you get a 401 Unauthorized response in return.</td>
</tr>
</tbody>
</table>
**Troubleshooting**

If you cannot invoke the API's HTTPS endpoint (this causes the SSLPeerUnverified exception), it could be because the security certificate issued by the server is not trusted by your browser. To resolve this issue, access the HTTPS endpoint directly from your browser and accept the security certificate.

If the API Manager has a certificate signed by a Certificate Authority (CA), the HTTPS endpoints should work out of the box.

Note the response for the API invocation. As we used a valid phone number in this example, the response is valid.
You have invoked an API using the Swagger API Console.

**Edit and Consume APIs using Swagger Tools**

WSO2 API Manager has an integrated Swagger Editor, which is part of the Swagger project.

**Swagger** is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code, and interact with the service with a minimal amount of implementation logic. Swagger helps describe a service in the same way that interfaces describe lower-level programming code.

The **Swagger Editor** is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation, and more discoverability. The Swagger Editor has JSON code and its UI facilitates easier code indentation, keyword highlighting and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger Editor.

API Manager supports **Open API 3.0** and **Open API 2.0** specifications and you can simply create, import, edit and consume the APIs defined in both specifications.

In this tutorial, let's see how you can add interactive documentation to an API by directly editing the Swagger code via the API Publisher UI.

This tutorial uses the PhoneVerification API created in Create and Publish an API.

1. Sign in to the API Publisher and choose to design a new REST API.
   
   https://<hostname>:9443/publisher
Let’s get started!

Add New API

1. Click Start Creating. In the Design tab, give an API name, a context and a version, and click Edit Source under the API Definition section. The Swagger UI opens.

2. You can select the OpenAPI specification version when you start creating a new REST API. To see the options, click Show More Options. Select the supported API specification from the drop-down list.

3. Click Start Creating.

4. In the Design tab, give an API name, a context and a version, and click Edit Source under the API Definition section. The Swagger UI opens.

5. Add a GET and POST method for the API.
   a. Under the paths object, remove the {} and add the following code, as shown in the screenshot below.

   In the code below, note that you have a resource defined with the URL pattern /CheckPhoneNumber under the paths object. This is followed by the HTTP methods GET and POST. For each HTTP method, the following parameters are defined.
• **responses**: An object to hold responses that can be used across operations. See the Swagger specification for details.
• **x-auth-type**: WSO2-specific object to define the authentication type of the method.
• **x-throttling-tier**: WSO2-specific object to define the throttling tier of the method.

```
/CarePhoneNumber:
get:
 responses:
  '200':
    description: ''
    x-auth-type: Application & Application User
    x-throttling-tier: Unlimited
post:
 responses:
  '200':
    description: ''
    x-auth-type: Application & Application User
    x-throttling-tier: Unlimited
```

**Troubleshooting**

If you get an error after adding the API definition in the Swagger UI, first check the indentation of the code that you added, which defines the API, because Swagger throw errors if the indentation is not correct.

b. Click **Apply Changes**.  
   This adds a resource with two HTTP methods into the API, which is visible in the API Publisher.

Let's assume that the backend of this API sends the response in XML format. Let's document this under the GET method in the resource that we just added.

5. Change the response content type to XML.
   a. Click **Edit Source** and add the following code under the GET method.

   ```yaml
   produces:
   - text/xml
   ```

   b. Click **Apply Changes**. The response content type that you updated is visible when you expand the GET method in the API Publisher.

You can use this attribute to document the type of the response message that the backend sends. **It does not do any message type conversion.** You can add multiple values as a comma-separated list. Example:

```yaml
produces:
- text/xml, application/json
```
6. Define parameters that correspond to the GET method.
   a. Click **Edit Source** and add the following code, which defines two parameters to the method, under the GET method.

   ```
   parameters:
   - name: PhoneNumber
     in: query
     required: true
     type: string
     description: Give the phone number to be validated
   - name: LicenseKey
     in: query
     required: true
     type: string
     description: Give the license key as 0 for testing purpose
   ```

   b. Click **Apply Changes**.
   
   The two parameters with their descriptions that you added are visible when you expand the GET method in API Publisher.
7. Add a summary and description for the GET method.
   a. Click Edit Source and add the following code, which defines a summary and description, to the GET method.
   
   ```
   summary: Check the validity of your phone number
   description: "Phone Verification validates a telephone number and returns carrier information, location routing etc."
   ```
   
   b. Click Apply Changes.
   The summary and description of the GET method that you added is visible when you expand the GET method in the API Publisher.
8. Add parameters to the POST method and change the POST method datatype.
   a. Click **Edit Source** and add the following code under the POST method, which defines two parameters named `PhoneNumber` and `LicenseKey` to pass in the payload. It also changes the datatypes of the parameters to `application/x-www-form-urlencoded` as the backend expects that datatype.

   ```
   consumes:
   - application/x-www-form-urlencoded
   parameters:
     - name: PhoneNumber
       in: formData
       required: true
       type: string
       description: Give the phone number to be validated
     - name: LicenseKey
       in: formData
       required: true
       type: string
       description: Give the license key as 0 for testing purpose
   ```

   b. Click **Apply Changes**.
   The two parameters with their descriptions that you added are visible when you expand the POST method in the API Publisher.
9. Change the title of the API,
   a. Click Edit Source, and add the following code in the Swagger UI.
      This is the title that is visible to the consumers in the API Store after the API is published.

```
info:
  title: PhoneVerificationAPI
```
You can see how this change is reflected in the API Store in step 15.

b. Click Apply Changes and complete the API creation process.

10. Complete the rest of the API creation process.
For more information, see step 7 onwards under Create and Publish an API.
11. Click Go to APIStore and the API you just published appears.
12. Click API Console.
Note that the changes that you made earlier now appear in the API Store for consumers.
In this tutorial, you have seen how the integrated Swagger Editor can be used to design, describe, and document your API, so that the API consumers get a better understanding of the API's functionality.

### API Publishing

How do I...

- Add API Documentation
- Manage the API Lifecycle
- Publish to Multiple External API Stores
- Publish through Multiple API Gateways
- Block Subscription to an API
- Enforce Throttling and Resource Access Policies
- Change the Default Mediation Flow of API Requests
- Map the Parameters of your Backend URLs with the API Publisher URLs
- Convert a JSON Message to SOAP and SOAP to JSON
- Invoke an API using a SOAP Client
- Create and Publish an API from a Swagger Definition
- Create a WebSocket API
- Create a Prototyped API with an Inline Script
- Pass a Custom Authorization Token to the Backend
- Create and Publish a SOAP API
- Disable Message Chunking
- Enable API Indexing on Remote Publisher and Store Nodes
- Remove Specific Request Headers From Response
Add API Documentation

This section covers the following:
- Add API Documentation In-line, using a URL or a File
- Add Apache Solr-Based Indexing

Add API Documentation In-line, using a URL or a File

**API documentation** helps API subscribers understand the functionality of the API, and API publishers to market their APIs better and sustain competition. Using the API Publisher, you can add different types of documentation from different sources. All documents created in the API Publisher have unique URLs to help improve SEO support.

The documentation types supported in the API Publisher are as follows:

- **In-line**: Hosts documentation (How-tos, Samples, SDK, forums etc.) in WSO2 API Publisher itself and allows it to be edited directly from the UI.
- **URL**: Links to file references (URLs) of an external configuration management system.
- **File**: Allows to upload the documentation directly to the server.
- **Using the integrated API Console**

Do you want to set different visibility levels to the API documentation than the API? See API documentation visibility.

1. Sign in to the WSO2 API Publisher.
   
   https://<hostname>:9443/publisher

2. Click the API (e.g., PhoneVerification 1.0.0) for which you want to add documentation.

3. Add in-line documentation.
   
   a. Select the **Docs** tab of the API and click **Add New Document**.
b. Enter the following details to create documentation in-line.

<table>
<thead>
<tr>
<th>Name</th>
<th>PhoneVerification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Check the validity of a phone number</td>
</tr>
<tr>
<td>Type</td>
<td>How To</td>
</tr>
<tr>
<td>Source</td>
<td>In-line</td>
</tr>
</tbody>
</table>

Update button can be used to update/change the document information.

c. Click Add Document.
d. Click Edit Content to open an embedded editor.
e. Edit the document content in-line using the embedded editor and click Save and Close.

PhoneVerification

```
Determine whether a phone line is wireless or a landline - in real time.
Even if the phone number has been ported between service providers, Phone Verification will return the latest information. Phone Verification validates the ten digits of a U.S. telephone number and returns carrier information as well as the Location Routing Number (LRN) for telephone numbers administered by the North American Numbering Plan Administration. LRN is a unique 10-digit number that represents a telephone switch through which multiple phone numbers are routed. The LRN enables Local Number Portability (LNP), which allows phone numbers to be ported to different carriers.
The benefits of using the Phone Verification API include:
- Determine if a phone number is valid
- Remove dashes, parenthesis and spaces
- Receive latest SMTP email string data
- Define timezone to restrict calling times
- Validate wireless vs. landline data for compliance
```

The API’s **Doc** tab opens.

4. **Add documentation using a URL**.
   a. Click **Add New Document** to add another doc type.
   b. Enter the following information to create another doc using a URL.

<table>
<thead>
<tr>
<th>Name</th>
<th>CDYNE Wiki</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td><strong>CDYNE Phone Notify API</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Other (specify)</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>URL <a href="https://cdyne.com/downloads/SPECS_Phone-Notify.pdf">https://cdyne.com/downloads/SPECS_Phone-Notify.pdf</a></td>
</tr>
</tbody>
</table>

PhoneVerification - 1.0.0

5. **Add documentation using a file**.
   a. Click **Add New Document** to add yet another document using a file.
   b. Enter the following information.

<table>
<thead>
<tr>
<th>Name</th>
<th>API Manager Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td><strong>API Manager Samples</strong></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Samples &amp; SDK</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>You can provide any file format (common formats are .pdf, .html, .doc, .txt) of any size. For example, use the sample PDF file here.</td>
</tr>
</tbody>
</table>
c. Click Add Document.
   You have now added three documents to the API: in-line, using a URL, and a file.

6. Log in to the WSO2 API Store and click the PhoneVerification 1.0.0 API.
   https://<hostname>:9443/store
6. Go to the API's **Documentation** tab and see the documents listed by type.
   Click the links to see the documentation content. As a subscriber, you can read the documentation and learn about the API.

You have created documentation using the API Publisher and viewed them as a subscriber in the API Store.

**Add Apache Solr-Based Indexing**

WSO2 API Manager has Apache Solr based indexing for API documentation content. It provides both the API Publisher and Store a full-text search facility to search through the API documentation, and find the documents and related APIs. The search syntax is `doc:keyword`. The search criteria looks for the keyword in any word/phrase in the documentation content and returns both the matching documents and associated APIs.
The following media types have Apache Solr based indexers by default, which are configured using the `<Indexers>` element in `<APIM_HOME>/repository/conf/registry.xml` file.

- Text : text/plain
- PDF : application/pdf
- MS word : application/msword
- MS Powerpoint : application/vnd.ms-powerpoint
- MS Excel : application/vnd.ms-excel
- XML : application/xml

Writing a custom index

In addition to the default indexes, you can write your own indexer implementation and register it as follows:

1. Write a custom indexer.
   The following is the sample indexer code.

   ```java
   package org.wso2.indexing.sample;
   import java.util.HashMap;
   import java.util.List;
   import java.util.Map;
   import java.util.Arrays;
   import org.apache.solr.common.SolrException;
   import org.wso2.carbon.registry.core.exceptions.RegistryException;
   import org.wso2.carbon.registry.core.utils.RegistryUtils;
   import org.wso2.carbon.registry.indexing.IndexingConstants;
   import org.wso2.carbon.registry.indexing.AsyncIndexer.File2Index;
   import org.wso2.carbon.registry.indexing.indexer.Indexer;
   import org.wso2.carbon.registry.indexing.solr.IndexDocument;
   public class PlainTextIndexer implements Indexer {
       public IndexDocument getIndexedDocument(File2Index fileData) throws SolrException,
       RegistryException {
           /* Create index document with resource path and raw content*/
           IndexDocument indexDoc = new IndexDocument(fileData.path,
           RegistryUtils.decodeBytes(fileData.data), null);
           /* You can specify required field/value pairs for this indexing document.
           * When searching we can query on these fields */
           Map<String, List<String>> fields = new HashMap<String, List<String>>();
           fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
           Arrays.asList(fileData.mediaType));
           if (fileData.mediaType != null) {
               fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
               Arrays.asList(fileData.mediaType));
           } else {
               fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
               Arrays.asList("text/plain"));
           }
           /* set fields for index document*/
           indexDoc.setFields(fields);
           return indexDoc;
       }
   }
   ``

2. Add the custom indexer JAR file to the `<API-M_HOME>/repository/components/lib` directory.
3. Update the `<Indexers>` element in the `<API-M_HOME>/repository/conf/registry.xml` file with the new indexer.
   The content is indexed using this media type. For example,

   ```xml
   <indexers>
       <indexer class="org.wso2.indexing.sample.PlainTextIndexer" mediaTypeRegEx="text/plain"
       profiles="default,api-store,api-publisher"/>
   </indexers>
   ```
The attributes of the above configuration are described below:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>Java class name of the indexer.</td>
</tr>
<tr>
<td>mediaTypeRegEx</td>
<td>A regular expression (regex) pattern to match the media type.</td>
</tr>
<tr>
<td>profiles</td>
<td>API-M profiles in which the indexer is available.</td>
</tr>
</tbody>
</table>

4. Restart the server.
5. Add the API documentation using the new media type and thereafter search some term in the documentation using the syntax (doc:keyword).
   You can now see how the documentation has got indexed according to the media type.

**Manage the API Lifecycle**

In order to provide flexibility of integration between APIs with internal and external environments the lifecycle of an API is important. APIs provided by WSO2 API Manager has a lifecycle containing six stages which allows you to identify in which state that the API is currently on.

In this section, we show you how to
- Create a New API Version
- Deploy and Test as a Prototype
- Publish the New Version and Deprecate the Old
- Customize API Life Cycle

**Create a New API Version**

A new API version is created when you want to change a published API's behavior, authentication mechanism, resources, throttling tiers, target audiences etc. It is not recommended to modify a published API that has subscribers plugged to it.

After creating a new version, you typically deploy it as a prototype for early promotion. A prototype can be used for testing, without subscription, along with the published versions of the API. After a period of time during which the new version is used in parallel with the older versions, the prototyped API can be published and its older versions can be deprecated.

The example here uses the PhoneVerification API, which you created in the Create and Publish an API section.

The steps below show you how to create a new version of an existing API.

1. Sign in to the WSO2 API Publisher.
   https://<hostname>:9443/publisher
   Refer step 1 of Create and Publish an API to sign in to the Publisher.
2. Click on the API name that corresponds to the API for which you want to create a new version (e.g., PhoneVerification 1.0.0).
   The API opens.
3. Click Create New Version.
4. Enter a version number, select the default version option, and click **Done**. The **APIS** page opens.

PhoneVerification - 1.0.0

Create New Version

New Version:* 2.0.0

Make this the Default Version

Info!
No default version defined for the current API

- The **Default Version** option means that you make this version the default in a group of different versions of the API. A default API can be invoked without specifying the version number in the URL. For example, if you mark http://host:port/youtube/2.0 as
4. Click the **Edit** icon of the new API version to edit it.

5. Do the required modifications to the API.
   For example, let's assume that the POST method is redundant, and let's delete it from the resource that we added to the API at the time it was created.

6. There is a known issue in API Manager 2.2.0 which is when creating a new version of an API with the SOAP Endpoint, it cannot be
You have created a new version of an API. In the next tutorial, let's learn how to deploy this API as a prototype and test it with its older versions.

### Deploy and Test as a Prototype

An **API prototype** is created for the purpose of early promotion and testing. You can deploy a new API or a new version of an existing API as a prototype. It gives subscribers an early implementation of the API that they can try out without a subscription or monetization, and provide feedback to improve. After a period of time, publishers can make changes that the users request and publish the API.

The example here uses the API **PhoneVerification 2.0.0**, which you created in the previous tutorial.

1. Sign in to the WSO2 API Publisher and select the API (e.g., **PhoneVerification 2.0.0**) that you want to prototype.
   
   https://<hostname>:9443/publisher

---

**Tip**: By default, only the latest version of an API is shown in the API Store. If you want to display multiple versions, set the `<DisplayMultipleVersions>` element to `true` in the `<API-M_HOME>/repository/conf/api-manager.xml` file, and restart the server.
2. Click GO TO OVERVIEW.
3. Click the Lifecycle tab of the API and click Deploy as Prototype.
   After creating a new version, you typically deploy it as a prototype for the purpose of testing and early promotion.

PhoneVerification - 2.0.0

- Leave the Requires Re-Subscription check box cleared if you want all users who are subscribed to the older version of the API to be automatically subscribed to the new version. If not, they need to subscribe to the new version again.
- You can choose to deprecate old versions of this API at this stage by selecting the Deprecate Old Versions check box.

4. Sign in to the API Store and click on the newly prototyped API.
   https://<hostname>:9443/store
The APIs Overview page opens. Note the following:
- There are no subscription options.
- There are two sets of URLs (with and without the version). This is because you marked the 2.0.0 version as the default version in step 4 of the previous tutorial.
- Other features such as documentation, social media, and forums are available.
5. Click the **API Console** tab.
   Note that the POST method is not available as we removed that in the new version.
Let's invoke the prototyped API.

6. In the **API Console** of the prototyped API, expand the GET method, click **Try it out**, and enter the following parameter values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneNumber</td>
<td>E.g., 18006785432</td>
</tr>
<tr>
<td>LicenseKey</td>
<td>Give 0 for testing purposes</td>
</tr>
</tbody>
</table>

7. Click **Execute** to invoke the API.

Note the response that appears in the console. You do not have to subscribe to the API or pass an authorization key to invoke a prototyped API.
8. Similarly, try to invoke the 1.0.0 version of the API without an access token. Note that you get an authentication error as "Missing credentials", because version 1.0.0 is a published API.
In this tutorial, you have prototyped an API and tested it along with its older and published versions. In the next tutorial, you can learn how to publish the prototyped API and deprecate its older versions.

Publish the New Version and Deprecate the Old

You publish an API to make it available for subscription in the API Store. If you set up multiple tenants, your tenant store will be visible to other tenants as well. Therefore, users of the other tenants can view the APIs that are published in your default API Store. This allows you to advertise your APIs to a wider audience. Although the APIs that are published in your tenant store are visible to the users of other tenant stores, they need to sign in to your tenant store in order to subscribe and use them.

For more details on the API lifecycle stages, see API lifecycle.

The steps below show you how to publish an API to its default API Store:

1. Sign in to the WSO2 API Publisher as a user who has the publisher role assigned to themselves.
   https://<hostname>:9443/publisher
2. Click on the API that you prototyped in the previous tutorial (e.g., PhoneVerification 2.0.0).

3. Click GO TO OVERVIEW.
4. Go to the API's Lifecycle tab and click Publish.

The Lifecycle tab is only visible to users with publisher privileges.

PhoneVerification - 2.0.0

Current State: PROTOTYPED

- Leave the Requires Re-Subscription check box cleared if you want all users who are subscribed to the older version of the API to be automatically subscribed to the new version. If not, they need to subscribe to the new version again.
- You can choose to deprecate old versions of this API at this stage by selecting the Deprecate Old Versions check box.

Note that you should not change these checklist items through the methods mentioned below.

1. Editing the `<API-M_HOME>/repository/resources/lifecycles/APILifecycle.xml` file.
2. Editing through the Management Console via Extensions > Lifecycles.

For instructions on changing the text appearing in the API Publisher UI, see Adding Internalization and Localization.
The API is now published to the default API Store.

5. Sign in to the default Store and click on the APIs menu to see the API that you just published listed there.

6. Go back to the WSO2 API Publisher and click the API that you want to deprecate (e.g., PhoneVerification 1.0.0).

7. Go to the API's Lifecycle tab and click Deprecate.

The API is now deprecated.

8. Go back to the WSO2 API Store, click the Applications menu, click on TestApp to open the TestApp application, which you used to subscribe to the API. Click on the Subscriptions tab to view the deprecated API. The subscriptions made to the older API versions should be deprecated now.

You have published an API to the API Store and deprecated its previous versions.

Customize API Life Cycle

APIs created in WSO2 API Manager have their own life cycle consisting of the following: a set of life cycle states, specific actions for each state transition, and a checklist of items before a state transition occurs. An API has a predefined life cycle consists of six states. This tutorial demonstrates how you can edit the default API lifecycle and customize it according to your requirements.

This capability is not available in WSO2 API Manager versions prior to 1.10.0.

Follow the steps below to add a new state to the default life cycle.

1. Sign in to the API Publisher and click on the API that you created previously (e.g., PizzaShackAPI - 1.0.0).
2. Click Lifecycle to view the current states available by default.
3. Open the WSO2 API Manager Management Console: https://localhost:9443/carbon
4. Navigate to Extensions > Configure > Lifecycles.

5. Click the View/Edit link corresponding to the default API LifeCycle.
The APILifeCycle configurations appear.

```xml
<aspect name="APILifeCycle"
      class="org.wso2.carbon.governance.registry.extensions.aspects.DefaultLifeCycle">
  <configuration type="literal">
    <lifecycle>
      <scxml xmlns="http://www.w3.org/2005/07/scxml"
             version="1.0"
             initialstate="Created">
        <state id="Created">
          <datamodel>
            <data name="checkItems">
              <item name="Deprecate old versions after publish the API"
                    forEvent=""/>
              <item name="Requires re-subscription when publish the API"
                    forEvent=""/>
            </data>
            <data name="transitionExecution">
                <execution forEvent="Deploy as a Prototype"
                           class="org.wso2.carbon.apimgt.impl.executors.APIExecutor"/>
                <execution forEvent="Publish"
                           class="org.wso2.carbon.apimgt.impl.executors.APIExecutor"/>
            </data>
        </state>
        <state id="Prototyped">
          <datamodel>
            <data name="checkItems">
              <item name="Deprecate old versions after publish the API"
                    forEvent=""/>
              <item name="Requires re-subscription when publish the API"
                    forEvent=""/>
            </data>
            <data name="transitionExecution">
                <execution forEvent="Publish"
                           target="Published"/>
                <execution forEvent="Deploy as a Prototype"
                           target="Prototyped"/>
            </data>
        </state>
      </lifecycle>
    </scxml>
  </configuration>
</aspect>
```
<state id="Published">
  <datamodel>
    <data name="transitionExecution">
      <execution forEvent="Block" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
      <execution forEvent="Deprecate" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
      <execution forEvent="Demote to Created" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
      <execution forEvent="Deploy as a Prototype" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
    </data>
  </datamodel>
  <transition event="Block" target="Blocked"/>
  <transition event="Deploy as a Prototype" target="Prototyped"/>
  <transition event="Demote to Created" target="Created"/>
  <transition event="Deprecate" target="Deprecated"/>
  <transition event="Publish" target="Published"/>
</state>

<state id="Blocked">
  <datamodel>
    <data name="transitionExecution">
      <execution forEvent="Re-Publish" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
      <execution forEvent="Deprecate" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
    </data>
  </datamodel>
  <transition event="Deprecate" target="Deprecated"/>
  <transition event="Re-Publish" target="Published"/>
</state>

<state id="Deprecated">
  <datamodel>
    <data name="transitionExecution">
      <execution forEvent="Retire" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
    </data>
  </datamodel>
</state>
<transition event="Retire" target="Retired"/>
</state>

<state id="Retired">
</state>
</scxml>
6. Update the API lifecycle definition.
   a. Add a sample state to the API Lifecycle.
      Copy and paste the following sample code in the file.

      ```xml
      <state id="Rejected">
        <datamodel>
          <data name="checkItems">
            <item name="Deprecate old versions after rejecting the API" forEvent=""></item>
            <item name="Remove subscriptions after rejection" forEvent=""></item>
          </data>
          <data name="transitionExecution">
            <execution forEvent="Re-Submit" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor"></execution>
            <execution forEvent="Retire" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor"></execution>
          </data>
        </datamodel>
        <transition event="Re-Submit" target="Published"/>
        <transition event="Retire" target="Retired"/>
      </state>
      ```

      The sample `REJECTED` state is added between `PUBLISHED` and `RETIRED`. It uses the `Re-submit` and `Retire` state transition events to change to the consequent states. Custom checklist items are also given under "checkItems", which are tasks to be done in a state transition. You can select/deselect these items in the management console.

      The same execution class is used (`org.wso2.carbon.apimgt.impl.executors.APIExecutor`) for all state transitions. However, you can plug your own execution code when modifying the life cycle configuration.

      For example, if you want to add notifications for a specific state transition, you can plug your own custom execution class for that particular state in the API life cycle. Any changes are updated in the Lifecycle tab accordingly.

   b. Add a new transition event under the `PUBLISHED` state, to show the state change to `REJECTED`.

      ```xml
      <transition event="Reject" target="Rejected"/>
      ```

7. Make the transition event visible in API Publisher.
   b. Add the following code in the file. Note that the key value in the JSON pair should be lowercase.

      ```json
      "reject": "Rejected",
      ```

8. Restart the WSO2 API Manager server.
9. Re-open API Publisher and check the Lifecycle to see the changes.
Consider the following points when extending and customizing the API lifecycle XML configuration.

- Do not change the life cycle name since it needs to be engaged with the APIs dynamically.
- Make sure you keep the **PUBLISHED** and **PROTOTYPED** states as those two states will be used by API Publisher in the API creation wizard.

For more details on customizing the API lifecycle, see *Extending the API Life Cycle*.

**Publish to Multiple External API Stores**

You can share an API with application developers who are subscribed to the API Stores of other tenants. This allows you to advertise your APIs to a wider community. Subscribers of other tenant stores can view and browse your APIs; however, the users must visit your (the original publisher’s) Store to subscribe to the APIs.

The following diagram illustrates the process involved when an API Publisher publishes to multiple API Stores.
The API Publisher of Tenant_1, who is located in Node_1, is publishing an API to its API Store. In addition to that, the API Publisher publishes the API to the following three external stores:

- API Store of Tenant_2 in same node.
- API Store of Tenant_3 in same node.
- API Store of Tenant_1 in Node 2

The capability to publish to external API Stores is not configured by default. Follow the steps below to configure it. In this guide, we use two separate instances of WSO2 API Manager and we publish from one instance to the Store of the other instance.

1. Copy the WSO2 API Manager product pack to two different locations. If needed, you can download the WSO2 API Manager product pack from here.
2. Go to the `<API-M_HOME>/repository/conf/carbon.xml` file of the second instance and change its port by an offset of 1. The port offset is set to avoid the port conflicts that occur when you run more than one WSO2 product on the same host.

```
<Offset>1</Offset>
```

3. Start both API-M servers.
   Let's publish from the first instance of WSO2 API Manager to the Store of the second instance, which in this tutorial we consider as the external API Store.
In a multi-tenant environment, you must sign in using the tenant’s credentials.

5. Click **Main > Resources > Browse**.

   ![Diagram of WSO2 API Manager interface]

   The Registry opens.


   ![Screenshot of the Registry interface]

   ![Tree view of the Registry]

   In this example, `http://localhost:9764/store` is the API Store of the second WSO2 API Manager instance.

7. Click the **Edit as Text** link, uncomment the `<StoreURL>` element under the `<ExternalAPIStores>` element, and add the details of each external API store that you need to publish APIs to.

   In this example, `- http://localhost:9764/store` is the API Store of the second WSO2 API Manager instance.
You publish to its super tenant’s Store (admin/admin).

For this tutorial change the DisplayName to Store2, so that it is clear that we are referring to the second WSO2 API-M instance, which we are using as the external Store.

The port is 9764 as you incremented it by 1 in step 2.

If the second WSO2 API Manager instance has multiple tenants and you want to publish to a tenant’s Store, the tenant’s Store URL and credentials must be given here.

```xml
<ExternalAPIStores>
  <StoreURL>http://localhost:9763/store</StoreURL>
  <ExternalAPIStore id="Store2" type="wso2"
    className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher">
    <DisplayName>Store2</DisplayName>
    <Endpoint>http://localhost:9764/store</Endpoint>
    <Username>admin</Username>
    <Password>admin</Password>
  </ExternalAPIStore>
</ExternalAPIStores>
```

If you want to configure more than one external store, change the configuration in `<ExternalAPIStore>` and add it to the `external-api-stores.xml`.

For example, if we have three API Stores, and one is a super tenant and other two are tenant stores, you can configure these three external stores as shown below.

```xml
<ExternalAPIStores>
  <!--Configuration to set the store URL of the current running APIM deployment. APIs published to external stores will be redirected to this URL-->
  <StoreURL>http://<ip_address>:<port>localhost:9763/store</StoreURL>

  <ExternalAPIStore id="SLStore" type="wso2"
    className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher">
    <DisplayName>SL-Store</DisplayName>
    <Endpoint>http://<ip_address>:<port>/store</Endpoint>
    <Username>admin</Username>
    <Password>admin</Password>
  </ExternalAPIStore>

  <ExternalAPIStore id="USStore" type="wso2"
    className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher">
    <DisplayName>US-Store</DisplayName>
    <Endpoint>http://<ip_address>:<port>/store</Endpoint>
    <Username><tenantadmin_username>@{tenant_domain}</Username>
    <Password><tenantadmin_password></Password>
  </ExternalAPIStore>

  <ExternalAPIStore id="UKStore" type="wso2"
    className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher">
    <DisplayName>UKStore</DisplayName>
    <Endpoint>http://<ip_address>:<port>/store</Endpoint>
    <Username><tenantadmin_username>@{tenant_domain}</Username>
    <Password><tenantadmin_password></Password>
  </ExternalAPIStore>
</ExternalAPIStores>
```

In a multi-tenant environment, each tenant can publish to different external Stores by changing the above file in their tenant space.

For more information on how APIs appear and are available for subscription in a multi-tenant environment, see API visibility and subscription. Note that publishing to an external Store only means that the API is advertised there. To subscribe, you must always register and sign in to the original publisher's tenant Store.
Note the following in the configuration above:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ExternalAPIStore id=&quot;&quot; type=&quot;&quot; className=&quot;&quot;&gt;</td>
<td></td>
</tr>
</tbody>
</table>
  * id: The external store identifier, which is a unique value.  
  * type: The type of the Store. This can be a WSO2-specific API Store or an external one, which has a different implementation from the default API Store.  
  * className: The implementation class inside the WSO2 API Manager distribution.  
  The default className specified is org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher, which is used when WSO2 specific API Store is used. However, if you are using an external API Store, the class should be customized by extending the org.wso2.carbon.apimgt.api.model.APIPublisher interface, and the fully qualified class name of the new implementation should be used as the className parameter. |
| <StoreURL> | The URL of the API Store of the current API-M deployment. This is the URL to the API in the original publisher's store. APIs that are published to external stores are redirected to this URL. |
| <DisplayName> | The name of the external API Store that is displayed in the Publisher UI. |
| <Endpoint> | The URL of the external API Store. |
| <Username> and <Password> | The credentials of a user who has permissions to create and publish APIs. |

The registry changes are applied dynamically. You do not need to restart the server.

8. Click **Save Content**.
9. Sign in to the API Publisher of the first instance as admin/admin and if you do not have any APIs that are in the published state created, **create an API**.

   In a multi-tenant environment, sign in to the API Publisher using your tenant's credentials.

10. Click on the newly created or existing API.  
    Here you see a new tab named **External API Stores** added to the API Publisher console.

    This tab is only visible when viewing API's that are in the published state.
10. Select the Store that you want to publish to (in this case, Store2) and click **Save**.

11. Sign in to the external API Store (in this case, http://localhost:9764/store) and click on the API that you just published.

A link appears as **Visit Publisher Store**, which directs you to the original publisher’s store (in this case, http://localhost:9763/store) through which you can subscribe to the API.
You have successfully added multiple external stores to your registry and published your APIs to them.

**Publish through Multiple API Gateways**

You can configure multiple API Gateway environments in WSO2 API Manager that publish to a single API Store when you require distributing the gateway load comes in. It helps you to distribute the API Gateway load to multiple nodes and also gives you some logical separation (e.g., production vs. sandbox) between the APIs in the API Store. When you publish an API through multiple Gateway environments, the APIs in the API Store will have different server hosts and ports.

The steps below explain how to configure and publish to multiple Gateways. In this guide, we set up three (3) WSO2 API Manager (WSO2 API-M) instances in the same server. In a typical production environment, the Gateways will ideally be in separate servers.

- **Instance 1**: Acts as the node that provides the API Publisher, Store, and the Key Manager functionality.
- **Instance 2**: Acts as a production Gateway node.
- **Instance 3**: Acts as a sandbox Gateway node.

1. Copy the WSO2 API Manager (WSO2 API-M) product pack into three (3) separate folders.
2. Let's add offsets to the default ports of the two Gateway instances. A port offset ensures that there are no port conflicts when more than one WSO2 product runs on the same server.
   - Open the `<API-M_HOME>/repository/conf/carbon.xml` file in the second API Manager instance, and add an offset of 1 to its default port. This increments its default server port, which is 9443, by 1.
   ```xml
   <Offset>1</Offset>
   ```
   - Open the `<API-M_HOME>/repository/conf/carbon.xml` file in the third API Manager instance and add an offset of 2 to its default port. This increments its default server port, which is 9443, by 2.
   ```xml
   <Offset>2</Offset>
   ```
3. Open the `<API-M_HOME>/repository/conf/api-manager.xml` files in the second and third API Manager instances and set the `<EnableThriftServer>` property to false. This is done to disable the thrift server in the two Gateway instances. Thrift server is needed for the Key Manager functionality. It is not needed in the Gateway instances.
   ```xml
   <EnableThriftServer>false</EnableThriftServer>
   ```
4. Open the `<API-M_HOME>/repository/conf/api-manager.xml` files in the second and third API Manager instances and change the following. This is done for the two Gateway instances to be able to communicate with the Key Manager that is in the first API Manager instance.
<AuthManager>
    <ServerURL>https://<IP of the first instance>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
</AuthManager>

...<APIKeyValidator>
    <ServerURL>https://<IP of the first instance>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
    <RevokeAPIURL>https://<IP of the first instance>:8243/revoke</RevokeAPIURL>
</APIKeyValidator>

6. Open the <API-M_HOME>/repository/conf/api-manager.xml files in all the Gateway instances and uncomment the following configuration:

<ThriftClientPort>10397</ThriftClientPort>

You are done configuring the two API Gateway instances.

7. Open the <API-M_HOME>/repository/conf/api-manager.xml file in the first API Manager instance, add two API Gateway environments under the <Environments> element, and comment out the <environment> element that comes by default.

This is done to point to the two API Gateway instances from the first instance.

- There can be different types of environments, and the allowed values are production, hybrid, and sandbox.
- An API deployed on a production type gateway will only support production keys.
- An API deployed on a sandbox type gateway will only support sandbox keys.
- An API deployed on a hybrid type gateway will support both production and sandbox keys.
- The api-console element specifies whether the environment should be listed in API Console or not.
- The Gateway environment names must be unique.

Example

<Environment type="production" api-console="true">
    <Name>Production Gateway</Name>
    <Description>Production Gateway Environment</Description>
    <ServerURL>https://localhost:9444/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
</Environment>

<Environment type="hybrid" api-console="true">
    <Name>Production and Sandbox</Name>
    <Description>Hybrid Gateway Environment</Description>
    <ServerURL>https://localhost:9445/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
</Environment>

If you have multiple gateways that support one type of key (e.g., when there are two gateways that support the production keys, as seen in the above code snippet), the environments you add via the <API-M_HOME>/repository/conf/api-manager.xml file will be visible in a drop-down list of the API Console tab, which is in the API Store of instance 1. This allows subscribers to send API requests to any selected Gateway.
Start all the WSO2 API-M instances.
Make sure to start instance 1 first before starting the other two instances.

Sign in to the API Publisher in the first WSO2 API-M instance and choose to edit an API.

To stop a given Gateway environment from being displayed in the API Console tab, you can set the `api-console` attribute to `false` in the `<Environment>` element in the `api-manager.xml` file. For example,

```
<Environment type="production" api-console="false">
```

8. Start all the WSO2 API-M instances.
Make sure to start instance 1 first before starting the other two instances.

9. Sign in to the API Publisher in the first WSO2 API-M instance and choose to edit an API.

10. Navigate to the Manage tab, expand the API Gateways section.
11. Select both Gateways and Save and Publish the API in order to be able to publish to both the gateways.
12. Sign in to the API Store (of the first instance) and click the API to open it.

13. In the API's Overview tab, note that it has two sets of URLs for the two Gateway instances:
You have published an API to the API Stores through multiple Gateway environments.

If you have published your API through more than one Gateway,

When you have generated keys for the Applications, the sample cURL command shows how to generate an access token using the Password Grant type provides the Gateway URL of the first published Gateway Environments listed in API Publisher as shown in the step 10.

Generating Access Tokens

The following cURL command shows how to generate an access token using the Password Grant type.

```bash
curl -x -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://localhost:8244/token
```

In a similar manner, you can generate an access token using the Client Credentials grant type with the following cURL command.

```bash
curl -x -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://localhost:8244/token
```

Change this gateway URL according to the Gateway that you want to publish the API if you are using this cURL command to generate access tokens.

If you wish to use the API-M pack that you used as the first instance to try-out other tutorials, please ensure to delete the API Gateway configurations that you added in step 7, and uncomment the default `<Environment>` configurations in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

Block Subscription to an API

An API publisher blocks subscription to an API as a way of disabling access to it and managing its usage and monetization. A subscription blocking can be temporary or permanent. There is an unblocking facility to allow API invocations back.
You block APIs by subscriptions. That is, a given user is blocked access to a given API that s/he has subscribed to using a given application. If a user is subscribed to two APIs using the same application and you block access to only one of the APIs, s/he can still continue to invoke the other APIs that s/he subscribed to using the same application. Also, s/he can continue to access the same API subscribed to using different applications.

API level subscription blocking is useful to control only the subscriptions created for a specific API by a user. If you want to block all API requests from a specific application/user/specific IP address or to a specific API, you can use request blacklisting.

Blocking can be done at two levels:

- **Block production and sandbox access**: API access is blocked with both production and sandbox keys.
- **Block production access only**: Allows sandbox access only. This is useful when you want to fix and test an issue in an API. Rather than blocking all access, you can block production access only, allowing the developer to fix and test it.

When API Gateway caching is enabled (it is enabled by default), even after blocking a subscription, consumers might still be able to access APIs until the cache expires, which happens approximately every 15 minutes. Likewise, the API Gateway cache applies even when an API is unblocked.

See the following topics for the descriptions on the concepts that you need to know when you block subscriptions to an API:

- Applications
- Throttling
- Access tokens

1. Create two APIs.
   a. Sign in to the WSO2 API Publisher.
   b. Create two APIs by the names *TestAPI1* and *TestAPI2* and publish them to the WSO2 API Store.
   For more information, see Create and Publish an API.

   ![All APIs](image)

2. Subscribe to both the APIs.
   a. Sign in to the WSO2 API Store and click *APIS*.
      Note that the two APIs are visible in the APIs page.
   b. Subscribe to both APIs using the same application.
      You can use the default application or create your own.
c. Click the **View Subscriptions** button when prompted.
   
   The **Subscriptions** tab opens.

d. Click the **Production Keys** tab and click **Generate Keys** to create an application access token.
   
   If you have already generated an access token before, click **Re-generate** to renew the token.
3. Invoke both APIs using the access token you received in the previous step.

   **Format Example Response**

   The following is the cURL command format.

   ```
   curl -k -H "Authorization: Bearer <access_token>" '<API_URL>'
   ```

   ```
   curl -k -H "Authorization :Bearer 5fe40a53-2db6-3693-a9a9-bfccaaa5707d"  
   'https://localhost:8243/TestAPI1/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
   ```

   The placeholders mentioned in the format cURL command are replaced as follows:

   - `<access_token>` : Give the token generated in step 2 (d).
   - `<API_URL>` : Go to the API's Overview tab in the API Store and copy the production URL and append the payload to it.
You have subscribed to two APIs and invoked them successfully. Let's block one subscription and see the outcome.

4. Block an API:
   a. Sign back in to the API Publisher.
   b. Click **API** and click on the API that you need to block.
      In this case, click on the **TestAPI1** API.
   c. Click **Subscriptions** to navigate to the managed subscription section.
4. Click **Block**. Note that the **Block** link immediately turns to **Unblock**, allowing you to activate the subscription back at any time.

5. Invoke the APIs to test the blocked API.
   a. Sign back in to the API Store.
   b. Invoke the two APIs (TestAPI1 and TestAPI2) again as mentioned in step 3.

You might have to **regenerate the access token** for the respective application that you subscribed the APIs to (for example in this case it will be DefaultApplication) if the access token expiration time (1 hour by default) has passed since the last time you generated it.

Note that you can invoke TestAPI2 again, but when you invoke TestAPI1, it gives a message that the requested API is temporarily blocked. Neither the API publisher nor any subscriber can invoke the API until the block is removed.

When **Gateway caching** is enabled, which is the case by default, the subscription blocking will take place only after the token cache expires (the default token cache expiry time is 15min). However, if the token is regenerated after the API is blocked, then the API will be blocked immediately.

```
<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
    <ams:code>900907</ams:code>
    <ams:message>The requested API is temporarily blocked</ams:message>
    <ams:description>Access failure for API: /TestAPI1/1.0.0, version: 1.0.0 status: (900907) - The requested API is temporarily blocked</ams:description>
</ams:fault>
```

If you click **Applications** in the API Store, and select the application that you used to subscribe to the API, the details of the blocked subscription appears.
6. Unblock the API.
   a. Go back to the API Publisher.
   b. Click on the respective API.
      In this case click TestAPI1 1.0.0.
   c. Click Subscriptions and click Unblock corresponding to the respective subscription.
      Make sure to click on the subscription that corresponds to the correct Application.

   If you invoke TestAPI1 again, you will notice that you can invoke the API as usual.

   When Gateway caching is enabled, which is the case by default, the subscription unblocking will take place only after the token cache expires (the default token cache expiry time is 15min). However, if the token is regenerated after the API is unblocked, then the API will be unblocked immediately.

You have subscribed to two APIs, blocked subscription to one and tested that you cannot invoke the blocked API.

**Enforce Throttling and Resource Access Policies**

**Throttling** allows you to limit the number of hits to an API during a given period, typically to protect your APIs from security attacks and your backend services from overuse, regulate traffic according to infrastructure limitations and to regulate usage for monetization. For information on different levels of throttling in WSO2 API Manager (WSO2 API-M), see Throttling tiers.

This tutorial uses the PhoneVerification API, which has one resource, GET and POST methods to access it and a throttling policy enforced.

**Before you begin**, follow the Create and Publish an API to create and publish the PhoneVerification API and then the Subscribe to an API to subscribe to the API using the Bronze throttling tier.

After you created, published, and subscribed to the API, let's see how the API Gateway enforces throttling and resource access policies to the API.

1. Sign in to the API Store and select the PhoneVerification API.
   Make sure to subscribe to the API as mentioned in the before you begin section.
2. Go to the Default Application, click the **Production Keys** tab and generate an access token. If you already have an access token for the application, you have to regenerate it after 1 hour.
Let's invoke this API.

3. Click on the API, then go to its API Console tab.
4. Expand the GET method and click Try it out. Provide the required parameters and click Execute to invoke the API. For example,

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneNumber</td>
<td>E.g., 18006785432</td>
</tr>
<tr>
<td>LicenseKey</td>
<td>Give 0 for testing purpose</td>
</tr>
</tbody>
</table>

**GET** /CheckPhoneNumber
4. Note the response that appears in the API Console. As we used a valid phone number in this example, the response returns as valid.

5. Note that you subscribed to the API on the Bronze throttling tier. The Bronze tier allows you to make a 1000 calls to the API per minute. If you exceed your quota, you get a throttling error as shown below.

6. Let's try to invoke the API using an unavailable resource name.

5. Go to the API's **Overview** page in the API Store and get the API's URL.
6. Install cURL or any other REST client.
7. Go to the command-line invoke the API using the following cURL command.

   ```bash
   curl -k -H "Authorization :Bearer <access-token-in-step-3>" 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
   ``

   Note that the PhoneVerification API’s resource name is `CheckPhoneNumber`, but we use an undefined resource name as `CheckPhoneNum`. Here’s an example:

   ```bash
   curl -k -H "Authorization :Bearer 63cc9779d6557f4346a9a28b5cfd8b53" 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
   ``

   Note that the call gets blocked by the API Gateway with a 'no matching resource' message. It doesn't reach your backend services as you are trying to access a REST resource that is not defined for the API.

   ```xml
   ``

You have seen how the API Gateway enforces throttling and resource access policies for APIs.

### Change the Default Mediation Flow of API Requests

The API Gateway has a default mediation flow for the API invocation requests that it receives. You can extend this default mediation flow to do additional custom mediation for the messages in the API Gateway. An extension is provided as a synapse mediation sequence. You design all sequences using a tool such as the WSO2 API Manager Tooling Plug-in and then store the sequence in the Gateway’s registry.

Let’s see how to create a custom sequence using the WSO2 API Manager Tooling Plug-in and then deploy and use it in your APIs.

1. Sign in to the API Publisher.
2. Click **Add** to create an API with the following information and then click **Next: Implement >**.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>YahooWeather</td>
</tr>
<tr>
<td>Context</td>
<td>/weather</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Resources</td>
<td>URL pattern</td>
</tr>
<tr>
<td></td>
<td>current/{country}/{zipcode}</td>
</tr>
<tr>
<td>Request types</td>
<td>GET method to return the current weather conditions of a zip code that belongs to a particular country</td>
</tr>
</tbody>
</table>
3. The **Implement** tab opens. Select **Managed API**, provide the information given in the table below and click **Manage**.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP/REST endpoint</td>
</tr>
</tbody>
</table>
| Production endpoint  | You can find the Yahoo weather API's endpoint from [https://developer.yahoo.com/weather/](https://developer.yahoo.com/weather/). Copy the part before the '?' sign to get this URL: [https://query.yahooapis.com/v1/public/yql](https://query.yahooapis.com/v1/public/yql).
|                      | To verify the URL, click the **Test** button next to it. |
| Sandbox endpoint     | [https://query.yahooapis.com/v1/public/yql](https://query.yahooapis.com/v1/public/yql) To verify the URL, click the **Test** button next to it. |
4. Click **Next: Manage >** to go to the **Manage** tab, provide the following information and click **Save & Publish** once you are done.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tier Availability</td>
<td>Gold</td>
</tr>
</tbody>
</table>

Keep the default values for the other attributes
5. Download and install the WSO2 API Manager Tooling Plug-in by following one of the three possible methods described in Installing the API Manager Tooling Plug-In if you have not done so already. Start Eclipse by double clicking on the Eclipse application, which is inside the downloaded folder.

6. Navigate to the Window menu, click Perspective, Open Perspective, and Other to open the Eclipse perspective selection window.

7. On the dialog box that appears, click WSO2 API Manager and click OK.
8. On the APIM perspective, click the Login icon as shown below.

![Login Icon](image)

9. On the dialog box that appears, enter the URL, username, and password of the Publisher server.

![Add Registry](image)

10. On the tree view that appears, expand the folder structure of the existing API.

11. Right-click on the in sequence folder and click Create to create a new in sequence.

![Create](image)

This is because you want the custom sequence to be invoked in the In direction or the request path. If you want it to be involved in the Out or Fault paths, select the respective folder under customsequences.

**Tip:** If you prefer not to use the registry to upload the sequence or want to engage a sequence to all APIs in WSO2 API-M at once, you can do so by saving the mediation sequence XML file in the file system. See Adding Mediation Extensions for details.

12. Name the sequence YahooWeatherSequence.

13. Your sequence now appears on the Developer Studio console. From under the Mediators section, drag and drop a Property mediator to your sequence and give the following values to the property mediator.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>New Property</th>
</tr>
</thead>
</table>

The Property Mediator has no direct impact on the message, but rather on the message context flowing through Synapse. You can retrieve the properties set on a message later through the Synapse XPath Variables or the get-property() extension function. In this sequence, we are using two property mediators and set a Synapse XPath variable and a get-property() function to the two mediators respectively to retrieve the properties set to the message context during the execution.
<table>
<thead>
<tr>
<th>New Property Name</th>
<th>YQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value Type</td>
<td>Expression</td>
</tr>
<tr>
<td>Value Expression</td>
<td>For the XPath expression, we take a query part in the Yahoo API's endpoint (<a href="https://developer.yahoo.com/weather/">https://developer.yahoo.com/weather/</a>) and concatenate:</td>
</tr>
<tr>
<td></td>
<td><code>concat('?q=select%20*%20from%20weather.forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D%22', syn:get-property('uri.var.zipcode'), ',', syn:get-property('uri.var.country'), '%22)&amp;format=json')</code></td>
</tr>
<tr>
<td>Property Scope</td>
<td>Synapse</td>
</tr>
</tbody>
</table>

Note that the full URL of the Yahoo endpoint of is https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D%22nome%2C%20ak%22)ables.org%2Falltableswithkeys and we are extracting the query part (q=) from the endpoint highlighted and provide the zip code and country with concatenation.

Since this is a mediation level Property keep the Property Scope as Synapse. This is the default scope set when no Property Scope is defined.

14. Similarly, add another property mediator with the following values. This is an HTTP transport property that appends its value to the address.
endpoint URL. Once you are done, save the sequence.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>New Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Property Name</td>
<td>REST_URL_POSTFIX</td>
</tr>
<tr>
<td>Value Type</td>
<td>Expression</td>
</tr>
<tr>
<td>Value Expression</td>
<td>get-property('YQL')</td>
</tr>
<tr>
<td>Property Scope</td>
<td>Axis2</td>
</tr>
</tbody>
</table>

Since this is a transport level Property you need to set the Property Scope as **Axis2**.

15. Navigate to the **File** menu, and click **Save** to save the sequence.
16. Right-click on the sequence and click **Commit File** to push the changes to the Publisher server.

Alternatively, you can create a CAR file including the sequences and can deploy it in API Manager. For more information, see **Deploying Composite Applications in the Server**, which is in the WSO2 Administration Guide.

17. Sign in to the API Publisher again, select the API that you created earlier, and click the **Edit** link right next to its name to go to the edit wizard.
18. Navigate to the API's **Implement** tab, select the **Enable Message Mediation** check box and select the sequence that you created for the In flow. Next, click **Manage** and **Save & Publish** the API again.

**Tip**: It might take a few minutes for the sequence to be uploaded into the API Publisher. If it isn't there, please check again later.
19. Sign in to the API Store, subscribe to the API that you just published, and generate the access tokens in order to invoke the API.

20. Click the **API Console** tab of the API. It opens the integrated API Console using which you can invoke the API.
21. Give the following values for the parameters and invoke the API. You can also give any other value of your choice.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>usa</td>
</tr>
<tr>
<td>zipcode</td>
<td>95004</td>
</tr>
</tbody>
</table>

Note the response that you get as a JSON object from Yahoo.

```json
{
  "query": {
    "count": 1,
    "created": "2017-05-04T12:49:03Z",
    "lang": "en-US",
    "results": {
      "channel": {
        "units": {
          "distance": "mi",
          "pressure": "in",
          "speed": "mph",
          "temperature": "F"
        },
        "title": "Yahoo! Weather - Aromas, CA, US",
        "description": "Yahoo! Weather for Aromas, CA, US",
        "language": "en-us",
        "lastBuildDate": "Thu, 04 May 2017 05:49 AM PDT",
        "ttl": "60",
        "location": {
          "city": "Aromas",
```
"country": "United States",
"region": "CA",
"wind": {
  "chill": "50",
  "direction": "245",
  "speed": "4"
},
"atmosphere": {
  "humidity": "98",
  "pressure": "999.0",
  "rising": "0",
  "visibility": "7.5"
},
"astronomy": {
  "sunrise": "6:9 am",
  "sunset": "7:58 pm"
},
"image": {
  "title": "Yahoo! Weather",
  "width": "142",
  "height": "18",
  "link": "http://weather.yahoo.com",
  "url": "http://l.yimg.com/a/i/brand/purplelogo//uh/us/news-wea.gif"
},
"item": {
  "title": "Conditions for Aromas, CA, US at 05:00 AM PDT",
  "pubDate": "Thu, 04 May 2017 05:00 AM PDT",
  "condition": {
    "code": "33",
    "date": "Thu, 04 May 2017 05:00 AM PDT",
    "temp": "51",
    "text": "Mostly Clear"
  },
  "forecast": [
    {
      "code": "30",
      "date": "04 May 2017",
      "day": "Thu",
      "high": "74",
      "low": "55",
      "text": "Partly Cloudy"
    },
    {
      "code": "28",
      "date": "05 May 2017",
      "day": "Fri",
      "high": "71",
      "low": "53",
      "text": "Mostly Cloudy"
    },
    {
      "code": "30",
      "date": "06 May 2017",
      "day": "Sat",
      "high": "65",
      "low": "47",
      "text": "Partly Cloudy"
    },
    {
      "code": "12",
      "date": "07 May 2017",
      "day": "Sun",
      "high": "62",
      "low": "48",
      "text": "Rain"
In this tutorial, you created a sequence to change the default mediation flow of API requests, deployed it in the API Gateway and invoked an API using the custom mediation flow.

Please note that following mediators are not usable within custom sequences since they are not supported by API Gateway custom medications.
- Call mediator in non-blocking mode
- Send mediator

Map the Parameters of your Backend URLs with the API Publisher URLs

This tutorial uses the WSO2 API Manager Tooling Plug-in.

This tutorial explains how to map your backend URLs to the pattern that you want in the API Publisher. Note the following:

1. The URL pattern of the APIs in the Publisher is `http://<hostname>:8280/<context>/<version>/<API resource>.
2. You can define variables as part of the URI template of your API's resources. For example, in the URI template `/business/{businessId}/address` `{businessId}` is a variable.
3. The variables in the resources are read during mediation runtime using property values with the "uri.var." prefix. For example, this HTTP endpoint gets the businessId that you specify in the resource: `http://localhost:8280/businesses/{uri.var.businessId}/details`
4. The URI template of the API's resource is automatically appended to the end of the HTTP endpoint at runtime. You can use the following mediator setting to remove the URL postfix from the backend endpoint: `<property name="REST_URL_POSTFIX" scope="axis2" action="remove"/>`

We do the following mapping in this tutorial:

<table>
<thead>
<tr>
<th>API Publisher</th>
<th>Endpoint</th>
</tr>
</thead>
</table>

Before you begin, note that a mock backend implementation is set up in this tutorial for the purpose of demonstrating the API invocation. If you have a local API Manager setup, save this file in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder to set up the mock backend.

1. Log in to the API Publisher, design a new API with the following information, click Add and then click Next: Implement >.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>TestAPI</td>
</tr>
<tr>
<td>Context</td>
<td>/test</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Visibility</td>
<td>Public</td>
</tr>
<tr>
<td>Resources</td>
<td>URL pattern</td>
</tr>
<tr>
<td>Request types</td>
<td>/business/{businessId}/address/</td>
</tr>
</tbody>
</table>
2. The **Implement** tab opens. Give the information in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP/REST endpoint</td>
</tr>
<tr>
<td>Production</td>
<td><code>http://localhost:8280/businesses/{uri.var.businessId}/details</code></td>
</tr>
<tr>
<td>Sandbox endpoint</td>
<td><code>http://localhost:8280/businesses/{uri.var.businessId}/details</code></td>
</tr>
</tbody>
</table>
TestAPI: /test/1.0.0

3. Click Next: Manage > to go to the Manage tab, select the Gold tier and publish the API.
As the API's resource is appended to its endpoint by Synapse at runtime, let's write a custom sequence to remove this appended resource.

4. Copy the following to a text editor and save the file in XML format (e.g., TestSequence.xml).

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="TestSequence">
  <property name="REST_URL_POSTFIX" scope="axis2" action="remove"/>
</sequence>
```

5. Download and install the WSO2 API Manager Tooling Plug-in if you have not done so already. Open Eclipse by double clicking the Eclipse.app file inside the downloaded folder.

6. Click Window > Open Perspective > Other to open the Eclipse perspective selection window. Alternatively, click the Open Perspective icon shown below at the top right corner.
6. On the dialog box that appears, click WSO2 APIManager and click OK.

7. On the APIM perspective, click the Login icon as shown below.

8. On the APIM perspective, click the Login icon as shown below.
9. On the dialog box that appears, enter the URL, username and password of the Publisher server.

10. On the tree view that appears, expand the folder structure of the existing API.
11. Right-click on the **in** sequence folder and click **Import Sequence** to import the sequence you create above.

13. Your sequence now appears on the APIM perspective. Right-click on the imported sequence and click **Commit File** to push the changes to the Publisher server.
14. Log back into the API Publisher, click Edit and go to the Implement tab. Select the Enable Message Mediation check box and engage the In sequence that you created earlier.

TestSequence.xml removes the URL postfix from the backend endpoint, since the URI template of the API's resource is automatically appended to the end of the URL at runtime. Therefore the request URL is modified by adding this sequence to the In flow.

15. Save and Publish the API.
You have created an API. Let's subscribe to the API and invoke it.
16. Log in to the API Store and subscribe to the API.
17. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.
18. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. If you have already generated a token before, click **Re-generate** to renew the access token.
19. Click the **API Console** tab of your API.
20. Note that the `businessId` is added in the UI as a parameter. Give a `businessId` and click **Try it out** to invoke the API.

21. Note the response that you get. According to the mock backend used in this tutorial, you get the response **Received Request**.

In this tutorial, you mapped the URL pattern of the APIs in the Publisher with the endpoint URL pattern of a sample backend.

**Convert a JSON Message to SOAP and SOAP to JSON**

This tutorial uses the WSO2 API Manager Tooling Plug-in and the PhoneVerification API created in Create and Publish an API.

The API Gateway has a default mediation flow for the API invocation requests that it receives. You can extend this default mediation flow to do additional custom mediation for the messages in the API Gateway. An extension is provided as a synapse mediation sequence. You can design sequences using a tool such as the WSO2 API Manager Tooling Plug-in and then store the sequence in the Gateway’s registry.

Let’s see how to convert message types using custom sequences. In this tutorial, we convert a JSON payload to SOAP before sending it to a SOAP backend. Then we receive the response in SOAP and convert it back to JSON.

1. Log in to the API Publisher and click the PhoneVerification API.
2. Click the **Edit** icon to go to its edit mode.
3. Create the following resource and add it to the API.

**Tip:** The resource you create here invokes the SOAP 1.2 Web service of the backend. Therefore, the recommended method is HTTP POST. As you do not include the payload in a query string, avoid giving any specific name in the URL pattern, which will be amended to the actual backend URL.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>URL pattern</td>
</tr>
<tr>
<td></td>
<td>/*</td>
</tr>
<tr>
<td>Request types</td>
<td>POST</td>
</tr>
</tbody>
</table>

4. After the resource is added, expand it and edit the parameter as follows. This parameter is used to pass the payload to the backend.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Parameter Type</th>
<th>Data Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td>Pass the phone number and license key</td>
<td>body</td>
<td>string</td>
<td>True</td>
</tr>
</tbody>
</table>
Next, let's write a sequence to convert the JSON payload to a SOAP request. We do this because the backend accepts SOAP requests.

5. Navigate to the Implement page and change the endpoint of the API to http://ws.cdyne.com/phoneverify/phoneverify.asmx?WSDL. Once the edits are done, click Save.

6. Download and install the WSO2 API Manager Tooling Plug-in if you have not done so already. Open Eclipse by double clicking the Eclipse.ap file inside the downloaded folder.

7. Click Window > Open Perspective > Other to open the Eclipse perspective selection window. Alternatively, click the Open Perspective icon shown below at the top right corner.

8. On the dialog box that appears, select WSO2 API Manager and click OK.
9. On the API-M perspective, click the Login icon as shown below.

10. On the dialog box that appears, enter the URL, username and password of the Publisher server.

11. On the tree view that appears, expand the folder structure of the existing API.
12. Right-click on the In sequence folder and click Create to create a new In sequence.
13. Name the sequence `JSONtoSOAP`.

14. Your sequence now appears on the APIM perspective. From under the Mediators section, drag and drop a PayloadFactory mediator to your sequence and give the following values to the mediator.

**Tip:** The PayloadFactory mediator transforms the content of your message. The `<args>` elements define arguments that retrieve values at runtime by evaluating the provided expression against the SOAP body. You can configure the format of the request/response and map it to the arguments.

For example, in the following configuration, the values for the format parameters PhoneNumber and LicenseKey will be assigned with values that are taken from the `<args>` elements (arguments,) in that particular order.

For details on how you got this configuration, see PayloadFactory Mediator in the WSO2 ESB documentation.
15. Similarly, add a **Property** mediator to the same sequence and give the following values to the property mediator. This mediator changes the payload type of the outgoing message to soap+xml. More information about the Property mediator can be found [here](#).
16. Save the sequence, which is in XML format (e.g., JSONtoSOAP.xml). This will be the In sequence for your API. Next, create an Out sequence.

17. Right-click on the Out sequence folder and click **Create** to create a new Out sequence.

18. Name the sequence **SOAPtoJSON**.
19. Add a **Log** mediator to the sequence and give the following values. Note that the property value provided is a string literal.

<table>
<thead>
<tr>
<th>Log Category</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Level</td>
<td>CUSTOM</td>
</tr>
<tr>
<td>Log Separator</td>
<td>,</td>
</tr>
</tbody>
</table>
| Properties   | Name: TRACE  
|              | Type: LITERAL  
|              | Value/Expression: Global Mediation Extension |

The **Log mediator** is used to log mediated messages. Having a custom log level allows logging only the properties added to the Log mediator configuration. More information can be found [here](#).

20. Similarly, add a **PayloadFactory** mediator with the following values. This mediator in the out sequence is used to transform the SOAP message content returned from the backend into JSON.

<table>
<thead>
<tr>
<th>Payload Format</th>
<th>Inline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td><code>&lt;CheckPhoneNumber xmlns=&quot;http://ws.cdyne.com/PhoneVerify/query&quot;&gt;\&lt;PhoneNumber&gt;$1&lt;/PhoneNumber&gt;\&lt;LicenseKey&gt;$2&lt;/LicenseKey\&gt;&lt;/CheckPhoneNumber&gt;</code></td>
</tr>
</tbody>
</table>
| Args | Type: Expression  
|      | Value: //request/PhoneNumber  
|      | Evaluator: xml  
|      | Type: Expression  
|      | Value: //request/LicenseKey  
|      | Evaluator: xml  
| Media Type | xml  

21. Finally, add a **Property** mediator with the following values. This mediator changes the payload type of the incoming message to json.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>messageType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Action</td>
<td>set</td>
</tr>
<tr>
<td>Value Type</td>
<td>Literal</td>
</tr>
<tr>
<td>Property Data Type</td>
<td>String</td>
</tr>
<tr>
<td>Value</td>
<td>application/json</td>
</tr>
<tr>
<td>Value String Capturing Group</td>
<td>0</td>
</tr>
<tr>
<td>Property Scope</td>
<td>axis2</td>
</tr>
</tbody>
</table>

22. Save the sequence, which is in XML format (e.g., SOAPtoJSON.xml). This will be the **Out** sequence for your API.

23. Click the **Push all changes to the server** icon shown below to commit your changes to the Publisher server.
24. Log back in to the API Publisher, click the **Edit** link associated with the API and navigate to the **Implement** tab. Select the **Enable Message Mediation** check box and engage the **In** and **Out** sequences that you created earlier.

**Message Mediation Policies**

Enable Message Mediation

**In Flow**

- **JSONtoSOAP**
  - Upload In Flow

**Out Flow**

- **SOAPtoJSON**
  - Upload Out Flow

**Fault Flow**

- **None**

**JSONtoSOAP** in sequence will serve the purpose of transforming the JSON payload to SOAP before sending it to the SOAP backend. **SOAPtoJSON** out sequence will transform the SOAP message returned from the backend to JSON.

25. **Save** the API.

You have created an API, a resource to access the SOAP backend and engaged sequences to the request and response paths to convert the message format from JSON to SOAP and back to JSON. Let's subscribe to the API and invoke it.

26. Log in to the API Store and subscribe to the API and create an access token if you have not done so already.
27. Go to the API Console tab and expand the POST method.
28. Give the payload in the body parameter in JSON format and click **Try it out**. Here's a sample JSON payload:
   
   ```json
   {"request":{"PhoneNumber":"18006785432","LicenseKey":"0"}}
   ```

29. Note that you get a JSON response to the JSON request whereas the backend accepts SOAP messages. The request and response are converted by the sequences that you engaged.
In this tutorial, you converted a message from JSON to SOAP and back to JSON using **In** and **Out** sequences.

### Invoke an API using a SOAP Client

You can use any SOAP client to **invoke an API**. We use the SOAP UI in this example.

See the following topics for a description of the concepts that you need to know when invoking an API:

- Applications
- Throttling
- Access tokens

The examples here use the **PhoneVerification** API, which is created in section **Create and Publish an API**.

Let's invoke the **PhoneVerification** API using a SOAP client.

1. Sign in to the API Store and click an API that you want to invoke (e.g., **PhoneVerification**).
2. The API's **Overview** page opens. Select an application (e.g., **DefaultApplication**), the **Bronze tier** and subscribe to the API.

3. Click the **Applications** menu, open the default application using which you subscribed to the API, and generate a production key.
4. Copy the access token to the clipboard as you need it later to invoke the API.

   Make sure the flash plug-in of your web browser is updated in order to get the **copy button** in the Store UI working.

5. Download the SOAP UI installation that suits your operating system from [https://www.soapui.org/downloads/soapui.html](https://www.soapui.org/downloads/soapui.html) and open its console.
6. In the SOAP UI, right click on the **Projects** menu and create a new SOAP project.
7. Give your API's WSDL and click OK. In this case, the WSDL is as follows: http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl

8. The WSDL defines two operations. Let's work with CheckPhoneNumber. Double click on Request 1. Then, click the Header tab and add an authorization header to your request by clicking on the add icon.
8. Give the value of the Authorization header in the following format: 
**Bearer <the-access-token-you-copied-in-step4>**

9. Add the following dummy values and submit the request.
   - Change the endpoint with the production URL of the API.
     You can copy the production URL from the API's Overview tab in the API Store (i.e., It is https://localhost:8243/phoneverify/ for the PhoneVerification API). Append the resources to the end of the URL, if any. The resource is /PhoneVerification/CheckPhoneNumber or the API that we use here.
   - In the SOAP request, change the parameters, which are PhoneNumber and LicenseKey.
     - **PhoneNumber**
       Give a dummy phone number E.g., 123456
     - **LicenseKey**
       Give 0 for testing purpose

10. Add the following dummy values and submit the request.
    a. Change the endpoint with the production URL of the API.
    You can copy the production URL from the API's Overview tab in the API Store (i.e., It is https://localhost:8243/phoneverify/ for the PhoneVerification API). Append the resources to the end of the URL, if any. The resource is /PhoneVerification/CheckPhoneNumber for the PhoneVerification API that we use here.
    b. In the SOAP request, change the parameters, which are PhoneNumber and LicenseKey.
        - **PhoneNumber**
          Give a dummy phone number E.g., 123456
        - **LicenseKey**
          Give 0 for testing purpose
Note the result on the right-hand side panel. As you gave a dummy phone number in this example, you get the result as invalid.

11. Add the following values and submit the request:
   a. Maintain the same endpoint that you used in the previous step.
   b. In the SOAP request, change the parameters, which are PhoneNumber and LicenseKey.

<table>
<thead>
<tr>
<th>PhoneNumber</th>
<th>LicenseKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>Give a proper phone number E.g., 18006785432</td>
<td>Give 0 for testing purpose</td>
</tr>
</tbody>
</table>

Note the result on the right-hand side panel. As you gave a valid phone number in this example, you get the result as valid.
You have invoked an API using a SOAP client.

You can treat the Admin Services APIs as if they were back-end server APIs, and get all the benefits of API management for the admin services.

Do the following WSO2 API Manager to expose SOAP APIs with OAUTH2.0.

1. Create a SOAP API for an admin service in the Publisher.
2. Add backend admin credentials for basic auth in the Implement page.
3. Publish the app with a scope protecting access - use the admin role for your scope.

Admins can then subscribe to the app and invoke it using OAUTH2.0 security.

Create and Publish an API from a Swagger Definition

A Swagger definition is a format that describes REST APIs. In this tutorial, you create and publish an API in WSO2 API Manager using a Swagger definition for an existing API.

1. Sign in to the WSO2 API Publisher.
   https://<hostname>:9443/publisher
2. In the APIS menu, click Add New API.
3. Select I Have an Existing API. Click Swagger URL and give http://petstore.swagger.io/v2/swagger.json as the URL. Click Start Creating.

Let's get started!

Add New API

- **I Have an Existing API**
  - Use an existing APIs endpoint or the API Swagger definition to create an API.
  - **Swagger URL**
    - http://petstore.swagger.io/v2/swagger.json

- **I Have a SOAP Endpoint**
  - Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

- **Design a New REST API**
  - Design and prototype a new REST API.

- **Design a New Websocket API**
  - Design and prototype a new Websocket API.

4. Edit the information as given below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Petstore</td>
</tr>
<tr>
<td>Context</td>
<td>/petstore</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>
Design API

5. Notice that all the **API resources** are created automatically when the Swagger URL is specified.

6. Click **Edit Source** to edit the Swagger file and remove security headers. This is required to invoke the API in the Store using the Swagger UI.
7. Remove the security tag from the `/pet` POST resource given below. This is required to enable API invocation using the API (store) console.

```java
//remove the following code snippet
security:
- petstore_auth:
  - 'write:pets'
  - 'read:pets'
```

8. Remove the security `/pet/{petId}` GET resource given below:

```java
//remove the following code snippet
security:
- api_key: []
```

9. After removing the security tags, click Apply Changes to save the changes.

```java
{} Swagger Editor
File ▼ Edit ▼ Generate Server ▼ Generate Client ▼
```

Then, click Next: Implement.

10. Click the Managed API option. Enter the information shown below and click Next: Manage.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP/REST endpoint</td>
</tr>
<tr>
<td>Production endpoint</td>
<td><a href="http://petstore.swagger.io/v2/">http://petstore.swagger.io/v2/</a></td>
</tr>
<tr>
<td>Sandbox endpoint</td>
<td>Providing only the production endpoint is sufficient.</td>
</tr>
</tbody>
</table>
11. Select the options described in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transports</td>
<td>HTTP and HTTPS</td>
<td>The transport protocol on which the API is exposed. Both HTTP and HTTPS transports are selected by default. If you want to limit API availability to only one transport (e.g., HTTPS), un-check the other transport.</td>
</tr>
<tr>
<td>Subscription Tiers</td>
<td>Gold, Silver</td>
<td>The API can be available at different levels of service. They allow you to limit the number of successful hits to an API during a given period of time.</td>
</tr>
</tbody>
</table>
12. Click **Save & Publish**.

Now you have successfully published an API using a Swagger definition.

Let's invoke the API from the API Store.

**Invoking the API**

1. Sign in to the WSO2 API Store.
   
   https://<hostname>:9443/store
2. Select the **PetStore** API to open it.
3. Subscribe to the API using the Default Application and the Gold tier.

4. Click **View Subscriptions** to view all the APIs to which the Default Application has subscribed.
5. Click the Production Keys tab and generate a token.

6. Navigate to the API Console for the PetStore API.
7. Expand the POST resource, click Try it out, and give the following example as the request body. Click Execute and view the response.

```json
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "Dogs"
  },
  "name": "Rover",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "lion_shepard"
    }
  ],
  "status": "available"
}
```

```json
{
  "id": 9123612807670061000,
  "category": {
    "id": 0,
    "name": "Dogs"
  },
  "name": "Rover",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "lion_shepard"
    }
  ],
  "status": "available"
}
```

You have successfully invoked the Petstore API.
Create a WebSocket API

WebSocket is a protocol similar to HTTP that is part of the HTML5 specification. It enables simultaneous two-way communication (full-duplex communication) between the client and the server over a single connection. The WebSocket protocol is designed to achieve the following:

- Reduce unnecessary network traffic and latency
- Allow streaming through proxies and firewalls while simultaneously supporting upstream and downstream communication
- Be backward compatible with the pre-WebSocket world by starting up as an HTTP connection before switching to WebSocket frames

A WebSocket API allows an API creator to expose a WebSocket backend as an API to offer services via a WebSocket protocol while providing OAuth, security, throttling, analytics, etc.

In this tutorial, you create and publish an API with a WebSocket backend and then invoke it using a Netty-based WebSocket client. You can use any WebSocket client to invoke the API.

1. Sign in to the API Publisher.
   https://<hostname>:9443/publisher (e.g. https://localhost:9443/publisher). Use admin as the username and password.
2. In the APIS menu, click Add New API.
3. Select the option to design a new WebSocket API and click Start Creating.
4. The Design tab of the API opens. Give the information in the table below and click Next: Implement > to proceed to the implementation phase.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>EchoWebSocket</td>
</tr>
<tr>
<td>Context</td>
<td>/echowebsocket</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
</tbody>
</table>
4. Click the Managed API option.

5. Provide the production endpoint and sandbox endpoint, which is ws://echo.websocket.org:80 in this example, and click Next: Manage >.

With WSO2 API Manager, you can maintain a production and a sandbox endpoint for a given API. The production endpoint is the actual location of the API, whereas the sandbox endpoint points to its testing/pre-production environment.

The **Test** button for the production and sandbox endpoints does not work for WebSocket APIs and is a known issue.
7. In the Manage tab, select the Gold tier, scroll down and click Save & Publish.

You have now published the WebSocket API to the API Store. Let's subscribe to it.

8. When prompted, choose to open the newly published API in the API Store.

9. The EchoWebSocket API opens. Select an application (e.g., DefaultApplication), the Gold tier and subscribe to the API.
10. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.
11. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. If you have already generated keys before, click **Regenerate**.

You can also add a **Callback URL** if you have not added it already when creating the API. You have now subscribed to an API in the API Store and can invoke it using a WebSocket client. In this tutorial, you invoke it using a Netty-based WebSocket client.

12. In your client application, set the WebSocket API URL as shown in the API Store.
13. In this example, make sure that the URL in the sample-ws-client/src/main/java/io/netty/example/http/websocketx/client/WebSocketClient.java file matches the one in the API Store.
14. In the same file, copy and paste the Authorization Bearer access token you generated in step 11 as shown below.
15. Save your changes.
16. Open the sample-ws-client directory you downloaded in step 11 using an IDE. This tutorial uses IntelliJ Idea 15 CE as the IDE.
17. Run the WebSocket client as shown below.
18. Type a message in the WebSocket client and you will see that it echoes the message as intended by the WebSocket API.
Create a Prototyped API with an Inline Script

In this tutorial, you create a prototyped API with an inline script, deploy it as a prototype, and invoke it using the API Console integrated in the API Store. Typically, you create APIs with inline scripts for testing purposes. An API prototype is created for the purpose of early promotion and testing. You can deploy a new API or a new version of an existing API as a prototype. It gives subscribers an early implementation of the API that they can try out without a subscription or monetization, and provide feedback to improve. After a period of time, publishers can make changes that the users request and publish the API.

1. Sign in to the API Publisher using admin as the username and password.

2. Select the option to design a new REST API and click Start Creating.

Let's get started!

Add New API

- I Have an Existing API
  Use an existing API's endpoint or the API Swagger definition to create an API.

- I Have a SOAP Endpoint
  Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

- Design a New REST API
  Design and prototype a new REST API.

Show More Options

Start Creating

3. Give the information in the table below. To add resources, click the Add button. Since the URL Pattern used here is a variable, it is denoted within curly braces.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Location_API</td>
</tr>
<tr>
<td>Context</td>
<td>/location</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Resources URL pattern</td>
<td>{town}</td>
</tr>
<tr>
<td>Request types</td>
<td>GET</td>
</tr>
</tbody>
</table>
3. After the resource is added, expand its GET method and note that a parameter by the name `town` is added under the resource. You use it to pass the payload to the backend. Once done, click Next: Implement >.

4. To specify multiple parameters in the API resource, separate the parameters with a forward slash.

   `{param1}/param2`

5. In the Prototyped API section under the Implement tab, select the implementation method as Inline.
6. Expand the `GET` method and give the following as the script. It reads the payload that the user sends with the API request and returns it as a JSON value. The value `mc` is the message context.

   ```javascript
   mc.setProperty('CONTENT_TYPE', 'application/json');
   var town = mc.getProperty('uri.var.town');
   mc.setPayloadJSON('{ "Town" : "'+town+'"}');
   ```

7. Click **Deploy as a Prototype**.
8. When prompted, choose to open the newly published API in the API Store.

   **Tip**: You can invoke prototyped APIs without signing in to the API Store or subscribing to the API. The purpose of a prototype is advertising and giving an early implementation for users to test.

9. The Location API opens. Click the **API Console** tab.
10. Expand the GET method, click **Try it out**. Give any value for the town (e.g. London) and click **Execute** to invoke the API.

11. Note the payload you gave as a JSON output in the response.

You have created an API with inline script, deployed it as a prototype and invoked it through the integrated API Console.
An API can also be prototyped by moving the API to the prototyped state in the API lifecycle. For more information, see the Deploy and Test as a Prototype tutorial.

**Related Tutorials**
- Create and Publish an API
- Create a WebSocket API
- Create and Publish an API from a Swagger definition

**Pass a Custom Authorization Token to the Backend**

This tutorial uses the WSO2 API Manager Tooling Plug-in.

When you send an API request to the backend, you pass a token in the Authorization header of the request. The API Gateway uses this token to authorize access, and then drops it from the outgoing message. If you wish to use a different (or a custom generated) authorization token than the application generated access token, you can use it as a token exchange mechanism in mediation logic of the API. In this tutorial, we explain how to pass a custom authorization token that is different to the authorization token generated for the application.

**In this tutorial**, you have a sample JAX-RS backend and it always expects 1234 as the authorization token. In your API request, you pass the token that is generated in the Authorization header, and 1234 in a Custom header. The mediation extension you write extracts the value of the Custom header, and sets it as the Authorization header before sending it to the backend.

Here's a summary:

Client (headers: Authorization, custom) -> Gateway (drop: Authorization, convert: custom->Authorization) -> Backend

Let's get started.

1. Download and install the WSO2 API Manager Tooling Plug-in if you have not done so already. Open Eclipse by double clicking the Eclipse.app file inside the downloaded folder.
2. Click Window > Open Perspective > Other to open the Eclipse perspective selection window. Alternatively, click the Open Perspective icon shown below at the top right corner.

3. On the dialog box that appears, click WSO2 APIManager and click OK.
4. On the APIM perspective, click the **Login** icon as shown below.

![Login icon](image)

5. On the dialog box that appears, enter the URL, username and password (by default `admin`) of the Publisher server.

![Login dialog](image)

6. On the tree view that appears, expand the folder structure of the existing API.

7. Right-click on the `in` sequence folder and click **Create** to create a new `in` sequence.
8. Name the sequence **TokenExchange**.

9. Your sequence now appears on the APIM perspective. From under the **Mediators** section, drag and drop a **Property** mediator to your sequence and give the following values to the mediator.

   **Tip**: The **Property Mediator** has no direct impact on a message, but rather on the message context flowing through **Synapse**. For more information, see **Property Mediator** in the WSO2 EI documentation.

The following property mediator is used to assign the Custom transport level property to another property called **Custom**.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>New Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Property Name</td>
<td>Custom</td>
</tr>
<tr>
<td>Value Type</td>
<td>EXPRESSION</td>
</tr>
<tr>
<td>Value Expression</td>
<td><code>get-property('transport', 'Custom')</code></td>
</tr>
</tbody>
</table>
10. Similarly, add another Property mediator to your sequence and give the following values to the mediator. This property mediator is used to construct a transport level property called Authorization and assign itself the value of the Custom property created above.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Property Name</td>
<td>Authorization</td>
</tr>
<tr>
<td>Value Type</td>
<td>EXPRESSION</td>
</tr>
<tr>
<td>Value Expression</td>
<td>get-property('Custom')</td>
</tr>
<tr>
<td>Property Scope</td>
<td>transport</td>
</tr>
</tbody>
</table>
11. Add a third **Property** mediator to your sequence and give the following values to the mediator. This property mediator is used to remove the **Custom** property from the transport level.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Property Name</td>
<td>Authorization</td>
</tr>
<tr>
<td>Property Action</td>
<td>set</td>
</tr>
<tr>
<td>Value Type</td>
<td>EXPRESSION</td>
</tr>
<tr>
<td>Value Expression</td>
<td>get-property('Custom')</td>
</tr>
<tr>
<td>Property Scope</td>
<td>transport</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
12. Save the sequence.
13. Right-click on the sequence and click **Commit File** to push the changes to the Publisher server.
Let's create a new API and engage the sequence you created to it.

14. Log in to the API Publisher, click the Add link and give the information in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>TestAPI1</td>
</tr>
<tr>
<td>Context</td>
<td>/test1</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Visibility</td>
<td>Public</td>
</tr>
</tbody>
</table>

15. Leave the Resources section blank, and click Next: Implement >. Add a wildcard resource (/*) when prompted. Click Next: Implement > again to move to the Implement tab.
16. The Implement tab opens. Give the information in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP endpoint</td>
</tr>
<tr>
<td>Production endpoint</td>
<td><a href="http://wso2cloud-custom-auth-header-sample-1-0-0.wso2apps.com/custom-auth-header/validate-header">http://wso2cloud-custom-auth-header-sample-1-0-0.wso2apps.com/custom-auth-header/validate-header</a></td>
</tr>
<tr>
<td>Sandbox endpoint</td>
<td><a href="http://wso2cloud-custom-auth-header-sample-1-0-0.wso2apps.com/custom-auth-header/validate-header">http://wso2cloud-custom-auth-header-sample-1-0-0.wso2apps.com/custom-auth-header/validate-header</a></td>
</tr>
</tbody>
</table>

TestAPI: /test/1.0.0

17. Select the Enable Message Mediation check box, engage the In sequence that you created earlier and click Manage.
In the Manage tab, select the Gold tier and click Save and Publish to publish the API to the API Store.

Let's subscribe to the API and invoke it.

Log in to the API Store and subscribe to the API using an available application and the Gold tier. If there are no applications available by default, create one.
20. Click the View Subscriptions button when prompted. The Subscriptions tab opens.

21. Click the Production Keys tab and click Generate Keys to create an application access token.

22. Install any REST client in your machine. We use cURL here.

23. Go to the command line, and invoke the API using the following cURL command. In this command, you pass the token that the backend expects, i.e., 1234, in the Custom header with the authorization token that the system generates in the Authorization header.

```
curl -H "Authorization: Bearer <access token>" -H "Custom: Bearer 1234" <API URL>
```

Note the following:
- `<access token>` is the token that you got in step 20.
- `<API URL>` appears on the API’s Overview page in the API Store. Copy the HTTP endpoint. If you select the HTTPS endpoint, be sure to run the cURL command with the -k option.

Here’s an example:

```
curl -k -H "Authorization: Bearer 2e25097b2b3fbbfb44f5642fa8a495a1" -H "Custom: Bearer 1234"
https://localhost:8243/test/1.0.0
```

24. Note the response that you get in the command line. According to the sample backend used in this tutorial, you get the response as “Request Received.”

```
$ curl -k -H "Authorization: Bearer 2e25097b2b3fbbfb44f5642fa8a495a1" -H "Custom: Bearer 1234" https://localhost:8243/test/1.0.0
<Response><code>200</code><message>Request Received</message><description>Request Received successfully</description></Response>
```

In this tutorial, you passed a custom token that the backend expects along with the system-generated Authorization token, and invoked an API successfully by swapping the system’s token with your custom token.

**Create and Publish a SOAP API**

WSO2 API Manager supports the management of both REST and SOAP APIs.
In this tutorial, we create and publish an **API with a SOAP endpoint** and then invoke it using the integrated API Console.

See the following topics for a description of the concepts that you need to know when invoking an API:

- Applications
- Throttling
- Access tokens

Let's get started.

1. Log in to the API Publisher and click **ADD NEW API**.
2. Select the option to design an API with an existing SOAP endpoint, select **Pass Through**, give the endpoint URL and click **Start Creating**.

This example uses the WSDL `http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl` from CDYNE as the endpoint here, but you can use any SOAP backend of your choice.

3. The **Design** tab of the API opens. Give the information in the table below and click **Next: Implement** > to proceed to the implementation phase.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SoapTest</td>
</tr>
<tr>
<td>Context</td>
<td>/soapTest</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
</tbody>
</table>
4. Click the **Managed API** option.
5. Select **HTTP/SOAP Endpoint**, provide the production endpoint (in this example, `http://ws.cdyne.com/phoneverify/phoneverify.asmx`), and click **Manage**.

6. In the **Manage** tab, select the **Gold** tier, scroll down and click **Save & Publish**.

If you wish to add scopes to the resources that were created, click **Add Scopes** and specify the scopes you want to add. If you specify a scope, you need to use the same scope when generating access tokens for the subscribed application and when invoking the API.

For more information on working with the scopes, see [OAuth scopes](#).
You have now published the SOAP API to the API Store. Let's subscribe to it.

7. When prompted, choose to open the newly published API in the API Store.
8. The SoapTest API opens. Select an application (e.g., DefaultApplication), the **Gold** tier and subscribe to the API.

You have now published the SOAP API to the API Store. Let's subscribe to it.

9. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.
10. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. If you have already generated keys before, click **Regenerate**. By default, access tokens expire an hour after creation, unless you change the expiration time.
Let's invoke the API.

11. On the APIs menu, select the API and click the API Console tab.

12. Expand the POST method and click Try it out. Enter the following, and click Execute to invoke the API.
13. Note the API response that appears on the console.
In this tutorial, you created an API with a SOAP backend and invoked it using the integrated Swagger API Console.

You can also invoke this API using a third-party tool such as SOAP UI. For more information on how to invoke an API using a SOAP client, see [Invoke an API using a SOAP Client](#).

### Disable Message Chunking

When processing large messages, message chunking facilitates sending the message as multiple independent chunks. Message chunking is set using the `Transfer-Encoding: chunked` header. However, some legacy backends might not support chunked messages. To disable sending chunked messages to the backend for a specific API, follow the steps below:

1. Create an XML file with the following content:

   ```xml
   <sequence xmlns="http://ws.apache.org/ns/synapse" name="disableChunkingSeq">
     <property name="DISABLE_CHUNKING" value="true" scope="axis2"/>
   </sequence>
   ``

2. Use the same sequence and apply it as a mediation extension to the inflow of this particular API. For more details, see [Creating per-API extensions](#). Once the API is published, chunking is disabled for the message that is sent to the backend.

   To stop chunked messages from being sent to the client, you can apply the same mediation extension to the out sequence as well.

### Enable API Indexing on Remote Publisher and Store Nodes

WSO2 API Manager uses Solr indexing to display API details on API Publisher and Store. In the tenant mode, the indexing process starts when a user of a tenant logs in to the Store. However, when there are large numbers of APIs, the indexing process can take a significant time and any new APIs may not be visible in Store nodes until the indexing process finishes. Therefore, in order to avoid the latter mentioned delay, you can start the API indexing for tenants on a remote Publisher/Store nodes as follows:
Starting API indexing on remote Publisher/Store nodes depends on Hazelcast clustering, therefore; clustering needs to be enabled in the Publisher/Store nodes and they should be in the same clustering domain.

1. Enable Hazelcast clustering for Publisher and Store nodes. Make sure that all Publisher and Store nodes are in the same cluster domain and the nodes are properly joined to the cluster.
2. Open `<API-M_HOME>/bin/wso2server.sh` file.
3. Add the Java System Property `enableTenantLoadNotification=true` at the bottom of the file:

   ```
   ....
   -DworkerNode=false \ 
   -DenableTenantLoadNotification=true \ 
   org.wso2.carbon.bootstrap.Bootstrap $*
   status=$?
   done
   ```

4. Repeat this on all Publisher and Store nodes, and restart them.

Now, when a tenant user logs in to a particular Publisher/Store the other Publisher/Store nodes are notified. Therefore, the indexing jobs will start immediately.

### Remove Specific Request Headers From Response

Web services may contain response headers with sensitive information. This tutorial explains how to remove HTTP request headers from the responses for security reasons.

To remove request headers from responses for per API or globally, add the name of the header to be removed as a property in your custom out sequence.

**Template Example**

```
<property name="<name of the header to be removed>" scope="transport" action="remove"/>
```

```
<property name="Accept" scope="transport" action="remove"/>
<property name="X-JWT-Assertion" scope="transport" action="remove"/>
<property name="Cookie" scope="transport" action="remove"/>
```

**For details on adding custom sequences per API, see Creating per-API extensions.**

**For details on adding custom sequences to all APIs of a specific tenant at once, see Creating global extensions.**

To handle error responses, follow the instructions below.

1. To address the scenario where the API does not exist, open the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/main.xml` file.
2. Add the name of the header to be removed as a property, just before the beginning of `send` mediator, as shown below

   **Template Example**

   ```
   <property name="<name of the header to be removed>" scope="transport" action="remove"/>
   ```

   ```
   <property name="Accept" scope="transport" action="remove"/>
   <property name="X-JWT-Assertion" scope="transport" action="remove"/>
   <property name="Cookie" scope="transport" action="remove"/>
   <send/>
   ```

3. To address the scenario where an error occurs during execution of API requests, open the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/fault.xml` file.
4. Add the name of the header to be removed as a property, just before the beginning of "CORS request handler" sequence, as shown below.

   **Template Example**
1. The swagger definition of the API needs to be changed in order to achieve this. Define the "x-wso2-security" key will define a scope for the API. The value should be nested under "x-wso2-security". Note that the role should be an already available role in the system. To define a role see Adding and Managing Users and Roles. Find the sample Swagger definition below for reference.

```json
"x-wso2-security": {
   "apim": {
      "x-wso2-scopes": [
         {
            "name": "scope1",
            "description": "scope 1",
            "key": "scope1",
            "roles": ["role1"]
         }
      ]
   }
}
```

2. The swagger definition of the API needs to be changed inorder to achieve this. Define the "x-scope" key under the resource to attach a scope to a resource. The value to this key should be the scope name. Note that the scope should be an already defined. Find the sample Swagger definition below for reference.

```json
"x-wso2-security": {
   "apim": {
      "x-wso2-scopes": [
         {
            "name": "scope1",
            "description": "scope 1",
            "key": "scope1",
            "roles": "role1"
         }
      ]
   }
}
```
"paths": {"/CheckPhoneNumber":
    "get": {
        "x-auth-type": "Application",
        "x-throttling-tier": "Unlimited",
        "x-scope": "scope1",
        "responses":{"200":{"description": "OK"}}
    }
}

Once you have done the changes to the swagger definition of the API as above, you can use this swagger to create the API as described in Create and Publish API from a Swagger Definition.

Generate REST API from SOAP Backend

This feature allows users to expose their legacy SOAP backends as REST APIs through WSO2 API Manager. WSO2 API Manager supports WSDL 1.1 based SOAP backends.

The following instructions explain how to generate REST APIs in WSO2 API Manager for an existing SOAP backend.

Before you begin...

Make sure that you have a valid WSDL URL from the SOAP backend. It should belong to the WSDL 1.1 version.

1. Sign in to the API Publisher and click ADD NEW API, which is under the API menu.
2. Select I Have a SOAP Endpoint. You see the following two options to create APIs for SOAP backend.
   - Pass Through – Create a pass through proxy for SOAP requests coming to the API Gateway.
   - Generate REST APIs – This option is used to generate REST API definitions from the given WSDL URL.

Select WSDL URL to provide a URL for your API. Alternatively, you can upload a WSDL archive/file

3. Select Generate REST APIs (The default option is Pass Through). Provide the WSDL URL given below for the SOAP backend and click Start Creating.

Sample WSDL URL

http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl

The WSDL URL and WSDL Archive/File options that are shown in the following screenshot, which allow you to modify API resources, are only available as a WUM update and is effective from 11th January 2019 (11-01-2019). For more information on updating WSO2 API Manager, see Updating WSO2 Products.
3. Click on Edit Source to edit the Swagger specification of the API.

The automatically generated REST resources of the API appears. You can describe your REST API better by modifying the Open API Specification that you see on this page. The following is a sample of how this particular API has been described.

4. Optionally, update the Swagger definition of your API and click Apply Changes to save your API.

Let’s update the API by replacing the existing Swagger definition with the following Swagger definition.
Expand to see the edited Swagger definition...

```yaml
swagger: "2.0"
paths:
  /checkPhoneNumber:
    get:
      operationId: CheckPhoneNumber
      parameters:
        - in: query
          name: PhoneNumber
          required: true
          type: string
        - in: query
          name: LicenceKey
          required: true
          type: string
      responses:
        default:
          description: ""
          schema:
            $ref: "#/definitions/checkPhoneNumberOutput"
      x-wso2-soap:
        soap-action: "http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber"
        soap-operation: CheckPhoneNumber
        namespace: "http://ws.cdyne.com/PhoneVerify/query"
        x-soap-version: "1.2"
  /checkPhoneNumbers:
    post:
      operationId: CheckPhoneNumbers
      parameters:
        - in: body
          name: Payload
          required: true
          schema:
            $ref: "#/definitions/checkPhoneNumbersInput"
      responses:
        default:
          description: ""
          schema:
            $ref: "#/definitions/checkPhoneNumbersOutput"
      x-wso2-soap:
        soap-action: "http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumbers"
        soap-operation: CheckPhoneNumbers
        namespace: "http://ws.cdyne.com/PhoneVerify/query"
        x-soap-version: "1.2"

info:
  title: PhoneVerificationAPI
  version: 1.0.0

definitions:
  checkPhoneNumberInput:
    type: object
    properties:
      CheckPhoneNumber:
        $ref: "#/definitions/CheckPhoneNumber"
  checkPhoneNumberOutput:
    type: object
    properties:
      CheckPhoneNumberResponse:
        $ref: "#/definitions/CheckPhoneNumberResponse"
  checkPhoneNumbersInput:
    type: object
    properties:
      CheckPhoneNumbers:
        $ref: "#/definitions/CheckPhoneNumbers"
  checkPhoneNumbersOutput:
    type: object
    properties:
      CheckPhoneNumbersResponse:
        $ref: "#/definitions/CheckPhoneNumbersResponse"
  StringArray:
    type: string
```

properties:
  String:
    type: array
    items:
      type: string
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
CheckPhoneNumber:
  type: object
  properties:
    PhoneNumber:
      type: string
    LicenseKey:
      type: string
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
CheckPhoneNumbersResponse:
  type: object
  properties:
    CheckPhoneNumbersResult:
      $ref: "#/definitions/ArrayOfPhoneReturn"
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
ArrayOfString:
  type: object
  properties:
    string:
      type: array
      items:
        type: string
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
ArrayOfPhoneReturn:
  type: object
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
CheckPhoneNumbers:
  type: object
  properties:
    PhoneNumbers:
      $ref: "#/definitions/ArrayOfString"
    LicenseKey:
      type: string
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
PhoneReturn:
  type: object
xml:
  namespace: "http://www.w3.org/2001/XMLSchema"
  prefix: s
CheckPhoneNumberResponse:
  type: object
  properties:
    CheckPhoneNumberResult:
      $ref: "#/definitions/PhoneReturn"
The generated API definitions are added to the API as shown below.

A REST API method is mapped with the corresponding SOAP operation through a Swagger, vendor-specific field - `x-wso2-soap`. Note that this should not be edited. A parameter without this field cannot be mapped with a SOAP backend operation.

See the sample REST operation given below.
Edit the details of the API as shown below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>PhoneVerification</td>
</tr>
<tr>
<td>Context</td>
<td>/phoneverify</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Access Control</td>
<td>All</td>
</tr>
<tr>
<td>Visibility on Store</td>
<td>Public</td>
</tr>
<tr>
<td>Tags</td>
<td>phone</td>
</tr>
</tbody>
</table>

For more information on the meanings of the latter mentioned fields, see Create and Publish an API.

The `operationId` parameter is used to map the REST resource with the corresponding SOAP operation.

8. Go to the Implement tab. Click on Managed API.

9. Select the Endpoint Type as the HTTP/SOAP Endpoint. Enter the SOAP endpoint URL given below.

```
SOAP Endpoint URL

http://ws.cdyne.com/phoneverify/phoneverify.asmx
```
10. Navigate to the **SOAP to REST Mapping** section. Click on a resource to view the In and Out sequences of the API.

The following is a sample Synapse based XML configuration which reads some parameters from the request path and constructs the SOAP payload required for the backend (target) services.
The incoming JSON message parameters are stored using properties. The SOAP payload needed for the backend is generated using a payload factory mediator.

11. Go to the Manage tab and select the throttling tiers for the API. Click Publish to publish the API to the API Store. For instructions, see Create and Publish an API.

12. Navigate to the API Store and subscribe to an API. After generating the production and sandbox keys you receive a valid access token to access the API.

13. Run the following HTTP request to invoke the API.
You receive the actual backend response.

**HTTP Request**

```bash
curl -k -X GET "https://localhost:8243/phoneverify/1.0.0/checkPhoneNumber?PhoneNumber=1234567&LicenceKey=0" -H "accept: application/json" -H "Authorization: Bearer <access-token>"
```

```json
{"CheckPhoneNumberResponse":{"CheckPhoneNumberResult":{"Valid":false,"OriginalNumber":1234567,"CleanNumber":234567,"Wireless":false}}}
```

---

**Developer Portal**

How do I...

- Include Additional Headers in the API Console
Include Additional Headers in the API Console

The Swagger API Console is a JavaScript client that runs in the API Store and makes JavaScript calls from the Store to the API Gateway. You must specify any additional headers that you want to add to the API Console under the CORS (Cross Origin Resource Sharing) configuration.

Before you begin...

Open the CORS configuration in the `<API-M_HOME>/repository/conf/api-manager.xml` file, enable CORS if it is not enabled already and specify the additional headers (SOAPAction, in this case) under the `<Access-Control-Allow-Headers>` element:

```
<CORSConfiguration>
    <Enabled>true</Enabled>
    <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
    <Access-Control-Allow-Methods>GET,PUT,POST,DELETE,PATCH,OPTIONS</Access-Control-Allow-Methods>
    <Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction</Access-Control-Allow-Headers>
</CORSConfiguration>
```

This configuration is only valid for APIs created through the API manager Publisher application. All the other Oauth token related APIs (/authorize, /revoke, /token, /userinfo) are not affected from this. To enable CORS configuration to these APIs as well, see Enabling CORS for Oauth Token related APIs.

Next, let's see how to add the header as a parameter to the API Console.

1. Log in to the API Publisher and click the API that you want to invoke (e.g., PhoneVerification).
2. Click the Edit link next to the API's name, navigate down to the API Definition section and click on the POST method to expand it.

3. Update the Produces and Consumes fields to `text/xml` and create the following header using the Add Parameter button.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAPAction</td>
<td>Description: Set to <code>http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</code></td>
</tr>
<tr>
<td></td>
<td>Parameter Type: Header</td>
</tr>
<tr>
<td></td>
<td>Data Type: String</td>
</tr>
<tr>
<td></td>
<td>Required: False</td>
</tr>
</tbody>
</table>
4. Once you are done, click Save.
5. Log in to the API Store, subscribe to the API and generate an access token for the application you subscribed with. If it’s an API that you are already subscribed to, you might have to re-generate the access token from the Applications page.

6. Click on the API again to open it and then click its API Console tab.
6. Expand the POST method and click **Try it out**.

7. Fill the parameter values and click **Execute** to invoke the API. For example,
This is the example SOAP request that we copied from the SOAP UI of the previous tutorial:

```xml
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CheckPhoneNumber xmlns="http://ws.cdyne.com/PhoneVerify/query">
      <PhoneNumber>650 745 4499</PhoneNumber>
      <!-- Optional LicenseKey parameter-->
      <LicenseKey>0</LicenseKey>
    </CheckPhoneNumber>
  </soap:Body>
</soap:Envelope>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Content Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAPAction</td>
<td><a href="http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber">http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</a></td>
</tr>
</tbody>
</table>

9. A result similar to the following appears on the console.
You have added SOAP parameters to the API Console and invoked a SOAP service using the API Console.

Enabling CORS for OAuth Token related APIs

Enabling CORS configuration through `api-manager.xml` is only valid for APIs created through the API manager Publisher application. Hence, enabling CORS for OAuth token related APIs (`authorize`, `revoke`, `token`, `userinfo`) can be carried out as follows.

Based on the API that you need to enable CORS, add the following handler configuration to the relevant API synapse file present in `<API-M_HOME>/repository/deployment/server/synapse-configs/default/api/` folder. It should be added within the `<handlers>` parent element.
The following are the mappings of the synapse files corresponding to the OAuth token related APIs.

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Synapse configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>/authorize</td>
<td><em>AuthorizeAPI</em>.xml</td>
</tr>
<tr>
<td>/revoke</td>
<td><em>RevokeAPI</em>.xml</td>
</tr>
<tr>
<td>/token</td>
<td><em>TokenAPI</em>.xml</td>
</tr>
<tr>
<td>/userinfo</td>
<td><em>UserInfoAPI</em>.xml</td>
</tr>
</tbody>
</table>

Log in to the API Store using Social Media

You can integrate WSO2 Identity Server with WSO2 API Manager and use your social media credentials to log in to the API Store and API Publisher. This tutorial shows you how to integrate Facebook authentication and log in to the API Store. Before following these steps, configure WSO2 Identity Server to provide Single Sign On for WSO2 API Manager by following Configuring External IDP through Identity Server for SSO.

- Create a Facebook application
- Configure Facebook login with Identity Server
- Configuring requested claims for user authentication in Facebook Identity Provider
- Configure service providers to the Publisher and Store with the Facebook Identity Provider
- Test Facebook authentication
- Configure the associated social login in IS dashboard

Create a Facebook application

1. Go to [https://developers.facebook.com/](https://developers.facebook.com/) and log in using your Facebook credentials.
2. Select My Apps in the navigation and create a new app by clicking Add a New App.
3. Enter the name of your app and your email address. Click Create App ID.
4. Click Set Up to create a Facebook Login product.
5. Select **Web** to work with this sample. You can select any other platform you wish to use.

   **Use the Quickstart to add Facebook Login to your app. To get started, select the platform for this app.**

   ![Platform Options](image)

   Change the port offset to 1 by modifying the `<Offset>` element value in `<IAM_HOME>/repository/conf/carbon.xml` as following.

   `<Offset>1</Offset>`

6. Add the **serverURL** of WSO2 Identity Server (which is configured with offset=1) `https://localhost:9444/` and click **Save** and **Continue**.

   ![Form](image)

   If you have changed the hostname of identity server use that instead of `localhost`.

   For example, if the host name is `identity.com`, then the server URL is `https://identity.com:9444/`

7. Go to **Set Up the Facebook SDK for JavaScript**. Click **Next**

8. Click **Settings** and select **Basic**. You can find your App ID and the App Secret as shown in the image below.
9. Select a **Category** for your application.

Add the correct **Site URL** as shown below and click **Save Changes**.

10. Click on the new **Facebook Login** product you have added and configure it as follows.
Now you have configured Facebook as your Identity Provider

### Configure Facebook login with Identity Server

Let's see how to configure WSO2 Identity Server to work with Facebook for user authentication, so that when you try to login to the API Publisher or Store, WSO2 Identity Server will redirect to Facebook to do the authentication. As a prerequisite, you have to configure WSO2 Identity Server by adding a new identity provider.

1. Download the WSO2 Identity Server 5.5.0 [here](#).
2. Configure Single Sign On with WSO2 API Manager 2.5.0.
3. Log in to the Management Console of WSO2 Identity Server as an administrator.
4. Go to the Identity section under the Main tab. Click Add under Identity Providers and enter following details.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client OAuth Login</td>
<td>Yes</td>
</tr>
<tr>
<td>Web OAuth Login</td>
<td>Yes</td>
</tr>
<tr>
<td>Valid OAuth Redirect URIs</td>
<td><a href="https://localhost:9444/commonauth">https://localhost:9444/commonauth</a></td>
</tr>
</tbody>
</table>

After the user authorizes the application, the authorization server redirects the user back to the application with access token or the authorization code in the URL. Since the redirected URL contains sensitive information, it is required to assure that the service does not redirect to arbitrary locations. The best way to ensure that the user is directed to the appropriate location is to define an OAuth Redirect URL as shown above.

### Login from Devices

Enables the OAuth client login flow for devices like a smart TV.
5. Go to Facebook Configuration under Federated Authenticators.
6. Enter the Client ID and Client Secret values obtained from the Facebook app created in the previous section.
7. Select Enable Facebook Authenticator and select Default to make it the default authentication method.
8. Enter the User information fields you want to retrieve separated by commas under User Information fields.
9. Click Register.

The Scope defines the permission to access particular information from a Facebook profile. See the Permissions Reference for a list of the different permission groups in Facebook APIs.
Configuring requested claims for user authentication in Facebook Identity Provider

We need to acquire the identity information by configuring claims for use Authentication in facebook. Let's see how you can configure Identity Server with Facebook by mapping the claims. For more information on claim Mapping refer Claim Management.

1. Go to the **Identity** section under the **Main** tab. Select **List** under **Identity Providers**.
2. Click **Edit** to edit the facebook identity provider you created.
3. Go to **Basic Claim Configuration** under Claim Configuration
4. Select the **Define Custom Claim Dialect** option under **Select Claim mapping Dialect**. Click **Add Claim Mapping** to add custom claim mappings as follows.

Add Identity Provider

<table>
<thead>
<tr>
<th>Basic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity Provider Name:</td>
</tr>
<tr>
<td>Display Name:</td>
</tr>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>Federation Hub Identity Provider:</td>
</tr>
<tr>
<td>Home Realm Identifier:</td>
</tr>
<tr>
<td>Identity Provider Public Certificate:</td>
</tr>
<tr>
<td>Alias:</td>
</tr>
</tbody>
</table>

Claim Configuration

<table>
<thead>
<tr>
<th>Basic Claim Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Claim mapping Dialect:</td>
</tr>
<tr>
<td>Identity Provider URIs:</td>
</tr>
<tr>
<td>User ID Claim URI:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

If you prefer to use the User ID as your first name of Facebook account, configure first_name claim as above. You need to select the same claim as **User ID Claim URI**.

5. The following are some common attribute names. You can map these names to any suitable **Local Claim URI**. (Local Claim is a set of standard claim values which are local to the WSO2 Identity Server)

- id
- email
- name
- first_name
- last_name
- link
- gender
- locale
- age_range

For more information, see Permissions Reference - Facebook Login.

Configure service providers to the Publisher and Store with the Facebook Identity Provider

To federate logging in to the Publisher and Store with Facebook, you need to configure the service provider with the Facebook Identity Provider.

You have to allow the usage of email addresses as usernames, to use email addresses. For instructions, Setting up an e-mail login.

1. Go to the Management console of WSO2 Identity Server (https://localhost:9444/carbon) and click on Service Providers.
2. Click **Edit** to edit the **API_PUBLISHER**.

3. Go to the **Local and Outbound Authentication Configuration** section. Select the Identity Provider you created from the dropdown list under **Federated Authentication**.

4. Make sure that **Federated Authentication** is selected. Click **Update** to save the changes.

5. Repeat steps 1 to 4 and configure the **API_STORE** service provider.

**Test Facebook authentication**

1. Access the API Publisher via `https://localhost:<port-number>/publisher`. Observe the request redirect to the WSO2 IS SAML2.0 based SSO login page and then Facebook login page.
2. Enter the username and password of your Facebook account.

3. After successfully authenticating the login, you will be logged into the API Publisher. Your username will be the first name of your Facebook account. This is because you have already configured the first name as the **UserID Claim URI**.

   If you configure your **UserID Claim URI** with the `last_name`, your username will be the last name of your Facebook account.

**Configure the associated social login in IS dashboard**

Identity Server has a dashboard which offers multiple options for users to maintain user accounts. Associating a social login for their account is one of the options provided in this dashboard. This dashboard can be accessed in the following URL: `https://<IS_HOST>:<IS_PORT>/dashboard`. By associating the social login you have the option to use local claims, instead of showing the logged name as Facebook username you can use logged users as the username in user local user store.
1. Login to the dashboard with API Store user account.
2. Click View Details in the Social Login gadget.

3. Click Associate Social Login to give your facebook account details.

4. Enter your IDP ID (facebook) and your username (as configured in Subject Claim URI) and click Register.

5. Select Local & Outbound Configuration and check Assert identity using mapped local subject identifier.

After logging in to API Publisher, you will see the configured local claim appearing as your username.

You have now successfully logged in to the API Publisher using your facebook credentials.
Test an API using a Testing Tool

When an enterprise exposes its APIs for internal or external consumption, application developers (both internal and external) write applications using these exposed APIs. Before the API is embedded within an application, it needs to be tested in order to make sure that the API can be successfully adopted. SmartBear’s Ready API! is an API testing tool widely used for this purpose. The WSO2 API Manager plugin, developed in partnership with SmartBear, allows seamless integration between the two products allowing application developers to work with the Ready! API platform to test APIs exposed via WSO2 API Manager.

This tutorial explains how to integrate Ready! API with WSO2 API Manager and then test APIs that are exposed in the API Manager. It also explains how Ready! API can be used to generate OAuth 2.0 tokens with different grant types and the benefits this integration can bring to an application developer.

This tutorial uses the PhoneVerification API, which is created in Create and Publish an API.

Installing the WSO2 API Manager plugin in Ready! API

1. Download and run Ready! API.
2. Click the Plugins button shown below to open the Plugin Manager.

![Plugin Manager](image)

3. Browse the plugin repository, locate the WSO2 API Manager plugin for Ready! API and click Install/Upgrade Plugin.

![Plugin Manager](image)

4. Click Yes on the confirmation message that appears.
Let's test an API exposed via WSO2 API Manager.

**Testing APIs using Ready! API**

1. Run the WSO2 API Manager server.
2. Run Ready! API if it's not already open.

If you are using a self-signed certificate with WSO2 API Manager, you need to explicitly configure Ready! API to trust this certificate. Click **File > Preferences** and then select the **Internal Browser** tab. Select the **Trust the self-signed certificate** check box and click **OK**.

3. Start by creating a new project in Ready! API. Click **File > New Project**.

4. In the **Description File** tab, select the **Import from WSO2 API Manager** option and click **Import**.
5. On the dialog box that appears, enter the information of the API Store from which API information needs to be extracted and click OK.

   It is possible to import APIs from tenant stores as well.

<table>
<thead>
<tr>
<th>Project Name</th>
<th>API Manager Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Store URL</td>
<td><a href="https://localhost:9443/store/">https://localhost:9443/store/</a></td>
</tr>
<tr>
<td>API Store User Name</td>
<td>admin</td>
</tr>
<tr>
<td>API Store Password</td>
<td>admin</td>
</tr>
<tr>
<td>API Manager Version</td>
<td>2.0.0</td>
</tr>
</tbody>
</table>
5. Select one or more APIs from the list of APIs available in the API Store and click OK.

6. Select an API (e.g. PhoneVerification) and expand the API to see all the HTTP verbs associated with it. Select the required HTTP verb (e.g. GET) to test the API.
7. Any query parameters that were added when creating the API in the API Manager appear here. You can also add your own resource path or query parameters to test the API.

8. To add your own parameters, click the **Parameters** field and then click the **Plus** icon.

9. Enter the required information for the API (e.g. 18006785432 as the phone number).

10. Once all the required information is added for the API, you need to add the API OAuth 2.0 token to invoke the API. Click the **Auth** tab at the bottom of the screen.
You can either get a test access token from the API Store or use the inbuilt OAuth 2.0 access token generation option. In this example, the inbuilt token generation option is used.

The inbuilt OAuth 2.0 access token generation option allows an access token to be generated with different grant types, which can be used to test the key generation process of WSO2 API Manager without requiring an application to perform the OAuth 2.0 key generation. This option can also be used in cases where you want to test access to different HTTP verbs and resource paths using different types of scopes and API keys.

11. In the Auth tab, click Get Access Token.
11. On the dialog box that appears, enter the following information.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth 2 Flow</td>
<td>You can choose from different grant types to generate your token. In this example, we have used the <strong>Client Credentials Grant</strong> type.</td>
</tr>
<tr>
<td>Client Identification and Client Secret</td>
<td>Get these values from the API Store. Browse to the application that the API being tested is subscribed to (e.g. DefaultApplication) and copy the values from the <strong>Production Keys</strong> tab.</td>
</tr>
<tr>
<td>Access Token URI</td>
<td>Provide the URL of the access token endpoint of the API Manager. By default, this URL is <code>https://localhost:9443/oauth2/token</code>. If you are using a componentized API manager deployment, the URI should point to the Key Manager component of the deployment.</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Scope</td>
<td>Define the scope under which the token is generated. If you have not defined any scope restrictions when creating the API you can leave this blank.</td>
</tr>
</tbody>
</table>

13. Once done, click **Get Access Token**.

14. The access token is retrieved from the server.
15. Now you can invoke the API by sending the request. The response is displayed as shown below. If you encounter an error, make sure that the values given for the endpoint, resource and parameters are correct.

GET - default_request

You have successfully tested an API.

To use the access token taken from the API Store,

a. Login to the WSO2 API Manager Store and browse to the application that the API being tested is subscribed to (e.g. DefaultApplication).

b. In the Production Keys tab, generate (or regenerate) a test access token and copy it.
b. Go back to the Ready! API, paste the access token for the API and send the request.

The response is displayed as shown below. If you encounter an error, make sure that the values given for the endpoint, resource and parameters are correct.

d. The response is displayed as shown below. If you encounter an error, make sure that the values given for the endpoint, resource and parameters are correct.
Use the Community Features

The API Store provides several useful features to build and nurture an active community of users, for your APIs. This is required to advertise APIs, learn user requirements and market trends.

Let's see what community features are available in the API Store:

- Use the search facility
- Rate and comment
- Share on social media/e-mail
- Embed an API widget
- Participate in the forum

Use the search facility

You can search for APIs in the API Publisher or Store in the following ways:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>By the API's name</td>
<td>As this is the default option, simply enter the API's name and search.</td>
<td></td>
</tr>
<tr>
<td>By the API provider</td>
<td>provider:xxxxx. For example, provider:admin</td>
<td>Provider is the user who created the API.</td>
</tr>
<tr>
<td>By the API version</td>
<td>version:xxxx. For example, version:1.0.0</td>
<td>A version is given to an API at the time it is created.</td>
</tr>
<tr>
<td>By the context</td>
<td>context:xxxx. For example, context:phoneverify</td>
<td>Context is the URL context of the API that is specified as /&lt;context_name&gt; at the time the API is created.</td>
</tr>
<tr>
<td>By the API's status</td>
<td>status:xxxx. For example, status: PUBLISHED</td>
<td>A state is any stage of an API's lifecycle. The default lifecycle stages include created, prototyped, published, deprecated, retired and blocked.</td>
</tr>
<tr>
<td>By description</td>
<td>description:xxxx</td>
<td>A description can be given to an API at the time it is created or later. There can be APIs without descriptions as this parameter is optional.</td>
</tr>
<tr>
<td>By the subcontext</td>
<td>subcontext:xxxx. For example, subcontext:checkphonenumber.</td>
<td>A subcontext is the URL pattern of any resource of the API. API resources are created at the time the API is created or later when it is modified. For example, if you create a resource by the name checkphonenumber, then /checkphonenumber becomes one subcontext of the API.</td>
</tr>
<tr>
<td>By the content of the API documentation</td>
<td>doc:xxxx</td>
<td>You can create API documentation in-line (using the API Publisher UI itself), by uploading a file or referring to an external URL. This search enables you to give a sentence or word phrase that is inside the in-line documentation and find the API that the documentation is added for.</td>
</tr>
</tbody>
</table>

Rate and comment

Rates and comments give useful insights to potential API consumers on the quality and usefulness of an API. You can rate and comment on each API version.

1. Log in to the API Store and click on a published API.
2. The API's Overview page opens. Note the rating and commenting options there:
3. Add a rating and a comment. Note that the comments appear sorted by the time they were entered, alongside the author's name.
Share on social media/e-mail

1. Log in to the API Store and click on a published API.
2. On the API's Overview page, you get the social media options using which you can share and advertise APIs.

Embed an API widget

A widget is an embeddable version of the API in HTML that you can share on your website or other web pages. This is similar to how YouTube videos are embedded in a web page.

1. Log in to the API Store and click on a published API.
2. Note the Embed tab under the API's sharing options.
Participate in the forum

1. Log in to the API Store.
2. Click the **Forum** tab to go to the forum, where you can initiate conversations and share your opinions with other users.

   Click **Create New Topic** to start a new topic in the Forum.

3. Enter the **Subject** and **Description** of your topic and click Create.
Write a Client Application Using the SDK

A software development kit (SDK) is a set of software development tools that allows to create applications for a specific platform. If an API consumer wants to create an application, they can generate a client side SDK for a supported language/framework and use it to write a software application to consume the subscribed APIs. This tutorial shows you how to write a client application using an SDK.

In this example, we use the sample API in WSO2 API Manager as a demonstration. To deploy the sample API, log in to the API Publisher and click the Deploy Sample API button. Note that the button only appears if no APIs have been created so far, in the given tenant space.

If a different API is used, the SDK functions to invoke the API are based on the specifications of that API.

1. Follow the steps in Invoke your first API, to deploy the sample API, subscribe and generate keys.

   **Access Token**

   Once the keys are generated, copy the access token. You can use this token to invoke APIs that you subscribe to using the same application.

2. Go to the API Store. Select your API and download the SDK for Java. For more details, see Generating client SDKs in the API Store.
In this example, you would have downloaded the `PizzaShackAPI_1.0.0_java.zip` file. This file name includes the API name, version, and language of the SDK. Unzip the `PizzaShackAPI_1.0.0_java.zip` file. Expand to see the folder structure of the unzipped file...
4. **Build the SDK using maven.**

When it’s done, you can include this SDK as a dependency in your software project. Details of this maven dependency are included in the README.md file.
Maven dependency

```xml
<dependency>
  <groupId>org.wso2</groupId>
  <artifactId>org.wso2.client.PizzaShackAPI</artifactId>
  <version>1.0.0</version>
  <scope>compile</scope>
</dependency>
```

Build using maven

You can build the SDK using the `mvn clean install` command inside the root directory. For more information see Maven Start Guide.

5. After creating a maven project, import the following with respect to the SDK. These classes will be accessible from the code once the SDK is built using maven and will be included as maven dependencies in the project.

```java
import org.wso2.client.api.ApiClient;
import org.wso2.client.api.PizzaShackAPI.DefaultApi;
import org.wso2.client.model.PizzaShackAPI.Menu;
```

6. Create an instance of the `DefaultApi` object in the java code. This instance is needed to get the API client which handles the operations related to consuming the API, using the resources of the API.

```java
DefaultApi defaultApi = new DefaultApi();
```

7. The API client of the `DefaultApi` object instance is used to set HTTP request headers with the required data. Note that these HTTP request headers might differ from one API to another, depending on the implementation of the API. A sample is show below.

```java
ApiClient apiClient = defaultApi.getApiClient();
apiClient.addDefaultHeader("Accept", "application/json");
```

8. Include the access token as a header in the API client object, to invoke the API.

```java
String accessToken = "bc392b16-6ce2-3208-9023-8938fbc376ea";
apiClient.addDefaultHeader("Authorization", "Bearer " + accessToken);
```

You need an access token to invoke the API. It is important to have a valid subscription before using the SDK, to obtain an access token. Note that the obtained access token has an expiration time.

9. Set the base path to the API client.

```java
apiClient.setBasePath("http://localhost:8280/pizzashack/1.0.0");
```

The base path for the client application is the production (or sandbox) URL of the API, found in the Overview tab of the API in the API Store.
10. Once the `ApiClient` object has all the required data, set the `ApiClient` for the instance of the `DefaultApi` object.

```java
defaultApi.setApiClient(apiClient);
```

11. Finally, we can call the available function in the SDK to get the response from the API.

```java
List<MenuItem> menuItems = (List<MenuItem>) defaultApi.menuGet();
```

`MenuItem` is a model class generated with SDK

Complete java code can be found below

[Java Code](pom file)
import org.wso2.client.api.ApiClient;
import org.wso2.client.api.ApiException;
import org.wso2.client.api.PizzaShackAPI.DefaultApi;
import org.wso2.client.model.PizzaShackAPI.MenuItem;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class APIClient {

    public static void main(String[] args) throws ApiException {
        DefaultApi defaultApi = new DefaultApi();
        String accessToken = "bc392b16-6ce2-3208-9023-8938fbc376ea";
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Accept", "application/json");
        headers.put("Authorization", "Bearer " + accessToken);
        ApiClient apiClient = defaultApi.getApiClient();
        apiClient.addDefaultHeader("Accept", "application/json");
        apiClient.addDefaultHeader("Authorization", "Bearer " + accessToken);
        apiClient.setLenientOnJson(true);
        apiClient.setBasePath("http://localhost:8280/pizzashack/1.0.0");
        defaultApi.setApiClient(apiClient);
        List<MenuItem> menuItems = (List<MenuItem>) defaultApi.menuGet();
        System.out.println(menuItems);
    }
}

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.pizzashack.client</groupId>
    <artifactId>pizzashack-api</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.wso2</groupId>
            <artifactId>org.wso2.client.PizzaShackAPI</artifactId>
            <version>1.0.0</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
</project>

Obtaining User Profile Information with OpenID Connect

OpenID Connect is an authentication protocol that is a simple identity layer on top of the OAuth 2.0 protocol. It allows clients to verify the identity of the end-user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner.

You can use WSO2 API Manager to obtain basic profile information about the user who generates the access token. To obtain this information, the openid scope needs to be passed, when generating the access token. API manager will send a JWT which contains information about the user who generates the token, as part of the response for this request. You can configure the information returned with the JWT token.
Follow the instructions below to obtain user profile information with OpenID connect with WSO2 API Manager.

1. Obtain a token using password grant type and openid scope. For more information on token generation with password grant type, see Password Grant Type. The format of the curl command and a sample is given below:

```
```

```
curl -k -d "grant_type=password&username=testuser&password=testuserpassword&scope=openid" -H "Authorization: Basic M1J6RFNrRFI5mSczRqY296R2xvJh0Q5UIYIpXeElqgKFJd0q6RVYOHDlZGFfcmGMIW194RjRh, Content-Type: application/x-www-form-urlencoded" https://apim.wso2.com:8243/token
```

You will receive a response in the format shown below. Note that the id_token parameter contains the JWT related to user information.

```
{
    "access_token": "83705add-d77e-3cc8-9b6a-53d210ed3fed",
    "refresh_token": "4b283fb8-942f-316d-ba90-44b4c6ae19",
    "scope": "openid",
    "id_token": "eyJ4NXQlO1i0bUp7ODweE16WmXakDyWkRgA5EUWmxZVEExkpxaPuUl1PVOwU7dJNk0PyKz1pVGMxWkEiLCJrzWQiOjkwMjV5NTExOYyZ3mOdhyXMGf12DeyYT14NDA2O7lZGZGViYT1kIiw6iXWkNeUjoiU3MyNTyfQ.eyJh0YXoijWioiY012vSd8yV0BgTI1ekekPUI15SImFojci6In16VybJpYWN1Omu29bWmuOmlhcDpsaWkZ2X1IiLCJzdWJjOiJzdWJzY3JpYmVyQGQncmVib3VzIiwic3Vic3RlZCI6MzEwMjvyb3VzIiwiaWRfY29zdHJvZHkiOiJwcmVzc2ZyaXRob3MgY2xhc3QieCIsImVtYWlsIjoiQWxhc3QgeCIsIm5ldmVyIjoiY2F0YWxvZ2UifQ.gdj0jn4PX5R4j5Y0ZNyEwi2G-NPq3_iW89NqkRxesdcmLvDp-nCRWMyyyUYc-bQqADekTdTQC4ACSVUI3BKau30y8uu-AO8pajIm-hWEX_PBqoMrt5ztzgmgKFAhL0Gdr8B1u8LZl51bX2cTKss_zYWNaMcPdKv2DqO5L098Wg",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

2. The following two options are available to view the actual user information.

- Decoding the id_token
- Invoking the userinfo endpoint

Decoding the id_token

By decoding the id_token, a payload similar to the following can be obtained, with user information such as email, organization, etc.

```
{
    "at_hash": "cXhWlVwBpgT0ALmazJHQ",
    "acct": "urn:naacme:incommon:iap:silver",
    "sub": "user1@carbon.super",
    "aud": [
        "KoNdleSrF3naXWhavao4bBoMYCa"
    ],
    "azp": "KoNdleSrF3naXWhavao4bBoMYCa",
    "organization": "WSO2",
    "iss": "https://172.16.2.111:9443/oauth2/token",
    "exp": 1511950413,
    "iat": 1511946813,
    "email": "user1@gmail.com"
}
```

For an online tool to decode the JWT, go to https://jwt.io/
Invoking the userinfo endpoint

You can obtain user information as a payload by invoking the userinfo endpoint with the access token obtained in step 1. The format of the curl command and a sample is given below.

```
```

```
```

The response will be a JSON payload as shown below:

```
{
  "sub": "user1@carbon.super",
  "organization": "WSO2",
  "email": "user1@gmail.com"
}
```

By default, only the username (sub) information will be available in the response. You can customize the user information returned by configuring the claims of the relevant Service Provider generated for the Application created in Store. For more information, see Service Provider Claim Configuration.

Cleaning Up Partially Created Keys

An application created in WSO2 API Manager has a corresponding OAuth application in the Key Manager node. An application can be created or deleted partially, where the OAuth application is successfully created/deleted but there is stale data left in the API Manager node. This can happen due to network failures between the API Manager and the Key Manager nodes, partial deletion of applications, etc.

To delete the remaining application data from API Manager, go to the Production Keys tab, and click Clean up.

Add Custom Attributes to Applications

Applications in the API Store are created and subscribed to APIs allowing users to consume the APIs, and helping developers to expose monetize their APIs. Applications can be created with predefined set of properties. This feature enables the functionality to provide set-of custom attributes while creating application.

Follow the steps below to configure adding custom attributes to applications

- Step 1 - Configuring custom attributes for applications
- Step 2 - Creating an application with custom attributes through the API Store

**Step 1 - Configuring custom attributes for applications**

Custom attributes can be configured in the following ways.

- Server-specific configuration
- Tenant-specific configuration
Server-specific configuration

To configure server-specific custom attributes, open the `<API-M-home>/repository/conf/api-manager.xml` file. Add `<ApplicationAttributes>` as shown in the example below.

```xml
<ApplicationConfiguration>
  <ApplicationAttributes>
    <Attribute required="true">
      <Name>External Reference Id</Name>
      <Description>Sample description of the attribute</Description>
    </Attribute>
  </ApplicationAttributes>
</ApplicationConfiguration>
```

Tenant-specific configuration

To configure server-specific custom attributes, open the `<API-M-home>/repository/resources/tenant-conf.json` file. Add `ApplicationConfigs` under `Attributes` as shown below.

```json
"ApplicationConfigs": {
  "Attributes": [
    {
      "Attribute": "External Reference Number",
      "Required": true,
      "Description": "Sample description"
    },
    {
      "Attribute": "Tenant Billing tier"
    }
  ]
}
```

**Step 2 - Creating an application with custom attributes through the API Store**

After configuring the custom attributes as shown in the previous section, you will be able to add these attributes to the applications you create in the API Store.

1. Log in to the API Store. Click **Add Application**, to add a new application.

2. You will see the configured custom attribute as shown below.
Add Application

An application is a logical collection of APIs. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe created and allows unlimited access by default.

Name

Per Token Quota

Unlimited

Allows unlimited requests

This feature allows you to assign an API request quota per access token. The allocated quota will be shared among all the subscribed APIs of the application.

Description

Billing Tier:

External Reference Id:

Add Cancel
Using WSO2 API Manager

This section provides information about the features, functionality, solution development, testing and debugging options of WSO2 API Manager.

- Installation Guide
- WSO2 API Manager Tooling
- Product Administration
- Configuring the API Manager
- Working with the API Microgateway
- Extending the API Manager
- Working with Security
- Working with Throttling
- Working with Endpoints
- Working with Analytics
- Reference Guide
- Developer Guide

Installation Guide

The following topics show how to download, install, run and get started quickly with WSO2 API Manager.

- Installation Prerequisites
- Installing the Product
- Running the Product
- Basic Health Checks

Installation Prerequisites

Before installing any WSO2 Carbon based product, it is necessary to have the appropriate prerequisite software installed on your system. Verify that the computer has the supported operating system and development platforms before starting the installation.

**System requirements**

| Physical | • 3 GHz Dual-core Xeon/Opteron (or latest)  
| 4 GB RAM (2 GB for JVM and 2 GB for the operating system)  
| 10 GB free disk space  
| ~ Recommended minimum - 2 Cores. For high concurrencies and better performances - 4 Cores.  
| Disk space is based on the expected storage requirements that are calculated by considering the file uploads and the backup policies. For example, if three WSO2 product instances are running in a single machine, it requires a 4 GHz CPU, 8 GB RAM (2 GB for the operating system and 6 GB (2 GB for each WSO2 product instance)) and 30 GB of free space. |
| Virtual Machine (VM) | • 2 compute units minimum (each unit having 1.0-1.2 GHz Opteron/Xeon processor)  
| 4 GB RAM  
| 10 GB free disk space  
| One CPU unit for the operating system and one for JVM.  
| Three WSO2 product instances running would require VM of 4 compute units, 8 GB RAM, and 30 GB free space.  
| ~ 512 MB heap size. This is generally sufficient to process typical SOAP messages but the requirements vary with larger message sizes and the number of messages processed concurrently. |
| EC2 | • 1 c5.large instance to run one WSO2 product instance.  
| Three WSO2 product instances can be run in 1 EC2 Extra-Large instance. Based on the I/O performance of the c5.large instance, it is recommended to run multiple instances in a larger instance (c5.xlarge). |

Environment compatibility
### Operating systems/ Databases

- All WSO2 Carbon-based products are Java applications that can be run on any platform that is Oracle JDK 1.8 or OpenJDK 8 compliant.
- For more information on JDKs that WSO2 products are tested with, see [Tested Operating Systems and JDKs](#).
- All WSO2 Carbon-based products are generally compatible with most common DBMSs. The embedded H2 database is suitable for development, testing, and some production environments. For most enterprise production environments, however, we recommend you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. For more information, see [Working with Databases](#). Additionally, we do not recommend the H2 database as a user store.
- It is **not recommended to use Apache DS** in a production environment due to scalability issues. Instead, use an LDAP like OpenLDAP for user management.
- On a production deployment, it is recommended that WSO2 products are installed on latest releases of RedHat Enterprise Linux or Ubuntu Server LTS.
- For environments that WSO2 products are tested with, see [Compatibility of WSO2 Products](#).
- To find out if this version of the product has issues running on your OS due to the JDK version, see [Known Incompatibilities](#).
- If you have difficulty in setting up any WSO2 product in a specific platform or database, please [contact us](#).

### Required applications

The following applications are required for running the API Manager and its samples or for building from the source code. Mandatory installs are marked with `*`.

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Version</th>
<th>Download Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java SE Development Kit (JDK)*</td>
<td>To launch the product as each product is a Java application.</td>
<td>OpenJDK 8</td>
<td>Oracle JDK</td>
</tr>
<tr>
<td></td>
<td>To build the product from the source distribution (both JDK and Apache Maven are required).</td>
<td>Oracle JDK 1.8.*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>To run Apache Ant.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Important!

Some updates of JDK 1.8 (for example, JDK1.8.0_151) are affected by a known issue related to GZIP decoding. Until this issue is fixed, we recommend one of the following approaches:

- Use either JDK1.8.0_144 or JDK1.8.0_077 updates. We have verified that these versions are not affected by the known issue.
- Alternatively, you can disable GZIP decoding for your product by following the steps given below. This will ensure that your product is not affected by the known issue.
  1. Open the `catalina-server.xml` file from the `<APIM_HOME>/repository/conf/tomcat/at/` directory.
  2. Set the `compression` parameter (under each of the connector configurations) to false as shown below:

        compression="off"

  3. Restart the server.
### Apache ActiveMQ JMS Provider
- To enable the product's JMS transport and try out JMS samples. The ActiveMQ client libraries must be installed in the product's classpath before you can enable the JMS transport.
  - 5.5.0 or later
  - If you use any other JMS provider (e.g., Apache Qpid), install any necessary libraries and/or components.

### Apache Ant
- To compile and run the product samples in `<API-M_HOME>/samples`
  - 1.7.0 or later

### Apache Maven
- To build the product from the source distribution (both JDK and Apache Maven are required). If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do not need to install Maven.
  - 3.0.0

### Web Browser
- To access the Management Console. The Web browser must be JavaScript enabled to take full advantage of the Management console.
  - On Windows Server 2003, you must not go below the medium security level in Internet Explorer 6.x.

---

You are now ready to install. Click one of the following links for instructions:
- Installing on Linux or OS X
- Installing on Solaris
- Installing on Windows
- Installing as a Linux Service

## Installing the Product
Installing WSO2 is very fast and easy. Before you begin, be sure you have met the installation prerequisites, and then follow the installation instructions for your platform. For information, contact team WSO2.

- Installing on Linux or OS X
- Installing on Solaris
- Installing on Windows
- Installing as a Linux Service

### Installing on Linux or OS X

#### Before you begin:
- See our compatibility matrix to find out if this version of the product is fully tested on Linux or OS X.
- See the known incompatibilities section to find out if this version of the product has issues running on your OS due to the JDK version.

Follow the instructions below to install API Manager on Linux or Mac OS X.

#### Installing the required applications
1. Log in to the command line (Terminal on Mac).
2. Ensure that your system meets the Installation Prerequisites. Java Development Kit (JDK) is essential to run the product.

#### Installing the API Manager
1. Download the latest version of the API Manager from https://wso2.com/api-management/install/.
2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API-M_HOME>`.

#### Setting up JAVA_HOME
You must set your JAVA_HOME environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer.

Environment variables are global system variables accessible by all the processes running under the operating system.

1. In your home directory, open the BASHRC file (.bash_profile file on Mac) using editors such as vi, emacs, pico, or mcedit.
2. Assuming you have JDK 1.7.0_80 in your system, add the following two lines at the bottom of the file, replacing /usr/java/jdk1.7.0_80 with the actual directory where the JDK is installed.

```bash
On Linux:
export JAVA_HOME=/usr/java/jdk1.7.0_80
export PATH=${JAVA_HOME}/bin:${PATH}

On OS X:
export JAVA_HOME=/System/Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home
```

3. Save the file.

   If you do not know how to work with text editors in a Linux SSH session, run the following command: `cat >> .bashrc`. Paste the string from the clipboard and press "Ctrl+D."

4. To verify that the JAVA_HOME variable is set correctly, execute the following command:

   5. The system returns the JDK installation path.

```bash
On Linux:
echo $JAVA_HOME

On OS X:
which java
```

If the above command gives you a path like /usr/bin/java, then it is a symbolic link to the real location. To get the real location, run the following:

```bash
ls -l `which java`
```

### Setting system properties

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script**: Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.

- **Set the properties from an external registry**: If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the secure vault.

When using SUSE Linux, it ignores /etc/resolv.conf and only looks at the /etc/hosts file. This means that the server will throw an exception on startup if you have not specified anything besides localhost. To avoid this error, add the following line above 127.0.0.1 localhost in the /etc/hosts file:

```
<ip_address> <machine_name> localhost
```

You are now ready to run the product.

### Installing on Solaris

#### Installing the required applications

1. Establish an SSH connection to the Solaris machine or log in on the text console.

2. Be sure your system meets the Installation Prerequisites. Java Development Kit (JDK) is essential to run the product.

#### Installing the API Manager

1. Download the latest version of the API Manager from [https://wso2.com/api-management/install/](https://wso2.com/api-management/install/).

2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API-M_HOME>`.
Setting up JAVA_HOME

You must set your JAVA_HOME environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer.

Environment variables are global system variables accessible by all the processes running under the operating system.

1. In your home directory, open the BASHRC file in your favorite text editor, such as vi, emacs, pico, or mcedit.
2. Assuming you have JDK 1.7.0_80 in your system, add the following two lines at the bottom of the file, replacing /usr/java/jdk1.7.0_80 with the actual directory where the JDK is installed.

   ```
   export JAVA_HOME=/usr/java/jdk1.7.0_80
   export PATH=${JAVA_HOME}/bin:${PATH}
   ```

   The file should now look like this:

   ```
   #JAVA_HOME
   export JAVA_HOME=/usr/java/jdk1.7.0_80
   export PATH=${JAVA_HOME}/bin:${PATH}
   ```

3. Save the file.

4. To verify that the JAVA_HOME variable is set correctly, execute the following command:

   ```
   echo $JAVA_HOME
   ```

   The system returns the JDK installation path.

Setting system properties

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script**: Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
- **Set the properties from an external registry**: If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the secure vault.

You are now ready to run the product.

Installing on Windows

Follow the instructions below to install API Manager on Windows.

Installing the required applications

1. Be sure your system meets the Installation Prerequisites. Java Development Kit (JDK) is essential to run the product.
2. Be sure that the PATH environment variable is set to “C:\Windows\System32”, because the findstr windows exe is stored in this path.

Installing the API Manager

1. Download the latest version of the API Manager from https://wso2.com/api-management/install/.
2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as <API-M_HOME>.

Installing and setting up snappy-java

1. Download the snappy-java_1.1.1.7.jar from here.
2. Copy the jar to <API-M_HOME>\repository\components\lib.
3. If the API Manager server is currently running, restart it to apply the changes.

**Setting up JAVA_HOME**

You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under `C:/Program Files/Java`, such as `C:/Program Files/Java/jdk1.7.0_80`. If you have multiple versions installed, choose the latest one, which you can find by sorting by date.

Environment variables are global system variables accessible by all the processes running under the operating system. You can define an environment variable as a system variable, which applies to all users, or as a user variable, which applies only to the user who is currently logged in.

You set up `JAVA_HOME` using the System Properties, as described below. Alternatively, if you just want to set `JAVA_HOME` temporarily for the current command prompt window, set it at the command prompt.

**Setting up JAVA_HOME using the system properties**

1. Right-click the **My Computer** icon on the desktop and click **Properties**.

2. In the System Properties window, click the **Advanced** tab, and then click **Environment Variables**.

3. Click **New** under **System variables** (for all users) or under **User variables** (just for the user who is currently logged in).
4. Enter the following information:
   - In the Variable name field, enter: JAVA_HOME
   - In the Variable value field, enter the installation path of the Java Development Kit, such as: c:/Program Files/Java/jdk1.7.0_80

The JAVA_HOME variable is now set and will apply to any subsequent command prompt windows you open. If you have existing command prompt windows running, you must close and reopen them for the JAVA_HOME variable to take effect, or manually set the JAVA_HOME variable in those command prompt windows as described in the next section. To verify that the JAVA_HOME variable is set correctly, open a command window (from the Start menu, click Run, and then type CMD and click Enter) and execute the following command:

```
set JAVA_HOME
```

The system returns the JDK installation path. You are now ready to run the product.

Setting JAVA_HOME temporarily using the Windows command prompt (CMD)

You can temporarily set the JAVA_HOME environment variable within a Windows command prompt window (CMD). This is useful when you have an existing JAVA_HOME command prompt window running and you do not want to restart it.

1. In the command prompt window, enter the following command where <JDK_INSTALLATION_PATH> is the JDK installation directory and press Enter.

```
set JAVA_HOME=<JDK_INSTALLATION_PATH>
```

   For example: set JAVA_HOME=c:/Program Files/java/jdk1.7.0_80

   The JAVA_HOME variable is now set for the current CMD session only.

2. To verify that the JAVA_HOME variable is set correctly, execute the following command:

```
set JAVA_HOME
```

   The system returns the JDK installation path.

Setting system properties

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script**: Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.

- **Set the properties from an external registry**: If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the secure vault.

You are now ready to run the product.

Installing as a Linux Service

Follow the sections below to run a WSO2 product as a Linux service:

- **Prerequisites**
- **Setting up CARBON_HOME**
- **Running the product as a Linux service**

**Before you begin:**

- See our compatibility matrix to find out if this version of the product is fully tested on your OS.
- See the known incompatibilities section to find out if this version of the product has issues running on your OS due to the JDK version.

**Prerequisites**

- Install JDK and set up the JAVA_HOME environment variable.

**Setting up CARBON_HOME**
Extract the WSO2 product to a preferred directory in your machine and set the environment variable CARBON_HOME to the extracted directory location.

**Running the product as a Linux service**

1. To run the product as a service, create a startup script and add it to the boot sequence. The basic structure of the startup script has three parts (i.e., start, stop and restart) as follows:

```bash
#!/bin/bash

case "$1" in
    start)
        echo "Starting the Service"
    ;;
    stop)
        echo "Stopping the Service"
    ;;
    restart)
        echo "Restarting the Service"
    ;;
    *)
        echo "$0 {start|stop|restart}"
        exit 1
esac
```

Given below is a sample startup script. <API-M_HOME> can vary depending on the WSO2 product's directory.

```bash
#!/bin/sh
export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_07"

startcmd='<API-M_HOME>/bin/wso2server.sh start > /dev/null &'
restartcmd='<API-M_HOME>/bin/wso2server.sh restart > /dev/null &'
stopcmd='<API-M_HOME>/bin/wso2server.sh stop > /dev/null &'

case "$1" in
    start)
        echo "Starting the WSO2 Server ..."
        su -c "${startcmd}" user1
    ;;
    restart)
        echo "Re-starting the WSO2 Server ..."
        su -c "${restartcmd}" user1
    ;;
    stop)
        echo "Stopping the WSO2 Server ..."
        su -c "${stopcmd}" user1
    ;;
    *)
        echo "$0 {start|stop|restart}"
        exit 1
esac
```

In the above script, the server is started as a user by the name user1 rather than the root user. For example, `su -c "${startcmd}" user1`

2. Add the script to `/etc/init.d/` directory.

3. If you want to keep the scripts in a location other than `/etc/init.d/` folder, you can add a symbolic link to the script in `/etc/init.d/` and keep the actual script in a separate location. Say your script name is prodserver and it is in `/opt/WSO2/` folder, then the commands for adding a link to `/etc/init.d/` is as follows:

- Make executable: `sudo chmod a+x /opt/WSO2/prodserver`
- Add a link to `/etc/init.d/`: `sudo ln -snf /opt/WSO2/prodserver /etc/init.d/prodserver`

3. Install the startup script to respective runlevels using the `update-rc.d` command. For example, give the following command for the sample script shown in step1:
The defaults option in the above command makes the service to start in runlevels 2, 3, 4 and 5 and to stop in runlevels 0,1 and 6.

A runlevel is a mode of operation in Linux (or any Unix-style operating system). There are several runlevels in a Linux server and each of these runlevels is represented by a single digit integer. Each runlevel designates a different system configuration and allows access to a different combination of processes.

4. You can now start, stop and restart the server using service <service name> {start|stop|restart} command. You will be prompted for the password of the user (or root) who was used to start the service.

Installing as a Windows Service

WSO2 Carbon and any Carbon-based product can be run as a Windows service as described in the following sections:

- Prerequisites
- Setting up the YAJSW wrapper configuration file
- Setting up CARBON_HOME
- Running the product in console mode
- Working with the WSO2CARBON service

Before you begin:

- See our compatibility matrix to find out if this version of the product is fully tested on your OS.
- See the known incompatibilities section to find out if this version of the product has issues running on your OS due to the JDK version.

Prerequisites

- Install JDK and set up the JAVA_HOME environment variable.
- Download and install a service wrapper library to use for running WSO2 API Manager as a Windows service. WSO2 recommends Yet Another Java Service Wrapper (YAJSW) version 11.03, and several WSO2 products provide a default wrapper.conf file in their <PRODUCT_HOME>/bin/yajsw/ directory. The instructions below describe how to set up this file.

Setting up the YAJSW wrapper configuration file

The configuration file used for wrapping Java Applications by YAJSW is wrapper.conf, which is located in the <PRODUCT_HOME>/conf/ directory of many WSO2 products. Following is the minimal wrapper.conf configuration for running a WSO2 product as a Windows service. Open your wrapper.conf file, set its properties as follows, and save it in <YAJSW_HOME>/conf/ directory.

If you want to set additional properties from an external registry at runtime, store sensitive information like usernames and passwords for connecting to the registry in a properties file and secure it with secure vault.

Manual Configurations

Add the following class path to the wrapper.conf file manually to avoid errors in the WSO2 API Manager Management Console:

wrapper.java.classpath.3 =
$\{carbon_home\}\repository\components\plugins\commons-lang_2.6.0.wso2v1.jar

Minimal wrapper.conf configuration

```ini
# working directory
#**********************************************************************************
wrapper.working.dir=${carbon_home}\
# Java Main class.
# YAJSW: default is "org.rzo.yajsw.app.WrapperJVMMain"
# DO NOT SET THIS PROPERTY UNLESS YOU HAVE YOUR OWN IMPLEMENTATION
# wrapper.java.mainclass=
#**********************************************************************************
# tmp folder
```
## yajsw creates temporary files named in... out... err... jna...
# per default these are placed in jna.tmpdir.
# jna.tmpdir is set in setenv batch file to <yajsw>/tmp
#*******************************************************************************
wrapper.tmp.path = ${jna tmpdir}
#*******************************************************************************

# Application main class or native executable
# One of the following properties MUST be defined
#*******************************************************************************
# Java Application main class
wrapper.java.app.mainclass=org.wso2.carbon.bootstrap.Bootstrap
# Log Level for console output. (See docs for log levels)
wrapper.console.loglevel=INFO
# Log file to use for wrapper output logging.
wrapper.logfile=${wrapper_home}\log\wrapper.log
# Format of output for the log file. (See docs for formats)
#wrapper.logfile.format=LPTM
# Log Level for log file output. (See docs for log levels)
#wrapper.logfile.loglevel=INFO
# Maximum size that the log file will be allowed to grow to before
# the log is rolled. Size is specified in bytes. The default value
# of 0, disables log rolling by size. May abbreviate with the 'k' (kB) or
# 'm' (mB) suffix. For example: 10m = 10 megabytes.
# If wrapper.logfile does not contain the string ROLLNUM it will be automatically added as suffix of
# the file name
wrapper.logfile.maxsize=10m
# Maximum number of rolled log files which will be allowed before old
# files are deleted. The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
#*******************************************************************************
# Wrapper Windows Service and Posix Daemon Properties
#*******************************************************************************
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
#*******************************************************************************
# Wrapper System Tray Properties
#*******************************************************************************
# enable system tray
wrapper.tray = true
# TCP/IP port. If none is defined multicast discovery is used to find the port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002
#*******************************************************************************
# Exit Code Properties
#*******************************************************************************
# Restart on non zero exit code
#*******************************************************************************
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
#*******************************************************************************
# Trigger actions on console output
#*******************************************************************************
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\trayMessage.gv
wrapper.filter.script.0.args=Exception
#*******************************************************************************
# genConfig: further Properties generated by genConfig
#*******************************************************************************
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 = -Xbootclasspath/a:${carbon_home}\lib\xboot\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 = -XX:HeapDumpPath=${carbon_home}\repository\logs\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 = -Djava.endorsed.dirs=${carbon_home}\lib\endorsed;${java_home}\jre\lib\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=/
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\bin\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\tmp
wrapper.java.additional.14 = -Dcatalina.base=${carbon_home}\lib\tomcat
wrapper.java.additional.15 = -Djava.util.logging.config.file=${carbon_home}\repository\conf\tomcat\tomcat-log.properties
wrapper.java.additional.16 = -Dcarbon.config.dir.path=${carbon_home}\repository\conf
wrapper.java.additional.17 = -Dcarbon.config.dir.path=${carbon_home}\repository\config
wrapper.java.additional.18 = -Dcomponents.repo=${carbon_home}\repository\plugins
wrapper.java.additional.19 = -Dcom.atomikos.icatch.file=${carbon_home}\lib\transactions.properties
wrapper.java.additional.20 = -Dcom.atomikos.icatch.hide_init_file_path=true
wrapper.java.additional.21 = -Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true
wrapper.java.additional.22 = -Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false
wrapper.java.additional.23 = -Dorg.terracotta.quartz.skipUpdateCheck=true
wrapper.java.additional.24 = -Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false
wrapper.java.additional.27 = -Dfile.encoding=UTF8
wrapper.java.additional.28 = -DworkerNode=false
wrapper.java.additional.29 = -Dorg.wso2.ignoreHostnameVerification=true

Setting up CARBON_HOME

Extract WSO2 API Manager that you want to run as a Windows service, and then set the Windows environment variable CARBON_HOME to the extracted product directory location (e.g., wso2am-2.6.0) here.

Running the product in console mode

You will now verify that YAJSW is configured correctly for running the WSO2 API Manager as a Windows service.

1. Open a Windows command prompt and go to the <YAJSW_HOME>/bat/ directory. For example:

   cd C:\Documents and Settings\yajsw_home\bat

2. Start the wrapper in console mode using the following command:

   runConsole.bat

   For example:

   C:\Documents and Settings\yajsw_home\bat>runConsole.bat

   If the configurations are set properly for YAJSW, you will see console output similar to the following and can now access the WSO2 management console from your web browser via https://localhost:9443/carbon.

   C:\Documents and Settings\yajsw_home\bat>runConsole.bat
   C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\n   C:\Documents and Settings\yajsw_home\bat>call setenv.bat "java" -XX:MaxRAMSize=-Djna.topdir="C:\Documents and Settings\yajsw_home\bat/.../tmp" -jar C:\Documents and Settings\yajsw_home\bat/.../wrapper.jar -c "C:\Documents and Settings\yajsw_home\bat/.../conf\wrapper.conf"
   YAJSW: yajsw-stable-11.03
   OS : Windows XP/5.2/amd64
   JVM : Oracle Corporation/1.7.0_06
   INFO: Using "C:\DOCUME"1\ADMINI"1\LOCALS"1\Temp\vfs_cache" as temporary files store.

Working with the WSO2CARBON service

To install the Carbon-based product WSO2 API Manager as a Windows service, execute the following command in the <YAJSW_HOME>/bat/ directory:
installService.bat

The console will display a message confirming that the WSO2CARBON service was installed.

C:\Documents and Settings\yajsw_home\bat\installService.bat
C:\Documents and Settings\yajsw_home\bat\cd C:\Documents and Settings\yajsw_home\bat\n
C:\Documents and Settings\yajsw_home\bat\call setenv.bat
"java" -mx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat/./tmp" -jar "C:\Documents and Settings\yajsw_home\bat/./wrapper.jar" -i "C:\Documents and Settings\yajsw_home\bat/./conf/wrapper.conf"

YAJSH: yajsw-stable-1.1.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0.06
INFO: Using "C:\DOCUME1\ADMINI1\LOCALS\1Temp\vfs_cache" as temporary files store.
platform null
================================================================================
Service "WSO2CARBON" installed
Press any key to continue . . .

To start the service, execute the following command in the same console window:

startService.bat

The console will display a message confirming that the WSO2CARBON service was started.

C:\Documents and Settings\yajsw_home\bat\startService.bat
C:\Documents and Settings\yajsw_home\bat\cd C:\Documents and Settings\yajsw_home\bat\n
C:\Documents and Settings\yajsw_home\bat\call setenv.bat
"java" -mx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat/./tmp" -jar "C:\Documents and Settings\yajsw_home\bat/./wrapper.jar" -t "C:\Documents and Settings\yajsw_home\bat/./conf/wrapper.conf"

YAJSH: yajsw-stable-1.1.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0.06
Dec 30, 2012 1:09:00 PM org.apache.commons.vfs2.VfsLog info
INFO: Using "C:\DOCUME1\ADMINI1\LOCALS\1Temp\vfs_cache" as temporary files store.
platform null
================================================================================
Service "WSO2CARBON" starting
Press any key to continue . . .

To stop the service, execute the following command in the same console window:

stopService.bat

The console will display a message confirming that the WSO2CARBON service has stopped.
To uninstall the service, execute the following command in the same console window:

```bat
uninstallService.bat
```

The console will display a message confirming that the WSO2CARBON service was removed.

```
C:\Documents and Settings\yajsw_home\bat\uninstallService.bat
C:\Documents and Settings\yajsw_home\bat\cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat\call setenv.bat
"java" -Xmx30m -Djava_tmpdir="C:\Documents and Settings\yajsw_home\bat\./tmp" -jar "C:\Documents and Settings\yajsw_home\bat\./wrapper.jar" -p "C:\Documents and Settings\yajsw_home\bat\./conf\wrapper.conf"
Yajsw: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
INFO: Using "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\vfs_cache" as temporary files store.
platform null
*************************** STOPPING "WS02CARBON" ****************************

Service "WS02CARBON" stopped
Press any key to continue . . .
```

Running the Product

To run WSO2 products, you start the product server at the command line. You can then run the Management Console to configure and manage the product.

The Management Console uses the default HTTP-NIO transport, which is configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file. (<`PRODUCT_HOME`> is the directory where you installed the WSO2 product you want to run.) In order to access the Management Console, you must configure the HTTP-NIO transport properly in this file. For more information on the HTTP-NIO transport, see the related topics section at the bottom of this page.

The following sections describe how to run the product.

- Starting the server
- Accessing the Management Console
- Accessing the API Publisher
- Accessing the API Store
- Stopping the server
- Restricting access to the Management Console and web applications
Starting the server

Follow the instructions below to start your WSO2 product based on the operating system you use.

On Windows/Linux/Mac OS

To start the server, run `<PRODUCT_HOME>/bin/wso2server.bat` (on Windows) or `<PRODUCT_HOME>/bin/wso2server.sh` (on Linux/Mac OS) from the command prompt as described below. Alternatively, you can install and run the server as a Windows or Linux service (see the related topics section at the end of this page).

1. Open a command prompt by following the instructions below.
   - On Windows: Click **Start** > **Run**, type `cmd` at the prompt, and then press Enter.
   - On Linux/Mac OS: Establish an SSH connection to the server, log on to the text Linux console, or open a terminal window.
2. Navigate to the `<PRODUCT_HOME>/bin/` directory using the command prompt.
3. Execute one of the following commands:
   - To start the server in a typical environment:
     - On Windows: `wso2server.bat --run`
     - On Linux/Mac OS: `sh wso2server.sh`
   - To stop the server running in this mode, you will enter: `sh wso2server.sh stop`
   - To provide access to the production environment without allowing any user group (including admin) to log in to the Management Console:
     - On Windows: `wso2server.bat --run -DworkerNode`
     - On Linux/Mac OS: `sh wso2server.sh -DworkerNode`
   - To check for additional options you can use with the startup commands, type `-help` after the command, such as: `sh wso2server.sh -help` (see the related topics section at the end of this page).
4. The operation log appears in the command window. When the product server has started successfully, the log displays the message "WSO2 Carbon started in ‘n’ seconds”.

On Solaris

To start the server, run `<PRODUCT_HOME>/bin/wso2server.sh` from the command prompt as described below.

Following instructions are tested on an Oracle Solaris 10 8/11 x86 environment.

1. Click **Launch** > **Run Applications**, type `dtterm` at the prompt, and then press Enter to open a command prompt.
2. Navigate to the `<PRODUCT_HOME>/bin/` directory using the command prompt.
3. Execute the following command: `bash wso2server.sh`
4. The operation log appears in the command window. When the product server has started successfully, the log displays the message "WSO2 Carbon started in ‘n’ seconds”.

If you are starting the product in service/nohup mode in Solaris, do the following:

1. Update the `<PRODUCT_HOME>/bin/wso2server.sh` file as follows:
   a. Search for the following occurrences:
      ```
      nohup sh "$/CARBON_HOME"/bin/wso2server.sh $args > /dev/null 2>&1 &
      
      bash "$/CARBON_HOME"/bin/wso2server.sh $args > /dev/null 2>&1 &
      ```
   b. Replace those occurrences with the following: `nohup bash "$/CARBON_HOME"/bin/wso2server.sh $args > /dev/null 2>&1 &`
   
   The only change is replacing `sh` with `bash`. This is required only for Solaris.

2. Update your `PATH` variable to have `/usr/xpg4/bin/sh` as the first element. This is because `/usr/xpg4/bin/sh` contains an `sh` shell that is newer than the default `sh` shell. You can set this variable as a system property in the `wso2server.sh` script or you can run the following command on a terminal:
   ```
   export PATH=/usr/xpg4/bin/sh:$PATH
   ```
3. Start the product by following the above instructions.

Accessing the Management Console

Once the server has started, you can run the Management Console by typing its URL in a web browser. The following sections provide more information about running the Management Console:

- Working with the URL
- Signing in
Getting help

Configuring the session time-out

**Working with the URL**

The URL appears next to Mgt Console URL in the start script log that is displayed in the command window. For example:

```
[2017-12-12 18:40:34,811] INFO - StartupFinalizerServiceComponent Server : WSO2 API Manager 2.1.0
[2017-12-12 18:40:34,832] INFO - StartupFinalizerServiceComponent WSO2 Carbon started in 31 sec
[2017-12-12 18:40:34,998] INFO - JMSListener Started to listen on destination : throttleData of type topic for listener Siddhi JMS Consumer#throttleData
```

The URL should be in the following format: https://<Server Host>:9443/carbon

You can use this URL to access the Management Console on this computer from any other computer connected to the Internet or LAN. When accessing the Management Console from the same server where it is installed, you can type localhost instead of the IP address as follows: https://localhost:9443/carbon

You can change the Management Console URL by modifying the value of the `<MgtHostName>` property in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file. When the host is internal or not resolved by a DNS, map the hostname alias to its IP address in the `/etc/hosts` file of your system, and then enter that alias as the value of the `<MgtHostName>` property in carbon.xml. For example:

```
In /etc/hosts:
127.0.0.1 localhost

In carbon.xml:
<MgtHostName>localhost</MgtHostName>
```

**Signing in**

At the sign-in screen, you can sign in to the Management Console using `admin` as both the username and password.

When the Management Console sign-in page appears, the web browser typically displays an "insecure connection" message, which requires your confirmation before you can continue.

The Management Console is based on the HTTPS protocol, which is a combination of HTTP and SSL protocols. This protocol is generally used to encrypt the traffic from the client to server for security reasons. The certificate it works with is used for encryption only, and does not prove the server identity. Therefore, when you try to access the Management Console, a warning of untrusted connection is usually displayed. To continue working with this certificate, some steps should be taken to "accept" the certificate before access to the site is permitted. If you are using the Mozilla Firefox browser, this usually occurs only on the first access to the server, after which the certificate is stored in the browser database and marked as trusted. With other browsers, the insecure connection warning might be displayed every time you access the server.

This scenario is suitable for testing purposes, or for running the program on the company’s internal networks. If you want to make the Management Console available to external users, your organization should obtain a certificate signed by a well-known certificate authority, which verifies that the server actually has the name it is accessed by and that this server actually belongs to the given organization.

**Getting help**

The tabs and menu items in the navigation pane on the left may vary depending on the features you have installed. To view information about a particular page, click the Help link at the top right corner of that page, or click the Docs link to open the documentation for full information on managing the product.

**Configuring the session time-out**

If you leave the Management Console unattended for a defined time, its login session will time out. The default timeout value is 15 minutes, but you can change this in the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/WEB-INF/web.xml` file as follows:

```
<session-config>
   <session-timeout>15</session-timeout>
</session-config>
```

In products like WSO2 API Manager where web applications such as API Publisher/API Store exist, you can configure a session timeout for those web apps by changing the `repository/conf/tomcat/web.xml` file as follows:
Accessing the API Publisher

Once the server has started, you can run the API Publisher by typing its URL in a web browser. The following sections provide more information about running the API Publisher:

- Working with the URL
- Signing in

Working with the URL

The URL appears next to API Publisher Default Context in the start script log that is displayed in the command window. For example:

```
```

The URL should be in the following format: https://<Server Host>:9443/publisher

You can use this URL to access the API Publisher on this computer from any other computer connected to the Internet or LAN. When accessing the API Publisher from the same server where it is installed, you can type localhost instead of the IP address as follows:

```
https://localhost:9443/publisher
```

Signing in

At the sign-in screen, you can sign in to the API Publisher using admin as both the username and password.

When the API Publisher sign-in page appears, the web browser typically displays an “insecure connection” message, which requires your confirmation before you can continue.

The API Publisher is based on the HTTPS protocol, which is a combination of HTTP and SSL protocols. This protocol is generally used to encrypt the traffic from the client to server for security reasons. The certificate it works with is used for encryption only, and does not prove the server identity. Therefore, when you try to access the API Publisher, a warning of untrusted connection is usually displayed. To continue working with this certificate, some steps should be taken to “accept” the certificate before access to the site is permitted. If you are using the Mozilla Firefox browser, this usually occurs only on the first access to the server, after which the certificate is stored in the browser database and marked as trusted. With other browsers, the insecure connection warning might be displayed every time you access the server.

This scenario is suitable for testing purposes, or for running the program on the company’s internal networks. If you want to make the API Publisher available to external users, your organization should obtain a certificate signed by a well-known certificate authority, which verifies that the server actually has the name it is accessed by and that this server actually belongs to the given organization.

Accessing the API Store

Once the server has started, you can run the API Store by typing its URL in a web browser. The following sections provide more information about running the API Store:

- Working with the URL
- Signing in

Working with the URL

The URL appears next to API Store Default Context in the start script log that is displayed in the command window. For example:

```
```

The URL should be in the following format: https://<Server Host>:9443/store

You can use this URL to access the API Store on this computer from any other computer connected to the Internet or LAN. When accessing the API Store from the same server where it is installed, you can type localhost instead of the IP address as follows:

```
https://localhost:9443/store
```

Signing in

At the API Store home page, you can click sign in link at top right corner to sign-in to the API Publisher using admin as both the username and password.

When the API Store home page appears, the Web browser typically displays an “insecure connection” message, which requires your confirmation before you can continue.
Stopping the server

To stop the server, press Ctrl+C in the command window, or click the Shutdown/Rerstart link in the navigation pane in the Management Console. If you started the server in background mode in Linux, enter the following command instead:

```
sh <PRODUCT_HOME>/bin/wso2server.sh stop
```

Restricting access to the Management Console and web applications

You can restrict access to the management console of your product by binding the management console with selected IP addresses. Note that you can either restrict access to the management console only, or you can restrict access to all web applications in your server as explained below.

- To control access only to the management console, add the IP addresses to the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml` file as follows:

  ```xml
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
  allow="<IP-address-01>|<IP-address-02>|<IP-address-03>"/>
  ```

  The `RemoteAddrValve` Tomcat valve defined in this file will only apply to the Carbon management console, and thereby all outside requests to the management console will be blocked.

- To control access to all web applications deployed in your server, add the IP addresses to the `<PRODUCT_HOME>/repository/conf/context.xml` file as follows:

  ```xml
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
  allow="<IP-address-01>|<IP-address-02>|<IP-address-03>"/>
  ```

  The `RemoteAddrValve` Tomcat valve defined in this file will apply to each web application hosted on the Carbon server. Therefore, all outside requests to any web application will be blocked.

- You can also restrict access to particular servlets in a web application by adding a Remote Address Filter to the `web.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory), and by mapping that filter to the servlet url. In the Remote Address Filter that you add, you can specify the IP addresses that should be allowed to access the servlet.

  The following example from a `web.xml` file illustrates how access to the management page (`/carbon/admin/login.jsp`) is granted only to one IP address:

  ```xml
  <filter> <filter-name>Remote Address Filter</filter-name>
  <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class> <init-param>
  <param-name>allow</param-name> <param-value>127.0.0.1</param-value> </init-param> </filter>
  <filter-mapping> <filter-name>Remote Address Filter</filter-name>
  <url-pattern>/carbon/admin/login.jsp</url-pattern> </filter-mapping>
  ```

Any configurations (including valves defined in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file) apply to all web applications and are available globally across the server, regardless of the host or cluster. For more information on using remote host filters, see the Apache Tomcat documentation.

Related Topics

- Installing as a Windows Service
- Installing as a Linux Service
Basic Health Checks

Basic health checks can be performed on an API Manager node by connecting to relevant ports. See the following table for the ports that can be used for health checks in a fully distributed deployment.

<table>
<thead>
<tr>
<th>API Manager Profile</th>
<th>Ports that can be used for health checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway Manager</td>
<td>9763 (HTTP), 9443 (HTTPS)</td>
</tr>
<tr>
<td>Gateway Worker</td>
<td>8280 (HTTP), 8243 (HTTPS)</td>
</tr>
<tr>
<td>API Store</td>
<td>9673 (HTTP), 9443 (HTTPS)</td>
</tr>
<tr>
<td>API Publisher</td>
<td>9673 (HTTP), 9443 (HTTPS)</td>
</tr>
<tr>
<td>Traffic Manager</td>
<td>5672 (TCP), 7611 (TCP), 7711 (TCP)</td>
</tr>
<tr>
<td>Key Manager</td>
<td>9673 (HTTP), 9443 (HTTPS)</td>
</tr>
</tbody>
</table>

For more information on each profile, see API Manager Profiles.

There can be scenarios where even though the ports are responding, the Services are not properly started. It is advisable to use Service level health checks to ensure that the services are started. For example, API Manager by default is shipped with the simple axis2 service named Version. This service returns the version of the API Manager instance that is running currently.

A sample cURL command and the response from the Version service are given below.

cURL command:
```plaintext
curl -v http://<HOSTNAME>:<PORT>/services/Version
```

WSO2 API Manager Tooling

This section describes how you can use the tooling support provided via WSO2 API Manager tooling to create custom sequences and use it with the API Manager.

See the following topics for detailed information:

- **Installing the API Manager Tooling Plug-In** - install WSO2 API Manager tooling
- **Adding Mediation Extensions** - extend the default mediation flow by adding custom mediation sequences
- **Pass a Custom Authorization Token to the Backend** - pass a custom authorization token that is different to the authorization token generated for the application
- **Change the Default Mediation Flow of API Requests** - create a custom sequence and then deploy and use it in your APIs
- **Convert a JSON Message to SOAP and SOAP to JSON** - convert message types using custom sequences
- **Map the Parameters of your Backend URLs with the API Publisher URLs** - map your backend URLs to the pattern that you want in the API Publisher

**Installing the API Manager Tooling Plug-In**

The API Manager tooling plug-in gives the capabilities of a complete Eclipse-based development environment for the API Manager. You can develop services, features, and artifacts, and manage their links and dependencies through a simplified graphical editor.

You can install multiple WSO2 product tooling plug-ins on top of the same Eclipse instance.

WSO2 API Manager supports OpenJDK as well as Oracle JDK. By default, some MacOS versions are shipped with JDK 1.9 or newer, which is not compatible with WSO2 API-M Tooling. Hence, you need to override it by installing Oracle JDK 1.8 and updating the folder path of the JDK 1.8 distribution in the Eclipse distribution when installing WSO2 API-M Tooling.
For example, if you are using **MacOS High Sierra (Version 10.13.4)** with **Oracle JDK 1.8.0_151**, add the following properties to the `<ECLIPSE_HOME>/Contents/Info.plist` file.

**Format Example**

```
<string>-vm</string>
<string><JAVA_HOME>/Contents/Home/bin/java</string>
```

There are 2 possible methods you can use to install the tooling plug-in.

- **Install the plug-in with pre-packaged Eclipse** - This method uses a complete plug-in installation with **pre-packaged Eclipse**, so that you do not have to install Eclipse separately.
- **Install the plug-in on Eclipse Oxygen using the P2 URL** - This method requires you to **install Eclipse Oxygen separately** in your system if you do not have it already.

**Install the plug-in with pre-packaged Eclipse**

On the [API Manager tooling page](https://wso2iam.github.io), click **DOWNLOAD** under **Developer Studio** and click the respective link under **Full Distribution** to download the distribution according to your operating system. Note that you are prompted to enter your email address here.

**Install the plug-in on Eclipse Oxygen using the P2 URL**

1. Make sure you have **Eclipse IDE for Java EE Developers (Oxygen 3A)** installed.
   
   Note that its recommended to use Eclipse Oxygen, since the plug-in has been tested on this version.

2. Open Eclipse and on the **Help** menu click **Install New Software**.
3. Click **Add** in the dialog box that appears.
4. Add the repository by entering **API-M Tool** as the name and [http://product-dist.wso2.com/p2/developer-studio-kernel/4.3.0/apim-tools/releases/2.5.0/](http://product-dist.wso2.com/p2/developer-studio-kernel/4.3.0/apim-tools/releases/2.5.0/) as the location and click **OK**.

5. Select the required software and click **Next**. Select **WSO2 API Manager Tools** and **WSO2 ESB Tools** check box. You need to install the WSO2 ESB Tools as WSO2 API-M tooling totally depends on WSO2 Enterprise Service Bus (WSO2 ESB).

If you are using a fresh instance of Eclipse with no other **WSO2 product tooling plug-ins** installed, you need to select the Developer Studio checkboxes as well.
6. Review the items to be installed and click **Next**.
7. Read and accept the license agreements and click **Finish**.
8. If a security warning appears saying that the authenticity or validity of the software cannot be established, click **OK**.
9. Select the WSO2 Developer Studio related certificates and click **OK**.
9. Click OK to restart Eclipse to complete the installation.

**Product Administration**

WSO2 API Manager is shipped with default configurations that allow you to download, install and get started with your product instantly. However, when you go into production, it is recommended to change some of the default settings to ensure that you have a robust system that is suitable for your operational needs. Also, you may have specific use cases that require specific configurations to the server. If you are a product administrator, the follow content will provide an overview of the administration tasks that you need to perform when working with WSO2 API Manager (WSO2 API-M).

### What is the WSO2 Administration Guide?

The WSO2 Administration Guide is a WSO2 guide that is common to all WSO2 products, and it explains all you need to know about how to set up and configure a WSO2 product to meet your enterprise requirements. In the subsequent sections we point you to the WSO2 Administration Guide where necessary.

[Upgrading from a previous release] [Changing the default database] [Configuring users, roles, and permissions] [Configuring security] [Configuring multitenancy] [Configuring the registry] [Performance tuning] [Changing the default ports] [Configuring custom proxy paths] [Customizing error pages] [Customizing the management console] [Applying patches] [Monitoring the server]

---

### Upgrading from a previous release

If you are upgrading from WSO2 API Manager 2.0.0 to WSO2 API Manager 2.1.0, see the upgrading instructions for WSO2 API Manager.

### Changing the default database

By default, WSO2 products are shipped with an embedded H2 database, which is used for storing user management and registry data. We recommend that you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. when you set up your production environment. You can change the default database configuration by simply setting up a new physical database and updating the configurations in the product server to connect to that database.
For instructions on setting up and configuring databases, see the following sections in the WSO2 Administration Guide.

<table>
<thead>
<tr>
<th>Setting up the Physical Database</th>
<th>You can use the scripts provided with WSO2 API-M to install and configure several other types of relational databases. For more information, see the following sections.</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Setting up Embedded H2</td>
<td>• Setting up Remote H2</td>
</tr>
<tr>
<td>• Setting up IBM DB2</td>
<td>• Setting up IBM Informix</td>
</tr>
<tr>
<td>• Setting up MySQL</td>
<td>• Setting up Microsoft SQL</td>
</tr>
<tr>
<td>• Setting up Oracle</td>
<td>• Setting up Oracle RAC</td>
</tr>
<tr>
<td>• Setting up PostgreSQL</td>
<td>• Setting up a MySQL Cluster</td>
</tr>
<tr>
<td>• Setting up Embedded Derby</td>
<td>• Setting up Remote Derby</td>
</tr>
<tr>
<td>• Setting up MariaDB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Changing the Carbon Database</th>
<th>WSO2 API-M is shipped with an H2 database, which serves as the default Carbon database. You can change this default database to one of the standard databases listed below.</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Changing to Embedded Derby</td>
<td>• Changing to Embedded H2</td>
</tr>
<tr>
<td>• Changing to IBM DB2</td>
<td>• Changing to IBM Informix</td>
</tr>
<tr>
<td>• Changing to MariaDB</td>
<td>• Changing to MSSQL</td>
</tr>
<tr>
<td>• Changing to MySQL</td>
<td>• Changing to Oracle RAC</td>
</tr>
<tr>
<td>• Changing to Oracle</td>
<td>• Changing to PostgreSQL</td>
</tr>
<tr>
<td>• Changing to Remote H2</td>
<td></td>
</tr>
</tbody>
</table>

When changing the default database, which is H2, to any other RDBMS, be sure to change all the other required datasources (WSO2AM_DB, WSO2AM_STATS_DB, WSO2_MB_STORE_DB, WSO2_METRICS_DB, etc.) to the same RDBMS. For more information, see Changing the Default API-M Databases.

**Configuring users, roles, and permissions**

The user management feature in your product allows you to create new users and define the permissions granted to each user. You can also configure the user stores that are used for storing data related to user management.

- For more information on working with user management, see the following sections in the WSO2 Administration Guide.

<table>
<thead>
<tr>
<th>Configuring the System Administrator</th>
<th>The admin user is the super tenant who is able to manage all other users, roles and permissions in the system by using the management console of the product. Therefore, the user that has admin permissions has to be stored in the primary user store when you start the system for the first time. The documentation on setting up primary user stores explains how to configure the administrator while configuring the user store. The information under this topic explains the main configurations that are relevant to setting up the system administrator.</th>
</tr>
</thead>
</table>
## Configuring the Authorization Manager

According to the default configuration in WSO2 products, the users, roles and permissions are stored in the same repository (i.e., the default, embedded H2 database). However, you can change this configuration in such a way that the users and roles are stored in one repository (user store) and the permissions are stored in a separate repository. A user store can be a typical RDBMS, a LDAP, or an external Active Directory.

The repository that stores permissions should always be a RDBMS. The Authorization Manager configuration in the `user-mgt.xml` file connects the system to this RDBMS. The information under this topic instructs you through setting up and configuring the Authorization Manager.

## Configuring User Stores

The user management feature in WSO2 API-M allows you to maintain multiple user stores for storing users and their roles. See the following topics for instructions:

- Configuring the Primary User Store
- Configuring a JDBC User Store
- Configuring a Read-Only LDAP User Store
- Configuring a Read-Write Active Directory User Store
- Configuring a Read-Write LDAP User Store
- Configuring Secondary User Stores
- Working with Properties of User Stores
- Writing a Custom User Store Manager

## Managing Users, Roles and Permissions

The user management functionality is provided by default in WSO2 API-M, and it supports the role-based authentication model where privileges of a user are based on the role attached. For more information on managing users, roles and permissions see the following sections in the WSO2 Administration Guide.

- Changing a Password
- Configuring Roles
- Configuring Users
- Role-based Permissions
- Managing User Attributes

* For WSO2 API Manager specific user management related information, see the following sections in the WSO2 API Manager Guide.

### Configuring security

After you install WSO2 API Manager, it is recommended to change the default security settings according to the requirements of your production environment. As API Manager is built on top of the WSO2 Carbon Kernel (version 4.4.11), the main security configurations applicable to API Manager are inherited from the Carbon kernel.

For instructions on configuring security in your server, see the following topics in the WSO2 Administration Guide.

#### Important!

If you are configuring your production environment, be sure to check the Security Guidelines for Production Deployment before applying any security configurations.

### Securing Passwords in Configuration Files

All WSO2 Carbon products contain some configuration files with sensitive information such as passwords. Let’s take a look at how such plain text passwords in configuration files can be secured using the Secure Vault implementation that is built into Carbon products.

The following topics will be covered under this section:

- Encrypting Passwords with Cipher Tool
- Encrypting passwords using the automated process
- Encrypting passwords manually
- Changing encrypted passwords
- Resolving Encrypted Passwords
- Enter password in command-line
- Start server as a background job
- Carbon Secure Vault Implementation
- Customizing the Secure Vault configuration
  - Creating a Secret Callback Handler
  - Creating a custom Secret Repository
## Configuring Transport-Level Security

The transport level security protocol of the Tomcat server is configured in the `<PRODUCT_HOME>/core/conf/tomcat/catalina-server.xml` file. Note that the `sslProtocol` attribute is set to "TLS" by default.

The following topics will guide you through the configuration options:
- Enabling TLS and disabling SSL support
- Disabling SSL support for products with JDK 1.8
- Disabling weak ciphers
- Disabling weak ciphers for the Tomcat transport
- Enables weak ciphers for the PassThrough transport
- Changing the server name in HTTP response headers

## Enabling Java Security Manager

The Java Security Manager is used to define various security policies that prevent untrusted code from manipulating your system. Enabling the Java Security Manager for WSO2 products activates the Java permissions that are in the `<PRODUCT_HOME>/core/repository/conf/sec.policy` file. You modify this file to change the Java security permissions as required.

## Using Asymmetric Encryption

WSO2 products use asymmetric encryption by default for the purposes of authentication and data encryption. In asymmetric encryption, keystores (with key pairs and certificates) are created and stored for the product. It is possible to have multiple keystores so that the keys used for different use cases are kept unique. The following topics explain more details on keystores.
- Understanding keystores
- Setting up keystores for WSO2 products
- Default keystore settings in WSO2 products
- Managing keystores

## Using Symmetric Encryption

WSO2 Carbon-based products use **asymmetric encryption** by default as explained in the previous section. From Carbon 4.4.3 onwards, you have the option of switching to **symmetric encryption** in your WSO2 product. Using symmetric encryption means that a single key will be shared for encryption and decryption of information.

## Mitigating Cross Site Request Forgery Attacks

Cross Site Request Forgery (CSRF) attacks trick you to send a malicious request, by forcing you to execute unwanted actions on an already authenticated web browser. For more information, see the following sections that describe the impact of the Cross Site Request Forgery (CSRF) attack and how to mitigate it.
- How can CSRF attacks be harmful?
- Mitigating CSRF attacks
  - Securing web applications
  - Securing Jaggery applications

## Mitigating Cross Site Scripting Attacks

Cross Site Scripting (XSS) attacks use web applications to inject malicious scripts or a malicious payload, generally in the form of a client side script, into trusted legitimate web applications. For more information, see the following sections that describe the impact of XSS attack and the approaches you can use to mitigate it.
- How can XSS attacks be harmful?
- Mitigating XSS attacks

## Enabling HostName Verification

Hostname verification is enabled in WSO2 products by default, which means that when a hostname is being accessed by a particular client, it will be verified against the hostname specified in the product's SSL certificate.

## Configuring TLS Termination

When you have Carbon servers fronted by a load balancer, you have the option of terminating SSL for HTTPS requests. This means that the load balancer will be decrypting incoming HTTPS messages and forwarding them to the Carbon servers as HTTP. This is useful when you want to reduce the load on your Carbon servers due to encryption. To achieve this, the load balancer should be configured with TLS termination and the Tomcat RemoteIpValve should be enabled for Carbon servers.

## Configuring multitenancy

You can create multiple tenants in your product server, which will allow you to maintain tenant isolation in a single server/cluster. For instructions on configuring multiple tenants for your server, see Working with Multiple Tenants in the WSO2 Administration Guide and to understand how multitenancy works with the API Store, see Managing Tenants.
Configuring the registry

A registry is a content store and a metadata repository for various artifacts such as services, WSDLs and configuration files. In WSO2 products, all configurations pertaining to modules, logging, security, data sources and other service groups are stored in the registry by default.

For instructions on setting up and configuring the registry for your server, see Working with the Registry in the WSO2 Administration Guide.

Performance tuning

You can optimize the performance of your WSO2 server by using configurations and settings that are suitable to your production environment. At a basic level, you need to have the appropriate OS settings, JVM settings etc. Since WSO2 products are all based on a common platform called Carbon, most of the OS, JVM settings recommended for production are common to all WSO2 products. Additionally, there will be other performance enhancing configuration recommendations that will depend on very specific features used by your product.

For instructions on the Carbon platform-level performance tuning recommendations, see Performance Tuning in the WSO2 Administration Guide.

For instructions on performance tuning recommendations that are specific to WSO2 API Manager, see tuning performance in the WSO2 API-M Guide.

Changing the default ports

When you run multiple WSO2 products, multiple instances of the same product, or multiple WSO2 product clusters on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts.

What is port offset?

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be offset. For example, if the HTTP port is defined as 9763 and the portOffset is 1, the effective HTTP port will be 9764. Therefore, for each additional WSO2 product, instance, or cluster you add to a server, set the port offset to a unique value (the default is 0).

For the list of ports in all WSO2 products, see Default Ports of WSO2 Products in the WSO2 Administration Guide.

For instructions on configuring ports, see Changing the Default Ports in the WSO2 Administration Guide.

Configuring custom proxy paths

This feature is particularly useful when multiple WSO2 products (fronted by a proxy server) are hosted under the same domain name. By adding a custom proxy path you can host all products under a single domain and assign proxy paths for each product separately.

Click here to see how to add custom proxy paths...

Before you begin...

Follow steps 1 to 8 in the Administration guide, to install Nginx if you haven't done so already.

The sample usecase to route requests is shown below.
1. Open the /etc/nginx/sites-enabled/wso2 file and enter the following configurations.

Some of the Nginx configuration syntaxes may slightly vary based on the Nginx version you use. The configurations below are tested with version 1.15.0.
server {
    listen 443 ssl;
    ssl_certificate    nginx.crt;
    ssl_certificate_key nginx.key;

    location ~ ^/apimanager/publisher/(.*)registry/(.*)$ {
        index index.html;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass https://127.0.0.1:9443/$1registry/$2;
    }

    location /apimanager/publisher {
        index index.html;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass https://localhost:9443/publisher;
        proxy_redirect https://localhost:9443/publisher
                          https://localhost/apimanager/publisher;
        proxy_cookie_path /publisher /apimanager/publisher;
    }

    location ~ ^/apimanager/store/(.*)registry/(.*)$ {
        index index.html;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass https://127.0.0.1:9443/$1registry/$2;
    }

    location /apimanager/store {
        index index.html;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass https://localhost:9443/store;
        proxy_redirect https://localhost:9443/store
                          https://localhost/apimanager/store;
        proxy_cookie_path /store /apimanager/store;
    }

    location /apimanager/admin {
        index index.html;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass https://localhost:9443/admin;
        proxy_redirect https://localhost:9443/admin
                          https://localhost/apimanager/admin;
        proxy_cookie_path /admin /apimanager/admin;
    }
}

2. Configure the reverseProxy for the following.

Open the <API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json file and do the following configurations.
Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` file and do the following configurations.

```
"reverseProxy": {
    "enabled": true, // values true, false, "auto" - will look for X-Forwarded-* headers
    "host": "localhost", // If reverse proxy do not have a domain name use IP
    "context": "/apimanager/publisher", // Use only if different path is used for registry
    "regContext": "" // Use only if different path is used for registry
},
```

Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json` file and do the following configurations.

```
"reverseProxy": {
   "enabled": true, // values true, false, "auto" - will look for X-Forwarded-* headers
   "host": "localhost", // If reverse proxy do not have a domain name use IP
   "context": "/apimanager/admin" // Use only if different path is used for registry
},
```

3. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file. Set the URL as shown below, to get the View in Store link on API overview tab in the API Publisher.

```
<APIStore>
  ..........
  <URL>https://localhost/apimanager/store</URL>
  ..........
</APIStore>
```

You have now added custom proxy paths for the API Publisher, API Store, and Admin Portal.

For instructions on configuring custom proxy paths, see Adding a Custom Proxy Path in the WSO2 Administration Guide.

---

Customizing error pages

You can make sure that sensitive information about the server is not revealed in error messages, by customizing the error pages in your product.

For instructions, see Customizing Error Pages in the WSO2 Administration Guide.

---

Customizing the management console

Some of the WSO2 products, such as WSO2 API Manager consist of a web user interface named the management console. This allows administrators to configure, monitor, tune, and maintain the product using a simple interface. You can customize the look and feel of the management console for your product.

For instructions, see Customizing the Management Console in the WSO2 Administration Guide.
Applying patches

For information on updating WSO2 API-M with the latest available patches (issued by WSO2) using the WSO2 Update Manager (WUM) tool, see Getting Started with WUM in the WSO2 Administration Guide.

Monitoring the server

Monitoring is an important part of maintaining a product server. Listed below are the monitoring capabilities that are available for WSO2 API Manager.

- Monitoring server logs: A properly configured logging system is vital for identifying errors, security threats and usage patterns in your product server. For instructions on monitoring the server logs, see Monitoring Logs in the WSO2 Administration Guide.
- Monitoring using WSO2 metrics: WSO2 API Manager is shipped with JVM Metrics, which allows you to monitor statistics of your server using Java Metrics. For instructions on setting up and using Carbon metrics for monitoring, see Using WSO2 Metrics in the WSO2 Administration Guide.
- JMX-based monitoring: For information on monitoring your server using JMX, see JMX Monitoring in the WSO2 Administration Guide.

Upgrading from the Previous Release

The following information describes how to upgrade your WSO2 API Manager server (WSO2 API-M) from APIM 1.8.0/1.9.0/1.9.1/2.0.0/2.1.0/2.2.0/2.5.0 to 2.6.0.

To upgrade from a version older than 1.8.0, follow the instructions in the document that was released immediately after your current release and upgrade incrementally.

Before you begin

This release is a WUM-only release. This means that there are no manual patches. Any further fixes or latest updates for this release can be updated through the WSO2 Update Manager (WUM).

- If you are upgrading to this version, in order to use this version in your production environment, use the WSO2 Update Manager and get the latest available updates for WSO2 API Manager 2.6.0. For more information on how to do this, see Updating WSO2 Products.

Upgrading from 2.5.0 to 2.6.0

Follow the instructions below to upgrade your WSO2 API Manager server from WSO2 API-M 2.5.0 to 2.6.0.

If you are using WSO2 Identity Server (WSO2 IS) as a Key Manager, follow the instructions in Upgrading from the Previous Release when WSO2 IS is the Key Manager.

- Step 1 - Migrate the WSO2 API-M configurations
- Step 2 - Upgrade WSO2 API-M to 2.6.0
- Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics
  - Step 3.1 - Configure WSO2 API-M Analytics 2.6.0
  - Step 3.2 - Configure WSO2 API-M 2.6.0 for Analytics
- Step 4 - Start the WSO2 API-M server

Step 1 - Migrate the WSO2 API-M configurations

Do not copy entire configuration files from the current version of WSO2 API Manager to the new one, as some configuration files (e.g., api-manager.xml) may have changed. Instead, redo the configuration changes in the new configuration files.

Follow the instructions below to move all the existing API Manager configurations from the current environment to the new one.

1. Back up all databases in your API Manager instances along with the Synapse configurations of all the tenants and the super tenant.
   - The Synapse configurations of the super tenant are in the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory.
   - The Synapse configurations of tenants are in the `<CURRENT_API-M_HOME>/repository/tenants` directory.
1. **If you use a clustered/distributed API Manager setup**, back up the available configurations in the API Gateway node.

2. Download API Manager 2.6.0 from [https://wso2.com/api-management](https://wso2.com/api-management).

3. Open the `<API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases. You can copy the configuration values from the same file in the current API Manager instance that you are already using.
   - User Store
   - Registry database/s
   - API Manager databases
   - Statistics database (if statistics are configured)

4. Edit the registry configurations in the `<API-M_2.6.0_HOME>/repository/conf/registry.xml` file and the user database in the `<API-M_2.6.0_HOME>/repository/conf/user-mgt.xml` file, similar to the configurations of the current WSO2 API Manager.

   If you are working with a **clustered/distributed API Manager setup**, follow step 5 on the Gateway node.

5. Move all your Synapse configurations to API-M 2.6.0 pack.
   - Move your Synapse super tenant configurations.
     Copy the contents in the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory and replace the contents in the `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default` directory with the copied contents.
   - Move all your tenant Synapse configurations.
     Copy the contents in the `<CURRENT_API-M_HOME>/repository/tenants` directory and replace the contents in the `<API-M_2.6.0_HOME>/repository/tenants` directory with the copied contents.

**Step 2 - Upgrade WSO2 API-M to 2.6.0**

1. Stop all WSO2 API Manager server instances that are running.
2. Make sure you backed up all the databases and Synapse configurations as instructed in step 1 of the previous section.
3. Upgrade the WSO2 API Manager database from version 2.5.0 to version 2.6.0.
   a. Download `apimgt-db-migration-scripts-2.5to2.6.zip` and extract it.
   b. Execute the relevant database script, from the scripts that are provided in the extracted directory, on the `WSO2AM_DB` database.
4. Upgrade the Identity component in WSO2 API Manager from version 5.6.0 to 5.7.0.

   As WSO2 API-M shares identity components with WSO2 Identity Sever (WSO2 IS), this step is necessary to upgrade those components (even if you are not using WSO2 IS as a Key Manager).

   a. Download the `wso2is-5.x.x-migration.zip` and extract it.
   b. Copy the `migration-resources` folder from the extracted folder to the `<API-M_2.6.0_HOME>` directory.
   c. Open the `migration-config.yaml` file in the `migration-resources` directory and make sure that the `currentVersion` element is set to 5.6.0, as shown below.

   ```yaml
   migrationEnable: "true"
   currentVersion: "5.6.0"
   migrateVersion: "5.7.0"
   ```

   Make sure you have enabled migration by setting the `migrationEnable` element to `true` as shown above.

   d. Copy the `org.wso2.carbon.is.migration-5.7.0.jar` from the extracted folder to the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.
   e. Copy the keystores (i.e., `client-truststore.jks`, `wso2carbon.jks` and any other custom JKS) used in the previous version and replace the existing keystores in the `<API-M_2.6.0_HOME>/repository/resources/security` directory.
   f. Start WSO2 API Manager 2.6.0 as follows to carry out the complete Identity component migration.

   - [Linux](#)
   - [Mac OS](#)
   - [Windows](#)
If you run WSO2 API-M as a Windows Service when doing the identity component migration, then you need to add the following parameters in the command line arguments section (CMD_LINE_ARGS) of the wso2server.bat file.

```bash
-Dmigrate -Dcomponent=identity
```

Please note that depending on the number of records in the identity tables, this identity component migration will take a considerable amount of time to finish. Do not stop the server during the migration process and please wait until the migration process finish completely and server get started.

### Troubleshooting

When running the above step if you encounter the following error message, follow the steps in this section. Note that this error could occur only if the identity tables contain a huge volume of data.

**Sample exception stack trace**

```
ERROR {org.wso2.carbon.registry.core.dataaccess.TransactionManager} - Failed to start new registry transaction.
{org.wso2.carbon.registry.core.dataaccess.TransactionManager}
Timeout: Pool empty. Unable to fetch a connection in 60 seconds, none available[size:50; busy:50; idle:0; lastwait:60000
```

1. Change the value in `<indexingFrequencyInSeconds>` in `<API-M_HOME>/repository/conf/registry.xml` file to a higher value (e.g., 10)
2. Re-run the command above.

**Make sure to revert the change done in Step 1, after the migration is complete.**

- After you have successfully completed the migration, stop the server and remove the following files and folders.
  - Remove the `org.wso2.carbon.is.migration-5.7.0.jar` file, which is in the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.
  - Remove the `migration-resources` directory, which is in the `<API-M_2.6.0_HOME>` directory.
  - If you ran WSO2 API-M as a Windows Service when doing the identity component migration, then you need to remove the following parameters in the command line arguments section (CMD_LINE_ARGS) of the wso2server.bat file.

```bash
-Dmigrate -Dcomponent=identity
```

5. Re-index the artifacts in the registry.
   - Run the `reg-index.sql` script against the `REG_DB` database.
Please note that depending on the number of records in the REG_LOG table, this script will take a considerable amount of time to finish. Do not stop the execution of script until it is completed.

b. Add the `tenantloader-1.0.jar` to `<API-M_2.6.0_HOME>/repository/components/dropins` directory

c. Rename the `<lastAccessTimeLocation>` element in the `<API-M_2.6.0_HOME>/repository/conf/registry.xml` file.

d. If you use a clustered/distributed API Manager setup, change the file in the API Publisher node. For example, change the `/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccess time registry path to /_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime _1`

e. If the `<API-M_2.6.0_HOME>/solr` directory exists, take a backup and thereafter delete it.

f. Start the WSO2 API-M server.

g. Stop the WSO2 API-M server and remove the `tenantloader-1.0.jar` from the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.

**Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics**

This step is only required if you have WSO2 API-M-Analytics configured in your current deployment.

Follow the steps below to migrate APIM Analytics 2.5.0 to APIM Analytics 2.6.0

### Step 3.1 - Configure WSO2 API-M Analytics 2.6.0

1. Download the WUM updated pack for WSO2 API Manager Analytics 2.6.0.

   Note that it is mandatory to use a WUM updated WSO2 API Manager Analytics 2.6.0 pack when migrating the configurations for WSO2 API-M Analytics.

2. Create a new database for the new statistics database and configure it in the `conf/worker/deployment.yaml` as follows:

   In this example shows to configure MySQL database configurations.

   ```yaml
   - name: APIM_ANALYTICS_DB
     description: "The datasource used for APIM statistics aggregated data."
     jndiConfig:
       name: jdbc/APIM_ANALYTICS_DB
       definition:
         type: RDBMS
         configuration:
           jdbcUrl: 'jdbc:mysql://localhost:3306/spDatabase?autoReconnect=true'
           username: username
           password: password
           driverClassName: com.mysql.jdbc.Driver
           maxPoolSize: 50
           idleTimeout: 60000
           connectionTestQuery: SELECT 1
           validationTimeout: 30000
           isAutoCommit: true
   ```

   In previous versions of WSO2 API-M, even though the `defaultAutoCommit` value was set to false in the `STAT_DB`, when performing Analytics migration in WSO2 API-M Analytics 2.6.0, you need to set the `isAutoCommit` value to true in the `APIM_ANALYTICS_DB` datasource.
configuration.

3. Copy the relevant JDBC driver to the `<APIM_ANALYTICS_2.6.0_HOME>/lib` folder.

4. Start the new API-M Analytics server.

```shell
cd <APIM_ANALYTICS_2.6.0_HOME>/bin
sh worker.sh
```

When the WSO2 API-M Analytics 2.6.0 server starts, the new statistics tables will get generated in the database that you configured in step 3.1 - (2.) above.

**Step 3.2 - Configure WSO2 API-M 2.6.0 for Analytics**

Follow the instructions below to configure WSO2 API Manager for the WSO2 API-M Analytics migration in order to migrate the statistics related data.

1. Configure the statistics related datasources for WSO2 API Manager Analytics.

Configure the following 3 datasources in the `<API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file.

The following 3 datasources should be configured in the `<API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file **only until the stats migration is complete**. After the statistics migration is complete remove the APIM_ANALYTIC S_DB datasource configuration, which was added for the new statistics database, from the latter mentioned file.

The following is an example of how the configurations should be defined when using MySQL.

- The first datasource points to the [previous API-M version’s AM_DB datasource](#).

```xml
<datasource>
    <name>WSO2AM_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/AM_DB?autoReconnect=true</url>
            <username>username</username>
            <password>password</password>
            <defaultAutoCommit>true</defaultAutoCommit>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

- The second datasource points to the WSO2 Data Analytics (WSO2 DAS) based previous datasource WSO2AM_STATS_DB for statistics.

```xml
<datasource>
    <name>WSO2AM_STATS_DB</name>
    <description>The datasource used for WSO2 Data Analytics</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_STATS_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/WSO2AM_STATS_DB?autoReconnect=true</url>
            <username>username</username>
            <password>password</password>
            <defaultAutoCommit>true</defaultAutoCommit>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```
1. Download and extract the WSO2 API Manager migration client and carry out the following steps to upgrade to WSO2 API Manager Analytics 2.6.0.

2. Copy the migrate.client-2.6.x-1.jar to the <API-M_2.6.0_HOME>/repository/components/dropins folder.

3. Copy the relevant JDBC driver to the <API-M_2.6.0_HOME>/repository/components/lib folder.

4. Make sure that WSO2 API-M Analytics is disabled in the <API-M_2.6.0_HOME>/repository/conf/api-manager.xml file.

```
<Analytics>
  <Enabled>false</Enabled>
  ...
</Analytics>
```
6. After setting the above configurations in place, start up the WSO2 API-M 2.6.0 server with the following commands.

```bash
cd <API-M_2.6.0_HOME>/bin
sh wso2server.sh -DmigrateStats=true
```

## Table-wise Migration

The migration client provides the support for table-wise migration.

The following are the table names that needs to be migrated.

- ApiPerDestinationAgg
- ApiResPathPerApp
- ApiVersionPerAppAgg
- ApiLastAccessSummary
- ApiFaultInvocationAgg
- ApiUserBrowserAgg
- GeoLocationAgg
- ApiExeTimeDay
- ApiExeTimeHour
- ApiExeTimeMinute
- ApiThrottledOutAgg
- APIM_ReqCountAgg
- ApiUserPerAppAgg

You need to use the `statTable` migration property for table wise migration.

For example, if you need to migrate only the `ApiPerDestinationAgg` and `ApiResPathPerApp` tables use the following command:

```bash
sh wso2server.sh -DmigrateStats=true -DstatTable=ApiPerDestinationAgg,ApiResPathPerApp
```

When you need to migrate multiple tables, the comma separated table names should without spaces before or after the comma as shown below.

```bash
sh wso2server.sh -DmigrateStats=true -DstatTable=ApiPerDestinationAgg,ApiResPathPerApp
```

If the default 3 datasource names/name, which you specified above, changed from the details that you specified in the `<API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file. Use the following command to pass the values to the migration client.

- Click here to see a code snippet of the definition of the datasource name...

```xml
<jndiConfig>
  <name>jdbc/WSO2AM_DB</name>
</jndiConfig>
```

**Format Example**

```bash
sh wso2server.sh -DdataSource=<AM_DATASOURCE_NAME>,<OLD_STAT_DATASOURCE_NAME>,<NEW_STAT_DATASOURCE_NAME>
```

```bash
sh wso2server.sh -DdataSource=jdbc/WSO2AM_DB,jdbc/WSO2AM_STATS_DB,jdbc/APIM_ANALYTICS_DB
```

The following warning message could occur during migration if the relevant consumer key does not exist in the `AM_APPLICATION_KEY_MAPPING` table due to Application deletion.
7. Stop the WSO2 API-M server and remove the migration JAR copied under Step 3.2 -(3.) above.
8. Remove both the old and new STAT_DB datasources from the <API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml file, which you defined in step 3.2 - (1.).
9. Enable analytics in WSO2 API-M by setting the following configuration to true in the <API-M_2.6.0_HOME>/repository/conf/api-manager.xml file.

```xml
<Analytics>
  <Enabled>true</Enabled>
  ...
</Analytics>
```


```
   cd <API-M_2.6.0_HOME>/bin
   sh wso2server.sh
```

**Step 4 - Start the WSO2 API-M server**

Linux / Mac OS Windows

```
   sh wso2server.sh
   wso2server.bat
```

Follow the instructions below to upgrade your WSO2 API Manager server from **WSO2 API-M 2.2.0 to 2.6.0**.

If you are using WSO2 Identity Server (WSO2 IS) as a Key Manager, follow the instructions in Upgrading from the Previous Release when WSO2 IS is the Key Manager.

- Step 1 - Migrate the WSO2 API-M configurations
- Step 2 - Upgrade WSO2 API-M to 2.6.0
- Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics
- Step 4 - Restart the WSO2 API-M server

**Step 1 - Migrate the WSO2 API-M configurations**

Do not copy entire configuration files from the current version of WSO2 API Manager to the new one, as some configuration files (e.g., api-manager.xml) may have changed. Instead, redo the configuration changes in the new configuration files.

Follow the instructions below to move all the existing API Manager configurations from the current environment to the new one.

1. Back up all databases in your API Manager instances along with the Synapse configs of all the tenants and the super tenant.  
   - The Synapse configs of the super tenant are in the <CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default directory.
   - The Synapse configs of tenants are in the <CURRENT_API-M_HOME>/repository/tenants directory.
   - If you use a clustered/distributed API Manager setup, back up the available configurations in the API Gateway node.
2. Download WSO2 API Manager 2.6.0 from [https://wso2.com/api-management/](https://wso2.com/api-management/).
3. Open the <API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml file and provide the datasource configurations for the following databases. You can copy the configuration values from the same file in the current WSO2 API Manager instance already being used.
   - User Store
4. Edit the registry configurations in the `<API-M_2.6.0_HOME>/repository/conf/registry.xml` file and the user database in the `<API-M_2.6.0_HOME>/repository/conf/user-mgt.xml` file, similar to the configurations of the current API Manager.

If you are working with a clustered/distributed API Manager setup, follow step 5 on the Gateway node.

5. Move all your Synapse configurations to API-M 2.6.0 pack.
   - Move your Synapse super tenant configurations.
     Copy the contents in the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory and replace the contents in the `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default` directory with the copied contents.
   - Move all your tenant Synapse configurations.
     Copy the contents in the `<CURRENT_API-M_HOME>/repository/tenants` directory and replace the contents in the `<API-M_2.6.0_HOME>/repository/tenants` directory with the copied contents.

Step 2 - Upgrade WSO2 API-M to 2.6.0

1. Stop all WSO2 API Manager server instances that are running.
2. Make sure you backed up all the databases and Synapse configs, as instructed in step 1 of the previous section.
3. To start the migration process, run the respective migration script based on your environment.

   ```bash
   ./apim220_to_apim260_gateway_artifact_migrator.sh <API-definitions-path>
   ```

   `<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.2.0 deployment, reside.

   The API definition paths `<API-definitions-path>` are as follows:
   - Super Tenant - `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default`
   - Tenant - `<API-M_2.6.0_HOME>/repository/tenants`

   Where, `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`. 

   ```
   Super Tenant
   Tenants
   ```
<table>
<thead>
<tr>
<th>Run the PowerShell script <code>apim220_to_apim260_gateway_artifact_migrator.ps1</code> as shown below, to migrate from WSO2 API Manager 2.2.0 to 2.6.0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Open a Windows command prompt and type the following command.</td>
</tr>
<tr>
<td><code>powershell</code></td>
</tr>
<tr>
<td>A message about PowerShell appears, and the shell changes to PowerShell (PS).</td>
</tr>
<tr>
<td>b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.</td>
</tr>
<tr>
<td><code>.\apim220_to_apim260_gateway_artifact_migrator.ps1 &lt;API-definitions-path&gt;</code></td>
</tr>
<tr>
<td>&lt;API-definitions-path&gt; - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.2.0 deployment, reside.</td>
</tr>
<tr>
<td>• Super Tenant - <code>&lt;API-M_2.6.0_HOME&gt;/repository/deployment/server/synapse-configs/default&gt;</code></td>
</tr>
<tr>
<td>Where <code>&lt;API-M_2.6.0_HOME&gt;</code> can be, for example, <code>/Users/user12/Documents/wso2am-2.6.0</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Run the PowerShell script <code>apim220_to_apim260_gateway_artifact_migrator_for_tenants.ps1</code> as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Open a Windows command prompt and type the following command.</td>
</tr>
<tr>
<td><code>powershell</code></td>
</tr>
<tr>
<td>A message about PowerShell appears, and the shell changes to PowerShell (PS).</td>
</tr>
<tr>
<td>b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.</td>
</tr>
<tr>
<td><code>.\apim220_to_apim260_gateway_artifact_migrator_for_tenants.ps1 &lt;API-definitions-path&gt;</code></td>
</tr>
<tr>
<td>&lt;API-definitions-path&gt; - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.2.0 deployment, reside.</td>
</tr>
<tr>
<td>• Tenants - <code>&lt;API-M_2.6.0_HOME&gt;/repository/tenants</code></td>
</tr>
<tr>
<td>Where <code>&lt;API-M_2.6.0_HOME&gt;</code> can be, for example, <code>/Users/user12/Documents/wso2am-2.6.0</code>.</td>
</tr>
</tbody>
</table>

It may take a considerable amount of time, which is proportionate to the amount of artifacts, to complete the migration process.
4. Upgrade the API Manager database from version 2.2.0 to version 2.6.0.
   a. Download `apimgt-db-migration-scripts-2.2to2.6.zip` and extract it.
   b. Execute the relevant database script, from the scripts that are provided in the extracted directory, on the `WSO2AM_DB` database.

5. Upgrade the Identity component in WSO2 API Manager from version 5.5.0 to 5.7.0.
   a. Download the `wso2is-5.x.x-migration.zip` and extract it.
   b. Copy the `migration-resources` folder from the extracted folder to the `<API-M_2.6.0_HOME>` directory.
   c. Open the `migration-config.yaml` file in the `migration-resources` directory and make sure define the `currentVersion` element to 5.5.0, as shown below.

   ```yaml
   migrationEnable: "true"
   currentVersion: "5.5.0"
   migrateVersion: "5.7.0"
   ```

   Make sure you have enabled migration by setting the `migrationEnable` element to true as shown above.

d. Copy the `org.wso2.carbon.is.migration-5.7.0.jar` from the extracted folder to the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.
e. Copy the keystores (i.e., `client-truststore.jks`, `wso2carbon.jks` and any other custom JKS) used in the previous version and replace the existing keystores in the `<API-M_2.6.0_HOME>/repository/resources/security` directory.
f. Start WSO2 API Manager 2.6.0 via the terminal as follows to carry out the complete Identity component migration.
   ```
   sh wso2server.sh -Dmigrate -Dcomponent=identity
   wso2server.bat -Dmigrate -Dcomponent=identity
   ```

If you run WSO2 API-M as a Windows Service when doing the identity component migration, then you need to add the following parameters in the command line arguments section (`CMD_LINE_ARGS`) of the `wso2server.bat` file.

```
-Dmigrate -Dcomponent=identity
```
Troubleshooting

When running the above step if you encounter the following error message, please follow the steps in this section. Please note that this error could occur only if the identity tables contain a huge volume of data.

Sample exception stack trace

```
ERROR {org.wso2.carbon.registry.core.dataaccess.TransactionManager} - Failed to start new registry transaction.
{org.wso2.carbon.registry.core.dataaccess.TransactionManager}
    Timeout: Pool empty. Unable to fetch a connection in 60 seconds, none available[size:50; busy:50; idle:0; lastwait:60000
```

1. Change the value in `<indexingFrequencyInSeconds>` in `<API-M_HOME>/repository/conf/registry.xml` to a higher value (e.g., 10)

2. Re-run the command above.

Make sure to revert the change done in Step 1, after the migration is complete.

Please note that depending on the number of records in the REG_LOG table, this script will take a considerable amount of time to finish. Do not stop the execution of script until it is completed.

Troubleshooting

When running the above step if you encounter the following error message, please follow the steps in this section. Please note that this error could occur only if the identity tables contain a huge volume of data.

Sample exception stack trace

```
ERROR {org.wso2.carbon.registry.core.dataaccess.TransactionManager} - Failed to start new registry transaction.
{org.wso2.carbon.registry.core.dataaccess.TransactionManager}
    Timeout: Pool empty. Unable to fetch a connection in 60 seconds, none available[size:50; busy:50; idle:0; lastwait:60000
```

1. Change the value in `<indexingFrequencyInSeconds>` in `<API-M_HOME>/repository/conf/registry.xml` to a higher value (e.g., 10)

2. Re-run the command above.

Make sure to revert the change done in Step 1, after the migration is complete.

Please note that depending on the number of records in the REG_LOG table, this script will take a considerable amount of time to finish. Do not stop the execution of script until it is completed.

b. Add the `tenantloader-1.0.jar` to `<API-M_2.6.0_HOME>/repository/components/dropins` directory.

c. Rename the `<lastAccessTimeLocation>` element in the `<API-M_2.6.0_HOME>/repository/conf/registry.xml` file.

d. If you use a clustered/distributed API Manager setup, change the file in the API Publisher node.

For example, change the `/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime` registry path to `/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime_1`

e. If the `<API-M_2.6.0_HOME>/solr` directory exists, take a backup and thereafter delete it.

f. Start the WSO2 API-M server.

g. Stop the WSO2 API-M server and remove the `tenantloader-1.0.jar` from the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.

Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics
Step 4 - Restart the WSO2 API-M server

Windows

sh wso2server.sh

wso2server.bat

Follow the instructions below to upgrade your WSO2 API Manager server from WSO2 API-M 2.1.0 to 2.6.0.

This step is only required if you have WSO2 API-M-Analytics configured in your current deployment.

As you are upgrading from WSO2 API-M Analytics 2.2.0, in order migrate the configurations required to run WSO2 API-M Analytics for WSO2 API-M 2.6.0 carryout the same instructions as mentioned in Upgrading from 2.5.0 to 2.6.0 - Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics.

Step 1 - Migrate the WSO2 API-M configurations

Follow the instructions below to move all the existing API Manager configurations from the current environment to the new one.

Do not copy entire configuration files from the current version of WSO2 API Manager to the new one, as some configuration files (e.g., api-manager.xml) may have changed. Instead, redo the configuration changes in the new configuration files.

1. Back up all databases in your API Manager instances along with the Synapse configs of all the tenants and the super tenant.
   - The Synapse configs of the super tenant are in the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory.
   - The Synapse configs of tenants are in the `<CURRENT_API-M_HOME>/repository/tenants` directory.
   - If you use a clustered/distributed API Manager setup, back up the available configurations in the API Gateway node.
2. Download API Manager 2.6.0 from https://wso2.com/api-management/.
3. Open the `<API-M-2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases.
   - User Store
   - Registry database/s
   - API Manager databases
   - Statistics database (if statistics are configured)
4. Edit the registry configurations in the `<API-M-2.6.0_HOME>/repository/conf/registry.xml` file and the user database in the `<API-M-2.6.0_HOME>/repository/conf/user-mgt.xml` file, similar to the configurations of the current API Manager.

   In a clustered/distributed API Manager setup, follow steps 5 and 6 for the Gateway node.

5. Move all your Synapse configurations with the exception of the files mentioned below, by copying the contents in the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory and by using it to replace the contents of the `<API-M-2.6.0_HOME>/repository/deployment/server/synapse-configs/default` directory.

If you are using WSO2 Identity Server (WSO2 IS) as a Key Manager, follow the instructions in Upgrading from the Previous Release when WSO2 IS is the Key Manager.

Step 2 - Upgrade WSO2 API-M to 2.6.0

Step 3 - Optionally, migrate the configurations for API-M Analytics

Step 4 - Restart the WSO2 API-M server
6. Move all your tenant Synapse configurations by copying the contents in the `<CURRENT_API-M_HOME>/repository/tenants` directory and adding the copied contents to the `<API-M_2.6.0_HOME>/repository/tenants` directory.

**NOTE:** Do not replace the files listed below from the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` folder to API Manager 2.6.0. These are application-specific APIs and sequences. If you made any custom changes to the files below, you need to merge the changes to the corresponding files in 2.6.0.

/api/_AuthorizeAPI_.xml
/api/_RevokeAPI_.xml
/api/_TokenAPI_.xml
/api/_UserInfoAPI_.xml
/sequences/_auth_failure_handler_.xml
/sequences/_build_.xml
/sequences/_cors_request_handler_.xml
/sequences/fault.xml
/sequences/main.xml
/sequences/_production_key_error_.xml
/sequences/_resource_mismatch_handler_.xml
/sequences/_sandbox_key_error_.xml
/sequences/_throttle_out_handler_.xml
/sequences/_token_fault_.xml
/proxy-services/WorkflowCallbackService.xml

**NOTE:** Copy the files listed below from the `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory and replace the corresponding files in the `<API-M_2.6.0_HOME>/repository/tenants/` directory.

_sequence/_auth_failure_handler_.xml
_sequence/_cors_request_handler_.xml
_sequence/fault.xml
_sequence/main.xml
_sequence/_production_key_error_.xml
_sequence/_resource_mismatch_handler_.xml
_sequence/_sandbox_key_error_.xml
_sequence/_throttle_out_handler_.xml

**NOTE:** In a clustered/distributed API Manager setup, do this step on the Gateway node.

**Step 2 - Upgrade WSO2 API-M to 2.6.0**

1. Stop all WSO2 API Manager server instances that are running.
2. Make sure you backed up all the databases and Synapse configs, as instructed in step 1 of the previous section.
3. To start the migration process, run the respective migration script based on your environment.

   **Linux/Mac OS/Windows**

   Run the `apim210_to_apim260_gateway_artifact_migrator.sh` script, as shown below, to migrate from WSO2 API Manager 2.1.0 to 2.6.0.

   ```bash
   ./apim210_to_apim260_gateway_artifact_migrator.sh <API-definitions-path>
   ```

   `<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the WSO2 API-M 2.1.0 deployment, reside.

   The API definition paths `<API-definitions-path>` are as follows:

   - **Super Tenant** - `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default`
   - **Tenant** - `<API-M_2.6.0_HOME>/repository/tenants`

   Where, `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`. 
Run the PowerShell script `apim210_to_apim260_gateway_artifact_migrator.ps1` as follows:

a. Open a Windows command prompt and type the following command.
   
   ```powershell```

   A message about PowerShell appears, and the shell changes to PowerShell (PS).

b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.
   
   ```
   .\apim210_to_apim260_gateway_artifact_migrator.ps1 <API-definitions-path>
   ```

   `<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the WSO2 API-M 2.1.0 deployment, reside.

   - Super Tenant - `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default`

   Where `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Docs/wso2am-2.6.0`.

Run the PowerShell script `apim210_to_apim260_gateway_artifact_migrator_for_tenants.ps1` as follows:

a. Open a Windows command prompt and type the following command.
   
   ```powershell```

   A message about PowerShell appears, and the shell changes to PowerShell (PS).

b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.
   
   ```
   .\apim210_to_apim260_gateway_artifact_migrator_for_tenants.ps1 <API-definitions-path>
   ```

   `<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the WSO2 API-M 2.1.0 deployment, reside.

   - Tenants - `<API-M_2.6.0_HOME>/repository/tenants`

   Where `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Docs/wso2am-2.6.0`.

It may take a considerable amount of time, which is proportionate to the amount of artifacts, to complete the migration process.
### Troubleshooting

Why do I get the following error - `apim210_to_apim260_gateway_artifact_migrator.ps1` / `apim210_to_apim260_gateway_artifact_migrator_for_tenants.ps1` cannot be loaded because the execution of scripts is disabled on this system?

When running the `apim210_to_apim260_gateway_artifact_migrator.ps1` script, if the execution process is aborted with the above error, it means that the execution of unknown scripts is disabled in the system. To overcome this issue and allow the execution of such scripts, run the following command in the terminal as the Administrator.

```
Set-ExecutionPolicy RemoteSigned
```

4. Upgrade the WSO2 API Manager database from version 2.1.0 to version 2.6.0.
   a. Download the `apimgt-db-migration-scripts-2.1to2.6.zip` and extract it.
   b. Execute the relevant database script, from the scripts that are provided in the extracted directory, on the `WSO2AM_DB` database.

5. If your MB Store database on WSO2 API-M 2.1.0 is not the default H2 DB, upgrade the MB Store database from version 3.1.0 to version 3.2.0.
   a. Download `mb-store-migration-scripts-3.1.0_to_3.2.0.zip` and extract it.
   b. Execute the relevant database script, from the scripts that are provided in the extracted directory, on the `WSO2_MB_STORE_DB` database.

6. Upgrade the Identity component in WSO2 API Manager from version 5.3.0 to 5.7.0.
   a. Download the `wso2is-5.x.x-migration.zip` and extract it.
   b. Copy the `migration-resources` folder from the extracted folder to the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.
   c. Open the `migration-config.yaml` file in the `migration-resources` directory and edit the `currentVersion` element to 5.3.0, as shown below.

   ```yaml
   migrationEnable: "true"
   currentVersion: "5.3.0"
   migrateVersion: "5.7.0"
   ```

   Make sure you have enabled migration by setting the `migrationEnable` element to `true`.

   d. Remove the following entry from the `migration-config.yaml` file.

   ```yaml
   name: "EventPublisherMigrator"
   order: 11
   ```

   e. Copy the `org.wso2.carbon.is.migration-5.7.0.jar` from the extracted folder to the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.
   f. Copy the keystores (i.e., `client-truststore.jks`, `wso2carbon.jks` and any other custom JKS) used in the previous version and replace the existing keystores in the `<API-M_2.6.0_HOME>/repository/resources/security` directory.
   g. Start WSO2 API Manager 2.6.0 via the terminal to carry out the complete Identity component migration.
If you have successfully completed the migration, stop the server and remove the following files and folders.

Remove the file, which is in the `org.wso2.carbon.is.migration-5.7.0.jar` directory.

Remove the folder, which is in the `migration-resources` directory.

If you ran WSO2 API-M as a Windows Service when doing the identity component migration, then you need to remove the following parameters in the command line arguments section (`CMD_LINE_ARGS`) of the `wso2server.bat` file.

```sh
-Dmigrate -Dcomponent=identity
```

Please note that depending on the number of records in the identity tables, this identity component migration will take a considerable amount of time to finish. Do not stop the server during the migration process and please wait until the migration process finish completely and server get started.

### Troubleshooting

When running the above step if you encounter the following error message, please follow the steps in this section. Please note that this error could occur only if the identity tables contain a huge volume of data.

#### Sample exception stack trace

```
ERROR {org.wso2.carbon.registry.core.dataaccess.TransactionManager} - Failed to start new registry transaction.
{org.wso2.carbon.registry.core.dataaccess.TransactionManager}
  Timeout: Pool empty. Unable to fetch a connection in 60 seconds, none available[size:50; busy:50; idle:0; lastwait:60000
```

1. Change the value in `<indexingFrequencyInSeconds>` in `<APIM_HOME>/repository/conf/registry.xml` to a higher value (e.g., 10)
2. Re-run the command above.

Make sure to revert the change done in Step 1, after the migration is complete.

If you are upgrading from a vanilla distribution of WSO2 API Manager 2.1.0 or a WUM updated distribution taken before the 5th December 2017 of WSO2 API Manager 2.1.0, you have to run the following migration client for migrating Access Control support for API Publisher because Publisher Access Control is enabled by default in WSO2 API Manager 2.6.0. **If not, you can skip this step.**
Step 1 - Migrate the WSO2 API-M configurations

Do not copy entire configuration files from the current version of API Manager to the new one, as some configuration files (e.g., api-manager.

Step 2 - Upgrade WSO2 API-M to 2.6.0

Step 3 - Optionally, migrate the configurations for API-M Analytics

As you are upgrading from WSO2 API-M Analytics 2.1.0, in order migrate the configurations required to run WSO2 API-M Analytics for WSO2 API-M 2.6.0 carryout the same instructions as mentioned in Upgrading from 2.5.0 to 2.6.0 - Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics.

Step 4 - Restart the WSO2 API-M server

Follow the instructions below to upgrade your WSO2 API Manager server from WSO2 API-M 2.0.0 to 2.6.0.

If you are using WSO2 Identity Server (WSO2 IS) as a Key Manager, follow the instructions in Upgrading from the Previous Release when WSO2 IS is the Key Manager.

- Step 1 - Migrate the WSO2 API-M configurations
- Step 2 - Upgrade WSO2 API-M to 2.6.0
- Step 3 - Optionally, migrate the configurations for API-M Analytics
- Step 4 - Restart the WSO2 API-M server
In this section, you move all existing API Manager configurations from the current environment to the new one.

1. Back up all databases in your API Manager instances along with the Synapse configs of all the tenants and the super tenant.
   - The Synapse configs of the super tenant are in the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory.
   - The Synapse configs of tenants are in the `<CURRENT_API-M_HOME>/repository/tenants/<tenant-id>/synapse-configs/default` directory.
   - If you use a clustered/distributed API Manager setup, back up the available configurations in the API Gateway node.

2. Download API Manager 2.6.0 from https://wso2.com/api-management/.

3. Open the `<API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases.
   - User Store
   - Registry database/s
   - API Manager databases
   - Statistics database (if statistics are configured)

4. Edit the registry configurations in the `<API-M_2.6.0_HOME>/repository/conf/registry.xml` file and the user database in the `<API-M_2.6.0_HOME>/repository/conf/user-mgt.xml` file similar to the configurations of the current WSO2 API Manager.

   In a clustered/distributed API Manager setup, follow steps 5 and 6 for the Gateway node.

5. Move all your Synapse configurations, except the files mentioned below, by copying the contents of the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` directory and by using it to replace the contents of the `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default` directory.

   **NOTE:** Do not replace the files listed below from the `<CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default` folder to WSO2 API Manager 2.6.0. These are application-specific APIs and sequences. If you made any custom changes to the files below, you need to merge the changes to the corresponding files in WSO2 API-M 2.6.0.

   ```
   /api/_AuthorizeAPI_.xml
   /api/_RevokeAPI_.xml
   /api/_TokenAPI_.xml
   /api/_UserInfoAPI_.xml
   /sequences/_auth_failure_handler_.xml
   /sequences/_build_.xml
   /sequences/_cors_request_handler_.xml
   /sequences/fault.xml
   /sequences/main.xml
   /sequences/_production_key_error_.xml
   /sequences/_resource_mismatch_handler_.xml
   /sequences/_sandbox_key_error_.xml
   /sequences/_throttle_out_handler_.xml
   /proxy-services/WorkflowCallbackService.xml
   ```

6. Move all your tenant Synapse configurations by updating the configurations made in the `<CURRENT_API-M_HOME>/repository/tenants` directory to the `<API-M_2.6.0_HOME>/repository/tenants` directory.

   **NOTE:** Get the files listed below from the `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory and replace the corresponding files in the `<API-M_2.6.0_HOME>/repository/tenants/<tenant-id>/sequences` directory.

   ```
   _auth_failure_handler_.xml
   _cors_request_handler_.xml
   fault.xml
   main.xml
   _production_key_error_.xml
   _resource_mismatch_handler_.xml
   _sandbox_key_error_.xml
   _throttle_out_handler_.xml
   ```
Step 2 - Upgrade WSO2 API-M to 2.6.0

1. Stop all WSO2 API Manager server instances that are running.
2. Make sure you backed up all the databases and Synapse configs as instructed in step 1 of the previous section.
3. To start the migration process, run the respective migration script based on your environment.

   **Linux/Mac OS/Windows**

Run the `apim200_to_apim260_gateway_artifact_migrator.sh` script, as shown below, to migrate from API Manager 2.0.0 to 2.6.0.

```
./apim200_to_apim260_gateway_artifact_migrator.sh <API-definitions-path>
```

* `<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the WSO2 API-M 2.0.0 deployment, reside.

The API definition paths `<API-definitions-path>` are as follows:

- **Super Tenant** - `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default`
- **Tenant** - `<API-M_2.6.0_HOME>/repository/tenants`

Where, `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`.

### Super Tenant

#### Tenants

Run the PowerShell script `apim200_to_apim260_gateway_artifact_migrator.ps1` as follows:

a. Open a Windows command prompt and type the following command.

```
powershell
```

A message about PowerShell appears, and the shell changes to PowerShell (PS).

b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.

```
./apim200_to_apim260_gateway_artifact_migrator.ps1 <API-definitions-path>
```

* `<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which are copied from the WSO2 API-M 2.0.0 deployment, reside.

- **Super Tenant** - `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default`

Where `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`. 
Run the PowerShell script `apim200_to_apim260_gateway_artifact_migrator_for_tenants.ps1` as follows:

a. Open a Windows command prompt and type the following command.

```
powershell
```

A message about PowerShell appears, and the shell changes to PowerShell (PS).
b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.

```
..\apim200_to_apim260_gateway_artifact_migrator_for_tenants.ps1 <API-definitions-path>
```

`<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.0.0 deployment, reside.
- **Tenants** - `<API-M_2.6.0_HOME>/repository/tenants`

Where `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`.

It may take a considerable amount of time, which is proportionate to the amount of artifacts, to complete the migration process.

**Troubleshooting**

*Why do I get the following error - `apim200_to_apim260_gateway_artifact_migrator.ps1` / `apim200_to_apim260_gateway_artifact_migrator_for_tenants.ps1` cannot be loaded because the execution of scripts is disabled on this system?*

When running the `apim200_to_apim260_gateway_artifact_migrator.ps1` script, if the execution process is aborted with the above error, it means that the execution of unknown scripts is disabled in the system. To overcome this issue and allow the execution of such scripts, run the following command in the terminal as the Administrator.

```
Set-ExecutionPolicy RemoteSigned
```

4. Upgrade the WSO2 API Manager database from version 2.0.0 to version 2.6.0.
   a. Download the `apimgt-db-migration-scripts-2.0to2.6.zip` ZIP and extract it.
   b. Execute the relevant database script, from the scripts that are provided in the extracted directory, on the `WSO2AM_DB` database.

5. Upgrade the Identity component in WSO2 API Manager from version 5.2.0 to 5.7.0.

As WSO2 API-M shares identity components with WSO2 IS, this step is necessary to upgrade those components (even if you are not using WSO2 IS as a Key Manager).

a. Download the `wso2is-5.x.x-migration.zip` and extract it.
   b. Copy the `migration-resources` folder in the extracted folder to the `<API-M_2.6.0_HOME>`.
   c. Open the `migration-config.yaml` file in the `migration-resources` directory and edit the `currentVersion` element to 5.2.0, as shown below.
- migrationEnable: "true"
  currentVersion: "5.2.0"
  migrateVersion: "5.7.0"

Make sure you have enabled migration by setting the migrationEnable element to true as defined above.

d. Remove the following entries from the migration-config.yaml file, which is in the migration-resources directory.

```yaml
- name: "EventPublisherMigrator"
  order: 11

and

- name: "ChallengeQuestionDataMigrator"
  order: 6
  parameters:
    schema: "identity"
```

e. Copy the org.wso2.carbon.is.migration-5.7.0.jar from the extracted folder to the <API-M_2.6.0_HOME>/repository/components/dropins directory.

f. Copy the keystores (i.e., client-truststore.jks, wso2carbon.jks and any other custom JKS) used in the previous version and replace the existing keystores in the <API-M_2.6.0_HOME>/repository/resources/security directory.

g. Start WSO2 API Manager 2.6.0 via the terminal to carry out the complete Identity component migration.

For Linux / Mac OS / Windows:

```
sh wso2server.sh -Dmigrate -Dcomponent=identity

wso2server.bat -Dmigrate -Dcomponent=identity
```

If you run WSO2 API-M as a Windows Service when doing the identity component migration, then you need to add the following parameters in the command line arguments section (CMD_LINE_ARGS) of the wso2server.bat file.

```
-Dmigrate -Dcomponent=identity
```

Please note that depending on the number of records in the identity tables, this identity component migration will take a considerable amount of time to finish. Do not stop the server during the migration process and please wait until the migration process finish completely and server get started.

Troubleshooting

When running the above step if you encounter the following error message, please follow the steps in this section. Please note
that this error could occur only if the identity tables contain a huge volume of data.

### Sample exception stack trace

```
ERROR {org.wso2.carbon.registry.core.dataaccess.TransactionManager} - Failed to start new registry transaction.
{org.wso2.carbon.registry.core.dataaccess.TransactionManager}
```

1. Change the value in `<indexingFrequencyInSeconds>` in `<API-M_HOME>/repository/conf/registry.xml` to a higher value (e.g., 10)
2. Re-run the command above.

**Make sure to revert the change done in Step 1, after the migration is complete**

h. After you have successfully completed the migration, stop the server and remove the following files and folders.
   - Remove the `file`, which is in the `<API-M_2.6.0_HOME>/repository/components/dropins` directory.
   - Remove the `directory`, which is in the `<API-M_2.6.0_HOME>` directory.
   - If you ran WSO2 API-M as a Windows Service when doing the identity component migration, then you need to remove the following parameters in the command line arguments section (`CMD_LINE_ARGS`) of the `wso2server.bat` file.

```
-Dmigrate -Dcomponent=identity
```

6. Download the WSO2 API Manager Migration Client and copy it to the `<API-M_HOME>/repository/components/dropins` directory.
7. Stop the WSO2 API-M server if you have started it.
8. Migrate Access Control support for API Publisher.

You have to run the following migration client to migrate the Access Control support for the API Publisher, because Publisher Access Control is enabled by default in WSO2 API Manager 2.6.0.

```
sh wso2server.sh -DmigrateAccessControl=true
```

9. Carryout migration for fault sequence from API-M 2.0.0 to API-M 2.6.0.

```
sh wso2server.sh -DmigrateFromVersion=2.0.0
```

10. Preserve the case sensitive behavior for the migrated resources by adding the following property to the `<API-M_HOME>/repository/conf/registry.xml` file under `<AuthorizationManager>` as follows:

```
<AuthorizationManager>
  <Property name="PreserveCaseForResources">false</Property>
</AuthorizationManager>
```

11. Re-index the artifacts in the registry.
   a. Shut down WSO2 API Manager 2.6.0 (if you have already started it).
   b. Run the `reg-index.sql` script against the `REG_DB` database.

Please note that depending on the number of records in the REG_LOG table, this script will take a considerable amount of time to finish. Do not stop the execution of script until it is completed.
Step 3 - Optionally, migrate the configurations for API-M Analytics

This step is only required if you have WSO2 API-M Analytics configured in your current deployment.

As you are upgrading from WSO2 API-M Analytics 2.0.0, in order migrate the configurations required to run WSO2 API-M Analytics for WSO2 API-M 2.6.0 carryout the same instructions as mentioned in Upgrading from 2.5.0 to 2.6.0 - Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics.

Step 4 - Restart the WSO2 API-M server

Follow the instructions below to upgrade your WSO2 API Manager server from APIM 1.10.0 to 2.6.0.

If you are using WSO2 Identity Server (WSO2 IS) as a Key Manager, follow the instructions in Upgrading from the Previous Release when WSO2 IS is the Key Manager.

- Step 1 - Migrate the WSO2 API-M configurations
- Step 2 - Upgrade WSO2 API-M to 2.6.0
- Step 3 - Optionally, migrate the statistics related data from WSO2 DAS to API Manager Analytics
- Step 4 - Restart the WSO2 API-M server

Step 1 - Migrate the WSO2 API-M configurations

Do not copy entire configuration files from the current version of API Manager to the new one, as some configuration files (e.g., api-manager.xml) may have changed. Instead, redo the configuration changes in the new configuration files.

Follow the instructions below to move all the existing API Manager configurations from the current environment to the new one.

1. Back up all databases in your API Manager instances along with the Synapse configs of all the tenants and the super tenant.
   - The Synapse configs of the super tenant are in the <CURRENT_API-M_HOME>/repository/deployment/server/synapse-configs/default directory.
   - The Synapse configs of tenants are in the <CURRENT_API-M_HOME>/repository/tenants directory.
   - If you use a clustered/distributed API Manager setup, back up the available configurations in the API Gateway node.
3. Open the <API-M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml file and provide the datasource configurations for the following databases. You can copy the configuration values from the same file in the current WSO2 API Manager instance that is already being used.
   - User Store
   - Registry database/s
   - API Manager databases
   - Statistics database (if statistics are configured)
4. Edit the registry configurations in the <API-M_2.6.0_HOME>/repository/conf/registry.xml file and the user database in the <API-M_2
4. File similar to the configurations of the current WSO2 API Manager.

   `WSO2_API_M_2.6.0_HOME>/repository/conf/user-mgt.xml`

5. Move all your Synapse configurations, except the files mentioned below, by copying the contents in the `<CURRENT_API_M_HOME>/repository/deployment/server/synapse-configs/default` directory and by using it to replace the contents in the `<API_M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default` directory.

   **NOTE:** Do not replace the files listed below from the `<CURRENT_API_M_HOME>/repository/deployment/server/synapse-configs/default` folder to WSO2 API-M 2.6.0. These are application-specific APIs and sequences. If you made any custom changes to the files below, you need to merge the changes to the corresponding files in 2.6.0.

   `/api/_AuthorizeAPI_.xml
   `/api/_RevokeAPI_.xml
   `/api/_TokenAPI_.xml
   `/api/_UserInfoAPI_.xml
   `/sequences/_auth_failure_handler_.xml
   `/sequences/_build_.xml
   `/sequences/_cors_request_handler_.xml
   `/sequences/fault.xml
   `/sequences/main.xml
   `/sequences/_production_key_error_.xml
   `/sequences/_resource_mismatch_handler_.xml
   `/sequences/_sandbox_key_error_.xml
   `/sequences/_resource_mismatch_handler_.xml
   `/sequences/_sandbox_key_error_.xml
   `/sequences/_throttle_out_handler_.xml
   `/sequences/_token_fault_.xml
   `/proxy-services/WorkflowCallbackService.xml

6. Move all your tenant Synapse configurations.

   Copy the contents in the `<CURRENT_API_M_HOME>/repository/tenants` directory. Replace the contents in the `<API_M_2.6.0_MANAGER_HOME>/repository/tenants` directory with the copied contents.

   **NOTE:** Get the files listed below from the `<API_M_2.6.0_MANAGER_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory and replace the corresponding files in the `<API_M_2.6.0_MANAGER_HOME>/repository/tenants/<tenant-id>/synapse-configs/default/sequences` directory.

   `_auth_failure_handler_.xml
   _cors_request_handler_.xml
   fault.xml
   main.xml
   _production_key_error_.xml
   _resource_mismatch_handler_.xml
   _sandbox_key_error_.xml
   _throttle_out_handler_.xml

7. If you have documentation defined for your APIs, make sure you add a value to the **Documentation Summary** field.

   Documentation summary is mandatory in APIM 2.0.0 onwards. Therefore, you may face issues when trying to migrate with an empty documentation summary field from an older version.

**Step 2 - Upgrade WSO2 API-M to 2.6.0**

1. Stop all WSO2 API Manager server instances that are running.
2. Make sure you backed up all the databases and Synapse configs as instructed in step 1 of the previous section.
3. Before you run the migration client, open the `<API_M_2.6.0_HOME>/repository/conf/datasources/master-datasources.xml` file, and set the `<username>`, and `<password>` elements of the `AM_DB` JNDI to that of a user who has permissions to alter tables in the database.

   **Tip:** After you are done running the migration client, you can switch these credentials back to a user with lesser privileges.
Example

```xml
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <username>xxxxxx</username>
      <password>xxxxxx</password>
      ...
    </configuration>
  </definition>
</datasource>
```

4. Upgrade the identity component in WSO2 API Manager from version 5.1.0 to 5.2.0.

As WSO2 API-M shares identity components with WSO2 IS, this step is necessary to upgrade those components (even if you are not using WSO2 IS as a Key Manager).

a. Download the migration DB scripts for version 5.1.0 to 5.2.0 from here.
b. Unzip and find the correct DB script in the `wso2is-5.2.0-migration/dbscripts/identity/migration-5.1.0_to_5.2.0` directory.
c. Manually execute the DB scripts that correspond to the RDBMS that you are working with.

Apply the respective DB script found inside the following directories against the respective DB as follows:

<table>
<thead>
<tr>
<th>DB script directory</th>
<th>Applicable DB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wso2is-5.2.0-migration/dbscripts/identity/migration-5.1.0_to_5.2.0</code></td>
<td>AM_DB</td>
<td>Run the script against AM_DB.</td>
</tr>
<tr>
<td><code>wso2is-5.2.0-migration/dbscripts/migration-5.1.0_to_5.2.0</code></td>
<td>UM_DB, REG_DB</td>
<td>This script includes insert operations against the UM_CLAIM table and some index creation scripts against REG_DB, which are required in order for you to upgrade from your previous WSO2 API-M version to WSO2 API-M 2.0.0. Please run the index creation queries against the REG_DB and UM_CLAIM table insert queries against the UM_DB.</td>
</tr>
</tbody>
</table>

For example, if you are working with a MySQL DB, run the `mysql.sql` script.

5. Download and extract the [WSO2 API Manager migration client](https://wso2.com/products/api-manager/migration) and do the following to upgrade your version of WSO2 API Manager to WSO2...
This migration client upgrades the WSO2 API Manager components from WSO2 API-M 1.10.0 to WSO2 API-M 2.0.0 first. Thereafter, from step 6 onwards, you need to carryout the rest of the changes that are required to upgrade from version WSO2 API-M 2.0.0 to WSO2 API-M 2.6.0.

### API-M 2.0.0:

This migration client upgrades the WSO2 API Manager components from WSO2 API-M 1.10.0 to WSO2 API-M 2.0.0 first. Thereafter, from step 6 onwards, you need to carryout the rest of the changes that are required to upgrade from version WSO2 API-M 2.0.0 to WSO2 API-M 2.6.0.

#### a. Copy the `org.wso2.carbon.apimgt.migrate.client-2.0.X.jar` file to the `<API-M_2.6.0_HOME>/repository/component` directory.

If you use a **clustered/distributed API Manager setup**, copy the JAR file to the Publisher and Gateway nodes.

#### b. Copy the `migration-script` folder into `<API-M_2.6.0_HOME>`.

If you use a **clustered/distributed API Manager setup**, copy the migration-script folder to a Publisher node.

#### i. Start API Manager 2.6.0 with the following command-line options to migrate the database, registry, and the file system, separately, in the given order.

<table>
<thead>
<tr>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>To migrate the database only. This migrates the AM_DB database. Please ensure that the <code>&lt;API_M_PUBLISHER&gt;/repository/conf/datasources/master-datasources.xml</code> file has an entry for AM_DB.</td>
<td><code>-DmigrateDB=true -Dcomponent=apim -DmigrateFromVersion=1.1</code></td>
</tr>
<tr>
<td>To migrate the registry only. This migrates the registry-related resources such as .rxt and Swagger definitions.</td>
<td><code>-DmigrateReg=true -Dcomponent=apim -DmigrateFromVersion=1.1</code></td>
</tr>
<tr>
<td>You may notice the &quot;Fault sequence migration from APIM 2.0.0 to 2.1.0 has started&quot; log message in this step which is used for fault sequence migration. This migration is also done in this step.</td>
<td></td>
</tr>
<tr>
<td>To migrate the file system only. This migrates the Synapse config files such as APIs that reside in the file system. Therefore, you must run this command on the Gateway node(s) of a distributed WSO2 API Manager setup.</td>
<td><code>-DmigrateFS=true -Dcomponent=apim -DmigrateFromVersion=1.1</code></td>
</tr>
<tr>
<td>To migrate to the new throttling engine and generate throttle policies. This migrates Synapse config files, the API-M database with new throttle policies and generates throttle policies to the <code>&lt;API-M_2.6.0_HOME&gt;/executionplans</code> directory. Therefore, you must run this command against a node that has Synapse config files and the AM_DB. After the migration, copy the <code>&lt;API-M_2.6.0_HOME&gt;/executionplans</code> directory to the <code>&lt;API-M_2.6.0_HOME&gt;/repository/deployment/server/executionplans</code> directory of the Traffic Manager node. Please note that you need to use either this command or the one below in order to migrate to the new throttling engine of APIM 2.0.0.</td>
<td><code>-Dcomponent=apim -DmigrateFromVersion=1.1 -Doptions=migrateThrottling</code></td>
</tr>
<tr>
<td>To migrate to the new throttling engine and deploy throttle policies to the Traffic Manager node. This migrates Synapse config files, the API-M database with new throttle policies, and deploys policies to the Traffic Manager node. Therefore, you must run this command against a node that has Synapse config files, the AM_DB, and is pointed to the Traffic Manager node. Please use this command only if you want to deploy the throttle policies directly to a seperate Traffic Manager node while performing the migration.</td>
<td><code>-Dcomponent=apim -DmigrateFromVersion=1.1 -Doptions=migrateThrottling</code></td>
</tr>
</tbody>
</table>

If you are using a **clustered/distributed API Manager setup** -

- Run with the following options `-DmigrateDB=true -DmigrateReg=true -Dcomponent=apim`
To migrate to the throttling engine of WSO2 API-M 2.6.0, you have to use a node that has AM_DB pointed as well as API Gateway Synapse configurations in place in the file system.

- If you use the migrateThrottling option, you can use the API Gateway node. Make sure that AM_DB is pointed to the Gateway at the time of migration. You can start the Gateway server with the -Dcomponent=apim -DmigrateFromVersion=1.10.0 -Doptions=migrateThrottling command. After completing this execution, make sure you copy the generated throttle policies in the <API-M_2.6.0_HOME>/executionplans directory to the <API-M_2.6.0_HOME>/repository/deployment/server/executionplans directory of the Traffic Manager Node.

- If you use both the migrateThrottling,deployPolicies options make sure that the Traffic Manager node is configured with the Gateway node, and that you avoid passing the -Dprofile option while performing the migration. When using these options, you do not have to manually copy the throttling policies to the Traffic Manager. You can start the Gateway server with the -Dcomponent=apim -DmigrateFromVersion=1.10.0 -Doptions=migrateThrottling,deployPolicies command.

**Troubleshooting**

If you have enabled token encryption in your database, those consumer keys need to be decrypted as part of the database migration. If you encounter the following error, it means that the migration client has failed to decrypt some of the keys in the old database.

```
ERROR (org.wso2.carbon.apimgt.migration.client.MigrateFrom19to110) -
Cannot decrypt consumer key :
DR8bt68uSU4+7xtCEIzu642CMqXhNy13OYVJOVL/H6EnFo8GR8a2GuhLH17W1HxYyvxeOp
b1YbavvRRIN/b6VqO9m8Hu0O1uLkkd AM_APPLICATION_KEY_MAPPING table
```

If this error occurs, by default, the migration client will terminate without updating databases in order to maintain database integrity. However, you can change this behavior by adding the following argument and running the WSO2 API-M migration client again from the beginning. Please note that the database is then updated with the keys where decryption was successful and failed keys are permanently deleted.

```
-DremoveDecryptionFailedKeysFromDB=true
```

**Expected Errors**

The following are some errors that you may come across. **Do not get alarmed** as these errors are to be expected, because you are yet to complete the migration process.

- You may notice the following exception being thrown in the console when WSO2 API Manager 2.6.0 is started at this point,

```
org.wso2.carbon.idp.mgt.IdentityProviderManagementException:
Error occurred while retrieving Identity Provider information for
  tenant : carbon.super and Identity Provider name : LOCAL
```

This is due to the fact that the IDP_METADATA table does not exist in the WSO2 API Manager database. The instructions given from step 8 onwards address the creation of the IDP_METADATA table after which
6. Run the respective migration script, based on your environment.

**Linux/Mac OS/Windows**

Run the `apim200_to_apim260_gateway_artifact_migrator.sh` script, as shown below, to migrate from WSO2 API Manager 2.0.0 to 2.6.0.

```
./apim200_to_apim260_gateway_artifact_migrator.sh <API-definitions-path>
```

`<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.0.0 deployment, reside.

The API definition paths `<API-definitions-path>` are as follows:

- **Super Tenant** - `<API-M-2.6.0_HOME>/repository/deployment/server/synapse-configs/default`
- **Tenant** - `<API-M-2.6.0_HOME>/repository/tenants`

Where, `<API-M-2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`.

This exception will no longer be thrown.

- You may notice the following error during the first stage of running the WSO2 API Manager Migration Client.

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'WSO2AM_DB.AM_POLICY_APPLICATION' doesn't exist
```

This error will get resolved after the first run of the migration client.

- You may notice Gateway artifact deployment related errors. The following is one such exception that is thrown in the console when WSO2 API Manager 2.6.0 is started at this point.

```
Error loading class : org.wso2.carbon.apimgt.usage.publisher.APIMgtResponseHandler – Class not found
```

This error gets eliminated after executing the Gateway artifact migration script in step 6.

- You may also notice the following exceptions when WSO2 API Manager 2.6.0 is started at this point.

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'WSO2AM_DB.IDN_CLAIM_DIALECT' doesn't exist
```

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'WSO2AM_DB.IDN_OIDC_SCOPE' doesn't exist
```

These errors will be resolved after following the instructions given under step 8 below.
Run the PowerShell script `apim200_to_apim260_gateway_artifact_migrator.ps1` as follows:

a. Open a Windows command prompt and type the following command.

```
powershell
```

A message about PowerShell appears, and the shell changes to PowerShell (PS).

b. Run the PowerShell script by passing the location of the gateway artifacts that you need to migrate.

```
.apim200_to_apim260_gateway_artifact_migrator.ps1 <API-definitions-path>
```

`<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.0.0 deployment, reside.

- **Super Tenant** - `<API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default`

Where `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`.

---

Run the PowerShell script `apim200_to_apim260_gateway_artifact_migrator_for_tenants.ps1` as follows:

a. Open a Windows command prompt and type the following command.

```
powershell
```

A message about PowerShell (PS) appears, and the shell changes to PS.

b. Run the PowerShell script by passing the location of the Gateway artifacts that you need to migrate.

```
.apim200_to_apim260_gateway_artifact_migrator_for_tenants.ps1 <API-definitions-path>
```

`<API-definitions-path>` - This is the location where the WSO2 API-M 2.6.0 API definitions, which were copied from the API-M 2.0.0 deployment, reside.

- **Tenants** - `<API-M_2.6.0_HOME>/repository/tenants`

Where `<API-M_2.6.0_HOME>` can be, for example, `/Users/user12/Documents/wso2am-2.6.0`.

---

It may take a considerable amount of time, which is proportionate to the amount of artifacts, to complete the migration process.

---

**Troubleshooting**
7. Upgrade the WSO2 API Manager database from version 2.0.0 to version 2.6.0.
   a. Download the apimgt-db-migration-scripts-2.0to2.6.zip and extract it.
   b. Execute the relevant database script, from the scripts that are provided in the extracted directory, on the WSO2AM_DB database.

8. Upgrade the Identity component in WSO2 API Manager from version 5.2.0 to 5.7.0.
   a. Download the wso2is-5.x.x-migration.zip and extract it.
   b. Copy the migration-resources folder in the extracted folder to <API-M_2.6.0_HOME>.
   c. Open the migration-config.yaml file in the migration-resources directory and edit the currentVersion element to 5.2.0, as shown below.

   ```yaml
   migrationEnable: "true"
   currentVersion: "5.2.0"
   migrateVersion: "5.7.0"
   ```

   d. Remove the following entries from the migration-config.yaml file.

   ```yaml
   - name: "EventPublisherMigrator"
     order: 11
   ```

   and

   ```yaml
   - name: "ChallengeQuestionDataMigrator"
     order: 6
     parameters:
       schema: "identity"
   ```

e. Copy the org.wso2.carbon.is.migration-5.7.0.jar from the extracted folder to the <API-M_2.6.0_HOME>/repository/components/dropins directory.

f. Copy the keystores (i.e., client-truststore.jks, wso2carbon.jks and any other custom JKS) used in the previous version and replace the existing keystores in the <API-M_2.6.0_HOME>/repository/resources/security directory.

g. Start API Manager 2.6.0 via the terminal to carry out the complete Identity component migration.

   - Linux / Mac OS
   - Windows
If you run WSO2 API-M as a Windows Service when doing the identity component migration, then you need to add the following parameters in the command line arguments section (CMD_LINE_ARGS) of the wso2server.bat file.

```wso2server.bat -Dmigrate -Dcomponent=identity```

Please note that depending on the number of records in the identity tables, this identity component migration will take a considerable amount of time to finish. Do not stop the server during the migration process and please wait until the migration process finish completely and server get started.

**Troubleshooting**

When running the above step if you encounter the following error message, please follow the steps in this section. Please note that this error could occur only if the identity tables contain a huge volume of data.

Sample exception stack trace

```java
ERROR {org.wso2.carbon.registry.core.dataaccess.TransactionManager} - Failed to start new registry transaction.
{org.wso2.carbon.registry.core.dataaccess.TransactionManager}
  Timeout: Pool empty. Unable to fetch a connection in 60 seconds, none available[size:50; busy:50; idle:0; lastwait:60000

1. Change the value in <indexingFrequencyInSeconds> in <API-M_HOME>/repository/conf/registry.xml to a higher value (e.g., 10)
2. Re-run the command above.
```

You have to run the following migration client for migrating Access Control support for API Publisher, because Publisher Access Control is enabled by default in API Manager 2.6.0.

```sh wso2server.sh -Dmigrate -Dcomponent=identity```
a. Download WSO2 API Manager Migration Client (You can use the same JAR that downloaded in step 5) and copy it to the &lt;API-M_HOME&gt;/repository/components/dropins directory.
b. Start the WSO2 API-M server as follows:

```
sh wso2server.sh -DmigrateAccessControl=true
```

t. Preserve the case sensitive behavior for the migrated resources by adding the following property to the &lt;API-M_HOME&gt;/repository/conf/us er-mgt.xml file under &lt;AuthorizationManager&gt; as follows:

```
<AuthorizationManager class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
  
  <Property name="PreserveCaseForResources">false</Property>

</AuthorizationManager>
```

11. Re-index the artifacts in the registry.
a. Shut down WSO2 API Manager 2.6.0 (if you have already started it).
b. Run the reg-index.sql script against the REG_DB database.

c. Add the tenantloader-1.0.jar to &lt;API-M_2.6.0_HOME&gt;/repository/components/dropins directory
d. Rename the lastAccessTimeLocation element in the &lt;API-M_2.6.0_HOME&gt;/repository/conf/registry.xml file.
e. If you use a clustered/distributed API Manager setup, change the file in the API Publisher node. For example, change the _/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime registry path to _/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime_ 1
f. If the &lt;API-M_2.6.0_HOME&gt; / solr directory exists, take a backup and thereafter delete it.
g. Start the WSO2 API-M server.
h. Stop the WSO2 API-M server and remove the tenantloader-1.0.jar from the &lt;API-M_2.6.0_HOME&gt;/repository/components/dropins directory.

Step 3 - Optionally, migrate the statistics related data from WSO2 DAS to API Manager Analytics

This step is only required if you have WSO2 Data Analytics Server (WSO2 DAS) configured in your current deployment for analytics.

From WSO2 API Manager 2.0.0 onwards, statistics can be configured only for RDBMS, because the WSO2 API Manager 1.10.0 REST based analytics configurations no longer exists. Follow the instructions below to migrate statistics data from a previous versions of API Manager to WSO2 API-M 2.6.0.

If you have configured analytics using RDBMS -

1. Take a backup of the API Manager statistics DB [STAT_DB].
2. Execute the relevant DB script found in migration-scripts/110-200-migration/stat/ on the STAT_DB database that you configured in WSO2 DAS for API-M 2.0.0.
3. Directly migrate to WSO2 API-M 2.6.0 as mention in Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics, under the Upgrade from 2.5.0 to 2.6.0 tab.

You may notice that the following exceptions are thrown in the console when migrating at this point. These exceptions occur because the following tables were introduced only after WSO2 API-M 2.0.0. Therefore, you can safely ignore these exceptions.

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'TestStatsDB_new.API_REQ_USER_BROW_SUMMARY' doesn't exist.
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'TestStatsDB_new.API_EXE_TME_SUMMARY' doesn't exist
```
If you configured analytics using the REST client in API-M 1.10.0 with DAS 3.0.x -

1. Configure analytics using WSO2 API Manager 1.10.0 and WSO2 DAS 3.0.x with the RDBMS client based on the instructions in Publishing API Runtime Statistics Using RDBMS, which is in the WSO2 API-M 1.10.0 documentation.
2. Wait for data to appear on the RDBMS and WSO2 API Manager statistics dashboard.
3. Take a backup of the WSO2 API Manager statistics DB (STAT_DB).
4. Execute the relevant DB script found in migration-scripts/110-200-migration/stat/ on the STAT_DB that you configured in WSO2 Data Analytics (WSO2 DAS) for WSO2 API Manager 1.10.0.
   After executing the DB script, now, your STAT_DB is similar to the STAT_DB in WSO2 API-M 2.0.0.
5. Directly migrate to WSO2 API-M 2.6.0 as mention in Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics, under the Upgrade from 2.5.0 to 2.6.0 tab.

Step 4 - Restart the WSO2 API-M server

Linux / Mac OS / Windows

```
sh wso2server.sh
wso2server.bat
```

Follow the instructions below to upgrade your WSO2 API Manager server from APIM 1.8.0/1.9.0/1.9.1 to 2.6.0.

If you are using WSO2 Identity Server (WSO2 IS) as a Key Manager, follow the instructions in Upgrading from the Previous Release when WSO2 IS is the Key Manager.

Step 1 - Upgrade WSO2 API-M to 2.0.0

It is not possible to directly upgrade from WSO2 API Manager 1.8.0/1.9.0/1.9.1 to 2.6.0.

Upgrade your current WSO2 API-M version (1.8.0/1.9.0/1.9.1) to WSO2 API-M 2.0.0 as expained in the WSO2 API-M 2.0.0 documentation.

Step 2 - Upgrade WSO2 API-M to 2.6.0

After you have successfully migrated your current WSO2 API-M version to 2.0.0, upgrade from API-M 2.0.0 to API-M 2.6.0. For more information, see Upgrading from 2.5.0 to 2.6.0 - Step 3 - Optionally, migrate the configurations for WSO2 API-M Analytics.

This concludes the upgrade process.

Tip 1: The migration client that you use in this guide automatically migrates your tenants, workflows, external user stores, etc. to the upgraded environment. Therefore, there is no need to migrate them manually.

Tip 2: From 2.6.0 onwards, WSO2 API Manager is configured by default to trigger token clean up during token generation, token refreshing and token revocation. For more details, see Configuring API Manager for token cleanup. Therefore, when the state of the token (TOKEN_STATE) is changed during any of the latter mentioned processes for tokens that were ACTIVE before the migration process and/or for tokens that will be generated after the migration process, by default, such tokens will be removed from the IDN_OAUTH2_ACCESS_TOKEN table and stored in an audit table. However, the old tokens which were already inactive, revoked, or expired before the migration will not be cleaned by the same token cleanup process. Therefore, alternatively, after the migration, you could follow step 3 in the Using stored procedures for token cleanup section and use the given token clean up script to manually clean up the old inactive, revoked, and expired tokens from the IDN_OAUTH2_ACCESS_TOKEN table.

Tip 3: If you are using an SVN based deployment synchronizer, start with a clean SVN repository and point the new deployment nodes to the new SVN repository. In addition, you need to remove any existing .svn directories in the new deployment's <API-M_2.6.0_HOME>/repository/deployment/server/synapse-configs/default directory and the <API-M_2.6.0_HOME>/repository/tenants/<tenant-id>/synapse-configs/default directory before starting the servers. For more details, see Configuring Deployment Synchronization.
The following information describes how to upgrade your WSO2 API Manager (WSO2 API-M) environment from APIM 1.8.0/1.9.0/1.9.1/1.10.0/2.0.0/2.1.0/2.2.0/2.5.0 to 2.6.0 when WSO2 Identity Server (WSO2 IS) is the Key Manager.

You can download the pre-packaged WSO2 Identity Server 5.7.0, which is also known as Identity Server as the Key Manager 5.7.0 from here.

Upgrading from 2.5.0 to 2.6.0 when using WSO2 IS as the Key Manager:

1. Migrate the WSO2 Identity Server (WSO2 IS) from version 5.6.0 to 5.7.0 as described in the WSO2 Identity Server 5.7.0 documentation.
2. Migrate WSO2 API-M from 2.5.0 to 2.6.0.
   Follow the instructions mentioned in the Upgrading from 2.5.0 to 2.6.0 tab, which is in Upgrading from the Previous Release, but skip step 2 - (4), which explains how to migrate the WSO2 API-M Identity component.

You can use the following steps in either one of the following situations:

- You are currently using a WSO2 IS 5.6.0 vanilla distribution that has WSO2 API Management related Key Manager features installed on top of it.
- You are currently using a pre-packaged WSO2 Identity Server 5.6.0 distribution.
- If you wish to upgrade your APIM environment from API-M 2.5.0 to 2.6.0, which is using the internal WSO2 Identity Server (WSO2 IS) capabilities, follow the instructions in Upgrading from the Previous Release.

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.5.0 to 2.6.0 when using WSO2 IS as the Key Manager:

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.2.0 to 2.6.0 when using WSO2 IS as the Key Manager:

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.1.0 to 2.6.0 when using WSO2 IS as the Key Manager:

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.0.0 to 2.6.0 when using WSO2 IS as the Key Manager:

You can use the following steps in either one of the following situations:

- You are currently using a WSO2 IS 5.5.0 vanilla distribution that has WSO2 API Management related Key Manager features installed on top of it.
- You are currently using a pre-packaged WSO2 Identity Server 5.5.0 distribution.
- If you wish to upgrade your APIM environment from API-M 2.2.0 to 2.6.0, which is using the internal WSO2 Identity Server (WSO2 IS) capabilities, follow the instructions in Upgrading from the Previous Release.

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.2.0 to 2.6.0 when using WSO2 IS as the Key Manager:

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.1.0 to 2.6.0 when using WSO2 IS as the Key Manager:

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.0.0 to 2.6.0 when using WSO2 IS as the Key Manager:

You can use the following steps in either one of the following situations:

- You are currently using a WSO2 IS 5.3.0 vanilla distribution that has WSO2 API Management related Key Manager features installed on top of it.
- You are currently using a pre-packaged WSO2 Identity Server 5.3.0 distribution.
- If you wish to upgrade your APIM environment from API-M 2.1.0 to 2.6.0, which is using the internal WSO2 Identity Server (WSO2 IS) capabilities, follow the instructions in Upgrading from the Previous Release.

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.1.0 to 2.6.0 when using WSO2 IS as the Key Manager:

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.0.0 to 2.6.0 when using WSO2 IS as the Key Manager:

You can use the following steps in either one of the following situations:

- You are currently using a WSO2 IS 5.2.0 vanilla distribution that has WSO2 API Management related Key Manager features installed on top of it.
- You are currently using a pre-packaged WSO2 Identity Server 5.2.0 distribution.
- If you wish to upgrade your APIM environment from API-M 2.0.0 to 2.6.0, which is using the internal WSO2 Identity Server (WSO2 IS)
Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 2.0.0 to 2.6.0 when using WSO2 IS as the Key Manager:

1. Migrate the WSO2 Identity Server (WSO2 IS) from version 5.2.0 to 5.7.0.
   Follow the instructions given in Upgrading From an Older Version of WSO2 IS section of the WSO2 Identity Server 5.7.0 documentation.

2. Migrate WSO2 API-M from 2.0.0 to 2.6.0.
   Follow the instructions mentioned in the Upgrading from 2.0.0 to 2.6.0 tab, which is in Upgrading from the Previous Release, but skip step 2 - (5), which explains how to migrate the WSO2 API-M Identity component.

   - You can use the following steps in either one of the following situations:
     - You are currently using a WSO2 IS 5.1.0 vanilla distribution that has WSO2 API Management related Key Manager features installed on top of it.
     - You are currently using a pre-packaged WSO2 Identity Server 5.1.0 distribution.

   - If you wish to upgrade your APIM environment from 1.10.0 to 2.6.0, which is using the Internal WSO2 Identity Server (WSO2 IS) capabilities that exists in WSO2 API Manager (WSO2 API-M), go to Upgrading from the Previous Release.

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 1.10.0 to 2.6.0 when using WSO2 IS as the Key Manager:

1. Migrate the WSO2 Identity Server (WSO2 IS) from version 5.1.0 to 5.7.0.
   Follow the instructions given in Upgrading From an Older Version of WSO2 IS section of the WSO2 Identity Server 5.7.0 documentation.

2. Migrate WSO2 API-M from 1.10.0 to 2.6.0.
   Follow the instructions mentioned under Upgrading from 1.10.0 to 2.6.0, which is in Upgrading from the Previous Release, but skip step 2 - (4) and (8), which explains how to migrate the WSO2 API-M Identity component.

   - You can use the following steps in either one of the following situations:
     - You are currently using a WSO2 IS 5.0.0 vanilla distribution that has WSO2 API Management related Key Manager features installed on top of it.
     - You are currently using a pre-packaged WSO2 Identity Server 5.0.0 distribution.

   - If you wish to upgrade your APIM environment from 1.8.0/1.9.0/1.9.1 to 2.6.0, which is using the Internal WSO2 Identity Server (WSO2 IS) capabilities that exists in WSO2 API Manager (WSO2 API-M), go to Upgrading from the Previous Release.

Follow the instructions below to upgrade WSO2 API-M from WSO2 API-M 1.8.0/1.9.0/1.9.1 to 2.6.0 when using WSO2 IS as the Key Manager:

1. Migrate the WSO2 Identity Server (WSO2 IS) from version 5.0.0 to 5.7.0.
   Follow the instructions given in Upgrading From an Older Version of WSO2 IS section of the WSO2 Identity Server 5.7.0 documentation.

2. Migrate WSO2 API-M from 1.8.0/1.9.0/1.9.1 to 2.6.0.
   a. Migrate WSO2 API-M from 1.8.0/1.9.0/1.9.1 to 2.0.0.
      Follow the instructions mentioned in Upgrading from the Previous Release, but skip steps (7) to (10), which explains how to migrate the WSO2 API-M Identity component.
   b. Migrate WSO2 API-M from 2.0.0 to 2.6.0.
      Follow the instructions mentioned under Upgrading from 2.0.0 to 2.6.0, which is in Upgrading from the Previous Release, but skip step 2 - (5), which explains how to migrate the APIM Identity component.

Deploying WSO2 API Manager

In a typical production environment, you set up the different WSO2 API Manager (WSO2 API-M) components (API Publisher, Store, Gateway, Key Manager, and Traffic Manager) in separate servers so that you can scale them independently. You also install multiple instances of a component in a cluster to ensure proper load balancing. When one node becomes unavailable or is experiencing high traffic, another node handles the requests.

The following topics explain the deployment aspect of WSO2 API Manager in detail.

- Deployment Patterns
- Deploying API Manager using Single Node Instances
- Configuring the keyStore and client trustStore for a Selected Pattern
- Distributed Deployment of API Manager
- Configuring the Proxy Server and the Load Balancer
- Installing and Configuring the Databases
- Configuring rsync for Deployment Synchronization
- Configuring WSO2 Identity Server as a Key Manager
- Configuring Admin App Event Publishing for Traffic Manager HA Setup

Deployment Patterns

WSO2 API Manager includes five main components as the Publisher, Store, Gateway, Traffic Manager and Key Manager. In a stand-alone APIM setup,
these components are deployed in a single server. However, in a typical production setup, they need to be deployed in separate servers for better performance. Installing and configuring each or selected component/s in different servers is called a distributed setup.

This topic includes the following sections:

- Main components of a distributed setup
- WSO2 API Manager deployment patterns
- Scaling the Gateway

Main components of a distributed setup

The following diagram illustrates the main components of an API Manager distributed deployment.

![Diagram of Main components of an APIM distributed setup](image)

*Figure: Main components of an APIM distributed setup*

Let's take a look at each component in the above diagram. For more information on these components, see **Architecture**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Gateway</td>
<td>This component is responsible for securing, protecting, managing, and scaling API calls. The API gateway is a simple API proxy that intercepts API requests and applies policies such as throttling and security checks. It is also instrumental in gathering API usage statistics. We use a set of handlers for security validation and throttling purposes in the API Gateway. Upon validation, it will pass Web service calls to the actual back end. If it is a token request call, it will then directly pass the call to the Key Manager Server to handle it.</td>
</tr>
<tr>
<td>API Store</td>
<td>This component provides a space for consumers to self-register, discover API functionality, subscribe to APIs, evaluate them, and interact with API publishers. Users can view existing APIs and create their own application by bundling multiple APIs together into one application.</td>
</tr>
<tr>
<td>API Publisher</td>
<td>This component enables API providers to easily publish their APIs, share documentation, provision API keys, and gather feedback on API features, quality, and usage. You can create new APIs by pointing to the actual back end service and also define rate-limiting policies available for the API.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>API Key Manager Server</strong></td>
<td>This component is responsible for all security and key-related operations. When an API call is sent to the Gateway, it calls the Key Manager server and verifies the validity of the token provided with the API call. If the Gateway gets a call requesting a fresh access token, it forwards the username, password, consumer key, and consumer secret key obtained when originally subscribing to the API to the Key Manager. All tokens used for validation are based on OAuth 2.0 protocol. All secure authorization of APIs is provided using the OAuth 2.0 standard for Key Management. The API Gateway supports API authentication with OAuth 2.0, and it enables IT organizations to enforce rate limits and throttling policies for APIs by consumer. Login/Token API in the Gateway node should point to the token endpoint of Key Manager node. The token endpoint of the Key Manager node is at <a href="https://localhost:9443/oauth2endpoints/token">https://localhost:9443/oauth2endpoints/token</a>. In a distributed setup, it should be https://&lt;IP of key manager node&gt;:&lt;port-with-offset-of-keymanager-node&gt;/oauth2endpoints/token.</td>
</tr>
<tr>
<td><strong>API Traffic Manager</strong></td>
<td>The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests.</td>
</tr>
<tr>
<td><strong>LB (load balancers)</strong></td>
<td>The distributed deployment setup depicted above requires two load balancers. We set up the first load balancer, which is an instance of NGINX Plus, internally to manage the cluster. The second load balancer is set up externally to handle the requests sent to the clustered server nodes, and to provide failover and autoscaling. As the second load balancer, you can use an instance of NGINX Plus or any other third-party product.</td>
</tr>
<tr>
<td><strong>RDBMS (shared databases)</strong></td>
<td>The distributed deployment setup depicted above shares the following databases among the APIM components set up in separate server nodes.</td>
</tr>
<tr>
<td></td>
<td>• <strong>User Manager Database</strong>: Stores information related to users and user roles. This information is shared among the Key Manager Server, Store, and Publisher. Users can access the Publisher for API creation and the Store for consuming the APIs.</td>
</tr>
<tr>
<td></td>
<td>• <strong>API Manager Database</strong>: Stores information related to the APIs along with the API subscription details. The Key Manager Server uses this database to store user access tokens required for verification of API calls.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Registry Database</strong>: Shares information between the Publisher and Store. When an API is published through the Publisher, it is made available in the Store via the sharing registry database.</td>
</tr>
</tbody>
</table>

**Message flows**

The three main use cases of API Manager are API publishing, subscribing and invoking. Described below is how the message flow happens in these use cases.

- **Publishing APIs**
  A user assigned to the **publisher role** can publish APIs. This is done via the Publisher server. When an API is published in the API Publisher, it will be available in the API Store. Furthermore, the API Gateway must be updated with this API so that users can invoke it. As we are using a clustered Gateway, all Gateway server nodes in the cluster are updated with the published API details, enabling any of these Gateway nodes to serve API calls that are received. When an API is published, it is also pushed to the registry database, so that it can be made available on the store via the shared database.

- **Subscribing to APIs**
  A user with the subscriber role logs into the API Store and subscribes to an API. The user must then generate an access token to be able to invoke the API. When the subscriber requests to generate the token, a request is sent to the Key Manager Server cluster. The token is then generated, and the access token details are displayed to the subscriber via the Store.

- **Invoking APIs**
  Subscribed users can invoke an API to which they have subscribed. When the API is invoked, the request is sent to the API Gateway server cluster. The Gateway server then forwards the request to the Key Manager server cluster for verification. Once the request is verified, the Gateway connects to the back-end implementation and obtains the response, which is sent back to the subscriber via the Gateway server.

The diagram below summarizes these message flows where the Publisher is referred to as the publishing portal and the Store is referred to as the developer portal.
**Figure**: WSO2 API Manager message flows

WSO2 API Manager deployment patterns

[ Pattern 1 ] [ Pattern 2 ] [ Pattern 3 ] [ Pattern 4 ] [ Pattern 5 ]

**Pattern 1**

Single node (all-in-one) deployment.
You can use this pattern when you are working with a low throughput.

**Pattern 2**

Deployment with a separate Gateway and separate Key Manager.

You can use this pattern when you require a high throughput scenario that requires a shorter token lifespan.

**Pattern 3**
Fully distributed setup.

**Pattern 4**

Internal and external (on-premise) API Management.
You can use this pattern when you require a separate internal and external API Management with separated Gateway instances.

**Pattern 5**

Internal and external (public and private cloud) API Management.
You can use this pattern when you wish to maintain a cloud deployment as an external API Gateway layer.

**Traffic Manager scalable deployment patterns**

See the article on [Scalable Traffic Manager deployment patterns part 1](#) and [part 2](#).

**Scaling the Gateway**

Scaling the Gateway requires considering the load that the Gateways will handle. We recommend that a load test be carried out on the Gateways in the environment. For more information on what factors affect the load, and how the numbers have to be derived in order to scale, refer to the article on [Capacity Planning](#). Make sure to check the backends as well for the expected load.

**Clustering Gateways and Key Managers with key caching**

For key validation, the Gateway can usually handle 3,000 transactions per second (TPS), whereas the Key Manager can only handle 500 TPS. To improve performance, the key cache is enabled on the Gateway by default, which allows the system to handle 3,000 TPS. However, if you need better security, you can enable the cache on the Key Manager instead. Note the following about clustering with key caching:

- When the cache is enabled at the Gateway, you can have two Gateways per Key Manager.
- When the cache is enabled at the Key Manager and disabled at the Gateway, you can have only one Gateway per Key Manager.
- If both caches are disabled (not recommended), even with only one Gateway per Key Manager, the system may not be able to handle the load, as the Key Manager will only be able to handle 500 TPS.

For more information, see [Key cache](#) in the WSO2 API Manager documentation.

**Scaling Gateways and Traffic Managers**

Based on the performance numbers attained at the WSO2 Labs, the recommended ratio is 10 Gateways to 1 Traffic Manager (10:1, one Traffic Manager is tested to have successfully handled the load of 10 Gateways, one Gateway handling the load of ~5000 TPS). Note that Traffic Managers do not scale horizontally. If the requirement arises to have more than 10 Gateways, WSO2 recommends to have Traffic Managers placed in dedicated clusters of a maximum of 10 Gateways each. However, the number of Gateways in a cluster is not a hard limit and can vary according to your requirement.
Deploying API Manager using Single Node Instances

In a typical production deployment, API Manager is deployed as components (Publisher, Store, Gateway, Key Manager and Traffic Manager). While this provides very high performance and a high level of scalability, it may be too complex if you want to run API Manager as a small to medium scale API Management solution. A WSO2 API-M single node deployment, which has all the API-M components in one instance, would be simple to set up and requires less resources when compared with a distributed deployment. It is ideal for any organization that wants to start small and iteratively build up a robust API Management Platform.

WSO2 provides two options for organizations that are interested in setting up a small to medium scale API Management solution.

- Setting up on WSO2 API Cloud, which is a subscription based API Management solution. You can access this service by creating an account in WSO2 API Cloud.
- If you are interested in setting up a single node API Manager instance, which has all the API-M components in one instance, on-premise, you can download the latest version of API Manager and follow the instructions given below to set up the instance.

Prerequisites

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. Since this is an all-in-one deployment, it is recommended to use a higher hardware specification. You can further fine tune your operating system for production by tuning performance. For more information on installing the product on different operating systems, see Installing the Product.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>Oracle JDK 1.8 or Open JDK 8</td>
</tr>
</tbody>
</table>

You can deploy a single node API Manager instance in the following methods:

- Single node deployment
- Active/active deployment

**Single node deployment**

In this setup, API traffic is served by one all-in-one instance of WSO2 API Manager.
Pros

• Production support is required only for a single API Manager node (you receive 24*7 WSO2 production support).
• Deployment is up and running within hours.
• Can handle up to 43 million API calls a day (up to 500 API calls a second)
• Minimum hardware/cloud infrastructure requirements (only one node).
• Suitable for anyone new to API Management.

Cons

• Deployment does not provide High Availability.
• Not network friendly. Deploying on a demilitarized zone (DMZ) would require a Reverse Proxy.

For more information on manually configuring the production servers from scratch, see Configuring a Single Node.
Active/active deployment

In this setup, API traffic is served by two single node (all-in-one) instances of WSO2 API Manager.

Pros

- The system is highly available.
- Production support is required for 2 API Manager nodes (you receive 24*7 WSO2 production support).
- Can handle up to 86 million API calls a day (up to 1000 API calls a second)
- Deployment is up and running within hours.

Cons

- Not network friendly. Deploying on a DMZ would require a Reverse Proxy.

Configuring a Single Node

This page walks you through how to manually configure and deploy WSO2 API Manager in a standalone single instance, without using a distributed or HA
deployment patterns. For an overview of Single Node and Active-Active deployments, please see Deploying API Manager using Single Node Instances.

Follow the instructions below to configure and deploy API-M using a single node:

- Step 1 - Create a SSL certificate
- Step 2 - Configure the load balancer
- Step 3 - Configure the databases
- Step 4 - Configure hostnames that are used to expose APIs
- Step 5 - Configure your deployment with production hardening
- Step 6 - Configure API-M Analytics
- Step 7 - Start the WSO2 API-M server

Step 1 - Create a SSL certificate
Create a SSL certificate on the WSO2 API Manager node. For more information, see Creating SSL Certificates in the Administration Guide.

**Step 2 - Configure the load balancer**

For information on configuring the load balancer see Configuring the Proxy Server and the Load Balancer.

**Step 3 - Configure the databases**

For information on configuring the databases, see Installing and Configuring the Databases.

**Step 4 - Configure hostnames that are used to expose APIs**

This step is only required if you are using a hostname to expose APIs.

Click here for more information.

Add this hostname in the `<API-M_HOME>/repository/conf/api-manager.xml` file. Update the `<GatewayEndpoint>` element with your chosen hostname as shown below. In this case we are using `localhost` as the hostname:

```
<!-- Endpoint URLs for the APIs hosted in this API gateway.-->
```

**Step 5 - Configure your deployment with production hardening**

Ensure that you have taken into account the respective security hardening factors (e.g., changing and encrypting the default passwords, configuring JVM security etc.) before deploying WSO2 API-M. For more information, see the Production Deployment Guidelines in the Administration Guide.

**Step 6 - Configure API-M Analytics**

If you wish to view reports, statistics, and graphs related to the APIs deployed in the Store, you need to configure API-M Analytics. Follow the standard setup to configure API-M Analytics in a production setup, and follow the quick setup to configure API-M Analytics in a development setup.

**Step 7 - Start the WSO2 API-M server**

Before you start the server

If you want to deploy WSO2 API-M using a hybrid single node deployment, where WSO2 Identity Server is used as the Key Manager while the rest of the WSO2 API-M components are all in one node, configure and start the Key Manager (e.g., configure and start WSO2 Identity Server as the Key Manager) before starting the API-M server.

Start the server using the following standard start-up script. For more information, see Starting the server.

```
Linux/Mac OS

  cd <API-M_HOME>/bin/
  sh wso2server.sh
```

Windows
Configuring an Active-Active Deployment

This page walks you through how to manually configure WSO2 API Manager (WSO2 API-M) with two active nodes that each have all the components of the API-M together in one instance (all-in-one instance).

Follow the instructions below to configure and deploy API-M by using an Active-Active deployment:

- Step 1 - Create a SSL certificate
- Step 2 - Configure the load balancer
- Step 3 - Configure the databases
- Step 4 - Configure the Publisher with the Gateway
- Step 5 - Configure the content synchronization mechanism
- Step 6 - Configure Throttling
- Step 7 - Configure the second WSO2 API-M node
- Step 8 - Configure your deployment with production hardening
Step 1 - Create a SSL certificate
Create a SSL certificate on the first WSO2 API-M all-in-one active node. For more information, see Creating SSL Certificates in the Administration Guide.

Step 2 - Configure the load balancer
For information on configuring the load balancer, see Configuring the Proxy Server and the Load Balancer.

Step 3 - Configure the databases
For information on configuring the databases, see Installing and Configuring the Databases.

Step 4 - Configure the Publisher with the Gateway

This step is required only if you are using rsync to share files.

When you use rsync the file synchronization will happen in only one direction. Therefore, use the following configuration to enable synchronization in both directions between two nodes.

For more details, see the next step.

Configure the API Publisher in both nodes to be able to publish to the API-M Gateway of one of the nodes. Do this by pointing the \(<ServerURL>\) to the same Gateway node.

You need to configure this in the \(<API-M_HOME>/repository/conf/api-manager.xml\) file.

```xml
<APIGateway>
<ServerURL>https://localhost:${mgt.transport.https.port}${carbon.context}services/</ServerURL>
</APIGateway>
```

Step 5 - Configure the content synchronization mechanism
Configure a shared file system as the content synchronization mechanism. You can use a common shared file system such as Network File System (NFS) or any other shared file system that is available. You need to mount the \(<API-M_HOME>/repository/deployment/server\) directory of the two nodes to the shared file system, in order to share all APIs and throttling policies between all the nodes.

When using rsync the file synchronization will happen in only one direction. Therefore, use the following configuration to enable synchronization in both directions between two nodes.

1. Open \(<API-M_HOME>/repository/conf/api-manager.xml\) file in node-1.
2. Configure Gateway Server URL to point to its own (localhost):

```xml
<APIGateway>
<ServerURL>https://localhost:${mgt.transport.https.port}${carbon.context}services/</ServerURL>
</APIGateway>
```

3. Open \(<API-M_HOME>/repository/conf/api-manager.xml\) file in node-2.
4. Configure Gateway Server URL to point to the node-1:

```xml
<APIGateway>
<ServerURL>https://localhost:${mgt.transport.https.port}${carbon.context}services/</ServerURL>
</APIGateway>
```
Step 6 - Configure Throttling

Click here for information on configuring Throttling.

1. Configure the data publisher in the `<DataPublisher>` section which comes under the `<ThrottlingConfigurations>` section in the `<API-M_HOME>/repository/conf/api-manager.xml` file. You need to update these configurations so that the Gateway can publish data to the Traffic Manager in its own node and the Traffic Manager in the other node, so that the same event is sent to both servers at the same time. The WSO2 Complex Event Processor (WSO2 CEP) component that lies within the Traffic Manager acts as the data receiver and process the data to come up with the Throttling decisions.

```
<DataPublisher>
    <Enabled>true</Enabled>
    <Type>Binary</Type>
    <ReceiverUrlGroup>{tcp://<node1-hostname>:<node1-port>},{tcp://<node2-hostname>:<node2-port>}
    <AuthUrlGroup>{ssl://<node1-hostname>:<node1-port>},{ssl://<node2-hostname>:<node2-port>}
    ............
</DataPublisher>
```

2. Save your changes.

Step 7 - Configure the second WSO2 API-M node

Make a copy of the active instance configured above and use this copy as the second active instance.

When making a copy of the node, you need to also make a copy of the SSL certificate that you created for node 1 in step 1.

Step 8 - Configure your deployment with production hardening

Ensure that you have taken into account the respective security hardening factors (e.g., changing and encrypting the default passwords, configuring JVM security etc.) before deploying WSO2 API-M. For more information, see the Production Deployment Guidelines in the Administration Guide.

Step 9 - Configure Analytics

If you wish to view reports, statistics, and graphs related to the APIs deployed in the Store, you need to configure API-M Analytics. Follow the standard setup to configure API-M Analytics in a production setup, and follow the quick setup to configure API-M Analytics in a development setup.

Step 10 - Start the WSO2 API-M servers

Before you start the servers
Start the WSO2 API-M servers using the standard start-up script. For more information, see Starting the server.

**Windows**

```
cd <API-M_HOME>/bin/
wso2server.bat --run
```

**Linux/Mac OS**

```
cd <API-M_HOME>/bin/
sh wso2server.sh
```

### Configuring the keyStore and client trustStore for a Selected Pattern

The WSO2 API-M GitHub repository includes a custom keyStore and client trustStore in the `<API-M_HOME>/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepacked` module as well.

This `wso2carbon.jks` keyStore is created for `wso2am_analytics`, and its self-signed certificate is imported into the `client-truststore.jks`.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see Creating New Keystores.

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. **Generate a Java keyStore and key pair with a self-signed certificate.**
   
   **Example**
   
   ```
   
   ```

2. **Export the certificate from the latter mentioned keyStore.**
   
   **Example**
   
   ```
   keytool -export -keystore <keystore-name> -alias <alias-of-the-certificate> -file <output-file-name>.cer
   
   keytool -export -keystore wso2carbon.jks -alias wso2carbon -file wso2carbon.cer
   ```

3. **Import the latter mentioned certificate into a trustStore.**
   
   **Example**
### Distributed Deployment of API Manager

**WSO2 API Manager** (WSO2 API-M) is a complete API management solution, used for creating and publishing APIs, creating and managing a developer community, and routing API traffic scalably. The WSO2 API-M includes the following five components: Publisher, Store, Gateway, Key Manager, and Traffic Manager.

Typically, when you get started with WSO2 API Manager in a development environment, you deploy WSO2 API Manager as a single instance with all its components on a single server. For details, see [Deploying API Manager using Single Node Instances](#).

However, in a production deployment, these components are deployed in a distributed manner. Therefore, you can create a distributed deployment of WSO2 API-M’s five main components. This page describes how to set up and deploy WSO2 API-M as a distributed deployment.

Note that your configurations may vary depending on the WSO2 API Manager deployment pattern that you choose. If you are using multi-tenancy, all nodes should use the same user store, as all servers are servicing the same set of tenants, and it has to share the same Governance Registry space across all nodes.

- Understanding the Distributed Deployment of WSO2 API-M
- Deploying WSO2 API-M in a Distributed Setup

#### Understanding the Distributed Deployment of WSO2 API-M

Before understanding how to deploy WSO2 API Manager (WSO2 API-M), let's understand the WSO2 API-M distributed deployment better.

- Understanding the WSO2 API-M architecture
- Understanding the distributed deployment

**Understanding the WSO2 API-M architecture**

WSO2 API Manager uses the following main components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publisher</td>
<td>Enables API providers to easily publish their APIs, share documentation, provision API keys, and gather feedback on API features, quality, and usage.</td>
</tr>
<tr>
<td>Store</td>
<td>Enables consumers to self-register, discover API functionality, subscribe to APIs, evaluate them, and interact with API publishers.</td>
</tr>
<tr>
<td>Key Manager</td>
<td>Responsible for all security and key-related operations.</td>
</tr>
<tr>
<td>Gateway</td>
<td>Responsible for securing, protecting, managing, and scaling API calls.</td>
</tr>
<tr>
<td>Traffic Manager</td>
<td>Used to make a decision on throttling.</td>
</tr>
</tbody>
</table>

For more information on the above, see the main components of a distributed system.

Additionally, API Manager uses the following databases, which are shared among the server nodes.

- **User Manager database** - Stores information related to users and user roles. This information is shared among the Key Manager Server, Store, and Publisher. Users can access the Publisher for API creation and the Store for consuming the APIs. The User Manager database is also referred to as `WSO2UM_DB` and `userdb`.
- **API Manager database** - Stores information related to the APIs along with the API subscription details. The Key Manager Server uses this database to store user access tokens that are used for verification of API calls. The API Manager database is also referred to as `WSO2_AM_DB` and `apimgt_db`.
- **Registry database** - Shares information between the Publisher and Store. When an API is published through the Publisher, it is made available in the Store via the shared registry database. Although you would normally share information between the Publisher and Store components only, if you are planning to create this setup for a multi-tenanted environment (create and work with tenants), it is required to share the information in this database between the Gateway and Key Manager components as well. The Registry database is also referred to as `WSO2REG_DB` and `regdb`.
- **Statistics database** - Stores information related to API statistics. After you configure API-M analytics, it writes summarized data to this database. The Publisher and Store can then query this database to display the statistics data. The Statistics database is also referred to as `WSO2_STAT_DB` and `statdb`.
- **Message Broker database** - Traffic Manager uses this database as the message store for broker when advanced throttling is used. The Message
Broker DB is also referred to as WSO2_MB_STORE_DB and mbstoredb.

WSO2 API Manager components use the databases as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>API Manager database</th>
<th>User Manager Database</th>
<th>Registry Database</th>
<th>Statistics Database</th>
<th>Message Broker Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publisher</td>
<td>Used</td>
<td>Used</td>
<td>Used</td>
<td>Used</td>
<td>Not used</td>
</tr>
<tr>
<td>Store</td>
<td>Used</td>
<td>Used</td>
<td>Used</td>
<td>Used</td>
<td>Not used</td>
</tr>
<tr>
<td>Key Manager</td>
<td>Used</td>
<td>Used</td>
<td>Used (in multi-tenancy mode)</td>
<td>Not used</td>
<td>Not used</td>
</tr>
<tr>
<td>Gateway</td>
<td>Not used</td>
<td>Used (in multi-tenancy mode)</td>
<td>Used (in multi-tenancy mode/ in multiple gateway mode when Google Analytics is used)</td>
<td>Not used</td>
<td>Not used</td>
</tr>
<tr>
<td>Traffic Manager</td>
<td>Not used</td>
<td>Not used</td>
<td>Not used</td>
<td>Not used</td>
<td>Used</td>
</tr>
</tbody>
</table>

**Notes**

- Although the Gateway does not use the WSO2 API Manager database, this connection is required; therefore, do not remove the default configuration in the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file. This connection should be your default database.
- The Gateway node creates a connection at the startup with the WSO2 API Manager database, but this connection will not be used later on.
- If you have more than one Traffic Manager node, each Traffic Manager node must have its own Message Broker database (WSO2_MB_STORE_DB).

When we consider a distributed deployment of WSO2 API Manager, we have the option of separating the five components and clustering each component as needed. Let’s look more closely at how the API Manager components are deployed separately.

**Understanding the distributed deployment**

In the following diagram, the five components are set up in a distributed deployment, and the five databases are connected to the relevant components respectively. The entire setup is also fronted by a load balancer.

In a clustered setup, if the **Key Manager** is NOT fronted by a load balancer, you have to set the `KeyValidatorClientType` element to `ThriftClient` in the `<API-M_HOME>/repository/conf/api-manager.xml` file, to enable Thrift as the communication protocol. You need to configure this in all the Gateway and Key Manager components.
Deploying WSO2 API-M in a Distributed Setup

Follow the instructions below to deploy WSO2 API Manager (WSO2 API-M) in a distributed environment, as depicted in the following deployment diagram:

- Step 1 - Install and configure WSO2 API-M
- Step 2 - Install and configure the databases
- Step 3 - Configure your deployment with production hardening
- Step 4 - Create and import SSL certificates
- Step 5 - Configure API-M Analytics
- Step 6 - Configure the connections among the components and start the servers
Step 1 - Install and configure WSO2 API-M

Click here for information on installing and configuring WSO2 API-M.

The following steps describe how to download, install, and configure WSO2 API Manager, with five instances.

1. Download the WSO2 API Manager in each of the five servers in the cluster for distributed deployment.
2. Unzip the WSO2 API Manager zipped archive, and rename each of those directories respectively as Key Manager, Gateway, Publisher, Store, and Traffic Manager. These five directories are located in a server of their own and are used for each component of WSO2 API-M. Each of these unzipped directories are referred to as `<API-M_HOME>` or `<PRODUCT_HOME>` in this document.
3. In each of the five servers, replace the default certificates (where `CN=localhost`) with new certificates generated with proper common name (CN) values. You need to do this in order to avoid getting an error with regard to the fact that the hostname in the certificate did not match.
Step 2 - Install and configure the databases

You can create the required databases for the API-M deployment in a separate server and point to the databases from the respective nodes. For information on configuring the databases, see Installing and Configuring the Databases.

Step 3 - Configure your deployment with production hardening

Ensure that you have taken into account the respective security hardening factors (e.g., changing and encrypting the default passwords, configuring JVM security, etc.) before deploying WSO2 API-M. For more information, see the Production Deployment Guidelines in the Administration Guide.

Step 4 - Create and import SSL certificates

Create a SSL certificate for each of the WSO2 API-M nodes (e.g., Publisher, Store, Key Manager, Gateway, and Traffic Manager) and import them to the keyStore and the trustStore. For more information, see Creating SSL Certificates in the Administration Guide.

Step 5 - Configure API-M Analytics

If you wish to view reports, statistics, and graphs related to the APIs deployed in the Store, you need to configure API-M Analytics. Follow the standard setup to configure API-M Analytics in a production setup, and follow the quick setup to configure API-M Analytics in a development setup.

Step 6 - Configure the connections among the components and start the servers

You will now configure the inter-component relationships of the distributed setup by modifying their `<API-M_HOME>/repository/conf/api-manager.xml` files. It is recommended to start the components in the following order: Key Manager, Publisher, Store, Traffic Manager, and Gateway.

In a clustered environment, you use session affinity (sticky sessions) to ensure that requests from the same client always get routed to the same server.

It is mandatory to set up Session Affinity in the load balancers that front the Publisher and Store clusters, and it is optional in the load balancer (if any) that fronts a Key Manager cluster or Gateway Cluster. However, you need to enable Session Affinity if you are working with multiple Gateway Managers in a Gateway High Availability (HA) deployment.

However, authentication via session ID fails when session affinity is disabled in the load balancer.

First time authentication happens via Basic Auth and the Gateway gets a cookie. This cookie is used in every consequent request along with the Basic Auth credentials. The admin service validates the cookie and if the validation fails it re-authenticates it using Basic Auth and issues a new cookie.

Click here for information on configuring the connections among the components and starting the servers.

- Step 6.1 - Configure the common configurations
- Step 6.2 - Configure and start the Key Manager
- Step 6.3 - Configure and start the API Publisher
- Step 6.4 - Configure and start the API Store
- Step 6.5 - Configure and start the Traffic Manager
- Step 6.6 - Configure and start the Gateway

Step 6.1 - Configure the common configurations

The common configurations can be done automatically when starting up the server. For instructions, see Starting an API-M profile.

When a node starts, it starts all the components and features bundled with it. If you are concerned about resource utilization, you can run the product on a specific profile, so that only the components and features that are required for that node and common features start up.

**Example**

```
sh <PRODUCT_HOME>/bin/wso2server.sh -Dprofile=api-store
```

For more information on using multi-profile support, see Product Profiles.
Step 6.2 - Configure and start the Key Manager

This section involves setting up the Key Manager node and enabling it to work with the other components in a distributed deployment.

Skip this step if you are using WSO2 Identity Server as the Key Manager and follow the instructions mentioned in Configuring WSO2 Identity Server as a Key Manager to configure and start the Key Manager.

1. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file in the Key Manager node and change the `<ServerURL>` element that appears under the `<APIGateway>` section, so that it points to the API Manager Gateway.

   You need to add these configurations so that when a user is deleted or when the role of a user is updated in the Key Manager, it will update the Gateway cache by clearing the cache entries of a particular user.

   - If you are working with a single Gateway in distributed set up, you need to replace `[GATEWAY_SERVER_HOST]` with the host of the Gateway node.
   - If you are working with Gateways in a High Availability (HA) setup that uses a shared file system (e.g., NFS), you need to replace `[GATEWAY_SERVER_HOST]` with the host of the Gateway load balancer node.
   - If you are working with Gateways in a High Availability (HA) setup that uses rsync, you need to replace `[GATEWAY_SERVER_HOST]` with the host of the Gateway Manager node.
   - You need to replace `[port]` with the management transport port. For more information, see Default Product Ports.

   ```xml
   <ServerURL>https://[GATEWAY_SERVER_HOST]:[port]/services/</ServerURL>
   ```

2. Configure the API key validator in the Key Manager.

   The Thrift protocol is normally enabled by default. However, if you have disabled the Thrift protocol, enable it as follows in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

   **Single Key Manager Key Manager with HA**

   When you are using a single Key Manager, add ThriftClient for the `<KeyValidatorClientType>` element to use the Thrift protocol.

   ```xml
   <APIKeyValidator>
     <KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
     <EnableThriftServer>true</EnableThriftServer>
     <ThriftServerHost=localhost</ThriftServerHost>
     <ThriftServerPort>10397</ThriftServerPort>
     ...
   </APIKeyValidator>
   ```

   The default Thrift server port is 10397. You need to only uncomment the following code to set the Thrift server port if you need to use a port that differs from the default value.

   ```xml
   <ThriftServerPort>[port]</ThriftServerPort>
   ```

   When you are using multiple Key Managers fronted by a load balancer, you need to add WSClient for the `<KeyValidatorClientType>` element to use the Web Service Client, and change `<EnableThriftServer>` to `false` to optimize performance.

   ```xml
   <APIKeyValidator>
     ...
     <KeyValidatorClientType>WSClient</KeyValidatorClientType>
     <EnableThriftServer>false</EnableThriftServer>
     <ThriftServerHost=localhost</ThriftServerHost>
     <ThriftServerPort>10397</ThriftServerPort>
     ...
   </APIKeyValidator>
   ```

   If you wish to encrypt the Auth Keys (access tokens, client secrets, and authorization codes), see Encrypting OAuth Keys.
3. Optionally, configure High Availability (HA) for the Key Manager.

These steps are ONLY applicable if you need to configure HA for the Key Manager.

a. Make a copy of the active instance configured above and use this copy as the second Key Manager active instance.
b. Configure a load balancer to front the two Key Manager nodes.
For information on configuring the load balancer, see Configuring the Proxy Server and the Load Balancer.

4. Start the WSO2 API-M Key Manager node(s) by typing the following command in the command prompt. For more information on starting a WSO2 server, see Starting the server.

**Linux/Mac OS**

```bash
cd <API-M_HOME>/bin/
sh wso2server.sh
```

**Windows**

```batch
cd <API-M_HOME>\bin\
wso2server.bat --run
```

It is not recommended to share the Solr directory between the Store and Publisher servers. You need to have separate Solr directories for each of the latter mentioned servers so that they will perform Solr indexing separately.

If you get an error similar to the following in both or one of the nodes, check whether you have shared the Solr directory.

```
org.apache.solr.common.SolrException:
SolrCore 'registry-indexing' is not available due to init failure: Index locked
for write for core registry-indexing
```

Step 6.3 - Configure and start the API Publisher

This section involves setting up the API Publisher node and enabling it to work with the other components in the distributed deployment.

1. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file in the API Publisher node and make the following changes.
   a. Configure the Publisher with the Key Manager.
      You need to update the following configuration ONLY when you do not wish to share the user stores with the WSO2 API-M instance.

      This step is not applicable if you are enabling Single Sign-on (SSO).

Single Key Manager

**Key Manager with HA**

Configure the Publisher with a single Key Manager as follows:

```xml
<AuthManager>
  <ServerURL>https://[Key-Manager-host]:9443/services/</ServerURL>
  <Username>admin</Username>
  <Password>admin</Password>
</AuthManager>
```

Configure the Publisher with multiple Key Managers that are fronted by a load balancer as follows:
Traffic Manager with HA

1. Configure the Publisher with the Traffic Manager.
   This configuration enables the publishing of throttling policies, custom templates, and block conditions to the Gateway node.

   **Single Traffic Manager**
   Traffic Manager with HA

   Configure the **Publisher with a single Traffic Manager** as follows:

   ```xml
   <ThrottlingConfigurations>
   <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
   <DataPublisher>
     <Enabled>true</Enabled>
     Type>Binary</Type>
     <ReceiverUrlGroup>tcp://[Traffic-Manager-host]:9611</ReceiverUrlGroup>
     <AuthUrlGroup>ssl://[Traffic-Manager-host]:9711</AuthUrlGroup>
   </DataPublisher>
   <PolicyDeployer>
     <ServiceURL>https://[Traffic-Manager-Host]:9443/services/</ServiceURL>
     <Username>${admin.username}</Username>
     <Password>${admin.password}</Password>
   </PolicyDeployer>
   <BlockCondition>
     <Enabled>false</Enabled>
   </BlockCondition>
   <JMSConnectionDetails>
     <Enabled>false</Enabled>
   </JMSConnectionDetails>
   </ThrottlingConfigurations>
   ```

   Configure the **Publisher with multiple Traffic Managers** that are fronted by a load balancer as follows:
<ThrottlingConfigurations>
  <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
  <DataPublisher>
    <Enabled>true</Enabled>
    <Type>Binary</Type>
    <ReceiverUrlGroup>{tcp://[Traffic-Manager-host]:9611},
    {tcp://[Traffic-Manager-host]:9611}</ReceiverUrlGroup>
    <AuthUrlGroup>{ssl://[Traffic-Manager-host]:9711},
    {ssl://[Traffic-Manager-host]:9711}</AuthUrlGroup>
    <PolicyDeployer>
      <ServiceURL>https://[Traffic-Manager-LB-Host]:9443/services/</ServiceURL>
      <Username>${admin.username}</Username>
      <Password>${admin.password}</Password>
    </PolicyDeployer>
    <BlockCondition>
      <Enabled>false</Enabled>
    </BlockCondition>
    <JMSConnectionDetails>
      <Enabled>false</Enabled>
    </JMSConnectionDetails>
  </DataPublisher>
</ThrottlingConfigurations>

Configure the Publisher with the Gateway.

You need to add these configurations, because when creating an API, it calls the Gateway endpoint to create the actual Synapse file.

- If you are using a single Gateway node, configure the Publisher with the Gateway as follows:

  <APIGateway>
    <Environments>
      <Environment type="hybrid" api-console="true">
        <Name>Production and Sandbox</Name>
        <Description>This is a hybrid gateway that handles both production and sandbox token traffic.</Description>
        <ServerURL>https://[API-Gateway-Host-or-IP]:9443/services/</ServerURL>
        <Username>${admin.username}</Username>
        <Password>${admin.password}</Password>
      </Environment>
    </Environments>
    <GatewayEndpoint>http://[API-Gateway-Host]:8243</GatewayEndpoint>
  </APIGateway>

- If you are using multiple Gateway nodes, configure the Publisher with the Gateway nodes as follows:

Configure the Publisher when working with multiple Gateways that are fronted by a load balancer when using a shared file system (e.g., NFS), to synchronize the data between your Gateway nodes as follows:
Configure the **Publisher with a multiple Gateways** that are fronted by a load balancer when using Remote Synchronization (rsync), to synchronize the data between your Gateway nodes as follows:

```xml
<APIGateway>
  <Environments>
    <Environment type="hybrid" api-console="true">
      <Name>Production and Sandbox</Name>
      <Description>This is a hybrid gateway that handles both production and sandbox token traffic.</Description>
      <ServerURL>https://[API-Gateway-LB-Host-or-IP]:9443/services/</ServerURL>
      <Username>${admin.username}</Username>
      <Password>${admin.password}</Password>
      <GatewayEndpoint>http://[API-Gateway-LB-Host]:8280,https://[API-Gateway-LB-Host]:8243</GatewayEndpoint>
    </Environment>
  </Environments>
</APIGateway>
```

Configure the **Publisher with a single API Store** as follows:

```xml
<APIStore>
  <DisplayURL>true</DisplayURL>
  <URL>https://[Store-hostname]:9443/store</URL>
</APIStore>
```

Configure the **Publisher with multiple API Stores** that are fronted by a load balancer as follows:

To change the admin password, see **Changing the super admin password**. Furthermore, make sure to adhere to the note given under step 2 in the latter mentioned section if your password has special characters.

d. Configure the Store URL to appear in the Publisher UI. For this purpose you need to set the `<DisplayURL>` to `true` and provide the URL of the Store.

**Single Store Store with HA**

Example

```xml
<APIStore>
  <DisplayURL>true</DisplayURL>
  <URL>https://[Store-hostname]:9443/store</URL>
</APIStore>
```
2. Configure the blocked apps and API notifications to go to the Topic.
   Open the `<API-M_HOME>/repository/conf/jndi.properties` file and make the following changes.

   - The following configuration is related to the Admin App in WSO2 API-M. In this guide it is assumed that the WSO2 API-M Admin App is configured in the Publisher node.
   - If you change the default username (i.e., admin) and password (i.e., admin) in the `user-mgt.xml` file, that username and password should be changed at the broker connection URL as well.

   ```
   connectionfactory.TopicConnectionFactory =
   amqp://admin:admin@clientid/carbon?brokerlist='tcp://[Traffic-Manager-host]:[Port-number]'
   topic.throttleData = throttleData
   ```

3. Disable the Thrift Server to optimize performance.
   You need to configure this in the Publisher `<API-M_HOME>/repository/conf/api-manager.xml` file.

   ```
   <APIKeyValidator>
   ...
   <EnableThriftServer>false</EnableThriftServer>
   </APIKeyValidator>
   ```

4. Optionally, configure High Availability (HA) for the Publisher.

   These instructions are ONLY applicable if you need to configure HA for the Publisher.

   a. Make a copy of the active Publisher instance configured above and use this copy as the second active Publisher instance.
   b. Configure a load balancer to front the two Publisher nodes.
   For information on configuring the load balancer, see Configuring the Proxy Server and the Load Balancer.

5. Start the WSO2 API-M Publisher node(s) by typing the following command in the command prompt.
   For more information on starting a WSO2 server, see Starting the server.

   ```
   cd <API-M_HOME>/bin/
   sh wso2server.sh
   ```

   ```
   cd <API-M_HOME>/bin\
   wso2server.bat --run
   ```

Step 6.4 - Configure and start the API Store

This section involves setting up the API Store node and enabling it to work with the other components in the distributed deployment.

1. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file in the API Store node and make the following changes.
   a. Configure the API Store with the Key Manager.
      Single Key Manager
      Key Manager with HA

   Configure the API Store with a single Key Manager as follows:
1. Configure the API key validator.
   When you are connecting the API Store directly to the Key Manager, add ThriftClient for the `<KeyValidatorClientType>` element to use the Thrift protocol.

```xml
<APIKeyValidator>
  <ServerURL>https://[Key-Manager-host]:9443/services/</ServerURL>
  <Username>${admin.username}</Username>
  <Password>${admin.password}</Password>
  <KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
  <ThriftClientConnectionTimeOut>10000</ThriftClientConnectionTimeOut>
  <!--ThriftClientPort>10397</ThriftClientPort-->
  <EnableThriftServer>true</EnableThriftServer>
  <ThriftServerHost>[Key-Manager_host]</ThriftServerHost>
  <!--ThriftServerPort>10397</ThriftServerPort-->
  ...
</APIKeyValidator>
```

2. Configure the Authentication Manager, so that the API Store can connect to the Key Manager.
   You need to update the following configuration ONLY when you do not wish to share the user stores with the WSO2 API-M instance.

   **This step is not applicable if you are enabling Single Sign-on (SSO).**

   ```xml
   <AuthManager>
     <ServerURL>https://[Key-Manager-host]:9443/services/</ServerURL>
     <Username>${admin.username}</Username>
     <Password>${admin.password}</Password>
   </AuthManager>
   ```

Configure the **Store with multiple Key Managers** that are fronted by a load balancer as follows:

1. Configure the API key validator.
   When you are using multiple Key Managers fronted by a load balancer, you need to add WSClient for the `<KeyValidatorClientType>` element to use the Web Service Client.

   ```xml
   <RevokeAPIURL>https://[Gateway-host]:8243/revoke</RevokeAPIURL>
   <APIKeyValidator>
     <ServerURL>https://[Key-Manager-LB-host]:9443/services/</ServerURL>
     <Username>${admin.username}</Username>
     <Password>${admin.password}</Password>
     <KeyValidatorClientType>WSClient</KeyValidatorClientType>
     <ThriftClientConnectionTimeOut>10000</ThriftClientConnectionTimeOut>
     <!--ThriftClientPort>10397</ThriftClientPort-->
     <EnableThriftServer>false</EnableThriftServer>
     <ThriftServerHost>localhost</ThriftServerHost>
     <!--ThriftServerPort>10397</ThriftServerPort-->
     ...
   </APIKeyValidator>
   ```

2. Configure the Authentication Manager, so that the API Store can connect to the Key Manager.
   You need to update the following configuration ONLY when you do not wish to share the user stores with the WSO2 API-M instance.

   **This step is not applicable if you are enabling Single Sign-on (SSO).**

   ```xml
   <AuthManager>
     <ServerURL>https://[Key-Manager-host]:9443/services/</ServerURL>
     <Username>${admin.username}</Username>
     <Password>${admin.password}</Password>
   </AuthManager>"
b. Make the following throttling related changes that correspond to the Traffic Manager.

```xml
<ThrottlingConfigurations>
  <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
  <DataPublisher>
    <Enabled>false</Enabled>
    …
  </DataPublisher>
  <BlockCondition>
    <Enabled>false</Enabled>
    …
  </BlockCondition>
  <JMSConnectionDetails>
    <Enabled>false</Enabled>
    …
  </JMSConnectionDetails>
  …
</ThrottlingConfigurations>
```

Configure the Store with the Gateway.

* If you are using a single Gateway node, configure the Store with the Gateway as follows:

```xml
<APIGateway>
  <Environments>
    <Environment type="hybrid">
      …
      <ServerURL>https://[API-Gateway-host-or-IP]:9443/services/</ServerURL>
      …
      <GatewayEndpoint>http://[API-Gateway-host-or-IP]:8280,https://[API-Gateway-host]:8443</GatewayEndpoint>
    </Environment>
    …
  </Environments>
</APIGateway>
```

* If you are using multiple Gateway nodes, configure the Store with the Gateway nodes as follows:

Configure the Store when working with multiple Gateways that are fronted by a load balancer, and when using a shared file system (e.g., NFS), to synchronize the data between your Gateway nodes as follows:
Configure the **Store with multiple Gateways** that are fronted by a load balancer, and when using Remote Synchronization (rsync), to synchronize the data between your Gateway nodes as follows:

```xml
<APIGateway>
    <Environments>
        <Environment type="hybrid">
            ...
            <ServerURL>https://[API-Gateway-LB-Host-or-IP]:9443/services/</ServerURL>
            <Username>${admin.username}</Username>
            <Password>${admin.password}</Password>
            <GatewayEndpoint>http://[API-Gateway-LB-Host]:8280,https://[API-Gateway-LB-Host]:8243</GatewayEndpoint>
        </Environment>
    </Environments>
    ...
</APIGateway>
```

Configure the **Token Revoke endpoint** to point to Gateway.

**Single Gateway Gateway with HA**

Configure the **Token Revoke endpoint with a single Gateway** as follows:

```xml
<OAuthConfigurations>
    ...
    <RevokeAPIURL>https://[API-Gateway-host]:8243/revoke</RevokeAPIURL>
</OAuthConfigurations>
```

Configure the **Token Revoke endpoint with multiple Gateways**, which are fronted by a load balancer as follows:

```xml
<OAuthConfigurations>
    ...
    <RevokeAPIURL>https://[API-Gateway-Worker-LB-host]:8243/revoke</RevokeAPIURL>
</OAuthConfigurations>
```

2. Disable the Thrift Server to optimize performance.
   You need to configure this in the **Store** `<API-M_HOME>/repository/conf/api-manager.xml` file.

```xml
<APIKeyValidator>
    ...
    <EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```
3. Optionally, configure High Availability (HA) for the Store.

This is **ONLY applicable** if you need to configure HA for the Store.

Make a copy of the active instance configured above and use this copy as the second API Store active instance.

4. Start the API Store node(s) by typing the following command in the command prompt. For more information on starting a WSO2 server, see *Starting the server*.

   **Linux/Mac OS**

   ```bash
   cd <API-M_HOME>/bin/
   sh wso2server.sh
   
   cd <API-M_HOME>
   startserver.bat --run
   ```

**Step 6.5 - Configure and start the Traffic Manager**

This section involves setting up the Traffic Manager node(s) and enabling it to work with the other components in a distributed deployment.

1. Delete the `<API-M_HOME>/repository/conf/registry.xml` and rename the `<API-M_HOME>/repository/conf/registry_tm.xml` file as the `registry.xml` file.

   To disable registry indexing when setting up the Traffic Manager, see *Registry indexing configurations*.

2. Delete the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file and rename the `<API-M_HOME>/repository/conf/axis2/axis2_TM.xml` file.

   Note that all the jaggery apps and webapps in the `<API-M_HOME>/repository/deployment/server/` directory are removed when you optimize the profile. For more details, see *Product Profiles*.

3. Optionally, mount the `<API-M_HOME>/repository/deployment/server` directory of all the Traffic Manager nodes to the shared file system.

   This step is **ONLY applicable** if you are configuring the Traffic Manager with HA and **shared file system** as the content synchronization mechanism.

   You need to do this to share all the Throttling policies between traffic management nodes.

4. Disable the Thrift Server to optimize performance.

   You need to configure this in the Traffic Manager `<API-M_HOME>/repository/conf/api-manager.xml` file.

   ```xml
   <APIKeyValidator>
   ...  
   <EnableThriftServer>false</EnableThriftServer>
   </APIKeyValidator>
   ```

5. Optionally, configure High Availability (HA) for the Traffic Manager.

   This is **ONLY applicable** if you need to configure HA for the Traffic Manager.

   Make a copy of the active instance configured above and use this copy as the second active Traffic Manager instance.

6. Start the WSO2 API-M Traffic Manager node(s) by typing the following command in the command prompt. For more information on starting a WSO2 server, see *Starting the server*.

   **Linux/Mac OS**

   ```bash
   cd <API-M_HOME>/bin/
   sh wso2server.sh -Dprofile=traffic-manager
   ```
Always start the Traffic Manager using the -Dprofile=traffic-manager profile to avoid FATAL errors such as the following.

```
FATAL - ServiceBusInitializer Couldn't initialize the ESB...
org.apache.synapse.SynapseException: The synapse.xml location ././
  ./repository/deployment/server/synapse-configs/default doesn't exist
```

### Troubleshooting

If you have a firewall between the Traffic Manager and the Gateway, you need to configure the heartbeat value to keep the JMS connection alive. To configure this, open the `<APIM_HOME>/repository/conf/advanced/qpid-config.xml` file and set the heartbeat to a non-zero value as shown below.

```
<heartbeat>
  <delay>60</delay>
  <timeoutFactor>2.0</timeoutFactor>
</heartbeat>
```

## Step 6.6 - Configure and start the Gateway

This section involves setting up the Gateway node and enabling it to work with the other components in the distributed deployment.

### Steps 1 to 5

Steps 1 to 5 in the following section are common irrespective of your API-M deployment, such as deploying a single Gateway node or deploying multiple Gateway nodes for High Availability (HA). However, if you are using two Gateway nodes for high availability (HA), first follow the instructions that is available in the Distributed Deployment of the Gateway document, and then carry out the following steps to configure the connections from Gateway(s) to other components.

1. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file in the Gateway node.
2. Modify the api-manager.xml file as follows. This configures the connection to the Key Manager component.

#### Single Key Manager with HA

Configure the Gateway with a single Key Manager as follows:

a. Configure the API Key Validator.

```
<APIKeyValidator>
  <ServerURL>https://[Key-Manager-host]:9443/services/</ServerURL>
  <Username>${admin.username}</Username>
  <Password>${admin.password}</Password>
  ...
  <ThriftServerHost>[Key-Manager-host]</ThriftServerHost>
  ...
</APIKeyValidator>
```

[Key-Manager-host] - If you have a single Key Manager node, this should be the host of the Key Manager (i.e., the host of the WSO2 Identity Server).

To change the admin password, see Changing the super admin password. If your password has special characters, follow the guidelines mentioned as a note under step 2 in the latter mentioned section.
b. Use ThriftClient as the KeyValidatorClientType in the <API-M_HOME>/repository/conf/api-manager.xml file.

You can only use the Thrift protocol when the Key Manager cluster is NOT fronted by a load balancer.

```xml
<KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
```

c. Disable the Thrift Server to optimize performance.
You need to configure this in the Gateway <API-M_HOME>/repository/conf/api-manager.xml file.

```xml
<APIKeyValidator>
  ...
  <EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

d. Uncomment ThriftClientPort element if its commented out and set the Thrift Client port in the Gateway to the same port as the Thrift Server port, which you defined in the Key Manager. This enables the Gateway to communicate with the Key Manager.

Do this step only if one of the following scenarios are applicable to your set up.

- You started or you plan on starting the Key Manager or Gateway with a port offset.
- Your Thrift Server port in the Key Manager differs from the default value, which is 10397.

```xml
<ThriftClientPort>[port]</ThriftClientPort>
```

e. Specify the hostname or IP of the Key Manager.
The default is localhost. In a distributed deployment you must set this parameter in both Key Manager nodes and Gateway nodes only if the Key Manager is running on a separate machine. Gateway uses this parameter to connect to the key validation Thrift service.

```xml
<ThriftServerHost>[Key-Manager-host]</ThriftServerHost>
```

The parameter ThriftClientConnectionTimeOut is used to specify the client side time-out when connecting to Thrift Key Validation Service in Key Manager. The default value is 10000 milliseconds, which is sufficient for most cases.

Configure the Gateway with multiple Key Managers, which are fronted by a load balancer as follows:

a. Configure the APIKeyValidator as follows:

```xml
<APIKeyValidator>
  <ServerURL>https://[Key-Manager-LB-host]:9443/services/</ServerURL>
  <Username>${admin.username}</Username>
  <Password>${admin.password}</Password>
  ...
</APIKeyValidator>
```

- [Key-Manager-LB-host]: If there are multiple Key Managers (i.e., Multiple WSO2 Identity Servers as the Key Manager) fronted by a load balancer, this should be the host of the Key Manager's load balancer. For example, in the configuration we have defined key-manager as the load balancer host in the Key Manager section.
To change the admin password, see Changing the super admin password. If your password has special characters, follow the guidelines mentioned as a note under step 2 in the latter mentioned section.

b. Use WSClent as KeyValidatorClientType in the <API-M_HOME>/repository/conf/api-manager.xml file. Note that you can only use the Web Service Client when the Key Manager cluster is fronted by a load balancer.

```xml
<KeyValidatorClientType>WSClent</KeyValidatorClientType>
```

c. Ensure that Thrift is disabled in the Gateway. This is enabled by default in all instances of the product, so you need to disable the Thrift server by setting EnableThriftServer to false in the <API-M_HOME>/repository/conf/api-manager.xml file of each node.

```xml
<EnableThriftServer>false</EnableThriftServer>
```

3. If you need to enable JSON Web Token (JWT), you have to enable it in all Gateway and Key Manager components. For more information on configuring JWT, see Generating JSON Web Token.

4. Configure the Gateway to communicate with the Traffic Manager.

   You need to do this to enable Throttling for the Traffic Manager node(s).

   These configurations vary based on whether you have a single Traffic Manager node or multiple Traffic Manager nodes.

**Single Traffic Manager HA of Traffic Manager**

Configure the **Gateway with a single Traffic Manager** as follows:

9611 and 9711 are the Traffic Manager receiver ports for the binary type.

   a. Update the Throttling configurations as follows:

```xml
<T throttlingCon</t
<ironAdvanceThrottling>true</ironAdvanceThrottling>
<DataPublisher>
  <Enabled>true</Enabled>
  <Type>Binary</Type>
  <ReceiverUrlGroup>tcp://[Traffic-Manager-host]:9611</ReceiverUrlGroup>
  <AuthUrlGroup>ssl://[Traffic-Manager-host]:9711</AuthUrlGroup>
  ...
</DataPublisher>

PolicyDeployer
  <Enabled>false</Enabled>
  <ServiceURL>https://[Traffic-Manager-host]:9443/services/</ServiceURL>
  ...
</PolicyDeployer>

JMSConnectionDetails
  <Enabled>true</Enabled>
  <ServiceURL>tcp://[Traffic-Manager-host]:5672</ServiceURL>
  ...
</JMSConnectionDetails>
</ThrottlingConfigurations>
```

   In the Gateway profile of WSO2 API Manager, disable the <PolicyDeployer> configuration to prevent blocking the API Publisher.

   b. Configure JMSConnectionParameters to connect to the broker running within the Traffic Manager.
Configure the Gateway with multiple Traffic Managers, which are fronted by a load balancer as follows:

The Gateway publishes all Throttling events to the two Traffic Manager instances, and it fetches the throttle decisions from the Traffic Manager instances. Follow the instructions below to configure the API Gateway worker to communicate with the Traffic Managers and to push throttle events to both Traffic Manager instances.

a. Configure the receiver URL group `<ReceiverUrlGroup>` and Authentication URL Group `<AuthUrlGroup>` values, which are under the `<DataPublisher>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file, in order to contain all the Traffic Manager receiver URLs.

This is required when you have more than one Traffic Manager instance, and you are publishing to both as per the deployment pattern selected. As an example, if you are using two Traffic Manager instances and data should be published to both of them, the ReceiverUrlGroup and AuthUrlGroup should be configured as follows:

```
<ThrottlingConfigurations>
  <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
  <DataPublisher>
    <Enabled>true</Enabled>
    <Type>Binary</Type>
    <ReceiverUrlGroup>{tcp://[Traffic-Manager-1-host]:9611},
                  {tcp://[Traffic-Manager-2-host]:9611}</ReceiverUrlGroup>
    <!--ReceiverUrlGroup>tcp://${carbon.local.ip}:9612</ReceiverUrlGroup-->
    <AuthUrlGroup>{ssl://[Traffic-Manager-1-host]:9711},
                  {ssl://[Traffic-Manager-2-host]:9711}</AuthUrlGroup>
    <!--AuthUrlGroup>ssl://${carbon.local.ip}:9712</AuthUrlGroup-->
    <Username>${admin.username}</Username>
    <Password>${admin.password}</Password>
    <DataPublisherPool>
      <MaxIdle>1000</MaxIdle>
      <InitIdleCapacity>200</InitIdleCapacity>
    </DataPublisherPool>
    <DataPublisherThreadPool>
      <CorePoolSize>200</CorePoolSize>
      <MaxPoolSize>1000</MaxPoolSize>
      <KeepAliveTime>200</KeepAliveTime>
    </DataPublisherThreadPool>
  </DataPublisher>
  ...
</ThrottlingConfigurations>
```

[Traffic-Manager-1-host] and [Traffic-Manager-2-host] are the IPs/hostnames of two Traffic Manager nodes.

Based on the above configuration, the API Gateway publishes events to both the Traffic Managers.

b. Configure `<JMSConnectionParameters>` to connect to multiple brokers running within each Traffic Manager using fail over mechanism.
Distributed Deployment of the Gateway

This topic provides instructions on how to configure the multiple Gateways in WSO2 API Manager (WSO2 API-M) in a distributed deployment to facilitate high availability (HA). The instructions in this topic are based on the Distributed Deployment of API Manager and these configurations will only work if the configurations in that topic are done correctly.

Furthermore, these instructions use a shared file system as the content synchronization mechanism.

Why use a shared file system?

WSO2 recommends using a shared file system as the content synchronization mechanism to synchronize the artifacts among the WSO2 API-M Gateway nodes, because a shared file system does not require a specific node to act as a Gateway Manager, instead all the nodes have the worker manager capabilities. As a result, this helps to share all the APIs with any of the nodes; thereby, avoiding the vulnerability of a single point of failure. For this purpose you can use a common shared file system such as Network File System (NFS) or any other shared file system.

Follow the instructions below to configure the API-M Gateway in a distributed environment:

- Step 1 - Configure the load balancer
- Step 2 - Configure the Gateway
- Step 3 - Optionally, configure Hazelcast
- Step 4 - Start the Gateway Nodes

Note that the configurations in this topic are done based on the following pattern.
1. Configure the load balancer

For more information see, Configuring the Proxy Server and the Load Balancer.

Step 2 - Configure the Gateway

When using the shared file system, all nodes have the manager worker capability. Therefore, there is no need of having a separate manager node. Follow the instructions below to set up the Gateway nodes and enable it to work with the other components in the distributed setup.

Click here for information on configuring the Gateway.


   The following configurations are done in the `<API-M_GATEWAY_HOME>/repository/conf/carbon.xml` file.
   a. Open `<API-M_GATEWAY_HOME>/repository/conf/carbon.xml`.
   b. Locate the `<HostName>` tag and add the cluster hostname. For an example, if the hostname is `gw.am.wso2.com`:

      ```xml
      <HostName>gw.am.wso2.com</HostName>
      ```

   c. Locate the `<MgtHostName>` tag and uncomment it. Make sure that the management hostname is defined as follows:

      ```xml
      <MgtHostName>gw.am.wso2.com</MgtHostName>
      ```

2. Configure the catalina-server.xml file.

   Specify the following configurations in the `catalina-server.xml` file, which is located in the `<API-M_GATEWAY_HOME>/repository/catalina-server.xml` directory.
The TCP port number is the value that this Connector uses to create a server socket and waits for incoming connections. In your operating system, only one server application can listen to a particular port number on a particular IP address.

3. Map the hostnames to IPs.
   Open the server’s /etc/hosts file and add the following.

   ```
   <GATEWAY-IP> gw.am.wso2.com
   ```

   **Example Format**

   ```
   xxx.xxx.xxx.xx4 gw.am.wso2.com
   ```

   **Replicate the configurations** in all the other Gateway nodes.

4. Mount the directory required for the shared file system.
   Mount the `<API-M_HOME>/repository/deployment/server` directory of all the Gateway nodes to the shared file system to share all APIs between all the Gateway nodes.

**Step 3 - Optionally, configure Hazelcast**

You can seamlessly deploy WSO2 API Manager using local caching in a clustered setup without Hazelcast clustering. However, there are edge case scenarios where you need to enable Hazelcast clustering. For more information, see *Working with Hazelcast Clustering* to identify whether you need Hazelcast clustering and to configure it.

**Step 4 - Start the Gateway Nodes**

Start the WSO2 API-M Gateway node by typing the following command in the command prompt.

```
sh <API-M_GATEWAY_HOME>/bin/wso2server.sh
```

**What if I am unable to use a shared file system?**

If you are unable to have a shared file system, you can use remote synchronization (rsync) instead, but note that when using rsync there is the vulnerability of a single point of failure, because rsync needs one node to act as the Gateway Manager as it only provides write permission to one node. For more information, see *Configuring the Gateway in a Distributed Environment with rsync*.

**Why can’t I use SVN based deployment synchronization (Dep Sync)?**

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onward WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

**Configuring the Gateway in a Distributed Environment with rsync**

As the first preference WSO2 recommends using a shared file system over rsync as the content synchronization mechanism. For more information, see *Distributed Deployment of the Gateway*.

You need to use remote synchronization (rsync) only if you are unable to have a shared file system, because when using rsync it needs one node to act as
the Gateway Manager as it only provides write permission to one node. Thereby, when using rsync there is the vulnerability of a single point of failure.

Follow the instructions below to configure the API-M Gateway in a distributed environment when using rsync as a content synchronization mechanism:

- Step 1 - Configure the load balancer
- Step 2 - Configure the Gateway Manager
- Step 3 - Configure the Gateway Worker
- Step 4 - Optionally configure Hazelcast
- Step 5 - Start the Gateway Nodes

Note that the configurations in this topic are done based on the following pattern. This pattern is used as a basic Gateway cluster where the worker nodes and manager nodes are separated.

**NGINX Load Balancer**

Load Balancer host

**Gateway Cluster**

Gateway Manager host

Gateway Worker host

**Gateway Cluster**

Gateway Manager

Gateway Worker

Gateway Worker host

Gateway Worker host

Step 1 - Configure the load balancer

For more information, see Configuring the Proxy Server and the Load Balancer.

Step 2 - Configure the Gateway Manager

These nodes refer to the management nodes that specialize in the management of the setup. Only management nodes are authorized to add new artifacts into the system or make configuration changes. Management nodes are usually behind an internal firewall and are exposed to clients running within the organization only. This section involves setting up the Gateway node and enabling it to work with the other components in the distributed setup.

   The following configurations are done in the `<API-M_GATEWAY_MANAGER_HOME>/repository/conf/carbon.xml` file.
   a. Open `<API-M_GATEWAY_MANAGER_HOME>/repository/conf/carbon.xml` file on the management node.
   b. Locate the `<HostName>` tag and add the cluster hostname.

   ```xml
   <HostName>am.wso2.com</HostName>
   ```

   c. Locate the `<MgtHostName>` tag and uncomment it. Make sure that the management hostname is defined as follows:

   ```xml
   <MgtHostName>mgt.am.wso2.com</MgtHostName>
   ```

2. Configure the catalina-server.xml file.
   Specify the following configurations in the `<API-M_GATEWAY_MANAGER_HOME>/repository/conf/tomcat/catalina-server.xml` file.
The TCP port number is the value that this Connector uses to create a server socket and await incoming connections. The operating system will allow only one server application to listen to a particular port number on a particular IP address.

3. Map the hostnames to IPs.
   Open the server's /etc/hosts file and add the following.

   ```
   <GATEWAY-MANAGER-IP> mgt.am.wso2.com
   ```

   Example Format

   ```
   xxx.xxx.xxx.xx4 am.wso2.com
   ```

   Once you replicate these configurations for all the manager nodes, your Gateway manager is configured.

Step 3 - Configure the Gateway Worker

Worker nodes specialize in serving requests to deployment artifacts and reading them. They can be exposed to external clients.

   a. Open `<GATEWAY_WORKER_HOME>/repository/conf/carbon.xml` file on each worker node.
   b. Specify the hostname as follows:

   ```
   <HostName>am.wso2.com</HostName>
   ```

2. Configure the catalina-server.xml file.
   Make the following configuration changes in the `<API-M_GATEWAY_WORKER_HOME>/repository/conf/tomcat/catalina-server.xml` file.

   ```
             port="9763"
             proxyPort="80"
             />
             port="9443"
             proxyPort="443"
             />
   ```

3. Map the hostnames to IPs.
   Open the server's /etc/hosts file and add the following in order to map the hostnames with the specified real IPs.

   ```
   <GATEWAY-MANAGER-IP> mgt.am.wso2.com
   ```

4. Configure rsync.
   For information on configuring rsync, see Configuring rsync for Deployment Synchronization.
Step 4 - Optionally configure Hazelcast

You can seamlessly deploy WSO2 API Manager using local caching in a clustered setup without Hazelcast clustering. However, there are edge case scenarios where you need to enable Hazelcast clustering. For more information, see Working with Hazelcast Clustering to identify whether you need Hazelcast clustering and to configure it.

Step 5 - Start the Gateway Nodes

Start the Gateway Manager and then the Gateway Worker nodes.

Click here for information on starting the Gateway nodes.

Step 5.1 - Start the Gateway Manager

Start the Gateway Manager by typing the following command in the command prompt.

```
sh <API-M_GATEWAY_MANAGER_HOME>/bin/wso2server.sh
```

Step 5.2 - Start the Gateway Worker

Start the Gateway Worker by typing the following command in the command prompt.

```
sh <GATEWAY_WORKER_HOME>/bin/wso2server.sh -Dprofile=gateway-worker
```

The additional `-Dprofile=gateway-worker` argument indicates that this is a worker node specific to the Gateway. You need to use this parameter to make a server read-only. Changes (i.e., writing or making modifications to the deployment repository, etc.) cannot be made in the Gateway worker nodes. Furthermore, starting a node as a Gateway worker ensures that the Store and Publisher related functions are disabled in the respective node. This parameter also ensures that the node starts in the worker profile, where the UI bundles are not activated and only the backend bundles are activated when the server starts up. For more information on the various product profiles available in WSO2 API Manager, see API Manager product profiles.

Working with Hazelcast Clustering

Hazelcast is an in-memory data grid, which is used in clustering and distributed shared memory. When using Hazelcast as a clustering implementation, data is evenly distributed among the nodes in a cluster. Hazelcast clustering is disabled in WSO2 API Manager (WSO2 APIM) by default, because you can successfully deploy WSO2 API-M without Hazelcast. The following subsections explain as to when you need Hazelcast clustering with WSO2 API Manager (WSO2 APIM) and how you can configure it.

When to use Hazelcast

When you can seamlessly deploy WSO2 API Manager using local caching in a clustered setup without Hazelcast clustering. However, the following are the edge case scenarios where you need to enable Hazelcast clustering.

- **Immediate revocation of tokens among the gateways**
  If you deploy WSO2 API-M without Hazelcast clustering, you can set the time-to-live (TTL) value in secs in order to define the period after which the token will automatically be revoked in the other gateways. However, if you want the token in all the gateways to be immediately revoked, then you need to enable Hazelcast clustering.

- **Backend service throttling**
  The endpoint throttling limits and the spike arrest throttling limits will not be shared across the cluster when Hazelcast clustering is disabled.

Deploying WSO2 API-M Manager with Hazelcast clustering

Follow the instructions below to deploy WSO2 API-M with Hazelcast clustering:

1. Carryout the instructions related to deploying WSO2 API-M.
2. Carryout the instructions to enable Hazelcast clustering.
3. Start the WSO2 API-M servers.

Enabling Hazelcast clustering

Follow the instructions below to enable Hazelcast clustering when deploying WSO2 API-M:

- **Step 1 - Enable Hazelcast clustering in the Gateway Manager**
- **Step 2 - Enable Hazelcast clustering in the Gateway Worker**
Step 1 - Enable Hazelcast clustering in the Gateway Manager

1. Open the `<GATEWAY_MANAGER_HOME>/repository/conf/axis2/axis2.xml` file.
2. Locate the clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default).
   a. Enable clustering for this node:

   ```
   <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true"/>
   ```

   b. Set the membership scheme to `wka` to enable the well-known address registration method (this node will send cluster initiation messages to WKA members that we will define later):

   ```
   <parameter name="membershipScheme">wka</parameter>
   ```

c. Specify the name of the cluster this node will join:

   ```
   <parameter name="domain">wso2.am.domain</parameter>
   ```

d. Specify the host used to communicate cluster messages. This is the IP of the Gateway manager node.

   ```
   <parameter name="localMemberHost">xxx.xxx.xxx.xx3</parameter>
   ```

e. Specify the port used to communicate cluster messages:

   ```
   <parameter name="localMemberPort">4500</parameter>
   ```

   This port number will not be affected by the port offset in `carbon.xml`. If this port number is already assigned to another server, the clustering framework will automatically increment this port number. However, if two servers are running on the same machine, you must ensure that a unique port is set for each server.

   f. Do the following port mapping configurations for the Gateway manager node. There are two types of transports in API Manager and when a request comes into the API Manager, it always goes to the default transport which is the PTT/NIO transport. So when you access the management console of the Gateway Manager node, you send a servlet request. If you do not specify the port mapping parameter in the manager node, it would hit the PTT/NIO transport and the request would fail.

   ```
   <parameter name="properties">
     <property name="backendServerURL" value="https://${hostName}:${httpsPort}/services/">
     <property name="mgtConsoleURL" value="https://${hostName}:${httpsPort}/">
     <property name="subDomain" value="mgt"/>
     <property name="port.mapping.80" value="9763"/>
     <property name="port.mapping.443" value="9443"/>
   </parameter>
   ```

   The receiver's HTTP/HTTPS port values are without the `portOffset` addition; they get auto-incremented by `portOffset`. In the case of an ESB cluster, the `WSDLEPRPrefix` parameter should point to the worker node's host name (am.wso2.com) and load balancer's http (80)/https (443) transport ports.

g. Change the members listed in the `<members>` element. This defines the WKA members.
Here we configure the manager node and worker node as the well-known members.

Step 2 - Enable Hazelcast clustering in the Gateway Worker

1. Open the `<GATEWAY_WORKER_HOME>/repository/conf/axis2/axis2.xml` file.
2. Locate the clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default):
   a. Enable clustering for this node:
      ```
      <clustering class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true"/>
      ```
   b. Set the membership scheme to `wka` to enable the Well Known Address registration method (this node will send cluster initiation messages to WKA members that we will define later):
      ```
      <parameter name="membershipScheme">wka</parameter>
      ```
   c. Specify the name of the cluster this node will join:
      ```
      <parameter name="domain">wso2.am.domain</parameter>
      ```
   d. Specify the host used to communicate cluster messages. This is the IP address of the Gateway worker.
      ```
      <parameter name="localMemberHost">xxx.xxx.xxx.xx4</parameter>
      ```
   e. Specify the port used to communicate cluster messages (if this node is on the same server as the manager node, or another worker node, be sure to set this to a unique value, such as 4000 and 4001 for worker nodes 1 and 2).
      ```
      <parameter name="localMemberPort">4200</parameter>
      ```
   f. Define the sub-domain as worker by adding the following property under the `<properties>` element:
      ```
      <property name="subDomain" value="worker"/>
      ```
   g. Define the manager and worker nodes as well-known members of the cluster by providing their host name and `localMemberPort` values. The manager node is defined here because it is required for the Deployment Synchronizer to function in an efficient manner. The deployment synchronizer uses this configuration to identify the manager and synchronize deployment artifacts across the nodes of a cluster.
it is recommended to add at least two well-known members here. This is done to ensure that there is high availability for the cluster.

You can also use IP address ranges for the `hostName`. For example, 192.168.1.2-10. This should ensure that the cluster eventually recovers after failures. One shortcoming of doing this is that you can define a range only for the last portion of the IP address. You should also keep in mind that the smaller the range, the faster the time it takes to discover members, since each node has to scan a lesser number of potential members.

h. See the instructions on configuring hazelcast properties given below.

Configuring Hazelcast properties

You can configure the hazelcast properties for the product nodes by following the steps given below.

1. Create the `hazelcast.properties` file with the following property configurations, and copy the file to the `<GATEWAY_HOME>/repository/conf/` directory.

    ```
    # Disabling the hazelcast shutdown hook
    hazelcast.shutdownhook.enabled=false
    # Setting the hazelcast logging type to log4j
    hazelcast.logging.type=log4j
    ```

    The above configurations are explained below.
    - **Hazelcast shutdown hook**: This configuration disables the shutdown hook in hazelcast, which ensures that the hazelcast instance shuts down gracefully whenever the product node shuts down. If the hazelcast shutdown hook is enabled (which is the default behavior of a product), you will see errors such as “Hazelcast instance is not active” at the time of shutting down the product node. This is because the hazelcast instance shuts down too early when the shutdown hook is enabled.
    - **Hazelcast logging type**: This configuration sets the hazelcast logging type to log4j, which allows hazelcast logs to be written to the `wso2carbon.log` file.

2. If you have enabled log4j for hazelcast logging as shown above, be sure to enter the configuration shown below in the `log4j.properties` file (stored in the `<GATEWAY_HOME>/repository/conf/` directory). This can be used to configure the log level for hazelcast logging. For a clustered production environment, it is recommended to use INFO as the log level as shown below.

    ```
    log4j.logger.com.hazelcast=INFO
    ```

Configuring the Proxy Server and the Load Balancer

A load balancer or reverse proxy is required to map external traffic with ports and URLs that WSO2 API Manager (WSO2 API-M) uses internally. Follow the instructions below to configure load balancing together with reverse proxying.

- Step 1 - Create a SSL certificate for the load balancer
- Step 2 - Configure the load balancer/reverse proxy server
- Step 3 - Configure the reverse proxy settings in the product

Step 1 - Create a SSL certificate for the load balancer

This step is only applicable for a High Availability (HA) setup where multiple nodes are fronted by a load balancer.

Create a SSL certificate for the load balancer using the following instructions.

1. Create the Server Key.
sudo openssl genrsa -des3 -out <key_name>.key 1024

2. Submit the certificate signing request (CSR).

sudo openssl req -new -key <key_name>.key -out server.csr

3. Remove the password.

sudo cp <key_name>.key <key_name>.key.org
sudo openssl rsa -in <key_name>.key.org -out <key_name>.key

4. Sign your SSL Certificate.

sudo openssl x509 -req -days 365 -in server.csr -signkey <key_name>.key -out <certificate_name>.crt

5. Copy the key and certificate files that you generated in the above step to the /etc/nginx/ssl/ location.

Step 2 - Configure the load balancer/reverse proxy server

In the following instructions, you are instructed to use NGINX to handle the load balancing requirements.

Although the following section instructs you to use NGINX as the load balancer, you can use any load balancer in your deployment based on your preference.

What is NGINX?

NGINX is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server. For more information, see https://www.nginx.com/.

Carry out the following steps to configure the load balancer to front multiple nodes.

1. Install NGINX in a server configured in your cluster.

<table>
<thead>
<tr>
<th>Deployment</th>
<th>API-M Nodes</th>
<th>LB</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single all-in-one deployment</td>
<td>N/A</td>
<td>NGINX Community</td>
<td>This deployment does not need Sticky Sessions (Session Affinity).</td>
</tr>
<tr>
<td>Active-active deployment using single all-in-one nodes</td>
<td>N/A</td>
<td>NGINX Plus</td>
<td>This deployment requires Sticky Sessions, but NGINX Community version does not support it. You can use ip_hash as the sticky algorithm.</td>
</tr>
<tr>
<td>Distributed deployment</td>
<td>Gateway with a single Gateway Manager</td>
<td>NGINX Community version</td>
<td>The Gateway node in this deployment does not need Sticky Sessions.</td>
</tr>
<tr>
<td></td>
<td>Gateway with multiple Gateway Managers</td>
<td>NGINX Plus</td>
<td>The Gateway Manager nodes require Sticky Sessions, but NGINX Community version does not support it. You can use ip_hash as the sticky algorithm. Sticky Sessions are needed for port 9443 in the Gateway, and not needed for the pass through ports in the Gateway (8243, 8280).</td>
</tr>
</tbody>
</table>
For more information on installing NGINX, see NGINX community version and NGINX Plus.

2. Configure NGINX to direct the HTTP and HTTPs requests based on your deployment.
   a. Run the following command to identify the exact location of the `<NGINX_HOME>` directory. Inspect the output and identify the --prefix tag as it provides the location of the `<NGINX_HOME>` directory.

   ```bash
   nginx -V
   ```

   b. Update the `nginx.conf` file with the required NGINX configuration given below. If not, you can create a file with the `.conf` suffix and copy it to the `<NGINX_HOME>/conf.d` directory.

   ```
   • All ports are default ports assuming no port offsets are used.
   • The key and the certificate for SSL is assumed to be at the `<NGINX_HOME>/ssl` location. The placeholders `{cert_name}` and `{key_name}` are the name of the certificate and key generated.
   • The directories used for access and error logs should be created if they do not exist.
   ```

   Single node deployment
   Active-Active Deployment
   HA for Gateway
   HA for Publisher
   HA for Store
   HA for Key Manager

   ```
   • The placeholder `{node-ip-address}` corresponds to the IP address of the backend node in which the WSO2 API-M server is running.
   • In the sample configuration given below, the hostname `api.am.wso2.com` is used to access all portals (publisher, store, admin, and carbon) and `gw.am.wso2.com` is used to invoke APIs. Only HTTPS is allowed.
   ```
The placeholders `{node-1-ip-address}` and `{node-2-ip-address}` correspond to the IP addresses of the backend nodes in which APIM servers are running.

In the sample configuration given below, the hostname `api.am.wso2.com` is used to access all portals (publisher, store, admin and carbon) and `gw.am.wso2.com` is used to invoke APIs. Only HTTPS is allowed.

This configuration uses a session cookie to configure stickiness. However, if you are using Nginx community version, configuring sticky sessions based on session cookie is not supported. It is possible to use the `ip_hash` method instead.
In an Active-Active deployment, it is **mandatory** to set up sticky sessions (session affinity) in the load balancers that front the **Publisher** and **Store**, and it is **optional** in the load balancer (if any) that fronts **Key Manager** or **Gateway**.

However, authentication via session ID fails when sticky sessions are disabled in the load balancers of Publisher and store.
The placeholder (gwm-ip-address) corresponds to the IP addresses of the backend nodes in which Gateway Manager server is running. Similarly, (gw-1-ip-address) and (gw-2-ip-address) are the nodes in which Gateway Workers are running.

In the sample configuration given below, the hostname mgtgw.am.wso2.com is used to access management console of the Gateway.
Manager and gw.am.wso2.com is used to invoke APIs. Only HTTPS is allowed.

- If you are using multiple Gateway Managers when using a shared file system (e.g., NFS), then you need to enable sticky sessions.

```
upstream mgtgw.am.wso2.com {
    server (gwm-ip-address):9443;
}

upstream sslgw.am.wso2.com {
    server (gw-1-ip-address):8243;
    server (gw-2-ip-address):8243;
}

test { 
    listen 80;
    server_name mgtgw.am.wso2.com;
    rewrite ^/(.*) https://mgtgw.am.wso2.com/$1 permanent;
}

test { 
    listen 443;
    server_name mgtgw.am.wso2.com;
    proxy_set_header X-Forwarded-Port 443;
    ssl on;
    ssl_certificate /etc/nginx/ssl/{cert_name};
    ssl_certificate_key /etc/nginx/ssl/{key_name};
    location / { 
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_read_timeout 5m;
        proxy_send_timeout 5m;
        proxy_pass https://mgtgw.am.wso2.com;
    }

    access_log /etc/nginx/log/gwm/https/access.log;
    error_log /etc/nginx/log/gwm/https/error.log;
}

test { 
    listen 443;
    server_name gw.am.wso2.com;
    proxy_set_header X-Forwarded-Port 443;
    ssl on;
    ssl_certificate /etc/nginx/ssl/{cert_name};
    ssl_certificate_key /etc/nginx/ssl/{key_name};
    location / { 
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_read_timeout 5m;
        proxy_send_timeout 5m;
        proxy_pass https://sslgw.am.wso2.com;
    }

    access_log /etc/nginx/log/gw/https/access.log;
    error_log /etc/nginx/log/gw/https/error.log;
}
```

- The placeholders {publisher-1-ip-address} and {publisher-2-ip-address} correspond to the IP addresses of the backend nodes in which APIM Publishers are running.
- In the sample configuration given below, the hostname publisher.am.wso2.com is used to access publisher portal. Only HTTPS is allowed.
- This configuration uses a session cookie to configure stickiness. However, if you are using Nginx community version, configuring sticky
The placeholders `{store-1-ip-address}` and `{store-2-ip-address}` correspond to the IP addresses of the backend nodes in which APIM Stores are running.

In the sample configuration given below, the hostname `store.am.wso2.com` is used to access Publisher portal. Only HTTPS is allowed.

This configuration uses a session cookie to configure stickiness. However, if you are using Nginx community version, configuring sticky sessions based on session cookie is not supported. It is possible to use `ip_hash` method instead.
upstream store.am.wso2.com {
  server {store-1-ip-address}:9443;
  server {store-2-ip-address}:9443;
  #ip_hash;
  sticky learn create=$upstream_cookie_jsessionid
    lookup=$cookie_jsessionid
    zone=client_sessions:1m;
}

server {
  listen 80;
  server_name store.am.wso2.com;
  rewrite ^/(.*) https://store.am.wso2.com/$1 permanent;
}

server {
  listen 443;
  server_name store.am.wso2.com;
  proxy_set_header X-Forwarded-Port 443;
  ssl on;
  ssl_certificate /etc/nginx/ssl/{cert_name};
  ssl_certificate_key /etc/nginx/ssl/{key_name};
  location / {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_read_timeout 5m;
    proxy_send_timeout 5m;
    proxy_pass https://store.am.wso2.com;
  }

  access_log /etc/nginx/log/store/https/access.log;
  error_log /etc/nginx/logs/store/https/error.log;
}

- The placeholders {km-1-ip-address} and {km-2-ip-address} correspond to the IP addresses of the backend nodes in which APIM Key Managers are running.
- In the sample configuration given below, the hostname km.am.wso2.com is used to access Key Manager. Only HTTPS is allowed.
- This configuration uses a session cookie to configure stickiness. However, if you are using Nginx community version, configuring sticky sessions based on session cookie is not supported. It is possible to use ip_hash method instead.
upstream km.am.wso2.com {
    server {km-1-ip-address}:9443;
    server {km-2-ip-address}:9443;
    #ip_hash;
    sticky_learn create=$upstream_cookie_jsessionid
    lookup=$cookie_jsessionid
    zone=client_sessions:1m;
}

server {
    listen 80;
    server_name km.am.wso2.com;
    rewrite ^/(.*) https://km.am.wso2.com/$1 permanent;
}

server {
    listen 443;
    server_name km.am.wso2.com;
    proxy_set_header X-Forwarded-Port 443;
    ssl on;
    ssl_certificate /etc/nginx/ssl/{cert_name};
    ssl_certificate_key /etc/nginx/ssl/{key_name};
    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_read_timeout 5m;
        proxy_send_timeout 5m;
        proxy_pass https://km.am.wso2.com;
    }

    access_log /etc/nginx/log/km/https/access.log;
    error_log /etc/nginx/log/km/https/error.log;
}

The ports and URLs that are used internally by API Manager are given below:

<table>
<thead>
<tr>
<th>Usage</th>
<th>URL</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS Servlet (UI Consoles)</td>
<td>localhost</td>
<td>9443</td>
</tr>
<tr>
<td>NIO transport (HTTP API Traffic)</td>
<td>localhost</td>
<td>8280</td>
</tr>
<tr>
<td>NIO transport (HTTPS API Traffic)</td>
<td>localhost</td>
<td>8243</td>
</tr>
</tbody>
</table>

Restart the NGINX server:

```
sudo service nginx restart
```

You do not need to restart the server if you are simply making a modification to the VHost file. The following command is sufficient in such cases.

```
sudo service nginx reload
```
Step 3 - Configure the reverse proxy settings in the product

- This step is only relevant to the Store and Publisher nodes when fronting multiple nodes with a load balancer.
- These configurations need to be added in all the Publisher and Store nodes.

When using a load balancer, you need to configure Reverse Proxy for the API Store URL, API Publisher URL, Admin Portal URL, and whatever other URLs that you are working with for the portals to be able to work with the Proxy Server configuration.

1. Update the API Store node.
   Update the following configuration in the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.js` file as shown below, to configure reverse proxying for the API Store.

   ```
   "reverseProxy" : {
     "enabled" : true, // values true, false, "auto" - will look for X-Forwarded-* headers
     "host" : "<hostname>", // If reverse proxy do not have a domain name use IP
     "context":"/store",
     "//regContext":"" // Use only if different path is used for registry
   }
   ```

2. Update the API Publisher node.
   a. Configure reverse proxying for the API Publisher.
      Update the following configuration in the `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` file as shown below:

      ```
      "reverseProxy" : {
        "enabled" : true, // values true, false, "auto" - will look for X-Forwarded-* headers
        "host" : "<hostname>", // If reverse proxy do not have a domain name use IP
        "context":"/publisher",
        "//regContext":"
      }
      ```

   b. Configure reverse proxying for the Admin Portal.
      Update the following configuration in the `<API-M_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.js` file as shown below.

      ```
      "reverseProxy" : {
        "enabled" : true, // values true, false, "auto" - will look for X-Forwarded-* headers
        "host" : "<hostname>", // If reverse proxy do not have a domain name use IP
        "context":"/admin",
        "//regContext":"
      },
      ```

Installing and Configuring the Databases

The following steps describe how to download and install a RDBMS, which in this case is a MySQL Server, create the databases, configure the data sources, and configure the API Manager components to connect to them.

Although the following section instructs you to use MySQL Server, you can use any RDBMS in your deployment based on your preference. For information on working with other databases, see Changing the Default API-M Databases.

The steps involved in installing and configuring the databases are the same irrespective of whether you are using a single node (standalone) deployment, an active-active deployment, or a distributed deployment.

1. Unzip the WSO2 API Manager pack. Let's call it `<API-M_HOME>`.
2. Download and install MySQL Server.
3. Download the MySQL JDBC driver.
4. Unzip the downloaded MySQL driver archive, and copy the MySQL JDBC driver JAR (mysql-connector-java-x.x.xx-bin.jar) into the `<API-M_HOME>/repository/components/lib` directory in all the nodes in the cluster.
5. Define the hostname for configuring permissions for the new database by opening the /etc/hosts file and adding the following:

Do this step only if your database is not on your local machine and on a separate server.

```
<MYSQL-DB-SERVER-IP> carbondb.mysql-wso2.com
```

6. Install mysql-client in each of the API-M servers in which WSO2 API-M is deployed. You need to do this in order to check if the servers can access the MySQL database.

```
sudo apt install mysql-client
mysql -h <mysqldb_host_ip> -u username -p
```

7. Enter the following command in a command prompt, where `username` is the username that you used to access the databases.

```
mysql -u username -p
```

When prompted, specify the password that will be used to access the databases with the username you specified.

8. Create the databases using the following commands, where `</API-M_HOME>` is the path to any of the API Manager instances you installed, and `username` and `password` are the same as those you specified in the previous steps.

```
WSO2 API Manager is shipped with an H2 database. This embedded H2 database is suitable for development and testing environments. However, for production environments, it is recommended to use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. The following steps explain how to create MySQL DBs.

**About using MySQL in different operating systems**

For users of Microsoft Windows, when creating the database in MySQL, it is important to specify the character set as latin1. Failure to do this may result in an error (error code: 1709) when starting your cluster. This error occurs in certain versions of MySQL (5.6.x) and is related to the UTF-8 encoding. MySQL originally used the latin1 character set by default, which stored characters in a 2-byte sequence. However, in recent versions, MySQL defaults to UTF-8 to be friendlier to international users. Hence, you must use latin1 as the character set as indicated below in the database creation commands to avoid this problem. Note that this may result in issues with non-latin characters (like Hebrew, Japanese, etc.). The following is how your database creation command should look.

```
mysql> create database <DATABASE_NAME> character set latin1;
```

For users of other operating systems, the standard database creation commands are sufficient. For these operating systems, the following is how your database creation command should look.

```
mysql> create database <DATABASE_NAME>;
```

If you are using MySQL to configure your datasources, we recommend that you use a case sensitive database collation. For more information, see the MySQL Official Manual. The default database collation, which is `latin1_swedish_ci`, is case insensitive. However, you need to maintain case sensitivity for database collation, because when the database or table has a case-insensitive collation in MySQL 5.6 or 5.7, if a user creates an API with letters using mixed case, deletes the API, and then creates another API with the same name, but in lower case letters, then the later created API loses its permission information, because when deleting the API, it keeps the Registry collection left behind.

This issue could be avoided if you use a case sensitive collation for database and tables. In that case, when creating the second API (which has the same name, but is entirely in lowercase letters), it will create a new record with the lowercase name in the `UM_PERMISSION` table.

**Additional notes**

- Ensure that MySQL is configured so that all nodes can connect to it.
- From WSO2 API Manager 2.0.0 onwards there are two MySQL DB scripts available in the product distribution. Click here to identify as to which version of the MySQL script to use.
- Table creation of the statistics database is handled by the Analytics scripts when you configure APIM Analytics, so you will
create the statistics database in this step but will not specify a source script.

- To access the databases from remote instances, its required to grant permission to the relevant username defined in the <API-M_HOME>/repository/conf/datasources/master-datasources.xml file, by using the grant command. See the following sample commands.

### Format Sample

```sql
mysql> create database apimgtdb;
mysql> use apimgtdb;
mysql> source <API-M_HOME>/dbscripts/apimgt/mysql.sql;
mysql> grant all on apimgtdb.* TO '<username>'@'%' identified by '<password>';

mysql> create database userdb;
mysql> use userdb;
mysql> source <API-M_HOME>/dbscripts/mysql.sql;
mysql> grant all on userdb.* TO '<username>'@'%' identified by '<password>';

mysql> create database regdb;
mysql> use regdb;
mysql> source <API-M_HOME>/dbscripts/mysql.sql;
mysql> grant all on regdb.* TO '<username>'@'%' identified by '<password>';

mysql> create database mbstoredb;
mysql> use mbstoredb;
mysql> source <API-M_HOME>/dbscripts/mb-store/mysql-mb.sql;
mysql> grant all on mbstoredb.* TO '<username>'@'%' identified by '<password>';
```

```sql
mysql> create database apimgtdb;
mysql> use apimgtdb;
mysql> source <API-M_HOME>/dbscripts/apimgt/mysql.sql;
mysql> grant all on apimgtdb.* TO 'wso2user'@'%' identified by 'wso2123';

mysql> create database userdb;
mysql> use userdb;
mysql> source <API-M_HOME>/dbscripts/mysql.sql;
mysql> grant all on userdb.* TO 'wso2user'@'%' identified by 'wso2123';

mysql> create database regdb;
mysql> use regdb;
mysql> source <API-M_HOME>/dbscripts/mysql.sql;
mysql> grant all on regdb.* TO 'wso2user'@'%' identified by 'wso2123';

mysql> create database mbstoredb;
mysql> use mbstoredb;
mysql> source <API-M_HOME>/dbscripts/mb-store/mysql-mb.sql;
mysql> grant all on mbstoredb.* TO 'wso2user'@'%' identified by 'wso2123';
```

In the sample commands above, its assumed that the username and password defined in the datasource configurations in <API-M_HOME>/repository/conf/datasources/master-datasources.xml file is wso2user and wso2123 respectively.

### In the sample commands above, its assumed that the username and password defined in the datasource configurations in <API-M_HOME>/repository/conf/datasources/master-datasources.xml file is wso2user and wso2123 respectively.

10. Configure the data sources for the five databases as follows:
    a. Open the <API-M_HOME>/repository/conf/datasources/master-datasources.xml file.
       This file contains the different datasources used by WSO2 API Manager. By default, the API Manager connects to the local H2 database and it is recommended to use a separate RDBMS server for a production deployment.
       - If you are configuring API-M in a single node, open the master-datasources.xml in the single WSO2 API-M instance.
       - If you are configuring API-M in a distributed setup, open the master-datasources.xml in all five WSO2 API-M components.

For more information, see Configuring master-datasources.xml in the Administration Guide.

**Note:** When configuring clustering, ignore the WSO2_CARBON_DB data source configuration.
b. Enable the components to access the WSO2 API Manager database by modifying the `WSO2AM_DB` data source in the `master-datasources.xml` file by changing the URL as indicated below. Make sure to also replace `db.mysql-wso2.com` with the hostname you specified in step 5 (carbondb.mysql-wso2.com).

- If you are configuring API-M in a single node, open the `master-datasources.xml` in the single WSO2 API-M instance.
- If you are configuring API-M in a distributed setup, open the `master-datasources.xml` in the Publisher, Store, and Key Manager nodes.

### Notes

- Although the Gateway does not use the WSO2 API Manager database, this connection is required; therefore, do not remove the default configuration in the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file. This connection should be your default database.
- The Gateway node creates a connection at the startup with the WSO2 API Manager database, but this connection will not be used later on.
- If you have more than one Traffic Manager node, each Traffic Manager node must have its own Message Broker database (WSO2_MDB_STORE_DB).

```xml
<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for the API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://db.mysql-wso2.com:3306/apimgt?autoReconnect=true</url>
      <username>user</username>
      <password>password</password>
      <defaultAutoCommit>false</defaultAutoCommit>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

c. Enable the Key Manager, Publisher, and Store components to access the user management database. You need to do this by adding the following code in the `master-datasources.xml` file and changing `db.mysql-wso2.com` to `carbondb.mysql-wso2.com` in order to configure the `WSO2UM_DB` data source.

- If you are configuring API-M in a single node, open the `master-datasources.xml` in the single WSO2 API-M instance.
- If you are configuring API-M in a distributed setup, open the `master-datasources.xml` in the Publisher, Store, and Key Manager nodes.
d. Enable access to registry databases by adding the `WSO2REG_DB` data sources related configuration in their `master-datasources.xml` files as follows. The components that need to access the registry database differs, based on whether the setup is multi-tenanted or not.

Single Tenant Setup

Multi Tenanted Setup

The configuration needs to be added to API Publisher and Store components.

The configuration needs to be added to API Publisher, Store, Key Manager, and Gateway components.

e. Enable the Traffic Manager component to access the Message Broker database by configuring the `WSO2_MB_STORE_DB` data source, which is in its `master-datasources.xml` file as follows:

- Do not share the `WSO2_MB_STORE_DB` database among the nodes in an Active-Active set-up or Traffic Manager HA scenario, because each node should have its own local `WSO2_MB_STORE_DB` database to act as separate Traffic Managers.
- The latter mentioned DBs can be either H2 DBs or any RDBMS such as MySQL.
- If the database gets corrupted then you need to replace the database with a fresh database that is available in the product distribution.
11. To give the Key Manager, Publisher, and Store components access to the user management database with shared permissions, open the `<API-M_HOME>/repository/conf/user-mgt.xml` file in each of these three components and add or modify the `dataSource` property that corresponds to the `configuration` element as follows:

For more information, see Configuring User Stores.

```xml
<configuration>
  ...
  <Property name="dataSource">jdbc/WSO2UM_DB</Property>
</configuration>

<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
  <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
  <Property name="ReadOnly">false</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="IsEmailUserName">false</Property>
  <Property name="DomainCalculation">default</Property>
  <Property name="PasswordDigest">SHA-256</Property>
  <Property name="StoreSaltedPassword">true</Property>
  <Property name="ReadGroups">true</Property>
  <Property name="WriteGroups">true</Property>
  <Property name="UserNameUniqueAcrossTenants">false</Property>
  <Property name="PasswordJavaRegEx"^^\S\{5,30}\$</Property>
  <Property name="UsernameJavaRegEx"^^\S\{5,30}\$</Property>
  <Property name="RolenameJavaRegEx"^^\S\{5,30}\$</Property>
  <Property name="UserRolesCacheEnabled">true</Property>
  <Property name="MaxRoleNameListLength">100</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="SharedGroupEnabled">false</Property>
  <Property name="SCIMEnabled">false</Property>
</UserStoreManager>
```

If you are using the WSO2UM_DB to store users, remember to change the administrator's username and password. For more information, see Maintaining Logins and Passwords.

12. To enable access to the registry database, open the `<API-M_HOME>/repository/conf/registry.xml` file in each of these components and configure them as follows. The components that need to mount the governance registry space differs based on whether the setup is multi-tennanted or not.

* Single Tenant Setup
* Multi Tenanted Setup

```xml
<datasource>
  <name>WSO2_MB_STORE_DB</name>
  <description>The datasource used for message broker database</description>
  <jndiConfig>
    <name>WSO2MBStoreDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://db.mysql-wso2.com:3306/mbstoredb?autoReconnect=true</url>
      <username>user</username>
      <password>password</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```
**Configuration needs to be added to API Publisher and Store components.**

**Configuration needs to be added to API Publisher, Store, Key Manager, and Gateway components.**

Do not replace the following configuration when adding in the mounting configurations mentioned below. The registry mounting configurations mentioned in the following steps must be added beneath the following entry, which is already in the configuration file.

```xml
<dbConfig name="wso2registry">
  <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>
```

This configuration points to the local H2 database. This configuration is necessary and must always exist in this file.

a. In the Publisher component's `registry.xml` file, add or modify the `dataSource` attribute of the `<dbConfig name="govregistry"` element as follows:

```xml
<dbConfig name="govregistry">  
  <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>
<remoteInstance url="https://localhost:9443/registry">
  <id>gov</id>
  <cacheId>user@jdbc:mysql://db.mysql-wso2.com:3306/regdb</cacheId>
  <dbConfig>govregistry</dbConfig>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>
<mount path="/_system/governance" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/_system/governance</targetPath>
</mount>
<mount path="/_system/config" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/_system/config</targetPath>
</mount>
```

b. In the Store component's `registry.xml` file, add or modify the `dataSource` attribute of the `<dbConfig name="govregistry"` element as follows (note that this configuration is almost identical to the previous step except for the `remoteInstance` URL):

```xml
<dbConfig name="govregistry">  
  <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>
<remoteInstance url="https://localhost:9443/registry">
  <id>gov</id>
  <cacheId>user@jdbc:mysql://db.mysql-wso2.com:3306/regdb</cacheId>
  <dbConfig>govregistry</dbConfig>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>
<mount path="/_system/governance" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/_system/governance</targetPath>
</mount>
<mount path="/_system/config" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/_system/config</targetPath>
</mount>
```

CacheId is a unique identification of remote instance. When you configure the remote instance, WSO2 recommends that you modify the `<cacheId>` with the corresponding values of your setup, based on the following format: `<username>@<JDBC_URL to_registry_database>`
13. Skip caching.

Uncomment the following configuration under the `<indexingConfiguration>` element in the `<API-M_HOME>/repository/conf/registry.xml` file.

```
<skipCache>true</skipCache>
```

Configuring rsync for Deployment Synchronization

Deployment synchronization can be done using rsync, which is a file copying tool. These changes must be done in the manager node and in the same `rsync` directory.

1. Create a file named `workers-list.txt`, somewhere in your machine, that lists all the worker nodes in the deployment. The following is a sample of the file where there are two worker nodes.

```
Different nodes are separated by new lines.

workers-list.txt

ubuntu@192.168.1.1:~/setup/192.168.1.1/as/as_worker/repository/deployment/server
ubuntu@192.168.1.2:~/setup/192.168.1.2/as/as_worker/repository/deployment/server
```

If you have configured tenants in worker nodes, you need to add the `repository/tenants` directory of the worker node to the list to synchronize tenant space. For example, if the node `ubuntu@192.168.1.1` needs to be synced with both the super tenant and the `ubuntu@192.168.1.1` tenant space, the following two entries should be added to the `workers-list.txt` file.

```
Different nodes are separated by new lines.

workers-list.txt

ubuntu@192.168.1.1:~/setup/192.168.1.1/apim/apim_worker/repository/deployment/server
ubuntu@192.168.1.1:~/setup/192.168.1.1/apim/apim_worker/repository/tenants
```

2. Create a file to synchronize the `<API-M_HOME>/repository/deployment/server` folders between the manager and all worker nodes.

```
You must create your own SSH key and define it as the `pem_file`. Alternatively, you can use an existing SSH key. For information on generating and using the SSH Keys, go to the SSH documentation. Specify the `manager_server_dir` depending on the location in your local machine. Change the `logs.txt` file path and the lock location based on where they are located in your machine.
```
rsync-for-carbon-depsync.sh

#!/bin/sh

manager_server_dir=/wso2as-5.2.1/repository/deployment/server/
pem_file=/home/ubuntu/carbon-440-test.pem

#delete the lock on exit
trap 'rm -rf /var/lock/depsync-lock' EXIT

mkdir /tmp/carbon-rsync-logs/

#keep a lock to stop parallel runs
if mkdir /var/lock/depsync-lock; then
  echo "Locking succeeded" >&2
else
  echo "Lock failed - exit" >&2
  exit 1
fi

#get the workers-list.txt
pushd `dirname $0` > /dev/null
  SCRIPTPATH=`pwd`
popd > /dev/null
  echo $SCRIPTPATH

  for x in `cat $SCRIPTPATH/workers-list.txt`
do
    echo ============================================================== >> /tmp/carbon-rsync-logs/logs.txt;
    echo Syncing $x;
    rsync --delete -arve "ssh -i $pem_file -o StrictHostKeyChecking=no" $manager_server_dir $x
    >> /tmp/carbon-rsync-logs/logs.txt
    echo ============================================================== >> /tmp/carbon-rsync-logs/logs.txt;
  done

3. Create a Cron job that executes the above file every minute for deployment synchronization. Do this by running the following command in your command line.

You can only run the Cron job on one given node (master) at a given time. If you switch the Cron job to another node, you must stop the Cron job on the existing node and start a new Cron job on the new node after updating it with the latest files.

```bash
*/ * * * * /home/ubuntu/setup/rsync-for-depsync/rsync-for-depsync.sh
```

Configuring WSO2 Identity Server as a Key Manager

**Before you begin check the product compatibility**

- Click here to see the compatibility matrix.

The compatibility matrix with regard to WSO2 API Manager (WSO2 API-M) and WSO2 Identity Server Key Manager (WSO2 IS-KM) product distribution is as follows:
- What is referred to as the prepackaged WSO2 Identity Server as a Key Manager?

The prepackaged WSO2 Identity Server as a Key Manager 5.7.0 comes with the necessary configurations already installed in order to connect WSO2 Identity Server as the Key Manager for WSO2 API Manager, and is therefore different to the default version (vanilla pack) of WSO2 Identity Server 5.7.0. The prepackaged WSO2 Identity Server as a Key Manager 5.7.0 is compatible with WSO2 API Manager 2.6.0 and is supported by WUM.
Step 1 - Download WSO2 IS as a Key Manager (WSO2 IS-KM)

Download the prepackaged WSO2 Identity Server from here and unzip it. `<IS_KM_HOME>` will refer to the root folder of the unzipped WSO2 IS pack.

It is assumed that you have already downloaded WSO2 API Manager. `<APIM_HOME>` will refer to the root folder of the unzipped WSO2 API-M pack.

The product distributions are also available as a WUM update. For more information, see Getting Started with WUM in the Administration Guide.

Step 2 - Optionally, configure port offset for WSO2 IS

This is only required if you are running both WSO2 API Manager and WSO2 Identity Server on the same Virtual Machine (VM).

Click here for more information on port offsetting

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be offset. For example, if the HTTP port is defined as 9763 and the port offset is 1, the effective HTTP port will be 9764. Therefore, for each additional WSO2 product, instance, or cluster you add to a server, set the port offset to a unique value (the default is 0).

Open the `<IS_KM_HOME>/repository/conf/carbon.xml` file and change the offset to 1. This increments the product's default port by one.

```
carbon.xml

<Offset>1</Offset>
```

Step 3 - Install and configure the databases

You can create the required databases for the API-M deployment on a separate server and point to the databases from the respective nodes.

If you have already created the databases, you need to only configure the data source configurations, so that WSO2 IS, which acts as the Key Manager, can connect to the required databases.

Click here for information on installing and configuring the databases.

The following diagram illustrates how the databases are shared between WSO2 IS and WSO2 API-M.

<table>
<thead>
<tr>
<th>WSO2 API-M</th>
<th>WSO2 IS-KM</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>API-M 2.6.0 GA</td>
<td>GA or WUM update for WSO2 IS-KM-5.7.0</td>
<td></td>
</tr>
<tr>
<td>API-M-2.6.0 WUM update</td>
<td>GA or WUM update for WSO2 IS-KM-5.7.0</td>
<td></td>
</tr>
<tr>
<td>API-2.6.0-update-X</td>
<td>-</td>
<td>APIIM-2.6.0-update-X (e.g., WSO2 APIM-2.6.0-update-1) releases are not compatible with the GA or WUM updates that are available for WSO2 IS-KM.</td>
</tr>
</tbody>
</table>
1. **WSO2REG_DB** - This database (DB) stores the registry information. The registry database is shared between WSO2 IS as the Key Manager and WSO2 API-M to share artifacts such as, metadata configurations, policies, and API details.

2. **WSO2UM_DB** - This DB stores the permissions (i.e., permission store) and the internal roles of the users.

3. **WSO2AM_DB** - This DB stores the identity data and API-related data and it includes OAuth tokens and keys. When serving key-validation requests, the Key Manager accesses the WSO2AM_DB, validates whether there are subscriptions made by the particular key.

4. **LDAP** - This DB stores the users and their role mapping. You do not need to configure the data source configuration in the master-datasources.xml file for the LDAP.

For more information, see Configuring the Databases for IS as the Key Manager.

**Step 4 - Configure the Key Manager (WSO2 IS) with WSO2 API-M**

The following instructions are only applicable in the following scenarios:

- If you are deploying WSO2 API-M using a hybrid single node deployment with WSO2 Identity Server as the Key Manager.
- If you are deploying WSO2 API-M using a hybrid active-active deployment pattern with WSO2 Identity Server as the Key Manager.

1. Configure the Key Manager to enable communication between the Key Manager and the Gateway.

   This step is only applicable if you have enabled Hazelcast clustering in the API-M Gateway node.

   - Click here for information on configuring the WSO2 API-M Gateway with or without Hazelcast.

   - When should I use Hazelcast?
   - Distributed Deployment of the Gateway using a shared file system
   - Configuring the Gateway in a Distributed Environment with rsync

   a. When users and roles are removed via the Key Manager, if the corresponding user tokens are cached on the Gateway, these tokens will only get invalidated when the cache is timed out. However, if Hazelcast clustering is enabled, token invalidation takes place immediately. Therefore, you need to enable communication between the Key Manager and Gateway to enable immediate token invalidation.

   For this purpose open the `<IS_KM_HOME>/repository/conf/api-manager.xml` file and change the `<ServerURL>` element that appears under the `<APIGateway>` section, so that it points to the API Manager server.
1. a. If you are working with a hybrid single node or active-active node deployment where WSO2 IS is the Key Manager and the rest of the API-M components are in one node, you need to replace `{gateway-server-host}` with the host of the WSO2 API Manager node.
   b. If you are working with a single Gateway in distributed set up, you need to replace `{gateway-server-host}` with the host of the Gateway node.
   c. If you are working with Gateways in a High Availability (HA) setup that uses a shared file system (e.g., NFS), you need to replace `{gateway-server-host}` with the host of all Gateway nodes as all nodes have the worker manager capability when using shared file system.
   d. If you are working with Gateways in a High Availability (HA) setup that uses rsync, you need to replace `{gateway-server-host}` with the host of the Gateway Manager node.

   ▪ The port value you enter here should be the management transport port. For more information, see Default Product Ports.

2. Configure the JSON Web Token (JWT) in the `<IS_KM_HOME>/repository/conf/api-manager.xml` file in the WSO2 Identity Server. For more information on JWT Token generation, see Passing Enduser Attributes to the Backend Using JWT.
   - Enable `ClaimsRetrieverImplClass`, `ConsumerDialectURI`, and `SignatureAlgorithm` by uncommenting the respective elements.
   - Set `<SignatureAlgorithm>` to one of the following values: `NONE` or `SHA256withRSA`.

3. Change the datasource in the `<IS_KM_HOME>/repository/conf/user-mgt.xml` file to point to the `WSO2UM_DB` datasource. You need to do this in order to point to the correct database for user management purposes. By default, this configuration points to the embedded H2 database.

4. Make sure that the data source name defined under `JDBCPersistenceManager` is `jdbc/WSO2UM_DB` in the `<IS_KM_HOME>/repository/conf/identity/identity.xml` file.

Make sure you add the user store configuration correctly. This is the same configuration that you added in WSO2 API Manager. For more information on how to do this, see Configuring User Stores in the Administration Guide.
Step 5 - Configure WSO2 API-M with the Key Manager (WSO2 IS)

The following instructions are only applicable in the following scenarios:

- If you are deploying WSO2 API-M using a hybrid single node deployment with WSO2 Identity Server as the Key Manager.
- If you are deploying WSO2 API-M using a hybrid active-active deployment pattern with WSO2 Identity Server as the Key Manager.

1. Change the ServerURL of the AuthManager and the ServerURL of the APIKeyValidator to point to WSO2 IS in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

   You need to add this configuration so that WSO2 API Manager will be aware of the URL of the Key Manager, which in this case is WSO2 Identity Server, in order to handover the Key validation and Authorization related tasks.

   Make sure to import the Key Manager's public certificate to WSO2 API-M's `client-truststore.jks`. For more information, see Creating New Keystores.

   ```xml
   <APIManager>
   ...
   <!-- Authentication manager configuration for API publisher and API store. This is a required configuration for both web applications as their user authentication logic relies on this. -->
   <AuthManager>
   <!-- Server URL of the Authentication service -->
   <ServerURL>https://${IS-server-host}:${port}/services/</ServerURL>
   ...
   </AuthManager>
   ...
   </APIManager>
   ...
   <APIKeyValidator>
   <!-- Server URL of the API key manager -->
   <ServerURL>https://${IS-server-host}:${port}/services/</ServerURL>
   ...
   </APIKeyValidator>
   ...
   </APIManager>
   ```

   - If you are working with a single Key Manager in hybrid single node setup where WSO2 IS is the Key Manager and the rest of the API-M components are in one node, you need to replace `{IS-server-host}` with the actual host of the WSO2 IS server.
   - If you are working with a single Key Manager in distributed setup, you need to replace `{IS-server-host}` with the actual host of the WSO2 IS server.
   - If you are working with Key Managers in High Availability (HA) mode with a hybrid active-active node setup, where WSO2 IS is the Key Manager and the rest of the API-M components are in one node, you need to replace `{IS-server-host}` with the host of the Key Manager Load Balancer node.
   - If you are working with Key Managers in a High Availability (HA) setup, you need to replace `{IS-server-host}` with the host of the Key Manager Load Balancer node.

2. Enable WS Client and disable the Thrift Client.

   Do this by changing the `<KeyValidatorClientType>` from ThriftClient to WSClient, and by setting `<EnableThriftServer>` to false to optimize performance.

   As API Gateway and Key Manager are two separate components (distributed) they talk to each other via API calls. This API call happens out-of-the-box via Thrift in WSO2 API Manager. However, when it comes to a production environment with high availability, it’s
3. If you are using the embedded LDAP that comes with WSO2 IS, then you need to point to the particular LDAP user store from WSO2 API Manager. You can copy this configuration from the `<IS_KM_HOME>/repository/conf/user-mgt.xml` file to the `<API-M_HOME>/repository/conf/user-mgt.xml` file.

When copying configurations, note that you must update the ports. For instance, when configuring the `ConnectionURL` property, you must update the port, because otherwise it will point to the port of the Identity Server when starting up if you copy it directly.

```xml
<Property name="ConnectionURL">ldap://<ip_address_of_IS>:10389</Property>
```

Note that if you have offset the IS port, then this port value 10389 should be incremented by the given WSO2 IS port offset.


Make sure you add the user store configuration correctly. This is the same configuration that you added in the Identity Server. For more information on how to do this, see Configuring User Stores in the Administration Guide.

```xml
<JDBCPersistenceManager>
  <DataSource>
    <Name>jdbc/WSO2AM_DB</Name>
  </DataSource>
  ...
</JDBCPersistenceManager>
```

Step 6 - Optionally, configure High Availability (HA) for the Key Manager

These steps are ONLY applicable if you need to configure HA for the Key Manager.

1. Make a copy of the active instance configured above and use this copy as the second Key Manager active instance.
2. Configure a Load Balancer to front the two Key Manager nodes.

For more information, see Configuring the Proxy Server and the Load Balancer.

Step 7 - Start the Key Manager(s)

Start WSO2 Identity Server for the changes to take effect. For more information, see Running the Product in the WSO2 Identity Server documentation.

```bash
sh <IS_KM_HOME>/bin/wso2server.sh
```

Click here for information on troubleshooting

- You may notice the following error messages when starting up the server. This occurs because some API Manager directories are not available in the Identity Server. These are not critical errors, so they can be ignored. Alternatively, you can create the listed directories in the Identity Server pack.
If you have configured the hostnames for WSO2 API Manager and WSO2 Identity Server in the server start up, you will see the following warning in the WSO2 API Manager backend logs.

```
ERROR {org.wso2.carbon.apimgt.impl.utils.APIUtil} - Custom sequence template location unavailable for custom sequence type in : repository/resources/customsequences/in
ERROR {org.wso2.carbon.apimgt.impl.utils.APIUtil} - Custom sequence template location unavailable for custom sequence type out : repository/resources/customsequences/out
ERROR {org.wso2.carbon.apimgt.impl.utils.APIUtil} - Custom sequence template location unavailable for custom sequence type fault : repository/resources/customsequences/fault
```

The reason for this is that the default certificates that come with WSO2 Servers are created for localhost. Therefore, when WSO2 API Manager boots up, it makes an HTTP call to a webapp that is in the Key Manager (throttle data at KM_URL/throttle/data/v1/keyTemplates). Thereafter, WSO2 API Manager decides the URL of the Key Manager base on the URL that is configured in the `api-manager.xml` which is localhost.

To overcome this issue, you need to create self-signed certificates for WSO2 API-M and WSO2 IS host names. Then export the public certs of WSO2 API-M to the trust-store.jks of WSO2 IS and vice versa. This should resolve the SSL handshake failure.

**Step 8 - Configure the other API-M components**

Follow the instructions below to configure the other WSO2 API-M components, namely the Publisher, Store, Traffic Manager, and Gateway:

- If you are working with a single WSO2 API-M instance, which has all the API-M components in one instance, and a separate Key Manager, then configure the steps mentioned in **Configuring a Single Node**.
- If you are working with an Active-Active setup, which has two all-in-one instances of API-M, and a separate Key Manager in high availability (HA) mode, then configure the steps mentioned in **Configuring an Active-Active Deployment**.
- If you are working with a distributed API-M setup, see **Deploying WSO2 API-M in a Distributed Setup** to configure the other API-M components.

**Configuring the Databases for IS as the Key Manager**

The following steps describe how to download and install a RDBMS, which in this case is a MySQL Server, create the databases, configure the data sources, and configure the WSO2 Identity Server (WSO2 IS) components to connect to them.

Although the following section instructs you to use MySQL Server, you can use any RDBMS in your deployment based on your preference. For information on working with other databases, see **Changing the Default API-M Databases**.

The steps involved in installing and configuring the databases are the same irrespective of whether you are using a single node (standalone) deployment, an active-active deployment, or a distributed deployment.

1. Download the MySQL JDBC driver.
2. Unzip the downloaded MySQL driver archive, and copy the MySQL JDBC driver JAR (mysql-connector-java-x.x.xx-bin.jar) into the `<IS_HOME>/repository/components/lib` directory in all the nodes in the cluster where `<IS_HOME>` refers to the location of the extracted WSO2 Identity Server (WSO2 IS) distribution.
3. Define the hostname for configuring permissions for the new database by opening the `/etc/hosts` file and adding the following:

```plaintext
<MYSQL-DB-SERVER-IP> carbondb.mysql-wso2.com
```

4. Install mysql-client in each of the Identity Server nodes in which WSO2 Identity Server is deployed. You need to do this in order to check if the servers can access the MySQL database.
6. When prompted, specify the password that will be used to access the databases with the username you specified.

```
mysql -u username -p
```

7. Configure the data sources for the user (userdb) and the registry (regdb) databases as follows in the `<IS_HOME>/repository/conf/datasources/master-datasources.xml` file.

This file contains the different datasources used by WSO2 Identity Server. By default, the Identity Server connects to the local H2 database and it is recommended to use a separate RDBMS for a production deployment.

For more information, see Configuring `master-datasources.xml` in the Administration Guide.

**Note:** When configuring clustering, ignore the `WSO2_CARBON_DB` data source configuration.

a. Enable the components to access the WSO2 API Manager database by changing the URL as indicated below. Make sure to replace `db.mysql-wso2.com` with the hostname you specified in step 5 (carbondb.mysql-wso2.com) in the `WSO2AM_DB` data source.

```
<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for the API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://db.mysql-wso2.com:3306/apimgtdb?autoReconnect=true</url>
      <username>user</username>
      <password>password</password>
      <defaultAutoCommit>false</defaultAutoCommit>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

b. Enable the Key Manager to access the user management database.

You need to do this by adding the following code and changing the `db.mysql-wso2.com` to `carbondb.mysql-wso2.com` in order to configure the `WSO2UM_DB` data source.
c. Enable the Key Manager to have access to registry databases by adding the following **WSO2REG_DB** data source related configuration.

```xml
<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used by the registry</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://db.mysql-wso2.com:3306/regdb?autoReconnect=true</url>
            <username>user</username>
            <password>password</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

This is only applicable in a multi-tenanted environment.

8. Make the following changes in the `<IS_HOME>/repository/conf/user-mgt.xml` file.
   a. To give the Key Manager component access with shared permissions to the user management database, add or modify the `dataSource` property that corresponds to the `<configuration>` element as follows so that it points to the **WSO2UM_DB** database. By default, this configuration points to the embedded H2 database.

```xml
<configuration>
    ...<Property name="dataSource">jdbc/WSO2UM_DB</Property>
</configuration>
```

This is only applicable in a multi-tenanted environment.
b. Make sure you add the user store configuration correctly so that both the WSO2 Identity Server and WSO2 API Manager server point to the same user store. For more information, see Configuring User Stores in the Administration Guide.

You must change the `<UserStoreManager>` element here as the internal LDAP user store is used by default. You need to remove or modify the `class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager"` code block with the correct LDAP that you are using. You could alternatively use the embedded LDAP in the WSO2 Identity Server as your user store. For more information, see Configuring the Primary User Store in the Administration Guide.

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <Property name="ReadOnly">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="IsEmailUserName">false</Property>
    <Property name="DomainCalculation">default</Property>
    <Property name="PasswordDigest">SHA-256</Property>
    <Property name="StoreSaltedPassword">true</Property>
    <Property name="ReadOnly">true</Property>
    <Property name="UserNameUniqueAcrossTenants">false</Property>
    <Property name="PasswordJavaRegEx">^[\S\s]{5,30}$</Property>
    <Property name="PasswordJavaScriptRegEx">^[\S\s]{5,30}$</Property>
    <Property name="UserNameJavaRegEx">^[\S\s]{3,30}$</Property>
    <Property name="UserNameJavaScriptRegEx">^[\S\s]{3,30}$</Property>
    <Property name="RolenameJavaRegEx">^[\S\s]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">^[\S\s]{3,30}$</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="SharedGroupEnabled">false</Property>
</UserStoreManager>
```

If you are using the `WSO2UM_DB` to store users, remember to change the administrator's username and password. For more information, see Maintaining Logins and Passwords.

9. To enable the Key Manager to access to the registry database, open the `<IS_HOME>/repository/conf/registry.xml` file in the Key Manager node and add or modify the `dataSource` attribute of the `<dbConfig name="govregistry">` element as follows in order to mount the Key Manager to the governance registry space. For more information on the properties and values related to the remote mount configurations, see Configuring registry.xml in the Administration Guide.

```
<dbConfig name="wso2registry">
    <dataSource>jdbc/WSO2CarbonDB</dataSource>
</dbConfig>
```

Note that this is only applicable in a multi-tenanted environment.

Do not replace the following configuration when adding in the mounting configurations mentioned below. The registry mounting configurations mentioned in the following steps must be added beneath the following entry, which is already in the configuration file.

```
<dbConfig name="wso2registry">
    <dataSource>jdbc/WSO2CarbonDB</dataSource>
</dbConfig>
```

This configuration is related to the local H2 DB, which is used to store the mount information, indexing configuration and anything local to the node, and hence should not be removed even if separate governance and config registries are used.
<dbConfig name="govregistry">
  <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost:9443/registry">
  <id>gov</id>
  <cacheId>user@jdbc:mysql://db.mysql-wso2.com:3306/regdb</cacheId>
  <dbConfig>govregistry</dbConfig>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>\registryRoot</registryRoot>
</remoteInstance>

<mount path="/system/governance" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/system/governance</targetPath>
</mount>

<mount path="/system/config" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/system/config</targetPath>
</mount>

CacheId is a unique identification of the remote instance. When you configure the remote instance, WSO2 recommends that you modify the <cacheId> with the corresponding values of your setup, based on the following format: <username>@<JDBC_URL_to_registry_database>

- You do not need to specify the remoteInstance URL in the above configuration, because WS mounting in not used in WSO2 API-M 2.1.0 onward.
- In the above code snippet the governance registry and the config registry are pointed to the same database. If required, you can use two databases for the two registries that are shared with Key Manager nodes.

Configuring Admin App Event Publishing for Traffic Manager HA Setup

The Admin App, which is accessed via the Admin Portal (https://<API-host>:<API-port>/admin) is generally setup in the Publisher node when WSO2 API Manager (WSO2 API-M) is deployed in a distributed setup.

When the Traffic Manager (TM) is set up for high availability (HA), you need to configure the Admin App to publish events to the blockingData topic in a failover manner to enable event publishing in the Admin App. When the blockingData topic receives the message from the Admin App, it publishes the messages to the throttleData topics in both the Traffic Managers, and the Gateways receive the messages from throttleData topics.

If there is a failure in Traffic Manager 1 (TM1), the Admin App starts publishing to the blockingData event in the other Traffic Manager (blockingData 2). When TM1 is restored, blockingData2 publishes the latest events to throttleData1. As a result, the Gateways that are connected to TM1 always receive the latest changes with regard to the blacklist conditions.
Furthermore, as illustrated in the diagram below a JMS receiver listens to the `blockingData` topic. When the JMS receiver receives an event, it sends it through an event stream and publishes it through the attached JMS publishers to the `throttleData` topics.

Follow the instructions below to configure Admin App event publishing when the Traffic Managers are setup for high availability.

**Step 1 - Configure the Admin App**

1. Change the topic name to `blockingData` in the Event publisher configurations, which is in the `<API-M_HOME>/repository/conf/api-manager.xml` file.
2. Configure the **blockingData** topic in the `<API-M_HOME>/repository/conf/jndi.properties` file.

### jndi.properties

```properties
connectionfactory.TopicConnectionFactory = amqp://admin:admin@clientID/carbon?failover='roundrobin'%26cyclecount='2'%26brokerlist='tcp://127.0.0.1:5674?retries='5'%26connectdelay='50';tcp://127.0.0.1:5675?retries='5'%26connectdelay='50'

topic.blockingData = blockingData
```

Step 2 - Configure the Traffic Manager nodes

You need to carry out the following configurations in both the Traffic Manager nodes in order to send the JMS messages from the **blockingData** topic to the **throttleData** topics.

1. Add a JMS receiver.
   Create a file named `blockingEventReceiver.xml` with the following content, and move it to the `<API-M_HOME>/repository/deployment/server/eventreceivers` directory, to add a new event receiver.

### blockingEventReceiver.xml

```xml
<eventReceiver xmlns="http://wso2.org/carbon/eventreceiver" name="blockingEventReceiver" statistics="enable" trace="enable">
    <from eventAdapterType="jms">
        <property name="transport.jms.DestinationType">topic</property>
        <property name="transport.jms.Destination">blockingData</property>
        <property name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
        <property name="java.naming.provider.url">repository/conf/jndi.properties</property>
        <property name="transport.jms.SubscriptionDurable">false</property>
        <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    </from>
    <mapping customMapping="disable" type="map" />
    <to streamName="org.wso2.blocking.condition.stream" version="1.0.0" />
</eventReceiver>
```

2. Configure the **blocking event receiver**.
   Create a file named `jndi2.properties` in the `<API-M_HOME>/repository/conf` directory with the following content.
You need to add this configuration so that the JMS receiver namely `blockingEventReceiver` can listen to the local blockingData topic.

3. Add an event stream.

Create a file named `org.wso2.blocking.condition.stream_1.0.0.json` in the `<API-M_HOME>/repository/deployment/server/eventstreams` directory.

4. Add JMS publishers to publish blocking data.
   a. Add a JMS publisher to publish events from `blockingData1` to `throttleData1`.

   Create a file named `blockingconditionpublisher1.xml` in the `<API-M_HOME>/repository/deployment/server/eventpublishers` directory, to add an event publisher.
### Changing the Default API-M Databases

When you use WSO2 API Manager (WSO2 APIM), you need the following databases in addition to the Carbon database. By default, WSO2 API-M is shipped with embedded H2 databases for the following in addition to the Carbon database. These databases are stored in the `<API-M_HOME>/repository/database` directory.

- **WSO2AM_DB**: For API-M-specific data.
- **WSO2MB_DB**: For message brokering data.
- **WSO2METRICS_DB**: For storing data for Metrics monitoring.

For instructions on changing the default Carbon database, see Changing the Carbon Database in the WSO2 Product Administration Guide.

---

### blockingconditionpubblisher1.xml

```xml
<eventPublisher xmlns="http://wso2.org/carbon/eventpublisher" name="blockingconditionpubblisher1" processing="enable" statistics="disable" trace="enable">
  <from streamName="org.wso2.blocking.condition.stream" version="1.0.0" />
  <mapping customMapping="disable" type="map" />
  <to eventAdapterType="jms">
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.Destination">throttleData</property>
    <property name="transport.jms.ConcurrentPublishers">allow</property>
    <property name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
    <property name="java.naming.provider.url">repository/conf/jndi.properties</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
  </to>
</eventPublisher>
```

### blockingconditionpubblisher2.xml

```xml
<eventPublisher xmlns="http://wso2.org/carbon/eventpublisher" name="blockingconditionpubblisher2" processing="enable" statistics="disable" trace="enable">
  <from streamName="org.wso2.blocking.condition.stream" version="1.0.0" />
  <mapping customMapping="disable" type="map" />
  <to eventAdapterType="jms">
    <property name="transport.jms.DestinationType">topic</property>
    <property name="transport.jms.Destination">throttleData</property>
    <property name="transport.jms.ConcurrentPublishers">allow</property>
    <property name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
    <property name="java.naming.provider.url">repository/conf/jndi3.properties</property>
    <property name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
  </to>
</eventPublisher>
```

b. Add a JMS publisher to publish events from blockingData1 to throttleData2.

Create a file named `blockingconditionpubblisher2.xml` in the `<API-M_HOME>/repository/deployment/server/eventpublishers` directory, to add another event publisher.
Given below are the steps you need to follow in order to change the default databases listed above.

- **Step 1 - Set up the database**
  - **Step 2 - Create the datasource connection**
    - Create the datasource connection for the API-M database
    - Create the datasource connection for the MB database (MB Store in WSO2 API-M)
    - Create the datasource connection for the Metrics database
    - Create the datasource connection for the Analytics database
  - **Step 5 - Configure keystores**
  - **Step 3 - Create database tables**
    - Create database tables in the API-M database
    - Create database tables in the MB database
    - Create database tables in the Metrics database

### Step 1 - Set up the database

You can set up the following database types for the API-M-specific databases:

- Setting up a MySQL database
- Setting up an MS SQL database
- Setting up an Oracle database
- Setting up an IBM DB2 database
- Setting up a PostgreSQL database

Note that we recommend to use Fail Over configuration over Load Balanced configuration with the MySQL clusters.

### Step 2 - Create the datasource connection

A datasource is used to establish the connection to a database. By default, datasource connections for the API-M database, API-M statistics database, and the Message Brokering database are configured in the master-datasources.xml file. The datasource connection for the Metrics database is configured in the master-datasources.xml file. These datasource configurations point to the default H2 databases, which are shipped with the product. After setting up new databases to replace the default H2 databases, you can either change the default configurations in the above-mentioned files or configure new datasources.

Create the datasource connection for the API-M database

Follow the steps below.

1. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and locate the `<datasource>` configuration element.
2. Update the URL pointing to your database, the `username` and `password` required to access the database, and the `driver` details as shown below.
   - Optionally, you can update the other elements for your database connection.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>maxActive</code></td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td><code>maxWait</code></td>
<td>The maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td><code>minIdle</code></td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created. You can enter zero to create none.</td>
</tr>
<tr>
<td><code>testOnBorrow</code></td>
<td>The indication of whether objects are validated before being borrowed from the pool. If the object fails to validate, it is dropped from the pool, and another attempt is made to borrow another.</td>
</tr>
<tr>
<td><code>validationQuery</code></td>
<td>The SQL query that is used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation but has been validated previously within this interval, it is not validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is only applicable to the MB Store database of WSO2 API-M, where this property should be explicitly set to false. In all other database connections explained above, auto committing is enabled or disabled at the code level as required for that database, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Note that auto committing is typically enabled for an RDBMS by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<API-M_HOME>/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

MySQL, MS SQL, Oracle, IBM DB2, PostgreSQL:

```xml
<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://localhost:3306/WSO2AM_DB</url>
      <username></username>
      <password></password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:jtds:sqlserver://localhost:1433/WSO2AM_DB</url>
      <username></username>
      <password></password>
      <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driverClassName>
      <maxActive>200</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```
<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:oracle:thin:@localhost:1521/orcl</url>
      <username></username>
      <password></password>
      <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
      <maxActive>100</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1 FROM DUAL</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:db2://SERVER_NAME:PORT/WSO2AM_DB</url>
      <username></username>
      <password></password>
      <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>360000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
<datasource>
  <name>WSO2AM_DB</name>
  <description>The datasource used for API Manager database</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:postgresql://localhost:5432/WSO2AM_DB</url>
      <username></username>
      <password></password>
      <driverClassName>org.postgresql.Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1; COMMIT</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

Create the datasource connection for the MB database (MB Store in WSO2 API-M)

Follow the steps below.

1. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and locate the `<datasource>` configuration element.
2. Update the URL pointing to your database, the `username` and `password` required to access the database, and the `driver` details as shown below. Further, be sure to set the `<defaultAutoCommit>` element to `false` for the MB database.

   - Optionally, you can update the other elements for your database connection.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created. You can enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects are validated before being borrowed from the pool. If the object fails to validate, it is dropped from the pool, and another attempt is made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that is used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation but has been validated previously within this interval, it is not validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is only applicable to the MB Store database of WSO2 APIM, where this property should be explicitly set to <code>false</code>. In all other database connections explained above, auto committing is enabled or disabled at the code level as required for that database, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Note that auto committing is typically enabled for an RDBMS by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<API-M_HOME>/conf/datasources/master-datasources.xml` file, see Tomcat JDBC Connection Pool.

MySQL MS SQL Oracle IBM DB2 PostgreSQL

Copyright © WSO2 Inc. 2015-2018
<datasource>
  <name>WSO2_MB_STORE_DB</name>
  <description>The datasource used for message broker database</description>
  <jndiConfig>
    <name>WSO2MBStoreDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://localhost:3306/WSO2MB_DB</url>
      <username></username>
      <password></password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

<datasource>
  <name>WSO2_MB_STORE_DB</name>
  <description>The datasource used for message broker database</description>
  <jndiConfig>
    <name>WSO2MBStoreDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <username></username>
      <password></password>
      <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driverClassName>
      <maxActive>200</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
<datasource>
  <name>WSO2_MB_STORE_DB</name>
  <description>The datasource used for message broker database</description>
  <jndiConfig>
    <name>WSO2MBStoreDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:oracle:thin:@localhost:1521/orcl</url>
      <username></username>
      <password></password>
      <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
      <maxActive>100</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1 FROM DUAL</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

<datasource>
  <name>WSO2_MB_STORE_DB</name>
  <description>The datasource used for message broker database</description>
  <jndiConfig>
    <name>WSO2MBStoreDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:db2://SERVER_NAME:PORT/WSO2MB_DB</url>
      <username></username>
      <password></password>
      <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>360000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
Create the datasource connection for the Metrics database

Follow the steps below.

1. Open the `<API-M_HOME>/repository/conf/datasources/metrics-datasources.xml` file and locate the `<datasource>` configuration element.
2. Update the URL pointing to your database, the username and password required to access the database, and the driver details as shown below.
   - Optionally, you can update the other elements for your database connection.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created. You can enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects are validated before being borrowed from the pool. If the object fails to validate, it is dropped from the pool, and another attempt is made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that is used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation but has been validated previously within this interval, it is not validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is only applicable to the MB Store database of WSO2 APIIM, where this property should be explicitly set to false. In all other database connections explained above, auto committing is enabled or disabled at the code level as required for that database, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Note that auto committing is typically enabled for an RDBMS by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<API-M_HOME>/conf/datasources/ master-datasources.xml` file, see Tomcat JDBC Connection Pool.

MySQLMS SQL Oracle PostgreSQL
<datasource>
  <name>WSO2_METRICS_DB</name>
  <description>The MySQL datasource used for WSO2 Carbon Metrics</description>
  <jndiConfig>
    <name>jdbc/WSO2MetricsDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <url>jdbc:mysql://localhost:3306/WSO2METRICS_DB</url>
      <username>root</username>
      <password>root</password>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

<datasource>
  <name>WSO2_METRICS_DB</name>
  <description>The MSSQL datasource used for WSO2 Carbon Metrics</description>
  <jndiConfig>
    <name>jdbc/WSO2MetricsDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <driverClassName>net.sourceforge.jtds.jdbc.Driver</driverClassName>
      <url>jdbc:jtds:sqlserver://localhost:1433/wso2_metrics</url>
      <username>sa</username>
      <password>sa</password>
      <maxActive>200</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>true</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
Create the datasource connection for the Analytics database

This section is only applicable if you have downloaded the WSO2 API Analytics distribution to use WSO2 API Analytics with WSO2 API-M.

The API Manager integrates with the WSO2 Analytics platform to provide reports, statistics, and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs, and detect unusual activity, manage locations via geo location statistics, and carry out detailed analysis of the logs.

Follow the steps below to create the datasource connection for the Analytics database:

When working with Analytics, ensure that the WSO2AM_DB database is of the same RDBMS type as the Analytics database. For example, if the Analytics related DBs are created in MySQL, the API-M databases (WSO2AM_DB) should also be created in MySQL.
The following is a list of database versions that are compatible with WSO2 API-M Analytics.

- Postgres 9.5 and later
- MySQL 5.6
- MySQL 5.7
- Oracle 12c
- MS SQL Server 2012
- DB2

1. Stop the WSO2 API-M Analytics server if it is running already.
2. To configure the dashboard profile, open the `<API-M_ANALYTICS_HOME>/conf/dashboard/deployment.yaml` file and edit the `APIM_ANALYTICS_DB` section. A sample for MySQL is shown below.

```yaml
- name: APIM_ANALYTICS_DB
description: "The datasource used for APIIM statistics aggregated data."
jndiConfig:
  name: jdbc/APIM_ANALYTICS_DB
definition:
type: RDBMS
configuration:
  jdbcUrl: 'jdbc:mysql://localhost:3306/stats_db'
  username: 'root'
  password: '123'
  driverClassName: com.mysql.jdbc.Driver
  maxPoolSize: 50
  idleTimeout: 60000
  connectionTestQuery: SELECT 1
  validationTimeout: 30000
  isAutoCommit: false
```

3. To configure the worker profile, open the `<API-M_ANALYTICS_HOME>/conf/worker/deployment.yaml` file and repeat the above step.
4. Configure `WSO2AM_MGW_ANALYTICS_DB` to the same database as `APIM_ANALYTICS_DB` in the `<API-M_ANALYTICS_HOME>/conf/worker/deployment.yaml` file.

```yaml
- name: WSO2AM_MGW_ANALYTICS_DB
description: "The datasource used for APIM MGW analytics data."
jndiConfig:
  name: jdbc/WSO2AM_MGW_ANALYTICS_DB
definition:
type: RDBMS
configuration:
  jdbcUrl: 'jdbc:mysql://localhost:3306/stats_db'
  username: 'root'
  password: '123'
  driverClassName: com.mysql.jdbc.Driver
  maxPoolSize: 50
  idleTimeout: 60000
  connectionTestQuery: SELECT 1
  validationTimeout: 30000
  isAutoCommit: false
```

5. Create the `AM_USAGE_UPLOADED_FILES` table in the `WSO2AM_STATS_DB` database.
A sample MySQL script to create the table is given below.

```sql
CREATE TABLE IF NOT EXISTS AM_USAGE_UPLOADED_FILES (  
  FILE_NAME varchar(255) NOT NULL,  
  FILE_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  FILE_PROCESSED tinyint(1) DEFAULT 0,  
  FILE_CONTENT MEDIUMBLOB DEFAULT NULL,  
  PRIMARY KEY (FILE_NAME, FILE_TIMESTAMP)
);```

6. Do the following to integrate third-party products when configuring databases.
WSO2 SP is an OSGi-based product. Therefore, when you integrate third party products such as Oracle with WSO2 API-M Analytics, you need to check whether the libraries you need to add are OSGi based. If they are not, you need to convert them to OSGi bundles before adding them to the <API-M_ANALYTICS_HOME>/lib directory.

To convert the jar files to OSGi bundles, follow the steps below.

1. Download the non-OSGi jar for the required third party product, and save it in a preferred directory in your machine.
2. Go to the <API-M_ANALYTICS_HOME>/bin directory. Run the command given below, to generate the converted file in the <API-M_ANALYTICS_HOME>/lib directory.

```
./jartobundle.sh <PATH_TO_NON-OSGi_JAR> ../lib
```

7. Start the WSO2 API-M Analytics server.

**Step 5 - Configure keystores**

In the SSL handshake between the API Manager and API Manager Analytics servers, the client (i.e. API Manager) needs to verify the certificate presented by the server (i.e. API Manager Analytics). For this purpose, the client stores the trusted certificate of the server in the client-truststore.jks.

If you use a custom keystore in API Manager and/or API Manager Analytics, import the public key certificate of API Manager Analytics into the client-truststore.jks file of the API Manager. To export the public key from the server and import it into the client's trust store, follow the steps given in Adding CA-signed certificates to keystores in the Administration Guide.

**Step 3 - Create database tables**

To create the database tables, connect to the databases that you created earlier and run the scripts provided in the product pack.

**Create database tables in the API-M database**

The DB scripts corresponding to the database type are provided in the <API-M_HOME>/dbscripts/apimgt directory.

To create the necessary database tables:

1. Connect to the database and run the relevant script.
   For example, run the following command to create the API-M tables in a MySQL database.

```
mysql -u root -p -DWSO2AM_DB < '<API-M_HOME>/dbscripts/apimgt/mysql.sql';
```

**<API-M_HOME>/dbscripts/mb-store/apimgt/mysql.sql** is the script that should be used for MySQL 5.6 and prior versions. If you database is MySQL 5.7 or later version, use **<API-M_HOME>/dbscripts/mb-store/mysql5.7.sql** script file.

2. Restart the WSO2 API-M server.

**Create database tables in the MB database**

The DB scripts corresponding to the database type are provided in the <API-M_HOME>/dbscripts /mb-store directory.

To create the necessary database tables:

1. Connect to the database and run the relevant script.
   For example, run the following command to create the MB tables in a MySQL database.

```
mysql -u root -p -DWSO2MB_DB < '<API-M_HOME>/dbscripts/mb-store/mysql.sql';
```

**<API-M_HOME>/dbscripts/mb-store/mb-store/mysql.sql** is the script that should be used for MySQL 5.6 and prior versions. If you database is MySQL 5.7 or later version, use **<API-M_HOME>/dbscripts/mb-store/mysql5.7.sql** script file.

2. Restart the WSO2 API-M server.

**Create database tables in the Metrics database**
The DB scripts corresponding to the database type are provided in the `<API-M_HOME>/dbscripts/metrics directory.

To create the necessary database tables:

1. Connect to the database and run the relevant script.
   For example, run the following command to create the MB tables in a MySQL database.
   
   ```
   mysql -u root -p -D WSO2_METRICS_DB < `<API-M_HOME>/dbscripts/metrics/mysql.sql';
   ```

   `<API-M_HOME>/dbscripts/metrics/mysql.sql` is the script that should be used for MySQL 5.6 and prior versions. If your database is MySQL 5.7 or later version, use `<API-M_HOME>/dbscripts/metrics/mysql5.7.sql` script file.

2. Restart the WSO2 API-M server.

### Administration - API-M Analytics

The following sections cover the administration tasks carried out for WSO2 API-M Analytics.

- **Re-indexing Existing Data**

**Re-indexing Existing Data**

This section covers reindexing data that are already stored in databases configured for WSO2 API-M Analytics.

Data indexed and stored in databases may need to be reindexed due to the following reasons:

- If the index data is corrupted.
- If you change the database to another database, the data in the new database needs to be re-indexed.

To reindex existing data, follow the steps below:

1. Shut down API-M Analytics.
2. Remove all the index data stored in the `<API-M_ANALYTICS_HOME>/repository/data` directory.
3. In the `<API-M_ANALYTICS_HOME>/repository/conf/analytics/local-shard-allocation-config.conf` file, change the mode for all the shards from `NORMAL` to `INIT`.
4. Restart API-M Analytics.

### Common Runtime and Configuration Artifacts

The following are the artifacts used commonly in a WSO2 API Manager and API Manager Analytics deployment.

- **Persistent Runtime Artifacts** - directories in API Manager which includes deployable files which are valid from a specified date and time at runtime.

- **Persistent Configuration Artifacts** - directories in API Manager where the configuration files are included which are used for configurations.

Persistent runtime artifacts could be updated at the runtime, and are expected to be available across instance restarts, VM re-creation or container re-spawning. Persistent file storage systems should be used to ensure this.

Ex.: In a Kubernetes based container environment, its possible to use Persistent Volumes to persist these artifacts.

- **API Manager**
  - Persistent Runtime Artifacts
  - Persistent Configuration Artifacts

- **APIM Analytics**
  - Persistent Runtime Artifacts
  - Persistent Configuration Artifacts

### API Manager

**Persistent Runtime Artifacts**

- `<API-M_HOME>/repository/deployment/server` - Contains webapps that are related to customizing WSO2 API Manager during a deployment. Required for deploying a super tenant
- `<API-M_HOME>/repository/tenants` - This is only used when the deployment involves multi-tenancy. For more information, see Configuring Multiple Tenants
- `<API-M_HOME>/solr` - Contains files for Apache solr indexing. For additional information, see Add Apache Solr-Based Indexing
- `<API-M_HOME>/repository/database` - H2 database (For solr indexing)
Shared Artifacts
The following artifacts can be shared among API Manager nodes

- `<API-M_HOME>/repository/deployment/server`
- `<API-M_HOME>/repository/tenants`

Persistent Configuration Artifacts

- `<API-M_HOME>/repository/resources` - This folder/artifact contains such as keystores, templates, scripts, synapse configurations and custom sequences etc.
- `<API-M_HOME>/repository/conf` - This folder contains the configuration files related to servers, datasources, registry, user management, etc.
- `<API-M_HOME>/bin` - Contains files for JVM changes, profile changes, etc.

APIM Analytics

Persistent Runtime Artifacts

- `<API-M_ANALYTICS_HOME>/repository/deployment/server` - Contains webapps, execution plans, event receivers, etc that are related to customizing WSO2 APIM Analytics during a deployment. Required for deploying a super tenant.
- `<API-M_ANALYTICS_HOME>/repository/data` - Contains the indexing files
- `<API-M_ANALYTICS_HOME>/repository/conf/analytics/-` - My node id and shard allocation related data are stored in this directory.

Shared Artifacts
The following artifacts can be shared among API Manager nodes

- `<API-M_ANALYTICS_HOME>/repository/deployment/server`

Persistent Configuration Artifacts

- `<API-M_ANALYTICS_HOME>/repository/resources` - Resources such as key stores, templates, etc.
- `<API-M_ANALYTICS_HOME>/repository/conf` - This folder contains the configuration files related to servers, datasources, registry, user management, etc.
- `<API-M_ANALYTICS_HOME>/bin` - Contains files for JVM changes, profile changes, etc.

JMX Monitoring
Java Management Extensions (JMX) is a technology that lets you implement management interfaces for Java applications. JConsole is a JMX-compliant monitoring tool, which comes with the Java Development Kit (JDK) 1.5 or later versions. Therefore, when you use a WSO2 product, JMX is enabled by default, which allows you to monitor the product using JConsole.

Go to the WSO2 Administration Guide for detailed instructions on how to configure JMX for a WSO2 product and how to use JConsole for monitoring a product.

MBeans for WSO2 API Manager

When JMX is enabled, WSO2 ESBAPI Manager exposes a number of management resources as JMX MBeans that can be used for managing and monitoring the running server. When you start JConsole, you can monitor these MBeans from the MBeans tab. While some of these MBeans (ServerAdmin and DataSource) are common to all WSO2 products, some MBeans are specific to WSO2 API Manager.

The common MBeans are explained in detail in the WSO2 Administration Guide. Listed below are the MBeans that are specific to WSO2 API Manager.
This section summarizes the attributes and operations available for the following ESB specific MBeans:

- Connection MBeans
- Latency MBeans
- Threading MBeans
- Transport MBeans

**Connection MBeans**

These MBeans provide connection statistics for the HTTP and HTTPS transports.

You can view the following Connection MBeans:

- `org.apache.synapse/PassThroughConnections/http-listener`
- `org.apache.synapse/PassThroughConnections/http-sender`
- `org.apache.synapse/PassThroughConnections/https-listener`
- `org.apache.synapse/PassThroughConnections/https-sender`

**Attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveConnections</td>
<td>Number of currently active connections.</td>
</tr>
<tr>
<td>LastXxxConnections</td>
<td>Number of connections created during last Xxx time period.</td>
</tr>
<tr>
<td>RequestSizesMap</td>
<td>A map of number of requests against their sizes.</td>
</tr>
<tr>
<td>ResponseSizesMap</td>
<td>A map of number of responses against their sizes.</td>
</tr>
<tr>
<td>LastResetTime</td>
<td>Last time connection statistic recordings was reset.</td>
</tr>
</tbody>
</table>
Operations

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset()</td>
<td>Clear recorded connection statistics and restart recording.</td>
</tr>
</tbody>
</table>

Latency MBeans

This view provides statistics of the latencies from all backend services connected through the HTTP and HTTPS transports. These statistics are provided as an aggregate value.

You can view the following Latency MBeans:

- org.apache.synapse/PassthroughLatencyView/nio-http-http
- org.apache.synapse/PassthroughLatencyView/nio-https-https

Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg_Latency</td>
<td>Average latency since latency recording was last reset.</td>
</tr>
<tr>
<td>txxx_AvgLatency</td>
<td>Average latency for last xxx time period. For example, LastHourAvgLatency returns the average latency for the last hour.</td>
</tr>
<tr>
<td>LastResetTime</td>
<td>Last time latency statistic recording was reset.</td>
</tr>
<tr>
<td>Avg_Client_To_Esb_RequestReadTime</td>
<td>Average Time taken to read request by API Manager which is sent by the client</td>
</tr>
<tr>
<td>xxx_Avg_Client_To_Esb_RequestReadTime</td>
<td>Average Time taken to read request by gateway which is sent by the client for last xxx time period. For example, 15m_Avg_Client_To_Esb_RequestReadTime means average Time taken to read request by API Manager which is sent by the client for last 15 minutes.</td>
</tr>
<tr>
<td>Avg_Esb_To_Backend_RequestWriteTime</td>
<td>Average Time taken to write the request from gateway to the backend.</td>
</tr>
<tr>
<td>xxx_Avg_Esb_To_Backend_RequestWriteTime</td>
<td>Average Time taken to write the request from gateway to the backend in last xxx time period. For example 15m_Avg_Esb_To_Backend_RequestWriteTime is average Time taken to write the request from gateway to the backend in last 15 minutes.</td>
</tr>
<tr>
<td>Avg_Backend_To_Esb_ResponseReadTime</td>
<td>Average Time taken to read the response from gateway to backend.</td>
</tr>
<tr>
<td>xxx_Avg_Backend_To_Esb_ResponseReadTime</td>
<td>Average Time taken to read the response from gateway to backend in last xxx time period.</td>
</tr>
<tr>
<td>Avg_Esb_To_Client_ResponseWriteTime</td>
<td>Average time taken to write the response from gateway to the client application.</td>
</tr>
<tr>
<td>xxx_Avg_Esb_To_Client_ResponseWriteTime</td>
<td>Average time taken to write the response from gateway to the client application in last xxx time period.</td>
</tr>
<tr>
<td>Avg_ClientWorker_Queued_Time</td>
<td>Average time where the ClientWorker get queued.</td>
</tr>
<tr>
<td>xxx_Avg_ClientWorker_Queued_Time</td>
<td>Average time where the ClientWorker get queued in last xxx time period.</td>
</tr>
<tr>
<td>Avg_ServerWorker_Queued_Time</td>
<td>Average time where the ServerWorker get queued.</td>
</tr>
<tr>
<td>xxx_Avg_ServerWorker_Queued_Time</td>
<td>Average time where the ServerWorker get queued in last xxx time period.</td>
</tr>
<tr>
<td>Avg_Latency_Backend</td>
<td>Average backend latency.</td>
</tr>
<tr>
<td>xxx_Avg_Latency_Backend</td>
<td>Average backend latency in last xxx time period.</td>
</tr>
<tr>
<td>Avg_Request_Mediation_Latency</td>
<td>Average latency of mediating the requests.</td>
</tr>
<tr>
<td>Avg_Response_Mediation_Latency</td>
<td>Average latency of mediating the responses.</td>
</tr>
</tbody>
</table>

Operations

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset()</td>
<td>Clear recorded latency statistics and restart recording.</td>
</tr>
</tbody>
</table>

Threading MBeans

These MBeans are only available in the NHTTP transport and not in the default Pass Through transport.

You can view the following Threading MBeans:
Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalWorkerCount</td>
<td>Total worker threads related to this server/client.</td>
</tr>
<tr>
<td>AvgUnblockedWorkerPercentage</td>
<td>Time-averaged unblocked worker thread percentage.</td>
</tr>
<tr>
<td>AvgBlockedWorkerPercentage</td>
<td>Time-averaged blocked worker thread percentage.</td>
</tr>
<tr>
<td>LastXxxBlockedWorkerPercentage</td>
<td>Blocked worker thread percentage averaged for last Xxx time period.</td>
</tr>
<tr>
<td>DeadLockedWorkers</td>
<td>Number of deadlocked worker threads since last statistics reset.</td>
</tr>
<tr>
<td>LastResetTime</td>
<td>Last time thread statistic recordings was reset.</td>
</tr>
</tbody>
</table>

Operations

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset()</td>
<td>Clear recorded thread statistic and restart recording.</td>
</tr>
</tbody>
</table>

Transport MBeans

For each transport listener and sender enabled in the ESB, there will be an MBean under the `org.apache.axis2/Transport` domain. For example, when the JMS transport is enabled, the following MBean will be exposed:

- `org.apache.axis2/Transport/jms-sender-n`

You can also view the following Transport MBeans:

- `org.apache.synapse/Transport/passthru-http-receiver`
- `org.apache.synapse/Transport/passthru-http-sender`
- `org.apache.synapse/Transport/passthru-https-receiver`
- `org.apache.synapse/Transport/passthru-https-sender`

Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveThreadCount</td>
<td>Threads active in this transport listener/sender.</td>
</tr>
<tr>
<td>AvgSizeReceived</td>
<td>Average size of received messages.</td>
</tr>
<tr>
<td>AvgSizeSent</td>
<td>Average size of sent messages.</td>
</tr>
<tr>
<td>BytesReceived</td>
<td>Number of bytes received through this transport.</td>
</tr>
<tr>
<td>BytesSent</td>
<td>Number of bytes sent through this transport.</td>
</tr>
<tr>
<td>FaultsReceiving</td>
<td>Number of faults encountered while receiving.</td>
</tr>
<tr>
<td>FaultsSending</td>
<td>Number of faults encountered while sending.</td>
</tr>
<tr>
<td>LastResetTime</td>
<td>Last time transport listener/sender statistic recording was reset.</td>
</tr>
<tr>
<td>MaxSizeReceived</td>
<td>Maximum message size of received messages.</td>
</tr>
<tr>
<td>MaxSizeSent</td>
<td>Maximum message size of sent messages.</td>
</tr>
<tr>
<td>MetricsWindow</td>
<td>Time difference between current time and last reset time in milliseconds.</td>
</tr>
<tr>
<td>MinSizeReceived</td>
<td>Minimum message size of received messages.</td>
</tr>
<tr>
<td>MinSizeSent</td>
<td>Minimum message size of sent messages.</td>
</tr>
<tr>
<td>MessagesReceived</td>
<td>Total number of messages received through this transport.</td>
</tr>
<tr>
<td>MessagesSent</td>
<td>Total number of messages sent through this transport.</td>
</tr>
<tr>
<td>QueueSize</td>
<td>Number of messages currently queued. Messages get queued if all the worker threads in this transport thread pool are busy.</td>
</tr>
<tr>
<td>ResponseCodeTable</td>
<td>Number of messages sent against their response codes.</td>
</tr>
</tbody>
</table>
### Timeouts

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving</td>
<td>Message receiving timeout.</td>
</tr>
<tr>
<td>Sending</td>
<td>Message sending timeout.</td>
</tr>
</tbody>
</table>

### Operations

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start()</td>
<td>Start this transport listener/sender.</td>
</tr>
<tr>
<td>stop()</td>
<td>Stop this transport listener/sender.</td>
</tr>
<tr>
<td>resume()</td>
<td>Resume this transport listener/sender which is currently paused.</td>
</tr>
<tr>
<td>resetStatistics()</td>
<td>Clear recorded transport listener/sender statistics and restart recording.</td>
</tr>
<tr>
<td>pause()</td>
<td>Pause this transport listener/sender which has been started.</td>
</tr>
<tr>
<td>maintenenceShutdown(long gracePeriod)</td>
<td>Stop processing new messages, and wait the specified maximum time for in-flight requests to complete before a controlled shutdown for maintenence.</td>
</tr>
</tbody>
</table>

### Working with Observability

Observability in WSO2 API Manager (WSO2 API-M) is really important to debug issues in a short period. WSO2 API-M facilitates observability by logging the following important points of the system with the time taken to achieve them.

- Method Calls
- External Calls (HTTP/HTTPS)
- Database Calls (JDBC and LDAP)

Furthermore, when observability is enabled in WSO2 API-M, a random correlation ID is generated within the WSO2 API-M for each transaction allowing you to correlate the latter three types of calls. Thereby, the requests and the responses that correspond to a specific API call will be logged under one correlation ID making it easier to analyze the information. If required, you can provide a unique correlation ID by adding the `activityid` in the header to the request sent to WSO2 API-M.

- Observability is not enabled by default as it impacts WSO2 API Manager's performance.
- In order to use this feature, apply the WUM update for WSO2 API-M 2.6.0 released on 2018-11-24.

If you want to deploy a WUM update in to production, you need to have a paid subscription. If you do not have a paid subscription, you can use this feature with the next version of WSO2 API Manager when it is released. For more information on updating WSO2 API Manager using WUM, see [Getting Started with WUM](#) in the WSO2 Updates Guide.

The following sections provide for more information on observability with regard to WSO2 API Manager.

- Enabling observability on WSO2 API-M
- Types of correlation logs
- Using the correlation logs
- Reading and analyzing the correlation logs
- Advanced use cases

### Enabling observability on WSO2 API-M

- **Step 1 - Configure observability**
  - **Step 2 - Enable observability**
  - **Step 3 - Start the WSO2 API-M server**

**Step 1 - Configure observability**

1. Add the Log4J configurations so that a log file is created for the purpose of observability when API Manager server is started.
   
   Add the following code to the `<API-M_HOME>/repository/conf/log4j.properties` file.
# correlation logs
log4j.logger.correlation=INFO, CORRELATION
log4j.additivity.correlation=false

# Appender config for correlation logs
log4j.appender.CORRELATION=org.apache.log4j.RollingFileAppender
log4j.appender.CORRELATION.File=${carbon.home}/repository/logs/${instance.log}/correlation.log
log4j.appender.CORRELATION.MaxFileSize=10MB
log4j.appender.CORRELATION.layout=org.apache.log4j.PatternLayout
log4j.appender.CORRELATION.Threshold=INFO
log4j.appender.CORRELATION.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS}|%X{Correlation-ID}|%t|%m%n

Note that the maximum file size of the correlation log is set to 10MB by default in the above configuration, which means that when the size of the file exceeds 10MB, a new log file is created. If required, you can change this file size.

2. Add the valve configurations for the purpose of deriving the correlation ID.
   Add the following configurations under the <Host> tag in the <API-M_HOME>/repository/conf/tomcat/catalina-server.xml file.

   <Valve className="org.wso2.carbon.tomcat.ext.valves.RequestCorrelationIdValve"
   headerToCorrelationIdMapping="{'activityid':'Correlation-ID'}"
   queryToCorrelationIdMapping="{'RelayState':'Correlation-ID'}"/>

3. Add the Synapse Handler configurations to facilitate external call logging.
   Add the following configurations as a handler after the <handler> tag in the <API-M_HOME>/repository/conf/synapse-handlers.xml file.

   <handler name="externalCallLogger"
   class="org.wso2.carbon.apimgt.gateway.handlers.LogsHandler"/>

**Step 2 - Enable observability**

If you want correlation logs to be enabled every time the server starts, add the following system property to the product startup script (stored in the <API-M_HOME>/bin/ directory) and set it to true.

Skip this step if you want to pass the system property to enable observability at the time of starting the WSO2 API-M server.

**Linux/Mac OS Windows**

Add the following configuration as a system property to the <API-M_HOME>/bin/wso2server.sh file. By default, this is set to false.

```
-DenableCorrelationLogs=true 
```

Make sure to add it before the org.wso2.carbon.bootstrap.Bootstrap $* line.

In a Windows environment, append the following system property to CMD_LINE_ARGS in the <API-M_HOME>/bin\wso2server.bat file as follows:

```
-DenableCorrelationLogs=true 
```

**Step 3 - Start the WSO2 API-M server**
Start the WSO2 API-M server.

- If you have **enabled observability before**, navigate to the `<API-M_HOME>/bin` directory and run the following command.
  
  ```bash
  sh wso2server.sh
  ```

- If you have **not enabled observability before**, navigate to the `<API-M_HOME>/bin` directory and enable observability at the time of starting the WSO2 API-M server as follows:
  
  ```bash
  sh wso2server.sh -DenableCorrelationLogs=true start
  ```

- If you have **enabled observability before**, navigate to the `<API-M_HOME>/bin` directory and run the following command.
  
  ```cmd
  wso2server.bat --run
  ```

- If you have **not enabled observability before**, navigate to the `<API-M_HOME>/bin` directory and enable observability at the time of starting the WSO2 API-M server as follows:
  
  ```cmd
  wso2server.bat --run -DenableCorrelationLogs=true start
  ```

When observability is enabled in WSO2 API Manager, a separate log file named `correlation.log` is created in the `<API-M_HOME>/repository/logs` directory.

### Types of correlation logs

The following are the types of logs that are available when working with observability in WSO2 API-M.

- Method call logs
- External call logs
- Database call logs

### Method call logs

When correlation logging is enabled, the API Manager logs the time taken to execute certain important methods of the following modules.

- `org.wso2.carbon.apimgt.gateway`
- `org.wso2.carbon.apimgt.keymgt`
- `org.wso2.carbon.apimgt.impl`

In API Manager, by default the important methods are marked with the `@MethodStats` annotation, and this annotation can be found at both the method level and the class level. All the methods of the respective class are included for logging for the classes that have the latter mentioned annotation. The format of a method log entry is as follows:

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2015-2018
The following is a detailed description of the method call log entry.

<table>
<thead>
<tr>
<th>timestamp</th>
<th>The time at which the log is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>2018-11-28 10:10:56,293</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>correlationID</th>
<th>Each log contains a correlation ID, which is unique to the HTTP request. A client can send a unique correlation ID in the header of the HTTP request. If this correlation ID is missing in the incoming request, WSO2 API-M will generate a random correlation ID for the request. The HTTP header that carries the correlation ID is configured in WSO2 API-M.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>a783f7c3-647f-4d10-9b72-106faa01bba8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>threadName</th>
<th>The identifier of the thread.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>PassThroughMessageProcessor-3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>duration</th>
<th>The time gap (in milliseconds) between two states of the message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>callType</th>
<th>METHOD - The call type is METHOD in order to indicate that it is a method level log.</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>Class name of the method which was invoked.</td>
</tr>
<tr>
<td>methodName</td>
<td>Name of the method which was invoked.</td>
</tr>
<tr>
<td>methodArguments</td>
<td>Parameters of the method that was invoked.</td>
</tr>
</tbody>
</table>

If you need to log all the methods that correspond to a package, you need to specify the package name as the value of the `logAllMethods` system property. For more information, see Logging all methods.

**External call logs**
You can enable correlation logs in WSO2 API-M to track the complete round trip of an individual HTTP message, which means the monitoring of individual HTTP requests from the point that a message is received by WSO2 API-M until the corresponding response message is sent back to the original message sender (client API-M back-end API-M client). Thereby, you can use the correlation log file to monitor and analyze external calls in detail. The following are the two types of external call logs that can be tracked via observability in WSO2 API-M.

- Synapse global handler level external call logs
- Synapse passthrough transport level external call logs

### Synapse global handler level external call logs

All external calls done by the API Manager is logged via this category. Note that this does not include DB calls. This is done via a Synapse Global Handler that logs the important information of the external calls. The format for a Synapse global handler level external call log entry is as follows:

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
</tr>
</tbody>
</table>

**Example**

```
2018-11-28 10:10:56,316|a783f7c3-647f-4d10-9b72-106faa01bba8|PassThroughMessageProcessor-4|20|HTTP|admin--Pizza ShackAPI:v1.0.0|GET|/pizzashack/1.0.0/menu|pizzashack/1.0.0/menu|null|null|null|null|null|71|2238|20 0|DefaultApplication|Fslkdjfnlsdfjiefnlsdf|21
```

Click here for more details on the Synapse global handler level external call log entry.

The following is a detailed description of the Synapse global handler level external call log entry.

<table>
<thead>
<tr>
<th>timestamp</th>
<th>The time at which the log is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>2018-11-28 10:10:56,316</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>correlationID</th>
<th>Each log contains a correlation ID, which is unique to the HTTP request. A client can send a unique correlation ID in the header of the HTTP request. If this correlation ID is missing in the incoming request, WSO2 API-M will generate a random correlation ID for the request.</th>
<th>The HTTP header that carries the correlation ID is configured in WSO2 API-M.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a783f7c3-647f-4d10-9b72-106faa01bba8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>threadName</th>
<th>The identifier of the thread.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>PassThroughMessageProcessor-4</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>duration(BE latency)</td>
<td>The time gap (in milliseconds) between two states of the message.</td>
</tr>
<tr>
<td>callType</td>
<td>HTTP - The call type identifies logs that correspond to either back-end latency or round-trip latency states. Thereby, in the case of an individual request, one log will be recorded to identify back-end latency, and another log for the round-trip latency. These logs are categorized using the HTTP call type because these logs relate to HTTP calls between WSO2 API-M and external clients.</td>
</tr>
<tr>
<td>apiName</td>
<td>Name of the API that was invoked.</td>
</tr>
<tr>
<td>apiMethod</td>
<td>HTTP method utilized.</td>
</tr>
<tr>
<td>apiContext</td>
<td>The API context which was invoked.</td>
</tr>
<tr>
<td>apiResourcePath</td>
<td>Resource path of the API that was invoked.</td>
</tr>
<tr>
<td>authHeader</td>
<td>Logs the Authorization header.</td>
</tr>
<tr>
<td>orgIdHeader</td>
<td>Logs the organization-id header.</td>
</tr>
<tr>
<td>SrcIdHeader</td>
<td>Logs the source-id header.</td>
</tr>
<tr>
<td>applIdHeader</td>
<td>Logs the application-id header.</td>
</tr>
<tr>
<td>uuIdHeader</td>
<td>Logs the uuid header.</td>
</tr>
</tbody>
</table>
### requestSize
Size of the request payload.

**Example**

```
71
```

### responseSize
Size of the response payload.

**Example**

```
2238
```

### apiResponseStatuscode
Status code of the response.

**Example**

```
200
```

### applicationName
Name of the application that was used to subscribe to the API.

**Example**

```
DefaultApplication
```

### consumerKey
This refers to the consumer key that you get when you generate keys for your production and sandbox environments.

**Example**

```
Fslkdjfn1sdfjiefn1sdf
```

### responseTime
Roundtrip time of the request.

**Example**

```
21
```

---

#### Synapse passthrough transport level external call logs

In contrast to the information provided by the Synapse global handler level, the passthrough transport level gives certain additional data such as, the Synapse internal state of the request. The format for a Synapse passthrough transport level external call log entry is as follows:

**Format**

```
timestamp|correlationID|threadName|duration|callType|connectionName|methodType|connectionURL|httpStatus
```
Example - HTTP State Transition

<table>
<thead>
<tr>
<th>timestamp</th>
<th>The time at which the log is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>2018-11-28 10:10:56,314</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>correlationID</th>
<th>Each log contains a correlation ID, which is unique to the HTTP request. A client can send a unique correlation ID in the header of the HTTP request. If this correlation ID is missing in the incoming request, WSO2 API-M will generate a random correlation ID for the request. The HTTP header that carries the correlation ID is configured in WSO2 API-M.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>a783f7c3-647f-4d10-9b72-106faa01bba8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>threadName</th>
<th>The identifier of the thread.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>HTTPS-Sender I/O dispatcher-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>duration</th>
<th>The time gap (in milliseconds) between two states of the message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>callType</th>
<th>The following are the two possible call types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>This call type identifies logs that correspond to either back-end latency or round-trip latency states. Thereby, in the case of an individual request, one log will be recorded to identify back-end latency, and another log for the round-trip latency. These logs are categorized using the HTTP call type because these logs relate to HTTP calls between WSO2 API-M and external clients.</td>
</tr>
<tr>
<td>HTTP State Transition</td>
<td>This call type identifies logs that correspond to the state transition in the HTTP transport related to a particular message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>connectionName</th>
<th>This is a name that is generated to identify the connection between WSO2 API-M and the external client (back-end or message sender).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>http-outgoing-1</td>
</tr>
</tbody>
</table>
**methodType**

The HTTP method used for the request.

**Example**

GET

**connectionURL**

The connection URL of external client to which the message is passed from WSO2 API-M.

**Example**

https://localhost:9443/am/sample/pizzashack/v1/api/menu

**httpState**

Listed below are the state changes that a message goes through when it flows through WSO2 API-M, and when the message flows between WSO2 API-M and external clients. A new log is generated for the message to record each of the following states.

- **REQUEST_HEAD**: All HTTP headers in the incoming request are being written to the backend.
- **REQUEST_BODY**: The body of the incoming request is being written to the backend.
- **REQUEST_DONE**: The request is completely received (content decoded) and written to the backend.
- **BACKEND LATENCY**: The response message is received by WSO2 API-M. This status corresponds to the time total time taken by the backend to process the message.
- **RESPONSE_HEAD**: All HTTP headers in the response message are being written to the client.
- **RESPONSE_BODY**: The body of the response message is being written to the client.
- **RESPONSE_DONE**: The response is completely received and written to the client.
- **ROUND-TRIP LATENCY**: The response message is completely written to the client. This status corresponds to the total time taken by the HTTP request to complete the round trip (from the point of receiving the HTTP request from a client until the response message is sent back to the client).

---

**Database call logs**

The database call logging for observability includes two types of DB calls, namely LDAP calls and JDBC calls. This will help to track down any latencies caused by a database calls in an instance.

The format for a database call log entry is as follows:

- **JDBC call logs**

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
</tr>
</tbody>
</table>

**Example**

```
```

Click here for more details on the JDBC call log entry.

The following is a detailed description of the JDBC call log entry.
<table>
<thead>
<tr>
<th><strong>timestamp</strong></th>
<th>The time at which the log is created.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>2018-11-28 10:10:43,202</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>correlationID</strong></th>
<th>Each log contains a correlation ID, which is unique to the HTTP request. A client can send a unique correlation ID in the header of the HTTP request. If this correlation ID is missing in the incoming request, WSO2 API-M will generate a random correlation ID for the request. The HTTP header that carries the correlation ID is configured in WSO2 API-M.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>a783f7c3-647f-4d10-9b72-106faa01bba8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>threadName</strong></th>
<th>The identifier of the thread.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>PassThroughMessageProcessor-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>duration</strong></th>
<th>The time gap (in milliseconds) between two states of the message.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>callType</strong></th>
<th>jdbc - This indicates JDBC level logs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>startTime</strong></th>
<th>Time in milliseconds at which the query started.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>1543380043202</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>methodName</strong></th>
<th>SQL statement method type that was called.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>executeQuery</td>
</tr>
</tbody>
</table>
**query**  
**SQL query**

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT REG_NAME, REG_VALUE FROM REG_PROPERTY P, REG_RESOURCE_PROPERTY RP WHERE P.REG_ID=RP.REG_PROPERTY_ID AND RP.REG_VERSION=? AND P.REG_TENANT_ID=RP.REG_TENANT_ID AND RP.REG_TENANT_ID=?</td>
</tr>
</tbody>
</table>

**connectionUrl**  
**Database connection URL**

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdbc:h2:repository/database/WSO2CARBON_DB</td>
</tr>
</tbody>
</table>

- **LDAP call logs**

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
</tr>
<tr>
<td>providerUrl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-11-05T14:05:18.599</td>
</tr>
</tbody>
</table>

**Click here for more details on the LDAP call log entry.**

The following is a detailed description of the LDAP call log entry.

<table>
<thead>
<tr>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>The time at which the log is created.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-11-05T14:05:18,599</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>correlationID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each log contains a correlation ID, which is unique to the HTTP request. A client can send a unique correlation ID in the missing in the incoming request, WSO2 API-M will generate a random correlation ID for the request.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>86b56b19-7872-4e2f-84f3-5a14f92e18c1</td>
</tr>
<tr>
<td><strong>threadName</strong></td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td><strong>Example</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>duration</strong></th>
<th>The time gap (in milliseconds) between two states of the message.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>callType</strong></th>
<th>ldap - Determines the LDAP level logs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>startTime</strong></th>
<th>Time in milliseconds at which the query started.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>1541406918591</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>methodName</strong></th>
<th>LDAP method type that was called.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>search</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>providerUrl</strong></th>
<th>LDAP connection URL.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>ldap://localhost:10389</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>principal</strong></th>
<th>Login name of the user.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>uid=admin,ou=system</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>argsLength</strong></th>
<th>Length of arguments.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Using the correlation logs

Follow the instructions below to check the correlation logs:

**Step 1 - Setup WSO2 API-M**

Enable observability with WSO2 API-M and start the WSO2 API-M server.
For more information, see [Enabling observability with WSO2 API-M](#).

**Step 2 - Invoke an API**

Follow the instructions below to invoke an API.

1. Create and publish an API.  
   For more information, see [Create and Publish an API](#).
2. Subscribe to an API.  
   For more information, see steps 1 to 10 under [Subscribe to an API](#).
3. Invoke the API.
   The following is based on the PhoneVerification API.

   ```bash
   curl -k -H "Authorization :Bearer <access-token>" "activityid:<example-correlation-ID>" --data "PhoneNumber=<phone_number>&LicenseKey=<license_key>" <api_url>
   
   curl -k -H "Authorization :Bearer 3dfafa3a-b1e3-3550-8a25-88e4b4fe2fb3" "activityid:a783f7c3-647f-106fas01bba8" --data "PhoneNumber=18006785432&LicenseKey=0" https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber
   ```

**Step 3 - Check the correlation logs**

1. Open a terminal and navigate to the `<API-M_HOME>/repository/logs` directory where the `correlation.log` file is saved.
2. Isolate the logs that are correlated.
   Replace `<correlation_ID>` with the required value.

   ```bash
   cat correlation.log | grep "<correlation_ID>"
   ```

Reading and analyzing the correlation logs

Let's analyze the following sample correlation log.

```
2018-11-29 15:19:13,859|ff0c8866-d8a8-4189-930d-016b9d92f1e8|HTTP-Listener I/O dispatcher-2|1|HTTP State Transition|http-incoming-2|GET|/testing/1|REQUEST_HEAD
2018-11-29 15:19:13,859|ff0c8866-d8a8-4189-930d-016b9d92f1e8|HTTP-Listener I/O dispatcher-2|0|HTTP State Transition|http-incoming-2|GET|/testing/1|REQUEST_DONE
2018-11-29 15:19:13,862|ff0c8866-d8a8-4189-930d-016b9d92f1e8|PassThroughMessageProcessor-17|0|METHOD|org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler|handleRequest|messageContext]
```
<table>
<thead>
<tr>
<th>Line No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>HTTP State Transition when receiving the request</td>
</tr>
<tr>
<td>3-13</td>
<td>Methods invoked in Gateway handlers in the request path</td>
</tr>
<tr>
<td>14-15</td>
<td>Database calls relevant to the API call</td>
</tr>
<tr>
<td>16</td>
<td>HTTP State Transition for the request</td>
</tr>
<tr>
<td>17</td>
<td>Backend Latency</td>
</tr>
<tr>
<td></td>
<td>Here the backend latency log have reflected the 800ms delay that was added</td>
</tr>
<tr>
<td></td>
<td>to the backend for this example.</td>
</tr>
<tr>
<td>18-20</td>
<td>HTTP State Transition for response</td>
</tr>
<tr>
<td>21</td>
<td>Synapse global handler level for the backend call log</td>
</tr>
<tr>
<td>22-28</td>
<td>Methods invoked in the Gateway handlers in the response path</td>
</tr>
<tr>
<td>29-31</td>
<td>HTTP State Transition for dispatching response</td>
</tr>
<tr>
<td>32</td>
<td>HTTP Roundtrip Latency</td>
</tr>
</tbody>
</table>

Using these logs we can determine that the latency is caused by the backend call.

Advanced use cases

The following are the advanced use cases that you may run into when working with observability in WSO2 API-M.

- **Logging all methods**
- **Blacklisting threads**

**Logging all methods**

Currently, when using method logging, it only logs special methods that are suspected to give latencies, because logging all methods can pose performance issues. There can be instances where you may need to log other methods too.

Follow the instructions below to configure the logging of all methods.

**Linux/Mac OS Windows**

Add the following configuration as a system property to the `<API-M_HOME>/bin/wso2server.sh` file to log all methods of a given package.

**Format**

```
-DlogAllMethods=<package_name> 
```

For example, let's consider that you need to log all the methods for the gateway package.

**Example**

```
-DlogAllMethods=org.wso2.carbon.apimgt.gateway 
```

Make sure to add it before the `org.wso2.carbon.bootstrap.Bootstrap` $* line.

In a Windows environment, append the following system property to `CMD_LINE_ARGS` in the `<API-M_HOME>/bin\wso2server.bat` file to log all methods of a given package.
For example, let's consider that you need to log all the methods for the gateway package.

Example

```
-DlogAllMethods=org.wso2.carbon.apimgt.gateway
```

Blacklisting threads

Blacklisting of threads is needed because some threads keep on printing unnecessary logs continuously. Therefore, by blacklisting unwanted threads from printing logs it helps in terms of readability of the logs.

Follow the instructions below to enable blacklisting of threads:

**Linux/Mac OS**

Add the following configuration as a system property to the `<API-M_HOME>/bin/wso2server.sh` file to stop printing logs for JDBC calls from the threads that starts with the names that you defined.

Format

```
-Dorg.wso2.CorrelationLogInterceptor.BlacklistedThreads=<threadName1>,<threadName2> \
```

Example

```
-Dorg.wso2.CorrelationLogInterceptor.BlacklistedThreads=pool-10-thread-1,metrics-jdbc-reporter-1-thread-1 \
```

In a Windows environment, append the following system property to `CMD_LINE_ARGS` in the `<API-M_HOME>/bin/wso2server.bat` file to stop printing logs for JDBC calls from the threads that starts with the names that you defined.

Format

```
-Dorg.wso2.CorrelationLogInterceptor.BlacklistedThreads=<threadName1>,<threadName2>
```

Example

```
```

If the above configuration is not added, by default the `MessageDeliveryTaskThreadPool` thread will be blacklisted as it is found to print a considerable amount for messages for API-M instances. However, if the above configuration is added the default value will be overridden.

Blacklisting of threads is not needed by default, as all unnecessary threads are already blacklisted.
Configuring the API Manager

This section explains how to configure the API Manager:

- Adding Custom Properties to APIs
- Adding Internationalization and Localization
- Adding Links to Navigate Between the Store and Publisher
- Changing the Default Transport
- Configuring API Monetization Category Labels
- Configuring Caching
- Configuring Keystores in WSO2 API Manager
- Configuring Multiple Tenants
- Configuring Single Sign-on
- Configuring Transports
- Configuring User Stores
- Customizing the API Store
- Directing the Root Context to the API Store
- Enabling Access Control Support for API Publisher
- Enabling CORS for APIs
- Enabling Notifications
- Generating SDKs
- Logging
- Maintaining Separate Production and Sandbox Gateways
- Managing Users and Roles
- Message Tracing
- Migrating the APIs and Applications to a Different Environment
- Performance Tuning and Testing Results
- Prevent API Suspension
- Removing Unused Tokens from the Database
- Revoke OAuth2 Application
- Sharing Applications Between Multiple Groups
- Transforming API Message Payload
- Whitelisting Host Names for API Store
- Working with Access Tokens
- Working with Databases
- Changing the Hostname
- Whitelisting Hostnames for API Store

Adding Custom Properties to APIs

Usually, APIs have a pre-defined set of properties such as the name, version, context, etc. However, there may be instances where you want to add specific custom properties to your API. You can do this in either of the following ways:

- Add custom properties via the API Publisher
- Add custom properties via the REST API

When adding custom properties, note the following:

- Property name should be unique.
- Property name should not contain spaces.
- Property name cannot be case-sensitive.
- Property name cannot be any of the following as they are reserved keywords: provider, version, context, status, description, subcontext, doc, lcState, name, tags.

After the custom properties have been added, you can search for APIs using custom property values.

Add custom properties via the API Publisher

1. Sign in to the API Publisher as an API creator using the following URL: https://<localhost>:9443/publisher.
2. Create a new API or edit an existing API.
3. In the Manage tab, expand the API Properties area, enter a custom property name and value (e.g. property name: environment, property value: preprod) and click the plus (+) sign to add it.
Add custom properties via the REST API

You can use the existing REST API to add a new API with custom properties. Add the following element to the request body including the relevant properties,

```
"additionalProperties" : {
  "environment": "preprod",
  "secured": "true"
}
```

Search using custom properties

You can use the following format to search for an API using the custom properties:

```
<property_name>:<property_value>
```

For example, if you want to search for the `environment` property with a specific value (e.g., preprod) in the Publisher, you can search as shown below:
When you click on the name of the API in the above screen, you are taken to the respective API Overview page that has details with regard to the API properties that you added.
Adding Internationalization and Localization

The API Manager comes with two Web interfaces as API Publisher and API Store. The following steps show an example of how to localize the API Publisher UI. Same instructions apply to localize the API Store.

Changing the browser settings

1. Follow the instructions in your Web browser's user guide and set the browser's language to a preferred one. For example, in Google Chrome, you set the language using the Settings -> Show advanced settings -> Languages menu.
2. Set the browser's encoding type to UTF-8.

Introduction to resource files

1. Go to `<APIM_HOME>/repository/deployment/server/jaggeryapps/publisher` directory where `<APIM_HOME>` is the API Manager distribution's home.
2. There are two types of resource files used to define localization strings in the API Manager.
   - The resource file used to store the strings defined in .jag files according to browser locale (For example, `locale_en.json`) is located in `.../publisher/site/conf/locales/jaggery` folder.
   - The resource file `i18nResources.json`, which is used to store strings defined in client-side javascript files such as pop-up messages when a UI event is triggered, is located in `.../publisher/site/conf/locales/js` folder.

For example,
To implement localization support for jaggery, we use its in-built script module 'i18n'. For more information, refer to `http://jaggeryjs.org/documentation.jag?api=i18n`.

**Localizing strings in Jaggery files**

3. To localize the API publisher to Spanish, first localize the strings defined in jaggery files. Create a new file by the name `locale_{localeCode}.json` inside `...publisher/site/conf/locales/jaggery` folder. For example, if the language set in the browser is Spanish, the locale code is `es` and the file name should be `locale_es.json`.

```
The file name which includes the locale code will differ according to the language selected in your browser. Create a new file for the language you select, if the selected language is not available. The file Name should be `locale_{localeCode}.json` where the `localeCode` refers to the sub tag of the particular language. Please refer the IANA Language Subtag Registry page for a list of sub tags.

By matching the accept Language header, the particular file will be selected from the resource file directory of each web application for the localization by the server. If there is no locale file found with the matching sub tag, `locale_default.json` file will be served for the localization.
```

Add the key-value pairs to `locale_es.json` file. For an example on adding key value pairs, refer to `locale_en.json` in `...publisher/site/conf/locales/jaggery` folder. It is the default resource file for jaggery.

In addition, a section of a sample `locale_es.json` file is shown below for your reference.

```json
{
    "name" : "Nombre",
    "context" : "Contexto",
    "version" : "Versión",
    "description" : "Descripción",
    "visibility" : "Visibilidad",
    "thumbnail" : "Uña del pulgar",
    "endpoint" : "Producción URL",
    "sandbox" : "Cajón de arena URL",
}
```

**Localizing strings in client-side Javascript files**

1. To localize the javascript UI messages, navigate to `publisher/site/conf/locales/js` folder and update `i18nResources.json` file with relevant values for the key strings.

2. Once done, open the API Publisher web application in your browser (`https://<YourHostName>:9443/publisher`).

3. Note that the UI is now changed to Spanish.

**Adding Links to Navigate Between the Store and Publisher**

By default, there are no links in the UIs of the API Store and API Publisher applications to traverse between the two apps.

To add a link in the WSO2 API Publisher to the WSO2 API Store:

1. Set the `<DisplayURL>` to `true`, and provide the URL of the Store in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

   If you are using a distributed/clustered API Manager setup, this configuration must be done in the API Publisher node/s.

   **Example**

   ```xml
   <APIStore>
       <DisplayURL>true</DisplayURL>
       <URL>https://<hostname>:9443/store</URL>
   </APIStore>
   ```

   • `<hostname>` - The hostname of the API Publisher node.
2. Restart the WSO2 API-M server.
   Note that a URL that points to the API Store appears on the top, right-hand corner of the WSO2 API Publisher.
   Example:

![WSO2 API Publisher](image)

- For information on clustering, see Clustering WSO2 API Manager.
- For information on deployment patterns, see Deployment Patterns of WSO2 API Manager.

## Changing the Default Transport

On the back end, APIs are Apache Synapse configurations that WSO2 API Manager accesses through a transport. The default API Manager transport is the PassThrough transport, but you can configure a different default transport in your `axis2.xml` file. For example, to use the HTTP-NIO transport as the default, go to the `<APIM_HOME>/repository/conf/axis2` folder, open the `axis2.xml` file, and then in the "Transport Ins" and "Transport Outs" sections, comment out the PassThrough configurations and uncomment the configurations for the HTTP-NIO transport.

WSO2 products do not use the HTTP/S servlet transport configurations that are in `axis2.xml` file. Instead, they use Tomcat-level servlet transports, which are used by the management console in `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.

The following topics provide more information on these transports:
- HTTP PassThrough transport
- HTTP-NIO transport
- Transport receiver parameters
- Transport sender parameters
- Connection throttling

### HTTP PassThrough transport

HTTP PassThrough Transport is the default, non-blocking HTTP transport implementation based on HTTP Core NIO and is specially designed for streaming messages. It is similar to the old message relay transport, but it does not care about the content type and simply streams all received messages through. It also has a simpler and cleaner model for forwarding messages back and forth. The two classes that implement the receiver and sender APIs are `org.apache.synapse.transport.passthru.PassThroughHttpListener` and `org.apache.synapse.transport.passthru.PassThroughHttpSender`, respectively. The PassThrough Transport does not require the binary relay builder and expanding formatter.

### HTTP-NIO transport

The HTTP-NIO transport is a module of the Apache Synapse project. Apache Synapse ships the HTTP-NIO transport as the default, non-blocking HTTP transport implementation. The two classes that implement the receiver and sender APIs are `org.apache.synapse.transport.http.HttpCoreNIOListener` and `org.apache.synapse.transport.http.HttpCoreNIOSender`, respectively. These classes are available in the JAR file named `synapse-http-transport.jar`. The transport implementation is based on Apache HTTP Core - NIO and uses a configurable pool of non-blocking worker threads to grab incoming HTTP messages off the wire. The PassThrough transport is the preferred default transport for WSO2 API Manager, but HTTP-NIO is supported for backward compatibility.

### Transport receiver parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Required</td>
<td>Possible Values</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>port</td>
<td>The port on which this transport receiver should listen for incoming messages.</td>
<td>No</td>
<td>A positive integer less than 65535</td>
<td>8280</td>
</tr>
<tr>
<td>non-blocking</td>
<td>Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.</td>
<td>Yes</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>bind-address</td>
<td>The address of the interface to which the transport listener should bind.</td>
<td>No</td>
<td>A host name or an IP address</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>WSDL_EPR_Prefix</td>
<td>A URL prefix which will be added to all service EPRs and EPRs in WSDLs etc.</td>
<td>No</td>
<td>A URL of the form &lt;protocol&gt;://&lt;hostname&gt;:&lt;port&gt;/</td>
<td></td>
</tr>
<tr>
<td>httpGetProcessor</td>
<td>An extension point used to execute a special interceptor for HTTP GET requests.</td>
<td>Yes</td>
<td>An extension point</td>
<td>org.wso2.carbon.mediation.transport.handlers.PassThroughNHGetProcessor</td>
</tr>
<tr>
<td>priorityConfigFile</td>
<td>The location of the file containing the configuration for priority based dispatching.</td>
<td>No</td>
<td>A file location</td>
<td></td>
</tr>
</tbody>
</table>

**Transport sender parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-blocking</td>
<td>Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.</td>
<td>Yes</td>
<td>true or false</td>
<td>true</td>
</tr>
<tr>
<td>warnOnHTTP500</td>
<td>Logs warnings for HTTP 500 responses only for the specified content-types. For example, `&lt;parameter name=&quot;warnOnHTTP500 locked=&quot;false&quot;&gt;x-application/hessian</td>
<td>none&lt;/parameter&gt;` would log warnings for HTTP 500 responses of content-type 'x-application/hessian' or messages missing a content-type.</td>
<td>No</td>
<td>A list of content types separated by &quot;</td>
</tr>
<tr>
<td>http.proxyHost</td>
<td>If the outgoing messages should be sent through an HTTP proxy server, use this parameter to specify the target proxy.</td>
<td>No</td>
<td>A host name or an IP address</td>
<td></td>
</tr>
<tr>
<td>http.proxyPort</td>
<td>The port through which the target proxy accepts HTTP traffic.</td>
<td>No</td>
<td>A positive integer less than 65535</td>
<td></td>
</tr>
<tr>
<td>http.nonProxyHosts</td>
<td>The list of hosts to which the HTTP traffic should be sent directly without going through the proxy.</td>
<td>No</td>
<td>A list of host names or IP addresses separated by &quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

**Connection throttling**

With the HTTP PassThrough and HTTP NIO transports, you can enable connection throttling to restrict the number of simultaneous open connections. To enable connection throttling, edit the `<PRODUCT_HOME>/repository/conf/nhttp.properties` (for the HTTP NIO transport) or `<PRODUCT_HOME>/repository/conf/passthru.properties` (for the PassThrough transport) and add the following line: `max_open_connections = 2`

This will restrict simultaneous open incoming connections to 2. To disable throttling, delete the `max_open_connections` setting or set it to `-1`.

Connection throttling is never exact. For example, setting this property to 2 will result in roughly two simultaneous open connections at any given time.

**Configuring API Monetization Category Labels**
When defining throttling tiers using the Admin Portal, you have the option to specify a given billing plan for tiers. A tier is defined as either a free or paid tier. Depending on the tiers available for a given API, the following API monetization categories are displayed as labels in the store.

- **Free** - If all subscription tiers are defined as Free, the API uses the **Free billing plan** and the API is labeled as Free in the Store.
- **Paid** - If all subscription tiers are defined as Paid, the API uses the **Commercial billing plan** and the API is labeled as Paid in the Store.
- **Freemium** - If the API has a combination of Free and Paid subscription tiers, the API uses the **Freemium billing plan** and the API is labeled as Freemium in the Store.

Follow the configuration steps below to enable API monetization category labels:

2. Navigate to the **Main** menu, and click **Browse**, which is under the **Resources** tab.
3. Enter the following path in the **Location** text-box and click **Go**.
   
   `/_system/config/apimgt/applicationdata/tenant-conf.json`

4. In the **Contents** panel, click the **Edit as text** link and the `tenant-conf.json` file opens.
5. To enable monetization categories for APIs, set the `EnableMonetization` property to true. By default, it is set to false.
6. Define the subscription tiers as required.
   
   For example if you are working with the unlimited tier,
   
   - To define the unlimited tier as **paid**, set the `IsUnlimitedTierPaid` property to true.
   - To define the unlimited tier as **free**, set the `IsUnlimitedTierPaid` property to false.

   As Freemium APIs has a combination of paid and free subscription tiers, the configuration involved in defining the subscription tiers will be the same as above. However, Freemium APIs need to have a minimum of one subscription defined as paid and free.

7. After the edits, click **Save Content**.

Note that the above configuration can be done independently on a per tenant basis.

When the above `EnableMonetization` property is set to true for the respective tenant, the API monetization category labels are displayed in the tenant API store.
Configuring Caching

When an API call hits the API Gateway, the Gateway carries out security checks to verify if the token is valid. During these verifications, the API Gateway extracts parameters (i.e., access token, API name, and API version) that are passed on to it. Since the entire load of traffic to APIs goes through the API Gateway, this verification process needs to be fast and efficient in order to prevent overhead and delays. WSO2 API Manager uses caching for this purpose, where the validation information is cached with the token, API name, and version, and the cache is stored in either the API Gateway or the Key Manager server.

The default cache size of any type of cache in a WSO2 product is 10,000 elements/records. Cache eviction occurs from the 10001st element. All caches in WSO2 products can be configured using the `<PRODUCT_HOME>/repository/conf/carbon.xml` file. The default `CacheTimeout` is 15 minutes which comes by default.

```
<Cache>
  <!-- Default cache timeout in minutes -->
  <DefaultCacheTimeout>15</DefaultCacheTimeout>
</Cache>
```

These configurations apply globally to all caches. You can override these values for specific caches using the UI or different configuration files as discussed under each section below.

This section covers the following:
- API Gateway cache
- Resource cache
- Key Manager cache
- Response cache
- API Store cache

In a distributed environment, the caching configurations you do in one node replicates equally in all nodes.

Apart from response caching, all the other caches are enabled by product. When the WSO2 API Manager components are clustered, they work as distributed caches. This means that a change done by one node is visible to another node in the cluster.
API Gateway cache

When caching is enabled at the Gateway and a request hits the Gateway, it first populates the cached entry for a given token. If a cache entry does not exist in the cache, it calls the Key Manager server. This process is carried out using Web service calls. After the Key Manager server returns the validation information, it gets stored in the Gateway. As the API Gateway issues a Web service call to the Key Manager server only, if it does not have a cache entry, this method reduces the number of Web service calls to the Key Manager server. Therefore, it is faster than the alternative method.

By default, the API Gateway cache is enabled because the `<EnableGatewayTokenCache>` element is set to true in the `<API-M_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayTokenCache>true</EnableGatewayTokenCache>
```

If you need to enable Gateway caching across the entire cluster, see Working with Hazelcast Clustering.

Clearing the API Gateway cache

If you wish to remove old tokens that might still remain active in the Gateway cache, you need to configure the `<RevokeAPIURL>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file by providing the URL of the Revoke API that is deployed in the API Gateway node. The revoke API invokes the cache clear handler, which extracts information form transport headers of the revoke request and clears all associated cache entries. If there’s a cluster of API Gateways in your setup, provide the URL of the revoke API deployed in one node in the cluster. This way, all revoke requests route to the OAuth service through the Revoke API.

Given below is how to configure this in a distributed API Manager setup.

1. In the `api-manager.xml` file of the **API Store node**, point the revoke endpoint as follows:

   ```
   <RevokeAPIURL>https://${carbon.local.ip}:${https.nio.port}/revoke</RevokeAPIURL>
   ```

2. In the API Gateway, point the Revoke API to the OAuth application deployed in the key manager node. For example,

   ```
   <api name="_WSO2AMRevokeAPI_" context="/revoke">
    <resource methods="POST" url-mapping="/" faultSequence="_token_fault_">
     <inSequence>
      <send>
       <endpoint>
        <address uri="https://keymgmt.wso2.com:9445/oauth2/revoke"/>
       </endpoint>
        </send>
      </inSequence>
    </resource>
    <handlers>
     <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerCacheExtensionHandler"/>
    </handlers>
   </api>
   ```

Resource cache

An API’s resources are HTTP methods that handle particular types of requests such as GET, POST etc. They are similar to methods of a particular class. Each resource has parameters such as its throttling level, Auth type etc.
Users can make requests to an API by calling any one of the HTTP methods of the API's resources. The API Manager uses the resource cache at the Gateway node to store the API's resource-level parameters (Auth type and throttling level). The cache entry is identified by a cache key, which is based on the API's context, version, request path and HTTP method. Caching avoids the need to do a separate back-end call to check the Auth type and throttling level of a resource, every time a request to the API comes. It improves performance.

Note that if you update an API, the resource cache gets invalidated and the changes are reflected within a few minutes.

By default, the resource cache is enabled as the `<EnableGatewayResourceCache>` element is set to true in the `<APIM_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayResourceCache>true</EnableGatewayResourceCache>
```

Key Manager cache

The following caches are available:

- Key cache
- OAuth cache

Key cache

In a typical API Manager deployment, the Gateway is deployed in a DMZ while the Key Manager is in MZ. By default, caching is enabled at the Gateway. To avoid caching token-related information in a leniently secured zone, you can store the cache on the Key Manager side. If you do, for each and every API call that hits the API Gateway, the Gateway issues a Web service call to the Key Manager server. If the cache entry is available in the Key Manager server, it is returned to the Gateway. Else, the database is checked for the validity of the token.

Storing the cache in the Key Manager causes lower performance than when storing it in the Gateway, but it is more secure. If you enable the key cache in a clustered environment, you should have only one Gateway per Key Manager, whereas you can have two Gateways per Key Manager when the Gateway cache is enabled instead. Note that you should always have one of the caches enabled, but we do not recommend using both caches combined. For more information, see Clustering Gateways and Key Managers with key caching in the WSO2 Clustering Guide.

You configure the key cache by editing the following elements in the `<APIM_HOME>/repository/conf/api-manager.xml` file:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Configuration Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable the API Gateway cache.</td>
<td>Under <code>&lt;CacheConfigurations&gt;</code>, set <code>&lt;EnableGatewayTokenCache&gt;false&lt;/EnableGatewayTokenCache&gt;</code></td>
</tr>
<tr>
<td>Enable the Key Manager cache.</td>
<td>Under <code>&lt;CacheConfigurations&gt;</code>, set <code>&lt;EnableKeyManagerTokenCache&gt;true&lt;/EnableKeyManagerTokenCache&gt;</code></td>
</tr>
<tr>
<td>Change the key cache duration, which expires after 900 seconds by default.</td>
<td><code>&lt;Token Cache Expiry&gt;900&lt;/Token Cache Expiry&gt;</code></td>
</tr>
</tbody>
</table>

OAuth cache

The OAuth token is saved in this cache, which is enabled by default. Whenever a new OAuth token is generated, it is saved in this cache to prevent constant database calls. Unless an OAuth expires or is revoked, the same token is sent back for the same user. Therefore, you do not need to change this cached token most of the time.

Response cache

The API Manager uses WSO2 ESB's cache mediator to cache response messages for each API. Caching improves performance, because the backend server does not have to process the same data for a request multiple times. You need to set an appropriate timeout period to offset the risk of stale data in
You need to enable response caching when creating a new API or editing an existing API using the API Publisher. Go to the API Publisher and click **Add New API** (to create a new API) or click the **Edit** icon associated with an existing API. Then, navigate to the **Manage** tab where you find the response caching section. You can set Response caching to **Enabled** and give a timeout value. This enables the default response caching settings.

To change the default response caching settings, edit the following cache mediator properties in the `<API-M_HOME>/repository/resources/api_templates/velocity_template.xml` file:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collector</td>
<td>• true: specifies that the mediator instance is a response collection instance.</td>
</tr>
<tr>
<td></td>
<td>• false: specifies that the mediator instance is a cache serving instance.</td>
</tr>
<tr>
<td>max MessageSize</td>
<td>Specifies the maximum size of a message to be cached in bytes. An optional attribute, with the default value set to <strong>unlimited</strong>.</td>
</tr>
<tr>
<td>maxSize</td>
<td>Defines the maximum number of elements to be cached.</td>
</tr>
<tr>
<td>hashGenerator</td>
<td>Defines the hash generator class.</td>
</tr>
<tr>
<td></td>
<td>When caching response messages, a hash value is generated based on the request's URI, transport headers and the payload (if available). WSO2 has a default REQUESTHASHGenerator class written to generate the hash value. See sample here.</td>
</tr>
<tr>
<td></td>
<td>If you want to change this default implementation (for example, to exclude certain headers), you can write a new hash generator implementation by extending the REQUESTHASHGenerator and overriding its <code>getDigest()</code> method. Once done, add the new class as the <code>hashGenerator</code> attribute of the <code>&lt;cache&gt;</code> element in the <code>velocity_template.xml</code> file.</td>
</tr>
</tbody>
</table>

When running a distributed deployment, you need to enable the stream builders on the API Gateway and maintain the standard builders on the API Store.

Follow the instructions below to enable the stream builders in the API gateway:

1. Open the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file.
2. Disable the default standard builders by commenting the following.

```
<!--messageBuilder contentType="application/json"
class="org.apache.synapse.commons.json.JsonBuilder"/>
```

3. Enable the stream builders by uncommenting the following.

```
<messageBuilder contentType="application/json"
class="org.apache.synapse.commons.json.JsonStreamBuilder"/>
```

Invalidating Cached Responses Remotely

You can invalidate all cached response remotely by using any JMX monitoring tool such as Jconsole using the exposed MBeans. You can use the `Invalidating cached datemediaticache()` operation of the `org.wso2.carbon.mediation` MBean for this as shown below.
**API Store cache**

The API Store has several caches to reduce the page-load times and increase its responsiveness when multiple users access it simultaneously.

- **Tag cache**: This cache saves the API's tags after they have been retrieved from registry. If your APIs and associated tags change frequently, it is recommended to configure a smaller cache refresh time (in milliseconds). This cache is disabled by default. To enable it, uncomment the `<TagCacheEnabledDuration>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

- **Recently-added-API cache**: This cache saves the five most recently added APIs. It is disabled by default. If you have multiple API modifications during a short time period, it is recommended to not enable this cache. To enable it, set the `<EnableRecentlyAddedAPICache>` to `true` in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

**Configuring Keystores in WSO2 API Manager**

WSO2 products use asymmetric encryption by default for the purposes of authentication and data encryption. In asymmetric encryption, keystores (with key pairs and certificates) are created and stored for the product. It is possible to have multiple keystores so that the keys used for different use cases are kept unique. For more information about creating and configuring keystores, see [Using Asymmetric Encryption](#).

After you have created a new keystore and updated the `client-truststore.jks` file, you must update a few configuration files in order to make the keystore work.

For instructions on the default carbon keystore configurations, see Configuring Keystores in WSO2 Products in the WSO2 Product Administration Guide.

The following common configurations should be completed before you begin the configurations given in this page.

- Configuring the primary keystore
- Configuring a keystore for SSL connections
Make sure you do the configurations below to configure a keystore in WSO2 API Manager.

- Configuring keystores for AMQP and MQTT transports
- Configuring keystores for Jaggery Apps SSO configuration
- Configuring keystores for security
- Configuring keystores for endpoints
- Configuring keystores for advanced transport handling
- Configuring keystores for Analytics

### Configuring keystores for AMQP and MQTT transports

To configure AMQP and MQTT transports, open the `<API-M_HOME>/repository/conf/broker.xml` file. The values for the `location` and `password` parameters under `keyStore` and `trustStore` must be updated. The code below shows the default values.

```xml
<sslConnection enabled="true" port="8672">
  <keyStore>
    <location>repository/resources/security/wso2carbon.jks</location>
    <password>wso2carbon</password>
  </keyStore>
  <trustStore>
    <location>repository/resources/security/client-truststore.jks</location>
    <password>wso2carbon</password>
  </trustStore>
</sslConnection>
```

### Configuring keystores for Jaggery Apps SSO configuration

Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` file. Update the values for `keyStorePassword` as shown below.

```json
"ssoConfiguration" : {
  "enabled" : "true",
  "issuer" : "API_PUBLISHER",
  "identityProviderURL" : "https://localhost:9444/samlsso",
  "keyStorePassword" : "wso2carbon",
  "identityAlias" : "wso2carbon",
  "responseSigningEnabled":"true",
  "assertionSigningEnabled":"true",
  "keyStoreName" :"wso2carbon.jks",
},
```

### Configuring keystores for security

Open the `<API-M_HOME>/repository/conf/identity/identity.xml` file and update the values for `Location` and `Password` under the `KeyStore` section. The default configurations are shown below.

```xml
<EntitlementSettings>
  <ThriftBasedEntitlementConfig>
    <EnableThriftService>false</EnableThriftService>
    <ReceivePort>${Ports.ThriftEntitlementReceivePort}</ReceivePort>
    <ClientTimeout>10000</ClientTimeout>
    <KeyStore>
      <Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
      <Password>wso2carbon</Password>
    </KeyStore>
    <ThriftHostName>${carbon.host}</ThriftHostName>
  </ThriftBasedEntitlementConfig>
</EntitlementSettings>
```
Configuring keystores for endpoints

Open the `<API-M_HOME>/repository/conf/identity/EndpointConfig.properties` file and update `client.keyStore` and `client.trustStore` with the location of the keystore and truststore respectively. The default configurations are shown below.

```properties
tenantListEnabled=false
hostname.verification.enabled=true
mutual.ssl.username=admin
client.keyStore=./repository/resources/security/wso2carbon.jks
client.trustStore=./repository/resources/security/client-truststore.jks
#identity.server.serviceURL=https://localhost:9443/services/
username.header=UserName
key.manager.type=SunX509
trust.manager.type=SunX509
tls.protocol=TLSv1.2
```

Configuring keystores for advanced transport handling

To have more advanced transport handling functions using keystores, you must update the `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` and the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file.

Configuring keystores for Analytics

Open the `<API-M_HOME>/repository/conf/data-bridge/data-bridge-config.xml` file and update `keyStoreLocation` and `keyStorePassword` with the location of the keystore and its password respectively. The default configurations are shown below.

```xml
<keyStoreLocation>${carbon.home}/repository/resources/security/wso2carbon.jks</keyStoreLocation>
<keyStorePassword>wso2carbon</keyStorePassword>
```

The `<API-M_HOME>/repository/conf/data-bridge/data-agent-config.xml` file is used by the publishing client. Therefore, a trustore with the public cert of the server is required here. The `<API-M_HOME>/repository/conf/data-bridge/data-bridge-config.xml` file is used by the listening server. This needs to include a keystore with the public and private certs to support SSL.

Configuring Multiple Tenants

The goal of multi-tenancy is to maximize resource sharing by allowing multiple users (tenants) to log in and use a single server/cluster at the same time, in a tenant-isolated manner. That is, each user is given the experience of using his/her own server, rather than a shared environment. Multi-tenancy ensures optimal performance of the system's resources such as memory and hardware and also secures each tenant's personal data.

You can register tenant domains using the Management Console of WSO2 products.

For common instructions on configuring multiple tenants for your WSO2 server, see Working with Multiple Tenants in the WSO2 Administration Guide and to understand how multi-tenancy works with the API Store, see Managing Tenants.

Managing Tenants

You can add a new tenant in the management console and then view it by following the procedure below. In order to add a new tenant, you should be logged in as a super user.

1. Click Add New Tenant in the Configure tab of the management console.
2. Enter the tenant information in **Register A New Organization** screen as follows, and click **Save**.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>The domain name for the organization, which should be unique (e.g., abc.com). This is used as a unique identifier for your domain. You can use it to log into the admin console to be redirected to your specific tenant. The domain is also used in URLs to distinguish one tenant from another.</td>
</tr>
<tr>
<td>Select Usage Plan for Tenant</td>
<td>The usage plan defines limitations (such as number of users, bandwidth, etc.) for the tenant. For on-premises deployment, there is only one default plan, i.e., Demo.</td>
</tr>
<tr>
<td>First Name/Last Name</td>
<td>The name of the tenant admin.</td>
</tr>
<tr>
<td>Admin Username</td>
<td>The login username of the tenant admin. The username always ends with the domain name (e.g., <a href="mailto:admin@abc.com">admin@abc.com</a>)</td>
</tr>
<tr>
<td>Admin Password</td>
<td>The password used to log in using the admin username specified.</td>
</tr>
<tr>
<td>Admin Password (Repeat)</td>
<td>Repeat the password to confirm.</td>
</tr>
<tr>
<td>Email</td>
<td>The email address of the admin.</td>
</tr>
</tbody>
</table>

Note that all the above parameters are required parameters.

3. After saving, the newly added tenant appears in the **Tenants List** page as shown below. Click **View Tenants** in the **Configure** tab of the management console to see information of all the tenants that currently exist in the system. Enter the domain name in the **Enter the Tenant Domain** parameter and click **Find** to find the newly added tenant in the list.

When you create multiple tenants in an API Manager deployment, the API Stores of each tenant are displayed in a multi-tenanted view for all users to browse and permitted users to subscribe to as shown below:

1. Access the API Store URL (by default, `https://localhost:9443/store`) using a Web browser. You see the storefronts of all the registered tenant domains listed there. For example,
This is called the public store. Each icon here is linked to the API Store of a registered tenant, including the super tenant, which is carbon.super. That is, the super tenant is also considered a tenant.

2. Click the Visit Store link associated with a given store to open it.

3. Anonymous users can browse all stores and all public APIs that are published to them. However, in order to subscribe to an API, the user must log in.

For example, if you are a user in the domain1.com tenant domain,

- You can access the public store (https://localhost:9443/store), go to the domain1.com store, log in to it and subscribe to its APIs.
- You can also browse the other tenant stores listed in the public store. But, within other tenant stores, you can only subscribe to the APIs to which your tenant domain is permitted to subscribe to. At the time an API is created, the API creator can specify which tenants are allowed to subscribe to the API. For information, see API Subscriptions.

Other tenant management operations such as activating, deactivating, updating and deleting, which are not available in the Management Console UI, can be done through the RemoteTenantManager Admin Service. You can invoke these operations using a SOAP client like SOAP UI. Follow the steps below to do the configurations using SOAP UI.

i. Open the <API-M_HOME>/repository/conf/carbon.xml file and set HideAdminServiceWSDLs parameter to false.

ii. Start SOAP UI client, and import the WSDL https://localhost:9443/services/RemoteTenantManagerService?wsdl. This assumes that you are running the SOAP UI client from the same machine as the API Manager instance.

iii. Note that there are several operations shown in the SOAP UI after importing the wsdl file:

   - Click on each operation to open the request view. For an example, for activateTenant operation, you can see the following request view:
3. Configuring Single Sign-on

Single Sign-On (SSO) allows users, who are authenticated against one application, to gain access to multiple other related applications without having to repeatedly authenticate themselves. It also allows the web applications to gain access to a set of back-end services with the logged-in user’s access rights, and the back-end services can authorize the user based on different claims like the user role. In WSO2 API Manager, you can enable SSO by using either SAML2 or OAuth2 OpenID Connect (OIDC).

The following sections explain more details on how you can use SSO in WSO2 API Manager.

- Configuring Single Sign-on with SAML2
- Configuring Single Sign-on with OpenID Connect

Configuring Single Sign-on with SAML2

SAML stands for Security Assertion Markup Language, which is an XML based data format for exchanging authentication and authorization data between an identity provider and a service provider. The single most important requirement that SAML addresses is web browser single sign-on (SSO). Three main roles are defined in SAML Specification.

- The Principal: This is typically the user who tries to access a protected resource or service of a service provider.
- The Identity Provider: An Identity Provider (IdP) is responsible for authenticating users and issuing assertions which include authentication/authorization decisions and user attributes.
- The Service Provider: A Service Provider (SP) consumes the assertions issued by Identity Provider and provides services to the principals.

The main use case scenario covered by SAML is the Principal (the user) requesting access to resource or service from the Service Provider. Then the Service Provider, using SAML, communicates with the Identity Provider to obtain identity assertion. The Service Provider makes the access control decision, depending on this assertion.

SAML 2.0 is the latest version of SAML, which uses security tokens containing assertions to pass information about a user between an identity provider and a service provider.

This section covers the following topics.

- Configuring API Manager for SSO
- Configuring External IDP through Identity Server for SSO
- Configuring Identity Server as IDP for SSO

For more information on SAML related terminologies discussed in the sections above, go to Assertions and Protocols for the OASIS SAML 2.0 documentation.

Configuring API Manager for SSO

You can configure the API Manager for SAML SSO by following the instructions below.
**Step 1 - Configure the Carbon Console for SSO**

This step is done in order to have SSO between the API Manager's and Identity Server's management consoles.

Open the `<API-M_HOME>/repository/conf/security/authenticators.xml` file and give the configurations as shown below.

- Set `disabled` attribute in the `<Authenticator>` element to `false`.
- `ServiceProviderID`: The issuer name of the service provider.
- `IdentityProviderSSOServiceURL`: The URL of the IDP. In this example, it is the URL of the Identity Server.

```
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
  <Priority>10</Priority>
  <Config>
    <Parameter name="LoginPage">/carbon/admin/login.jsp</Parameter>
    <Parameter name="ServiceProviderID">carbonserver</Parameter>
    <Parameter name="IdentityProviderSSOServiceURL">https://localhost:9444/samlsso</Parameter>
    <Parameter name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
  </Config>
</Authenticator>
```

Make sure the `<priority>` element of the `SAML2SSOAuthenticator` is less than that of the `BasicAuthenticator` handler. For more information see the FAQ.

**NameIDPolicyFormat**

Service provider and Identity Provider usually communicate with each other about a subject. That subject should be identified through NAME-ID. It should be in some format so that it is easy for the other party to identify it based on the format. The possible values for the NameIDPolicyFormat are as follows:

- `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`
- `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`

In this scenario use `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`, which is the default NameIDPolicyFormat. However, the value of the NameIDPolicyFormat is totally based on the entities that you wish to implement. For more information on NameIdentifiers, refer to section 8.3 in SAML Core.

If there are many WSO2 products in your environment, you can configure SSO for the management consoles to gain one-time access to all of the consoles without repeated authentication. You can do this by changing the `SAML2SSOAuthenticator` configuration in the `<PRODUCT_HOME>/repository/conf/security/authenticators.xml` file as shown above.

**Step 2 - Configure Publisher/Store for SSO**

To configure SSO for the API Publisher, open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` file and give the configurations as shown below.

Note that following parameters should be matched with the Service Provider configurations of Identity Server that you are configuring.
ssoConfiguration": {
    "enabled": "true",
    "issuer": "API_PUBLISHER",
    "identityProviderURL": "https://localhost:9444/samlsso",
    "keyStorePassword": "",
    "identityAlias": "wso2carbon",
    "verifyAssertionValidityPeriod": "true",
    "timestampSkewInSeconds": "300",
    "audienceRestrictionsEnabled": "true",
    "responseSigningEnabled": "true",
    "assertionSigningEnabled": "true",
    "keyStoreName": "",
    "signRequests": "true",
    "assertionEncryptionEnabled": "false",
    "idpInit": "false",
    "idpInitSSOURL": "https://localhost:9444/samlsso?spEntityID=API_PUBLISHER",
    "externalLogoutPage": "https://localhost:9444/samlsso?slo=true"
},

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Set this value to true to enable SSO in the application.</td>
</tr>
<tr>
<td>issuer</td>
<td>API_PUBLISHER. This value can change depending on the Issuer value defined in WSO2 IS SSO configuration above.</td>
</tr>
<tr>
<td>identityProviderURL</td>
<td><a href="https://localhost:9444/samlsso">https://localhost:9444/samlsso</a>. Change the IP and port accordingly. This is the redirecting SSO URL in your running WSO2 IS server instance.</td>
</tr>
<tr>
<td>keyStoreName</td>
<td>The keystore of the running IDP. As you use a remote instance of WSO2 IS here, you can import the public certificate of the IS keystore to the APIM and then point to the APIM keystore. The default keystore of the APIM is &lt;API-M_HOME&gt;/repository/sources/security/wso2carbon.jks. Be sure to give the full path of the keystore here.</td>
</tr>
<tr>
<td>keyStorePassword</td>
<td>Password for the above keystore. The default keyStorePassword is wso2carbon.</td>
</tr>
<tr>
<td>identityAlias</td>
<td>wso2carbon</td>
</tr>
</tbody>
</table>

By default, in the ssoConfiguration in site.json described above signRequests parameter decide whether to sign the AuthRequest or not. We have enabled it by setting the value true in default configuration so that, all the AuthRequests passing will be signed.

The identityAlias parameter is set to wso2carbon in the above example. You can configure an external server by importing the certificate of the IdP to APIM, and changing the identityAlias parameter value according to the certificate. To configure an IDP initiated SSO, you have to include the following additional parameters in the ssoConfiguration section.

```
... "idpInit": "true",
    "idpInitSSOURL": "https://localhost:9444/samlsso?spEntityID=API_PUBLISHER",
    "externalLogoutPage": "https://localhost:9444/samlsso?"slo=true"
...
```

To configure SSO for the API Store, open the <API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file and change the ssoConfiguration section similarly. This allows users to gain access to the Store applications directly, if they are authenticated against the Publisher.

idpInitSSOURL consists of <SAML2.IdPURL> and <SAML2.SPEntityId>.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAML2.IdPURL= <a href="https://localhost:9443/samlsso">https://localhost:9443/samlsso</a></td>
<td>The URL of the SAML 2.0 Identity Provider</td>
</tr>
<tr>
<td>SAML2.SPEntityId=API_PUBLISHER</td>
<td>A unique identifier for this SAML 2.0 Service Provider application</td>
</tr>
</tbody>
</table>

The SAML2.SPEntityId should be the value of issuer you specify under ssoConfiguration in the <API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file, which uniquely identifies your SAML identity provider. Therefore, it differs between the Store and Publisher according to the issuer that you configure.
**Step 3 - Configure the API Store for SSO in passive mode**

If the passive mode is disabled and Single Sign-On (SSO) is enabled, it redirects the user to the SSO login page. Therefore, as the WSO2 API Store allows anonymous access, passive mode is enabled by default, so that irrespective of whether SSO is enabled or not it directs the user to the API Store URL, and enables the SSO workflow only when the Sign In button is clicked.

To disable the passive mode, set the property named `passive` to `false` in the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` file.

```
"ssoConfiguration": {
    ...
    "passive": "false",
    ...
},
```

By enabling passive mode in SSO Configuration, WSO2 API Manager enables **Passive Authentication** on Single Sign On.

> From the two fundamental authentication models which are active and passive, **active authentication** is based on WS-Trust protocol on which a relying party is responsible of issuing the security token associated with the user credentials. But in **passive authentication** which is based on SAML 2.0 and WS-Federation protocols, the relying party does not control the login logic and relies on the IdP to issue the credentials.

**Step 4 - Configure an Identity Provider**

*Configuring WSO2 IS Configuring an external IdP*

You can configure WSO2 Identity Server as the IdP. For instructions on how to configure, see Configuring Identity Server as IdP for SSO. You can also opt to configure an external identity provider. For instructions on how to configure, see Configuring External IdP through Identity Server for SSO.

**Configuring External IDP through Identity Server for SSO**

The topics below explain the configurations you need to make to configure an external IdP through WSO2 Identity Server.

- Sharing the user store
- Sharing the registry space
- Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider
- Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

**Sharing the user store**

First you need to point both WSO2 IS and WSO2 API Manager to a single user store.

You do this to make sure that a user who tries to log in to the API Manager console, the API Store or the API Publisher is authorized. When a user tries to log in to either of the three applications, s/he is redirected to the configured identity provider (WSO2 IS, in this case) where s/he provides the login credentials to be authenticated. In addition to this, the user should also be authorized by the system as some user roles do not have permission to perform certain actions. For the purpose of authorization, the IS and API Manager need to have a shared user store and user management database (by default, this is the H2 database in the `<API-M_HOME>/repository/conf/user-mgt.xml` file) where the user’s role and permissions are stored.

For example, let's take a common JDBC user store (MySQL) for both the IS and API Manager.

Refer Configuring User Stores for more information on Configuring different types of user stores.

1. Create a MySQL database (e.g., `410_um_db`) and run the `<API-M_HOME>/dbscripts/mysql.sql` script on it to create the required tables.
   If you are using a different database type, find the relevant script from the `<API-M_HOME>/dbscripts` directory.
   ```
   <API-M_HOME>/dbscripts/mysql.sql is the script that should be used to run on the 410_um_db database for MySQL 5.6 and prior versions. If you database is MySQL 5.7 or later version, use <API-M_HOME>/dbscripts/mysql5.7.sql script file.
   ```

2. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and add the datasource configuration for the database that you use for the shared user store and user management information. For example,
2. <datasource>
   <name>WSO2_UM_DB</name>
   <description>The datasource used for registry and user manager</description>
   <jndiConfig>
     <name>jdbc/WSO2UMDB</name>
   </jndiConfig>
   <definition type="RDBMS">
     <configuration>
       <url>jdbc:mysql://localhost:3306/410_um_db</url>
       <username>username</username>
       <password>password</password>
       <driverClassName>com.mysql.jdbc.Driver</driverClassName>
       <maxActive>50</maxActive>
       <maxWait>60000</maxWait>
       <testOnBorrow>true</testOnBorrow>
       <validationQuery>SELECT 1</validationQuery>
       <validationInterval>30000</validationInterval>
     </configuration>
   </definition>
</datasource>

Refer Configuring master-datasources.xml for descriptive information about each property of the datasource configuration.

3. Add the same datasource configuration above to the <IS_HOME>/repository/conf/datasources/master-datasources.xml file.
4. Copy the database driver JAR file to the <IS_HOME>/repository/components/lib and <API-M_HOME>/repository/components/lib directories.
5. Open the <API-M_HOME>/repository/conf/user-mgt.xml file. The dataSource property points to the default H2 database. Change it to the jndiConfig name given above (i.e., jdbc/WSO2UMDB). This changes the datasource reference that is pointing to the default H2 database.

6. Add the same configuration above to the <IS_HOME>/repository/conf/user-mgt.xml file.
7. The Identity Server has an embedded LDAP user store by default. As this is enabled by default, follow the instructions in Internal JDBC User Store Configuration to disable the default LDAP and enable the JDBC user store instead.

Sharing the registry space

In a multi-tenant environment, by default, the Identity Server uses the key store of the super tenant to sign SAML responses. The API Store and API Publisher are already registered as SPs in the super tenant. However, if you want the Identity Server to use the registry key store of the tenant that the user belongs to, you can create a common registry database and mount it on both the IS and the API Manager.

1. Create a MySQL database (e.g., registry) and run the <IS_HOME>/dbscripts/mysql.sql script on it to create the required tables.
   If you are using a different database type, find the relevant script from the <IS_HOME>/dbscripts directory.
2. Add the following datasource configuration to both the <IS_HOME>/repository/conf/datasources/master-datasources.xml and <API-M_HOME>/repository/conf/datasources/master-datasources.xml files.
3. Create the registry mounts by inserting the following sections into the `<IS_HOME>/repository/conf/registry.xml` file.

When doing this change, do not replace the existing `<dbConfig>` for "wso2registry". Simply add the following configuration to the existing configurations. Refer Configuring registry.xml for more information on the parameters used in configuring registry.xml.

```
<dbConfig name="govregistry">
  <dataSource>jdbc:WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
  <id>gov</id>
  <dbConfig>govregistry</dbConfig>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
</remoteInstance>

<mount path="/system/governance" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>system/governance</targetPath>
</mount>

<mount path="/system/config" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>system/config</targetPath>
</mount>
```

4. Repeat the above step in the `<API-M_HOME>/repository/conf/registry.xml` file as well.

Next, let us look at the SSO configurations.

**Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider**

1. Start the IS server and log in to its Management Console (https://localhost:9444/carbon).

If you use login pages that are hosted externally to log in to the Identity Server, give the absolute URLs of those login pages in the authenticators.xml and application-authenticators.xml files in the `<IS_HOME>/repository/conf/identity` directory.

2. Click Add under the Service Providers menu.
3. Give a service provider name and click **Register**.

**Add New Service Provider**

In a multi tenanted environment, for all tenants to be able to log in to the API Manager web applications, do the following:

- Click the **SaaS Application** option that appears after registering the service provider.

If not, only users in the current tenant domain (the one you are defining the service provider in) will be allowed to log in to the web application and you have to register new service providers for all web applications (API Store and API Publisher in this case) from each tenant space separately. For example, let's say you have three tenants as TA, TB and TC and you register the service provider in TA only. If you tick the **SaaS Application** option, all users in TA, TB, TC tenant domains will be able to log in. Otherwise, only users in TA will be able to log in.

- Add the following inside the `<SSOService>` element in the `<IS_HOME>/repository/conf/identity/identity.xml` file and restart the server.

```xml
<SSOService>
  <UseAuthenticatedUserDomainCrypto>true</UseAuthenticatedUserDomainCrypto>
  ...
</SSOService>
```

If not, you get an exception stating that the SAML response signature verification fails.
3. You are navigated to the detailed configuration page. Inside the Inbound Authentication Configuration section, expand SAML2 Web SSO Configuration and click Configure.

   Service Providers

   To enable tenant specific SSO with IS 5.7.0 for API_PUBLISHER and API_STORE, enable Use tenant domain in local subject identifier under the Local & Outbound Authentication Configuration section.

5. Provide the configurations to register the API Publisher as the SSO service provider. These sample values may change depending on your configuration.
   - Issuer: API_PUBLISHER
   - Assertion Consumer URL: https://localhost:9443/publisher/jagg/jaggery_acs.jag Change the IP and port accordingly. This is the URL for the acs page in your running publisher app.
   - Select the following options:
     - Enable Response Signing
     - Enable Single Logout
     - Enable Attribute Profile
     - Include Attributes in the Responses Always

   For example:
6. Similarly, provide the configurations to register the API Store as the SSO service provider. These sample values may change depending in your configuration:

- **Issuer**: API_STORE
- **Assertion Consumer URL**: https://localhost:9443/store/jagg/jaggery_acs.jag Change the IP and port accordingly. This is the URL for the acs page in your running store app.
- **Select the following options:**
  - Enable Response Signing
  - Enable Single Logout
  - Enable Attribute Profile
  - Include Attributes in the Responses Always

7. Make sure that the responseSigningEnabled element is set to true in both the following files:
8. Add a new Identity Provider in WSO2 Identity Server. For more details on configuring external IDPs in WSO2 IS, see Configuring an Identity Provider.
   - Identity Provider Name: ExternalIS
   - Do the following changes under Federated Authenticators > SAML2 Web SSO Configurations
     - Enable SAML2 Web SSO
     - Check Default
     - Set Service Provider Entity ID
     - Set SSO URL for the external IDP (e.g., https://localhost/:9453/samlsso)
   - Enable Logout

9. Enable JIT Provisioning for the external IDP. For more information, see Configuring Just-In-Time Provisioning for an Identity Provider.

10. Map the external IDP roles to the roles configured in API Manager. For more information on mapping roles, see Configuring Roles for an Identity Provider.

11. Open the management console, and click Edit under Service Providers.
12. Under Local & Outbound Authentication Configuration select Federated Authentication. Select the newly created external IDP.

13. Add http://wso2.org/claims/role as the Claim URI under Claim Configuration. Select the Mandatory Claim check box. Add http://wso2.org/claims/username as the Subject Claim URI.

   Additionally, you might need to configure claims to map them to the available claims in WSO2 Identity Server. For more details, see Configuring Claims for an Identity Provider.

**Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers**

1. Open the <API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json and modify the following configurations found under ssoConfiguration
   - enabled: Set this value to true to enable SSO in the application
   - issuer: API_PUBLISHER. This value can change depending on the Issuer value defined in the WSO2 IS SSO configuration above.
1. identityProviderURL: https://localhost:9444/samlsso. Change the IP and port accordingly. This is the redirecting SSO URL in your running WSO2 IS server instance.
2. keyStoreName: The keystore of the running IDP. As you use a remote instance of WSO2 IS here, you can import the public certificate of the IS keystore to the API Manager and then point to the API Manager keystore. The default keystore of the API Manager is <API-M_HOME>/repository/resources/security/wso2carbon.jks. Make sure you give the full path of the keystore here.
3. keyStorePassword: Password for the above keystore
4. identityAlias: wso2carbon

You do this as a workaround for a known issue that will be fixed in a future release. The SAML2SSOAuthenticator handler does not process only SAML authentication requests at the moment. If you set its priority higher than that of the BasicAuthenticator handler, the SAML2SSOAuthenticator tries to process the basic authentication requests as well. This causes login issues in the API Publisher/Store.

```xml
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
  <Priority>0</Priority>
  ....
</Authenticator>
```

You can skip this step if you are using Identity Server 5.7.0 as the IDP.

4. Access the API Publisher (e.g., https://localhost:9443/publisher). Observe the request redirecting to the WSO2 IS SAML2.0 based SSO login page. For example,

```
WSO2 IDENTITY SERVER
```

```
SIGN IN

Username
Password

Remember me on this computer

SIGN IN
```

5. Enter the user credentials. If the user authentication is successful against WSO2 IS, it redirects to the API Publisher with the user that is already authenticated.
6. Access the API Store, click its Login link (top, right-hand corner) and verify that the same user is already authenticated in the API Store.

Even with SSO enabled, if the user doesn't have sufficient privileges to access the API Publisher/Store or any other application, s/he will not be authorized to access them.

To learn more about Single Sign-On with WSO2 Identity Server, see SAML 2.0 Web SSO in the WSO2 Identity Server documentation.

Configuring Identity Server as IDP for SSO

The Single Sign-On with SAML 2.0 feature in the API Manager is implemented according to the SAML 2.0 browser-based SSO support that is facilitated by WSO2 Identity Server (WSO2 IS). This feature is available in any WSO2 IS version from 4.1.0 onwards. We use WSO2 IS 5.7.0 in this guide. WSO2 Identity Server acts as an identity service provider of systems enabled with single sign-on, while the Web applications act as SSO service providers. Using this feature, you can configure SSO across the API Publisher and Store. After configuring, you can access the API Store or API Publisher in a single authentication attempt.
The topics below explain the configurations.

- Sharing the user store
- Sharing the registry space
- Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider
- Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

**Sharing the user store**

Initially, configure your user store(s), if you have not done so already, by following the instructions in Configuring User Stores. Thereafter, point both WSO2 IS and WSO2 API Manager to your user stores(s) using the instructions given below. You do this to make sure that a user who tries to log in to the API Manager console, the API Store or the Publisher is authorized. When a user tries to log in to either of the three applications, s/he is redirected to the configured identity provider (WSO2 IS in this case) where s/he provides the login credentials to be authenticated. In addition to this, the user should also be authorized by the system as some user roles do not have permission to perform certain actions. For the purpose of authorization, the IS and API Manager need to have a shared user store and user management database (by default, this is the H2 database in the <API-M_HOME>/repository/conf/user-mgt.xml file) where the user's role and permissions are stored.

For example, let's share a JDBC user store (MySQL) with both the WSO2 Identity Server and WSO2 API Manager as follows:

1. Download WSO2 API Manager from here and unzip it. <API-M_HOME> refers to the root folder where WSO2 API-M was unzipped.
2. Create a MySQL database (e.g., 410_um_db) and run the <API-M_HOME>/dbscripts/mysql.sql script on it to create the required tables.
3. Open the <API-M_HOME>/repository/conf.datasources/master-datasources.xml file and add the datasource configuration for the database that you use for the shared user store and user management information. For example, you can share as single user store as follows. If you are sharing multiple datasources, you need to define a datasource for each of the user stores that you are working with, so that they can be shared.

**Example**

```xml
<dataSource>
  <name>WSO2_UM_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2UMDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://localhost:3306/410_um_db</url>
      <username>username</username>
      <password>password</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</dataSource>
```

Change the database url to the url of the MySQL database you have created above. Modify the username and password parameters in above configuration with your mysql database credentials.

Refer Configuring master-datasources.xml for descriptive information about each property of the datasource configuration.

4. Download WSO2 Identity Server (WSO2 IS) 5.7.0 from here and unzip it. <IS_HOME> refers to the root folder where WSO2 IS was unzipped.
5. Add the same datasource configuration above to `<IS_HOME>/repository/conf/datasources/master-datasources.xml` file.
6. Copy the database driver JAR file to the `<IS_HOME>/repository/components/lib` and `<API-M_HOME>/repository/components/lib` directories.
7. Open the `<API-M_HOME>/repository/conf/user-mgt.xml` file. The `dataSource` property points to the default H2 database. Change it to the `jndiConfig` name given above (i.e., `jdbc/WSO2UMDB`). This changes the datasource reference that is pointing to the default H2 database.

```
<Realm>
  <Configuration>
    ...
    <Property name="dataSource">jdbc/WSO2UMDB</Property>
    ...
  </Configuration>
</Realm>
```

8. Add the same configuration above to the `<IS_HOME>/repository/conf/user-mgt.xml` file.
9. The Identity Server has an embedded LDAP user store by default. As this is enabled by default, follow the instructions in Internal JDBC User Store Configuration to disable the default LDAP and enable the JDBC user store instead.

In WSO2 API Manager, the JDBC User Store is enabled by default. By changing the default user store of WSO2 Identity server to JDBC User Store, we are pointing both WSO2 API Manager and WSO2 Identity Server to the same user store so that, their user stores are shared.

**Sharing the registry space**

In a multi-tenanted environment, by default, the Identity Server uses the key store of the super tenant to sign SAML responses. The API Store and Publishers are already registered as SPs in the super tenant. However, if you want the Identity Server to use the registry key store of the tenant that the user belongs to, you can create a common registry database and mount it on both the IS and the APIM.

1. Create a MySQL database (e.g., registry) and run the `<IS_HOME>/dbscripts/mysql.sql` script on it to create the required tables. If you are using a different database type, find the relevant script from the `<IS_HOME>/dbscripts` directory.

```
There are two MySQL DB scripts available in the product distribution from WSO2 Carbon Kernel 4.4.6 onwards. Click here to identify as to which version of the MySQL script to use.
```

2. Add the following datasource configuration to both the `<IS_HOME>/repository/conf/datasources/master-datasources.xml` and `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` files.

```
<datasource>
  <name>WSO2REG_DB</name>
  <description>The datasource used for registry</description>
  <jndiConfig>
    <name>jdbc/WSO2REG_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://localhost:3306/registry?autoReconnect=true&amp;relaxAutoCommit=true&amp;</url>
      <username>apiuser</username>
      <password>apimanager</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
    </definition>
  </datasource>
```

Modify the username and password parameters of above configuration with your mysql database credentials.
3. Create the registry mounts by inserting the following sections into the `<IS_HOME>/repository/conf/registry.xml` file.

When doing this change, do not replace the existing `<dbConfig>` for "wso2registry". Simply add the following configuration to the existing configurations.

```xml
<dbConfig name="govregistry">
  <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
  <id>gov</id>
  <dbConfig>govregistry</dbConfig>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>

<mount path="/\system/governance" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/\system/governance</targetPath>
</mount>

<mount path="/\system/config" overwrite="true">
  <instanceId>gov</instanceId>
  <targetPath>/\system/config</targetPath>
</mount>
```

Refer Configuring registry.xml for more details on configuration details and usage of registry.xml

4. Repeat the above step in the `<API-M_HOME>/repository/conf/registry.xml` file as well.

Next, let us look at the SSO configurations.

**Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider**

1. Start WSO2 Identity Server.

   ```
   ./wso2server.sh -DportOffset=1
   ```

   You also can change Port offset value by changing `<Offset>` 1 `</Offset>` under `<Ports>` in `<IS_HOME>/repository/conf/carbon.xml` file.

**What is port offset?**

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime, such as the HTTP/S ports, will be offset. For example, if the HTTPS port is defined as 9443 and the portOffset is 1, the effective HTTPS port will be 9444.


   If you use signin pages that are hosted externally to sign in to the Identity Server, give the absolute URLs of those login pages in the `application-authenticators.xml` and `application-authenticators.xml` files in the `<IS_HOME>/repository/conf/identity` directory.
3. Select **Add** under the **Service Providers** menu.

4. Give a service provider name and click **Register**.

In a multi-tenanted environment, for all tenants to be able to log in to the APIM Web applications, do the following:

- Click the **SaaS Application** option that appears after registering the service provider.

If not, only users in the current tenant domain (the one you are defining the service provider in) will be allowed to log in to the Web application and you have to register new service providers for all Web applications (API Store and API Publisher in this case) from each tenant space separately. For example, let's say you have three tenants as TA, TB, and TC and you register the service provider in TA only. If you tick the **SaaS Application** option, all users in TA, TB, TC tenant domains will be able to log in. Else, only users in TA will be able to log in.

- Add the following inside the `<SSOService>` element in the `<IS_HOME>/repository/conf/identity/identity.xml` file and restart the server.

```xml
<SSOService>
  <UseAuthenticatedUserDomainCrypto>true</UseAuthenticatedUserDomainCrypto>
  ...
</SSOService>
```

If not, you get an exception as SAML response signature verification fails.

- Because the servers in a multi-tenanted environment interact with all tenants, all nodes should share the same user store. Therefore, make sure you have a shared registry (JDBC mount, WSO2 Governance Registry etc.) instance across all nodes.
5. You are navigated to the detailed configuration page. Inside the Inbound Authentication Configuration section, expand SAML2 Web SSO Configuration and click Configure.

To enable tenant specific SSO with IS 5.7.0 for API_PUBLISHER and API_STORE, enable Use tenant domain in local subject identifier under the Local & Outbound Authentication Configuration section.

6. Provide the configurations to register the API Publisher as the SSO service provider. These sample values may change depending in your configuration.
   - **Issuer**: API_PUBLISHER
   - **Assertion Consumer URL**: `https://localhost:9443/publisher/jagg/jaggery_acs.jag`. Change the IP and port accordingly. This is the URL for the Assertion Consumer Services (ACS) page in your running publisher app.
   - Select the following options:
     - Enable Response Signing
     - Enable Single Logout
     - Click Register once done.

For example:
6. Similarly, provide the configurations to register the API Store as the SSO service provider. These sample values may change depending on your configuration.

- **Issuer**: API_STORE
- **Assertion Consumer URL**: https://localhost:9443/store/jagg/jaggery_acs.jag. Change the IP and port accordingly. This is the URL for the Assertion Consumer Services (ACS) page in your running Store app.
  - Select the following options:
    - **Enable Response Signing**
    - **Enable Single Logout**
  - Click *Register* once done.

7. Make sure that the `responseSigningEnabled` element is set to `true` in both the following files:

```
<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json
<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json
```

**Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers**

1. Open `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` and modify the following configurations found under `ssOConfiguration`:

- **enabled**: Set this value to `true` to enable SSO in the application
- **Issuer**: API_PUBLISHER. This value can change depending on the `Issuer` value defined in WSO2 IS SSO configuration above.
- **IdentityProviderURL**: https://localhost:9444/samlsso. Change the IP and port accordingly. This is the redirecting SSO URL in your running WSO2 IS server instance.
- **keyStoreName**: The keystore of the running IDP. As you use a remote instance of WSO2 IS here, you can import the public certificate of the IS keystore to the APIM and then point to the APIM keystore. The default keystore of the APIM is `<API-M_HOME>/repository/resources/security/wso2carbon.jks`. Be sure to give the full path of the keystore here.
- **keyStorePassword**: Password for the above keystore. The default keyStorePassword is wso2carbon.
- **IdentityAlias**: wso2carbon

2. Similarly, configure the API Store with SSO. The only difference in API Store SSO configurations is setting `API_STORE` as the `issuer`.

3. Reduce the priority of the SAML2SSOAuthenticator configuration in the `<API-M_HOME>/repository/conf/security/authenticators.xml` file.

You do this as a workaround for a known issue that will be fixed in a future release. The SAML2SSOAuthenticator handler does not process only SAML authentication requests at the moment. If you set its priority higher than that of the BasicAuthenticator handler, the SAML2SSOAuthenticator tries to process the basic authentication requests as well. This causes login issues in the API Publisher/Store.
3. <Authenticator name="SAML2SSOAuthenticator" disabled="false">
   <Priority>0</Priority>
   ....
   </Authenticator>

You can skip this step if you are using Identity Server 5.7.0 as the IDP.

4. Access the API Publisher: https://localhost:<port_number>/publisher (e.g., https://localhost:9443/publisher). Observe the request redirect to the WSO2 IS SAML2.0 based SSO login page. For example,

```
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
  <Priority>0</Priority>
  ....
  </Authenticator>
```

5. Enter user credentials. If the user authentication is successful against WSO2 IS, it will redirect to the API Publisher Web application with the user already authenticated.

6. Access the API Store application, click its Login link (top, right-hand corner) and verify that the same user is already authenticated in API Store.

Even with SSO enabled, if the user doesn’t have sufficient privileges to access API Publisher/Store or any other application, s/he will not be authorized to access them.

To learn more about Single Sign-On with WSO2 Identity Server, see SAML 2.0 Web SSO in the WSO2 Identity Server documentation.

Configuring Single Sign-on with OpenID Connect

OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.

OpenID Connect (OIDC) is a simple identity layer on top of the OAuth 2.0 (OAauth2) protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

OpenID Connect allows clients of all types, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. The specification suite is extensible, allowing participants to use optional features such as encryption of identity data, discovery of OpenID Providers, and session management, when it makes sense for them.

- Configuring SSO with OpenID Connect
- Using SSO in API Manager with OpenID Connect

Configuring SSO with OpenID Connect

Follow the instructions below to configure Single Sign-on (SSO) using OpenID Connect in WSO2 API Manager (WSO2 API-M) using WSO2 Identity Server (WSO2 IS):

- Step 1 - Download WSO2 IS
- Step 2 - Download WSO2 API-M
Step 1 - Download WSO2 IS

Download WSO2 Identity Server 5.7.0.

For testing purposes if you want to run both the WSO2 API-M and WSO2 IS server on the same server, then you can go to the `<IS_HOME>/repository/conf/carbon.xml` file and offset the port by 1 as follows:

```xml
<Offset>1</Offset>
```

Step 2 - Download WSO2 API-M

Download WSO2 API Manager 2.6.0.

For testing purposes if you want to run both the WSO2 API-M and WSO2 IS server on the same server, then you can change the hostname in WSO2 API-M.

- Click here for more information on changing the hostname in WSO2 API-M.

Follow the instructions below to change the hostname in WSO2 API-M:

2. Change the hostname and the management hostname based on your choice.
   
   ```xml
   <HostName>wso2.am</HostName>
   <MgtHostName>wso2.am</MgtHostName>
   ```

3. Open the `/etc/hosts` file.

   ```bash
   vim /etc/hosts
   ```

4. Add the new hostname in the `/etc/hosts` file.

Step 3 - Configure the user database

Configure a database of your choice with SSL support. This example uses a MySQL database.

- Click here to configure a MySQL DB.

Follow the instructions below to configure your user DB if you are using MySQL. For more information, see Installing and Configuring the Databases.

1. Download and install MySQL Server 5.7.
   For more information on DB compatibility, see Tested DBMSs.
2. Download the MySQL JDBC driver.
3. Unzip the downloaded MySQL driver archive, and copy the MySQL JDBC driver JAR (`mysql-connector-java-x.x.xx-bin.jar`) into the `<API-M_HOME>/repository/components/lib` directory and into the `<IS_HOME>/repository/components/lib` directory.
4. Access the database.
   Enter the following command in a command prompt. Where `<username>` is the username.
   ```bash
   mysql -u<username> -p
   ```
   Error rendering macro 'code': Invalid value specified for parameter 'firstline'
   ```bash
   mysql -u<username> -p
   ```
   Error rendering macro 'code': Invalid value specified for parameter 'firstline'
   ```bash
   mysql -uroot -p
   ```
   When prompted, specify the password that corresponds to the username that you specified to access the database.
5. Create the databases using the following commands, where `<API-M_HOME>` is the path to the WSO2 API Manager instance that you installed, and the username and password are the same credentials that you specified in the previous steps.
Step 4 - Configure the user stores

Follow the instructions below to share the user stores between WSO2 API-M and WSO2 IS.

1. Navigate to the `master-datasources.xml` file in the following directories.
   - WSO2 IS - `<IS_HOME>/repository/conf/datasources`
   - WSO2 API-M - `<API-M_HOME>/repository/conf/datasources`

2. Add the `WSO2UM_DB` related datasource configurations in order to share the user stores between WSO2 API-M and WSO2 IS.

By default, WSO2 API-M uses a JDBC user store, while WSO2 IS uses a LDAP user store. This example uses a JDBC user store and MySQL DB.

Format Example

```xml
<datasource>
  <name>WSO2UM_DB</name>
  <description>The datasource used by user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2UM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://[host_name_of_mysql_server]:3306/userdb?autoReconnect=true</url>
      <username>[user]</username>
      <password>[password]</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

Make sure to replace the following placeholders:

- `[host_name_of_mysql_server]`
- `[user]`
- `[password]`
<datasource>
  <name>WSO2UM_DB</name>
  <description>The datasource used by user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2UM_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <username>root</username>
      <password>root</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>

SSL is enabled by default. However, only for testing purposes you can disable SSL by updating the URL as follows in the WSO2 IS and WSO2 API-M <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. However, this is not recommended for a production environment and you need to make sure that SSL is enabled.

```xml
<url>jdbc:mysql://localhost:3306/userdb?autoReconnect=true&amp;useSSL=false</url>
```

3. Update the user-mgt.xml file in the <IS_HOME>/repository/conf directory and the <API-M_HOME>/repository/conf directory. Update the default configurations with the following property configurations.

```xml
<configuration>
...<Property name="dataSource">jdbc/WSO2UM_DB</Property>
</configuration>
```

4. Configure the user store manager properties. In this example, as you are using JDBC as the user store, you need to update the <IS_HOME>/repository/conf/user-mgt.xml file as follows:
   a. Comment out the default LDAP user store details, which are defined in the <UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager"> section.
   b. Uncomment the following code block with regard to the to the JDBC user store.
Step 5 - Configure the Identity Provider

Configure an Identity Provider of your choice. In this example we use WSO2 IS as the Identity Provider (IdP).

Follow the instructions below to configure WSO2 IS as the IdP:

1. Start WSO2 Identity Server.
   - On Windows: `<IS_HOME>/bin/wso2server.bat --run`
   - On Linux/Mac OS: `sh <IS_HOME>/bin/wso2server.sh`

2. Create a service provider (SP) for the Store. For more detailed information, see Adding and Configuring a Service Provider in the WSO2 Identity Server 5.7.0 documentation.
   a. Create a service provider (SP) as API_STORE with a inbound OAuth/oidc callback URL as `https://<APIM-hostname>:9443/store/jagg/jaggery_oidc_acs.jag` and enable the authorization-code grant type, by checking the Code checkbox that corresponds to Allowed Grant Types.  
      <APIM-hostname> - Replace this with the hostname of the WSO2 API Manager Server.
   b. Add claim mapping and set the `http://wso2.org/claims/displayName` as the local claim and make it a mandatory claim.
      You can either use a local claim or a custom claim.
      Your configurations should look similar to the following after creating the service provider.
3. Create a service provider for the Publisher:
   a. Create a service provider as API_PUBLISHER with an inbound OAuth/oidc callback URL as https://<APIM-hostname>:9443/publi sher/jagg/jaggery_oidc_acs.jag and enable the authorization-code grant type.
   <APIM-hostname> - Replace this with the hostname of the WSO2 API Manager Server.
   b. Add claim mapping and set the http://wso2.org/claims/displayName as the local claim and make it a mandatory claim. 
   You can either use a local claim or a custom claim.
   Your configurations should look similar to the following after creating the service provider.
Step 6 - Configure WSO2 API-M

Configure WSO2 API-M with the Identity Provider.

Follow the instructions below to configure WSO2 API-M with WSO2 IS, which is the Identity Provider in this example.

1. Configure the API Store. For more detailed information, see Adding and Configuring a Service Provider in the WSO2 Identity Server 5.7.0 documentation.
   b. Edit the `oidcConfiguration` section to point to the IdP that you configured in step 5.

Make sure to replace the following placeholders with actual values.

- Replace these with the credentials that you got when creating the API_STORE service provider.
  - <client-id>
  - <client-secret>
- Replace this with the hostname of the IdP.
  - <IdP-hostname>
  - <IdP-port>
- Replace this with the hostname of the WSO2 API Manager Server.
  - <APIM-hostname>
  - <APIM-port>

In this example WSO2 IS is port offset by 1.


Make sure to replace the following placeholders with actual values.

- Replace these with the credentials that you got when creating the API_STORE service provider.
  - <client-id>
  - <client-secret>
- Replace this with the hostname of the WSO2 Identity Server.
  - <IdP-hostname>
- Replace this with the WSO2 IS port
  - <IdP-port>
- Replace this with the hostname of the WSO2 API Manager Server.
  - <APIM-hostname>
- Replace this with the WSO2 APIM port

2. Configure the API Publisher.
a. Navigate to the `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` file.
b. Edit the `oidcConfiguration` section to point to the IdP that you configured in step 5.

Format Example

```
"oidcConfiguration" : {
  "enabled" : "true",
  "issuer" : "API_PUBLISHER",
  "identityProviderURI" : "https://<IdP-hostname>:<IdP-port>/oauth2/token",
  "authorizationEndpointURI" : "https://<IdP-hostname>:<IdP-port>/oauth2/authorize",
  "tokenEndpointURI" : "https://<IdP-hostname>:<IdP-port>/oauth2/token",
  "userInfoURI" : "https://<IdP-hostname>:<IdP-port>/oauth2/userinfo",
  "jwksURI" : "https://<IdP-hostname>:<IdP-port>/oauth2/jwks",
  "logoutEndpointURI" : "https://<IdP-hostname>:<IdP-port>/oidc/logout",
  "authHttpMethod": "POST",
  "clientConfiguration" : {
    "clientId" : "<client-id>",
    "clientSecret" : "<client-secret>",
    "responseType" : "code",
    "authorizationType" : "authorization_code",
    "scope" : "phone email address openid profile",
    "redirectURI" : "https://<APIM-hostname>:<APIM-port>/publisher/jagg/jaggery_oidc_acs.jag",
    "postLogoutRedirectURI" : "https://<APIM-hostname>:<APIM-port>/publisher/"
  }
},
```

Make sure to replace the following placeholders with actual values.

- `<client-id>` and `<client-secret>` - Replace these with the credentials that you got when creating the `API_PUBLISH` service provider.
- `<IdP-hostname>` - Replace this with the hostname of the IdP.
- `<IdP-port>` - Replace this with the IdP port
- `<APIM-hostname>` - Replace this with the hostname of the WSO2 API Manager Server.
- `<APIM-port>` - Replace this with the WSO2 APIM port

In this example WSO2 IS is port offset by 1.

```
"oidcConfiguration" : {
  "enabled" : "true",
  "issuer" : "API_PUBLISHER",
  "authorizationEndpointURI" : "https://localhost:9444/oauth2/authorize",
  "tokenEndpointURI" : "https://localhost:9444/oauth2/token",
  "userInfoURI" : "https://localhost:9444/oauth2/userinfo",
  "jwksURI" : "https://localhost:9444/oauth2/jwks",
  "logoutEndpointURI" : "https://localhost:9444/oidc/logout",
  "authHttpMethod": "POST",
  "clientConfiguration" : {
    "clientId" : "BB5qBB8mr54JJJJI5T56uH8Gvfkk",
    "clientSecret" : "hiAk_Y_vVbbbrDP_6sJJchJgKlwca",
    "responseType" : "code",
    "authorizationType" : "authorization_code",
    "scope" : "phone email address openid profile",
    "redirectURI" : "https://wso2.am:9443/publisher/jagg/jaggery_oidc_acs.jag",
    "postLogoutRedirectURI" : "https://wso2.am:9443/publisher/"
  }
},
```

Make sure to replace the following placeholders with the actual values.

- `<client-id>` and `<client-secret>` - Replace these with the credentials that you got when creating the `API_PUBLISH` service provider.
- `<IdP-hostname>` - Replace this with the hostname of the WSO2 Identity Server.
- `<IdP-port>` - Replace this with the WSO2 IS port.
2. Optionally, map the username claim with a claim of your choice.

   In order to use this feature, apply the WUM update for WSO2 API-M 2.6.0 released on 15-11-2018 (15th Nov 2018).

   If you want to deploy a WUM update in to production, you need to have a paid subscription. If you do not have a paid subscription, you can use this feature with the next version of WSO2 API Manager when it is released. For more information on updating WSO2 API Manager using WUM, see Getting Started with WUM in the WSO2 Updates Guide.

   By default, in WSO2 API-M the username is mapped to the preferred_username claim. However, if you need WSO2 API-M to map the username to a different claim, follow the instructions below.
   a. Open the <API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file and the <API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json files.
   b. Add the usernameclaim property under the oidcconfiguration configurations sections in both the files and specify the OIDC claim to which you want to map the username claim.

   FormatExample

```
"oidcConfiguration" : {
  ...
  "usernameclaim":"<oidc_claim_name>",
}
```

   For example, if you want to use the user's email address as their username, then you would configure the site.json as follows:

```
"oidcConfiguration" : {
  ...
  "usernameclaim":"email",
  ...
}
```

   **Step 7 - Import the public certificate of the Identity Provider**

   Import the public certificate of the Identity Provider (IdP), which corresponds to the WSO2 Identity Server, into the client-trustore.jks of WSO2 API-M.

   1. Navigate to the <IS_HOME>/repository/resources/security directory.

```
  cd <IS_HOME>/repository/resources/security
```

   2. Export the public certificate to a .pem file.

```
  keytool -export -alias wso2carbon -keystore wso2carbon.jks -file publickey.pem
```

   Enter the password as wso2carbon when requested. This is the default password for keystores.

```
Certificate stored in file <publickey.pem>
Warning:
The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format using "keytool -importkeystore -srckeystore wso2carbon.jks -destkeystore wso2carbon.jks -deststoretype pkcs12".
```

   3. Copy the <IS_HOME>/repository/resources/security/publickey.pem file to the <API-M_HOME>/repository/resources/security directory.
4. Navigate to the `<API-M_HOME>/repository/resources/security` directory.

```
  cd <API-M_HOME>/repository/resources/security
```

5. Import the .pem file in to the client trust store (client-truststore.jks).

```
  keytool -import -alias wso2is -file publickey.pem -keystore client-truststore.jks -storepass wso2carbon
```

Type `yes` when the question that you see in the second line is printed.

```
Certificate already exists in keystore under alias <wso2carbon>
Do you still want to add it? [no]: yes
Certificate was added to keystore
```

6. Check the details of the imported certificate that corresponds to the Identity Provider.

```
  keytool -list -alias wso2is -keystore client-truststore.jks -v
```

Step 8 - Start WSO2 API Manager

Start WSO2 API-M based on your OS.

- On Windows: `<API-M_HOME>/bin/wso2server.bat --run`
- On Linux/Mac OS: `sh <API-M_HOME>/bin/wso2server.sh`

Using SSO in API Manager with OpenID Connect

Follow the instructions below to use OpenID Connect based SSO with the API Store and the API Publisher.
Let’s consider a scenario where you first log in to the Publisher and then log in to the Store.

1. Configure OpenID Connect for SSO.
   For more information, see Configuring SSO with OpenID Connect.
2. Access the API Publisher.
   https://<APIM-hostname>:<APIM-port>/publisher/
   In this example, access the Publisher as follows:
   https://wso2.am:9443/publisher/
3. Provide your username and password and click SIGN IN.
4. Enter your username as the display name and click **SIGN IN**.

5. Check Select All to select the mandatory user claims related to `API_PUBLISHER` and also check one of the approve options (Approve Once or Approve Always) based on your preference.
If you select **Approve Once**, you will have to approve OpenID user claim related data each time that you sign in to the Publisher.

6. Click **Continue**.
   You are now logged in to the Publisher interface.

7. Access the Store.
   https://<APIM-hostname>:<APIM-port>/store/
In this example, access the Store as follows:
https://wso2.am:9443/store/

8. Click **Sign In**.

9. Check **Select All** to select the mandatory user claims related to `API_STORE` and also check one of the approve options (Approve Once or Approve Always) based on your preference. If you select Approve Once, you will have to approve OpenID user claim related data each time that you sign in to the Store.

10. Click **Continue**.
    You are directly logged in to the Store without needing to add any user credentials.
Configuring Transports

A transport is responsible for carrying messages that are in a specific format. WSO2 API Manager supports all the widely used transports including HTTP/s, JMS, Pass-through and VFS, and domain-specific transports like FIX. All WSO2 transports are directly or indirectly based on the Apache Axis2 transports framework. This framework provides two main interfaces that each transport implementation has.

- org.apache.axis2.transport.TransportListener: Implementations of this interface specify how incoming messages are received and processed before handing them over to the Axis2 engine for further processing.
- org.apache.axis2.transport.TransportSender: Implementations of this interface specify how a message can be sent out from the Axis2 engine.

Because each transport has to implement the two interfaces above, each transport generally contains a transport receiver/listener and a transport sender. You configure, enable, and manage transport listeners and senders independently to each other. For example, you can enable just the JMS transport sender without having to enable the JMS transport listener.

For more information, see the following topics in the WSO2 ESB documentation:
- Available transports
- How to configure transports
- HTTP Transport Properties

Configuring User Stores

A user store is the database where information of the users and/or user roles is stored. User information includes log-in name, password, first name, last name, e-mail etc.

All WSO2 products have an embedded H2 database except for WSO2 Identity Server, which has an embedded LDAP as its user store. Permission is stored in a separate database called the user management database, which by default is H2. However, users have the ability to connect to external user stores as well.

The user stores of Carbon products can be configured to operate in either one of the following modes.

- User store operates in read/write mode - In Read/Write mode, WSO2 Carbon reads/writes into the user store.
- User store operates in read only mode - In Read Only mode, WSO2 Carbon guarantees that it does not modify any data in the user store. Carbon maintains roles and permissions in the Carbon database but it can read users/roles from the configured user store.

The sections below provide configuration details:
- Realm Configuration
- Changing the RDBMS
- Configuring Primary User Stores

Realm Configuration

The complete functionality and contents of the User Management module is called a user realm. The realm includes the user management classes, configurations and repositories that store information. Therefore, configuring the User Management functionality in a WSO2 product involves setting up the relevant repositories and updating the relevant configuration files.

The following diagram illustrates the required configurations and repositories:
See the following topics for more details:

- Configuring the system administrator
- Configuring the authorization manager

**Configuring the system administrator**

The **admin** user is the super tenant that will be able to manage all other users, roles and permissions in the system by using the management console of the product. Therefore, the user that should have admin permissions is required to be stored in the primary user store when you start the system for the first time. The documentation on setting up primary user stores will explain how to configure the administrator while configuring the user store. The information under this topic will explain the main configurations that are relevant to setting up the system administrator.

If the primary user store is read-only, you will be using a user ID and role that already exists in the user store, for the administrator. If the user store is read/write, you have the option of creating the administrator user in the user store as explained below. By default, the embedded H2 database (with read/write enabled) is used for both these purposes in WSO2 products.

Note the following key facts about the system administrator in your system:

- The admin user and role is always stored in the primary user store in your system.
- An administrator is configured for your system by default. This **admin** user is assigned to the **admin** role, which has all permissions enabled.
- The permissions assigned to the default **admin** role cannot be modified.

**Updating the administrator**

The `<Configuration>` section at the top of the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file allows you to configure the administrator user in your system as well as the RDBMS that will be used for storing information related to user authentication (i.e. role-based permissions).
<Realm>
  <Configuration>
    <AddAdmin>true</AddAdmin>
    <AdminRole>admin</AdminRole>
    <AdminUser>
      <UserName>admin</UserName>
      <Password>admin</Password>
    </AdminUser>
    <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role see the registry root -->
    <Property name=""></Property>
  </Configuration>
</Realm>

Note the following regarding the configuration above.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;AddAdmin&gt;</td>
<td>When true, this element creates the admin user based on the AdminUser element. It also indicates whether to create the specified admin user if it doesn’t already exist. When connecting to an external read-only LDAP or Active Directory user store, this property needs to be false if an admin user and admin role exist within the user store. If the admin user and admin role do not exist in the user store, this value should be true, so that the role is added to the user management database. However, if the admin user is not there in the user store, we must add that user to the user store manually. If the AddAdmin value is set to true in this case, it will generate an exception.</td>
</tr>
<tr>
<td>&lt;AdminRole&gt;</td>
<td>wso2admin</td>
</tr>
<tr>
<td>&lt;AdminUser&gt;</td>
<td>Configures the default administrator for the WSO2 product. The user store is read-only, the admin user must exist in the user store or the system will not start. If the external user store is read-only, you must select a user already existing in the external user store and add it as the admin user that is defined in the AdminUser element. If the external user store is in read/write mode, and you set AddAdmin to true, the user you specify will be automatically created.</td>
</tr>
<tr>
<td>&lt;UserName&gt;</td>
<td>This is the username of the default administrator or super tenant of the user store. If the user store is read-only, the admin user MUST exist in the user store for the process to work.</td>
</tr>
<tr>
<td>&lt;Password&gt;</td>
<td>If the user store is read-only, this element and its value are ignored after the server starts for the first time. Therefore we can reset this password back to the original value/variable after server starts for the first time. This password is used only if the user store is read-write and the AddAdmin value is set to true. Note that the password in the user-mgt.xml file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the user-mgt.xml file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the user-mgt.xml file. To change the admin password, you must use the Change Password option from the management console.</td>
</tr>
<tr>
<td>&lt;EveryOneRoleName&gt;</td>
<td>The name of the &quot;everyone&quot; role. All users in the system belong to this role.</td>
</tr>
</tbody>
</table>

Configuring the authorization manager

According to the default configuration in WSO2 products, the Users, Roles and Permissions are stored in the same repository (i.e., the default, embedded H2 database). However, you can change this configuration in such a way that the Users and Roles are stored in one repository (User Store) and the Permissions are stored in a separate repository. A user store can be a typical RDBMS, an LDAP or an external Active Directory.

The repository that stores Permissions should always be an RDBMS. The Authorization Manager configuration in the user-mgt.xml file (stored in the <PRODUCT_HOME>/repository/conf/ directory) connects the system to this RDBMS.

Follow the steps given below to set up and configure the Authorization Manager.

**Step 1: Setting up the repository**
By default, the embedded H2 database is used for storing permissions. You can change this as follows:

1. Change the default H2 database or set up another RDBMS for storing permissions.
2. When you set up an RDBMS for your system, it is necessary to create a corresponding datasource, which allows the system to connect to the database.
   - If you are replacing the default H2 database with a new RDBMS, update the `master-datasource.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/datasources/` directory) with the relevant information.
   - Alternatively, create a new XML file with the datasource information of your new RDBMS and store it in the same `<PRODUCT_HOME>/repository/conf/datasources/` directory.

Refer the related topics for detailed information on setting up databases and configuring datasources.

**Step 2: Updating the user realm configurations**

Once you have set up a new RDBMS and configured the datasource, the `user-mgt.xml` file (user realm configuration) should be updated as explained below.

1. Set up the database connection by update the datasource information using the `<Property>` element under `<Configuration>`. The jndi name of the datasource should be used to refer to the datasource. In the following example, the jndi name of the default datasource defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file is linked from the `user-mgt.xml` file.

   ```xml
   <Realm>
   <Configuration>
   ..........
   <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
   </Configuration>
   ...
   </Realm>
   ```

   You can add more configurations using the `<Property>` element:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>testOnBorrow</td>
<td>It is recommended to set this property to 'true' so that object connections will be validated before being borrowed from the JDBC pool. For this property to be effective, the <code>validationQuery</code> parameter in the <code>&lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml</code> file should be a non-string value. This setting will avoid connection failures. See the section on performance tuning of WSO2 products for more information.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

2. The default Authorization Manager section in the `user-mgt.xml` file is shown below. This can be updated accordingly.

   ```xml
   <AuthorizationManager
   class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
   <Property name="AdminRoleManagementPermissions">/permission</Property>
   <Property name="AuthorizationCacheEnabled">true</Property>
   </AuthorizationManager>
   ```

   - The `org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager` class enables the Authorization Manager for your product.
   - The `AdminRoleManagementPermissions` property sets the registry path where the authorization information (role-based permissions) are stored. Note that this links to the repository that you defined in Step 1.
   - It is recommended to enable the `GetAllRolesOfUserEnabled` property in the AuthorizationManager as follows:

     ```xml
     <Property name="GetAllRolesOfUserEnabled">true</Property>
     ```

Although using the user store manager does not depend on this property, you must consider enabling this if there are any performance issues in your production environment. Enabling this property affects the performance when the user logs in. This depends on the users, roles and permission stats.

- By default, the rules linked to a permission (role name, action, resource) are not case sensitive. If you want to make them case sensitive, enable the following property:
<Property name="CaseSensitiveAuthorizationRules">true</Property>

Related topics

1. See Maintaining Logins and Passwords for information on how to change the super admin credentials.

Changing the RDBMS

The default database of user manager is the H2 database that comes with WSO2 products. You can configure it to point to databases by other vendors.

1. Add the JDBC driver to the classpath by dropping the JAR into <PRODUCT_HOME>/repository/components/lib.
2. Change values of properties given in on the Realm Configuration page appropriately.
3. Create the database by running the relevant script in <PRODUCT_HOME>/dbscripts and restart the server:
   - For Linux: sh wso2server.sh or sh wso2server.sh
   - For Windows: wso2server.bat or wso2server.bat

Configuring Primary User Stores

Every WSO2 product comes with an embedded, internal user store, which is configured in <PRODUCT_HOME>/repository/conf/user-mgt.xml. In WSO2 Identity Server, the embedded user store is LDAP, and in other products it is JDBC. Because the domain name (unique identifier) of this default user store is set to PRIMARY by default, it is called the primary user store.

Instead of using the embedded user store, you can set your own user store as the primary user store. Since the user store you want to connect to might have different schemas from the ones available in the embedded user store, it needs to go through an adaptation process. WSO2 products provide the following adapters to enable you to authenticate users from different types of user stores and plug into LDAP, Active Directory, and JDBC to perform authentication:

<table>
<thead>
<tr>
<th>User store manager class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</td>
<td>Use ReadOnlyLDAPUserStoreManager to do read-only operations for external LDAP user stores.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldapReadWriteLDAPUserStoreManager</td>
<td>Use ReadWriteLDAPUserStoreManager for external LDAP user stores. This is the user store configuration uncommented in the code in the user-mgt.xml file.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager</td>
<td>Use ActiveDirectoryUserStoreManager to configure an Active Directory Service (AD DS) or Active Directory Lightweight Directory Service (AD LDS). This can be used only for read/write operations. If you need to use AD as read-only you must use org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</td>
<td>Use JDBCUserStoreManager for both internal and external JDBC stores.</td>
</tr>
</tbody>
</table>

The user-mgt.xml file already has sample configurations for all of the above user stores. To enable these configurations, you must uncomment them in the code and comment out the ones that you do not need.

The following topics provide details on the various primary user stores you can configure.

- Configuring an external LDAP or Active Directory user store
- Configuring an internal/external JDBC user store

If you are using ldaps (secured) to connect to the Active Directory as shown below, you need to import the certificate of Active Directory to the client-truststore.jks of the WSO2 product. See the topic on configuring keystores for information on how to add certificates to the trust-store.

(Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
Configuring an external LDAP or Active Directory user store

All WSO2 products can read and write users and roles from external Active Directory or LDAP user stores. You can configure WSO2 products to access these user stores in one of the following modes:

- Read-only mode
- Read/write mode

Read-only mode

Before you begin

- If you create the user-mgt.xml file yourself, be sure to save it in the <PRODUCT_HOME>/repository/conf directory.
- The class attribute for a read-only LDAP is `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">

When you configure a product to read users/roles from your company LDAP in read-only mode, it does not write any data into the LDAP.

1. Comment out the following user store which is enabled by default in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file.
   `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">

2. Given below is a sample for the LDAP user store. This configuration is found in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file, however, you need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.
Update the connection details to match your user store. For example:

```xml
<Property name="ConnectionURL">ldap://localhost:10389</Property>
```

Obtain a user who has permission to read all users/attributes and perform searches on the user store from your LDAP/Active Directory administrator. For example, if the privileged user is "AdminLDAP" and the password is "2010#Avrudu", update the following sections of the realm configuration as follows:

```xml
<Property name="ConnectionName">uid=AdminLDAP,ou=system</Property>
<Property name="ConnectionPassword">2010#Avrudu</Property>
```

Update with the directory name where the users are stored. When LDAP searches for users, it will start from this location of the directory.

```xml
<Property name="UserSearchBase">ou=system</Property>
```

Set the attribute to use as the username, typically either cn or uid for LDAP. Ideally, `<Property name="UserNameAttribute">` and `<Property name="UserNameSearchFilter">` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.
For example:

```xml
<Property name="UserNameAttribute">uid</Property>
```

e. Set the ReadGroups property to 'true', if it should be allowed to read roles from this user store. When this property is 'true', you must also specify values for the GroupSearchBase, GroupSearchFilter and NameAttribute properties as shown in the following example:

```xml
<Property name="ReadGroups">true</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
```

If the ReadGroups property is set to 'false', only Users can be read from the user store.

f. Optionally, configure the realm to read roles from the user store by reading the user/role mapping based on a membership (user list) or backlink attribute. The following code snippet represents reading roles based on a membership attribute. This is used by the Apache Directory server and OpenLDAP.

```xml
<Property name="ReadGroups">false</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MembershipAttribute">member</Property>
```

g. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the use store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.

3. Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

**Read/write mode**

**Before you begin**

- To read and write to an Active Directory user store, set the ReadGroups property to `true` instead of `false`.
- To write user entries to an LDAP user store (roles are not written, just user entries), you follow the steps in the **Read-only mode** section but specify the following class instead:

```xml
<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
```

- Use the following class for Active Directory.

```xml
<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
```

The `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file has commented-out configurations for external LDAP/AD user stores.

1. Enable the `<ReadWriteLDAPUserStoreManager>` or the `<ActiveDirectoryUserStoreManager>` in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file by uncommenting the code. When it is enabled, the user manager reads/writes into the LDAP/AD user store. Note that these configurations already exist in the `user-mgt.xml` file so you only need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.
2. The default configuration for the external read/write user store in the `user-mgt.xml` file is as follows. Change the values according to your requirements.
LDAP User Store Active Directory User Store

LDAP user store sample:

```xml
<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
  <Property name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
  <Property name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServerPort}</Property>
  <Property name="ConnectionName">uid=admin,ou=system</Property>
  <Property name="PasswordHashMethod">SHA</Property>
  <Property name="UserNameListFilter">(objectClass=person)</Property>
  <Property name="UserEntryObjectClass">wso2Person</Property>
  <Property name="UserNameSearchFilter">(&amp;(objectClass=person)(uid=?))</Property>
  <Property name="UserNameAttribute">uid</Property>
  <Property name="PasswordJavaScriptRegEx">\S{5,30}</Property>
  <Property name="UsernameJavaScriptRegEx">\S{3,30}</Property>
  <Property name="RolenameJavaScriptRegEx">\S{3,30}</Property>
  <Property name="ReadGroups">true</Property>
  <Property name="WriteGroups">true</Property>
  <Property name="EmptyRolesAllowed">true</Property>
  <Property name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
  <Property name="GroupEntryObjectClass">groupOfNames</Property>
  <Property name="GroupNameSearchFilter">(&amp;(objectClass=groupOfNames)(cn=?))</Property>
  <Property name="GroupNameAttribute">cn</Property>
  <Property name="GroupNameSearchBase">ou=Groups,dc=wso2,dc=org</Property>
  <Property name="GroupNameSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
  <Property name="GroupNameEntryObjectClass">groups</Property>
  <Property name="GroupNameAttribute">member</Property>
  <Property name="GroupNameAttribute">cn</Property>
  <Property name="UserRolesCacheEnabled">true</Property>
  <Property name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</Property>
</UserStoreManager>
```

**Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up:

`APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

Active directory user store sample:
<UserStoreManager class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
  <Property name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
  <Property name="defaultRealmName">WSO2.ORG</Property>
  <Property name="KdcEnabled">false</Property>
  <Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
  <Property name="ConnectionName">CN=admin,CN=Users,DC=WSO2,DC=Com</Property>
  <Property name="ConnectionPassword">Alb2c3d4</Property>
  <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
  <Property name="UserSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
  <Property name="UserNameAttribute">cn</Property>
  <Property name="isADLDSRole">false</Property>
  <Property name="UserAccountControl">512</Property>
  <Property name="UserNameListFilter">(objectClass=user)</Property>
  <Property name="UserNameSearchFilter">(&(objectClass=user)(cn=?))</Property>
  <Property name="UsernameJavaScriptRegEx">^[\S]{3,30}$</Property>
  <Property name="PasswordJavaScriptRegEx">^[\S]{5,30}$</Property>
  <Property name="RolenameJavaScriptRegEx">^[\S]{3,30}$</Property>
  <Property name="GroupSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
  <Property name="GroupNameAttribute">cn</Property>
  <Property name="GroupNameListFilter">(objectcategory=group)</Property>
  <Property name="GroupNameSearchFilter">(&(objectcategory=group)(cn=?))</Property>
  <Property name="UserRolesCacheEnabled">true</Property>
  <Property name="Referral">follow</Property>
  <Property name="BackLinksEnabled">true</Property>
  <Property name="MaxRoleNameListLength">100</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="SCIMEnabled">false</Property>
</UserStoreManager>

**Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up - `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

When working with Active Directory it is best to enable the `GetAllRolesOfUserEnabled` property in the AuthorizationManager as follows.

```xml
<AuthorizationManager class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
  <Property name="AdminRoleManagementPermissions">/permission</Property>
  <Property name="AuthorizationCacheEnabled">true</Property>
  <Property name="GetAllRolesOfUserEnabled">true</Property>
</AuthorizationManager>
```

While using the user store manager does not depend on this property, you must consider enabling this if there are any performance issues in your production environment. Enabling this property affects the performance when the user logs in. This depends on the users, roles and permissions stats.
3. Set the attribute to use as the username, typically either cn or uid for LDAP. Ideally, `<Property name="UserNameAttribute">` and `<Property name="UserNameSearchFilter">` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

```
LDAP Active Directory
```

```
<Property name="UserNameAttribute">uid</Property>
```

```
<Property name="UserNameAttribute">sAMAccountName</Property>
```

4. The following code snippet represents reading roles based on a backlink attribute. This is used by the Active Directory.

```
<Property name="ReadGroups">true</Property>
<Property name="GroupSearchBase">cn=users,dc=wso2,dc=lk</Property>
<Property name="GroupSearchFilter">(objectcategory=group)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MemberOfAttribute">memberOf</Property>
```

5. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the use store configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.

6. Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

When configuring an external LDAP for Governance Registry or API Manager, the user name and password for the default admin will change to the LDAP admin. As a result, the `<PRODUCT_HOME>/repository/conf/api-manager.xml` file must be updated with the new LDAP admin credentials.

### Configuring an internal/external JDBC user store

The default internal JDBC user store reads/writes into the internal database of the Carbon server. JDBC user stores can be configured using the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file's JDBCUserStoreManager configuration section. Additionally, all Carbon-based products can work with an external RDBMS. You can configure Carbon to read users/roles from your company RDBMS and even write to it. Therefore, in this scenario, the user core connects to two databases:

- The Carbon database where authorization information is stored internally.
- Your company database where users/roles reside.

Therefore, the `user-mgt.xml` file must contain details for two database connections. The connection details mentioned earlier are used by the authorization manager. If we specify another set of database connection details inside the UserStoreManager, it reads/writes users to that database. The following are step-by-step guidelines for connecting to an internal and external JDBC user store in read-only mode:

1. Uncomment the following section in `<PRODUCT_HOME>/repository/conf/user-mgt.xml`:

```
Ensure that you comment out the configurations for other user stores which you are not using when uncommenting JDBCUserStoreManager.
```
The following are samples for the internal and external JDBC user store configuration:

**Internal JDBC User Store**

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
  <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
  <Property name="ReadOnly">false</Property>
  <Property name="WriteGroups">true</Property>
  <Property name="UsernameJavaRegEx">^\[\S\]{3,30}$</Property>
  <Property name="UsernameJavaScriptRegEx">^\[\S\]{3,30}$</Property>
  <Property name="UsernameJavaRegExViolationErrorMsg">Username pattern policy violated</Property>
  <Property name="PasswordJavaRegEx">^\[\S\]{5,30}$</Property>
  <Property name="PasswordJavaScriptRegEx">^\[\S\]{5,30}$</Property>
  <Property name="PasswordJavaRegExViolationErrorMsg">Password length should be within 5 to 30 characters</Property>
  <Property name="RolenameJavaRegEx">^\[\S\]{3,30}$</Property>
  <Property name="RolenameJavaScriptRegEx">^\[\S\]{3,30}$</Property>
  <Property name="CaseInsensitiveUsername">true</Property>
  <Property name="SCIMEnabled">false</Property>
  <Property name="IsBulkImportSupported">true</Property>
  <Property name="PasswordDigest">SHA-256</Property>
  <Property name="StoreSaltedPassword">true</Property>
  <Property name="MultiAttributeSeparator">,</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="MaxRoleNameListLength">100</Property>
  <Property name="UserRolesCacheEnabled">true</Property>
  <Property name="UserNameUniqueAcrossTenants">false</Property>
  <Property name="SelectUserSQL">SELECT * FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
  <Property name="GetRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM..."
1. `WHERE UM_ROLE_NAME LIKE ? AND UM_TENANT_ID=? AND UM_SHARED_ROLE = '0' ORDER BY UM_ROLE_NAME`<Property>
   
   `SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_SHARED_ROLE = '1' ORDER BY UM_ROLE_NAME`<Property>
   
   
   `SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?`<Property>
   
   
   
   `SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?`<Property>
   
   
   
   
   `SELECT DISTINCT UM_PROFILE_ID FROM UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=?`<Property>
   
   `SELECT DISTINCT UM_PROFILE_ID FROM UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=?`<Property>
   
   `SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?`<Property>
   
   `SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?`<Property>
   
   `SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?`<Property>
   
   `INSERT INTO UM_USER (UM_USER_NAME, UM_USER_PASSWORD, UM_SALT_VALUE, UM_REQUIRE_CHANGE, UM_CHANGED_TIME, UM_TENANT_ID) VALUES (?, ?, ?, ?, ?, ?)`<Property>
   
   `INSERT INTO UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?),(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?),?)`<Property>
   
   `INSERT INTO UM_ROLE (UM_ROLE_NAME, UM_TENANT_ID) VALUES (?,?)`<Property>
   
   `UPDATE UM_ROLE SET UM_SHARED_ROLE = ? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID = ?`<Property>
   
   `INSERT INTO UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?),(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?),?)`<Property>
   
   `INSERT INTO UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?),(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?),?)`
<Property name="AddSharedRoleToUserSQL">INSERT INTO UM_SHARED_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_USER_TENANT_ID, UM_ROLE_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?)</Property>

<Property name="RemoveUserFromSharedRoleSQL">DELETE FROM UM_SHARED_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?);</Property>

<Property name="RemoveUserFromRoleSQL">DELETE FROM UM_USER_ROLE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?);</Property>

<Property name="RemoveRoleFromUserSQL">DELETE FROM UM_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?);</Property>

<Property name="DeleteRoleSQL">DELETE FROM UM_ROLE WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID=?;</Property>

<Property name="OnDeleteRoleRemoveUserRoleMappingSQL">DELETE FROM UM_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?;</Property>

<Property name="DeleteUserSQL">DELETE FROM UM_USER WHERE UM_USER_NAME = ? AND UM_TENANT_ID=?;</Property>

<Property name="OnDeleteUserRemoveUserRoleMappingSQL">DELETE FROM UM_USER_ROLE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?;</Property>


<Property name="UpdateRoleNameSQL">UPDATE UM_ROLE set UM_ROLE_NAME=? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>

<Property name="AddUserPropertySQL">INSERT INTO UM_USER_ATTRIBUTE (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?, ?, ?)</Property>


<Property name="UserNameUniqueAcrossTenantsSQL">SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=?</Property>

<Property name="IsDomainExistingSQL">SELECT UM_DOMAIN_ID FROM UM_DOMAIN WHERE UM_DOMAIN_NAME=? AND UM_TENANT_ID=?</Property>

<Property name="AddDomainSQL">INSERT INTO UM_DOMAIN (UM_DOMAIN_NAME, UM_TENANT_ID) VALUES (?, ?)</Property>

<Property name="AddUserRoleToRole-mssql">INSERT INTO UM_USER_ROLE (UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), (?)</Property>

<Property name="AddRoleToUser-mssql">INSERT INTO UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (?)</Property>


UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
WSO2 API Manager, WSO2 Inc.

Copyright © WSO2 Inc. 2015-2018

489

The sample for the external JDBC user store consists of properties pertaining to various SQL statements. This is because the schema may be different for an external user store, and these adjustments need to be made to streamline the configurations with WSO2 products.

You can define a data source in <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml and refer to it from the user-mgt.xml file. This takes the properties defined in the master-datasources.xml file and reuses them in the user-mgt.xml file. To do this, you need to define the following property:

```xml
<Property name="dataSource">jdbc/WSO2CarbonDB</Property>
```

2. Find a valid user that resides in the RDBMS. For example, say a valid username is AdminSOA. Update the Admin user section of your configuration as follows. You do not have to update the password element; leave it as is.

```xml
<AdminUser>
    <UserName>AdminSOA</UserName>
    <Password>XXXXXX</Password>
</AdminUser>
```

3. Add the PasswordHashMethod property to the UserStoreManager configuration for JDBCUserStoreManager. For example:

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="PasswordHashMethod">SHA</Property>
    ...
</UserStoreManager>
```

The PasswordHashMethod property specifies how the password should be stored. It usually has the following values:

- SHA - Uses SHA digest method.
- MD5 - Uses MD5 digest method.
- PLAIN_TEXT - Plain text passwords.

In addition, it also supports all digest methods in http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html.

4. Update the connection details found within the <UserStoreManager> class based on your preferences. For more information on parameters need to be configured refer Configuring a JDBC User Store.

5. In the realm configuration section, add the property MultiTenantRealmConfigBuilder and set the value to org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder in order to construct tenant specific realm configurations.

For example:

```xml
<Property name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
```

6. Add the JDBC driver to the classpath by copying its JAR file into the <PRODUCT_HOME>/repository/components/lib directory.

7. Edit the SQLs in the user-mgt.xml file according to your requirements, and then start the server.

### Related Links

- For a comprehensive understanding on the configuration details, see Working with Properties of User Stores in the WSO2 Administration guide.
- For details on writing a simple custom user store manager for WSO2 products, see Writing a Custom User Store Manager in the WSO2 Administration guide.
Customizing the API Store

You can customize the API Store in the following ways:
- Enabling or disabling self signup
- Enabling or disabling the walkthrough
- Enabling or disabling the forum
- Changing the theme
- Changing language settings
- Setting single login for all apps
- Categorizing and grouping APIs
- Customizing the API group
- Customizing error Pages

Enabling or disabling self signup

In a multi-tenanted API Manager setup, self signup to the API Store is disabled by default to all tenants except the super tenant. A tenant admin can enable it as follows:

1. Sign in to the management console (https://<HostName>:9443/carbon) as admin (or tenant admin).
2. In the Main menu, click Add under Users and Roles.
3. Click Add New Role.
4. Add a role by the name subscriber (or any other name you prefer).
5. Click **Next** and add the following permissions:

6. Go to the **Resources > Browse** menu.
7. Navigate to the `/system/governance/apimgt/applicationdata/` directory.
8. Click on `sign-up-config.xml` to load the resource in the registry browser UI and select the "Edit as text " option to edit the configurations.
9. Do the following changes in the signup configuration and save.
   - Set `<EnableSignup>` to `true`.
   - Set `<RoleName>` to `subscriber` and `<IsExternalRole>` to `true`. Note that you must have the subscriber role created at this point.
   - Set `<AdminUserName>` and `<AdminPassword>` to the credentials of the super admin, or if you are in a multitenant setup and you are not the super admin, to the tenant admin's credentials. Note that the super admin's credentials are admin/admin by default. If you changed the default super admin's credentials, using admin/admin will cause errors.

```
<selfsignup>
  <enablesignup>true</enablesignup>
  <!-- user storage to store users -->
  <signupdomain>PRIMARY</signupdomain>
  <!-- Tenant admin information. (for clustered setup credentials for AuthManager) -->
  <adminusername>xxxx</adminusername>
  <adminpassword>xxxx</adminpassword>
  <!-- List of roles for the tenant user -->
  <signuproles>
    <signuprole>
      <rolename>subscriber</rolename>
      <isexternalrole>true</isexternalrole>
    </signuprole>
  </signuproles>
</selfsignup>
```

10. Restart the server and open the API Store (https://<HostName>:9443/store)
    Note the Sign-up link that appears in the top, right-hand corner of the window.
11. To disable the self signup capability, navigate to the `/system/governance/apimgt/applicationdata/sign-up-config.xml` file in the registry again and set the `<SelfSignUp><EnableSignup>` element to false.

**Tip:** To engage your own signup process, see *Adding a User Signup Workflow*.

**Enabling or disabling the walkthrough**

To disable the API Store walkthrough, open the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/interactiveTutorial.json` file.

Set the `isEnabledTutorial` parameter to false as show below.

```
{
  "isEnabledTutorial" : false,
  "blackListedTenantDomains" : []
}
```

**Enabling or disabling the forum**

The Forum is enabled by default in the API Manager Store.
Follow the instructions below to disable the Forum:

1. Navigate to the `<API_HOME>/repository/conf/api-manager.xml` file.
2. Uncomment the following code.
   ```xml
   <isStoreForumEnabled>false</isStoreForumEnabled>
   ```
3. Restart WSO2 API Manager.
   If you access the API Store, you will notice that the Forum is no longer available.

**Changing the theme**

See Adding a New API Store Theme.

**Changing language settings**

To change the language of the API Store, see Adding Internationalization and Localization.

**Setting single login for all apps**

Single sign-on (SSO) allows users who are logged in to one application to automatically log in to multiple other applications using the same credentials. They do not have to repeatedly authenticate themselves. To configure, see Configuring Single Sign-on with SAML2.

**Categorizing and grouping APIs**

API providers add tags to APIs when designing them using the API Publisher. Tags allow API providers to categorize APIs that have similar attributes. When a tagged API gets published to the API Store, its tags appear as clickable links to the API consumers, who can use them to quickly jump to a category of interest. The font size of the tag in the Store varies based on the number of APIs that are assigned to it. Therefore, for example the font size of a tag which has 10 APIs assigned to it will be bigger than the font size of a tag that has only 2 APIs assigned to it.
If you want to see the APIs grouped according to different topics in the API Store, add an API group:

Although the way in which you add a Tag and API group appears to be similar there are differences. Therefore, you need to note the following:

- **The group name should always have the suffix** `-group` **and it can have spaces** in it (e.g., APIs groups-group).
- **The tag name should not have a suffix or prefix**, but it **can have spaces**.

1. Go to `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf` directory, open the `site.json` file and set the `tagWiseMode` attribute as true.
2. Add an API group to the APIs that you wish to group.
   a. Go to the API Publisher (https://<HostName>:9443/publisher).
   b. Click on the edit link of the respective API as shown below.
c. Add a group name to the APIs that you wish to group.

For example add the "APIs groups-group" tag to the Workflow and Integration APIs.

d. Save the API for the tag to appear in the Store.

e. Repeat steps 2 (a) to (d) to add another APIs to the newly created group.

Sign in to the API Store and note the API groups.

If you wish, you can click on a group to see the APIs that belong to a specific group.
Customizing the API group

If you want to change the descriptions and the thumbnail images that come by default, do the following:

1. Sign in to the Management Console and click the Resources > Browse menu to open the registry.

2. Create a collection named tags under the registry location /_system/governance/apimgt/applicationdata.
3. Give read permission to the `system/wso2.anonymous.role` role.

4. Add each tag as collections under the tags collection (e.g., Workflow APIs-group, Integration APIs-group, Quote APIs-group.)

5. Navigate to each tag collection and upload the following:
   - `description.txt` with the description of the tag
   - `thumbnail.png` for the thumbnail image

6. Back in the API Store, note the changes you did in the registry.

**Customizing error Pages**

In API Manager store/publisher and admin webapps, `jaggery.conf` is the Jaggery configuration file specifies the application specific configurations. In that file we can find following code block which have configured the error pages.
If such a specified error occurs due to an operation or page redirection in the web application, it redirects to the specified html page. As an example, if you request for [https://localhost:9443/store/site/conf.site.json](https://localhost:9443/store/site/conf.site.json), it gives a 403 response, it serves the html page site/pages/error-pages/403.html specified above.

### Error 403 : Forbidden

You don't have permission to access anything with that kind of request.

---

These error pages are located in `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/pages/error-pages` directory. You can customize these html pages according to your preference (adding css, javascript or jQuery functionalities). And also you can create your own html pages to be viewed for errors occurred by adding it to the jaggery.conf.

### Directing the Root Context to the API Store

WSO2 API Manager includes separate Web applications as the API Publisher and the API Store. The root context of the API Manager is set to go to the API Publisher by default. For example, assume that the API Manager is hosted on a domain named `apis.com` with default ports. The URLs of the API Store and API Publisher will be as follows:

- API Store - [https://apis.com:9443/store](https://apis.com:9443/store)
- API Publisher - [https://apis.com:9443/publisher](https://apis.com:9443/publisher)

If you open the root context, which is [https://apis.com:9443](https://apis.com:9443) in your browser, it directs to the API Publisher by default. You can set this to go to the API Store as follows:

1. Open the bundle `<AM_HOME>/repository/components/plugins/org.wso2.am.styles_l.x.x.jar`.
2. Open the `component.xml` file that is inside META-INF directory.
3. Change the `<context-name>` element, which points to publisher by default, to store:

```
<context>
  <context-id>default-context</context-id>
  <context-name>store</context-name>
  <protocol>http</protocol>
  <description>API Publisher Default Context</description>
</context>
```
4. Compress the JAR and put it back in the `<API-M_HOME>/repository/components/plugins` directory.
5. Restart the server.
6. Open the default context (https://apis.com:9443) again in a browser and note that it directs to the API Store.

Tip: If you want to configure the API Publisher and Store to pass proxy server requests, configure a reverse proxy server.

Enabling Access Control Support for API Publisher

Visibility settings prevent certain user roles from viewing and modifying APIs created by another user role. This feature allows you to restrict the ability to view and modify APIs for a set of users.

- Enabling Access Control
- Using the API Publisher UI
- Using the REST API

Enabling Access Control

To enable this feature, open the `<API-M_HOME>/repository/conf/api-manager.xml` file. Add the code given below under `<APIPublisher>`.

```xml
<APIPublisher>
  ...
  <EnableAccessControl>true</EnableAccessControl>
</APIPublisher>
```

Instructions to existing users

Re-indexing the artifacts in the registry.

- Rename the `<lastAccessTimeLocation>` element in the `<API-M_HOME>/repository/conf/registry.xml` file. If you use a clustered/distributed API Manager setup, change the file in the API Publisher node. For example, change the `<lastAccessTimeLocation>` path to `/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstimexyz`.

```xml
<lastAccessTimeLocation>/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstimexyz
</lastAccessTimeLocation>
```

- Shut down API Manager if you have already started it, backup and delete the `<API-M_HOME>/solr` directory if it exist

Note that the registry indexing takes some time depending on the number of APIs you have in your store, so the existing APIs may not appear if you are accessing the publisher/store immediately after you start the server.

Restart the server after doing these changes.

Using the API Publisher UI

1. Log in to API Publisher as an API Creator. For more information on User Roles, see Managing Users and Roles.
2. Create an API. Select Restricted by roles for Access Control in the Design tab.
3. Add the roles that have permission to view or modify this API.

Using the REST API

You can use the existing REST API to add a new API. To create an API with publisher access control restriction, add the two elements shown below in your request body,

```json
"accessControl" : "RESTRICTED",
"accessControlRoles" : ["admin"]
```

Note that the roles should be valid. If the API creator is not an API-M admin he/she should at least have one of his/her roles in the accessControlRoles field.

The publisher role cache is enabled by default in API Manager. This is to avoid sending repeated requests to the Key Manager node in a distributed deployment, to authenticate user roles.

This WUM update allows you to disable the feature by disabling `<EnablePublisherRoleCache>` under `<CacheConfigurations>`. We recommend enabling the elements shown in the example below.
Enabling CORS for APIs

Cross-Origin Resource Sharing (CORS) is a mechanism that allows accessing restricted resources (i.e., fonts, images, scripts, videos and iframes) from domains outside the domain from which the requesting resource originated. By default, web browsers apply the same-origin policy to avoid interactions between different origins. CORS defines a way in which a browser and a server can interact to determine whether or not it is safe to allow the cross-origin requests.

In API Manager, you can enable Cross-Origin Resource Sharing per API or as a global configuration that is applied across all APIs.

- Enabling CORS Globally
- Enabling CORS Per API

Enabling CORS Globally

You can enable CORS globally for API Manager by configuring api-manager.xml located in `<API-M_HOME>/repository/conf` directory.

Follow the steps below to enable CORS response headers globally. Once this configuration is enabled, it will be applied across all the APIs served by the API Gateway.

1. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file.
2. Locate the following configuration and set the `<Enabled>` attribute to `true` with the required CORS headers in the response. Once this configuration is applied in the API Gateway, it will affect all the API calls served by the Gateway.

```xml
<CacheConfigurations>
  <EnablePublisherRoleCache>true</EnablePublisherRoleCache>
  ...
</CacheConfigurations>

<!-- Configuration to enable/disable sending CORS headers in the Gateway response and define the Access-Control-Allow-Origin header value.-->  
<!-- Configuration to enable/disable sending CORS headers from the Gateway-->  
<Enabled>true</Enabled>
<!-- The value of the Access-Control-Allow-Origin header. Default values are API Store addresses, which is needed for swagger to function. -->  
<Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
<!-- Configure Access-Control-Allow-Methods -->  
<Access-Control-Allow-Methods>GET,PUT,POST,DELETE,PATCH,OPTIONS</Access-Control-Allow-Methods>
<!-- Configure Access-Control-Allow-Headers -->  
<Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction</Access-Control-Allow-Headers>
<!-- Configure Access-Control-Allow-Credentials -->  
<Access-Control-Allow-Credentials>false</Access-Control-Allow-Credentials>
</CORSConfiguration>

Note that if disabled it results in lowering performance due to repeatedly accessing the Key Manager.

Enabling CORS Per API

It is required to enable CORS globally before you enable CORS Per API. Therefore if you haven’t done it yet, follow the steps in Enabling CORS Globally before starting the below steps.

1. Sign in to API Publisher and choose to design a new API.
2. Click **Start Creating**.
3. Give the information in the table below and click **Add** to add the resource.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>WeatherAPI</td>
</tr>
<tr>
<td>Context</td>
<td>/weather</td>
</tr>
<tr>
<td>Version</td>
<td>v1.0.0</td>
</tr>
<tr>
<td>Resources</td>
<td>URL Pattern: current/(country)/(zipcode)</td>
</tr>
<tr>
<td></td>
<td>Request types: GET method to return the current weather conditions of a zip code that belongs to a particular country</td>
</tr>
</tbody>
</table>
4. Once done, click Next: Implement >
5. In the Implementation tab, provide the following endpoint details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint type</td>
<td>HTTP/REST endpoint</td>
</tr>
<tr>
<td>Production endpoint</td>
<td>You can find the Yahoo weather API's endpoint from <a href="https://developer.yahoo.com/weather/">https://developer.yahoo.com/weather/</a>. Copy the part before the '?' sign to get this URL: <a href="https://query.yahooapis.com/v1/public/yql">https://query.yahooapis.com/v1/public/yql</a></td>
</tr>
</tbody>
</table>

6. Select the Enable API based CORS Configuration check box to enable CORS for the API.
7. Once you enable CORS, you will be able to see the CORS response header configuration section. Listed below are the CORS specific response headers supported by the API Gateway and how to configure them.

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
<th>Sample values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access-Control-Allow-Origin</td>
<td>Determines whether a resource can be shared with the resource of a given origin. The API Gateway validates the origin request header value against the list of origins defined under the Access Control Allow Origins configuration (this can be All Allow Origins or a specific value like localhost). If the host is in the allowed origin list, it will be set as the Access-Control-Allow-Origin response header in the response.</td>
<td>All Allow Origins(*), localhost</td>
</tr>
<tr>
<td>Access-Control-Allow-Headers</td>
<td>Determines, as part of the response to a preflight request (a request that checks to see if the CORS protocol is understood), which header field names can be used during the actual request. The gateway will set the header values defined under Access Control Allow Headers configurations.</td>
<td>authorization, Access-Control-Allow-Origin, Content-type, SOAPAction</td>
</tr>
<tr>
<td>Access-Control-Allow-Methods</td>
<td>This header specifies the method(s) allowed when accessing the resource in response to a preflight request. Required methods can be defined under the Access Control Allow Method configuration.</td>
<td>GET, PUT, POST, DELETE, PATCH, OPTIONS</td>
</tr>
<tr>
<td>Access-Control-Allow-Credentials</td>
<td>Determines whether or not the response to the request can be exposed to the page. It can be exposed when the header value is true. The header value can be set to true/false by enabling/disabling the Access Control Allow Credentials configuration.</td>
<td>true, false</td>
</tr>
</tbody>
</table>

8. Once the CORS configurations are done, click Next: Manage >.

9. Select the Unlimited subscription tier and click Save and Publish to create and publish the API to the API Store.
You have successfully enabled CORS for a specific API.

### Enabling Notifications

You can enable notifications at the creation of new API versions in order to notify existing subscribers (via email) that a new version of the API is available. If you want to send a different type of notification, you can extend the `org.wso2.carbon.apimgt.impl.notification.Notifier` abstract class on your own accordingly.

Follow the instructions below to enable notifications for new API versions:

1. Set the email server configurations in the `<API-M_HOME>/repository/conf/output-event-adapters.xml` file under the `<adapterConfig type="email">` section.
<adapterConfig type="email">
  <!-- Comment mail.smtp.user and mail.smtp.password properties to support connecting
  SMTP servers which use trust based authentication rather username/password authentication -->
  <property key="mail.smtp.from">abcd@gmail.com</property>
  <property key="mail.smtp.user">abcd</property>
  <property key="mail.smtp.password">xxxx</property>
  <property key="mail.smtp.host">smtp.gmail.com</property>
  <property key="mail.smtp.port">587</property>
  <property key="mail.smtp.starttls.enable">true</property>
  <property key="mail.smtp.auth">true</property>
  <!-- Thread Pool Related Properties -->
  <property key="minThread">8</property>
  <property key="maxThread">100</property>
  <property key="keepAliveTimeInMillis">20000</property>
  <property key="jobQueueSize">10000</property>
</adapterConfig>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail.smtp.from</td>
<td>The email address you use to send emails</td>
</tr>
<tr>
<td>mail.smtp.user</td>
<td>The email address used to authenticate the mail server. This can be same as mail.smtp.from</td>
</tr>
<tr>
<td>mail.smtp.password</td>
<td>Password used to authenticate the mail server.</td>
</tr>
</tbody>
</table>

2. Log in to the Management Console and click **Main > Resource > Browse**.
3. Browse to the `/_system/config/apimgt/applicationdata/tenant-conf.json` file and click **Edit as Text**.
4. Set the `NotificationsEnabled` property to `true` as shown below:

```json
"NotificationsEnabled": "true",
"Notifications": [{
  "Type": "new_api_version",
  "Notifiers": [{
    "Class": "org.wso2.carbon.apimgt.impl.notification.NewAPIVersionEmailNotifier",
    "ClaimsRetrieverImplClass": "org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever",
    "Title": "Version $2 of $1 Released",
    "Template": "<html> <body> <h3 style="color:Black;">We're happy to announce the arrival of the next major version $2 of $1 API which is now available in Our API Store.</h3><a href="https://localhost:9443/store">Click here to Visit WSO2 API Store</a></body></html>"
  }
}]
```

A notification type can have multiple notifier classes that help send multiple notifications. In this case, notification sends via EMail but it could be SMS notification. Each notifier has a class attribute containing the full class path. The following properties should be set for the default `NewAPIVersionEmailNotifier` class:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>The full class path of the notifier class.</td>
</tr>
</tbody>
</table>
Subscriber email addresses are extracted from user claims. The default pack uses the org.wso2.carbon.apimg.t.impl.token.DefaultClaimsRetriever class to read the claim values from the user store.

**Claims**

A claim is a piece of information about a particular subject. It can be anything that the subject is owned by or associated with, such as name, group, preferences, etc. For information on how to add new claim mappings, see Configuring Claims.

<table>
<thead>
<tr>
<th>Title</th>
<th>The subject of the email.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>The template of the email body. This can be string values or a valid registry path to a template file.</td>
</tr>
</tbody>
</table>

The following strings are replaced with API specific values in the Title and Template properties.

- $1 - API name
- $2 - New API version

If you create the subscriber in the Management Console, you need to add the subscriber’s email in the subscriber user profile. You can find the user profile when you list the users in the management console.

If you are using a Google mail account, note that Google has restricted third-party apps and less secure apps from sending emails by default. Therefore, you need to configure your account to disable this restriction. For more information about Setting Gmail, see Creating Users using the Ask Password Option.

**Generating SDKs**

Software Development Kits (SDKs) contain the necessary toolkits to create a client application to invoke a particular API. If an API consumer wants to create an application, they can generate a server stub or client side SDK for a supported language/framework and use it to write a software application to consume the subscribed APIs.

SDK generation is not supported with the APIs that are created using OpenAPI 3.0 support.

- Generating client SDKs in the API Store
- Generating server stubs and client SDKs in the API Publisher

**Generating client SDKs in the API Store**

**Tip:** A valid API subscription must exist in order to use the SDK. SDK generation is allowed per API.

By default, SDK generation is enabled for Java and Android. If those two languages are sufficient, move to step 3. To enable SDK generation for other languages,

1. Open the `<APIM_Home>/repository/conf/api-mamanger.xml` file and edit the `<SupportedLanguages>` property of the `<SwaggerCodegen>` element to include any of the supported languages/frameworks. An example is shown below,
<SwaggerCodegen>
   <ClientGeneration>
      <GroupId>org.wso2</GroupId>
      <ArtifactId>org.wso2.client.</ArtifactId>
      <ModelPackage>org.wso2.client.model.</ModelPackage>
      <ApiPackage>org.wso2.client.api.</ApiPackage>
      <!-- Configure supported languages/Frameworks as comma separated values, Supported Languages/Frameworks : android, java, scala, csharp, cpp, dart, flash, go, groovy, javascript, jmeter, nodejs, perl, php, python, ruby, swift, clojure, aspNet5, asyncScala, spring, csharpDotNet2, haskell--> <SupportedLanguages>java,android,ruby,php</SupportedLanguages>
   </ClientGeneration>
</SwaggerCodegen>

In this example, the supported languages are defined as Java, Android, Ruby and PHP.

2. Once the above configuration is saved, start the API Manager server.
3. Log in to the API Store, open the API and click the **SDKs** tab. Download options are available for each programming language specified earlier in the `api-manager.xml` file.

4. Download the SDK for the preferred language and use it to write software applications to consume the API.

   **Tip:** Make sure that you are logged in to the API Store before attempting to download the SDK. As mentioned above, a valid subscription should exist for the API. The SDK cannot be used without a valid access token for the subscription.

---

**Generating server stubs and client SDKs in the API Publisher**

The API Publisher has an embedded **swagger editor** with the ability to generate server code and client SDKs then and there.

Client SDK and Server Stub generation in API Publisher is only supported for Rest APIs.

1. Open an existing API and choose to edit it.
2. Click **Edit Source** to open the embedded swagger editor.

3. To generate and download a server stub, click **Generate Server** and select a server stub from the list. API developers can use the **Generate Server** option to generate the REST API structure based on the swagger definition. The actual backend implementation can be developed on top of the code generated using swagger code generator. You can select from a list of frameworks to generate the actual backend implementation stub of the REST API.
4. To generate and download a client SDK, click **Generate Client** and select a client from the list.
The generated server stubs and client SDKs are generated using Swagger Codegen.

Logging

Logging is one of the most important monitoring tools of a production server. A properly configured logging system is vital for identifying errors, security threats, and usage patterns. WSO2 API Manager uses a log4j based logging mechanism through the Apache Commons Logging facade library. The log4j.properties file, which governs how logging is performed by the server, is in the <API-M_HOME>/repository/conf directory. In general, you should not modify the log4j.properties file directly. Instead, you set up and modify logging using the API-M management console. The settings in the management console override the settings in the log4j.properties file.

A logger is used to log messages for a specific system or application component. Loggers are normally named using a hierarchical, dot-separated namespace and have a child-parent relationship. For example, the logger named root.sv is a parent of the logger named root.sv.sf and a child of root.sv.sf.sf.

It is recommended to add the `securityDefinitions` in the swagger definition to be able to pass access tokens when invoking an API. Edit the source of the API from the API Publisher and add the code given below.

```json
securityDefinitions:
  default:
    type: oauth2
    authorizationUrl: 'https://<GW-HOST>:<GW-PORT>/authorize'
    flow: implicit
    scopes: {}
  security:
    - default: []

write a client application using the SDK

execute a client application using the SDK

What's Next

Write a Client Application Using the SDK
The following topics provide more information about logging:

- Application Logs
- Setting Up Logging
- System Logs
- Monitoring Access Logs

### Application Logs

Application logs are events that are recorded when they are invoked by an application or a program running in a system. Similarly, the application logs of a running Carbon instance display the log events of its deployed web applications and web services. The Application Logs page has been introduced as a fine-grained view of system logs. While system logs display log events of the entire system holistically, the application logs page allows the user to select a particular application and view only those logs.

The application logs show logs from the proxy service, not from any logs inside its sequences. To view logs from the log mediator, see System Logs.

The log files can be retrieved in two ways:

- If syslog-ng is configured, log files are taken from the remote location where the log files are hosted using the syslog-ng server.
- If syslog-ng is not configured, log files are taken from the local file system (super-tenant or Stand-alone apps).

For more information on logs and how to change log properties according to your preferences, see Logging. Follow the instructions below to access statistics on application logs.

1. Log in to the product's Management Console and click Monitor > Application Logs.

2. The Application Logs page appears. This page displays logs of a selected application in a bottom-up manner. For example,
2. Use the drop-down list shown below to select a deployed web service or web application to view its log files.

Application Logs

3. In the View list, select the category of logs you want to view. The available categories are:
   - TRACE - Trace messages
   - DEBUG - Debug messages
   - INFO - Information messages
   - WARN - Warning messages
   - ERROR - Error messages
   - FATAL - Fatal error messages
   - ALL - Displays all categories of logs

   For example,

Application Logs

4. You can also search for a specific log using the search function. Enter a keyword (or part of a keyword) and click Search. When a search criteria is given, the View field is displayed as Custom.

For example,
Setting Up Logging

There are two ways to configure log4j logging in WSO2 API Manager. You can manually edit the log4j.properties file, or configure logging through the management console. Configuration made through the management console can be persisted in the WSO2 registry so that it is available even after the server restarts. There is also an option to restore the original Log4j configuration from the log4j.properties file using the management console. However, if you modify the log4j.properties file and restart the server, the earlier log4j configuration persisted in the registry is overwritten.

- Configure logging
  - Global Log4J configuration
  - Configure Log4J appenders
  - Configure Log4J loggers

For information on viewing the contents of the logs, see Application Logs and System Logs.

Configure logging

Follow the instructions below to set up logging.

1. Sign in to the API-M Management Console.
2. Click Configure > Logging.
3. If you want your modifications to be persisted and available even after a server restart, select the Persist All Configurations Changes check box.
4. Use the options in the following sections to configure the layout and the amount of information about system activity that you want to record.

- Global Log4J configuration
- Configure Log4J appenders
- Configure Log4J loggers

**Global Log4J configuration**

This section allows you to assign a single log level and log pattern to all loggers.

- **Log Level** - Reflects a minimum level that this logger cares about. You can view the hierarchy of levels below.
- **Log Pattern** - Defines the output format of the log file.

**Configure Log4J appenders**

Log4j allows logging requests to print to multiple destinations. These output destinations are called appenders. You can attach several appenders to one logger.

- **Name** - The name of an appender. By default, WSO2 API Manager comes with the following log appenders configured:
  - **CARBON_CONSOLE** - Logs to the console when the server is running.
  - **CARBON_LOGFILE** - Writes the logs to the `<API-M_HOME>/repository/logs/wso2carbon.log` file.
  - **SERVICE_APPENDER** - Writes mediation time audit messages to the `<API-M_HOME>/repository/logs/wso2-apigw-service.log` file.
  - **TRACE_APPENDER** - Writes mediation time tracing/debug messages to the `<API-M_HOME>/repository/logs/wso2-apigw-trac
e.log file for tracing enabled services.
- **TRACE_MEMORYAPPENDER**
- **CARBON_MEMORY**
- **CARBON_SYS_LOG** - Allows separation of the software that generates messages from the system that stores them and the software that reports and analyzes them.
- **Log pattern** - Defines the output format of the log file.
- **Threshold** - Filters log entries based on their level. For example, if the threshold is set to WARN, log entries are allowed to pass into the appender if its level is WARN, ERROR or FATAL, while other entries are discarded.

Hierarchy of levels
- **TRACE** - Designates informational events that are more fine-grained than DEBUG.
- **DEBUG** - Designates fine-grained informational events that are most useful to debug an application.
- **INFO** - Designates informational messages that highlight the progress of the application at coarse-grained level.
- **WARN** - Designates potentially harmful situations.
- **ERROR** - Designates error events that might still allow the application to continue running.
- **FATAL** - Designates very severe error events that will presumably lead the application to abort.

**Configure Log4J loggers**

This section allows you to browse through all loggers, define a log level, and switch on/off additivity to any of them. You can filter loggers using the first few characters (use the **Starts With** button) or using a combination of characters (use the **Contains** button).

- **Logger** - The name of a logger.
- **Parent Logger** - The name of a parent logger.
- **Level** - Allows to select the level (threshold) from the drop-down menu. After you specify the level for a certain logger, a log request for that logger is enabled only if its level is equal or higher to that of the logger’s. If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. See hierarchy of levels above.
- **Additivity** - Allows to inherit all the appenders of the parent Logger if set to True.

**Example**

Use the following procedure to enable logs to view HTTP headers and messages:

1. In the **Filter Loggers by** field, enter **wire** and then click **Contains**. You see `org.apache.synapse.transport.http` displayed under **Logger**.

2. Change the level of this logger to **DEBUG**.

3. Search for the `org.apache.synapse.transport.http.headers` logger and change the level to **DEBUG**.

Alternatively, you can uncomment the entry for the two loggers as follows:

1. Go to the `<API-M_HOME>/repository/conf` directory and open the `log4j.properties` file with a text editor.
2. Edit the entries for the two loggers as follows by removing the commented (#).
   ```
   log4j.logger.org.apache.synapse.transport.http.headers=DEBUG
   log4j.logger.org.apache.synapse.transport.http.wire=DEBUG
   ```
3. Save the changes.

**System Logs**

The **System Logs** page displays information about the log files of the current product. The log files can be retrieved in two ways:
If syslog-ng is configured, log files are taken from the remote location where the log files are hosted using the syslog-ng server. If syslog-ng is not configured, log files are taken from the local file system (super-tenant or stand-alone apps).

This page contains the following sections:

- Viewing system logs
- Displaying log mediator logs

For more information on logs, see Logging.

Viewing system logs

To view system logs, click System Logs on the Monitor tab in the Management Console. The log messages displayed on this page are obtained from a memory appender. Therefore, the severity (log level) of the displayed log messages is equal to or higher than the threshold of the memory appender.

- **File Name** - The name of the file containing logs pertaining to a certain period.
- **Date** - The date at which the log file was generated.
- **File Size** - The size of the file in bytes.
- **Action** - Allows to view and download files.

### System Logs

Log files are taken from the local file system.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Date</th>
<th>File Size</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>wso2carbon.log</td>
<td>Current Log</td>
<td>1.0 MB</td>
<td>View Download</td>
</tr>
<tr>
<td>wso2carbon.log.2011-07-22</td>
<td>2011-07-22</td>
<td>44.4 KB</td>
<td>View Download</td>
</tr>
<tr>
<td>wso2carbon.log.2011-07-26</td>
<td>2011-07-26</td>
<td>295 MB</td>
<td>View Download</td>
</tr>
</tbody>
</table>

Now, you can download an individual log file, or view it in the Management Console. When viewing a log file, you can choose a category such as ERROR in the View list to filter the messages by that category. You can also enter a search term to find a specific log message, and you can use the Log Head field to specify how many log lines to display.

### System Log View

Now you can choose a category such as ERROR to filter the messages by that category. You can also enter a search term to find a specific log message, and you can use the Log Head field to specify how many log lines to display.

The location of the log files on disk is specified in the **log4j.configuration** file.

Displaying log mediator logs

When you use the Log mediator inside sequences and proxy services in WSO2 API Manager, the logs are stored in the `<PRODUCT_HOME>/repository/logs/wso2carbon.log` file by default. To view these logs from the System Logs page, open the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file and uncomment the following element:

```xml
<handler class="org.wso2.carbon.utils.logging.handler.TenantDomainSetter" name="TenantDomainSetter"/>
```

When you restart the server, the logs will be available on the System Logs page.

Monitoring Access Logs
HTTP access logs help you monitor information such as the persons who access the product, how many hits are received, what the errors are, etc. This information is useful for troubleshooting errors.

All WSO2 products can enable access logs for the HTTP servlet transport. This servlet transport works on 9443/9763 ports, and it receives admin/operation requests. Therefore, access logs for the servlet transport is useful for analysing operational/admin-level access details. Additionally, in WSO2 API Manager (WSO2 API-M), WSO2 Enterprise Service Bus (WSO2 ESB), and WSO2 Enterprise Integrator (WSO2 EI) you can generate access logs for the PassThrough and NIO transport as well. The PassThrough and NIO transport works on 8280/8243 ports and is used for API/Service invocations. By default, the access logs from both the Servlet transport and the PassThrough transport are written to a common access log file located in the <API-M_HOME>/repository/logs directory.

To configure the default behaviour of HTTP access logs in WSO2 API-M, see the following topics.

- Configuring access logs for the HTTP Servlet transport
- Configuring access logs for the PassThrough or NIO transports (Service/API invocation)
- Supported log pattern formats for the PassThrough or NIO transports

### Configuring access logs for the HTTP Servlet transport

The HTTP Servlet transport related access logs, logs details of the request as well as the response on a single log line.

As the runtime of WSO2 products is based on Apache Tomcat, you can use the Access_Log_Valve variable in Tomcat as explained below to configure access logs for the HTTP Servlet transport:

1. Open the `<API-M_HOME>/repository/conf/tomcat/catalina-server.xml` file, which is the server descriptor file for the embedded Tomcat integration.
2. Customize the attributes for the `Access_Log_Valve` variable shown below.

```xml
<Va[...]
className="org.apache.catalina.valves.AccessLogValve"
    directory="${carbon.home}/repository/logs"
    prefix="http_access_"
    suffix=".log"
    pattern="combined"/>
```

The attributes that are used by default are explained below. See the descriptions of the Tomcat-supported Access Log Valve attributes and customize the required values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>The path to the directory that will store the access log file. By default, this location is set to <code>${carbon.home}/repository/logs</code> in all WSO2 products.</td>
</tr>
<tr>
<td>prefix</td>
<td>The prefix added to the log file's name. By default, this is set to: &quot;http_access_&quot;</td>
</tr>
<tr>
<td>suffix</td>
<td>The suffix added to the log file's name. By default, this is set to: &quot;_.log&quot;</td>
</tr>
</tbody>
</table>
The attribute defines the format for the log pattern, which consists of the information fields from the requests and responses that should be logged. The pattern format is created using the following attributes:

- A standard value to represent a particular string. For example, "%h" represents the remote host name in the request. See the list of string replacement values supported by the Tomcat valve.
- %{xxx}i is used to represent the header in the incoming request (xxx=header value).
- %{xxx}o is used to represent the header in the outgoing request (xxx=header value).

While you can use the above attributes to define a custom pattern, the standard patterns shown below can be used.

- **common (Apache common log pattern):**

  ```
  pattern=%h %l %u %t "%r" %s %b
  ```

- **combined (Apache combined log pattern):**

  ```
  pattern=%h %l %u %t "%r" %s %b "%{Referer}i" "%{User-Agent}i"
  ```

Note that, by default, the "combined" pattern is enabled in WSO2 API-M.

### 3. Restart the server.

According to the default configurations, a log file named `http_access_.{DATE}.log` is created inside the `<API-M_HOME>/repository/log` directory. The log is rotated on a daily basis.

### Configuring access logs for the PassThrough or NIO transports (Service/API invocation)

The PassThrough and/or NIO transport related access logs, logs the request and the response on two separate log lines.

By default, access logs related to service/API invocation are disabled for performance reasons in the above products. You should enable these access log only for troubleshooting errors. Follow the steps given below to enable access logs for the PassThrough or NIO transport:

1. **Add the following entry in the `<API-M_HOME>/repository/conf/log4j.properties` configuration file.**

   ```
   log4j.logger.org.apache.synapse.transport.http.access=INFO
   ```

2. **Create a file named `<API-M_HOME>/repository/conf/access-log.properties` with the following configuration and customize it as required.**

   You can customize the format of your PassThrough or NIO access logs based on the configurations in your `access-log.properties` file.

   ```
   # Default access log pattern
   #access_log_pattern="%(X-Forwarded-For)i \%h \%l \%u \%t "\"\" \%s \%b "%(Referer)i"
   \"%(User-Agent)i"
   # combined log pattern
   #access_log_pattern="%(X-Forwarded-For)i \%h \%l \%u \%t "\"\" \%s \%b "%(Referer)i"
   \"%(User-Agent)i"
   access_log_pattern=time=%t remoteHostname=%h localPort=%p localIP=%A requestMethod=%m
   requestURL=%U remoteHost=%% requestProtocol=%H HTTPStatusCode=%s queryString=%q
   # common log pattern
   #access_log_pattern="%(X-Forwarded-For)i \%h \%l \%u \%t "\"\" \%s \%b
   # file prefix
   access_log_prefix=http_gw
   # file suffix
   access_log_suffix=.log
   # file date format
   access_log_file_date_format=yyyy-MM-dd
   #access_log_directory="/logs"
   ```
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>access_log_directory</code></td>
<td>Add this property <strong>ONLY</strong> if you want to change the default location of the log file. By default, the product is configured to store access logs in the <code>&lt;API-M_HOME&gt;/repository/logs</code> directory.</td>
</tr>
<tr>
<td><code>access_log_prefix</code></td>
<td>The prefix added to the name of the log file.</td>
</tr>
<tr>
<td><code>access_log_suffix</code></td>
<td>The suffix added to the name of the log file.</td>
</tr>
<tr>
<td><code>access_log_file_date_format</code></td>
<td>The date format used in access logs.</td>
</tr>
</tbody>
</table>
| `access_log_pattern`         | The attribute defines the format for the log pattern, which consists of the information fields from the requests and responses that should be logged. The pattern format is created using the following attributes:  
  - A standard value to represent a particular string. For example, `\%h` represents the remote host name in the request. Note that all the string replacement values supported by Tomcat are **NOT** supported for the pass-through transport's access logs. The list of supported values are given below.  
  - `\%{xxx}i` is used to represent the header in the incoming request (`xxx=header value`).  
  - `\%{xxx}o` is used to represent the header in the outgoing request (`xxx=header value`).  
  
  While you can use the above attributes to define a custom pattern, the standard patterns shown below can be used.  
  - **common** (Apache common log pattern):  
    
    ```
    access_log_pattern=%h %l %u %t "%r" %s %b
    ```  
  - **combined** (Apache combined log pattern):  
    
    ```
    access_log_pattern=%h %l %u %t "%r" %s %b "%{Referer}i" "%{User-Agent}i"
    ```  
  
  By default, a modified version of the Apache combined log format is enabled in WSO2 API-M as shown below. Note that the "X-Forwarded-For" header is appended to the beginning of the usually combined log format. This correctly identifies the original node that sent the request (in situations where requests go through a proxy such as a load balancer). The "X-Forwarded-For" header must be present in the incoming request for this to be logged.  
    
    ```
    access_log_pattern=%{X-Forwarded-For}i %h %l %u %t "%r" %s %b 
    "%{Referer}i" "%{User-Agent}i"
    ``` |

3. Add the following configuration in the `<API-M_HOME>/repository/conf/nhttp.properties` file.
   You need to add this configuration in order to make sure that the access logs related to the PassThrough and NIO transports are rotated on a daily basis. If this configuration is not set, all the access log details related to the PassThrough and NIO transports will get logged in a single file. The date will be appended to the access log when it is rotated.

   ```
   nhttp.is.log.rotatable=true
   ```

4. Restart the server.
5. Invoke an API in WSO2 APIM.
   For testing purposes, use the artifacts in the quick start guide.
The access log file for the service/API will be created in the `<API-M_HOME>/repository/logs` directory in the following format. `<access_log_prefix>.log

For example, `http_gw.log`

Note that there will be delay in printing the logs to the access log file.

**Supported log pattern formats for the PassThrough or NIO transports**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>Remote IP address</td>
</tr>
<tr>
<td>%A</td>
<td>Local IP address</td>
</tr>
<tr>
<td>%b</td>
<td>Bytes sent, excluding HTTP headers, or '-' if zero</td>
</tr>
<tr>
<td>%B</td>
<td>Bytes sent, excluding HTTP headers</td>
</tr>
<tr>
<td>%c</td>
<td>Cookie value</td>
</tr>
<tr>
<td>%C</td>
<td>Accept header</td>
</tr>
<tr>
<td>%e</td>
<td>Accept Encoding</td>
</tr>
<tr>
<td>%E</td>
<td>Transfer Encoding</td>
</tr>
<tr>
<td>%h</td>
<td>Remote host name (or IP address if enableLookups for the connector is false)</td>
</tr>
<tr>
<td>%l</td>
<td>Remote logical username from identd (always returns '-')</td>
</tr>
<tr>
<td>%L</td>
<td>Accept Language</td>
</tr>
<tr>
<td>%k</td>
<td>Keep Alive</td>
</tr>
<tr>
<td>%m</td>
<td>Request method (GET, POST, etc.)</td>
</tr>
<tr>
<td>%n</td>
<td>Content Encoding</td>
</tr>
<tr>
<td>%r</td>
<td>Request Element</td>
</tr>
<tr>
<td>%s</td>
<td>HTTP status code of the response</td>
</tr>
<tr>
<td>%S</td>
<td>Accept Chatset</td>
</tr>
<tr>
<td>%t</td>
<td>Date and time, in Common Log Format</td>
</tr>
<tr>
<td>%T</td>
<td>Time taken to process the request in seconds.</td>
</tr>
<tr>
<td>%u</td>
<td>Remote user that was authenticated (if any), else '-'</td>
</tr>
<tr>
<td>%U</td>
<td>Requested URL path</td>
</tr>
<tr>
<td>%v</td>
<td>Local server name</td>
</tr>
</tbody>
</table>
Maintaining Separate Production and Sandbox Gateways

With WSO2 API Manager, you can maintain a production and a sandbox endpoint for a given API. The production endpoint is the actual location of the API, whereas the sandbox endpoint points to its testing/pre-production environment.

When you publish an API using the API Publisher, it gets deployed on the API Gateway. By default, there's a single Gateway instance (deployed either externally or embedded within the publisher), but you can also set up multiple Gateways:

- Single Gateway to handle both production and sandbox requests
- Multiple Gateways to handle production and sandbox requests separately

Single Gateway to handle both production and sandbox requests

This is the default scenario. Because this Gateway instance handles both production and sandbox token traffic, it is called a hybrid API Gateway. When an API request comes to the API Gateway, it checks whether the requesting token is of type PRODUCTION or SANDBOX and forwards the request to the appropriate endpoint. The diagram below depicts this scenario.

Multiple Gateways to handle production and sandbox requests separately

Having a single Gateway instance to pass through both types of requests can negatively impact the performance of the production server. To avoid this, you can set up separate API Gateways. The production API Gateway handles requests that are made using PRODUCTION type tokens and the sandbox API Gateway handles requests that are made using SANDBOX type tokens.

The diagram below depicts this using two Gateways:
In either of the two approaches, if an API Gateway receives an invalid token, it returns an error to the requesting client saying that the token is invalid.

You configure production and sandbox Gateways using the `<Environments>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file in API Publisher nodes, as shown in the following example:

```xml
<Environments>
  <Environment type="production">
    <Name>Production</Name>
    <ServerURL>https://localhost:9445/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
  </Environment>
  <Environment type="sandbox">
    <Name>Sandbox</Name>
    <ServerURL>https://localhost:9448/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
  </Environment>
</Environments>
```

The `<ServerURL>` parameter should have the value of the environment instance. For information about the `<GatewayEndpoint>`, see Working with Endpoints.

The `type` attribute of the `<Environment>` element can take the following values:

- **Production**: A production type Gateway
- **Sandbox**: A sandbox type Gateway
- **Hybrid**: The Gateway handles both types of tokens

If you work with Gateways in different geographical locations, configuring multiple environments using the `<APIGateway>` element in the `<API-M_HOME>/` file is recommended. The diagram below depicts a sample setup:
Managing Users and Roles

Before you begin, note the following:

- Only system administrators can add, modify and remove users and roles. To set up administrators, see Realm Configuration.
- Your product has a primary user store where the users/roles that you create using the management console are stored by default. It’s default RegEx configurations are as follows. RegEx configurations ensure that parameters like the length of a user name/password meet the requirements of the user store.

```plaintext
PasswordJavaRegEx-------- ^\S{5,30}$
PasswordJavaScriptRegEx-- ^\S{5,30}$
UsernameJavaRegEx-------- ^!#$%;*+={}\{3,30}$
UsernameJavaScriptRegEx-- ^\S{3,30}$
RolenameJavaRegEx-------- ^!#$%;*+={}\{3,30}$
RolenameJavaScriptRegEx-- ^\S{3,30}$
```
When creating users/roles, if you enter a username, password etc. that does not conform to the RegEx configurations, the system throws an exception. You can either change the RegEx configuration or enter values that conform to the RegEx. If you change the default user store or set up a secondary user store, configure the RegEx accordingly under the user store manager configurations in `<APIM_HOME>/repository/conf/user-mgt.xml` file. For details about writing a simple custom user store manager for WSO2 products, see Writing a Custom User Store Manager.

This chapter contains the following information:

- Adding User Roles
- Adding Users

### Adding User Roles

Roles contain permissions for users to manage the server. They can be reused and they eliminate the overhead of granting permissions to users individually.

Throughout this documentation, we use the following roles that are typically used in many enterprises. You can also define different user roles depending on your requirements.

- **admin**: The API management provider who hosts and manages the API Gateway and is responsible for creating users in the system, assigning them roles, managing databases, security, etc. The Admin role is also used to access the WSO2 Admin Portal (https://<APIM_Host>:<APIM_Port>/admin), where you can define workflow tasks, throttling policies, analytics configurations, etc. The Admin role is available by default with the credentials admin/admin. By default, this role contains all the permissions (including super admin permissions) in the permission tree.

- **creator**: A creator is typically a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions etc.) and uses the API publisher to provision APIs into the API store. The creator uses the API Store to consult ratings and feedback provided by API users. Creator can add APIs to the store but cannot manage their lifecycle. Governance permission gives to a creator to govern, manage and configure the API artifacts.

- **publisher**: A person in a managerial role and overlooks a set of APIs across the enterprise and controls the API lifecycle, subscriptions and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.

- **subscriber**: A user or an application developer who searches the API store to discover APIs and use them. S/he reads the documentation and forums, rates/comments on the APIs, subscribes to APIs, obtains access tokens and invokes the APIs.

Follow the instructions below to create the creator, publisher and subscriber roles in the API Manager for example.

Create user roles

1. Log in to the management console (https://<APIM_Host>:<APIM_Port>/admin) as admin (default credentials are admin/admin).
2. In the **Main** menu, click **Add** under **Users and Roles**.
3. Click **Add New Role**.
4. Enter the name of the user role (e.g., creator) and click Next.

**Step 1 : Enter role details**

<table>
<thead>
<tr>
<th>Enter role details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
</tr>
<tr>
<td><strong>Role Name</strong></td>
</tr>
</tbody>
</table>

**Tip:** The Domain drop-down list contains all user stores configured in the system. By default, you only have the PRIMARY user store. To configure secondary user stores, see Configuring Secondary User Stores.

5. The permissions page opens. Select the permissions according to the role that you create. The table below lists the permissions of the creator, publisher and subscriber roles which are available by default:

<table>
<thead>
<tr>
<th>Roles</th>
<th>Permissions</th>
<th>UI</th>
</tr>
</thead>
</table>

Configuring Secondary User Stores
creator
- Configure > Governance and all underlying permissions.
- Login
- Manage > API > Create
- Manage > Resources > Govern and all underlying permissions

Step 2: Select permissions to add to Role

**Admin Permissions**
- Configure
  - Configure Data Sources
  - Governance
  - Inbound Endpoints
  - Configure ntask integration
  - Configure ESLI ntask
  - Priority Executor
- Security
  - Schedule Tasks
  - Theme
- Login
- Manage
  - Add
  - API
  - Create
  - Subscribe
  - Publish
- Manage
- Extensions
- Identity
  - Manage Tiers
  - Mediation
- Configure
- Resources
  - Associations
  - Browse
  - Community Features
  - Govern
- Notifications
- WS-API

publisher
- Login
- Manage > API > Publish
6. Click **Finish** once you are done adding permissions.

When a user creates an application and generates application keys, a role is created automatically in the following format.

```
"Application/<username>_<applicationName>_PRODUCTION"
```

These roles do not have any permissions assigned to it, but it is used to manage the visibility of the corresponding service provider that is created in the format of "<username>_<applicationName>_PRODUCTION" within the Key Manager. The created service provider is only visible to users with the latter mentioned role that has been generated automatically. Only if a user with admin privileges assigns the latter mentioned role to a user, will that user be able to view the details of the service provider that is created per application.

### Adding Users

Users are consumers who interact with your enterprise's applications, databases or any other systems. These users can be persons, devices or applications/programs within or outside of the enterprise's network. Since these users interact with internal systems and access data, the need to define which user is allowed to do what, is critical. This is called user management.

Follow the steps below to create users and assign them to roles via the Management console. Also, if you want to authenticate users via e-mail, social media, multiple user store attributes, see [Maintaining Logins and Passwords](#).

1. Log in to the Management Console (https://<hostname>:9443/carbon) and click **Add** under **Users and Roles** in the **Main** menu.
2. Click **Add New User**.

Add Users and Roles

3. The **Add User** page opens. Provide the username and password and click **Next**.

Add New User

**Step 1: Enter user name**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>User Name*</td>
<td>apicreator</td>
</tr>
<tr>
<td>Password*</td>
<td>********</td>
</tr>
<tr>
<td>Password Repeat*</td>
<td>********</td>
</tr>
</tbody>
</table>

**Tip:** The **Domain** drop-down list contains all user stores configured in the system. By default, you only have the PRIMARY user store. To configure secondary user stores, see **Configuring Secondary User Stores**.

4. Select the roles you want to assign to the user. In this example, we assign the **creator** role defined in the previous section.
4. Click **Finish** to complete. The new user appears in the **Users** list. You can change the user's password, assign it different roles or delete it.

You cannot change the user name of an existing user.

5. Click **Finish** to complete. The new user appears in the **Users** list. You can change the user's password, assign it different roles or delete it.

Accessing the Admin Dashboard

The Admin Dashboard is intended to be used by API Manager admins. The admin user has special permissions specified in the `/permission/admin/ma`
If a new user needs to access the admin dashboard, follow the steps below:

1. Create a user.
2. Create a new role. For more information, see Adding User Roles.
3. Assign the following permissions to the new role you just created: /permission/admin/manage/apim_admin and /permission/admin/configure/login.
4. Assign the role created in step 2, to the user created in step 1.

Now this user is able to login and perform administrative tasks using the Admin Dashboard.

**Message Tracing**

Message Tracing refers to the process of identifying each message flow of each of the transactions that go through the Gateway. You can do message tracing on WSO2 API Manager by installing the Message Tracer feature that has been specifically developed for WSO2 products. You can use the Message Tracer to derive logging, auditing, and debugging related information with regard to message content and its direction. The Message Tracer is a part of carbon-analytics and it is built into WSO2 API Manager.

- Configuring message tracing
- Publishing event tracing data to API Manager Analytics

**Configuring message tracing**

Follow the steps below to configure the Message Tracer in API Manager to dump trace events to WSO2 APIM logs, which can be viewed via the terminal or the `wso2carbon` log file.

1. Start the WSO2 API Manager server.
3. Click Message Tracing, which is under the Configure tab, to navigate to the Message Tracing Configurations.
4. Select the following options to enable message tracing and click Update.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump Message Content</td>
<td>Enable Message tracing for the Content of the Message as well.</td>
</tr>
<tr>
<td>Enable Logging</td>
<td>Enable Logging in the available logging handler in order to log the tracing message.</td>
</tr>
<tr>
<td>Enable Analytics Event Publishing</td>
<td>Publish tracing events to WSO2 API Manager Analytics</td>
</tr>
</tbody>
</table>

5. Add an event publisher to log the trace messages in the APIM the `wso2carbon` log file.
   a. Go to Main > Event > Publishers and click Add Event Publisher.
b. In Create a New Event Publisher page, add the following details and click Add Event Publisher.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name</td>
<td>message_tracer_logger_publisher</td>
</tr>
<tr>
<td>Event Source</td>
<td>DAS_MESSAGE_TRACE:1.0.0</td>
</tr>
<tr>
<td>Stream Attributes</td>
<td>Keep default values</td>
</tr>
<tr>
<td>Output Event Adapter Type</td>
<td>logger</td>
</tr>
<tr>
<td>Message Format</td>
<td>text</td>
</tr>
</tbody>
</table>

Leave the Unique Identifier field blank.

After enabling message tracing, dump message content, and logging, you will see a log message similar to the following on the API Console/terminal for events such as API invocation etc.
Publishing event tracing data to API Manager Analytics

As an additional step you can publish these trace messages to WSO2 API Manager Analytics server by following the steps below.

Before you begin,

Make sure you have configured API Manager Analytics. For more information, see Configuring APIM Analytics.

1. Sign in to WSO2 APIM Management Console (https://localhost:9443/carbon) if you have not done so already.
2. Select Enable Analytics Event Publishing, which is in the Message Tracing Configuration page and click Update.
After you have saved the changes, WSO2 APIM will generate a stream definition in the `<APIM_HOME>/repository/deployment/server/events/streams/DAS_MESSAGE_TRACE_1.0.0.json` file.

3. Copy the `DAS_MESSAGE_TRACE_1.0.0.json` file and add it to the `<APIM_ANALITICS_HOME>/repository/deployment/server/events/streams` directory to deploy the same definition in WSO2 API Manager Analytics.

4. Start the WSO2 API Manager Analytics server.


6. Persist the event stream so that the data is saved into a table.
   a. Navigate to Main > Event > Streams to view the available event streams.

   You will see the deployed stream under event streams.

   ![Stream Deployment](image)

   You will see the deployed stream under event streams.

   ![Stream Deployment](image)

   b. Click the Edit option that is relevant to the stream definition `DAS_MESSAGE_TRACE:1.0.0` file so that it opens in the edit view.
   c. Click Next[Persist Event] at the bottom of the edit view.
   d. In the next page select Persist Event Stream and select all the attribute check-boxes in order to persist all the information and click Save Event Stream.
7. Add an Event receiver to point to the Event stream.
   a. Go to **Main -> Event -> Receivers** in WSO2 API Manager Analytics and click **Add Event Receiver**.
   b. Add the following details and click **Add Event Receiver**.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Receiver Name</td>
<td>message_trace_receiver</td>
</tr>
<tr>
<td>Input Event Adapter Type</td>
<td>wso2event</td>
</tr>
<tr>
<td>Is events duplicated in cluster</td>
<td>false</td>
</tr>
<tr>
<td>Event Stream</td>
<td>DAS_MESSAGE_TRACE:1.0.0</td>
</tr>
<tr>
<td>Message format</td>
<td>wso2event</td>
</tr>
</tbody>
</table>
8. Configure a publisher that can publish events to WSO2 API Manager Analytics.
   a. Sign in to WSO2 APIM Management console (https://localhost:9443/carbon) if you have not done so already.
   b. Go to Main > Event > Publishers and click Add Event Publisher.
   c. In Create a New Event Publisher page, add the following details and click Add Event Publisher.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Publisher Name</td>
<td>message_tracer_analytics_publisher</td>
</tr>
<tr>
<td>Event Source</td>
<td>DAS_MESSAGE_TRACE:1.0.0</td>
</tr>
<tr>
<td>Stream Attributes</td>
<td>Keep default values</td>
</tr>
<tr>
<td>Output Event Adapter Type</td>
<td>wso2event</td>
</tr>
<tr>
<td>Receiver URL</td>
<td>tcp://localhost:7612</td>
</tr>
<tr>
<td>User Name</td>
<td>admin</td>
</tr>
<tr>
<td>Password</td>
<td>admin</td>
</tr>
<tr>
<td>Protocol</td>
<td>thrift</td>
</tr>
<tr>
<td>Publishing mode</td>
<td>non-blocking</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Publishing Timeout</td>
<td>0</td>
</tr>
<tr>
<td>Message Format</td>
<td>wso2event</td>
</tr>
</tbody>
</table>

Leave the Authenticator URL field blank.

**Create a New Event Publisher**

If you have also enabled Analytics event Publishing in addition to enabling message tracing, dump message content, and logging, you can see the results in WSO2 API Manager Analytics by following the steps below:

1. Sign in to the WSO2 API Manager Analytics Management Console ([https://localhost:9444/carbon](https://localhost:9444/carbon)).
2. Navigate to Main > Data Explorer.
3. Select the table **DAS MESSAGE TRACE** and click Search.
   You will see the event data traced in the WSO2 Analytics Data Explorer as shown below.
Migrating the APIs and Applications to a Different Environment

WSO2 API Manager (WSO2 API-M) allows you to maintain multiple environments running on the same WSO2 API-M version. This feature allows you to import and export applications or APIs between your environments. For example, if you have an application or API running in the development environment, you can import it and export to the test (QA) environment. Thereby, APIs and applications do not have to be created from scratch in different environments.

- Getting Started
- Migrating APIs to different environments
- Migrating applications to different environments
- Reset user
- Check the version of the CLI
- Set HTTP request timeout
- Set export directory
- Understanding the API import/export tool

**Getting Started**

After running the CLI tool make sure to add an environment before you start working with the import/export CLI commands, because all APIs and applications need to be imported or exported to a specific environment.

- Running the CLI tool
- Adding an environment
- Removing an environment
- List environments

**Running the CLI tool**

- **Step 1 - Deploy the API import/export tool**
- **Step 2 - Run the CLI tool**

**Step 1 - Deploy the API import/export tool**

2. Start WSO2 API Manager.
3. Download the latest WSO2 API import/export tool (api-import-export-2.6.0-v1.war) from [here](http://wso2.com/products/api-manager/).

   Note that the import/export tool attached is specific to this version of WSO2 API Manager.

4. Copy the downloaded `api-import-export-2.6.0-v1.war` file to the `<API-M_HOME>/repository/deployment/server/webapps` folder. The file is automatically deployed as hot deployment is enabled.

**Step 2 - Run the CLI tool**

1. Navigate to the API Management Tooling page - [https://wso2.com/api-management/tooling/](https://wso2.com/api-management/tooling/)
2. Click **Download** under CLI.
3. Select a generated archive suitable for your platform (i.e., Mac, Windows, Linux) and extract it the CLI tool that you downloaded to a desired location and `cd` into it.
4. Navigate to the working directory where the executable CLI Tool resides.
5. Execute the following command to start the CLI tool.

```bash
./apimcli
```

6. Add the location of the extracted folder to your system's `$PATH` variable to be able to access the executable from anywhere.

For further instructions execute the following command.

```bash
apimcli --help
```

**Global flags for CLI tool**

The following are some global flags that you can use with the CLI tool.

```bash
--verbose
--insecure, -k
--help, -h
```

Display information and example usage of a command

**Adding an environment**

You can add environments by either manually editing the `$HOME/.wso2apimcli/main_config.yaml` file or by running the following CLI command.

```bash
apimcli add-env
```

The directory structure for the configuration files (`$HOME/.wso2apimcli`) will be created upon execution of the `apimcli` command.

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to add an environment.

```bash
Command Format: Example Response
```

```
**Linux Format**

```bash
apimcli add-env -n <environment-name> \
--registration <registration-endpoint> \ 
--apim <API-Manager-endpoint> \ 
--token <token-endpoint> \ 
--import-export <endpoint-for-environment> \ 
--admin <admin-REST-API-endpoint> \ 
--api_list <API-listing-REST-API-endpoint> \ 
--app_list <application-listing-REST-API-endpoint>
```

```
**Mac Format**

```bash
apimcli add-env -n <environment-name> --registration <registration-endpoint> --apim \ 
<API-Manager-endpoint> --token <token-endpoint> --import-export <endpoint-for-environment> \ 
--admin <admin-REST-API-endpoint> --api_list <API-listing-REST-API-endpoint> --app_list \ 
<application-listing-REST-API-endpoint>
```

**Flags:**
• Required flags:
  • --name, -n
  There are no short flags for the following flags.
  • --registration
  • --apim
  • --token
  • --import-export
  • --admin
  • --api_list
  • --app_list

Example 1

apimcli add-env -n production
--registration
https://localhost:9443/client-registration/v0.14/register
--apim https://localhost:9443
--token https://localhost:8243/token
--import-export https://localhost:9443/api-import-export-2.6.0-v1
--admin https://localhost:9443/api/am/admin/v0.14
--api_list https://localhost:9443/api/am/publisher/v0.14/apis
--app_list https://localhost:9443/api/am/store/v0.14/applications

Example 2

apimcli add-env -n dev
--registration
https://localhost:9444/client-registration/v0.14/register
--apim https://localhost:9444
--token https://localhost:8244/token
--import-export https://localhost:9444/api-import-export-2.6.0-v1
--admin https://localhost:9444/api/am/admin/v0.14
--api_list https://localhost:9444/api/am/publisher/v0.14/apis
--app_list https://localhost:9444/api/am/store/v0.14/applications

Example 1

apimcli add-env -n production --registration
https://localhost:9443/client-registration/v0.14/register
--apim https://localhost:9443
--token https://localhost:8243/token
--import-export https://localhost:9443/api-import-export-2.6.0-v1
--admin https://localhost:9443/api/am/admin/v0.14
--api_list https://localhost:9443/api/am/publisher/v0.14/apis
--app_list https://localhost:9443/api/am/store/v0.14/applications

Example 2

./apimcli add-env -n dev --registration
--token https://localhost:8244/token --import-export
https://localhost:9444/api-import-export-2.6.0-v1 --admin
https://localhost:9444/api/am/admin/v0.14 --api_list
https://localhost:9444/api/am/publisher/v0.14/apis --app_list
https://localhost:9444/api/am/store/v0.14/applications
Successfully added environment '<environment-name>'

Successfully added environment 'production'

Removing an environment

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to remove an environment.

   **Command Format**
   ```
   apimcli remove-env -n <environment-name>
   ```
   ```
   apimcli remove-env --name <environment-name>
   ```

   **Flags:**
   - Required flags
     - `--name`, `-n`

   ```
   apimcli remove-env -n production
   ```

   **Example**
   ```
   Successfully removed environment '<environment-name>'
   Execute 'apimcli add-env --help' to see how to add a new environment
   ```
   ```
   Successfully removed environment 'uat'
   Execute 'apimcli add-env --help' to see how to add a new environment
   ```

List environments

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to list the environments.
   
   There are no flags for the following CLI command.

   **Command**
   ```
   apimcli list envs
   ```
Environments available in file `~/Users/kim/.wso2apimcli/main_config.yaml`

<table>
<thead>
<tr>
<th>NAME</th>
<th>PUBLISHER ENDPOINT</th>
<th>REGISTRATION ENDPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TOKEN ENDPOINT</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://localhost:9444/client-registration/v0.14/register">https://localhost:9444/client-registration/v0.14/register</a></td>
<td></td>
</tr>
</tbody>
</table>

Migrating APIs to different environments

- Exporting an API
- Importing an API
- Importing/exporting an API in a tenanted environment
- List APIs

Exporting an API

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to export an existing API as a .zip archive.

Command Format

```bash
apimcli export-api -n <API-name> -v <version> -r <provider> -e <environment> -u <username> -p <password> -k
```

```bash
apimcli export-api --name <API-name> --version <version> --provider <provider> --environment <environment> --username <username> --password <password> --insecure
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

- Required flags:
  - `--name, -n`
  - `--version, -v`
  - `--provider, -r`
  - `--environment, -e`
  - `--insecure, -k`: This allows connections to SSL sites without certificates

- Optional flags:
  - `--username, -u`
  - `--password, -p`

```bash
apimcli export-api -n PhoneVerification -v 1.0.0 -r admin -e dev -u admin -p admin -k
```

Successfully exported API!
Find the exported API at `~/Users/kim/.wso2apimcli/exported/apis/dev/PhoneVerification_1.0.0.zip`

Importing an API
When you import an API, regardless the status of the imported API it will be added with the created state and you need to sign in to the Publisher and publish the API.

You can use the archive created in the previous section to import APIs to an API Manager instance.

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.

### For Secure Endpoint Enabled APIs:

If you have enabled secure endpoints when creating the API and your username or/and password differs in the two environments, please follow the steps below before importing the API.

1. Unzip the .zip archive created in the previous section.
2. Go to the `<API-name-version>/Meta-information` directory and open the `api.json` file. For example, `PhoneVerification_1.0.0/Meta-information` directory and open the `api.json` file.
3. Modify the `endpointUTPassword` with your endpoint password and save the `api.json` file.
4. Compress the `PhoneVerification_1.0.0` folder to a folder named `myExportedAPI`.

#### Command Format

- **Required flags:**
  - `--file, -f`: The file path of the exported API. For example, if your file path is `/Users/kim/./wso2apimcli/exported/apis/dev/PhoneVerification_1.0.0.zip`, then you need to enter `dev/PhoneVerification_1.0.0.zip` as the value for this flag.
  - `--environment, -e`: The environment to which you what to import the API to.
  - `--insecure, -k`: This allows connections to SSL sites without certificates

- **Optional flags:**
  - `--username, -u`
  - `--password, -p`

#### Example

```
apimcli import-api -f <environment>/<file> -e <environment> -u <username> -p <password> -k
```

#### Sample Response

```
ZipFilePath: /Users/kim/.wso2apimcli/exported/apis/dev/PhoneVerification_1.0.0.zip
Successfully imported API 'dev/PhoneVerification_1.0.0.zip'
Successfully imported API!
```

You must add a parameter named `--preserve-provider` to the CLI command and set its value to false if the API is imported to a different domain than its exported one. This parameter sets the provider of the imported API to the user who is issuing the CLI command. Here's an example:

```
apimcli import-api -f dev/PhoneVerification_1.0.0.zip -e production -u admin -p admin -k
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

### Flags:

- **Required flags:**
  - `--file, -f`: The file path of the exported API. For example, if your file path is `/Users/kim/./wso2apimcli/exported/apis/dev/PhoneVerification_1.0.0.zip`, then you need to enter `dev/PhoneVerification_1.0.0.zip` as the value for this flag.
  - `--environment, -e`: The environment to which you what to import the API to.
  - `--insecure, -k`: This allows connections to SSL sites without certificates

- **Optional flags:**
  - `--username, -u`
  - `--password, -p`

```
apimcli import-api -f dev/PhoneVerification_1.0.0.zip -e production -u admin -p admin -k
```

---

Command Format: `apimcli import-api -f <environment>/<file> -e <environment> -u <username> -p <password> -k`
**Importing/exporting an API in a tenanted environment**

To export an API from a tenant, follow the steps in Export an API. Use the tenant-specific encoded credentials in the CLI command. Here's an example:

```
apimcli import-api -k -f <environment>/<file> -e <environment> -u <username> -p <password> --preserve-provider <preserve_provider>
```

Flags:

- **Required flags:**
  - `--insecure`, `-k`: This allows connections to SSL sites without certificates
  - `--file`, `-f`
  - `--environment`, `-e`: The environment to which you want to import the API to.
  - `--preserve-provider` (This does not have a short flag)

- **Optional flags:**
  - `--username`, `-u`
  - `--password`, `-p`

```
apimcli import-api --insecure --file <environment>/<file> --environment <environment> -u <username> -p <password> --preserve-provider=<preserve_provider>
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

The `--preserve-provider` flag is used to decide whether to keep the actual Provider as the provider of the API or change the provider to the user who is importing the API to a different environment.

As an example, if `--preserve-provider` is set to true, when importing an API created by user-1 in environment-1 will be preserved as the provider when and after importing that API to environment-2 by user-2. If `--preserve-provider` is set to false, when importing that API created by user-1 to the environment-2, the provider will be changed (not preserved) to user-2 who is importing the API.

**Troubleshooting**

After importing, if the APIs are not visible in the API Publisher UI, do the following to re-index the artifacts in the registry.

1. Rename the `<lastAccessTimeLocation>` element in the `<API-M_2.6.0_HOME>/repository/conf/registry.xml` file. If you use a clustered/distributed API Manager setup, change the file in the API Publisher node. For example, change the `/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime` registry path to `/system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime_1`.
2. Shut down the API Manager 2.6.0, backup and delete the `<API-M_2.6.0_HOME>/solr` directory.

For more information, see Upgrading the API Manager to 2.6.0.

Importing/exporting an API in a tenanted environment

The environments that you create will be common to the admin and the tenants. Therefore, you do not need to create environments again when exporting and importing APIs between tenanted environments.

- To export an API from a tenant, follow the steps in Export an API. Use the tenant-specific encoded credentials in the CLI command. Here's an example:

```
Command FormatExampleResponse
```

```apimcli import-api -k -f dev/PhoneVerification_1.0.0.zip -e production -u admin -p admin --preserve-provider=false
```

Developed by WSO2 Inc.
apimcli export-api -n <API-name> -v <version> -r <provider> -e <environment> -u <username> -p <password> -k

apimcli export-api --name <API-name> -version <version> --provider <provider> --environment <environment> --username <username> --password <password> --insecure

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

- **Required flags:**
  - --name, -n
  - --version, -v
  - --provider, -r
  - --environment, -e : The environment to which you want to export the API to.
  - --insecure, -k : This allows connections to SSL sites without certificates

- **Optional flags:**
  - --username, -u
  - --password, -p

apimcli export-api -n PizzaShackAPI -v 1.0.0 -r chris@test.com -e dev -u chris@test.com -p chris123 -k

Sample Response

Successfully exported API!
Find the exported API at /Users/kim/.wso2apimcli/exported/apis/dev/PizzaShackAPI_1.0.0.zip

To import the API in another tenant, follow the steps in Importing an API. Use the encoded credentials for this tenant in the CLI command. Here’s an example:

apimcli import-api -f <environment>/<file> -e <environment> -u <username> -p <password> --preserve-provider <preserve_provider>

apimcli import-api --file <environment>/<file> --environment <environment> --username <username> --password <password> --preserve-provider <preserve_provider>

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

- **Required flags:**
  - --file, -f
  - --environment, -e : The environment to which you want to import the API to.
--insecure, -k : This allows connections to SSL sites without certificates

Optional flags:

- --username, -u
- --password, -p

Examples:

```
apimcli import-api -f dev/PizzaShackAPI_1.0.0.zip -e production -u nick@example.com -p nick123 --preserve-provider=false -k
```

ZipFilePath: /Users/kim/.wso2apimcli/exported/apis/dev/PizzaShackAPI_1.0.0.zip
Successfully imported API 'dev/PizzaShackAPI_1.0.0.zip'
Successfully imported API!

Note that the --preserve-provider flag value should be set to false.

List APIs

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to list the APIs.

Command Format: `apimcli list apis -e <environment> -u <username> -p <password> -k`

```
apimcli list apis -e dev -u admin -p admin -k
```

Flags:

- Required flags:
  - --environment, -e
  - --insecure, -k
- Optional flags:
  - --username, -u
  - --password, -p

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

```
apimcli list apis -e dev -u admin -p admin -k
```

Environment: dev
No. of APIs: 2

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>VERSION</th>
<th>CONTEXT</th>
<th>STATUS</th>
<th>PROVIDER</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneVerification</td>
<td>1.0.0</td>
<td>/phoneverify</td>
<td>PUBLISHED</td>
<td>admin</td>
<td>2f25b332-4007-4c83-8249-b14b8af04848</td>
</tr>
<tr>
<td>PizzaShackAPI</td>
<td>1.0.0</td>
<td>/pizzashack</td>
<td>PUBLISHED</td>
<td>admin</td>
<td>59e81b69-24d2-4fc4-9aa-40665e119261</td>
</tr>
</tbody>
</table>
```

Copyright © WSO2 Inc. 2015-2018

547
Migrating applications to different environments

- Managing application lifecycle
- Exporting an application
- Importing an application
- Importing applications in a single tenant environment
- Importing applications in a multi-tenant environment
- List apps

Managing application lifecycle

The lifecycle of an application could be defined as the stages of an application between the development and production environments. The feature facilitates to manage the application life cycle by allowing the user to migrate the applications within desired environments. The user should have admin permissions in order to use this.

Exporting an application

You can export applications in the API Store and download them as a zipped file.

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to export an existing application as a .zip archive.

Command Format:ExampleResponse

```
apimcli export-app -n <application-name> -o <owner> -e <environment> -u <username> -p <password> -k
```

```
apimcli export-app --name <application-name> --owner <owner> --environment <environment> --username <username> --password <password> --insecure
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

- Required flags:
  - `--name, -n`
  - `--owner, -o`
  - `--environment, -e`: The environment to which you want to export the application to.
  - `--insecure, -k`: This allows connections to SSL sites without certificates

- Optional flags:
  - `--username, -u`
  - `--password, -p`

```
apimcli export-app -n SampleApp -o admin -e dev -u admin -p admin -k
```
Successfully exported Application!
Find the exported Application at /Users/kim/.wso2apimcli/exported/apps/dev/admin_SampleApp.zip

The zipped file will be as follows:
<exported-Application>.zip
<Application-Name>
<Application-Name>.json

Importing an application

You can import an application to your environment as a zipped application. When you import an application as a zipped file, a new application is created within the target environment.

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to import an existing application as a .zip archive.

Command Format Example Response

```
apimcli import-app -f <environment>/<file> -e <environment> -s <skip_subscriptions> -o <owner> -r <preserve_owner> -k
```

```
apimcli import-app --file <environment>/<file> --environment <environment> --skipSubscriptions --owner <owner> --preserveOwner --insecure
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

- **Required flags**
  - **--file, -f**: The file path of the exported App. For example, if your file path is /Users/kim/.wso2apimcli/exported/apps/dev/admin_SampleApp.zip, then you need to enter dev/admin_SampleApp.zip as the value for this flag.
  - **--environment, -e**: The environment to which you want to import the application to.
  - **--insecure, -k**: This allows connections to SSL sites without certificates

- **Optional flags**
  - **--skipSubscriptions, -s**: You can opt to skip importing the subscriptions of the application by defining this flag. This parameter is set to false by default.
  - **--owner, -o**: The owner of the imported application can be specified by providing an username of a valid user based on your preference. The application importer can set the preferred owner’s username as the value of the --owner or -o flag.
  - **--preserveOwner, -r**: You can also import the application by preserving the application owner information, from the previous environment. The application importer can add the --preserveOwner or -r flag in order to define that this flag is set to true. This parameter is set to false by default. Therefore, the default value is used when you do not define this flag. If you import the application without specifying any of the optional flags, you will be added as the owner of the application in the imported environment. If both the --owner and the --preserveOwner flags are set, then the --owner flag gets higher priority over the --preserveOwner flag.

```
apimcli import-app -f dev/admin_SampleApp.zip -e production -o admin --preserveOwner
--skipSubscriptions -u admin -p admin -k
```

```
ZipFilePath: /Users/kim/.wso2apimcli/exported/apps/dev/admin_SampleApp.zip
Completed importing the Application 'dev/admin_SampleApp.zip'
Successfully imported Application!
```
There are three options to import applications in a single tenant environment.

- Application A in environment 1 is migrated to environment 2. Some APIs may not be available in environment 2 (API B in this case) and the relevant subscriptions are not added in such cases (skipped).

- A different owner can be specified while importing an application to environment 2, without preserving the original user of environment 1.
The original owner of the application can be preserved when the application is imported to environment 2 by adding the `--preserveOwner` flag.
Importing applications in a multi-tenant environment

In a situation where an application has API subscriptions in different tenant domains, such subscriptions are added if the relevant APIs with the target tier, are available in the importing environment. Note that the provider of the API may not be the same in the importing environment.

List apps

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to list the apps.

   **Command Format**
   
   ```
apimcli list apps -e <environment> -u <username> -o <owner> -p <password> -k
   napimcli list apps --environment <environment> --owner <owner> --username <username> --password <password> --insecure
   ```

   **Flags:**

   - `-e <environment>`: Environment.
   - `-u <username>`: Username.
   - `-o <owner>`: Owner.
   - `-p <password>`: Password.
   - `--insecure`: Insecure mode.
**Example**

**Command Format**

1. **Required flags**
   - `--environment, -e`
   - `--owner, -o`
   - `--insecure, -k`: This allows connections to SSL sites without certificates

2. **Optional flags**
   - `--username, -u`
   - `--password, -p`

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

```plaintext
apimcli list apps -e dev -o admin -u admin -p admin -k
```

**Response**

Environment: production
No. of Applications: 2

+--------------------------------------+--------------------+-------+----------+----------+
|                  ID                  |        NAME        | OWNER |  STATUS  | GROUP-ID |
+--------------------------------------+--------------------+-------+----------+----------+
| e13b5bcf-dee5-48fe-9f23-bf46fc17a378 | DefaultApplication |       | APPROVED |          |
| 153ad3d3-fa26-4dda-af54-27ee3327848  | TestApp          |       | APPROVED |          |
+--------------------------------------+--------------------+-------+----------+----------+

**Reset user**

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to reset user details.

   **Command Format**

   ```plaintext
   apimcli reset-user -e <environment>
   apimcli reset-user --environment <environment>
   ```

   **Flags:**
   - **Required flags**
     - `--environment, -e`

   ```plaintext
   apimcli reset-user -e dev
   ```

   Successfully cleared user data for environment: dev

**Check the version of the CLI**

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to check the version of the CLI.

   **Command Format**

   ```plaintext
   apimcli version
   ```
Set HTTP request timeout

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to set the HTTP request timeout.
   Command Format
   ```
   apimcli set --http-request-timeout <http-request-timeout>
   ```
   Flags:
   ```
   --http-request-timeout
   ```
   ```
   apimcli set --http-request-timeout 10000
   ```

Set export directory

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to change the default location of the export directory.
   Command Format
   ```
   apimcli set --export-directory <export-directory-path>
   ```
   Flags:
   ```
   --export-directory
   ```
   ```
   apimcli set --export-directory /Users/kim/Downloads/MyExports
   ```

Understanding the API import/export tool

The API import/export tool uses a RESTful API, protected by basic authentication.

Only the following types of users are allowed to access the API import/export tool.

- A user with the admin role.
- A user with a role that has the API-M Admin, Login, and API Create permissions.

Click here to see a screen shot of the above listed permissions.
To allow access to the import/export feature only for a particular tenant, log in to WSO2 API Manager's Management Console and add the downloaded archive file as a web application to the server.

The 'admin' role is the default role which is specified in the Realm configuration in the `<API-M-HOME>/repository/conf/user-mgt.xml` file. It will be changed if you have changed the value of the `<AdminRole>` parameter as shown below.

```
<Realm>
  <Configuration>
    <AddAdmin>true</AddAdmin>
    <AdminRole>admin</AdminRole>
    ....
  </Configuration>
</Realm>
```

The export functionality

The API export functionality retrieves the information required for the requested API from the registry and databases and generates a ZIP file, which the exporter can download. This exported ZIP file has the following structure:
The structure of the ZIP file is explained below:

<table>
<thead>
<tr>
<th>Sub directory/File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Information</td>
<td></td>
</tr>
<tr>
<td>api.json</td>
<td>contains all the basic information required for an API to be imported to another environment</td>
</tr>
<tr>
<td>swagger.json</td>
<td>contains the API swagger definition</td>
</tr>
<tr>
<td>Documents</td>
<td></td>
</tr>
<tr>
<td>docs.json</td>
<td>contains the summary of all the documents available for the API</td>
</tr>
<tr>
<td>documents with type 'file'</td>
<td>Add the uploaded files for API documentation also</td>
</tr>
<tr>
<td>Image</td>
<td></td>
</tr>
<tr>
<td>icon.&lt;extension&gt;</td>
<td></td>
</tr>
<tr>
<td>WSDL</td>
<td></td>
</tr>
<tr>
<td>&lt;APIName&gt;--&lt;version&gt;.wsdl</td>
<td>WSDL file of the API</td>
</tr>
<tr>
<td>Sequences</td>
<td></td>
</tr>
<tr>
<td>In Sequence</td>
<td></td>
</tr>
<tr>
<td>&lt;Sequence Name&gt;.xml</td>
<td></td>
</tr>
<tr>
<td>Out Sequence</td>
<td></td>
</tr>
<tr>
<td>&lt;Sequence Name&gt;.xml</td>
<td></td>
</tr>
<tr>
<td>Fault Sequence</td>
<td></td>
</tr>
<tr>
<td>&lt;Sequence Name&gt;.xml</td>
<td></td>
</tr>
</tbody>
</table>

The import functionality

The import functionality uploads the exported ZIP file of the API to the target environment. It creates a new API with all the registry and database resources exported from the source environment. Note the following:

- The lifecycle status of an imported API will always be **CREATED** even when the original API in the source environment has a different state. This is to enable the importer to modify the API before publishing it.
- Tiers and sequences are provider-specific. If an exported tier is not already available in the imported environment, that tier is not added to the new environment. However, if an exported sequence is not available in the imported environment, it is added.
- The importer can decide whether to keep the original provider’s name or replace it. Set the **--preserve-provider** flag to true to keep it. If you set it to false, the original provider is replaced by the user who is sending the CLI command.
- Cross-tenant imports are not allowed by preserving the original provider. For example, if an API is exported from tenant A and imported to tenant B, the value of the **--preserve-provider** flag must always be false.

Performance Tuning and Testing Results

The topics in this section provide performance tuning recommendations for the WSO2 API Manager (WSO2 API-M) and the results of performance tests that were carried out on WSO2 API-M.

- **Tuning Performance**
- **WSO2 API-M Performance and Capacity Planning**

Tuning Performance

This section describes some recommended performance tuning configurations to optimize the API Manager. It assumes that you have set up the API Manager on Unix/Linux, which is recommended for a production deployment. We also recommend a distributed API Manager setup for most production systems. Out of all components of an API Manager distributed setup, the API Gateway is the most critical, because it handles all inbound calls to APIs. Therefore, we recommend you to have at least a 2-node cluster of API Gateways in a distributed setup.

- **OS-level settings**
When it comes to performance, the OS that the server runs plays an important role.

Following are the configurations you can apply to optimize OS-level performance:

1. To optimize network and OS performance, configure the following settings in the `/etc/sysctl.conf` file of Linux. These settings specify a larger port range, a more effective TCP connection timeout value, and a number of other important parameters at the OS-level.

   ```
   net.ipv4.tcp_fin_timeout = 30
   fs.file-max = 2097152
   net.ipv4.tcp_tw_recycle = 1
   net.ipv4.tcp_tw_reuse = 1
   net.core.rmem_default = 524288
   net.core.wmem_default = 524288
   net.core.rmem_max = 67108864
   net.core.wmem_max = 67108864
   net.ipv4.tcp_rmem = 4096 87380 16777216
   net.ipv4.tcp_wmem = 4096 65536 16777216
   net.ipv4.ip_local_port_range = 1024 65535
   ```

2. To alter the number of allowed open files for system users, configure the following settings in the `/etc/security/limits.conf` file of Linux (be sure to include the leading `*` character).

   ```
   * soft nofile 4096
   * hard nofile 65535
   ```

   Optimal values for these parameters depend on the environment.

3. To alter the maximum number of processes your user is allowed to run at a given time, configure the following settings in the `/etc/security/limits.conf` file of Linux (be sure to include the leading `*` character). Each carbon server instance you run would require up to 1024 threads (with default thread pool configuration). Therefore, you need to increase the nproc value by 1024 per each carbon server (both hard and soft).

   ```
   * soft nproc 20000
   * hard nproc 20000
   ```

**Important:**

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to familiarize yourself with these files using Unix/Linux documentation before editing them.
- The values we discuss here are general recommendations. They might not be the optimal values for the specific hardware configurations in your environment. We recommend you to carry out load tests on your environment to tune the API Manager accordingly.

**OS-level settings**

**JVM-level settings**

- JVM-level settings
- WSO2 Carbon platform-level settings
- APIM-level settings
- Throttle data and Analytics-related settings

If you are running MacOS Sierra and experiencing long startup times for WSO2 products, try mapping your Mac hostname to 127.0.0.1 and ::1 in the `/etc/hosts` file as described. For example, if your Macbook hostname is "john-mbpro.local", then add the mapping to the canonical 127.0.0.1 address in the `/etc/hosts` file, as shown in the example below.

```
127.0.0.1 localhost john-mbpro.local
```

It is not recommended to use `net.ipv4.tcp_tw_recycle = 1` when working with network address translation (NAT), such as if you are deploying products in EC2 or any other environment configured with NAT.

```
net.ipv4.tcp_fin_timeout = 30
fs.file-max = 2097152
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.core.rmem_default = 524288
net.core.wmem_default = 524288
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.ip_local_port_range = 1024 65535
```
When an XML element has a large number of sub elements and the system tries to process all the sub elements, the system can become unstable due to a memory overhead. This is a security risk.

To avoid this issue, you can define a maximum level of entity substitutions that the XML parser allows in the system. You do this using the entity expansion limit as follows in the `<API-M_HOME>/bin/wso2server.bat` file (for Windows) or the `<API-M_HOME>/bin/wso2server.sh` file (for Linux/Solaris). The default entity expansion limit is 64000.

```
-DentityExpansionLimit=10000
```

In a clustered environment, the entity expansion limit has no dependency on the number of worker nodes.

**WSO2 Carbon platform-level settings**

In multitenant mode, the WSO2 Carbon runtime limits the thread execution time. That is, if a thread is stuck or taking a long time to process, Carbon detects such threads, interrupts and stops them. Note that Carbon prints the current stack trace before interrupting the thread. This mechanism is implemented as an Apache Tomcat valve. Therefore, it should be configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file as shown below.

```
<V valve className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve" threshold="600"/>
```

- The `className` is the Java class used for the implementation. Set it to `org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve`.
- The `threshold` gives the minimum duration in seconds after which a thread is considered stuck. The default value is 600 seconds.

**APIM-level settings**

- Timeout configurations for an API call
- General APIM-level recommendations
- Registry indexing configurations
- Tuning data-agent parameters

Timeout configurations for an API call

The following diagram shows the communication/network paths that occur when an API is called. The timeout configurations for each network call are explained below.

- **Key validation**
  Key validation occurs via a Servlet HTTP call and the connection timeout can be configured by changing the following configuration details in the `<API-M_HOME>/repository/conf/axis2/axis2_client.xml` file. All timeout values are in milliseconds.
If the Key Manager caching is enabled, the calls between the API Gateway and Key Manager are cached. As a result, the Key Manager is not invoked for each API call.

- **Client call API Gateway + API Gateway call Backend**
  For backend communication, the API Manager uses PassThrough transport. This is configured in the `<API-M_HOME>/repository/conf/passthru-http.properties` file. For more information, see Configuring passthru-http.properties in the EI documentation.

  Note that the default value for `http.socket.timeout` differs between WSO2 products. In WSO2 API-M, the default value for `http.socket.timeout` is 180000ms.

General APIM-level recommendations

Some general APIM-level recommendations are listed below:

<table>
<thead>
<tr>
<th>Improvement Area</th>
<th>Performance Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Gateway nodes</td>
<td>Increase memory allocated by modifying the <code>/bin/wso2server.sh</code> file with the following setting:</td>
</tr>
<tr>
<td></td>
<td>-Xms2048m -Xmx2048m -XX:MaxPermSize=1024m</td>
</tr>
<tr>
<td></td>
<td>Set the following in the <code>&lt;API-M_HOME&gt;/repository/conf/axis2/axis2_client.xml</code> file:</td>
</tr>
<tr>
<td></td>
<td>&lt;parameter name=&quot;defaultMaxConnPerHost&quot;&gt;1000&lt;/parameter&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;parameter name=&quot;maxTotalConnections&quot;&gt;30000&lt;/parameter&gt;</td>
</tr>
<tr>
<td></td>
<td>The following Axis2 client configurations are only applicable when Web Services key validation (WS key validation) is enabled.</td>
</tr>
<tr>
<td></td>
<td>The default values mentioned are the values identified at the time of releasing API-M. However, if you want high concurrency, use the values mentioned below:</td>
</tr>
</tbody>
</table>

  The above configurations are only applicable when WS key validation is enabled.
Recommended values for the `<API-M_HOME>/repository/conf/nhttp.properties` file are given below. Note that the commented out values in this file are the default values that will be applied if you do not change anything.

### Property descriptions:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>snd_t_core</code></td>
<td>Transport sender worker pool's initial thread count</td>
</tr>
<tr>
<td><code>snd_t_max</code></td>
<td>Transport sender worker pool's maximum thread count</td>
</tr>
<tr>
<td><code>snd_io_threads</code></td>
<td>Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.</td>
</tr>
<tr>
<td><code>snd_alive_sec</code></td>
<td>Sender-side keep-alive seconds</td>
</tr>
<tr>
<td><code>snd_qlen</code></td>
<td>Sender queue length, which is infinite by default</td>
</tr>
</tbody>
</table>

### Recommended values:

- **HTTP Sender thread pool parameters**
  - `snd_t_core`=200
  - `snd_t_max`=250
  - `snd_alive_sec`=5
  - `snd_qlen`=-1
  - `snd_io_threads`=16

- **HTTP Listener thread pool parameters**
  - `lst_t_core`=200
  - `lst_t_max`=250
  - `lst_alive_sec`=5
  - `lst_qlen`=-1
  - `lst_io_threads`=16

- **timeout parameters**
  - `http.socket.timeout.receiver`: Recommended socket timeout for listener is 180000 ms.
  - `http.socket.timeout.sender`: Recommended socket timeout for sender is 180000 ms.

---

Recommended values for the `<API-M_HOME>/repository/conf/passthru-http.properties` file are given below. Note that the commented out values in this file are the default values that will be applied if you do not change anything.

### Property descriptions:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>worker_thread_keepalive_sec</code></td>
<td>Defines the keep-alive time for extra threads in the worker pool</td>
</tr>
<tr>
<td><code>worker_pool_queue_length</code></td>
<td>Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool</td>
</tr>
<tr>
<td><code>io_threads_per_reactor</code></td>
<td>Defines the number of IO dispatcher threads used per reactor</td>
</tr>
<tr>
<td><code>http.max.connection.per.host.port</code></td>
<td>Defines the maximum number of connections per host port</td>
</tr>
<tr>
<td><code>worker_pool_queue_length</code></td>
<td>Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.</td>
</tr>
</tbody>
</table>

### Recommended values:

- `worker_thread_keepalive_sec`: Default value is 60s. This should be less than the socket timeout.
- `worker_pool_queue_length`: Set to -1 to use an unbounded queue. If a bound queue is used and the queue gets filled to its capacity, any further attempts to submit jobs will fail, causing some messages to be dropped by Synapse. The thread pool starts queuing jobs when all the existing threads are busy and the pool has reached the maximum number of threads. So, the recommended queue length is -1.
- `io_threads_per_reactor`: Value is based on the number of processor cores in the system. (Runtime.getRuntime().availableProcessors())
- `http.max.connection.per.host.port`: Default value is 32767, which works for most systems but you can tune it based on your operating system (for example, Linux supports 65K connections).
  - `worker_pool_size_core`: 400
  - `worker_pool_size_max`: 500
  - `io_buffer_size`: 16384
  - `http.socket.timeout`: 180000

Make the number of threads equal to the number of processor cores.

---

### Timeout configurations

The API Gateway routes the requests from your client to an appropriate endpoint. The most common reason for your client getting a timeout is when the Gateway’s timeout is larger than the client’s timeout values. You can resolve this by either increasing the timeout on the client’s side or by decreasing it on the API Gateway’s side.

Here are a few parameters, in addition to the timeout parameters discussed in the previous sections.
**synapse.global_timeout_interval**

Defines the maximum time that a callback waits in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in the Gateway.

The global timeout is defined in the `<API-M_HOME>/repository/conf/synapse.properties` file. The recommended value is 120000 ms.

**Endpoint-level timeout**

You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.

The example below sets the endpoint to 50 seconds (50000 ms) and executes the fault handler in case of a timeout.

```xml
<timeout>
  <duration>50000</duration>
  <responseAction>fault</responseAction>
</timeout>
```

Alternatively, you can set this through the Publisher UI as well, by following the steps below:

1. Log in to the API Publisher (https://<HostName>:9443/publisher). Select your API and click Edit API.
2. Click the Implement tab and click the cogwheel icon next to the endpoint you want to re-configure.
3. In the Advanced Settings dialog box that appears, increase the duration by modifying the default property set as 3000 ms.

   Note that when the endpoint is suspended, the default action is defined here as invoking the fault sequence.

4. Click Save and re-publish the API.

   The `http.socket.timeout` parameter needs to be adjusted based on the endpoint-level timeout so that it's value is equal or higher than the highest endpoint-level timeout.
If your API is marked as the default version, it has a different template (without the version number) that comes with a pre-defined timeout for the endpoint. This timeout does not change with the changes you do to the API by editing the Advanced Endpoint Configuration. Therefore, if this predefined timeout (60 seconds) is less than the actual API timeout, it triggers the timeout before the actual configured API timeout.

To overcome this, update the default_api_template.xml residing in the `<API-M_HOME>/repository/resources/api_templates` directory by removing the endpoint timeout configuration from the default API. Then, the APIs marked as the default version also trigger the timeout when the actual API timeout is met.

Follow the steps below to update the default_api_template.xml to remove the endpoint configuration for the default APIs.

If you are using a distributed (clustered) setup, follow these steps in the Publisher node as it is the API Publisher that creates the API definition and pushes it to the Gateway.
1. Open the `<API-M_HOME>/repository/resources/api_templates/default_api_template.xml` file and remove the following configuration:

```
<timeout>
  <duration>60000</duration>
  <responseAction>fault</responseAction>
</timeout>
<suspendOnFailure>
  <progressionFactor>1.0</progressionFactor>
</suspendOnFailure>
<markForSuspension>
  <retriesBeforeSuspension>0</retriesBeforeSuspension>
  <retryDelay>0</retryDelay>
</markForSuspension>
```

Add the following configuration to the same place in the `default_api_template.xml` file.

```
<suspendOnFailure>
  <errorCodes>-1</errorCodes>
  <initialDuration>0</initialDuration>
  <progressionFactor>1.0</progressionFactor>
  <maximumDuration>0</maximumDuration>
</suspendOnFailure>
<markForSuspension>
  <errorCodes>-1</errorCodes>
</markForSuspension>
```

2. Add the following configuration to the same place in the `default_api_template.xml` file.

By adding this configuration, you ensure that the APIs marked as the default version never timeout or are suspended using the endpoint configuration defined in the synapse file of the API.

```
<suspendOnFailure>
  <errorCodes>-1</errorCodes>
  <initialDuration>0</initialDuration>
  <progressionFactor>1.0</progressionFactor>
  <maximumDuration>0</maximumDuration>
</suspendOnFailure>
<markForSuspension>
  <errorCodes>-1</errorCodes>
</markForSuspension>
```

3. Go to the API Publisher and republish the default API by clicking **Save and Publish**.
Key Manager nodes

Set the MySQL maximum connections:

```bash
mysql> show variables like "max_connections";
max_connections was 151
set to global max_connections = 250;
```

Set the open files limit to 200000 by editing the `/etc/sysctl.conf` file:

```bash
sudo sysctl -p
```

Set the following in the `<API-M_HOME>/repository/conf/tomcat/catalina-server.xml` file.

If you use WSO2 Identity Server (WSO2 IS) as the Key Manager, then the root location of the above path and the subsequent path needs to change from `<API-M_HOME>` to `<IS_HOME>`.

```xml
<maxThreads>750</maxThreads>
<minSpareThreads>150</minSpareThreads>
<disableUploadTimeout>true</disableUploadTimeout>
<enableLookups>false</enableLookups>
<connectionUploadTimeout>120000</connectionUploadTimeout>
<maxKeepAliveRequests>600</maxKeepAliveRequests>
<acceptCount>600</acceptCount>
```

Set the following connection pool elements in the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file. Time values are defined in milliseconds.

```xml
<maxActive>50</maxActive>
<maxWait>60000</maxWait>
<testOnBorrow>true</testOnBorrow>
<validationQuery>SELECT 1</validationQuery>
<validationInterval>30000</validationInterval>
```

Note that you set the `<testOnBorrow>` element to `true` and provide a validation query (e.g., in Oracle, `SELECT 1 FROM DUAL`), which is run to refresh any stale connections in the connection pool. Set a suitable value for the `<validationInterval>` element, which defaults to 30000 milliseconds. It determines the time period after which the next iteration of the validation query will be run on a particular connection. It avoids excess validations and ensures better performance.

Registry indexing configurations

The registry indexing process is only required to be run on the API Publisher and API Store nodes. To disable the indexing process from running on the other nodes (Gateways and Key Managers), you need to set the `<wso2registry><indexingConfiguration><startIndexing>` element to false in the `<API-M_HOME>/repository/conf/registry.xml` file of the relevant nodes.

**Throttle data and Analytics-related settings**

This section describes the parameters you need to configure to tune the performance of API-M Analytics and Throttling when it is affected by high load, network traffic etc. You need to tune these parameters based on the deployment environment.

**Tuning data-agent parameters**

The following parameters should be configured in the `<APIM_ANALYTICS_HOME>/repository/conf/data-bridge/data-agent-config.xml` file. Note that there are two sub-sections in this file, named **Thrift** and **Binary**.
The Thrift section is related to Analytics and the Binary section is related to Throttling. Same set of parameters mentioned below can be found in both sections. The parameter descriptions and recommendations are intended towards the for performance tuning of Analytics, but the same recommendations are relevant for Throttling data related tuning in the Binary section. Note that the section for Thrift is relevant only if Analytics is enabled.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Tuning Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueueSize</td>
<td>The number of messages that can be stored in WSO2 API-M at a given time before they are published to the Analytics Server.</td>
<td>32768</td>
<td>This value should be increased when the Analytics Server is busy due to a request overload or if there is high network traffic. This prevents the generation of the queue full, dropping message error. When the Analytics server is not very busy and when the network traffic is relatively low, the queue size can be reduced to avoid an overconsumption of memory. The number specified for this parameter should be a power of 2.</td>
</tr>
<tr>
<td>BatchSize</td>
<td>The WSO2 API-M statistical data sent to the Analytics Server to be published in the Analytics Dashboard are grouped into batches. This parameter specifies the number of requests to be included in a batch.</td>
<td>200</td>
<td>This value should be tuned in proportion to the volume of requests sent from WSO2 API-M to the Analytics Server. This value should be reduced if you want to reduce the system overhead of the Analytics Server. This value should be increased if WSO2 API-M is generating a high amount of statistics and if the QueueSize cannot be further increased without causing an overconsumption of memory.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Value</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CorePoolSize</td>
<td>The number of threads allocated to publish WSO2 API-M statistical data to the Analytics Server via Thrift at the time WSO2 API-M is started. This value increases when the throughput of statistics generated increases. However, the number of threads will not exceed the number specified for the MaxPoolSize parameter.</td>
<td>1</td>
<td>The number of available CPU cores should be taken into account when specifying this value. Increasing the core pool size may improve the throughput of statistical data published in the Analytics Dashboard, but latency will also be increased due to context switching.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>The maximum number of threads that should be allocated at any given time to publish WSO2 API-M statistical data to the Analytics Server.</td>
<td>1</td>
<td>Increasing the maximum core pool size may improve the throughput of statistical data published in the Analytics Dashboard, since more threads can be spawned to handle an increased number of events. However, latency will also increase since a higher number of threads would cause context switching to take place more frequently.</td>
</tr>
<tr>
<td>MaxTransportPoolSize</td>
<td>The maximum number of transport threads that should be allocated at any given time to publish WSO2 API-M statistical data to the Analytics Server.</td>
<td>250</td>
<td>This value must be increased when there is an increase in the throughput of events handled by WSO2 API-M Analytics. The value of the tcpMaxWorkerThreads parameter in the <code>&lt;APIM-ANALYTICS_HOME&gt;/itsory/conf/data-bridge/data-bridge-config.xml</code> must change based on the specified for this parameter and the number of data publishers publishing statistics. e.g., the value for this parameter is 250 and the number of data publishers is 7, the value for tcpMaxWorkerThreads parameter must be 1750 (i.e., 7 * 250). This is because you need to ensure that there are enough receiver threads to handle the number of messages published by the publishers.</td>
</tr>
<tr>
<td>SecureMaxTransportPoolSize</td>
<td>The maximum number of secure transport threads that should be allocated at any given time to publish WSO2 API-M statistical data to the Analytics Server.</td>
<td>250</td>
<td>This value must be increased when there is an increase in the throughput of events handled by WSO2 API-M Analytics. The value of the sslMaxWorkerThreads parameter in the <code>&lt;APIM-ANALYTICS_HOME&gt;/itsory/conf/data-bridge/data-bridge-config.xml</code> must change based on the specified for this parameter and the number of data publishers publishing statistics. e.g., the value for this parameter is 250 and the number of data publishers is 7, the value for sslMaxWorkerThreads parameter must be 1750 (i.e., 7 * 250). This is because you need to ensure that there are enough receiver threads to handle the number of messages published by the publishers.</td>
</tr>
</tbody>
</table>

**WSO2 API-M Performance and Capacity Planning**

The following sections analyze the results of WSO2 API Manager performance tests done in the Amazon EC2 environment.
**Summary**

The performance of WSO2 API Manager was measured using the following APIs, which invoke a simple “Netty HTTP Echo Service”. As the name suggests, the Netty service echoes back any request posted to the service.

1. **Echo API**: This is a secured API, which directly invokes the back-end service.
2. **Mediation API**: This is also a secured API, which has a “sequence” as a mediation extension to modify the message.

Tests were done using 100, 200, 300, 1000, and 2000 concurrent users. Concurrent Users mean that there are multiple users accessing the API Gateway at the same time. Different Message Sizes (Payload) were used for the tests with different back-end service delays. The message sizes used are 50B, 1KiB, 10KiB, and 100KiB. The back-end delays were 0ms, 30ms, 500ms, and 1s.

Two key performance metrics were used to measure the performance of each test.

1. **Throughput**: This measures the number of API invocations that the API Manager Gateway server processed during a specific time interval (e.g. per second).
2. **Response Time**: This measures end-to-end processing time for an operation (of invoking an API using HTTPS protocol). The complete distribution of response times were recorded.

The heap size of WSO2 API Manager was increased to 4GB from 2GB, which is the default. Except for increasing the heap size of API Manager, there were no other specific configurations used to optimise the performance of WSO2 API Manager.

With WSO2 API Manager, an average user uses ~1KiB messages and most of the back-ends usually respond in ~30ms. Therefore, let’s look at some charts to understand performance test results for above APIs when using 1KiB messages with 30ms backend delay.

The deployment used was **All-in-one API Manager**.

The following figure shows how the Throughput changes for different number of concurrent users.

Throughput (Requests/sec) vs Concurrent Users for 1KiB messages with 30ms backend delay

![Throughput Chart](image)

Key observations:

- More concurrent users mean more requests to the API Manager Gateway. Therefore, the Throughput of the API Manager Gateway increases as the number of concurrent users accessing the APIs increases. The maximum throughput is obtained for 1000 concurrent users for both “Echo API” and “Mediation API” and the throughput degrades after 1000 concurrent users due to resource contentions in the system. The degradation point mainly depends on the hardware resources.
- Echo API throughput is much better than the Mediation API. Main reason is that the Mediation API has a mediation extension, which uses a “Payload
Ad Factory® Mediator. This mediation in the sequence does a JSON to JSON message transformation. That means, the Mediation API reads the message (payload) to do the message transformation and it has a considerable overhead than the “Echo API”, which is similar to a “Direct Proxy”. A “Direct Proxy” does not perform any processing on the messages that pass through it.

The following figure shows how the Average Response Time changes for different number of concurrent users.

**Key observations:**

- The Average Response Time increases with the number of concurrent users. Since the number of requests to serve increases with more users, there are more resource contentions. Therefore, the number of concurrent users served by the API Gateway needs to be decided on the required response time limits. For example, in order to keep average response time below 100ms for APIs with 30ms backend delay, the maximum concurrent users accessing the API Gateway should be limited to 300.

- The Mediation API response times are higher than Echo API due to the performance overhead of mediation extension.

Let’s look at the 90th, 95th, and 99th Response Time percentiles. This is useful to measure the percentage of requests exceeded the response time value for a given percentile. A percentile can also tell the percentage of requests completed below the particular response time value.

For example, when there are 100 concurrent users, 90th response time percentile for Echo API is 37ms. This means that 10% of the requests have taken more than 37ms to respond. Similarly, 99th response time percentile for Echo API is 86ms, which means that 99% of the requests have completed within 86ms.
Key observations:

- Mediation API is slower than the Echo API due to the performance overhead of mediation extension.
- Response Times percentiles are less than 300ms up to 300 concurrent users.
- For higher concurrent users, 99th percentile of Mediation API response times goes beyond 1 second, which means that 1% of the API invocations took 1 second to 1.4 seconds.

To see the memory usage, the Garbage Collection (GC) logs in the API Manager were enabled and the GC log for each performance test was analyzed using the GCViewer.

The GC Throughput was calculated for each test to check whether GC operations are not impacting the performance of the server. The GC Throughput is the time percentage of the application, which was not busy with GC operations. For example, if the application ran for 10 minutes and 30 seconds were taken for GC operations, the GC Throughput is 1 - \( \frac{30}{60 + 30} \) = 95%. A GC Throughput over 90% is good and that means the allocated heap was enough to handle all concurrent requests, which allocate objects in the memory.

The following chart shows the GC Throughput (%) for different numbers of concurrent users.

---

1000 to 2000 concurrent users mean a lot and it is not very common. To support more concurrent users with acceptable response times, it is recommended to scale horizontally or vertically. When scaling horizontally, two or more Gateway nodes need to be used with a load balancer. To measure the performance after scaling, another load test must be carried out.
Key observations:

- GC Throughput for Mediation API is better than Echo API. The Mediation API processes requests slower than the Echo API, which means that the object allocation rate is also less than the Echo API.

**WSO2 API Manager All-in-one Deployment**

In this deployment, the WSO2 API Manager is deployed in one EC2 instance and a RDS instance is used for databases. The back-end service, which is a simple Netty HTTP Echo Service is deployed in a separate EC2 instance. There are two JMeter servers with a JMeter client in three EC2 instances.

See: JMeter Remote Test. Two JMeter servers are used to simulate high number of concurrent users.

The following are the EC2 instances that the API-M 2.6.0 performance tests were carried out on.

**Synapse Gateway**
<table>
<thead>
<tr>
<th>Name</th>
<th>EC2 Instance Type</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache JMeter Client</td>
<td>c3.large</td>
<td>2</td>
<td>3.75</td>
</tr>
<tr>
<td>Apache JMeter Server 01</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>Apache JMeter Server 02</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>WSO2 API Manager (Synapse)</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>Netty HTTP Backend</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>MySQL</td>
<td>db.m3.medium (RDS)</td>
<td>1</td>
<td>3.75</td>
</tr>
</tbody>
</table>

Ballerina Gateway

<table>
<thead>
<tr>
<th>Name</th>
<th>EC2 Instance Type</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache JMeter Client</td>
<td>c3.large</td>
<td>2</td>
<td>3.75</td>
</tr>
<tr>
<td>Apache JMeter Server 01</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>Apache JMeter Server 02</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>Microgateway</td>
<td>c3.large</td>
<td>4</td>
<td>7.5</td>
</tr>
<tr>
<td>Netty HTTP Backend</td>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
</tr>
</tbody>
</table>

WSO2 API Manager version is 2.2.0 and Apache JMeter version is 3.2.

See the following links for more details on Amazon Instance Types
- [https://aws.amazon.com/ec2/instance-types/](https://aws.amazon.com/ec2/instance-types/)

The operating system is Ubuntu 16.04.2 LTS.
MySQL version in RDS instance was 5.7

Observations from all results

There are key observations for the average user scenario of accessing APIs with 1KiB messages and the back-end service having 30ms delay.

The following are the key observations from the all performance tests done with different message sizes and different backend delays. (See Comparison of results for all charts used to derive the pointed mentioned below)

Key observations related to throughput:

- Throughput increases up to a certain limit when the number of concurrent users increase. Mediation API throughput increase rate is much lower than the Echo API.
- Throughput decreases when the message sizes increase. The Mediation API throughput decrease rate is much higher than the Echo API.
- Throughput decreases when the backend sleep time increase. This observation is similar to both APIs. This means that if the backend takes more time, the request processing rate at the API Manager gateway will be less.

Key observations related to response time:

- Average response time increases when the number of concurrent users increase. The increasing rate of average response time for Mediation API is much higher than the Echo API.
- Average response time increases considerably for Mediation API when the message sizes increase due to the message processing. The average response time of Echo API is not increasing as much as the Mediation API.
- Average Response Time increases when the backend sleep time increases. This observation is similar to both APIs.

Key observations related to GC Throughput:

- The GC throughput decreases when the number of concurrent users increase. When there are more concurrent users, the object allocation rate increases.
- The GC throughput increases when the message sizes increases. The request processing rate slows down due to the time taken to process large messages. Therefore, the object allocation rate decreases when the message sizes increases.
- The GC throughput increases when the backend sleep time increases. The object allocation rate will be low when the backend takes more time to respond.

Backend Service

The backend service used for tests were developed using Netty. Since there can be up to 2000 users, 2000 threads were used for Netty (to avoid the back-end becoming a bottleneck)

The Netty server also has a parameter to specify the number of seconds to sleep to simulate the delays in the backend service)

Performance Testing Tool

As mentioned above, Apache JMeter was used to run load tests. In JMeter, the number of concurrent users was specified and the following details were taken after each test.

- # Samples - The number of requests sent with the given number of concurrent users.
- Error Count - How many request errors were recorded.
- Error % - Percent of requests with errors
- Average - The average response time of a set of results
- Min - The shortest time taken for a request
- Max - The longest time taken for a request
- 90th Percentile - 90% of the requests took no more than this time. The remaining samples took at least as long as this
- 95th Percentile - 95% of the requests took no more than this time. The remaining samples took at least as long as this
- 99th Percentile - 99% of the requests took no more than this time. The remaining samples took at least as long as this
- Throughput - The Throughput is measured in requests per second.
- Received KB/sec - The throughput measured in received Kilobytes per second
- Sent KB/sec - The throughput measured in sent Kilobytes per second

In addition, to above details, some additional details were recorded for every test.

- GC Throughput - Time percentage the application was not busy with GC

GC throughput and other GC related details were obtained from the GC logs produced by the WSO2 API Manager.

The following are the GC flags used:

-XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:$CARBON_HOME/repository/logs/gc.log

The process memory was not considered as Java is working on an already reserved heap area.

Performance Test Scripts

All scripts used to run the performance tests and analyze results are in the following repositories.

**Comparison of results**

- **Throughput Comparison**
  - Throughput (Requests/sec) vs Concurrent Users
  - Throughput (Requests/sec) vs Message Size (Bytes)
  - Throughput (Requests/sec) vs Sleep Time (ms)
- **Average Response Time Comparison**
  - Average Response Time (ms) vs Concurrent Users
  - Average Response Time (ms) vs Message Size (Bytes)
  - Average Response Time (ms) vs Sleep Time (ms)
- **GC Throughput Comparison**
  - API Manager GC Throughput (%) vs Concurrent Users
  - API Manager GC Throughput (%) vs Message Size (Bytes)
  - API Manager GC Throughput (%) vs Sleep Time (ms)

Throughput Comparison

The Echo API has some errors with 100KiB message size for 1000 and 2000 concurrent users.
The following charts show what happens to the server throughput when considering all results.

- Throughput (Requests/sec) vs Concurrent Users

```
Throughput (Requests/sec) vs Concurrent Users

• Throughput (Requests/sec) vs Message Size (Bytes)
```
Throughput (Requests/sec) vs Message Size (Bytes)

- Throughput (Requests/sec) vs Sleep Time (ms)
Average Response Time Comparison
The following charts show what happens to the average response time when considering all results.

- Average Response Time (ms) vs Concurrent Users
Average Response Time (ms) vs Concurrent Users

- Average Response Time (ms) vs Message Size (Bytes)
Average Response Time (ms) vs Message Size (Bytes)

- Average Response Time (ms) vs Sleep Time (ms)
GC Throughput Comparison

The following chart shows the GC throughput behavior when considering all results.
API Manager GC Throughput (%) vs Concurrent Users
API Manager GC Throughput (%) vs Concurrent Users

- API Manager GC Throughput (%) vs Message Size (Bytes)
API Manager GC Throughput (%) vs Message Size (Bytes)

- API Manager GC Throughput (%) vs Sleep Time (ms)
1. Refer for more details on the charts.

### Prevent API Suspension

WSO2 API Manager suspends your API if the endpoint of your API cannot be reached. The default suspension time is 30 seconds. Any request to your API will not be able to reach your endpoint for 30 seconds and will return an error message, as shown below.

```xml
<am:fault xmlns:am="http://wso2.org/apimanager">
  <am:code>303001</am:code>
  <am:type>Status report</am:type>
  <am:message>Runtime Error</am:message>
  <am:description>Currently, Address endpoint : [ Name : somename-AT-sometenant--test_me_APIproductionEndpoint_0 ] [ State : SUSPENDED ]</am:description>
</am:fault>
```

To prevent or turn off API suspension, do the following:

1. Log into API Publisher (https://<HostName>:9443/publisher). Select your API and click **Edit API**.
2. In the **Implement** tab, click the cogwheel icon next to the endpoint you want to re-configure.

3. In the dialog box that appears, set the values for **Initial Duration** and **Max Duration** to zero to turn off suspension.
4. Click Save and re-publish the API.

For more details on creating and publishing an API, see Create and Publish an API.

To avoid backend endpoint suspension,

1. Navigate to the <API-M_HOME>/repository/deployment/server/synapse-configs/default/api folder.
2. Open the configuration file of the API that has to be prevented from being suspended. (e.g. admin--testApi_v1.0.0.xml)
3. Add the following configurations,

```
<endpoint name="admin--testApi_APIproductionEndpoint_0">
  <address url="http://localhost:9000/services/SimpleStockQuoteService">
    <timeout>
      <duration>30000</duration>
      <responseAction>fault</responseAction>
    </timeout>
    <suspendOnFailure>
      <errorCodes>-1</errorCodes>
      <initialDuration>0</initialDuration>
      <progressionFactor>1.0</progressionFactor>
      <maximumDuration>0</maximumDuration>
    </suspendOnFailure>
    <markForSuspension>
      <errorCodes>-1</errorCodes>
    </markForSuspension>
  </address>
</endpoint>
```

For more details on configuring different timeouts, see Timeout configurations for an API call in the Performance Tuning guide.
Removing Unused Tokens from the Database

As you use WSO2 API Manager, the number of revoked, inactive and expired tokens accumulates in the IDN_OAUTH2_ACCESS_TOKEN table. These tokens are kept in the database for logging and audit purposes, but they can have a negative impact on the server’s performance over time. Therefore, it is recommended to clean them periodically.

The following methods can be used for token cleanup:

- Configuring API Manager for token cleanup
- Using stored procedures for token cleanup

**Configuring API Manager for token cleanup**

WSO2 API Manager triggers token cleanup during the following instances.

- Token generation
- Token refresh
- Token revocation

To enable token cleanup, open the `<API-M_HOME>/repository/conf/identity/identity.xml` file and do the following changes.

```
<!-- token cleanup feature config-->
<TokenCleanup>
  <!-- old access token cleaning feature -->
  <EnableTokenCleanup>true</EnableTokenCleanup>
  <!-- old access token will be retain in audit table -->
  <RetainOldAccessToken>true</RetainOldAccessToken>
</TokenCleanup>
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;EnableTokenCleanup&gt;</code></td>
<td>Set this property to <strong>true</strong> to enable token cleanup. Set it to <strong>false</strong> to disable token cleanup.</td>
</tr>
<tr>
<td><code>&lt;RetainOldAccessToken&gt;</code></td>
<td>Set this property to <strong>true</strong> to move the old, invalid tokens to the Audit table when token cleaning is enabled. Set it to <strong>false</strong> if you do not wish to store old tokens in the Audit table.</td>
</tr>
</tbody>
</table>

**Using stored procedures for token cleanup**

Alternatively, you can also use the stored procedures provided below, to run a token cleanup task periodically to remove the old and invalid tokens.

1. Take a backup of the running database.
2. Set up the database dump in a test environment and test it for any issues.

   For more information on setting up a database dump, go to the [MySQL](https://dev.mysql.com/), [SQL Server](https://docs.microsoft.com/sql ), and [Oracle](https://docs.oracle.com) official documentation.

   **Tip**: We recommend you to test the database dump before the cleanup task as the cleanup can take some time.

3. Depending on your database, select the appropriate token cleanup script from here and run it on the database dump. This takes a backup of the necessary tables, turns off SQL updates and cleans the database of unused tokens.
4. Once the cleanup is over, start the API Manager pointing to the cleaned-up database dump and test thoroughly for any issues.
   You can also schedule a cleanup task that will automatically run after a given period of time. Here's an example:
   MySQL: `SQL Server`
USE 'WSO2AM_DB';
DROP EVENT IF EXISTS 'cleanup_tokens_event';
CREATE EVENT 'cleanup_tokens_event'
ON SCHEDULE
    EVERY 1 WEEK STARTS '2018-01-01 00:00.00'
DO
    CALL 'WSO2AM_DB'.'cleanup_tokens'();
-- 'Turn on the event_scheduler'
SET GLOBAL event_scheduler = ON;

USE WSO2AM_DB;
GO
-- Creates a schedule named CleanupTask.
-- Jobs that use this schedule execute every day when the time on the server is 01:00.
EXEC sp_add_schedule
    @schedule_name = N'CleanupTask',
    @freq_type = 4,
    @freq_interval = 1,
    @active_start_time = 010000;
GO
-- attaches the schedule to the job BackupDatabase
EXEC sp_attach_schedule
    @job_name = N'BackupDatabase',
    @schedule_name = N'CleanupTask';
GO

Replace WSO2AM_DB with the name of your API Manager database in the above script.

Revoke OAuth2 Application

1. An OAuth client is created when an application access token is generated. When a subscriber creates an application and generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth client and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database.

You can revoke the access tokens issued for the application by following the instructions below
   a. Log in to the management console (https://<HostName>:9443/carbon)
   b. In the Main menu, click Service Providers List.
c. Select the **Service Provider** and click **Edit**.

Only a user with the relevant permission to manage Service Providers will be able to view/edit the existing Service Providers. A user with only subscribe permission is not able to view or edit the Service Providers. The Admin user by default has permission only to view and edit Service Providers of Applications created by the same user. However, it's possible for the Admin user to assign the relevant permission for a particular Service Provider, to itself by navigating to **Users and Roles List Roles**, and then view/edit the Service Provider.

d. Expand the **Inbound Authentication Configurations** section and select **OAuth/OpenID Configuration**

e. You can revoke/deactivate the OAuth application by clicking **Revoke**. This will revoke all the tokens given for the application.
1. To regenerate the secret of the OAuth Application, click **Regenerate Secret**.

After an OAuth application is revoked, the consumer secret and all the generated access tokens and authorization codes will be invalid. You will not be able to regenerate access tokens in the API store or using the Token API.

To re-activate the application, the consumer secret must be regenerated as shown in the next step.

You have to generate new access tokens and authorization codes for the OAuth application through the API Store or using the Token API after regenerating the consumer secret.

### Sharing Applications Between Multiple Groups

WSO2 API Manager provides the facility for users to share their applications and subscriptions with a specific logical group/groups such as an organization. Users can view and modify applications and subscriptions belonging to other users in the same group.

By default, API Manager considers the organization name you enter when signing up to the API Store as the group ID. All the users specifying the same organization name belong to the same group and therefore, can view each others’ subscriptions and applications. API Manager also provides flexibility to change this default authentication implementation:

- Enabling Multi-Group Sharing
- Using the group sharing feature
- Extending the group ID extractor

#### Enabling Multi-Group Sharing

You can enable application sharing between multiple groups by following the steps below.

1. Shutdown the server if its running

```xml
<GroupingExtractor>org.wso2.carbon.apimgt.impl.DefaultGroupIDExtractorImpl</GroupingExtractor>
```

This default extractor does not work with SAML SSO. To enable SAML SSO, you need to write a custom implementation using the `SA_MLGroupIDExtractorImpl.java` class.

3. Restart the server.

#### Using the group sharing feature

Group IDs are extracted using a `GroupingExtractor` class which is an implementation of `NewPostLoginExecutor` interface. The default implementation is done through the `DefaultGroupIDExtractorImpl` class. The organization claim is extracted using the group ID. If a particular user is in more than one organization, provide the organizations as a string separated by commas.

The steps below show how to use the group sharing feature:

1. Start WSO2 API Manager and click **Sign-up**.
2. Sign up to the API store as two different users (e.g., usera, userb) belonging to the same organizations. Click Show Additional Details to set the organization.
3. Sign in as **usera** and add application App_A.
4. Enter the Group ID as **org1** and press enter. Click **Add.** App_A will be shared with all the users in **org1** group.
5. Sign out of the API Store. Sign in as `userb`.
6. Go to the Applications tab. You will see App_A which was added by `usera`.

Note that the name of the application creator is appended to the application name to differentiate the applications.

7. Subscribe to the default API using App_A.
8. Log in to the API Store as usera. The subscriptions for App_A by userb will be displayed.

Extending the group ID extractor

The default implementation picks the organization claim as the group ID. The organization names are returned in a string array. To use a different claim or a different type of group ID, should create your own group ID extractor class by extending NewPostLoginExecutor interface and overriding the method below.

```java
String[] getGroupingIdentifierList(String response);
```

This particular method will be called when a user is logging into the store and it will return all the groupIds for the logged in user. After logging in users will be able to see the applications created by themselves, and the applications shared with groupIds returned by getGroupingIdentifierList Method.

Transforming API Message Payload

When a request comes to the API Manager, it sends the response in the same format of the request. For example, the API Manager handles JSON to JSON transformations out of the box. If the backend does not accept messages of the same content type of the request message, it must be transformed to a different format. The API Gateway of the API Manager handles these transformations using message builders and formatters.

When a message comes in to the API Gateway, the receiving transport selects a message builder based on the message's content type. It uses that builder to process the message's raw payload data and convert it into JSON. Conversely, when sending a message out from the Gateway, a message formatter is used to build the outgoing stream from the message. As with message builders, the message formatter is selected based on the message's content type.

- JSON message builders and formatters
- XML representation of JSON payloads
- Converting a payload between XML and JSON

Note that if you edit an API's synapse configuration as mentioned in this guide and then go back to the API Publisher and save the API, your changes will be overwritten. Therefore, we do not recommend changing the API's synapse configuration directly. The recommended way to extend an API's mediation flow is by engaging In/Out sequences.

Also see the following sections in the WSO2 EI documentation. WSO2 EI is used to implement the API Gateway through which API messages are
transformed:

- Accessing content from JSON payloads
- Logging JSON payloads
- Constructing and transforming JSON payloads
- Troubleshooting, debugging, and logging

**JSON message builders and formatters**

There are two types of message builders and formatters for JSON. The default builder and formatter keep the JSON representation intact without converting it to XML. You can access the payload content using the JSON Path or XPath and convert the payload to XML at any point in the mediation flow.

- `org.apache.synapse.commons.json.JsonStreamBuilder`
- `org.apache.synapse.commons.json.JsonStreamFormatter`

If you want to convert the JSON representation to XML before the mediation flow begins, use the following builder and formatter instead. Note that some data loss can occur during the JSON -> XML -> JSON conversion process.

- `org.apache.synapse.commons.json.JsonBuilder`
- `org.apache.synapse.commons.json.JsonFormatter`

The builders and formatters are configured respectively in the `messageBuilders` and `messageFormatters` sections of the Axis2 configuration files located in the `<PRODUCT_HOME>/repository/conf/axis2` directory. Both types of JSON builders use StAXON as the underlying JSON processor.

The following builders and formatters are also included for compatibility with older API Manager versions:

- `org.apache.axis2.json.JSONBuilder/JSONMessageFormatter`
- `org.apache.synapse.commons.json.JsonStreamBuilder/JSONStreamFormatter`
- `org.apache.axis2.json.JSONBadgerfishOMBuilder/JSONBadgerfishMessageFormatter`

If you want to handle JSON payloads that are sent using a media type other than `application/json`, you must register the JSON builder and formatter for that media type in the following two files at minimum (for best results, register them in all Axis2 configuration files found in the `<PRODUCT_HOME>/repository/conf/axis2` directory):

- `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml`
- `<PRODUCT_HOME>/repository/conf/axis2/axis2_blocking_client.xml`

For example, if the media type is `text/javascript`, register the message builder and formatter as follows:

```xml
<messageBuilder contentType="text/javascript" class="org.apache.synapse.commons.json.JsonStreamBuilder"/>
<messageFormatter contentType="text/javascript" class="org.apache.synapse.commons.json.JsonStreamFormatter"/>
```

Always use the same type of builder and formatter combination. Mixing different builders and formatters will cause errors at runtime.

To support having spaces inside JSON attributes, change the default JSON builder and formatter to the following pair in either `<API-M_HOME>/repository/conf/axis2/axis2.xml (super tenant scenario)` or `<API-M_HOME>/repository/conf/axis2/tenant-axis2.xml (tenant scenario)` file:

```xml
<messageBuilder contentType="application/json" class="org.apache.synapse.commons.json.JsonStreamBuilder"/>
<messageFormatter contentType="application/json" class="org.apache.synapse.commons.json.JsonStreamFormatter"/>
```

To support use cases for JSON payloads with arrays, change the default JSON builder and formatter to the following pair in either `<API-M_HOME>/repository/conf/axis2/axis2.xml (super tenant scenario)` or `<API-M_HOME>/repository/conf/axis2/tenant-axis2.xml (tenant scenario)` file:
XML representation of JSON payloads

When building the XML tree, JSON builders attach the converted XML infoset to a special XML element that acts as the root element of the final XML tree. If the original JSON payload is of type object, the special element is `<jsonObject/>`. If it is an array, the special element is `<jsonArray/>`. Following are examples of JSON and XML representations of various objects and arrays.

**Null objects**

**JSON:**

```json
{"object":null}
```

**XML:**

```xml
<jsonObject>
 <object/>
</jsonObject>
```

**Empty objects**

**JSON:**

```json
{"object":{}}
```

**XML:**

```xml
<jsonObject>
 <object/>
</jsonObject>
```
Empty strings

JSON:

{ "object": "" }

XML:

<jsonObject>
   <object></object>
</jsonObject>

Empty array

JSON:

[ ]

XML (JsonStreamBuilder):

<jsonArray></jsonArray>

XML (JsonBuilder):

<jsonArray>
   <?xml-multiple jsonElement?>
</jsonArray>

Named arrays

JSON:

{ "array": [1, 2] }

XML (JsonStreamBuilder):

<jsonObject>
   <array>1</array>
   <array>2</array>
</jsonObject>

XML (JsonBuilder):

<jsonObject>
   <array>1</array>
   <array>2</array>
</jsonObject>
Anonymous arrays

JSON:

`[1, 2]`

XML (JsonStreamBuilder):

```xml
<jsonArray>
  <jsonElement>1</jsonElement>
  <jsonElement>2</jsonElement>
</jsonArray>
```

XML (JsonBuilder):

```xml
<jsonArray>
  <jsonElement>1</jsonElement>
  <jsonElement>2</jsonElement>
</jsonArray>
```
XML (JsonBuilder):

```xml
<jsonArray>
  <jsonElement>1</jsonElement>
  <jsonElement>
    <jsonArray>1</jsonArray>
  </jsonElement>
</jsonArray>
```

**XML processing instructions (PIs)**

Note that the addition of `xml-multiple` processing instructions to the XML payloads whose JSON representations contain arrays. JsonBuilder (via StAXON) adds these instructions to the XML payload that it builds during the JSON to XML conversion so that during the XML to JSON conversion, JsonFormatter can reconstruct the arrays that are present in the original JSON payload. JsonFormatter interprets the elements immediately following a processing instruction to construct an array.

**Special characters**

When building XML elements, the `$` character and digits are handled in a special manner when they appear as the first character of a JSON key. Following are examples of two such occurrences. Note the addition of the `_JsonReader_PS_` and `_JsonReader_PD_` prefixes in place of the `$` and digit characters, respectively.

**JSON:**

```json
{"$key":1234}
```

**XML:**

```xml
<jsonObject>
  <_JsonReader_PS_key>1234</_JsonReader_PS_key>
</jsonObject>
```

**JSON:**

```json
{"32x32":"image_32x32.png"}
```

**XML:**

```xml
<jsonObject>
  <_JsonReader_PD_32X32>image_32x32.png</_JsonReader_PD_32X32>
</jsonObject>
```

**Converting a payload between XML and JSON**

To convert an XML payload to JSON, set the `messageType` property to `application/json` in the axis2 scope before sending message to an endpoint. Similarly, to convert a JSON payload to XML, set the `messageType` property to `application/xml` or `text/xml`. For example:
An example command to invoke above API:

curl -v -X POST -H "Content-Type:application/xml" -H "Authorization: Bearer xxx" -d@request1.xml "http://10.100.1.110:8280/tojson/1.0"

If the request payload is as follows:
The response payload will look like this:

```
{
  "coordinates":{
    "location":[
      {
        "name":"Bermuda Triangle",
        "n":25.0000,
        "w":71.0000
      },
      {
        "name":"Eiffel Tower",
        "n":48.8582,
        "e":2.2945
      }
    ]
  }
}
```

Note that we have used the Property mediator to mark the outgoing payload to be formatted as JSON. For more information about the Property Mediator, see the Property Mediator page on WSO2 EI documentation.

```
<property name="messageType" value="application/json" scope="axis2"/>
```

Similarly if the response message needs to be transformed, set the messageType property in the outSequence.
<api name="admin--TOJSON" context="/tojson" version="1.0" version-type="url">
  <resource methods="POST GET DELETE OPTIONS PUT" url-mapping="/">
    <inSequence>
      <property name="POST_TO_URI" value="true" scope="axis2"/>
      <filter source="$ctx:AM_KEY_TYPE" regex="PRODUCTION">
        <then>
          <send>
            <endpoint name="admin--Test_APIproductionEndpoint_0">
              <http uri-template="http://localhost:9767/services/StudentService">
                <timeout>
                  <duration>30000</duration>
                  <responseAction>fault</responseAction>
                </timeout>
                <suspendOnFailure>
                  <errorCodes>-1</errorCodes>
                  <initialDuration>0</initialDuration>
                  <progressionFactor>1.0</progressionFactor>
                  <maximumDuration>0</maximumDuration>
                </suspendOnFailure>
                <markForSuspension>
                  <errorCodes>-1</errorCodes>
                </markForSuspension>
              </http>
            </endpoint>
          </send>
        </then>
        <else>
          <sequence key="_sandbox_key_error_"/>
        </else>
      </filter>
    </inSequence>
    <outSequence>
      <property name="messageType" value="application/json" scope="axis2"/>
      <send/>
    </outSequence>
  </resource>
  <handlers>
    <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
    <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
      <property name="id" value="A"/>
      <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/>
    </handler>
    <handler class="org.wso2.carbon.apimgt.usage.publisher.APIUsageHandler"/>
    <handler class="org.wso2.carbon.apimgt.usage.publisher.APIManagementTrackingHandler"/>
    <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
  </handlers>
</api>

**XML to JSON Transformation Parameters**

See [JSON Transformation Parameters](#) for additional parameters for converting XML to JSON.

**Whitelisting Host Names for API Store**

API Manager provides the capability to whitelist multiple host names if you use different host names to access API Store in your environment.

In this case, **localhost** is by default considered as a whitelisted host name.

Similarly you can whitelist multiple host names for store as follows.
You need to add the host names to the `whiteListedHostNames` attribute in `<APIM_HOME>/repository/deployment/server/jaggeryapp/store/site/conf/site.json` as comma separated values.

See the following example configuration.

```
"whiteListedHostNames": ["www.wso2.org", "www.example.com"]
```

**Note:**

When you try to access API Store with a host which is not whitelisted, or is not specified in `<APIM_HOME>/repository/conf.carbon.xml`, you will notice the following warning being logged in the server logs.

```
Possible HOST Header Attack is identified. Hence, rewriting to default host in configuration.
```

### Working with Access Tokens

When an Application Developer registers an Application on the API Store, the Application is given a consumer-key and a consumer-secret, which represents the credentials of the Application that is being registered. The consumer-key becomes the unique identifier of the Application, similar to a user's username, and is used to authenticate users. When an Access Token is issued for the Application, it is issued against the latter mentioned consumer-key.

API consumers generate access tokens and pass them in the incoming API requests. The API key (i.e., the generated access token) is a simple string that you need to pass as an HTTP header. For example, "Authorization: Bearer NtBQkXoKElu0H1a1fQ0DWfo6IX4a." This works equally well for SOAP and REST calls.

Authorizing requests, which come to published APIs, using access tokens helps you prevent certain types of denial-of-service (DoS) attacks. If the token that is passed with a request is invalid, WSO2 API Manager (WSO2 API-M) discards that request in the first stage of processing itself.

WSO2 API Manager provides two types of access tokens for authentication:

- **Application Access Tokens**: Tokens to identify and authenticate an entire application. An application is a logical collection of many APIs. With a single application access token, you can invoke all of these APIs.
- **User Access Tokens**: Tokens to identify the final user of an application. For example, the final user of a mobile application deployed on different devices.

In WSO2 API-M the access token must be unique for the following combinations - `CONSUMER_KEY`, `AUTH2_USER`, `USER_TYPE`, `TOKEN_STATE`, `TOKEN_STATE_ID` and `TOKEN_SCOPE`. The latter mentioned constraint is defined in the `IDN_OAUTH2_ACCESS_TOKEN` table. Therefore, it is not possible to have more than one Access Token for any of the above combinations.

Let's take a look at how to generate and renew each type of access token.

- Generating application access tokens
- Generating user access tokens
- Generating access tokens per device
- Renewing application access tokens
- Renewing user access tokens
- Changing the default token expiration time

### Generating application access tokens

Application access tokens are tokens that authenticate an application, which is a logical collection of APIs. You can access all APIs associated with an application using a single token, and also subscribe multiple times to a single API with different service level agreement (SLA) levels/tiers. Application access tokens leverage OAuth2 to provide simple key management.

The steps below describe how to generate/renew application access tokens:

1. Sign in to WSO2 API Store.
2. Click the **Applications** menu and open the application for which you want to generate an access token.
3. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. You can use this token to invoke all APIs that you subscribe to using the same application.
Generating user access tokens

User access tokens are tokens that authenticate the final user of an API, and are valid for all APIs subscribed to a user via a particular application. User access tokens allow you to invoke an API even from a third-party application such as a mobile app. You generate/renew a user access token by calling the Login API through a REST client. For more information, see Token API.

By default, access tokens and consumer secrets are not saved in an encrypted format in the database. An admin can enable encryption following the instructions in Encrypting OAuth Keys.

Tip: If you want to maintain authorization headers in messages, which are going out from the API Gateway, an admin can go to the <API_Gateway_node>/repository/conf/api-manager.xml file, uncomment the <RemoveOAuthHeadersFromOutMessage> element, set its value to false, and then restart the server to apply the changes.

Note that when a user is deleted, the access token is automatically invalidated.

Generating access tokens per device

In the Access token validity period field, you can set an expiration period to determine the validity period of the token after generation. Set this to a negative value to ensure that the token never expires. Also see Changing the default token expiration time.

Tip: When you generate access tokens to APIs protected by scope/s, you can select the scope/s and then generate the token for it.
WSO2 API Manager returns the same token repeatedly if a valid token exists for the requesting Application, on behalf of the user. However, the latter mentioned scenario becomes an issue if the same user is using the same Application in two devices (e.g., if you have two instances of the same Application running on your iPhone and iPad, and your iPhone already has a token on behalf of you, your iPad will get the same token if you requested for it within the same validity period. Therefore, if one of your devices revoke this token (e.g., revoke on logout), the token that you obtained for your other device becomes invalid as the devices use the identical tokens.

To overcome this problem, WSO2 API Manager provides a mechanism, with the use of OAuth2.0 Scopes, for obtaining a unique Access Token for each device that uses the same Application. Thereby, allowing users to request tokens for different scopes. You need to prefix the scope names with the string "device_". WSO2 API Manager uses special treatment for the scopes that are prefixed with the latter mentioned string by ignoring the usual validations it does when issuing tokens that are associated to scopes. The following is a sample cURL command that you can use to request a token with a "device_" scope.

```
```

Each token request that is made with a different scope, results in a different access token being issued. For example if you received a token named abc as a result of the scope device_ipad, you will not receive abc when you request for the token with the scope device_iphone. Note that you can use device_scopes in conjunction with other scopes as usual.

**Renewing application access tokens**

When an application access token expires, consumers can refresh the token by signing into the API Store, opening the application, and clicking Re-generate that appears in the Production Keys tab. You can also specify a token expiration time for the application access token. Set this to a negative value to ensure that the token never expires.
Renewing user access tokens

To renew a user token, issue a REST call to the WSO2 Login API through a REST client. For more information, see Token API.

Changing the default token expiration time

Access tokens have an expiration time, which is set to 60 minutes by default.

- To change the default expiration time of application access tokens,
• Change the value of the `<AccessTokenDefaultValidityPeriod>` element in the `<API-M_HOME>/repository/conf/identity/identity.xml` file. Set this to a negative value to ensure that the token never expires. Changes to this value are applied only to the new applications that you create.

**Example**

```
<AccessTokenDefaultValidityPeriod>-3600</AccessTokenDefaultValidityPeriod>
```

• Alternatively, you can set a default expiration time through the UI when generating/regenerating the application access token. This is explained in previous sections.

• Similarly, to change the default expiration time of user access tokens, edit the value of the `<UserAccessTokenDefaultValidityPeriod>` element in the `<API-M_HOME>/repository/conf/identity/identity.xml` file.

**Example**

```
<UserAccessTokenDefaultValidityPeriod>3800</UserAccessTokenDefaultValidityPeriod>
```

Also see Configuring Caching for several caching options available to optimize key validation.

### Working with Databases

The default database that WSO2 API Manager uses to store registry, user manager and product-specific data is the `WSO2CARBON_DB.h2.db` H2 database located in the `<PRODUCT_HOME>/repository/database` folder.

This embedded H2 database is suitable for development and testing. For most production environments, however, we recommend you to use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. You can use the scripts provided with WSO2 products to install and configure several other types of relational databases, including MySQL, IBM DB2, Oracle, and more.

The following sections explain how to change the default database:

- Setting up the Physical Database
- Managing Datasources

### Managing Datasources

A datasource provides information that a server can use to connect to a database. Datasource management is provided by the following feature in the WSO2 feature repository:

Name: WSO2 Carbon - datasource management feature
Identifier: org.wso2.carbon.datasource.feature.group

If datasource management capability is not included in your product by default, add it by installing the above feature, using the instructions given under the Feature Management section of this documentation.

Click **Data Sources** on the **Configure** tab of the product's management console to view, edit, and delete the datasources in your product instance.

You can view, edit, and delete the datasources in your product instance by clicking **Data Sources** on the **Configure** tab of the product management console. However, you cannot edit or delete the default `<WSO2_CARBON_DB>` datasource.

### Adding Datasources

If the datasource management feature is installed in your WSO2 product instance, you can add datasources that allow the server to connect to databases and other external data stores.

Use the following steps to add a datasource:

1. In the product management console, click **Data Sources** on the **Configure** tab.
2. Click Add Data Source.
3. Specify the required options for connecting to the database. The available options are based on the type of datasource you are creating:
   - Configuring a RDBMS Datasource
   - Configuring a Custom Datasource

After adding datasources, they will appear on the Data Sources page. You can edit and delete them as needed by clicking Edit or Delete links.

When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to <PRODUCT_HOME>/repository/components/lib.

Configuring an RDBMS Datasource

When adding a datasource, if you select RDBMS as the datasource type, the following screen appears:

This is the default RDBMS datasource configuration provided by WSO2. You can also write your own RDBMS configuration by selecting the custom datasource option. Enter values for the following fields when using the default RDBMS datasource configuration:

- **Data Source Type**: RDBMS
- **Name**: Name of the datasource (must be a unique value)
- **Data Source Provider**: Specify the datasource provider.
- **Driver**: The class name of the JDBC driver to use. Make sure to copy the JDBC driver relevant to the database engine to the <PRODUCT_HOME>/repository/components/lib/ directory. For example, if you are using MySQL, specify com.mysql.jdbc.Driver as the driver and copy mysql-connector-java-5.XX-bin.jar file to this directory. If you do not copy the driver to this directory when you create the datasource, you will get an exception similar to Cannot load JDBC driver class com.mysql.jdbc.Driver.
- **URL**: The connection URL to pass to the JDBC driver to establish the connection.
**User Name**: The connection user name that will be passed to the JDBC driver to establish the connection.

**Password**: The connection password that will be passed to the JDBC driver to establish the connection.

**Expose as a JNDI Data Source**: Allows you to specify the JNDI datasource.

**Data Source Configuration Parameters**: Allows you to specify the datasource connection pool parameters when creating a RDBMS datasource.

For more details on datasource configuration parameters, see [Apache Tomcat JDBC Connection Pool guide](#).

After creating datasources, they appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

---

**Configuring the Datasource Provider**

A datasource provider connects to a source of data such as a database, accesses its data, and returns the results of the access queries. When creating a RDBMS datasource, use the default provider or link to an external provider. Default datasource provider

To use the default datasource provider, select **default**, and then enter the Driver, URL, User Name, and Password connection properties as follows:

![New Data Source](#)

External datasource provider

If you need to add a datasource supported by an external provider class such as `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`, select **External Data Source**, click **Add Property**, and then enter the name and value of each connection property you need to configure. Following is an example datasource for an external datasource provider:

![New Data Source](#)

---

**Configuring a JNDI Datasource**

Java Naming and Directory Interface (JNDI) is a Java Application Programming Interface (API) that provides naming and directory functionality for Java
software clients, to discover and look up data and objects via a name. It helps decoupling object creation from the object look-up. When you have registered a datasource with JNDI, others can discover it through a JNDI look-up and use it.

When adding a datasource, to expose a RDBMS datasource as a JNDI datasource, click **Expose as a JNDI Data Source** to display the JNDI fields as follows:

Following are descriptions of the JNDI fields:

- **Name**: Name of the JNDI datasource that will be visible to others in object look-up.
- **Use Data Source Factory**: To make the datasource accessible from an external environment, you must use a datasource factory. When this option is selected, a reference object will be created with the defined datasource properties. The datasource factory will create the datasource instance based on the values of the reference object when accessing the datasource from an external environment. In the datasource configuration, this is set as: `<jndiConfig useDataSourceFactory="true">`.
- **JNDI Properties**: Properties related to the JNDI datasource (such as password). When you select this option, set the following properties:
  - `java.naming.factory.initial`: Selects the registry service provider as the initial context.
  - `java.naming.provider.url`: Specifies the location of the registry when the registry is being used as the initial context.

### Configuring the Datasource Connection Pool Parameters

When the server processes a database operation, it spawns a database connection from an associated datasource. After using this connection, the server returns it to the pool of connections. This is called datasource connection pooling. It is a recommended way to gain more performance/throughput in the system. In datasource connection pooling, the physical connection is not dropped with the database server, unless it becomes stale or the datasource connection is closed.

RDBMS datasources in WSO2 products use Tomcat JDBC connection pool (`org.apache.tomcat.jdbc.pool`). It is common to all components that access databases for data persistence, such as the registry, user management (if configured against a JDBC userstore), etc.

You can configure the datasource connection pool parameters, such as how long a connection is persisted in the pool, using the datasource configuration parameters section that appears in the product management console when creating a datasource. Click and expand the option as shown below:
Following are descriptions of the parameters you can configure. For more details on datasource configuration parameters, see Apache Tomcat JDBC.
### Connection Pool guide.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction isolation</td>
<td>The default TransactionIsolation state of connections created by this pool are as follows:</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_UNKNOWN&lt;br&gt;- TRANSACTION_NONE&lt;br&gt;- TRANSACTION_READ_COMMITTED&lt;br&gt;- TRANSACTION_READ_UNCOMMITTED&lt;br&gt;- TRANSACTION_REPEATABLE_READ&lt;br&gt;- TRANSACTION_SERIALIZABLE</td>
</tr>
<tr>
<td>Initial Size (int)</td>
<td>The initial number of connections created, when the pool is started. Default value is zero.</td>
</tr>
<tr>
<td>Max. Active (int)</td>
<td>Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.</td>
</tr>
<tr>
<td>Max. Idle (int)</td>
<td>Maximum number of connections that should be kept in the pool at all times. Default value is 8. Idle connections are checked periodically (if enabled), and connections that have been idle for longer than minEvictableIdleTimeMillis will be released. (also see testWhileIdle).</td>
</tr>
<tr>
<td>Min. Idle (int)</td>
<td>Minimum number of established connections that should be kept in the pool at all times. The connection pool can shrink below this number, if validation queries fail. Default value is zero. For more information, see testWhileIdle.</td>
</tr>
<tr>
<td>Max. Wait (int)</td>
<td>Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. Default value is 30000 (30 seconds).</td>
</tr>
<tr>
<td>Validation Query (String)</td>
<td>The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are SELECT 1 (mysql), select 1 from dual (oracle), SELECT 1 (MS Sql Server).</td>
</tr>
<tr>
<td>Test On Return (boolean)</td>
<td>Used to indicate if objects will be validated before returned to the pool. The default value is false.</td>
</tr>
<tr>
<td></td>
<td>For a true value to have any effect, the validationQuery parameter must be set to a non-null string.</td>
</tr>
<tr>
<td>Test On Borrow (boolean)</td>
<td>Used to indicate if objects will be validated before borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. Default value is false.</td>
</tr>
<tr>
<td></td>
<td>For a true value to have any effect, the validationQuery parameter must be set to a non-null string. In order to have a more efficient validation, see validationInterval .</td>
</tr>
<tr>
<td>Test While Idle (boolean)</td>
<td>The indication of whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool. The default value is false and this property has to be set in order for the pool cleaner/test thread to run. For more information, see timeBetweenEvictionRunsMillis .</td>
</tr>
<tr>
<td></td>
<td>For a true value to have any effect, the validationQuery parameter must be set to a non-null string.</td>
</tr>
<tr>
<td>Time Between Eviction Runs Mills (int)</td>
<td>Number of milliseconds to sleep between runs of the idle connection validation/cleaner thread. This value should not be set under 1 second. It indicates how often we check for idle, abandoned connections, and how often we validate idle connections. The default value is 5000 (5 seconds).</td>
</tr>
<tr>
<td>Minimum Evictable Idle Time (int)</td>
<td>Minimum amount of time an object may sit idle in the pool before it is eligible for eviction. The default value is 60000 (60 seconds).</td>
</tr>
<tr>
<td>Remove Abandoned (boolean)</td>
<td>Flag to remove abandoned connections if they exceed the removeAbandonedTimeout. If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the removeAbandonedTimeout. Setting this to true can recover database connections from applications that fail to close a connection. For more information, see logAbandoned. The default value is false.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Remove Abandoned Timeout (int)</td>
<td>Timeout in seconds before an abandoned (in use) connection can be removed. The default value is 60 (60 seconds). The value should be set to the longest running query that your applications might have.</td>
</tr>
<tr>
<td>Log Abandoned (boolean)</td>
<td>Flag to log stack traces for application code which abandoned a connection. Logging of abandoned connections, adds overhead for every connection borrowing, because a stack trace has to be generated. The default value is false.</td>
</tr>
<tr>
<td>Auto Commit (boolean)</td>
<td>The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default. If not set, then the setAutoCommit method will not be called.</td>
</tr>
<tr>
<td>Default Read Only (boolean)</td>
<td>The default read-only state of connections created by this pool. If not set then the setReadOnly method will not be called. (Some drivers don't support read only mode. For example: Informix)</td>
</tr>
<tr>
<td>Default Catalog (String)</td>
<td>The default catalog of connections created by this pool.</td>
</tr>
<tr>
<td>Validator Class Name (String)</td>
<td>The name of a class which implements the org.apache.tomcat.jdbc.pool.Validator interface and provides a no-arg constructor (may be implicit). If specified, the class will be used to create a Validator instance, which is then used instead of any validation query to validate connections. The default value is null. An example value is com.mycompany.project.SimpleValidator.</td>
</tr>
<tr>
<td>Connection Properties (String)</td>
<td>Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be [propertyN\ame=property];*. The default value is null. The user and password properties will be passed explicitly, so that they do not need to be included here.</td>
</tr>
<tr>
<td>Init SQL</td>
<td>Ability to run a SQL statement exactly once, when the connection is created.</td>
</tr>
<tr>
<td>JDBC Interceptors</td>
<td>Flexible and pluggable interceptors to create any customizations around the pool, the query execution and the result set handling.</td>
</tr>
<tr>
<td>Validation Interval (long)</td>
<td>To avoid excess validation, only run validation at most at this frequency - time in milliseconds. If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).</td>
</tr>
<tr>
<td>JMX Enabled (boolean)</td>
<td>Register the pool with JMX or not. The default value is true.</td>
</tr>
<tr>
<td>Fair Queue (boolean)</td>
<td>Set to true, if you wish that calls to getConnection should be treated fairly in a true FIFO fashion. This uses the org.apache.tomcat.jdbc.pool.FairBlockingQueue implementation for the list of the idle connections. The default value is true. This flag is required when you want to use asynchronous connection retrieval. Setting this flag ensures that threads receive connections in the order they arrive. During performance tests, there is a very large difference in how locks and lock waiting is implemented. When fairQueue=true, there is a decision making process based on what operating system the system is running. If the system is running on Linux (property os.name=Linux), then to disable this Linux specific behavior and still use the fair queue, simply add the property org.apache.tomcat.jdbc.pool.FairBlockingQueue.ignoreOS=true to your system properties, before the connection pool classes are loaded.</td>
</tr>
<tr>
<td>Abandon When Percentage Full (int)</td>
<td>Connections that have been abandoned (timed out) will not get closed and reported up, unless the number of connections in use are above the percentage defined by abandonWhenPercentageFull. The value should be between 0-100. The default value is zero, which implies that connections are eligible for closure as soon as removeAbandonedTimeout has been reached.</td>
</tr>
<tr>
<td>Max Age (long)</td>
<td>Time in milliseconds to keep this connection. When a connection is returned to the pool, the pool will check to see if the current time when connected, is greater than the maxAge that has been reached. If so, it closes the connection rather than returning it to the pool. The default value is zero, which implies that connections will be left open and no age check will be done upon returning the connection to the pool.</td>
</tr>
<tr>
<td>Use Equals (boolean)</td>
<td>Set to true, if you wish the ProxyConnection class to use String.equals, and set to false when you wish to use == when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.</td>
</tr>
<tr>
<td>Suspect Timeout (int)</td>
<td>Timeout value in seconds. Default value is zero. Similar to the removeAbandonedTimeout value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if logAbandoned is set to true. If this value is equal or less than zero, no suspect checking will be performed. Suspect checking only takes place if the timeout value is larger than zero, and the connection was not abandoned, or if abandon check is disabled. If a connection is suspected, a warning message gets logged and a JMX notification will be sent.</td>
</tr>
</tbody>
</table>
Alternate User Name Allowed (boolean)

By default, the `jdbc-pool` will ignore the `DataSource.getConnection(username,password)` call, and simply return a previously pooled connection under the globally configured properties `username` and `password`, for performance reasons.

The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the `DataSource.getConnection(username,password)` call, simply set the property `alternateUsernameAllowed`, to true. If you request a connection with the credentials `user1/password1`, and the connection was previously connected using different `user2/password2`, then the connection will be closed, and reopened with the requested credentials. This way, the pool size is still managed on a global level, and not on a per-schema level. The default value is false.

Configuring a Custom Datasource

When adding a datasource, if you select the custom datasource type, the following screen will appear:

![New Data Source Screen]

Following are descriptions of the custom datasource fields:

- **Data Source Type**: Custom
- **Custom Data Source Type**: Specify whether the data is in a table or accessed through a query as described below.
- **Name**: Enter a unique name for this datasource
- **Description**: Description of the datasource
- **Configuration**: XML configuration of the datasource

### Custom datasource type

When creating a custom datasource, specify whether the datasource type is DS_CUSTOM_TABULAR (the data is stored in tables), or DS_CUSTOM_QUERY (non-tabular data accessed through a query). More information about each type are explained below.

#### Custom tabular datasources

Tabular datasources are used for accessing tabular data, that is, the data is stored in rows in named tables that can be queried later. To implement tabular datasources, the interface `org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS` is used. For more information, see a sample implementation of a tabular custom datasource at `Carbon datasources also support tabular data with the `InMemoryDataSource` implementation. If you have Data Services Server installed, for more information see the `<PRODUCT_HOME>/repository/conf/datasources/custom-datasources.xml` file, which is a sample Carbon datasource configuration.

#### Custom query datasources

Custom query-based datasources are used for accessing non-tabular data through a query expression. To implement query-based datasources, the `org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryBasedDS` interface is used. You can create any non-tabular datasource using the query-based approach. Even if the target datasource does not have a query expression format, you can create and use your own. For example, you can support any NoSQL type datasource using this type of a datasource.

For more information, see a sample implementation of a custom query-based datasource at `EchoDataSource`, and a sample data service descriptor with custom query datasources in `InMemoryDSSample`. Carbon datasources also support query-based data with the `org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryDataSourceReader` datasource reader implementation. If you have Data Services Server installed, for more information see the `<PRODUCT_HOME>/repository/conf/datasources/custom-datasources.xml` file, which is a sample Carbon datasource configuration.
In the `init` methods of all custom datasources, user-supplied properties will be parsed to initialize the datasource accordingly. Also, a property named `<__DATASOURCE_ID__>`, which contains a UUID to uniquely identify the current datasource, will be passed. This can be used by custom datasource authors to identify the datasources accordingly, such as datasource instances communicating within a server cluster for data synchronization.

Shown below is an example configuration of a custom datasource of type `<DS_CUSTOM_TABULAR>`:

![Example configuration of a custom datasource](image)

After creating datasources, they will appear on the Data Sources page. You can edit and delete them as needed by clicking Edit or Delete links.

### Changing the Hostname

By default, WSO2 products identify the hostname of the current machine through the Java API. However, this value sometimes yields erroneous results on some environments. Therefore, users are recommended to configure the hostname. The following procedure explains how to change the hostname and management hostname of WSO2 API Manager (WSO2 API-M) as required for your production environment.

1. Open the `<API-M_HOME>/repository/conf/carbon.xml` file and set the `HostName` and `MgtHostName` property as shown below.

   ```xml
   <HostName>{hostname}</HostName>
   <MgtHostName>{management-hostname}</MgtHostName>
   ````

   *(hostname)* - Hostname or IP address of the machine hosting this server. This is will become part of the End Point Reference of the services deployed on this server instance.

   *(management-hostname)* - Hostname to be used for the WSO2 API-M Management console.

2. Generate a key store, export the public certificate from the keystore, and import that certificate to the `client-truststore.jks` file. For more information, see Creating New Keystores in the WSO2 Administration guide.

### Whats Next?

After changing your hostname and management hostname, make sure to whitelist your hostname for the API Store.

#### Whitelisting Hostnames for API Store

After changing your hostname and management hostname, make sure to whitelist your hostname for the API Store.
API Manager provides the capability to whitelist multiple host names if you use different host names to access API Store in your environment.

In this case, **localhost** is by default considered as a whitelisted host name.

Similarly you can whitelist multiple host names for store as follows.

- You need to add the host names to the `whiteListedHostNames` attribute in `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` as comma separated values.

```json
"whiteListedHostNames": ["www.wso2.org", "www.example.com"]
```

**Note:**
When you try to access API Store with a host which is not whitelisted, or is not specified in `<API-M_HOME>/repository/conf.carbon.xml`, you will notice the following warning being logged in the server logs.

```
Possible HOST Header Attack is identified. Hence, rewriting to default host in configuration.
```

---

**Working with the API Microgateway**

The API Microgateway is a specialized form of the WSO2 API Gateway. Its main characteristics are:

- The ability to execute in isolation without mandatory connections to other components (Key Manager, Traffic Manager, etc).
- The ability to host only a subset of specific APIs (defined in the API Publisher), instead of all.
- Immutability; if you update an API, you need to re-create the container/instance and hot deployment is not possible.

Microgateway offers you a proxy that is capable of performing security validations (OAuth, Signed JWT), in-memory (local) rate limiting and operational analytics.

The following topics explain further details about the API Microgateway, its architecture and capabilities, and how to configure, deploy and troubleshoot it:

- Overview of the Microgateway
- Microgateway Quick Start
- Grouping APIs with Labels
- Overriding Endpoint Information
- Importing Certificates to the API Microgateway Truststore
- Analytics for Microgateway
- Deploying the API Microgateway in Docker
- Deploying the API Microgateway in Kubernetes
- Working with the Microgateway in a Windows Environment
- Troubleshooting the Microgateway

**Overview of the Microgateway**

- **Introduction**
- **Capabilities of the Microgateway**
- **When to use the Microgateway**
- **When to use the API Gateway**
- **How the Microgateway works**

**Introduction**

The WSO2 API Microgateway is a lightweight message processor for APIs. The Microgateway is used for message security, transport security, routing, and other common API Management related quality of services. It can process incoming and outgoing messages while collecting information required for usage metering and throttling capabilities. The Microgateway natively supports scaling in highly decentralized environments including microservice architecture. An immutable, ephemeral Microgateway fits well with the microservice architecture. The Microgateway is also capable of operating in lockdown environments such as IoT devices, since connectivity from the Microgateway to the API Management system is not mandatory.

Setting up Microgateways close to your backend servers and on-premise deployments not only reduces network latency, but also adds additional security as you can run it in a fully controlled environment with a selected set of APIs. You can run the Microgateway in a regional data center, in the same server...
instance as your service implementation, or even in your local workstation.

In summary, the Microgateway is a simplified lightweight version of the API Gateway. However, the Microgateway only supports a subset of the capabilities of the API Gateway.

**Capabilities of the Microgateway**

- **Fits the microservice architecture** - Since the Microgateway has only a small startup delay and consumes less resources, it natively supports container-based deployments. When there is a requirement to scale the Microgateway, it can be done easily because of its design. Immutable, ephemeral gateways help to build cloud-native API Management solutions and simple and infinitely repeatable deployments that use continuous integration/delivery would be much easier with the Microgateway. Due to these capabilities, server and node failures do not result in your service going down.

- **Accelerates rapid development** - Unlike in the typical complete API Gateway, users can easily run a set of selected APIs in a controlled environment within seconds in the Microgateway. For example, if you need to do a quick debug on an API invocation flow, you can simply setup the Microgateway in a local workstation without doing a full-blown deployment.

- **Low resource consumption and efficiency** - The Microgateway is capable of running on very low memory and CPU consumption. Its lightweight design allows the Microgateway to load just the minimum required components to process API calls. The following are the recommended system requirements.

<table>
<thead>
<tr>
<th>CPU</th>
<th>2 Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>256MB</td>
</tr>
</tbody>
</table>

- **Reduces network latency** - You can deploy the Microgateway close to backend services depending on the requirement. If necessary, the Microgateway can be deployed in regional datacenters or within the same subnet where your services reside. Therefore, you do not need to route your API calls through the internet or different network segments.

- **Dev-ops friendly** - No programming is needed to set up and manage the Microgateway and it can be done only using configurations or environment variables that can override configurations. If needed, you can easily integrate the Microgateway management with a container management system and automate the deployment process.

- **Enhanced security** - If your API consumer and producer both reside within the same network or in close proximity, there is no point in routing that traffic through a Cloud Gateway. You can simply route that traffic through a Microgateway residing within the same network. This helps to reduce network latency. Routing traffic within the same network boundaries also helps to address security concerns and compliance with certain regulations.

**When to use the Microgateway**

- To run a Gateway in a lockdown environment or offline mode with no connection to the API Management system.
- To run a set of selected APIs in a controlled environment such as in private jet gateway mode, particularly when you see unusual traffic patterns on an API and need to scale it by itself.
- To scale only a set of selected APIs according to traffic patterns. An independently scalable set of APIs can be run using the Microgateway.
- In internal or on-premise API Gateway deployments where consumers (delivery channels) and service instances reside within the same network. In such cases, having the Microgateway within close proximity to a backend server reduces latency.
- When there is a requirement to run a Gateway in sidecar mode along with an application server within the same runtime (or same pod in Kubernetes).

**When to use the API Gateway**

- When there is requirement to throttle API calls based on counters across all Gateway nodes.
- To run an API Gateway as a centralized Gateway that can handle requests for many different APIs and different backend servers.
- When using traditional SOAP architecture requiring the Gateway to perform mediations and orchestrations.

**How the Microgateway works**
API Manager comes with a Microgateway that is responsible for Microgateway related management tasks. This command line tool accepts issued commands and you are able to communicate with the API Management core runtime (API Publisher) and build the Microgateway runtime. The API Microgateway communicates with the API Publisher to retrieve the API metadata required to create the Microgateway runtime.

Upon API Gateway CLI setup, users provide the label name (logical group of APIs), the API Management core base URL, the username and password. With this information, the CLI communicates with APIs exposed from the API core and creates a Gateway runtime project for the given label where all APIs are included under a given label. All throttling policies required for runtime are also created.

When you issue a build command, the project build is triggered and a single distribution is created with API definitions. You can directly run this on the Cloud, a premise deployment, or a local workstation, as required.

For details on how analytics works with Microgateway and how to configure it, see Analytics for Microgateway.

**What’s Next?**

Microgateway Quick Start

What is the API Microgateway?

The API Manager Microgateway is a specialized form of the WSO2 API Gateway. Its main characteristics are:

- The ability to execute in isolation without mandatory connections to other components (Key Manager, Traffic Manager, etc).
- The ability to host only a subset of specific APIs (defined in the API Publisher), instead of all.
- Immutability; if you update an API, you need to re-create the container/instance and hot deployment is not possible.

Microgateway offers you a proxy that is capable of performing security validations (OAuth, Signed JWT), in-memory (local) rate limiting and operational analytics.

Before you begin,

1. Download the Microgateway distribution and extract it.
2. Append the full path of the /bin folder of the extracted Microgateway distribution to the PATH environment variable.
Let's go through the main use case of the Microgateway:

- Deploy and subscribe to the sample API
- Generate a distribution for a single API
- Generate a JWT token and invoke the API

**Deploy and subscribe to the sample API**

1. Open the API Publisher (https://<hostname>:9443/publisher) and sign in with admin/admin credentials.
2. Close the interactive tutorial that starts automatically if you are a first-time user.
3. Click the **Deploy Sample API** button. It deploys a sample API called PizzaShackAPI into the API Manager.

The **Deploy Sample API** option is available only when there are no APIs in API Publisher. If you have already created a API, this option is not available.

4. Click PizzaShackAPI to open it.
5. Go to the **Lifecycle** tab and note that the **State** is **PUBLISHED**. The API is already published to the API Store.
6. Sign in to the API Store (https://<hostname>:9443/store) with the admin/admin credentials and click the **PizzaShackAPI** API.
7. Select the default application and an available tier, and click **Subscribe**.
8. When the subscription is successful, click **View Subscriptions** on the information message that appears.

You have now successfully subscribed to an API. Let's generate a distribution for this API.

**Generate a distribution for a single API**

It is possible to create a Microgateway distribution for a single API that is in the **Published** state.

For details on how to create a Microgateway distribution for a group of APIs, see **Grouping APIs with Labels**.

1. Navigate to a preferred workspace folder using the command line.
   This location is used to run the Microgateway commands and to generate Microgateway artifacts.
2. Set up a project using the command given below,

   ```
   micro-gw setup <project_name> -a <API_name> -v <version>
   ```

   When the tool requests for the username and password, use **admin** in both instances. Use the default values for the other parameters by pressing Enter.

   Here is an example:

   ```
   $ micro-gw setup pizzashack-project -a PizzaShackAPI -v 1.0.0
   Enter Username:
   admin
   Enter Password for admin:
   
   Enter API Manager REST API [https://localhost:9443]:
   
   Enter Trust store location: [lib/platform/bre/security/ballerinaTruststore.p12]
   
   Enter Trust store password: [ use default? ]
   
   Setting up project pizzashack-project is successful.
   ```

   When the above command is issued, the tool connects with API Manager REST APIs and retrieves the API specified above. The source artifacts are generated in the current folder location.

   The folder structure looks similar to the following,
2. Build the microgateway distribution for the project using the following command:

    micro-gw build <project_name>

Here is an example:

    $ micro-gw build pizzashack-project

    Build successful for the project - pizzashack-project

Once the above command is executed, the generated source files are built and a Microgateway distribution is created in the target folder.

The name of the distribution will have the format `micro-gw-<project_name>.zip`. In the above example, the name will be `micro-gw-pizzashack-project.zip`.

4. Next, unzip the `micro-gw-pizzashack-project.zip` and run the gateway shell script inside the bin folder of the extracted zip using the following command:
The Microgateway will now start for the Pizzashack API. For example,

```bash
micro-gw-pizzashack-project/bin$ bash gateway

ballerina: HTTP access log enabled
ballerina: initiating service(s) in
'/home/user/workspace/pizzashack-project/target/micro-gw-pizzashack-project/exec/pizzashack-project.balx'
ballerina: started HTTPS/WSS endpoint 0.0.0.0:9096
ballerina: started HTTPS/WSS endpoint 0.0.0.0:9095
ballerina: started HTTP/WS endpoint 0.0.0.0:9090
```

You have successfully generated and started a Microgateway distribution for the Pizzashack API.

**Generate a JWT token and invoke the API**

Once you start the Microgateway, you can use a JWT or an OAuth2 token to invoke the API. For the invocation URL, you can use either `https://localhost:9095/` (HTTPS) or `http://localhost:9090/` (HTTP).

A signed JWT token is a self-contained access token that can be validated by the Microgateway itself without connecting to any external party. In contrast, a typical OAuth2 token needs to be validated by connecting to the Key Manager. This is one of the key advantages of the Microgateway, if you need to run it in offline mode.

1. Log in to the API Store, click **Applications** and choose to edit the Default Application. Set the token type to JWT.

![Edit Application](image)
WSO2 API Manager, WSO2 Inc.

2. Click the Production Keys tab and click Generate Keys to generate an access token to invoke the API.
If you have already generated keys before, click Regenerate.

3. Invoke the API using the JWT token using the following command:

Format
curl -k -i -H "Authorization: Bearer <JWT_token>" <API_url>

For example,

Example
curl -k -i -H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlVCX0JReTJIRlYzRU1UZ3E2NFEtMVZpdFliRSJ9.eyJhdWQi
OiJodHRwOlwvXC9vcmcud3NvMi5hcGltZ3RcL2dhdGV3YXkiLCJzdWIiOiJhZG1pbiIsImFwcGxpY2F0aW9uIjp7ImlkI
joxLCJuYW1lIjoiRGVmYXVsdEFwcGxpY2F0aW9uIiwidGllciI6IlVubGltaXRlZCJ9LCJzY29wZSI6ImFtX2FwcGxpY2
F0aW9uX3Njb3BlIGRlZmF1bHQiLCJpc3MiOiJodHRwczpcL1wvbG9jYWxob3N0OjgyNDNcL3Rva2VuIiwia2V5dHlwZSI
6IlBST0RVQ1RJT04iLCJzdWJzY3JpYmVkQVBJcyI6W3sibmFtZSI6IlBpenphU2hhY2tBUEkiLCJjb250ZXh0IjoiXC9w
aXp6YXNoYWNrXC8xLjAuMCIsInZlcnNpb24iOiIxLjAuMCIsInB1Ymxpc2hlciI6ImFkbWluIiwic3Vic2NyaXB0aW9uV
GllciI6IlVubGltaXRlZCIsInN1YnNjcmliZXJUZW5hbnREb21haW4iOiJjYXJib24uc3VwZXIifV0sImV4cCI6MTUyOD
gwMjg3ODExMCwiaWF0IjoxNTI4ODAyODgwMTEwLCJqdGkiOiJlZmQxZGVhZC1iNWI3LTQ0M2MtOTIyMC1iYzJjZTJjY2M
wYjEifQ==.b8P2uPRkVai0O7PcbvbjANLuHlJQzX1eHplweDpE6ItbEHRTkN2U_h6b39tz14dKUmigzASinj5LheUWGB7
gEDRqlc39ckhRX2qpolQpITZvpzYo8ky9AcxlJXLxrfPwgdht36zfIQwlPN_s2A5nY7c9pDBMu0OOOlYpmK81SrtipFST
AyPiRg5VyY3n-4POnjkEF-LQKCCTq7ef0uLOFTcoCT-gqNsXeKqt15suCYj5QMHJ8VP5bKsKZy9-1o9oFNlwc1QE0qE01
fPuGuz-4J22OvkHyrasbjhhGaaDgdpdERl9ElUDuL0C9AdX6Fb1sz54gnAiU3RUBK3RQUDK7Q=="
https://localhost:9095/pizzashack/1.0.0/menu

You receive a response similar to the following:

HTTP/1.1 200 OK
Date: Thu, 26 Jul 2018 16:52:38 GMT
Content-Type: application/json
Transfer-Encoding: chunked
server: WSO2 Carbon Server
[{"name":"BBQ Chicken Bacon","description":"Grilled white chicken, hickory-smoked bacon and
fresh sliced onions in barbeque
sauce","price":"13.99","icon":"/images/6.png"},{"name":"Chicken
Parmesan","description":"Grilled chicken, fresh tomatoes, feta and mozzarella
cheese","price":"9.99","icon":"/images/1.png"},{"name":"Chilly Chicken Cordon
Bleu","description":"Spinash Alfredo sauce topped with grilled chicken, ham, onions and
mozzarella","price":"17.99","icon":"/images/10.png"},{"name":"Double Bacon
6Cheese","description":"Hickory-smoked bacon, Julienne cut Canadian bacon, Parmesan,
mozzarella, Romano, Asiago and and Fontina
cheese","price":"16.99","icon":"/images/9.png"},{"name":"Garden Fresh","description":"Slices
onions and green peppers, gourmet mushrooms, black olives and ripe Roma
tomatoes","price":"9.99","icon":"/images/3.png"},{"name":"Grilled Chicken
Club","description":"Grilled white chicken, hickory-smoked bacon and fresh sliced onions
topped with Roma tomatoes","price":"13.99","icon":"/images/8.png"},{"name":"Hawaiian BBQ
Chicken","description":"Grilled white chicken, hickory-smoked bacon, barbeque sauce topped
with sweet pine-apple","price":"27.99","icon":"/images/7.png"},{"name":"Spicy
Italian","description":"Pepperoni and a double portion of spicy Italian
sausage","price":"13.99","icon":"/images/2.png"},{"name":"Spinach
Alfredo","description":"Rich and creamy blend of spinach and garlic Parmesan with Alfredo
sauce","price":"27.99","icon":"/images/5.png"},{"name":"Tuscan Six Cheese","description":"Six
cheese blend of mozzarella, Parmesan, Romano, Asiago and
Fontina","price":"11.99","icon":"/images/4.png"}]

Alternatively, you can invoke the API using an OAuth2 token.

Copyright © WSO2 Inc. 2015-2018

622


API Microgateway Walkthrough

This section provides you with a series of screencasts that give you a quick tour on how to work with the Microgateway. For more information on the Microgateway capabilities and usage, see Overview of the Microgateway.

- **Screencast 1 - Setting up a Microgateway project**
- **Screencast 2 - Build and run the Microgateway**
- **Screencast 3 - Running a Microgateway in different environments**

**Screencast 1 - Setting up a Microgateway project**

This screencast walks you through the steps of setting up a Microgateway project using the built-in CLI tool that comes with the WSO2 API Manager Microgateway.

For more information on how to get started, see our Microgateway Quick Start.

**Screencast 2 - Build and run the Microgateway**

This screencast demonstrates how to build and run the Microgateway. Note that the project that you created in the previous screencast is used for this screencast.

**Screencast 3 - Running a Microgateway in different environments**

This is the 3rd and the final screencast of this series. This screencast shows you how you can run a Microgateway in Docker and Kubernetes environments.
Grouping APIs with Labels

It is possible to create a Microgateway distribution for a group of APIs. In order to group APIs, a label needs to be created and attached to the APIs that need to be in a single group.

- Creating and attaching a Microgateway to an API
- Generating a Microgateway distribution for a label
- Viewing Microgateway labels in the API Store

Creating and attaching a Microgateway to an API

1. Log in to the Admin portal (https://<hostname>:9443/admin). Use admin as the username and password.
2. To add a new Microgateway label, click LABELS under MICROGATEWAY, and then click ADD MICROGATEWAY.

3. Create a new label (e.g. MARKETING_STORE), add a host (e.g. https://localhost:9095) and click Save.
3. Navigate to the API Publisher (https://<hostname>:9443/publisher). Sign in using admin as the username and password.

4. Deploy the sample Pizzashack API by clicking Deploy Sample API if you have not already done.

5. Choose to edit the created Pizzashack API.
7. Navigate to the Manage section and click Gateway Environments.
8. Select the newly created label to attach it to the Pizzashack API.

9. Click Save & Publish.
10. Similarly, you can select MARKETING_STORE label for few other Published APIs.

Generating a Microgateway distribution for a label

1. Run the initial setup command for the MARKETING_STORE label after navigating to a preferred workspace folder.

   ```bash
   micro-gw setup <project name> -l <label name>
   ```

   Here is a sample:

   ```bash
   micro-gw setup marketing_project -l MARKETING_STORE
   ```

2. Build the microgateway distribution for the project using the following command:
Once the above command is executed, the generated source files are built and a Microgateway distribution is created under the target folder.

3. Next, unzip the micro-gw-marketing-project.zip and run the gateway shell script inside the bin folder of the extracted zip using the following command:

```
bash gateway
```

The Microgateway starts for the APIs grouped with MARKETING_STORE label you just created.

```
micro-gw-marketing-project/bin$ bash gateway

ballerina: HTTP access log enabled
ballerina: initiating service(s) in
'/home/user/workspace/marketing-project/target/micro-gw-marketing-project/exec/marketing-project.balx'
2018-06-29 21:16:44,142 INFO  [wso2/gateway:0.0.0] - Subscribing writing method to event stream
ballerina: started HTTPS/WSS endpoint 0.0.0.0:9096
ballerina: started HTTPS/WSS endpoint 0.0.0.0:9095
ballerina: started HTTP/WS endpoint 0.0.0.0:9090
```

Viewing Microgateway labels in the API Store

2. Any attached Microgateways are shown in the Overview tab of the API.
Overriding Endpoint Information

The backend URLs and credentials for endpoints can be provided for each Gateway node as system properties. Following are the supported functionalities:

- Override endpoints
- Define backend basic auth credentials

Override endpoints

To override the production endpoint of an API (e.g., TestAPI v1), provide the following system variable at server startup,

```
bash gateway -e TestAPI.v1.prod.endpoint.0="http://wso2.com"
```

For the sandbox endpoint, provide the following system variable at server startup,

```
bash gateway -e TestAPI.v1.sand.endpoint.0="http://wso2.com"
```
If there is more than one endpoint (e.g. load balance endpoint), use the following method to define each endpoint:

```bash
gateway -e TestAPI.v1.prod.endpoint.0="http://wso2.com" -e TestAPI.v1.prod.endpoint.1="http://support.wso2.com"
```

To set the endpoint using system properties, use the following method (replace . with _ in a Linux environment):

```bash
export TestAPI_v1_prod_endpoint_0="http://wso2.com"
```

Define backend basic auth credentials

If the backend endpoint is protected with basic auth, provide credentials related to that endpoint. Use the following system variables to define them:

```bash
gateway -e TestAPI.v1.prod.basic.username="admin" -e TestAPI.v1.basic.password="admin"
```

If a username is not defined, the username defined in the API (during API implementation) is used. To set the credentials using system properties, use the following method (replace . with _ in a Linux environment):

```bash
export TestAPI_v1_prod_basic_username="admin"
export TestAPI_v1_prod_basic_password="admin"
```

**Importing Certificates to the API Microgateway Truststore**

For signature validation of JAVTs, you need to add the public certificate of the Identity Provider to the truststore of the API Microgateway. Follow the steps given below to import the certificate.

1. Convert the public certificate to a PEM format. For example,

   ```bash
   openssl x509 -inform der -in public_certificate.cert -out certificate.pem
   ```

2. Import the certificate to the truststore. The `ballerinaTruststore.p12` resides in the generated distribution of the API Microgateway at <MICROGW_HOME>/runtime/bre/security/ballerinaTruststore.p12. Use the keytool that comes in JDK 8u60 or later.

   ```bash
   keytool -import -keystore <MICROGW_HOME>/runtime/bre/security/ballerinaTruststore.p12 -alias wso2carbonjwt -file certificate.pem
   ```

   Use the keytool that comes in JDK 8u60 or later.

3. Update the `certificateAlias` configuration in the `micro-gw.conf` file residing in the `<MICROGW_HOME>/conf` directory. The `certificateAlias` value is `wso2carbonjwt`, which is also used in step 2.

   ```bash
   [jwtTokenConfig]
   issuer="https://localhost:8243/token"
   audience="http://org.wso2.apimgt/gateway"
   certificateAlias="wso2carbonjwt"
   trustStore.path="$[ballerina.home]/bre/security/ballerinaTruststore.p12"
   trustStore.password="ballerina"
   ```

**Analytics for Microgateway**

The following sections explain how to configure the WSO2 API-M Analytics server to work with the Microgateway.
Architecture

The current architecture depicting the connection between the API Microgateway and the Analytics server is shown below.

![Architecture Diagram]

How Analytics works with the Microgateway

The Microgateway receives the requests from the API. To process the requests, there are two main tasks that run in the Microgateway.

When an API is invoked through the Microgateway, the subsequent events related to the invocations are populated and written in a file. To avoid the files attaining a large size, the following tasks are executed at particular time intervals.

- **File rotating task**
  - This task creates large files by rotating them at particular time intervals. The size of the files depends on the TPS (Transactions Per Second) of the environment in which the microgateway is running. Once the file is rotated, it is compressed into a zipped file. The `rotatingPeriod` can be configured as shown in the following section.

- **File uploading task**
  - This task uploads the zipped file created by the previous task, to a microservice running on the Analytics Server. This zip file is then persisted in the `WSO2AM_STATS_DB` database in the Analytics server node. A task running in the Analytics server processes the persisted data and sends it to the event stream. The time interval in which the zipped files are uploaded can be configured by setting the `uploadingTimeSpanInMillis` parameter.

After successfully completing the upload and persistence is achieved, the corresponding files are deleted from the Microgateway filesystem.

Configuring Analytics for the Microgateway

The following sections describe how to configure the WSO2 API-M Analytics Server for Microgateway.

Step 1 - Configuring the API-M Analytics Server

Before you begin...

Configure API Manager Analytics using the **Standard Setup**. For instructions, see Configuring APIM Analytics.

1. Create the `AM_USAGE_UPLOADED_FILES` table in the `WSO2AM_STATS_DB` database. A sample MySQL script to create the table is given below.
CREATE TABLE IF NOT EXISTS AM_USAGE_UPLOADED_FILES (  
FILE_NAME varchar(255) NOT NULL,  
FILE_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
FILE_PROCESSED tinyint(1) DEFAULT 0,  
FILE_CONTENT MEDIUMBLOB DEFAULT NULL,  
PRIMARY KEY (FILE_NAME, FILE_TIMESTAMP)  
);

Make sure that you correctly configured the WSO2AM_STATS_DB between WSO2 API Manager and WSO2 API-M Analytics.

Java system properties
The following system properties can be passed to the server if the default values need to be changed.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dusage.publishing.frequency=20000</td>
<td>300000 (5 minutes)</td>
<td>The frequency in which the database is read by the event source. This parameter is defined in milliseconds.</td>
</tr>
<tr>
<td>-Dusage.cleanup.frequency=36000</td>
<td>1800000 (30 minutes)</td>
<td>The frequency in which the cleanup task is run, to remove the processed files. This parameter is defined in milliseconds.</td>
</tr>
<tr>
<td>-Dfile.retention.days=5</td>
<td>N/A</td>
<td>The allowed duration for files to remain in the database. This parameter is defined in days</td>
</tr>
<tr>
<td>-Dusage.publishing.thread.count=3</td>
<td>3</td>
<td>The number of worker threads used to read data from the database. Each thread will read from a single file.</td>
</tr>
</tbody>
</table>

Example
-Dusage.cleanup.frequency=48000  
-Dusage.publishing.frequency=20000  
-Dfile.retention.days=3

Step 2 - Configuring the Microgateway for Analytics
To do the configurations for Microgateway analytics, open the <MICROGW_HOME>/conf/micro-gw.conf file. The sample below shows the analytics-related configurations.

enable=false
uploadingTimeSpanInMillis=600000
uploadingEndpoint="https://localhost:9444/micro-gateway-analytics/v0.10/usage/upload-file"
rotatingPeriod=60000
task.uploadFiles=true
username="admin"
password="admin"

The configurations are described in the table below.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Set this to true to enable Microgateway analytics. When enabled, the Microgateway starts persisting files with the analytics data in the &lt;MICROGW_HOME&gt;/api-usage-data folder.</td>
</tr>
</tbody>
</table>
uploadingTimeSpanInMillis | The time interval in which the uploading task runs.
uploadingEndpoint | The endpoint URL of the web application, to which the file has to be uploaded. This web app is deployed in the Analytics server, to retrieve files containing analytics data.
rotatingPeriod | The time interval, after which the file is rotated and compressed. This depends on the TPS (Transactions Per Second) capacity of the environment.

**Best Practice**
To avoid creating large files, we recommend setting a low `rotatingPeriod` if your environment has a higher TPS.

task.uploadFiles | Set this to true to enable the file upload task.

If this property is disabled, the analytics files are not uploaded to the analytics server, although the files are persisted in the Microgateway system.

**Best Practice**
In a distributed setup, the analytics data is uploaded to a shared location from multiple Microgateway nodes. To avoid multiple nodes competing to upload the same file, we recommend enabling the `task.uploadFiles` property in **only one node**. Make sure that you disable this in the rest of the Microgateway nodes.

You can also opt to have your own design as a workaround.

username | Username used in the Analytics server.
password | Password for the corresponding username.

**Deploying the API Microgateway in Docker**

**Before you begin, make sure you have downloaded the following:**

- WSO2 API Manager
- WSO2 Microgateway ToolKit

1. Start the API Manager and log in to the API Publisher (https://<hostname>:9443/publisher) using admin as the username and password.
2. Create and publish an API (e.g. Petstore).
3. Log in to the API Store and create an application that supports JWT tokens.
4. Subscribe to the Petstore API and generate a JWT token to invoke the API.
5. Create a `deployment.toml` file containing the relevant deployment configurations such as docker image name, registry, tag, etc. as shown below.
5. [docker]
   [docker.dockerConfig]
   enable = true
   name = "petstore"
   registry = 'docker.wso2.com'
   tag = 'v1'
   #buildImage = '
   #dockerHost = '
   #dockerCertPath = '
   #baseImage = '
   #enableDebug = '
   #debugPort = '
   #push = '
   #username = '
   #password = ''
   [docker.dockerCopyFiles]
   enable = true
   [{docker.dockerCopyFiles.files}]
   source = '<MICROGW_TOOLKIT_HOME>/resources/conf/micro-gw.conf'
   target = '/home/ballerina/conf/micro-gw.conf'
   isBallerinaConf = true

   Note that the docker image name must be in lower case.

6. Copy the micro-gw.conf file to the docker image as it contains the key manager configurations, JWT configurations, etc. This can be done by enabling the docker copy files configuration as shown above.

   Let's create a project called petstore_project and provide the deployment.toml file as an input.

   7. Navigate to the <MICROGW_TOOLKIT_HOME>/bin directory and run the following command,

   ```bash
   ./micro-gw setup <project_name> -a <API_name> --version <version> --deployment-config deployment.toml
   ```

   For example,

   ```bash
   ./micro-gw setup petstore_project -a petstore -v 1.0.0 --deployment-config deployment.toml
   ```

   This command creates the following folders under the petstore_project folder.

   petstore_project
   conf
deployment-config.toml
src
extension_filter.bal
petstore.bal
listeners.bal
policies
    application_10PerMin.bal
    application_20PerMin.bal
    application_50PerMin.bal
    subscription_Bronze.bal
    subscription_Gold.bal
    subscription_Silver.bal
subscription_Unauthenticated.bal
throttle_policy_initializer.bal
target
temp
hashes.json
test.toml

8. Build the project using the following command,

```
./micro-gw build <project_name>
```

The docker image is created in your local registry and can be used to spawn an API Microgateway docker container.

9. Run the docker container using the following command,

```
docker run -d docker.wso2.com/<API_name>:<version>
```

If you are working in a Mac environment, start the docker container with the following command to bind the docker container ports to the localhost or the docker host machine:

```
docker run -d -p 9090:9090 -p 9095:9095 docker.wso2.com/<API_name>:<version>
```

If you are working in a Linux environment, you can also start the docker container using the host network driver for your container as follows:

```
docker run --network host -d docker.wso2.com/<api_name>:<version>
```

For more information on working with Docker in different environments, see the relevant Docker documentation: Docker for Mac, Docker for Windows.

10. Retrieve the docker container IP address as follows,

```
docker ps - This provides the container id
docker inspect <Container_ID> | grep "IPAddress"
```

11. Using a REST client or a cURL command, access the API using the following details:

If you are working in a Mac environment, the URL is https://<localhost>:9095/<API_name>/<version>/<resource>.

URL: https://<Container_IP>:9095/<API_name>/<version>/<resource>
Headers: Authorization : Bearer <JWT_TOKEN>
Method: GET

As JWT is a self-contained access token, the Microgateway does not need to connect to the Key Manager. However, if you are using an Oauth2 access token, point the Microgateway to the Key Manager using the Key Manager details in the micro-gw.conf configuration file of the Microgateway. You can provide the Key Manager serverUrl as shown below. The serverUrl has to be accessible from the Microgateway.
Deploying the API Microgateway in Kubernetes

1. Start the API Manager and log in to the API Publisher (https://<hostname>:9443/publisher) using admin as the username and password.

2. Create an API with the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>hello_world - v1</td>
</tr>
<tr>
<td>Context</td>
<td>/hello/v1</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Access Control</td>
<td>All</td>
</tr>
<tr>
<td>Visibility on Store</td>
<td>Public</td>
</tr>
<tr>
<td>Production URL</td>
<td><a href="http://bk.test.com">http://bk.test.com</a></td>
</tr>
<tr>
<td>Tier Availability</td>
<td>Gold, Unlimited</td>
</tr>
</tbody>
</table>

You can create an application that supports JWT tokens, subscribe to the API and get a JWT token to invoke the API.

3. Create a deployment.toml file enabling Kubernetes deployment, service and config map resources.

The config map is used to copy the micro-gw.conf file.
Let's create a project called `k8s_project` and provide the `deployment.toml` file as an input.

4. Navigate to the `wso2am-micro-gw-toolkit-2.5.0/bin` directory and run the following command,

   ```
   ./micro-gw setup k8s_project -a hello_world -v v1 --deployment-config deployment.toml
   ```

This command creates the following folders under the `k8s_project` folder.

```
k8s_project
  conf
    deployment-config.toml
  src
    extension_filter.bal
    hello_world_v1.bal
    listeners.bal
  policies
    application_10PerMin.bal
    application_20PerMin.bal
    application_50PerMin.bal
    subscription_Bronze.bal
    subscription_Gold.bal
```
subscription_Silver.bal
subscription_Unauthenticated.bal
throttle_policy_initializer.bal
target
temp
hashes.json

5. Build the project using the following command,

```
./micro-gw build k8s_project
```

This generates the following Kubernetes resources.

```
k8s_project
docker
   Dockerfile
k8s_project_config_map.yaml
k8s_project_deployment.yaml
k8s_project_svc.yaml
```

The docker image to be deployed in Kubernetes is created in your local registry. You can find the image `k8s_project:latest` when you execute the docker images command.

6. Deploy the docker image in a Kubernetes environment.

```
You can push the docker image to the docker hub or to a private docker registry. If you change the docker image name, you need to change the image name in the k8s_project_deployment.yaml file.
```

7. You can also SCP the image to the Kubernetes nodes as follows:
   a. Save the docker image to a `tar` file. For example,

```
docker save k8s_project:latest > image.tar
```

   b. SCP the image to the Kubernetes nodes. For example,

```
scp -i <identity file> image.tar username@K8s_NODE_IP:
```

c. Load the docker image in the Kubernetes nodes. This needs to be executed in the Kubernetes nodes. For example,

```
docker load < image.tar
```

8. Deploy the API Microgateway in Kubernetes by deploying the Kubernetes resources using the following command,

```
kubectl create -f k8s_project/target/kubernetes/
```

9. Access the API in HTTPS using the following details:

```
The NodePort service type has been used in Kubernetes. For that service type, you can access the API using any of the Kubernetes node IP addresses and node ports.

https://<Any_Kubernetes_Node_IP>:<NodePort>/hello/v1/check
```

You can use the `kubectl` get services command to list down the services that run on Kubernetes.
Node port - 32616
URL - https://<Any_Kubernetes_Node_IP>:32616/hello/v1/check
Headers - Authorization Bearer <JWT_TOKEN>
Method - GET

As JWT is a self-contained access token, the Microgateway does not need to connect to the Key Manager. However, if you are using an Oauth2 access token, point the Microgateway to the Key Manager using the Key Manager details in the micro-gw.conf configuration file of the Microgateway. If you are running Key Manager in Kubernetes, you can provide the Key Manager serverUrl as shown below. The serverUrl has to be accessible from the Microgateway pods.

```yaml
[keyManager]
serverUrl="https://localhost:9443"
username="admin"
password="admin"
tokenContext="oauth2"
timestampSkew=5000
```

Working with the Microgateway in a Windows Environment

You can work with the WSO2 API Manager Microgateway in a Microsoft Windows environment as follows:

- Running the Microgateway on a Windows operation system
- Run the Microgateway as a Windows service
- Testing the Microgateway
- Resetting the Microgateway CLI tool

**Before you begin**

Make sure to have the following prerequisites in place to initialize a Microgateway instance,

- Oracle Java 1.8
- WSO2 API Manager (WSO2 API-M) 2.5.0 or later.
  This is required to bootstrap the Microgateway.
- WSO2 Microgateway CLI tool
- NSSM (the Non-Sucking Service Manager)
  This is required if you need to run the Microgateway as a Windows service.

**Running the Microgateway on a Windows operation system**

Follow the steps below run a WSO2 API Manager Microgateway instance in a Windows environment.

- Step 1 - Set up JAVA_HOME
- Step 2 - Set up a Microgateway project
- Step 3 - Build the Microgateway
- Step 4 - Run the Microgateway

**Step 1 - Set up JAVA_HOME**

You need to set up JAVA_HOME in order to run the Microgateway CLI tool.

1. Install the Oracle Java 1.8 or higher version.
2. Set the JAVA_HOME environment variable as a user variable via the edit system properties UI.
Step 2 - Set up a Microgateway project

1. Start the WSO2 API Manager server.
2. Create a new API in the API Publisher and publish it.
   If you have not created any APIs so far, you can deploy the sample Pizzashack API.
3. Create a Microgateway label to assign it to the newly created API.
   For more information, see Generating a Microgateway distribution for a label.
   
   It is not mandatory to create a Microgateway label because you can deploy a Microgateway in either one of the following instances based on your requirement.
   
   - Deploy a Microgateway for an API in order to deploy it for a single API.
   - Deploy a Microgateway for a Microgateway label in order to deploy the Microgateway for a group of APIs.

4. Set up the Microgateway CLI tool.
   Add the Microgateway executable location to the path variable.
   a. Download the Microgateway (wso2am-micro-gw-toolkit-x.x.x.zip).
   b. Unzip the downloaded ZIP file to a location of your choice (e.g., C:\Program Files\wso2\micro gateway cli) .
   c. Copy the path to the Microgateway bin folder (path-to-wso2am-micro-gw-toolkit\bin).
   d. Click Edit the system preferences.
**e.** Navigate to the **Advanced** tab and click **Environment Variables**.

**f.** Navigate to the path system variable and click **Edit**.
g. Click **New**, add the path to the Microgateway `bin` folder that you copied previously and click **OK** three times to save the newly added system variable.
4. Initialize the Microgateway project.
   a. Open the Microsoft command line (cmd.exe).
   b. Change the location to where you want to set up the Microgateway project.

   ```
   cd <directory-location-to-set-up-microgateway-project>
   ```

c. Run the following command to bootstrap a Microgateway instance.
   In this guide, as we have assigned a label to the API, you only need to run the first command.
   
   • If you want to set up a Microgateway for a Microgateway label, use the following command.

   ```
   micro-gw setup <project-name> -l <label-name>
   ```

   • If you want to set up a Microgateway project for a single API, use the following command.

   ```
   micro-gw setup <project-name> -a <API-name> -v <API-version>
   ```

   This will create a project for that particular API name and version.

   The remainder of the process is the same for a label-based project and a private jet API project.

d. Enter your username and password when prompted.

   ```
   If you have run the Microgateway CLI tool before, you are not prompted to enter your username again unless you have reset the Microgateway CLI tool.
   ```

e. Press **Enter** for all the other questions when prompted so that it picks up the default values.

   ```
   If you have run the Microgateway CLI tool before, you are not prompted to enter the following details unless you have reset the Microgateway CLI tool.
   ```

Enter APIIM base URL [https://localhost:9443/]:
Enter Trust store location: [lib/platform/bre/security/ballerinaTruststore.p12]
Enter Trust store password: [ use default? ]

Setting up project my_sample_project is successful.

E:\micro_gateway_projects:

After this process is complete, a directory is created with the given project name. The structure of the project is similar to the following:
Step 3 - Build the Microgateway

1. Navigate to the Microgateway project root directory.
   If you have not changed the directory after step 5 when setting up the Microgateway project, this is the same directory where you ran the set up command.
2. Run the following command to build the project.
   Use the same project name that you defined when setting up the Microgateway project.

```
micro-gw build <project-name>
```

After running the build command, you can find the Microgateway distribution (micro-gw-<project-name>.zip) specifically designed for the given Microgateway label or the respective API version in the <project-name>	arget directory.

Step 4 - Run the Microgateway

1. Copy the <micro-gw-project-name>.zip file from the <project-name>	arget directory and extract the files to where you want to run the Microgateway.
2. Navigate to the <micro-gw-project-name>/bin directory.

```
cd <micro-gw-project-name>/bin
```

3. Run the Microgateway.
This runs the Microgateway instance in the foreground.

```
.\gateway.bat
```

**Run the Microgateway as a Windows service**

- Step 1 - Set up JAVA_HOME
- Step 2 - Set up a Microgateway project
- Step 3 - Build the Microgateway
- Step 4 - Install the Gateway script as a Windows service

**Step 1 - Set up JAVA_HOME**

You need to set up Java in order to run the Microgateway CLI tool.

1. Install Oracle Java 1.8 or later.
2. Set the JAVA_HOME environment variable as a server variable via the edit system properties UI.

**Step 2 - Set up a Microgateway project**

1. Start the WSO2 API Manager server.
2. Create a new API in the API Publisher and publish it.
   If you have not created any APIs so far, you can deploy the sample Pizzashack API.
3. Create a Microgateway label to assign it to the newly created API.
   For more information, see [Generating a Microgateway distribution for a label](#).

It is not mandatory to create a Microgateway label because you can deploy a Microgateway in either one of the following instances based on your requirement.

- Deploy a Microgateway for an API in order to deploy it for a single API.
- Deploy a Microgateway for a Microgateway label in order to deploy the Microgateway for a group of APIs.

4. Set up the Microgateway CLI tool.
   Add the Microgateway executable location to the path variable.
a. **Download the Microgateway** (wso2am-micro-gw-toolkit-x.x.x.zip).
b. Unzip the downloaded ZIP file to a location of your choice (e.g., `C:\Program Files\wso2\micro gateway cli\`).
c. Copy the path to the Microgateway bin folder (`<path-to-wso2am-micro-gw-toolkit>\bin`).
d. Click **Edit the system preferences**.

![Edit the system preferences](image1.png)

```
Navigate to the Advanced tab and click **Environment Variables**.
```

![Environment Variables](image2.png)

```
f. Navigate to the path system variable and click **Edit**.
```
g. Click **New**, add the path to the Microgateway `bin` folder that you copied previously and click **OK** three times to save the newly added system variable.
5. Initialize the Microgateway project.
   a. Open the Microsoft command line (cmd.exe).
   b. Change the location to where you want to set up the Microgateway project.
   
   ```
   cd <directory-location-to-set-up-microgateway-project>
   ```

c. Run the following command to bootstrap a Microgateway instance.
   In this guide, as we have assigned a label to the API, you only need to run the first command.

   - If you want to set up a Microgateway for a Microgateway label, use the following command.
     
     ```
     micro-gw setup <project-name> -l <label-name>
     ```

   - If you want to set up a Microgateway project for a single API, use the following command.
     
     ```
     micro-gw setup <project-name> -a <API-name> -v <API-version>
     ```

   This will create a project for that particular API name and version.

   The remainder of the process is the same for a label-based project and a private jet API project.

d. Enter your username and password when prompted.

   ```
   Enter username and password when prompted.
   Press Enter for all the other questions when prompted so that it picks up the default values.
   ```

   If you have run the Microgateway CLI tool before, you are not prompted to enter your username again unless you have reset the Microgateway CLI tool.

e. Enter your username and password when prompted.

   ```
   Enter your username and password when prompted.
   ```

   If you have run the Microgateway CLI tool before, you are not prompted to enter the following details unless you have reset the Microgateway CLI tool.

   Enter APIM base URL [https://localhost:9443/]:
   Enter Trust store location: [lib/platform/bre/security/ballerinaTruststore.p12]
   Enter Trust store password: [ use default? ]

   Setting up project my_sample_project is successful.
   E:\micro_gateway_projects:

   After this process is complete, a directory is created with the given project name. The structure of the project is similar to the following:
Step 3 - Build the Microgateway

1. Navigate to the Microgateway project root directory.
   If you have not changed the directory after step 5 when setting up the Microgateway project, this is the same directory where you ran the set up command.
2. Run the following command to build the project.
   Use the same project name that you defined when setting up the Microgateway project.

   ```bash
   micro-gw build <project-name>
   ```

   After running the build command, you can find the Microgateway distribution (micro-gw-<project-name>.zip) specifically designed for the given Microgateway label or the respective API version in the `<project-name>\target` directory.

Step 4 - Install the Gateway script as a Windows service

1. Download the NSSM (Non-Sucking Service Manager) tool and extract the ZIP file.
2. Copy the extracted ZIP file to the location where you want to run the Microgateway.
3. Install the `gateway.bat` script as a Windows service.
   a. Navigate to the `win64` folder, which is inside the new location of the extracted NSSM tool.

   ```bash
   cd apim_2_5_microgateway_bat\nssm-2.24\win64
   ```
b. Run the following command to install the `gateway.bat` script as a Windows service.

```
nssm.exe install microgw
```

c. Click Yes when the following User Account Control message appears.
You need to do this in order to allow the NSSM tool to make changes to your Microgateway distribution.

![User Account Control](image)

You need to do this in order to allow the NSSM tool to make changes to your Microgateway distribution.

Click [Yes] and select the location of the `gateway.bat` file, which is in the `bin` folder of your unzipped Microgateway distribution.

![NSSM service installer](image)

d. Click [Yes] and select the location of the `gateway.bat` file, which is in the `bin` folder of your unzipped Microgateway distribution.

e. Click Details, and enter the details of the Microgateway service.

![NSSM service installer](image)

f. Click Install service.
A confirmation message appears.
Testing the Microgateway

You can invoke the API through the Microgateway using the following cURL command in order to test the Microgateway. By default, the Microgateway serves the HTTPS/WSS request via the 9095 port.

```
```

```
curl -v -k https://mg.knnect.com:9095/wolaa/1.1.1/getmeall -H "Authorization: Bearer 104cd62f-a8ab-3089-bc12-f7cc36e73e77"
```

Resetting the Microgateway CLI tool

When you run the Microgateway CLI tool for the first time, it preserves all the information (i.e., your username, APIM base URL, TrustStore location and the TrustStore password) with the exception of your password, for later use. If you want to reset the Microgateway CLI tool in order to clear all the data that has been preserved by the Microgateway CLI tool, you can use the `--reset` argument as follows. Note that you can invoke the Microgateway CLI tool reset command from any directory in the system.

```
micro-gw reset
```

Troubleshooting the Microgateway

When errors/exceptions occur in the system, the API Microgateway throws error responses to the client by default. The following sections explain about the different ways of troubleshooting common problems that might occur while you use the Microgateway.

- Common exceptions and solutions
- Error responses
- Error codes

Common exceptions and solutions

The table below shows the common exceptions that might occur when you are trying to use the API Microgateway, and how to fix them. These exceptions can happen due to misconfigurations.

<table>
<thead>
<tr>
<th>Error log</th>
<th>Possible cause</th>
<th>Res</th>
</tr>
</thead>
<tbody>
<tr>
<td>micro-gw: Error occurred while trying to connect with server. Is the server running at <a href="https://localhost:9443">https://localhost:9443</a>?</td>
<td>The API Manager node (Publisher) is down when running the setup command in the Microgateway.</td>
<td>Verify and fix the setup command.</td>
</tr>
</tbody>
</table>
Error responses

A connection to the backend could not be established because the network is unavailable.
Verify

Micro-gw: ERROR [src:0.0.0] - Error in client response: {message: "Connection refused: localhost/127.0.0.1:8080", cause: null}
Connection to the backend is refused.
Check

error [docker plugin]: Unable to build docker image: {message: "Invalid reference format: repository name must be lowercase"}
The name of the docker images should be in lowercase.
Verify and Provide the

ERROR [wso2/gateway:0.0.0] - Error occurred while reading the key validation response: {message: "Connection refused: localhost/127.0.0.1:9443", cause: null}
ERROR [wso2/gateway:0.0.0] - Error occurred while converting the authorized value from the key validation response to a string value: {message: "null" cannot be cast to 'string', cause: null}
ERROR [wso2/gateway:0.0.0] - Error occurred while reading the key validation response: {message: "General SSLEngine problem/192.168.8.101:9443", cause: null}
ERROR [wso2/gateway:0.0.0] - Error occurred while converting the authorized value from the key validation response to a string value: {message: "null" cannot be cast to 'string', cause: null}
ERROR [ballerina/http] - Error while validating JWT token: {message: "Invalid signature", cause: null}
JWT signature verification has failed.
Verify

This is due to an issue in the Service template.

Copyright © WSO2 Inc. 2015-2018
<table>
<thead>
<tr>
<th>Error code</th>
<th>Error response</th>
<th>Possible reasons</th>
</tr>
</thead>
</table>
| 101503     | { "fault": {  
|            |    "code": "101503",  
|            |    "message": "Runtime Error",  
|            |    "description": "Error connecting to the back end"  
|            | } }             | • The network is not reachable by the backend.  
|            |                | • The connection from the backend has been refused. |
| 101504     | { "fault": {  
|            |    "code": "101504",  
|            |    "message": "Runtime Error",  
|            |    "description": "Connection timed out"  
|            | } }             | • The connection has timed out.  
|            |                | • The connection has timed out from the Microgateway end.  
|            |                | • Response is getting delayed and hence timeout occurs in the Microgateway side. |
| 900900     | { "fault": {  
|            |    "code": 900900,  
|            |    "message": "Unclassified Authentication Failure",  
|            |    "description": "Unclassified Authentication Failure"  
|            | } }             | • The Key Manager is down. |

**Error codes**

Given below are some API Manager-specific error codes and their meanings. For more details on error codes and their descriptions, see Error Handling.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Possible reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>900901</td>
<td>The production/sandbox key offered by the requested endpoint is not specified.</td>
</tr>
<tr>
<td>900900</td>
<td>API-M authentication related error.</td>
</tr>
<tr>
<td>900901</td>
<td>Invalid access token.</td>
</tr>
<tr>
<td>900902</td>
<td>Missing credentials.</td>
</tr>
<tr>
<td>900903</td>
<td>Access token expired.</td>
</tr>
<tr>
<td>900904</td>
<td>Access token inactive.</td>
</tr>
<tr>
<td>900905</td>
<td>Incorrect access token type provided.</td>
</tr>
<tr>
<td>900906</td>
<td>No matching resource found in the API for the given request.</td>
</tr>
<tr>
<td>900907</td>
<td>The requested API is temporarily blocked.</td>
</tr>
<tr>
<td>900908</td>
<td>Resource forbidden.</td>
</tr>
</tbody>
</table>
Extending the API Manager

This page summarizes the extension points supported by WSO2 API Manager.

[ Mediation extensions ] [ Security extensions ] [ Branding extensions ] [ Workflow extensions ] [ API life cycle extensions ]

Mediation extensions

- Writing Custom Handlers
- Adding Mediation Extensions
- Customized Error Handling
- Editing API Templates

Security extensions

- Writing Custom Grant Types
- Customizing the JWT Generation
- Configuring a Third-Party Key Manager
- Extending Key Validation
- Extending the Key Manager Interface
- Extending Scope Validation
- Engaging Multiple Throttling Policies to a Single API
- Sharing Applications Between Multiple Groups
- Securing OAuth Token with HMAC Validation

Branding extensions

- Adding a new or customizing an existing API Store Theme
- Customize the API Store and Gateway URLs for Tenants
- Customizing Login Pages for API Store and API Publisher
- Customizing User SignUp in API Store

Workflow extensions

- Adding an Application Creation Workflow
- Adding an Application Registration Workflow
- Adding an API Subscription Workflow
- Adding a User Signup Workflow
- Invoking the API Manager from the BPEL Engine
- Customizing a Workflow Extension
- Configuring Workflows for Tenants
- Configuring Workflows in a Cluster
- Changing the Default User Role in Workflows

API life cycle extensions

- Extending the API Life Cycle

Managing Workflow Extensions

The workflow feature enables you to add more control and constraints to the tasks executed within it. For example, you can add a constraint through a workflow where an approval from a manager is required for a user to sign-up to the API Store. You can engage a workflow if you require a third party interventions/approvals for actions which have a pre-defined path such as signing up, registering an application, subscribing to an API, etc.

Use workflow extensions to attach a workflow to the following API Store/API Publisher operations:
Adding an Application Creation Workflow

This section explains how to attach a custom workflow to the application creation operation in WSO2 API Manager (WSO2 API-M). First, see Workflow Extensions for information on different types of workflow executors.

Attaching a custom workflow to application creation allows you to control the creation of applications within the Store. An application is the entity that holds a set of subscribed API's that would be accessed by an authorization key specified for that particular application. Hence, controlling the creation of these applications would be a decision based on the organization's requirement. Some example use cases would be:

- Review the information of the application by a specific reviewer before the application is created.
- The application creation would be offered as a paid service.
- The application creation should be allowed only to users who are in a specific role.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Change the credentials of the workflow configurations in the following registry resource: _system/governance/apimgt/aplicationdata/workflow-extensions.xml.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if any exist.
- Unzip the `<API-M>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip` file, update the role as follows in the `ApplicationsApprovalTask.ht` file, and ZIP the `ApplicationsApprovalTask-1.0.0` folder.

**Format**

```xml
<htd:argument name="role">
  [new-role-name]
</htd:argument>
```

**Configuring the Business Process Server**

1. Download WSO2 Enterprise Integrator.
2. Set an offset of 2 to the default BPS port in the `<EI_HOME>/wso2/business-process/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

```xml
<Offset>2</Offset>
```

**Tip:** If you change the BPS port offset to a value other than 2 or run WSO2 API-M and WSO2 EI on different machines (therefore, want to set the hostname to a different value than localhost), you need to search and replace the value 9765 in all the files (.epr) inside the `<API-M_HOME>/business-processes` directory with the new port (i.e., the value of 9763 + `<port-offset>`).


```xml
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```
4. Copy the following from the `<API-M_HOME>/business-processes/epr` directory to the `<EI_HOME>/wso2/business-process/repository/conf/epr` directory.

- If the `<EI_HOME>/wso2/business-process/repository/conf/epr` directory does not exist, create it.
- Make sure to give the correct credentials in the `<EI_HOME>/wso2/business-process/repository/conf/epr` files.

- Update the `<EI_HOME>/business-processes/epr/ApplicationCallbackService.epr` file according to API Manager.

  ```xml
  <wsa:Address>https://localhost:8243/services/WorkflowCallbackService</wsa:Address>
  ```

- Update the `<EI_HOME>/business-processes/epr/ApplicationService.epr` file according to EI.

  ```xml
  <wsa:Address>http://localhost:9765/services/ApplicationService</wsa:Address>
  ```


If you are using Mac OS with High Sierra, you may encounter the following warning when logging in to the Management Console due to a compression issue that exists in the High Sierra SDK.

```text
WARN {org.owasp.csrfguard.log.JavaLogger} - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST, uri:/carbon/admin/login_action.jsp, error:required token is missing from the request)
```

To avoid this issue, open the `<EI_HOME>/wso2/business-process/repository/conf/tomcat/catalina-server.xml` file and change the compression="on" to compression="off" in the Connector configuration, and restart the EI.

6. Click Add under Processes and upload the `<API-M_HOME>/business-processes/application-creation/BPEL/ApplicationApprovalWorkflowProcess_1.0.0.zip` file to EI. This is the business process archive file.

7. Select Add under the Human Tasks menu and upload the `<API-M_HOME>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip` file to EI. This is the human task archived file.

Before you begin, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Change the credentials of the workflow configurations in the following registry resource: `_system/governance/apimgt/applicationdata/workflow-extensions.xml`.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if any exist.
- Unzip the `<API-M>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip` file, update the role as follows in the `ApplicationsApprovalTask.ht` file, and ZIP the `ApplicationsApprovalTask-1.0.0` folder.

```xml
<htd:argument name="role">
  [new-role-name]
</htd:argument>
```
Configuring the Business Process Server

1. **Download WSO2 Business Process Server.**

2. Set an offset of 2 to the default BPS port in the `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

   `<Offset>2</Offset>`

   **Tip:** If you change the BPS port offset to a value other than 2 or run WSO2 API-M and WSO2 BPS on different machines (therefore, want to set the hostname to a different value than localhost), you need to search and replace the value 9765 in all the files (.epr) inside the `<API-M_HOME>/business-processes` directory with the new port (i.e., the value of 9763 + `<port-offset>`).

3. Open the `<BPS_HOME>/repository/conf/humantask.xml` and `<BPS_HOME>/repository/conf/b4p-coordination-config.xml` file and set the `TaskCoordinationEnabled` property to true.

   `<TaskCoordinationEnabled>true</TaskCoordinationEnabled>`

4. Copy the following from the `<API-M_HOME>/business-processes/epr` directory to the `<BPS_HOME>/repository/conf/epr` directory. If the `<BPS_HOME>/repository/conf/epr` directory does not exist, create it.

   **Make sure to give the correct credentials in the `<BPS_HOME>/repository/conf/epr` files.**

   - Update the `<API-M_HOME>/business-processes/epr/ApplicationCallbackService.epr` file according to API Manager.

     `<wsa:Address>https://localhost:8243/services/WorkflowCallbackService</wsa:Address>`

   - Update the `<API-M_HOME>/business-processes/epr/ApplicationService.epr` file according to BPS.

     `<wsa:Address>http://localhost:9765/services/ApplicationService</wsa:Address>`


   If you are using Mac OS with High Sierra, you may encounter the following warning when logging in to the Management Console due to a compression issue that exists in the High Sierra SDK.

   ```java
   WARN (org.owasp.csrfguard.log.JavaLogger) - potential cross-site request forgery (CSRF) attack thwarted (user:anonymous), ip:xxx.xxx.xx.xx, method:POST, url:/carbon/admin/login_action.jsp, error:required token is missing from the request)
   ```

   To avoid this issue, open the `<BPS_HOME>/repository/conf/tomcat/catalina-server.xml` file and change the compression="on" to compression="off" in the Connector configuration, and restart the BPS.

6. Click **Add** under Processes and upload the `<API-M_HOME>/business-processes/application-creation/BPEL/ApplicationApprovalWorkflowProcess_1.0.0.zip` file to BPS.

   This is the business process archive file.
7. Select **Add** under the **Human Tasks** menu and upload the `<API-M_HOME>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip` file to BPS. This is the human task archived file.

**Configuring WSO2 API Manager**

Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json` file and configure "workFlowServer URL" under "workflows" to point to the BPS server (e.g., "workFlowServerURL": "https://localhost:9445/services/*")

**Engaging the WS Workflow Executor in the API Manager**

First, enable the application creation workflow.

1. Sign in to WSO2 API-M Management Console (`https://<Server-Host>:9443/carbon`) and select **Browse** under **Resources**.

2. Go to the `/system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor, and enable WS Workflow Executor. In addition, specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

   ```xml
   <WorkFlowExtensions>
     ... 
     <ApplicationCreation
       executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationCreationWSWorkflowExecutor">
       <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationApprovalWorkFlowProcess/</Property>
       <Property name="username">admin</Property>
       <Property name="password">admin</Property>
       <Property name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
     </ApplicationCreation>
     ... 
   </WorkFlowExtensions>
   ``

   All the workflow process services of the EI/BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

   The application creation WS Workflow Executor is now engaged.

   **Using EI/Using BPS**

   When enabling the workflow features, make sure you import the certificate of the API Manager into the **client-truststore** located inside the `<EI_HOME>/wso2/business-process/repository/resources/security` directory.

   When enabling the workflow features, make sure you import the certificate of the API Manager into the **client-truststore** located inside the `<BPS_HOME>/repository/resources/security` directory.

3. Go to the API Store, click **Applications** and create a new application.
   It invokes the application creation process and creates a Human Task instance that holds the execution of the BPEL process until some action is performed on it.

   Note that the **Status** field of the application states **INACTIVE (Waiting for approval)** if the BPEL is invoked correctly, indicating that the request is successfully submitted.
4. Sign in to the Admin Portal (https://localhost:9443/admin), list all the tasks for application creation and approve the task. It resumes the BPEL process and completes the application creation.
5. Go back to the Applications page in the WSO2 API Store and see the created application.

Whenever a user tries to create an application in the API Store, a request is sent to the workflow endpoint. A sample is shown below:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/*
xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
  <soapenv:Body>
    <wor:createApplication xmlns:wor="http://workflow.application.apimgt.carbon.wso2.org">
      <wor:applicationName>application1</wor:applicationName>
      <wor:applicationTier>Gold</wor:applicationTier>
      <wor:applicationCallbackUrl>http://webapp/url</wor:applicationCallbackUrl>
      <wor:applicationDescription>Application 1</wor:applicationDescription>
      <wor:tenantDomain>ws02.com</wor:tenantDomain>
      <wor:userName>user1</wor:userName>
      <wor:workflowExternalRef>c0aad878-278c-4439-8d7e-712ee71d3f1c</wor:workflowExternalRef>
    </wor:createApplication>
  </soapenv:Body>
</soapenv:Envelope>
```

Elements of the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Name of the application the user creates.</td>
</tr>
<tr>
<td>applicationTier</td>
<td>Throttling tier of the application.</td>
</tr>
<tr>
<td>applicationCallbackUrl</td>
<td>When the OAuth2 Authorization Code grant type is applied, this is the endpoint on which the callback needs to happen after the user is authenticated. This is an attribute of the actual application registered on the API Store.</td>
</tr>
<tr>
<td>applicationDescription</td>
<td>Description of the application</td>
</tr>
<tr>
<td>tenantDomain</td>
<td>Tenant domain associated with the application (domain of the user creating the application).</td>
</tr>
<tr>
<td>userName</td>
<td>Username of the user creating the application.</td>
</tr>
<tr>
<td>workflowExternalRef</td>
<td>The unique reference against which a workflow is tracked. This needs to be sent back from the workflow engine to the API Manager at the time of workflow completion.</td>
</tr>
<tr>
<td>callBackURL</td>
<td>This property is configured in the <code>&lt;callBackURL&gt;</code> element in the <code>workflow-extensions.xml</code> registry file.</td>
</tr>
</tbody>
</table>

### Adding an Application Registration Workflow

This section explains how to attach a custom workflow to the application registration operation in the API Manager. First, see Workflow Extensions for information on different types of workflow executors.

**Introduction to Application registration (Key generation) workflow**

Application creation and Application registration are different workflows. After an application is created, you can subscribe to available APIs, but you get the consumer key/secret and access tokens only after registering the application. There are two types of registrations that can be done to an application: production and sandbox. You change the default application registration workflow in situations such as the following:

1. To issue only sandbox keys when creating production keys is deferred until testing is complete.
2. To restrict untrusted applications from creating production keys. You allow only the creation of sandbox keys.
3. To make API subscribers go through an approval process before creating any type of access token.

**Before you begin.** If you have changed the API Manager's default user and role, make sure you do the following changes:
1. Change the credentials of the workflow configurations in the registry resource _system/governance/apimgt/applicationdata/workflow-extensions.xml.
   b. Click on browse under Resources in left Navigation under Main tab.
   c. Go to _system/governance/apimgt/applicationdata/workflow-extensions.xml location in registry browser and open the workflow-extensions.xml clicking Edit as text.
   d. Uncomment the following two sections and change the credentials of API Manager's default user credentials you have given.

   ```xml
   <ProductionApplicationRegistration executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
     <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
     <Property name="username">admin</Property>
     <Property name="password">admin</Property>
     <Property name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
   </ProductionApplicationRegistration>
   <SandboxApplicationRegistration executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
     <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
     <Property name="username">admin</Property>
     <Property name="password">admin</Property>
     <Property name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
   </SandboxApplicationRegistration>
   ```

   This configuration is provided assuming that WSO2 EI is running with offset 2. If you are running WSO2 EI in a different offset change the port of `serviceEndpoint` properties in following configuration according to the changed port offset.

   ```xml
   <ProductionApplicationRegistration executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
     <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
     <Property name="username">admin</Property>
     <Property name="password">admin</Property>
     <Property name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
   </ProductionApplicationRegistration>
   <SandboxApplicationRegistration executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
     <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
     <Property name="username">admin</Property>
     <Property name="password">admin</Property>
     <Property name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
   </SandboxApplicationRegistration>
   ```
2. Point the database that has the API Manager user permissions to EI.
   In this step you need to share the user store database in WSO2 API Manager with WSO2 EI.
   
   a. Copy the following datasource configuration in `<API-M_HOME>/repository/conf/datasources/master-datasources.xml`:

   ```xml
   <datasource>
       <name>WSO2UM_DB</name>
       <description>The datasource used by user manager</description>
       <jndiConfig>
           <name>jdbc/WSO2UM_DB</name>
       </jndiConfig>
       <definition type="RDBMS">
           <configuration>
               <username>user</username>
               <password>password</password>
               <driverClassName>com.mysql.jdbc.Driver</driverClassName>
               <maxActive>50</maxActive>
               <maxWait>60000</maxWait>
               <testOnBorrow>true</testOnBorrow>
               <validationQuery>SELECT 1</validationQuery>
               <validationInterval>30000</validationInterval>
           </configuration>
       </definition>
   </datasource>
   ```

   b. Change the datasource to point the WSO2UM_DB by changing the realm configuration in `<API-M_HOME>/repository/conf/user-mgt.xml` as shown below.

   ```xml
   <UserManager>
       <Realm>
           <Configuration>
               ....
               <Property name="dataSource">jdbc/WSO2UM_DB</Property>
           </Configuration>
           ....
       </Realm>
   </UserManager>
   ```

   c. Do the configuration described in (a) and (b) in `<EI_HOME>/wso2/business-process/conf/datasources/master-datasources.xml` and `<EI_HOME>/wso2/business-process/conf/user-mgt.xml` respectively.

3. Share any LDAPs, if exist.

4. Unzip the `<API-M>/business-processes/application-registration/HumanTask/ApplicationRegistrationTask-1.0.0.zip` file, update the role as follows in the `ApplicationRegistrationTask.ht` file, and ZIP the `ApplicationRegistrationTask`.
Configuring the Business Process Server

1. Download WSO2 Enterprise Integrator.
2. Set an offset of 2 to the default EI port in the `<EI_HOME>/wso2/business-process/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

```xml
<Offset>2</Offset>
```


```xml
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```


- If the `<EI_HOME>/wso2/business-process/conf/epr` folder does not exist, create it.
- Make sure you give the correct credentials in the `<EI_HOME>/wso2/business-process/conf/epr` files.

```xml
<wsa:Address>https://localhost:8243/services/WorkflowCallbackService</wsa:Address>
```

5. Start the BPS server and log in to its management console (https://<Server Host>:9443+<port offset>/carbon).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.
6. Log into Management console of WSO2 EI, select **Add > BPEL** under the **Processes** menu and upload the **<APIM_HOME>/business-processes/application-registration/BPEL/ApplicationRegistrationWorkflowProcess_1.0.0.zip** file to EI. This is the business process archive file.

7. Select **Add** under the **Human Tasks** menu and upload the **<APIM_HOME>/business-processes/application-registration/HumanTask/ApplicationRegistrationTask-1.0.0.zip** file to EI. This is the human task archived file.

To avoid this issue open **<EI_HOME>/repository/conf/tomcat/catalina-server.xml** and change the compression="on" to compression="off" in Connector configuration and restart the EI.

---

**WARN** [org.owasp.csrfguard.log.JavaLogger] - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST, uri:/carbon/admin/login_action.jsp, error:required token is missing from the request)

---

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

1. Change the credentials of the workflow configurations in the registry resource **/_system/governance/apimgt/applicationdata/workflow-extensions.xml**.
   a. Login to the Management console of WSO2 API Manager in **https://localhost:9443/carbon**.
   b. Click on browse under Resources in left Navigation under Main tab.
   c. Go to **/_system/governance/apimgt/applicationdata/workflow-extensions.xml** location in registry browser and open the workflow-extensions.xml clicking **Edit as text**.
d. Uncomment the following two sections and change the credentials of API Manager's default user credentials you have given.

This configuration is provided assuming that WSO2 BPS is running with offset 2. If you are running WSO2 BPS in a different offset change the port of serviceEndpoint properties in following configuration according to the changed port offset.

```xml
<ProductionApplicationRegistration
    executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
    <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
    <Property name="username">admin</Property>
    <Property name="password">admin</Property>
    <Property name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
</ProductionApplicationRegistration>

<SandboxApplicationRegistration
    executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
    <Property name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
    <Property name="username">admin</Property>
    <Property name="password">admin</Property>
    <Property name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
</SandboxApplicationRegistration>

Make sure to comment out the existing ProductionApplicationRegistration and SandboxApplicationRegistration executors as shown below.
2. Point the database that has the API Manager user permissions to BPS.
In this step you need to share the user store database in WSO2 API Manager with WSO2 BPS.

   a. Copy the following datasource configuration in `<API-M_HOME>/repository/conf/datasources/master-datasources.xml`:

   ```xml
   <datasource>  
   <name>WSO2UM_DB</name>  
   <description>The datasource used by user manager</description>  
   <jndiConfig>  
   <name>jdbc/WSO2UM_DB</name>  
   </jndiConfig>  
   <definition type="RDBMS">  
   <configuration>  
   <username>user</username>  
   <password>password</password>  
   <driverClassName>com.mysql.jdbc.Driver</driverClassName>  
   <maxActive>50</maxActive>  
   <maxWait>60000</maxWait>  
   <testOnBorrow>true</testOnBorrow>  
   <validationQuery>SELECT 1</validationQuery>  
   <validationInterval>30000</validationInterval>  
   </configuration>  
   </definition>  
   </datasource>
   ``

   b. Change the datasource to point the WSO2UM_DB by changing the realm configuration in `<API-M_HOME>/repository/conf/user-mgt.xml` as shown below.

   ```xml
   <UserManager>  
   <Realm>  
   <Configuration>  
   ....  
   <Property name="dataSource">jdbc/WSO2UM_DB</Property>  
   ....  
   </Configuration>  
   ....  
   </Realm>  
   </UserManager>
   ``

   c. Do the configuration described in (a) and (b) in `<BPS_HOME>/repository/conf/datasources/master-datasources.xml` and `<BPS_HOME>/repository/conf/user-mgt.xml` respectively.

3. Share any LDAPs, if exist.

4. Unzip the `<API-M>/business-processes/application-registration/HumanTask/ApplicationRegistrationTask-1.0.0.zip` file, update the role as follows in the `ApplicationRegistrationTask.html` file, and ZIP the `ApplicationRegistrationTask-1.0.0` folder.
Configuring the Business Process Server

2. Set an offset of 2 to the default BPS port in the `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

   `<Offset>2</Offset>`

3. Open the `<BPS_HOME>/repository/conf/humantask.xml` file and set the `<TaskCoordinationEnabled>true</TaskCoordinationEnabled>` property to true.

4. Copy the following from the `<API-M_HOME>/business-processes/epr` folder to the `<BPS_HOME>/repository/conf/epr` folder. If the `<BPS_HOME>/repository/conf/epr` folder does not exist, create it.

   Make sure you give the correct credentials in the `<BPS_HOME>/repository/conf/epr` files.

   * RegistrationService.epr
   * RegistrationCallbackService.epr

5. HERE Start the BPS server and log in to its management console (https://<Server Host>:9443+<port offset>/carbon).

   If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

   WARN {org.owasp.csrfguard.log.JavaLogger} - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>), ip:xxx.xxx.xx.xx, method:POST, uri:carbon/admin/login_action.jsp, error:required token is missing from the request

   To avoid this issue open `<BPS_HOME>/repository/conf/tomcat/catalina-server.xml` and change the compression="on" to compression="off" in Connector configuration and restart the BPS.

6. Log into Management console of WSO2 BPS, select Add > BPEL under the Processes menu and upload the `<API-M_HOME>/business-processes/application-registration/BPEL/ApplicationRegistrationWorkflowProcess_1.0.0.zip` file to BPS. This is the business process archive file.
7. Select **Add** under the **Human Tasks** menu and upload the `<APIM_HOME>/business-processes/application-registration/HumanTask/ApplicationRegistrationTask-1.0.0.zip` file to BPS. This is the human task archived file.

---

**Configuring the API Manager**

Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json` file and configure the "workFlowServerURL" under "workflows" to point to the EI/BPS server (e.g. "workFlowServerURL": "https://localhost:9445/services/")

```json
{
    ....
    "context": "/admin",
    "request_url": "READ_FROM_REQUEST",
    "tasksPerPage": 10,
    "allowedPermission": "/permission/admin/manage/apim_admin",
    "workflows": {
      "workFlowServerURL": "https://localhost:9445/services/",
    }
    ....
}
```

---

**Engaging the WS Workflow Executor in the API Manager**

First, enable the application registration workflow.

1. Start WSO2 API Manager and login to the APIM management console (https://<Server Host>:9443/carbon) and select **Browse** under Resources.

2. Go to the `_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor as described in the tip provided at the start of this documentation if you haven't done already.
<WorkFlowExtensions>
  ...
  <ProductionApplicationRegistration
    executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
    <Property
      name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkFlowProcess/
    </Property>
    <Property
      name="username">admin</Property>
    <Property
      name="password">admin</Property>
    <Property
      name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
  </ProductionApplicationRegistration>
  ...
  <SandboxApplicationRegistration
    executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
    <Property
      name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkFlowProcess/
    </Property>
    <Property
      name="username">admin</Property>
    <Property
      name="password">admin</Property>
    <Property
      name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
  </SandboxApplicationRegistration>
  ...
</WorkFlowExtensions>

Note that all workflow process services of the EI/BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

Using EI/Using BPS

When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the `<EI_HOME>/wso2/business-process/repository/resources/security` directory.

When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the `<BPS_HOME>/repository/resources/security` directory.

3. Log into the API Store (https://localhost:9443/store) as a Store user and open the application with which you subscribed to the API.

If you do not have an already created API and an Application subscribed to it, follow Create and Publish an API and upto step 8 of Subscribe to an API to create an API and subscribe to it.

4. In the Production Keys tab of the Application, click the Generate Keys button.
   It invokes the ApplicationRegistrationWorkflowProcess.bpel that is bundled with the ApplicationRegistrationWorkflowProcess_1.0.0.zip file and creates a HumanTask instance that holds the execution of the BPEL process until some action is performed on it.

5. Note that a message appears saying that the request is successfully submitted if the BPEL was invoked correctly.
6. Log in to the Admin Portal (https://<Server Host>:9443/admin) with admin credentials and list all the tasks for application registrations. Click Start to start the Human Task and then change its state. Once you select Approve and click Complete the task, it resumes the BPEL process and completes the registration.
7. Go back to the API Store and view your application.

It shows the application access token, consumer key and consumer secret.

After the registration request is approved, keys are generated by invoking the APIKeyMgtSubscriber service hosted in Key Manger nodes. Even when the request is approved, key generation can fail if this service becomes unavailable. To address such failures, you can configure to trigger key generation at a time Key Manager nodes become available again. Given below is the message used to invoke the BPEL process:

```xml
<applicationregistrationworkflowprocessrequest
 xmlns:wor="http://workflow.application.apimgt.carbon.wso2.org"
 xmlns="http://workflow.application.apimgt.carbon.wso2.org">
 <applicationname>NewApp5</applicationname>
 <applicationtier>Unlimited</applicationtier>
 <applicationcallbackurl></applicationcallbackurl>
 <applicationdescription></applicationdescription>
 <tenantdomain>carbon.super</tenantdomain>
 <username>admin</username>
 <workflowexternalref>4a20749b-a10d-4fa5-819b-4fae5f57ffaf</workflowexternalref>
 <callbackurl>https://localhost:8243/services/WorkflowCallbackService</callbackurl>
 <keytype>PRODUCTION</keytype>
</applicationregistrationworkflowprocessrequest>
```

Adding an API Subscription Workflow

This section explains how to attach a custom workflow to the API subscription operation in the API Manager. First, see Workflow Extensions for information on different types of workflows executors.

Attaching a custom workflow to API subscription enables you to add throttling tiers to an API that consumers cannot choose at the time of subscribing. Only admins can set these tiers to APIs. When a consumer subscribes to an API, he/she has to subscribe to an application in order to get access to the API. However, when API subscription workflow is enabled, when the consumer subscribes to an application, it initially is in the On Hold state, and he/she cannot use the API, using its production or sandbox keys, until their subscription is approved.

Configuring the Business Process Server

1. Download WSO2 Enterprise Integrator.
2. Set an offset of 2 to the default EI port in `<EI_HOME>/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than
one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.


   `TaskCoordinationEnabled=true</TaskCoordinationEnabled>`

4. Copy the following from `<API-M_HOME>/business-processes/epr` to `<EI_HOME>/epr` folder. If the `<EI_HOME>/wso2/business-process/repository/conf/epr` folder isn't there, please create it.

   Make sure to give the correct credentials in the `<EI_HOME>/wso2/business-process/repository/conf/epr` files.

   - Update the `<API-M_HOME>/business-processes/epr/SubscriptionCallbackService.epr` file according to API Manager.

     `<wsa:Address>https://localhost:8243/services/WorkflowCallbackService</wsa:Address>`

   - Update the `<API-M_HOME>/business-processes/epr/SubscriptionService.epr` file according to EI.

     `<wsa:Address>http://localhost:9765/services/SubscriptionService/</wsa:Address>`

5. Start the EI server and sign in to its management console (https://<Server Host>:9443+<port offset>/carbon).

   If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

   WARN {org.owasp.csrfguard.log.JavaLogger} - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST, uri:/carbon/admin/login_action.jsp, error:required token is missing from the request)

   To avoid this issue open `<EI_HOME>/conf/tomcat/catalina-server.xml` and change the compression="on" to compression="off" in Connector configuration and restart the EI.

6. Select Add under the Processes menu and upload the `<API-M_HOME>/business-processes/subscription-creation/BPEL/SubscriptionApprovalWorkFlowProcess_1.0.0.zip` file to EI. This is the business process archive file.
7. Select **Add** under the **Human Tasks** menu and upload the `<API-M_HOME>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip` file to EI. This is the human task archived file.

**Before you begin,** if you have changed the API Manager’s default user and role, make sure you do the following changes:

- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Unzip the `<API-M>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip` file, update the role as follows in the `SubscriptionsApprovalTask.htm` file, and ZIP the `SubscriptionsApprovalTask-1.0` folder.

```xml
<htd:argument name="role">[new-role-name]</htd:argument>
```

**Configuring the Business Process Server**

1. Download **WSO2 Business Process Server**.
2. Set an offset of 2 to the default BPS port in `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

```xml
<Offset>2</Offset>
```

**Tip:** If you change the BPS port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside the `<API-M_HOME>/business-processes` folder with the new port (9763 + port offset.)

3. Open the `<BPS_HOME>/repository/conf/humantask.xml` file and `<BPS_HOME>/repository/conf/b4p-coordination-config.xml` file and set the `TaskCoordinationEnabled` property to true.

```xml
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```

4. Copy the following from `<API-M_HOME>/business-processes/epr` to `<BPS_HOME>/repository/conf/epr` folder. If the `<BPS_HOME>/repository/conf/epr` folder isn’t there, please create it.

**Make sure to give the correct credentials in the `<BPS_HOME>/repository/conf/epr` files.**

- Update the `<API-M_HOME>/business-processes/epr/SubscriptionCallbackService.epr` file according to API Manager.

```xml
<wsa:Address>https://localhost:8243/services/WorkflowCallbackService</wsa:Address>
```

- Update the `<API-M_HOME>/business-processes/epr/SubscriptionService.epr` file according to BPS.

```xml
<wsa:Address>http://localhost:9765/services/SubscriptionService/</wsa:Address>
```

5. Start the BPS server and sign in to its management console ([https://<Server Host>:9443+<port offset>/carbon](https://<Server Host>:9443+<port offset>/carbon)).
5. Select **Add** under the **Processes** menu and upload the `<API-M_HOME>/business-processes/subscription-creation/BPEL/SubscriptionApprovalWorkFlowProcess_1.0.0.zip` file to BPS. This is the business process archive file.

6. Select **Add** under the **Processes** menu and upload the `<API-M_HOME>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip` file to BPS. This is the human task archived file.

### Configuring the API Manager

Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json` file and configure "workFlowServerURL" under "workflows" to point to the EI/BPS server (e.g. "workFlowServerURL": "https://localhost:9443/services/"

### Engaging the WS Workflow Executor in the API Manager

1. Sign in to API Manager Management Console (`https://<Server Host>:9443/carbon`) and select **Browse** under **Resources**.

2. Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```xml
<WorkFlowExtensions>
  ...
  <SubscriptionCreation executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationWSWorkflowExecutor">
    <Property name="serviceEndpoint">http://localhost:9765/services/SubscriptionApprovalWorkFlowProcess/</Property>
    <Property name="username">admin</Property>
    <Property name="password">admin</Property>
    <Property name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
  </SubscriptionCreation>
  ...
</WorkFlowExtensions>
```

**Note** that all workflow process services of the EI/BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST, uri:/carbon/admin/login_action.jsp, error:required token is missing from the request)
```

To avoid this issue open `<BPS_HOME>/repository/conf/tomcat/catalina-server.xml` and change the compression="on" to compression="off" in Connector configuration and restart the BPS.
The application creation WS Workflow Executor is now engaged.

Using EI

Using BPS

When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the <EI_HOME>/wso2/business-process/repository/resources/security directory.

When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the <BPS_HOME>/repository/resources/security directory.

3. Go to the API Store Web interface and subscribe to an API. It invokes the API subscription process and creates a Human Task instance that holds the execution of the BPEL until some action is performed on it.

4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.

5. Sign in to the Admin Portal (https://<Server Host>:9443/admin), list all the tasks for API subscription and approve the task. It resumes the BPEL process and completes the API subscription.

6. Go back to the API Store and see that the user is now subscribed to the API.

Whenever a user tries to subscribe to an API, a request of the following format is sent to the workflow endpoint:

```xml
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
  <soapenv:Header/>
  <soapenv:Body>
    <wor:createSubscription>
      <wor:apiName>sampleAPI</wor:apiName>
      <wor:apiVersion>1.0.0</wor:apiVersion>
      <wor:apiContext>/sample</wor:apiContext>
      <wor:apiProvider>admin</wor:apiProvider>
      <wor:subscriber>subscriber1</wor:subscriber>
      <wor:applicationName>application1</wor:applicationName>
      <wor:tierName>gold</wor:tierName>
      <wor:workflowExternalRef></wor:workflowExternalRef>
      <wor:callBackURL>?</wor:callBackURL>
    </wor:createSubscription>
  </soapenv:Body>
</soapenv:Envelope>
```

Elements of the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiName</td>
<td>Name of the API to which subscription is requested.</td>
</tr>
<tr>
<td>apiVersion</td>
<td>Version of the API the user subscribes to.</td>
</tr>
<tr>
<td>apiContext</td>
<td>Context in which the requested API is to be accessed.</td>
</tr>
<tr>
<td>apiProvider</td>
<td>Provider of the API.</td>
</tr>
<tr>
<td>subscriber</td>
<td>Name of the user requesting subscription.</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application through which the user subscribes to the API.</td>
</tr>
<tr>
<td>tierName</td>
<td>Throttling tiers specified for the application.</td>
</tr>
<tr>
<td>workflowExternalRef</td>
<td>The unique reference against which a workflow is tracked. This needs to be sent back from the workflow engine to the API Manager at the time of workflow completion.</td>
</tr>
<tr>
<td>callBackURL</td>
<td>The URL to which the Workflow completion request is sent to by the workflow engine, at the time of workflow completion. This property is configured in the &lt;callBackURL&gt; element in the workflow-extensions.xml registry file.</td>
</tr>
</tbody>
</table>
Adding a User Signup Workflow

This section explains how to attach a custom workflow to the user signup operation in the API Manager. First, see Workflow Extensions for information on different types of workflow executors.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Change the credentials of the workflow configurations in the registry resource _system/governance/apimgt/applicationdata/workflow-extensions.xml.
- Point the database that has the API Manager user permissions to EI.
- Share any LDAPs, if exist.
- Unzip the <API-M>/business-processes/user-signup/UserApprovalTask-1.0.0.zip file, update the role as follows in the UserApprovalTask.ht file, and ZIP the UserApprovalTask.ht folder.

```xml
<htd:argument name="role">
  [new-role-name]
</htd:argument>
```

**Format**

**Configuring the Business Process Server**

1. Download WSO2 Enterprise Integrator.
2. Set an offset of 2 to the default EI port in the <EI_HOME>/wso2/business-process/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

```xml
<Offset>2</Offset>
```

3. Open the <API-M_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip file and set the TaskCoordinationEnabled property to true. For further information on this configuration see Configuring Human Task Coordination.

```xml
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```

4. Copy the following from the <API-M_HOME>/business-processes/epr folder to the <EI_HOME>/wso2/business-process/repository folder.

**Tip**: If you change the EI port offset to a value other than 2 or run the API Manager and EI on different machines (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <API-M_HOME>/business-processes folder with the new port (9763 + port offset).

**Note**: Make sure that the port offset is updated in the following files as well. Note that the zipped files should be unzipped for you to be able to see the files

- <API-M_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip/UserApprovalTask.wsdl
- <API-M_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip/UserApprovalTask.wsdl
- <API-M_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip/WorkflowCallbackService.wsdl
Before you begin, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Change the credentials of the workflow configurations in the registry resource `system/governance/apimgt/apimgt/applicationdata/workflow-extensions.xml`.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Unzip the `<API-M_HOME>/business-processes/user-signup/UserApprovalTask-1.0.0.zip` file, update the role as follows in the `UserApprovalTask.ht` file, and ZIP the `UserApprovalTask.ht` folder.

```
<htd:argument name="role">
  [new-role-name]
</htd:argument>
```
Configuring the Business Process Server


2. Set an offset of 2 to the default BPS port in the `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. For more information, see Changing the Default Ports with Offset.

   ```
   <Offset>2</Offset>
   ```

   **Tip:** If you change the BPS port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the hostname to a different value than localhost), you do the following:

   - Search and replace the value 9765 in all the files (.epr) inside `<API-M_HOME>/business-processes` folder with the new port (9763 + port offset).

   **Note:** Make sure that the port offset is updated in the following files as well. Note that the zipped files should be unzipped for you to be able to see the files:
   - `<API-M_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip/UserApprovalTask.wsdl`
   - `<API-M_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip/UserApprovalTask.wsdl`
   - `<API-M_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip/WorkflowCallbackService.wsdl`

3. Open the `<BPS_HOME>/repository/conf/humantask.xml` file and `<BPS_HOME>/repository/conf/b4p-coordination-config.xml` file and set the `TaskCoordinationEnabled` property to true. For further information on this configuration see Configuring Human Task Coordination.

   ```
   <TaskCoordinationEnabled>true</TaskCoordinationEnabled>
   ```

4. Copy the following from the `<API-M_HOME>/business-processes/epr` folder to the `<BPS_HOME>/repository/conf/epr` folder.

   - If the `<BPS_HOME>/repository/conf/epr` folder isn’t there, please create it.
   - Make sure to give the correct credentials in the `<BPS_HOME>/repository/conf/epr` files.

   - Update the `<API-M_HOME>/business-processes/epr/UserSignupProcess.epr` file according to API Manager.

     ```
     <wsa:Address>https://localhost:8243/services/WorkflowCallbackService</wsa:Address>
     ```

   - Update the `<API-M_HOME>/business-processes/epr/UserSignupService.epr` file according to BPS.

     ```
     ```

5. Start the BPS server and log in to its management console (https://<Server Host>:9443+<port offset>/carbon).

   If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.
5. Select Add under the Processes menu and upload the `<API-M_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalsProcess_1.0.0.zip` file to BPS. This is the business process archive file.

6. Select Add under the Processes menu and upload the `<API-M_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip` file to BPS. This is the human task archived file.

### Configuring the API Manager

1. Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json` file and configure "workFlowServerURL" under "workflows" to point to the EI/BPS server (e.g. "workFlowServerURL": "https://localhost:9445/services/"

### Engaging the WS Workflow Executor in the API Manager

First, enable the user signup workflow.

1. Log in to APIM management console (https://<Server Host>:9443/carbon) and select Browse under Resources.

2. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```xml
<WorkflowExtensions>
  ... 
  <UserSignUp executor="org.wso2.carbon.apimgt.impl.workflow.UserSignUpWSWorkflowExecutor">
    <Property name="serviceEndpoint">http://localhost:9765/services/UserSignupProcess/</Property>
    <Property name="username">admin</Property>
    <Property name="password">admin</Property>
    <Property name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
  </UserSignUp>
  ... 
</WorkflowExtensions>
```

**Note** that all workflow process services of the EI/BPS run on port 9765 because you changed its default port (9763) with an offset of 2.
Using EI/Using BPS

When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the <EI_HOME>/wso2/business-process/repository/resources/security directory.
When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the <BPS_HOME>/repository/resources/security directory.

3. Go to the API Store Web interface and sign up.
It invokes the signup process and creates a Human Task instance that holds the execution of the BPEL until some action is performed on it.
4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
5. Log in to the Admin Portal (https://<Server Host>:9443/admin) and approve the user signup task. It resumes the BPEL process and completes the signup process.
6. Go back to the API Store and see that the user is now registered.

Whenever a user tries to sign up to the API Store, a request of the following format is sent to the workflow endpoint:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
  <soapenv:Header />
  <soapenv:Body>
      <wor:userName>sampleuser</wor:userName>
      <wor:tenantDomain>foo.com</wor:tenantDomain>
      <wor:workflowExternalRef>c0aad878-278c-4439-8d7e-712ee71d3f1c</wor:workflowExternalRef>
    </wor:registerUser>
  </soapenv:Body>
</soapenv:Envelope>
```

Elements of the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>The user name requested by the user</td>
</tr>
<tr>
<td>tenantDomain</td>
<td>Domain to which the user belongs to</td>
</tr>
<tr>
<td>workflowExternalRef</td>
<td>The unique reference against which a workflow is tracked. This needs to be sent from the workflow engine to the API Manager at the time of workflow completion.</td>
</tr>
<tr>
<td>callBackURL</td>
<td>The URL to which the workflow completion request is sent by the workflow engine, at the time of workflow completion. This property is configured under the <code>callBackURL</code> property in the <code>Workflow-extensions.xml</code> registry file.</td>
</tr>
</tbody>
</table>

Invoking the API Manager from the BPEL Engine

Once the workflow configurations are finalized at the BPEL, the call-back URL of the APIM, which is originally configured in the `<APIM_HOME>/repository/conf/api-manager.xml` file and sent to the BPEL engine in the outflow will be called to progress the workflow. In APIM, the endpoint is available in both SOAP and REST variants as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>URI</th>
<th>WSDL Location</th>
</tr>
</thead>
</table>

Both the endpoints are secured via basic authentication. Therefore, when you invoke either endpoint, you need to include an authorization header with a base64-encoded value of the username and password with the request. E.G., Authorization: Basic <base64 encoded `username:password`>.

The endpoint expects the following list of parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Customizing a Workflow Extension

Each workflow executor in the WSO2 API Manager is inherited from the `org.wso2.carbon.apimgt.impl.workflow.WorkflowExecutor` abstract class, which has the following abstract methods:

- **execute**: contains the implementation of the workflow execution
- **complete**: contains the implementation of the workflow completion
- **getWorkflowType**: abstract method that returns the type of the workflow as a String
- **getWorkflowDetails(String workflowStatus)**: abstract method that returns a list of `WorkflowDTO` objects. This method is not used at the moment and it returns null for the time being.

To customize the default workflow extension, you override the `execute()` and `complete()` methods with your custom implementation. For example, the following class is a sample implementation of the Subscription Creation workflow. It returns an email to an address provided through the configuration on each subscription creation:
private String emailPassword;
@override
public List<WorkflowDTO> getWorkflowDetails(String arg0) throws WorkflowException {
    return null;
}
@override
public String getWorkflowType() {
    return WorkflowConstants.WF_TYPE_AM_SUBSCRIPTION_CREATION;
}
@override
public WorkflowResponse execute(WorkflowDTO workflowDTO) throws WorkflowException {
    SubscriptionWorkflowDTO subsCreationWFDTO = (SubscriptionWorkflowDTO) workflowDTO;
    Properties props = new Properties();
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.host", "smtp.gmail.com");
    props.put("mail.smtp.port", "587");
    Session session = Session.getInstance(props, new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(emailAddress, emailPassword);
        }
    });
    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(emailAddress));
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(adminEmail));
        message.setSubject("Subscription Creation");
        message.setText("Subscription created for API " + subsCreationWFDTO.getApiName() + " using
Application " + subsCreationWFDTO.getApplicationName() + " by user " +
subsCreationWFDTO.getSubscriber());
        Transport.send(message);
        System.out.println("Sent email to notify subscription creation");
        //Call the execute method of the parent class. This will create a reference for the
        //workflow execution in the database.
        super.execute(workflowDTO);
        //Set the workflow Status to APPROVED and immediately complete the workflow since we
        //are not waiting for an external party to complete this.
        workflowDTO.setUpdatedTime(System.currentTimeMillis());
        super.complete(workflowDTO);
        //Set the workflow Status to APPROVED and immediately complete the workflow since we
        //are not waiting for an external party to complete this.
        workflowDTO.setStatus(WorkflowStatus.APPROVED);
        complete(workflowDTO);
    } catch(MessagingException e) {
        e.printStackTrace();
        throw new WorkflowException(e.getMessage());
    } catch(Exception e) {
        e.printStackTrace();
        throw new WorkflowException(e.getMessage());
    }
    return new GeneralWorkflowResponse();
}
@override
public WorkflowResponse complete(WorkflowDTO workflowDTO) throws WorkflowException {
    workflowDTO.setUpdatedTime(System.currentTimeMillis());
    super.complete(workflowDTO);
    ApiMgtDAO apiMgtDAO = ApiMgtDAO.getInstance();
    try {
        Integer.parseInt(workflowDTO.getWorkflowReference());
    } catch(APIManagementException e) {
        throw new WorkflowException("Could not complete subscription creation workflow", e);
    }
    return new GeneralWorkflowResponse();
}
public String getAdminEmail() {
    return adminEmail;
}
public void setAdminEmail(String adminEmail) {
    this.adminEmail = adminEmail;
}
public String getEmailAddress() {
    return emailAddress;
}
public String getEmailPassword() {
    return emailPassword;
}
public void setEmailPassword(String emailPassword) {
    this.emailPassword = emailPassword;
}

this.emailPassword = emailPassword;
}
}

Note the following regarding the above sample:

- The `execute()` method takes in a `WorkflowDTO` object (`.SubscriptionWorkflowDTO` class) that contains information about the subscription that is being created.
- The `adminEmail`, `emailAddress` and `emailPassword` are private String variables with public `getter` and `setter` methods. The values for these variables are populated through the server configuration.
- After sending the email, a call is made to the super class's `execute()` method in order to create a reference entry in the database. This entry is generally used to look up the workflow when the workflow happens asynchronously (via a human approval).
- The `complete()` method contains the code to mark the subscription active. Until then, the subscription is in ON_HOLD state.
- In this sample, the `complete()` method is called immediately to make the subscription active instantly. If the completion of your workflow happens asynchronously, you must not call the `complete()` method from the `execute()` method.
- The `WorkflowException` is thrown to roll back the subscription in case of a failure.

In a distributed setup, the custom workflows should be deployed in the Store node.

After the implementation of the class is done, follow the steps below to implement the new workflow extension in the API Manager:

1. Compile the class and export it as a JAR file. Make sure you have the following JARs in the classpath before compilation.
   - `<API-M_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.impl_6.1.66.jar`
   - `<API-M_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.api_6.1.66.jar`
   - `javax.mail.jar`: see https://java.net/projects/javamail/pages/Home to download the JAR
2. After exporting the JAR, copy it to `<API-M_HOME>/repository/components/lib` directory.
3. Log in to APIM management console (https://<Server Host>:9443/carbon) and select Browse under Resources.

4. Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable the WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication). For example:

```xml
<WorkflowExtensions>
  ...
  <!--SubscriptionCreation executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationSimpleWorkflowExecutor="/-->
  <SubscriptionCreation executor="org.wso2.sample.workflow.SubsCreationEmailSender">
    <Property name="adminEmail">to_user@email.com</Property>
    <Property name="emailAddress">from_user@email.com</Property>
    <Property name="emailPassword">from_user_password</Property>
  </SubscriptionCreation>
  ...
</WorkflowExtensions>
```

Note that the `adminEmail`, `emailAddress` and `emailPassword` properties will be assigned to the appropriate variables defined in the class through the public `setter` methods of those variables.

If you use the same or similar sample to return an email, you must remove the `org.jaggeryjs.hostobjects.email_0.9.0.ALPHA4.wso2v1.jar` file from the `<API-M_HOME>/repository/components/plugins` directory. Removing it results in a `ClassNotFoundException` thrown at server startup, but it does not affect the server's functionality.

### Configuring HTTP Redirection for Workflows

This section walks you through how to redirect to a third party entity using the redirect URL as part of a custom workflow extension. For example, consider an API Manager user publishes an API and wants to make that API a chargeable API. If there are no payment details of the subscriber, that subscriber is forwarded to a third party entity that handles the payment detail collection etc.
Writing the custom workflow executor

Deploying the custom workflow executor

Using the workflow

Invoking the API Manager

Each workflow executor in the WSO2 API Manager is inherited from the `org.wso2.carbon.apimgt.impl.workflow.WorkflowExecutor` abstract class, which has the following abstract methods:

- `execute`: contains the implementation of the workflow execution
- `complete`: contains the implementation of the workflow completion
- `getWorkflowType`: abstract method that returns the type of the workflow as a String
- `getWorkflowDetails(String workflowStatus)`: abstract method that returns a list of `WorkflowDTO` objects. This method is not used at the moment and it returns null for the time being.

To customize the default workflow extension, you override the `execute()` and `complete()` methods with your custom implementation.

**Writing the custom workflow executor**

1. Extend the `WorkflowExecutor` class found in the `org.wso2.carbon.apimgt.impl.workflow` package.
2. Upon extension of the `WorkflowExecutor` class, override the `complete()` and `execute()` methods.
3. The `execute()` method is the first method called by API Manager. Call the `super.execute` method inside the `execute()` method to add the workflow entry to the database.
4. Create a response of type `WorkflowResponse`. For HTTP responses, WSO2 API Manager has an inbuilt object named `HTTPWorkflowResponse` found at `org.wso2.carbon.apimgt.api.WorkflowResponse`. When creating the HTTP workflow response object, specify the additional parameters and the redirect URL. The usage of these parameters are listed below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redirect URL</td>
<td>Yes</td>
<td>The URL that API Manager will redirect the user to upon workflow execution.</td>
</tr>
<tr>
<td>Redirect Confirmation Message</td>
<td>No</td>
<td>A redirection notification can be specified for a notification to appear at the point of redirecting. By default, this is set to a null value so the text must be specified in order for the notification to appear.</td>
</tr>
<tr>
<td>Additional Parameters</td>
<td>Yes</td>
<td>If you need to redirect back to the API Manager, call the workflow call back service to complete the workflow. To invoke this service, set the callback URL and the workflow reference ID in the additional parameters. These parameters are sent to the third party entity by query parameters.</td>
</tr>
</tbody>
</table>

5. Implement the `complete()` method, which the third party entity invokes to complete the workflow. Update the workflow status with the workflow status received by the third party entity.
6. A sample implementation of a custom workflow executor is shown below:

```java
package org.wso2.sample.workflow;
/*
 * Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 * WSO2 Inc. licenses this file to you under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except
 * in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

import org.wso2.carbon.apimgt.api.APIManagementException;
import org.wso2.carbon.apimgt.api.WorkflowResponse;
import org.wso2.carbon.apimgt.impl.APIConstants;
import org.wso2.carbon.apimgt.impl.dao.ApiMgtDAO;
import org.wso2.carbon.apimgt.impl.dto.WorkflowDTO;
import org.wso2.carbon.apimgt.impl.workflow.*;

import java.util.List;
/***/
public class SubscriptionCreationSampleWorkflowExecutor extends WorkflowExecutor {

    @Override
    public String getWorkflowType() {
        return WorkflowConstants.WF_TYPE_AM_SUBSCRIPTION_CREATION;
    }

    @Override
    public List<WorkflowDTO> getWorkflowDetails(String workflowStatus) throws WorkflowException {
        return null;
    }

    @Override
    public WorkflowResponse execute(WorkflowDTO workflowDTO) throws WorkflowException {
        super.execute(workflowDTO);

        HttpWorkflowResponse httpworkflowResponse = new HttpWorkflowResponse();
        httpworkflowResponse.setRedirectUrl("http://google.lk");
        httpworkflowResponse.setAdditionalParameters("CallbackUrl",
                gg");
        httpworkflowResponse.setAdditionalParameters("workflowRefId", workflowDTO.getExternalWorkflowReference());
        httpworkflowResponse.setRedirectConfirmationMsg("you will be redirected to http://google.lk");
        return httpworkflowResponse;
    }

    @Override
    public WorkflowResponse complete(WorkflowDTO workflowDTO) throws WorkflowException {
        workflowDTO.setUpdatedTime(System.currentTimeMillis());
        super.complete(workflowDTO);
        ApiMgtDAO apiMgtDAO = new ApiMgtDAO();
        if(WorkflowStatus.APPROVED.equals(workflowDTO.getStatus())) {
            try {
                apiMgtDAO.updateSubscriptionStatus(Integer.parseInt(workflowDTO.getWorkflowReference()), APIConstants.SubscriptionStatus.UNBLOCKED);
            } catch (APIManagementException e) {
                log.error("Could not complete subscription creation workflow", e);
                throw new WorkflowException("Could not complete subscription creation workflow", e);
            }
        } else if(WorkflowStatus.REJECTED.equals(workflowDTO.getStatus())) {
            try {
            }
        }
    }
}
apiMgtDAO.updateSubscriptionStatus(Integer.parseInt(workflowDTO.getWorkflowReference()), APIConstants.SubscriptionStatus.REJECTED);
    } catch (APIManagementException e) {
        log.error("Could not complete subscription creation workflow", e);
        throw new WorkflowException("Could not complete subscription creation workflow", e);
    }
    return new GeneralWorkflowResponse();
Deploying the custom workflow executor

1. Once you have written the custom workflow executor, compile it to a .jar file.
2. Place the .jar file in the <APIM_HOME>/repository/components/lib directory and restart the server.

Using the workflow

1. Log in to API Manager Management Console (https://<Server Host>:9443/carbon) and select Browse under Resources.
2. Navigate to the _system/governance/apimgt/applicationdata/workflow-extensions.xml resource and disable the simple workflow executor.
3. Add and enable the implemented executor.
4. Specify the service endpoint where the workflow engine is hosted and the credentials required to access the service, via basic authentication (i.e., username/password based authentication).
5. An example configuration is shown below:

```xml
<WorkFlowExtensions>
  ...
  <SubscriptionCreation
  executor="org.wso2.sample.workflow.SubscriptionCreationSampleWorkflowExecutor">
  </SubscriptionCreation>
  ...
</WorkFlowExtensions>
```

Invoking the API Manager

To invoke the API Manager from a third party entity, see Invoking the API Manager from the BPEL Engine.

Configuring Workflows for Tenants

Using the API Manager, you can configure custom workflows that get invoked at the event of a user signup, application creation, registration, subscription etc. You do these configurations in the workflow-extensions.xml as described in the previous sections.

However, in a multi-tenant API Manager setup, not all tenants have access to the file system and not all tenants want to use the same workflow that the super admin has configured in the api-manager.xml file. For example, different departments in an enterprise can act as different tenants using the same API Manager instance and they can have different workflows. Also, an enterprise can combine WSO2 API Manager and WSO2 EI (Enterprise Integrator) to provide API Management As a Service to the clients. In this case, each client is a separate enterprise represented by a separate tenant. In both cases, the authority to approve business operations (workflows) resides within a tenant's space.

To allow different tenants to define their own custom workflows without editing configuration files, the API Manager provides configuration in tenant-specific locations in the registry, which you can access through the UI.

The topics below explain how to deploy a BPEL/human task using WSO2 EI and how to point them to services deployed in the tenant spaces in the API Manager.

Deploying a BPEL and a HumanTask for a tenant

Only the users registered in the EI can deploy BPELs and human tasks in it. Registration adds you to the user store in the EI. In this guide, the API Manager and EI use the same user store and all the users present in the EI are visible to the API Manager as well. This is depicted by the diagram below:
Follow the steps below to deploy a BPEL and a human task for a tenant in the API Manager:

**Sharing the user/permission stores with the EI and API Manager**

1. Create a database for the shared user and permission store as follows:

   ```
   mysql> create database workflow_ustore;
   Query OK, 1 row affected (0.00 sec)
   ```

   Make sure you copy the database driver (in this case, mysql driver) to the `/repository/components/lib` folder before starting each server.

2. Run the `<APIM_HOME>/dbscripts/mysql.sql` script (the script may vary depending on your database type) on the database to create the required tables.

   From WSO2 Carbon Kernel 4.4.6 onwards there are two MySQL DB scripts available in the product distribution. Click here to identify as to which version of the MySQL script to use.

3. Open the `<APIM_HOME>/repository/conf/datasources/master-datasources.xml` and create a datasource pointing to the newly created database. For example,
<datasource>
    <name>USTORE</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/ustore</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://127.0.0.1:3306/workflow_ustore?autoReconnect=true&amp;relaxAutoCommit=true</url>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

4. Repeat step 3 in the EI as well.
5. Point the datasource name in `<APIM_HOME>/repository/conf/user-mgt.xml to the new datasource. (note that the user store is configured using the `<UserStoreManager>` element).

If you already have a user store such as the IDAP in your environment, you can point to it from the `user-mgt.xml` file, instead of the user store that we created in step1.

In the following example, the same JDBC user store (that is shared by both the API Manager and the EI) is used as the permission store as well:

```
<Configuration>
    <AddAdmin>true</AddAdmin>
    <AdminRole>admin</AdminRole>
    <AdminUser>
        <UserName>admin</UserName>
        <Password>admin</Password>
    </AdminUser>
    <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role sees the registry root -->
    <Property name="dataSource">jdbc/ustore</Property>
</Configuration>
```

6. Repeat step 5 in the EI as well.

Sharing the data in the registry with the EI and API Manager

To deploy BPELs in an API Manager tenant space, the tenant space should be accessible by both the EI and API Manager, and certain tenant-specific data such as key stores needs to be shared with both products. Follow the steps below to create a registry mount to share the data stored in the registry:

1. Create a separate database for the registry:

   ```
   mysql> create database workflow_regdb;
   Query OK, 1 row affected (0.00 sec)
   ```

2. Run the `<APIM_HOME>/dbscripts/mysql.sql` script (the script may vary depending on your database type) on the database to create the required tables.

3. Create a new datasource in `<APIM_HOME>/repository/conf/datasources/master-datasources.xml as done before:`
4. Add the following entries to \texttt{<APIM_HOME>/repository/conf/registry.xml}:

```xml
<dbConfig name="sharedregistry">
  <dataSource>jdbc/regdb</dataSource>
</dbConfig>

<remoteInstance url="https://localhost:9443/registry">
  <id>mount</id>
  <dbConfig>sharedregistry</dbConfig>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/_system</registryRoot>
</remoteInstance>

<!-- This defines the mount configuration to be used with the remote instance and the target path for the mount -->
<mount path="//_system/config" overwrite="true">
  <instanceId>mount</instanceId>
  <targetPath>/_system/nodes</targetPath>
</mount>
<mount path="//_system/governance" overwrite="true">
  <instanceId>mount</instanceId>
  <targetPath>/_system/governance</targetPath>
</mount>
```

5. Repeat the above three steps in the EI as well.

**Creating a BPEL**

In this section, you create a BPEL that has service endpoints pointing to services hosted in the tenant's space. This example uses the Application Creation workflow.

1. Set a port offset of 2 to the EI using the \texttt{<EI_HOME>/repository/conf/carbon.xml} file. This prevents any port conflicts when you start more than one WSO2 products on the same server.
2. Log in to the API Manager's management console (\texttt{https://localhost:9443/carbon}) and create a tenant using the Configure -> Multitenancy menu.
3. Create a copy of the BPEL located in `<APIM_HOME>/business-processes/application-creation/BPEL`.
4. Extract the contents of the new BPEL archive.
5. Copy `ApplicationService.epr` and `ApplicationCallbackService.epr` from `<APIM_HOME>/business-processes/epr` folder to the folder extracted before. Then, rename the two files as `ApplicationService-Tenant.epr` and `ApplicationCallbackService-Tenant.epr` respectively.
6. Open `ApplicationService-Tenant.epr` and change the `wsa:Address` to `http://localhost:9765/services/t/<tenant_domain>` and add the tenant admin credentials.

   In a distributed setup, the `ApplicationService-Tenant.epr`'s `wsa:Address` should point to the proxy/load balancer of Enterprise Integrator (EI cluster) (http://<BPS_LB_hostname_here>/services/t/<tenant_domain>/ApplicationService). Also, the `ApplicationCallbackService-Tenant.epr`'s `wsa:Address` should point to APIM cluster's Workflow Callback service endpoint. This is normally deployed at the gateway nodes. The `wsa:Address` should point to the gateway nodes (https://<API_gateway_LB_hostname_here>/services/WorkflowCallbackService) and the user credentials which grant access to that service should be used.

7. Point the `deploy.xml` file of the extracted folder to the new .epr files provided in the BPEL archive. For example,

   ```xml
   <invoke partnerLink="AAPL">
     <service name="applications:ApplicationService" port="ApplicationPort">
       <endpoint xmlns="http://wso2.org/bps/bpel/endpoint/config" endpointReference="ApplicationService-Tenant.epr"></endpoint>
     </service>
   </invoke>

   <invoke partnerLink="CBPL">
     <service name="callback.workflow.apimgt.carbon.wso2.org:WorkflowCallbackService" port="WorkflowCallbackServiceHttpsSoap11Endpoint">
     </service>
   </invoke>
   ```

8. Zip the content and create a BPEL archive in the following format:
9. Log into the EI as the tenant admin and upload the BPEL.

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST, url:/carbon/admin/login_action.jsp, error:required token is missing from the request)
```

To avoid this issue open `<EI_HOME>/repository/conf/tomcat/catalina-server.xml` and change the compression="on" to compression="off" in Connector configuration and restart the EI.

Creating a Tenant for Authentication

Step 1: Create a registry resource in the tenant's configuration registry

1. Start the EI server if it is not started already.
2. Navigate to Registry->Browse in the Main menu of the management console and click on `/_system/config.`
3. Click on Entries>Add Resource and fill the form using the values listed below for guidance. See Adding a Resource for more information.

<table>
<thead>
<tr>
<th>Method</th>
<th>Name</th>
<th>Media Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Text Content</td>
<td>TaskCoordination</td>
<td>text/plain</td>
</tr>
</tbody>
</table>

4. Click Add to finish adding the resource.

---

Step 2: Create username and password registry properties and define credentials

1. Click on the registry resource you created (Task Coordination) found under the Entries section.

2. Add two new registry properties for the resource called "Username" and "Password", and define the tenant coordination user credentials. To do
this, click Properties>Add New Property and enter the following values. See Managing Properties for more information.

<table>
<thead>
<tr>
<th>Username Property</th>
<th>Password Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: username</td>
<td>Name: password</td>
</tr>
<tr>
<td>Value: (username value)</td>
<td>Value: (password value)</td>
</tr>
</tbody>
</table>

3. Click Add to finish adding the property.

Creating a human task

Similar to creating a BPEL, create a HumanTask that has service endpoints pointing to services hosted in the tenant's space.

2. Edit the SOAP service port-bindings in ApplicationApprovalTaskService.wsdl. For example,

   ```xml
   <wsdl:service name="ApplicationService">
     <wsdl:port name="ApplicationPort" binding="tns:ApplicationSoapBinding">
       <soap:address location="http://localhost:9765/services/t/<tenant_domain>/ApplicationService" />
     </wsdl:port>
   </wsdl:service>
   
   <wsdl:service name="ApplicationReminderService">
     <wsdl:port name="ApplicationReminderPort" binding="tns:ApplicationSoapBindingReminder">
       <soap:address location="http://localhost:9765/services/t/<tenant_domain>/ApplicationReminderService" />
     </wsdl:port>
   </wsdl:service>
   
   <wsdl:service name="ApplicationServiceCB">
     <wsdl:port name="ApplicationPortCB" binding="tns:ApplicationSoapBindingCB">
       <soap:address location="http://localhost:9765/services/t/<tenant_domain>/ApplicationServiceCB" />
     </wsdl:port>
   </wsdl:service>
   
   In a distributed setup, the above addresses should be changed to point to the EI proxy/loadbalancer. A sample is shown below.
   ```

   ```xml
   <soap:address location="http://<BPS_LB_hostname_here>/services/t/<tenant_domain>/ApplicationServiceCB" />
   
   3. Create the HumanTask archive by zipping all the extracted files. When creating the human task, make sure all the files are at the top level of the zip.
4. Log into the EI as the tenant admin and upload the HumanTask.
5. Log into the API Manager's management console as the tenant admin and select Resources > Browse menu.
6. Go to the /_system/governance/apimgt/applicationdata/workflow-extensions.xml in the registry and change the service endpoint as a tenant-aware service URL (e.g., http://localhost:9765/services/t/<tenant_domain>/ApplicationApprovalWor
kFlowProcess). Also set the credentials as the tenant admin's credentials of the ApplicationCreationWSWorkflowExecutor file. For example,

```java
<WorkflowExecutor xmlns="urn:wsdd:wsdl:1.1">
    <Process kFlowProcess="ApplicationCreation">
        <!-- Application creation workflow details -->
    </Process>
</WorkflowExecutor>
```

Be sure to disable the SimpleWorkflowExecutor and enable the ApplicationCreationWSWorkflowExecutor.

### Testing the workflow

You have now completed configuring the Application Creation workflow for a tenant. Whenever a tenant user logs in to the tenant store and create an application, the workflow will be invoked. You log in to the Admin Portal ([https://<Server Host>:9443/admin](https://<Server Host>:9443/admin)) as the tenant admin and browse Application Creation menu to see all approval tasks have been created for newly created applications.

![Approval Tasks](https://via.placeholder.com/150)

### Configuring Workflows in a Cluster

If you are working in a clustered API Manager setup with the API Store, Publisher, Gateway and Key Manager in separate servers, do the workflow configurations that are discussed in the previous topics in the API Store node. In addition, do the following configurations.

In this guide, you access the Admin Portal ([https://<Server Host>:9443/admin](https://<Server Host>:9443/admin)) Web application using the same node as the API Publisher. This is recommended because workflow management is an administrative task and is meant to reside within a private network as the Publisher. Typically, the Admin Portal from the same user store as the API Manager. Therefore, you can use the Admin Portal residing in the Publisher node instead of having it separately. This eliminates the need for a dedicated workflow management node. You need a dedicated node if the Admin Portal users reside in a separate user store.

1. If you want to change the user roles that can access the Admin Portal, open the `<APIM_HOME>/repository/deployment/server/jaggery`
1. File that is in the node from where you access the Admin Portal (the API Publisher node in this example) and change its Allowed Roles parameter. You can add multiple user roles as a comma-separated list.

2. By default, workflow related configuration files have the port of the Business Process Server with an offset of 2. If you set up the BPS with a different port offset, change the workflow server URLs in the site.json file accordingly.

3. Point the <Address> sub element of the <endpoint> element to the API Store node in the <APIM_HOME>/repository/deployment/serve r/synapse-configs/default/proxy-services/WorkflowCallbackService.xml file of the API Store node.

4. Add the IP address and the port of the API Store to the <Address> element of the .epr file of the workflow that you configure. You can find the .epr file by the name of the workflow in the <APIM_HOME>/business-processes/epr folder. For example, <APIM_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.epr.

5. Go to the <APIM_HOME>/business-processes/<workflow name>/BPEL folder and unzip the file that is there by the name of the workflow. For example, <APIM_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip.

6. Go inside the unzipped folder and do the following:

<table>
<thead>
<tr>
<th>Action</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open the ApprovalTask WSDL file and point</td>
<td>In the &lt;APIM_HOME&gt;/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0/ApprovalTask.wsdl</td>
</tr>
<tr>
<td>the address elements of the server where the BPEL runs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;wsdl:service name=&quot;UserApprovalService&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;wsdl:port name=&quot;UserApprovalPort&quot; binding=&quot;tns:UserApprovalBinding&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;soap:address location=&quot;<a href="http://localhost:9765/services/UserApprovalService">http://localhost:9765/services/UserApprovalService</a>&quot; /&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/wsdl:port&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/wsdl:service&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;wsdl:service name=&quot;UserApprovalServiceCB&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;wsdl:port name=&quot;UserApprovalPortCB&quot; binding=&quot;tns:UserApprovalBindingCB&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;soap:address location=&quot;<a href="http://localhost:9765/services/UserApprovalServiceCB">http://localhost:9765/services/UserApprovalServiceCB</a>&quot; /&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/wsdl:port&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/wsdl:service&gt;</td>
</tr>
</tbody>
</table>

Note that all workflow process services of the BPS run on port 9765 because you changed its default port (9763)
Open the Callbac
kService WSDL
file and point the
address elements
to the API Store
node in NIO port.

In the `<APIM_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0/WorkflowCallbackService.wsdl`

```
<wsdl:service name="WorkflowCallbackService">
  <wsdl:port name="WorkflowCallbackServiceHttpsSoap11Endpoint" binding="ns:WorkflowCallbackServiceSoap11Binding">
    <soap:address location="https://localhost:8243/services/WorkflowCallbackService.WorkflowCallbackServiceHttpsSoap11Endpoint"/>
  </wsdl:port>
  <wsdl:port name="WorkflowCallbackServiceHttpSoap11Endpoint" binding="ns:WorkflowCallbackServiceSoap11Binding">
  </wsdl:port>
  <wsdl:port name="WorkflowCallbackServiceHttpsSoap12Endpoint" binding="ns:WorkflowCallbackServiceSoap12Binding">
    <soap12:address location="https://localhost:8243/services/WorkflowCallbackService.WorkflowCallbackServiceHttpsSoap12Endpoint"/>
  </wsdl:port>
  <wsdl:port name="WorkflowCallbackServiceHttpSoap12Endpoint" binding="ns:WorkflowCallbackServiceSoap12Binding">
    <soap12:address location="http://localhost:8280/services/WorkflowCallbackService.WorkflowCallbackServiceHttpSoap12Endpoint"/>
  </wsdl:port>
  <wsdl:port name="WorkflowCallbackServiceHttpsEndpoint" binding="ns:WorkflowCallbackServiceHttpBinding">
  </wsdl:port>
  <wsdl:port name="WorkflowCallbackServiceHttpEndpoint" binding="ns:WorkflowCallbackServiceHttpBinding">
  </wsdl:port>
</wsdl:service>
```

7. Go to the `<APIM_HOME>/business-processes/<workflow name>/HumanTask` folder and unzip the file that is there by the name of
the workflow. For example, `<APIM_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip`.
8. Go inside the unzipped folder and do the following:

<table>
<thead>
<tr>
<th>Action</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you changed the default admin role, open the ApprovalTask HT file and apply the changes there.</td>
<td>Change the admin instances in <code>&lt;APIM_HOME&gt;/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0/UserApprovalTask.ht</code> file. Here's an example, assuming that the new admin role is apimadmin.</td>
</tr>
</tbody>
</table>

```
<htd:peopleAssignments>
  <htd:potentialOwners>
    <htd:from logicalPeopleGroup="admin">
      <htd:argument name="role">apimadmin</htd:argument>
    </htd:from>
  </htd:potentialOwners>
</htd:peopleAssignments>
```
Open the ApprovalTask WSDL file and point the two address elements to the Business Process Server node.

In the `<APIM_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0/UserApprovalTask.wsdl` file:

```xml
<wSDL:service name="UserApprovalService">
  <wSDL:port name="UserApprovalPort" binding="tns:UserApprovalBinding">
    <soap:address location="http://localhost:9765/services/UserApprovalService" />
  </wSDL:port>
</wSDL:service>
<wSDL:service name="UserApprovalServiceCB">
  <wSDL:port name="UserApprovalPortCB" binding="tns:UserApprovalBindingCB">
    <soap:address location="http://localhost:9765/services/UserApprovalServiceCB="/>
  </wSDL:port>
</wSDL:service>
```

**Note** that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

### Changing the Default User Role in Workflows

The default user role in the workflow configuration files is the admin role. If you change this to something else, you need to change the following files:

1. Change the credentials in the `.epr` files of the `<BPS_HOME>/repository/conf/epr` folder.
2. Change the credentials in workflow configurations in API Manager Registry (`/_system/governance/apimgt/applicationdata/workflow-extensions.xml`)
3. Point the same database that has the permissions used by the API Manager to the BPS.
4. Share the LDAP, if it exists.
5. If you change the default user role, change the `.ht` file of the relevant human task.

### Cleaning Up Workflow Tasks

When workflow extensions are enabled using default workflow executors, they create respective approval tasks in WSO2 Business Process Server (BPS). Each of these tasks are visible to administrators in the Admin Portal. The administrator has the option to accept or reject each of the requests made by other users. At the same time, users have the option to delete the application, subscription or key they created before the administrator accepts or rejects their requests. This leaves unnecessary approval requests in the Admin Portal, which can confuse the administrator.

API Manager provides a task clean up feature to prevent deleted items from showing up in the Admin Portal. The `WorkflowExecutor` class is introduced with the `cleanUpPendingTask(String workflowExtRef)` method, which is triggered by application or subscription deletion. This method implements the logic to notify WSO2 BPS that a task with the `workflowExtRef` ID has been deleted.

The BPEL process in WSO2 BPS should contain a cancel event to support process cancellation. Each BPEL process should support correlation and event cancellation in order to successfully cleanup unnecessary tasks. For more information on BPEL correlation, see Process Correlation and the BPEL Correlation Guide.

The final BPEL should have a design similar to the following diagram,
Follow the steps below to test this out.

This example assumes that workflows are enabled for application creation, application registration and subscription creation.

1. Log in to the API Store and create two new applications.
2. Log in to the Admin Portal (https://<Server Host>:9443/admin) and approve the creation of one application.
3. In the API Store, subscribe an API to the approved application.
4. Generate production and/or sandbox key(s) for the approved application.
5. Check the pending approval tasks in the Admin Portal. You see tasks pending for application creation, application registration and subscription creation.
6. Delete the items you created from the API Store and notice that the respective administrator approval tasks are removed.
7. If the application with pending subscription and key generation approvals is deleted from the API Store, all the pending subscription and key generation approval tasks are deleted for that application.

**Adding an API State Change Workflow**

This section explains how to add a custom workflow to control the API state changes in the API Manager. First see Workflow Extensions for more information on different types of workflow executors. For more details on API states see API Lifecycle.

**Configuring the Business Process Server**

1. Download WSO2 Enterprise Integrator.
2. Set an offset of 2 to the default BPS port in `<EI_HOME>/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see Changing the Default Ports with Offset.
Tip: If you change the EI port offset to a value other than 2 or run the API Manager and EI on different machines (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM_HOME>/business-processes folder with the new port (9763 + port offset).


If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential cross-site request forgery (CSRF) attack thwarted (user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST, uri:/carbon/admin/login_action.jsp, error:required token is missing from the request)
```

To avoid this issue open <EI_HOME>/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the EI.

Tip: If you change the EI port (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM_HOME>/business-processes folder with the new port (9763 + port offset).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

4. Select Processes > Add > BPMN and upload the <APIM_HOME>/business-processes/api-state-change/APIStateChangeApprovalProcess.bar file to EI.

**Configuring the Business Process Server**

2. Set an offset of 2 to the default BPS port in <BPS_HOME>/repository/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see Changing the Default Ports with Offset.

```xml
<Offset>2</Offset>
```

Tip: If you change the BPS port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the hostname to a different value than localhost), you do the following:

- Search and replace the value 9765 in all the files (.epr) inside <APIM_HOME>/business-processes folder with the new port (9763 + port offset).


If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.
4. Select Processes > Add > BPMN and upload the `<APIM_HOME>/business-processes/api-state-change/APIStateChangeApprovalProcess.bar` file to BPS.

**Configuring the API Manager**

1. Open `<APIM_HOME>/repository/conf/api-manager.xml` and set `<Enabled>true</Enabled>` in the `<WorkflowConfigurations>` section.
2. Change the `<ServerUrl>` if you have configured the BPS/EI to run on a different port offset.

**Engaging the WS Workflow Executor in the API Manager**

First, enable the API state change workflow.

1. Log in to the APIM management console (https://<Server Host>:9443/carbon) and select Browse under Resources.

2. Go to the `/system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor.

```xml
<WorkflowExtensions>
    ....
    <!-- Publisher related workflows -->
    <APIStateChange
        executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeSimpleWorkflowExecutor"/>
    <APIStateChange
        executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeWSWorkflowExecutor">
        <Property name="processDefinitionKey">APIStateChangeApprovalProcess</Property>
        <Property name="stateList">Created:Publish,Published:Block</Property>
    </APIStateChange>
    ....
</WorkflowExtensions>
```

You have now engaged the API WS Workflow. The default configuration is set for **Created to Publish** and **Published to Block** state changes.

**To avoid this issue** open `<BPS_HOME>/repository/conf/tomcat/catalina-server.xml` and change the compression="on" to compression="off" in Connector configuration and restart the BPS.
See Advanced Configurations for information on configuring more state changes.

When enabling the workflow features, make sure you import the certificate of the API Manager into the client-truststore located inside the `<BPS_HOME>/repository/resources/security` directory.

3. Log in to the API Publisher (https://<Server Host>:9443/publisher) and publish an API. See Create and Publish an API. A message related to the publish workflow will be displayed because the workflow is enabled for Created to Publish state change.

![API Publisher](image1)

Note that the Save & Publish button will be disabled until the workflow task is completed or deleted.

![Lifecycle state change request has been submitted](image2)

4. You can revoke the state change by clicking Delete Task in the Lifecycle tab.

![WorldBank - 1.0.0](image3)

5. Log in to the Admin Portal (https://<Server Host>:9443/admin) and click API State Change to see the list of tasks awaiting approval.

![Approval Tasks](image4)
Click **Assign to Me** to approve the task. Select **Approve** and click **Complete** to resume and complete the API state change.

### Approval Tasks

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Suspended</th>
<th>Created On</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Approval request for API State change action 'Publish' from 'Created' state for the API 'WorldBank': 1.0.9 by admin</td>
<td>false</td>
<td>Wed Nov 30 2016 14:01:45 GMT+0300 (IST)</td>
</tr>
</tbody>
</table>

Show 10 entries, Showing 1 to 1 of 1 entries

Using WSO2 EI

**Using WSO2 BPS**

### Configuring the BPS for tenants

1. Log in to the EI with the credentials of the tenant. Select **Processes > Add > BPMN** and upload the `<APIM_HOME>/business-processes/api-state-change/APIStateChangeApprovalProcess.bar` file to EI.
3. To engage the WS Workflow Executor, log in to the admin console using the credentials of the tenant and repeat step 2 from **Engaging the WS Workflow Executor in the API Manager**.

### Configuring the BPS for tenants

1. Log in to the BPS with the credentials of the tenant. Select **Processes > Add > BPMN** and upload the `<APIM_HOME>/business-processes/api-state-change/APIStateChangeApprovalProcess.bar` file to BPS.
2. Copy the `<BPS_HOME>/repository/deployment/server/webapps/bpmn.war` web app to `<BPS_HOME>/repository/tenants/<tenant_id>/webapps`.
3. To engage the WS Workflow Executor, log in to the admin console using the credentials of the tenant and repeat step 2 from **Engaging the WS Workflow Executor in the API Manager**.

### Advanced Configurations

Given below are the configurations that can be changed by editing `<APIM_HOME>/repository/conf/api-manager.xml`

```xml
<WorkflowConfigurations>
  <Enabled>true</Enabled>
  <ServerUrl>https://localhost:9445/bpmn</ServerUrl>
  <ServerUser>${admin.username}</ServerUser>
  <ServerPassword>${admin.password}</ServerPassword>
  <WorkflowCallbackAPI>https://localhost:${mgt.transport.https.port}/api/am/publisher/v0.10/workflows/update-workflow-status</WorkflowCallbackAPI>
  <TokenEndPoint>https://localhost:${https.nio.port}/token</TokenEndPoint>
  <DCREndPoint>https://localhost:${mgt.transport.https.port}/client-registration/v0.10/register</DCREndPoint>
  <DCREndPointUser>${admin.username}</DCREndPointUser>
  <DCREndPointPassword>${admin.password}</DCREndPointPassword>
</WorkflowConfigurations>
```

The elements of the above configuration are explained below.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables the Admin Portal to approve state change tasks.</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>ServerUrl</td>
<td>The URL of the BPMN server.</td>
</tr>
<tr>
<td>ServerUser</td>
<td>User accessing the BPMN REST API.</td>
</tr>
<tr>
<td>ServerPassword</td>
<td>Password of the user accessing the BPMN REST API.</td>
</tr>
<tr>
<td>WorkflowCallbackAPI</td>
<td>The REST API invoked by the BPMN to complete the workflow.</td>
</tr>
<tr>
<td>TokenEndpoint</td>
<td>The API call to generate the access token is passed to the BPMN process. Once the access token is received, it is used to call the workflow callback API.</td>
</tr>
<tr>
<td>DCREndPoint</td>
<td>Endpoint to generate OAuth application. This application is used by the BPMN process to generate the token.</td>
</tr>
<tr>
<td>DCREndPointUser</td>
<td>Endpoint user.</td>
</tr>
<tr>
<td>DCREndPointPassword</td>
<td>Endpoint password.</td>
</tr>
</tbody>
</table>

**Setting a DCREndPointUser**

Create a user with exclusive `apim:apiworkflow` scope permissions when setting a DCREndPointUser. Please avoid using super admin credentials. If super admin credentials are used, the created OAuth application will have all the permissions related to scopes in the other REST APIs. Follow the steps below to create a user with `apim:apiworkflow` scope permissions:

1. Log in to APIM management console (https://<Server Host>:9443/carbon) and create a role named `workflowCallbackRole`. Set create and publisher or subscriber permissions to this role.
2. Go to Resources and click Browse. Go to `/_system/config/apimgt/applicationdata/tenant-conf.json` and update the role related to `apim:api_workflow` scope with the newly created role.

```
... {
  "Name": "apim:api_workflow",
  "Roles": "workflowCallbackRole"
...}
```

3. Assign this role to a user.
4. Update `<DCREndPointUser>` and `<DCREndPointPassword>` with this user's credentials.

For more details on how to create users and roles see managing users and roles.

The configurations that can be changed by editing the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` are given below.

**Simple WorkFlow**

```
<APIStateChange executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeSimpleWorkflowExecutor"/>
```

**WS WorkFlow**

```
<APIStateChange executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeWSWorkflowExecutor">
  <Property name="processDefinitionKey">APIStateChangeApprovalProcess</Property>
  <Property name="stateList">Created:Publish,Published:Block</Property>
</APIStateChange>
```

The elements of the above configuration are explained below.
Writing Custom Handlers

This section introduces handlers and using an example, explains how to write a custom handler:

- Introducing Handlers
- Writing a custom handler
- Engaging the custom handler

Introducing Handlers

When an API is created, a file with its synapse configuration is added to the API Gateway. You can find it in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration. You find the default handlers in any API's Synapse definition as shown below.

```xml
<handlers>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler">
    <property name="apiImplementationType" value="ENDPOINT"/>
  </handler>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
    <property name="id" value="A"/>
    <property name="policyKeyResource" value="gov:/apimgt/applicationdata/res-tiers.xml"/>
    <property name="policyKeyApplication" value="gov:/apimgt/applicationdata/app-tiers.xml"/>
  </handler>
  <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
  <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler">
    <property name="configKey" value="gov:/apimgt/statistics/ga-config.xml"/>
  </handler>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
</handlers>
```

Let's see what each handler does:

- **CORSRequestHandler**: Sets the CORS headers to the request and executes the CORS sequence mediation logic. This handler is thereby responsible for returning the CORS headers from the gateway or routing the requests to the backend and letting the backend send the CORS headers.
- **APIAuthenticationHandler**: Validates the OAuth2 bearer token used to invoke the API. It also determines whether the token is of type Production or Sandbox and sets MessageContext variables as appropriate.
- **APIThrottleHandler**: Throttles requests based on the throttling policy specified by the `policyKey` property. Throttling is applied both at the application level as well as subscription level.
- **APIMgtUsageHandler**: Publishes events to WSO2 Stream Processor (WSO2 SP) for collection and analysis of statistics. This handler only comes into effect if API usage tracking is enabled. See the Working with Analytics section for more information.
- **APIMgtGoogleAnalyticsTrackingHandler**: Publishes events to Google Analytics. This handler only comes into effect if Google analytics tracking is enabled. See Integrating with Google Analytics for more information.
- **APIManagerExtensionHandler**: Triggers extension sequences. By default, the extension handler is listed at last in the handler chain, and therefore is executed last. You cannot change the order in which the handlers are executed, except the extension handler. To configure the API Gateway to execute extension handler first, uncomment the `<ExtensionHandlerPosition>` section in the `<APIM_HOME>/repository/conf/api-manager.xml` file and provide the value top. This is useful when you want to execute your own extensions before our default handlers in situations like doing additional security checks such as signature verification on access tokens before executing the default security handler. See Adding Mediation Extensions.
Using APILogMessageHandler

Message logging is handled by APIManagerExtensionHandler. APILogMessageHandler is a sample handler that comes with WSO2 API Manager that can be used for logging.

Why are logs removed from APIManagerExtensionHandler?

The primary purpose of ExtensionHandler is handling extensions to mediation and not for logging messages. When the logs are also included in ExtensionHandler, there's a limitation to improve the ExtensionHandler for developing features because it breaks the logs.

For example, When the ExtensionHandler moves to the top of the handlers set, most of the logs print null values since the handler runs before the APIAuthenticationHandler. Therefore, the logs are removed from the extension handler and APILogMessageHandler introduced as a sample.

To achieve logging requirements, this handler is not the only approach and with custom sequences also it is possible to log messages using the Log Mediator.

In order to enable logging by invoking APILogMessageHandler, follow the steps below.

To enable Message Logging per API:

1. Open the synapse Configuration of the API located in <APIM_HOME>/repository/deployment/server/synapse-configs/default/api directory and add below handler before </Handlers>.

   <handler class="org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageHandler"/>

2. Copy the following code into the <APIM_HOME>/repository/conf/log4j.properties file to enable printing DEBUG logs.

   log4j.logger.org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageHandler = DEBUG

3. Restart API Manager.

To enable Message Logging into APIS created from publisher automatically:

1. Open the <APIM_HOME>/repository/resources/api_templates/velocity_template.xml file and copy the following handler before </Handlers>.

   <handler class="org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageHandler"/>

2. Restart API Manager.

To perform analytics with the logs, see Analyzing the Log Overview.

Writing a custom handler

The outcome of using a Class Mediator vs. a Synapse Handler are very similar. However, when using a custom handler you need to maintain a customized velocity template file that needs to be manually merged when you upgrade your product to a newer version. Therefore, it is recommended to use custom Handlers when you wish to specify the exact order of execution of JARs as this can not be done with Mediators.

Custom Handler is a way of extending API Manager which the product offer to change the API flow through the API Gateway. What is happening in custom handler can be decided by the code you are writing to extend the product. It can be adding extra security, logging database invocation or something else. This custom handler must extend the org.apache.synapse.rest.AbstractHandler class and implement handleRequest() and handleResponse() methods.

Let's see how you can write a custom handler and apply it to the API Manager. In this example, we extend the authentication handler. Make sure your custom handler name is not the same as the name of an existing handler.

WSO2 API Manager provides the OAuth2 bearer token as its default authentication mechanism. A sample implementation is here. Similarly, you can extend the API Manager to support any custom authentication mechanism by writing your own authentication handler class.

Given below is an example implementation. Please find the complete project archive org.wso2.carbon.test.authenticator.zip. You can download, unzip and build the project using maven and Java 7/8.
package org.wso2.carbon.test;

import org.apache.synapse.MessageContext;
import org.apache.synapse.core.axis2.Axis2MessageContext;
import org.apache.synapse.rest.AbstractHandler;
import java.util.Map;

public class CustomAPIAuthenticationHandler extends AbstractHandler {

    public boolean handleRequest(MessageContext messageContext) {
        try {
            if (authenticate(messageContext)) {
                return true;
            }
        } catch (APISecurityException e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean handleResponse(MessageContext messageContext) {
        return true;
    }

    public boolean authenticate(MessageContext synCtx) throws APISecurityException {
        Map headers = getTransportHeaders(synCtx);
        String authHeader = getAuthorizationHeader(headers);
        if (authHeader.startsWith("userName")) {
            return true;
        }
        return false;
    }

    private String getAuthorizationHeader(Map headers) {
        return (String) headers.get("Authorization");
    }

    private Map getTransportHeaders(MessageContext messageContext) {
        return (Map) ((Axis2MessageContext) messageContext).getAxis2MessageContext().getProperty(org.apache.axis2.context.MessageContext.TRANSPORT_HEADERS);
    }
}

### Engaging the custom handler

1. Build the custom authenticator code downloaded previously, and copy the resulting jar to `<API-M_HOME>/repository/components/dropins` directory.
2. Engage the custom handler using the API template as explained below:

   You can engage a custom handler to all APIs at once or only to selected APIs. To engage a custom handler to APIs, you need to add the custom handler with its logic in the `<APIM_HOME>/repository/resources/api_templates/velocity_template.xml` file.

   **It is not recommended to update the API source code via the source view UI or file system when engaging a custom handler to selected APIs, because the customizations get overridden by the publisher updates.**

   For example, the following code segment adds the custom authentication handler that you wrote earlier to the `velocity_template.xml` file while making sure that it skips the default `APIAuthenticationHandler` implementation:
You can select to which API(s) you need to engage the handler. Given below is an example of adding only the CustomAPIAuthenticationHandler to the sample PizzaShackAPI.

```xml
<handlers xmlns="http://ws.apache.org/ns/synapse">
    <handler class="org.wso2.carbon.sample.auth.CustomAPIAuthenticationHandler"/>
</handlers>
```

3. Restart the API Manager server.

Adding Mediation Extensions

This tutorial uses the WSO2 API Manager Tooling Plug-in.

The API Gateway has a default mediation flow, which you can extend by adding custom mediation sequences. In API Manager there are 3 default sequences engaged as in, out and fault. You create a custom mediation sequence either manually or using a tool such as the WSO2 API Manager Tooling Plug-in, and then engage it per API or globally to all APIs of a specific tenant. With custom mediation sequences you can modify the default mediation flow for different usabilities according to your requirement. Log the mediation flow, execute operations on Message context properties, to customize, format the requests and responses are some of them.

- Default mediation flow
- Creating per-API extensions
- Creating global extensions

- The following mediators are not usable within custom sequences because they are not supported by the API Gateway.
  - Call mediator in non-blocking mode
  - Send mediator
- When using the Loopback mediator, it is mandatory to set the following property before defining the Loopback mediator in the custom mediator sequence in the following manner.
Default mediation flow

You cannot dynamically construct the back-end endpoint of an API using the address endpoints in the WSO2 API Manager. To achieve the requirement of a dynamic endpoint, you can use the default endpoint instead. The default endpoint sends the message to the address specified in the To header. The To header can be constructed dynamically. For example,

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="default-endpoint-seq">
  <property name="service_ep" expression="fn:concat('http://jsonplaceholder.typicode.com/',
    'posts/')"/>
  <header name="To" expression="get-property('service_ep')"/>
</sequence>
```

In this example, you have constructed the service_ep property dynamically and assigned the value of this property to the To header. The default endpoint sends the message to the address specified in the To header, in this case, http://jsonplaceholder.typicode.com/posts/. For more details about working with dynamic endpoints, see Dynamic Endpoints.

Adding a non-blocking send operation

In this example, the Send mediator in a proxy service using the VFS transport is transferring a file to a VFS endpoint. VFS is a non-blocking transport by default, which means a new thread is spawned for each outgoing message. The Property mediator added before the Send mediator removes the ClientAPINonBlocking property from the message to perform the mediation in a single thread. This is required when the file being transferred is large and you want to avoid out-of-memory failures.

```xml
<inSequence>
  <property name="ClientApiNonBlocking" value="true" scope="axis2" action="remove"/>
  <send>
    <endpoint name="FileEpr">
      <address uri="vfs:file:///home/shammi/file-out"/>
    </endpoint>
  </send>
</inSequence>
```

Creating per-API extensions

- Create and upload using the WSO2 API Manager Tooling Plug-in
- Create and upload manually in the API Publisher
- Editing a mediation policy
- Create manually and save in the file system

Create and upload using the WSO2 API Manager Tooling Plug-in

The recommended way to engage a mediation extension sequence per API is to create a custom sequence using the WSO2 API Manager Tooling Plug-in, upload it via its APIM Perspective and then engage it using the API Publisher. The following tutorial demonstrates how to do this: Change the Default Mediation Flow of API Requests.

Create and upload manually in the API Publisher

You can also create a mediation sequence manually and upload it from the API Publisher itself. For instance, you can copy the above default mediation flow content into an XML file. In the Implement tab of the API, select the Enable Message Mediation check box and click the Upload In Flow or Upload Out Flow field (for the example above, it needs to be uploaded to the In flow). Once the file is uploaded, save and publish the API. When you invoke the API, the request is sent to the endpoint referred to in the To header.
Editing a mediation policy

If you want to edit an already uploaded mediation policy,

1. Select the mediation policy from the drop down list.
2. Click the download icon next to it, as shown below:

3. Edit the downloaded mediation XML file and re-upload it.
Create manually and save in the file system

Alternatively, you can name the mediation XML file in the pattern `<API_NAME>:v<VERSION>--<DIRECTION>` and save it directly in the following location:

- In the single-tenant mode, save the XML file in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory.
- In the multi-tenant mode, save the XML file in the tenant's synapse sequence folder. For example, if tenant id is 1, then save it in `<API_Gateway>/repository/tenants/1/synapse-configs/default/sequences` folder.

In the naming pattern, the `<DIRECTION>` can be In or Out. When it is In, the extension is triggered on the in-flow (request path) and when it is Out, the extension is triggered on the out-flow (response path). To change the default fault sequence, you can either modify the default sequence or write a custom fault sequence and engage it to APIs through the API Publisher.

An example synapse configuration of a per-API extension sequence created for the API `admin--TwitterSearch` version 1.0.0 is given below.

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="admin--TwitterSearch:v1.0.0--In">
  <log level="custom">
    <property name="TRACE" value="API Mediation Extension"/>
  </log>
</sequence>
```

You can copy this content into an XML file (e.g., `twittersearch_ext.xml`) and save it in the `<API_Gateway>/repository/deployment/server/synapse-configs/default/sequences` directory.

The above sequence prints a log message on the console whenever the TwitterSearch API is invoked.

**Creating global extensions**

You can also engage mediation extension sequences to all APIs of a specific tenant at once. To do that, simply create the XML with the naming pattern `WSO2AM--Ext--<DIRECTION>` and save it in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory.

An example synapse configuration of a global extension sequence is given below:

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="WSO2AM--Ext--In">
  <property name="Authentication" expression="get-property('transport', 'Authentication')"/>
  <property name="Authorization" expression="get-property('Authentication')" type="STRING"/>
  <property name="Authentication" scope="transport" action="remove" />
</sequence>
```

This custom sequence assigns the value of your basic authentication to Authorization header.

You can copy this content into an XML file (e.g., `global_ext.xml`) and save it in the `<API_Gateway>/repository/deployment/server/synapse-configs/default/sequences` directory.

When you invoke your REST API via a REST Client, configure that client to have a custom header (Authentication) for your basic authentication credential and configure the Authorization header to contain the bearer token for the API. When you send the Authentication and Authorization headers, the Gateway drops the Authorization header, converts the Authentication to Authorization headers and sends to the backend.

**Class Mediator** is one specific example of mediation extension. When creating a class mediator, we are allowed to write a Java class which extends the `org.apache.synapse.mediators.AbstractMediator` class.

This class implements the mediate() function which access the message context and provide the facility to customize the mediation flow of the API. Through that we can read properties of the message context into variables and perform operations.
package samples.mediators;

import org.apache.synapse.MessageContext;
import org.apache.synapse.mediators.AbstractMediator;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.soap.SOAPFactory;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import javax.xml.namespace.QName;

public class SimpleClassMediator extends AbstractMediator {
    private String variable1 = xxx;
    private static final Log log = LogFactory.getLog(SimpleClassMediator.class);

    public SimpleClassMediator() {}

    public boolean mediate(MessageContext mc) {
        // Do something useful..
        // Implementation of reading the property values of Message context and modifying request/logging properties
        return true;
    }

    public String getType() {
        return null;
    }

    public void setTraceState(int traceState) {
        traceState = 0;
    }

    public int getTraceState() {
        return 0;
    }

    public void setVariable1(String newValue) {
        variable1 = newValue;
    }

    public String getVariable1() {
        return variable1;
    }
}

Then we can export this class as a jar file and add as a library to <API-M_HOME>/repository/components/lib directory.

By referring this class with the fully qualified class name in a class mediator in the API as below, we can execute it in the insequence or outsequence of the API globally or per API as described above.

<class name="samples.mediators.SimpleClassMediator">
    <property name="propertyName" value="propertyValue"/>
    ...
</class>

If any properties are specified in the java class of the class mediator, the corresponding setter methods are invoked once on the class during initialization.
Writing Custom Grant Types

OAuth 2.0 authorization servers provide support for four main grant types according to the OAuth 2.0 specification. They also allow you to add custom grant types and extend the existing ones.

See Writing a Custom OAuth 2.0 Grant Type in the WSO2 identity Server documentation to implement custom grant types for the API Manager. Note that API Manager has already customized the Grant Type handlers for authorization_code, password, client_credentials and saml2-bearer grant types. If you require any additional functionality for these grant types, its advisable to extend the following grant handler implementations.

<table>
<thead>
<tr>
<th>Grant Type</th>
<th>Existing Handler Class (which can be extended if required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>authorization_code</td>
<td>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedAuthorizationCodeGrantHandler</td>
</tr>
<tr>
<td>password</td>
<td>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedPasswordGrantHandler</td>
</tr>
<tr>
<td>client_credentials</td>
<td>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedClientCredentialsGrantHandler</td>
</tr>
<tr>
<td>urn:ietf:params:oauth:grant-type:saml2-bearer</td>
<td>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedSAML2BearerGrantHandler</td>
</tr>
</tbody>
</table>

Extending Key Validation

In WSO2 API Manager (WSO2 API-M) versions prior to 1.9.0, the components were tightly coupled with the Key Manager and token validation was done by directly accessing the databases. However, from WSO2 API-M 1.9.0 onwards, you can plug different OAuth2 providers to the key validation. When you call an API providing an access token, the execution flows through the handlers specified in the API. Among them, the API authentication handler extracts the token from the header and calls APIKeyValidationService to get the token validated. Upon validating the token, the API Gateway receives APIKeyValidationInforDTO as the response, using which the rest of the operations are performed.

Before decoupling was done, the entire key validation process was executed inside a single method named validateKey(), which performed all the operations by running a single query. After decoupling, that single query was broken down into smaller parts by introducing KeyValidationHandler, which runs inside the validateKey() operation, providing a way to extend each step.

The KeyValidationHandler has four main operations that are executed in the following order:
**WSO2 API Manager, WSO2 Inc.**

- **validateToken** - Validates the token. The existing implementation should work for most cases.
- **validateSubscription** - Skips/changes the domain validation.
- **validateScopes** - Relaxes/reduces scope restrictions.
- **GenerateConsumerToken** - Creates different types of tokens.

The default implementation of the KeyValidationService is written in a way where you are able to complete the entire key validation flow only by extending the `getTokenMetaData()` method in the KeyManagerInterface.

However, there are situations where you need to customize the default key validation flow according to different requirements. In such situations, WSO2 API-M provides the facility to extend the KeyValidationHandler and its methods.

A few examples are listed below.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>You need to skip trivial steps, because its validation does not add value.</td>
<td>When creating a key via the API Store, the subscriber can specify which domains are allowed to make calls using a token granted against a particular consumer key. If this validation does not add any value, these trivial steps can be ignored and skipped by extending the KeyValidationHandler.</td>
</tr>
<tr>
<td></td>
<td>For another example scenario, see Skipping Role Validation for Scopes.</td>
</tr>
<tr>
<td>You need to avoid going into detail when validating scopes</td>
<td>Consider a situation where a scope is assigned to a resource and you only need to verify if the token that is used to access the API has at least one or more scopes defined for that API without going into much detail. This requirement can be achieved by extending the validateScope() method.</td>
</tr>
<tr>
<td>You need to send a different type of token instead of JSON Web Token (JWT) to pass details of API invocation to the backend</td>
<td>A JSON Web Token (JWT) is used to pass details of an API invocation to the backend. If a different type of token is required, you can extend the generateConsumerToken() method to achieve this purpose.</td>
</tr>
</tbody>
</table>

**Extending the Key Manager Interface**

In a typical WSO2 API Manager (WSO2 API-M) deployment, different components talk to the KeyManager interface to achieve different tasks. For instance -

- After creating an application in API store, subscribers would click on the generate button to register an application. At this point, the API store talks to the KeyManager to create an OAuth client and get the Consumer Key/Secret and the Application Access token.
- When the Gateway receives a request, it talks to KeyManager and get the token validated. The KeyManager checks if the token is active, and whether the token is usable to invoke the resource being accessed. If the token is valid, the KeyManager sends additional details about the token (i.e., the Throttling Tier for the subscription and Consumer key) to the Gateway in the response. In turn the Gateway uses these details to determine if the request should be passed to the backend or not.

Therefore, the KeyManager interface acts as the bridge between the OAuth Provider and WSO2 API Manager (WSO2 API-M).

Implement the KeyManager interface, which is a Java extension point in WSO2 API-M, when you are writing your own implementation to plug an external OAuth2 authorization server, which will act as the Key Manager. For this purpose uncomment and update the API Key Manager details under the `<APIKeyManager>` element and specify the custom class implementation under the `<KeyManagerClientImpl>` element.

```
<KeyManagerClientImpl>org.wso2.carbon.mit.OpenIDClientImpl</KeyManagerClientImpl>
```

The following are the methods that the KeyManager interface uses to carry out operations.

- **createApplication** - Creates a new OAuth application in the Authorization Server.
- **updateApplication** - Updates an OAuth application.
- **retrieveApplication** - Retrieves an OAuth application.
- **getNewApplicationAccessToken** - The Store calls this method to get a new application Access Token. This method is called when getting the token for the first time and when the Store needs to refresh the existing token.
- **getTokenMetaData** - Gets details about an access token.
- **getKeyManagerConfiguration** - Gets Key Manager implementation from api-manager.xml file.
- **buildAccessTokenRequestFromJSON** - This method will parse the JSON input and add those additional values to the Access Token Request. If it is needed to pass parameters in addition to those specified in the AccessTokenRequest, those parameters can be provided in the JSON input.
- **mapOAuthApplication** - You need to use this method when creating an OAuth application in semi-manual mode when you have a consumer key and secret already generated from a Key Manager and you need to map the key and secret with the existing API-M application.
- **buildAccessTokenRequestFromOAuthApp** - This method creates an Access Token Request using the OAuth Application information. If the token request is null, this method creates a new object, else it modifies the provided Access Token request.
- **loadConfiguration**
Adding a New API Store Theme

A theme consists of UI elements such as logos, images, copyrights messages, landing page text, background colors etc. WSO2 API Store comes with a default theme. You can extend the existing theme by writing a new one or customising the existing one.

Folder structure of the API Store themes

The default theme of the API Store is named wso2. You find it inside the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/wso2` folder. If you do not have access to the file system, download the default theme from here.

The easiest way to create a new theme is to copy the files of an existing theme to a folder that is named after your new theme, and do the modifications you want to the files inside it. All themes have the same folder structure as shown below:

```
- themes
  - wso2
    - bower.json
    - css
    - images
    - js
    - libs
    - package.json
    - subthemes
    - templates
```

You can add a new theme as a main theme or a sub-theme.

- **A main theme** is saved inside the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes` directory.
- **A sub theme** is saved inside the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/<main-theme-directory>/subtheme` directory.

As a sub-theme is saved inside a main theme, it needs to contain only the files that are different from the main theme. Any file that you add inside the sub-theme overrides the corresponding files in the main theme. The rest of the files are inherited from the main theme.

**Tip**: How to customize a theme

Themes are located in the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes` folder. There are separate folders for each theme, typically by the name of the theme (e.g., wso2), inside the themes folder. In addition, there are CSS folders, which contain the CSS files of those themes, inside the individual theme folders. If you need to customize an existing theme, you need to change the corresponding CSS files.

Let's see how to create a new theme and set it to the API Store:

- Writing a sub theme of the main theme
- Setting the new theme as the default theme

**Writing a sub theme of the main theme**

As a main theme already has most of the UIs, the syntax, and logic of Jaggery code defined, in a typical scenario, you do not have to implement a theme from scratch. Rather, you just add in your edits as a sub-theme of the existing main theme as given below:

1. Download the default main theme from here, unzip it, and rename the folder according to the name of your new theme (e.g., ancient). Let's refer to this folder as `<THEME_HOME>`.
2. Make any changes you want to the theme.
   - For example, make the following changes in the CSS styles in the `<THEME_HOME>/css/custom.css` file using a text editor and save.
     - Add the following code to change the color of the header to red.
Update the color given for the search box to \#0be2e2.

As you plan to upload the theme as a sub-theme of the default main theme, delete all the files in your \$THEME_HOME\$ folder except the ones that you edited. The rest of the files are automatically applied from the main theme.

### Setting the new theme as the default theme

The following are the two methods in which you can set your new theme as the default theme:

- Saving directly in the file system
- Uploading through the Admin Portal

#### Saving directly in the file system

If you have access to the file system, do the following:

1. Save the \$THEME_HOME\$ folder that contains the sub-theme of the main theme inside the \$APIM_HOME\$/repository/deployment/server/jaggeryapps/store/site/themes/\$main-theme-directory$/subtheme/\$sub-theme$/css folder. This makes your new theme a sub-theme of \$wso2\$.
2. Open the \$APIM_HOME\$/repository/deployment/server/jaggeryapps/store/site/conf/site.json file, and add the following code to it. It specifies the base theme as \$wso2\$, which is overridden by the sub-theme \$ancient\$.

   ```json
   "theme" : {
     "base" : "wso2",
     "subtheme" : "ancient"
   }
   ```

3. Open the API Store.
   Note the new theme that is applied to it.

---

The custom.css file related to the sub-theme should always have the entire set of CSS styles. Therefore, do not delete the code related to the CSS styles that you have not changed in the \$APIM_HOME\$/repository/deployment/server/jaggeryapps/store/site/themes/\$main-theme-directory$/subtheme/\$sub-theme$/css/custom.css file.
Uploading through the Admin Portal (Tenants Only)

If you do not have access to the file system, you can upload the theme through the Admin Portal as shown below:

1. Navigate inside the `<THEME_HOME>` folder that contains the sub-theme of the main theme, select all the folders inside it, and right-click to compress all the selected files and folders. Then rename the ZIP file based on the name of your sub-theme. For this example use the `ancient.zip` file.
2. Sign in to the WSO2 Admin Portal ([https://<server-host>:9443/admin](https://<server-host>:9443/admin)) with your tenant username (format `<username>@<domain>.co m.kim@testorg.com`) and password.
3. Expand the Settings menu, click Upload Tenant Theme and upload your ZIP file.

4. Access the API Store ([https://<server-host>:9443/store](https://<server-host>:9443/store)) using your tenant username and password. Note the new theme that is applied.

Extending Scope Validation

OAuth scopes, which were introduced from WSO2 API Manager 1.7.0 onwards, allow you to have fine grained access control to API resources based on the user roles. It allows you to define scopes per API and associate defined scopes with API resources. OAuth 2.0 bearer tokens are obtained for a set of requested scopes and the token obtained is not allowed to access any API resources beyond the associated scopes. For more information, see OAuth Scopes.

API manager uses scopes as a way of defining permissions for a resource. If a resource is assigned a scope, then the token accessing the resource should be generated with that scope. By associating a scope with a role, you can control which users are permitted to have tokens under certain scopes. In this instance, associating a role to a scope seems legitimate.

Validating the role of a requester does not make much sense in some scenarios. For instance, when the scope is used as a means of generating an access token and not for securing a resource (e.g. openid scope). In such situations, scope validation can be extended to skip role validation for certain scopes.

Skipping role validation for scopes

When scopes which cannot be associated to roles are requested, the token should be issued without validating the scope. In WSO2 API Manager, you do this by whitelisting the scope through configuration. Patterns of the whitelisted scopes are specified via a configuration under the `<OAuthConfiguration>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file. Scopes that match the pattern are not validated by role and are available to anyone requesting it.

The following steps show a demonstration:

1. Start the API Manager server and log into the API Store.
2. Create an application. On the Production Keys tab of your application, click Generate Keys.
3. Obtain the **Base64** encoded representation of the Consumer Key and the Consumer Secret separated by a colon according to the following format.

```
Base64(consumer-key:consumer-secret)
```

You can also use the curl request listed under the Generate Access Tokens section for the steps 3 and 4 based on the grant type.

4. Use the Base64 encoded value obtained above in the Authorization header when invoking the following command. This is used to get the token by calling the token API.

```
curl -k -d "grant_type=password&username=admin&password=admin&scope=some_random_scope" -H "Authorization: Basic WmRFUFBvZmZwYVFnR25ScG5iZldtcUtSS3IwYTpSaG5ocEVJYUVCMENIT1FReWpiZTJwaDBzc1Vh" -H "Content-Type: application/x-www-form-urlencoded" https://10.100.0.3:8243/token
```

Along with the token, you receive a response from the server similar to the one below.

```
{"scope":"default","token_type":"bearer","expires_in":3600,"refresh_token":"23fac44e9b7e1ae95a33b85f4f26dec6","access_token":"9474fa104ccbi96303f41c8a5ee6f48"}
```

You may not see the scope you requested for in this response as it has not been whitelisted yet.

5. Shut down the server.

6. To whitelist the scope, add the following under the `<OAuthConfigurations>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file and restart the server.
7. Call the token API using the same request used in step 4. You will receive a response similar to the one below.

```xml
<ScopeWhitelist>
  <Scope>device_.*</Scope>
  <Scope>some_random_scope</Scope>
</ScopeWhitelist>
```

You see a successful response along with the whitelisted scope for which you requested.

### Extending the API Life Cycle

APIs created in WSO2 API Manager have their own life cycle consisting of the following: a set of life cycle states, specific actions for each state transition, and a checklist of items before a state transition occurs. In previous API Manager versions, an API had a predefined life cycle consists of six states which could not be customized or extended. From API Manager 1.10.0 onwards, you can extend the API life cycle with the WSO2 registry based life cycle in API Manager.

- Default API Lifecycle in WSO2 API Manager
- Extension Points of API Lifecycle

**Default API Lifecycle in WSO2 API Manager**

The WSO2 registry based life cycle provides a configurable way to define the life cycle of an artifact, which can be extended easily, as the default API life cycle is defined as an XML configuration.

Note that this extending capability of the API life cycle is not available in API Manager versions prior to 1.10.0.

To see the default API life cycle configuration, follow the steps below.

1. Start the API Manager server and log into the management console: `https://localhost:9443/carbon`
2. Navigate to Extensions > Configure > Lifecycles.

![Lifecycles](image)

3. Click the View/Edit link corresponding to the API LifeCycle. The default API life cycle configuration opens.

```xml
<aspect name="APILifeCycle"
  class="org.wso2.carbon.governance.registry.extensions.aspects.DefaultLifeCycle">
  <configuration type="literal">
    <lifecycle>
      <scxml xmlns="http://www.w3.org/2005/07/scxml"
        version="1.0"
        initialstate="Created">
        <state id="Created">
          <datamodel>
            <data name="checkItems">
              <item name="Deprecate Old Versions" forEvent="">
              </item>
            </data>
          </datamodel>
        </state>
      </scxml>
    </configuration>
  </lifecycle>
</aspect>
```
<item name="Require Re-Subscription" forEvent=""/>
</item>
</data>
<data name="transitionExecution">
  <execution forEvent="Deploy as a Prototype" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
  </execution>
  <execution forEvent="Publish" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
  </execution>
</data>
</datamodel>
<Transition event="Publish" target="Published"/>
<Transition event="Deploy as a Prototype" target="Prototyped"/>
</state>
<state id="Prototyped">
<datamodel>
  <data name="transitionExecution">
    <execution forEvent="Publish" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Demote to Created" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
  </data>
</datamodel>
<Transition event="Publish" target="Published"/>
<Transition event="Demote to Created" target="Created"/>
</state>
<state id="Published">
<datamodel>
  <data name="transitionExecution">
    <execution forEvent="Block" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Deprecate" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Demote to Created" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Demote to Prototyped" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
  </data>
</datamodel>
<Transition event="Block" target="Blocked"/>
<Transition event="Demote to Prototyped" target="Prototyped"/>
<Transition event="Demote to Created" target="Created"/>
<Transition event="Deprecate" target="Deprecated"/>
<Transition event="Publish" target="Published"/>
</state>
<state id="Blocked">
<datamodel>
  <data name="transitionExecution">
    <execution forEvent="Re-Publish" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Deprecate" class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
  </data>
</datamodel>
<Transition event="Block" target="Blocked"/>
<Transition event="Demote to Prototyped" target="Prototyped"/>
<Transition event="Demote to Created" target="Created"/>
<Transition event="Deprecate" target="Deprecated"/>
<Transition event="Publish" target="Published"/>
</state>
<transition event="Deprecate" target="Deprecated"/>
<transition event="Re-Publish" target="Published"/>
</state>
</state>
<state id="Deprecated">
<datamodel>
<data name="transitionExecution">
<execution forEvent="Retire"
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
</execution>
</data>
</datamodel>
</state>
<transition event="Retire" target="Retired"/>
</state>
<state id="Retired">
</state>
</scxml>
The above configuration includes the following important information:

1. Lifecycle name: APILifeCycle
2. Set of six default states: CREATED, PROTOTYPED, PUBLISHED, BLOCKED, DEPRECATED, RETIRED
3. Set of checklist items to be satisfied: For example, when the API is in the CREATED state and has multiple versions, there are two checks that occur: deprecate old versions and re-subscriptions required.
4. State transition events: Defines from which state to which target state an API can be moved.
5. Actions for each state transition: A triggered action that executes during each state transition. For example, when an API state changes from CREATED to PUBLISHED, an execution occurs as a relative synapse API where an XML element is created and the related API data is saved in the database. This execution is defined for each state transition in the above registry life cycle configuration.

The state transition events that occur in the default API life cycle is shown in the following diagram:
The **Lifecycle** tab shows the current state of an API, the target events defined in the API life cycle for that state, and the set of checklist items.
Extension Points of API Lifecycle

With the integration of the registry life cycle to the API life cycle of WSO2 API Manager, it is possible to extend the existing API life cycle and customize it according to your preference. Following are some extension points where the default API life cycle can be extended by modifying above mentioned XML configuration of the API life cycle.

Consider the following points when extending and customising the API life cycle XML configuration.

- Do not change the life cycle name since it needs to be engaged with the APIs dynamically.
- Make sure you keep the **PUBLISHED** and **PROTOTYPED** states as those two states will be used by API Publisher in the API creation wizard.
Following are some extension points that can be used:

- Define your own life cycle states in the API life cycle
- Change the state transition events as per the environmental preferences
- Add custom checklist items for specific state transitions
- Change the execution code for each state transition

For all state transitions, the same execution class is used (org.wso2.carbon.apimgt.impl.executors.APIExecutor). However, you can plug your own execution code when modifying the life cycle configuration. For example, if you want to add notifications for a specific state transition, you can plug your own custom execution class for that particular state in the API life cycle. Any changes are updated in the Lifecycle tab accordingly.

When a new transition event is introduced to the life cycle, an entry must be made to the locale_default.json file in order to view that life cycle transition event in the Publisher Lifecycle tab. This is introduced to support multi-language facility. For example, let's say a transition event called Notify Users is introduced in the DEPRECATED state as follows,

```xml
<state id="Deprecated">
  <datamodel>
    <data name="transitionExecution">
      <execution forEvent="Retire"
        class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
        </execution>
    </data>
  </datamodel>
  <transition event="Retire" target="Retired"/>
  <transition event="Notify Users" target="Retired"/>
</state>
```

You need to add "notify users" : "Notify Users" as an entry in the <APIM_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/locales/jaggery/locale_default.json file. Note that the key value in this entry should be in lower case (e.g. notify users).

For other languages, add the entry to the relevant file. For further information, see Adding Internationalization and Localization.

**Customize the API Store and Gateway URLs for Tenants**

The default URL of WSO2 API Manager Store is https://<HostName>:9443/store. Follow the steps below to change the URL of the Gateways and API Store tenants in WSO2 API Manager.

- Install Nginx and create SSL certificates
- Setup custom domain mapping in the registry
- Configure the store webapp

**Install Nginx and create SSL certificates**

Follow the steps below to install Nginx and create SSL certificates.

**Install nginx in Mac OS**

If you are using Mac OS, you need to install Nginx using the brew package manager. The commands are as follows.

- Command to install nginx
  ```bash
  brew install nginx
  ```
  
- Command to run nginx
  ```bash
  sudo nginx
  ```

1. Run the following command and install Nginx, if not already available.
sudo apt-get install nginx

2. Create an SSL certificate.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/ssl/nginx.key -out /etc/nginx/ssl/nginx.crt
```

3. Navigate to the `<API-M_HOME>/repository/resources/security` directory and use the following command to add the certificate to the client trust store.

```
keytool -import -file /etc/nginx/ssl/nginx.crt -keystore client-truststore.jks -storepass wso2carbon -alias wso2carbon2
```

4. Navigate to the `/etc/nginx/sites-enabled/default` directory in your terminal and add the following configurations with your custom domain name.

```
server {
    listen 443;
    ssl on;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;
    location / {
        proxy_set_header X-WSO2-Tenant "ten5.com";
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_pass https://localhost:9443/store/;
        proxy_redirect https://localhost:9443/store/ /;
        proxy_redirect <custom URL> /;
        proxy_cookie_path /store/ /;
    }
}
```

Setup custom domain mapping in the registry

Only the super tenant can add custom URLs in their registry space for other tenants. Tenants cannot configure custom URLs for their Store or Gateway.

1. Log in to the management console (https://<HostName>:9443/carbon) as the super admin.
2. In the **Main** menu, click **Browse** under **Resources**.
3. Navigate to the /_system/governance registry path and create the following directory structure in the registry.

   In API Cloud, this directory structure is created automatically when specifying the custom URL through the UI.

   `customurl/api-cloud/<tenant-domain>/urlMapping`
To create a directory in the registry path,

1. Navigate to the location in the registry browser, click and open the location.

2. Click **Add Collection** and specify the name of the directory and click **Add**.
5. Add the following resource configurations to the registry and click Add.

```json
{    "tenantDomain": "<tenant domain name>",
    "store": {
        "customUrl": "<custom domain for store>",
    },
    "gateway": {
        "customUrl": "<custom domain for gateway>",
    }
}
```

The URLs of the Store and Gateway are updated accordingly.

Configure the store webapp

1. Go to <API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf directory, open the site.json file and add the tenant header parameter as shown below.

```json
"reverseProxy": {
    "enabled": "auto",
    "host": "sample.proxydomain.com",
    "context": ".",
    "tenantHeader": "X-WSO2-Tenant"
},
```

[Image of the navigation steps and configuration options]
[Image of the site.json file configuration]

[Image of the Interface Manager screen with the Add Resource option highlighted]
You can choose any name for the header and set the virtual host to create the specific domain.

For details on how to create and manage multiple tenants, see Managing Tenants. You can also see Multi-tenant Architecture for more information about tenants.

**Editing API Templates**

Each API in API manager is represented by an XML file. The elements of this XML file and their attributes are defined in `<APIM_HOME>/repository/resources/api_templates/velocity_template.xml` file, which is the default API template that comes with the API Manager. By editing the default template definitions, you can change the synapse configuration of all APIs that are created.

If you are using a distributed API Manager setup (i.e., Publisher, Store, Gateway and Key Manager components are running on separate JVMs), edit the template in the Publisher node.

**Customizing Login Pages for API Store and API Publisher**

Custom pages for logging into the server are available for SAML2 SSO, OAuth and OpenID. This section guides you through this customization.

The login pages and other pages like error and notification screens of SAML SSO, OAuth, OpenID and Passive STS are located in the `authenticationendpoint` webapp file found at `<APIM_HOME>/repository/deployment/server/webapps`.

You can easily customize these pages within this web application by changing the respective JSPs, JavaScript and CSS. If you want to point to a different web application, you can do so by redirecting or forwarding from `authenticationendpoint` to your webapp. In the case of SAML SSO, the 'issuer' id of the service provider is also sent to this webapp. Therefore, different login pages can be given to different service providers by looking at the 'issuer' request parameter.

The following is a sample of how to customize the login page for SAML2 SSO.

**Customizing the login page for SAML SSO service providers**

Usually WSO2 API Manager displays a default login page for all the SAML SSO service providers that send authentication requests to it. The following steps indicate how to change the default login page into a customized one.

1. **Registering the two service providers in the Identity Server**
2. **Configuring the login page**

**Registering the two service providers in the Identity Server**

1. Download WSO2 Identity Server and extract it.
2. Run the server by executing `wso2is-5.0.0/bin/wso2server.sh` if on a Unix-based systems, or `/bin/wso2server.bat` if on Windows.
3. On the management console, click Add under Service Providers in the Main menu.
4. Enter "publisher" as the Service Provider Name in the form that appears and click Register.

5. In the page that appears next, expand the **Inbound Authentication Configuration** section and the **SAML2 Web SSO Configuration** section. Click **Configure**. The Register New Service Provider page appears.
5. Configure the following details for publisher.
   - Issuer: api_publisher
   - Assertion Consumer URL: https://10.100.5.83:9443/publisher/jagg/jaggery_acs.jag
   - Select Enable Response Signing
   - Select Enable Single Logout

6. Repeat steps 1 to 5 and configure the following details for store.
   - Issuer: api_store
   - Assertion Consumer URL: https://10.100.5.83:9443/store/jagg/jaggery_acs.jag
   - Select Enable Response Signing
   - Select Enable Single Logout

7. When attempting to login with SAML from WSO2 Identity Server in API publisher and API store, you can see the following default page located at <IS_HOME>/repository/deployment/server/webapps/authenticationendpoint/login.jsp

For instructions on configuring WSO2 Identity Server as an identity provider, see Configuring Identity Server as IDP for SSO.

Configuring the login page

Understanding the authenticationendpoint web application

The login page that is displayed during SAML2 SSO, OAuth, OpenID and Passive-STS flows is located inside the webapp named authenticationendpoint. The reason for storing this in a web app is:
   - to easily customize the page according to user requirements
   - if needed, place that whole web application in an external application server

The Identity Server knows the location of this web application as it is specified in the <IS_HOME>/repository/conf/identity/application-authentication.xml configuration file. This is referenced as shown below.
By default it points to a location inside the Identity Server itself, thus the relative path is given. If it is necessary to point to an external application, the full path should be given instead.

If this web app is moved outside the Identity Server, we must ensure that no one can access the login credentials that are passed between this application and the Identity Server. This means that the external location should ideally be either inside a secured intranet or the transport should be HTTPS. Other similar precautions may be necessary to secure the communication.

The following is the structure of this web app.

The **authenticationendpoint** web application uses a carbon component called `org.wso2.carbon.identity.application.authentication.endp oint.util`. This bundle includes a filter called the `org.wso2.carbon.identity.application.authentication.endpoint.util.filter.Au thenticationEndpointFilter`, which acts as the Front Controller.

When a request is made to the **authenticationendpoint** web application, based on the authentication protocol type identified by the request parameter `type`, the controller first forwards the request to the protocol based login url patterns defined. For example, if the request to the **web application** is initiated as a result of a SAML SSO authentication request, the controller will forward the request to the url pattern `/samlssologin.do`. If you look inside the `web.xml`, you will see that this url pattern is mapped to the `login.jsp` file. The request is finally forwarded to this `login.jsp` page.

Everything on the **authenticationendpoint** web application is customizable. You can customize it by adding JSP pages or modifying them and configuring the `web.xml` respectively.

The only restriction involved is that the content already sent back by the pages inside the default web app must be submitted to the Identity Server. Additionally, you must point to the correct location via the `<IS_HOME>/repository/conf/identity/application-authentication.xml` file.

**Customizing the login page**

When a request comes to the default login page, you can see several parameters being passed in the address bar. For this customization, the focus is on the following two parameters:

- **sessionDataKey**: This is an identifier used by the Identity Server to maintain state information related to this particular request by the service provider.
- **relyingParty**: This is the value we gave for the "Issuer" field when we registered the SAML2 SSO service provider (e.g., `travelocity.com`). This value is used to display different login pages to different service providers.
When customizing the pages, ensure that the following is applied.

1. Form submissions should happen to the "commonauth" servlet as a POST.
   `<form id="form" name="form" action="../../commonauth" method="POST">`
2. Make sure to send back the "sessionDataKey" with the form submission, by using a hidden input field.
   `<input type="hidden" name="sessionDataKey" value="<%=request.getParameter("sessionDataKey")%>">

**Using a JSP to redirect to SP relevant pages**

1. Rename the existing 'login.jsp' to 'default_login.jsp'
2. Create a new file with the name 'login.jsp' including the following code.

```jsp
<% String relyingParty = request.getParameter("relyingParty");
    if (relyingParty.equals("api_publisher")) {
        RequestDispatcher dispatcher = request.getRequestDispatcher("publisher_login.jsp");
        dispatcher.forward(request, response);
    } else if (relyingParty.equals("api_store")) {
        RequestDispatcher dispatcher = request.getRequestDispatcher("store_login.jsp");
        dispatcher.forward(request, response);
    } else {
        RequestDispatcher dispatcher = request.getRequestDispatcher("default_login.jsp");
        dispatcher.forward(request, response);
    }
%>

This code snippet forwards the request to a different login page by checking the value of relyingParty parameter.
3. Get the 'publisher_login.jsp' from [here](#) and place it at the same level as 'login.jsp'. Also, download the contents of the 'css' folders from that same link and put them inside the respective folders in the authenticationendpoint.
4. Log in to the publisher web app again. You are presented with a different page.

5. Follow steps 1 to 4 to configure the custom login page to the store web app.

**API Gateways with Dedicated Backends**

We can extend the [multiple gateway environments](#) feature by utilizing dynamic endpoint capabilities of WSO2 API Manager to have each gateway point to a different back-end endpoint. API Gateway is the actual runtime of the APIs that are developed and published from the API Publisher. WSO2 API Manager is capable of publishing APIs to different Gateways where API users connect to those API Gateways in order to do the actual API calls through the applications to which they are subscribed.

However, the API Publisher can only provide a single static endpoint for an API in the implementation. Therefore, the API call is directed to a single endpoint in whichever Gateway the API is deployed in, as depicted in the diagram below.
However, in most situations, you would want to have each Gateway proxying to a dedicated backend API. To provide that capability, WSO2 API Manager provides the ability to specify dynamic endpoint URLS at the time of specifying the API endpoint URL. This UEL is resolved at runtime with the details (host and port) specified at the startup of each Gateway. Each gateway then points to a dedicated backend API, as depicted in the diagram below.

**Configuring dynamic endpoints**

Follow the steps below to configure a dynamic endpoint as the API endpoint.

1. Start the WSO2 API Manager server that includes the API Publisher component and create an API.
2. Go to the Implement tab of the API and replace the host and port of the API endpoint with `{uri.var.host}` and `{uri.var.port}` respectively, as shown below.
3. Save and publish the API.
4. Navigate to the `<API-M_HOME>/repository/deployment/server/synapse-configs/sequences` directory of each Gateway and create the following sequence.
Java system properties are used at the server start-up process of each Gateway to resolve the variables that are defined as properties in this sequence.

Alternatively, you can resolve this host and port using a class mediator. To do that, follow the steps below as an alternative to step 4.

1. Create a java class extending the AbstractMediator class of org.synapse.core as shown below and create the JAR file out of it.

```java
import org.apache.synapse.MessageContext;
import org.apache.synapse.mediators.AbstractMediator;

public class EnvironmentResolver extends AbstractMediator {
    @Override
    public boolean mediate(MessageContext messageContext) {
        String host = System.getProperty("environment.host");
        String port = System.getProperty("environment.port");

        messageContext.setProperty("uri.var.host", host);
        messageContext.setProperty("uri.var.port", port);

        return true;
    }

    @Override
    public boolean isContentAware(){
        return false;
    }
}
```

2. Add the created JAR file into the <API-M_HOME>/repository/components/lib folder of each Gateway. You can download a sample JAR file here.

3. Add the following sequence to the <API-M_HOME>/repository/deployment/server/synapse-configs/sequences folder of each Gateway.

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="WSO2AM--Ext--In">
    <class name="org.wso2.carbon.env.EnvironmentResolver"/>
</sequence>
```

`org.wso2.carbon.env.EnvironmentResolver` is the fully qualified name of the class that contains the code responsible for converting system variables into properties. It is a special class that needs to be extended from the `org.apache.synapse.mediators.AbstractMediator` class and requires overriding of the `mediate` function.

5. Execute the following command when starting up each Gateway to set the system variables at the server start up from within the `<API-M_HOME>/bin` directory by replacing the following values.

```
<ip_of_backend_environment>  <port_of_backend_environment>
host IP of the Gateway       port where the Gateway is running in the dedicated machine or VM
```
5.
6.
./wso2server.sh -Dhost=<ip_of_backend_environment> -Dport=<port_of_backend_environment>

Now the Gateways have started with the dedicated backend host/port combinations.

6. Invoke the API.

You receive the response from the API, which is sent through the dedicated backend, from the Gateway that this API is published.

Securing OAuth Token with HMAC Validation

Implementing security measures in order to prevent the possible attacks is a need in using enterprise software. Keyed-Hash Message Authentication Code (HMAC) validation is such measure which involved a cryptographic hash function and used to verify both the data integrity and authentication of a Message as with any Message Authentication code. In this tutorial you will use the HMAC to validate the OAuth tokens created in WSO2 API Manager and and WSO2 Identity Server.

- Preventing miss-use of OAuth Tokens
- WSO2 IS Extension - OAuth Token Generator Extension
- WSO2 API Manager extension - HMAC and timestamp verification handler

Preventing miss-use of OAuth Tokens

In API Manager, the main use case of HMAC is preventing miss-use of expired OAuth tokens or randomly generated OAuth tokens. Stolen or randomly generated tokens can be used to employ DOS/DDOS attacks effectively.

If an attacker uses random tokens to send API requests, API Manager will try to verify the token and it will hit through the critical path of verification. This is a costly transaction and it can cause high latencies and instability in API Manager clusters. Implementation of this particular solution is done using extensions developed for standard extension points of WSO2 API Manager and WSO2 Identity Server.

WSO2 IS Extension - OAuth Token Generator Extension
Engage the HMAC OAuth handler in order to do the Keyed-Hash Message Authentication Code (HMAC) validation by adding following into `<IS_HOME>/repository/conf/identity/identity.xml`

```xml
<IdentityOAuthTokenGenerator>com.sample.lahiru.wso2.hmac.oauth</IdentityOAuthTokenGenerator>
```

This extension is responsible for enhancing the OAuth token with HMAC(Hash-based Message Authentication Code), so that above mentioned attacks will be less effective. Following two parts will be added to the token in addition to the default token created in WSO2 IS.

- HMAC
- Expiry timestamp

The format of the access token will be as follows thereafter. The token has 3 parts, delimited by ".".

**Part I**—original access token issued from WSO2 Identity Server

**Part II**—Hex value for token expiry time

**Part III**—HMAC calculation of ("Part I" + "." + "Part II")

**Access token format**: `<Part I>.<Part II>.<Part III>`

Example: `ba13cf7473cfbde970ae6e8b60973f64.0000015fc1ebabde.67830f2f2886256eb80faa9dad85c3d2c9be7db1`

**WSO2 API Manager extension - HMAC and timestamp verification handler**

You can engage this handler by adding following entry before `#foreach($handler in $handlers)` line of `velocity_template.xml` file located in `<AM_HOME>/repository/resources/api_templates/` directory.

```xml
<handler class="com.sample.lahiru.wso2.hmac.handler.HMACTokenValidatorHandler"/>
```

Refer [Writing Custom Handlers](#) for understanding how to develop and engage WSO2 API handler extensions. Find the code for APIM handler in GitHub in `oauth-hmac-extension`.

This custom handler verifies the HMAC of the token before it tries to authenticate using default authentication handler, which will be an expensive operation usually. It will also make sure the token is not expired. These verification will avoid any API calls to WSO2 API Manager, in case of the token is expired or HMAC is invalid.

HMAC validation handler calculates the HMAC using Part I and Part II (See Access token format), extracted from the token and validates by comparing that value with HMAC value included in the token (Part III).

**Customizing User SignUp in API Store**

WSO2 API Manager (WSO2 API-M) allows new users to gain access to the API store via a Self Sign-up page. The default sign-up page has a set of mandatory and optional fields where the user can use to provide their details. However, there can be cases where API Store owners need to customize the available fields by modifying the available fields or/and adding new fields.

This customization can be easily achieved in WSO2 API Manager because the fields are loaded dynamically from the user claim attributes. The fields that are available in the Sign-up page together with instructions on how to customize the default Sign-up page are explained as follows.

- User sign-up page
- Adding a new field to the user sign up form
- Modifying existing fields in the user sign up form
- Adding consent purposes

**User sign-up page**

By default, the API Store Sign-up page looks as follows:
It has following fields by default.

<table>
<thead>
<tr>
<th>Mandatory Fields</th>
<th>Additional Fields (Optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>Organization</td>
</tr>
<tr>
<td>Password</td>
<td>Country</td>
</tr>
<tr>
<td>Re-type Password</td>
<td>Land Phone</td>
</tr>
<tr>
<td>FirstName</td>
<td>Mobile Phone</td>
</tr>
<tr>
<td>LastName</td>
<td>IM</td>
</tr>
<tr>
<td>Email</td>
<td>URL</td>
</tr>
</tbody>
</table>

Adding a new field to the user sign up form
If you want to add a new field to the API Store Sign-up page that will be filled up when a new user signs up to the API Store, you can do this by adding a local claim via the WSO2 API-M Management Console.

Let's add a field named City by following the instructions below:

1. Start the WSO2 API Manager server, and go to the WSO2 API-M Management Console (https://localhost:9443/carbon/)
2. Navigate to the Main menu and click Add which is under the Claims tab.
3. Click Add Local Claim.
4. Enter the following values as the local claim details.

<table>
<thead>
<tr>
<th>Claim URI</th>
<th><a href="http://wso2.org/claims/city">http://wso2.org/claims/city</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name</td>
<td>City</td>
</tr>
<tr>
<td>Description</td>
<td>City</td>
</tr>
<tr>
<td>Mapped Attribute</td>
<td>city</td>
</tr>
<tr>
<td>Supported By Default</td>
<td>select</td>
</tr>
</tbody>
</table>

5. If you need to define this claim as a required field (Mandatory field in Sign-up), make sure to check the Required check box.
6. Click Add.
7. Navigate to API Store Sign-up page and refresh it.
   You should see the newly added field.

The claims which are Supported by Default, are only displayed in the Sign-up page. Therefore, when you are adding new claims make sure to select the Supported by Default check box.
Modifying existing fields in the user sign up form

You can modify the fields of the user sign-up page by editing the existing claims mapped to the respective fields.

Let's make the field named “City” a mandatory field and also change the display order of the field by following the instructions below:

1. Start the WSO2 API Manager server and navigate to WSO2 API-M Management Console (https://localhost:9443/carbon/).
2. Navigate to the Main menu and click List which is under the Claims tab.
3. Click \texttt{http://wso2.org/claims} in the list of claims that appear.
4. Edit the respective claim.
   
   a. Click on the \textbf{Edit} link that corresponds to the "City" claim.

   b. Select the \textbf{Required} check box. To change the display order, change the display order of all the city claim to 4.

   c. To change the display order, change the \textbf{Display Order} field to 4.

5. Navigate to API Store Sign-up page and refresh it.
   You will see that the "City" field is re-ordered and marked as a mandatory.
Similarly, you can modify the other existing fields in the user sign-up page by editing the claims based on your requirement.

Adding consent purposes

WSO2 API-M allows you to specify different consent purposes of collecting Personally Identifiable Information (PII) from users who sign up to the API Store. For example, the email address or the telephone number can be used for marketing purposes.

Follow the steps below to add consent purposes.

1. Sign in to the WSo2 API-M Management console (https://<host>:<port>/carbon)
2. Navigate to the Main tab. Select Consent Purposes, which is under the Manage tab, and click Add.
3. Fill in the details of the PII categories as shown below.
4. Click Finish.

If a specific tenant admin has previously specified consent purposes via the WSO2 API-M Management Console, during the sign-up process, the list of those consent purposes will be shown to the respective users who belong to the same tenant domain in order to obtain their consent.

Working with Security

After you install the API-M, it is recommended to change the default security settings according to the requirements of your production environment. As the API-M is built on top of the WSO2 Carbon platform, some security configurations are inherited from the Carbon platform.

Important!
If you are configuring your production environment, be sure to check the Security Guidelines for Production Deployment before applying any security configurations.

The following topics explain the platform-specific, and product-specific configurations:

- API-M-specific security configurations
- WSO2 Carbon platform-based security configurations
- API Endpoint Security

API-M-specific security configurations

See the following topics:
- Passing Enduser Attributes to the Backend Using JWT
- Dynamic SSL Certificate Installation
- Maintaining Logsins and Passwords
- Saving Access Tokens in Separate Tables
- Configuring WSO2 Identity Server as the Key Manager
- Configuring a Third-Party Key Manager
- Enabling Role-Based Access Control Using XACML
- Encrypting OAuth Keys
- Provisioning Out-of-Band OAuth Clients
- Pass a Custom Authorization Token to the Backend

WSO2 Carbon platform-based security configurations

The following security configurations are common to all WSO2 products that are built on top of the WSO2 Carbon platform.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>
### Configuring transport-level security

WSO2 products support a variety of transports that make them capable of receiving and sending messages over a multitude of transport, and application-level protocols. By default, all WSO2 products are shipped with the HTTP transport. The transport receiver implementation of the HTTP transport is available in Carbon platform. The transport sender implementation comes from the Tomcat HTTP connector, which is configured in the `<API-M_HOME>/repository/conf/tomcat/catalina-server.xml` file.

For more information on securing the HTTP transport, see Configuring transport level security in the WSO2 Administration Guide.

### Configuring keystores
A keystore is a repository that stores the cryptographic keys and certificates. These artifacts are used for encrypting sensitive information, and establishing trust between your server and outside parties that connect to your server.

All WSO2 products come with a default keystore (ws2carbon.jks). In a production environment, it is recommended to replace it with one. You can also configure multiple keystores for different purposes.

See the following in the WSO2 Administration Guide:

- Learn how public key encryption and keystores are used.
- Learn how to create new keystores and replace the default one.
- Learn how configuration files should be updated to use the relevant keystore for different purposes.

To download a keystore in WSO2 API Manager, do the following:

a. Sign in to https://<hostname>:9443/carbon as the tenant admin and click on Configure.

b. Select Keystores.

c. Click Public Key to download the keystore for the selected tenant.
Securing sensitive passwords
As a secure vault implementation is available in all WSO2 products, you can encrypt the sensitive data (i.e., passwords in configuration files and passwords for mediation flows) using the Cipher tool. For more information, see the following sections.

- Working with Encrypted Passwords
- Encrypting Secure Endpoint Passwords

Enabling JAVA security manager
See Enabling JAVA security manager in the WSO2 Administration Guide on how to prevent untrusted code from manipulating your system.

API Endpoint Security

Look into the following topics under enabling endpoint security for the APIs.

- Basic Auth
- Digest Auth

Passing Enduser Attributes to the Backend Using JWT

JSON Web Token (JWT) is used to represent claims that are transferred between two parties such as the end user and the backend.

A claim is an attribute of the user that is mapped to the underlying user store. It is encoded as a JavaScript Object Notation (JSON) object that is used as the payload of a JSON Web Signature (JWS) structure, or as the plain text of a JSON Web Encryption (JWE) structure. This enables claims to be digitally signed.

A set of claims is called a dialect (e.g., http://wso2.org/claims). The general format of a JWT is {token info}.{claims list}.{signature} . The API implementation uses information such as logging, content filtering and authentication/authorization that is stored in this token. The token is Base64-encoded and sent to the API implementation in a HTTP header variable. The JWT is self-contained and is divided into three parts as the header, the payload and the signature. For more information on JWT, see JSON Web Token (JWT) Overview.

To authenticate end users, the API Manager passes attributes of the API invoker to the backend API implementation using JWT. In most production deployments, service calls go through the API Manager or a proxy service. If you enable JWT generation in the API Manager, each API request will carry a JWT to the back-end service. When the request goes through the API manager, the JWT is appended as a transport header to the outgoing message. The back-end service fetches the JWT and retrieves the required information about the user, application, or token.

An example of a JWT is given below:

```json
{
   "typ": "JWT",
   "alg": "NONE"
}
{"iss": "wso2.org/products/am",
  "exp": 1345183492181,
  "http://wso2.org/claims/subscriber": "admin",
  "http://wso2.org/claims/applicationname": "app2",
  "http://wso2.org/claims/apicontext": "placeFinder",
  "http://wso2.org/claims/version": "1.0.0",
  "http://wso2.org/claims/tier": "Silver",
  "http://wso2.org/claims/enduser": "sumedha"}
```

The above JWT token contains the following information.

**JWT Header**: The header section declares that the encoded object is a JSON Web Token (JWT) and the JWT is in plaintext, that is not signed using any encryption algorithm.

**JWT Claims set**:  
- "iss" - The issuer of the JWT  
- "exp" - The token expiration time  
- "http://wso2.org/claims/subscriber" - Subscriber to the API, usually the app developer  
- "http://wso2.org/claims/applicationname" - Application through which API invocation is done  
- "http://wso2.org/claims/apicontext" - Context of the API  
- "http://wso2.org/claims/version" - API version  
- "http://wso2.org/claims/tier" - Tier/price band for the subscription  
- "http://wso2.org/claims/enduser" - Enduser of the app who's action invoked the API

Let's see how to enable and pass information in the JWT or completely alter the JWT generation logic in the API Manager:

- Configuring JWT  
- Customizing the JWT generation  
- Changing the JWT encoding to Base64URL encoding  
- Expiry time of the JWT
Configuring JWT

Before passing enduser attributes, you enable and configure the JWT implementation in the `<API-M_HOME>/repository/conf/api-manager.xml` file. The relevant elements are described below. If you do not configure these elements, they take their default values.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;EnableJWTGeneration&gt;</code></td>
<td>Uncomment <code>&lt;EnableJWTGeneration&gt;</code> property and set this value to <code>true</code> to enable JWT.</td>
</tr>
<tr>
<td><code>&lt;JWTHeader&gt;</code></td>
<td>The name of the HTTP header to which the JWT is attached.</td>
</tr>
<tr>
<td><code>&lt;ClaimsRetrieverImplClass&gt;</code></td>
<td>By default, the <code>&lt;ClaimsRetrieverImplClass&gt;</code> parameter is commented out in the <code>api-manager.xml</code> file. Enable it to add all user claims in the JWT token: <code>&lt;ClaimsRetrieverImplClass&gt;org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever&lt;/ClaimsRetrieverImplClass&gt;</code></td>
</tr>
</tbody>
</table>

By default, the following are encoded to the JWT:
- subscriber name
- application name
- API context
- API version
- authorized resource owner name

In addition, you can also write your own class by extending the interface `org.wso2.carbon.apimgt.impl.token.ClaimsRetriever` following methods of the interface:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void init() throws</td>
<td>Used to perform initialization tasks. Is executed once, right before the very first SortedMap&lt;String, String&gt; getClaims(String endUserName) throws</td>
</tr>
<tr>
<td>APIManagementException;</td>
<td>APIManagementException;</td>
</tr>
<tr>
<td>String getDialectURI(String endUserName);</td>
<td>The dialect URI to which the attribute names returned by the <code>getClaims()</code> method if the <code>getClaims</code> method returns <code>{email:user1@wso2.com, gender:male, wso2.org/claims, the JWT will contain &quot;http://wso2.org/claims/email&quot;:&quot;user1@wso2.com&quot;, &quot;http://wso2.org/claims/gender&quot;:&quot;male&quot;,&quot;http://wso2.org/claims/email&quot;:&quot;user1@wso2.com&quot; as part of the body. The default implementation </code>org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever<code>attributes defined under the dialect URI</code><a href="http://wso2.org/claims%60">http://wso2.org/claims`</a> and the URI. The order of encoding the user's attributes is the natural order of the attributes will be encoded, except the 6 default attributes.</td>
</tr>
</tbody>
</table>

| `<ConsumerDialectURI>`      | The dialect URI under which the user's claims are be looked for. Only works with the default value of the `<ClaimsRetrieverImplClass>` element. The default value of this element users to be included in the JWT, simply uncomment this element after enabling the JWT. It will include all claims in the JWT. |

| `<SignatureAlgorithm>`      | The signing algorithm used to sign the JWT. The general format of the JWT is `{token info}.{claims list}.{signature}` algorithm, signing is turned off and the JWT looks as `{token info}.{claims list}` with two strings delimited by a period. This element can have only two values - the default value, which is SHA256withRSA or NONE. |

You can use TCPMon or API Gateway debug logs to capture JWT token header with enduser details. To enable gateway DEBUG logs for wire messages,

1. Go to the `<APIM_GATEWAY>/repository/conf` directory and open the `log4j.properties` file with a text editor.
2. Edit the entries for the two loggers as follows (remove the # in order to enable debug logs):
   ```
   log4j.logger.org.apache.synapse.transport.http.headers=DEBUG
   log4j.logger.org.apache.synapse.transport.http.wire=DEBUG
   ```
Customizing the JWT generation

The JWT that is generated by default (see example above) has predefined attributes that are passed to the backend. These include basic application-specific details, subscription details, and user information that are defined in the JWT generation class that comes with the API Manager by the name `org.wso2.carbon.apimgt.keymgt.token.JWTGenerator`. If you want to pass additional attributes to the backend with the JWT or completely change the default JWT generation logic, do the following:

1. Write your own custom JWT implementation class by extending the default `JWTGenerator` class. A typical example of implementing your own claim generator is given below. It implements the `populateCustomClaims()` method to generate some custom claims and adds them to the JWT.

```java
import org.wso2.carbon.apimgt.keymgt.APIConstants;
import org.wso2.carbon.apimgt.keymgt.dto.APIKeyValidationInfoDTO;
import org.wso2.carbon.apimgt.keymgt.token.JWTGenerator;

public class CustomTokenGenerator extends JWTGenerator {
    // Custom implementation...
}
```

2. Build your class and add the JAR file to the `<API-M_HOME>/repository/components/lib` directory.
3. Add your class in the `<JWTGeneratorImpl>` element of the `<API-M_HOME>/repository/conf/api-manager.xml` file.

```xml
<JWTConfiguration>
    
    <JWTGeneratorImpl>org.wso2.carbon.test.CustomTokenGenerator</JWTGeneratorImpl>

</JWTConfiguration>
```
4. Set the `<EnableJWTGeneration>` element to `true` in the `api-manager.xml` file.
5. Restart the server.

**Changing the JWT encoding to Base64URL encoding**

The default JWT generator, `org.wso2.carbon.apimgt.impl.token.JWTGenerator`, encodes the value of the JWT using Base64 encoding. However, for certain apps you might need to have it in Base64URL encoding. To encode the JWT using Base64URL encoding, add the `URLSafeJWTGenerator` class in the `<TokenGeneratorImpl>` element in the `api-manager.xml` file as shown below.

```xml
<JWTConfiguration>
  ....
  <JWTGeneratorImpl>org.wso2.carbon.keymgt.token.URLSafeJWTGenerator</JWTGeneratorImpl>
  ....
</JWTConfiguration>
```

### Expiry time of the JWT

JWT expiry time depends directly on whether caching is enabled in the Gateway Manager or Key Manager. The WSO2 API-M Gateway caching is enabled by default. However, if required, you can enable or disable the caching for the Gateway Manager or the Key Manager using the `<EnableGatewayTokenCache>` or `<EnableKeyManagerTokenCache>` elements respectively in the `api-manager.xml` file. If caching is enabled for the Gateway Manager or the Key Manager, the JWT expiry time will be the same as the default cache expiry time.

The claims that are retrieved for the JWT Token generation are cached. The expiry time of these JWT claims can be set by setting the `JWTClaimCacheExpiry` in the `api-manager.xml`, under CacheConfigurations element:

```xml
<CacheConfigurations>
  ...
  <!-- JWT claims Cache expiry in seconds -->
  <JWTClaimCacheExpiry>900</JWTClaimCacheExpiry>
  ...
</CacheConfigurations>
```

When both Gateway and Key Manager caches are disabled, the JWT expiry time can be set by adding the `JWTExpiryTime` property under `APIKeyValidator` element in `api-manager.xml`:

```xml
<APIKeyValidator>
  ...
  <JWTExpiryTime>900</JWTExpiryTime>
  ...
</APIKeyValidator>
```

### Dynamic SSL Certificate Installation

If you have a backend with a self-signed certificate (or a certificate which is not signed by a CA) you need to import it to the client-truststore and restart the server. This feature enables you to upload the backend certificate through API Publisher while creating or editing your API. Follow the steps below to add a new certificate to any endpoint. Note that this feature supports only HTTP/REST and HTTP/SOAP endpoints.

#### Prerequisites

1. Ensure that you have downloaded the latest WUM update. For more details, see [Updating WSO2 Products](#) in the WSO2 Administration Guide.
2. If you are an existing user, follow the instructions given below.
   a. Run the scripts inside the `<API-M_HOME>/dbscripts/apimgt` directory, according to your preferred database. For instructions on configuring databases, see [Set up the database](#). Verify that the table `AM_CERTIFICATE_METADATA` has been created in your database.
   b. The configurations for the `PassThroughHTTPSSLSender` parameter is available by default in the `api-manager.xml` file as shown below.

```xml
<PassThroughHTTPSSLSender>
  ...
  <PassThroughHTTPSSLSender>true</PassThroughHTTPSSLSender>
  ...
</PassThroughHTTPSSLSender>
```
c. If you use a different Trust Store/Keystore configuration in the axis2.xml or carbon.xml files, modify the KeyStore and TrustStore location in `<API-M_HOME>/repository/resources/security/sslprofiles.xml` file accordingly. The `sslprofiles.xml` file is configured with the existing client-truststore.jks.

This feature currently supports only the following formats for keystores and certificates.

- **Keystore**: .jks
- **Certificate**: .crt

If you need to use a certificate in any other format, you can convert it using a standard tool before uploading.

After configuring, the certificate will be added to the Gateway nodes which are defined under the Environments in `api-manager.xml`. In a clustered setup, as gateway configurations are identical, sync the `<API-M_HOME>/repository/resources/security/sslprofiles.xml` and `<API-M_HOME>/repository/resources/security/client-truststore.jks` among the gateway nodes. After the configured interval, the synapse transport will be reloaded in all the gateway nodes.

**Adding a certificate**

1. Log in to the API Publisher. Create a new API or edit an existing API.
2. Go to the Implement tab. Click Manage Certificates and click Add New Certificate.

   ![Endpoint: *(specify at least one)*](image)

   **Production**: https://localhost:9443/am/sample/pizzasheek/v1/api
   **Sandbox**: https://localhost:9443/am/sample/pizzasheek/v1/api

   **Manage Certificates**

   You do not have any uploaded Certificates for above endpoints.

   **Add New Certificate**

3. Enter the following information and click **Upload**.
3. Enter a name for your certificate.

4. Select an endpoint from the dropdown list.

5. Enter the location of your certificate file or click Browse to select through the UI.

The uploaded certificate aliases will be displayed.

You can repeat from step 2 to add a certificate to the sandbox endpoint.

5. You can repeat from step 2 to add a certificate to the sandbox endpoint.

Deleting a certificate

To delete a certificate, click the icon adjacent to the certificate, as shown below.
Changing the super admin credentials

Follow the instructions below to change the default admin password:

1. Change the user credentials in the following files.
   - The `<UserName>` and `<Password>` values in `<API-M_HOME>/repository/conf/user-mgt.xml` file.

   ```xml
   <UserManager>
       <Realm>
           ...  
           <AdminUser>
               <UserName>admin</UserName>
               <Password>admin</Password>
           </AdminUser>
           ...  
       </Realm>
   </UserManager>
   ```

   Note that the password in the `user-mgt.xml` file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the `user-mgt.xml` file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the `user-mgt.xml` file. To change the super admin password, you must use the Change Password option from the management console.

   To change the password from Management Console (https://localhost:9443/carbon), follow the steps in Changing a Password corresponding to API Manager.

   - The `<API-M_HOME>/repository/conf/jndi.properties` file.

   ```properties
   connectionfactory.TopicConnectionFactory = amqp://admin:admin@clientID/carbon?brokerlist='tcp://localhost:5672'
   connectionfactory.QueueConnectionFactory = amqp://admin:admin@clientID/test?brokerlist='tcp://localhost:5672'
   ```

   If you have Configured API Manager Analytics, when changing the super admin credentials you have to change credentials in `<API-M_HOME>/repository/conf/api-manager.xml` and `<API-M_HOME>/repository/conf/log4j.properties` as well.

   - The `<API-M_HOME>/repository/conf/api-manager.xml` file.
<Analytics>
  <!-- Enable Analytics for API Manager -->
  <Enabled>true</Enabled>
  ....
  <StreamProcessorServerURL>{tcp://localhost:7612}</StreamProcessorServerURL>
  <!--StreamProcessorAuthServerURL>{ssl://localhost:7712}</StreamProcessorAuthServerURL-->
  <!-- Administrator username to login to the remote Stream Processor server. -->
  <StreamProcessorUsername>${admin.username}</StreamProcessorUsername>
  <!-- Administrator password to login to the remote Stream Processor server. -->
  <StreamProcessorPassword>${admin.password}</StreamProcessorPassword>
  ....
  <StatsProviderImpl>org.wso2.carbon.apimgt.usage.client.impl.APIUsageStatisticsRdbmsClientImpl</StatsProviderImpl>
  ...
  <StreamProcessorRestApiURL>https://localhost:9444</StreamProcessorRestApiURL>
  <StreamProcessorRestApiUsername>${admin.username}</StreamProcessorRestApiUsername>
  <StreamProcessorRestApiPassword>${admin.password}</StreamProcessorRestApiPassword>
  ....
</Analytics>

- The `<API-M_HOME>/repository/conf/log4j.properties` file:

  log4j.appender.DAS_AGENT.userName=admin
  log4j.appender.DAS_AGENT.password=admin
  
  log4j.appender.LOGEVENT.userName=admin
  log4j.appender.LOGEVENT.password=admin

### Do you have any special characters in passwords?

- If you specify passwords inside XML files, take care when giving special characters in the user names and passwords. According to XML specification (http://www.w3.org/TR/xml), some special characters can disrupt the configuration. For example, the ampersand (&) must not appear in the literal form in XML files. It can cause a Java Null Pointer exception. You must wrap it with CDATA (http://www.w3schools.com/xml/xml_cdata.asp) as shown below or remove the character:

  `<Password>
   <![CDATA[xnvYh?AVHAc7q2%Jv855%A4a,%M8B]]>
  </Password>`

- Note the following if you have special characters in the passwords on your `jndi.properties` file:
  - It is not possible to use the `@` symbol in the username or password.
  - It is also not possible to use the percentage (%) sign in the password. When building the connection URL, the URL is parsed. This parsing exception happens because the percentage (%) sign acts as the escape character in URL parsing. If using the percentage (%) sign in the connection string is required, use the respective encoding character for the percentage (%) sign in the connection string. For example, if you need to pass adm%in as the password, then the % symbol should be encoded with its respective URL encoding character. Therefore, you have to send it as adm%25in.

For a list of possible URL parsing patterns, see [URL encoding reference](http://www.w3schools.com/xml/xml_url.asp).
**Recovering a password**

For information, see *How can I recover the admin password used to log in to the management console?*

**Login in via multiple user attributes in API Store**

For information, see *Authentication using multiple Attributes* in the WSO2 IS documentation.

**Setting up an e-mail login**

For information, see *Email Authentication* in the WSO2 IS documentation.

- When setting up email login, specify the complete username with tenant domain. If you are in the super tenant mode the username should be as follows. `<username>@<email>@carbon.super`
  
  **Example:** admin@wso2.com@carbon.super

- When configuring the `<DataPublisher>` section under `<ThrottlingConfiguration>` in the `<PRODUCT_HOME>/repository/conf/api-manager.xml` file, specify the fully qualified username with tenant domain.
  
  **Example:** <Username>admin@wso2.com@carbon.super</Username>

- The `@` character is a reserved character in the WSO2 messaging component. Therefore, when specifying username in JMS Connection URL, under `<JMSConnectionParameters>` section in the `<PRODUCT_HOME>/repository/conf/api-manager.xml` file, `@` characters should be replaced by `!` character. An example is shown below.

  ```xml
  <ConnectionFactory.TopicConnectionFactory><![CDATA[amqp://admin@wso2.com@carbon.super:admin@clientid/carbon?failover='roundrobin'&cyclecount='2'&brokerlist='tcp://10.100.0.3:5682?retries='5'&connectdelay='50';tcp://10.100.0.3:5692?retries='5'&connectdelay='50']]></ConnectionFactory.TopicConnectionFactory>
  ```

**Setting up a social media login**

You can auto-provision users based on a social network login by integrating the API Manager with WSO2 Identity Server. Refer *Log in to the API Store using Social Media* for more information.

---

Note that auto-provision users based on a social network login is not supported in a multi-tenant environment.

In a multi-tenant environment, the system cannot identify the tenant domain in the login request that comes to the API Manager’s Publisher/Store. Therefore, the service provider is registered as a SaaS application within the super tenant’s space. Configuring user provisioning is part of creating the service provider. In order to authenticate the user through a third party identity provider such as a social network login, you must enable identity federation. As the service provider is created in the super tenant’s space, the provisioned user is also created within the super tenant’s space. As a result, it is not possible to provision the user in the tenant’s space.

To overcome this limitation, you can write a custom authenticator to retrieve the tenant domain of the user and write a custom login page where the user can enter the tenant domain, which is then added to the authenticator context. Then, write a custom provisioning handler to provision the user in the tenant domain that is maintained in the context.

- For information on writing a custom authenticator, see *Creating Custom Authenticators* in the WSO2 IS documentation.
- For information on writing a custom login page, see *Customizing Login Pages* in the WSO2 IS documentation.

**Saving Access Tokens in Separate Tables**

This feature has been deprecated as it is redundant. Although it was introduced as a security measure, a compromise in the database would result in a compromise in all its tables.

You can configure the API Manager instances to store access tokens in different tables according to their user store domains. This is referred to as *user token partitioning* and it ensures better security when there are multiple user stores configured in the system. To configure user stores other than the default one, see *Configuring Secondary User Stores*.

The following topics explain how to enable user token partitioning:

- Enabling assertions
- Storing keys in different tables
Enabling assertions

You use assertions to embed parameters into tokens and generate a strong access token. You can also use these parameters later for other processing. At the moment, the API Manager only supports UserName as an assertion.

By default, assertions are set to `false` in `<APIM_HOME>/repository/conf/identity/identity.xml`. To enable it, set the `<UserName>` element to `true`. You can add a user name to an access token when generating the key, and verify it by encoding the retrieved access token with Base64.

```
<APIM_HOME>/repository/conf/identity/identity.xml

<EnableAssertions>
    <UserName>true</UserName>
</EnableAssertions>
```

Storing keys in different tables

1. If the `<UserName>` assertion is enabled, set the `<EnableAccessTokenPartitioning>` element in `<APIM_HOME>/repository/conf/identity/identity.xml` file to `true`. It determines whether you want to store the keys in different tables or not.

```
<EnableAccessTokenPartitioning>true</EnableAccessTokenPartitioning>
```

2. Set the user store domain names and mappings to new table names. For example,
   - if userId = foo.com/admin where 'foo.com' is the user store domain name, then a 'mapping:domain' combo can be defined as 'A:foo.com'
   - 'A' is the mapping for the table that stores tokens relevant to users coming from the 'foo.com' user store

   In this case, the actual table name is IDN_OAUTH2_ACCESS_TOKEN_A. We use a mapping simply to prevent any issues caused by lengthy table names when lengthy domain names are used. You must manually create the tables you are going to use to store the access tokens in each user store (i.e., manually create the tables IDN_OAUTH2_ACCESS_TOKEN_A and IDN_OAUTH2_ACCESS_TOKEN_B according to the following defined domain mapping). This table structure is similar to the IDN_OAUTH2_ACCESS_TOKEN table defined in the api-manager dbscript, which is inside the `<APIM_HOME>/dbscripts/apimgt` directory.

   You can provide multiple mappings separated by commas as follows. Note that the domain names need to be specified in upper case.

```
<AccessTokenPartitioningDomains>A:FOO.COM, B:BAR.COM</AccessTokenPartitioningDomains>
```

3. According to the information given above, change the `<OAuth>` element in the `<APIM_HOME>/repository/conf/identity/identity.xml` file as shown in the following example:

```
<!-- Assertions can be used to embed parameters into access token.-->  
<EnableAssertions>
    <UserName>true</UserName>
</EnableAssertions>

<!-- This should be set to true when using multiple user stores and keys should saved into different tables according to the user store. By default all the application keys are saved in to the same table. UserName Assertion should be 'true' to use this.-->
<EnableAccessTokenPartitioning>true</EnableAccessTokenPartitioning>

<!-- user store domain names and mappings to new table names. eg: if you provide 'A:foo.com', foo.com should be the user store domain name and 'A' represent the relevant mapping of token storing table, i.e. tokens relevant to the users coming from foo.com user store will be added to a table called IDN_OAUTH2_ACCESS_TOKEN_A. -->  
<AccessTokenPartitioningDomains>A:foo.com, B:bar.com</AccessTokenPartitioningDomains>
```

Configuring WSO2 Identity Server as the Key Manager

The **Key Manager** handles all clients, security and access token-related operations. For more information, see [Key Manager](#).
To configure WSO2 Identity Server as the Key Manager of the API Manager, see Configuring WSO2 Identity Server as a Key Manager.

**Configuring a Third-Party Key Manager**

The **Key Manager** handles all clients, security, and access token-related operations. In a typical API Manager production deployment, different components talk to the Key Manager component for achieving different tasks. The API Gateway connects with the Key Manager to check the validity of OAuth tokens, subscriptions, and API invocations. When a subscriber generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth App and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database. For more information, see Key Manager.

The Key Manager decouples the OAuth client and access token management from the rest of its operations, so that you can plug in a third-party OAuth provider for managing OAuth clients and access tokens. When working with an external Key Manager, you need to extend the required Key Manager interface(s), which are explained below, based on your requirements.

- **Key Manager interface** - This interface handles functionalities of the API Store. It contains methods to create, update, get, and delete OAuth2 applications, to map the existing consumer keys and secrets, and to generate the application access tokens. For more information, see Extending the Key Manager Interface.
- **Key Validation handler** - This interface handles functionalities of the Key Manager component. It contains methods to implement at API runtime to validate the token, subscriptions, and scopes, and also to generate JSON Web Tokens (JWTs). For more information, see Extending Key Validation.

Let's see what basic steps you need to follow when writing a Key Manager implementation that acts as the bridge between a third-party OAuth provider and WSO2 API Manager.

Follow the instructions below to configure the third-party Key Manager:

- **Step 1: Start the authorization server**
- **Step 2: Configure WSO2 API Manager**
- **Step 3: Run the sample**

**Step 1: Start the authorization server**

1. Download the binary located [here](#) and deploy it in a Tomcat server.

   Alternatively, you can build the OAuth Server from scratch and start the server by issuing the `mvn jetty:run` command in the `api-authORIZATION-server-war` folder. Detailed steps for building and starting the server are provided [here](#).

   **Tip**: The Surf OAuth web application that you just downloaded has the following customizations:
   - The `apis.application.properties` file is copied to the classpath.
   - All the URLs starting with `localhost` are replaced by the loop back IP (127.0.0.1).
   - `org.surfnet.oaaas.noop.NoopAuthenticator` authenticator is set as the default authenticator.
   - Token expiry time is increased to 99999 seconds. This ensures that the tokens issued for the web client last several months.

2. Move the web application to the ROOT context to ensure that the SurfOauth web applications work on Tomcat.

   ```
   rm -rf tomcat7/webapps/ROOT
   mv tomcat7/webapps/surf-oauth tomcat7/webapps/ROOT
   ```

3. Access [http://127.0.0.1:8080/](http://127.0.0.1:8080/) to see the following page:
The server is now up and running.

4. Follow the steps below to create a resource server.
   a. In Surf OAuth, click the **Resource Servers** link where all the OAuth clients are grouped together.
   b. Register a resource server representing WSO2 API Manager.
   c. Add two scopes named **test** and **scope1** and save your changes.
      You will use them when creating clients.

5. Follow the steps to create an OAuth Client.
   a. Click the **Access Tokens** link and note all the tokens issued for the web client.
      These tokens are obtained at the time you sign in, by a Javascript client running on the browser. The same token is then used for subsequent operations.
b. Pick an active access token from the above list. You use it to create clients through WSO2 API Manager.

c. Get a registration endpoint that is needed to register the client.

As Surf OAuth doesn’t support a spec-compliant client registration yet, you can use an endpoint with similar capabilities. For example, as shown below, you can enable Developer Tools in Google Chrome to see the URL and the request:

\[\text{Step 2: Configure WSO2 API Manager}\]

2. Copy the JAR files that you built in to the `<API-M_HOME>/repository/components/lib` directory.

If you are setting up a distributed environment, copy and paste the JAR files that you built in to the respective directories given below in the Key Manager node and the Store node respectively.

- API Key Manager - `<API-M_KEY_MANAGER_HOME>/repository/components/lib`
- API Store - `<API-M_STORE_HOME>/repository/components/lib`

3. Uncomment the `<APIKeyManager>` element in the `/repository/conf/api-manager.xml` file, which is in the API Key Manager and API Store and change the values based on your third-party implementation.

**Tip:** Be sure to replace the `<RegistrationEndpoint>` and `<AccessToken>` elements with the client registration endpoint and the access token you obtained earlier in step 7 and 6. ConsumerKey and Secret should be that of the created resource server. Also change the `<hostname>` in the `<IntrospectionURL>` accordingly.

The `nl.surfnet.demo.SurfOAuthClient` class, which is mentioned in the following example, extends the Key Manager interface.
### Step 3: Run the sample

You have connected WSO2 API Manager with a third-party authorization server. Let's see how WSO2 API Manager creates OAuth clients at Surf OAuth when applications are registered in the API Store. In this guide, we use Product APIs to test invoke this process.

1. Start WSO2 API Manager.
2. Sign in to the WSO2 API Store and create an application.

   ```bash
   -d 'action=login&username=admin&password=admin'
   -d 'action=addApplication&application=SurfClientApp&tier=Unlimited&description=&callbackUrl='
   ``

3. Register an OAuth client of the type PRODUCTION in the authorization server. As shown below, you need to send the specific parameters required by the OAuth Server in JSON.

   ```bash
   -d 'action=generateApplicationKey&application=SurfClientApp&authorizedDomains=ALL&keytype=PRODUCTION&validityTime=3600&jsonParams="scopes": ["test"], "contactName": "John Doe", "contactEmail": "john@doe.com"
   ```

4. Go to the Client Applications link in the Surf OAuth UI. Note the newly created client listed there.

   ![Client Applications](image)

### Example

```xml
<APIKeyManager>
  <KeyManagerClientImpl>nl.surfnet.demo.SurfOAuthClient</KeyManagerClientImpl>
  <Configuration>
    <RegistrationEndpoint><Give the client registration endpoint you got in step 7></RegistrationEndpoint>
    <AccessToken><Give the access token you got in step 6></AccessToken>
    <IntrospectionURL>http://<hostname>:port/v1/tokeninfo</IntrospectionURL>
    <ConsumerKey>xxx</ConsumerKey>
    <ConsumerSecret>xxx</ConsumerSecret>
  </Configuration>
</APIKeyManager>
```

For a sample on Key Manager implementation, see the WSO2 default Key Manager implementation.
You have now created an application and registered an OAuth Client corresponding to it.

5. Validate tokens by subscribing to a SurfClient application and obtaining a token.
   a. Sign in to the API Publisher and deploy the sample API (PizzaShackAPI) if you haven’t done so already.

b. Assuming you still have the OAuth client created earlier, subscribe to this API as follows:

   ```bash
   curl -k -X POST -b cookies
   'action=addAPISubscription&name=PizzaShackAPI&version=1.0.0&provider=admin&tier=Unlimited&applicationName=SurfClientApp'
   
   Let’s obtain a token from the OAuth Provider.
   c. Go to the Edit view of the OAuth client and make sure the client_credentials grant type is enabled, and a token expiration time is specified.
5. Obtain a token.
   Replace `<ConsumerKey:ConsumerSecret>` with the Base64 encoded `ConsumerKey:ConsumerSecret` of the client application you just created.

   ```
curl -k -d "grant_type=client_credentials&scope=test" -H "Authorization: Basic <ConsumerKey:ConsumerSecret>" -H "Content-Type: application/x-www-form-urlencoded"
   https://localhost:8243/token
   ```

   The Token endpoint format for above e, f, and g steps is: http://<surf-hostname>:port/v1/token. A sample change done in `_TokenAPI_.xml` is as follows:
Enabling Role-Based Access Control Using XACML

Many organizations expose their business capabilities through APIs. One of the key challenges is controlling access to these exposed APIs in such a way that all authorized users are able to access its APIs without any interruption, while at the same time making sure that any unauthorized users are kept out. In order to achieve this, parameters such as the user role can be used in determining whether to grant or deny access to an API for a given user. There are two ways to control access to users, namely OAuth 2.0 scope and XACML. This section explains how an external extensible Access Control Markup Language (XACML) entitlement server can be integrated with WSO2 API Manager to provide role-based access control to APIs exposed via WSO2 API Manager. XACML is a declarative access control policy language based on XML that can provide a standardized way of validating authorization requests.

WSO2 API Manager provides the capability to authorize users based on OAuth 2.0 tokens and this mechanism can be extended to provide role-based access control using OAuth 2.0 scopes. However, as opposed to using OAuth 2.0 scope to provide authorization, XACML provides a standardized way of validating authorization requests. Authorization policies can be written in a standardized way using XACML and can be stored and managed through a policy administration point (PAP). Since the policies are standardized, policies written to one XACML engine can be ported to another engine from a different vendor without any issue. Similarly, XACML provides more control on how access should be enforced as different parameters and possibilities can be evaluated. XACML also provides ‘Obligations’ and ‘Advice’ as part of the XACML response that can be used by the API Manager when enforcing the policy decision to implement fine-grained access control for APIs.

How XACML is used with WSO2 API Manager

The diagram shown below depicts the scenario where WSO2 API Manager uses the XACML entitlement server to validate API requests that come into the API Manager. In this case, WSO2 Identity Server has been used as the XACML entitlement server.
The process is initiated by an administrator who creates the XACML policies and adds them to the PAP. The created policies are stored in a policy repository and promoted to the policy decision point (PDP) by an authorized user. Once the policy is deployed, authorization requests are evaluated against this policy. There can be more than one policy deployed in the PDP.

The API Manager acts as the policy enforcement point (PEP). Whenever an API invocation comes to the API Manager, an authorization request is sent to the PDP with the required attributes. In this case, it can be the name of the user, resource path and the HTTP verb. The PDP receives the request along with these attributes and evaluates the request against the existing policies deployed in the PDP. If the request requires more information, the PDP tries to obtain that information from a policy information point (PIP). In this case, the request from the API Manager can contain the username and the policy that is deployed requires the role of the user. In such a scenario, the PDP gets this information from the user store that is defined as a PIP. Once the PDP has the required information to evaluate the request, a response is sent back to the API Manager with its policy decision.

**Enabling role-based access control**

The steps below demonstrate how WSO2 Identity Server, acting as a XACML entitlement server, can validate authentication requests from the API Manager based on a set of predefined XACML entitlement policies. This allows a standardized way of defining entitlement policies that can be enforced from WSO2 API Manager. For detailed information on XACML, see XACML Architecture in the WSO2 Identity Server documentation.

Let’s take the following requirement in exposing an API via the API manager.

<table>
<thead>
<tr>
<th>API Path</th>
<th>HTTP Verb</th>
<th>Who can access</th>
</tr>
</thead>
<tbody>
<tr>
<td>menu</td>
<td>GET</td>
<td>webuser</td>
</tr>
<tr>
<td>pizzashack/1.0.0</td>
<td>POST</td>
<td>admin</td>
</tr>
<tr>
<td>order</td>
<td>POST</td>
<td>webuser</td>
</tr>
<tr>
<td>order/{orderid}</td>
<td>GET</td>
<td>webuser</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>admin</td>
</tr>
</tbody>
</table>

Based on the requirement, a single API is exposed to add or retrieve order information. Each member type (webuser or admin) is identified from the resource path. The operation (GET or POST) that needs to be performed is distinguished by the HTTP verb. Follow the steps below to implement this kind of role-based access control.

1. Let’s start by creating the required users. First, you need to link both the API Manager and the Identity Server to the same user store in order to share users, roles and other related information. This can be done by linking the API manager with the LDAP user store within WSO2 Identity.
1. Start the WSO2 API Manager server and log in to the Management Console. Create user information with the following permission structure.

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>api_user</td>
<td>webuser</td>
</tr>
<tr>
<td>api_admin</td>
<td>admin</td>
</tr>
</tbody>
</table>

When adding the webuser role, set the Login and Subscribe permissions from permission tree.

2. Share the registry of both WSO2 API Manager and WSO2 Identity Server. Refer Sharing the registry space for the steps.

3. Start the WSO2 API Manager server and log in to the Management Console. Create user information with the following permission structure.

4. When adding the webuser role, set the Login and Subscribe permissions from permission tree.

5. Start the WSO2 Identity Server and log in to its Admin Console.

6. Under the Entitlement section, click Policy Administration > Add New Entitlement Policy.

By default, in API Manager JDBCUserStore is enabled. When you are moving to the ReadWriteLDAPUserStore, make sure you have commented the configuration of JDBCUserStore and keep only one user store configuration.

<PRODUCT_HOME>/repository/conf/user-mgt.xml in both nodes.

In an actual deployment, both these servers can share the user store of your organization.
6. You are redirected to a page listing all available policy editors. Select **Standard Policy Editor** from the list and add the values shown below in the policy editor. Refer to the [Creating a XACML Policy](#) in WSO2 Identity Server for more information.

7. **Entitlement Policy Name:** PizzaShackPolicy
6. **Rule Combining Algorithm:** Deny unless Permit

When the rule combination algorithm is set to **Deny Unless Permit**, you need to set the permit criteria as a rule.

8. In the **Define Entitlement Rule(s)** area, set the following 2 rules to define the kind of requests and from which user they should be permitted.

   a. **Rule Name:** AdminGrant
   
   b. **Conditions:** Under **Define your conditions by using followings...**, select drop down options as **Subject**, **is/are**, **at-least-one-member-of** in order and enter **admin** in the last field.

   *Click the icon next to **END** shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.*
Select the attributes as given below. Note that this needs to be done for all the rules.

**Select Attribute ID:** Role
**Select Attribute Data Type:** String
**Entitlement Data Module:** Carbon Attribute Finder Module

Click on **Add** button after providing above values as shown below.

b. GetOrder- allows web users to get order information from the API. Give the information below,

**Rule Name:** GetOrder
**Conditions:** Under **Rule's conditions are evaluated**..., select drop down options as **Action**, **is**, **equal** in order and enter **GET** in the last field.

Under **Define your conditions by using followings**..., select drop down options as **Subject**, **is/are**, **at-least-one-member-of** in order and enter **webuser** in the last field.

Click the icon next to **END** shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.
Select the attributes as given below. Note that this needs to be done for all the rules.

Select Attribute ID: Role
Select Attribute Data Type: String
Entitlement Data Module: Carbon Attribute Finder Module

Click on Add button after providing above values as shown below.

9. Click Add once done.

10. The rules are added to the policy. Click Finish to save the policy.
11. In the Policy Administration page, click Publish to My PDP to publish the policy to the PDP.

   ![Publish Policy](image_url)

   Keep the default selected values in the Publish Policy window and select publish.

   ![Policy Administration](image_url)

   You can test the service by clicking Try option in front of the entitlement policy. It is the tryIt tool developed for XACML in WSO2 Identity Server and you can access the same tryIt tool by navigating to Tools > XACML > TryIt.

   ![TryIt](image_url)

   You need to enter your username as the Subject Name. Refer Evaluating a XACML Policy for more information on how to use the TryIt tool for XACML Policy evaluation.

12. Download the entitlement-1.0-SNAPSHOT.jar and add it to the <API-M_HOME>/repository/components/lib directory. This JAR file contains the APIEntitlementCallbackHandler class which passes the username, HTTP verb and the resource path to the XACML entitlement server. If you want to view the source code of the JAR, go here.

13. Restart the server once the JAR file is added.

14. Now, you need to create a sequence containing the entitlement policy mediator that can be attached to each API required to authorize users with the entitlement server. Create an XML file with the following configuration and name EntitlementMediator.xml.
The Entitlement Mediator intercepts requests and evaluates the actions performed by a user against an eXtensible Access Control Markup Language (XACML) policy. Here, WSO2 Identity Server is used as the XACML Policy Decision Point (PDP) where the policy is set, and WSO2 API Manager serves as the XACML Policy Enforcement Point (PEP) where the policy is enforced. Refer Entitlement Mediator for more information on parameters and usage of this mediator.

The attributes in the `<entitlementService>` element above should be modified according to the services' endpoint configuration as follows.

- `remoteServiceUrl` - Service URL of WSO2 Identity Server, acting as the XACML entitlement server in this scenario.
- `remoteServiceUserName` - Username
- `remoteServicePassword` - Password used to connect to the service

15. Log in to the API Publisher and create an API.
16. Attach the custom sequence to the inflow of the message as shown below.

17. Save, publish and test the API to make sure that the requests specified in the 2 rules defined in step 8 are accessible according to the user role specified. For example, the POST operation is only available to users with the role `admin`. If an anonymous user tries to access the POST operation, it should fail.

If you encounter an error stating "org.apache.axis2.transport.jms.JMSSender cannot be found by axis2_1.6.1.wso2v16" when publishing the API, comment out the following JMSSender configuration in the `<APIM_HOME>/repository/conf/axis2/axis2_blocking_client.xml` file and restart the server.

```xml
<transportSender name="jms" class="org.apache.axis2.transport.jms.JMSSender"/>
```

18. If you want to debug the entitlement mediator, enable debug logs in the Management Console for the `org.wso2.sample.handlers.entitlement` package.
Encrypting OAuth Keys

WSO2 API Manager allows you to encrypt any sensitive OAuth2.0 keys that are created. The API Manager encrypts access tokens, client secrets and authorization codes (this can be extended to any other OAuth2.0 keys if needed) using the primary keystore. The result is encoded in Base64 and stored in the database. The RSA algorithm is used by default and the key strength (1024, 2048, etc) is based on the private key strength of the primary keystore. If SymmetricEncryption is enabled, the API Manager uses the AES algorithm by default, or the algorithm specified for the SymmetricEncryption in the carbon.xml file.

Symmetric Encryption is a form of encryption where the same key is used to encrypt and decrypt the message along with a mathematical algorithm. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.

It is recommended to switch this configuration on/off before any keys have been generated in your system. Once token encryption is switched on, the system encrypts all sensitive OAuth2.0 data such as Access Tokens, Consumer Secrets, etc. When reading that information, the system assumes that they are in the encrypted format and attempts to decrypt them. Therefore, switching this configuration on after any keys are created would break the system, unless the data is converted back into plain text.

In order to encrypt the OAuth keys, change the following configurations.

1. In the <APIM_HOME>/repository/conf/api-manager.xml file, set the <EncryptPersistedTokens> property to true.
3. Restart the server(s) after the above configuration changes are performed.

Tip

- If you use a Distributed API Manager setup, the changes must be made on both the API Store and Key Manager nodes.
- If you use WSO2 Identity Server (WSO2 IS) as the Key Manager setup, you need to make changes in both WSO2 IS and WSO2 API Manager.

Provisioning Out-of-Band OAuth Clients

When an application access token is generated, an OAuth client is created underneath. The consumer key and consumer secret shown under a key type actually belongs to the OAuth client. There can be situations where an OAuth client is created elsewhere, but needs to be associated with an application in the API Store. For instance, in an organization where WSO2 Identity Server is used as the authoritative server, OAuth clients may only be created through the Identity Server. Similarly, when a third party OAuth provider is used, users might want to use previously created OAuth clients with the API Manager. To achieve this, you can provision the OAuth clients created outside the API Store into the WSO2 API Manager (WSO2 APIM), thereby associating the OAuth client with an application in the API Store. Once the mapping is done, you can use it in the same way as an OAuth client created through the API Store.

Note that when you delete an application after Out-of-Band OAuth client is provisioned, the underlying OAuth client is not deleted. It just deletes the association of the OAuth client with the application. Therefore that OAuth client will be able to create an association to another application, which means we can map the same OAuth clients' keys to a new application created in the API Store.

The steps below describe how to provision OAuth clients created outside the API Store into the WSO2 APIM:

In this example, we use a standalone API Manager instance and do this via the WSO2 APIM Management Console.

In a setup where WSO2 Identity Server is used as the Key Manager, this step is performed in the Identity Server.

1. Sign in to the WSO2 APIM Management Console (https://<Server Host>:9443/carbon) and click Add under Service Providers.
2. Enter the name of the service provider and click Register.

3. Click Configure under Inbound Authentication Configuration > OAuth/OpenId Connect Configuration to add a new OAuth client.
4. Provide a callback URL and click Add. If you do not have a callback URL, you can clear the Code and Implicit authorization grant types and add the OAuth client.

You have now created the OAuth client and are provided with the OAuth client key and OAuth client secret.
5. Enable the option to provide out-of-band keys by opening the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` file and changing the "mapExistingAuthApps" setting to true.

```
"mapExistingAuthApps" : true
```

Note that the ability to provision Out-of-Band Auth client will only be available for the applications that you created after doing this configuration.

6. Sign into the WSQ2 API Store.
   https://<Server Host>:9443/store

7. Click Applications.

8. Click on the respective application to view the subscriptions details for the application.


The following steps explain how you can provision an out-of-band OAuth client for the production environment. If you wish to generate keys for your sandbox, you can follow the steps below using the Sandbox Keys tab.

a. Click Production Keys.
   Notice that you now see a Provide Keys button for your application.
b. Click **Provide Keys**, paste the consumer key and consumer secret pair, which you received in step 4, and click **Save**.

You have successfully provisioned an out-of-band OAuth client.

**Basic Auth**

When you create an API using the API Publisher, you can specify the endpoint of the API backend implementation in the **Implement** tab as Production and Sandbox endpoints.

If this endpoint is secured, there is an option for you to set the Auth type and credentials for the endpoint under **Show More Options**.
Here you can click the Show More Options link to select the endpoint security scheme. If you select Secured, you are prompted to select the authentication type for the endpoint and also to give its credentials.

Then select the endpoint authentication type according to the authentication scheme that is supported by your endpoint. If your endpoint supports basic authentication, you can select the Basic Auth option from the drop down list and give your credentials.

To give more context on the above scenario, a secured endpoint is where we have access-protected resources. You have to specify the username and the password when a request is sent to a secured endpoint. The endpoint authentication mechanism can either be Basic Authentication or Digest Authentication. They differ on how the credentials are communicated and how access is granted by the backend server.

Basic Authentication is the simplest mechanism used to enforce access controls to web resources. Here, the HTTP user agent provides the username and the password when making a request. The string containing the username and the password separated by a colon is Base64 encoded and sent in the authorization header when calling the backend when authentication is required.

If the user name and password is admin, the following header will be sent to the backend

Authorization: Basic YWRtaW46YWRtaW4= (where YWRtaW46YWRtaW4= is equivalent to Base64Encoded(admin:admin))
Digest Auth

When you create an API using the API Publisher, you can specify the endpoint of the API backend implementation in the Implement tab as Production and Sandbox endpoints.

If this endpoint is secured, there is an option for you to set the Auth type and credentials for the endpoint under Show More Options.

Click Show More Options to select the endpoint security scheme. If you select Secured, you are prompted to select the authentication type for the endpoint and also to give its credentials.

Select the endpoint authentication type according to the authentication scheme that is supported by your endpoint. If your endpoint supports Digest authentication, you can select the Digest Auth option from the drop down list and give your credentials.

To give more context on the above scenario, a secured endpoint is where we have access-protected resources. You have to specify the username and the password when a request is sent to a secured endpoint. The endpoint authentication mechanism can either be Basic Authentication or Digest Authentication. They differ on how the credentials are communicated and how access is granted by the backend server.

The selected Endpoint Auth Type should match with the authentication mechanism supported by the secured endpoint.

Digest Authentication applies a hash function to the username and the password before sending them over the network. It is a process of applying MD5 cryptographic hashing with the usage of nonce values to prevent replay attacks. It is a simple challenge-response authentication mechanism that may be used by a server to challenge a client request and by a client to provide authentication information for the secured endpoint.

This approach is safer than Basic Authentication, which uses unencrypted base64 encoding instead of a hashing mechanism.
Working with Encrypted Passwords

All WSO2 products are shipped with a Secure Vault implementation that allows you to store encrypted passwords that are mapped to aliases. This approach allows you to use the aliases instead of the actual passwords in your configurations for better security. For example, some configurations require the admin username and password. If the admin user's password is "admin", you could use `UserManager.AdminUser.Password` as the password alias.

You will then map that alias to the actual "admin" password using Secure Vault. The WSO2 product will then look up this alias in Secure Vault during runtime, decrypt and use its password.

In all WSO2 products, Secure Vault is commonly used for encrypting passwords and other sensitive information in configuration files. When you use WSO2 API-M, you can encrypt sensitive information contained in synapse configurations (i.e., mediation flow) in addition to the information in configuration files. For more information, see the following topics:

- Encrypting passwords in configuration files
- Encrypting passwords for mediation flow
- Using encrypted passwords in mediation flow

Encrypting passwords in configuration files

To encrypt passwords in configuration files, you simply have to update the `cipher-text.properties` and `cipher-tool.properties` files that are stored in the `<API-M_HOME>/conf/security` directory and then run the Cipher tool that is shipped with the product. Go to the links given below to see instructions in the WSO2 administration guide:

- Encrypting passwords using the automated process.
- Encrypting passwords using the manual process.
  - The manual encryption process is relevant when the location of the configuration files, which contain the elements to be encrypted, cannot be specified using an xpath in the `cipher-tool.properties` file. The `log4j.properties` and `jndi.properties` file are two such files which require the manual password encryption process.
- Changing already encrypted passwords.
- Resolving already encrypted passwords.

Encrypting passwords for mediation flow

Before you begin

If you are using Windows, you need to have Ant (http://ant.apache.org/) installed before using the Cipher Tool.

The WSO2 API-M provides a UI that can be used for encrypting passwords and other sensitive information in synapse configurations. Follow the steps below:

1. If you are using the Cipher tool for the first time in your environment, you must first enable the Cipher tool by executing the `-Dconfigure` command with the cipher tool script:
   a. Open a terminal and navigate to the `<API-M_HOME>/bin` directory.
   b. Execute one of the following commands based on your OS:
      - On Linux: `./ciphertool.sh -Dconfigure`
      - On Windows: `./ciphertool.bat -Dconfigure`

   If you are using the Cipher tool for the first time, this command first initializes the tool for your product.

2. When prompted, enter the primary key password, which is by default `wso2carbon`.
   Enter the password and proceed.

3. When prompted, enter the plain text password that you want to encrypt.
   Enter the following element as the password and proceed.

---

For more information on the Secure Vault implementation in WSO2 products, see Carbon Secure Vault Implementation.

---

Authorization: Digest username="Admin", realm="admin@wso2.com", nonce="dc9b87102dd2f0e8b1df0600bfb0c093", uri="/dir/index.html", qop=auth, nc=00000001, cnonce="0ad4f113b", response="6629fae49393a05397450978507c4e1", opaque="5ccc069c403ebaf9f0171e9517f40e41"
Enter Plain Text Value :admin

Now, you will receive the encrypted value.

Example

```
Encrypted value is:
ghkPzT assassHills123Xpml1i4H1OM/srL5pB8yjknnRKQ2s7TNuCvt1+qEkE1LWwlroh3z1kuE0KFuapXcCSs5pxfGM0Lr4/k7dNa281wbsG8C++/
ZfuUt1t116c7v0RM55fQwzCPfybl713hVb3aJ9VkgSbvh11q6zqzg=
```

4. Start WSO2 API-M and sign in to the management console:
   a. Open a terminal and navigate to the `<API-M_HOME>/bin` directory.
   b. Execute one of the following scripts:
      - On Windows: wso2server.bat --run
      - On Linux/Mac OS: sh wso2server.sh
   c. Sign in to the management console.
      `https://<server-host>:9443/carbon`

5. Select Browse under Resources to access the registry browser and go to the `/_system/config/repository/components/secure-vault` location.

6. Add the aliases and the encrypted value as a property.

---

### Encrypting passwords in api-manager.xml file

The `api-manager.xml` file does not contain any plaintext passwords. Therefore, encrypting the plaintext passwords in `user-mgt.xml` file as shown in *Encrypting Passwords with Cipher Tool* is sufficient. We recommend having the same admin user on all nodes in a clustered setup.

However, if you need to use an alias to encrypt passwords in the `<API-M_HOME>/repository/conf/api-manager.xml` file using the cipher tool, do the following.

To derive the alias, ignore the root XML element (i.e., `<APIManager>`) in the `api-manager.xml` file. Denote the subsequent XML elements separated by a dot (.), according to the hierarchy. You can derive the alias `AuthManager.Password` from the XML example given below.

```
<APIManager xmlns:svns="http://org.wso2.securevault/configuration">
  <DataSourceName>jdbc/WSO2AM_DB</DataSourceName>
  <GatewayType>Synapse</GatewayType>
  <EnableSecureVault>false</EnableSecureVault>
  <AuthManager>
    <Username>/${admin.username}</Username>
    <Password>/${admin.password}</Password>
  </AuthManager>
</APIManager>
```

Following are some sample aliases derived for other passwords in the `api-manager.xml` file:

- ThrottlingConfigurations.DataPublisher.Password
- ThrottlingConfigurations.PolicyDeployer.Password
- ThrottlingConfigurations.jmsConnectionDetails.Password
- Analytics.DASPassword
- Analytics.DASRestApiPassword

---

Using encrypted passwords in mediation flow
To use the alias of an encrypted password in a mediation flow, you need to add the \{wso2:vault-lookup('alias')\} custom path expression when you define the mediation flow. For example, instead of hard coding the admin user's password as \<Password>admin</Password>\, you can encrypt and store the password using the adminUser.Password alias as follows:

\<Password>{wso2:vault-lookup('adminUser.Password')}</Password>.

This password in the mediation flow can now be retrieved by using the \{wso2:vault-lookup('alias')\} custom path expression to logically reference the password mapping.

**Encrypting Secure Endpoint Passwords**

When creating an API using the API Publisher, you specify the endpoint of its backend implementation in the **Implement** tab. If you select the endpoint as secured, you are prompted to give credentials in plain-text.

**Test: /test/1.0.0**

The steps below show how to secure the endpoint's password that is given in plain-text in the UI.

1. Shut down the server if it is already running and set the element `<EnableSecureVault>` in the `<APIM_HOME>/repository/conf/api-manager.xml` file to `true`. By default, the system stores passwords in configuration files in plain text because this value is set to `false`.
2. Run the cipher tool available in the `<APIM_HOME>/bin` directory. If you are running Windows, it is the `ciphertool.bat` file. If you are using the default keystore, give `wso2carbon` as the primary keystore password when prompted.

```
sh ciphertool.sh -Dconfigure
```

3. Restart the server (Publisher) after the above steps have been performed. From there onwards, the Basic Authentication header which is written to the API definition xml file will be encrypted. For APIs which were already created and published before this step was performed, an update to the particular API would trigger the encryption process of the credentials. For an example, see below for example of the same API when endpoint password is not encrypted and encrypted:

```
Not Encrypted
```

```
Encrypted
```

Here, the Basic authentication header is in base64 encoded format and can be decoded to get the actual credentials of the endpoint.
<property name="Authorization" expression="fn:concat('Basic ', 'dGVzdDp0ZXN0MTIz')" scope="transport"/>

Here, the password is first looked up from the secret repository, and then set as a transport header.

<property name="password" expression="wso2:vault-lookup('<api-identifier>')"/>
<property name="unpw" expression="fn:concat('test',':',get-property('password'))"/>
<property name="Authorization" expression="fn:concat('Basic ', base64Encode(get-property('unpw')))" scope="transport"/>

Mutual SSL between API Gateway and Backend

In contrast to the usual one-way SSL authentication where a client verifies the identity of the server, in mutual SSL the server validates the identity of the client so that both parties trust each other. This builds a system that has a very tight security and avoids any requests made to the client to provide the username/password, as long as the server is aware of the certificates that belong to the client.

This section explains how to secure your backend by enabling mutual SSL between the API Gateway and your backend. To establish a secure connection with the backend service, API Manager needs to have the public key of the backend service in the truststore. Similarly, the backend service should have the public key of API Manager in the truststore.

- Export the certificates
- Configure API Manager to enable dynamic SSL profiles
- Test Mutual SSL between API Gateway and backend

Export the certificates

1. Generate the keys for the backend. A sample command is given below.

   keytool -keystore backend.jks -genkey -alias backend

   The keystore will be generated in your target folder.

2. Export the certificate from the keystore. A sample command is given below.

   keytool -export -keystore backend.jks -alias backend -file backend.crt

3. Import the generated backend certificate to the API Manager truststore file as shown below

   keytool -import -file backend.crt -alias backend -keystore</APIM_HOME>/repository/resources/security/client-truststore.jks

4. Export the public certificate from API Manager’s keystore. The <APIM_HOME>/repository/resources/security/wso2carbon.jks file which is the default keystore shipped with WSO2 API Manager is used in this example. Use the command below to generate the certificate for the default keystore. Give the default password wso2carbon when prompted.

   keytool -export -keystore wso2carbon.jks -alias wso2carbon -file wso2PubCert.cert

   To change the default keystore, generate a keystore file and copy it to the <APIM_HOME>/repository/resources/security folder. After copying the keystore, generate the certificate as shown in step 2.

5. Import the generated certificate to your backend truststore.
You have now successfully exported the certificates for mutual SSL.

**Configure API Manager to enable dynamic SSL profiles**

To configure API Manager for Dynamic SSL Profiles for HTTPS transport Sender, you need to create a new XML file `<APIM_HOME>/repository/deployment/server/multi_ssl_profiles.xml` (this path is configurable) and copy the below configuration into it. This will configure client-truststore.jks as Trust Store for all connections to `<localhost:port>`.

```xml
<parameter name="customSSLProfiles">
  <!-- For SSL Handshake configure only trust store-->
  <profile>
    <servers>localhost:port</servers>
    <TrustStore>
      <Location>repository/resources/security/client-truststore.jks</Location>
      <Type>JKS</Type>
      <Password>wso2carbon</Password>
    </TrustStore>
  </profile>

  <!-- For Mutual SSL Handshake configure both trust store and key store-->
  <profile>
    <servers>10.100.5.130:9444</servers>
    <TrustStore>
      <Location>repository/resources/security/client-truststore.jks</Location>
      <Type>JKS</Type>
      <Password>wso2carbon</Password>
    </TrustStore>
    
    <KeyStore>
      <Location>repository/resources/security/wso2carbon.jks</Location>
      <Type>JKS</Type>
      <Password>xxxxxx</Password>
      <KeyPassword>xxxxxx</KeyPassword>
    </KeyStore>
  </profile>
</parameter>
```

To enable dynamic loading of this configuration, add below configurations to the Transport Sender configuration `<APIM_HOME>/repository/conf/axis2.xml`. Set above file’s path as the `filePath` parameter.

```xml
<parameter name="dynamicSSLProfilesConfig">
  <filePath>repository/deployment/server/multi_ssl_profiles.xml</filePath>
  <fileReadInterval>3600000</fileReadInterval>
</parameter>
```

Now both the backend service and ESB is configured to use default key stores and API Manager is configured to load dynamic SSL profiles. Restart API Manager.

**You can start API Manager using the following options, to see the SSI debug logs.**

-Djavax.net.debug=ssl:handshake
-Djavax.net.debug=all
-Djavax.net.debug=all:handshake:verbose
**Test Mutual SSL between API Gateway and backend**

You can do the following to test your mutual SSL configurations:

1. Create an API
2. Subscribe to the API
3. Invoke the API

**Securing APIs**

APIs published on the WSO2 API Gateway are secured using OAuth2.0 by default. Any client application invoking a secure published API needs to have a valid subscription to the particular API and present a valid OAuth2.0 Access Token to the API Gateway. Please see steps 7 and 8 of the quick start guide to understand how you can subscribe an application to an API and how to get credentials for your application.

Once you have the got the required credentials, namely the consumer key and consumer secret, for your application, you (application users) can get access tokens to invoke APIs that are subscribed to the particular application. To understand how you can get tokens for different types of applications, see Token API.

**Authentication**

HTTP Authorization header is the most common method of providing authentication information for REST APIs. The application needs to have the access token in an authorization header for the client application to authenticate the API that is being accessed. The format of the header is as follows.

**Default Format**

```
Authorization : Bearer <access-token>
```

**Format for customized authorization header**

```
<customized-authorization-header> : Bearer <access-token>
```

**Example with default authorization header**

```
Authorization : Bearer NtBQkXoKElu0H1a1fQ0DWfo6IX4a
```

**Example with customized authorization header**

```
SERVER_AUTH : Bearer NtBQkXoKElu0H1a1fQ0DWfo6IX4a
```

The string `NtBQkXoKElu0H1a1fQ0DWfo6IX4a` is a sample value of the access token that is being sent from the client application.

**Customizing the authorization header**

If the value of the authorization header needs to be changed due to organizational policies, legacy APIs, or for any other reason, it can be done at three levels in WSO2 API Manager (WSO2 API-M) as described below.

- Server wide customization
- Tenant wide customization
- Per API customization

**Server wide customization**

Follow the instructions below to change the value of the authorization header at the server level configurations:

1. Navigate to the `<API-M_HOME>/repository/conf/api-manager.xml` file and make the following changes.
   a. Uncomment the `<AuthorizationHeader>` section and add the customized authorization header.

   You need to make this change on all the profiles of WSO2 API Manager.
### FormatExample

```xml
<AuthorizationHeader>{authorization-header}</AuthorizationHeader>
```

```xml
<AuthorizationHeader>SERVER_AUTH</AuthorizationHeader>
```

**b. Add the custom authorization header to the list of allowed headers defined in the** `<CORSConfiguration>` **section.**

```xml
<Access-Control-Allow-Headers>{authorization-header}</Access-Control-Allow-Headers>
```

```xml
<Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction,SERVER_AUTH</Access-Control-Allow-Headers>
```

2. Restart the WSO2 API Manager server to reload the changes.
3. If you have already published APIs, sign in to the API Publisher and republish those APIs.

![Click here to test out the customized authorization header at the server level.](image)

---

**Before you begin,** deploy the sample PizzaShackAPI as explained in the Quick Start Guide, as the following example is based on that API.

#### Sample cURL Request

```bash
curl -H "<customized-authorization-header>: Bearer <access-token>" -H "accept: application/json" "https://<server-IP>:<port>/pizzashack/1.0.0/menu" -k -v
```

```bash
curl -H "SERVER_AUTH: Bearer 3c536e3f-397c-3df9-a89c-9c40efedfa9e" -H "accept: application/json" "https://10.100.0.112:8243/pizzashack/1.0.0/menu" -k -v
```
Note that the customized authorization header appears in the Store UI as well.

Tenant wide customization

Each tenant can define their own name for the authorization header by following the instructions below:
The tenant wide authorization header customizations take precedence over the server wide authorization header customizations.

1. Sign in to the WSO2 API Manager Management Console with your tenant credentials.
   https://<server-host>:9443/carbon

   For information on creating a tenant, see Managing Tenants.

2. Click Main, then click Browse which is under Resources.

3. Enter /_system/config/apimgt/applicationdata/tenant-conf.json as the location and click Go to access the tenant-conf.json file that is in the WSO2 Registry.
4. Click Edit as text to be able to edit the JSON file.
5. Add the following configuration with the customized authorization header to the JSON file and save the file.

   Format: 
   ```
   "AuthorizationHeader" : "<tenant-authorization-header>"
   
   "AuthorizationHeader" : "TENANT_AUTH"
   
   {
   
   "EnableMonetization":false,
   "IsUnlimitedTierPaid":false,
   "ExtensionHandlerPosition":"bottom",
   "RESTAPIScopes":{
   "Scope":{
   "Name":"apim:api_publish",
   "Roles":"admin,Internal/publisher"
   },
   
   "Name":"apim:api_create",
   "Roles":"admin,Internal/creator"
   },
   
   "Name":"apim:api_view",
   "Roles":"admin,Internal/publisher,Internal/creator"
   },
   
   "Name":"apim:subscribe",
   "Roles":"admin,Internal/subscriber"
   },
   
   "Name":"apim:tier_view",
   "Roles":"admin,Internal/publisher,Internal/creator"
   ```
   
   The tenant wide authorization header customizations take precedence over the server wide authorization header customizations.
{
    "Roles": "admin",
    "Name": "apim:subscription_view",
    "Roles": "admin, Internal/creator"
},
{
    "Name": "apim:subscription_block",
    "Roles": "admin, Internal/creator"
},
{
    "Name": "apim:mediation_policy_view",
    "Roles": "admin"
},
{
    "Name": "apim:mediation_policy_create",
    "Roles": "admin"
},
{
    "Name": "apim:api_workflow",
    "Roles": "admin"
}
],
"NotificationsEnabled": "false",
"Notifications": [
    {
        "Type": "new_api_version",
        "Notifiers": [
            {
                "Class": "org.wso2.carbon.apimgt.impl.notification.NewAPIVersionEmailNotifier",
                "ClaimsRetrieverImplClass": "org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever",
                "Template": "<html><body><h3 style="color:Black;">We're happy to announce the arrival of the next major version $2 of $1 API which is now available in Our API Store.</h3><a href="https://localhost:9443/store">Click here to Visit WSO2 API Store</a></body></html>""
            }
        ]
    }
],
"DefaultRoles": [
    "PublisherRole": {
        "CreateOnTenantLoad": true,
        "RoleName": "Internal/publisher"
    },
    "CreatorRole": {
        "CreateOnTenantLoad": true,
        "RoleName": "Internal/creator"
    },
    "SubscriberRole": {
        "CreateOnTenantLoad": true
    }
]
6. If you have already published APIs, sign in to the API Publisher using your tenant credentials and republish those APIs.

Click here to test out the customized authorization header at the tenant level.

Before you begin, deploy the sample PizzaShackAPI as explained in the Quick Start Guide, as the following example is based on that API.

**Sample cURL Request**

```bash
curl -H "<customized-authorization-header>: Bearer <access-token>" -H "accept: application/json" "https://<server-IP>:<port>/pizzashack/1.0.0/menu" -k -v
```

```bash
curl -H "TENANT_AUTH: Bearer 3c536e3f-397c-3df9-a89c-9c40efeda9e" -H "accept: application/json" "https://10.100.0.112:8243/pizzashack/1.0.0/menu" -k -v
```

> GET /t/test.com/pizzashack/1.0.0/menu HTTP/1.1
> Host: 10.100.0.112:8243
> User-Agent: curl/7.54.0
> accept: application/json
> TENANT_AUTH: Bearer 57aca573-6fa8-3b03-a340-dd662b44fe8
> HTTP/1.1 200 OK
> Access-Control-Allow-Origin: *
> Access-Control-Allow-Methods: GET
> Access-Control-Allow-Headers: authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction,SERVER_AUTH,TENANT_AUTH
> Content-Type: application/json
> Date: Fri, 19 Jan 2018 06:35:47 GMT
> Transfer-Encoding: chunked
>
> [{"name":"BBQ Chicken Bacon","description":"Grilled white chicken, hickory-smoked bacon and fresh sliced onions in barbeque sauce","price":"22.99","icon":"/images/6.png"},
{name":"Chicken Parmesan","description":"Grilled chicken, fresh tomatoes, feta and mozzarella cheese","price":"21.99","icon":"/images/1.png"},
{name":"Chilly Chicken Cordon Bleu","description":"Spinach Alfredo sauce topped with grilled chicken, ham, onions and mozzarella","price":"28.99","icon":"/images/10.png"},
{name":"Double Bacon 6Cheese","description":"Hickory-smoked bacon, Julienne cut Canadian bacon, Parmesan, mozzarella, Romano, Asiago and and Fontina cheese","price":"25.99","icon":"/images/9.png"},
{name":"Garden Fresh","description":"Slices onions and green peppers, gourmet mushrooms, black olives and ripe Roma tomatoes","price":"14.99","icon":"/images/3.png"},
{name":"Grilled Chicken Club","description":"Grilled white chicken, hickory-smoked bacon and fresh sliced onions topped with Roma tomatoes","price":"15.99","icon":"/images/8.png"},
{name":"Hawaiian BBQ Chicken","description":"Grilled white chicken, hickory-smoked bacon, barbeque sauce topped with sweet pine-apple","price":"19.99","icon":"/images/7.png"},
{name":"Spicy Italian","description":"Pepperoni and a double portion of spicy Italian sausage","price":"14.99","icon":"/images/2.png"},
{name":"Spinach Alfredo","description":"Rich and creamy blend of spinach and garlic Parmesan with Alfredo sauce","price":"20.99","icon":"/images/5.png"},
{name":"Tuscan Six Cheese","description":"Six cheese blend of mozzarella, Parmesan, Romano, Asiago and Fontina","price":"22.99","icon":"/images/4.png"}]
```

Note that the customized authentication header appears in the Store UI as well.
Per API customization

The API Publisher application allows an API Developer or Product Manager to specify the name of the authorization header when creating or modifying an API.

The customized authorization header defined per API takes precedence over the customized authorization headers that are defined server and tenant wide.

Follow the instructions below to add a customized authorization header for an API:

1. Sign in to the Publisher.  
   https://<server-host>:9443/publisher
2. When creating or updating an API, define the customized authorization header in the Manage tab.  
   UI without tenants UI with tenants
The subscription field appears only if you have any tenants.

3. Save and Publish the API.

Click here to test out the customized authorization header at the API level.

Before you begin, deploy the sample PizzaShackAPI as explained in the Quick Start Guide, as the following example is based on that API.

Sample cURL Request

```
curl -H "<customized-authorization-header>: Bearer <access-token>" -H "accept: application/json" "https://<server-IP>:<port>/pizzashack/1.0.0/menu" -k -v
```

```
curl -H "PER_API_AUTH: Bearer 045d4999-8a9e-3897-945b-e7bea13d78fe" -H "accept: application/json" "https://10.100.0.112:8243/pizzashack/1.0.0/menu" -k -v
```
Multi Factor Authentication (MFA) for Publisher and Developer Portals

Multi factor authentication provided a many-layered security for a user identity where even if one factor is compromised, an attacker cannot gain full access to the target due to the remaining authentication factors.

To configure MFA for Publisher and Store with WSO2 Identity Server, please follow the steps below:

1. Configure Single Sign On (SSO) with WSO2 Identity Server. Please see the documentation on Configuring Identity Server as IDP for SSO.
2. Configure Multi Factor Authentication in WSO2 Identity Server. Please see Configuring Multi-factor Authentication for WSO2 IS.

Gateway Threat Protectors for API Manager

WSO2 API Manager has three types of threat protectors for the Gateway.

- Regular Expression Threat Protection for API Gateway
- JSON Threat Protection for API Gateway
- XML Threat Protection for API Gateway

Combining threat protectors

You can use a combination of the threat protectors given above to validate the messages and protect your gateway from attacks. An example custom mediation policy which which validates the API request against XML and regex validators is given below.
Add a custom sequence

You can add custom sequences depending on the threats that you need to address. To add a custom sequence, do the following.

1. Create an xml file with your custom sequence, or edit and save the sequence given above.
2. Go to Message Mediation Policies as described in the previous section.
3. Click Upload In Flow.
4. Select and upload your custom sequence.
Regular Expression Threat Protection for API Gateway

WSO2 API Manager provides pre-defined regex patterns to sanitize the request from SQL injection attacks. The attacks may depend on the API traffic at runtime. The API developers should identify the common attacks and select the appropriate restrictive measures. This feature extracts the data from XML, JSON payloads, Queryparam, URI path, headers and validates the content against pre defined regular expressions. If any predefined regex keyword is matched with the content, the API request is considered as a threat and it is blocked and rejected. This secures the backend resources from activities that make the system vulnerable. You can configure your own restriction patterns to thwart various attacks such as the following:

- Javascript Injection
- Server-side Include Injection
- XPath Injection
- Java Exception Injection
- XPath Abbreviated Syntax Injection

Blacklisting patterns

We recommend the following patterns for blacklisting.

<table>
<thead>
<tr>
<th>Name</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Injection</td>
<td>&quot;.<em>'.</em></td>
</tr>
<tr>
<td>Server-side Include</td>
<td>.&quot;#include.&quot;</td>
</tr>
<tr>
<td>Injection Attack</td>
<td>.&quot;Exception in thread.&quot;</td>
</tr>
<tr>
<td>Java Exception Injection</td>
<td>.&quot;.<em>'.</em></td>
</tr>
<tr>
<td>Javascript Exception</td>
<td>&lt;\s<em>script\b[^&gt;]</em>&gt;[^&lt;]<em>\s</em>&lt;/script\s*&gt;</td>
</tr>
<tr>
<td>XPath Expanded Syntax</td>
<td>/{ancestor(-or-self)?</td>
</tr>
</tbody>
</table>

- Editing the sequence through registry artifacts
- Applying the Regular Expression Policy
- Testing the regex threat protector

Editing the sequence through registry artifacts

To edit the existing sequence follow the steps below.

1. Log in to the Management Console.
2. Navigate to `/system/governance/apimgt/customsequences/in/regex_policy.xml`
3. Edit the `regex_policy.xml` file.
4. Go to the API Publisher and re-publish your API for the changes to take effect.

Applying the Regular Expression Policy

You can apply the pre-defined Regular Expression Policy through the UI. Follow the instructions below to apply the regex_policy in sequence.

1. Create an API or edit an existing API.
2. Go to Message Mediation Policies under the Implement tab.
3. Select Enable Message Mediation. Select `regex_policy` from the drop-down menu for In Flow.
4. Click **Save and Publish** to save the changes.

Each request is sanitized through the regular expression threat protector. You can add or modify the regex patterns according to your requirement.

The `regex_policy` sequence is given below.

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="regex_policy">
  <log level="custom">
    <property name="IN_MESSAGE" value="Regular_expression_policy"/>
  </log>
  <property name="threatType" expression="get-property('threatType')" value="SQL-Injection"/>
  <property name="regex" expression="get-property('regex')" value=".*'.*|.*ALTER.*|.*ALTER TABLE.*|.*CREATE DATABASE.*|.*CREATE PROCEDURE.*|.*CREATE SCHEMA.*|.*CREATE VIEW.*|.*DELETE.*|.*DROP DATABASE.*|.*DROP PROCEDURE.*|.*DROP.*|.*SELECT.*|.*create table.*|.*CREATE VIEW.*|.*ALTER VIEW.*|.*ALTER.*|.*CREATE.*">
    <property name="enabledCheckBody" expression="get-property('checkBodyEnable')" value="true"/>
    <property name="enabledCheckHeaders" expression="get-property('enabledCheckHeaders')" value="true"/>
    <property name="enabledCheckPathParams" expression="get-property('enabledCheckPathParams')" value="true"/>
  </property>
  <class name="org.wso2.carbon.apimgt.gateway.mediators.RegularExpressionProtector"/>
</sequence>
```

If you need to validate only the request headers, you can disable the `enabledCheckBody` and `enabledCheckPathParams` properties by setting the value to `false`.

**Testing the regex threat protector**

You can test this feature by sending an SQL injection attack with the XML message body. The sample request and response is given below.
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
  <food>
    <name>Homestyle Breakfast</name>
    <price>drop table</price>
    <description>
      Two eggs, bacon or sausage, toast, and our ever-popular hash browns
    </description>
    <calories>950</calories>
  </food>
</breakfast_menu>

<am:fault xmlns:am="http://wso2.org/apimanager">
  <am:code>400</am:code>
  <am:message>Bad Request</am:message>
  <am:description>SQL-Injection Threat detected in Payload</am:description>
</am:fault>

Performance impact

The regex mediator builds the entire message and performs string processing to find potentially harmful constructs underneath the message body. This drops the performance of 10KB messages for 300 concurrent users by 3.6 times than the normal flow. The performance decrease may accelerate along with the message size.

JSON Threat Protection for API Gateway

The JSON threat protector in WSO2 API Manager validates the request body of the JSON message based on pre-configured to thwart payload attacks.

- Editing the sequence through registry artifacts
- Applying the JSON validator policy
- Testing the JSON threat protector

Detecting vulnerabilities before parsing the message

The json_validator sequence specifies the properties to be limited in the payload. A sample json_validator sequence is given below.

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="json_validator">
  <log level="custom">
    <property name="IN_MESSAGE" value="json_validator"/>
  </log>
  <property name="maxPropertyCount" value = "100"/>
  <property name="maxStringLength" value = "100"/>
  <property name="maxArrayElementCount" value = "100"/>
  <property name="maxKeyLength" value = "100"/>
  <property name="maxJsonDepth" value = "100"/>
  <property name="RequestMessageBufferSize" value="1024"/>
  <class name="org.wso2.carbon.apimgt.gateway.mediators.JsonSchemaValidator"/>
</sequence>
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxPropertyCount</td>
<td>100</td>
<td>Maximum number of properties</td>
</tr>
<tr>
<td>maxStringLength</td>
<td>100</td>
<td>Maximum length of string</td>
</tr>
<tr>
<td>maxArrayElementCount</td>
<td>100</td>
<td>Maximum number of elements in an array</td>
</tr>
<tr>
<td>maxKeyLength</td>
<td>100</td>
<td>Maximum number length of key</td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>maxJsonDepth</td>
<td>100</td>
<td>Maximum length of JSON</td>
</tr>
</tbody>
</table>

**Editing the sequence through registry artifacts**

To edit the existing sequence follow the steps below.

1. Log in to the Management Console.
2. Navigate to `/system/governance/apimgt/customsequences/in/json_validator.xml`
3. Edit the `json_validator.xml` file.
4. Go to the API Publisher and re-publish your API for the changes to take effect.

**Applying the JSON validator policy**

You can apply the pre-defined JSON Policy through the UI. Follow the instructions below to apply the `json_validator` in sequence.

- Create an API or edit an existing API.
- Go to Message Mediation Policies under the Implement tab.
- Select Enable Message Mediation. Select `json_validator` from the drop-down menu for In Flow.

**Testing the JSON threat protector**

You can edit the sequence to set the property values according to your requirements. A sample request and response for each property value set to 5 is given below.

**Request Response**

Note that this exceeds the JSON property count

```bash
The request message:
```
Disabling the XML payload validation

The XML threat protector in WSO2 API Manager validates the XML payload vulnerabilities based on the pre-configured limits. It uses following methodologies to thwart the gateway from XML based attacks.

- Detecting the malformed, vulnerable XML messages through limitations
- XML schema validation

**XML Threat Protection for API Gateway**

Detecting the malformed, vulnerable XML messages through limitations

The xml_validator sequence specifies the properties to be limited in the payload. A sample xml_validator sequence is given below.

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="xml_validator">
  <log level="custom">
    <property name="IN_MESSAGE" value="xml_validator"/>
  </log>
  <property name="xmlValidation" value="true"/>
  <property name="dtdEnabled" value="false"/>
  <property name="externalEntitiesEnabled" value="true"/>
  <property name="maxXMLDepth" value="100"/>
  <property name="maxElementCount" value="100"/>
  <property name="maxAttributeCount" value="100"/>
  <property name="maxAttributeLength" value="100"/>
  <property name="entityExpansionLimit" value="100"/>
  <property name="maxChildrenPerElement" value="100"/>
  <property name="schemaValidation" value="true"/>
  <switch source="get-property('To')">
    <case regex=".*\addResource.*">
      <property name="xsdURL" value="<Insert the XSD URL>"/>
    </case>
    </case regex=".*\update.*">
    <!--<property name="xsdURL" value="<insert XSD_URL>"/>-->
    </case regex=".*\delete.*">
    <!--<property name="xsdURL" value="<insert XSD_URL>d"/>-->
  </switch>
  <property name="RequestMessageBufferSize" value="1024"/>
  <class name="org.wso2.carbon.apimgt.gateway.mediators/XMLSchemaValidator"/>
</sequence>
```

Users can enable or disable XML payload limits and schema validation. Some examples are shown below.

Disabling the XML payload validation Disabling the XML schema validation

```xml
<property name="xmlValidation" value="false"/>
```

**Performance impact**

The JSON schema mediator builds the message at the mediation level. This impacts the performance of 10KB messages for 300 concurrent users by 5.2 times than the normal flow.
XML payload validation properties

- Disable the DTD payload in the XML properties to avoid attacks
- You can turn on/off external entities of the payload. An example is given below with the elements of the XML request body, that can be configured.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtdEnabled</td>
<td>false</td>
<td>The DTD can be enabled/disabled according to your requirement.</td>
</tr>
<tr>
<td>externalEntitiesEnabled</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>maxXMLDepth</td>
<td>100</td>
<td>Maximum depth of the XML request message.</td>
</tr>
<tr>
<td>maxElementCount</td>
<td>100</td>
<td>Maximum number of allowed elements in the XML request message.</td>
</tr>
<tr>
<td>maxAttributeCount</td>
<td>100</td>
<td>Maximum count of allowed attributes in the XML request message.</td>
</tr>
<tr>
<td>maxAttributeLength</td>
<td>100</td>
<td>Maximum allowed length of each attribute value in characters.</td>
</tr>
<tr>
<td>entityExpansionLimit</td>
<td>100</td>
<td>Maximum allowed entity expansion limit of the XML request message.</td>
</tr>
<tr>
<td>maxChildrenPerElement</td>
<td>100</td>
<td>Maximum number of child elements allowed in the XML request message.</td>
</tr>
</tbody>
</table>

XML schema validation

You can define XML schemas per resource to validate each request. For example, to add an XML schema to the resource /userapi/1.0.0/addResource/value follow the steps below.

1. Define the resource in the case regex
2. Define the relevant schema URL and add it as shown below.
3. You can define the buffer size of the request message depending on your requirement. An example is given below.

Each request is sanitized through the XML threat protector. API developer can modify each properties according to your requirement.

- Editing the sequence through registry artifacts
- Applying the XML validator policy
- Testing the XML threat protector
- Testing the schema validation

Editing the sequence through registry artifacts

To edit the existing sequence follow the steps below.

1. Log in to the Management Console.
2. Navigate to /_system/governance/apimgt/customsequences/in/xml_validator.xml
3. Edit the xml_validator.xml file.
4. Go to the API Publisher and re-publish your API for the changes to take effect.

Applying the XML validator policy

You can apply the pre-defined XML Policy through the UI. Follow the instructions below to apply the xml_validator in sequence.

- Create an API or edit an existing API.
- Go to Message Mediation Policies under the Implement tab.
- Select Enable Message Mediation. Select xml_validator from the drop-down menu for In Flow.
Testing the XML threat protector

You can edit the sequence to set the property values according to your requirements. A sample request and response for the value of the properties set to 30 is given below. Note that the .xsd URL for the relevant resource has been hosted.

```curl
<inline_model>
<Name>string</Name>
<Age>string</Age>
<Address>string</Address>
<phone>string</phone>
<home>string</home>
<path>string</path>
<class>string</class>
<team>string</team>
<location>string</location>
<brand>string</brand>
<summary>string</summary>
<data>string</data>
</inline_model>
```

```
<am:fault xmlns:am="http://wso2.org/apimanager">
  <am:code>400</am:code>
  <am:message>Bad Request</am:message>
  <am:description>XML Validation Failed: due to Maximum Element Count limit (30) Exceeded</am:description>
</am:fault>
```

Testing the schema validation

A sample request and response to test the schema validation is given below.
curl -X POST "https://192.168.8.101:8243/xmlPolicy/1.0.0/addResource" -H "accept: application/json" -H "Content-Type: application/xml" -H "Authorization: Bearer 2901c002-f626-372c-9be3-f054b2c8d65f" -d "<?xml version="1.0"?><inline_model> <location>string</location> <brand>string</brand> <summary>string</summary> <data>string</data> <Name>string</Name> <Age>string</Age> <Address>string</Address> <phone>string</phone> <home>string</home> <path>string</path> <class>string</class> <team>string</team> <location>string</location> <brand>string</brand> <summary>string</summary> <data>string</data></inline_model>"

<am:fault xmlns:am="http://wso2.org/apimanager">
  <am:code>400</am:code>
  <am:message>Bad Request</am:message>
  <am:description>Error occurred while parsing XML payload : org.xml.sax.SAXParseException: cvc-elt.1: Cannot find the declaration of element 'inline_model'.</am:description>
</am:fault>
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="stringtype">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="inttype">
    <xs:restriction base="xs:positiveInteger"/>
  </xs:simpleType>
  <xs:simpleType name="dectype">
    <xs:restriction base="xs:decimal"/>
  </xs:simpleType>
  <xs:simpleType name="orderidtype">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{6}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="shiptotype">
    <xs:sequence>
      <xs:element name="name" type="stringtype"/>
      <xs:element name="address" type="stringtype"/>
      <xs:element name="city" type="stringtype"/>
      <xs:element name="country" type="stringtype"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="itemtype">
    <xs:sequence>
      <xs:element name="title" type="stringtype"/>
      <xs:element name="note" type="stringtype" minOccurs="0"/>
      <xs:element name="quantity" type="inttype"/>
      <xs:element name="price" type="dectype"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="shipordertype">
    <xs:sequence>
      <xs:element name="orderperson" type="stringtype"/>
      <xs:element name="shipto" type="shiptotype"/>
      <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
    </xs:sequence>
    <xs:attribute name="orderid" type="orderidtype" use="required"/>
  </xs:complexType>
  <xs:element name="shiporder" type="shipordertype"/>
</xs:schema>

Performance impact

The XML mediator builds the message at the mediator level. This impacts the performance of 10KB messages for 300 concurrent users by 5.6 times than the normal flow. The performance may slow down along with the message size.

General Data Protection Regulation (GDPR) for WSO2 API Manager

The Forget-Me tool pre-packed with API-M can be used to remove identities of an external user who is deleted according to the system administrator’s request. This tool removes user identities stored in the database and also in log files in order to meet GDPR requirements. The following sections guide you through configuring and running this tool in WSO2 API Manager.

- Changing the default configurations of the tool.
- Changing the default configurations location
Running the tool in API Manager

This tool is packaged with WSO2 API Manager by default. Follow the steps below to run this tool.

Before you begin...

- Note that this tool is designed to run in offline mode (i.e., the server should be shut down or run on another machine) in order to prevent unnecessary load to the server. If this tool runs in online mode (i.e., when the server is running), DB lock situations on the H2 databases may occur. This DB lock may happen if at least one of your databases point to H2. Let's say you have User, REG and AM databases pointed to Mysql but your Carbon DB is in H2, then also you can get this DB lock error when running in online mode.
- If you have configured a database other than the default H2 database, copy the relevant driver to the `<API-M_HOME>/repository/components/tools/forget-me/lib` directory.

1. Open a new terminal window and navigate to the `<API-M_HOME>/bin` directory.
2. Execute one of the following commands depending on your operating system:
   - On Linux/Mac OS: `./forgetme.sh -U <username>`
   - On Windows: `forgetme.bat -U <username>`

   The command specified above uses only the `-U <username>` option, which is the only mandatory option to run the tool. There are several other optional command line options that you can specify based on your requirement. The supported options are described in detail below.

3. The following is the list of all the command line options that can be used with this command.

<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
<th>Required</th>
<th>Default Value</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>The name of the user whose identity references you want to remove.</td>
<td>Yes</td>
<td></td>
<td><code>-U alex.doe</code></td>
</tr>
<tr>
<td>d</td>
<td>The configuration directory to use when the tool is run.</td>
<td>No</td>
<td></td>
<td><code>-d /users/alex/forgetme/config</code></td>
</tr>
<tr>
<td>d</td>
<td>If you do not specify a value for this option, the default conf directory will be used.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>The tenant domain</td>
<td>No</td>
<td>carbon.super</td>
<td><code>-T example-company</code></td>
</tr>
</tbody>
</table>
### Running the toolkit in standalone mode

This tool can run standalone and therefore cater to multiple products. This means that if you are using multiple WSO2 products and need to delete the user's identity from all products at once, you can do so by running the tool in standalone mode.

For information on how to build and run the Forget-Me tool, see [Removing References to Deleted User Identities in WSO2 Products](#) in the WSO2 Administration Guide.

### Running the tool in API Manager Analytics

Shown below is an example data stream used by API Manager Analytics. Note that the userId/username, emails and the ip are personally identifiable information (PII) of the user.

<table>
<thead>
<tr>
<th>Stream Name</th>
<th>Attribute List</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>org.wso2.analytics.apim.ipAccessSummary</code></td>
<td><code>userId</code>, <code>ip</code></td>
</tr>
<tr>
<td><code>org.wso2.analytics.apim.alertStakeholderInfo</code></td>
<td><code>userId</code>, <code>emails</code></td>
</tr>
</tbody>
</table>

These PII references can be removed from the Analytics database by using the Forget-Me tool. Follow the steps given below.

1. Add the relevant drivers for your Analytics-specific databases to the `<API-M_ANALYTICS_HOME>/repository/components/tools/forget-me/lib` directory. For example, if you have changed your Analytics databases from the default H2 instances to MySQL, copy the MySQL driver to this given directory.
2. Create a folder named `streams` in the `<API-M_ANALYTICS_HOME>/repository/components/tools/forget-me/conf` directory.
3. Create a new file named `streams.json` with content similar to what is shown below based on the streams used, and store it in the `streams` directory that you created in the previous step. This file holds the details of the streams and the attributes with PII that we need to remove from the database.

---

**TID**  | The tenant ID  | No  | -TID 1234

**D**  | The userstore domain  | No  | PRIMARY  | -D Finance-domain

**pu**  | The pseudonym with which the username should be replaced.  | No  | A random UUID value is generated as the pseudonym.  | -pu "123-343-435-545-dfd-4"

**carbon**  | The CARBON HOME.  | No  |  | -carbon  "usr/bin/wso2am/wso2am-2.6.0"

---

4. All references to the user are removed from WSO2 API Manager. You can view the generated reports inside the `<API-M_HOME>/repository/components/tools/forget-me/conf` directory.
The above configuration includes the following:

- **Stream Name**: The name of the stream.
- **Attributes**: The list of attributes that contain PII.
- **id**: The ID attribute, which holds the value that needs to be anonymized (replaced with a pseudonym).

4. Create a new file named `config.json` in `<API-M_ANALYTICS_HOME>/repository/components/tools/forget-me/conf/` directory with the content shown below.

```json
{
  "processors": [
    "analytics-streams"
  ],
  "directories": [
    {
      "dir": "streams",
      "type": "analytics-streams",
      "processor": "analytics-streams"
    }
  ]
}
```

5. Open a command prompt and navigate to the `<API-M_ANALYTICS_HOME>/bin` directory.

6. Execute one of the following commands depending on your operating system:

- On Linux/Mac OS: `. /forgetme.sh -U <username>`
- On Windows: `forgetme.bat -U <username>`

### Identity Management for the API Store

- Password Recovery
- Account locking
- Password policies

#### Password Recovery

Password recovery for a store user can be done by enabling the password recovery feature. This feature will not work by default as an email server is not configured. Follow the steps below to configure it in the API Store.

1. Open the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file and uncomment the following tag to configure a mail server. Change the default values with details of your mail server. See `MailTo Transport` for more information.

```xml
<transportSender name="mailto" class="org.apache.axis2.transport.mail.MailTransportSender">
  <parameter name="mail.smtp.host">smtp.gmail.com</parameter>
  <parameter name="mail.smtp.port">587</parameter>
  <parameter name="mail.smtp.starttls.enable">true</parameter>
  <parameter name="mail.smtp.auth">true</parameter>
  <parameter name="mail.smtp.user">synapse.demo.0</parameter>
  <parameter name="mail.smtp.password">mailpassword</parameter>
  <parameter name="mail.smtp.from">synapse.demo.0@gmail.com</parameter>
</transportSender>
```
2. After you configure the mail server, the user can click on the **Forgot Password** link on the Sign In page of the API Store and request a password change.

![Sign In to Your Account](image)

3. You need to have a user account with an email configured for this feature to work. Click the **Forgot Password** link. You will be directed to the Password Reset page. Enter the username you are trying to recover the password of.

![Password Reset](image)

4. After you add the username, click **Request Password Change**.

![Password Reset](image)

You will receive a notification that the password recovery instructions have been sent to your email address. Note that this is the email address stored during user sign up.

![Password Recovery Instructions](image)

5. You will receive an email with the following information. A sample is shown below. Click on the link given in the email.
Hi Ashley

We received a request to change the password on the Ashley account associated with this e-mail address.

If you made this request, please click the link below to securely change your password:


If clicking the link doesn't seem to work, you can copy and paste the link into your browser's address window.

If you did not request to have your Ashley password reset, simply disregard this email and no changes to your account will be made.

Best Regards,
WSO2 Carbon Team
http://www.wso2.com

6. You will be redirected to the page shown below to change your password.

Account locking

You can lock user accounts with the Account recovery and credential management feature, pre-installed in WSO2 API Manager.

- Account locking by failed login attempts
- Account locking by an administrative user

Account locking by failed login attempts

The following steps show how to enable account locking.

1. Open the <API-M_HOME>/repository/conf/identity/identity.xml file. Ensure that the IdentityMgtEventListener with the orderId=50 is set to true.

```xml
<EventListener enable="true"
    name="org.wso2.carbon.identity.mgt.IdentityMgtEventListener"
    orderId="50"
    type="org.wso2.carbon.user.core.listener.UserOperationEventListener"/>
```

   ```
   Authentication.Policy.Enable=true
   ```

   Change the following properties according to your preference.

   ```
   ```

   The description for the properties are given in the table below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

3. Restart the server for the changes to be applied.

   An error message similar to the following will be logged in `wso2carbon.log` when the account is locked:

   ```
   ERROR - AuthenticationAdmin System error while Authenticating/Authorizing User : 17003 User account is locked for user : testUser. cannot login until the account is unlocked
   ```

Account locking by an administrative user

An administrative user can lock a user account by editing the user’s profile in the management console.

2. Go to Claims > List on the Configure menu and select the http://wso2.org/claims claim dialect.
3. Select the Account Locked claim and click Edit.

   ![Account Locked Claim](image)

   - **Claim URI:** http://wso2.org/claims/identity/accountLocked
   - **Description:** Account Locked
   - **Mapped Attribute:** User Store Domain Name
   - **Regular Expression:** PRIMARY
   - **Display Order:** 0
   - **Supported by Default:** false
   - **Required:** false
   - **Read only:** false

4. Select the Supported by Default checkbox and click Update. This is done to make the Account Locked status appear in the user’s profile.
5. Go to **Users and Roles > List > Users** on the **Main** menu and click on **User Profile** of the user you want to lock.

6. Tick the checkbox in front of the **Account Locked** field to lock the account for the user and click **Update**.

### Paasword policies

You can define custom password policies for store user signup as follows.

1. Open the identity.xml file in the `<API-M_HOME>/repository/conf/identity/` folder and set the `org.wso2.carbon.identity.mgt.IdentityMgtEventListener` under the `<EventListeners>` tag to enable="true".

```xml
<EventListener enable="true" name="org.wso2.carbon.identity.mgt.IdentityMgtEventListener" orderId="50" type="org.wso2.carbon.user.core.listener.UserOperationEventListener"/>
```
2. Uncomment and edit the following entries in identity-mgt.properties file in <API-M_HOME>/repository/conf/identity folder based on your preference.

```properties
# Define password policy enforce extensions
Password.policy.extensions.1=org.wso2.carbon.identity.mgt.policy.password.DefaultPasswordLengthPolicy
Password.policy.extensions.1.min.length=6
Password.policy.extensions.1.max.length=12
Password.policy.extensions.2=org.wso2.carbon.identity.mgt.policy.password.DefaultPasswordNamePolicy
Password.policy.extensions.3=org.wso2.carbon.identity.mgt.policy.password.DefaultPasswordPatternPolicy
Password.policy.extensions.3.pattern="{?=.*[^\d]{6}.?=.*[^a-z]{6}.?=.*[^A-Z]{6}.?=.*[!@#$%&*]{6}).{0,100}$"
Password.policy.extensions.3.errorMsg='Password pattern policy violated. Password should contain a digit[0-9], a lower case letter[a-z], an upper case letter[A-Z], one of !@#$%&* characters'
```

For more information, see Writing a Custom Password Validator.

Customizing signup validation in the API Store

API Store self signup UI page input password field validation is based on the default password policy. You can change it to match with your custom policy by extending the self signup page with your custom html changes via a sub theme.

Securing APIs with Mutual SSL

This is available only as a WUM update and is effective from 22nd October 2018 (2018-10-22). For more information on updating WSO2 API Manager, see Updating WSO2 API Manager.

In contrast to the usual one-way SSL authentication where a client verifies the identity of the server, in mutual SSL the server validates the identity of the client so that both parties trust each other. This builds a system that has a very tight security and avoids any requests made to the client to provide the username/password, as long as the server is aware of the certificates that belong to the client.

This section explains how to APIs in WSO2 API Manager can be secured using mutual SSL in addition to OAuth2.

- Enable securing APIs with mutual SSL
- Create an API secured with mutual SSL
- Invoke an API secured with Mutual SSL from the API Store

Enable securing APIs with mutual SSL

Follow the steps below to enable this feature in WSO2 API Manager.

1. Create the AM_API_CLIENT_CERTIFICATE table in the APIM DB using the appropriate script given below. Note that the database name will depend on the databases present in your environment.

   ```sql
   CREATE TABLE IF NOT EXISTS `AM_API_CLIENT_CERTIFICATE` (
    `TENANT_ID` INT(11) NOT NULL,
    `ALIAS` VARCHAR(45) NOT NULL,
    `API_ID` INTEGER NOT NULL,
    `CERTIFICATE` BLOB NOT NULL,
    `REMOVED` BOOLEAN NOT NULL DEFAULT 0,
    `TIER_NAME` VARCHAR (512),
    FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (`ALIAS`, `TENANT_ID`, `REMOVED`)
   )
   ```
IF NOT EXISTS (SELECT * FROM SYS.OBJECTS WHERE OBJECT_ID = OBJECT_ID(N'[dbo].[AM_API_CLIENT_CERTIFICATE]') AND TYPE IN (N'U'))
CREATE TABLE AM_API_CLIENT_CERTIFICATE (
    TENANT_ID INTEGER NOT NULL,
    ALIAS VARCHAR(45) NOT NULL,
    API_ID INTEGER NOT NULL,
    CERTIFICATE VARBINARY(MAX) NOT NULL,
    REMOVED BIT NOT NULL DEFAULT 0,
    TIER_NAME VARCHAR(512),
    PRIMARY KEY (ALIAS, TENANT_ID, REMOVED),
    FOREIGN KEY (API_ID) REFERENCES AM_API(API_ID) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS `AM_API_CLIENT_CERTIFICATE` (
    `TENANT_ID` INT(11) NOT NULL,
    `ALIAS` VARCHAR(45) NOT NULL,
    `API_ID` INTEGER NOT NULL,
    `CERTIFICATE` BLOB NOT NULL,
    `REMOVED` BOOLEAN NOT NULL DEFAULT 0,
    `TIER_NAME` VARCHAR (512),
    FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (`ALIAS`, `TENANT_ID`, `REMOVED`)
) ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS `AM_API_CLIENT_CERTIFICATE` (
    `TENANT_ID` INT(11) NOT NULL,
    `ALIAS` VARCHAR(45) NOT NULL,
    `API_ID` INTEGER NOT NULL,
    `CERTIFICATE` BLOB NOT NULL,
    `REMOVED` BOOLEAN NOT NULL DEFAULT 0,
    `TIER_NAME` VARCHAR (512),
    FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (`ALIAS`, `TENANT_ID`, `REMOVED`)
) ENGINE=MDB;

CREATE TABLE AM_API_CLIENT_CERTIFICATE ( 
    TENANT_ID INTEGER NOT NULL,
    ALIAS VARCHAR2(45) NOT NULL,
    API_ID INTEGER NOT NULL,
    CERTIFICATE BLOB NOT NULL,
    REMOVED INTEGER DEFAULT 0 NOT NULL,
    TIER_NAME VARCHAR2 (512),
    FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE,
    PRIMARY KEY (ALIAS, TENANT_ID, REMOVED)
);
CREATE TABLE AM_API_CLIENT_CERTIFICATE ( 
  TENANT_ID INTEGER NOT NULL,
  ALIAS VARCHAR2(45) NOT NULL,
  API_ID INTEGER NOT NULL,
  CERTIFICATE BLOB NOT NULL,
  REMOVED INTEGER DEFAULT 0 NOT NULL,
  TIER_NAME VARCHAR2(512),
  FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE,
  PRIMARY KEY (ALIAS, TENANT_ID, REMOVED)
)
/

DROP TABLE IF EXISTS AM_API_CLIENT_CERTIFICATE;
CREATE TABLE AM_API_CLIENT_CERTIFICATE ( 
  TENANT_ID INTEGER NOT NULL,
  ALIAS VARCHAR(45) NOT NULL,
  API_ID INTEGER NOT NULL,
  CERTIFICATE BYTEA NOT NULL,
  REMOVED BOOLEAN NOT NULL DEFAULT '0',
  TIER_NAME VARCHAR(512),
  FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE,
  PRIMARY KEY (ALIAS, TENANT_ID, REMOVED)
);

CREATE TABLE AM_API_CLIENT_CERTIFICATE ( 
  TENANT_ID INT NOT NULL,
  ALIAS VARCHAR(45) NOT NULL,
  API_ID INTEGER NOT NULL,
  CERTIFICATE BLOB NOT NULL,
  REMOVED SMALLINT NOT NULL DEFAULT 0,
  TIER_NAME VARCHAR (512),
  FOREIGN KEY (API_ID) REFERENCES AM_API (API_ID) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY (ALIAS, TENANT_ID, REMOVED)
)/

2. Open the <API-M_HOME>/repository/conf/axis2/axis2.xml file. Locate the transportReceiver for https as shown below.

```xml
<transportReceiver name="https"
  class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener">
```

Change the class name to `org.apache.synapse.transport.passthru.PassThroughHttpMultiSSLListener`. The sample is given below.

```xml
<transportReceiver name="https"
  class="org.apache.synapse.transport.passthru.PassThroughHttpMultiSSLListener">
```

3. Add the following parameters under `transportReceiver` similar to the sample given below.

```xml
<parameter name="dynamicSSLProfilesConfig">
  <filePath>repository/resources/security/listenerprofiles.xml</filePath>
  <fileReadInterval>600000</fileReadInterval>
</parameter>

<parameter name="SSLVerifyClient">optional</parameter>
```

The `dynamicSSLProfilesConfig` specifies the file read to load the dynamic SSL profile and the time interval in which it will be read.
Note that if the `fileReadInterval` parameter is set to 600000, it will take at least 10 minutes for the gateway to accept a newly added client certificate.

Expand to see the transportReceiver segment after the changes mentioned above have been made...

```xml
<transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpMultiSSLListener">
    <parameter name="port" locked="false">8243</parameter>
    <parameter name="non-blocking" locked="false">true</parameter>
    <!--parameter name="bind-address" locked="false">hostname or IP address</parameter-->
    <!--parameter name="WSDLEPRPrefix" locked="false">https://apachehost:port/somepath</parameter-->
    <parameter name="httpGetProcessor">
        <parameter name="keystore" locked="false">
            <KeyStore>
                <Location>repository/resources/security/wso2carbon.jks</Location>
                <Type>JKS</Type>
                <Password>wso2carbon</Password>
                <KeyPassword>wso2carbon</KeyPassword>
            </KeyStore>
        </parameter>
        <parameter name="truststore" locked="false">
            <TrustStore>
                <Location>repository/resources/security/client-truststore.jks</Location>
                <Type>JKS</Type>
                <Password>wso2carbon</Password>
            </TrustStore>
        </parameter>
    </parameter>
    <parameter name="dynamicSSLProfilesConfig">
        <filePath>repository/resources/security/listenerprofiles.xml</filePath>
        <fileReadInterval>600000</fileReadInterval>
    </parameter>
    <parameter name="SSLVerifyClient">optional</parameter>
</transportReceiver>
```

4. Copy the code given below to create the `listenerprofiles.xml` file in the `<API-M_HOME>/repository/resources/security` directory.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<parameter name="SSLProfiles">
    <profile>
        <bindAddress>0.0.0.0</bindAddress>
        <KeyStore>
            <Location>repository/resources/security/wso2carbon.jks</Location>
            <Type>JKS</Type>
            <Password>wso2carbon</Password>
            <KeyPassword>wso2carbon</KeyPassword>
        </KeyStore>
        <TrustStore>
            <Location>repository/resources/security/client-truststore.jks</Location>
            <Type>JKS</Type>
            <Password>wso2carbon</Password>
        </TrustStore>
    </profile>
</parameter>
```
5. Open the <API-M_HOME>/repository/conf/api-manager.xml file. Set the EnableMTLSForAPIs parameter to true.

```xml
<APIManager>
    ...
    <EnableMTLSForAPIs>true</EnableMTLSForAPIs>
</APIManager>
```

6. Restart the server.

Create an API secured with mutual SSL

1. Create an API.
2. Edit the API and navigate to the Manage tab.

   Make this the default version: 
   [ ] No default version defined for the current API

   Transports: *
   [ ] HTTPS  [ ] HTTP

   Response Caching: [ ]

   API Security: *
   [ ] OAuth2  [ ] Mutual SSL

   Manage Certificates

You can select both OAuth2 and Mutual SSL options. This means that the user can access the API using a valid OAuth2 token or using a valid client certificate.

4. Click Manage Certificates to upload a new client certificate. Select Add New Certificate.

   API Security: *
   [ ] OAuth2  [ ] Mutual SSL

   Manage Certificates

This feature currently supports only the following formats for keystores and certificates.

- Keystore: .jks
- Certificate: .crt

If you need to use a certificate in any other format, you can convert it using a standard tool before uploading.

After configuring, the certificate will be added to the Gateway nodes which are defined under the Environments in api-manager.xml. In a clustered setup, as gateway configurations are identical, sync the <API-M_HOME>/repository/resources/security/listenerprofiles.xml and <API-M_HOME>/repository/resources/security/client-truststore.jks among the gateway nodes. After the configured interval, the synapse transport will be reloaded in all the gateway nodes.

5. Provide an alias and public certificate. Select the tier that should be used to throttle out the calls using this particular client certificate and click Upload.
6. Save and Publish the API

**Invoke an API secured with Mutual SSL from the API Store**

**Before you begin...**

Add the relevant certificate to your browser according to your private certificate.

1. Invoke an API from the API Store.
2. When you click Execute the browser will send a prompt similar to the one shown below. Select the corresponding certificate for the API.

Limitations

Listed below are the known limitations for this feature.

- Application subscription is not permitted for APIs that are only protected with mutual SSL. Hence, subscription/application level throttling is not applicable for these type of APIs.
- Resource level throttling is not applicable for the APIs that are only protected with mutual SSL.
- Resource level security will not be applicable for the APIs that are only protected with mutual SSL.
- Scope level security will not be applicable for the APIs that are only protected with mutual SSL.

**Working with Throttling**

The following sections explain how to work with throttling in the API Manager:

- Introducing Throttling Use-Cases
- Setting Maximum Backend Throughput Limits
- Setting Throttling Limits
- Adding New Throttling Policies
- Managing Throttling
- Enforcing Throttling to an API
- Engaging a new Throttling Policy at Runtime
Engaging Multiple Throttling Policies to a Single API

Introducing Throttling Use-Cases

Throttling allows you to limit the number of successful hits to an API during a given period, typically in cases such as the following:

- To protect your APIs from common types of security attacks such as certain types of denial of service (DOS) attacks
- To regulate traffic according to infrastructure availability
- To make an API, application, or a resource available to a consumer at different levels of service, usually for monetization purpose

The API Gateway architecture model, which solves the API management problem, comprises the following:

- The back-end services/systems hosting the actual business logic
- The APIs in the API Gateway that proxy the back-end services
- The applications that consume the APIs in the API Gateway
- The users of the applications

The following sections describe the type of throttling policy applicable to each of the above areas and why the relevant stakeholders must consider each of them carefully.

- Implications on back-end services/systems
- Implications on the APIs in the Gateway
- Advanced throttling policies: API Publisher
- Implications on applications that consume APIs

Implications on back-end services/systems

Maximum backend throughput (Applies per API): API Publisher

According to the API Gateway architecture, an API in the Gateway is actually a proxy to an actual service hosted within your organization, cloud, etc. This usually means that there is a physical capacity that your backend services can handle. Although you expose your API on defined limits (subscription tiers), as the number of applications that consume your API grows, the number of requests being served by your API rise, which in turn means that the number of
requests served by your backend system rise as well. Therefore, although none of the applications may exceed their own allocated quotas, their combined load might hit the maximum capacity that can be handled by your backend system. To prevent your backend system from getting overloaded, the limits enforced by the Maximum Backend Throughput in the API act as a hard stop on the number of requests that your backend system can serve within a given time period. The counters maintained when evaluating the maximum backend throughput are shared across all nodes of the Gateway cluster and apply across all users using any application that accesses that particular API. For information on how to specify maximum backend throughput limits, see Setting Maximum Backend Throughput Limits.

Implications on the APIs in the Gateway

Subscription tiers: API Publisher

When an API Publisher publishes an API to be consumed by applications, s/he can choose to make the API available over different limits. For example, the Gold tier allows an application to access the API at 5000 requests per minute while a Silver tier allows an application to access the API at 2000 requests per minute. For information on how to define a throttling tier to an API, see API-level throttling (API publisher).

The subscription tiers are used to gain monetary value for the API; you can charge more from app developers who require larger quotas of your API’s functionality and lesser from developers who require less. The limits can be enforced either by the number of requests over time (5000 req/min) or the amount of data bandwidth over time (500 mb/hour). The limits enforced by subscription tiers are applied across all users of the application that use that particular API and can be considered as a shared quota among all users of an application that access that API. When using a cluster of Gateway nodes, the counters maintained while evaluating the subscription tiers are shared across all nodes. For information on how to define a subscription tier to an API, see Subscription-level throttling (API publisher).

Burst Control

Burst control limits are enforced for subscription tiers in order to distribute the load across the specified time period. For example, if you have a subscription tier that allows you to send 1000 requests per hour, you can ensure that a particular application does not consume the full quota of 1000 requests within the first 2 minutes by setting a burst control limit within the subscription tier allowing only a maximum of 25 requests per minute. Therefore, the time periods set for burst control limits must always be smaller than the time period specified for its corresponding subscription tier. Burst control limits can be set only to control the number of requests for a given period of time and does not allow you to control the data bandwidth for a given time period. The burst control limits are enforced for each individual Gateway node. Although the request counters are replicated across the cluster, since burst control time periods are usually quite small, the replication frequency can be quite high compared to the burst rate of incoming requests. Therefore, it is safe to assume that the burst control values are applied on a per-node basis. For information on how to define burst control limits, see Rate limiting (burst control).

Advanced throttling policies: API Publisher

Advanced throttling policies allow an API Publisher to control access to his API resources using advanced rules. Advanced policies include the ability to apply limits by filtering requests based on the following properties and their combinations. The counters maintained when evaluating advanced throttling policies are shared across all nodes in the Gateway cluster.

- IP address and address range
- HTTP request headers
- JWT claims
- Query parameters

Let's look at how each of these can be important for serving requests through your APIs.

IP address and address range

You can control/restrict access to your API or its selected resources for a given IP address or address range. For example, if you need to grant permission for internal applications to consume a larger quota of your API resource than your external consumers, you can define an advanced policy with higher limits for your internal IP address range and lower limits for the rest. For information on how to define IP throttling, see Advanced throttling (API publisher).

HTTP request headers

Advanced policies allow you to apply limits to APIs by filtering requests based on HTTP headers. For example, assume you need to apply a special limit for JSON requests. To do that, you can filter JSON messages by using a policy that inspects the HTTP request headers and checks if the Content-Type header is application/json and apply a special limit for those requests while allowing a default value for the rest.

Here is a sample for configuring JWT claim condition by considering the "Content-Type" header.
JWT claims

A JWT claim contains meta information of an API request. It can include application details, API details, user claims, etc. Advanced throttling policies based on JWT claims allow you to filter requests by JWT claim values and apply limits for requests. For example, if you need to allow special limits for users in a specific user role, you can create an advanced policy that checks for a particular regular expression on the role claim of the user and apply special limits for the ones that match.

Here is a sample for configuring JWT claim condition by considering the version of the API ("http://wso2.org/claims/version").

Query parameters

Filtering based on query parameters almost always apply to HTTP GET requests when doing search type of operations. For example, if you have a search API with category as a query parameter, you can have different limits for searching different categories.
Eg: 'sales' category can be allocated with more requests than 'hr' category

**Implications on applications that consume APIs**

Per token quota: Application Developer

When an application developer subscribes their application to an API, they select a tier (limit) for their application to invoke the API. This limit applies across all users of the application when accessing the particular API. To ensure that a fair distribution of the quota is available among all the users, it is important to consider setting a per user quota for the application, since a user is identified by a token (in OAuth2.0, this limit is known as the per token quota). It is important to note that the limit enforced by this setting applies to a single user (token) accessing all APIs of the application. The counters maintained when evaluating a per token quota are shared across all nodes in the Gateway cluster. For information on how to define a throttling tier to an application, see Application-level throttling (application developer).

**The below diagram shows how throttle policies are applied at different levels.**

If advanced throttling is enabled, it will also be applied here. Requests will be allowed/rejected based on the conditions specified in advanced throttling policies as well.
Setting Maximum Backend Throughput Limits

The maximum backend throughput setting limits the total number of calls a particular API in the API Manager is allowed to make to the backend. While the other throttling levels define the quota the API invoker gets, they do not ensure that the backend is protected from overuse. The maximum backend throughput setting limits the quota the backend can handle. The counters maintained when evaluating the maximum backend throughput are shared across all nodes of the Gateway cluster and apply across all users using any application that accesses that particular API.

You set a maximum backend throughput using the Manage tab of the API Publisher when creating or editing an API. Under the Throttling Settings section, select the Specify option for the maximum backend throughput and specify the limits of the Production and Sandbox endpoints separately, as the two endpoints can come from two servers with different capacities.

Alternatively, you can go to the synapse configuration of the API, which is created at the point of creating the API, in the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/api` directory (for a tenant user, the location would be the `<API-M_HOME>/repository/tenants/<TenantID>/synapse-configs/default/api` directory), and specify the maximum backend throughput by modifying the synapse configuration. Maximum backend throughput limits are usually counted over a duration of 1 second, but you can increase the duration using the `productionUnitTime` and `sandboxUnitTime` properties in the API's synapse configuration. For example,

If you want to accept only 600 requests by the production endpoint within a minute of duration and 700 total requests within 5 minutes by sandbox endpoint, you can modify the synapse configuration as below.

```xml
<handlers>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler">
    <property name="apiImplementationType" value="ENDPOINT"/>
  </handler>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
  <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.ThrottleHandler">
    <property name="id" value="A"/>
    <property name="productionMaxCount" value="600"/>
    <property name="productionUnitTime" value="60000"/>
    <property name="sandboxMaxCount" value="700"/>
    <property name="sandboxUnitTime" value="300000"/>
    <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/>
  </handler>
  <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
  ...
</handlers>
```

Note that the duration is specified in milliseconds.

Setting Throttling Limits

Throttling allows you to limit the number of successful hits to an API during a given period, typically in cases such as the following:
To protect your APIs from common types of security attacks such as certain types of denial of service (DOS) attacks
To regulate traffic according to infrastructure availability
To make an API, application, or a resource available to a consumer at different levels of service, usually for monetization purpose

You can define throttling at the API, application, resource and subscription levels. The final throttle limit granted to a given user on a given API is ultimately defined by the consolidated output of all throttling tiers together.

**Example**: Let’s say two users are subscribed to an API using the Gold subscription, which allows 20 requests per minute. They both use the application App1 for this subscription, which again has a throttling tier set to 20 requests per minute. All resource level throttling tiers are unlimited. In this scenario, although both users are eligible for 20 requests per minute access to the API, each ideally has a limit of only 10 requests per minute. This is due to the application-level limitation of 20 requests per minute.

### Different levels of throttling

It is possible to throttle requests for each tier based on the request count per unit time or the amount of data (bandwidth) transferred through the Gateway per unit time. You can configure this via the Admin Portal as explained under Adding New Throttling Policies.

Let’s take a look at the different levels of throttling:

- **Subscription-level throttling (API publisher)**
  - Rate limiting (burst control)
- **Subscription-level throttling (API subscriber)**
- **Advanced throttling (API publisher)**
  - API level throttling
  - Resource-Level Throttling
  - Advanced Throttling tiers
- **Application-level throttling (application developer)**
  - Application-level Throttling tiers

**Subscription-level throttling (API publisher)**

Subscription-level throttling tiers are also defined when managing APIs using the API Publisher portal.
Subscription-level Throttling tiers

The default throttling tiers are as follows:

- **Bronze**: 1000 requests per minute
- **Silver**: 2000 requests per minute
- **Gold**: 5000 requests per minute
- **Unlimited**: Allows unlimited access (you can disable the Unlimited tier by editing the `<EnableUnlimitedTier>` element in the `<ThrottlingConfigurations>` node of the `<API-M_HOME>/repository/conf/api-manager.xml` file.)

Advanced Throttling is enabled by default with following configuration in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

```
<ThrottlingConfigurations>
    <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
    .......
</ThrottlingConfigurations>
```

If you are disabling **Advanced Throttling** in any case by setting the value of `<EnableAdvanceThrottling>` to `false`, Advanced Throttling is disabled and the basic Throttling mechanism is enabled thereafter. In such a scenario, if you want to disable the Unlimited Throttling tier of the basic Throttling configurations, you need to disable it under `<TierManagement>` by setting `<EnableUnlimitedTier>` to `false` in the of the `<API-M_HOME>/repository/conf/api-manager.xml` file.

```
<TierManagement>
    <EnableUnlimitedTier>true</EnableUnlimitedTier>
</TierManagement>
```

It is also possible to specify a bandwidth per unit time instead of a number of requests. This can be done by an API Manager administrator. For information
on editing the values of the existing tiers, defining new tiers, and specifying a bandwidth per unit time, see Adding a new subscription-level throttling tier.

**Note** that when you edit an API with active subscribers, certain things like tier changes do not get automatically reflected to the subscribers. For such changes to take effect, the subscribers should resubscribe to the API and regenerate the access token.

### Rate limiting (burst control)

With rate limiting, you can define tiers with a combination of, for example, a 1000 requests per day and 10 requests per second. Users are then throttled at two layers. Enforcing a rate limit protects the backend from sudden request bursts and controls the usage at a subscription and API level.

For instance, if there's a subscription level policy enforced over a long period, you may not want users to consume the entire quota within a short time span. Sudden spikes in usage or attacks from users can also be handled via rate limiting. You can define a spike arrest policy when the subscription level tier is created. For more information on using rate limiting in subscription tiers, refer Adding a new subscription-level throttling tier.

**Spike Arrest Policy** is used to protect the API backend against large number of traffic spikes and DoS attacks. Unlike setting one definite throttling tier (Quota), it helps to limit the sudden increase of number of requests at any point in time.

As an example, if we specify a quota policy as 20 requests per minute, it is possible to send all 20 requests in first few seconds in one minute so that we cannot limit it. By defining a spike arrest policy as 10 requests per second, it equally scatters the number of requests over the given one minute. Therefore, by doing rate limiting we can protect the backend from sudden spikes and DoS attacks through spike arrest policy.

For each subscription level throttle key, a WS policy is created on demand. The request count is calculated and throttling occurs at the node level. If you are using a clustered deployment, the counters are replicated across the cluster.

**Subscription-level throttling (API subscriber)**

After subscription-level throttling tiers are set and the API is published, at subscription time, the consumers of the API can sign in to the API Store and select which tier (out of those enabled for subscribers) they are interested in, as shown below:

According to the selected tiers, the subscribers are granted a maximum number of requests to the API.

**Advanced throttling (API publisher)**

Advanced throttling policies are applied when you are Publishing an API. It can be further divided into the following two levels base on the applicability.

- **API level throttling**
- **Resource level throttling**

**API level throttling**

API level policies are defined when managing APIs using the API Publisher by selecting **Apply to API** under **Advanced Throttling** policies as shown below.
This will specify the maximum throttling level per minute for the API.

**Resource-Level Throttling**

An API is made up of one or more resources. Each resource handles a particular type of request and is similar to a method (function) in a larger API. You can use this method when handling a large number of request at resource level such as Financial transactions. For example, Imagine API have two resources and one resource take more request than other you do not need to throttle it in API level in that case you can use this. Resource-level throttling tiers are set to HTTP verbs of an API's resources. You can apply resource-level throttling through the Manage tab as shown below:
Advanced Throttling tiers

The default throttling tiers are as follows:

- **10KPerMin**: 10,000 requests per minute
- **20KPerMin**: 20,000 requests per minute
- **50KPerMin**: 50,000 requests per minute
- **Unlimited**: Unlimited access (you can disable the Unlimited tier by editing the `<EnableUnlimitedTier>` element in `<ThrottlingConfigurations>` node which is defined in the `<API-M_HOME>/repository/conf/api-manager.xml` file)

It is also possible to specify a bandwidth per unit time instead of a number of requests. This can be defined via the API Manager Admin Portal. For information on editing the values of the existing tiers, defining new tiers, and specifying a bandwidth per unit time, see Adding a new advanced throttling policy.

Application-level throttling (application developer)
Application-level throttling tiers are defined at the time an application is created in the API Store as shown below. The limits are restricted per token for a specific application.

An application is a logical collection of one or more APIs and is required to subscribe to an API. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe to one API multiple times with different SLA levels.

An application is available to a consumer at different levels of service. For example, if you have infrastructure limitations in facilitating more than a certain number of requests to an application at a time, the throttling tiers can be set accordingly so that the application can have a maximum number of requests within a defined time.

### Application-level Throttling tiers

The default throttling levels are as follows:

- **10PerMin**: 10 requests per minute
- **20PerMin**: 20 requests per minute
- **50PerMin**: 50 requests per minute
- **Unlimited**: Unlimited access. The Default Application, which is provided out of the box has the tier set to Unlimited.

It is also possible to specify a bandwidth per unit time instead of a number of requests. This can be done through the Admin Portal of API Manager. For information on editing the values of the existing tiers, defining new tiers and specifying a bandwidth per unit time, see Adding a new application-level throttling tier.

### Adding New Throttling Policies

WSO2 API Manager admins can add new throttling policies and define extra properties to the throttling policies. To get started, click the level of throttling that you want to add a new policy to:

- Adding a new advanced throttling policy
- Adding a new application-level throttling tier
- Adding a new subscription-level throttling tier

To make changes in the throttling configurations, set the `EnableAdvanceThrottling` parameter in `api-manager.xml`. This parameter is set to `true` by default. If you set it to `false`, you only see the available tiers.
Adding a new advanced throttling policy

You can add advanced throttling policies to both APIs and resources.

2. Click Advanced Throttling under the Throttle Policies section to see the set of existing throttling tiers.
3. To add a new tier, click Add Tier.
4. Fill in the required details and click Save.
4. To add throttling limits with different parameters to the conditions below, click Add Conditional Group.

5. To add throttling limits with different parameters to the conditions below, click Add Conditional Group.

Request Count and Request Bandwidth are the two options for default limit. You can use the option according to your requirement. For example, if you are using API for file sharing, data transmission you can use request bandwidth option and limit the data bandwidth for a given time unit.

Note that if you want to add a header, query param, or JSON Web Token (JWT) claim condition, you need to set the <EnableHeaderConditions>, <EnableJWTClaimConditions> or <EnableQueryParamConditions> element to true (depending on which is required).
You can add Description about condition group by click **Sample description about condition group** under **Condition Group**.

### Conditional Groups

![Conditional Groups](image)

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Condition</td>
<td>Allows you to set a throttling limit for a specific IP address or a range of IP addresses.</td>
</tr>
<tr>
<td>Header Condition</td>
<td>Allows you to set a throttling limit to specific headers and parameters.</td>
</tr>
<tr>
<td>Query Param Condition</td>
<td>Allows you to set a throttling limit to specific query parameters.</td>
</tr>
<tr>
<td>JWT Claim Condition</td>
<td>Allows you to set a throttling limit to specific claims.</td>
</tr>
</tbody>
</table>

Conditional group Execution policy used only for that condition. For example, If you add IP condition and set request count as shown in above diagram then only 5 requests allow per 1 minute using that IP condition. Any request comes with false condition(Outside that condition) taken to the default limit.

6. Turn on the required condition and enter a condition and value.
7. Header condition and JWT claim condition values allow regex patterns to be defined.
   You can configure it to make either an exact match or a pattern match for the value using the regex values. For example,
Once done, click Add.

You have added a new advanced throttling policy. You can now apply it to an API or a resource.

You possibly can configure multiple conditional groups when defining a tier for advanced throttling policies. For example, it's possible to apply IP based throttling and query parameter condition both in one advanced policy tier.

**Adding a new application-level throttling tier**

Application-level throttling policies are applicable per access token generated for an application.

2. Click **Application Tiers** under the **Throttle Policies** section to see the set of existing throttling tiers.
3. To add a new tier, click **Add New Policy**.
4. Fill in the required details and click **Save**.

### Add Application Level Policy

#### General Details

**Name**

100PerMin

**Description**

Allows 100 requests per min

#### Quota Limits

- **Request Count**: 100
- **Unit Time**: 1 minute

You have added a new application-level throttling policy.

**Adding a new subscription-level throttling tier**

1. Sign in to the Admin Portal using the URL [https://localhost:9443/admin](https://localhost:9443/admin) and your admin credentials.
2. Click **Subscription Policies** under the **Throttling Policies** section. The existing set of throttling tiers are displayed.
3. To add a new tier, click **Add New Policy**.
When you are going to add a new Subscription level throttling tier, you can see the existing list of subscription tiers in Subscription Tier List. In this list, you will find a tier named Unauthenticated which have a request quota of 500. This is a subscription tier which automatically applied when the authentication type of your resources is 'None'. That is, when you can invoke APIs without tokens. And this tier is not visible in the Throttling tier list of the application.

4. Fill in the details required by this form and click Save once you are done.
Given below are the descriptions of the fields you find in the form:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Count/Request Bandwidth</td>
<td>The maximum number of requests/maximum bandwidth allowed to the API within the time period given in the next field.</td>
</tr>
<tr>
<td>Unit Time</td>
<td>Time within which the number of requests given in the previous field is allowed to the API. This can be defined in minutes, hours, days, weeks, months or years.</td>
</tr>
</tbody>
</table>
Burst Control (Rate Limiting) You can define the request count/bandwidth per unit time on an addition layer by using rate limiting. This is usually a smaller number of requests/bandwidth for a shorter time span than what is enforced in the above fields. For instance, if there’s a subscription level policy enforced over a long period, you may not want users to consume the entire quota within a short time span. Enforcing a rate limit protects the backend from sudden request bursts and controls the usage at a subscription and API level.

Stop On Quota Reach This indicates the action to be taken when a user goes beyond the allocated quota. If the check box is selected, the user’s requests are dropped and an error response (HTTP Status code 429) is given. If the check box is cleared, the requests are allowed to pass through.

Billing Plan This field only makes sense if you have API Monetization enabled. The available billing plans are Free, Commercial, and Freemium. An API is tagged/labeled as Free, Paid, or Freemium depending on its subscription tiers (e.g., Unlimited, Gold, etc.), which are the tiers selected when creating an API.

- **Free** - If all subscription tiers are defined as Free, the API uses the Free billing plan and the API is labeled as Free in the Store.
- **Paid** - If all subscription tiers are defined as Paid, the API uses the Commercial billing plan and the API is labeled as Paid in the Store.
- **Freemium** - If the API has a combination of Free and Paid subscription tiers, the API uses the Freemium billing plan and the API is labeled as Freemium in the Store.

This labeling happens on the API Store only if monetization has been enabled. For information on how to enable monetization and how to tag subscription tiers, see Configuring API Monetization Category Labels.

Custom Attributes You can choose to display additional information about tiers using custom attributes, during custom implementations. The main objective of these fields is to provide more information regarding the tier to Application Developers at the time of API subscription. An example usage of custom attributes is API Monetization. See Enabling Monetization of APIs for more information on practical usage of custom attributes in the subscription tier.

Permissions You can allow or deny permission for specific roles. Once permission is denied to a role, the new subscription tier that you add here will not be available to that role in the API Store.

You have added a new subscription-level throttling policy.

**Managing Throttling**

This section guides you through the following areas:

- **IP Whitelisting**
  - Creating the Advanced Throttling policy
  - Engage the policy with an API
- **Blacklisting requests**
  - Blacklisting PhoneVerification API
- **Custom throttling**

**IP Whitelisting**

IP whitelisting is a way of configuring a filter to extract a particular set of known IP addresses and grant the access to the given assets for requests comes from those IPs only. With introducing Advanced Throttling in WSO2 API Manager you can achieve IP whitelisting via the features provided by Traffic Manager. For this we are creating an Advanced Throttling policy and attach it to the API.

**Creating the Advanced Throttling policy**

1. Login to the admin portal of WSO2 API Manager (https://<ip_address>:9443/admin).
2. Open Throttling Policies tab and navigate to Advanced Throttling.
3. Click ADD NEW POLICY to add a new Throttling tier.

![Advanced Throttling Policies](image)

4. Fill the details as below and click Add Conditional Group.
5. Open the Conditional Group added and fill the details.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Condition Policy</td>
<td>Checked</td>
</tr>
<tr>
<td>IP Condition Type</td>
<td>Specific IP</td>
</tr>
<tr>
<td>IP Address</td>
<td><code>&lt;IP_Address_to_be_whitelisted&gt; E.g. 193.100.3.106</code></td>
</tr>
<tr>
<td>Invert Condition</td>
<td>Checked (If Invert Condition check then condition only apply to the IP Address above)</td>
</tr>
<tr>
<td>Request Count</td>
<td>0</td>
</tr>
</tbody>
</table>

Following is a example configuration.
5. In above configuration we are whitelisting a Specific IP.

You can whitelist a range of IP as well by selecting **IP Range** for the IP Condition Type in the Conditional Group and specifying the range.

6. Click **Save**.

**Advanced Throttling Policies**

<table>
<thead>
<tr>
<th>Name</th>
<th>Quota Policy</th>
<th>Quota</th>
<th>Unit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10KPerMin</td>
<td>requestCount</td>
<td>10000</td>
<td>1 min</td>
</tr>
<tr>
<td>25KPerMin</td>
<td>requestCount</td>
<td>20000</td>
<td>1 min</td>
</tr>
<tr>
<td>50KPerMin</td>
<td>requestCount</td>
<td>50000</td>
<td>1 min</td>
</tr>
<tr>
<td>whitelistIP</td>
<td>requestCount</td>
<td>8</td>
<td>0 min</td>
</tr>
</tbody>
</table>
You have successfully created the policy. Now we should engage this policy to an API.

**Engage the policy with an API**

1. Login to API Publisher https://<IP_address>:9443/publisher.
2. Edit API and go to Manage tab.
3. Enable **Apply to API** under **Advance Throttling Policies** and select the newly created Throttling policy.

4. Save and Publish the API.
   Now the API will be accessible only by the IP specified in the throttling policy.

Since it takes some time to deploy the policy, the first few requests from the IPs other than the white-listed IP/IPs will be passed through. After the policy is successfully deployed, non-white-listed IP access will be blocked.

**API-M Throttling is asynchronous. When you apply a new whitelisting condition, note that at least one request has to go through for the condition to be applied**

**Blacklisting requests**

By blacklisting requests, you can protect servers from common attacks and abuse by users. For example, if a malicious user misuses the system, all requests received from that particular user can be completely blocked. Tenant administrative users can block requests based on the following parameters:

- Block calls to specific APIs
- Block all calls from a given application
- Block requests coming from a specific IP address
- Block a specific user from accessing APIs

To blacklist a request,

2. Click **Black List** under the **Throttle Policies** section and click **Add Item**.
Select the item to black list, enter a value and click Blacklist.

Select Item to Blacklist

Select Condition Type*

- API Context
- Application
- IP Address
- User

Value: *

Format: ${context}

Eg: test/1.0.0

Note that you have to use "/" always infront of the ${context} value when blacklisting the APIs with API context. E.g. /test/1.0.0. The sample provided in the product does not include "/" due to a known issue.

You can temporary on/off the blacklisting condition by enabling/disabling the Condition status that is auto enabled when a blacklisting condition is created.
Blacklisting PhoneVerification API

As described above you can blacklist requests for APIs, by Applications, to IP Addresses and for Users. Let's see how we can blacklist the requests come to the PhoneVerification API that we published in Quick Start Guide.

2. Click Black List under the Throttle Policies section and click Add Item.
3. Select API Context and provide the Context of PhoneVerification API with version as the Value.
4. Click Blacklist.
5. Now login to API Store using the URL https://localhost:9443/store and invoke the API.
   You will see the following response.
Custom throttling

Custom throttling allows system administrators to define dynamic rules for specific use cases, which are applied globally across all tenants. When a custom throttling policy is created, it is possible to define any policy you like. The Traffic Manager acts as the global throttling engine and is based on the same technology as WSO2 Complex Event Processor (CEP), which uses the Siddhi query language. Users are therefore able to create their own custom throttling policies by writing custom Siddhi queries. The specific combination of attributes being checked in the policy need to be defined as the key (also called the key template). The key template usually includes a predefined format and a set of predefined parameters. It can contain a combination of allowed keys separated by a colon (:), where each key must start with the prefix $. The following keys can be used to create custom throttling policies:

- resourceKey
- userId
- apiContext
- apiVersion
- appTenant
- apiTenant
- appId

For example, the following sample custom policy allows the admin user to send 5 requests per minute to the Pizza Shack API.

**Key Template**

$userId:$apiContext:$apiVersion

**Siddhi query**

```
FROM RequestStream
SELECT userId, (userId == 'admin@carbon.super' and apiContext == '/pizzashack/1.0.0' and apiVersion == '1.0.0') AS isEligible,
str:concat('admin@carbon.super','://','/pizzashack/1.0.0:1.0.0') as throttleKey

INSERT INTO EligibilityStream
FROM EligibilityStream[[isEligible=true]]#window.time(1 min)
SELECT throttleKey, (count(throttleKey) >= 5) as isThrottled group by throttleKey
INSERT ALL EVENTS into ResultStream;
```
As shown in the above Siddhi query, the throttle key must match the key template format. If there is a mismatch between the key template format and the throttle key, requests will not be throttled.

### Enforcing Throttling to an API

**Throttling** allows you to limit the number of hits to an API during a given period, typically to protect your APIs from security attacks and your backend services from overuse, regulate traffic according to infrastructure limitations and to regulate usage for monetization. For information on different levels of throttling in WSO2 API Manager (WSO2 API-M), see [Throttling tiers](#).

This tutorial uses the **PhoneVerification** API, which has one resource, GET and POST methods to access it and a throttling policy enforced.

**Before you begin**, follow the Create and Publish an API to create and publish the **PhoneVerification** API and then the Subscribe to an API to subscribe to the API using the **Bronze** throttling tier.

After you created, published, and subscribed to the API, let's see how the API Gateway enforces throttling and resource access policies to the API.

1. Sign in to the API Store and select the PhoneVerification API. Make sure to subscribe to the API as mentioned in the before you begin section.
2. Go to the Default Application, click the **Production Keys** tab and generate an access token. If you already have an access token for the application, you have to regenerate it after 1 hour.
Let's invoke this API.

3. Click on the API, then go to its API Console tab.
4. Expand the GET method and click Try it out. Provide the required parameters and click **Execute** to invoke the API. For example,

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneNumber</td>
<td>E.g., 18006785432</td>
</tr>
<tr>
<td>LicenseKey</td>
<td>Give 0 for testing purpose</td>
</tr>
</tbody>
</table>
Note the response that appears in the API Console. As we used a valid phone number in this example, the response returns as valid.

Note that you subscribed to the API on the Bronze throttling tier. The Bronze tier allows you to make a 1000 calls to the API per minute. If you exceed your quota, you get a throttling error as shown below.

Let's try to invoke the API using an unavailable resource name.

5. Go to the API's Overview page in the API Store and get the API's URL.
6. Install cURL or any other REST client.
7. Go to the command-line invoke the API using the following cURL command.

```
```

Note that the PhoneVerification API's resource name is CheckPhoneNumber, but we use an undefined resource name as CheckPhoneNumber. Here's an example:

```
curl -k -H "Authorization :Bearer 63cc9779d6557f4346a9a28b5cfd8b53" 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

Note that the call gets blocked by the API Gateway with a 'no matching resource' message. It doesn't reach your backend services as you are trying to access a REST resource that is not defined for the API.

You have seen how the API Gateway enforces throttling and resource access policies for APIs.

### Engaging a new Throttling Policy at Runtime

WSO2 API Manager allows you to control the number of successful requests to your API during a given period. You can enable for APIs in the CREATED and PUBLISHED state and also for published APIs at runtime. This feature protects your APIs, regulates traffic and access to the resources.

The steps below show how to engage a throttling policy to an API at runtime.

1. Write a new throttling policy. For example, the following sample throttling policy points to a backend service and allows 1000 concurrent requests to a service.

```
xmlns:throttle="http://www.wso2.org/products/wso2commons/throttle"
wsu:Id="WSO2MediatorThrottlingPolicy">
  <throttle:MediatorThrottleAssertion>
    <throttle:MaximumConcurrentAccess>1000</throttle:MaximumConcurrentAccess>
  </wsp:Policy>
</wsp:Policy>
```

**Attributes**

- Throttle policy - This section is used to specify the policy for throttling.
- Maximum concurrent accesses - The maximum number of messages that are served at a given time.
- Throttle assertion - Assertion for a concurrency-based policy.

2. Log in to the API Manager’s management console (https://localhost:9443/carbon) and go to the Resource > Browse menu to view the registry.
3. Click the `/system/governance/apimgt/applicationdata` path to go to its detailed view.

4. In the detail view, click Add Resource.

5. Upload the policy file to the server as a registry resource.

6. Open the synapse configuration file of a selected API you want to engage the policy, from the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/api` directory.

7. To engage the policy to a selected API, add it to your API definition. In this example, we add it to the login API under APIThrottleHandler.
<api xmlns="http://ws.apache.org/ns/synapse" name="_WSO2AMLoginAPI_" context="/login">
  <resource methods="POST" url-mapping="/"/>
    <inSequence>
      <send>
        <endpoint>
          <address uri="https://localhost:9493/oauth2/token"/>
        </endpoint>
      </send>
    </inSequence>
    <outSequence>
      <send/>
    </outSequence>
  </resource>
  <handlers>
    <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
      <property name="id" value="A"/>
      <property name="policyKey" value="gov:/apimgt/applicationdata/throttle.xml"/>
    </handler>
    <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
      <property name="id" value="B"/>
      <property name="policyKey" value="gov:/apimgt/applicationdata/throttle-l2.xml"/>
    </handler>
    <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
  </handlers>
</api>

You have successfully engaged a throttling policy to an API at runtime, without restarting the server.

Engaging Multiple Throttling Policies to a Single API

You can apply different throttling policies at the same time to a single API. This is called **multi-layer throttling**.

The following example shows how to have two throttling policies for a single API at a given time. The table below shows the throttling information of the two throttling policies.

<table>
<thead>
<tr>
<th>Tier</th>
<th>throttle-l1</th>
<th>throttle-l2</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>300 per month</td>
<td>5 per 3 minutes</td>
</tr>
<tr>
<td>Silver</td>
<td>2000 per month</td>
<td>1 per 5 seconds</td>
</tr>
<tr>
<td>Gold - Unlimited</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
</tbody>
</table>

To engage the two throttling layers, you add two throttling tier definitions and engage them to the API using the steps below:

1. Go to the Synapse configuration file of the particular API located in `<API-M_HOME>/repository/deployment/server/synapse-configs/default/api`.
2. Copy the following content inside the `<handlers>` section in the API configuration.

```xml
<handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
<handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
  <property name="id" value="A"/>
  <property name="policyKey" value="gov:/apimgt/applicationdata/throttle.xml"/>
</handler>
<handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
  <property name="id" value="B"/>
  <property name="policyKey" value="gov:/apimgt/applicationdata/throttle-l2.xml"/>
</handler>
```

3. Replace the existing content of the `/_system/governance/apimgt/applicationdata/tiers.xml` file with following content.
Create an XML as `throttling-l1.xml` with the following content and add it to `/_system/governance/apimgt/applicationdata` registry location. The code adds two policies for each role (free, Silver, Gold) and engages them to the APIs with different keys. Both throttling layers execute in runtime sequentially.
Working with Endpoints

An endpoint is a specific destination for a message such as an address, WSDL, a failover group, a load-balance group etc. WSO2 API Manager supports a range of different endpoint types, allowing the API Gateway to connect with advanced types of backends.

<table>
<thead>
<tr>
<th>Endpoint Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP endpoint</td>
<td>A REST service endpoint based on a URI template.</td>
</tr>
<tr>
<td>Address endpoint</td>
<td>The direct URL of the service.</td>
</tr>
</tbody>
</table>
Failover Group endpoint
- The endpoints that the service tries to connect to in case of a failure. Selecting the endpoint when the primary endpoint gets failed happens in a round robin manner.
- Failover Group is a group of leaf endpoints (i.e., address endpoint, HTTP endpoint, and WSDL endpoint). The failover group endpoint tries to send the message to another endpoint when failure occurs in the current endpoint (while sending a message). Failover group ensures that a message is delivered as long as there is at least one active endpoint among the listed endpoints.

Load Balance endpoint
- The endpoints where the incoming requests are directed to in a round robin manner. They automatically handle fail-over as well.

Dynamic endpoint
- The dynamic endpoint sends the message to the address specified in the To header. You can configure dynamic endpoints by setting mediation extensions with a set of conditions to dynamically change the To header. For details of configuring endpoints to change the default mediation flow, see Adding Mediation Extensions.

Note the following:
- You can expose both REST and SOAP services to consumers through APIs.
- You cannot call backend services secured with OAuth through APIs created in the API Publisher. At the moment, you can call only services secured with username/password.
- The system reads gateway endpoints from the `<APIM_HOME>/repository/conf/api-manager.xml` file. When there are multiple gateway environments defined, it picks the gateway endpoint of the production environment. You can define both HTTP and HTTPS gateway endpoints as follows:

```xml
```

If both types of endpoints are defined, the HTTPS endpoint will be picked as the server endpoint.

**Tip:** When you define secure (HTTPS) endpoints, set the `<parameter name="HostnameVerifier"> element to `AllowAll` in the `<APIM_HOME>/repository/conf/axis2/axis2.xml` file's HTTPS transport sender configuration:

```xml
<parameter name="HostnameVerifier">AllowAll</parameter>
```

If not, the server throws an exception.

### Configuring Load Balancing Endpoints

WSO2 API Manager provides configuring load balancing endpoints through the API Publisher.

To configure load balanced endpoints go to the edit view of the API and navigate to the Implement tab and select the `Load Balanced` check box.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>

Following are the other configurations that you need to do in order to specify a load balancing endpoint.
The set of production endpoints can be specified here where the requests need to be load balanced. You can specify more than one endpoint by clicking on the "+" sign and can delete the endpoints by clicking on the "−" sign.

<table>
<thead>
<tr>
<th>Production Endpoints</th>
<th>The set of production endpoints can be specified here where the requests need to be load balanced. You can specify more than one endpoint by clicking on the &quot;+&quot; sign and can delete the endpoints by clicking on the &quot;−&quot; sign.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandbox endpoints</td>
<td>The set of sandbox endpoints can be specified here where the requests need to be load balanced. You can specify more than one endpoint by clicking on the &quot;+&quot; sign and can delete the endpoints by clicking on the &quot;−&quot; sign.</td>
</tr>
</tbody>
</table>

The load balancing algorithm is specified here.

The default is Round Robin Algorithm which has the className of `org.apache.synapse.endpoints.algorithms.RoundRobin`. If you select other from the dropdown list of Algorithms you need to specify the class name of the algorithm. Classnames of other algorithms can be found here.

<table>
<thead>
<tr>
<th>Session Management</th>
<th>A session management method from the load balancing group. The possible values are as follows.</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>If this is selected, session management is not used.</td>
</tr>
<tr>
<td>Transport</td>
<td>If this is selected, session management is done on the transport level using HTTP cookies.</td>
</tr>
<tr>
<td>SOAP</td>
<td>If this is selected, session management is done using SOAP sessions.</td>
</tr>
<tr>
<td>Client ID</td>
<td>If this is selected, session management is done using an ID sent by the client.</td>
</tr>
</tbody>
</table>

The number of milliseconds after which the session would time out.

After completing the configuration click save and publish the API.

**Configuring Failover Group of Endpoints**

WSO2 API Manager provides configuring failover group endpoints through the API Publisher.
To configure failover endpoints go to the edit view of the API and navigate to the **Implement** tab and click **Failover** under endpoint type.

At least one failover endpoint need to be added for production and sandbox (if you have specified) Endpoints.

You can specify more than one endpoint by clicking on "+" sign and can delete the endpoints by clicking on "-" sign.

After completing the adding endpoints, click save and publish the API.

**Advanced Endpoint Configuration**

WSO2 API Manager provides controlling the production and sandbox endpoints with the Advanced Endpoint Configuration.

To configure your endpoints with this feature, go to the edit view of the API, navigate to the **Implement** tab and click the cogwheel icon next to the endpoint you want to configure.

The Advanced Endpoint Configuration dialog box appears as below.
Following are the configurations that we can do with the Advanced Endpoint Configurations. You can do advanced configurations for both production and sandbox endpoints.

<table>
<thead>
<tr>
<th>Endpoint Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>
| Endpoint Suspend State | If you want to configure the suspension of an endpoint specifying error codes, maximum suspension time, suspension factors etc., you can use the endpoint suspension state in the Advanced Endpoint Configuration.  
**Error Codes:** Error codes in the drop down list which need to make the endpoint suspension. If the selected error codes are received from the endpoint, the endpoint will be suspended. Specify the transport error codes where the endpoint suspension should be triggered. You can select single or error codes here.  
**Initial duration:** The time duration for which the endpoint will be suspended, when one or more suspend error codes are received from it for the first time.  
When creating (or updating) Failover endpoints through the Publisher UI (in the Implement tab), you need to go into this configuration box of each endpoint and specify a set of Error Codes for the endpoint to fail over on and take off the Initial Duration by setting its value to -1.  
**Max duration:** The maximum time duration for which the endpoint is suspended when suspend error codes are received from it.  
**Factor:** The duration to suspend can vary from the first time suspension to the subsequent time. The factor value decides the suspense duration variance between subsequent suspensions. |
### Endpoint Timeout state
Configurations of retry, error codes and delays in terms of endpoint timeout can be configured with the endpoint timeout state in the Advanced Endpoint Configuration.

**Error Codes:** A list of error codes. If these error codes are received from the endpoint, the request will be subjected to a timeout.

**Retries Before Suspension:** The number of re-tries in case of a timeout, caused by the above listed error codes.

**Retry Delay:** The delay between retries in milliseconds.

### Connection Timeout
Duration and the response actions after a connection timeout can be configured here in the Advanced Endpoint Configuration.

**Action:** Response action to be performed after connection timeout. You can select from **Never Timeout, Discard Message, and Execute Fault Sequence**. The default value is **Execute Fault Sequence**.

**Duration:** The time duration of connection timeout in milliseconds.

To configure the endpoint connection timeout:

1. Open the `<APIM_HOME>/repository/conf/synapse.properties` file and change the value of the timeout as given below.

   ```properties
   synapse.global_timeout_interval=30000
   ```

2. Open the `<ESB_HOME>/repository/conf/passthru-http.properties` file and change the socket timeout value.

   ```properties
   http.socket.timeout=30000
   ```

   Note that the **socket timeout value** should be greater than both the synapse global timeout and any endpoint timeouts given for your API.

For more information about endpoints and how to add, edit or delete them, see the WSO2 ESB documentation.

## Working with Analytics

The API Manager integrates with API Manager Analytics to provide reports, statistics and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs and detect unusual activity, manage locations via geo-location statistics and carry out a detailed analysis of the logs. WSO2 API Manager has an enhanced distribution of Analytics to cater to API Manager specific scenarios.

API Manager Analytics consists of two components: **Worker** and **Dashboard**. The worker is the server that processes the data streams that are sent from WSO2 API Manager and publishes the statistics to a database. The dashboard reads the statistics published by the worker and displays the statistics on the dashboard. The worker and dashboard are connected through the database.

See the following topics for more details:

- Configuring API Manager Analytics
- Analyzing API Manager Analytics
- Managing Alerts
- Integrating with Google Analytics
- Monitoring with WSO2 Carbon Metrics
- Purging Analytics Data
- Default Ports of WSO2 API Manager Analytics
- Enabling API Manager Analytics Features in WSO2 Stream Processor

### Configuring API Manager Analytics

This section explains how to configure analytics for WSO2 API Manager (WSO2 API-M). The API Manager integrates with the WSO2 Stream Processor 4.3.0 to provide reports, statistics and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and carry out detailed analysis of the logs. WSO2 API Manager has an enhanced distribution of Analytics to cater to API Manager specific scenarios which is used here to configure API Manager Analytics.

By default, WSO2 API Manager has a port offset of 0 (no port offset) and WSO2 API Manager Analytics has an offset of 1. Therefore, this guide assumes that you do not have any other Carbon servers running on the same machine with port offsets of 0 or 1.

Click on the **Quick Setup** tab to set up analytics for a quick demo in order to try-out the scenarios, or click on the **Standard Setup** tab to set up analytics for a production environment.
Follow the instructions below if you wish to set up API-M Analytics for a quick demo in order to try-out the scenarios.

1. Download and install WSO2 API-M.
   Download WSO2 API-M via the WSO2 API Manager page. For more information on installing WSO2 API-M, see the Installation Guide.

2. Download and install WSO2 API-M Analytics.
   Download WSO2 API-M Analytics via the WSO2 API-M Analytics page. As the process of installing API-M Analytics is identical to installing WSO2 Stream Processor Server 4.3.0 (WSO2 SP), the following steps direct you to the WSO2 SP documentation for details.
   a. Ensure that you have met the Installation Prerequisites.
   b. Install WSO2 API-M Analytics based on the installation instructions that is relevant to your operating system:
      - Installing on Linux
      - Installing on Windows
      - Installing as a Windows Service
      - Installing as a Linux Service

3. To enable Analytics, open the <API-M_HOME>/repository/conf/api-manager.xml file and set the Enabled property under Analytics to true as shown below. Save this change.

   ```xml
   <Analytics>
   <Enabled>true</Enabled>
   ...
   </Analytics>
   ```

If you are working on a distributed (clustered) setup of WSO2 API Manager, do the configurations instructed to be done in API Manager in the Publisher, Store, and Gateway nodes as well.

4. Start the Analytics Server.
   Navigate to the <API-M_ANALYTICS_HOME>/bin directory in your console and execute one of the following scripts based on your OS.
   - On Windows: worker.bat --run
   - On Linux/Mac OS: sh worker.sh

5. Start the API Manager server.
   Navigate to the <API-M_HOME>/bin directory in your console and execute one of the following scripts based on your OS.
   - On Windows: wso2server.bat --run
   - On Linux/Mac OS: sh wso2server.sh

If API-M Analytics is properly configured in WSO2 API Manager, when you start up the API Manager server, which is after the WSO2 API-M Analytics server, you will see the following log message in the terminal that is running the API-M Analytics server.

```
INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
```

6. Optionally, if you wish to access the business rules via the Dashboard node, navigate to the <API-M_ANALYTICS_HOME>/bin directory in your console and execute one of the following scripts based on your OS.
   - On Windows: dashboard.bat --run
   - On Linux/Mac OS: sh dashboard.sh

You can now start using the WSO2 API Manager for its usual operations and the required Analytics functionality.

Follow the instructions below if you wish to set up API-M Analytics for a production environment.

- Step 1 - Download and install WSO2 API-M
- Step 2 - Download and install WSO2 API-M Analytics
- Step 3 - Configure WSO2 API Manager to publish statistics
- Step 4 - Configure databases
Step 5 - Configure keystores

Step 1 - Download and install WSO2 API-M

Download WSO2 API-M via the WSO2 API Manager page. For more information on installing WSO2 API-M, see the Installation Guide.

Step 2 - Download and install WSO2 API-M Analytics

Download WSO2 API-M Analytics via the WSO2 API-M Analytics page. As the process of installing API-M Analytics is identical to installing WSO2 Stream Processor Server 4.3.0 (WSO2 SP), the following steps direct you to the WSO2 SP documentation for details.

1. Ensure that you have met the Installation Prerequisites.
2. Install WSO2 API-M Analytics based on the installation instructions that is relevant to your operating system:
   - Installing on Linux
   - Installing on Windows
   - Installing as a Windows Service
   - Installing as a Linux Service

Step 3 - Configure WSO2 API Manager to publish statistics

Follow the instructions below to do the required configurations for WSO2 API-M to publish statistics in the WSO2 API-M Analytics server.

To download the WSO2 API Manager distribution, click DOWNLOAD and then click DOWNLOAD Server in the WSO2 API Manager page.

If you are working on a distributed (clustered) setup of API Manager, carry out the instructed configurations in the Publisher, Store and Gateway nodes of the API Manager.

1. Open the <API-M_HOME>/repository/conf/api-manager.xml file.
2. Under the <Analytics> sub element, set the Enabled parameter to true.
3. Configure the following parameters if required.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;StreamProcessorServerURL&gt;</td>
<td>&lt;protocol&gt;://&lt;hostname&gt;:&lt;port&gt;/</td>
<td>The server URL of the remote WSO2 API-M Analytics server. An event can also be published to multiple receiver groups delimited by curly braces, whereas receivers e.g., Three receivers within a single group can be specified as tcp://localhost:7612,tcp://localhost:7613,tcp://localhost:7614. Two receiver groups with two receivers each can be specified as '{tcp://localhost:7612,tcp://localhost:7613},{tcp://localhost:7712,tcp://localhost:7713}'.</td>
</tr>
<tr>
<td>&lt;StreamProcessorUsername&gt;</td>
<td>A valid administrator username</td>
<td>The administrator username to log into the remote WSO2 API-M. If you enable email user, you need to configure the email user credentials. Email user credentials must be <a href="mailto:demo@wso2.com">demo@wso2.com</a> @carbon.super. It is required to change the values StreamProcessorUsername and StreamProcessorPassword if the user of WSO2 API-M is different to the user of WSO2 API-M Analytics.</td>
</tr>
</tbody>
</table>

...
4. **Save the changes.**

**Step 4 - Configure databases**

Configuring databases allow you to persist data relating to APIs, process them and analyze. Follow the procedure below to configure databases.

The following is a list of database versions that are compatible with WSO2 API-M Analytics.

- Postgres 9.5 and later
- MySQL 5.6
- MySQL 5.7
- Oracle 12c
- MS SQL Server 2012
- DB2

1. Stop the WSO2 API-M Analytics server if it is running already.
2. To configure the dashboard profile, open the `<API-M_ANALYTICS_HOME>/conf/dashboard/deployment.yaml` file and edit the `APIM_ANALYTICS_DB` section. A sample for MySQL is shown below.

```yaml
- name: APIM_ANALYTICS_DB
description: "The datasource used for APIM statistics aggregated data."
jndiConfig:
  name: jdbc/APIM_ANALYTICS_DB
definition:
type: RDBMS
configuration:
jdbcUrl: 'jdbc:mysql://localhost:3306/stats_db'
username: 'root'
password: '123'
driverClassName: com.mysql.jdbc.Driver
maxPoolSize: 50
idleTimeout: 60000
connectionTestQuery: SELECT 1
validationTimeout: 30000
isAutoCommit: false
```

3. To configure the worker profile, open the `<API-M_ANALYTICS_HOME>/conf/worker/deployment.yaml` file and repeat the above step.
4. Configure `WSO2AM_MGW_ANALYTICS_DB` to the same database as `APIM_ANALYTICS_DB` in the `<API-M_ANALYTICS_HOME>/conf/worker/deployment.yaml` file.
4. **jndiConfig**
   - name: WSO2AM_MGW_ANALYTICS_DB
     - description: "The datasource used for APIM MGW analytics data."
     - configuration:
       - type: RDBMS
       - definition:
         - jdbcUrl: 'jdbc:mysql://localhost:3306/stats_db'
         - username: 'root'
         - password: '123'
         - driverClassName: com.mysql.jdbc.Driver
         - maxPoolSize: 50
         - idleTimeout: 60000
         - connectionTestQuery: SELECT 1
         - validationTimeout: 30000
         - isAutoCommit: false

5. **Create the AM_USAGE_UPLOADED_FILES table in the WSO2AM_STATS_DB database.**
   A sample MySQL script to create the table is given below.
   ```sql
   CREATE TABLE IF NOT EXISTS AM_USAGE_UPLOADED_FILES (
   FILE_NAME varchar(255) NOT NULL,
   FILE_TIMESTAMP TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FILE_PROCESSED tinyint(1) DEFAULT 0,
   FILE_CONTENT MEDIUMBLOB DEFAULT NULL,
   PRIMARY KEY (FILE_NAME, FILE_TIMESTAMP)
   );
   ```

6. **Do the following to integrate third-party products when configuring databases.**

   WSO2 SP is a OSGi-based product. Therefore, when you integrate third party products such as Oracle with WSO2 API-M Analytics, you need to check whether the libraries you need to add are OSGi based. If they are not, you need to convert them to OSGi bundles before adding them to the `<API-M_ANALYTICS_HOME>/lib` directory.

   To convert the jar files to OSGi bundles, follow the steps below.
   1. Download the non-OSGi jar for the required third party product, and save it in a preferred directory in your machine.
   2. Go to the `<API-M_ANALYTICS_HOME>/bin` directory. Run the command given below, to generate the converted file in the `<API-M_ANALYTICS_HOME>/lib` directory.

   ```bash
   ./jartobundle.sh <PATH_TO_NON-OSGi_JAR> ../lib
   ```

7. **Start the WSO2 API-M Analytics server.**

   **Step 5 - Configure keystores**

   In the SSL handshake between the API Manager and API Manager Analytics servers, the client (i.e. API Manager) needs to verify the certificate presented by the server (i.e. API Manager Analytics). For this purpose, the client stores the trusted certificate of the server in the `client-truststore.jks` keystore.

   If you use a custom keystore in API Manager and/or API Manager Analytics, import the public key certificate of API Manager Analytics into the `client-truststore.jks` file of the API Manager. To export the public key from the server and import it into the client's trust store, follow the steps given in Adding CA-signed certificates to keystores in the Administration Guide.

   **Analyzing APIM Statistics**

   The following topics cover how statistics relating to the APIs deployed in the WSO2 API Manager are analyzed via APIM - Analytics.
Introducing the WSO2 API Manager Statistics Model

Introduction

This section describes and illustrates the API Manager statistic publishing and generating model. It describes the internal components of API Manager, external analyzer information and other data retrieval components. API Manager generates events based on the API Manager invocation pattern and publishes them to all the listening event analyzers. The analyzer is responsible for the accumulation of all events and generates summary data based on the defined summarization logic. After the summarized data is generated, the API Manager Dashboard can retrieve statistics from the analyzer data-source to the UI via the API Manager analytics client.

API Manager usage publisher

The internal API Manager component listens to the API Manager invocations and its behavior. Based on the request and responses, the event is generated and published to all the event receivers. This publisher publishes the following event streams,

- org.wso2.apimgt.statistics.request
- org.wso2.apimgt.statistics.fault
- org.wso2.apimgt.statistics.throttle

API Manager event streams

API-M provides the following types of event streams as listed below.

**org.wso2.apimgt.statistics.request**

This stream tracks information for the API request.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta_clientType</td>
<td>string</td>
<td>Meta information of Client type</td>
</tr>
<tr>
<td>applicationConsumerKey</td>
<td>string</td>
<td>Consumer key of API invoked client application</td>
</tr>
<tr>
<td>applicationName</td>
<td>string</td>
<td>Name of the client application</td>
</tr>
<tr>
<td>applicationId</td>
<td>string</td>
<td>ID of the client application</td>
</tr>
<tr>
<td>applicationOwner</td>
<td>string</td>
<td>Name of the Owner of the Application</td>
</tr>
<tr>
<td>apiContext</td>
<td>string</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiName</td>
<td>string</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>string</td>
<td>API version</td>
</tr>
<tr>
<td>apiResourcePath</td>
<td>string</td>
<td>API resource path of the API request</td>
</tr>
<tr>
<td>apiResourceTemplate</td>
<td>string</td>
<td>API resource URL pattern of API request</td>
</tr>
<tr>
<td>apiMethod</td>
<td>string</td>
<td>HTTP Verb of API request [e.g., GET/POST]</td>
</tr>
<tr>
<td>apiCreator</td>
<td>string</td>
<td>Creator of the API</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>string</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>apiTier</td>
<td>string</td>
<td>Subscription tier associated with the API request</td>
</tr>
<tr>
<td>apiHostname</td>
<td>string</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>username</td>
<td>string</td>
<td>Enduser of the API request</td>
</tr>
<tr>
<td>userTenantDomain</td>
<td>string</td>
<td>Tenant domain of the user associated with the request</td>
</tr>
<tr>
<td>userIp</td>
<td>string</td>
<td>IP address of the client</td>
</tr>
<tr>
<td>userAgent</td>
<td>string</td>
<td>User agent of the user</td>
</tr>
</tbody>
</table>
requestTimestamp : long : Timestamp of the API request when received to the gateway
throttledOut : bool : Describes whether the request was allowed after hitting the throttle tier
responseTime : long : Timestamp of the API response when received to the gateway
serviceTime : long : Time taken to serve the API request in APIM side
backendTime : long : Time taken to process the request at the backend
responseCacheHit : bool : Describes if response caching is enabled or not
responseSize : long : Response message size in bytes
protocol : string : Protocol used to send the response (HTTP/HTTPS) and the port
responseCode : int : HTTP Response Code
destination : string : URL of the endpoint
securityLatency : long : Time taken for authentication
throttlingLatency : long : Time taken for throttling the request/response
requestMediationLatency : long : Time taken to mediate the request
responseMediationLatency : long : Time taken to mediate the response
backendLatency : long : Time taken by the backend to return the response
otherLatency : long : Time taken to process tasks other than mentioned above
gatewayType : string : The gateway type (Synapse/Micro)
label : string : The label of the API (if specified)

org.wso2.apimgt.statistics.fault

This stream contains the fault API invocations. It includes the API with back end errors, timeout etc.

meta_clientType : string : Meta information of Client type
applicationConsumerKey : string : Consumer key of API invoked client application
apiName : string : API Name
apiVersion : string : API Version
apiContext : string : API context depending on the user's request
apiResourcePath : string : API resource path of the API request
apiMethod : string : HTTP Verb of API request [e.g.,GET/POST]
apiCreator : string : Creator of the API
username : string : Enduser of the API request
userTenantDomain : string : Tenant domain of the user associated with the request
apiCreatorTenantDomain : string : Tenant domain of the API creator
hostname : string : Hostname or Datacenter ID (if specified)
applicationId : string : ID of the client application
applicationName : string : Name of the client application
protocol : string : Protocol used to send the response (HTTP/HTTPS) and the port
errorCode : string : Synapse error code
errorMessage : string : Description of the synapse error message
requestTimestamp : long : Timestamp of the API request when received to the gateway
This stream contains the API invocation with throttle information. Throttling can happen due to any of the following reasons:

- The application limit has exceeded.
- The resource limit has exceeded.
- The API limit has exceeded.
- The hard level limit has exceeded.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta_clientType</td>
<td>string</td>
<td>Meta information of Client type</td>
</tr>
<tr>
<td>username</td>
<td>string</td>
<td>Enduser of the API request</td>
</tr>
<tr>
<td>userTenantDomain</td>
<td>string</td>
<td>Tenant domain of the user associated with the request</td>
</tr>
<tr>
<td>apiName</td>
<td>string</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>string</td>
<td>API Version</td>
</tr>
<tr>
<td>apiContext</td>
<td>string</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiCreator</td>
<td>string</td>
<td>Creator of the API</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>string</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>applicationId</td>
<td>string</td>
<td>ID of the client application</td>
</tr>
<tr>
<td>applicationName</td>
<td>string</td>
<td>Name of the client application</td>
</tr>
<tr>
<td>subscriber</td>
<td>string</td>
<td>Name of the subscriber of the Application</td>
</tr>
<tr>
<td>throttledOutReason</td>
<td>string</td>
<td>The reason describing why the request has been throttled out</td>
</tr>
<tr>
<td>gatewayType</td>
<td>string</td>
<td>The gateway type (Synapse/Micro)</td>
</tr>
<tr>
<td>throttledOutTimestamp</td>
<td>long</td>
<td>Timestamp when the request is throttled out</td>
</tr>
</tbody>
</table>

API Manager aggregate tables

The summarized tables are stored in the APIM Analytics internal storage. All events are stored in tables based on granularity (SECONDS, MINUTES, HOURS, DAYS, MONTHS, YEARS). Each such table contains the aggregation of stream data per granularity level. For example, each record of the _HOUR_S table contains the aggregate of the events that arrived each hour.

It is important to note that directly querying the tables is not recommended and instead you can use the WSO2 Stream Processor REST APIs (for retrieving from aggregations), similar to what has been done in API Manager Analytics for Publisher and Store statistic graphs.

The reason for this recommendation is because of the granular level aggregations that are present for a single event use case. Events are processed in-memory before adding them to the granular level tables. Therefore, if queried directly from the database, and if there are any aggregate data related to an ongoing operation, such aggregated data will not be taken into consideration. Furthermore, retrieving information directly from the tables will result in complex queries. Therefore, you can use the Stream Processor REST APIs for querying the tables in order to easily obtain accurate information, because all workings are handled internally by the Stream Processor REST APIs.

Siddhi Apps, which are deployed in the Worker profile of API Manager Analytics instance, are used in order to populate data to the aggregate tables. The latter mentioned Siddhi Apps are available in the `<API-M_ANALYTICS_HOME>/wso2/worker/deployment/siddhi-files` directory. The following are the Siddhi Apps that are used for storing data required for charts and graphs:

- `APIM_ACCESS_SUMMARY.siddhi`
- `APIM_FAULT_SUMMARY.siddhi`
- `APIM_THROTTLED_OUT_SUMMARY.siddhi`

The following subsections describe the table schema of each of the Aggregate tables that are present in the Statistics DB.

- `ApiUserPerAppAgg`
- `ApiResPathPerApp`
- `ApiPerDestinationAgg`
- `ApiVersionPerAppAgg`
- `ApiExeTime`
- `ApiUserBrowserAgg`
- `APIM_ReqCountAgg`
- `ApiLastAccessSummary`
- `GeoLocationAgg`
- `ApiFaultyInvocationAgg`
- `ApiThrottledOutAgg`

**ApiUserPerAppAgg**

This aggregation contains summary data from the request event stream and you can use it to obtain information about each API request.

**ApiUserPerAppAgg Table Schema**

This schema resembles the `API_REQUEST_SUMMARY` table from the previous Stat DB schema.
CREATE TABLE `ApiUserPerAppAgg_<granularity>` (  `AGG_TIMESTAMP` bigint(20) NOT NULL,  `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,  `apiContext` varchar(254) NOT NULL,  `apiHostname` varchar(254) NOT NULL,  `applicationId` varchar(254) NOT NULL,  `username` varchar(254) NOT NULL,  `userTenantDomain` varchar(254) NOT NULL,  `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,  `apiName` varchar(254) DEFAULT NULL,  `apiVersion` varchar(254) DEFAULT NULL,  `apiCreator` varchar(254) DEFAULT NULL,  `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,  `applicationName` varchar(254) DEFAULT NULL,  `applicationOwner` varchar(254) DEFAULT NULL,  `gatewayType` varchar(254) DEFAULT NULL,  `label` varchar(254) DEFAULT NULL,  `regionalID` varchar(254) DEFAULT NULL,  `AGG_COUNT` bigint(20) DEFAULT NULL,  PRIMARY KEY (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`apiHostname`,`applicationId`,`username`,`userTenantDomain`) )

Click here for detailed descriptions on the ApiUserPerAppAgg Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The start timestamp of the time range based in the granularity for the event timestamp (For instance if the event is generated at 11:30:21, the agg_event_timestamp for hour granularity is 11:00:00)</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiHostname</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>applicationId</td>
<td>ID of the client application which is used to invoke the API</td>
</tr>
<tr>
<td>username</td>
<td>End user of the API request</td>
</tr>
<tr>
<td>userTenantDomain</td>
<td>Tenant domain of the user associated with the request</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request(event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the client application used to invoke the API</td>
</tr>
<tr>
<td>applicationOwner</td>
<td>Name of the owner of the client application</td>
</tr>
<tr>
<td>gatewayType</td>
<td>Type of the gateway (Synapse/Micro)</td>
</tr>
<tr>
<td>label</td>
<td>The label of the API (if specified)</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occurred within the relevant time interval specified by AGG_TIMESTAMP based on granularity</td>
</tr>
</tbody>
</table>

_ApiResPathPerApp_

This aggregation contains summarized data about API usage by resources and it is also derived from the request event stream.
ApiResPathPerApp Table Schema
This schema resembles the API_Resource_USAGE_SUMMARY table from the previous Stat DB schema.

CREATE TABLE `ApiResPathPerApp_<granularity>` (  
  `AGG_TIMESTAMP` bigint(20) NOT NULL,  
  `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,  
  `apiContext` varchar(254) NOT NULL,  
  `apiHostname` varchar(254) NOT NULL,  
  `applicationId` varchar(254) NOT NULL,  
  `apiResourceTemplate` varchar(254) NOT NULL,  
  `apiMethod` varchar(254) NOT NULL,  
  `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,  
  `apiName` varchar(254) DEFAULT NULL,  
  `apiVersion` varchar(254) DEFAULT NULL,  
  `apiCreator` varchar(254) DEFAULT NULL,  
  `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,  
  `applicationName` varchar(254) DEFAULT NULL,  
  `gatewayType` varchar(254) DEFAULT NULL,  
  `label` varchar(254) DEFAULT NULL,  
  `regionalID` varchar(254) DEFAULT NULL,  
  `AGG_COUNT` bigint(20) DEFAULT NULL,  
  PRIMARY KEY  
  (`AGG_TIMESTAMP`, `AGG_EVENT_TIMESTAMP`, `apiContext`, `apiHostname`, `applicationId`, `apiResourceTemplate`, `apiMethod`)  
)

Click here for detailed descriptions on the ApiResPathPerApp Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table (e.g., start timestamp of the hour in the table with _HOURS granularity).</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiHostname</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>applicationId</td>
<td>ID of the client application which is used to invoke the API</td>
</tr>
<tr>
<td>apiResourceTemplate</td>
<td>API resource URL pattern of API request</td>
</tr>
<tr>
<td>apiMethod</td>
<td>HTTP Verb of API request [e.g., GET/POST]</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_EVENT_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the client application used to invoke the API</td>
</tr>
<tr>
<td>gatewayType</td>
<td>Type of the gateway (Synapse/Micro)</td>
</tr>
<tr>
<td>label</td>
<td>The label of the API (if specified)</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occured within the relevant time interval specified by AGG_TIMESTAMP based on granularity</td>
</tr>
</tbody>
</table>

ApiPerDestinationAgg
This aggregation contains summarized data of API destinations and is derived from the request event stream.

ApiPerDestinationAgg Table Schema
This schema resembles the API_DESTINATION_SUMMARY table from the previous Stat DB schema.

```sql
CREATE TABLE `ApiPerDestinationAgg_<granularity>` (
    `AGG_TIMESTAMP` bigint(20) NOT NULL,
    `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,
    `apiContext` varchar(254) NOT NULL,
    `apiHostname` varchar(254) NOT NULL,
    `destination` varchar(254) NOT NULL,
    `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,
    `apiName` varchar(254) DEFAULT NULL,
    `apiVersion` varchar(254) DEFAULT NULL,
    `apiCreator` varchar(254) DEFAULT NULL,
    `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,
    `gatewayType` varchar(254) DEFAULT NULL,
    `label` varchar(254) DEFAULT NULL,
    `regionalID` varchar(254) DEFAULT NULL,
    `AGG_COUNT` bigint(20) DEFAULT NULL,
    PRIMARY KEY (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`apiHostname`,`destination`)
)
```

Click here for detailed descriptions on the ApiPerDestinationAgg Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user’s request</td>
</tr>
<tr>
<td>apiHostname</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>destination</td>
<td>URL of the endpoint</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>gatewayType</td>
<td>Type of the gateway (Synapse/Micro)</td>
</tr>
<tr>
<td>label</td>
<td>The label of the API (if specified)</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occurred within the relevant time interval specified by AGG_TIMESTAMP based on granularity.</td>
</tr>
</tbody>
</table>

ApiVersionPerAppAgg

This aggregation contains summarized data about API usage and is also derived from the request event stream.

ApiVersionPerAppAgg Table Schema

This schema resembles the API_VERSION_USAGE_SUMMARY table from the previous Stat DB schema.
CREATE TABLE `ApiVersionPerAppAgg_<granularity>` (  
'AGG_TIMESTAMP' bigint(20) NOT NULL,  
'AGG_EVENT_TIMESTAMP' bigint(20) NOT NULL,  
'apiContext' varchar(254) NOT NULL,  
'apiHostname' varchar(254) NOT NULL,  
'applicationId' varchar(254) NOT NULL,  
'AGG_LAST_EVENT_TIMESTAMP' bigint(20) DEFAULT NULL,  
'apiName' varchar(254) DEFAULT NULL,  
'apiVersion' varchar(254) DEFAULT NULL,  
'apiCreator' varchar(254) DEFAULT NULL,  
'apiCreatorTenantDomain' varchar(254) DEFAULT NULL,  
'applicationName' varchar(254) DEFAULT NULL,  
'gatewayType' varchar(254) DEFAULT NULL,  
'label' varchar(254) DEFAULT NULL,  
'REGIONAL_ID' varchar(254) DEFAULT NULL,  
'AGG_COUNT' bigint(20) DEFAULT NULL,  
'AGG_SUM_quotaExceededValue' bigint(20) DEFAULT NULL,  
PRIMARY KEY (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`apiHostname`,`applicationId`)  
)

Click here for detailed descriptions on the ApiResPathPerApp Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiHostname</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>applicationId</td>
<td>ID of the client application which is used to invoke the API</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API Creator</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the client application used to invoke the API</td>
</tr>
<tr>
<td>gatewayType</td>
<td>Type of the gateway (Synapse/Micro)</td>
</tr>
<tr>
<td>label</td>
<td>The label of the API (if specified)</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occurred within the relevant time interval specified by AGG_TIMESTAMP based on granularity</td>
</tr>
<tr>
<td>AGG_SUM_quotaExceededValue</td>
<td>A binary value (0 or 1) to indicate whether the request is throttled(1) or not(0).</td>
</tr>
</tbody>
</table>

ApiExeTime

This aggregation contains information related to API invocation including timestamps and the time taken by the API at different stages of invocation such as service time, backend time etc. This is also derived from the request event stream.

ApiExeTime Table Schema
This schema resembles the API.EXE.TIME.<time>..SUMMARY table from the previous Stat DB schema.
CREATE TABLE `ApiExeTime_<granularity>` (  `AGG_TIMESTAMP` bigint(20) NOT NULL,  `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,  `apiContext` varchar(254) NOT NULL,  `apiHostname` varchar(254) NOT NULL,  `apiResourceTemplate` varchar(254) NOT NULL,  `apiMethod` varchar(254) NOT NULL,  `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,  `apiName` varchar(254) DEFAULT NULL,  `apiVersion` varchar(254) DEFAULT NULL,  `apiCreator` varchar(254) DEFAULT NULL,  `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,  `regionalID` varchar(254) DEFAULT NULL,  `AGG_SUM_responseTime` bigint(20) DEFAULT NULL,  `AGG_COUNT` bigint(20) DEFAULT NULL,  `AGG_SUM_serviceTime` bigint(20) DEFAULT NULL,  `AGG_SUM_backendTime` bigint(20) DEFAULT NULL,  `AGG_SUM_securityLatency` bigint(20) DEFAULT NULL,  `AGG_SUM_throttlingLatency` bigint(20) DEFAULT NULL,  `AGG_SUM_requestMedLat` bigint(20) DEFAULT NULL,  `AGG_SUM_responseMedLat` bigint(20) DEFAULT NULL,  `AGG_SUM_backendLatency` bigint(20) DEFAULT NULL,  `AGG_SUM_otherLatency` bigint(20) DEFAULT NULL,  PRIMARY KEY  (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`apiHostname`,`apiResourceTemplate`,`apiMethod`)  )

Click here for detailed descriptions on the ApiExeTime Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiHostname</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>apiResourceTemplate</td>
<td>API resource URL pattern of API request</td>
</tr>
<tr>
<td>apiMethod</td>
<td>HTTP Verb of API request [e.g.,GET/POST]</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_SUM_responseTime</td>
<td>Average time of the API response when received to the gateway</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occurred within the relevant time interval specified by AGG_TIMESTAMP based on granularity</td>
</tr>
<tr>
<td>AGG_SUM_serviceTime</td>
<td>Average time taken to serve the API request at the API-M server end</td>
</tr>
<tr>
<td>AGG_SUM_backendTime</td>
<td>Average time taken to process the request at the backend</td>
</tr>
<tr>
<td>AGG_SUM_securityLatency</td>
<td>Average time taken for authentication</td>
</tr>
<tr>
<td>AGG_SUM_throttlingLatency</td>
<td>Average time taken for throttling the request/response</td>
</tr>
<tr>
<td>AGG_SUM_requestMedLat</td>
<td>Average time taken to mediate the request</td>
</tr>
</tbody>
</table>
### AGG_SUM_aggregations

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_SUM_responseMedLat</td>
<td>Average time taken to mediate the response</td>
</tr>
<tr>
<td>AGG_SUM_backendLatency</td>
<td>Average time taken by the backend to return the response</td>
</tr>
<tr>
<td>AGG_SUM_otherLatency</td>
<td>Average time taken to process tasks other than those mentioned above</td>
</tr>
</tbody>
</table>

### ApiUserBrowserAgg

This aggregation contains information regarding the user browser summary derived from the request event stream.

### ApiUserBrowserAgg Table Schema

This schema resembles the API_REQ_USER_BROW_SUMMARY table from the previous Stat DB schema.

```sql
CREATE TABLE `ApiUserBrowserAgg_<granularity>` (
  `AGG_TIMESTAMP` bigint(20) NOT NULL,
  `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,
  `apiContext` varchar(254) NOT NULL,
  `apiCreator` varchar(254) NOT NULL,
  `apiCreatorTenantDomain` varchar(254) NOT NULL,
  `operatingSystem` varchar(254) NOT NULL,
  `browser` varchar(254) NOT NULL,
  `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,
  `apiName` varchar(254) DEFAULT NULL,
  `apiVersion` varchar(254) DEFAULT NULL,
  `gatewayType` varchar(254) DEFAULT NULL,
  `label` varchar(254) DEFAULT NULL,
  `regionalID` varchar(254) DEFAULT NULL,
  `AGG_COUNT` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`apiCreator`,`apiCreatorTenantDomain`,`operatingSystem`,`browser`) )
```

Click here for detailed descriptions on the ApiUserBrowserAgg Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API Creator</td>
</tr>
<tr>
<td>operatingSystem</td>
<td>Operating system</td>
</tr>
<tr>
<td>browser</td>
<td>Browser</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>gatewayType</td>
<td>Type of the gateway (Synapse/Micro)</td>
</tr>
<tr>
<td>label</td>
<td>The label of the API (if specified)</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occurred within the relevant time interval specified by AGG_TIMESTAMP based on granularity</td>
</tr>
</tbody>
</table>
**APIM_ReqCountAgg**

This aggregation is derived from the throttle event stream and the request event stream and provides information regarding the throttled out counts and success counts for a particular API.

**APIM_ReqCountAgg Table Schema**

```sql
CREATE TABLE `APIM_ReqCountAgg_<granularity>` (
    `AGG_TIMESTAMP` bigint(20) NOT NULL,
    `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,
    `apiName` varchar(254) NOT NULL,
    `apiVersion` varchar(254) NOT NULL,
    `apiCreator` varchar(254) NOT NULL,
    `apiCreatorTenantDomain` varchar(254) NOT NULL,
    `applicationName` varchar(254) NOT NULL,
    `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,
    `regionalID` varchar(254) DEFAULT NULL,
    `AGG_SUM_successCount` bigint(20) DEFAULT NULL,
    `AGG_SUM_throttleCount` bigint(20) DEFAULT NULL,
    PRIMARY KEY
    (`AGG_TIMESTAMP`, `AGG_EVENT_TIMESTAMP`, `apiName`, `apiVersion`, `apiCreator`, `apiCreatorTenantDomain`, `applicationName`) )
```

Click here for detailed descriptions on the APIM_ReqCountAgg Table Schema.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the client application used to invoke the API</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_SUM_successCount</td>
<td>Total successful API request count for the particular API</td>
</tr>
<tr>
<td>AGG_SUM_throttleCount</td>
<td>Total throttled out API request count for the particular API</td>
</tr>
</tbody>
</table>

**ApiLastAccessSummary**

This table is not an aggregation. It contains information regarding the last access times of the APIs and is derived from the request event stream.

**ApiLastAccessSummary Table Schema**

This schema resembles the API_LAST_ACCESS_TIME_SUMMARY table from the previous Stat DB schema.
CREATE TABLE `ApiLastAccessSummary` (  
  `apiContext` varchar(254) DEFAULT NULL,  
  `apiName` varchar(254) NOT NULL,  
  `apiVersion` varchar(254) DEFAULT NULL,  
  `apiCreator` varchar(254) NOT NULL,  
  `apiCreatorTenantDomain` varchar(254) NOT NULL,  
  `applicationOwner` varchar(254) DEFAULT NULL,  
  `lastAccessTime` bigint(20) DEFAULT NULL,  
  PRIMARY KEY (`apiName`, `apiCreator`, `apiCreatorTenantDomain`) )

GeoLocationAgg

This aggregation contains summarized information regarding the geolocation statistics of API requests and is derived from the request event stream.

GeoLocationAgg Table Schema

This schema resembles the `API_REQ_GEO_LOC_SUMMARY` table from the previous Stat DB schema.

CREATE TABLE `GeoLocationAgg_<granularity>` (  
  `AGG_TIMESTAMP` bigint(20) NOT NULL,  
  `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,  
  `apiContext` varchar(254) NOT NULL,  
  `country` varchar(254) NOT NULL,  
  `city` varchar(254) NOT NULL,  
  `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,  
  `apiName` varchar(254) DEFAULT NULL,  
  `apiVersion` varchar(254) DEFAULT NULL,  
  `apiCreator` varchar(254) DEFAULT NULL,  
  `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,  
  `username` varchar(254) DEFAULT NULL,  
  `userTenantDomain` varchar(254) DEFAULT NULL,  
  `totalCount` bigint(20) DEFAULT NULL,  
  `regionalID` varchar(254) DEFAULT NULL,  
  PRIMARY KEY (`AGG_TIMESTAMP`, `AGG_EVENT_TIMESTAMP`, `apiContext`, `country`, `city`) )

Field Name | Description
---|---
apiContext | API context depending on the user’s request
apiName | API Name
apiVersion | API Version
apiCreator | API Creator
apiCreatorTenantDomain | Tenant domain of the API creator
applicationOwner | Name of the owner of the client application
lastAccessTime | Timestamp of the latest API request occurrence

Field Name | Description
---|---
AGG_TIMESTAMP | The start timestamp of the time range based on the granularity of the table (e.g., the start timestamp of the hour in the table with the HOUR’S granularity).
AGG_EVENT_TIMESTAMP | The time at which the event is received at the event receiver
apiContext | API context depending on the user’s request
country | Country
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGG_TIMESTAMP</td>
<td>The start timestamp of the time range based on the granularity of the table. (e.g., start timestamp of the hour in the table with _HOURS granularity)</td>
</tr>
<tr>
<td>AGG_EVENT_TIMESTAMP</td>
<td>The time at which the event is received at the event receiver</td>
</tr>
<tr>
<td>apiContext</td>
<td>API context depending on the user's request</td>
</tr>
<tr>
<td>applicationId</td>
<td>ID of the client application</td>
</tr>
<tr>
<td>hostname</td>
<td>Hostname or Datacenter ID (if specified)</td>
</tr>
<tr>
<td>AGG_LAST_EVENT_TIMESTAMP</td>
<td>The timestamp at which the latest API request (event) belonging to the respective time range starting from when the AGG_TIMESTAMP was made</td>
</tr>
<tr>
<td>apiName</td>
<td>API Name</td>
</tr>
<tr>
<td>apiVersion</td>
<td>API Version</td>
</tr>
<tr>
<td>apiCreator</td>
<td>API Creator</td>
</tr>
</tbody>
</table>

**ApiFaultyInvocationAgg**

This aggregation contains summarized data of the faulty API invocations and is derived from the fault event stream.

**ApiFaultyInvocationAgg Table Schema**

This schema resembles the API_FAULT_SUMMARY table from the previous Stat DB schema.

```sql
CREATE TABLE `ApiFaultyInvocationAgg_<granularity>` (  
  `AGG_TIMESTAMP` bigint(20) NOT NULL,  
  `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,  
  `apiContext` varchar(254) NOT NULL,  
  `applicationId` varchar(254) NOT NULL,  
  `hostname` varchar(254) NOT NULL,  
  `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,  
  `apiName` varchar(254) DEFAULT NULL,  
  `apiVersion` varchar(254) DEFAULT NULL,  
  `apiCreator` varchar(254) DEFAULT NULL,  
  `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,  
  `applicationConsumerKey` varchar(254) DEFAULT NULL,  
  `applicationName` varchar(254) DEFAULT NULL,  
  `requestTimestamp` bigint(20) DEFAULT NULL,  
  `regionalID` varchar(254) DEFAULT NULL,  
  `AGG_COUNT` bigint(20) DEFAULT NULL,  
  PRIMARY KEY (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`applicationId`,`hostname`)  
)
```

[Click here for detailed descriptions on the ApiFaultyInvocationAgg Table Schema.]
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiCreatorTenantDomain</td>
<td>Tenant domain of the API creator</td>
</tr>
<tr>
<td>applicationConsumerKey</td>
<td>Consumer key of API invoking client application</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the client application used to invoke the API</td>
</tr>
<tr>
<td>requestTimestamp</td>
<td>Timestamp of the API request when received to the gateway</td>
</tr>
<tr>
<td>regionalID</td>
<td>The region ID if multi-data centers are configured for analytics</td>
</tr>
<tr>
<td>AGG_COUNT</td>
<td>The number of API requests that occurred within the relevant time interval specified by the AGG_TIMESTAMP based on granularity</td>
</tr>
</tbody>
</table>

**ApiThrottledOutAgg**

This aggregation contains summarized data of the throttled out API invocations. This information is derived from the throttle event stream.

**ApiThrottledOutAgg Table Schema**

This schema resembles the `API_THROTTLED_OUT_SUMMARY` table from the previous Stat DB schema.

```sql
CREATE TABLE `ApiThrottledOutAgg_<granularity>` (
    `AGG_TIMESTAMP` bigint(20) NOT NULL,
    `AGG_EVENT_TIMESTAMP` bigint(20) NOT NULL,
    `apiContext` varchar(254) NOT NULL,
    `applicationId` varchar(254) NOT NULL,
    `throttledOutReason` varchar(254) NOT NULL,
    `hostname` varchar(254) NOT NULL,
    `AGG_LAST_EVENT_TIMESTAMP` bigint(20) DEFAULT NULL,
    `apiName` varchar(254) DEFAULT NULL,
    `apiCreator` varchar(254) DEFAULT NULL,
    `apiCreatorTenantDomain` varchar(254) DEFAULT NULL,
    `username` varchar(254) DEFAULT NULL,
    `userTenantDomain` varchar(254) DEFAULT NULL,
    `applicationName` varchar(254) DEFAULT NULL,
    `subscriber` varchar(254) DEFAULT NULL,
    `gatewayType` varchar(254) DEFAULT NULL,
    `regionalID` varchar(254) DEFAULT NULL,
    `AGG_COUNT` bigint(20) DEFAULT NULL,
    PRIMARY KEY (`AGG_TIMESTAMP`,`AGG_EVENT_TIMESTAMP`,`apiContext`,`applicationId`,`throttledOutReason`,`hostname`)
) ;
```

Click here for detailed descriptions on the ApiThrottledOutAgg Table Schema.
API Manager statistics

API statistics are provided in both the API Publisher and the API Store. Apart from the number of subscriptions per API, all other statistical dashboards require an instance of WSO2 Data Analytics Server installed. For information on the available statistics and how to view them, see Viewing API Statistics.

Viewing API Statistics

API statistics are provided in both the API Publisher and the API Store. For instructions on how to set up Analytics, see Configuring APIM Analytics. Once Analytics is set up, follow the instructions below to view statistics through the API Publisher and API Store.

First, invoke a few APIs to generate traffic and see the statistics.

The following gadgets on the API Manager statistical dashboards, display real runtime statistics even when Analytics is not set up (as described in Configuring APIM Analytics).

- Created APIs Over Time
- Applications Created Over Time
- Developer Signups Over Time
- Subscriptions Created Over Time
- API subscriptions

The other gadgets you see on the API Manager statistical dashboards without setting up Analytics, are just samples and are not based on real runtime statistics of your server.

Please note that our data summarization logic summarizes the data on a per day basis.

The sections below explain how to access the statistical dashboards:

- API Publisher statistics
- API Store statistics
- Admin Portal Statistics

API Publisher statistics

The gadgets that display publisher statistics can only be viewed via users that have permission to create APIs. For more information, see Managing Users and Roles.

Log in to the API Publisher. Anyone who can create and/or publish APIs can view the API-level usage and subscription statistics by clicking on a selected API and referring to its Versions and Users tabs.
PizzaShackAPI - 1.0.0

API Subscriptions by Versions

<table>
<thead>
<tr>
<th>Version</th>
<th>Number of API calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0</td>
<td>4</td>
</tr>
</tbody>
</table>

API Usage by Versions

<table>
<thead>
<tr>
<th>Version</th>
<th>Number of API calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0</td>
<td>325</td>
</tr>
</tbody>
</table>
Given below are the statistical dashboards that are available from the Analytics menu.
In each of the dashboards, you can choose to view all APIs or if you are an API creator, only the APIs that you have created. You can also select the time period for which you wish to view the statistics.

Several examples of usage and performance statistics are given below:

- Created APIs Over Time
- API Usage
- API Last Access Times
- Usage by Resource Path
- Usage by Destination
- API Usage Comparison
- API Throttled Requests
- Faulty Invocations
- API Latency Time
- API Usage Across Geo Locations
- API Usage Across User Agent
- App Throttled Requests
- Applications Created Over Time
- API Subscriptions
- Developer Signups Over Time
- Subscriptions Created Over Time
- API Usage per Application
- Top Users per Application
- Resource Usage per Application
- Faulty Invocations per Application
- Availability of APIs

Created APIs Over Time

The number of APIs published over a given time period.

API Usage

The number of subscriptions of each API with a graphical view of the count.
API Last Access Times

A tabular representation of the last access times of an API, according to the version and the accessed subscriber.

API Last Access Times (Across All Versions)

<table>
<thead>
<tr>
<th>API</th>
<th>Version</th>
<th>Subscriber</th>
<th>Access Time (GMT+05:30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppAPI (admin)</td>
<td>1.0.0</td>
<td><a href="mailto:admin@carbon.super">admin@carbon.super</a></td>
<td>June 8, 2015 2:48:00 PM GMT+05:30</td>
</tr>
<tr>
<td>PizzaShackAPI (admin)</td>
<td>1.0.0</td>
<td><a href="mailto:sub1@carbon.super">sub1@carbon.super</a></td>
<td>June 8, 2015 2:31:00 PM GMT+05:30</td>
</tr>
<tr>
<td>MyAPI (admin)</td>
<td>1.0.0</td>
<td><a href="mailto:admin@carbon.super">admin@carbon.super</a></td>
<td>June 8, 2015 2:24:00 PM GMT+05:30</td>
</tr>
<tr>
<td>PetAPI (admin)</td>
<td>1.0.0</td>
<td><a href="mailto:admin@carbon.super">admin@carbon.super</a></td>
<td>June 8, 2015 12:44:00 PM GMT+05:30</td>
</tr>
</tbody>
</table>

Usage by Resource Path

The number of invocations made by resources for an API, represented in a tabular view.
## API Usage by Resource Path

<table>
<thead>
<tr>
<th>api</th>
<th>version</th>
<th>resourcePath</th>
<th>method</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap1</td>
<td>v1.0</td>
<td>/</td>
<td>GET</td>
<td>1</td>
</tr>
<tr>
<td>ap2</td>
<td>v1.0</td>
<td>/api</td>
<td>POST</td>
<td>1</td>
</tr>
<tr>
<td>mockapi</td>
<td>v1.0</td>
<td>/api/orders</td>
<td>GET</td>
<td>3</td>
</tr>
<tr>
<td>pizza</td>
<td>v1.0</td>
<td>/menus</td>
<td>GET</td>
<td>1</td>
</tr>
</tbody>
</table>

## Usage by Destination

The number of times an API was accessed based on the destination, represented in a tabular view.

### API Usage by Destination

<table>
<thead>
<tr>
<th>API</th>
<th>VERSION</th>
<th>CONTEXT</th>
<th>DESTINATION ADDRESS</th>
<th>NO OF ACCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyAPI</td>
<td>1.0.0</td>
<td>/myAPI/1.0.0</td>
<td><a href="https://localhost:5443/am/sample/pizzashack/v1/api/">https://localhost:5443/am/sample/pizzashack/v1/api/</a></td>
<td>9</td>
</tr>
</tbody>
</table>

## API Usage Comparison

The number of invocations made for an API represented as a combination of all resources and all versions of the API.
## API Usage Comparison (Across All Versions)

The API Usage Comparison tool allows you to compare API usage across different versions. You can filter by specific APIs and view the usage information over different time periods (hour, day, week, month).

### Filter by...

- All APIs

### API Usage Table

<table>
<thead>
<tr>
<th>API</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PizzaShackAPI</td>
<td>68</td>
</tr>
<tr>
<td>MyAPI</td>
<td>16</td>
</tr>
<tr>
<td>PetAPI</td>
<td>10</td>
</tr>
<tr>
<td>AppAPI</td>
<td>1</td>
</tr>
</tbody>
</table>

**API Throttled Requests**

The total count of the successful request count and throttled request count towards an API over time.

**Filter by ...**

- All APIs

**API Usage Table**

<table>
<thead>
<tr>
<th>API</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PizzaShackAPI</td>
<td>68</td>
</tr>
<tr>
<td>MyAPI</td>
<td>16</td>
</tr>
<tr>
<td>PetAPI</td>
<td>10</td>
</tr>
<tr>
<td>AppAPI</td>
<td>1</td>
</tr>
</tbody>
</table>

**API Throttled Requests**

The total count of the successful request count and throttled request count towards an API over time.
Faulty Invocations

A successful invocation is when an API receives the expected response. If it results in any kind of error response, that invocation is considered a faulty invocation. The total number of invocations of an API represented as a combination of successful and faulty invocations is denoted in the faulty invocations graph.

Faulty Invocations

API Latency Time

The execution time of the APIs represented as a combination of throttling, In mediation, Out mediation, backend response time, and authentication time.

You can also see a comparison view of the latencies.
API Usage Across Geo Locations

The data script that updates statistics related to geo locations is executed once a day. Therefore, at a given time, some of the statistics generated within the last 24 hours may not be displayed in this gadget.
API Usage Across Geo Locations

API Usage Across User Agent

The proportional distribution of the usage (invocation) of each API differentiated by the user agent HTTP header received in requests towards the API.

API Usage Across Usage Agent

App Throttled Requests

The successful request count and throttled request count of each API invoked by each application.
Application Throttled Out Requests

Applications Created Over Time

The number of applications created over a given time period.
App Registrations

API Subscriptions

The subscriptions created for each of the APIs as a distribution of API versions.

Overall API Subscriptions (Across All Versions)

Developer Signups Over Time

The number of developers who signed up to the API Store over time.
Subscriptions Created Over Time

The number of subscriptions created for an API over a given period.

First, select the API for which you wish to view subscriptions.
API Subscription per app

API Store statistics

Log in to the API Store. You can self-subscribe to the store. Next, click the Statistics menu.

Given below are the statistical dashboards that are available:

- **API Usage**: The usage of the API per application.
- **Top Users**: Users who make the most API invocations per application and the number of registered users per application.
- **Resource Usage**: The usage of an API and from which resource path per application.
- **Faulty Invocations**: The number of faulty API invocations per application.

In a faulty API invocation, the message is mediated through the fault sequence. By default, the API Manager considers an API invocation to be faulty when the backend service is unavailable.
Several examples of usage and performance statistics are given below:

**API Usage per Application**

The number of invocations made for each API by each application.

**Top Users per Application**

The users who made the largest number of API calls per application.
Resource Usage per Application

The usage of resources of the APIs by each application.

The statistics for the registered users for applications takes the number of users shared across each application into consideration. To enable application sharing among users within the same organization, see Sharing Applications Between Multiple Groups.

Only users who have generated access tokens using the password grant type are considered in these statistics.
Faulty Invocations per Application

The total number of invocations made by each application that are unsuccessful (faulty).

Faulty Invocations per Application

Admin Portal Statistics

Log in to the Admin Portal (https://localhost:9443/admin). API Availability is the only statistical view that exists in the Admin Portal. Admin users can view API Availability statistics by navigating to ANALYTICS > API AVAILABILITY.

Availability of APIs

The status of the APIs (all API versions) represented in a tabular view.
Status

This indicates the status of the API. There are two possible values; **Available** and **Limited**.

**Available** - This status indicates that the API has traffic with normal successful invocations. By default, if an API receives successful invocations for at least one out of five invocations within 30000 milliseconds, the status of the API becomes **Available**.

- Note that only the APIs that have traffic are represented in this tabular representation.

**Limited** - If an API receives an alert due to one of the reasons indicated in Availability of APIs (health monitoring), the API status changes to **Limited**.

For more information on how to view the generated alerts, see Viewing Alerts.

The availability of APIs statistics is directly related to the Availability of APIs (health monitoring) alert type. You can edit the default configurations of the numbers set as parameters to customize generating alerts by navigating to SETTINGS > ANALYTICS and by going to the Edit view of HealthAvailabilityPerMinAlert as shown below.

### Edit Configurations

<table>
<thead>
<tr>
<th>Template Name</th>
<th>HealthAvailabilityPerMinAlert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitors the API Availability</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration Name</th>
<th>HealthAvailabilityPerMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Monitoring API health</td>
</tr>
</tbody>
</table>

### Parameter Configurations

- **Number of continuous responses**
  - 5
  - Number of responses that should fail to pass the lower percentile

- **Number of continuous response time failures**
  - 5
  - number of minutes that responses should fail to pass the lower percentile

- **Time interval (in milliseconds) for API availability status change**
  - 300000
  - Time duration taken to recheck and change the availability of an API

- **Cache Time-out**
  - 300
  - Cache time-out value in seconds

- **Severity Level**
  - 2
  - Severity level of the alert: (1:severe, 2:moderate, 3:mild)

Save Configuration

The parameter configurations of this alert type are given below.
<table>
<thead>
<tr>
<th><strong>Number of continuous responses</strong></th>
<th>Considering the aspect of generating the alerts, number of responses that should fail to pass the lower percentile. The default value is 5 responses.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of continuous response time failures</strong></td>
<td>The number of minutes that responses should fail to pass the lower percentile for the alert to be generated. The default value is 5 minutes.</td>
</tr>
<tr>
<td><strong>Time interval</strong></td>
<td>The time duration taken to recheck and change the availability of an API. The availability of the API is rechecked after this time interval and the new status is retrieved if the behaviour changes. The default value is 300 seconds (300000 milliseconds).</td>
</tr>
<tr>
<td><strong>Cache time-out</strong></td>
<td>The cache time-out value in seconds.</td>
</tr>
<tr>
<td><strong>Severity level</strong></td>
<td>The severity level of the alert, which you can specify as 1, 2 or 3 where 1 is severe, 2 is moderate and 3 is mild.</td>
</tr>
</tbody>
</table>

### Using Geolocation Based Statistics

Geolocation based statistics are used to carry out detailed monitoring of geographic locations. The following sections explain how WSO2 API Manager is integrated with WSO2 Analytics - APIM to create alerts based on statistics generated for selected geographic locations.

- Configuring Geolocation Based Statistics
- Writing a Custom Geolocation Provider
- Updating Geo Location Data Set

#### Configuring Geolocation Based Statistics

In order to generate Geolocation based statistics, you need to pass an `x-forwarded-for` header with the relevant IP in the API request.

Note that this documentation uses MySQL as an example for configuring the GEO_LOCATION_DATA database.

1. Download the Geolocation data from [here](#).
2. Unzip the file you downloaded.

   If you have Geo Location Dataset already downloaded, you can update that dataset by following [Updating Geo Location Data Set](#).

3. Create the GEO_LOCATION_DATA database by executing one of the scripts in the Geolocation Data/dbscripts directory. In this example, `mysql.sql` is executed.

   This can be done using the **MySQL Workbench**.

   For detailed instructions to run the database script, see [MySQL Documentation - The Workbench Scripting Shell](#).

4. Restore data to the BLOCKS and LOCATION tables by importing data from the BLOCKS.csv and LOCATION.csv from the .Geolocation Data/data directory of the extracted zip using the commands given below.

   * **Importing Geolocation Data/data/LOCATION .csv**
     ```
     mysqlimport -u root -p --ignore-lines=2 --fields-terminated-by=, --fields-optionally-enclosed-by="" --local GEO_LOCATION_DATA <path_to_folder_location>/GeolocationData/data/LOCATION.csv
     ```

   * **Importing Geolocation Data/data/BLOCKS.csv**
     ```
     mysqlimport -u root -p --ignore-lines=2 --fields-terminated-by=, --fields-optionally-enclosed-by="" --local GEO_LOCATION_DATA <Extracted_location>/GeolocationData/data/BLOCKS.csv
     ```

   For more information, see [MySQL Documentation - Data Export and Import](#).

5. Check whether your imported dataset is working properly by executing following query in the MySQL Command Line.

   ```
   SELECT loc.country_name, loc.subdivision_1_name FROM BLOCKS block, LOCATION loc WHERE block.network_blocks = '<Network_part_of_ipv4>' AND <Long_value_of_public_IP> BETWEEN block.network AND block.broadcast AND block.geoname_id=loc.geoname_id;
   ```

   **Example query:**
SELECT loc.country_name, loc.subdivision_1_name FROM BLOCKS block, LOCATION loc WHERE block.network_blocks = '221.192' AND 3720398641 BETWEEN block.network AND block.broadcast AND block.geoname_id=loc.geoname_id;

6. Download a JDBC provider depending on the database you are using (MySQL, in this example), and copy it to the `<APIM_ANALYTICS_HOME>/lib` directory.

7. Configure the datasource for the Geo location.
   A default datasource for Geo location is packed by default in the `<APIM_ANALYTICS_HOME>/conf/worker/deployment.yaml` file under data sources.
   You can edit the following data source to point to his own db and import the geo location data.

   ```yaml
   - name: GEO_LOCATION_DATA
     description: "The data source used for geo location database"
     jndiConfig:
       name: jdbc/GEO_LOCATION_DATA
       definition:
         type: RDBMS
         configuration:
           jdbcUrl: 'jdbc:h2:${sys:carbon.home}/wso2/worker/database/GEO_LOCATION_DATA;AUTO_SERVER=TRUE'
           username: wso2carbon
           password: wso2carbon
           driverClassName: org.h2.Driver
           maxPoolSize: 50
           idleTimeout: 60000
           validationTimeout: 30000
           isAutoCommit: false
   ```

For information on how to configure alerts, see Configuring Alerts.

Writing a Custom Geolocation Provider

Each Geolocation Resolver implementation in WSO2 Analytics is inherited from the `org.wso2.carbon.analytics.apim.spark.geolocation.api.LocationResolver` abstract class has the following methods.

- `getLocation`: This contains the Geolocation Resolving implementation. Only this method needs to be implemented for this scenario.
- `init`: This contains the Geolocation Resolver implementation

To customize the default Geolocation Resolver extension, you should override the `getLocation()` method with your custom implementation. For example, the following class is a sample implementation of the Geolocation Resolving service It returns the Location according to the IP of the Geolocation API that provided through the configuration on each IP resolving through the UDF.

```java
package com.wso2.carbon.analytics.apim.spark.geolocation.impl;

import org.wso2.carbon.analytics.apim.spark.geolocation.api.Location;
import org.wso2.carbon.analytics.apim.spark.geolocation.api.LocationResolver;
import org.wso2.carbon.analytics.apim.spark.geolocation.exception.GeolocationResolverException;

import org.wso2.carbon.analytics.apim.spark.geolocation.api.Location;  
```
public class CustomLocationResolver extends LocationResolver {
    private String restUrl;
    private String username;
    private String password;

    @Override
    public void init() throws GeoLocationResolverException {
    }

    public Location getLocation(String ip) throws GeoLocationResolverException {
        return null;
    }

    public String getRestUrl() {
        return restUrl;
    }

    public void setRestUrl(String restUrl) {
        this.restUrl = restUrl;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

Compile as a jar file and copy into <Product-home>/repository/components/lib folder.
Configure the above class in geolocation.xml under <Product-home>/repository/conf/etc as following way.

<GeoLocation>
    <Implementation
class="com.wso2.carbon.analytics.apim.spark.geolocation.impl.CustomLocationResolver">
        <Property name="restUrl">http://localhost:80080/geolocation/service</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
    </Implementation>
    <Cache>
        <enabled>true</enabled>
    </Cache>
</GeoLocation>
Updating Geo Location Data Set

Follow the procedure below in order to update your existing Geo Location dataset to use in Configuring Geolocation Based Statistics.

1. Download the latest CSV file from [here](https://github.com/maxmind/geoip2-csv-converter/releases).
2. Prepare the database entries.
   1. Unzip the latest CSV file and the geoip-2-csv-converter you have downloaded in the steps above.
   2. Run `update-geolocation-data.sh` file using the command below.

   ```bash
   sh update-geolocation-data.sh
   ```

   - Enter the path to the extracted GeoLite2-City-Blocks-IPv4 directory which you downloaded first, as the response for `Enter path to GeoLite2-City-Blocks-IPv4 directory`:
     ```
     E.g.: /PATH_TO/GeoLite2-City-CSV_20171107
     ```
   - Enter the path to geoip2-csv-converter directory as the response for `Enter path to geoip2-csv-converter home directory`:
     ```
     E.g.: /PATH_TO/geoip2-csv-converter-v1.0.0
     ```
   - After executing the script, you will find the `final.csv` file inside your current directory.

     ```
     ** get first column form original
     ** change column name to 'network_cidr'
     ** Extract ip address data
     ** change column name to 'network_blocks'
     ** extract entries from original cut:
     /home/chamalee/xxx/geoip2-csv-converter-v1.0.0/GeoLite2-City-Blocks-IPv4-converted.csv:
     No such file or directory
     ** change column name to 'network'
     ** change column name to 'broadcast'
     ** merge csv files
     ```

3. Shut down both APIM and APIM-Analytics servers if you are running them already.
4. Truncate BLOCKS and LOCATION tables from the GEO_LOCATION_DATA database.

Alternatively you can drop the tables in the GEO_LOCATION_DATA database and create new tables.

Importing Data

1. Import the created `final.csv` file into BLOCKS table. Use the command given below.

   ```sql
   load data local infile '{PATH_TO_FINAL.CSV}/final.csv' into table BLOCKS
elements terminated by ','
enclosed by ''
lines terminated by '\n'
(network_cidr, network, broadcast, geoname_id, registered_country_geoname_id,
represented_country_geoname_id, is_anonymous_proxy, is_satellite_provider, postal_code,
latitude, longitude, network_blocks);
   ```

2. Import the `GeoLite2-City-Locations-en.csv` file located inside the extracted geoip-2-csv-converter directory (e.g. `geoip-2-csv-converterGeoLite2-City-CSV_2017110`) into LOCATION table. Use the command given below.

   ```sql
   ```
3. Restart WSO2 API Manager and WSO2 APIM-Analytics servers.

You have now updated the Geo Location Data Set.

Managing Alerts

WSO2 API Manager uses WSO2 Analytics to create alerts for different purposes. Common scenarios that require alerts include the sudden failure of one or more APIs, a sudden increase in the response time of one or more APIs and the change in the pattern of API resource access.

The following topics explain the different alert types that can be created and how to configure WSO2 API Manager to use them.

- Alert Types
- Configuring Alerts
- Subscribing for Alerts
- Viewing Alerts

Alert Types

WSO2 APIM currently supports the following alert types.

- Abnormal response time
- Abnormal backend time
- Abnormal request counts
- Abnormal resource access pattern
- Unseen source IP address
- Frequent tier limit hitting (tier crossing)
- Availability of APIs (API health monitoring)

Abnormal response time

<table>
<thead>
<tr>
<th>Reason for triggering</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If there is a sudden increase in the response time of a specific API resource.</td>
<td></td>
</tr>
</tbody>
</table>

| Indication | Slow WSO2 API Manager runtime, or slow backend.                                               |

| Description | This alert gets triggered if the response time of a particular API is higher than the configured value. These alerts could be treated as an indication of a slow WSO2 API Manager runtime or a slow backend. |

Abnormal backend time

<table>
<thead>
<tr>
<th>Reason for triggering</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If there is a sudden increase in the backend time corresponding to a particular API resource.</td>
<td></td>
</tr>
</tbody>
</table>

| Indication | Slow backend                                                                                 |

| Description | This alert gets triggered if the backend time corresponding to a particular API is higher than the configured value. These alerts could be treated as an indication of a slow backend. |

Abnormal request counts


### Abnormal resource access pattern

<table>
<thead>
<tr>
<th>Reason for triggering</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If there is a change in the resource access pattern of a user who uses a particular application.</td>
<td>A Markov Chain model is built for each application to learn its resource access pattern. For the purpose of learning the resource access patterns, no alerts are sent during the first 500 (default) requests. After learning the normal pattern of a specific application, WSO2 Analytics performs a real-time check on a transition done by a specific user, and sends an alert if it is identified as an abnormal transition. For a transition to be considered valid, it has to occur within 60 minutes by default, and it should be by the same user.</td>
</tr>
</tbody>
</table>

The above diagram depicts an example where a Markov Chain model is created during the learning curve of the system. Two states are recorded against Application A and the arrows show the directions of the transitions. Each arrow carries a probability value that stands for the probability of a specific transition taking place. Assume that the following two consecutive events are received by the application from user john@abc.com.

1. DELETE /API1/number/1
2. DELETE /API1/number/3

The above transition has happened from the DELETE /API1/number/{x} state to itself. According to the Markov chain model learnt by the system, the probability of this transition occurring is very low. Therefore, an alert is sent.

This alert is triggered if there is a change in the resource access pattern of a user of a particular application. These alerts could be treated as an indication of a suspicious activity made by a user over your application.

### Unseen source IP address

<table>
<thead>
<tr>
<th>Reason for Triggering</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If there is either a change in the request source IP for a specific API of an application, or if the request if from an IP used before 30 days (default).</td>
<td>If there is a sudden spike or a drop in the request count within a period of one minute by default for a particular API resource. These alerts could be treated as an indication of possible high traffic, suspicious activity, possible malfunction of the client application, etc.</td>
</tr>
</tbody>
</table>

This alert is triggered if there is a sudden spike in the request count within a period of one minute by default for a particular API for an application. These alerts could be treated as an indication of a possible high traffic, suspicious activity etc.
### Indication

These alerts could be treated as an indication of a suspicious activity made by a user over an application.

### Description

This alert is triggered if there is either a change in the request source IP for a particular application by a user or if the request is from an IP used before a time period of 30 days (default). The first 100 requests by a user are used only for learning purposes by default and therefore, no alerts are sent during that time. However, the learning would continue even after the first 100 requests by a user. This means, even if you receive continuous requests from the newly detected IP, you are alerted only once.

### Frequent tier limit hitting (tier crossing)

<table>
<thead>
<tr>
<th>Reason for Triggering</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reason for Triggering</strong></td>
<td>These alerts indicate that you need to subscribe to a higher tier.</td>
</tr>
</tbody>
</table>

### Availability of APIs (API health monitoring)

These alerts are triggered for the reasons specified in the tables below.

<table>
<thead>
<tr>
<th>Reason for Triggering</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>The response time of an API is greater than the configured value specified for the same. This should occur continuously for a specified number of times (5 times by default).</td>
<td>The response time is too high.</td>
</tr>
<tr>
<td>The response status code is greater than or equal to 500, but less than 600. This should occur continuously for a specified number of times (5 times by default) in order to trigger an alert.</td>
<td>A server side error has occurred.</td>
</tr>
</tbody>
</table>

For more information on API status changing over availability of APIs, see Viewing Availability Of APIs.
Configuring Alerts

WSO2 API Manager Analytics allows you to customize and configure the analytics-based alerts sent to the subscribers using the features listed below.

- Edit alerts as business rules
- Configuring alerts via the Publisher
- Configuring alerts via the Store

Before you begin...

Make sure that you have configured Analytics for API Manager. For instructions, see Configuring APIM Analytics.

Edit alerts as business rules

1. Go to the `<API-M_ANALYTICS_HOME>/ bin` directory and run the following command to start the dashboard, if you have not started it already.

   - On Windows
     
     ```bash
     dashboard.bat --run
     ```
   - On Linux
     
     ```bash
     sh dashboard.sh
     ```

2. Go to the Business Rules and Status Dashboard. (e.g., `https://localhost:9643/business-rules`)
3. You can view the existing business rules that are applied for API Manager. Depending on your privileges, you can view, edit, and delete business rules.

   For more details on working with business rules, see Managing Business Rules.

   ![Business Rules Dashboard](image)

Configuring alerts via the Publisher

The users of your APIs can subscribe to analytics-related alerts from the API Publisher and the API Store. Follow the instructions below to manage alerts via the Publisher:

You can not disable Health Availability related alerts, because they are enabled by default. However, you can enable and disable the email alerts that correspond to the Health Availability alerts.

- Creating an abnormal response time alert
- Updating an abnormal response time alert
- Enabling an abnormal backend time alert
Updating an abnormal backend time alert
Subscribing to emails based on Publisher alerts
Unsubscribing from emails based on Publisher alerts

Creating an abnormal response time alert

1. Start the WSO2 API-M Analytics and WSO2 API-M Servers.
   For more information see, Configuring APIM Analytics.
2. Sign in to the API Publisher. https://localhost:9443/publisher
3. Click Manage Alert Types.

The Manage Alert Types page appears.

Manage Alert Types

<table>
<thead>
<tr>
<th>Alert Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abnormal Response Time</td>
<td>This alert gets triggered if the response time of a particular API is higher than the predefined value. These alerts could be treated as an indication of a slow WSO2 API Manager runtime or a slow backend.</td>
</tr>
<tr>
<td>Abnormal Backend Time</td>
<td>This alert gets triggered if the backend time corresponding to a particular API is higher than the predefined value. These alerts could be treated as an indication of a slow backend. In technical terms, if the backend time of a particular API of a tenant lies outside the predefined value, an alert will be sent out.</td>
</tr>
</tbody>
</table>

- Health Availability
  - This alert gets triggered if at least one of the two cases below are satisfied: Response time of an API > Threshold response time value defined for that particular API or Response status code >= 500 (By Default) AND Response status code < 500 (By Default)

Email: Type an email address and press Enter

4. Click on the setting option that corresponds to the Abnormal Response Time option.

5. Select the API for which you need to set up the alerts.
6. Define the time period (in milliseconds).
   Immediately after the response period of the API exceeds the above defined time period an alert gets triggered, such alerts could be treated as an indication of a slow WSO2 API Manager runtime or a slow backend.
Configure 'Abnormal Response Time' Alert

New Alert

7. Click Save.

**Updating an abnormal response time alert**

1. Start the WSO2 API-M Analytics and WSO2 API-M Servers. For more information see, Configuring APIM Analytics.
2. Sign in to the API Publisher.
   https://localhost:9443/publisher
3. Click Manage Alert Types.
4. Click on the setting option that corresponds to the Abnormal Response Time option.
5. Update the alert.
   - **To edit a abnormal response time alert** -
     a. Select the API that corresponds to the existing alerts.
     b. Define the new time period (in milliseconds).
     c. Click Save. This will update the existing alert details.
   - **To delete a abnormal response time alert**, click the Remove option that corresponds to the respective API.

**Enabling an abnormal backend time alert**
1. Start the WSO2 API-M Analytics and WSO2 API-M Servers. For more information see, Configuring APIM Analytics.
2. Sign in to the API Publisher. https://localhost:9443/publisher
3. Click Manage Alert Types.

![Manage Alert Types](image)

4. Click on the setting option that corresponds to the Abnormal Backend Time option.

5. Select the API for which you need to set up the alerts.
6. Define the time period (in milliseconds).
    Immediately after the backend time of the API exceeds the above defined time period an alert gets triggered, such alerts could be treated as an indication of a slow backend. In technical terms, if the backend time of a particular API of a tenant lies outside the predefined value, an alert is sent.

**Configure 'Abnormal Backend Time' Alert**

![Configure Abnormal Backend Time Alert](image)

**7. Click Save.**

*Updating an abnormal backend time alert*

1. Start the WSO2 API-M Analytics and WSO2 API-M Servers. For more information see, Configuring APIM Analytics.
2. Sign in to the API Publisher. https://localhost:9443/publisher
3. Click Manage Alert Types.

![Manage Alert Types](image)

4. Click on the setting option that corresponds to the Abnormal Backend Time option.
5. Update the alert.
   - **To edit an existing abnormal backend time alert**: a. Select the API that corresponds to the existing alert. b. Define the new time period (in milliseconds). c. Click **Save**. This will update the existing alert details.
   - **To delete an abnormal backend time alert**, click the **Remove** option that corresponds to the respective API.

### Alert Configurations

<table>
<thead>
<tr>
<th>API Name</th>
<th>Version</th>
<th>Time (ms)</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PizzaShopAPI</td>
<td>1.0.0</td>
<td>2000</td>
<td><strong>Remove</strong></td>
</tr>
</tbody>
</table>

![Alert Configurations Table]

### Subscribing to emails based on Publisher alerts

1. Configure the sender's details for the email alerts.

   Follow the instructions below to configure an email address with API Manager to send email alerts to subscribers.
   a. Open the `<API-M_ANALYTICS_HOME>/conf/worker/deployment.yaml` file.
   b. Configure the sender email address. The sample code is shown below.

   ```yaml
   siddhi:
   extensions:
   ...
   - extension:
     name: email
     namespace: sink
     properties:
       username: alex@gmail.com
       address: alex@gmail.com
       password: password
   ...
   ```

   Note that you might have to bypass a security warning to configure this with a private email address.

   c. Go to the `<API-M_ANALYTICS_HOME>/resources/apim-analytics/` directory. Copy the `APIM_ALERT_EMAIL_NOTIFICATION.siddhi` file and paste it in the `<API-M_ANALYTICS_HOME>/wso2/worker/deployment/siddhi-files` directory.
   d. Restart the WSO2 API-M Analytics and WSO2 API-M Servers.

2. Sign in to the API Publisher.
   [https://localhost:9443/publisher](https://localhost:9443/publisher)

3. Click **Manage Alert Types**.

4. Select the alert options that you wish to start receiving emails on.
5. Click Save.

Unsubscribing from emails based on Publisher alerts

1. Start the WSO2 API-M Analytics and WSO2 API-M Servers.
   For more information see, Configuring APIM Analytics.
2. Sign in to the API Publisher.
   https://localhost:9443/publisher
3. Click Manage Alert Types.

   ![Image of Manage Alert Types]

4. Deselect the alert options that you wish to stop receiving emails on.
5. Click Unsubscribe.

Configuring alerts via the Store

The users of your APIs can subscribe to analytics-related alerts from the API Publisher and the API Store. Follow the instructions below to manage alert types via the Store:

You can not disable Abnormal Resource Access Alerts, Unseen Source IP Access Alerts and Tier Crossing Alerts, because they are enabled by default. However, you can enable and disable the email alerts that correspond to the latter mentioned alerts.

- Creating an abnormal request count alert
- Updating an abnormal request count alert
- Subscribing to emails based on Store alerts
- Unsubscribing from emails based on Store alerts

Creating an abnormal request count alert

1. Start the WSO2 API-M Analytics and WSO2 API-M Servers.
   For more information see, Configuring APIM Analytics.
2. Sign in to the API Store.
   https://localhost:9443/store
3. Click Manage Alert.
The Manage Alerts page appears.

4. Click on the setting option that corresponds to the Abnormal Request Count option.

5. Select the application for which you need to set up the alerts.
6. Select the API for which you need to set up the alerts.

Only the APIs that you have subscribed to using the latter mentioned application are displayed.

7. Define the request count (per minute).

Immediately after the request count of the API exceeds the above defined time period an alert gets triggered. These alerts could be treated as indications of possible high traffic, suspicious activity, possible malfunction of the client application etc.
Configure 'Abnormal Request Count' Alert

New Alert

Application Name * DefaultApplication
API * PizzaShackAPI-1.0.0
Request Count * 8000

Save

8. Click **Save**.

**Updating an abnormal request count alert**

1. Start the WSO2 API-M Analytics and WSO2 API-M Servers.
   For more information see, Configuring APIM Analytics.
2. Sign in to the API Store.
   https://localhost:9443/store
3. Click **Manage Alert**.

4. Click on the setting option that corresponds to the **Abnormal Request Count** option.

5. Update the alert.
   - **To edit an abnormal response count alert** -
     a. Select the application that corresponds to the existing alert.
     b. Select the API that corresponds to the existing alert.
     c. Define the new request count (per minute).
     d. Click **Save**.
     This will update the existing alert details.
   - **To delete an abnormal response count alert**, click the **Remove** option that corresponds to the respective API.
Alert Configurations

<table>
<thead>
<tr>
<th>Application Name</th>
<th>API Name</th>
<th>API Version</th>
<th>Request Count</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Application</td>
<td>PizzaHackAPI</td>
<td>1.0.0</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>Default Application</td>
<td>Test</td>
<td>1.0.0</td>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

Subscribing to emails based on Store alerts

1. Configure the sender's details for the email alerts.
   
   Follow the instructions below to configure an email address with API Manager to send email alerts to subscribers.
   
   a. Open the `<API-M_ANALYTICS_HOME>/conf/worker/deployment.yaml` file.
   
   b. Configure the sender email address. The sample code is shown below.

   ```yaml
   siddhi:
     extensions:
       ...
     -
       extension:
         name: email
         namespace: sink
         properties:
           username: alex@gmail.com
           address: alex@gmail.com
           password: password
       ...
   ```

   Note that you might have to bypass a security warning to configure this with a private email address.

   c. Go to the `<API-M_ANALYTICS_HOME>/resources/apim-analytics/` directory. Copy the `APIM_ALERT_EMAIL_NOTIFICATION.siddhi` file and paste it in the `<API-M_ANALYTICS_HOME>/wso2/worker/deployment/siddhi-files` directory.
   
   d. Restart the WSO2 API-M Analytics and WSO2 API-M Servers.

   For more information see, Configuring APIM Analytics.

2. Start the WSO2 API-M Analytics and WSO2 API-M Servers.
   
   For more information see, Configuring APIM Analytics.

3. Sign in to the API Store.
   
   https://localhost:9443/store

4. Click Manage Alert.

5. Select the alert options that you wish to start receiving emails on.

6. Click Save.

Unsubscribing from emails based on Store alerts

Start the WSO2 API-M Analytics and WSO2 API-M Servers.

For more information see, Configuring APIM Analytics.

https://localhost:9443/store

Click Manage Alert.

5. Select the alert options that you wish to stop receiving emails on.

6. Click Save.
1. Start the WSO2 API-M Analytics and WSO2 API-M Servers.
   For more information see, Configuring APIM Analytics.
2. Sign in to the API Store:
   https://localhost:9443/store
3. Click Manage Alert.

   ![WSO2 API Store](image)

4. Deselect the alert options that you wish to stop receiving emails on.
5. Click Unsubscribe.

   ![Manage Alerts](image)

Subscribing for Alerts

You can subscribe to events as a system administrator or as a API publisher/subscriber. These users can subscribe to any of the alert types listed in the Manage Alert Types page specific to them. For more information about different types of alerts and their importance, see Alert Types.

Click on the relevant tab to view the required procedure.

- System Administrators
- APIM Publisher/Subscriber

A system administrator is allowed to select one or more alert types to subscribe for, as well as specify a list of email addresses to which the alerts should be sent. Follow the procedure below to carry out the tasks mentioned above using the Admin Portal of WSO2 API Manager.

1. Log into the WSO2 API Manager Admin Portal using the URL:
   https://localhost:<SERVER_PORT>/admin
2. In ANALYTICS menu, Click MANAGE ALERT TYPES to open the Manage Alert Types page.
3. Select the relevant check boxes based on the alert types to which you want to subscribe.
4. Under Email list, enter the list of email addresses that should receive alerts. The email addresses should be those of system administrators. Each email address can be separated with a comma or you can type Email address and press Enter.
5. Click Save to save the information.

An API Manager publisher/subscriber can enable/disable alert types based on the alerts that he/she wants to receive individually, as well as specify a list of email addresses to which the alerts should be sent.

1. Log into the API publisher with the username and password of a user with permission to publish using WSO2 API Manager.
2. In ANALYTICS menu, Click MANAGE ALERT TYPES to open the Manage Alert Types page.
2. Select the relevant check boxes based on the alert types to which you want to subscribe.
3. Under Email list, enter the list of email addresses that should receive alerts.
4. Click Save to save the information.

**Viewing Alerts**

The following procedure to view alerts is relevant only for system administrators. System administrators, API publishers as well as API subscribers receive alerts via notification emails if they have subscribed to one or more alert type.

Follow the procedure below to view alerts that were generated for the APIs deployed in your WSO2 API Manager installation.

1. Access the WSO2 API Manager Admin Portal by using the following URL, and log in using your credentials.
   
   https://<API-M_HOST>:<API-M_port>/admin
2. Once the Admin Portal opens, click the following icon in the top right corner of the view.

This opens the Alerts History page as shown in the example below. The All Alerts value is selected in the Alert Type field by default as shown in the example below. Therefore, all the alerts that were generated in your WSO2 API Manager installation are displayed by default in this page.

3. If you want to filter the alerts displayed in the page by a specific alert type, select the required alert type in the Alert Type field as shown in the example below.
Integrating with Google Analytics

You can configure the API Manager to track runtime statistics of API invocations through Google Analytics (http://www.google.com/analytics). Google Analytics is a service that allows you to track visits to a website and generate detailed statistics on them.

This guide explains how to setup API Manager in order to feed runtime statistics to Google analytics for summarization and display.

1. Setup a Google Analytics account if not subscribed already and receive a Tracking ID, which is of the format "UA-XXXXXXXX-X". A Tracking ID is issued at the time an account is created with Google Analytics.
2. Log in to the API Manager management console (https://localhost:9443/carbon) using admin/admin credentials and go to Main -> Resources -> Browse menu.
4. Change the `<Enabled>` element to `true`, set your tracking ID in `<TrackingID>` element and Save.

```
<GoogleAnalyticsTracking>
  <Enabled>true</Enabled>
  <TrackingID>UA-XXXXXXX-X</TrackingID>
</GoogleAnalyticsTracking>
```

The following screen shot illustrates this change:

5. If you want to enable tracking for tenants, log in to the management console with a tenant’s credentials, click Source View, and then add the following parameter to the `org.wso2.carbon.mediation.registry.WSO2Registry` registry definition near the top (repeat this step for each tenant):

```
<parameter name="cachableDuration">15000</parameter>
```

The following screen shot illustrates this change:
6. API Manager is now integrated with Google Analytics. A user who has subscribed to a published API through the API Store should see an icon as Real-Time after logging into their Google Analytics account. Click on this icon and select Overview.

7. Invoke the above API using the embedded WSO2 REST Client (or any third-part rest client such as cURL).

Real-time statistics

8. This is one invocation of the API. Accordingly, Google Analytics graphs and statistics will be displayed at runtime. This example displays the Page Views per second graph and 1 user as active.

Report statistics

Google analytics reporting statistics take more than 24 hours from the time of invocation to populate. Shown below is a sample Dashboard with populated statistics.
There are widgets with statistics related to Audience, Traffic, Page Content, Visit Duration etc. You can add any widget of your preference to dashboard.

Monitoring with WSO2 Carbon Metrics

WSO2 Carbon Metrics provides an API for WSO2 Carbon Components to use the Metrics library. WSO2 API Manager is shipped with JVM Metrics to monitor statistics of your API Manager server using Java Metrics. The following sections provide a detailed description of how Carbon Metrics is used in API Manager for monitoring.

- Enabling Metrics and Storage Types
- Configuring Metrics Properties
- Using JVM Metrics

Enabling Metrics and Storage Types

Given below are the configurations that should be in place for your API Manager server in order to use the metrics feature. You need to first enable metrics for your server and then enable the required storage types (reporters) that are used for storing the metrics data. See the following topics for instructions:

- Enabling metrics
- Configuring the storage of metrics
- Sample configuration

Enabling metrics

To enable metrics for your product, set the Enabled parameter under the Metrics element to true in the <APIM_HOME>/repository/conf/metrics.xml file. Alternatively, you can enable metrics at the time of starting the API Manager server by using the following command:

```
-Dmetrics.enabled=true
```

Configuring the storage of metrics

WSO2 API Manager is configured by default to store the information from metrics in the following reporters: JMX, CSV and JDBC. These reporters are configured in the metrics.xml file (stored in the <APIM_HOME>/repository/conf directory). You can disable metrics for individual reporters by setting the Enabled parameter to false.

If you set the Enabled parameter under the metrics element to false in the metrics.xml file, metrics is disabled for all the reporters and it is not possible to enable metrics for individual reporters.

See the following topics for information on configuring each of the available storage types.

- JMX
- CSV
- JDBC
**JMX**

The following parameters in the `metrics.xml` file can be used to configure a JMX storage for metrics data.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether metrics monitoring is enabled for JMX or not.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**CSV**

The following parameters in the `metrics.xml` file can be used to configure a CSV storage for metrics data.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether metrics monitoring is enabled for CSV or not.</td>
<td>Boolean</td>
<td>false</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Location</td>
<td>The location where the CSV files are stored.</td>
<td>String</td>
<td>${carbon.home}/repository/logs/metrics/</td>
<td></td>
</tr>
<tr>
<td>PollingPeriod</td>
<td>The time interval between polling activities that are carried out to update the metrics dashboard based on latest information. For example, if the polling period is 60, polling would be carried out every 60 milliseconds.</td>
<td>Integer</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

**JDBC**

The following parameters in the `metrics.xml` file can be used to configure a JDBC storage for metrics data.

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory/Optional</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether metrics monitoring is enabled for JDBC or not.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>DataSourceName</td>
<td>The name of the datasource used.</td>
<td>String</td>
<td>jdbc/WSO2MetricsDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PollingPeriod</td>
<td>The time interval between polling activities that are carried out to update the metrics dashboard based on latest information. For example, if the polling period is 60, polling would be carried out every 60 seconds.</td>
<td>Integer</td>
<td>60</td>
<td></td>
<td>This value is specified in seconds.</td>
</tr>
</tbody>
</table>

If you have enabled JDBC, then you also need to specify a datasource configuration to be used to create the connection between WSO2 API Manager and the JDBC data storage system. The `metrics-datasources.xml` file is used for configuring this datasource for metrics.
Parameters that can be configured for a datasource are as follows:

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
<th>Default value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;datasources-configuration&gt;</td>
<td>xmlns</td>
<td>The root element. The namespace is specified as: xmlns:svns=&quot;<a href="http://org.wso2.securevault/configuration">http://org.wso2.securevault/configuration</a>&quot;</td>
<td></td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;providers&gt;</td>
<td></td>
<td>The container element for the datasource providers.</td>
<td></td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;provider&gt;</td>
<td></td>
<td>The datasource provider, which should implement org.wso2.carbon.ndatasource.common.spi.DataSourceReader. The datasources follow a pluggable model in providing datasource type implementations using this approach.</td>
<td>Fully qualified Java class</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;datasources&gt;</td>
<td></td>
<td>The container element for the datasources.</td>
<td></td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;datasource&gt;</td>
<td></td>
<td>The root element of a datasource.</td>
<td></td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td></td>
<td>Name of the datasource.</td>
<td>String</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;description&gt;</td>
<td></td>
<td>Description of the datasource.</td>
<td>String</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;jndiConfig&gt;</td>
<td></td>
<td>The container element that allows you to expose this datasource as a JNDI datasource.</td>
<td></td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td></td>
<td>The JNDI resource name to which this datasource should be bound.</td>
<td>String</td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;environment&gt;</td>
<td></td>
<td>The container element in which you specify the following JNDI properties:</td>
<td>Fully qualified Java class</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;definition&gt;</td>
<td>type</td>
<td>The container element for the data source definition. Set the type attribute to RDBMS, or to custom if you're creating a custom type. The &quot;RDBMS&quot; data source reader expects a &quot;configuration&quot; element with the sub-elements listed below.</td>
<td>String</td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;configuration&gt;</td>
<td></td>
<td>The container element for the RDBMS properties.</td>
<td></td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;url&gt;</td>
<td></td>
<td>The connection URL to pass to the JDBC driver to establish the connection.</td>
<td>URL</td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;username&gt;</td>
<td></td>
<td>The connection user name to pass to the JDBC driver to establish the connection.</td>
<td>String</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;password&gt;</td>
<td></td>
<td>The connection password to pass to the JDBC driver to establish the connection.</td>
<td>String</td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;driverClassName&gt;</td>
<td></td>
<td>The class name of the JDBC driver to use.</td>
<td>Fully qualified Java class</td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;maxActive&gt;</td>
<td></td>
<td>The maximum number of active connections that can be allocated from this pool at the same time.</td>
<td>Integer</td>
<td>100</td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;maxWait&gt;</td>
<td></td>
<td>Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception.</td>
<td>Integer</td>
<td>30000 (30 seconds)</td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;testOnBorrow&gt;</td>
<td></td>
<td>Specifies whether objects are validated before being borrowed from the pool. If the object fails to validate, it is dropped from the pool, and we will attempt to borrow another. When set to true, the validationQuery parameter must be set to a non-null string.</td>
<td>Boolean</td>
<td>false</td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;validationQuery&gt;</td>
<td></td>
<td>The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).</td>
<td>String</td>
<td>null</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
To avoid excess validation, only run validation at most at this frequency (interval time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).

Sample configuration

Shown below is a sample metrics.xml file with the default configurations specifying the types of storages enabled for metrics data. See the above topics for instructions.

The default configurations in the metrics.xml file

```xml
<Metrics xmlns="http://wso2.org/projects/carbon/metrics.xml">
    <!-- Enable Metrics -->
    <Enabled>false</Enabled>
    <!-- Metrics reporting configurations -->
    <Reporting>
        <JMX>
            <Enabled>true</Enabled>
        </JMX>
        <CSV>
            <Enabled>false</Enabled>
            <Location>${carbon.home}/repository/logs/metrics/</Location>
            <PollingPeriod>60</PollingPeriod>
        </CSV>
        <JDBC>
            <Enabled>true</Enabled>
            <Source>Carbon</Source>
            <DataSourceName>jdbc/WSO2MetricsDB</DataSourceName>
            <PollingPeriod>60</PollingPeriod>
            <ScheduledCleanup>
                <Enabled>true</Enabled>
                <ScheduledCleanupPeriod>86400</ScheduledCleanupPeriod>
                <DaysToKeep>7</DaysToKeep>
            </ScheduledCleanup>
        </JDBC>
    </Reporting>
</Metrics>
```
Configuring Metrics Properties

The `<APIM_HOME>/repository/conf/metrics.properties` file specifies properties that correspond to the gauges in the Metrics Dashboard. See the topic on using JVM metrics for details on using the metrics dashboard for JVM metrics. The level defined for a property in this file determines the extent to which the relevant gauge in the dashboard should be updated with information. The different levels that can be defined for properties are as follows:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>Designates no informational events.</td>
</tr>
<tr>
<td>Info</td>
<td>Designates informational metric events that highlight the progress of the application at coarse-grained level.</td>
</tr>
<tr>
<td>Debug</td>
<td>Designates fine-grained informational events that are most useful to debug an application.</td>
</tr>
<tr>
<td>Trace</td>
<td>Designates finer-grained informational events than the DEBUG.</td>
</tr>
<tr>
<td>All</td>
<td>Designates all the informational events.</td>
</tr>
</tbody>
</table>

If no specific level is configured for a property in the `metrics.properties` file, the metrics root level applies. The root level is defined as shown in the following example in the `metrics.properties` file.

```
metrics.rootLevel=OFF
```

If you want to change the current root level, you can also use the following command.

```
-Dmetrics.rootLevel=INFO
```

The levels in `metrics.properties` file can be configured to any hierarchy. However, if the level defined for an individual property is different to the level defined for its parent in the hierarchy, the level defined for the individual property overrules that of the parent. For example, if we have `metric.level.jvm.memory=INFO` in the `<APIM_HOME>/repository/conf/metrics.properties` file, all metrics under `jvm.memory` have INFO as the configured level. However, if you have `metric.level.jvm.memory.heap=TRACE`, the TRACE level would apply for the `metric.level.jvm.memory.heap` property even though it is a child property of `jvm.memory`.

The properties that are included in this file by default are as follows:

- JVM's direct and mapped buffer pools
- Class loading
- GC
- Memory
- Operating system load
- Threads

JVM's direct and mapped buffer pools

Class loading

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.class-loading</td>
<td>INFO</td>
<td>The gauge showing the number of classes currently loaded for the JVM.</td>
</tr>
</tbody>
</table>

GC

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.gc</td>
<td>DEBUG</td>
<td>The gauge for showing garbage collection and memory usage. Monitoring this allows you to identify memory leaks and memory thrash, which have a negative impact on performance.</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.memory</td>
<td>INFO</td>
<td>The gauge for showing the used and committed memory in WSO2 API Manager.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.heap</td>
<td>INFO</td>
<td>The gauge for showing the used and committed heap in WSO2 API Manager.</td>
</tr>
</tbody>
</table>
The gauge for showing the used code cache and used CMS Perm Gen in WSO2 API Manager.

The gauge for showing the total memory currently available for the JVM.

The gauge for showing the used and available memory for JVM in the memory pool.

### Operating system load

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.os</td>
<td>INFO</td>
<td>The gauge for showing the current load imposed by the JVM on the operating system.</td>
</tr>
</tbody>
</table>

### Threads

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.threads</td>
<td>OFF</td>
<td>The parent property of all the gauges relating to the JVM thread pool. The metric level defined for this property applies to all the rest of the properties in this table. The metric level set via this property to a child property can be overruled if a different level is set for it.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.count</td>
<td>DEBUG</td>
<td>The gauge for showing the number of active and idle threads currently available in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.daemon.count</td>
<td>DEBUG</td>
<td>The gauge for showing the number of active daemon threads currently available in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.blocked.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads that are currently blocked in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.deadlock.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads that are currently deadlocked in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.new.count</td>
<td>OFF</td>
<td>The gauge for showing the number of new threads generated in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threadsRunnable.count</td>
<td>OFF</td>
<td>The gauge for showing the number of runnable threads currently available in the JVM thread pool.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.terminated.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads terminated from the JVM thread pool since you started running the WSO2 API Manager instance.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.timed_waiting.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads in the Timed_Waiting state.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.waiting.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads in the Waiting state in the JVM thread pool. One or more other threads are required to perform certain actions before these threads can proceed with their actions.</td>
</tr>
</tbody>
</table>

### Viewing API specific Handler Metrics

Follow the below steps to view API specific metrics for handlers which are exposed through Dropwizard. These are timer metrics which allow us to see how much time is taken for each handler.

1. Copy the JMX Service URL displayed in the console at API Manager server startup.

```
INFO - JMXServerManager JMX Service URL :
```

2. Go to `<JDK_HOME>/bin` where `<JDK_HOME>` is the directory where the JDK(Java Development Kit) is installed. Find the `jconsole` executable file in the directory and run it on the command/shell prompt.

   You can type `jconsole` in the command/shell prompt to run the file.

3. In the prompted window, enter the JMX Service URL copied before as a new connection. Type `admin` in both username and password fields and click `Connect`.
4. Now invoke an API in WSO2 API Manager.

   Handler Metrics will only be visible when you invoke an API.
5. Go to the MBeans tab in the open jconsole window. You can view the implemented metrics by using Dropwizard Metrics Library which starts with the org.wso2.am prefix.

**Using JVM Metrics**

JVM metrics are Java metrics enabled by default in WSO2 products for the purpose of monitoring general statistics related to the server performance. Follow the procedure below to view the JVM metrics dashboard for a WSO2 product.

For detailed instructions to enable/disable JVM metrics and to configure metrics, see Enabling Metrics and Storage Types.

1. Log into the Management Console of the WSO2 product. Click Monitor -> Metrics -> JVM Metrics to open the View Metrics page.
2. Specify the source for the JVM metrics by selecting a value from the drop-down list for the Source parameter in the top panel.
3. Specify the time interval for which the statistics should be displayed in the dashboard by selecting a value from the following drop-down list in the top panel.
4. Click the required buttons opposite Views in the top panel to select the types of information you want to view in the dashboard, and refresh the web page.

Statistics corresponding to each button can be viewed as follows:

- **CPU**
  Click this button to view statistics relating to the CPU as shown below.

```
Source: WSO2x-MacBook-Air-59.local
CPU: View Metrics
Last 5 minutes
```

```
Source: WSO2x-MacBook-Air-59.local
Memory: View Metrics
Last 5 minutes
```

```
Source: WSO2x-MacBook-Air-59.local
Threading: View Metrics
Last 5 minutes
```

```
Source: WSO2x-MacBook-Air-59.local
Class Loading: View Metrics
Last 5 minutes
```

```
Source: WSO2x-MacBook-Air-59.local
File Descriptor: View Metrics
Last 5 minutes
```
• **Memory**
  
  Click **Memory** to view statistics relating to the memory as shown below.

![Memory Graph](image)

• **Threading**
  
  Click **Threading** to view statistics relating to threading as shown below.

![Threading Graph](image)
Class Loading
Click Class Loading to view statistics relating to class loading as shown below.

File Descriptor
Click File Descriptor to view information relating to the file descriptor count as shown below.

Purging Analytics Data
Data purging is an option to remove historical data in WSO2 API Manager Analytics. This is important because it is not possible to delete tables or table data in WSO2 API Manager Analytics. By purging data, you can achieve high performance in data analysis without removing analyzed summary data.

Make sure you do not purge data in tables other than those mentioned tables related to alerts, because data purging deletes your summarized historical data.
**Purging alert related tables**

Data Purging is enabled by default for all the aggregation tables. However, you can purge data in certain alert tables, namely `APIMIpAccessSummary` and `APIMIpAccessAlertCount` tables, which have the potential to grow quickly like aggregation tables. Follow the instructions below to purge the data in the latter mentioned alert tables.

1. Configure WSO2 API Manager (WSO2 API-M) with WSO2 API-M Analytics, and start WSO2 API-M Analytics, WSO2 API Manager, and the WSO2 API-M Analytics Dashboard node in the latter mentioned order.

   For more information, see Configuring APIM Analytics.


   https://<HOSTNAME>:<DASHBOARD_PORT>/business-rules

   For example: https://localhost:9643/business-rules

3. Click the edit link that corresponds to the APIM DATA PURGING business rule.

   ![Business Rules Manager](image)

<table>
<thead>
<tr>
<th>Business Rule</th>
<th>Status</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIM Unusual IP Access Alert</td>
<td>Deployed</td>
<td></td>
</tr>
<tr>
<td>APIM Frequent Tier Limit Hitting Alert</td>
<td>Deployed</td>
<td></td>
</tr>
<tr>
<td>APIM Health Availability Alert</td>
<td>Deployed</td>
<td></td>
</tr>
<tr>
<td>APIM Abnormal Back End Time</td>
<td>Deployed</td>
<td></td>
</tr>
<tr>
<td>APIM Abnormal Request Pattern</td>
<td>Deployed</td>
<td></td>
</tr>
<tr>
<td>APIM DATA PURGING</td>
<td>Deployment Failure</td>
<td></td>
</tr>
<tr>
<td>APIM Abnormal Request Count</td>
<td>Deployed</td>
<td></td>
</tr>
<tr>
<td>APIM Abnormal Response Time</td>
<td>Deployed</td>
<td></td>
</tr>
</tbody>
</table>

4. Click **Save and Deploy** to enable data purging in the `APIMIpAccessSummary` and `APIMIpAccessAlertCount` tables.

   When the above business rule is enabled, it will purge data from `APIMIpAccessSummary` and `APIMIpAccessAlertCount` tables every time the trigger is triggered. The default time interval for the trigger is 1 year.
Default Ports of WSO2 API-M Analytics

Given below are the specific ports used by WSO2 API-M Analytics

Ports inherited from WSO2 SP

- 7712 - Thrift SSL port for secure transport, where the client is authenticated to use WSO2 API-M Analytics.
- 7612 - Thrift TCP port where WSO2 API-M Analytics receives events from clients.
- 7444 - The default port for the Stream Processor Store API.
- 9444 - MSF4J HTTPS Port in the Stream Processor. This is used by the Microgateway.

Enabling API-M Analytics Features in WSO2 Stream Processor

WSO2 API Manager Analytics is powered by WSO2 Stream Processor. You can enable APIM Analytics features in Stream Processor 4.3.0 by following the steps below.

1. Open the `<SP_HOME>/conf/<PROFILE>/deployment.yaml` file.

   `<PROFILE>` refers to dashboard, editor, manager and worker profile directories. Each of these directories contain a deployment.yaml file and all of them need to be edited when you enable/disable solutions.

2. Enable API-M Analytics is enabled by setting the `APIM-analytics.enabled` parameter to `true` as shown in the sample below.

If you need to purge the data immediately or cannot wait for the next trigger, then you need to reduce the default time interval to trigger data purging by changing the `Time Interval For Trigger` value to a smaller value (e.g., one minute). You can set the time interval for trigger in seconds, minutes, hours, days, months, and years.
## Carbon Configuration Parameters

```yaml
wso2.carbon:
  # server type
  type: wso2-sp
analytics.solutions:
  APIM-analytics.enabled: true
```

You have now enabled Analytics for WSO2 API Manager in the WSO2 Stream Processor.

### Reference Guide

The following topics provide reference information for working with WSO2 API Manager:

- WSO2 API Manager Best Practices
- Product Profiles
- Default Product Ports
- Changing the Default Ports with Offset
- Error Handling
- Message Flow in the API Manager Gateway
- Updating WSO2 API Manager
- Accessing API Manager by Multiple Devices Simultaneously

### WSO2 API Manager Best Practices

Here are the guidelines and recommendations to design and deploy APIs using WSO2 API Manager:

- **Proper Naming APIs**
  - Naming Resources
  - Naming Complex Resources
- **Proper Versioning**
  - Advantages of API Versioning
  - Major, Minor, Patch Versions

#### Best practices for creating an API

- Create APIs for dedicated backend services.
- For each of the resources, decide on the HTTP methods that are used to perform the required application functions. This includes the use of applicable HTTP headers.
- Decide on special behaviors required by the application (e.g., concurrency control, long running requests).
- Identify potential error-prone situations and define corresponding error messages.

#### Proper Naming APIs

It's important to have proper names for services and service paths. For example, if you create an API related to camera-related operations, you can select a name with camera-related terms such as "camera-api". When you create a resource for the API, you can select names such as "capture-image", "delete-image". Resources must be named properly through URIs (Uniform Resource Identifiers). Proper naming of resources is key to increase the understandability of an API to its users.

Following are some of the guidelines for designing proper API/Resource paths and names. Note that these are not mandatory rules but best practices.

#### Naming Resources

Atomic resources, collection resources, and composite resources should be named as nouns because they represent 'things'; not 'actions'. Actions lean more towards verbs as names of resources. Processing-function resources and controller resources should be named as verbs because they represent actions. Function resources and controller resources should not be sub-resources of individual resources.

#### Naming Complex Resources

Use lower case characters only in names. This is because the rules about which URI element names are case sensitive and which are not might cause confusion. If multiple words are used to name a resource, separate the words by dashes (-) or some other separator. Use singular nouns to name atomic resources. Use pluralized names for collections. That is, the name should be the plural noun of the grouped concept (atomic resource or composite resource). Use forward slashes (/) to specify hierarchical relations between resources. A forward slash separates the names of the hierarchically structured resources. The parent name precedes the name of its immediate children.

#### Proper Versioning

The version of an API is part of its URI. It is usually given as a pair of integers (separated by a dot) referred to as the major and the minor number of the version, preceded by the lowercase letter "v". For example, a valid version string in the base path is v2.1, indicating the first minor version of the second major version of the corresponding API.

#### Advantages of API Versioning
A proper versioning strategy is helpful for all API users and clients to communicate with APIs easily and effectively. WSO2 API Manager has versioning support. You can copy existing APIs and create new versions of the same APIs. The recommended way to change the functionality of a running API is to create a new version from the existing API. You can then modify the new version, and test the functionalities of the new version before publishing it.

**Major, Minor, Patch Versions**

In general, a version number following the semantic versioning concept and have the structure major.minor.patch. The significance, in terms of client impact, increases from left to right. As a best practice, you should support the current major version and at least one previous, major version. In case you release new versions frequently (e.g., every few months), it is recommended to support more previous major versions.

1. **Patch Version** - An incremental patch number means that the underlying modification to the API cannot even be noticed by a client and therefore, the patch number is omitted from the version string. Only the internal implementation of the API has changed while the signature of the API is unchanged. A new patch number can be due to a bug fix, a minor internal modification, etc.
2. **Minor Version** - An incremented minor number means that new features have been added to the API, but this addition must be backward compatible. That is, the client can use the old API without failures. For example, the API might add new optional parameters or a completely new request.
3. **Major Version** - An incremented major number means that the changes are not backward compatible. For example, new mandatory parameters have been added, former parameters have been dropped, or complete former requests are no longer available.

**Product Profiles**

When a WSO2 product starts, it starts all components, features and related artifacts bundled with it. Multi-profile support allows you to run the product on a selected profile so that only the features specific to that profile along with common features start up with the server.

Starting a product on a preferred profile only blocks/allows the relevant OSGi bundles. As a result, even if you start the server on a profile such as the api-store for example, you can still access the API Publisher web application.

**OSGi Bundle**

OSGi bundle is a tightly coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any). In OSGi, a bundle is the primary deployment format. Bundles are applications that are packaged in JARs, and can be installed, started, stopped, updated, and removed.

- API-M Profiles
- Starting an API-M profile
  - Method 1 - Optimizing before starting the server
  - Method 2 - Optimizing while starting the server
- How multi-profiling works

**API-M Profiles**

The following are the different profiles available in WSO2 API Manager.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Command Option with Profile Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway worker</td>
<td>-Dprofile=gateway-worker</td>
<td>Only starts the components related to the API Gateway. You use this when the API Gateway acts as a worker node in a cluster. This profile starts the backend features for data processing and communicating with the manager node.</td>
</tr>
<tr>
<td>Key Manager</td>
<td>-Dprofile=api-key-manager</td>
<td>Only starts the features relevant to the Key Manager component of the API Manager.</td>
</tr>
<tr>
<td>Traffic Manager</td>
<td>-Dprofile=traffic-manager</td>
<td>Only starts the features relevant to the Traffic Manager component of the API Manager. The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests.</td>
</tr>
<tr>
<td>API Publisher</td>
<td>-Dprofile=api-publisher</td>
<td>Only starts the front end/backend features relevant to the API Publisher.</td>
</tr>
<tr>
<td>Developer Portal (API Store)</td>
<td>-Dprofile=api-store</td>
<td>Only starts the front end/backend features relevant to the Developer Portal (API Store).</td>
</tr>
</tbody>
</table>
**Starting an API-M profile**

You can start an API Manager profile in the following methods, based on your requirement

- Method 1 - Optimizing before starting the server
- Method 2 - Optimizing while starting the server

It is recommended to start the components in the following order: Key Manager, Publisher, Store, Traffic Manager, and Gateway.

**Method 1 - Optimizing before starting the server**

Create an optimized distribution for a particular API-M profile.

1. Run the `<PRODUCT_HOME>/bin/profileSetup.sh` script or `<PRODUCT_HOME>/bin/profileSetup.bat` script based on your operating system, with the profile flag.

   ```bash
   sh <PRODUCT_HOME>/bin/profileSetup.sh -Dprofile=<preferred-profile>
   ```

   **Starting the Publisher profile**

   ```bash
   sh <PRODUCT_HOME>/bin/profileSetup.sh -Dprofile=api-publisher
   ```

2. Start the server with the specified profile. A sample command is shown below.

   ```bash
   sh <PRODUCT_HOME>/bin/wso2server.sh -Dprofile=<preferred-profile>
   ```

   **Starting the Publisher profile**

   ```bash
   sh <PRODUCT_HOME>/bin/wso2server.sh -Dprofile=api-publisher
   ```

**Method 2 - Optimizing while starting the server**

1. Start the server using the script based on your operating system, using the command given below.

   ```bash
   sh <PRODUCT_HOME>/bin/wso2server.sh --optimize -Dprofile=<preferred-profile>
   ```

   **Starting the Publisher profile**

   ```bash
   sh <PRODUCT_HOME>/bin/wso2server.sh --optimize -Dprofile=api-publisher
   ```

Click here to see the sample output when you start the Publisher profile
Starting to optimize API Manager for the API Publisher profile
[2018-06-19 17:36:08:045] INFO - Disabled the <DataPublisher> from api-manager.xml file
[2018-06-19 17:36:08:055] INFO - Disabled the <JMSConnectionDetails> from api-manager.xml file
[2018-06-19 17:36:08:066] INFO - Disabled the <transportSender name="ws" class="org.wso2.carbon.websocket.transport.WebsocketTransportSender"> from axis2.xml file
[2018-06-19 17:36:08:072] INFO - Disabled the <transportSender name="wss" class="org.wso2.carbon.websocket.transport.WebsocketTransportSender"> from axis2.xml file
[2018-06-19 17:36:08:074] INFO - Removed the WebSocketInboundEndpoint.xml file from 
../repository/deployment/server/synapse-configs/default/inbound-endpoints/
../repository/deployment/server/synapse-configs/default/inbound-endpoints/
[2018-06-19 17:36:08:085] INFO - Removed the authenticationendpoint.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:087] INFO - Removed the api-am-admin#v0.13.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:090] INFO - Removed the throttle#data#v1.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:093] INFO - Removed the oauth2.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:097] INFO - Removed the am-sample#pizzashack#v1.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:099] INFO - Removed the api-identity#consent-mgt#v1.0.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:102] INFO - Removed the am-sample#calculator#v1.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:105] INFO - Removed the api-am-store#v0.13.war file from 
../repository/deployment/server/webapps
[2018-06-19 17:36:08:131] INFO - Removed store directory from 
../repository/deployment/server/jaggeryapps
Finished the optimizations
Starting the server...

Note that registry indexing for the Traffic Manager and Gateway worker profiles are disabled when you run the profile optimization. To enable registry indexing, open the <PRODUCT_HOME>/repository/conf/registry.xml file. Set the value of the <startIndexing> parameter in the <indexingConfiguration> section to true as shown below.

```
<indexingConfiguration>
  <startIndexing>true</startIndexing>
  . . .
</indexingConfiguration>
```

**How multi-profiling works**

Starting a product on a preferred profile starts only a subset of features bundled in the product. In order to identify what feature bundles apply to which profile, each product maintains a set of bundles.info files in the <PRODUCT_HOME>/repository/components/<profile-name>/configuration/or.../simpleconfigurator directories. The bundles.info files contain references to the actual bundles. Note that the bundles.info file in the corresponding directory path refers to the name of the profile. For example, when there’s a product profile named webapp, references to all the feature bundles required for webapp profile to function are in bundles.info file saved in the <PRODUCT_HOME>/repository/components/webapp/configuration/or...simpleconfigurator directory.

Note that when you start the server without using a preferred profile, the server refers to the <PRODUCT_HOME>/repository/components/default/configuration/or.../simpleconfigurator/bundles.info file by default. This file contains references to all bundles in the <PRODUCT_HOME>/repository/components/plugins directory, which is where all components/bundles of a product are saved.

**Default Product Ports**

This page describes the default ports that are used for each WSO2 product when the port offset is 0.

Copyright © WSO2 Inc. 2015-2018
Note that it is recommended to disable the HTTP transport in an API Manager production setup. Using the Bearer token over HTTP is a violation of the OAuth specification and can lead to security vulnerabilities.

- Product-specific ports
- Disabling HTTP Transports

Common ports

The following ports are common to all WSO2 products that provide the given feature. Some features are bundled in the WSO2 Carbon platform itself and therefore are available in all WSO2 products by default.

Management console ports

WSO2 products that provide a management console use the following servlet transport ports:

- 9443 - HTTPS servlet transport (the default URL of the management console is https://localhost:9443/carbon)
- 9763 - HTTP servlet transport

WSO2 Enterprise Integrator (WSO2 EI) uses the following ports to access the management console:

- 9443 - HTTPS servlet transport for the ESB runtime (the default URL of the management console is https://localhost:9443/carbon)
- 9445 - HTTPS servlet transport for the EI-Business Process runtime (the default URL of the management console is https://localhost:9445/carbon)
- 9444 - Used for the EI-Analytics management console

LDAP server ports

Provided by default in the WSO2 Carbon platform.

- 10389 - Used in WSO2 products that provide an embedded LDAP server

KDC ports

- 8000 - Used to expose the Kerberos key distribution center server

JMX monitoring ports

WSO2 Carbon platform uses TCP ports to monitor a running Carbon instance using a JMX client such as JConsole. By default, JMX is enabled in all products. You can disable it using <PRODUCT_HOME>/repository/conf/etc/jmx.xml file.

- 11111 - RMIRegistry port. Used to monitor Carbon remotely
- 9999 - RMIServer port. Used along with the RMIRegistry port when Carbon is monitored from a JMX client that is behind a firewall

Clustering ports

To cluster any running Carbon instance, either one of the following ports must be opened.

- 45564 - Opened if the membership scheme is multicast
- 4000 - Opened if the membership scheme is wka

Random ports

Certain ports are randomly opened during server startup. This is due to specific properties and configurations that become effective when the product is started. Note that the IDs of these random ports will change every time the server is started.

- A random TCP port will open at server startup because of the -Dcom.sun.management.jmxremote property set in the server startup script. This property is used for the JMX monitoring facility in JVM.
- A random UDP port is opened at server startup due to the log4j appender (SyslogAppender), which is configured in the <PRODUCT_HOME>/repository/conf/log4j.properties file.

Product-specific ports

Some products open additional ports.

API Manager | BPS | Data Analytics Server | Complex Event Processor | Elastic Load Balancer | ESB | Enterprise Integrator | Identity Server | Message Broker | Machine Learner | Storage Server | Enterprise Mobility Manager | IoT Server

API Manager

- 5672 - Used by the internal Message Broker.
- 7611 - Authenticate data published when Thrift data publisher is used for throttling.
- 7612 - Publish Analytics to the API Manager Analytics server.
- 7711 - Port for secure transport when Thrift data publisher is used for throttling.
• **7711** - Port offset of the APIM Analytics Server - Thrift SSL port for secure transport when publishing analytics to the API Manager Analytics server.

• **8280, 8243** - NIO/PT transport ports.

• **9611** - Publish data to the Traffic Manager. Required when binary data publisher for throttling.

• **9711** - Authenticate data published to the Traffic Manager. Required when binary data publisher for throttling.

• **10397** - Thrift client and server ports.

• **9099** - Web Socket ports.

• **Traffic Manager related ports**
  - **9160** - Cassandra port on which Thrift listens to clients
  - **7711** - Thrift SSL port for secure transport, where the client is authenticated to WSO2 CEP
  - **7611** - Thrift TCP port to receive events from clients to WSO2 CEP
  - **11224** - Thrift TCP port for HA management of WSO2 CEP

If you change the default API Manager ports with a port offset, most of its ports will be changed automatically according to the offset except a few exceptions described in the APIM Manager documentation.

**BPS**

• **2199** - RMI registry port (datasources provider port)

**Data Analytics Server**

Given below are the specific ports used by WSO2 DAS.

Ports inherited from WSO2 BAM

WSO2 DAS inherits the following port configurations used in its predecessor, **WSO2 Business Activity Monitor (BAM)**.

• **7711** - Thrift SSL port for secure transport, where the client is authenticated to use WSO2 DAS.

• **7611** - Thrift TCP port where WSO2 DAS receives events from clients.

Ports used by the Spark Analytics Engine

The Spark Analytics engine is used in 3 separate modes in WSO2 DAS as follows.

• Local mode

• Cluster mode

• Client mode

Default port configurations for these modes are as follows.

For more information on these ports, go to [Apache Spark Documentation](https://spark.apache.org/).

**Ports available for all modes**

The following ports are available for all three modes explained above.

<table>
<thead>
<tr>
<th>Description</th>
<th>Port number</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.ui.port</td>
<td>4040</td>
</tr>
<tr>
<td>spark.history.ui.port</td>
<td>18080</td>
</tr>
<tr>
<td>spark.blockManager.port</td>
<td>12000</td>
</tr>
<tr>
<td>spark.broadcast.port</td>
<td>12500</td>
</tr>
<tr>
<td>spark.driver.port</td>
<td>13000</td>
</tr>
<tr>
<td>spark.executor.port</td>
<td>13500</td>
</tr>
<tr>
<td>spark.fileserver.port</td>
<td>14000</td>
</tr>
<tr>
<td>spark.replClassServer.port</td>
<td>14500</td>
</tr>
</tbody>
</table>

**Ports available for the cluster mode**

The following ports are available only for the cluster mode.

<table>
<thead>
<tr>
<th>Description</th>
<th>Port number</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.master.port</td>
<td>7077</td>
</tr>
<tr>
<td>spark.master.rest.port</td>
<td>6066</td>
</tr>
<tr>
<td>spark.master.webui.port</td>
<td>8081</td>
</tr>
</tbody>
</table>
spark.worker.port 11000
spark.worker.webui.port 11500

**Complex Event Processor**

The following ports are also applicable when clustering and deploying the Traffic Manager in a distributed environment in WSO2 API Manager.

- 9160 - Cassandra port on which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to CEP
- 7611 - Thrift TCP port to receive events from clients to CEP
- 11224 - Thrift TCP port for HA management of CEP

**Elastic Load Balancer**

- 8280, 8243 - NIO/PT transport ports

**ESB**

Non-blocking HTTP/S transport ports: Used to accept message mediation requests. If you want to send a request to an API or a proxy service for example, you must use these ports. ESB_HOME/repository/conf/axis2(axis2.xml file.

- 8243 - Passthrough or NIO HTTPS transport
- 8280 - Passthrough or NIO HTTP transport

**Enterprise Integrator**

Integration runtime ports

- 9443 - HTTPS servlet transport (the default URL of the management console is https://localhost:9443/carbon)

Non-blocking HTTP/S transport ports: Used to accept message mediation requests. If you want to send a request to an API or a proxy service for example, you must use these ports: <EI_HOME>/conf/axis2(axis2.xml file.

- 8243 - Passthrough or NIO HTTPS transport
- 8280 - Passthrough or NIO HTTP transport

EI-Analytics runtime ports

- 9444 - Management console port
- 9161 - Cassandra port on which Thrift listens to clients
- 7712 - Thrift SSL port for secure transport, where the client is authenticated to DAS
- 7612 - Thrift TCP port to receive events from clients to DAS

EI-Business Process runtime ports

- 9445 - HTTPS servlet transport (the default URL of the management console is https://localhost:9445/carbon)
- 9765 - HTTP servlet transport

EI-Broker runtime ports

- 9446 - HTTPS servlet transport (the default URL of the management console is https://localhost:9446/carbon)
- 9766 - HTTP servlet transport

EI-Broker uses the following JMS ports to communicate with external clients over the JMS transport.

- 5675 - Port for listening for messages on TCP when the AMQP transport is used.
- 8675 - Port for listening for messages on TCP/SSL when the AMQP Transport is used.
- 1886 - Port for listening for messages on TCP when the MQTT transport is used.
- 8836 - Port for listening for messages on TCP/SSL when the MQTT Transport is used.
- 7614 - The port for Apache Thrift Server.

**Identity Server**

- 8000 - KDCServerPort. Port which KDC (Kerberos Key Distribution Center) server runs
- 10500 - ThriftEntitlementReceivePort

**Message Broker**

Message Broker uses the following JMS ports to communicate with external clients over the JMS transport.

- 5672 - Port for listening for messages on TCP when the AMQP transport is used.
- 8672 - Port for listening for messages on TCP/SSL when the AMQP Transport is used.
- 1883 - Port for listening for messages on TCP when the MQTT transport is used.
- 8833 - Port for listening for messages on TCP/SSL when the MQTT Transport is used.
- 7611 - The port for Apache Thrift Server.
Machine Learner

- 7077 - The default port for Apache Spark.
- 54321 - The default port for H2O.
- 4040 - The default port for Spark UI.

Storage Server

Cassandra:

- 7000 - For inter node communication within cluster nodes
- 7001 - For inter node communication within cluster nodes via SSL
- 9160 - For Thrift client connections
- 7199 - For JMX

HDFS:

- 54310 - Port used to connect to the default file system.
- 54311 - Port used by the MapRed job tracker
- 50470 - Name node secure HTTP server port
- 50475 - Data node secure HTTP server port
- 50010 - Data node server port for data transferring
- 50075 - Data node HTTP server port
- 50020 - Data node IPC server port

Enterprise Mobility Manager

The following ports need to be opened for Android and iOS devices so that it can connect to Google Cloud Messaging (GCM)/Firebase Cloud Messaging (FCM) and APNS (Apple Push Notification Service) and enroll to WSO2 EMM.

Android:

The ports to open are 5228, 5229 and 5230. GCM/FCM typically only uses 5228, but it sometimes uses 5229 and 5230.

GCM/FCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.

iOS:

- 5223 - TCP port used by devices to communicate to APNs servers
- 2195 - TCP port used to send notifications to APNs
- 2196 - TCP port used by the APNs feedback service
- 443 - TCP port used as a fallback on Wi-Fi, only when devices are unable to communicate to APNs on port 5223

The APNs servers use load balancing. The devices will not always connect to the same public IP address for notifications. The entire 17.0.0.0/8 address block is assigned to Apple, so it is best to allow this range in the firewall settings.

API Manager:

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10397</td>
<td>Thrift client and server ports</td>
</tr>
<tr>
<td>8280, 8243</td>
<td>NIO/PT transport ports</td>
</tr>
</tbody>
</table>

IoT Server

The following ports need to be opened for WSO2 IoT Server, and Android and iOS devices so that it can connect to Google Cloud Messaging (GCM)/Firebase Cloud Messaging (FCM) and APNS (Apple Push Notification Service), and enroll to WSO2 IoT Server.

Default ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8243</td>
<td>HTTPS gateway port.</td>
</tr>
<tr>
<td>9443</td>
<td>HTTPS port for the core profile.</td>
</tr>
<tr>
<td>8280</td>
<td>HTTP gateway port.</td>
</tr>
<tr>
<td>9763</td>
<td>HTTP port for the core profile.</td>
</tr>
<tr>
<td>1886</td>
<td>Default MQTT port.</td>
</tr>
<tr>
<td>9445</td>
<td>HTTPS port for the analytics profile.</td>
</tr>
<tr>
<td>9765</td>
<td>HTTP port for the analytics profile.</td>
</tr>
<tr>
<td>1039</td>
<td>HTTP port for the analytics profile.</td>
</tr>
</tbody>
</table>

Ports required for mobile devices to communicate with the server and the respective notification servers.
Disabling Servlet Transport

Disabling Passthru Transport

1. The ports to open are 5228, 5229 and 5230. Google Cloud Messaging (GCM) and Firebase Cloud Messaging (FCM) typically only uses 5228, but it sometimes uses 5229 and 5230. GCM/FCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.

2. The ports to open are 5223. Transmission Control Protocol (TCP) port used by devices to communicate to APNs servers.

3. The ports to open are 2195. TCP port used to send notifications to APNs.

4. The ports to open are 2196. TCP port used by the APNs feedback service.

Disabling HTTP Transports

API Manager has two HTTP transports. See below for instructions on how to disable the following:

1. Passthru (API Traffic) Transport
2. Servlet (UI Traffic and Admin service access) Transport

Disabling Passthru Transport

Disabling Servlet Transport

1. Open the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file.
2. Locate the Transport receiver `http` as shown below:

   ```xml
   <transportReceiver name="http" class="org.apache.synapse.transport.passthru.PassThroughHttpListener">
   ...</transportReceiver>
   ```

   3. Comment out the `http` transport receiver section.

Disabling Servlet Transport

1. Open the `<API-M_HOME>/repository/conf/tomcat/catalina-server.xml` file.
2. Locate the Connector with the port 9763 as shown below:

   ```xml
   ...</Connector>
   ```

   3. Comment out the `http` connector section.

The Server needs to be restarted for these changes to be effective.

Changing the Default Ports with Offset

When you run multiple WSO2 products/clusters or multiple instances of the same product on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts. An offset defines the number by which all ports in the runtime (e.g., HTTP/S ports) will be increased. For example, if the default HTTP port is 9763 and the offset is 1, the effective HTTP port will change to 9764. For each additional WSO2 product instance, you set the port offset to a unique value. The offset of the default port is considered to be 0.

There are two ways to set an offset to a port:

- Pass the port offset to the server during startup. The following command starts the server with the default port incremented by 3: `./wso2server.sh -DportOffset=3`
- Set the offset in the Ports section of `<PRODUCT_HOME>/repository/conf/carbon.xml`. E.g., `<offset>3</offset>`
When you offset the server's port, it automatically changes all ports it uses. However, you are also able to manually adjust the ports for the Thrift client and Thrift server if needed.

Changing the Thrift client and server ports

The port offset specified earlier in the `carbon.xml` file affects the ports of the Thrift client and server as well (the default port is 10397). However, since Thrift is run as a separate server within WSO2 servers, it is possible to adjust the ports manually in the `<APIM_HOME>/repository/conf/api-manager.xml` file. By default, the `<ThriftClientPort>` and `<ThriftServerPort>` elements are commented out. If you want to adjust those ports manually, first uncomment the elements and change the Thrift ports separately. For example,

```
<KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
<ThriftClientPort>10399</ThriftClientPort>
<ThriftClientConnectionTimeOut>10000</ThriftClientConnectionTimeOut>
<ThriftServerPort>10399</ThriftServerPort>
<ThriftServerHost>localhost</ThriftServerHost>
<EnableThriftServer>true</EnableThriftServer>
```

If you specify the Thrift client and server ports manually, the port offset specified in the `carbon.xml` file has no effect on those two ports and the value that is set manually is used instead.

When you run multiple instances of the API Manager in distributed mode, the Gateway and Key Manager (used for validation and authentication) can run on two different JVMs. When the API Gateway receives API invocation calls, it contacts the API Key Manager service for verification (given that caching is not enabled at the Gateway level). Communication between API Gateway and Key Manager happens in either of the following ways:

- Through a Web service call
- Through a Thrift call

The default communication mode is using Thrift. Assume that the Gateway port is offset by 2, Key Manager port by 5 and the default Thrift port is 10397. If the Thrift ports are changed by the offsets of Gateway and Key Manager, the Thrift client port (Gateway) will now be 10399 while the Thrift server port (Key Manager) will change to 10402. This causes communication between the Gateway and Key Manager to fail because the Thrift client and server ports are different.

To fix this, you must change the Thrift client and server ports of the Gateway and Key Manager to the same value. In this case, the difference between the two offsets is 3, so you can either increase the default Thrift client port by 3 or else reduce the Thrift server port by 3.

Changing the offset of the Workflow Callback Service

The API Manager has a Service which listens for workflow callbacks. This service configuration can be found at `<APIM_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/WorkflowCallbackService.xml`. Open this file and change the port value of the `<address uri>` accordingly.

For example,

```
<address uri="https://localhost:9445/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag" format="rest"/>
```

For a list of all default ports opened in WSO2 API Manager, see Default Product Ports.

Error Handling

When errors/exceptions occur in the system, the API Manager throws XML-based error responses to the client by default. To change the format of these error responses, you change the relevant XML file in the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory. The directory includes multiple XML files, named after the type of errors that occur. You must select the correct file.

For example, to change the message type of authorization errors, open the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/_auth_failure_handler.xml` file and change the `application/xml` to something like `application/json`.

```
<sequence name="_auth_failure_handler_" xmlns="http://ws.apache.org/ns/synapse">
  <property name="error_message_type" value="application/json"/>
  <sequence key="_cors_request_handler_"/>
</sequence>
```

Similarly, to change the error messages of throttling errors (e.g., quota exceeding), change the `_throttle_out_handler_.xml` file; resource mismatch errors, the `_resource_mismatch_handler_.xml` file, etc.
Given below are some error codes and their meanings.

### API handlers error codes

<table>
<thead>
<tr>
<th>Error code</th>
<th>Error Message</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>700700</td>
<td>API blocked</td>
<td>This API has been blocked temporarily. Please try again later or contact the system administrators.</td>
<td>Invoke an API which is in the <strong>BLOCKED</strong> lifecycle state</td>
</tr>
<tr>
<td>900800</td>
<td>Message throttled out</td>
<td>The maximum number of requests that can be made to the API within a designated time period is reached and the API is throttled for the user.</td>
<td>Invoke an API exceeding the tier limit</td>
</tr>
<tr>
<td>900801</td>
<td>Hard limit exceeded</td>
<td>Hard throttle limit has been reached</td>
<td>Invoke an API exceeding the hard throttle limit</td>
</tr>
<tr>
<td>900802</td>
<td>Resource level throttle out</td>
<td>Message is throttled out because resource level has exceeded</td>
<td>Sending/Receiving messages beyond authorized resource level</td>
</tr>
<tr>
<td>900803</td>
<td>Application level throttle out</td>
<td>Message is throttled out because application level is exceeded</td>
<td>Sending/Receiving messages beyond authorized application level</td>
</tr>
<tr>
<td>900804</td>
<td>Subscription level throttle out</td>
<td>Message is throttled out due to subscription level throttling limit reached.</td>
<td>Sending/Receiving messages beyond configured throttling limit of subscription level policy.</td>
</tr>
<tr>
<td>900805</td>
<td>Message blocked</td>
<td>Accessing an API which is blocked on user, IP, application, or API Context.</td>
<td>An admin user can block API invocations in real time by user, IP, application, or API context. The API invocation meets the blocked condition.</td>
</tr>
<tr>
<td>900806</td>
<td>Custom policy throttled out</td>
<td>Message throttled out due to exceeding the limit configured through the custom throttling policy rules.</td>
<td>The API invocations meet custom throttle policy rules, exceeding the limits of the configured custom policy.</td>
</tr>
<tr>
<td>900807</td>
<td>Message throttled out</td>
<td>Message throttled out because of exceeding the burst control/rate limit (requests per second) in the subscription level policy.</td>
<td>Sending/Receiving messages exceeding the configured burst control/rate limit within second.</td>
</tr>
<tr>
<td>900900</td>
<td>Unclassified authentication failure</td>
<td>An unspecified error has occurred</td>
<td>Backend service for key validation is not accessible when trying to invoke an API</td>
</tr>
<tr>
<td>900901</td>
<td>Invalid credentials</td>
<td>Invalid authentication information provided</td>
<td>Using an older access token after an access token has been renewed.</td>
</tr>
<tr>
<td>900902</td>
<td>Missing credentials</td>
<td>No authentication information provided</td>
<td>Accessing an API without <strong>Authorization</strong>: Bearer header</td>
</tr>
<tr>
<td>900905</td>
<td>Incorrect access token type is provided</td>
<td>The access token type used is not supported when invoking the API. The supported access token types are application and user accesses tokens. See Access Tokens.</td>
<td>Invoke an API with application token, where the resource only allows application user tokens</td>
</tr>
<tr>
<td>900906</td>
<td>No matching resource found in the API for the given request</td>
<td>A resource with the name in the request can not be found in the API.</td>
<td>Invoke an API resource that is not available</td>
</tr>
<tr>
<td>900907</td>
<td>The requested API is temporarily blocked</td>
<td>Happens when the API is blocked.</td>
<td>Invoke API resource with a subscription that has been blocked by the API publisher</td>
</tr>
<tr>
<td>900908</td>
<td>Resource forbidden</td>
<td>The user invoking the API has not been granted access to the required resource.</td>
<td>Invoke an unsubscribed API</td>
</tr>
<tr>
<td>900909</td>
<td>The subscription to the API is inactive</td>
<td>The status of the API has changed to an inaccessible/unavailable state.</td>
<td>Invoke an API resource with a subscription that has not yet been approved by the administrator.</td>
</tr>
<tr>
<td>900910</td>
<td>The access token does not allow you to access the requested resource</td>
<td>Can not access the required resource with the provided access token. Check the valid resources that can be accessed with this token.</td>
<td>Invoke API resource with an access token that is not generated to be used with the resource’s scope.</td>
</tr>
</tbody>
</table>
The payload sent with the request is too large and the client is unable to keep the connection alive until the payload is completely transferred to the API Gateway. Sending a large PDF file with the POST request.

The error codes 900903 (Access token expired) and 900904 (Access token inactive) are deprecated from API Manager 1.9.0 onwards. Alternatively, error code 900901 will be sent when the token is invalid or inactive.

### Sequences error codes

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>900901</td>
<td>Production/sandbox key offered to the API with no production/sandbox endpoint</td>
</tr>
<tr>
<td>400</td>
<td>Server cannot process the request due to an error in the request sent by the client</td>
</tr>
<tr>
<td>403</td>
<td>No matching resource found in the API for the given request</td>
</tr>
</tbody>
</table>

In addition to the above error codes, we have engaged Synapse-level error codes to the default fault sequence and custom fault sequences (e.g., `_token_.fault_.xml`) of the API Manager. For information, see Error Handling in WSO2 Enterprise Integrator (WSO2 EI) documentation.

The HTTP Status Codes and the corresponding error codes from the error responses are given below.

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>900901, 900902, 900905, 900907, 900909</td>
</tr>
<tr>
<td>403</td>
<td>900906, 900908, 900910</td>
</tr>
<tr>
<td>429</td>
<td>900800</td>
</tr>
<tr>
<td>500</td>
<td>900900</td>
</tr>
<tr>
<td>503</td>
<td>700700, 900801</td>
</tr>
</tbody>
</table>

### Transport error codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>101000</td>
<td>Receiver input/output error sending</td>
</tr>
<tr>
<td>101001</td>
<td>Receiver input/output error receiving</td>
</tr>
<tr>
<td>101500</td>
<td>Sender input/output error sending</td>
</tr>
<tr>
<td>101501</td>
<td>Sender input/output error receiving</td>
</tr>
<tr>
<td>101503</td>
<td>Connection failed</td>
</tr>
<tr>
<td>101504</td>
<td>Connection timed out (no input was detected on this connection over the maximum period of inactivity)</td>
</tr>
<tr>
<td>101505</td>
<td>Connection closed</td>
</tr>
<tr>
<td>101506</td>
<td>NHTTP protocol violation</td>
</tr>
<tr>
<td>101507</td>
<td>Connection canceled</td>
</tr>
<tr>
<td>101508</td>
<td>Request to establish new connection timed out</td>
</tr>
<tr>
<td>101509</td>
<td>Send abort</td>
</tr>
<tr>
<td>101510</td>
<td>Response processing failed</td>
</tr>
</tbody>
</table>

If the HTTP PassThrough transport is used, and a connection-level error occurs, the error code is calculated using the following equation:

```
Error code = Base error code + Protocol State
```
There is a state machine in the transport sender side, where the protocol state changes according to the phase of the message.

Following are the possible protocol states and the description for each:

<table>
<thead>
<tr>
<th>Protocol State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_READY(0)</td>
<td>Connection is at the initial stage ready to send a request</td>
</tr>
<tr>
<td>REQUEST_HEAD(1)</td>
<td>Sending the request headers through the connection</td>
</tr>
<tr>
<td>REQUEST_BODY(2)</td>
<td>Sending the request body</td>
</tr>
<tr>
<td>REQUEST_DONE(3)</td>
<td>Request is completely sent</td>
</tr>
<tr>
<td>RESPONSE_HEAD(4)</td>
<td>The connection is reading the response headers</td>
</tr>
<tr>
<td>RESPONSE_BODY(5)</td>
<td>The connection is reading the response body</td>
</tr>
<tr>
<td>RESPONSE_DONE(6)</td>
<td>The response is completed</td>
</tr>
<tr>
<td>CLOSING(7)</td>
<td>The connection is closing</td>
</tr>
<tr>
<td>CLOSED(8)</td>
<td>The connection is closed</td>
</tr>
</tbody>
</table>

Since there are several possible protocol states in which a request can time out, you can calculate the error code accordingly using the values in the table above. For example, in a scenario where you send a request and the request is completely sent to the backend, but a timeout happens before the response headers are received, the error code is calculated as follows:

In this scenario, the base error code is CONNECTION_TIMEOUT(101504) and the protocol state is REQUEST_DONE(3).

Therefore,

Error code = 101504 + 3 = 101507

These Transport error codes are used in Advanced Configurations of Endpoints.

**Custom error messages**

To send a custom message with a custom HTTP status code, you execute an additional sequence that can generate a new error message. You then override the message body, HTTP status code and other values.

The following steps demonstrate how to override a throttled-out message's HTTP status code as a custom error message:

1. Start the WSO2 API Manager.
2. Go to `<API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory and create the file `convert.xml` as follows.
3. Check the terminal logs to see whether there are issues in the deployment. If the deployment is successful, you see a message similar to the following in the system logs:

```
INFO - DependencyTracker Sequence : convert was added to the Synapse configuration successfully
INFO - SequenceDeployer Sequence named 'convert' has been deployed from file: <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/convert.xml
```

Alternatively, you can use the **Source View** of the API-M Management Console as follows to edit the synapse configuration:

- Sign in to the Management Console. ([https://<Server Host>:9443/carbon](https://<Server Host>:9443/carbon)).
- Go to **Manager -> Source View**.
- Copy the content of the sequence in **convert.xml**, paste it as a new sequence in the source view and update it.

4. Include the sequence that you just deployed in a sequence of your choice. For this example, let's add this custom sequence in the **_auth_failure_handler_sequence**.

```
<sequence name="_auth_failure_handler_" xmlns="http://ws.apache.org/ns/synapse">
  ...
  <sequence key="convert"/>
  <drop/>
</sequence>
```

5. Check the terminal and see whether there are any errors with the **_auth_failure_handler_** sequence deployment. If the deployment is successful, you see a message similar to the following in the system logs:

```
INFO - DependencyTracker Sequence : _auth_failure_handler_ was added to the Synapse configuration successfully
INFO - SequenceDeployer Sequence: _auth_failure_handler_ has been updated from the file: <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/_auth_failure_handler_.xml
```
6. Invoke the API with the respective criteria in order to trigger the sequence.
   In this example, let’s view the menu on the PizzaShack API and invoke the API with an incorrect token.

   **Sample Response**

   ```
curl -v -H "Authorization: Bearer <Access_Token>"
http://localhost:<API_name>/<version>/<context>
```

   ```
```

```
> GET /pizzashack/1.0.0/menu HTTP/1.1
> Host: localhost:8243
> User-Agent: curl/7.54.0
> accept: application/json
> Authorization: Bearer fb119e84-9542-3194-93dc-1dddaa1111
> ...
> HTTP/1.1 555
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: GET
< Access-Control-Allow-Headers: authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction
< Content-Type: application/json; charset=UTF-8
< Date: Fri, 04 Jan 2019 09:53:56 GMT
< Transfer-Encoding: chunked
<
{"fault":{"code":900901,"type":"Status report","message":"Runtime Error","description":"Invalid Credentials"}}
```

WSO2 API Manager has the following default fault sequences located in `<API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory.

<table>
<thead>
<tr>
<th>Fault Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fault.xml</td>
<td>This is the primary fault sequence that gets invoked when an error occurs during the execution of an API resources</td>
</tr>
<tr>
<td>main.xml</td>
<td>This sequence is called when the endpoint being called does not exist</td>
</tr>
<tr>
<td>_auth_failure_handler.xml</td>
<td>This sequence is called when an API authentication error is encountered</td>
</tr>
<tr>
<td>_production_key_error.xml</td>
<td>This sequence is called when a Production key is used to invoke an API that does not have a Production endpoint defined</td>
</tr>
<tr>
<td>_sandbox_key_error.xml</td>
<td>This sequence is called when a Sandbox key is used to invoke an API that does not have a Sandbox endpoint defined</td>
</tr>
<tr>
<td>_throttle_out_handler.xml</td>
<td>This sequence is called when a given request to an API gets throttled out</td>
</tr>
<tr>
<td>_token_fault.xml</td>
<td>This sequence is called when there is an error in invoking the token API</td>
</tr>
<tr>
<td>_resource_mismatch_handler.xml</td>
<td>This sequence is called when a matching resource cannot be found by the gateway to the corresponding resource being invoked</td>
</tr>
<tr>
<td><em>cors_request_handler</em>.xml</td>
<td>This sequence enables sending CORS specific headers when the CORS specific configuration (CORSConfig uration) is enabled in WSO2 API Manager in the <code>&lt;API-M_HOME&gt;/repository/conf/api-manager.xml</code> file.</td>
</tr>
<tr>
<td><em>threat_fault</em>.xml</td>
<td>This sequence is called to send error messages with regard to threat detection.</td>
</tr>
<tr>
<td>dispatchSeq.xml</td>
<td>This sequence is defined as a default handler for any inbound WebSocket calls.</td>
</tr>
<tr>
<td>outDispatchSeq.xml</td>
<td>This sequence is defined to handle any outbound WebSocket calls.</td>
</tr>
</tbody>
</table>

The default sequences can also be customized as shown in the section above.
Message Flow in the API Manager Gateway

The Gateway of an API Manager deployment is responsible for the main business functionality of serving API traffic. The following diagram illustrates the message flow in the Gateway at a very high level.

- The handlers
- Mediation extensions
- In sequence and out sequence

The handlers

The handlers are request and response interceptors. The list of API handlers in WSO2 API-M are as follows:

- APIMgtLatencyStatsHandler
- CORSRequestHandler
- APIAuthenticationHandler
- ThrottleHandler
- APIMgmtUsageHandler
- APIMgtGoogleAnalyticsTrackingHandler
- APIManagerExtensionHandler

Each handler performs a specific task as mentioned in the table below. Note that some handlers are functional both at the inflow and outflow of messages.

<table>
<thead>
<tr>
<th>Handler</th>
<th>Inflow</th>
<th>Outflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIMgtLatencyStatsHandler</td>
<td>Publish request and response latencies, if analytics is enabled</td>
<td>Publish request and response latencies, if analytics is enabled</td>
</tr>
<tr>
<td>CORSRequestHandler</td>
<td>Set CORS Headers</td>
<td>Set CORS Headers</td>
</tr>
<tr>
<td>APIAuthenticationHandler</td>
<td>Request authentication</td>
<td>N/A</td>
</tr>
<tr>
<td>ThrottleHandler</td>
<td>Request throttling</td>
<td>N/A</td>
</tr>
<tr>
<td>APIMgtUsageHandler</td>
<td>Publish request data, if analytics is enabled</td>
<td>N/A</td>
</tr>
<tr>
<td>APIMgtGoogleAnalyticsTrackingHandler</td>
<td>Publish data to Google Analytics, if Google Analytics is configured</td>
<td>N/A</td>
</tr>
<tr>
<td>APIManagerExtensionHandler</td>
<td>Execute custom mediation sequences at inflow</td>
<td>Execute custom mediation sequences at outflow</td>
</tr>
</tbody>
</table>

Mediation extensions

Mediation extensions are the custom mediation logic that can be executed in either the inflow or the outflow. For more details on how to configure mediation extensions, see Adding Mediation Extensions.

In sequence and out sequence

The in sequence and out sequence carry the main business logic of the request flow. The in sequence handles sending the request from the client to the backend, while the out sequence routes the response sent from the backed to the client.

Updating WSO2 API Manager

WSO2 introduces the WSO2 Update Manager (WUM), which is a command-line utility that allows you to get the latest updates that are available for a particular product release. These updates include the latest bug fixes and security fixes that are released by WSO2 after a particular product version is released. Therefore, you do not need to wait and upgrade to the next product release to get these bug fixes. For information on updating WSO2 API Manager with the latest available patches (issued by WSO2) using the WUM tool, see Getting Started with WUM in the WSO2 Updates Guide.

Persisting Index data

The indexing related information of WSO2 API Manager is stored in the `<API-M_HOME>/solr/data` directory. Once the data is indexed, it is stored in the index directory. Refer Add Apache Solr-Based Indexing for more information.

Before you discard the old API Manager instance,
Refer [Updating WSO2 API Manager Analytics](#) in order to update WSO2 API Manager Analytics binary distribution.

**Accessing API Manager by Multiple Devices Simultaneously**

When there are many users who use production deployment setups, accessing API Manager by multiple devices is more important. According to the architecture, if we logged out from one device and revoke the access token, then all the calls made with that token thereafter will get authentication failures.

In this case Applications should be smart enough to detect that authentication failure and should request for a new access token.

**Issue in having multiple access tokens**

Once user log in to the application, the user may need to provide username and password. We can use that information (username and passowrd) and consumer key, consumer secret pair to receive new token once the authentication failure is detected. We should handle this from client application side. If we allowed users to have multiple tokens at the same time, that will cause security related issues and finally users will end up with thousands of tokens that the user cannot even maintain. And also those affects to the usage of metering and statistics.

**Recommended Solution**

The recommended solution for this issue is having only one active user token at a given time. We need to make the client application aware about error responses sent from the API Manager Gateway. And use the refresh token Approach. When you request a user token you will get refresh token along with the token response, so that you can use that for refreshing the access token.

**How this should work**

Let's assume that same user is logged in to WSO2 API Manager from desktop and tablet. At that time client should provide username and password both when they are login into desktop and tablet apps. Then you can generate token request with the username - password and consumer key - consumer secret pair. So this request will be kept in memory until the user close or logout from application (We do not persist this data to anywhere so that there is no security issue.).

Then, when the user log out from the desktop or the application on the desktop, it decides to refresh the OAuth token first, and the user will be prompted to enter their username and password on the tablet, since the tablet has revoked or inactivated the OAuth token. But in here, you should not prompt username and password because the client is already provided them and you have the token request in memory. Once we detect authentication failure from the tablet, it will immediately send token generation request and get new token. Hence, the user will not aware about what happen underneath.

**Developer Guide**

This section provides instructions and information regarding the various developer functions of WSO2 API Manager for users. This may include methods of working with the available administration and user features available in the API Manager. The topics in this section are listed as follows:

- Working with the Source Code
- Java Documentation
- WSO2 Admin Services
- Product APIs
- Working with Audit Logs
- Enabling Authentication Session Persistence
- Using the Registry REST API
Working with the Source Code

The source code of all WSO2 products as well as the scripts that are used for building WSO2 products are maintained in GitHub repositories. If you are a developer, you can easily clone the source code from these Git repositories, and if required, you can do modifications and build a customized product on your own. For more information, see the following:

- WSO2 GitHub Repositories
- Using Maven to Build WSO2 Products
- Contributing to the Code Base

Java Documentation

The following Java documentation describes all the classes, interfaces, and methods of the API Manager, which you can use to create custom classes: http://product-dist.wso2.com/javadocs/api-manager/2.1.0/

WSO2 Admin Services

WSO2 products are managed internally using SOAP Web services known as admin services. WSO2 products come with a management console UI, which communicates with these admin services to facilitate administration capabilities through the UI.

A service in WSO2 products is defined by the following components:

- Service component: provides the actual service
- UI component: provides the Web user interface to the service
- Service stub: provides the interface to invoke the service generated from the service WSDL

There can be instances where you want to call back-end Web services directly. For example, in test automation, to minimize the overhead of having to change automation scripts whenever a UI change happens, developers prefer to call the underlying services in scripts. The topics below explain how to discover and invoke these services from your applications.

Discovering the admin services

By default, the WSDLs of admin services are hidden from consumers. Given below is how to discover them using the OSGi console.

1. Set the <HideAdminServiceWSDLs> element to false in the <PRODUCT_HOME>/repository/conf/carbon.xml file.
2. Go to <PRODUCT_HOME>/bin/ folder and start the WSO2 product as follows:

   - **In Linux Environment**
     ```bash
     sh wso2server.sh -DosgiConsole
     ```

   - **In Windows Environment**
     ```cmd
     wso2server.bat -DosgiConsole
     ```

3. When the server is started, hit the enter/return key several times to get the OSGi shell in the console.
4. In the OSGi shell, type: osgi> listAdminServices
5. The list of admin services of your product are listed. For example:
6. To see the service contract of an admin service, select the admin service’s URL and then paste it in your browser with ?wsdl at the end. For example:
https://localhost:9443/services/RemoteUserStoreManagerService?wsdl

7. Note that the admin service’s URL appears as follows in the list you discovered in step 6:


Invoking an admin service

Admin services are secured using common types of security protocols such as HTTP basic authentication, WS-Security username token, and session based authentication to prevent anonymous invocations. For example, the UserAdmin Web service is secured with the HTTP basic authentication. To invoke a service, you do the following:

1. Authenticate yourself and get the session cookie.
2. Generate the client stubs to access the back-end Web services.

To generate the stubs, you can write your own client program using the Axis2 client API or use an existing tool like SoapUI (4.5.1 or later) or wsdl2java.

The wsdl2java tool, which comes with WSO2 products by default hides all the complexity and presents you with a proxy to the back-end service. The stub generation happens during the project build process within the Maven POM files. It uses the Maven ant run plug-in to execute the wsdl2java tool.

You can also use the Java client program given here to invoke admin services. All dependency JAR files that you need to run this client are found in the /lib directory.

Authenticate the user

The example code below authenticates the user and gets the session cookie:
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
import org.wso2.carbon.authenticator.stub.AuthenticationAdminStub;
import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.apache.axis2.context.ServiceContext;
import java.rmi.RemoteException;

public class LoginAdminServiceClient {
    private final String serviceName = "AuthenticationAdmin";
    private AuthenticationAdminStub authenticationAdminStub;
    private String endPoint;

    public LoginAdminServiceClient(String backEndUrl) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        authenticationAdminStub = new AuthenticationAdminStub(endPoint);
    }

    public String authenticate(String userName, String password) throws RemoteException,
    LoginAuthenticationExceptionException {
        String sessionCookie = null;
        if (authenticationAdminStub.login(userName, password, "localhost")) {
            System.out.println("Login Successful");
            ServiceContext serviceContext = authenticationAdminStub.
            _getServiceClient().getLastOperationContext().getServiceContext();
            sessionCookie = (String) serviceContext.getProperty(HTTPConstants.COOKIE_STRING);
            System.out.println(sessionCookie);
        }
        return sessionCookie;
    }

    public void logOut() throws RemoteException, LogoutAuthenticationExceptionException {
        authenticationAdminStub.logout();
    }
}

Generate the client stubs

After authenticating the user, give the retrieved admin cookie with the service endpoint URL as shown in the sample below. The Remote user
management service name is RemoteUserStoreManagerService. You can find its URL (e.g., https://localhost:9443/services/RemoteU
serStoreManagerService) in the service.xml file in the META-INF folder in the respective bundle that you find in <PRODUCT_HOME>/rep
ository/components/plugins.
import org.apache.axis2.AxisFault;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.wso2.carbon.um.ws.api.stub.RemoteUserStoreManagerServiceStub;
import org.wso2.carbon.um.ws.api.stub.RemoteUserStoreManagerServiceUserStoreExceptionException;
import java.rmi.RemoteException;

public class RemoteUserStoreServiceAdminClient {
    private final String serviceName = "RemoteUserStoreManagerService";
    private RemoteUserStoreManagerServiceStub remoteUserStoreManagerServiceStub;
    private String endPoint;

    public RemoteUserStoreServiceAdminClient(String backEndUrl, String sessionCookie) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        remoteUserStoreManagerServiceStub = new RemoteUserStoreManagerServiceStub(endPoint);
        //Authenticate Your stub from sessionCooke
        ServiceClient serviceClient;
        Options option;
        serviceClient = remoteUserStoreManagerServiceStub._getServiceClient();
        option = serviceClient.getOptions();
        option.setManageSession(true);
        option.setProperty(org.apache.axis2.transport.http.HTTPConstants.COOKIE_STRING,
                sessionCookie);
    }

    public String[] listUsers() throws RemoteException, RemoteUserStoreManagerServiceUserStoreExceptionException {
        return remoteUserStoreManagerServiceStub.listUsers("*", 100);
    }
}

The following sample code lists the users in the APIM server:
import org.apache.axis2.AxisFault;
import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.wso2.carbon.um.ws.api.stub.RemoteUserStoreManagerServiceUserStoreExceptionException;
import java.rmi.RemoteException;

public class AdminServiceClientManager {
    public static void main (String[] args) {
        System.setProperty("javax.net.ssl.trustStore",
                "<API-M_HOME>/repository/resources/security/client-truststore.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "wso2carbon");
        try {
            LoginAdminServiceClient loginAdminServiceClient =
                    new LoginAdminServiceClient("https://localhost:9443");
            String sessionId = loginAdminServiceClient.authenticate("admin", "admin");
            RemoteUserStoreServiceAdminClient remoteUserStoreServiceAdminClient = new
                    RemoteUserStoreServiceAdminClient("https://localhost:9443", sessionId);
            String[] users = remoteUserStoreServiceAdminClient.listUsers();
            if(users != null){
                System.out.println("Listing user names of Carbon server...... ");
                for(String user : users){
                    System.out.println("User Name : " + user);
                }
            }
            loginAdminServiceClient.logOut();
        } catch (AxisFault axisFault) {
            axisFault.printStackTrace();
            throw new RuntimeException(axisFault);
        } catch (RemoteException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (LoginAuthenticationExceptionException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (RemoteUserStoreManagerServiceUserStoreExceptionException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (LogoutAuthenticationExceptionException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }
}

The complete maven project can be found at: org.wso2.carbon.sample.admin.service.invoker.zip

Product APIs

The following topics consist of a list of RESTful APIs, Token APIs, and deprecated APIs exposed by WSO2 API Manager.

- **RESTful APIs**
- **Token API**
- **Deprecated APIs**

  - **RESTful APIs** - These are the APIs exposed from the API Publisher, API Store, and Admin Portal, which can be used to create and manage APIs.
  - **Token API** - These APIs can be used to generate and renew user and application access tokens.
  - **Deprecated APIs** - These APIs are deprecated and will be unsupported in a future release.

**RESTful APIs**
The following topics list the APIs exposed from the API Publisher, API Store, and Admin Portal, which you can use to create and manage APIs. You can consume APIs directly through their UIs or, an external REST client like cURL, or the WSO2 REST client.

- Publisher APIs
- Store APIs
- Admin APIs

**Token API**

Users need access tokens to invoke APIs subscribed under an application. Access tokens are passed in the HTTP header when invoking APIs. The API Manager provides a Token API that you can use to generate and renew user and application access tokens. The response of the Token API is a JSON message. You extract the token from the JSON and pass it with an HTTP Authorization header to access the API.

The topics below explain how to generate access tokens and authorize them. WSO2 API Manager supports the following common authorization grant types and you can also define additional types.

- Kerberos OAuth2 Grant
- Refresh Token Grant
- Authorization Code Grant
- NTLM Grant
- Password Grant
- SAML Extension Grant
- Client Credentials Grant
- Implicit Grant
- JWT Grant

For more information on Token APIs, see the following topics.

- Revoking access tokens
- Configuring the token expiration time
- Token Persistence

**Revoking access tokens**

After issuing an access token, a user or an admin can revoke it in case of theft or a security violation. You can do this by calling the Revoke API using a utility like cURL. The Revoke API's endpoint URL is `http://localhost:8280/revoke`.

You can use any of the following cURL command options to revoke an access token:

**Option 1**

The parameters required to invoke the following API are as follows:

- `token` - The token to be revoked
- `<base64 encoded (clientId:clientSecret)>` - Use a base64 encoder (e.g., [https://www.base64encode.org/](https://www.base64encode.org/)) to encode your client ID and client secret using the following format: `<clientId>:<clientSecret>` Thereafter, enter the encoded value for this parameter.

```
```

```
curl -k -v -d "token=a0d210c7a3de7d548e03f1986e9a5c39" -H "Authorization: Basic OVRKKVJLZWFhVGZGeUpRSkRzam9aZmp4UkhjYTgpZnJ3ZXVeau1920TSSzFT2W1QWx1aXdVYmcd" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/revoke
```

You receive an empty response with the HTTP status as 200. The following HTTP headers are returned:

Note that if you use an invalid access token, you still receive an empty response with the HTTP status as 200 but only the following HTTP headers are returned:
Option 2

The parameters required to invoke the following API are as follows:

- **token** - The token to be revoked
- `<base64 encoded (clientId:clientSecret)>` - Use a base64 encoder (e.g., [https://www.base64encode.org/](https://www.base64encode.org/)) to encode your client ID and client secret using the following format: `<clientId>:<clientSecret>`. Thereafter, enter the encoded value for this parameter.
- **token_type_hint** - This parameter is optional. If you do not specify this parameter, then WSO2 Identity Server (WSO2 IS) will search in both key spaces (access and refresh) and if it finds a matching token then it will be revoked. Therefore, if this parameter is not specified the token revocation process takes longer. However, if you specify this parameter then WSO2 IS only searches in the respective token key space, hence the token revocation process is much faster.

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded; charset=UTF-8" -d "token=<REFRESH_TOKEN_TO_BE_REVOKED>&token_type_hint=access_token&client_id=<CLIENT_ID>" http://localhost:8243/revoke
```

Configuring the token expiration time

User access tokens have a fixed expiration time, which is set to 60 minutes by default. Before deploying the API Manager to users, extend the default expiration time by editing the `<AccessTokenDefaultValidityPeriod>` element in the `<APIM_HOME>/repository/conf/identity/identity.xml` file.

Also take the **time stamp skew** into account when configuring the expiration time. The time stamp skew is used to manage small time gaps in the system clocks of different servers. For example, let’s say you have two Key Managers and you generate a token from the first one and authenticate with the other.
If the second server's clock runs 300 seconds ahead, you can configure a 300s time stamp skew in the first server. When the first Key Manager generates a token (e.g., with the default life span, which is 3600 seconds), the time stamp skew is deducted from the token's life span. The new life span is 3300 seconds and the first server calls the second server after 3200 seconds.

You configure the time stamp skew using the `<TimestampSkew>` element in the `<APIM_HOME>/repository/conf/identity/identity.xml` file.

**Tip**: Ideally, the time stamp skew should not be larger than the token's life span. We recommend you to set it to zero if the nodes in your cluster are synchronized.

When a user access token expires, the user can try regenerating the token.

**Token Persistence**

The OAuth2 component in WSO2 API Manager (WSO2 API-M) has two implementations that you can use to handle token persistence in the database, which are namely synchronous and asynchronous token persistence. The following sections guide you through the difference between these two approaches and how to configure them in a production environment.

- Synchronous token persistence (When PoolSize = 0)
- Asynchronous token persistence (When PoolSize > 0)

The Synchronous or Asynchronous behavior is governed by the `PoolSize` property under `SessionDataPersist` element in the `identity.xml` file.

**Synchronous token persistence (When PoolSize = 0)**

The flow of synchronous token persistence is as follows:

1. The client sends an access token request.
2. The OAuth2 component in WSO2 API-M checks for an existing active access token for the given client/user/scope. It first checks the cache and if an active token is not found, it then checks the database.
3. If an active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component creates a new access token and persists it in the database using the same thread. Once it is persisted, the new token is returned to the client.

**Enabling synchronous token persistence**

To enable synchronous token persistence, follow the steps in the Enabling Authentication Session Persistence tutorial and set the `<poolSize>` property to 0.

**Asynchronous token persistence (When PoolSize > 0)**
The flow of asynchronous token persistence is as follows:

1. The client sends an access token request.
2. The OAuth2 component in WSO2 API-M checks for an existing active access token for the given client/user/scope. It first checks the cache and if an active token is not found, it then checks the database.
3. If an active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component creates a new access token and adds it to a persisting queue. Once the token is added to the queue, the token is returned to the client.
5. Thereafter, there are background threads that consume the above queue and they in-turn will persist these tokens to the DB.

Enabling asynchronous token persistence

To enable asynchronous token persistence, follow the steps in the Enabling Authentication Session Persistence tutorial and set the `<poolsize>` property to a value higher than 0. The value provided for the `<poolsize>` property determines the number of threads in the thread pool that are used to consume the token persisting queue.

Recovery flow for token persistence

This section explains the recovery flow triggered in WSO2 API Manager for exceptional cases that may occur in a production environment caused by the client application mishandling the `CON_APP_KEY` constraint that is explained below.

- `CON_APP_KEY` constraint
- Asynchronous token persistence
- Synchronous token persistence

`CON_APP_KEY` constraint

`CONSTRAINT CON_APP_KEY UNIQUE (CONSUMER_KEY, AUTHZ_USER, USER_TYPE, TOKEN_STATE, TOKEN_STATE_ID, TOKEN_SCOPE)`

As seen in the code snippet above for a given set of consumer key, user, and scope values, there can be only one `ACTIVE` access token. The `CON_APP_KEY` constraint in the `IDN_OAUTH2_ACCESS_TOKEN` table enforces this by allowing only one active access token for a given set of consumer key, user, and scope values. This constraint may be violated in a scenario where two or more identical token requests come from the same application.

The above scenario is unlikely, because in practice an application is usually designed to handle this situation using scopes, or in the case of a multi-threaded client, there is usually a separate thread to acquire access tokens so that other threads can retrieve from it.

Asynchronous token persistence

Flow

For instance, if the violation mentioned above occurs with two nodes of a cluster receiving identical access token requests, the flow of the asynchronous token persistence is as follows:

1. The client sends an access token request.
2. The OAuth2 component in both nodes of WSO2 API-M checks for an existing active access token for the given client/user/scope. Both nodes first check the cache and if an active token is not found, the it checks the database.
3. If an existing active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component in both nodes creates a new access token and adds it to the persisting queue. After adding it to the queue, the access token is returned to the client.
5. However, the background threads that consume the persisting queue in both servers (nodes) attempt to persist the token to the database. One of the servers succeed and successfully persist the access token to the database, and the other server receives an error due to violation of the `CON_APP_KEY` constraint. The violation is due to the fact that the same access token was already persisted by the first server in the cluster and is currently active.
Recovery flow

To handle this situation, WSO2 API Manager has a recovery flow for token persistence that does the following:

- As both access tokens were returned to the client, there must be a database entry for both tokens.
- As the access token that the second node is attempting to persist is not allowed due to the violation, the recovery flow takes the latest entry in the database, which is the active access token persisted by the first node, and marks it as INACTIVE.
- The access token that is received from the second node is now saved as an ACTIVE access token in the database. Therefore, one of the access tokens returned to the client is an INACTIVE token.

Tip: If the client application is not designed to handle the CON_APP_KEY constraint violation using scopes, you can avoid the situation described above and avoid any invalid tokens by using synchronous token persistence. To do this, set the <PoolSize> property in the <API-M _HOME>/repository/conf/identity/identity.xml file to 0.

Synchronous token persistence

Flow

The flow of the synchronous token persistence when receiving two identical access token requests is as follows:

1. The client sends an access token request.
2. The OAuth2 component in both nodes of WSO2 API-M checks for an existing active access token for the given client/user/scope. Both nodes first check the cache and if an active token is not found, the database is checked.
3. If an existing active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component in both nodes creates a new access token and persists the access token to the database using the same thread.
5. Either one of the nodes persist the token successfully and returns it to the client, but the other node receives an error due to violation of the CON_APP_KEY constraint.

Recovery flow

The process flow now moves on to the recovery flow described above in order to handle the CON_APP_KEY constraint violation and is executed as follows:

- As the same thread is being used, the OAuth2 component in the second node checks the database again for an ACTIVE access token.
- As there is now an ACTIVE token, which was persisted by the first node, the second node now returns the access token persisted by the first node to the client.
- Both access token requests receive the same access token.

Kerberos OAuth2 Grant

Kerberos is a security protocol that has support built into various operating systems and open-source distributions (e.g., Ubuntu, Windows, RedHat, Open Solaris, etc). In addition, a majority of browsers support some Kerberos functions as well. As WSO2 API Manager uses the OAuth 2.0 protocol, the Kerberos OAuth2 grant type allows organizations to exchange a Kerberos ticket for an OAuth2 token. Thereby, allowing organizations to re-use their existing Kerberos infrastructure, while easier adopting OAuth 2.0 within these organizations.

- Kerberos OAuth2 grant flow
- Configuring Kerberos Grant with API Manager

Kerberos OAuth2 grant flow

The following section describes the flow involved in exchanging a Kerberos ticket for an OAuth2 token.
1. The Kerberos client requests the Kerberos Service Ticket from the Kerberos Key Distribution Center (KDC) to invoke the service. The Kerberos Key Distribution Center can be any Kerberos Server.

2. The Kerberos Key Distribution Center sends a response with the Kerberos Service Ticket. If the client and the requested service is valid, the Key Distribution Center (KDC) sends a Kerberos ticket encrypted with the service owners private key. The API handles the exchanging of the Ticket Granting Ticket (TGT), Service Granting Ticket (SGT), and all other low level Kerberos details.

3. The Kerberos client requests the OAuth2 token. The message format of the OAuth2 token request should be as follows:

You can use one of the following two cURL commands to request for the OAuth2 token.

```bash
```

```bash
```

The "scope=my_scope" is an optional parameter that you can add to the string in the token request body.
Example

```
grant_type=kerberos&scope=my_scope&kerberos_realm=example.com&kerberos_token=YII1...
```

Example

```
POST /token HTTP/1.1
Host: idp.example.com:8243
Content-Type: application/x-www-form-urlencoded
Authorization: Basic MW91TDJmTzZTeGxmRDJMRhcxMjVjVG8wd1FrYTps1VUV0bTg5dFk2UVp1W1VtcVpmTDkyQkRGZUFh
grant_type=kerberos&scope=my_scope&kerberos_realm=example.com&kerberos_token=YII1...
```

4. The Kerberos client receives the OAuth2 token. The Kerberos Grant validates the received token with the provided Identity Provider (IDP) credentials and if it is a valid token, it issues an OAuth2 token to the client.

```
Example

```
{
  "access_token": "636ce45f-c7f6-3a95-907f-d1f8aca28403",
  "refresh_token": "831271d9-16ba-3bad-af18-b9f6592a8671",
  "scope": "my_scope",
  "token_type": "Bearer",
  "expires_in": 521
}
```

Configuring Kerberos Grant with API Manager

Follow the instructions below to configure Kerberos Grant with WSO2 API Manager:

Now API Manager comes with kerberos_grant_1.0.0_1.0.0.jar. Therefore you do not need to add this kerberos grant jar manually.

1. Add following entry under `<SupportedGrantTypes>` in the `<API-M_HOME>/repository/conf/identity/identity.xml` file.

```
<SupportedGrantType>
  <GrantTypeName>kerberos</GrantTypeName>
  <GrantTypeHandlerImplClass>org.wso2.carbon.identity.oauth2.grant.kerberos.ExtendedKerberosGrant</GrantTypeHandlerImplClass>
  <GrantTypeValidatorImplClass>org.wso2.carbon.identity.oauth2.grant.kerberos.KerberosGrantValidator</GrantTypeValidatorImplClass>
</SupportedGrantType>
```

2. Create a file named `jaas.conf` in the `<API-M_HOME>/repository/conf/identity` directory with the following content.
3. Copy the following JARs into the `<API-M_HOME>/repository/components/dropins` directory.
   
   - org.wso2.carbon.identity.application.authenticator.iwa-5.3.0.jar
   - org.wso2.carbon.identity.idp.metadata.saml2_1.0.1.jar

4. Configure OAuth2 for your client application with the Kerberos grant type.
   
   a. Start the WSO2 API-M server by navigating to the `<API-M_HOME>/bin` directory in your console and running one of the following scripts based on your OS.
      
      - On Windows:
        wso2server.bat --run
      - On Linux/Mac OS:
        sh wso2server.sh
   
   b. Sign into the API Store.
      
      https://<hostname>:9443/store
   
   c. Click **Applications** and click on the name of the application that you want to configure the OAuth2 with the Kerberos grant type.
   
   d. Generate the Production Keys.
      
      i. Click **Production Keys**.
      
      ii. Click on the Kerberos checkbox as shown in the screenshot.

   e. Generate the Sandbox Keys.
      
      i. Click **Sandbox Keys**.
      
      ii. Click on the Kerberos checkbox.
      
      iii. Click **Generate Keys** to generate the keys.

5. Configure the Service Principal Name (SPNName) and Service Principal Password (SPNPassword).

   A service principal name (SPN) is a unique identifier of a service instance. SPNs are used by Kerberos authentication to associate a service instance with a service logon account. This allows a client application to request that the service authenticate an account even if the client does not have the account name.

   a. Sign in to the WSO2 API-M Management Console.
      
      https://<Server-Host>:9443/carbon
   
   b. Navigate to the Main menu, click Add under the Identity Provider menu.
   
   c. Add a new Identity Provider (IDP).
6. Invoke the token endpoint using the message format discussed in step 3.

Note that for users to be counted in the Registered Users for Application statistics which takes the number of users shared each of the Application, they should have to generate access tokens using Password Grant type.

**Refresh Token Grant**

After an access token is generated, sometimes you might have to renew the old token due to expiration or security concerns. You can renew an access token using a refresh token, by issuing a REST call to the Token API with the following parameters. With this grant type, the refresh token acts as credentials that are issued to the client by the authorization server. Issuing a refresh token is optional. If the authorization server issues a refresh token, it is included when issuing an access token. Refresh tokens are issued for all other grant types other than the implicit grant as recommended by the OAuth 2.0 specification.

**Tip**: Be sure to keep the refresh token private, similar to the access token as this token issues access tokens without user interactions.

- Generating a new access token and refresh token
- Revoking a refresh token

**Generating a new access token and refresh token**

To use this grant type, you need a refresh token, using which you can get a new access token and a refresh token. This can be done by issuing a REST call to the Token API through a REST client like curl, with the following parameters:

- The Token API URL is `https://localhost:8243/token`, assuming that both the client and the Gateway are run on the same server.
- `payload`: `grant_type=refresh_token&refresh_token=<retoken>`.
- `headers`: `Authorization :Basic <base64 encoded string>, Content-Type: application/x-www-form-urlencoded`, `Replace <base64 encoded string> as appropriate`.

For example, the following curl command can be used to access the Token API.

```
```

You receive a response similar to the following:
The above REST message grants you a renewed access token along with a refresh token, which you can use the next time you renew the access token. A refresh token can be used only once. You can configure an expiration time for the refresh token by setting it in the `<RefreshTokenValidityPeriod>` element in the `<APIM_HOME>/repository/conf/identity/identity.xml` file.

**Revoking a refresh token**

After issuing an access token and refresh token, a user or an admin can revoke it in case of theft or a security violation. You can do this by calling the Revoke API using a utility like `cURL`. The Revoke API’s endpoint URL is `http://localhost:8280/revoke`.

**Option 1**

The parameters required to invoke the following API are as follows:

- `<refresh_token_to_be_revoked>` - The token to be revoked
- `<base64 encoded (clientId:clientSecret)>` - Use a base64 encoder (e.g., https://www.base64encode.org/) to encode your client ID and client secret using the following format: `<clientId>:<clientSecret>` Thereafter, enter the encoded value for this parameter.

```bash
```

```bash
curl -k -v -d "token=c8e8eec2-0092-3ac6-b23f-ef7492f345a6" -H "Authorization: Basic OVRRNVJLZWFhVGZGeUpRSkRzam9aZmp4UkhjYTpDZnJ3XRUa192OTdzSSZTZWlWQWxlaXdVWmt" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/revoke
```

Option 2

The parameters required to invoke the following API are as follows:

- `<refresh_token_to_be_revoked>` - The token to be revoked.
- `<base64 encoded (clientId:clientSecret)>` - Use a base64 encoder (e.g., https://www.base64encode.org/) to encode your
client ID and client secret using the following format: <clientId>:<clientSecret> Thereafter, enter the encoded value for this parameter.

- **token_type_hint** - This parameter is optional. If you do not specify this parameter, then WSO2 Identity Server (WSO2 IS) will search in both key spaces (access and refresh) and if it finds a matching token then it will be revoked. Therefore, if this parameter it not specified the token revokation process takes longer. However, if you specify this parameter then WSO2 IS will only searches in the respective token key space, hence the token revokation process is much faster.

```
curl -k -v -d "token=<refresh_token_to_be_revoked>&token_type_hint=<access_token_or_refresh_token>" -H "Authorization: Basic <base64 encoded (clientId:clientSecret)>" -H Content-Type: application/x-www-form-urlencoded https://localhost:8243/revoke
```

```
curl -k -v -d "token=4ed29669-a457-3f83-af1e-180cad271ccastoken_type_hint=refresh_token" -H "Authorization: Basic OVRRNVJLZWFhV0ZGeUpRSkRzam9aZmp4UkhjYTg0ZnJ3EXRual9ZOTdSSZFTZWIWxlaXdvVmtm" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/revoke
```

Note that for users to be counted in the Registered Users for Application statistics which takes the number of users shared each of the Application, they should have to generate access tokens using Password Grant type.

### Authorization Code Grant

Instead of requesting authorization directly from the resource owner (resource owner's credentials), in this grant type, the client directs the resource owner to an authorization server. The authorization server works as an intermediary between the client and resource owner to issues an authorization code, authenticate the resource owner and obtain authorization. As this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a Web browser) and receiving incoming requests (via redirection) from the authorization server.

The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint (you can use the /authorize endpoint for the authorization code grant type of OAuth 2.0). It includes the client identifier, response_type, requested scope, and a redirection URI to which the authorization server sends the user-agent back after granting access. The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner granted or denied the client's access request. Assuming the resource owner grants access, the authorization server then redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes an authorization code.

The client then requests an access token from the authorization server's /token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. It then includes the redirection URI used to obtain the authorization code for verification. The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI matches the URI used to redirect the client from the /authorize endpoint in the previous response. If valid, the authorization server responds back with an access token and, optionally, a refresh token.

Invoking the Token API to generate tokens

Assuming that both the client and the API Gateway are run on the same server, the Authorization API URL is `https://localhost:8243/authorize`.

- **query component:** response_type=code&client_id=<consumer_key>&scope=PRODUCTION&redirect_uri=<application_call
For example, the client directs the user-agent to make the following HTTP request using TLS:

```
GET /authorize?response_type=code&client_id=wU62Dj1yDBmq87GlBwp1aFvMa%20A&scope=PRODUCTION&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
```

The authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WlxSbIA
```

Now the client makes the following HTTP request using TLS to the /token endpoint:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic SVpzSWk2SER1QjVlO3ZL2FpBb1VpXZ2aM2Y4YTPbHbTSjZvVl1Y2ZkM1T1PMxGxjDmpzbEFDVzhh
Content-Type: application/x-www-form-urlencoded
grant_type=authorization_code&code=SplxlOBeZQQYbYS6WlxSbIA&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

The /token endpoint responds in the same way like in password grant type.

Note that if you are using a separate server for authentication (e.g., a distributed API Manager setup or an instance of WSO2 Identity Server as the authentication server), be sure to give the full URL of the authentication server in the `<APIM_HOME>/repository/conf/identity/application-authentication.xml` file. The default configuration has a relative path, which works in a standalone API Manager setup:

```
<AuthenticationEndpointURL>/authenticationendpoint/login.do</AuthenticationEndpointURL>
<AuthenticationEndpointRetryURL>/authenticationendpoint/retry.do</AuthenticationEndpointRetryURL>
```

**Try Authorization Code Grant**

The steps below show how access tokens are generated for the authorization code grant type.

**Before you begin,**

The following instructions use the sample playground webapp. For instructions on how to set up the sample webapp, see Setting up the Sample Webapp.

1. Log in to the API Manager Store and create a new application.
2. Go to the **Production Keys** tab.
3. Add the Callback URL of your playground app, select **Code** Grant type click **Generate Keys**.

   By default the implicit and code grant type selection checkboxes are disabled in the UI. You need to enter the callback URL first to enable selecting the code grant type.

4. Go to the playground app and click Import Photos.
5. Give the information in the table below and click Authorize.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Grant Type</td>
<td>Authorization Code</td>
</tr>
<tr>
<td>Client Id</td>
<td>Consumer Key obtained for your application</td>
</tr>
<tr>
<td>Scope</td>
<td>The scope you have selected for your application</td>
</tr>
<tr>
<td>Callback URL</td>
<td>The callback URL of your application</td>
</tr>
<tr>
<td>Authorize Endpoint</td>
<td><a href="https://localhost:9443/oauth2/authorize">https://localhost:9443/oauth2/authorize</a></td>
</tr>
</tbody>
</table>

6. The playground application redirects to the login page. Enter your username and password and click Sign In.
7. Click Approve to provide access to your information.

You will receive the access token as follows
NTLM Grant

**NTLM** is the successor of the authentication protocol in Microsoft LAN Manager (LANMAN), an older Microsoft product, and attempts to provide backwards compatibility with LANMAN. You can obtain an access token to your API in an API Manager instance running on Windows by providing a valid NTLM token as an authorization grant. The steps are given below:

1. Invoking the Token API to generate tokens
   - Get a valid consumer key and consumer secret pair. Initially, you generate these keys through the API Store by clicking the **Generate Keys** button on the Production Keys tab of the application.
   - Combine the consumer key and consumer secret keys in the format consumer-key:consumer-secret and encode the combined string using base64 (http://base64encode.org).
   - In order to generate an access token with NTLM, you must have an NTLM token.
   - Generate an NTLM token by running the sample provided in the directory. See the **Readme.txt** in the same folder for instructions.
   - Invoke the token API in the following manner to get an access token.

\[
\text{curl -k -d "grant_type=iwa:ntlm&windows_token=<give the NTLM token you got in step 3>" \-H "Authorization: Basic <give the string you got in step2>" \-H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token}
\]

Password Grant

You can obtain an access token by providing the resource owner's username and password as an authorization grant. It requires the base64 encoded string of the consumer-key:consumer-secret combination. You need to meet the following prerequisites before using the Token API to generate a token.

**Prerequisites**

- A valid user account in the API Store. You can self sign up if it is enabled by an admin.
- A valid consumer key and consumer secret pair. Initially, these keys must be generated through the API Store by clicking **Generate Keys** on the Production Keys tab of the application.
- A running API Gateway instance (typically an API Manager instance should be running). For instructions on API Gateway, see **Components**.
- If the Key Manager is on a different server than the API Gateway, change the server URL (host and ports) of the Key Manager accordingly in the `<AM_HOME>/repository/conf/api-manager.xml` file.
- If you have multiple Carbon servers running on the same computer, change the port with an offset to avoid port conflicts.

Invoking the Token API to generate tokens

1. Combine the consumer key and consumer secret keys in the format consumer-key:consumer-secret and encode the combined string using base64. Encoding to base64 can be done using the URL: http://base64encode.org.

Here's an example consumer key and secret combination: wU62Dj1yDBnq870I8wp1FqvmAbAa:k3dSdce6DGP7wpaElfqvmjDve. And here's
the string encoded from the example: d1U2MqgbhIEq5xODdhBB13cGxmcXZtQWJBTpro2RT2G91kREUd3cGFFbGZxdmlqRHV1. The encoded string should be used in the header of the cURL command.

2. Access the Token API by using a REST client such as cURL, with the following parameters.

- Assuming that both the client and the API Gateway are run on the same server, the token API url is https://localhost:8243/token
- payload - "grant_type=password&username=<username>&password=<password>&scope=<scope>". Replace the <username> and <password> values as appropriate.

Tip: <scope> is optional.

If you define a scope for an API's resource, the API can only be accessed through a token that is issued for the scope of the said resource. For example, if you define a scope named 'update' and issue one token for the scopes 'read' and 'update', the token is allowed to access the resource. However, if you issue the token for the scope named 'read', the request to the API will be blocked.

- headers - Authorization: Basic <base64 encoded string>, Content-Type: application/x-www-form-urlencoded. Replace the <base64 encoded string> as appropriate.

For example, use the following cURL command to access the Token API. It generates two tokens as an access token and a refresh token. You can use the refresh token at the time a token is renewed.

curl -k -d "grant_type=password&username=<username>&password=<password>" -H "Authorization: Basic d1U2MqgbhIEq5xODdhBB13cGxmcXZtQWJBTpro2RT2G91kREUd3cGFFbGZxdmlqRHV1" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token

You receive a response similar to the following:

```
Response:
{
  "scope":"default",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"ca5a51f18b2edf4eaa9e4b871e42b58a",
  "access_token":"f2c66f146278aaaf6513b585b5b68d1d"
}
```

Instead of using the Token API, you can generate access tokens from the API Store's UI.

Tip:

Note that for users to be counted in the Registered Users for Application statistics which takes the number of users shared each of the Application, they should have to generate access tokens using Password Grant type.

SAML Extension Grant

SAML 2.0 is an XML-based protocol. It uses security tokens containing assertions to pass information about an end-user between a SAML authority and a SAML consumer. A SAML authority is an identity provider (IdP) and a SAML consumer is a service provider (SP).

Enterprise applications that have SAML2 based SSO infrastructures sometimes need to consume OAuth-protected resources through APIs. However, these apps prefer to use the existing trust relationship with the IdP, even if the OAuth authorization server is entirely different from the IdP. The API Manager leverages this trust relationship by exchanging the SAML2.0 token to an OAuth token with the authorization server. It acts as the OAuth authorization server.

When SAML bearer token is used, the roles of the user can be retrieved from either the user store or the SAML assertion. When checkRolesFromSamlAssertion system property is set to true, the roles will be checked from the SAML assertion, not the user store. Refer the steps below to set this property:

1. Set the property -DcheckRolesFromSamlAssertion=true in the <API-M_HOME>/bin/wso2server.(sh|bat) file.
2. Restart the server.

The diagram below depicts the above with WSO2 Identity Server as the IdP.
The steps of the above diagram are explained below:

**Step [1]:** User initiates a login call to an enterprise application

**Step [2]:**
- As the application is a SAML SP, it redirects the user to the SAML2.0 IdP to log in.
- The user provides credentials at the IdP and is redirected back to the SP with a SAML2.0 token signed by the IdP.
- The SP verifies the token and logs the user to the application.
- The SAML 2.0 token is stored in the user’s session by the SP.

**Step [3]:**
- The enterprise application (SP) wants to access an OAuth2 protected API resource through WSO2 API Manager.
- The application makes a request to the API Manager to exchange the SAML2 bearer token for an OAuth2.0 access token.
- The API Manager validates the assertion and returns the access token.

**Step [4]:** User does API invocations through the API Manager by setting it as an Authorization header with the returned OAuth2 access token.

- Configuring the token exchange
- Invoking the Token API to generate tokens

A sequence diagram explaining the above flow would be as follows:
### Configuring the token exchange

Before you begin, make sure you have the following:

- A valid user account in the API Store.
- A valid consumer key and consumer secret. Initially, these keys must be generated in the API Store by clicking the **Generate Keys** button on the **Production Keys** tab of the application.
- A running API Gateway instance.
- If the Key Manager is on a different server than the API Gateway, change the server URL (host and ports) of the Key Manager accordingly in the `<ServerURL>` element of the `<APIKeyManager>` section in the `api-m-<tenantDomain>/repository/conf/api-manager.xml` file.
- A valid SAML2 assertion. For instructions on how to configure WSO2 API Manager with SAML2, see **Configuring API Manager for SSO**

In this example, **WSO2 Identity Server 5.7.0** is used as the IdP to get a SAML token and the API Manager is used as the OAuth server.


   If you are using a tenant to create the Identity Provider, use the credentials of tenant admin to log into the API Manager’s Management Console.

2. Click **Main > Identity Providers > Add**

3. Provide the following values to configure the IdP:
   - **Under Basic Information**
     - **Identity Provider Name**: Enter a unique name for the IdP.
     - **Identity Provider Public Certificate**: The certificate used to sign the SAML assertion. Export the public certificate of WSO2 IS and import it here. For more information, see **Exporting the public certificate** in the WSO2 Identity Server documentation.
       
       Alternatively, you can create a self-signed certificate and then export it as a `.cer` file using the following commands:

       ```bash
       keytool -genkey -alias wookie -keyalg RSA -keystore wookieKeystore.jks -keysize 4096
       keytool -v -export -file keystore1.cer -keystore wookiekeystore.jks -alias wookie
       ```
   - **Alias**: Give the name of the alias if the Identity Provider identifies this token endpoint by an alias. e.g., 9443/oauth2/token.
   - **Under Federated Authenticators > SAML2 Web SSO Configuration**
     - **Enable SAML2 Web SSO**: true
     - **Identity Provider Entity Id**: The SAML2 issuer name specified when generating the assertion token, which contains the unique identifier of the IdP. You give this name when configuring the SP.
     - **Service Provider Entity Id**: Issuer name given when configuring the SP.
     - **SSO URL**: Enter the IDP’s SAML2 Web SSO URL value. E.g., [https://localhost:9444/samlsso/](https://localhost:9444/samlsso/) if you have offset the default port, which is 9443.

   If you are in tenant mode, append the tenant domain to the SSO URL as a query parameter as below.

   ```text
   https://localhost:9443/samlsso?tenantDomain=<tenantDomain>
   ```
Next, let's register a service provider.

4. Sign in to the management console of the Identity Server and click **Main > Service Providers > Add**.

5. Choose to edit the service provider that you just registered and click **Inbound Authentication Configuration > SAML2 Web SSO Configuration**.
5. Provide the following values to configure the SP and click **Update**:

- **Issuer**: Give any name
- **Assertion Consumer URL**: The URL to which the IDP sends the SAML response. For example: `https://localhost:9443/store/jagg/jaggery_acs.jag`
- **Enable Response Signing**: true
- **Enable Audience Restriction**: true
  
  **Audience**: URL of the token API.
  
  For example: `https://localhost:9443/oauth2/token`
Let’s see how to get a signed SAML2 token (encoded assertion value) when authenticating against a SAML2 IDP. With the authentication request, you pass attributes such as the SAML2 issuer name, token endpoint and the restricted audience. In this guide, we use a command-line client program developed by WSO2 to create the 64-bit, URL-encoded SAML assertion.

**Invoking the Token API to generate tokens**

Follow the steps below to invoke the token API to generate access tokens from SAML2 assertions.

1. Combine the consumer key and consumer secret keys as `consumer-key:consumer-secret`. Encode the combined string using base64 ([http://base64encode.org](http://base64encode.org)). Here’s an example consumer key and secret combination: `wU62DjlyD8nq87GlBwplfqvmAbAa:ksdSdoefDPl7wpElfqvmjDue`.

   Let’s create a SAML2 assertion using the same command-line client that you used in the previous section.

2. Download the command-line tool from [here](https://github.com/wso2/carbon-saml) and extract the ZIP file.

3. Go to the extracted folder using the command line and execute the following command. We assume that both the client and the API Gateway run
on the same server. Therefore, the Token API URL is https://localhost:8243/token.

**Format**

```
java -jar SAML2AssertionCreator.jar <Identity_Provider_Entity_Id> <saml-subject>
<saml-recipient> <saml-identity> <identity_provider_JKS_file>
<Identity_Provider_JKS_password> <identity_provider_certificate_alias>
<identity_provider_private_key_password>
```

**Example**

```
java -jar SAML2AssertionCreator.jar localhost admin https://localhost:9443/oauth2/token
https://localhost:9443/oauth2/token
/home/user/wso2am-2.1.0/repository/resources/security/wso2carbon.jks wso2carbon wso2carbon
wso2carbon
```

The arguments are as follows:

- `<Identity_Provider_Entity_Id>` - This is the value of the saml:Issuer, which is a unique identifier of the identity provider.
- `<saml-subject>` - This is the value of the name ID, which is found in the saml:Subject.
- `<saml-recipient>` - This is the value of the subject confirmation data recipient, which is found in the saml:SubjectConfirmationData.Recipient.
- `<saml-identity>` - The alias of the public certificate.
- `<identity_provider_JKS_file>` - Pointer to the Java Key Store (JKS) file to be used for credentials.
- `<identity_provider_JKS_password>` - The JKS password.
- `<identity_provider_certificate_alias>` - The alias of the public certificate.
- `<identity_provider_private_key_password>` - The password of the private key that is used for signing.

This command returns a SAML2 assertion XML string and a base64-URL encoded assertion XML string.

You now have a SAML2 assertion.

4. Access the Token API using a REST client such as curl. For example, the following cURL command generates an access token and a refresh token. You can use the refresh token at the time a token is renewed.

```
```

Note that for users to be counted in the Registered Users for Application statistics which takes the number of users shared each of the Application, they should have to generate access tokens using Password Grant type.

**Client Credentials Grant**

Client credentials can be used when the authorization scope is limited to the protected resources belonging to the client. Client credentials are used as an authorization grant when the client requests access to protected resources based on an authorization previously arranged with the authorization server. The client application requests an access token from the authorization server, authenticating the request with its client key and client secret. If the client is successfully authenticated, an access token is returned.

Please refer to the WSO2 IS documentation for a detailed explanation on this grant type with the use of a sequence diagram.

**Invoking the Token API to generate the tokens**

1. Get a valid consumer key and consumer secret pair. Initially, you generate these keys through the API Store by clicking Generate Keys on the Production Keys tab of the application.
2. Combine the consumer key and consumer secret keys in the format consumer-key:consumer-secret and encode the combined string using base64 (http://base64encode.org).
3. Use the following sample cURL command to obtain the access token.

```
FormatExampleResponse
```
Tip: We use the Client Credentials grant type to regenerate access tokens for an application through the API Store. Therefore, you should enable this grant type to the application. To do that, go to the API Store, click the application name from under the APPLICATIONS menu, click the APPLICATIONS tab, and select the Production Keys check box under Client Credentials.

Setting a custom validity time for access tokens

You can set a validity period for access tokens through a cURL command. Pass the validity_period parameter as shown in the example below.
Implicit Grant

Implicit grant type is used to obtain access tokens if your application (client) is a mobile application or a browser based app such as a JavaScript client. Similar to authorization code grant, the implicit grant type is also based on redirection flow. The redirection URI includes the access token in the URI fragment. Therefore, the client application is capable of interacting with the resource owner user agent to obtain the access token from the redirection URI which is sent from the authorization server.

The implicit grant type does not require client authentication, and relies on the presence of the resource owner and the registration of the redirection URI. The resource owner is authenticated by the authorization server to obtain the access token. The access token is encoded into the redirection URI. This may be exposed to the resource owner and other applications residing inside the same device.

The diagram below depicts the flow of Implicit Grant.

1. The client requests for the access token with the client ID and grant type, and other optional parameters.
2. Since the resource owner authenticates directly with the authorization server, their credentials will not be shared with the client.
3. The Authorization Server sends the access token through a URI fragment to the client.
4. The client extracts the token from the fragment and sends the API request to the Resource Server with the access token.

The following parameters are required to implement the Implicit grant type in WSO2 API Manager.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope</td>
<td>The OAuth scope you are requesting for the particular token</td>
<td>openid</td>
</tr>
<tr>
<td>response_type</td>
<td>The required response format</td>
<td>id_token</td>
</tr>
<tr>
<td>redirect_uri</td>
<td>The URL of the OAuth application requesting for the token</td>
<td><a href="http://localhost:8080/playground2/oauth2client">http://localhost:8080/playground2/oauth2client</a></td>
</tr>
<tr>
<td>nonce</td>
<td>Any random value</td>
<td>13e2312637dg136e1</td>
</tr>
<tr>
<td>client_id</td>
<td>Client ID of the OAuth application</td>
<td>mzdQQ0RZOIqAf549ucIImB4h0SIa</td>
</tr>
</tbody>
</table>

An example is given below:

```
https://localhost:8243/authorize
scope=openid
&response_type=id_token
&redirect_uri=http://localhost:8080/playground2/oauth2client
&nonce=13e2312637dg136e1
&client_id=mzdQQ0RZOIqAf549ucIImB4h0SIa
```

Invoking the Token API to generate tokens

In this example we use the WSO2 Playground, which is hosted as a web application, to obtain the access token with implicit grant.

Before you begin,

The following instructions use the sample playground webapp. For instructions on how to set up the sample webapp, see Setting up the Sample Webapp.
1. Login to WSO2 API Manager Store and create an application as shown below.

   ![Add Application](image)

   An application is a logical collection of APIs. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe to one API multiple times with different SLA levels. The Default Application is pre-created and allows unlimited access by default.

   - **Name**: PlaygroundApp
   - **Per Token Quota**: Unlimited
   - **Description**:

   ![Add](image)

2. Go to the Production keys tab for the application. Add [http://localhost:8080/playground2/oauth2client](http://localhost:8080/playground2/oauth2client) as the **Callback URL**. Select **Implicit** from the list of grant types and click **Generate Keys**.

   The Implicit grant and Code grant type checkboxes are disabled by default in the UI. To enable selecting the checkboxes, enter the **Callback URL** for the application.

   ![Production Keys](image)

   - **No Keys Found**
   - **Grant Types**
     - **Refresh Token**, **SAML2**, **Implicit**, **Code**
   - **Callback URL**: [http://localhost:8080/playground2/oauth2client](http://localhost:8080/playground2/oauth2client)

3. Go to playground app [http://wso2is.local:8080/playground2/index.jsp](http://wso2is.local:8080/playground2/index.jsp) and click **import photos**.
4. Give the information in the table below and click Authorize.

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Grant Type</td>
<td>Implicit</td>
</tr>
<tr>
<td>Client Id</td>
<td>Consumer Key obtained for your application</td>
</tr>
<tr>
<td>Scope</td>
<td>The scope you have selected for your application</td>
</tr>
<tr>
<td>Callback URL</td>
<td>The callback URL of your application</td>
</tr>
<tr>
<td>Authorize Endpoint</td>
<td><a href="https://localhost:8243/authorize">https://localhost:8243/authorize</a></td>
</tr>
</tbody>
</table>

5. The playground application redirects to the login page. Enter your username and password and click Sign In.
Click Approve to provide access to your information.

You will receive the access token as follows

For users to be counted in the Registered Users for Application statistics, which takes the number of users shared each of the Application, they have to generate access tokens using Password Grant type.

JWT Grant

The JSON Web Token (JWT) bearer grant is simply a JSON string containing claim values that will be evaluated and validated by the JWT Grant Handlers at the Authorization Server end, before issuing an access token.

WSO2 API Manager, as an OAuth 2.0 Authorization Server with its Key Manager features, can accept JWT Assertions from OAuth 2.0 clients as means of resource owner authentication and authorization. Additionally, it can exchange the JWT token with OAuth 2.0 access tokens in order to access protected resources on behalf of the resource owner.

- Configuring the JWT grant
- Using the JWT grant
- JWT Bearer Grant
Configuring the JWT grant

1. Sign in to the WSO2 API Manager. Enter your username and password to log on to the Management Console (https://localhost:9443/carbon).
2. Navigate to the Identity Providers section under the Main tab of the management console and click Add.
3. Provide the following values to configure the IDP:
   - Identity Provider Name: Enter an issuer name as the identity provider name. This is used to generate the JWT assertion.
   - Identity Provider Public Certificate: The certificate used to sign the JWT assertion.

Identity provider Public Certificate

The Identity Provider Public Certificate is the public certificate belonging to the identity provider. Uploading this is necessary to authenticate the response from the identity provider.

This can be any certificate. If the identity provider is another API Manager or Identity Server, this can be a wso2.crt file.

To create the identity provider certificate from the wso2carbon.jks file, follow the steps below.

1. Open your Command Line interface, go to the <APIM_HOME>/repository/resources/security/directory. Run the following command.

   ```
   keytool -export -alias wso2carbon -file wso2.crt -keystore wso2carbon.jks -storepass wso2carbon
   ```

2. Once you run this command, the wso2.crt file is generated and can be found in the <APIM_HOME>/repository/resources/security/directory. Click Choose File and navigate to this location in order to select and upload this file.

   See Using Asymmetric Encryption in the WSO2 Product Administration Guide for more information on how public keys work and how to sign these keys by a certification authority.

   - **Alias**: Give the name of the alias if the Identity Provider identifies this token endpoint by an alias (e.g., https://localhost:9443/authorization2/token). For more information, see Adding a new identity provider.

Add New Identity Provider

<table>
<thead>
<tr>
<th>Basic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity Provider Name: Enter a unique name for this identity provider</td>
</tr>
<tr>
<td>Display Name: Enter the identity provider’s display name</td>
</tr>
<tr>
<td>Description: A meaningful description about the identity provider</td>
</tr>
<tr>
<td>Federation Real Identity Provider: Check if this point to a Federation real identity provider</td>
</tr>
<tr>
<td>Home Real Identity Provider: Enter the home real identity for this identity provider</td>
</tr>
<tr>
<td>Identity Provider Public Certificate: Select a certificate that belongs to the identity provider</td>
</tr>
</tbody>
</table>

Identity provider public certificate

Ch=localthost, O=wso20, L=Mountain View, ST=CA, C=US
Ch=localthost, O=wso20, L=Mountain View, ST=CA, C=US
11/02/2015
10/02/2016
1206562196
3

4. Navigate to the Main menu to access the Identity menu. Click Add under Service Providers.
5. Fill in the Service Provider Name and provide a brief Description of the service provider. See Adding a Service Provider for more information.
6. Expand the OAuth/OpenID Connect Configuration and click Configure.
7. Enter a Callback URL. For example, use http://localhost:8080/playground2/oauth2client and click Add.
   The OAuth Client Key and OAuth Client Secret will now be visible.
Using the JWT grant

The cURL commands below can be used to retrieve the access token and refresh the token using a JWT.

Request

```
curl -i -X POST -u <clientid>:<clientsecret> -k -d
```

The -u flag should specify the "<Client Id>:<Client Secret>" value. The assertion parameter value is the signed base64 encoded JWT. The value of the assertion parameter MUST contain a single JWT. You can refer JWT Bearer Grant for more information about assertion.

If you have configured the service provider and identity provider in a tenant, you have to add the tenant domain as a query parameter to the access token endpoint.

If the tenant domain is wso2.com, the access token endpoint will be as follows.

```
```

Sample request

```
curl -i -X POST -H 'Content-Type: application/x-www-form-urlencoded' -u bBhEoE2wIpU1zB8HA3GfvZz8xxAa:RKgXUC3pTRq9xPpWwuyTP0tnSpq -k -d
'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0NTgxNjY5ODUsInN1YiI6ImFkbWluIiwibmJmIjoxNDU4MTA2OTg1LCJhdWQiOiJqd3RJRFAiLCJqdGkiOiJUb2tlbjU2NzU2IiwiaWF0IjoxNDU4MTA2OTg1fQ.ZcxdoTVEsWoil80ne42QzmsfelMWyjrR0JE3UK1e2VMIJjt2ZnnWExyCA5tsN6ixYFAXC._7rkFuuNSgO151MNLP2z3WcgG152jefpGEW92v2i9z5RKOeLQLqLsfo8ae8KzLp7kuKZ2XunfQ2YWYU97vvlDm_vp5ruuYz3ZrJ0c' https://localhost:9443/oauth2/token
```

You would have now received the response from the token endpoint. The response would contain the access token, refresh token, expiry time and token type.
JWT Bearer Grant

JWT contains three parts that are separated by dots ".": header, payload, and a signature. The header identifies the algorithm used to generate the signature.

For example, see the following code block.

```
Sample header
{
    "alg":"RS256"
}
```

The payload contains the claims mentioned below:

- **iss (issuer)** - The JWT must contain an **iss** (issuer) claim that contains a unique identifier that identifies the identity provider that issued the JWT.
- **sub (subject)** - The JWT must contain a **sub** (subject) claim that identifies the entity that the identity provider or the entity that issued the JWT vouches for.
- **aud (audience)** - The JWT must contain an **aud** (audience) claim which containing a value that identifies the authorization server as an intended audience. This value should be registered as token endpoint alias in the Identity Provider.
- **exp (expiration time)** - The JWT must contain an **exp** (expiration) claim that limits the time window during which the JWT can be used.
- **nbf (not before)** - The JWT may contain a **nbf** (not before time) claim that forces a JWT to be used only after a specified time.
- **iat (issued at)** - The JWT may contain an **iat** (issued at) claim that identifies the time at which the JWT was issued.
- **jti (json web token ID)** - The JWT may contain **jti** (JWT ID) claim that provides a unique identifier for the token.
- **Other custom claims** - JWT may contain claims other than the above mentioned ones. This is the extension point of the JWT specification.

For example, see the following code block.

```
Sample payload
{
    "sub":"admin",
    "aud":[
        "https://localhost:9443/oauth2/token",
    ],
    "nbf":1507546100,
    "iss":"jwtIDP",
    "exp":1507606100,
    "iat":1507546100,
    "jti":"Token56756"
}
```

The signature is calculated by base64 URL encoding the header and payload and concatenating them with a period as a separator and signing it:

```
Signature = sign(encodeBase64(header) + '.' + encodeBase64(payload))
```

The signature must then be base64 URL encoded. JWT assertion can be generated by concatenating these three encoded values with a separator dot ".".

```
assertion = encodeBase64(header) + '.' + encodeBase64(payload) + '.' + encodeBase64(signature)
```

The result is as follows:
Sample assertion
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImF1dF9pZCI6IjB7InVzZXIiOm51bGwsImlhdCI6MTc3NzA5MzIwNiwiaWRtaW4iOjEwMDc1MjI3OCwiaHR0cDovL3NjaGVtZS5sYW5kcy5jb20vY3Jvc3QvMjAxODU4NzIwLzIwaWQvY29tcG9uZW50L2F1dGhvc2UuanBnIiwiZXhwIjoxNTA3NTQ2MTAwLCJjZGZlZCI6MTg5MzgwMTI0Nl0

Deprecated APIs

The following APIs are deprecated and will be unsupported in a future release. They are provided below for reference by existing users.

- Publisher APIs
- Store APIs

Publisher APIs

The following Publisher APIs are deprecated and will be unsupported in a future release. They are provided below for reference by existing users. For a complete list of the currently supported Publisher APIs, go to https://docs.wso2.com/display/AM260/apidocs/publisher.

- Login
- Logout
- Add API
- Add API with Path Parameter
- Update API
- Publishing an API to external Store
- Get All APIs
- Get an API
- Remove an API
- Copy an API
- Check Older Version
- Change API Status
- Add/Update an API Document
- Remove an API Document
- Get all Throttling Tiers
- Check if API Exists
- Validate Roles
- Upload a Certificate
- Remove a Certificate
- Analytics related APIs

Note: When you access any API other than the login and logout APIs through an external REST client such as cURL, first invoke the login API to ensure that user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations. The response is a JSON message.

Login

<table>
<thead>
<tr>
<th>Description</th>
<th>Log in to API Publisher web application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a></td>
</tr>
<tr>
<td>URI Parameters</td>
<td>action=login&amp;username=xxx&amp;password=xxx</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>POST</td>
</tr>
<tr>
<td>Example</td>
<td>curl -X POST -c cookies <a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a> -d action=login&amp;username=admin&amp;password=admin</td>
</tr>
</tbody>
</table>

Logout

<table>
<thead>
<tr>
<th>Description</th>
<th>Log out from API Publisher web application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a></td>
</tr>
</tbody>
</table>
### Add API

<table>
<thead>
<tr>
<th>Description</th>
<th>Add a new API.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a></td>
</tr>
</tbody>
</table>
| URI Parameters | Given below are the parameters that you can pass with an Add-API call. Mandatory ones are marked with a *.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action*</td>
<td>name=xxx</td>
</tr>
<tr>
<td>Name*</td>
<td>context=/xxx</td>
</tr>
<tr>
<td>Version*</td>
<td>version=x.x.x</td>
</tr>
<tr>
<td>API visibility*</td>
<td>visibility=&lt;public</td>
</tr>
<tr>
<td>Thumb image</td>
<td>- To add a thumbnail image as a file object, create the object and pass it with the apiThumb parameter. See sample in this method.</td>
</tr>
<tr>
<td></td>
<td>- To add a thumbnail image as a URL of the image, pass the URL with the thumbUrl parameter as thumbUrl=&lt;URL&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>description=xxx</td>
</tr>
<tr>
<td>Tags</td>
<td>tags=x,y,z</td>
</tr>
<tr>
<td>Resources*</td>
<td>resourceCount=0&amp;resourceMethod=GET&amp;resourceMethodAuthType=Application&amp;resourceMethodThrottlingTier=Unlimited</td>
</tr>
<tr>
<td></td>
<td>- resourceMethod can take any one of the following values: GET, POST, DELETE, PUT, OPTIONS</td>
</tr>
<tr>
<td></td>
<td>- resourceMethodAuthType can take any one of the following values: Application, Application User, Application and Application User, None</td>
</tr>
<tr>
<td></td>
<td>- resourceMethodThrottlingTier can take any one of the following default values: Unlimited, Gold, Silver, Bronze</td>
</tr>
<tr>
<td></td>
<td>- These can be changed or have additional tiers defined in the /_system/governance/apimgt/applicationdata/tiers.xml registry location.</td>
</tr>
</tbody>
</table>
Resources as Swagger

```
swagger="{paths": {"/CheckPhoneNumber": {"post": {"x-auth-type": "Application\&Application User", "x-scope": "read_number", "description": "Phone Number", "in": "query"}, {"name": "LicenseKey", "paramType": "query", "required": false, "type": "string", "description": "License Key", "in": "query"}}}, "/" : {"put": {}, "get": {}}},
{x-wso2-security": {"apim": {"x-wso2-scopes": [{"description": "", "name": "read_number", "roles": "admin", "key": "string", "description": "License Key", "in": "query"}]}},
"type": "string", "description": "Phone Number", "in": "query"}, {"name": "LicenseKey", "paramType": "query", "required": false, "type": "string", "description": "License Key", "in": "query"}}},
"x-wso2-scopes": ["{x-song-token": "", "x-scope": "read_number"}], "info": {"title": "PhoneVerification", "version": "1.0.0"}
```

In the above code, note that you have one resource path defined with the URL pattern /CheckPhoneNumber under the `/` etc.) You can have multiple similar resource paths to a single API and multiple HTTP methods to each resource path. For more information of the Swagger objects used in this example, see the Swagger 2.0 specification. Described below are the WSO2-specific ones.

- **x-wso2-scopes**: The list of scope elements that you want to define. Each element has the below fields. See OAuth Scopes
  - description: Scope description
  - roles: Allowed roles
  - name: Scope Name
  - key: Scope Key

- **x-auth-type**: Authentication type of the method.
- **x-throttling-tier**: Throttling tier of the method.
- **x-scope**: OAuth scope of the method. This must be one of the list of element you define in x-wso2-scopes.

The following image shows the WSO2-specific parameters we describe here. Also see Resources under Key Concepts for more information.

Endpoints

This example adds an HTTP production endpoint:

```
endpoint_config={"production_endpoints": [{"url": "http://service.endpoint.com", "endpoint_type": "load_balance"},
{"url": "http://service.endpoint1.com", "config": null}], "sandbox_failovers": [], "endpoint_type": "failover"}
```

To give advanced endpoint configurations, add the JSON implementation inside "config":. If you don't have any advance

You add sandbox endpoints in the same way. The only difference is that instead of production_endpoints, you give

If you want to add other types of endpoints, follow the examples below. Note that the endpoint_type of both HTTP:

- For address endpoints:
  ```
  endpoint_config={"production_endpoints": [{"url": "http://service.endpoint.com "},
  "config": null},
  "url": "http://service.endpoint.com", "config": null]}}, "sandbox"}
  ```

- For failover endpoints:
  ```
  endpoint_config={"production_endpoints": [{"url": "http://service.endpoint.com "},
  "config": null], "url": "http://service.endpoint.com", "config": null]], "sandbox"}
  ```

- For load balanced endpoints:
  ```
  endpoint_config={"production_endpoints": [{"url": "http://service.endpoint.com", "config": null}], "sandbox"}
  ```

Endpoint security scheme

```
endpointType=<secured|nonsecured>
```

The default is non-secured but if you select 'secured', you must pass the credentials as follows: endpointType=secure

```
epPassword=<the password>
```

Make default version

To mark this version of the API as the default version from a group of versions, give default_version_checked=default:

The Default Version option means that you make this version the default in a group of different versions of the API. To set

If you mark any version of an API as the default, you get two API URLs in its Overview page in the API Store. One URL is

If you mark an unpublished API as the default, the previous default, published API will still be used as the default until the

Tier Availability

```
tiersCollection=<Gold,Silver,Bronze,Unlimited>
```

Transports

```
http_checked=http&https_checked=https
```

Both are selected by default. If you want to set only the HTTP transport, leave the https_checked parameter empty as

Sequences

```
inSequence=<sequence name>&outSequence=<sequence name>
```

If you want to engage a custom sequence to the API, give inSequence=<sequence name>&outSequence=<sequence name>
## Response Caching

Response caching is enabled by default. If you want to disable it, pass the response cache timeout as follows:

```
responseCache=disabled&cacheTimeout=300
```

See [Configuring Caching](#) for more information.

## Subscriptions

By default, subscription is allowed to the current tenant only.

Add the argument `subscriptions=all_tenants` to enable subscriptions to this API by all tenants. To enable subscription for specific tenants, use `subscriptions=specific_tennants&tenants=<tenant_name>`. For example:

```
subscriptions=all_tenants
```

See [API visibility and subscription](#) for more information.

## Business Information

Add a section like this:

```
bizOwner=<name>&bizOwnerMail=<e-mail address>&techOwner=<name>&techOwnerMail=<e-mail address>
```

## HTTP Methods

### POST

**Example**

```
curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag -d "action=addAPI&name=PhoneVerification&context=/phoneverify&version=1.0.0&visibility=public&description=Verify a phone number&tags=phone,mobile,multimedia&endpointType=nonsecured&tiersCollection=Gold,Bronze&http_checked=http&https_checked=https&resourceCount=0&resourceMethod-0=GET&resourceMethodAuthType-0=Application&resourceMethodThrottlingTier-0=Unlimited&uriTemplate-0=/*&default_version_checked=default_version&bizOwner=xx&bizOwnerMail=xx@ee.com&techOwner=xx&techOwnerMail=ggg@ww.com" -d 'endpoint_config={"production_endpoints":{"url":"http://ws.cdyne.com/phoneverify/phoneverify.asmx","config":null}}"'
```

## Add API with Path Parameter

**Description**

Add a new API with path parameter

**URI**

```
http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag
```

**Parameters**

```
action=addAPI&name=xxx&context=/xxx&version=1.0.0&visibility=xxx&thumbsUrl=&description=xxx&tags=xxx&endpointType=xxx&tiersCollection=xxx&http_checked=xxx&https_checked=xxx
```

**HTTP Methods**

POST

**Example**

```
curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag -d "action=addAPI&name=SampleApi&context=/sample&version=1.0.0&visibility=public&description=Verify a phone number&tags=phone,mobile,multimedia&endpointType=nonsecured&tiersCollection=Gold,Bronze&http_checked=http&https_checked=https&resourceCount=0&resourceMethod-0=GET&resourceMethodAuthType-0=Application&resourceMethodThrottlingTier-0=Unlimited&uriTemplate-0=/*&default_version_checked=default_version&bizOwner=xx&bizOwnerMail=xx@ee.com&techOwner=xx&techOwnerMail=g gg@ww.com" -d 'endpoint_config={"production_endpoints":{"url":"http://gdata.youtube.com/feeds/api/standardfeeds","config":null}}"'
```

## Update API

**Description**

Update an existing API

**URI**

```
http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag
```

**Parameters**

Parameters are the same as in Add API except that `action = updateAPI` and you can only update the following parameters: visibility, thumbUrl, description, tags, endpointType, endpoint_config, ... URL etc., `http_checked`, `https_checked`, `tiersCollection`, `swagger`, etc.

**HTTP Methods**

POST

**Example**

```
Update API : curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag -d "action=updateAPI&name=PhoneVerification&context=/phoneverify&version=1.0.0&visibility=public&description=Youtube Live Feeds&endpointType=nonsecured&http_checked=http&https_checked=https&tags=phone,mobile,multimedia&uriTemplate-0=/*&default_version_checked=default_version&bizOwner=xx&bizOwnerMail=xx@ee.com&techOwner=xx&techOwnerMail=ggg@ww.com" -d 'endpoint_config={"production_endpoints":{"url":"http://ws.cdyne.com/phoneverify/phoneverify.asmx","config":null}}'
```

## Publishing an API to external Store

**Description**

Publish an API to external store

**URI**

```
http://localhost:9763/publisher/site/blocks/item-external/ajax/external.jag
```

**Parameters**

```
action=updateExternal&name=xxx&version=xxx&provider=xxx&externalAPIStores=<external-store-1>::<external-store-2>::<external-store-3>
```

:: sign is used to separate the list of API stores
### HTTP Methods

<table>
<thead>
<tr>
<th>Description</th>
<th>POST</th>
</tr>
</thead>
</table>

| Example | **updateExternal** : curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/item-external/ajax/external.jag -d "action=updateExternal&name=PhoneVerification&version=1.0.0&provider=admin&externalAPIStores=exstore2::exstore3` |

This API can be used to unpublish an API from a given API store as well. If we remove the particular store ID and call the API once again, that API will get unpublished from the external stores which are not mentioned in the request. For example, if you want to remove the API from exstore2 and keep it published in exstore3, the following is the cURL command you need to use.

```
curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-external/ajax/external.jag -d "action=updateExternal&name=sampleAPI&version=v1&provider=admin@wso2.com&externalAPIStores=exstore2::exstore3"
```

If you want to remove the API from all the stores, provide `::` as the `externalAPIStores` parameter value.

### Get All APIs

<table>
<thead>
<tr>
<th>Description</th>
<th>Lists all the created APIs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><strong><a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a></strong></td>
</tr>
<tr>
<td>URI Parameters</td>
<td><code>?action=getAllAPIs</code></td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>GET</td>
</tr>
</tbody>
</table>

| Example | curl -b cookies `http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag` ?action=getAllAPIs |

### Get an API

<table>
<thead>
<tr>
<th>Description</th>
<th>Get details of a specific API.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><strong><a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a></strong></td>
</tr>
<tr>
<td>URI Parameters</td>
<td><code>action=getAPI&amp;name=xxx&amp;version=xxx&amp;provider=xxx</code></td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>POST</td>
</tr>
</tbody>
</table>

| Example | curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag` ?action=getAPI&name=PhoneVerification&version=1.0.0&provider=admin |

### Remove an API

<table>
<thead>
<tr>
<th>Description</th>
<th>Remove an API.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><strong><a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag</a></strong></td>
</tr>
<tr>
<td>URI Parameters</td>
<td><code>action=removeAPI&amp;name=xxx&amp;version=xxx&amp;provider=xxx</code></td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>POST</td>
</tr>
</tbody>
</table>

| Example | curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag` ?action=removeAPI&name=PhoneVerification&version=1.0.0&provider=admin |

### Copy an API

<table>
<thead>
<tr>
<th>Description</th>
<th>Copy an API to a newer version.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><strong><a href="http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag">http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag</a></strong></td>
</tr>
<tr>
<td>URI Parameters</td>
<td><code>action=createNewAPI&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;newVersion=xxx</code></td>
</tr>
<tr>
<td>HTTP</td>
<td>Methods</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>POST</td>
</tr>
</tbody>
</table>

**Example**
curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag` -d "action=createNewAPI&provider=admin&apiName=PhoneVerification&version=1.0.0&newVersion=2.0.0&isDefaultVersion=default_version"

### Check Older Version

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does older version of API exist.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URI</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>URI Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>action=isAPIOlderVersionExist&amp;provider=xxx&amp;name=xxx&amp;version=xxx</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POST</td>
</tr>
</tbody>
</table>

**Example**
curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag` -d "action=isAPIOlderVersionExist&provider=admin&name=PhoneVerification&version=1.0.0"

### Change API Status

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the API's status.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URI</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>URI Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>action=updateStatus&amp;name=xxx&amp;version=1.0.0&amp;provider=apiCreateName&amp;status=PUBLISHED&amp;publishToGateway=true&amp;requireResubscription=true</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POST</td>
</tr>
</tbody>
</table>

**Example**
curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag` -d "action=updateStatus&name=PhoneVerification&version=1.0.0&provider=admin&status=PUBLISHED&publishToGateway=true&requireResubscription=true"

### Add/Update an API Document

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a new API document.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URI Parameters Add Document</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>action=addDocumentation&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx&amp;docType=xxx&amp;newType=xxx&amp;sourceType=xxx&amp;docUrl=&amp;docLocation=xxx&amp;docVisibility=owner_only/private</code></td>
</tr>
</tbody>
</table>

Note that docVisibility is applicable only if you have enabled it. See API documentation visibility.

<table>
<thead>
<tr>
<th>Add Document file</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>action=addDocumentation&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx&amp;docType=xxx&amp;newType=xxx&amp;sourceType=xxx&amp;docUrl=&amp;docLocation=@test.txt</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update Document</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>action=addDocumentation&amp;mode=xxx&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx&amp;docType=xxx&amp;newType=xxx&amp;sourceType=xxx&amp;docUrl=&amp;docLocation=xxx</code></td>
</tr>
</tbody>
</table>

The docType parameter can have values such as How To, Samples, Public Forum etc. If the docType is specified as "Other", the value of the sourceType can be inline, file or URL.

<table>
<thead>
<tr>
<th>HTTP</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POST</td>
</tr>
</tbody>
</table>

**Example**
Add Document:
curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag` -d "action=addDocumentation&provider=xxx&apiName=xxx&version=xxx&docName=xxx&summary=xxx&docType=xxx&newType=xxx&sourceType=xxx&docUrl=&docLocation=@test.txt"

Add Document file:
curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag` -F "action=addDocumentation" -F "provider=admin" -F "apiName=testAPi" -F "version=1.0.0" -F "docName=testDoc2" -F "docType=how to" -F "docLocation=@test.txt"

Update Document:
curl -X POST -b cookies `http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag` -d "action=addDocumentation&mode=Update&provider=xxx&apiName=PizzaShackAPI&version=1.0.0&docName=Sample&summary=sample updated&docType=other&newType=primary&sourceType=inline&docUrl=&docLocation="

### Remove an API Document

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove an API document.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</code></td>
</tr>
</tbody>
</table>
### Get all Throttling Tiers

**Description**
Get the throttling tiers that can be applied to APIs

**URI**
http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=getTiers

**Example**
curl -b cookies "http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=getTiers"

### Check if API Exists

**Description**
Check if an API by a given name exists in the API Publisher

**URI**
http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINameExist&apiName=<name of the API>

**Example**
curl -b cookies "http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINameExist&apiName=PhoneVerification"

### Validate Roles

**Description**
Check if the user logged in user is any one in a given list of users

**URI**

**Example**
curl -b cookies "http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=validateRoles&roles=admin"

### Upload a Certificate

**Description**
Upload a certificate to an endpoint

**URI**

**Example**
curl -X POST -b cookies "http://localhost:9763/publisher/site/blocks/item-design/ajax/add.jag?action=addCertificate&alias=wso2&ep=http%3A%2F%2Fwso2.com&certificate=Q2VydGlmaWNhdGlvbiBiYXNzZXRzIGRvY29kZSAtIGRvY29kZSAtIGRvY29kZSBieXBlbmVudHMgMywgQ2F0aXMgMjBd" -H 'content-type: application/x-www-form-urlencoded' -d "action=addCertificate&alias=wso2&ep=http%3A%2F%2Fwso2.com&certificate=Q2VydGlmaWNhdGlvbiBiYXNzZXRzIGRvY29kZSAtIGRvY29kZSAtIGRvY29kZSBieXBlbmVudHMgMywgQ2F0aXMgMjBd"

### Remove a Certificate

**Description**
Remove a certificate to given for a certain endpoint

**URI**
http://localhost:9763/publisher/site/blocks/item-design/ajax/add.jag?action=deleteCert

**Example**
curl -X POST -b cookies "http://localhost:9763/publisher/site/blocks/item-design/ajax/add.jag?action=deleteCert&alias=<certificate-alias>&endpoint=<endpoint>"
Example:

Analytics related APIs

Before using the following analytics related APIs, ensure to configure Analytics for API-M. For more information, see Configuring APIM Analytics.

- Get List of API Creators
- Get Subscriber Count
- Get API Usage By Resource Path
- Get API Usage By Destination
- Get API Usage By Provider
- Get API and Application Throttling Data
- Get API Response Fault Count

Get List of API Creators

<table>
<thead>
<tr>
<th>Description</th>
<th>Get the list of all the API creators.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/publisher/site/blocks/stats/api-usage-user/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/api-usage-user/ajax/stats.jag</a></td>
</tr>
</tbody>
</table>
| Request Headers             | "Content-Type" -> "application/x-www-form-urlencoded" \
                              "Cookie" -> \
                              "JSESSIONID=29FC6C9F818ED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9 \
                              Path=publisher/; Secure; HttpOnly" |
| HTTP Methods                | POST |
| Payload                     | action=getAPIUsageByUser&currentLocation=/publisher/site/pages/all-statistics.jag&apiFilter=allAPIs |
| Example                     | curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIUsageByUser&currentLocation=/publisher/site/pages/all-statistics.jag&apiFilter=allAPIs' |

Get Subscriber Count

<table>
<thead>
<tr>
<th>Description</th>
<th>Get the number of subscribers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/publisher/site/blocks/stats/api-subscriptions/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/api-subscriptions/ajax/stats.jag</a></td>
</tr>
</tbody>
</table>
| Request Headers             | "Content-Type" -> "application/x-www-form-urlencoded" \
                              "Cookie" -> \
                              "JSESSIONID=29FC6C9F818ED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9 \
                              Path=publisher/; Secure; HttpOnly" |
| HTTP Methods                | POST |
| Payload                     | action=getSubscriberCountByAPIs \
                              currentLocation=/publisher/site/pages/all-statistics.jag \
                              apiFilter=allAPIs |
| Example                     | curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getSubscriberCountByAPIs&currentLocation=' |
| Sample Response | < HTTP/1.1 200 OK < Server: nginx < Date: Mon, 09 Jan 2017 05:15:49 GMT < Content-Type: application/json;charset=UTF-8 < Content-Length: 198 < Connection: keep-alive < Strict-Transport-Security: max-age=15768000 < X-Frame-Options: DENY < X-Content-Type-Options: nosniff < X-XSS-Protection: 1; mode=block | |
|------------------|--------------------------------------------------------------------------------------------------|

Get API Usage By Resource Path

**Description**

Get the API usage based on the resource path.

**URI**


**Request Headers**

- "Content-Type" -> "application/x-www-form-urlencoded"
- "Cookie" -> "JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4AF4A4D3E6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9 Path=/publisher/; Secure; HttpOnly"

**HTTP Methods**

POST

**Payload**

- `action=getAPIUsageByResourcePath`
- `currentLocation=/publisher/site/pages/all-statistics.jag`
- `fromDate:2014-05-10 00:00:00`
- `toDate:2016-12-16 14:34:33`
- `apiFilter=allAPIs`

**Example**

`curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIUsageByResourcePath&currentLocation=/publisher/site/pages/all-statistics.jag&fromDate=2014-05-10+00%3A00%3A00&toDate=2016-12-16+14%3A34%3A33&apiFilter=allAPIs' http://localhost:9763/publisher/site/blocks/stats/api-usage-resource-path/ajax/stats.jag`

Get API Usage By Destination

**Description**

Get the API usage based on the destination

**URI**


**Request Headers**

- "Content-Type" -> "application/x-www-form-urlencoded"
- "Cookie" -> "JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4AF4A4D3E6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9 Path=/publisher/; Secure; HttpOnly"

**HTTP Methods**

POST

**Payload**

- `action=getAPIUsageByDestination`
- `currentLocation=/publisher/site/pages/all-statistics.jag`
- `fromDate:2014-05-10 00:00:00`
- `toDate:2016-12-16 14:34:33`
- `apiFilter=allAPIs`

**Example**

`curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIUsageByDestination&currentLocation=/publisher/site/pages/all-statistics.jag&fromDate=2014-05-10+00%3A00%3A00&toDate=2016-12-16+14%3A34%3A33&apiFilter=allAPIs' http://localhost:9763/publisher/site/blocks/stats/api-usage-destination/ajax/stats.jag`
| Sample Response | HTTP/1.1 200 OK  
|-----------------|----------------------------------|
|                 | < Server: nginx  
|                 | < Content-Type: application/json; charset=UTF-8  
|                 | < Content-Length: 359  
|                 | < Strict-Transport-Security: max-age=15768000  
|                 | < X-Frame-Options: DENY  
|                 | < X-Content-Type-Options: nosniff  
|                 | < X-XSS-Protection: 1; mode=block  

Get API Usage by Provider

**Description**
Get API Usage by Provider.

**URI**
http://localhost:9763/publisher/site/blocks/stats/api-usage/ajax/stats.jag

**Request Headers**
"Content-Type" -> "application/x-www-form-urlencoded"  
"Cookie" -> "JSESSIONID=29FC6DCF81BED3701B2F0FD00A71D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9; Path=/publisher/; Secure; HttpOnly"

**HTTP Methods**
POST

**Payload**
action:getProviderAPIUsage  
currentLocation:/publisher/site/pages/all-statistics.jag  
fromDate:2014-05-10 00:00:00  
toDate:2016-12-16 14:34:33  
apiFilter:allAPIs

**Example**
curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getProviderAPIUsage&currentLocation=%2Fpublisher%2Fsite%2Fpages%2Fall-statistics.jag&fromDate=2014-05-10+00%3A00%3A00&toDate=2016-12-16+14%3A34%3A33&apiFilter=allAPIs' 'http://localhost:9763/publisher/site/blocks/stats/api-usage/ajax/stats.jag'

Get API and Application Throttling Data

**Description**
Get the throttling related data related to the APIs and applications.

**URI**
http://localhost:9763/publisher/site/blocks/stats/api-throttledcounts/ajax/stats.jag

**Request Headers**
"Content-Type" -> "application/x-www-form-urlencoded"  
"Cookie" -> "JSESSIONID=29FC6DCF81BED3701B2F0FD00A71D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9; Path=/publisher/; Secure; HttpOnly"

**HTTP Methods**
POST

**Payload**
action:getThrottleDataOfAPIAndApplication  
currentLocation:/publisher/site/pages/all-statistics.jag  
fromDate:2014-05-10 00:00:00  
toDate:2016-12-16 14:34:33  
apiFilter:allAPIs  
apiName: Cedum  
appName: anuruddhal.wso2.com

**Example**
curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getThrottleDataOfAPIAndApplication&currentLocation=%2Fpublisher%2Fsite%2Fpages%2Fall-statistics.jag&fromDate=2014-05-10+00%3A00%3A00&toDate=2016-12-16+14%3A34%3A33&apiFilter=allAPIs&apiName=Cedum&appName=anuruddhal.wso2.com' 'http://localhost:9763/publisher/site/blocks/stats/api-throttledcounts/ajax/stats.jag'
Get API Response Fault Count

<table>
<thead>
<tr>
<th>Description</th>
<th>Get the response fault count of APIs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/publisher/site/blocks/stats/faulty-invocations/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/faulty-invocations/ajax/stats.jag</a></td>
</tr>
</tbody>
</table>
| Request Headers | "Content-Type" -> "application/x-www-form-urlencoded"  
| | "Cookie" ->  
| | JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4A43D6DA7CE1DB8AC82882E3CDBE32C2037AF6F6FCB9  
| | Path="/publisher/; Secure; HttpOnly" |
| HTTP Methods     | POST |
| Payload          |
| action:getThrottleDataOfAPIAndApplication  
| currentLocation:/publisher/site/pages/all-statistics.jag  
| fromDate:2014-05-10 00:00:00  
| toDate:2016-12-16 14:34:33  
| apiFilter=allAPIs |
| Example | curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIResponseFaultCount&currentLocation=/s.jag' |

Sample Response

< HTTP/1.1 200 OK  
< Server: nginx  
< Date: Fri, 16 Dec 2016 09:53:17 GMT  
< Content-Type: application/json; charset=UTF-8  
< Content-Length: 176  
< Connection: keep-alive  
< Strict-Transport-Security: max-age=15768000  
< X-Frame-Options: DENY  
< X-Content-Type-Options: nosniff  
< X-XSS-Protection: 1; mode=block  
< "error" : false, "usage" : [{"apiName" : "Cedum", "version" : "1.0.0", "context" : "/anuruddha/demo/1.0.0", "faultPercentage" : 12.5, "count" : 1}]

Store APIs

The following Store APIs are deprecated and will be unsupported in a future release. They are provided below for reference by existing users. For a complete list of the currently supported Store APIs, go to https://docs.wso2.com/display/AM260/apidocs/store/.

- Login
- Logout
- User Signup
- Search APIs
- Get all Paginated Published APIs
- Add an Application
- Update an Application
- Get Applications
- Get an Application by Name
- Remove an Application
- Generate an Application Key
- Update an Application Key
- Add a Subscription
List Subscriptions
List Subscriptions by Application
List Subscriptions by API
Remove a Subscription
Delete an OAuth Application
Provision an Out-of-Band OAuth Client
Clean Partially Created Keys
Get All Documentations
Get the Contents of a File Document
Add an API Comment
Get All Endpoint URLs
Get All Available Tiers
Update Grant Types

**Note:** When you access any API other than the login and logout APIs through an external REST client such as cURL, first invoke the login API to ensure that the user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations.

The responses is a JSON message.

### Login

**Description**
Log in to API Store.

**URI**
http://localhost:9763/store/site/blocks/user/login/ajax/login.jag

**URI Parameters**
action=login&username=<username>&password=<password>

**HTTP Methods**
POST

**Example**
curl -X POST -c cookies http://localhost:9763/store/site/blocks/user/login/ajax/login.jag -d 'action=login&username=admin&password=admin'

### Logout

**Description**
Log out from API Store.

**URI**

**URI Parameters**
?action=logout

**HTTP Methods**
GET

**Example**

### User Signup

**Description**
Add a new API Consumer.

**URI**
http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag

**URI Parameters**
action=addUser&username=<username>&password=<password>&allFieldsValues=<first_name>|<last_name>|<organization>|<country>|<email>|<land>|<mobile>|<IM>|<url>

**HTTP Methods**
POST

**Example**

### Search APIs

**Description**
Search for APIs using a given query.

**URI**
http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag
## Get all Paginated Published APIs

<table>
<thead>
<tr>
<th>Description</th>
<th>Get a list of all published APIs in paginated form so that browsing is easier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag">http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag</a></td>
</tr>
<tr>
<td>URI Parameters</td>
<td>action=getAllPaginatedPublishedAPIS, tenant, start, end, returnAPITags (optional)</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>GET</td>
</tr>
<tr>
<td>Example</td>
<td>To get the first 100 APIs in the API Store:</td>
</tr>
<tr>
<td></td>
<td>curl -b cookies &quot;<a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllPaginatedPublishedAPIS&amp;tenant=carbon.super&amp;start=1&amp;end=100">http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllPaginatedPublishedAPIS&amp;tenant=carbon.super&amp;start=1&amp;end=100</a>&quot;</td>
</tr>
</tbody>
</table>

Please note that the `getAllPublishedAPIS` API is now deprecated. You can get the same functionality from `getAllPaginatedPublishedAPIS`.

## Add an Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Add a new application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI Parameters</td>
<td>action=addApplication&amp;application=xxx&amp;tier=xxx&amp;description=xxx</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>POST</td>
</tr>
</tbody>
</table>

## Update an Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Update an existing application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI Parameters</td>
<td>action=updateApplication&amp;applicationOld=&lt;application_name&gt;&amp;applicationNew=&lt;application_name&gt;&amp;callbackUrlNew=xxx&amp;descriptionNew=xxx&amp;tier=&lt;tier_name&gt;</td>
</tr>
<tr>
<td>HTTP Methods</td>
<td>POST</td>
</tr>
</tbody>
</table>

## Get Applications
<table>
<thead>
<tr>
<th>Description</th>
<th>Get list of applications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><code>http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag</code></td>
</tr>
<tr>
<td><strong>URI Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?action=getApplications</td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>GET</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>curl -b cookies <code>http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag?action=getApplications</code></td>
</tr>
</tbody>
</table>

Get an Application by Name

<table>
<thead>
<tr>
<th>Description</th>
<th>Get details of a single application by name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><code>http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag</code></td>
</tr>
<tr>
<td><strong>URI Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?action=getApplicationByName&amp;applicationName=$APP_NAME</td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>GET</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>curl -b cookies <code>http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag?action=getApplicationByName&amp;applicationName=$APP_NAME</code></td>
</tr>
</tbody>
</table>

Remove an Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Remove an existing application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td><code>http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag</code></td>
</tr>
<tr>
<td><strong>URI Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>action=removeApplication&amp;application=&lt;application_name&gt;</td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
</tbody>
</table>

Generate an Application Key

<table>
<thead>
<tr>
<th>Description</th>
<th>Generate the key and secret values for a new application.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>action=generateApplicationKey&amp;application=&lt;app_name&gt;&amp;keytype=&lt;PRODUCTION</td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
</tbody>
</table>

Update an Application Key

<table>
<thead>
<tr>
<th>Description</th>
<th>Update the key and secret values for an application.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>action=updateClientApplication&amp;application=&lt;app_name&gt;&amp;keytype=&lt;PRODUCTION</td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2015-2018
### Add a Subscription

**Description**
Add a new API subscription.

**URI**

**URI Parameters**
- By application name: `action=addAPISubscription&name=xxx&version=xxx&provider=xxx&tier=xxx&applicationName=xxx`
- By application ID: `action=addSubscription&name=xxx&version=xxx&provider=xxx&tier=xxx&applicationId=xxx`

**HTTP Methods**
POST

**Examples**

### List Subscriptions

**Description**
List all applications with active subscriptions, along with the access key information of each application.

**URI**
http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag

**URI Parameters**
- action=getAllSubscriptions, selectedApp (optional)
  - You can give an application's name in the selectedApp parameter. The API then returns the given application's subscribed APIs and access key information. If you do not specify this parameter, only the first application in the retrieved application list will contain subscribed API details, in addition to the access key information.

**HTTP Methods**
GET

**Examples**
1. `curl -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions`

### List Subscriptions by Application

**Description**
List all API subscriptions of a given application.

**URI**
http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag

**URI Parameters**
- action=getSubscriptionByApplication&app=<application_name>

**HTTP Methods**
GET

**Examples**
1. `curl -b cookies http://localhost:9763/store/site/blocks/subscription/list/ajax/subscription-list.jag?action=getSubscriptionByApplication&app=DefaultApplication`
2. `curl -b cookies 'http://localhost:9763/store/site/blocks/subscription/list/ajax/subscription-list.jag?action=getSubscriptionByApplication&app=DefaultApplication&selectedApp=NewApp1'`
### List Subscriptions by API

<table>
<thead>
<tr>
<th>Description</th>
<th>List all subscriptions of a given API.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI</strong></td>
<td><code>http://localhost:9763/store/site/blocks/subscriptions/subscription-list/ajax/subscription-list.jag</code></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td><code>action=getSubscriptionByAPI&amp;apiName=xxx&amp;apiVersion=xxx&amp;provider=xxx</code></td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>GET</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>curl -b cookies 'http://localhost:9763/store/site/blocks/subscriptions/subscription-list/ajax/subscription-list.jag?action=getSubscriptionByAPI&amp;apiName=MyAPI&amp;apiVersion=1.0.0&amp;provider=admin'</code></td>
</tr>
</tbody>
</table>

### Remove a Subscription

<table>
<thead>
<tr>
<th>Description</th>
<th>Remove an API subscription.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI</strong></td>
<td><code>http://localhost:9763/store/site/blocks/subscriptions/subscription-remove/ajax/subscription-remove.jag</code></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>By application name:</strong> <code>action=removeSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx</code></td>
</tr>
<tr>
<td></td>
<td><strong>By application Id:</strong> <code>action=removeSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;applicationId=xxx</code></td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>By application Name:</strong> <code>curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscriptions/subscription-remove/ajax/subscription-remove.jag -d 'action=removeSubscription&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin&amp;applicationName=DefaultApplication'</code></td>
</tr>
<tr>
<td></td>
<td><strong>By application Id:</strong> <code>curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscriptions/subscription-remove/ajax/subscription-remove.jag -d 'action=removeSubscription&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin&amp;applicationId=1'</code></td>
</tr>
</tbody>
</table>

### Delete an OAuth Application

<table>
<thead>
<tr>
<th>Description</th>
<th>Deletes an OAuth application in a third-party Authorization Server. If you delete it through the API Store UI, only the mapping that is maintained in the API Manager side will be deleted.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI</strong></td>
<td><code>http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag</code></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td><code>action=deleteAuthApplication&amp;consumerKey=&lt;application_key&gt;</code></td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>curl -k -X POST -b cookies http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag -d 'action=deleteAuthApplication&amp;consumerKey=4lHddsxCtpFa2zJE1EbBpJy_NIQa'</code></td>
</tr>
</tbody>
</table>

### Provision an Out-of-Band OAuth Client

<table>
<thead>
<tr>
<th>Description</th>
<th>Provisions an OAuth client that was created out-of-band.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI</strong></td>
<td><code>http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag</code></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td><code>action=mapExistingOauthClient&amp;application=&lt;APPLICATION_NAME&gt;&amp;keytype=PRODUCTION/SANDBOX&amp;callbackUrl=&lt;URL&gt;&amp;authorizedDomains=&lt;authorized_domains&gt;&amp;mapExistingOauthClient=&lt;APPLICATION NAME&gt;&amp;validityTime=&lt;time duration in seconds&gt;&amp;client_id=&lt;client-ID&gt;</code></td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag -d 'action=mapExistingOauthClient&amp;application=http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag&amp;keytype=PRODUCTION&amp;callbackUrl=google.com&amp;authorizedDomains=ALL&amp;validityTime=3600&amp;client_id=mPbgvinvpEk1QcSrw962TLICriUa&amp;&lt;Authorized_domains&gt;=&quot;The domains from which requests are allowed to the APIs.&quot;</code></td>
</tr>
</tbody>
</table>

### Clean Partially Created Keys

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI</strong></td>
<td><code>http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag</code></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td><code>action=mapExistingOauthClient&amp;application=&lt;APPLICATION_NAME&gt;&amp;keytype=PRODUCTION&amp;callbackUrl=google.com&amp;authorizedDomains=ALL&amp;validityTime=3600&amp;client_id=mPbgvinvpEk1QcSrw962TLICriUa&amp;&lt;Authorized_domains&gt;=&quot;The domains from which requests are allowed to the APIs.&quot;</code></td>
</tr>
<tr>
<td><strong>HTTP Methods</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag -d 'action=mapExistingOauthClient&amp;application=http://localhost:9763/store/site/blocks/subscriptions/subscription-add/ajax/subscription-add.jag&amp;keytype=PRODUCTION&amp;callbackUrl=google.com&amp;authorizedDomains=ALL&amp;validityTime=3600&amp;client_id=mPbgvinvpEk1QcSrw962TLICriUa&amp;&lt;Authorized_domains&gt;=&quot;The domains from which requests are allowed to the APIs.&quot;'</code></td>
</tr>
</tbody>
</table>
### Cleans any partially created keys from the API Manager database, before adding a new subscription.

**Description**

Cleans any partially created keys from the API Manager database, before adding a new subscription. Partially created keys can remain in the API Manager databases when an OAuth application of a third-party authorization server gets deleted using the API Store UI. It only deletes the mapping that is maintained in the API Manager side.

**URI**


**URI Parameters**

- action=cleanUpApplicationRegistration
- applicationName=x
- keyType=PRODUCTION/SANDBOX

**HTTP Methods**

POST

**Example**

```bash
curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d 'action=cleanUpApplicationRegistration&applicationName=DefaultApplication&keyType=PRODUCTION'
```

### Get all Documentation

**Description**

Get all documents create for a given API

**URI**

http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag

**URI Parameters**

- action=getAllDocumentationOfApi
- name=xxx
- version=x.x.x
- provider=xxx

**HTTP Methods**

GET

**Example**

```bash
curl -b cookies "http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllDocumentationOfApi&name=PhoneVerification&version=1.0.0&provider=admin"
```

### Get the Contents of a File Document

**Description**

Get the contents of a file that is attached to API documentation of type 'File'

**URI**


**URI Parameters**

- action=getFileDocumentByFilePath
- filePath=xxx

**HTTP Methods**

GET

**Example**

```bash
```

### Add an API Comment

**Description**

Add a comment to an API.

**URI**

http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag

**URI Parameters**

- action=addComment
- name=xxx
- version=xxx
- provider=xxx
- comment=xxx

**HTTP Methods**

POST

**Example**

```bash
curl -X POST -b cookies http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag -d 'action=addComment&name=CalculatorAPI&version=1.0&provider=admin&comment=test comment'
```

### Get all Endpoint URLs

**Description**

Get all the endpoint URLs of the API Gateway environments configured for an API.

**URI**


**URI Parameters**

**HTTP Methods**

GET

**Example**

```bash
### Get all Available Tiers

**Description**
Get all the tiers available in the deployment.

**URI**
http://localhost:9763/store/site/blocks/item-add/ajax/add.jag

**HTTP Methods**
GET

**Example**

### Update Grant Types

**Description**
Edit default grant types and add new grant types

**URI**

**URI Parameters**
action=updateClientApplication&application=<Application_Name>&keytype=<Type of the key>&jsonParams=<URL encoded JSON>&callbackUrl=<callBackURL>

**HTTP Methods**
GET

**Example**
curl -b cookies 'https://localhost:9443/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag?action=updateClientApplication&application=DefaultApplication&keytype=PRODUCTION&jsonParams=%7B%22grant_types%22%3A%22refresh_token,urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer,password,iwa%3Antlm%2Cclient_credentials%22%7D&callbackUrl=' -k -b cookies

To create a list of the grant types to be encoded

1. Write a JSON string with the required grant types.

   ```json
   {"grant_types":"refresh_token,urn:ietf:params:oauth:grant-type:saml2-bearer,password,iwa:ntlm,client_credentials"}
   ```

2. Encode them with a URL encoder.
3. Use the encoded value for the `jsonParams` parameter as shown in the sample cURL command given above.

You can also invoke these APIs using mutual SSL authentication. Follow the instructions below to enable this:

1. Go to `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` and set the `clientAuth` attribute to want.

   ```xml
   port="9443"
   ...>
   clientAuth="want"
   ...
   
   </Connector>
   ```

2. For each Store API, attach the X509Certificate and pass the MutualAuthUserName parameter in the header.
3. Ensure that both client and the server have each other's certificates in the trust store.
Working with Audit Logs

Auditing is a primary requirement when it comes to monitoring production servers. For examples, DevOps need to have a clear mechanism for identifying who did what, and to filter possible system violations or breaches.

Audit logs or audit trails contain a set of log entries that describe a sequence of actions that occurred over a period of time. Audit logs allow you to trace all the actions of a single user, or all the actions or changes introduced to a certain module in the system etc. over a period of time. For example, it captures all the actions of a single user from the first point of logging in to the server.

Audit logs are enabled by default in WSO2 API Manager (WSO2 API-M) via the following configurations, which are in the `<API-M_HOME>/repository/conf/log4j.properties` file.

```
# Configure audit log for auditing purposes
log4j.logger.AUDIT_LOG=INFO, AUDIT_LOGFILE
log4j.appender.AUDIT_LOGFILE=org.apache.log4j.DailyRollingFileAppender
log4j.appender.AUDIT_LOGFILE.File=${carbon.home}/repository/logs/audit.log
log4j.appender.AUDIT_LOGFILE.Append=true
log4j.appender.AUDIT_LOGFILE.layout=org.wso2.carbon.utils.logging.TenantAwarePatternLayout
log4j.appender.AUDIT_LOGFILE.layout.ConversionPattern=[%d] %P%5p - %x %m %n
log4j.appender.AUDIT_LOGFILE.layout.TenantPattern=%U%@%D [%T] [%S]
log4j.appender.AUDIT_LOGFILE.threshold=INFO
log4j.additivity.AUDIT_LOG=false
```

The audit logs that get created when running WSO2 API-M are stored in the `audit.log` file, which is located in the `<API-M_HOME>/repository/logs` directory.

Audit log actions

In WSO2 API-M, audit logs can be enabled for the following user actions in the Publisher and Store.

### Publisher

<table>
<thead>
<tr>
<th>Action</th>
<th>Sample Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign in to the Publisher</td>
<td>[2017-06-07 22:26:22,506] INFO - '<a href="mailto:devona@carbon.super">devona@carbon.super</a> [-1234]' logged in at [2017-06-07 22:26:22,501+05</td>
</tr>
<tr>
<td>Create an API</td>
<td>[2017-06-07 22:28:06,027] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;created&quot;,&quot;typ&quot;:&quot;API&quot;,&quot;info&quot;:{&quot;provider&quot;:&quot;admin&quot;,&quot;name&quot;:&quot;PhoneVerification&quot;}]</td>
</tr>
<tr>
<td>Update an API</td>
<td>[2017-06-08 10:22:49,657] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;updated&quot;,&quot;typ&quot;:&quot;API&quot;,&quot;info&quot;:{&quot;provider&quot;:&quot;admin&quot;,&quot;name&quot;:&quot;PhoneVerification&quot;}]</td>
</tr>
<tr>
<td>Delete an API</td>
<td>[2017-06-08 10:15:55,369] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;deleted&quot;,&quot;typ&quot;:&quot;API&quot;,&quot;info&quot;:{&quot;provider&quot;:&quot;admin&quot;,&quot;name&quot;:&quot;PhoneVerification&quot;}]</td>
</tr>
</tbody>
</table>

### Store

<table>
<thead>
<tr>
<th>Action</th>
<th>Sample Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign in to the Store</td>
<td>[2017-06-07 22:34:54,684] INFO - '<a href="mailto:admin@carbon.super">admin@carbon.super</a> [-1234]' logged in at [2017-06-07 22:34:54,682+0</td>
</tr>
<tr>
<td>Create an application</td>
<td>[2017-06-07 22:40:17,625] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;created&quot;,&quot;typ&quot;:&quot;Application&quot;,&quot;info&quot;:{&quot;type&quot;:&quot;Gold&quot;,&quot;application_id&quot;:1}]</td>
</tr>
<tr>
<td>Update an application</td>
<td>[2017-06-07 22:44:25,931] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;updated&quot;,&quot;typ&quot;:&quot;Application&quot;,&quot;info&quot;:{&quot;type&quot;:&quot;APPROVED&quot;,&quot;application_id&quot;:1}]</td>
</tr>
<tr>
<td>Delete an application</td>
<td>[2017-06-07 22:45:59,093] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;deleted&quot;,&quot;typ&quot;:&quot;Application&quot;,&quot;info&quot;:{&quot;application_id&quot;:1}]</td>
</tr>
<tr>
<td>Subscribe to an application</td>
<td>[2017-06-07 22:36:48,826] INFO - [&quot;performerBy&quot;:&quot;admin&quot;,&quot;action&quot;:&quot;created&quot;,&quot;typ&quot;:&quot;Subscription&quot;,&quot;info&quot;:{&quot;application_id&quot;:1}]</td>
</tr>
</tbody>
</table>
Enabling Authentication Session Persistence

This topic is regarding sessions in the WSO2 API Manager (WSO2 API-M) and the process of enabling session persistence for these sessions. This is particularly useful when the remember me option is selected when logging into either the service provider or the WSO2 API-M.

Uncomment the following configuration in the `<API-M_HOME>/repository/conf/identity/identity.xml` file, under the `Server` and `JDBCPersistenceManager` elements to enable authentication session persistence.

```xml
<SessionDataPersist>
  <Enable>true</Enable>
  <Temporary>false</Temporary>
  <PoolSize>100</PoolSize>
  <SessionDataCleanUp>
    <Enable>true</Enable>
    <CleanUpTimeout>20160</CleanUpTimeout>
    <CleanUpPeriod>1140</CleanUpPeriod>
  </SessionDataCleanUp>
  <OperationDataCleanUp>
    <Enable>true</Enable>
    <CleanUpPeriod>720</CleanUpPeriod>
  </OperationDataCleanUp>
</SessionDataPersist>
```

The following table describes the elements of the configurations mentioned above.

<table>
<thead>
<tr>
<th>Configuration element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>This enables the persistence of session data. Therefore, this must be configured to <code>true</code> if you wish to enable session persistence.</td>
</tr>
<tr>
<td>Temporary</td>
<td>Setting this to <code>true</code> enables persistence of temporary caches that are created within an authentication request.</td>
</tr>
<tr>
<td>PoolSize</td>
<td>To improve performance, OAuth2 access tokens are persisted asynchronously in the database using a thread pool. This value refers to the number of threads in that thread pool.</td>
</tr>
<tr>
<td>SessionDataCleanUp</td>
<td>This section of the configuration is related to the cleaning up of session data.</td>
</tr>
<tr>
<td>Enable</td>
<td>Selecting <code>true</code> here enables the cleanup task and ensures that it starts running.</td>
</tr>
<tr>
<td>CleanUpTimeout</td>
<td>This is the timeout value (in minutes) of the session data that is removed by the cleanup task. The default value is 2 weeks.</td>
</tr>
<tr>
<td>CleanUpPeriod</td>
<td>This is the time period (in minutes) that the cleanup task would run. The default value is 1 day.</td>
</tr>
<tr>
<td>OperationDataCleanUp</td>
<td>This section of the configuration is related to the cleaning up of operation data.</td>
</tr>
</tbody>
</table>

**Note:** If Single Sign-On is to work, you must enable at least one of the two configurations mentioned in this topic.

### Related Topics
- See Configuring Single Sign-on with SAML2 for more information

### Using the Registry REST API

The registry REST API can be used to perform CRUD operations on registry resources. This is not packed with WSO2 API Manager by default. Follow the steps below to use the registry REST API with WSO2 API Manager.

1. Download the registry REST API webapp.
2. Copy the webapp to `<API-M_HOME>/repository/deployment/server/webapps`.
3. Invoke the registry REST API. For an example, to get the content of the `app-tiers.xml` file, in the registry path `/system/governance/apimgt/applicationdata`, the following curl command can be used:

   ```bash
curl -X GET http://localhost:9443/admin/apimgt/applicationdata/app-tiers.xml
   ```

   (Replace `localhost` with the IP address of your WSO2 API Manager instance.)
For a complete reference of the available REST API operations, go to Resources with REST API.
FAQ

• About WSO2 API Manager
  • What is WSO2 API Manager?
  • What is the open source license of the API Manager?
  • Is there commercial support available for WSO2 API Manager?
  • What are the default ports opened in the API Manager?
  • What are the technologies used underneath WSO2 API Manager?
  • Can I get involved in API-M development activities?
  • What is the default communication protocol of the API Manager?
  • Does WSO2 API-M support HTTP pipelining?

• Installation and start up
  • What are the minimum requirements needed to run WSO2 API Manager?
  • Which MySQL database script should I use?
  • How do I deploy a third-party library into the API Manager?
  • Can I connect the API Manager directly to an LDAP or Active Directory where corporate identities are stored?
  • Can I extend the management console UI to add custom UIs?
  • I don’t want some of the features that come with WSO2 API Manager. Can I remove them?
  • How can I change the memory allocation for the API Manager?
  • How do I start up only selected components of the API Manager?

• Deployment
  • What are the different deployment patterns and clustering configurations of the API Manager?
  • What are the container technologies that are supported in API Manager?
  • Is it recommended to run multiple WSO2 products on a single server?
  • Can I install features of other WSO2 products to the API Manager?
  • How can I continue to use my email address as the username in a distributed API-M deployment?

• Functionality
  • Why can’t I see all the APIs that I published on the API Store?
  • When editing an API’s resource parameters, how can I add multiple options to the Response Content Type parameter?
  • Why are the changes I did to the Response Content Type resource parameter of a published API not reflected in the API Store, even after saving?
  • How do I change the pass-through transport configurations?
  • How can I extend the default API Manager server by installing new features?
  • How can I preserve the CDATA element tag in API responses?

• Authentication and security
  • How can I manage authentication centrally in a clustered environment?
  • How can I manage the API permissions/visibility?
  • How can I add security policies (UT, XACML, etc.) for the services?
  • How can I enable self signup to the API Store?
  • How can I disable self signup to the API Store? I want to engage my own approval mechanism.
  • Is there a way to lock a user’s account after a certain number of failed login attempts to the API Store?
  • How do I change the default admin password and what files should I edit after changing it?
  • How can I recover the admin password used to log in to the management console?
  • How can I manage session timeouts for the management console?
  • How can I add authentication headers to messages going out of the API Gateway to the backend?
  • Can I give special characters in the passwords that appear in the configuration files?
  • How can I protect my product server from security attacks caused by weak ciphers?

• Troubleshooting
  • Why do I get an illegal access attempt error while trying to authenticate APIKeyValidationService?
  • How can I fix a mismatching certificate hostname exception?
  • How can I fix a EntityMismatch: our entity version is 1 but your entity version is 2 exception?
  • How can I fix a fatal alert: unknown_ca error when invoking the methods of an API via the API Console?
  • I get a Hostname verification failed exception when trying to send requests to a secured endpoint. What should I do?
  • When I add new users or roles, I get an error message stating that the entered user name is not conforming to policy. What should I do?
  • When I call a REST API, a lot of temporary files are created in my server and takes up a lot of space. What should I do?
  • Why do I get a Gateway Failures error?
  • How can I capture the state of a system?
  • How can I clean up the REG_LOG table?

About WSO2 API Manager

What is WSO2 API Manager?

WSO2 API Manager is a complete solution for creating, publishing and managing all aspects of an API and its life cycle. For more information, see the introduction.

What is the open source license of the API Manager?

Apache Software License Version 2.0

Is there commercial support available for WSO2 API Manager?
It is completely supported from evaluation to production. For more details, see WSO2 Support.

**What are the default ports opened in the API Manager?**

For a list of all default ports available, see Default Ports of WSO2 Products.

**What are the technologies used underneath WSO2 API Manager?**

The API Manager is built on top of WSO2 Carbon, an OSGi based components framework for SOA. For more details, see API Manager components.

**Can I get involved in API-M development activities?**

Not only are you allowed, but also encouraged. You can start by subscribing to the dev@wso2.org and architecture@wso2.org mailing lists. Feel free to provide ideas, feedback and help make our code better. For more information on contacts, mailing lists and forums, see our community page.

**What is the default communication protocol of the API Manager?**

The default communication protocol is Thrift.

**Does WSO2 API-M support HTTP pipelining?**

No, currently WSO2 API-M does not support HTTP pipelining.

### Installation and start up

**What are the minimum requirements needed to run WSO2 API Manager?**

For a list of system requirements, environment compatibility and required applications, see Installation Prerequisites.

**Which MySQL database script should I use?**

From Carbon kernel 4.4.6 onward, your product is shipped with two scripts for MySQL (click here to see if your product is based on this kernel version or newer):

- mysql.sql: Use this script for MySQL versions prior to version 5.7.
- mysql5.7.sql: Use this script for MySQL 5.7 and later versions.

MySQL 5.7 is only recommended for products that are based on Carbon 4.4.6 or a later version.

**How do I deploy a third-party library into the API Manager?**

Copy any third-party JARs to <API-M_HOME>/repository/components/lib directory and restart the server.

**Can I connect the API Manager directly to an LDAP or Active Directory where corporate identities are stored?**

Yes, you can configure the API Manager with multiple user stores. For more details, see Configuring User Stores.

**Can I extend the management console UI to add custom UIs?**

Yes, you can extend the management console (default URL is https://localhost:9443/carbon) easily by writing a custom UI component and simply deploying the OSGi bundle.

**I don’t want some of the features that come with WSO2 API Manager. Can I remove them?**

Yes, you can do this using the Features menu under the Configure menu of the management console (default URL is https://localhost:9443/carbon).

**How can I change the memory allocation for the API Manager?**

The memory allocation settings are stored in the <API-M_HOME>/bin/wso2server.sh file.

**How do I start up only selected components of the API Manager?**

Even though the API Manager bundles all components together, you can select which component(s) you want to start by using the -Dprofile command at product startup. For more information, see Product Profiles.
Deployment

What are the different deployment patterns and clustering configurations of the API Manager?

See Deploying WSO2 API Manager.

What are the container technologies that are supported in API Manager?

OpenShift, Docker, Kubernetes and Mesos are supported.

Is it recommended to run multiple WSO2 products on a single server?

This is not recommended in a production environment involving multiple transactions. If you want to start several WSO2 products on a single server, you must change their default ports to avoid port conflicts. See Changing the Default Ports with Offset.

Can I install features of other WSO2 products to the API Manager?

Yes, you can do this using the management console. The API Manager already has features of WSO2 Identity Server, WSO2 Governance Registry, WSO2 ESB etc. embedded in it. However, if you require more features of a certain product, it is recommended to use a separate instance of it rather than install its features to the API Manager.

How can I continue to use my email address as the username in a distributed API-M deployment?

To enable using your email (e.g., admin@wso2.com) as your username when deploying WSO2 API-M and WSO2 Identity Server (WSO2 IS), while doing master configurations, do the following.

Go to <API-M_HOME>/repository/conf/api-manager.xml. In the DataPublisher section, under ThrottlingConfiguration section, specify the username as follows: admin@wso2.com@carbon.super

The file only accepts configurations for the super tenant.

For more details, see Using Email Address as the Username.

How can I set up a reverse proxy server to pass server requests?

See Configuring the Proxy Server and the Load Balancer.

Functionality

Why can’t I see all the APIs that I published on the API Store?

If you have multiple versions of an API published, only the latest version is shown in the API Store. To display multiple versions, set the <DisplayMultipleVersions> element to true in the <API-M_HOME>/repository/conf/api-manager.xml file.

When editing an API’s resource parameters, how can I add multiple options to the Response Content Type parameter?

You cannot do this using the UI. Instead, edit the Swagger definition of the API as shown in the following example,

content_type: ["text/xml", "text/plain"]

Why are the changes I did to the Response Content Type resource parameter of a published API not reflected in the API Store, even after saving?

If you edited the Response Content Type using the UI, please open the API's Swagger definition, do your changes, and save. Then the changes should be reflected back in the API Store. This will be fixed in a future release.

How do I change the pass-through transport configurations?

If you have enabled the pass-through transport, you can change its default configurations by adding the following under the <transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener"> element in the <PRODUCT_HOME>/repository/conf/axis2/axis2.xml file. Be sure to stop the server before editing the file.

If you are using JDK 1.8.*, add the parameter given below:
How can I extend the default API Manager server by installing new features?

See Working with Features in the WSO2 Admin Guide.

How can I preserve the CDATA element tag in API responses?

Set the `javax.xml.stream.isCoalescing` property to `false` in the `<API-M_HOME>/XMLInputFactory.properties` file. Here’s an example:

```xml
<XacuteResponse xmlns="http://aaa/xI">  
  <Rowset>  
    <Row>  
      <outxml><![CDATA[<inSequence>  
        <send>  
          <endpoint>  
            <address uri="http://localhost:8080/my-webapp/echo"/>  
          </endpoint>  
        </send>  
        <inSequence>/]]></outxml>  
    </Row>  
  </Rowset>  
</XacuteResponse>
```

Authentication and security

How can I manage authentication centrally in a clustered environment?

You can enable centralized authentication using a WSO2 Identity Server based security and identity gateway solution, which enables SSO (Single Sign On) across all the servers.

How can I manage the API permissions/visibility?

To set visibility of the API only to selected user roles in the server, see API Visibility.

How can I add security policies (UT, XACML, etc.) for the services?

This should be done in the backend services in the Application Server or WSO2 ESB.

How can I enable self signup to the API Store?

See how to enable self signup.

How can I disable self signup to the API Store? I want to engage my own approval mechanism.

To disable the self signup capability, open the API-M management console and click the Resources > Browse menu. The registry opens. Navigate to the `/_system/governance/apimgt/applicationdata/sign-up-config.xml` file and set the `<SelfSignUp><Enabled>` element to `false`. To engage your own signup process, see Adding a User Signup Workflow.

Is there a way to lock a user’s account after a certain number of failed login attempts to the API Store?

If your identity provider is WSO2 Identity Server, this facility comes out of the box. If not, install the Account Recovery and Credentials Management feature (available under the User Management category) to the API Manager and configure it. For more information, see User Account Locking and Account Disabling in the Identity Server documentation. For more information on installing features, see Working with features in the Admin Guide.

How do I change the default admin password and what files should I edit after changing it?
To change the default admin password, log in to the management console with admin/admin credentials and use the Change my password option. After changing the password, do the following:

Change the following elements in the `<API-M_HOME>/repository/conf/api-manager.xml` file:

```xml
<AuthManager>
  <Username>admin</Username>
  <Password>newpassword</Password>
</AuthManager>

<APIGateway>
  <Username>admin</Username>
  <Password>newpassword</Password>
</APIGateway>

<APIKeyManager>
  <Username>admin</Username>
  <Password>newpassword</Password>
</APIKeyManager>
```

Go to the Resources > Browse menu in the management console to open the registry and update the credentials in the `/_system/governance/apimgt/applicationdata/sign-up-config.xml` registry location.

**How can I recover the admin password used to log in to the management console?**

Use the `<API-M_HOME>/bin/chpasswd.sh` script.

**How can I manage session timeouts for the management console?**

To configure session timeouts, see Configuring the session time-out.

**How can I add authentication headers to messages going out of the API Gateway to the backend?**

Uncomment the `<RemoveOAuthHeadersFromOutMessage>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file and set its value to `false`.

**Can I give special characters in the passwords that appear in the configuration files?**

If the config file is in XML, take care when giving special characters in the user names and passwords. According to XML specification (http://www.w3.org/TR/xml), some special characters can disrupt the configuration. For example, the ampersand character (&) must not appear in the literal form in XML files. It can cause a Java Null Pointer exception. You must wrap it with CDATA (http://www.w3schools.com/xml/xml_cdata.asp) as shown below or remove the character:

```xml
<Password>
  <![CDATA[ xnvYh?VHAkc?qZ4jv8556A4a,4M88h ]]> 
</Password>
```

**How can I protect my product server from security attacks caused by weak ciphers?**

You can protect your server from attacks such as the Logjam attack (Man-in-the-Middle attack) by disabling weak ciphers. For more details, see Disable weak ciphers in the WSO2 Admin Guide.

**Troubleshooting**

**Why do I get an illegal access attempt error while trying to authenticate APIKeyValidationService?**

If you get the following error: `org.wso2.carbon.server.admin.module.handler.AuthenticationHandler - Illegal access attempt.`, it may be due to the following reasons:

- Did you change the default admin password?
  If so, you need to change the credentials stored in the `<APIKeyValidator>` element of the `<API-M_HOME>/repository/conf/api-manager.xml` file of the API Gateway node(s).
- Have you set the priority of the SAML2SSOAuthenticator handler higher than that of the BasicAuthenticator handler in the authenticators.xml file?
If so, the SAML2SSOAuthenticator handler tries to manage the basic authentication requests as well. Set a lower priority to the SAML2SSOAuthenticator than the BasicAuthenticator handler as follows:

```xml
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
  <Priority>0</Priority>
  <Config>
    <Parameter name="LoginPage">/carbon/admin/login.jsp</Parameter>
    <Parameter name="ServiceProviderID">carbonServer</Parameter>
    <Parameter name="IdentityProviderSSOServiceURL">https://localhost:9444/samlssoserviceurl</Parameter>
    <Parameter name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
    <Parameter name="ISAuthnReqSigned">false</Parameter>
  </Config>
</Authenticator>
```

How can I fix a mismatching certificate hostname exception?

Reason for occurrence

The `javax.net.ssl.SSLException: hostname in certificate didn't match: <ip address> != <localhost>` exception is a very common exception that occurs whenever the WSO2 product server is accessed using a different IP address (e.g., https://10.100.0.77:9443/publisher) except localhost (e.g., https://localhost:9443/publisher).

The reason that the exception occurs is because the self-signed certificate that is shipped with WSO2 products is configured using the hostname as localhost, and as a result Apache Shindig does not allow any other HTTP requests that originate from other hostnames/IP addresses other than localhost.

Overcoming the issue

You have to create and add a certificate for the IP/domain name in order to overcome this issue. Follow the instructions below:

In the following instructions, assume that you are attempting to add a self-signed certificate for the domain 'foo.com'.

Step 1 - Create a self-signed Java KeyStore file and include your domain as the Common Name (CN)

1. Open a terminal and type the following command to generate a KeyStore.
   
   ```bash
   keytool -genkey -alias test.foo.com -keyalg RSA -keystore foo.jks -keysize 2048
   ```

2. Specify a preferred KeyStore password when prompted.
   
   ```bash
   Enter keystore password: <keystore_password>
   Re-enter new password: <keystore_password>
   ```

3. Enter the first name and last name as *.foo.com and fill out the other information accordingly when prompted.
Example

What is your first and last name?
[Unknown]: <new_host_name>

What is the name of your organizational unit?
[Unknown]:

What is the name of your organization?
[Unknown]: WSO2

What is the name of your City or Locality?
[Unknown]: Mountain View

What is the name of your State or Province?
[Unknown]: CA

What is the two-letter country code for this unit?
[Unknown]: US

Is CN=*\.foo.com, OU=Unknown, O=WSO2, L=Mountain View, ST=CA, C=US correct?
[no]: yes

4. Specify a preferred private Key password when prompted.

Enter key password for <keystore_password>
(RETURN if same as keystore password): <key_password>
Re-enter new password: <key_password>

* <key_password> - Enter the key password that you provided in step 1.2.

This generates a KeyStore with a private key and a public certificate with CN as *.foo.com

Step 2 - Configure the SSL KeyStore

Follow the instructions to configure the WSO2 product with the generated KeyStore:

1. Copy the generated self-signed keystore, foo.jks, which was created in step 1, into the <PRODUCT_HOME>/repository/resources/security directory.

2. Export the public certificate from the keystore and import that certificate to the client-truststore.jks file.
   a. Navigate to the <API-M_HOME>/repository/resources/security directory.
   b. Export the public certificate from the primary KeyStore.

   `keytool -export -alias test.foo.com -file test.foo.com -keystore foo.jks -storepass <KEYSTORE_PASSWORD_GIVEN_ABOVE>`

   c. Import the certificate to the client-truststore.jks file.

   `keytool -import -alias test.foo.com -file test.foo.com -keystore client-truststore.jks -storepass wso2carbon`

Step 3 - Update the KeyStoreFile and KeyStorePass parameters of the Tomcat HTTPS connector

1. Change the keystoreFile and keystorePass parameter of the Server.Service.Connector configuration with regard to port 9443 in the <API-M_HOME>/repository/conf/tomcat/catalina-server.xml file as follows, in order to locate the new SSL KeyStore.
1. Configure the new key store

Step 4 - Configure the new key store

Update the `<Password>`, `<KeyAlias>`, `<KeyPassword>` values under the `<KeyStore>` field in the `<API-M_HOME>/repository/conf/carbon.xml` file based on your new key store configuration.

```
<KeyStore>
    <!-- Keystore file location-->
    <Location>${carbon.home}/repository/resources/security/foo.jks</Location>
    <!-- Keystore type (JKS/PKCS12 etc.)-->
    <Type>JKS</Type>
    <!-- Keystore password-->
    <Password><KEYSTORE_PASSWORD></Password>
    <!-- Private Key alias-->
    <KeyAlias><NAME_OF_THE_ALIAS></KeyAlias>
    <!-- Private Key password-->
    <KeyPassword><KEY_PASSWORD></KeyPassword>
</KeyStore>
```

How can I fix a fatal alert: unknown_ca error when invoking the methods of an API via the API Console?

The root cause for the `javax.net.ssl.SSLException: Received fatal alert: unknown_ca` error is because the default pack is not shipped with a CA-signed certificate. When using the API Console, the web browser sends an HTTPS request to the API Gateway. As the certificate on the Gateway is not CA-signed, the browser does not accept it.

To resolve this issue, first access the Gateway URL via a new browser tab of the same browser and accept the certificate from the browser.

I get a Hostname verification failed exception when trying to send requests to a secured endpoint. What should I do?

Set the `<parameter name="HostnameVerifier">` element to `AllowAll` in `<API-M_HOME>/repository/conf/axis2/axis2.xml` file's HTTPS transport sender configuration. For example, `<parameter name="HostnameVerifier">AllowAll</parameter>`.

This parameter verifies the hostname of the certificate of a server when the API Manager acts as a client and does outbound service calls.
When I add new users or roles, I get an error message stating that the entered user name is not conforming to policy. What should I do?

This is because your user name or password length or any other parameter is not conforming to the RegEx configurations of the user store. See Managing Users and Roles.

When I call a REST API, a lot of temporary files are created in my server and takes up a lot of space. What should I do?

There might be multiple configuration context objects created per API invocation. Check whether your client creates a configuration context object per API invocation. You can also configure a HouseKeeping task in the `<API-M_HOME>/repository/conf/carbon.xml` file to clear the temporary folders. For example,

```xml
<HouseKeeping>
  <AutoStart>true</AutoStart>
  <!-- The interval in minutes, between house-keeping runs -->
  <Interval>10</Interval>
  <!-- The maximum time in minutes, temp files are allowed to live in the system. Files/directories which were modified more than "MaxTempFileLifetime" minutes ago will be removed by the house-keeping task -->
  <MaxTempFileLifetime>30</MaxTempFileLifetime>
</HouseKeeping>
```

Why do I get a Gateway Failures error?

The Gateway Failures UI error occurs when the ServerURL, username, password and/or GatewayEndpoint is incorrect. This can be rectified by checking and correcting the gateway configurations under `<Environments>` in the `<API-M>/repository/conf/api-manager.xml` file.

If you are using the API-M instance you used as the first instance in the Publish through Multiple API Gateways tutorial, you may receive the above error when trying out other tutorials. This is because you updated the environments configurations in that pack by adding two API Gateway environments under the `<Environments>` element, and commenting the `<environment>` element that comes by default. To overcome this error, uncomment the default configuration and delete the newly added configuration under `<Environments>` in the `<API-M>/repository/conf/api-manager.xml` file.

How can I capture the state of a system?

At the time of an error, you can use a tool called Carbon Dump (carbondump.sh) to collect all the necessary data (i.e., heap and thread dumps) from a running WSO2 API Manager instance in order to carryout a head dump and thread stack analysis. For more information on using this tool, see Capturing the state of the system in the Administration guide.

How can I clean up the REG_LOG table?

The REG_LOG table contains all the registry operations performed for all the registry resources in the system. When you clean up this table, you need to keep the latest record from every resource path to maintain atleast one resource reference in case of reindexing. Execute the following query to clean this table.

```
CREATE TABLE reg_log_ids_to_KEEP (  
  REG_LOG_ID INTEGER,  
  REG_TENANT_ID INTEGER
);

INSERT INTO reg_log_ids_to_KEEP (REG_LOG_ID, REG_TENANT_ID)  
SELECT MAX(REG_LOG_ID) AS REG_LOG_ID, REG_TENANT_ID FROM REG_LOG GROUP BY REG_PATH, REG_TENANT_ID;

DELETE FROM REG_LOG where REG_LOG_ID not in (SELECT REG_LOG_ID from reg_log_ids_to_KEEP);
drop table reg_log_ids_to_KEEP;

DELETE FROM REG_LOG WHERE REG_ACTION = 7;
```

Cleaning up the REG_LOG table periodically might be required if there is a large amount of data in the table, and as a result it takes a long time to process queries. Executing the query given above helps to improve the performance of the database.