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 following 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

If you are upgrading from WSO2 API Manager 2.5.0 to WSO2 API Manager 2.6.0, 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>
Changing the Carbon Database

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.

- Changing to Embedded Derby
- Changing to Embedded H2
- Changing to IBM DB2
- Changing to IBM Informix
- Changing to MariaDB
- Changing to MSSQL
- Changing to MySQL
- Changing to Oracle
- Changing to Oracle RAC
- Changing to PostgreSQL
- Changing to Remote H2

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_MTRIO2_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.

Configuring the System Administrator

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.

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.

<table>
<thead>
<tr>
<th>Securing Passwords in Configuration Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>The following topics will be covered under this section:</td>
</tr>
<tr>
<td>• Encrypting Passwords with Cipher Tool</td>
</tr>
<tr>
<td>• Encrypting passwords using the automated process</td>
</tr>
<tr>
<td>• Encrypting passwords manually</td>
</tr>
<tr>
<td>• Changing encrypted passwords</td>
</tr>
<tr>
<td>• Resolving Encrypted Passwords</td>
</tr>
<tr>
<td>• Enter password in command-line</td>
</tr>
<tr>
<td>• Start server as a background job</td>
</tr>
<tr>
<td>• Carbon Secure Vault Implementation</td>
</tr>
<tr>
<td>• Customizing the Secure Vault configuration</td>
</tr>
<tr>
<td>• Creating a Secret Callback Handler</td>
</tr>
<tr>
<td>• Creating a custom Secret Repository</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuring Transport-Level Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>The transport level security protocol of the Tomcat server is configured in the <code>&lt;PRODUCT_HOME&gt;/core/conf/tomcat/catalina-server.xml</code> file. Note that the <code>sslProtocol</code> attribute is set to &quot;TLS&quot; by default.</td>
</tr>
<tr>
<td>The following topics will guide you through the configuration options:</td>
</tr>
<tr>
<td>• Enabling TLS and disabling SSL support</td>
</tr>
<tr>
<td>• Disabling SSL support for products with JDK 1.8</td>
</tr>
<tr>
<td>• Disabling weak ciphers</td>
</tr>
<tr>
<td>• Disabling weak ciphers for the Tomcat transport</td>
</tr>
<tr>
<td>• Disables weak ciphers for the PassThrough transport</td>
</tr>
<tr>
<td>• Changing the server name in HTTP response headers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enabling Java Security Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>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 <code>&lt;PRODUCT_HOME&gt;/core/repository/conf/sec.policy</code> file. You modify this file to change the Java security permissions as required.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Using Asymmetric Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>• Understanding keystores</td>
</tr>
<tr>
<td>• Setting up keystores for WSO2 products</td>
</tr>
<tr>
<td>• Default keystore settings in WSO2 products</td>
</tr>
<tr>
<td>• Managing keystores</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Using Symmetric Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mitigating Cross Site Request Forgery Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>• How can CSRF attacks be harmful?</td>
</tr>
<tr>
<td>• Mitigating CSRF attacks</td>
</tr>
<tr>
<td>• Securing web applications</td>
</tr>
<tr>
<td>• Securing Jaggery applications</td>
</tr>
</tbody>
</table>
## 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 RemotepIpValve 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.

⚠️ 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.

```
```

```
```

```
https://localhost/apimanager/admin  ➔  https://localhost:9443/admin
```

⚠️ 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.

1. Open the `/etc/nginx/sites-enabled/wso2` file and enter the following configurations.
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_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_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_cookie_path /admin /apimanager/admin;
  }
}

2. Configure the reverseProxy for the following.

**Publisher**
Open the `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/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",
  //"regContext": "" // Use only if different path is used for
  registry
},

**Store**

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/store",
  //"regContext": "" // Use only if different path is used for
  registry
},

**Admin**

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"
  //"regContext": "" // 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.

```xml
<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 Update Manager 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.