WSO2 Administration Guide

Documentation

Carbon 4.4.x Platform
# Table of Contents

1. WSO2 Administration Guide ................................................................. 5
   1.1 Deployment and Clustering .......................................................... 5
      1.1.1 Key Concepts ........................................................................ 6
         1.1.1.1 Clustering Overview ......................................................... 6
         1.1.1.2 Load Balancing ............................................................... 7
         1.1.1.3 Separating the Worker and Manager Nodes ..................... 8
         1.1.1.4 Sharing Databases in a Cluster ....................................... 10
         1.1.1.5 Sticky Sessions with Manager Nodes ............................. 11
      1.1.2 Production Deployment Guidelines .......................................... 12
         1.1.2.1 Security Guidelines for Production Deployment ............. 19
         1.1.2.2 Troubleshooting in Production Environments ............... 25
      1.1.3 Configuring rsync for Deployment Synchronization ............... 28
   1.2 Data Stores ..................................................................................... 29
      1.2.1 Working with Databases ......................................................... 30
         1.2.1.1 Setting up the Physical Database .................................. 30
            1.2.1.1.1 Setting up IBM DB2 ............................................... 30
            1.2.1.1.2 Setting up IBM Informix ....................................... 33
            1.2.1.1.3 Setting up H2 ......................................................... 34
            1.2.1.1.4 Setting up MariaDB ............................................. 36
            1.2.1.1.5 Setting up Microsoft SQL .................................... 37
            1.2.1.1.6 Setting up MySQL .............................................. 38
            1.2.1.1.7 Setting up a MySQL Cluster ................................ 40
            1.2.1.1.8 Setting up Oracle ............................................... 40
            1.2.1.1.9 Setting up Oracle RAC ....................................... 41
            1.2.1.1.10 Setting up PostgreSQL ..................................... 43
            1.2.1.1.11 Setting up Derby .............................................. 44
         1.2.1.2 Changing the Carbon Database ......................................... 44
            1.2.1.2.1 Changing to Embedded Derby ................................ 45
            1.2.1.2.2 Changing to Embedded H2 ................................... 49
            1.2.1.2.3 Changing to IBM DB2 ........................................ 52
            1.2.1.2.4 Changing to IBM Informix .................................... 56
            1.2.1.2.5 Changing to MariaDB ........................................ 59
            1.2.1.2.6 Changing to MSSQL .......................................... 63
            1.2.1.2.7 Changing to MySQL ......................................... 67
            1.2.1.2.8 Changing to Oracle .......................................... 71
            1.2.1.2.9 Changing to Oracle RAC ................................... 75
            1.2.1.2.10 Changing to PostgreSQL .................................. 77
            1.2.1.2.11 Changing to Remote H2 .................................... 81
         1.2.1.3 Browsing the H2 Database .................................................. 86
      1.2.2 Working with the Registry ....................................................... 86
         1.2.2.1 Managing the Registry ..................................................... 87
            1.2.2.1.1 Metadata ............................................................. 88
            1.2.2.1.2 Properties .......................................................... 89
            1.2.2.1.3 Entries and Content .......................................... 90
            1.2.2.1.4 Role Permissions ............................................. 99
         1.2.2.2 Searching the Registry ..................................................... 100
      1.2.3 Using Remote Registry Instances for the Registry Partitions ....... 102
         1.2.3.1 All Partitions in a Single Server .................................... 102
         1.2.3.2 Config and Governance Partitions in a Remote Registry ...... 102
         1.2.3.3 Governance Partition in a Remote Registry ..................... 107
         1.2.3.4 Config and Governance Partitions in Separate Nodes ....... 111
      1.3 Security ....................................................................................... 115
         1.3.1 Maintaining Logins and Passwords ...................................... 158
         1.3.2 Securing Passwords in Configuration Files ......................... 161
            1.3.2.1 Encrypting Passwords with Cipher Tool ...................... 161

Copyright © WSO2 Inc. 2015-2019
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.2.2 Resolving Encrypted Passwords</td>
<td>168</td>
</tr>
<tr>
<td>1.3.2.3 Carbon Secure Vault Implementation</td>
<td>169</td>
</tr>
<tr>
<td>1.3.3 Configuring Transport Level Security</td>
<td>172</td>
</tr>
<tr>
<td>1.3.4 Enabling Java Security Manager</td>
<td>176</td>
</tr>
<tr>
<td>1.3.5 Using Asymmetric Encryption</td>
<td>180</td>
</tr>
<tr>
<td>1.3.5.1 Creating New Keystores</td>
<td>182</td>
</tr>
<tr>
<td>1.3.5.2 Renewing a CA-Signed Certificate in a Keystore</td>
<td>186</td>
</tr>
<tr>
<td>1.3.5.3 Configuring Keystores in WSO2 Products</td>
<td>188</td>
</tr>
<tr>
<td>1.3.5.4 Managing Keystores with the UI</td>
<td>192</td>
</tr>
<tr>
<td>1.3.6 Using Symmetric Encryption</td>
<td>193</td>
</tr>
<tr>
<td>1.3.7 Mitigating Cross Site Request Forgery Attacks</td>
<td>194</td>
</tr>
<tr>
<td>1.3.8 Mitigating Cross Site Scripting Attacks</td>
<td>199</td>
</tr>
<tr>
<td>1.3.9 Enabling HostName Verification</td>
<td>200</td>
</tr>
<tr>
<td>1.3.10 Configuring TLS Termination</td>
<td>201</td>
</tr>
<tr>
<td>1.4 Carbon Applications</td>
<td></td>
</tr>
<tr>
<td>1.4.1 Adding a Custom Proxy Path</td>
<td>202</td>
</tr>
<tr>
<td>1.4.2 Changing the Default Ports</td>
<td>203</td>
</tr>
<tr>
<td>1.4.2.1 Default Ports of WSO2 Products</td>
<td>208</td>
</tr>
<tr>
<td>1.4.3 Configuring SAML2 Single-Sign-On Across Different WSO2 Products</td>
<td>209</td>
</tr>
<tr>
<td>1.4.4 Configuring the Task Scheduling Component</td>
<td>214</td>
</tr>
<tr>
<td>1.4.5 Customizing Error Pages</td>
<td>215</td>
</tr>
<tr>
<td>1.4.6 Customizing the Management Console</td>
<td>217</td>
</tr>
<tr>
<td>1.4.7 Installing WSO2 Products as a Service</td>
<td>219</td>
</tr>
<tr>
<td>1.4.7.1 Installing as a Linux Service</td>
<td>224</td>
</tr>
<tr>
<td>1.4.7.2 Installing as a Windows Service</td>
<td>225</td>
</tr>
<tr>
<td>1.4.8 Managing Datasources</td>
<td>227</td>
</tr>
<tr>
<td>1.4.8.1 Adding Datasources</td>
<td>233</td>
</tr>
<tr>
<td>1.4.8.2 Configuring an RDBMS Datasource</td>
<td>234</td>
</tr>
<tr>
<td>1.4.8.2.1 Configuring the Datasource Provider</td>
<td>234</td>
</tr>
<tr>
<td>1.4.8.2.2 Configuring a JNDI Datasource</td>
<td>235</td>
</tr>
<tr>
<td>1.4.8.2.3 Configuring the Datasource Connection Pool Parameters</td>
<td>236</td>
</tr>
<tr>
<td>1.4.8.3 Configuring a Custom Datasource</td>
<td>237</td>
</tr>
<tr>
<td>1.4.8.9 Configuring a Custom Datasource</td>
<td>238</td>
</tr>
<tr>
<td>1.4.9 Managing Users, Roles and Permissions</td>
<td>240</td>
</tr>
<tr>
<td>1.4.9.1 Changing a Password</td>
<td>241</td>
</tr>
<tr>
<td>1.4.9.2 Configuring Roles</td>
<td>242</td>
</tr>
<tr>
<td>1.4.9.3 Configuring Users</td>
<td>243</td>
</tr>
<tr>
<td>1.4.9.4 Role-based Permissions</td>
<td>244</td>
</tr>
<tr>
<td>1.4.9.5 Managing User Attributes</td>
<td>248</td>
</tr>
<tr>
<td>1.4.9.6 Removing References to Deleted User Identities in WSO2 Products</td>
<td>251</td>
</tr>
<tr>
<td>1.4.10 Securing Carbon Applications</td>
<td>261</td>
</tr>
<tr>
<td>1.4.11 Working with Composite Applications</td>
<td>264</td>
</tr>
<tr>
<td>1.4.11.1 Introduction to Composite Applications</td>
<td>269</td>
</tr>
<tr>
<td>1.4.11.2 Packaging Artifacts into Composite Applications</td>
<td>269</td>
</tr>
<tr>
<td>1.4.11.3 Deploying Composite Applications in the Server</td>
<td>272</td>
</tr>
<tr>
<td>1.4.12 Working with Features</td>
<td>276</td>
</tr>
<tr>
<td>1.4.13 Working with Multiple Tenants</td>
<td></td>
</tr>
<tr>
<td>1.4.13.1 Introduction to Multitenancy</td>
<td>285</td>
</tr>
<tr>
<td>1.4.13.2 Configuring the Tenant Loading Policy</td>
<td>291</td>
</tr>
<tr>
<td>1.4.13.3 Adding New Tenants</td>
<td>291</td>
</tr>
<tr>
<td>1.4.14 Working with Transports</td>
<td>296</td>
</tr>
<tr>
<td>1.4.14.1 Introduction to Transports</td>
<td>298</td>
</tr>
<tr>
<td>1.4.14.2 Carbon Transports</td>
<td>298</td>
</tr>
<tr>
<td>1.4.14.2.1 HTTP Servlet Transport</td>
<td>299</td>
</tr>
<tr>
<td>1.4.14.2.2 HTTPS Servlet Transport</td>
<td>302</td>
</tr>
<tr>
<td>1.4.14.2.3 HTTP-NIO Transport</td>
<td>303</td>
</tr>
<tr>
<td>1.4.14.2.4 HTTPS-NIO Transport</td>
<td>304</td>
</tr>
<tr>
<td>1.4.14.2.5 VFS Transport</td>
<td>305</td>
</tr>
<tr>
<td>1.4.14.2.6 JMS Transport</td>
<td>307</td>
</tr>
<tr>
<td>1.4.14.2.7 MailToTransport</td>
<td>310</td>
</tr>
<tr>
<td>1.4.14.2.8 TCP Transport</td>
<td>311</td>
</tr>
<tr>
<td>1.4.14.2.9 Local Transport</td>
<td>312</td>
</tr>
<tr>
<td>1.4.14.2.10 UDP Transport</td>
<td>312</td>
</tr>
<tr>
<td>1.4.14.2.11 FIX Transport</td>
<td>313</td>
</tr>
</tbody>
</table>
## 1.5 Performance Tuning

---

## 1.6 Monitoring

- **1.6.1 Capturing System Data in Error Situations**
- **1.6.2 JMX-Based Monitoring**
- **1.6.3 Monitoring Logs**
  - **1.6.3.1 Monitoring Logs using Management Console**
  - **1.6.3.1.1 Configuring Log4j Properties**
  - **1.6.3.1.2 Configuring the Log Provider**
  - **1.6.3.1.3 View and Download Logs**
  - **1.6.3.2 HTTP Access Logging**
  - **1.6.3.3 Masking Sensitive Information in Logs**
- **1.6.4 Monitoring Message Flows**
- **1.6.5 Monitoring Performance Statistics**
- **1.6.6 Monitoring SOAP Messages**
- **1.6.7 Monitoring TCP-Based Messages**
  - **1.6.7.1 Starting TCPMon**
  - **1.6.7.2 Message Monitoring with TCPMon**
  - **1.6.7.3 Other Usages of TCPMon**
- **1.6.8 Monitoring with WSO2 Carbon Metrics**
  - **1.6.8.1 Setting Up Carbon Metrics**
    - **1.6.8.1.1 Enabling Metrics and Storage Types**
    - **1.6.8.1.2 Configuring Metrics Properties**
  - **1.6.8.2 Using JVM Metrics**
  - **1.6.8.3 Using Messaging Metrics**
- **1.6.9 Monitoring Server Health**

## 1.7 Getting WSO2 Updates

---

## 1.8 Reference Guide

- **1.8.1 Configuration Files**
  - **1.8.1.1 Configuring catalina-server.xml**
  - **1.8.1.2 Configuring master-datasources.xml**
  - **1.8.1.3 Configuring registry.xml**
  - **1.8.1.4 Configuring user-mgt.xml**
  - **1.8.1.5 Configuring config-validation.xml**
- **1.8.2 Database Upgrade Guide**
- **1.8.3 Product Startup Options**
- **1.8.4 Supported Cipher Suites**
- **1.8.5 Directory Structure of WSO2 Products**
- **1.8.6 WSO2 Patch Application Process**
Welcome to the WSO2 Administration Guide!

In this guide, you learn how to configure WSO2 products that are built on top of the WSO2 Carbon platform version 4.4.x. WSO2 Carbon is the base platform on which the products are built.

See the WSO2 release matrix to identify the products that are released for the Carbon 4.4.x platform. Once you have configured your product, see the product documentation on how to use each product.
Deployment and Clustering

See the following topics on how to deploy and cluster WSO2 products:

- Key Concepts
- Production Deployment Guidelines
- Configuring rsync for Deployment Synchronization

Key Concepts

The following are some key concepts associated with deployment and clustering.

- Clustering Overview
- Load Balancing
- Separating the Worker and Manager Nodes
- Sharing Databases in a Cluster
- Sticky Sessions with Manager Nodes

Clustering Overview

The following topics explain clustering basics:

- Introduction to clustering
- About membership schemes
- Clustering compatibility with WSO2 products
- Deciding how to set up your cluster

Introduction to clustering

You can install multiple instances of WSO2 products in a cluster. A cluster consists of multiple instances of a product that divide up the work and act as a single instance. This improves performance as requests are distributed among several servers instead of just one. It is also more reliable as one instance is there to handle requests when another becomes unavailable. Clustering provides the following benefits:

- **High availability**: Some systems require high availability percentages like two-nines (99%). A server may go down due to many reasons such as system failures, planned outage, or hardware or network problems. Clustering for high availability results in fewer service interruptions. Since downtime is costly to any business, clustering has a direct and positive impact on costs.
- **Increased scalability**: Scalability is the ability of a system to accommodate a growing amount of work. Scalability is using resources more effectively. By distributing processing, we can make vertical or horizontal scalability possible.
- **Failover and switchover capabilities**: Failover can occur automatically or manually. You can prepare a redundant backup system or use load-balanced servers to serve the failover function. You address failover through your system design and characteristics, and clustering helps you design your applications against interruptions and with improved recovery time. Even if a failover occurs, it is important to bring the system back up as quickly as possible.
- **Low cost**: Clustering improves scalability and fault tolerance, so business continuity is guaranteed even in the case of node failure. Also, it facilitates automatically scaling up the system when there is a burst load, which means the business will not lose any unforeseen opportunities.

These characteristics are essential for enterprise applications deployed in a production environment. You need a cluster when you go into production as that is when good performance and reliability are critical.

WSO2 provides Hazelcast Community Edition as its default clustering engine. For clustering on a secure channel (i.e., secure Hazelcast), use Hazelcast Enterprise. To integrate with Hazelcast Enterprise, there are provisions to provide license key under clustering configurations. Advanced users can fine-tune Hazelcast by creating a `<PRODUCT_HOME>/repository/conf/hazelcast.properties` file and adding the relevant Hazelcast properties as described in the Hazelcast Advanced Configuration Properties documentation. If you use Hazelcast Enterprise Edition or Hazelcast Management Center, see the Hazelcast documentation for details on configuring those products and also Advanced Configurations and Information for further details.

Add the following property to `hazelcast.properties` file to add the license key of Hazelcast Enterprise: `hazelcast.enterprise.license.key`.

### About membership schemes

A cluster should contain two or more instances of a product that are configured to run within the same domain. To make an instance a member of the cluster, configure it to either of the available membership schemes, which are as follows:

- Well Known Address (WKA) membership scheme
- Multicast membership scheme
- AWS membership scheme

All of these membership schemes are ready to be used in production. You can select based on your production environment. Here’s a comparison of the membership schemes:

<table>
<thead>
<tr>
<th>Multicast</th>
<th>WKA</th>
<th>AWS</th>
</tr>
</thead>
</table>

Copyright © WSO2 Inc. 2015-2019
All nodes should be in the same subnet | Nodes can be in different networks | Amazon EC2 nodes
---|---|---
All nodes should be in the same multicast domain | No multicasting requirement | No multicasting requirement
Multicasting should not be blocked | No multicasting requirement | No multicasting requirement
No fixed IP addresses or hosts required | At least one well-known IP address or host required. | No fixed IP addresses or hosts required
Failure of any member does not affect membership discovery | New members can join with some WKA nodes down, but not if all WKA nodes are down. | Failure of any member does not affect membership discovery
Does not work on IaaSs such as Amazon EC2 | IaaS-friendly | Works on Amazon EC2
No WKA requirement | Requires keepalive, elastic IPs, or some other mechanism for re-mapping IP addresses of WK members in cases of failure. | No WKA requirement

Note that some production environments do not support multicast. However, if your environment supports multicast, there are no issues in using this as your membership scheme.

### About Well-Known Addresses (WKA)

The Well-Known Addresses (WKA) feature is a mechanism that allows cluster members to discover and join a cluster using unicast instead of multicast. WKA is enabled by specifying a small subset of cluster members (referred to as WKA members) that are able to start a cluster. The WKA member starts the cluster and the other members join the cluster through this WKA member. When the WKA member is down, the cluster breaks, and the members cannot communicate with each other.

The system should have at least two well-known address (WKA) members in order to work correctly and to recover if a single WKA member fails.

### Clustering compatibility with WSO2 products

WSO2 products are compatible with each other if they are based on the same WSO2 Carbon version. See the release matrix for compatibility information.

### About performance of WSO2 products in a cluster

If you are setting up multiple WSO2 products in a cluster, it is recommended to set up each product on a separate server. For example, WSO2 ESB is used for message mediation, so a considerable amount of processing happens in the ESB. The DSS does data service hosting and has a different architecture layer from the ESB. If you deploy both the ESB and DSS in the same instance/runtime, it can negatively impact the performance of both, and it also makes scaling difficult. However, you can set up hybrid servers (installing selected DSS features on top of the ESB and vice versa) using WSO2 products without the above performance concerns.

### Deciding how to set up your cluster

When setting up your cluster, you must decide how you want to set up and share your databases, whether to front your cluster with a load balancer, and whether to use sticky sessions. You also need to make a decision on whether to separate the worker and manager nodes in the cluster.

#### Load Balancing

You cluster services in production environments to scale up applications and/or achieve high availability. By scaling up, the application can support more user requests and through high availability, the service is available even when a few servers are down.

You use a load balancer to distribute requests among the nodes in a cluster. The nodes that receive incoming traffic are a set of backend worker nodes. They are either pre-defined (static) or discovered dynamically. In the static mode, you cannot add new nodes to the pre-defined set of worker nodes at runtime. In the dynamic mode, you can add nodes to the load balancer at runtime without knowing the IPs and other connection details.

Among the many varieties of load balancers are hardware, DNS, transport-level (e.g., HTTP level like Apache Tomcat), and application-level load balancers (e.g., Synapse). High-level load balancers, like application-level load balancers, operate with more information about the messages they route and therefore, provide more flexibility but also incur more overhead. The choice of a load balancer is a trade-off between performance and flexibility.

There are many algorithms or methods for distributing the load between servers. Random or round-robin are simple approaches. More sophisticated
algorithms consider runtime properties in the system like the machine's load or the number of pending requests. The distribution can also be controlled by application-specific requirements like sticky sessions. With a reasonably diverse set of users, simple approaches perform as well as complex ones.

**Session affinity**

Stateful applications inherently do not scale well. State replication induces a performance overhead on the system. Instead of deploying stateful applications in a cluster, you can use session-affinity-based load balancing.

Session affinity ensures that, when a client sends a session ID, the load balancer forwards all requests containing the session ID to the same backend worker node, irrespective of the specified load balancing algorithm. Before the session is created, the request is dispatched to the worker node that is next-in-line and a session is established with that worker node.

**Separating the Worker and Manager Nodes**

WSO2 Carbon version 4.0.0 onwards supports deployment models that consist of 'worker' nodes and 'manager' nodes. A worker node serves requests received by clients, whereas a manager node deploys and configures artifacts (web applications, services, proxy services, etc.).

The worker/manager setup separates a WSO2 product's UI components, management console, and related functionality with its internal framework. Typically, the manager nodes are in read-write mode and authorized to make configuration changes. The worker nodes are in read-only mode and authorized only to deploy artifacts and read configurations.

**Why separate the worker and manager nodes**

1. **Improved security**: Manager nodes are typically behind a firewall that only allows admin access. They are exposed to clients running within the organization only, while worker nodes can be exposed to external clients.
2. **Proper separation of concerns**: Management nodes specialize in the management of the setup while worker nodes specialize in serving requests to deployment artifacts. Only management nodes are authorized to add new artifacts into the system or make configuration changes.
3. **Specific worker-node tasks**: Worker nodes can only deploy artifacts and read configuration. Worker nodes are limited only for specific tasks.
4. **Lower memory requirements**: There is a lower memory footprint in the worker nodes because the OSGi bundles related to the management console and UI are not loaded to them. This is also good for memory utilization.

**Worker/Manager separated clustering patterns**

Since all WSO2 products are built on the cluster-enabled Carbon platform, you can cluster them in the same way depending on which deployment pattern you use. The clustering pattern determines the process of separating the worker and manager nodes. You can select one of the following patterns based on your load and the target expenditure.

---

**Worker/Manager clustering deployment pattern 1**

This pattern involves two worker nodes in a cluster. The worker is in high-availability mode while the manager is not. This pattern is suitable in situations where it is rare to deploy applications or modify a running application and therefore, can run with only a single manager node. However, you need multiple worker nodes to ensure that the application runs continuously.

This mode is rarely used. The preferred mode is having two management nodes in Active/Passive mode as in deployment pattern 2.
This pattern has two manager nodes in one cluster and two worker nodes in a separate cluster. The management node is in Active/Passive mode for high availability. It is generally not recommended to put a manager node in the Active/Active mode, but if you want high availability for your data center and location-based services, it is useful.

This pattern is useful in scenarios where the application deployment/modification might be frequent and therefore, need a cluster of manager nodes. However, if the load is less, you can share a single load balancer with the worker cluster.

Worker/Manager clustering deployment pattern 3

This pattern has two manager nodes in one sub-domain and two worker nodes in a separate sub-domain. The manager and worker subdomains are parts of a single WSO2 product cluster domain. Both subdomains use their own load balancer while existing within the same cluster. Note that multiple load balancers require you to follow several unique configurations.

This pattern is similar to deployment pattern 2. However, the application/modification load (or any other administrative load) might be high, so there is a dedicated load balancer for the manager cluster to prevent this load from affecting the load of the worker cluster.
Sharing Databases in a Cluster

All WSO2 products are shipped with a default H2 database. WSO2 products use the underlying registry services in the WSO2 Carbon platform to establish their own registry space, which is utilized for storing data and persisting configurations. In addition to the registry space, all identity-related data and user permissions are also stored in this default database.

The registry space provided to each product contains three major partitions.

- The local data repository
- The configuration registry
- The governance registry

Each of these partitions can be separated in a clustered production environment. The following table provides more information on each partition and the type of data that would typically reside in them.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Description</th>
<th>Type of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local data repository</td>
<td>This partition of the registry space is specific to the node that the product resides in. This is not intended to be shared among multiple servers.</td>
<td>The local data repository contains system configuration as well as runtime data that is local to a single instance of a product. For example, the local data repository is used to store mount configurations that are local to each instance.</td>
</tr>
<tr>
<td>Configuration registry</td>
<td>These configurations can be shared across multiple instances of the same product (a cluster of ESB nodes for example). However, these cannot be shared in a cluster of multiple different products unless certain configurations are done.</td>
<td>The configuration registry, which is the most widely used partition of the registry space, contains product specific configuration.</td>
</tr>
<tr>
<td>Governance registry</td>
<td>The governance registry partition has been designed in a way that it can be made use of by multiple instances of various Carbon based products.</td>
<td>The governance registry contains data and configuration shared across the platform. The WSO2 Governance Registry makes use of this partition of the registry space to store services and related metadata such as WSDLs, schemas, policies and endpoints.</td>
</tr>
</tbody>
</table>

Databases in production environments

From most production environments, it is recommended to externalize the databases to a JDBC database of your choice and split the registry space to manage registry resources in a better way. The governance registry and configuration registry data can either be stored in a single database or in two databases (i.e., one for configuration and one for governance) depending on the amount of data used.

The following diagram depicts how the databases are configured in a typical WSO2 product cluster.
In the above diagram, the governance and configuration registry is shared for the whole WSO2 product cluster (assuming the cluster is comprised of the same WSO2 product; the configuration registry is not shared for different products). This means, each node in the cluster is configured to point to this database. These configurations involve changes to the `<PRODUCT_HOME>/repository/conf/registry.xml` file for each WSO2 product in the cluster.

The user management database is also shared among all nodes in the cluster, although the way it is shared differs slightly from the governance and configuration registry. The user management database is basically a user store and is configured using the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file for each WSO2 product in the cluster.

Furthermore, each WSO2 product has its own local data repository for runtime data.

Each of the WSO2 products in the cluster must have datasources defined for each of the databases that they point to. This is configured in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file for each WSO2 product.

**Mounting the registry**

The governance and configuration registry has to be mounted in order to be shared by all products in the cluster. These mounting configurations are done so that any changes to data in these databases are communicated to all nodes in the product cluster. In the case of a node failing, another node can take up the tasks required and will be able to do so as the governance and configuration registry is up to date across all the WSO2 product instances.

**Sticky Sessions with Manager Nodes**

When a single WSO2 product is interacting with a web application, a session object is created. It remains in the memory of the WSO2 product. However, in a clustered environment where you could have multiple WSO2 product servers fronted by a load balancer, the situation is different.

If the backend application is state-full, it may use sessions. You manage sessions in two different ways.

- **Using sticky sessions**: In this approach, once a session is created by the backend server, a session ID will be sent back to the client in the response message. This session ID will be tracked by the intermediate load balancers. If the user/client sent another request with the same session ID, that request will be sent to the same backend server.
- **Session replication**: In this approach, backend servers will replicate the sessions among all the nodes in the cluster, and the load balancers will be able to send any request to any node.

This session can be implemented at the HTTP level or at the SOAP level. One downside of this approach is if a node has failed, the sessions associated with that node are lost and need to be restarted. It is common to couple this solution with a shared database solution. With sticky sessions enabled, the session data is kept in memory, but persistent data is saved into a database.

If cluster nodes share state, and they are not stored in same shared persistent media like a database, all changes done at each node have to be disseminated to all other nodes in the cluster. Often, this is implemented using a group communication method. Group communication keeps track of the
members of groups of nodes defined by users and updates the group membership when nodes have joined or if they leave. When a message is sent using group communication, it guarantees that all nodes in the current group will receive the message. For WSO2 products, the clustering implementation uses Hazelcast as the form of group communication to disseminate all the changes to all other nodes in the cluster.

Production Deployment Guidelines

The requirements for deploying WSO2 products can change based on the deployment scenario and pattern. The recommendations in this topic are for general production use, assuming moderate load conditions. For situations where a high volume of traffic is expected and if there are large deployments, these guidelines may not be sufficient. See Troubleshooting in Production Environments for information on how to obtain and analyze information to solve production issues. The following are the topics addressed in this section.

Installation prerequisites

Prior to installing any WSO2 Carbon-based product, it is necessary to have the appropriate hardware and software for running the product.

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 | • One c5.large instance to run one WSO2 product instance.  
| Cassandra data nodes | • 4 core processors  
| • 8 GB RAM  
| For more information, see the Cassandra documentation on hardware recommendations for enterprise implementations.

Environment compatibility

- By default, WSO2 products are installed with OpenJDK, which allows you to run the product as soon as it is installed.

To use a different JDK, point the JAVA_HOME environment variable to the new JDK. Make sure your JDK version is compatible with the WSO2 product.

- All WSO2 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 in the Administration Guide. Also, 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.
- If you have difficulty in setting up any WSO2 product in a specific platform or database, contact us.

Installing the WSO2 product
Given below is how to install a WSO2 product:

Download and install the product

If the installation prerequisites are satisfied, follow the steps below:

1. Go to the product page and download the product installer (click Installer pkg).

   Note that there are several options for installing the product in various environments. Use the available links for more information on each option.

2. Double-click to open the installation wizard, which will guide you through the installation. When you finish, the product will be installed and ready for use.

Access the HOME directory

Let’s call the installation location of your product as the `<PRODUCT_HOME>` directory. This is located in a place specific to your OS as shown below:

<table>
<thead>
<tr>
<th>OS</th>
<th>Home directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS</td>
<td>/Library/WSO2/&lt;PRODUCT_NAME&gt;/&lt;VERSION&gt;</td>
</tr>
<tr>
<td>Windows</td>
<td>C:\Program Files\WSO2&lt;PRODUCT_NAME&gt;/&lt;VERSION&gt;</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>/usr/lib/wso2/&lt;PRODUCT_NAME&gt;/&lt;VERSION&gt;</td>
</tr>
<tr>
<td>CentOS</td>
<td>/usr/lib64/&lt;PRODUCT_NAME&gt;/&lt;VERSION&gt;</td>
</tr>
</tbody>
</table>

Uninstalling the product

To remove an already installed product, follow the instructions below:

<table>
<thead>
<tr>
<th>OS</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS</td>
<td>Open a terminal and run the following command as the root user:</td>
</tr>
<tr>
<td></td>
<td><code>sudo bash /Library/WSO2/&lt;PRODUCT_NAME&gt;/&lt;VERSION&gt;/uninstall.sh</code></td>
</tr>
<tr>
<td>Windows</td>
<td>Go to the Start Menu -&gt; Programs -&gt; WSO2 -&gt; Uninstall <code>&lt;PRODUCT_NAME_VERSION&gt;</code> or search Uninstall <code>&lt;PRODUCT_NAME_VERSION&gt;</code> and click the shortcut icon. This will uninstall the product from your computer.</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Open a terminal and run the following command:</td>
</tr>
<tr>
<td></td>
<td><code>sudo apt-get purge &lt;PRODUCT_DISTRIBUTION_NAME&gt;</code></td>
</tr>
<tr>
<td>CentOS</td>
<td>Open a terminal and run the following command:</td>
</tr>
<tr>
<td></td>
<td><code>sudo yum remove &lt;PRODUCT_DISTRIBUTION_NAME&gt;-x86_64</code></td>
</tr>
</tbody>
</table>

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 (i.e., the `<PRODUCT_HOME>/bin/wso2server.sh` file), 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.

  Be sure to set the org.wso2.ignoreHostnameVerification system property in the `<PRODUCT_HOME>/bin/wso2server.sh` file to false as follows:

  ```
  org.wso2.ignoreHostnameVerification=false
  ```

  This setting will enable hostname verification of HTTP requests and responses in the Carbon server, and thereby avoid security issues in production environments.

- **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 property file instead of in the Java code and secure the properties file with the secure vault.

  Note: 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.

---

**Running the product**

To run WSO2 products, you start the product server at the command line. You can then run the Management Console application to configure and manage the product.

**Before you begin**

- When you move into a production environment, it is recommended to grant restricted access to the management console. See Securing Carbon Applications for instructions.
- The `config-validation.xml` file in the `<PRODUCT_HOME>/repository/conf/etc` directory contains a list of recommended system parameters, which are validated against your system when the server starts. See Configuring config-validation.xml for details on modifying these parameters before starting the server.
- The Management Console uses the default HTTP-NIO transport, which is configured in the `catalina-server.xml` file in the `<PRODUCT_HOME>/repository/conf/tomcat` directory. This transport must be properly configured in this file for the management console to be accessible.
- As explained in the installation prerequisites, the default product installation uses OpenJDK. Therefore, you don't require a different JDK. However, if you have set up Oracle JDK or IBM JDK, be sure to apply the following settings to your product distribution.

**Oracle JDK/IBM JDK**

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 that you 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 `<PRODUCT_HOME>/repository/conf/tomcat/` directory.
2. Set the `compression` parameter (under each of the connector configurations) to false as shown below:

   ```
   compression="off"
   ```

3. Restart the server.

   If you are using IBM JDK 1.8, change the value of the org.wso2.csrfguard.PRGGuard.Provider property to "IBMJCE" in the `Owasp.CsrfGuard.d.Carbon.properties` file. This file is stored in the `<PRODUCT_HOME>/repository/conf/security/` directory.

**Starting the product profiles**

Open a command prompt and execute the name of the product distribution (For example, `wso2ei-6.30`):
Stopping the server

To stop the server, press Ctrl+C in the command window, or click the Shutdown/R restart link in the navigation pane in the Management Console.

Running recommendations for security

The following are security related recommendations to be followed when running the product.

- **Running as a different user**: For security reasons, it's recommended to run the product as an unprivileged user. After adding a user to the system, apply your organizational security policies to that user.
- **Running on a different port**: If you want to run on a different port, like port 80, the recommended way is to add a port forwarding rule from your firewall.
- **Running as a Unix daemon**: You have the option of running each product as a standard Unix service. You can start, stop, and restart the WSO2 product instances as follows.

```
# sh bin/wso2server.sh [start | stop | restart]
```

Tuning parameters

- The latency numbers (~50ms) are based on a two datacenter setup with a high-speed network connection. With the default configuration, you might notice intermittent behavior, so it is important to tune the system.
- It is not recommended to use Metrics for system monitoring (JVM, CPU, etc.) in a production deployment. You can use external monitoring tools for this purpose.

Hazelcast properties

WSO2 products use Hazelcast as its default clustering engine. The following configuration must be placed in the `<PRODUCT_HOME>/repository/conf/hazelcast.properties` file. Create this file if it does not exist.

```
hazelcast.shutdownhook.enabled=false
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.

Once you enable log4j for hazelcast as explained above, add log4j.logger.com.hazelcast=INFO to the `<PRODUCT_HOME>/repository/conf/log4j.properties` file. For more information on logging, see Monitoring Logs.

Additionally, Hazelcast indicates that if all members are not mentioned in the well-known member list, there can be a split-brain (network partition) situation. If the cluster spans across data centers, it is important to add all the members to the well-known members list in the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file.

Common guidelines and checklist

The following table lists out the common guidelines and details pertaining to them. These are common to all products and are followed for making an installed WSO2 product ready for production.

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Details</th>
</tr>
</thead>
</table>
### Security hardening

Guidelines for hardening the security of a WSO2 deployment in a production environment can be discussed under three high-level categories:

- **Product-level security**
- **OS-level security**
- **Network-level security**

See [Security Guidelines for Production Deployment](#) for the detailed list of security-related recommendations.

### 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 by setting the `HostName` parameter in the `carbon.xml` file:

```xml
<HostName>your.host.name</HostName>
```

To configure hostnames for WSDLs and endpoints, users are recommended to add the following parameter in the `<transportReceiver>` section in the `axis2.xml` file:

```xml
<parameter name="WSDLEPRPrefix" locked="false">[http]://your.host.name:[port]</parameter>
```

See the topics on changing hostnames shown below:

- Setting up hostnames and ports
- Changing the hostname

See [Working with Transports](#) for information on transports in WSO2 products.

### Registry and governance

All WSO2 products make use of an instance of a registry to store configurations. The registry uses a database as the persistent storage. By default, the registry uses an embedded H2 database. This embedded database might yield a lower performance and is less reliable compared to a standard database like MySQL when there are a large number of deployed artifacts. Hence, you should look at associated trade-offs, and we recommend that you switch to a database like Oracle, MySQL or MSSQL.

Moreover, it is worth noting that the default setup does not include database backup procedures. The production setup should obviously need to have regular database backup procedures configured.

When the registry database is pointed to a remote database, multiple running instances of the same product can boot up and run against the same configuration stored in the registry. This, in turn, helps with governance.

See [here](#) for more information on sharing a registry space across multiple WSO2 product instances.

### User stores

WSO2 products offer three choices to store user details:

- Using a database
- Using an LDAP server
- Using an Active Directory service

The default is to use the embedded H2 database, with the user store schema. Like in the case of the registry database, you can existing user bases and grant access privileges for WSO2 products based on those user stores.

See [Configuring User Stores](#) for more information on user stores, how they work, and how to configure them.
## Monitoring with JMX

WSO2 Products support JMX for monitoring. By default, JMX uses port 9999. You can configure this to the desired port by setting the JMX port parameter in the `carbon.xml` file.

```xml
<Ports>
  <JMX>9999</JMX>
</Ports>
```

**Related links**

See [JMX-Based Monitoring](#) for information on monitoring WSO2 products using JMX.

## Tuning WSO2 products

Most of the performance tuning recommendations are common to all WSO2 products. However, each WSO2 product may have additional guidelines for optimizing the performance of product-specific features.

**Related links**

- See [Performance Tuning](#) for the general guidelines, which are common to all WSO2 products.
- For performance tuning guidelines that are specific to each product, go to the product documentation for each product listed below and search for performance tuning guidelines.

Listed below are the main WSO2 products:

- API Manager
- Data Analytics Server
- Enterprise Integrator profiles
  - ESB profile
  - Business Process Management profile
  - Message Broker profile
  - Analytics profile
  - Micro Integrator profile
- IOT Server
- Identity Server
- Enterprise Service Bus

The following are now legacy products of WSO2:

- Application Server
- Business Rules Server
- Enterprise Mobility Manager
- Enterprise Store
- Governance Registry

## Firewalls

The following ports must be accessed when operating within a firewall.

- **9443** - Used by the management console and services that use the servlet transport, and is defined in the `catalina-server.xml` file.
- **9763** - Used by the services that use servlet transport, and is defined in the `catalina-server.xml` file.
- **9999** - Used for JMX monitoring, defined in the `carbon.xml` file.
- **8280** - Default HTTP port used by ESB for proxy services, and is defined in the `axis2.xml` file.
- **8243** - Default HTTPS port used by ESB for proxy services, and is defined in the `axis2.xml` file.

**Related links**

See [Default Ports of WSO2 Products](#) for a list of common and product-specific ports used by WSO2.
### Proxy servers

If the product is hosted behind a proxy such as ApacheHTTPD, users can configure products to use the proxy server by providing the following system properties at start-up:

```
-Dhttp.proxyHost=xxxx
-Dhttp.proxyPort=xxxx
```

Alternatively, this can be done by adding the following configurations in the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file:

```
<parameter name="Proxy">
  <Configuration>
    <proxyhost>you.proxy.host</proxyhost>
    <proxyport>your.proxy.port</proxyport>
  </configuration>
</parameter>
```

### High availability

For high availability, WSO2 products must run on a cluster. This enables the WSO2 products to still work in the case of failover. Use replication management provided by the RDBMS.

### Data backup and archiving

For data backup and archiving of data, use the functionality provided by the RDBMS.

### Backup and recovery recommendations

None of the WSO2 products persist data in the file systems or retain or generate artifacts. By default, we only store log files in the file system and data and artifacts in the databases and the repository.

**What you should back up**

1. **Database backups:**
   - Back up of all the databases defined in `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml`.
   - Back up any other databases configured in any files in the `<PRODUCT_HOME>/repository/conf/datasources` directory.

2. **Artifact backups:**
   - This includes hot-deployment artifacts, web applications, synapse files, tenant directories, etc. Back up of the `<PRODUCT_HOME>/repository` directory periodically. The frequency of the back ups depends on your usage. For example, if you are creating or updating APIs daily, take this backup daily.

3. **WSO2 product instance backups:**
   - A one-time-only backup that you take of the entire server directory. This includes all the configuration files, logs, server extensions, and deployment artifacts for both tenants and super tenants. This back up is ideally done when the server is ready to be deployed in a production environment.

### Backup recommendations

We recommend that you use a proper artifact management system such as Puppet to back up and manage your artifacts before deploying them in the WSO2 Carbon runtime. Also, use the WSO2 Update Manager (WUM) tool, which is a command-line utility that allows you to get the latest updates (bug fixes and security fixes) of a particular product release.
Recovery recommendations

Be sure to determine the following depending on your business-continuity requirements:

- **Recovery Point Objective (RPO)**: Up to what points are you to recover. This is determined by the latest, known, good point.
- **Recovery Time Objective (RTO)**: How long does it take to recover to the RPO.
- **Backup Frequency**: How frequently you should take backups. If your RPO is one day, your backup frequency should be daily.
- **Disaster Recovery Site**: The place where the latest copy of your backup is. This can be from a different shelf in your data center to a completely different geographical location.

We also recommend the following:

1. Align your artifact deployment and recovery processes.
2. Schedule disaster recovery drills to test the recoverability of the system.
3. Test your artifacts in an environment that is identical to the production environment before deploying them into production.

Recovery strategy

The following steps include how to recover your setup using the backups:

1. Recover the hot-deployment artifacts by replacing the `<PRODUCT_HOME>/repository` directory with the backed up copy.
2. Recover the entire WSO2 product by directly replacing the existing WSO2 server directory in the production setup with the backup server directory. This will ensure that all the files, logs, and configurations made to the product do not need to be redone.
3. To recover the databases, follow the recovery strategy recommended by the databases you are using. For information on supported and tested databases, see Tested Database Management Systems.

Security Guidelines for Production Deployment

Given below are the common security guidelines for deploying a WSO2 product in a production environment.

Also, see the production deployment guidelines and any other product-specific guidelines that might come in the respective product's documentation.

- **WSO2 product-level security**
- **OS-level security**
- **Network-level security**

### WSO2 product-level security

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram: managing your artifacts using a configuration management system
<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apply security updates</strong></td>
<td>Apply all the security patches relevant to your product version. If your WSO2 product version is supported by WSO2 Update Manager (WUM), you need to use WUM to get the latest fixes.</td>
</tr>
<tr>
<td></td>
<td>• If your WSO2 product is listed as a WUM-supported product, follow the instructions in Getting Started with WUM.</td>
</tr>
<tr>
<td></td>
<td>• If you are using an older WSO2 product version, which is not WUM-supported, you need to download the security patches relevant to your product from the WSO2 Security Patch Release page and apply them to your system manually. The instructions are given in WSO2 Patch Application Process.</td>
</tr>
<tr>
<td><strong>Note the following:</strong></td>
<td>• WSO2 releases security patch notifications monthly via the Support Portal and the above mentioned WSO2 Security Patch Releases page. However, for highly critical issues, patches are issued immediately to customers.</td>
</tr>
<tr>
<td></td>
<td>• The WSO2 Security Patch Release page has all the security patches for the latest product versions. WSO2 does not issue patches publicly for older product versions. Community users are encouraged to use the latest product version to receive all the security fixes.</td>
</tr>
<tr>
<td><strong>Change default keystores</strong></td>
<td>Change the default key stores and create new keys for all the cryptographic operations. WSO2 products by default come with a self-signed SSL key. Since these keys are public, it is recommended to configure your own keys for security purposes. Consider the following guidelines when creating the keystores:</td>
</tr>
<tr>
<td></td>
<td>• Select a key size of at least 2048 bits.</td>
</tr>
<tr>
<td></td>
<td>• Use an SHA256 certificate.</td>
</tr>
<tr>
<td></td>
<td>• Make sure that WSO2 default certificates do not exist in any of the keystores in your production environment. For example, be sure to delete the default public certificate in the default trust store that is shipped with the product.</td>
</tr>
<tr>
<td></td>
<td>See the recommendations for using keystores in WSO2 products for information. See Creating New Keystores for information on how to create and configure your own keys.</td>
</tr>
<tr>
<td><strong>Encrypt passwords in configuration files</strong></td>
<td>WSO2 products use a tool called Secure Vault to encrypt the plain-text passwords in configuration files. See Securing Passwords in Configuration Files for instructions.</td>
</tr>
<tr>
<td><strong>Change default ports</strong></td>
<td>All the default ports used by WSO2 products are listed in Default Ports of WSO2 Products. For example, the default HTTPS port is 9443 and the HTTP port is 9763. Also, Axis2 services are exposed over the following ports: 8243 and 8280.</td>
</tr>
<tr>
<td></td>
<td>To change a default port, update the <code>&lt;offset&gt;</code> element in the <code>carbon.xml</code> file as explained in Changing the Default Ports.</td>
</tr>
<tr>
<td><strong>Enable read-only access to external user stores (LDAPs etc.)</strong></td>
<td>If your WSO2 product is connecting to an external user store, such as Microsoft Active Directory, for the purpose of reading and retrieving user information, be sure to enable read-only access to that user store.</td>
</tr>
<tr>
<td></td>
<td>For example, see Configuring a Read-Only LDAP User Store under Configuring User Stores for instructions.</td>
</tr>
<tr>
<td><strong>Always communicate (with external user stores) over TLS</strong></td>
<td>All connections from your WSO2 product to external databases, userstores (LDAP), or other services, should be over TLS, to ensure adequate network-level protection. Therefore, be sure to use external systems (user stores, databases) that are TLS-enabled.</td>
</tr>
<tr>
<td><strong>Connect to data stores using a less privileged user</strong></td>
<td>When connecting the WSO2 product to external databases or user stores (LDAP), be sure to go through a user who does not have permission to change the data store's schema. Be sure not to use the root user of the data store because all permissions are generally granted to the root user.</td>
</tr>
</tbody>
</table>
**Configure strong HTTP(S) security**

To have strong transport-level security, use TLS 1.2 and disable SSL, TLS 1.0 and 1.1. The TLS protocol and strong ciphers are configured for an HTTP connector in the `catalina-server.xml` file (using the `sslEnabledProtocols` and `ciphers` attributes). See the following links for instructions:

- Configuring Transport-Level Security
- Supported Cipher Suites

Note the following:

- When deciding on the TLS protocol and the ciphers, consider the compatibility with existing client applications. Imposing maximum security might cause functional problems with client applications.
- Apply ciphers with 256 bits key length if you have applied the Unlimited strength policy. Note that Unlimited strength policy is recommended.
- Also, consider the following factors when deciding on the ciphers:
  - DES/3DES are deprecated and should not be used.
  - MD5 should not be used, due to known collision attacks.
  - RC4 should not be used, due to crypto-analytical attacks.
  - DSS is limited to a small 1024 bit key size.
  - Cipher-suites that do not provide Perfect Forward Secrecy/Forward Secrecy (PFS/FS).
  - GCM based ciphers are recommended over CBC ciphers.

**Remove weak ciphers for PassThrough transport**

Remove any weak ciphers from the PassThrough transport and ensure that the server does not accept connections using those weak ciphers. The PassThrough transport is configured using the `axis2.xml` file stored in the `<PROD_HOME>/repository/conf/axis2/` directory. You need to add the `PreferredCiphers` parameter under the "Transport Ins (Listeners)" section along with the list of relevant cipher suites.

See Configuring the PassThrough Transport for instructions.

**Update the HTTP Response header "Server" value**

By default, all WSO2 products pass "WSO2 Carbon Server" as the server value in HTTP headers when sending HTTP responses. This means that information about the WSO2 product stack will be exposed through HTTP responses. It is recommended to change this by configuring the server name in the `catalina-server.xml` file.

See Configuring Transport Level Security for instructions.

**Enabling HTTP Strict Transport Security Headers (HSTS)**

Be sure that HTTP Strict Transport Security (HSTS) is enabled for all the applications deployed in your WSO2 server. This includes the management console, and any other web applications and/or Jaggery applications.

Note that (for products based on Carbon 4.4.11 or later versions) HSTS is disabled for the applications with which the product is shipped by default. This is because HSTS validation can interrupt the development processes by validating signatures of self-signed certificates.

See the topic on Enabling HTTP Strict Transport Security Headers for instructions.

**Preventing browser caching**

If there are dynamic pages in your application with sensitive information, you need to prevent browser caching. This can be done by making sure that the applications deployed in your server will return the relevant HTTP response headers.

Note that cache prevention headers are enabled for the applications with which the product is shipped by default. Therefore, you need to manually enable cache prevention headers only for all the new applications that you deploy in your server.

See the topic on Preventing browser caching for instructions.

**Increase Ephemeral Diffie-Hellman Key size**

Before starting the server, open the product startup script (`wso2server.sh` in Linux and `wso2server.bat` in Windows) and enter the following with the other Java properties:

```
-Djdk.tls.ephemeralDHKeySize=2048 \
```
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable client-initiated renegotiation</td>
<td>Before starting the server, open the product startup script (wso2server.sh in Linux and wso2server.bat in Windows) and enter the following with the other Java properties:</td>
<td><code>Djdk.tls.rejectClientInitiatedRenegotiation=true \</code></td>
</tr>
<tr>
<td>Enable HostName Verification</td>
<td>If your product is using Carbon Kernel 4.4.17 or a later version, make sure that hostname verification is enabled in the product startup script (wso2server.sh in Linux and wso2server.bat in Windows) with the Strict mode. That is, you need to enable the following parameter:</td>
<td><code>Dhttpclient.hostnameVerifier=Strict \</code></td>
</tr>
<tr>
<td></td>
<td>In Carbon versions prior to 4.4.17, be sure that hostname verification is enabled by setting the following property to false.</td>
<td><code>Dorg.wso2.ignoreHostnameVerification=false \</code></td>
</tr>
<tr>
<td>Enable additional XSS Protection</td>
<td>XSS attacks are prevented on the latest WSO2 products by default. This is due to output encoding of the displaying values. However, if additional protection is required, an input validation valve can be configured.</td>
<td>See Enabling HostName Verification for instructions.</td>
</tr>
<tr>
<td>Increase JSESSIONID length</td>
<td>If required, increase the session ID length by changing the <code>sessionIDLength</code> attribute of the session manager in the context.xml file (stored in the <code>&lt;PRODUCT_HOME&gt;/repository/conf/tomcat/context.xml</code> directory) as shown below. The default value is 16 bytes.</td>
<td><code>&lt;Manager className=&quot;org.wso2.carbon.webapp.mgt.CarbonTomcatSessionManager&quot; sessionIdLength=&quot;16&quot;&gt;&lt;/Manager&gt;</code></td>
</tr>
<tr>
<td>Disable URL rewriting</td>
<td>By default, WSO2 products do not use URL rewriting that could expose session information in URLs. For additional protection, disable URL rewriting at the server level. This can be configured in the context.xml file (stored in the <code>&lt;PRODUCT_HOME&gt;/repository/conf/tomcat/context.xml</code> file).</td>
<td><code>&lt;Context disableURLRewriting=&quot;true&quot; &gt;</code></td>
</tr>
</tbody>
</table>
Change default admin credentials

All WSO2 products have the Administrator account configured by default. The default user name and password of the administrator account is “admin”. To change the administrator credentials, you need to first sign in to the management console of the product as “admin”, and then use the Change Password option under Home->Configure->User Management->Users in the navigator.

See Changing a Password for more information on how to change the password of the administrator.

Restrict access to the management console

Make sure that the permission for signing into the management console is granted only to the users that need to use the management console. For example, in products such as WSO2 Identity Server and WSO2 API Manager, the majority of users only need to login to the connected service providers via the WSO2 product. Therefore, such users should not have permission to log into the management console.

You can make sure that only administrative users have access to the product’s management console. Further, instead of granting all permission to one administrator, you can distribute the responsibilities among administrators by assigning different permissions for conducting various tasks.

See Managing Users, Roles and Permissions for instructions.

Enable log rotation and monitoring

Ensure that you have a relevant log rotation scheme to manage logs. Log4J properties for WSO2 products can be configured in the <PRODUCT_HOME>/repository/conf/log4j.properties file. To roll the wso2carbon.log based on size, the following configurations can be used:

```java
log4j.appender.CARBON_LOGFILE=org.apache.log4j.RollingFileAppender
log4j.appender.CARBON_LOGFILE=${carbon.home}/repository/logs/${instance.log}/wso2carbon${instance.log}.log
log4j.appender.CARBON_LOGFILE.MaxFileSize=1000KB
log4j.appender.CARBON_LOGFILE.MaxBackupIndex=10
```

See Monitoring Logs for details on how to configure logging details in WSO2 products.

Prevent Log Forging

Log forging can be prevented by appending a UUID to the log message.

See Monitoring Logs for more information on configuring the log4j.properties file.

Set appropriate JVM parameters

The recommended JDK version is JDK 1.7 or 1.8. See the installation pre-requisites for more information.

For JDK 1.7, set the appropriate Heap and Permgen values for the JVM based on your deployment scenario. These can be set in the <PRODUCT_HOME>/bin/wso2server.sh file. You do not need to set this in JDK 1.8 as the MaxPermSize value has been removed from Hotspot JVM.

For example

```bash
-Xms512m -Xmx2048m -XX:MaxPermSize=1024m
```

Tip: To run the JVM with 2 GB (-Xmx2048m), you should ideally have about 4GB of memory on the physical machine.

### OS-level security

This section provides the list of OS-level security guidelines for your production environment.

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run WSO2 processes with a specific OS-level user</td>
<td>Use a separate OS-level user to run WSO2 products. Make sure that the user is only granted the permissions required for running the product for that particular user. Do not use the root/administrator user of your OS because the root/administrator is granted all privileges by default.</td>
</tr>
<tr>
<td>Minimize software to avoid vulnerability</td>
<td>Make sure that you only install the software/packages that are relevant to your WSO2 product’s deployment. Also, continuously monitor the software that you install. See the Installation Prerequisites to identify the minimum software your WSO2 product will need.</td>
</tr>
</tbody>
</table>
Enable the Firewall | Enable a firewall at the OS level (for example, `iptables`). This will provide protection for inbound and outbound connections of your WSO2 product. Make sure that you only open the required outbound and inbound ports from the OS-level firewall.

Restrict access to TCP ports used for clustering | Apply a firewall at host-level to disallow access to TCP ports used for clustering (port 4000, 4001, ... by default) from unrecognized hosts. These ports should be accessible only from other members of the WSO2 product cluster.

Use Secure Shell (SSH) | Use Secure Shell (SSH) when interacting with servers and executing commands. Adhere to the following best practices when you configure SSH:

- Change the default ssh port to a higher value.
- Disable the root or administrator.
- Enable login with user keys.
- Display a legal banner or a security banner with security warnings before SSH authentication.

Keep the system up-to-date | If there are security updates available for the packages installed in your OS (including the Java runtime), be sure to perform the necessary testing in a staging environment, and then proceed to install them for your OS.

Monitor user activities | Monitor the activities of your OS users. You can do this by enabling OS-level logs and by reviewing them regularly. You can also set up a centralized log monitoring system for this purpose.

Session Data Cleanup | **Note**: This security guideline is specific only to WSO2 Identity Server.

In a production environment, there is a possibility for a deadlock/database lock to occur when running a session data cleanup task in high load scenarios. To mitigate this, configure the following property to clean data in chunks. Configure this property in the `<IS_HOME>/repository/conf/identity/identity.xml` file under `<SessionDataCleanUp>` with the required chunk size. This value is in the number of records and depends on the database type and server capacity. It also depends on the amount of load generated by single sign-on (SSO). A higher value increases the chances of deadlocks and a lower value increases the time it takes for a cleanup.

```xml
/DeleteChunkSize>50000</DeleteChunkSize>
```

For more information on configuring sessions in production, see Authentication Session Persistence in the WSO2 Identity Server documentation.

Make regular backups | Make sure to backup important files and archive them continuously. See Backup and Recovery Recommendations for more information.

---

**Network-level security**

This section provides the list of security guidelines for configuring the network.

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a failover setup</td>
<td>When your WSO2 products are clustered, you need to regularly monitor the health of your server instances. For example, you need to monitor resource-level factors such as the server's resource utilization, response time anomalies, and the number of incoming network connections. Server monitoring will help you identify when additional server instances (failover instances) are required. You can also make decisions about network routing changes that you need to do in order to avoid server downtime.</td>
</tr>
<tr>
<td></td>
<td>- See Deployment and Clustering/Key Concepts for information on configuring failover.</td>
</tr>
<tr>
<td></td>
<td>- See Monitoring WSO2 products for information on the monitoring options for WSO2 products.</td>
</tr>
<tr>
<td>Maintain network level logging</td>
<td>Be sure to maintain and monitor logs for your proxy servers, load balancers, and other network devices.</td>
</tr>
<tr>
<td>Check open ports and services</td>
<td>Periodically check for open ports using port scanning tools and make sure that only the necessary ports are open to both internal and external networks. Be sure that only the ports relevant to your WSO2 products are open for communication. If there are other ports started, be sure to monitor them.</td>
</tr>
<tr>
<td></td>
<td>See Default Ports of WSO2 Products for the full list of ports in all WSO2 products.</td>
</tr>
<tr>
<td>Configure device-level security</td>
<td>All your network devices should be periodically checked for anomalies. For example, you need to verify routing tables and firewall rules.</td>
</tr>
<tr>
<td></td>
<td>Also, make sure that the default credentials are changed before the first use of those devices.</td>
</tr>
</tbody>
</table>
**Troubleshooting in Production Environments**

The following sections provide information on how to troubleshoot various problems that may arise for deployment in production environments.

- Analyzing a stack trace
- Capturing the state of the system
- Viewing process threads in Solaris
- Checking the health of a cluster

### Analyzing a stack trace

When your Java process starts to spin your CPU, you must immediately analyze the issue using the following two commands and obtain the invaluable information required to tackle the issue. This is done based on the process ID (pid).

1. `jstack <pid> > thread-dump.txt`
2. `ps -C java -L -o pcpu,cpu,nice,state,cputime,pid,tid > thread-usage.txt`

Tip: OS X users can alternatively use the command `ps M <PID>` instead.

These commands provide you with the `thread-dump.txt` file and the `thread-usage.txt` file. After obtaining these two files, do the following.

1. Find the thread ID (the one that belongs to the corresponding PID) that takes up the highest CPU usage by examining the `thread-usage.txt` file.

```
% CPU  CPU   NI  S  TIME   PID   TID
.............
 0.0   -   0 S 00:00:00  1519  1602
 0.0   -   0 S 00:00:00  1519  1603
24.8 -   R 00:06:19  1519  1604
 2.4 -   0 S 00:00:37  1519  1605
 0.0 -   0 S 00:00:00  1519  1606
.............
```

In this example, the thread ID that takes up the highest CPU usage is 1604.

2. Convert the decimal value (in this case 1604) to hexadecimal. You can use an online converter to do this. The hexadecimal value for 1604 is 644.

3. Search the `thread-dump.txt` file for the hexadecimal obtained in order to identify the thread that spins. In this case, the hexadecimal value to search for is 644. The `thread-dump.txt` file should have that value as a thread ID of one thread.

4. That thread usually has a stack trace, and that's the lead you need to find the issue. In this example, the stack trace of the thread that spins is as follows.
Capturing the state of the system

Carbondump is a tool used to collect all the necessary data from a running WSO2 product instance at the time of an error. The carbondump generates a ZIP archive with the collected data that helps to analyze the system and determine the problem that caused the error. Therefore, it is recommended that you run this tool as soon as an error occurs in the WSO2 product instance.

When using the tool, you have to provide the process ID (pid) of the product instance and the `<PRODUCT_HOME>` location, which is where your unzipped Carbon distribution files reside. The command takes the following format:

```
sh carbondump.sh [-carbonHome path] [-pid of the carbon instance]
```

For example,
In Linux: sh carbondump.sh -carbonHome /home/user/wso2carbon-3.0.0/ -pid 5151
In Windows: carbondump.bat -carbonHome c:\wso2carbon-3.0.0\ -pid 5151

The tool captures the following information about the system:

- Operating system information** OS (kernel) version
- Installed modules lists and their information
- List of running tasks in the system
- Memory information of the Java process** Java heap memory dump
- Histogram of the heap
- Objects waiting for finalization
- Java heap summary. GC algo used, etc.
- Statistics on permgen space of Java heap
- Information about the running Carbon instance** Product name and version
  - Carbon framework version (This includes the patched version)
  - <PRODUCT_HOME>, <JAVA_HOME>
  - configuration files
  - log files
  - H2 database files
- Thread dump
- Checksum values of all the files found in the $CARBON_HOME

** Viewing process threads in Solaris

This information is useful to know in situations when the database processes are not fully utilizing the CPU’s threading capabilities. It gives you a better understanding on how 11g and 10g takes advantage of threading and how you can validate those queries from the system.

The following information provides insight on whether a Solaris process is parallelized and is taking advantage of the threading within the CPU.

1. Open a command line in Solaris.
2. Run prstat and have a look to the last column, labeled PROCESS/NLWP. NLWP is a reference to the number of lightweight processes and are the number of threads the process is currently using with Solaris as there is a one-to-one mapping between lightweight processes and user threads. A single thread process will show 1 there while a multi-threaded one will show a larger number. See the following code block for an example.

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>SIZE</th>
<th>RSS</th>
<th>STATE</th>
<th>PRI</th>
<th>NICE</th>
<th>TIME</th>
<th>CPU</th>
<th>PROCESS/NLWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12905</td>
<td>root</td>
<td>4472K</td>
<td>3640K</td>
<td>cpu0</td>
<td>59</td>
<td>0</td>
<td>0:00:01</td>
<td>0.4%</td>
<td>prstat/1</td>
</tr>
<tr>
<td>18403</td>
<td>monitor</td>
<td>474M</td>
<td>245M</td>
<td>run</td>
<td>59</td>
<td>17</td>
<td>1:01:28</td>
<td>9.1%</td>
<td>java/103</td>
</tr>
<tr>
<td>4102</td>
<td>oracle</td>
<td>12G</td>
<td>12G</td>
<td>run</td>
<td>59</td>
<td>0</td>
<td>0:00:12</td>
<td>4.5%</td>
<td></td>
</tr>
</tbody>
</table>

If you observe the PROCESS/NLWP value in the example above, you can identify that prstat and oracle are single thread processes, while java is a multi-threaded process.

3. Alternatively, you can analyze individual thread activity of a multi-threaded process by using the -L and -p options, like prstat -L -p pid. This displays a line for each thread sorted by CPU activity. In that case, the last column is labeled PROCESS/LWPID, where LWPID is the thread ID. If more than one thread shows significant activity, your process is actively taking advantage of multi-threading.

** Checking the health of a cluster

In Hazelcast, the health of a member in the cluster is determined by the heartbeats the member sends. If the well-known member does not receive a heartbeat within a given amount of time (this can be configured), then the node is assumed dead. By default, the given amount of time is 600 seconds (or 10mins), which might be too much for some scenarios.

Failure detectors used in distributed systems can be unreliable. In these sort of scenarios, Hazelcast uses heartbeat monitoring as a fault detection mechanism and the nodes send heartbeats to other nodes.

If a heartbeat message is not received by a given amount of time, Hazelcast assumes the node is dead. This is configured via the hazelcast.max.no.heartbeat.seconds property. The optimum value for this property depends on the system. Although the default is 600 seconds, it might be necessary to reduce the heartbeat to a lower value if nodes are to be declared dead in a shorter time frame. However, you must verify this in your system and adjust as necessary depending on your scenario.
Do the following steps to configure the maximum time between heartbeats.

1. Create a property file called hazelcast.properties, and add the following property to it:
   ```
   hazelcast.max.no.heartbeat.seconds=300
   ```
2. Place this file in the `<PRODUCT_HOME>/repository/conf/` directory in all the nodes in your cluster.
3. Restart the servers.

---

### 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 directory.

1. Create a file called `workers-list.txt` that lists all the worker nodes in the deployment. The following is a sample of the file where there are two worker nodes.

   ```
   Tip: 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
   ```

2. Create a file to synchronize the `<PRODUCT_HOME>/repository/deployment/server` folders between the manager and all worker nodes.

   ```
   Note: You must create your own SSH key and define it as the `pem_file`. Alternatively, you can use an existing SSH key. 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=/.ssh/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.

    * * * * *
   /home/ubuntu/setup/rsync-for-depsync/rsync-for-depsync.sh

Note: You can run the Cron job on one given node (master) at a given time. If you switch it 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 so far.
Data Stores

See the following topics for information on storing various data in WSO2 products:

- Working with Databases
- Working with the Registry
- Working with Users, Roles and Permissions

Working with Databases

All WSO2 products are shipped with embedded H2 databases for storing data. These default databases are located in the `<PRODUCT_HOME>/repository/database` directory of the product pack.

Default databases

Explained below are the default databases that you will find in the `database` directory.

- **Carbon database**: `WSO2CARBON_DB.h2.db` is the main Carbon database in a WSO2 product. This stores registry and user management data by default. In addition, if the product uses features of WSO2 Identity Server or WSO2 Enterprise Store (ES), data that are specific to those will by default reside in the embedded Carbon database. However, for production environments, we recommend separate RDBMSs to store identity-related and storage-related data.
- **Product-specific databases**: In addition to the main Carbon database, your product may have other databases for storing product-specific data.

Changing the default databases

The embedded H2 databases shipped with your product are suitable for development, testing, and some production environments. For most production environments, we recommend industry-standard RDBMSs such as Oracle, PostgreSQL, MySQL, MS SQL, etc. Further, if you have features of WSO2 Identity Server or WSO2 Enterprise Store (ES) in your product, it is recommended to use separate RDBMSs for each, i.e., identity-related and storage-related data.

WSO2 products are shipped with scripts for creating the required tables in all the required databases. The scripts for creating tables for user management and registry data are stored in the `<PRODUCT_HOME>/dbscripts` folder. If product-specific databases are required, and if features of WSO2 Identity Server or WSO2 Enterprise Store (ES) are used in the product, there will be subfolders in the `<PRODUCT_HOME>/dbscripts` directory with separate scripts.

Changing the default Carbon database:

You simply have to set up new physical databases, point the product server to the new databases by updating the relevant configuration files, and create the required tables using the scripts provided in the product pack. See the following topics for instructions:

- Setting up the Physical Database
- Changing the Carbon Database
- Browsing the H2 Database

Changing the default product-specific databases:

The process of setting up and configuring product-specific databases is similar to changing the default Carbon database. However, depending on the product, there may be additional configuration files to update. See the documentation for the respective product for instructions.

Setting up the Physical Database

The topics in this section describe how to use scripts in `<PRODUCT_HOME>/dbscripts/` folder to set up each type of physical database.

- Setting up IBM DB2
- Setting up IBM Informix
- Setting up H2
- Setting up MariaDB
- Setting up Microsoft SQL
- Setting up MySQL
- Setting up a MySQL Cluster
- Setting up Oracle
- Setting up Oracle RAC
- Setting up PostgreSQL
- Setting up Derby

Setting up IBM DB2

The following sections describe how to set up an IBM DB2 database to replace the default H2 database in your WSO2 product:

- Prerequisites
- Setting up the database and users
- Setting up DB2 JDBC drivers

Prerequisites
Download the latest version of DB2 Express-C and install it on your computer.

For instructions on installing DB2 Express-C, see this ebook.

Setting up the database and users
Create the database using either DB2 command processor or DB2 control center as described below.

Using the DB2 command processor
1. Run DB2 console and execute the `db2start` command on a CLI to open DB2.
2. Create the database using the following command:
   ```
   create database <DB_NAME>
   ```
3. Before issuing an SQL statement, establish the connection to the database using the following command:
   ```
   connect to <DB_NAME> user <USER_ID> using <PASSWORD>
   ```
4. Grant required permissions for users as follows:
   ```
   connect to DB_NAME
   grant <AUTHORITY> on database to user <USER_ID>
   ```

For example:

```
 db2 -> connect to regdb user greg using 18801980
 Database Connection Information
 Database server = DB2/LINUX 9.7.4
 SQL authorization ID = greg
 Local database alias = REGDB

 db2 -> GRANT DELETE, CREATE, BIND, CONNECT, CREATE_NOT_FENCED, INPLICIT_SCHEMA, LOAD on DATABASE TO USER user REGDB00001. The SQL command completed successfully.
 db2 ->
```

For more information on DB2 commands, see the DB2 Express-C Guide.

Using the DB2 control center
1. Open the DB2 control center using the `db2cc` command as follows:
   ```
   greg@wso2:~/sqlib/bin$ ./db2cc
   ```
2. Right-click All Databases in the control center tree (inside the object browser), click Create Database, and then click Standard and follow the steps in the Create New Database wizard.
3. Click **User and Group Objects** in the control center tree to create users for the newly created database.

4. Give the required permissions to the newly created users.
Setting up DB2 JDBC drivers

Copy the DB2 JDBC drivers (db2jcc.jar and db2jcc_license_cu.jar) from `<DB2_HOME>/SQLLIB/java` directory to the `<PRODUCT_HOME>/repository/components/lib/` directory.

```bash
cp db2jcc.jar db2jcc_license_cu.jar /home/user/wso2/greg/repository/components/lib/
```

`<DB2_HOME>` refers to the installation directory of DB2 Express-C, and `<PRODUCT_HOME>` refers to the directory where you run the WSO2 product instance.

**What's next**

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with IBM DB2, see Changing to IBM DB2.

**Setting up IBM Informix**

The following sections describe how to set up IBM Informix to replace the default H2 database in your WSO2 product:

- Prerequisites
- Creating the database
- Setting up Informix JDBC drivers

**Prerequisites**

Download the latest version of IBM Informix and install it on your computer.
Creating the database

Create the database and users in Informix. For instructions on creating the database and users, see Informix product documentation.

Do the following changes to the default database when creating the Informix database.

- Define the page size as 4K or higher when creating the dbspace as shown in the following command (i.e. denoted by \(-k\) 4):

  ```
  onspaces -c -S testspace4 -k 4 -p /usr/informix/logdir/data5.dat -o 100 -s 3000000
  ```

- Add the following system environment variables.

  ```
  export DB_LOCALE=en_US.UTF-8
  export CLIENT_LOCALE=en_US.UTF-8
  ```

- Create an sbspace other than the dbspace by executing the following command:

  ```
  onspaces -c -S testspace4 -k 4 -p /usr/informix/logdir/data5.dat -o 100 -s 3000000
  ```

- Add the following entry to the `<INFORMIX_HOME>/etc/onconfig` file, and replace the given example sbspace name (i.e. `testsparse4`) with your sbspace name:

  ```
  SBSPACENAME testspace4
  ```

Setting up Informix JDBC drivers

Download the Informix JDBC drivers and copy them to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib/directory`

Use Informix JDBC driver version 3.70.JC8, 4.10.JC2 or higher.

What's next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with IBM Informix, see Changing to IBM Informix.

Setting up H2

You can set up either an embedded H2 database or a remote H2 database using the instructions in the following topics:

- Setting up Embedded H2
- Setting up Remote H2
The following sections describe how to set up an embedded H2 database to replace the default H2 database in your WSO2 product:

- Setting up the database
- Setting up the drivers

## H2 is not recommended in production

The embedded H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead.

You can use the embedded H2 database in development environments and as the local registry in a registry mount.

### Setting up the database

Download and install the H2 database engine on your computer.

For instructions on installing DB2 Express-C, see H2 installation guide.

### Setting up the drivers

WSO2 currently ships H2 database engine version h2-1.2.140.* and its related H2 database driver. If you want to use a different H2 database driver, take the following steps:

1. Delete the following H2 database-related JAR file, which is shipped with WSO2 products:
   ```
   <PRODUCT_HOME>/repository/components/plugins/h2-database-engine_1.2.140.wso2v3.jar
   ```

2. Find the JAR file of the new H2 database driver (``<H2_HOME>/bin/h2-*.jar`` where ``<H2_HOME>`` is the H2 installation directory) and copy it to your WSO2 product's ``<PRODUCT_HOME>/repository/components/lib/`` directory.

### What's next

Next, you need to configure your product with Embedded H2 database. For more information, see Changing to Embedded H2.

### Setting up Remote H2

The following sections describe how to set up a remote H2 database to replace the default H2 database in your WSO2 product:

- Setting up the remote H2 database
- Setting up the drivers

## When to use the embedded H2 database?

The embedded H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead.

However, you can use the embedded H2 database as the local registry in a registry mount even in enterprise testing and production environments.

### Setting up the remote H2 database

Follow the steps below to set up a Remote H2 database.

1. Download and install the H2 database engine on your computer as follows:

   For instructions on installing, see the H2 installation guide.
2. Go to the `<H2_HOME>/bin` directory and run the H2 network server starting script as follows, where `<H2_HOME>` is the H2 installation directory:

   ```
   [Client@ws2:~/dtbsh]# get http://www.h2database.com/h2-211-09-11.zip
   Connecting to www.h2database.com (99.17.101:80) [Connected].
   GET /h2-211-09-11.zip HTTP/1.1
   Host: www.h2database.com
   User-Agent: curl/7.25.0
   Accept: */*
   Accept-Language: en-US,en;q=0.8
   Content-Type: application/octet-stream
   Content-Length: 1898
   Proxy-Connection: close
   
   1898 bytes sent in 149ms (12K/s)
   
   15% [==================================] 1923.304 111K/s eta 45s
   ```

3. Run the H2 database server with the following commands:
   - For Linux:
     ```
     [client@ws2:~]# h2.sh
     ```
   - For Windows:
     ```
     [client@ws2:~]# h2.bat
     ```

   The script starts the database engine and opens a pop-up window.

4. Click Start Browser to open a web browser containing a client application, which you use to connect to a database. If a database does not already exist by the name you provided in the JDBC URL text box, H2 will automatically create a database.

Setting up the drivers

WSO2 currently ships H2 database engine version h2-1.2.140.* and its related H2 database driver. If you want to use a different H2 database driver, take the following steps:

1. Delete the following H2 database-related JAR file, which is shipped with WSO2 products:
   ```
   <PRODUCT_HOME>/repository/components/plugins/h2-database-engine_1.2.140.wso2v3.jar
   ```
2. Find the JAR file of the new H2 database driver (<H2_HOME>/bin/h2-* .jar, where `<H2_HOME>` is the H2 installation directory) and copy it to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib` directory.

What's next

Next, you need to configure your product with Embedded H2 database. For more information, see Changing to Remote H2.

Setting up MariaDB

The following sections describe how to set up MariaDB to replace the default H2 database in your WSO2 product

- Setting up the database and users
- Setting up the drivers

Setting up the database and users

Follow the steps given below to set up MariaDB. See Tested DBMSs for information on the MariaDB versions that are tested with WSO2 products.

1. Download, install and start MariaDB on your computer. See https://downloads.mariadb.org/.

   You can install MariaDB standalone or as a galera cluster for high availability. Database clustering is independent of WSO2 product clustering.

   For instructions on installing MariaDB on MAC OS, go to Homebrew.

2. Log in to MariaDB as the root user (or any other user with database creation privileges).

   ```
   mysql -u root -p
   ```

3. Enter the password when prompted.
In most systems, there is no default root password. Press the Enter key without typing anything if you have not changed the default root password.

4. In the MySQL command prompt, create the database using the following command:

```sql
CREATE DATABASE regdb;
```

5. Give authorization to the regadmin user as follows:

```sql
GRANT ALL ON regdb.* TO regadmin@localhost IDENTIFIED BY "regadmin";
```

6. Once you have finalized the permissions, reload all the privileges by executing the following command:

```sql
FLUSH PRIVILEGES;
```

7. Log out from the MySQL prompt by executing the following command:

```sql
quit;
```

Setting up the drivers

Download the MySQL Java connector JAR file, and copy it to the <PRODUCT_HOME>/repository/components/lib/ directory.

Note that you must use the MySQL connector that is compatible with your MariaDB version. For example, mysql-connector-java-5.1.3 6-bin.jar is compatible with MariaDB version 10.0.20. See Tested DBMSs for information on the WSO2 products that have been tested for compatibility with different versions of MariaDB and MySQL connectors.

What's next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with MariaDB, see Changing to MariaDB.

Setting up Microsoft SQL

The following sections describe how to set up Microsoft SQL to replace the default H2 database in your WSO2 product:

- Setting up the database and users
- Setting up the JDBC driver

Setting up the database and users

Follow the steps below to set up the Microsoft SQL database and users.

Enable TCP/IP

1. In the start menu, click Programs and launch Microsoft SQL Server 2012.
2. Click Configuration Tools, and then click SQL Server Configuration Manager.
3. Enable TCP/IP and disable Named Pipes from protocols of your Microsoft SQL server.
4. Double click TCP/IP to open the TCP/IP properties window and set Listen All to Yes on the Protocol tab.
5. On the IP Address tab, disable TCP Dynamic Ports by leaving it blank and give a valid TCP port, so that Microsoft SQL server will listen on that port.
6. Similarly, enable TCP/IP from SQL Native Client Configuration and disable Named Pipes. Also, check whether the port is set correctly to 1433.

7. Restart Microsoft SQL server.

Create the database and user

1. Open the Microsoft SQL Management Studio to create a database and user.
2. Click New Database from the Database menu and specify all the options to create a new database.
3. Click New Login from the Logins menu, and specify all the necessary options.

Grant permissions

Assign newly created users the required grants/permissions to log in and create tables, to insert, index, select, update and delete data in tables in the newly created database. These are the minimum set of SQL server permissions.

Setting up the JDBC driver

Download and copy the sqljdbc4 Microsoft SQL JDBC driver file to the WSO2 product's `<PRODUCT_HOME>/repository/components/lib` directory. Use `com.microsoft.sqlserver.jdbc.SQLServerDriver` as the `driverClassName` in your datasource configuration in `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as explained in the next section.

In WSO2 IoT Server copy the driver file to the `<IOTS_HOME>/lib` directory

What's next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with MSSQL, see Changing to MSSQL.

Setting up MySQL

The following sections describe how to set up a MySQL database to replace the default H2 database in your WSO2 product:

- Setting up the database and users
- Setting up the drivers
- Executing db scripts on MySQL database

Setting up the database and users

Follow the steps below to set up a MySQL database:

1. Download and install MySQL on your computer using the following command:

   For instructions on installing MySQL on MAC OS, go to Homebrew.

   ```
   sudo apt-get install mysql-server mysql-client
   ```

2. Start the MySQL service using the following command:

   ```
   sudo /etc/init.d/mysql start
   ```

3. Log in to the MySQL client as the root user (or any other user with database creation privileges).

   ```
   mysql -u root -p
   ```
4. Enter the password when prompted.

In most systems, there is no default root password. Press the Enter key without typing anything if you have not changed the default root password.

5. In the MySQL command prompt, create the database using the following command:

```
create database regdb;
```

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 database creation command should be as follows:

```
mysql> create database <DATABASE_NAME> character set latin1;
```

For users of other operating systems, the standard database creation commands will suffice. For these operating systems, the database creation command should be as follows:

```
mysql> create database <DATABASE_NAME>;
```

6. Give authorization to the regadmin user as follows:

```
GRANT ALL ON regdb.* TO regadmin@localhost IDENTIFIED BY "regadmin";
```

7. Once you have finalized the permissions, reload all the privileges by executing the following command:

```
FLUSH PRIVILEGES;
```

8. Log out from the MySQL prompt by executing the following command:

```
quit;
```

Setting up the drivers

Download the MySQL Java connector JAR file, and copy it to the `<PRODUCT_HOME>/repository/components/lib/` directory.

Be sure to use the connector version that is supported by the MySQL version you use. If you come across any issues due to version incompatibility, follow the steps below:

1. Shut down the server and remove all existing connectors from `<PRODUCT_HOME>/repository/components/lib` and `<PRODUCT_HOME>/repository/components/dropins`.
2. Download the connector JAR that is compatible with your current MySQL version.
3. Copy the JAR file only to `<PRODUCT_HOME>/repository/components/lib`. Files will be copied automatically to the dropins folder during server startup.
4. Start the server with the `-Dsetup` parameter as `sh wso2server.sh -Dsetup`. 
Executing db scripts on MySQL database

To run the database script against the database you created, login to the MySQL client and point to the corresponding database.

```bash
use regdb;
```

Execute the mysql.sql database script against the pointed database using the following command.

```bash
mysql> source <path to the script>/mysql.sql;
```

If you are using MySQL 5.7 or later version, use mysql5.7.sql script and execute the above command modified as follows.

```bash
mysql> source <path to the script>/mysql5.7.sql;
```

**What’s next**

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with MySQL, see Changing to MySQL.

**Setting up a MySQL Cluster**

For instructions on setting up any WSO2 product with a MySQL cluster, see this article, which is published in the WSO2 library.

**Setting up Oracle**

The following sections describe how to set up an Oracle database to replace the default H2 database in your WSO2 product:

- Setting up the database and users
- Setting up the JDBC driver

**Setting up the database and users**

Follow the steps below to set up an Oracle database.

1. Create a new database by using the Oracle database configuration assistant (dbca) or manually.
2. Make the necessary changes in the Oracle tnsnames.ora and listener.ora files in order to define addresses of the databases for establishing connections to the newly created database.
3. After configuring the .ora files, start the Oracle instance using the following command:

   ```bash
   $ sudo /etc/init.d/oracle-xe restart
   ```

4. Connect to Oracle using SQL*Plus as SYSDBA as follows:

   ```bash
   $ ./<ORACLE_HOME>/config/scripts/sqlplus.sh sysadm/password as SYSDBA
   ```

5. Connect to the instance with the username and password using the following command:

   ```bash
   $ connect
   ```

6. As SYSDBA, create a database user and grant privileges to the user as shown below:
Create user `<USER_NAME>` identified by password account unlock; 
grant connect to `<USER_NAME>`;
grant create session, create table, create sequence, create trigger to 
`<USER_NAME>`;
alter user `<USER_NAME>` quota `<SPACE_QUOTA_SIZE_IN_MEGABYTES>` on 
`'<TABLE_SPACE_NAME>'`;
commit;

7. Exit from the SQL*Plus session by executing the quit command.

Setting up the JDBC driver

1. Copy the Oracle JDBC libraries (for example, `<ORACLE_HOME>/jdbc/lib/ojdbc14.jar`) to the `<PRODUCT_HOME>/repository/components/lib/` directory.
2. Remove the old database driver from the `<PRODUCT_HOME>/repository/components/dropins/` directory.

   If you get a "timezone region not found" error when using the ojdbc6.jar file with WSO2 servers, set the Java property as follows: export JAVA_OPTS="-Duser.timezone='+05:30'"  
   
   The value of this property should be the GMT difference of the country. If it is necessary to set this property permanently, define it inside the `wso2server.sh` as a new JAVA_OPTS property.

What's next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with Oracle, see Changing to Oracle.

Setting up Oracle RAC

The following sections describe how to set up Oracle RAC to replace the default H2 database in your WSO2 product:

- Setting up the database and users
- Setting up the JDBC driver

Oracle Real Application Clusters (RAC) is an option that facilitates clustering and high availability in Oracle database environments. In the Oracle RAC environment, some of the commands used in `oracle.sql` are considered inefficient. Therefore, the product has a separate SQL script (`oracle_rac.sql`) for Oracle RAC. The Oracle RAC-friendly script is located in the `dbscripts` folder together with other `.sql` scripts.

To test products on Oracle RAC, rename `oracle_rac.sql` to `oracle.sql` before running `-Dsetup`.

Setting up the database and users

Follow the steps below to set up an Oracle RAC database.

1. Set environment variables `<ORACLE_HOME>`, `PATH`, and `<ORACLE_SID>` with the corresponding values (/oracle/app/oracle/product/11.2.0/dbhome_1, `PATH: <ORACLE_HOME>/bin, and `orc11`) as follows:

   ```
   [oracle@node1 ~]$ export ORACLE_HOME=/oracle/app/oracle/product/11.2.0/dbhome_1
   [oracle@node1 ~]$ export PATH=$PATH:$ORACLE_HOME/bin
   [oracle@node1 ~]$ export ORACLE_SID=orc11
   ```

2. Connect to Oracle using SQL*Plus as SYSDBA.
2. Create a database user and grant privileges to the user as shown below:

```
CREATE user <USER_NAME> identified by password account unlock;
grant connect to <USER_NAME>;
grant create session, create table, create sequence, create trigger to <USER_NAME>;
alter user <USER_NAME> quota <SPACE_QUOTA_SIZE_IN_MEGABYTES> on '<TABLE_SPACE_NAME>'
commit;
```

3. Exit from the SQL*Plus session by executing the `quit` command.

Setting up the JDBC driver

Copy the Oracle JDBC libraries (for example, the `<ORACLE_HOME>/jdbc/lib/ojdbc14.jar` file) to the `<PRODUCT_HOME>/repository/components/lib` directory.

Remove the old database driver from the `<PRODUCT_HOME>/repository/components/dropins` directory when you upgrade the database driver.

What's next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with Oracle RAC, see Changing to Oracle RAC
Setting up PostgreSQL

The following sections describe how to set up PostgreSQL to replace the default H2 database in your WSO2 product:

- Setting up the database and login role
- Setting up the drivers

Setting up the database and login role

Follow the steps below to set up a PostgreSQL database.

1. Install PostgreSQL on your computer as follows:

   ```bash
   sudo apt-get install postgresql
   ```

2. Start the PostgreSQL service using the following command:

   ```bash
   sudo /etc/init.d/postgresql start
   ```

3. Create a database and the login role from a GUI using the PGAdminIII tool.

4. To connect PGAdminIII to a PostgreSQL database server, locate the server from the object browser, right-click the client and click Connect. This will show you the databases, tablespaces, and login roles as follows:

   ![PGAdminIII interface](image)

5. To create a database, click Databases in the tree (inside the object browser), and click New Database.

6. In the New Database dialog box, give a name to the database, e.g., gregdb and click OK.

7. To create a login role, click Login Roles in the tree (inside the object browser), and click New Login Role. Enter the role name and a password.

   These values will be used in the product configurations as described in the following sections. In the sample configuration, gregadmin will be used as both the role name and the password.

8. Optionally, enter other policies, such as the expiration time for the login and the connection limit.

9. Click OK to finish creating the login role.

Setting up the drivers
1. Download the PostgreSQL JDBC4 driver.
2. Copy the driver to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib` directory.

   For WSO2 IoT Server you need to copy the driver to the `<IOTS_HOME>/lib` directory.

What's next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with PostgreSQL, see Changing to PostgreSQL.

Setting up Derby

You can set up either an embedded Derby database or a remote database as described in the following topics.

- Setting up Embedded Derby
- Setting up Remote Derby

Setting up Embedded Derby

The following section describes how to set up an IBM DB2 database to replace the default H2 database in your WSO2 product:

Setting up the database

Follow the steps below to set up an embedded Derby database:

1. Download Apache Derby.
2. Install Apache Derby on your computer.

   For instructions on installing Apache Derby, see the Apache Derby documentation.

What’s next

By default, all WSO2 products are configured to use the embedded H2 database. To configure your product with it, see Changing to Embedded Derby.

Setting up Remote Derby

The following sections describe how to set up a remote Derby database to replace the default H2 database in your WSO2 product:

- Setting up the database
- Setting up the drivers

Setting up the database

Follow the steps below to set up a remote Derby database.

1. Download Apache Derby.
2. Install Apache Derby on your computer.

   For instructions on installing Apache Derby, see the Apache Derby documentation.

3. Go to the `<DERBY_HOME>/bin` directory and run the Derby network server start script. Usually, it is named `startNetworkServer`.

Setting up the drivers

Copy the derby.jar, derbyclient.jar JAR and the derbynet.jar JAR from the `<DERBY_HOME>/lib` directory to the `<PRODUCT_HOME>/repository/components/extensions/` directory (the classpath of the Carbon web application).

Changing the Carbon Database

WSO2 products are shipped with an H2 database, which serves as the default Carbon database. You can change this default database to one of the standard databases. See the links given 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 Embedded Derby

The following sections describe how to replace the default H2 database with embedded Derby:

- Setting up datasource configurations
- Creating database tables

Before you begin

You need to set up the embedded Derby before following the steps to configure your product with Embedded Derby. For more information, see Setting up Embedded Derby.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Embedded Derby database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource and point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.apache.derby.jdbc.EmbeddedDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user.</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<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 should elapse (when there are no available connections in the pool) before the system throws 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. Enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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 `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see [Updating WSO2 Products](#). This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see [Release Matrix](#).

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollbackOnReturn=false then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

Configuring the connection pool behavior on return

When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the ConnectionRollbackOnReturnInterceptor is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see [Updating WSO2 Products](#). This WUM update is only applicable to Carbon 4.4.11.

- Configure the connection pool to commit pending transactions on connection return
  1. Navigate to either one of the following locations based on your OS:
     - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh`
     - On Windows: `<PRODUCT_HOME>/bin/wso2server.bat`
  2. Add the following JVM option:

```
-Dndatasource.disable.rollbackOnReturn=true 
```

  4. Disable the defaultAutoCommit by defining it as false.
  5. Add the commitOnReturn property and set it to true for all the datasources, including the custom datasources.
Configure the connection pool to rollback pending transactions on connection return

1. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
2. Disable the `defaultAutoCommit` by defining it as `false`.
3. Add the `rollbackOnReturn` property to the datasources.

```
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <defaultAutoCommit>false</defaultAutoCommit>
      <rollbackOnReturn>true</rollbackOnReturn>
    </configuration>
  </definition>
</datasource>
```

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new database(s) you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
Creating database tables

You can create database tables by executing the database scripts as follows:

1. Run the ij tool located in the `<DERBY_HOME>/bin/` directory as illustrated below:

   ```
   client@ws02:~/dtb/db-derby-10.8.1.2-bin/bin$ /ij
   ij version 10.8
   ij>
   ```

2. Create the database and connect to it using the following command inside the ij prompt:

   ```
   connect 'jdbc:derby:repository/database/WSO2CARBON_DB;create=true';
   ```

   Replace the database file path in the above command with the full path to your database.

3. Exit from the ij tool by typing the `exit` command.

   ```
   exit;
   ```

4. Log in to the ij tool with the username and password that you set in `registry.xml` and `user-mgt.xml`:

   ```
   connect 'jdbc:derby:repository/database/WSO2CARBON_DB' user 'regadmin' password 'regadmin';
   ```

5. Use the scripts given in the following locations to create the database tables:
   - To create tables for the registry and user manager database (`WSO2CARBON_DB`), run the below command:

     ```
     run '<PRODUCT_HOME>/dbscripts/derby.sql';
     ```

     Now the product is running using the embedded Apache Derby database.

6. Restart the server.

   You can create database tables automatically **when starting the product for the first time** by using the `-Dsetup` parameter as follows.
   - For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
   - For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Deprecation of -DSetup**

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, from January 2018 onwards 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.

The product is configured to run using an embedded Apache Derby database.
Changing to Embedded H2

The following sections describe how to replace the default H2 database with Embedded H2:

- Setting up datasource configurations
- Creating database tables

H2 is not recommended in production

The embedded H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead.

You can use the embedded H2 database in development environments and as the local registry in a registry mount.

Before you begin

You need to set up Embedded H2 before following the steps to configure your product with it. For more information, see Setting up Embedded H2.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Embedded H2 database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```xml
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000</url>
      <username>ws2carbon</username>
      <password>ws2carbon</password>
      <driverClassName>org.h2.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```

In contrast to setting up with remote Derby, when setting up with the embedded mode, set the database driver name (the driverClassName element) to org.apache.derby.jdbc.EmbeddedDriver and the database URL (the url element) to the database directory location relative to the installation. In the above sample configuration, it is inside the <DERBY_HOME>/WSO2_CARBON_DB/ directory.

H2 is not recommended in production

The embedded H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead.

You can use the embedded H2 database in development environments and as the local registry in a registry mount.
The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollbackOnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see Tomcat JDBC Connection Pool.

The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see Release Matrix.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollbackOnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

Configuring the connection pool behavior on return

When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the ConnectionRollbackOnReturnInterceptor is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11.

- Configure the connection pool to commit pending transactions on connection return
  1. Navigate to either one of the following locations based on your OS.
     - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh`
     - On Windows: `<PRODUCT_HOME>\bin\wso2server.bat`
  2. Add the following JVM option:
Configure the connection pool to rollback pending transactions on connection return

1. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
2. Disable the `defaultAutoCommit` by defining it as `false`.
3. Add the `rollbackOnReturn` property and set it to `true` for all the datasources, including the custom datasources.

```
<Dndatasource.disable.rollbackOnReturn=true 
```

5. Disable the `defaultAutoCommit` by defining it as `false`.
6. Add the `commitOnReturn` property and set it to `true` for all the datasources, including the custom datasources.

```
<datasource> 
  ... 
  <definition type="RDBMS"> 
    <configuration> 
      ... 
      <defaultAutoCommit>false</defaultAutoCommit> 
      <commitOnReturn>true</commitOnReturn> 
      ... 
    </configuration> 
  </definition> 
</datasource> 
```

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new database(s) you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `<WSO2_CARBON_DB>` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

```
<dbConfig name="wso2registry"> 
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource> 
</dbConfig> 
```

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry"> 
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource> 
</dbConfig> 
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry"> 
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource> 
</dbConfig> 
```
Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in the H2 shell or web console:

- To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

```
<PRODUCT_HOME>/dbscripts/h2.sql
```

Follow the steps below to run the script in Web console:

1. Run the `.h2.sh` command to start the Web console.
2. Copy the script text from the SQL file.
3. Paste it into the console.
4. Click Run.
5. Restart the server.

Changing to IBM DB2

The following sections describe how to replace the default H2 database with IBM DB2:

- Setting up datasource configurations
- Creating database tables

**Before you begin**

You need to set up IBM DB2 before following the steps to configure your product with it. For more information, see [Setting up IBM DB2](#).

**Setting up datasource configurations**

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the IBM DB2 database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBON_DB datasource**

```
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```
Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:db2://SERVER_NAME:PORT/DB_NAME</url>
            <username>regadmin</username>
            <password>regadmin</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>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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 `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see Tomcat JDBC Connection Pool.
The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see Release Matrix.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollback OnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

Configuring the connection pool behavior on return

When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the ConnectionRollbackOnReturnInterceptor is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11.

- Configure the connection pool to commit pending transactions on connection return
  a. Navigate to either one of the following locations based on your OS.
     • On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh`
     • On Windows: `<PRODUCT_HOME>\bin\wso2server.bat`
  b. Add the following JVM option:

```
-Dndatasource.disable.rollbackOnReturn=true \
```

c. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
d. Disable the defaultAutoCommit by defining it as false.
e. Add the commitOnReturn property and set it to true for all the datasources, including the custom datasources.

```xml
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <defaultAutoCommit>false</defaultAutoCommit>
      <commitOnReturn>true</commitOnReturn>
      ...
    </configuration>
  </definition>
</datasource>
```

- Configure the connection pool to rollback pending transactions on connection return
b. Disable the defaultAutoCommit by defining it as false.
c. Add the rollbackOnReturn property to the datasources.
Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

   ```xml
   <Configuration>
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in the DB2 Express-C command editor.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

   ```sql
   <PRODUCT_HOME>/dbscripts/db2.sql
   ```

2. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`
Changing to IBM Informix

The following sections describe how to replace the default H2 databases with IBM Informix:

- Setting up datasource configurations
- Creating database tables

Before you begin

You need to set up IBM Informix before following the steps to configure your product with it. For more information, see Setting up IBM Informix.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the IBM Informix database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

```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>
            <!-- IP ADDRESS AND PORT OF DB SERVER -->
            <url>jdbc:informix-sqli://localhost:1533/AM_DB;CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</url>
            <username>wso2carbon</username>
            <password>wsocarbon</password>
            <driverClassName>com.informix.jdbc.IfxDriver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Deprecation of -DSetup

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, from January 2018 onwards 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.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000. You need to add the following configuration when specifying the connection URL as shown in the example above:</td>
</tr>
<tr>
<td>Add the following configuration to the connection URL when specifying it as shown in the example above: *CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IPX_USE_STRE NC=true;</td>
<td></td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollback OnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see [Updating WSO2 Products](#). This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see [Release Matrix](#).

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollback OnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

**Configuring the connection pool behavior on return**
When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the `ConnectionRollbackOnReturnInterceptor` is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see [Updating WSO2 Products](#). This WUM update is only applicable to Carbon 4.4.11.
• Configure the connection pool to commit pending transactions on connection return
  a. Navigate to either one of the following locations based on your OS.
     • On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh/`
     • On Windows: `<PRODUCT_HOME>\bin\wso2server.bat`
  b. Add the following JVM option:

     ```
     -Dndatasource.disable.rollbackOnReturn=true 
     ```

c. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
  d. Disable the `defaultAutoCommit` by defining it as `false`.
  e. Add the `commitOnReturn` property and set it to true for all the datasources, including the custom datasources.

```xml
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <defaultAutoCommit>false</defaultAutoCommit>
      <commitOnReturn>true</commitOnReturn>
      ...
    </configuration>
  </definition>
</datasource>
```

• Configure the connection pool to rollback pending transactions on connection return
  b. Disable the `defaultAutoCommit` by defining it as `false`.
  c. Add the `rollbackOnReturn` property to the datasources.

```xml
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <defaultAutoCommit>false</defaultAutoCommit>
      <rollbackOnReturn>true</rollbackOnReturn>
      ...
    </configuration>
  </definition>
</datasource>
```

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.
3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```xml
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

   ```
   <PRODUCT_HOME>/dbscripts/informix.sql
   ```

2. Restart the server.

**Deprecation of -DSetup**

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, from January 2018 onwards 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.

Changing to MariaDB

The following sections describe how to replace the default H2 database with MariaDB, which is a drop-in replacement for MySQL.

- Setting up datasource configurations
- Creating database tables

**Before you begin**

You need to set up MariaDB before following the steps to configure your product with it. For more information, see Setting up MariaDB.

Setting up datasource configurations
A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the MariaDB database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

### Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:mysql://localhost:3306/regdb</url>
         <username>regadmin</username>
         <password>regadmin</password>
         <defaultAutoCommit>false</defaultAutoCommit>
         <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>
       </configuration>
     </definition>
   </datasource>
   ```

   The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for MariaDB is 3306</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
</tbody>
</table>

Do not change the WSO2_CARBON_DB datasource name in the below configuration.
**defaultAutoCommit**

This property is **not** applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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.

---

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollbackOnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

**Configuring the connection pool behavior on return**

When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the ConnectionRollbackOnReturnInterceptor is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 Products, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11.

- **Configure the connection pool to commit pending transactions on connection return**
  - a. Navigate to either one of the following locations based on your OS.
    - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh/
    - On Windows: `<PRODUCT_HOME>in\wso2server.bat
  - b. Add the following JVM option:

    ```
    -Dndatasource.disable.rollbackOnReturn=true \
    ```

  - d. Disable the defaultAutoCommit by defining it as false.
  - e. Add the commitOnReturn property and set it to true for all the datasources, including the custom datasources.

---

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).
Configure the connection pool to rollback pending transactions on connection return
   b. Disable the defaultAutoCommit by defining it as false.
   c. Add the rollbackOnReturn property to the datasources.

<datasource>
    ...
    <definition type="RDBMS">
        <configuration>
            ...
            <defaultAutoCommit>false</defaultAutoCommit>
            <commitOnReturn>true</commitOnReturn>
            ...
        </configuration>
    </definition>
</datasource>

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the WSO2_CARBON_DB datasource above to the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file.

<dbConfig name="wso2registry">
    <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file.

<Configuration>
    <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

   ```
   mysql -u regadmin -p -D regdb < '<PRODUCT_HOME>/dbscripts/mysql.sql';
   ```

2. Restart the server.

You can create database tables automatically when starting the product for the first time by using the -Dsetup parameter as follows:

   - For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
   - For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`

**Deprecation of -DSetup**

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, from January 2018 onwards 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.

Changing to MSSQL

By default, WSO2 products use the embedded H2 database as the database for storing user management and registry data. Given below are the steps you need to follow in order to use a MSSQL database for this purpose.

- Setting up datasource configurations
  - Changing the default WSO2_CARBN_DB datasource
  - Configuring new datasources to manage registry or user management data
- Creating the database tables

**Before you begin**

You need to set up MSSQL before following the steps to configure your product with MSSQL. For more information, see Setting up Microsoft SQL.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBN_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Microsoft SQL database to replace the default H2 database, either change the default configurations of the WSO2_CARBN_DB datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CARBN_DB datasource**

Follow the steps below to change the type of the default WSO2_CARBN_DB datasource.

1. Edit the default datasourceconfiguration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:sqlserver://<IP>:1433;databaseName=wso2greg;SendStringParametersAsUnicode=false</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Give the URL to connect to the Microsoft SQL database. Shown below is the most simple form of the database connection URL. It includes the URL and two parameters:</td>
</tr>
<tr>
<td></td>
<td>jdbc:sqlserver://&lt;IP&gt;:&lt;PORT&gt;;databaseName=&lt;db_name_value&gt;;SendStringParametersAsUnicode=false</td>
</tr>
<tr>
<td></td>
<td>Change &lt;IP&gt; to the IP of the server. The best practice is to use port 1433 because you can use it in order processing services.</td>
</tr>
<tr>
<td></td>
<td>Set SendStringParametersAsUnicode to ‘false’ in order to overcome a limitation in the Microsoft SQL client driver. Without this parameter, the database driver will erroneously convert VARCHAR data into NVARCHAR and lower the database’s performance.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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 `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](#).

The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see [Updating WSO2 Products](#).

This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see [Release Matrix](#).

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If <code>defaultAutoCommit=false</code>, then you can set <code>commitOnReturn=true</code>, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if <code>rollbackOnReturn=true</code> then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If <code>defaultAutoCommit=false</code>, then you can set <code>rollbackOnReturn=true</code> so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

**Configuring the connection pool behavior on return**

When a database connection is returned to the pool, by default the product rolls back the pending transactions if `defaultAutoCommit=true`. However, if required you can disable the latter mentioned default behavior by disabling the `ConnectionRollbackOnReturnInterceptor`, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the `ConnectionRollbackOnReturnInterceptor` is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see [Updating WSO2 Products](#). This WUM update is only applicable to Carbon 4.4.11.

- **Configure the connection pool to commit pending transactions on connection return**
  a. Navigate to either one of the following locations based on your OS.
     - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh/
     - On Windows: `<PRODUCT_HOME>/bin\wso2server.bat
  b. Add the following JVM option:

     ```
     -Dndatasource.disable.rollbackOnReturn=true 
     ```

  c. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
  d. Disable the `defaultAutoCommit` by defining it as false.
  e. Add the `commitOnReturn` property and set it to true for all the datasources, including the custom datasources.
Configure the connection pool to rollback pending transactions on connection return

- Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
- Disable the `defaultAutoCommit` by defining it as false.
- Add the `rollbackOnReturn` property to the datasources.

```
<datasource>
  ...  
  <definition type="RDBMS">
    <configuration>
      ...  
      <defaultAutoCommit>false</defaultAutoCommit>
      <commitOnReturn>true</commitOnReturn>
      ...  
    </configuration>
  </definition>
</datasource>
```

Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `<WSO2_CARBON_DB>` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
Creating the database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

   `<PRODUCT_HOME>/dbscripts/mssql.sql`

2. Restart the server.

Changing to MySQL

By default, WSO2 products use the embedded H2 database as the database for storing user management and registry data. Given below are the steps you need to follow in order to use a MySQL database for this purpose.

- Creating the datasource connection to MySQL
- Updating other configuration files
- Creating database tables

Before you begin

You need to set up MySQL before following the steps to configure your product with MySQL. For more information, see Setting up MySQL.

Creating the datasource connection to MySQL

A datasource is used to establish the connection to a database. By default, WSO2.Carbon_DB datasource is configured in the `master-datasources.xml` file for the purpose of connecting to the default H2 database, which stores registry and user management data.

After setting up the MySQL database to replace the default H2 database, either change the default configurations of the WSO2.Carbon_DB datasource, or configure a new datasource to point it to the new database as explained below.

Follow the steps below to change the type of the default WSO2.Carbon_DB datasource.

1. Open the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file and locate the `<datasource>` configuration element.
2. You simply have to update the URL pointing to your MySQL database, the username, and password required to access the database and the MySQL driver details as shown below.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for MySQL is 3306</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
</tbody>
</table>

```
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/regdb</url>
            <username>regadmin</username>
            <password>regadmin</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>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

3. You can update the configuration elements given below 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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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 `<PRODUCT_HOME>/repository/conf/datasources.xml` file, see Tomcat JDBC Connection Pool.
### Configuring the connection pool behavior on return

When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the [ConnectionRollbackOnReturnInterceptor](https://docs.wso2.com/display/WSO2CAM/Configuration+of+Connection+Pool+Interceptors) and setting the connection pool behavior on return via the datasource configurations by using the following options.

- **Configure the connection pool to commit pending transactions on connection return**
  1. Navigate to either one of the following locations based on your OS.
     - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh`
     - On Windows: `<PRODUCT_HOME>/bin\wso2server.bat`
  2. Add the following JVM option:
     ```
     -Dndatasource.disable.rollbackOnReturn=true \n     ```
  4. Disable the `defaultAutoCommit` by defining it as false.
  5. Add the `commitOnReturn` property and set it to true for all the datasources, including the custom datasources.

```xml
<datasource>
  ... 
  <definition type="RDBMS">
    <configuration>
      ...
      <defaultAutoCommit>false</defaultAutoCommit>
      <commitOnReturn>true</commitOnReturn>
      ...
    </configuration>
    ... 
  </definition>
</datasource>
```

- **Configure the connection pool to rollback pending transactions on connection return**
  1. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
  2. Disable the `defaultAutoCommit` by defining it as false.
  3. Add the `rollbackOnReturn` property to the datasources.

### Table: Connection Pool Attributes and Description

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If <code>defaultAutoCommit=false</code>, then you can set <code>commitOnReturn=true</code>, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if <code>rollbackOnReturn=true</code> then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If <code>defaultAutoCommit=false</code>, then you can set <code>rollbackOnReturn=true</code> so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>
Updating other configuration files

- The registry.xml file (stored in the <PRODUCT_HOME>/repository/conf directory) specifies the datasource information corresponding to the database that stores registry information. Therefore, if you have changed the datasource name, you need to update the following accordingly:

```xml
<dbConfig name="wso2registry">
    <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

- The user-mgt.xml file (stored in the <PRODUCT_HOME>/repository/conf directory) specifies the datasource information corresponding to the database that stores user management information. Therefore, if you have changed the datasource name, you need to update the following accordingly:

```xml
<Configuration>
    <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

Creating database tables

To create the database tables, connect to the database that you created earlier and run the relevant scripts.

Alternatively, you can create database tables automatically when starting the product for the first time by using the -Dsetup parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup

**Deprecation of -DSetup**

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, from January 2018 onwards 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.

1. To create tables in the registry and user manager database (WSO2CARBON_DB), execute the relevant script as shown below.
Changing to Oracle

By default, WSO2 products use the embedded H2 database as the database for storing user management and registry data. Given below are the steps you need to follow in order to use an Oracle database for this purpose.

- Setting up datasource configurations
- Changing the default WSO2_CARBON_DB datasource
- Configuring new datasources to manage registry or user management data
- Creating the database tables

Before you begin

You need to set up Oracle before following the steps to configure your product with Oracle. For more information, see Setting up Oracle.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Oracle database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the file <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml as shown below.

For MySQL 5.7:

From Carbon kernel 4.4.6 onwards your product will be shipped with two scripts for MySQL as follows (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.

Note that if you are automatically creating databases during server startup using the -DSetup option, the mysql.sql script will be used by default to set up the database. Therefore, if you have MySQL version 5.7 set up for your server, be sure to do the following before starting the server:

a. First, change the existing mysql.sql file to a different filename.
b. Change the mysql.sql script to mysql.sql.
c. Change the mysql.sql script to mysql.sql.

MySQL 5.7 is only recommended for products that are based on Carbon 4.4.6 or a later version.

2. Restart the server.
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:oracle:thin:@SERVER_NAME:PORT/SID</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>oracle.jdbc.OracleDriver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1 FROM DUAL</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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>

The default port for Oracle is 1521.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see Tomcat JDBC Connection Pool.
### The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see Release Matrix.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if rollbackOnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

### Configuring the connection pool behavior on return

When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

**Disabling the ConnectionRollbackOnReturnInterceptor is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11.**

- **Configure the connection pool to commit pending transactions on connection return**
  a. Navigate to either one of the following locations based on your OS.
     - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh`
     - On Windows: `<PRODUCT_HOME>/bin\wso2server.bat`
  b. Add the following JVM option:

     ```
     -Dndatasource.disable.rollbackOnReturn=true \
     ```
  c. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
  d. Disable the defaultAutoCommit by defining it as false.
  e. Add the commitOnReturn property and set it to true for all the datasources, including the custom datasources.

```
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
    <defaultAutoCommit>false</defaultAutoCommit>
    <commitOnReturn>true</commitOnReturn>
    ...
  </configuration>
  </definition>
</datasource>
```

- **Configure the connection pool to rollback pending transactions on connection return**
  b. Disable the defaultAutoCommit by defining it as false.
  c. Add the rollbackOnReturn property to the datasources.
Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

   ```xml
   <dbConfig name="wso2registry">
     <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

   ```xml
   <Configuration>
     <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

Creating the database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in SQL*Plus:

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

   ```sql
   SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql
   ```

2. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`
Changing to Oracle RAC

By default, WSO2 products use the embedded H2 database as the database for storing user management and registry data. Given below are the steps you need to follow in order to use an Oracle RAC database for this purpose.

- Setting up datasource configurations
  - Changing the default WSO2_CARBON_DB datasource
  - Configuring new datasources to manage registry or user management data
- Creating the database tables

Before you begin

You need to set up Oracle RAC before following the steps to configure your product with Oracle RAC. For more information, see Setting up Oracle RAC.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Oracle RAC database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.
```xml
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on) (ADDRESS=(PROTOCOL=TCP)(HOST=racnode1) (PORT=1521)) (ADDRESS=(PROTOCOL=TCP)(HOST=racnode2) (PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=rac)))</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1 FROM DUAL</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be 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 will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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 `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](https://tomcat.apache.org/download-source.html).
Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the WSO2_CARBON_DB datasource above to the <PRODUCT_HOME>/repository/conf/da
tasources/master-datasources.xml file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the <PRODUCT_HOME>/repository/conf/registry.xml file.

   ```xml
   <dbConfig name="wso2registry">
   <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
   </dbConfig>
   ```

3. If you are setting up a separate database to store user management data, update the following configurations in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file.

   ```xml
   <Configuration>
   <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
   </Configuration>
   ```

Creating the database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in SQL*Plus:

1. To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

   ```sql
   SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql
   ```

2. Restart the server.

You can create database tables automatically when starting the product for the first time by using the -Dsetup parameter as follows:

   - For Windows: <PRODUCT_HOME>/bin/wso2server.bat -Dsetup
   - For Linux: <PRODUCT_HOME>/bin/wso2server.sh -Dsetup

Deprecation of -DSetup

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, from January 2018 onwards 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.

Changing to PostgreSQL

By default, WSO2 products use the embedded H2 database as the database for storing user management and registry data. Given below are the steps you need to follow in order to use PostgreSQL for this purpose.
• Setting up datasource configurations
  • Changing the default WSO2_CARBON_DB datasource
  • Configuring new datasources to manage registry or user management data
• Creating database tables

Before you begin
You need to set up PostgreSQL before following the steps to configure your product with PostgreSQL. For more information, see Setting up PostgreSQL.

Setting up datasource configurations
A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the PostgreSQL database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource
Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file as shown below.

   Make sure to set the following elements:
   - `validationQuery` - `SELECT 1; COMMIT`
   - `defaultAutoCommit` - `false`
   - `testOnBorrow` - `true`

   **Note:** If you are using WSO2 Identity Server 5.7.0 or a later version, set the `<defaultAutoCommit>` property to `true`.

   ```xml
   <datasource>
   <name>WSO2_CARBON_DB</name>
   <description>The datasource used for registry and user manager</description>
   <jndiConfig>
   <name>jdbc/WSO2CarbonDB</name>
   </jndiConfig>
   <definition type="RDBMS">
   <configuration>
   <url>jdbc:postgresql://localhost:5432/gregdb</url>
   <username>regadmin</username>
   <password>regadmin</password>
   <driverClassName>org.postgresql.Driver</driverClassName>
   <maxActive>80</maxActive>
   <maxWait>60000</maxWait>
   <minIdle>5</minIdle>
   <testOnBorrow>true</testOnBorrow>
   <validationQuery>SELECT 1; COMMIT</validationQuery>
   <defaultAutoCommit>false</defaultAutoCommit>
   <validationInterval>30000</validationInterval>
   </configuration>
   </definition>
   </datasource>
   
   The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a PostgreSQL instance is 5432.</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user.</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver.</td>
</tr>
<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 will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>Whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>Whether to commit database changes automatically or not.</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 will not be validated again.</td>
</tr>
</tbody>
</table>
| defaultAutoCommit        | This property is **not** applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for RDBMS drivers 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. **Note:** Set this property to true if you are using WSO2 Identity Server 5.7.0 or a later version.

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see **Tomcat JDBC Connection Pool**.

The following elements are available only as a **WUM** update and is effective from 14th September 2018 (2018-09-14). For more information, see **Updating WSO2 Products**. This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see **Release Matrix**.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If <code>defaultAutoCommit=false</code>, then you can set <code>commitOnReturn=true</code>, so that the pool can complete the transaction by calling the commit on the connection as it is returned to the pool. However, if <code>rollbackOnReturn=true</code> then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If <code>defaultAutoCommit=false</code>, then you can set <code>rollbackOnReturn=true</code> so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

**Configuring the connection pool behavior on return**

When a database connection is returned to the pool, by default the product rolls back the pending transactions if `defaultAutoCommit=true`. However, if required you can disable the latter mentioned default behavior by disabling the `ConnectionRollbackOnReturnInterceptor`, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the `ConnectionRollbackOnReturnInterceptor` is only possible with the **WUM** update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see **Updating WSO2 Products**. This WUM update is only applicable to Carbon 4.4.11.

- **Configure the connection pool to commit pending transactions on connection return**
  a. Navigate to either one of the following locations based on your OS.
     - On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh`
     - On Windows: `<PRODUCT_HOME>\bin\wso2server.bat`
  b. Add the following JVM option:
c. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. 
d. Disable the `defaultAutoCommit` by defining it as false. 
e. Add the `commitOnReturn` property and set it to true for all the datasources, including the custom datasources.

```
<datasource>
    ...
    <configuration>
        ...
        <defaultAutoCommit>false</defaultAutoCommit>
        <commitOnReturn>true</commitOnReturn>
        ...
    </configuration>
    ...
</definition>
</datasource>
```

- Configure the connection pool to rollback pending transactions on connection return
b. Disable the `defaultAutoCommit` by defining it as false. 
c. Add the `rollbackOnReturn` property to the datasources.

```
<datasource>
    ...
    <definition type="RDBMS">
        <configuration>
            ...
            <defaultAutoCommit>false</defaultAutoCommit>
            <rollbackOnReturn>true</rollbackOnReturn>
            ...
        </configuration>
        ...
    </definition>
</datasource>
```

2. Download the Postgres JDBC driver for the version you are using and copy it to the `<PRODUCT_HOME>/repository/components/lib` folder.

### Configuring new datasources to manage registry or user management data

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB` datasource above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.
2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.
If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

```xml
<Configuration>
    <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>
```

Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:

   ```bash
   <PRODUCT_HOME>/dbscripts/postgresql.sql
   ```

2. Restart the server.

Changing to Remote H2

The following sections describe how to replace the default H2 databases with Remote H2:

- Setting up datasource configurations
- Creating database tables

### Deprecation of -DSetup

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, from January 2018 onwards 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.

### H2 is not recommended in production

The embedded H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead.

You can use the embedded H2 database in development environments and as the local registry in a registry mount.
Before you begin

- Set up the database as explained in Setting up Remote H2.
- Download the H2 database driver and copy it to WSO2 IoT Server by following the steps under Setting up drivers.

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, WSO2_CARBON_DB datasource is used to connect to the default H2 database, which stores registry and user management data. After setting up the Remote H2 database to replace the default H2 database, either change the default configurations of the WSO2_CARBON_DB datasource, or configure a new datasource to point it to the new database as explained below.

Changing the default WSO2_CARBON_DB datasource

Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file as shown below.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
      <configuration>
        <url>jdbc:h2:tcp://localhost/~/registryDB;create=true</url>
        <username>regadmin</username>
        <password>regadmin</password>
        <driverClassName>org.h2.Driver</driverClassName>
        <maxActive>80</maxActive>
        <maxWait>60000</maxWait>
        <minIdle>5</minIdle>
        <testOnBorrow>true</testOnBorrow>
        <validationQuery>SELECT 1</validationQuery>
        <validationInterval>30000</validationInterval>
        <defaultAutoCommit>false</defaultAutoCommit>
      </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (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, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
</tbody>
</table>
The following elements are available only as a WUM update and is effective from 14th September 2018 (2018-09-14). For more information, see Updating WSO2 Products.
This WUM update is only applicable to Carbon 4.4.11 and will be shipped out-of-the-box with Carbon versions newer than Carbon 4.4.35. For more information on Carbon compatibility, see Release Matrix.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitOnReturn</td>
<td>If defaultAutoCommit=false, then you can set commitOnReturn=true, so that the pool can commit the transaction by calling the commit on the connection as it is returned to the pool. However, if rollbackOnReturn=true then this attribute is ignored. The default value is false.</td>
</tr>
<tr>
<td>rollbackOnReturn</td>
<td>If defaultAutoCommit=false, then you can set rollbackOnReturn=true so that the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool. The default value is false.</td>
</tr>
</tbody>
</table>

Configuring the connection pool behavior on return
When a database connection is returned to the pool, by default the product rolls back the pending transactions if defaultAutoCommit=true. However, if required you can disable the latter mentioned default behavior by disabling the ConnectionRollbackOnReturnInterceptor, which is a JDBC-Pool JDBC interceptor, and setting the connection pool behavior on return via the datasource configurations by using the following options.

Disabling the ConnectionRollbackOnReturnInterceptor is only possible with the WUM update and is effective from 14th September 2018 (2018-09-14). For more information on updating WSO2 API Manager, see Updating WSO2 Products. This WUM update is only applicable to Carbon 4.4.11.

- Configure the connection pool to commit pending transactions on connection return
  a. Navigate to either one of the following locations based on your OS.
     * On Linux/Mac OS: `<PRODUCT_HOME>/bin/wso2server.sh/
     * On Windows: `<PRODUCT_HOME>`\`bin\wso2server.bat
  b. Add the following JVM option:

```
-Dndatasource.disable.rollbackOnReturn=true
```
  c. Navigate to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.
  d. Disable the defaultAutoCommit by defining it as false.
  e. Add the commitOnReturn property and set it to true for all the datasources, including the custom datasources.
Configure the connection pool to rollback pending transactions on connection return
b. Disable the `defaultAutoCommit` by defining it as `false`.
c. Add the `rollbackOnReturn` property to the datasources.

```
<datasource>
  ...
  <definition type="RDBMS">
    <configuration>
      ...
      <defaultAutoCommit>false</defaultAutoCommit>
      <commitOnReturn>true</commitOnReturn>
      ...
    </configuration>
  </definition>
</datasource>
```

**Configuring new datasources to manage registry or user management data**

Follow the steps below to configure new datasources to point to the new databases you create to manage registry and/or user management data separately.

1. Add a new datasource with similar configurations as the `WSO2_CARBON_DB datasource` above to the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Change its elements with your custom values. For instructions, see Setting up datasource configurations.

2. If you are setting up a separate database to store registry-related data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/registry.xml` file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/MY_DATASOURCE_NAME</dataSource>
</dbConfig>
```

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
<Configuration>
  <Property name="dataSource">jdbc/MY_DATASOURCE_NAME</Property>
</Configuration>

Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts in H2 shell or web console:

- To create tables in the registry and user manager database (WSO2CARBON_DB), use the below script:

  `<PRODUCT_HOME>/dbscripts/h2.sql`

Follow the steps below to run the script in Web console:

1. Run the `./h2.sh` command to start the Web console.
2. Copy the script text from the SQL file.
3. Paste it into the console.
4. Click Run.
5. Restart the server.

You can create database tables automatically when starting the product for the first time by using the `-Dsetup` parameter as follows:

- For Windows: `<PRODUCT_HOME>/bin/wso2server.bat -Dsetup`
- For Linux: `<PRODUCT_HOME>/bin/wso2server.sh -Dsetup`
Deprecation of -DSetup

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, from January 2018 onwards 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.

Browsing the H2 Database

All WSO2 products are shipped with a default H2 database. In some product scenarios, you may need to access a database table to see how it works, to troubleshoot, or to try out the scenario.

Follow the instructions given below to connect to the H2 database and browse through it.

H2 is not recommended in production

The embedded H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead.

You can use the embedded H2 database in development environments and as the local registry in a registry mount.

1. Open the <PRODUCT_HOME>/repository/conf/carbon.xml file and paste the following configuration.

Do not uncomment the existing <H2DatabaseConfiguration> element. Just paste the following configuration below it.

```
<H2DatabaseConfiguration>
  <property name="web"/>
  <property name="webPort">8082</property>
  <property name="webAllowOthers"/>
</H2DatabaseConfiguration>
```

2. Restart the WSO2 product and access the following URL via your browser: http://localhost:8082

Tip: If you are logged in to the management console, logout before connecting to the database.

3. Enter the following details and click Connect.
   a. JDBC URL : jdbc:h2:[file path to <PRODUCT_HOME>/repository/database/WSO2CARBON_DB]
   b. username : wso2carbon
   c. password : wso2carbon

The database tables are listed on the left. You can now browse through them as required.

Working with the Registry

A registry is a content store and a metadata repository for various artifacts such as services, WSDLs and configuration files. These artifacts are keyed by unique paths where a path is similar to a Unix file path. In WSO2 products, all configurations pertaining to modules, logging, security, data sources and other service groups are stored in the registry by default.

The registry kernel of WSO2 provides the basic registry and repository functionality. WSO2 products use the services provided by the registry kernel to establish their own registry spaces, which are utilized for storing data and persisting configuration. Here are some of the features provided by the WSO2 Registry interface:

- Provides the facility to organize resources into collections.
- Keeps multiple versions of resources.
- Manages social aspects such as rating of resources.
- Provides AtomPub interfaces to publish, view and manage resources from remote or non-Java clients.
The Registry space of any WSO2 product contains three major partitions:

- **Local Repository**: Used to store configuration and runtime data that is local to the server. This partition is not to be shared with multiple servers. Mount point is `/system/local`.
- **Configuration Repository**: Used to store product-specific configurations. This partition can be shared across multiple instances of the same product (e.g., sharing ESB configurations across an ESB cluster). Mount point is `/system/config`.
- **Governance Repository**: Used to store configuration and data that are shared across the whole platform. This typically includes services, service descriptions, endpoints or datasources. Mount point of this registry is `/system/governance`.

You can browse the contents of the registry using the product's management console.

This section provides the following information:

- **Managing the Registry**
- **Searching the Registry**
- **Using Remote Registry Instances for the Registry Partitions**

### Managing the Registry

1. Log in to the product's management console and select Browse from the Registry menu that is under the Main menu.

   ![Registry](image)

2. The **Browse** page opens. For example,

   ![Browse](image)

3. Click a registry artifact from the tree view and you will be navigated to its detail view. For example,

   ![Browse](image)

   It gives the following four main components:
   - Metadata
   - Properties
• Entries and Content
• Role Permissions

Metadata

The Metadata panel allows you to manage resource metadata and revisions using the Create Checkpoint and View Versions options. Each time you create a check point, it is added as a new reversion of the resource. Revisions is a useful way to facilitate Disaster Recovery and Fault Tolerance in the registry. By creating a revision, a user essentially saves a snapshot of the current state of a resource or collection that can be restored at a later date. The registry’s checkpoint and restoration mechanisms are similar to that of System Restore of Microsoft Windows.

The Metadata panel displays the following properties of the resource or the collection:

- **Created** - Time the resource was created and the author of the resource.
- **Last Updated** - Time the resource was updated and the author of the alterations.
- **Media Type** - An associated Media type of the resource/collection. For more information about Media types, see Adding a Resource.
- **Checkpoint** - Allows to create a checkpoint (URL for the permanent link) of a resource/collection.
- **Versions** - Allows to view versions of a resource/collection.
- **Permalink** - Holds the resource URL in both HTTP and HTTPS. (e.g., http://10.100.2.76:9763/registry/resource/_system/governance/trunk/services/test)
- **Description** - Description of the resource/collection.

For example,

<table>
<thead>
<tr>
<th>Created:</th>
<th>By admin on 03 Aug 11:08:28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Updated:</td>
<td>By admin on 03 Aug 11:08:05</td>
</tr>
<tr>
<td>Media Type:</td>
<td>text/plain</td>
</tr>
<tr>
<td>Checkpoint:</td>
<td><img src="create_checkpoint.png" alt="Create Checkpoint" /></td>
</tr>
<tr>
<td>Versions:</td>
<td><img src="view_versions.png" alt="View versions" /></td>
</tr>
<tr>
<td>Permalink:</td>
<td><img src="http.png" alt="HTTP" /> <img src="https.png" alt="HTTPS" /></td>
</tr>
<tr>
<td>Description:</td>
<td><img src="edit.png" alt="Edit" /></td>
</tr>
</tbody>
</table>

**Creating a checkpoint**

To create a checkpoint, click on the **Create Checkpoint** link:

**NOTE**: When checkpoints are created, properties, comments, ratings and tags will also be taken into consideration. If you do not want them to be versioned along with resource content, you can disable it by making changes to the Static Configuration. However, these changes need to be done before the server starts for the first time.

**Viewing Versions**

To view the resource versions, click on the **View versions** link:
It opens the versions. For example,

**Versions of**

_/system/governance/trunk/services/test_

<table>
<thead>
<tr>
<th>Version</th>
<th>Last Modified Date</th>
<th>Last Modified By</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18m ago</td>
<td>admin</td>
<td>Details</td>
</tr>
</tbody>
</table>

This page gives the following information:

- The number of a resource/collection version
- Last date of modifications and the author who did the last alterations
- **Actions**
  - Details - Opens the **Browse** page of a resource/collection version to view its details
  - Restore - Restores a selected version
  - Delete Version History - Delete the version history

To learn more about restoring to a previous version, see [read here](#).

**NOTE**: Versions and checkpoints are not available for **Symbolic Links** and **Remote Links**.

### Properties

The **Properties** panel displays the properties of the currently selected resource or collection. New properties can be added, while existing properties can be edited or deleted.

1. To add a property, click on the **Add New Property** link in the **Properties** panel.

2. Enter a unique name for the property and a value and click **Add**.
3. After adding the property, you can edit or delete it using the **Edit** and **Delete** links associated with it.

**Entries and Content**

A collection includes a set of child collections and resources. If you select a collection in the registry, the **Entries** panel opens with details of the child collections and resources of the collection. For example,

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>prop1</td>
<td>val</td>
<td></td>
</tr>
</tbody>
</table>

The **Info** and **Actions** links in the **Entries** panel provide the following information:

<table>
<thead>
<tr>
<th>Links in the Entries panel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Collection</td>
<td>See <a href="#">Adding child collections</a></td>
</tr>
<tr>
<td>Add Resource</td>
<td>See <a href="#">Adding and Editing resources</a></td>
</tr>
<tr>
<td>Create Link</td>
<td>See <a href="#">Entries and Content</a> # Adding links</td>
</tr>
</tbody>
</table>

**Info**

- **Feed** - Generate an RSS feed for the resource
- **Rating** - Ratings of the resource
- **Media type** - Each collection or resource created and stored on the repository has an associated media type. If you leave the media type unspecified, it takes the default value. There are two main ways to configure media types for resources:
  1. Use a one-time configuration, by modifying the `mime.types` file found in the server configuration directory. Done just once before the initial start-up of the server. This method does not apply to collections. The only way to configure media types for a collection is using the second method.
  2. Use the management console as described when adding a collection and a resource.

Initially, the system contains media types defined in `mime.types` file. They are available for resources and a set of default media types is available for collections.

You can manage media types for resources by editing the properties of `/system/mime.types/index` collection. This collection contains two resources: `collection` and `custom.ui`. To manage media types of collections and custom user interfaces, edit the properties of these two resources.

---

Human-readable media types have shorter names in place of MIME names (for example, WSDL is used instead of application/wsdl+xml). This is achieved by introducing a new file as `<PRODUCT_HOME>/repository/conf/mime.mappings`. For more information, see Configuring Registry Files section in WSO2 Governance Registry documentation.
### Actions

| Allow you to rename, move, copy or delete a resource/child collection. |

You cannot move/copy resources and collections across registry mounts if they have dependencies or associations. You can only move/copy within a mount. For more information on mounts, read WSO2 Governance Registry documentation: Remote Instance and Mount Configuration Details.

These options are not available for all resources/collections.

#### Adding child collections

You can create a child collection to existing collections in a registry as shown below:

1. Select a collection. You can see the Entries panel with details of child collections and resources it has.
2. Click Add Collection.

3. Specify the following options:
   - A unique name for the collection and a description
   - Select a media type from the drop-down menu
4. Click Add.

#### Adding resources to a collection

You can add a resource to a collection for more convenient usage of resources as follows:

1. Select a collection. In its detailed view, you can see the Entries panel with details of child collections and resources it has.
2. In the Entries panel, click Add Resource.

3. In the Add Resource page that opens, select one of the following methods:
   - Uploading content from file
   - Importing content from URL
   - Creating text content
   - Creating custom content

**Uploading content from file**

Allows you to create a resource by fetching its content from a specified file (e.g., XML, WSDL, JAR). For example,
### Importing content from URL

Allows you to fetch a resource from a specified URL path. For example,

You can add external links (hyperlinks) as resources in the registry. To add such a link, create a text resource with the media type `application/vnd.wso2-hyperlink` and specify the URL as the resource's content.

### Creating text content

Allows you to write the content in the UI itself, using either the Rich Text Editor or Plain Text Editor. For example,
Creating custom content

Allows you to create your own type of content with a specified media type. For example, to add a user profile to the registry, create custom content with the media type `application/vnd.wso2-profiles+xml` and provide the user name. For example,

![Add Resource](image)

Editing resources

If you select a resource, in its detailed view, you can see the **Content** panel, which provides a UI to edit, upload, and download the content as follows:

- **Display as text**: Allows only to view the configuration of a resource. For example,

  ![Display as text](image)

- **Edit as text**: Allows to edit a resource either in plain text editor or rich text editor.

  ![Edit as text](image)

- **Upload**: Allows to upload a file to the resource. The existing content of the resource will be replaced by the you upload.

  **NOTE**: Be sure to upload a file of the same type as the existing resource, in order to avoid corrupting the resource. For resources such as WSDLs or Schemas, do not upload modifications that include changes to namespaces and imports.

- **Download**: Allows to download a resource from its host to a file in a specified location.

  If a Security Warning appears when you try to download a resource, click **Save** to start downloading.
Adding links

**Symbolic links** are much like hyperlinks to existing resources or collections in the local registry instance. When you access a symbolic link, the actual resource metadata, community features, associations and the content can be viewed from that location. If you make a change to the Symlink resource, the actual resource will get affected. These Symlink resources are soft links so that the actual resource does not keep a count for links from which it is referenced by. Instead the link shows a resource with a description saying that it could not make link to the original resource. Symbolic and remote links does not support versioning at the moment.

**Remote links** are created to mount a collection of a remotely deployed registry instance to the local instance. Any path/collection of the remote instance can be mounted at any point in the local instance. After mounting, the rest is very similar to symbolic links. You can work on the remote resource from the local instance.

Follow the instructions below to create a link on a resource/collection in the collection.

1. Symbolic links and Remote links can be created in a similar way to adding a normal resource. To add a link, click "Create Link" in the "Entries" panel.

2. From the drop-down menu, select a symbolic or a remote link to add:
   - Symbolic links
     When adding a Symbolic link, enter a name for the link and the path of the existing resource or collection being linked. It creates a link to the particular resource.

The created Symbolic link is shown by an icon with an arrow in the Entries panel.
Remote links

You can mount a collection in a remotely deployed registry instance to your registry instance by adding a remote link. Provide a name, the instance to which you are going to mount and also the path of the remote collection. If no path is given, the root collection will be mounted.

After mounting the Remote collection, you can access and work on that collection from your local instance.

Adding a Resource

You can add a resource to a certain collection for more convenient usage of the Registry.

Follow the instructions below to add a new child entry to a collection.

1. To add a new resource, click on the Add Resource link.

2. In the Add Resource panel, select Method from the drop-down menu.

The following methods are available:

- Upload content from file
- Import content from URL
- Create Text content
- Create custom content

Uploading Content from File

1. If this method was selected, specify the following options:
   - File - The path of a file to fetch content (XML, WSDL, JAR etc.) Use the Browse button to upload a file.
   - Name - The unique name of a resource.
   - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
• Description - The description of a resource.

2. Click Add once the information is added as shown in the example below.

**Importing Content from URL**

1. If this method was selected, specify the following options:
   - URL - The full URL of the resource to fetch content from URL.
   - Name - The unique name of a resource.
   - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
   - Description - The description of a resource.

2. Click Add once the information is added.

**Text Content Creation**

1. If this method was selected, specify the following options:
   - Name - The unique name of a resource.
   - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
   - Description - The description of a resource.
   - Content - The resource content. You can use either Rich Text Editor or Plain Text Editor to enter.

2. Click Add once the information is added.
Custom Content Creation

1. If this method was selected, choose the Media Type from the drop-down menu and click Create Content.

Media Types

Each collection and resource created and stored on the repository has an associated media type. However, you also have the option to leave this unspecified enforcing the default media type. There are two main ways to configure media types for resources.

- The first method is by means of a one-time configuration, which can be done by modifying the "mime.types" file found in <CARBON_HOME>/repo/sitory/conf directory. This can be done just once before the initial start-up of the server.
- The second method is to configure the media types via the server administration console. The first method does not apply for collections, and the only available mechanism is to configure the media types via the server administration console.

Initially the system contains the media types defined in the mime.types file will be available for resources and a set of default media types will be available for collections.

Managing media types for resources can be done via the server administration console, by editing the properties of the /system/mime.types/index collection. This collection contains two resources, collection and custom.ui. To manage media types of collections and custom user interfaces, edit the properties of these two resources.

Link Creation

Follow the instructions below to create a link on a resource/collection.

1. Symbolic links and Remote links can be created in a similar way to adding a normal resource. To add a link, click Create Link in the Entries panel.
2. Select a link to add from the drop-down menu.

A Symbolic Link

When adding a Symbolic link, enter a name for the link and the path of an existing resource or collection which is being linked. It creates a link to the particular resource.

A Remote Link

You can mount a collection in a remotely-deployed registry instance to your registry instance by adding a Remote link. Provide a name for the Remote link in the name field. Choose the instance to which you are going to mount and give the path of the remote collection which you need to mount for the path field, or else the root collection will be mounted.
If you select a collection, in its detailed view, you can see the **Entries** panel with details of child collections and resources it has. It provides a UI to view details, add new resources, collections and links as follows:

- Add Resource
- Add Collection
- Create Link
- Child resource/collection information such as name, created date and author
- The Info link specifying media type, feed, rating.
- The Actions link to rename, move, copy or delete a resource/collection

You cannot move/copy resources and collections across registry mounts if they have dependencies or associations. You can only move/copy within a mount. For more information on mounts, read WSO2 Governance Registry documentation: [Remote Instance and Mount Configuration Details](#).

These options are not available for all resources/collections.

### Role Permissions

When you select a collection in the registry, the **Permissions** panel opens with the defined role permissions available. It allows you to specify which role has access to perform which operations on a registry resource or a collection.

**Adding new role permissions**

1. In the **New Role Permissions** section, select a role from the drop-down list. This list is populated by all user roles configured in the system.

   ![New Role Permissions](#)

   The `wso2.anonymous.role` is a special role that represents a user who is not logged in to the management console. Granting Read access to this role means that you do not require authentication to access resources using the respective Permalinks.

   The `everyone` role is a special role that represents a user who is logged into the management console. Granting Read access to this role means that any user who has logged into the management console with sufficient permissions to access the Resource Browser can read the respective resource. Granting Write or Delete access means that any user who is logged in to the management console with sufficient permissions to access the Resource Browser can make changes to the respective resource.

2. Select one of the following actions:
   - **Read**
   - **Write**
   - **Delete**
   - **Authorize** - A special permission that gives a role the ability to grant and revoke permissions to/from others

3. Select whether to allow the action or deny and click **Add Permission**. For example

   ![Add Permission](#)
Deny permissions have higher priority over Allow. That is, a Deny permission always overrides an Allow permission assigned to a role.

Deny permission must be given at the collection level. For example, to deny the write/delete action on a given policy file, set Write/Delete actions for the role to Deny in /trunk/policies. If you set the Deny permission beyond the collection level (e.g., / or /_system etc.) it will not be applied for the user’s role.

4. The new permission appears in the list.

<table>
<thead>
<tr>
<th>Role</th>
<th>Read</th>
<th>Write</th>
<th>Delete</th>
<th>Authorize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Deny</td>
<td>Allow</td>
<td>Deny</td>
<td>Allow</td>
</tr>
</tbody>
</table>

Apply All Permissions  Reset

From here, you can edit the permissions by selecting and clearing the check boxes. After editing the permissions, click Apply All Permissions to save the alterations.

Searching the Registry

The management console provides facility to search all resources in the registry.

1. Log in to the product’s management console and select Search -> Metadata under the Registry menu.

2. The Search page opens.
You can define a search criteria using the following parameters:

- **Resource name**
- **Created/updated date range** - The date when a resource was created/updated
- **Created/updated author** - The person who created/updated the resource
- **Tags and comments**
- **Property Name**
- **Property Value**
- **Media Type**

Created/updated dates must be in MM/DD/YYYY format. Alternatively, you can pick it from the calendar interface provided.

To search for matches containing a specific pattern, use the "%" symbol.
3. Fill the search criteria and click **Search** to see the results in the **Search Results** page.

**Using Remote Registry Instances for the Registry Partitions**

You can configure and use the registry space in one of the following ways:

- Use the registry space shipped by default with the product.
- Use remote registry instances for different registry partitions. These partitions can also be shared across multiple product instances.

This guide explains the second option using WSO2 Governance Registry as the remote registry instance.

The registry space contains three major partitions as local, configuration and governance repositories. For more information on these partitions, see *Working with the Registry*. You can share two of these three partitions across multiple product instances in a typical production environment. Therefore, we identify four main deployment strategies for the three partitions as follows:

- All Partitions in a Single Server
- Config and Governance Partitions in a Remote Registry
- Governance Partition in a Remote Registry
- Config and Governance Partitions in Separate Nodes

In any of the above four sections, you can mount any WSO2 product to a remote WSO2 Governance Registry instance. Examples discussed here use JDBC-based configuration model as it is the recommended approach for a production setup.

**All Partitions in a Single Server**

**Strategy 1: Local Registry**

**Pattern #1 : Local Registry**

![Figure 1: All registry partitions in a single server instance.](image)

The entire registry space is local to a single server instance and not shared. This is recommended for a stand-alone deployment of a single product instance, but can also be used if there are two or more instances of a product that do not require sharing data or configuration among them.

This strategy requires no additional configuration.

**Config and Governance Partitions in a Remote Registry**

In this deployment strategy, the configuration and governance spaces are shared among instances of a group/cluster. For example, two WSO2 Application Server instances that have been configured to operate in a clustered environment can have a single configuration and governance registry that is shared across each node of the cluster. A separate instance of the WSO2 Governance Registry is used to provide the space used in common.
Figure 2: Config and governance partitions in the remote Governance Registry instance.

Configuration steps are given in the following sections.

- Creating the Database
- Configuring Governance Registry as the Remote Registry Instance
- Configuring Carbon Server Nodes

Creating the database

In a production setup, it is recommended to use an Oracle or MySQL database for the governance registry. As an example, we use MySQL database named 'registrydb'. Instructions are as follows:

1. Access MySQL using the command:

```
mysql -u root -p
```

2. Enter the password when prompted.

3. Create 'registrydb' database.

```
create database registrydb;
```

The MySQL database for G-Reg is now created.

Configuring Governance Registry as the remote registry instance

Database configurations are stored in $CARBON_HOME/repository/conf/datasources/master-datasources.xml file for all carbon servers. By default, all WSO2 products use the in-built H2 database. Since Governance Registry in this example is using a MySQL database named 'registrydb', the master-datasources.xml file needs to be configured so that the datasource used for the registry and user manager in Governance Registry is the said MySQL database.


2. Navigate to $G-REG_HOME/repository/conf/datasources/master-datasources.xml file where G-REG_HOME is the Governance Registry distribution home. Replace the existing WSO2_CARBON_DB datasource with the following configuration:
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://x.x.x.x:3306/registrydb</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>

Change the values of the following elements according to your environment.

- <url> : URL of the MySQL database.
- <username> and <password> : username and password of the mySQL database.
- <validationQuery> : Validate and test the health of the DB connection.
- <validationInterval> : specified time intervals at which the DB connection validations should run.

3. Navigate to $G-REG_HOME/repository/conf/axis2/axis2.xml file in all Carbon-based product instances to be connected with the remote registry, and enable tribes clustering with the following configuration.

```
<clustering class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
            enable="true"/>
```

The above configuration is required only when caching is enabled for the Carbon server instances and <enableCache> parameter is set to true. This provides cache invalidation at the event of any updates on the registry resources.


5. Start the Governance Registry server with -Dsetup so that all the required tables are created in the database. For example, in Linux

```
sh wso2server.sh -Dsetup
```

### Deprecation of -DSetup

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, from January 2018 onwards 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.
The Governance Registry server is now running with all required user manager and registry tables for the server also created in 'registrydb' database.

**Configuring server nodes**

Now that the shared registry is configured, let's take a look at the configuration of Carbon server nodes that use the shared, remote registry.

1. Download and extract the relevant WSO2 product distribution from the 'Products' menu of [https://wso2.com](https://wso2.com). In this example, we use two server instances (of any product) by the names CARBON-Node1 and CARBON-Node2.

2. We use the same datasource used for Governance Registry above as the registry space of Carbon-based product instances.

**Configuring master-datasources.xml file**

3. Configure `$CARBON_HOME/repository/conf/datasource/master-datasources.xml` where $CARBON_HOME is the distribution home of any WSO2 Carbon-based product you downloaded in step 1. Then, add the following datasource for the registry space.

```xml
<datasource>
  <name>WSO2_CARBON_DB_GREG</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB_GREG</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://x.x.x.x:3306/registrydb</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>
```

Change the values of the relevant elements accordingly.

**Configuring registry.xml file**

4. Navigate to `$CARBON_HOME/repository/conf/registry.xml` file and specify the following configurations for both server instances setup in step 1.

Add a new `db config` to the datasource configuration done in step 3 above. For example,

```xml
<dbConfig name="remote_registry">
  <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:
Define the registry partitions using the remote Governance Registry instance. In this deployment strategy, we are mounting the config and governance partitions of the Carbon-based product instances to the remote Governance Registry instance. This is graphically represented in Figure 2 at the beginning.

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

mount path: Registry collection of Carbon server instance that needs to be mounted

- `<mount path>`: Defines if an existing collection/resource at the given path should be overwritten or not. Possible values are:
  - true: The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  - false: The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
  - virtual: If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.

- `<targetPath>`: Path to the remote Governance Registry instance where the registry collection is mounted.

In each of the mounting configurations, we specify the actual mount path and target mount path. The targetPath can be any meaningful name.

**Configuring axis2.xml file**

1. Navigate to `$CARBON_HOME/repository/conf/axis2/axis2.xml` file where CARBON_HOME is the distribution home of any WSO2 Carbon-based products to be connected with the remote registry. Enable carbon clustering by copying the following configuration to all Carbon server instances:

```xml
<clustering
    class="org.apache.axis2.clustering.tribes.TribesClusteringAgent"
    enable="true"/>
```

**Note**

The above configuration is needed only when caching is enabled in the server instances and `<enableCache>` parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.

2. Copy ‘MySQL JDBC connector jar’ ([link](http://dev.mysql.com/downloads/connector/j/5.1.html)) to $G-REG_HOME/repository/components/lib in both Carbon server instances.
3. Start both servers and note the log entries that indicate successful mounting to the remote Governance Registry instance. For example,

```
[2012-12-14 14:02:33,369] INFO - CarbonCoreActivator Carbon Home : /home/gillian/Products/Mount/Node1
[2012-12-14 14:02:33,369] INFO - CarbonCoreActivator Java Temp Dir : /home/gillian/Products/Mount/Node1/tmp

[2012-12-14 14:02:33,383] INFO - AgentDS Successfully deployed Agent Client

[2012-12-14 14:02:35,145] INFO - EmbeddedRegistryService Configured Registry in 54ms
[2012-12-14 14:02:35,162] INFO - EmbeddedRegistryService Connected to mount at remote_registry in 4ms
```

4. Navigate to the registry browser in the Carbon server’s management console and note the config and governance partitions indicating successful mounting to the remote registry instance. For example,

```
Browse

Root /

Location: /

[Tree view] [Detail view]

system

config
governance

local
```

Governance Partition in a Remote Registry

In this deployment strategy, only the governance partition is shared among instances of a group/cluster. For example, a WSO2 Application Server instance and a WSO2 ESB instance that have been configured to operate in a clustered environment can have a single governance registry which is shared across each node of the cluster. A separate instance of the WSO2 Governance Registry (G-Reg) is used to provide the space used in common.

```
Figure 3: Governance partition in the remote Governance Registry instance.
```

Configuration steps are given in the following sections:

- Creating the Database
- Configuring Governance Registry Instance
- Configuring Carbon Server Nodes
Creating the database

In a production setup, it is recommended to use an Oracle or MySQL database for the governance registry. As an example, we use MySQL database named ‘registrydb’. Instructions are as follows:

1. Access MySQL using the command:

   ```
   mysql -u root -p
   ```

   Enter the password when prompted.

2. Create ‘registrydb’ database.

   ```
   create database registrydb;
   ```

   The MySQL database for G-Reg is now created.

Configuring Governance Registry instance

Database configurations are stored in `master-datasources.xml` file for all carbon servers. By default, all WSO2 products use the in-built H2 database. Since Governance Registry in this example is using a MySQL database named ‘registrydb’, the `master-datasources.xml` file needs to be configured so that the datasource used for the registry and user manager in Governance Registry is the said MySQL database.


2. Navigate to `$G-REG_HOME/repository/conf/datasources/master-datasources.xml` file where `$G-REG_HOME` is the Governance Registry distribution home. Replace the existing WSO2_CARBON_DB datasource with the following configuration:

   ```xml
   <datasource>
   <name>WSO2_CARBON_DB</name>
   <description>The datasource used for registry and user manager</description>
   <jndiConfig>
   <name>jdbc/WSO2CarbonDB</name>
   </jndiConfig>
   <definition type="RDBMS">
   <configuration>
   <url>jdbc:mysql://x.x.x.x:3306/registrydb</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>
   ```

   Change the values of the following elements according to your environment.

   - `<url>`: URL of the MySQL database.
   - `<username>` and `<password>`: username and password of the MySQL database.
   - `<validationQuery>`: Validate and test the health of the DB connection.
   - `<validationInterval>`: specified time intervals at which the DB connection validations should run.

3. Navigate to `$G-REG_HOME/repository/conf/axis2/axis2.xml` file in all Carbon-based product instances to be connected with the remote registry, and enable clustering with the following configuration.
<clustering
class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false"/>

The above configuration is required only when caching is enabled for the Carbon server instances and <enableCache> parameter is set to true. This provides cache invalidation at the event of any updates on the registry resources.

4. Copy the ‘mySQL JDBC connector jar’ to G-REG_HOME/repository/components/lib directory.

5. Start the Governance Registry server with -Dsetup so that all the required tables are created in the database. For example, in Linux

```sh
sh wso2server.sh -Dsetup
```

### Deprecation of -DSetup

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, from January 2018 onwards 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.

The Governance Registry server is now running with all required user manager and registry tables for the server also created in ‘registrydb’ database.

### Configuring server nodes

Now that the shared registry is configured, let's take a look at the configuration of Carbon server nodes that use the shared, remote registry.

1. Download and extract the relevant WSO2 product distribution from the ‘Products’ menu of https://wso2.com. In this example, we use two server instances (of any product) by the names CARBON-Node1 and CARBON-Node2 and the configuration is given for one server instance. Similar steps apply to the other server instance as well.

2. We use the same datasource used for Governance Registry above as the registry space of Carbon-based product instances.

#### Configure master-datasources.xml file

3. Configure $CARBON_HOME/repository/conf/datasource/master-datasources.xml where CARBON_HOME is the distribution home of any WSO2 Carbon-based product you downloaded in step 1. Then, add the following datasource for the registry space.
<datasource>
  <name>WSO2_CARBON_DB_GREG</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB_GREG</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://x.x.x.x:3306/registrydb</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>

Change the values of the relevant elements accordingly.

**Configuring registry.xml file**

4. Navigate to $CARBON_HOME/repository/conf/registry.xml file and specify the following configurations for both server instances setup in step 1.

Add a new db config to the datasource configuration done in step 3 above. For example,

```xml
<dbConfig name="remote_registry">
  <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:

```xml
<remoteInstance url="https://x.x.x.x:9443/registry">
  <id>instanceid</id>
  <dbConfig>remote_registry</dbConfig>
  <cacheId>root@https://x.x.x.x:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>
```

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

Define the registry partitions using the remote Governance Registry instance. In this deployment strategy, we are mounting the governance partition of the Carbon-based product instances to the remote Governance Registry instance. This is graphically represented in Figure 3 at the beginning.
**mount path** : Registry collection of Carbon server instance that needs to be mounted

**mount overwrite** : Defines if an existing collection/resource at the given path should be overwritten or not. Possible values are:
- true - The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
- false - The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
- virtual - If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.

**target path** : Path to the remote Governance Registry instance where the registry collection is mounted.

---

### Configuring axis2.xml file

5. Navigate to $CARBON_HOME/repository/conf/axis2/axis2.xml file where CARBON_HOME is the distribution home of any WSO2 Carbon-based products to be connected with the remote registry. Enable carbon clustering by copying the following configuration to all Carbon server instances:

```xml
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
    enable="false"/>
```

---

### Note

The above configuration is needed only when caching is enabled in the server instances and `<enableCache>` parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.


7. Start both servers and note the log entries that indicate successful mounting to the remote Governance Registry instance. Also navigate to the registry browser in the Carbon server’s management console and note the governance partition indicating successful mounting to the remote registry instance.

### Config and Governance Partitions in Separate Nodes

In this deployment strategy, let’s assume 2 clusters of Carbon-based product Foo and Carbon-based product Bar that share a governance registry space by the name G-Reg 1. In addition, the product Foo cluster shares a configuration registry space by the name G-Reg 2 and the product Bar cluster shares a configuration registry space by the name G-Reg 3.
Configuration steps are given in the following sections.

- Creating the Database
- Configuring the Remote Registry Instances
- Configuring Foo Product Cluster
- Configuring Bar Product Cluster

Creating the database

In a production setup, it is recommended to use an Oracle or MySQL database for the governance registry. As an example, we use MySQL database named 'registrydb'. Instructions are as follows:

1. Access MySQL using the command:

```
mysql -u root -p
```

2. Enter the password when prompted.
3. Create 'registrydb' database.

```
create database registrydb;
```

The MySQL database for G-Reg 1 is now created. Similarly create 'registrydb2' and 'registrydb3' as the MySQL databases for G-Reg 2 and G-Reg 3 respectively.

Configuring the Remote Registry instances

Database configurations are stored in $CARBON_HOME/repository/conf/datasources/master-datasources.xml file for all carbon servers. By default, all WSO2 products use the in-built H2 database. Since the Governance Registry nodes (G-Reg 1, G-Reg 2 and G-Reg 3) in this example are using MySQL databases ('registrydb', 'registrydb2' and 'registrydb3' respectively) the master-datasources.xml file of each node needs to be configured so that the datasources used for the registry, user manager and configuration partitions in Governance Registry are the said MySQL databases.

2. First, navigate to $G-REG_HOME/repository/conf/datasources/master-datasources.xml file where G-REG_HOME is the distribution home of Governance Registry of G-Reg 1. Replace the existing WSO2_CARBON_DB datasource with the following configuration:
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://10.20.30.41:3306/registrydb</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>

Change the values of the following elements according to your environment.

- `<url>`: URL of the MySQL database.
- `<username>` and `<password>`: username and password of the MySQL database.
- `<validationQuery>`: Validate and test the health of the DB connection.
- `<validationInterval>`: specified time intervals at which the DB connection validations should run.

3. Similarly, replace the existing WSO2_CARBON_DB datasource in G-Reg 2 with the following:

<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</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>

4. Repeat the same for G-Reg 3 as follows.
5. Navigate to $G-REG_HOME/repository/conf/axis2/axis2.xml file in all instances and enable clustering with the following configuration.

```xml
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
    enable="false"/>
```

The above configuration is required only when caching is enabled for the Carbon server instances and <enableCache> parameter is set to true. This provides cache invalidation at the event of any updates on the registry resources.


7. Start the Governance Registry servers with -Dsetup so that all the required tables will be created in the databases. For example, in Linux

```
sh wso2server.sh -Dsetup
```

---

**Deprecation of -DSetup**

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, from January 2018 onwards 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.

The Governance Registry server instances are now running with all required user manager and registry tables for the server created in 'registrydb', 'registrydb1' and 'registrydb2' databases.

**Configuring the foo product cluster**

Now that the shared registry nodes are configured, let's take a look at the configuration of Carbon server clusters that share the remote registry instances.
Namely, Foo product cluster shares G-Reg 1 and G-Reg 2 while Bar product cluster shares G-Reg 1 and G-Reg 3.

Include the following configurations in the master node of Foo product cluster.

**Configuring master-datasources.xml file**

1. Configure `${CARBON_HOME}/repository/conf/datasource/master-datasources.xml` where CARBON_HOME is the distribution home of any WSO2 Carbon-based product. Then, add the following datasource for the registry space.

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB_GREG</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB_GREG</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:mysql://10.20.30.41:3306/registrydb</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>
   
   <datasource>
     <name>WSO2_CARBON_DB_GREG_CONFIG</name>
     <description>The datasource used for configuration partition</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB_GREG_CONFIG</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>

Change the values of the relevant elements according to your environment.

**Configuring registry.xml file**

2. Navigate to `${CARBON_HOME}/repository/conf/registry.xml` and specify the following configurations.

Add a new db config to the datasource configuration done in step 1 above. For example,
Specify the remote Governance Registry instance with the following configuration:

```xml
<remoteInstance url="https://10.20.30.41:9443/registry">
  <id>governanceRegistryInstance</id>
  <dbConfig>governance_registry</dbConfig>
  <cacheId>root@https://10.20.30.41:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>

<remoteInstance url="https://10.20.30.42:9443/registry">
  <id>configRegistryInstance</id>
  <dbConfig>config_registry</dbConfig>
  <cacheId>root@https://10.20.30.42:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>
```

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

**Note**

When adding the corresponding configuration to the registry.xml file of a slave node, set `<readOnly>true</readOnly>`. This is the only configuration change.

Define the registry partitions using the remote Governance Registry instance.

```xml
<mount path="/._system/config" overwrite="true">
  <instanceId>configRegistryInstance</instanceId>
  <targetPath>/._system/config</targetPath>
</mount>

<mount path="/._system/governance" overwrite="true">
  <instanceId>governanceRegistryInstance</instanceId>
  <targetPath>/._system/governance</targetPath>
</mount>
```

- `<mount path>`: Registry collection of Carbon server instance that needs to be mounted
- `<mount overwrite>`: Defines if an existing collection/resource at the given path should be overwritten or not. Possible values are:
  - `true` - The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  - `false` - The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
- virtual - If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.
- target path: Path to the remote Governance Registry instance where the registry collection is mounted.

**Configuring axis2.xml file**

3. Navigate to `$CARBON_HOME/repository/conf/axis2/axis2.xml` file and enable carbon clustering by copying the following configuration to all Carbon server instances:

```xml
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
    enable="false"/>
```

**Note**

The above configuration is needed only when caching is enabled in the server instances and `<enableCache>` parameter set to true. Clustering enables cache invalidation in configured nodes at the event of any changes to the registry resources by any of the Carbon server nodes in the deployment setup.


**Configuring the bar product cluster**

The instructions here are similar to that of the Foo product cluster discussed above. The difference is that Bar product cluster shares G-Reg 1 (Governance space) and G-Reg 3 (Config space) remote registry spaces whereas Foo product cluster shares G-Reg 1 and G-Reg 2 (Config space).

Include the following configurations in the master node of Foo product cluster.

**Configure master-datasources.xml file**

1. Configure `$CARBON_HOME/repository/conf/datasource/master-datasources.xml` where `$CARBON_HOME` is the distribution home of any WSO2 Carbon-based product. Then, add the following datasource for the registry space.

```xml
<datasource type="mysql">
    <url>jdbc:mysql://localhost:3306/registry</url>
    <user>registry</user>
    <password>registry</password>
</datasource>
```
Change the values of the relevant elements according to your environment.

**Configuring registry.xml file**

2. Navigate to `$CARBON_HOME/repository/conf/registry.xml` file and specify the following configurations.

Add a new `db config` to the `datasource` configuration done in step 1 above. For example,

```xml
<dbConfig name="governance_registry">
  <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
<dbConfig name="config_registry">
  <dataSource>jdbc/WSO2CarbonDB_GREG_CONFIG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:
<remoteInstance url="https://10.20.30.41:9443/registry">
  <id>governanceRegistryInstance</id>
  <dbConfig>governance_registry</dbConfig>
  <cacheId>root@https://10.20.30.41:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>

<remoteInstance url="https://10.20.30.43:9443/registry">
  <id>configRegistryInstance</id>
  <dbConfig>config_registry</dbConfig>
  <cacheId>root@https://10.20.30.43:9443/registry</cacheId>
  <readOnly>false</readOnly>
  <enableCache>true</enableCache>
  <registryRoot>/</registryRoot>
</remoteInstance>

Change the values of the following elements according to your environment.

- `<remoteInstance url>`: URL of the remote G-Reg instance.
- `<dbConfig>`: The dbConfig name specified for the registry database configuration.
- `<cacheId>`: This provides information on where the cache resource resides.
- `<enableCache>`: Whether caching is enabled on the Carbon server instance.

Define the registry partitions using the remote Governance Registry instance.

```xml
<mount path="/system/config" overwrite="true">
  <instanceId>configRegistryInstance</instanceId>
  <targetPath>/system/config</targetPath>
</mount>

<mount path="/system/governance" overwrite="true">
  <instanceId>governanceRegistryInstance</instanceId>
  <targetPath>/system/governance</targetPath>
</mount>
```

- `mount path`: Registry collection of Carbon server instance that needs to be mounted
- `mount overwrite`: Defines if an existing collection/resource at the given path should be overwritten or not. Possible values are:
  - `true`: The existing collection/resource in the specified location will always be deleted and overwritten with the resource/s in the remote registry instance.
  - `false`: The resource/s will not be overwritten. An error will be logged if a resource exists at the existing location.
  - `virtual`: If the existing location has a resource/collection, it will be preserved but virtual view of the remote registry resource/s can be viewed. The original resource/collection can be viewed once the remote registry configuration is removed.
- `target path`: Path to the remote Governance Registry instance where the registry collection is mounted. In each of the mounting configurations, we specify the actual mount path and target mount path. The `targetPath` can be any meaningful name.

### Configuring axis2.xml file

3. Navigate to `$CARBON_HOME/repository/conf/axis2/axis2.xml` file and enable carbon clustering by copying the following configuration to all Carbon server instances:

5. Start both clusters and note the log entries that indicate successful mounting to the remote Governance Registry nodes.

6. Navigate to the registry browser in the Carbon server's management console of a selected node and note the config and governance partitions indicating successful mounting to the remote registry instances.

**Working with Users, Roles and Permissions**

The user management functionality allows you to configure the users that can access your product and the permissions that determine how each user can work with your system.

The default user management configuration in a WSO2 product is as follows:

- The default H2 database in the WSO2 product is configured as the User Store that stores all the information on users, roles and permissions.
- An Admin user and Admin password are configured by default.
- The default Admin role connected to the Admin user has all permissions granted.

According to the default configuration explained above, you can simply log into the management console of the product with the Admin user and get started right away.

Follow the links given below to understand how user management works in WSO2 products, and for step-by-step instructions on how to change/update the default configuration:

- Introduction to User Management
- Configuring the User Realm

**Introduction to User Management**

User management involves defining and managing users, roles, and their access levels in a system. A user management dashboard or console provides system administrators with a high-level view of a system's active user sessions, their login statuses, the privileges of each user, and their activity in the system. It enables system admins to make business-critical, real-time security decisions. A typical user management implementation involves a wide range of functionality such as adding/deleting users, controlling user activity through permissions, managing user roles, defining authentication policies, managing external user stores and manual/automatic logout, and resetting passwords.

Any user management system has the following basic components:

- **Users**: Users are consumers who interact with your organizational applications, databases, and other systems. A user can be a person, a device, or another application/program within or outside of the organization's network. Because users interact with internal systems and access data, organizations need to define which data and functionality each user can access by assigning permissions.

- **Permissions**: A permission is a delegation of authority or a right that is assigned to a user or a group of users to perform an action on a system. Permissions can be granted to or revoked from a user, user group, or user role automatically or by a system administrator. For example, if a user has the permission to log in to a system, the permission to log out is automatically granted as well.

- **User roles**: A user role is a grouping of permissions. In addition to assigning individual permissions to users, admins can create user roles and assign those roles to users. For example, you might create user roles called VP, Manager, and Employee, each of which has a different set of permissions, and then assign those roles to users based on their position in the company. Then, if you need to modify the permissions of all your managers, you can simply modify the Manager user role, and all the users with that role will have their permissions updated automatically.

The following diagram illustrates how the user management functionality is structured to work in WSO2 products:

- **User stores**: A user store is the database where information about the users and user roles is stored, including login name, password, first name, last name, and email address.

- **RDBMS (for Authentication and Authorization)**: This RDBMS stores information about the role-based permissions.

According to the default configuration in WSO2 products, the embedded H2 RDBMS that is shipped with the product is used as the
**Realm configuration:** The user realm consists of the configurations required to initialise the user realm. The `user-mgt.xml` file stored in the `<PRODUCT_HOME>/repository/conf/` directory is used as the realm configuration XML. This includes setting up the User Store Manager, the Authorization Manager and the System Administrator. These configurations are explained below.

### User Store Manager

The User Store Manager is responsible for managing the underlying user store. It is represented by the `UserStoreManager` Java interface. There can be different User Store Manager implementations to connect with different user stores, but you can configure only one User Store Manager implementation in a single user realm (that is, a single WSO2 Carbon instance). The User Store Manager can be operated in both read/write mode and read-only mode. In read-only mode, you can only connect with an existing user store. WSO2 products provide the following default User Store Manager implementations:

- `JDBCUserStoreManager` (read and write)
- `LDAPUserStoreManager` (read-only)
- `ApacheDSUserStoreManager` (read and write)

You can write a custom user store manager implementation by implementing `UserStoreManager` or by extending `AbstractUserStoreManager` or one of the default implementations.

#### Using `JDBCUserStoreManager`

The `JDBCUserStoreManager` class uses a schema that is specific to WSO2 Carbon. It contains the following tables:

- **UM_USER:** Contains user names and passwords
- **UM_ROLE:** Contains role names
- **UM_USER_ROLE:** Contains user role mappings
- **UM_USER_ATTRIBUTE:** Contains user attributes. There can be any attribute ID and a value for that attribute ID that is associated with a user’s profile.

You can find the full schema of these tables from the database script files in the `<PRODUCT_HOME>/dbscripts` directory. Note that these scripts also contain schemas for other tables that are used for user management and registry functions. If your organization contains an existing JDBC user store that you want to use with a WSO2 product, you must extend `JDBCUserStoreManager` and write a new implementation for your user store according to your schema.

### Authorization Manager

The Authorization Manager uses role-based access control (RBAC) to protect resources related to the WSO2 Carbon platform. The default implementation of the Authorization Manager is `JDBCAuthorizationManager`, which uses a permission model specific to WSO2 Carbon and uses the authorization data that is stored in tables in the JDBC database. You can replace this implementation with a custom implementation (for example, if you want to use a XACML authorization manager) and use it with WSO2 products.

### System Administrator

The system admin user is typically the super tenant user, who by default has permission to perform all administration tasks in the server. The admin user will thereby create other tenant users and define roles with permissions. Once this is done, the other tenant users will be able to log in to their respective tenant domains and use the server according to the permissions that have been granted. Note that the permissions granted to the Super Tenant user cannot be modified.

For information on how you can set up and configure the user management realm, see [Configuring the User Realm](#), and for information on how you can manage the users, roles and permissions using the Management Console, see [Managing Users, Roles and Permissions](#).
Configuring the User Realm

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. The following documentation explains the configurations that should be done in WSO2 products in order to set up the User Management module.

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:

The following sections include instructions on the above required configurations and repositories:

- Configuring the System Administrator
- Configuring the Authorization Manager
- Configuring User Stores

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.

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.

Before you begin:

Ensure that you have a primary user store (for storing users and roles) and an RDBMS (for storing information related to permissions). See the following documentation for instructions on how to set up these repositories.

- Configuring the Primary User Stores: This topic explains how the primary user store is set up and configured for your product.
## 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 the other repository.

### Configuration Example

```xml
<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>This is the role that has all administrative privileges of the WSO2 product, so all users having this role are admins of the product. You can provide any meaningful name for this role. This role is created in the internal H2 database when the product starts. This role has permission to carry out any actions related to the Management Console. If the user store is read-only, this role is added to the system as a special internal role where users are from an external user store.</td>
</tr>
<tr>
<td>&lt;AdminUser&gt;</td>
<td>Configures the default administrator for the WSO2 product. If 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 &lt;AdminUser&gt; element. If the external user store is in read/write mode, and you set &lt;AddAdmin&gt; 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>Do NOT put the password here but leave the default value. If the user store is read-only, this element and its value are ignored. This password is used only if the user store is read-write and the AddAdmin value is set to true.</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>

### Note

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 as explained [here](#).
Permissions are stored in a separate repository. A user store can be a typical RDBMS, an LDAP or an external Active Directory. For information on how the repositories for storing information about users and roles are configured, see Configuring User Stores.

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 instructions given below to set up and configure the Authorization Manager.

- Step 1: Setting up the repository
- Step 2: Updating the user realm configurations

**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.

For information on how you can set up a new RDBMS and configure it for your system, see Setting up the Physical Database, and for information on the purpose of defining datasources and how they are configured for a product, see Managing 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.

**Setting up the database connection**

Update the datasource information using the <Property> element under <Configuration>. Given below are the properties that are set by default.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataSource</td>
<td>This is the jndi name of the datasource that is used for referring to the datasource. In the following example, the jndi name of the default datasource defined in the &lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml file is linked from the user-mgt.xml file.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>isCascadeDeleteEnabled</td>
<td>This property is set to 'true' by default, which enables cascade delete for the UM_USER_PERMISSION and UM_ROLE_PERMISSION tables when a permission gets deleted from the UM_PERMISSION table. That is, if a record in the parent table is deleted the corresponding records in the child table will be automatically deleted.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

You can add more optional 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 validationQuery parameter in the &lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml 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>

**Configuring the Authorization Manager**

Shown below is how the Authorization Manager is enabled in the user-mgt.xml file.
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:

```xml
<Property name="CaseSensitiveAuthorizationRules">true</Property>
```

### Configuring User Stores

The user management feature in WSO2 products allows you to maintain multiple user stores for your system that are used to store the users and their roles. You can set up any of the following types of user stores:

- JDBC user stores
- Active Directory user stores
- Read-Only LDAP user stores
- Read-Write LDAP user stores

The **Primary User Store** in every WSO2 product is configured in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file. By default, the embedded H2 database (JDBC) that is shipped with WSO2 products is configured as the primary user store, except for WSO2 Identity Server, which has an embedded LDAP as its primary user store. You can change the default configuration by replacing the default database according to your requirement. The primary user store is shared among all the tenants in the system.

With the user management feature, any number of **Secondary User Stores** can be easily set up for your system using the management console. This will automatically create an XML file with the configurations corresponding to the secondary user store. Alternatively, you can manually create the configuration file and store it in this directory without using the management console. **Secondary User Stores** are specific to the created tenant, and they are not shared between multiple tenants.

Although, information about users and roles are stored in the repositories that we call User Stores, which can be of any of the types described above, the permissions attached to roles are always stored in an RDBMS. According to the default configuration in WSO2 products, the embedded H2 database is used for storing permissions as well as users and roles. The instructions in this section explain how you can change the default user store. For information on how to set up an RDBMS repository for storing permission, see Configuring the Authorization Manager.

The following topics include instructions on setting up user stores:

- Configuring the Primary User Store
- Configuring Secondary User Stores
- Working with Properties of User Stores
- Writing a Custom User Store Manager
Configuring the Primary User Store

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. This documentation explains the process of setting up a primary user store for your system.

### The default User Store

The primary user store that is configured by default in every WSO2 product is a JDBC user store, which reads/writes into the internal database of the product server. By default, the internal database is H2 (except for WSO2 IS, which uses an LDAP as the default user store). This database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles).

Instead of using the embedded database, you can set up a separate repository and configure it as your 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, for connecting to LDAP, Active Directory and JDBC. Thereby, these adapters enable you to authenticate users from different types of user stores.

<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 <code>ReadOnlyLDAPUserStoreManager</code> to do read-only operations for external LDAP user stores.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</td>
<td>Use <code>ReadWriteLDAPUserStoreManager</code> for external LDAP user stores to do both read and write operations. This is the user store configuration which is uncommented in the code in the <code>user-mgt.xml</code> file for WSO2 Identity Server.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager</td>
<td>Use <code>ActiveDirectoryUserStoreManager</code> to configure an Active Directory Domain 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 <code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code>.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</td>
<td>Use <code>JDBCUserStoreManager</code> for both internal and external JDBC user stores. This is the user store configuration which is uncommented in the code in the <code>user-mgt.xml</code> file for all WSO2 products, except WSO2 Identity Server (which uses the <code>ReadWriteLDAPUserStoreManager</code>).</td>
</tr>
</tbody>
</table>

The `user-mgt.xml` file already has sample configurations for all of the above user stores. To enable the required user store configuration, you must uncomment them in the code and comment out the ones that you do not need as explained in the following topics.

- 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 a JDBC User Store

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. This file is shipped with user store manager configurations for all possible user store types (JDBC, read-only LDAP/Active Directory, read-write LDAP and read-write Active directory). The instructions given below explains how to configure an RDBMS (JDBC) as the primary user store for the WSO2 server.

### The default User Store

The internal H2 database that is shipped with every WSO2 product (except WSO2 Identity Server) is configured as the default primary user store. This internal database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles). In the case of the WSO2 Identity Server, the default user store is an LDAP (Apache DS) that is shipped with the product.

When you configure a JDBC user store as the primary user store, you can either use the default configuration or you can change it in the following ways:

- You can set up two separate databases for the Authorization Manager and the User Store Manager.
- It is not recommended to use the default H2 database in production. Therefore, you can replace this. For instructions on replacing this by setting up a new RDBMS and configuring it for your system, see Setting Up the Physical Database.

Therefore, before you begin, ensure that the RDBMS that you want to use as the JDBC user store is correctly set up for your system. Then, follow the steps given below to configure a JDBC user store as the primary user store in your product.
Step 1: Configuring the JDBC user store manager

**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 JDBC is `<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">`.

To configure a JDBC user store as the primary user store, you must change the `JDBCUserStoreManager` section in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

1. Uncomment the following section:

   ```xml
   <UserStoreManager
       class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
   ```

2. Specify the connection to the RDBMS inside the JDBC user store manager according to your requirement. For more information on user store properties in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file which are used for configuring the primary user store, see Properties of Primary User Stores.

### Internal JDBC User Store

The following sample shows how to configure the internal RDBMS as the 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="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="UserNameUniqueAcrossTenants">false</Property>
     <Property name="PasswordJavaRegEx">[\S]{5,30}$</Property>
     <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
     <Property name="UsernameJavaRegEx">[a-zA-Z0-9._-|/]{3,30}$</Property>
     <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
     <Property name="RolenameJavaRegEx">[a-zA-Z0-9._-|/]{3,30}$</Property>
     <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
     <Property name="UserRolesCacheEnabled">true</Property>
 </UserStoreManager>
```

### External JDBC User Store

The following sample shows how to configure an external RDBMS as the JDBC user store:

```xml
<UserStoreManager
    class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property
        name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    </UserStoreManager>
```
<Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>

<Property name="driverName">com.mysql.jdbc.Driver</Property>

<Property name="url">jdbc:mysql://localhost:3306/tcsdev</Property>

<Property name="userName"></Property>

<Property name="password"></Property>

<Property name="Disabled">false</Property>

<Property name="MaxUserNameListLength">100</Property>

<Property name="MaxRoleNameListLength">100</Property>

<Property name="UserRoleCacheEnabled">true</Property>

<Property name="PasswordDigest">SHA-256</Property>

<Property name="ReadGroups">true</Property>

<Property name="IsEmailUserName">false</Property>

<Property name="DomainCalculation">default</Property>

<Property name="StoreSaltedPassword">true</Property>

<Property name="UserNameUniqueAcrossTenants">false</Property>

<Property name="PasswordJavaRegEx">^[\S]{5,30}$</Property>

<Property name="PasswordJavaScriptRegEx">^[\S]{5,30}$</Property>

<Property name="UsernameJavaRegEx">^[a-zA-Z0-9._\-|/]{3,30}$</Property>

<Property name="UsernameJavaScriptRegEx">^[\S]{5,30}$</Property>

<Property name="RolenameJavaRegEx">^[a-zA-Z0-9._\-|/]{3,30}$</Property>

<Property name="RolenameJavaScriptRegEx">^[\S]{5,30}$</Property>

<Property name="SCIMEnabled">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 UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_TENANT_ID=? AND UM_SHARED_ROLE = '0' ORDER BY UM_ROLE_NAME</Property>

<Property name="GetSharedRoleListSQL">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>

<Property name="UserFilterSQL">SELECT UM_USER_NAME FROM UM_USER WHERE UM_USER_NAME LIKE ? AND UM_TENANT_ID=? ORDER BY UM_USER_NAME</Property>


<Property name="UserSharedRoleSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE UM_USER.UM_USER_NAME = ? AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = ?</Property>

<Property name="UserRoleSQL">SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>


<Property name="GetUserListOfSharedRoleSQL">SELECT UM_USER_NAME FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE UM_USER.UM_USER_NAME = ? AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = ?</Property>

<Property name="GetUserPropertiesForProfileSQL">SELECT UM_ATTR_NAME, UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE
<Property name="GetUserPropertyForProfileSQL">SELECT
UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID =
UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER.UM_USER_NAME=?
AND UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=?
AND UM_USER.UM_TENANT_ID=?</Property>

<Property name="GetUserIsForPropertySQL">SELECT UM_USER_NAME FROM
UM_USER, UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID =
UM_USER.UM_ID AND UM_USER_ATTRIBUTE.UM_ATTR_NAME=? AND
UM_USER_ATTRIBUTE.UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=?
AND UM_USER.UM_TENANT_ID=?</Property>

<Property name="GetUserLisForPropertySQL">SELECT UM_USER_NAME FROM
UM_USER, UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID =
UM_USER.UM_ID AND UM_USER_ATTRIBUTE.UM_ATTR_NAME=? AND
UM_USER_ATTRIBUTE.UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=?
AND UM_USER.UM_TENANT_ID=?</Property>

<Property name="GetProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=?</Property>

<Property name="GetUserProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND
UM_TENANT_ID=?</Property>

<Property name="GetUserIDFromUserNameSQL">SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>

<Property name="GetUserNameFromTenantIDSQL">SELECT UM_USER_NAME FROM UM_USER WHERE UM_TENANT_ID=?</Property>

<Property name="GetTenantIDFromUserNameSQL">SELECT UM_TENANT_ID FROM UM_USER WHERE UM_USER_NAME=?</Property>

<Property name="AddUserSQL">INSERT INTO UM_USER (UM_USER_NAME,
UM_USER_PASSWORD, UM_SALT_VALUE, UM_REQUIRE_CHANGE, UM_CHANGED_TIME,
UM_TENANT_ID) VALUES (?, ?, ?, ?, ?, ?)</Property>

<Property name="AddUserToRoleSQL">INSERT INTO UM_USER_ROLE
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) VALUES ((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="AddRoleSQL">INSERT INTO UM_ROLE (UM_ROLE_NAME,
UM_TENANT_ID) VALUES (?, ?)</Property>

<Property name="AddSharedRoleSQL">UPDATE UM_ROLE SET
UM_SHARED_ROLE = ? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID = ?</Property>

<Property name="AddRoleToUserRoleSQL">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>

<Property name="AddSharedRoleToUserRoleSQL">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_TENANT_ID=? AND
UM_ROLE_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID FROM
UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND
UM_USER_TENANT_ID=? AND UM_ROL_TENANT_ID = ?</Property>

<Property name="RemoveRoleFromUserSQL">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=?)) 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_USER_ID=(SELECT UM_ID FROM
UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND
UM_USER_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="OnDeleteUserRemoveUserAttributeSQL">DELETE FROM
<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="AddUserToRoleSQL-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="AddRoleToUserSQL-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>


<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="AddUserToRoleSQL-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="AddRoleToUserSQL-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>


<Property name="DomainName">wso2.org</Property>
<Property name="Description"/>
</UserStoreManager>

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 in order to streamline the configurations with WSO2 products.

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](http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html).

4. If you are setting up an external JDBC user store, you need to set the following property to 'true' to be able to create roles in the primary user store.

```xml
<Property name="WriteGroups">false</Property>
```

**Step 2: Updating 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 user store when you start the system for the first time. If the JDBC user store is read-only, then we need to always use a user ID that is already in the user store as the super tenant. Otherwise, if the JDBC user store can be written to, you have the option of creating a new admin user in the user store when you start the system for the first time. For information on configuring the system administrator user, see **Configuring the System Administrator**.

These two alternative configurations can be done as explained below.

- If the user store is read-only, find a valid user that already resides in the RDBMS. For example, say a valid username is AdminSOA. Update the `<AdminUser>` section of your configuration as shown below. You do not have to update the password element as it is already set in the user store.

```xml
<AddAdmin>False</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
  <UserName>AdminSOA</UserName>
  <Password>XXXXXX</Password>
</AdminUser>
```

- If the user store can be written to, you can add the super tenant user to the user store. Therefore, `<AddAdmin>` should be set to **true** as shown below.
Step 3: Updating the datasources

Whenever there is an RDBMS set up for your system, it is necessary to create a corresponding datasource, which allows the system to connect to the database. The datasource for the internal H2 database that is shipped with WSO2 products by default, is configured in the master-datasources.xml file, which is stored in the <PRODUCT_HOME>/repository/conf/datasources/ directory. For detailed information on setting up databases, see Setting Up the Physical Database, and for information on the purpose of defining datasources and how they are configured for a product, see Managing Datasources.

1. There are two possible methods for updating datasources:

   a. Shown below is how the master-datasources.xml file is configured to connect to the default H2 database in your system. If you have replaced the default database with a new RDBMS, which you are now using as the JDBC users store, you have to update the master-datasources.xml file with the relevant information.

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000</url>
         <username>wso2carbon</username>
         <password>wso2carbon</password>
         <driverClassName>org.h2.Driver</driverClassName>
         <maxActive>50</maxActive>
         <maxWait>60000</maxWait>
         <testOnBorrow>true</testOnBorrow>
         <validationQuery>SELECT 1</validationQuery>
         <validationInterval>30000</validationInterval>
       </configuration>
     </definition>
   </datasource>
   ```

   b. Alternatively, instead of using the master-datasources.xml file, you can also 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.

   2. Now, the datasource configuration and the user store manager configuration in the user-mgt.xml file should be linked together. You can do this by referring to the datasource information (typically defined in the master-datasources.xml file) from the user-mgt.xml file as explained below.
The RDBMS that is used for storing Authorization information is configured under the `<Configuration>` section in the `user-mgt.xml` file, by adding `<Property name="dataSource">` as shown below. The following example refers to the default H2 database.

```
<Configuration>
    .......
    <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
</Configuration>
```

If you are using the same RDBMS as the user store in your system, this datasource reference will suffice. However, if you have set up a separate RDBMS as the user store, instead of using a common RDBMS for Authorization information as well as the user store, you must refer to the datasource configuration from within the User Store Manager configuration in the `user-mgt.xml` file by adding the `<Property name="dataSource">` property.

Step 4: Starting the server

1. Add the JDBC driver to the classpath by copying its JAR file into the `<PRODUCT_HOME>/repository/components/lib` directory.
2. Start the server.

**Configuring a Read-Only LDAP User Store**

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. This file is shipped with user store manager configurations for all possible user store types (JDBC, read-only LDAP/Active Directory, read-write LDAP and read-write Active directory). The instructions given below explains how to configure a read-only LDAP or Active Directory as the primary user store for the WSO2 server.

### The default User Store

The primary user store that is configured by default in the `user-mgt.xml` file is a JDBC user store, which reads/writes into the internal database of the product server. By default, the internal database is H2 for all WSO2 products excluding the Identity Server. This database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles). In the case of the WSO2 Identity Server, the default user store is an LDAP (Apache DS) that is shipped with the product.

Note that the RDBMS used in the default configuration can remain as the database used for storing Authorization information.

Follow the given steps to configure a read-only LDAP/AD as the primary user store:

- Step 1: Setting up the read-only LDAP/AD user store manager
- Step 2: Updating the system administrator
- Step 3: Starting the server

Step 1: Setting up the read-only LDAP/AD user store manager

#### 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/Active Directory is `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">`.

1. Uncomment the following user store in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file:
   `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">`. Also, ensure that you comment out the configurations for any other user stores in the same file.
2. Given below is a sample for the LDAP/AD user store configuration in read-only mode. You can change the values to match your LDAP/AD. For descriptions on each of the properties used in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file which are used for configuring the primary user store, see Properties of User Stores.
a. Update the connection details to match your user store. For example:

```
<Property name="ConnectionURL">ldap://localhost:10389</Property>
```

For Active Directory, the `ConnectionURL` should have the following format:
b. 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 user store configuration as shown below. Note that this user does NOT have to be the system administrator that you define here.

\[
\begin{align*}
\text{<Property name="ConnectionURL">lldaps://10.100.1.100:636</Property>}
\end{align*}
\]

Set the attribute to use as the username, typically either \textit{cn} or \textit{uid} for LDAP. Ideally, \text{<Property name="UserNameAttribute">} and \text{<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:

\[
\begin{align*}
\text{<Property name="UserNameAttribute">uid</Property>}
\end{align*}
\]

c. Update \text{<Property name="UserSearchBase">} with the directory name where the users are stored. When LDAP searches for users, it will start from this location of the directory.

\[
\begin{align*}
\text{<Property name="UserSearchBase">ou=system</Property>}
\end{align*}
\]

d. Set the attribute to use as the username, typically either \textit{cn} or \textit{uid} for LDAP. Ideally, \text{<Property name="UserNameAttribute">} and \text{<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:

\[
\begin{align*}
\text{<Property name="UserNameAttribute">uid</Property>}
\end{align*}
\]

e. Set the \text{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 \text{GroupSearchBase}, \text{GroupSearchFilter} and \text{NameAttribute} properties. If the \text{ReadGroups} property is set to 'false', only Users can be read from the user store. You can set the configuration to read roles from the user store by reading the user/role mapping based on a membership (user list) or backlink attribute as shown below.

To read the user/role mapping based on a membership (This is used by the ApacheDirectory server and OpenLDAP):

- Enable the \text{ReadGroups} property.
Step 2: Updating 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. The <Configuration> section in the user-mgt.xml file contains the super admin information. Update this configuration for the read-only LDAP/AD as explained below.

f. 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.

g. In WSO2 products based on Carbon 4.4.x, you can set the LDAPConnectionTimeout property: If the connection to the LDAP is inactive for the length of time (in milliseconds) specified by this property, the connection will be terminated.
<Configuration>
  <AddAdmin>False</AddAdmin>
  <AdminRole>admin</AdminRole>
  <AdminUser>
    <UserName>AdminSOA</UserName>
    <Password>XXXXXX</Password>
  </AdminUser>
  ............
</Configuration>

- <AddAdmin>: This should be set to 'False' as it will not be allowed to create users and roles in a read-only user store.
- <AdminRole>: The admin role you enter here should already exist in the read-only user store. Otherwise, you must enter an internal role, which will be saved to the internal database of the system when the system starts the first time.
- <AdminUser>: Since we are configuring a read-only LDAP as the primary user store, the user that should have admin permissions is required to be stored in the user store when you start the system for the first time. For example, say a valid username is AdminSOA. Update the <AdminUser> section of your configuration as shown above. You do not have to update the password element as it is already set in the user store.

For information about the system administrator user, see Configuring the System Administrator, and for information on how keystores are used in WSO2 products, see Using Asymmetric Encryption.

Configuring a Read-Write Active Directory User Store

The default User Store

The primary user store that is configured by default in the user-mgt.xml file is a JDBC user store, which reads/writes into the internal database of the product server. By default, the internal database is H2 for all WSO2 products excluding WSO2 Identity Server. This database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles). In the case of the WSO2 Identity Server, the default user store is an LDAP (Apache DS) that is shipped with the product.

Note that the RDBMS used in the default configuration can remain as the database used for storing Authorization information.

Follow the given steps to configure an external Active Directory as the primary user store:

- Step 1: Setting up the external AD user store manager
- Step 2: Updating the system administrator
- Step 3: Starting the server

Step 1: Setting up the external AD user store manager

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 an external AD is UserStoreManager class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager".

1. Enable the ActiveDirectoryUserStoreManager class 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 Active Directory 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 given below. Change the values according to your requirement. For more information on each of the properties used in the user-mgt.xml file for configuring the primary user store, see Properties of User Stores.
<UserStoreManager class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
    <Property name="TenantManager" org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager />
    <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">A1b2c3d4</Property>
    <Property name="AnonymousBind">false</Property>
    <Property name="UserSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
    <Property name="UserNameAttribute">cn</Property>
    <Property name="UserNameSearchFilter">(&amp;(objectClass=user)(cn=?))</Property>
    <Property name="UserNameListFilter">(objectClass=user)</Property>
    <Property name="GroupNameAttribute">cn</Property>
    <Property name="GroupNameSearchFilter">(&amp;(objectClass=group)(cn=?))</Property>
    <Property name="GroupNameListFilter">(object category=group)</Property>
    <Property name="UsernameJavaRegEx">^[a-zA-Z0-9-_.\-|/]{3,30}$</Property>
    <Property name="UsernameJavaScriptRegEx">^[\S]{3,30}$</Property>
    <Property name="PasswordJavaRegEx">^[\S]{5,30}$</Property>
    <Property name="PasswordJavaScriptRegEx">^[\S]{5,30}$</Property>
    <Property name="RolenameJavaRegEx">^[a-zA-Z0-9-_.\-|/]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">^[\S]{3,30}$</Property>
    <Property name="SCIMEnabled">false</Property>
    <Property name="IsBulkImportSupported">true</Property>
    <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
    <Property name="MultiAttributeSeparator">,</Property>
    <Property name="isADLDSRole">false</Property>
    <Property name="userAccountControl">512</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="kdcEnabled">false</Property>
    <Property name="defaultRealmName">WSO2.ORG</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="ConnectionPoolingEnabled">false</Property>
    <Property name="LDAPConnectionTimeout">5000</Property>
    <Property name="RetryAttempts" />
</UserStoreManager>
When working with Active Directory:

- It is best to enable the `GetAllRolesOfUserEnabled` property in the `AuthorizationManager` as follows. See the documentation on configuring the `AuthorizationManager` for more information.

```xml
<AuthorizationManager>
  class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager"/>
  <Property name="AdminRoleManagementPermissions">/permission</Property>
  <Property name="AuthorizationRoleManagement">true</Property>
  <Property name="GetAllRolesOfUserEnabled">true</Property>
</AuthorizationManager>
```

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.

- If you are using ldaps (secured LDAP) to connect to the Active Directory as shown in the example below, you need to import the certificate of Active Directory to the `client-truststore.jks` of the WSO2 product. For information on how to add certificates to the truststore and how keystores are configured and used in a system, see Using Asymmetric Encryption.

```xml
<Property name="ConnectionURL">ldaps://10.100.1.100:636</Property>
```

- You also need to enable connection pooling for LDAPS connections at the time of starting your server, which will enhance server performance.

3. 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.

4. 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.

```xml
<Property name="UserNameAttribute">sAMAccountName</Property>
```

5. In WSO2 products based on Carbon 4.4.x, you can set the `LDAPConnectionTimeout` property: If the connection to the LDAP is inactive for the length of time (in milliseconds) specified by this property, the connection will be terminated.

6. 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 `GroupNameAttribute` properties. If the `ReadGroups` property is set to 'false', only Users can be read from the user store. You can set the configuration to read roles from the user store by reading the user/role mapping based on a membership (user list) or backlink attribute as shown below.

To read the user/role mapping based on a membership (This is used by the ApacheDirectory server and OpenLDAP):

- Enable the `ReadGroups` property.

```xml
<Property name="ReadGroups">true</Property>
```

- Set the `GroupSearchBase` property to the directory name where the Roles are stored. That is, the roles you create using the
management console of your product will be stored in this directory location. Also, when LDAP searches for users, it will start from this location of the directory. For example:

```xml
<Property
    name="GroupSearchBase">ou=system,CN=Users,DC=wso2,DC=test</Property>
```

- Set the GroupSearchFilter and GroupNameAttributes. For example:

```xml
<Property
    name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
```

- Set the MembershipAttribute property as shown below:

```xml
<Property name="MembershipAttribute">member</Property>
```

To read roles based on a backlink attribute, use the following code snippet instead of the above:

```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>
<Property name="BackLinksEnabled">true</Property>
<Property name="MembershipOfAttribute">memberOf</Property>
```

Step 2: Updating 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 user store when you start the system for the first time. Since the Active Directory user store can be written to, you have the option of creating a new admin user in the user store when you start the system for the first time. Alternatively, you can also use a user ID that already exists in the user store. For more information on setting up the system administrator and the authorization manager, see Configuring the User Realm.

- These two alternative configurations can be done as explained below.
- Find a valid user that already resides in the user store. For example, say a valid username is AdminSOA. Update the `<AdminUser>` section of your configuration as shown below. You do not have to update the password element as it is already set in the user store.

```xml
<AddAdmin>False</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
    <UserName>AdminSOA</UserName>
    <Password>XXXXXX</Password>
</AdminUser>
```
Since the user store can be written to, you can add the super tenant user to the user store. Therefore, `<AddAdmin>` should be set to `true` as shown below.

```
<AddAdmin>true</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
  <UserName>admin</UserName>
  <Password>admin</Password>
</AdminUser>
```

Step 3: Starting the server

Start your server and try to log in as the admin user you specified.

### Configuring a Read-Write LDAP User Store

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. This file is shipped with user store manager configurations for all possible user store types (JDBC, read-only LDAP/Active Directory, read-write LDAP and read-write Active directory). The instructions given below explains how to configure a read-write LDAP as the primary user store for the WSO2 server.

#### The default User Store

The primary user store that is configured by default in the `user-mgt.xml` file of WSO2 products is a JDBC user store, which reads/writes into the internal database of the product server. By default, the internal database is H2. This database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles). In the case of the WSO2 Identity Server, the default user store is an LDAP (Apache DS) that is shipped with the product.

Note that the RDBMS used in the default configuration can remain as the database used for storing Authorization information.

Follow the given steps to configure a read-write LDAP as the primary user store:

- Step 1: Setting up the read-write LDAP user store manager
- Step 2: Updating the system administrator
- Step 3: Starting the server

#### Step 1: Setting up the read-write LDAP user store manager

Once the above points are made note of and completed, you can start configuring your external read-write LDAP as the primary user store.

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-write LDAP is `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager"`

Once the above points are made note of and completed, you can start configuring your external read-write LDAP as the primary user store.

1. Enable the `<ReadWriteLDAPUserStoreManager>` user store manager class in the `user-mgt.xml` file by uncommenting the relevant code. When it is enabled, the user manager reads/writes into the LDAP 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 that you are not using; in short, you can only configure one primary user store.

2. The default configuration for the external read/write user store in the `user-mgt.xml` file is as follows. You may have to change some of these values according to your requirements. For more information about each of the properties used in the `user-mgt.xml` file for configuring the primary user store, see Properties of User Stores.
To read and write to an LDAP userstore, it is important to ensure that the `ReadGroups` and `WriteGroups` properties in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file are set to true.

```
<Property name="ReadGroups">true</Property>
<Property name="WriteGroups">true</Property>
```

Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute">` and

```
<Property name="UserNameAttribute">uid</Property>
```
b. <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 administrator.

<Property name="UserNameAttribute">uid</Property>

c. Specify the following properties that are relevant to connecting to the LDAP in order to perform various tasks.

<Property name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServerPort}</Property>
<Property name="ConnectionName">uid=admin,ou=system</Property>
<Property name="ConnectionPassword">admin</Property>

In WSO2 products based on Carbon 4.4.x, you can set the property: LDAPConnectionTimeout

If the connection to the LDAP is inactive for the length of time (in milliseconds) specified by this property, the connection will be terminated.

Set the 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 GroupNameAttribute properties. If the ReadGroups property is set to ‘false’, only Users can be read from the user store. You can set the configuration to read roles from the user store by reading the user/role mapping based on a membership (user list) or backlink attribute as shown below.

To read the user/role mapping based on a membership (This is used by the ApacheDirectory server and OpenLDAP):

- Enable the ReadGroups property.

<Property name="ReadGroups">true</Property>

- Set the GroupSearchBase property to the directory name where the Roles are stored. That is, the roles you create using the management console of your product will be stored in this directory location. Also, when LDAP searches for groups, it will start from this location of the directory. For example:

<Property name="GroupSearchBase">ou=system,CN=Users,DC=wso2,DC=test</Property>

- Set the GroupSearchFilter and GroupNameAttributes. For example:
2.

- Set the `MembershipAttribute` property as shown below:

```
<Property name="MembershipAttribute">member</Property>
```

To read roles based on a backlink attribute, use the following code snippet instead of the above:

```
<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>
<Property name="BackLinksEnabled">true</Property>
<Property name="MembershipOfAttribute">memberOf</Property>
```

Step 2: Updating 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 user store when you start the system for the first time. Since the LDAP user store can be written to, you have the option of creating a new admin user in the user store when you start the system for the first time. Alternatively, you can also use a user ID that already exists in the LDAP. For information about the system administrator user, see Configuring the System Administrator.

These two alternative configurations can be done as explained below.

- If the user store is read-only, find a valid user that already resides in the user store. For example, say a valid username is `AdminSOA`. Update the `<AdminUser>` section of your configuration as shown below. You do not have to update the password element as it is already set in the user store.

```
<AddAdmin>False</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
  <UserName>AdminSOA</UserName>
  <Password>XXXXXXX</Password>
</AdminUser>
```

- If the user store can be written to, you can add the super tenant user to the user store. Therefore, `<AddAdmin>` should be set to `true` as shown below.

```
<AddAdmin>False</AddAdmin>
<AdminRole>admin</AdminRole>
<AddAdmin>true</AddAdmin>
<AdminUser>
  <UserName>AdminSOA</UserName>
  <Password>XXXXXXX</Password>
</AdminUser>
```
<AddAdmin>true</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
  <UserName>admin</UserName>
  <Password>admin</Password>
</AdminUser>

Step 3: Starting the server

Start your server and try to log in as the admin user you specified in Step 2.

Configuring Secondary User Stores

The default configurations of WSO2 products have a single, embedded user store (primary user store). If required, you can configure WSO2 products to connect to several secondary user stores as well. After configuration, users from different stores can log in and perform operations depending on their roles/permissions. You can also configure your own customized user stores and connect them with the products as secondary stores.

The topics below explain how to configure secondary user stores manually or using the management console:

- Configuring using the management console
- Configuring manually

Before you begin:

If you are setting up a database other than the default H2 that comes with the product to store user information, select the script relevant to your database type from the <PRODUCT_HOME>/dbscripts folder and run it on your database. It creates the necessary tables.

Configuring using the management console

1. Log in to the management console and click Add under the User Stores submenu in the Main menu.
2. The Add New User Store page opens.

   Note: You cannot update the PRIMARY user store at runtime, so it is not visible on this page.

3. In the User Store Manager Class list, select the type of user store you are creating.

<table>
<thead>
<tr>
<th>User store manager</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 LDAP user stores.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</td>
<td>Use ReadWriteLDAPUserStoreManager for LDAP user stores to do both read and write operations.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager</td>
<td>Use ActiveDirectoryUserStoreManager to configure an Active Directory Domain 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.ldapReadOnlyLDAPUserStoreManager.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</td>
<td>Use JDBCUserStoreManager for JDBC user stores. The JDBC user store can be configured for read-only mode or read/write mode using the following property: &lt;Property name=&quot;ReadOnly&quot;&gt;false/true&lt;/Property&gt;.</td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2015-2019
3. Use CarbonRemoteUserStoreManager to configure another WSO2 product based on WSO2 Carbon as the user store manager.

Note: Any secondary user stores that are configured to the WSO2 product are not supported. However, the primary user store of the WSO2 product you point to works as expected.

You can also populate this drop-down list with custom user store manager implementations by adding them to the server. Here we are maintaining the samples under the WSO2 Identity Server product repo itself and you can filter that based on the Identity Server version.

4. Enter a unique domain name with no underscore (_) characters, and optionally enter a description for this user store.

5. Enter values for the properties, using the descriptions in the Descriptions column for guidance. The properties that appear vary based on the user store manager class you selected, and there may be additional properties in Optional or Advanced sections at the bottom of the screen. For information on the properties that are used when defining user stores, see Properties of User Stores.

Make sure that you do not enter “federated” as the domain name as the term is reserved for federated users.

6. Ensure that all the mandatory fields are filled and a valid domain name is given and click Add.

7. A message appears saying that the user stores are being added.

Note: The above message does not imply that the user store is added successfully. It simply means that the server is attempting to add the new user store to the end of the available chain of stores.

8. Refresh the page after a few seconds to check the status.

9. If the new user store is successfully added, it will appear in the User Stores page. This can be viewed at any time by clicking List under User Stores in the Main menu.

10. After adding to the server, you can edit the properties of the new secondary user store and enable/disable it in a dynamic manner.

Configuring manually
By default, the configuration of the primary user store is saved in the user-mgt.xml file. When you create a secondary user store using the management console as explained above, its configuration is saved to an XML file with the same name as the domain name you specify. Alternatively, you can create this XML file manually and save it as follows:

- When you configure multiple user stores, you must **give a unique domain name to each user store** in the `<DomainName>` element. If you configure a user store without specifying a domain name, the server throws an exception at startup.
- If it is the configuration of a super tenant, save the secondary user store definitions in `<PRODUCT_HOME>/repository/deployment/server/userstores` directory.
- If it is a general tenant, save the configuration in `<PRODUCT_HOME>/repository/tenants/<tenantid>/userstores` directory.
- The secondary user store configuration file must have the same name as the domain with an underscore (_) in place of the period. For example, if the domain is `wso2.com`, name the file as `wso2_com.xml`
- Only one file contains the definition for one user store domain.

### Working with Properties of User Stores

The following table provides descriptions of the key properties you use to configure primary user stores.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxUserNameListLength</td>
<td>Controls the number of users listed in the user store of a WSO2 product. This is useful when you have a large number of users and don't want to list them all. Setting this property to 0 displays all users.</td>
</tr>
<tr>
<td>ConnectionURL</td>
<td>Connection URL to the user store server. In the case of default LDAP in Carbon, the port is specified in the carbon.xml file, and a reference to that port is included in this configuration.</td>
</tr>
<tr>
<td>ConnectionName</td>
<td>The username used to connect to the database and perform various operations. This user does not have to be an administrator in the user store or have an administrator role in the WSO2 product that you are using, but this user MUST have permissions to read the user list and users' attributes and to perform search operations on the user store. The value you specify is used as the DN (Distinguish Name) attribute of the user. This property is mandatory.</td>
</tr>
<tr>
<td>ConnectionPassword</td>
<td>Password for the ConnectionName user.</td>
</tr>
<tr>
<td>DisplayNameAttribute</td>
<td>This is an optional property. The Display Name Attribute is the name by which users will be listed when you search for users in the management console (Go to Configuration -&gt; Users tab).</td>
</tr>
<tr>
<td>PasswordHashMethod</td>
<td>Password hash method to use when storing user entries in the user store.</td>
</tr>
<tr>
<td>UserNameListFilter</td>
<td>Filtering criteria for listing all the user entries in the user store. This query or filter is used when doing search operations on users. In this case, the search operation only provides the objects created from the specified class. This query is the same as listing out all the available users in the management console.</td>
</tr>
<tr>
<td>UserEntryObjectClass</td>
<td>Object class used to construct user entries. By default, it is a custom object class defined with the name wso2Pers.</td>
</tr>
<tr>
<td>UserSearchBase</td>
<td>DN of the context or object under which the user entries are stored in the user store. In this case, it is the &quot;users&quot; container. When the user store searches for users, it will start from this location of the directory.</td>
</tr>
<tr>
<td>UserNameSearchFilter</td>
<td>Filtering criteria used to search for a particular user entry.</td>
</tr>
<tr>
<td>UserNameAttribute</td>
<td>The attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, UID, etc.</td>
</tr>
<tr>
<td>UsernameWithEmailJavaScriptRegEx</td>
<td>This property defines the JavaScript regular expression pattern when the EnableEmailUserName property is set to true in carbon.xml configuration file. If you need to support both email as a user name and normal user names, you can set this property as follows.</td>
</tr>
</tbody>
</table>

```xml
<Property name="UsernameWithEmailJavaScriptRegEx">^[^\s]{3,30}$</Property>
```

Different databases have different search bases.

For information on using email address to authenticate users, click [here](#).
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PasswordJavaScriptRegEx</td>
<td>Policy that defines the password format.</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for username validation.</td>
</tr>
<tr>
<td>UsernameJavaRegEx</td>
<td>A regular expression to validate usernames. By default, strings have a length of 5 to 30. Only non-empty characters are allowed. You can provide ranges of alphabets, numbers and also ranges of ASCII values in the RegEx properties.</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for role name validation.</td>
</tr>
<tr>
<td>RolenameJavaRegEx</td>
<td>A regular expression used to validate role names. By default, strings have a length of 5 to 30. Only non-empty characters are allowed.</td>
</tr>
<tr>
<td>CaseInsensitiveUsername</td>
<td>(JDBC) This property can be set to 'false' for case insensitive JDBC user stores for performance improvements.</td>
</tr>
<tr>
<td>ReadGroups</td>
<td>Specifies whether groups should be read from the user store. If this is disabled by setting it to false, none of the groups in the user store can be read, and the following group configurations are NOT mandatory: GroupSearchBase, GroupNameListFilter, or GroupNameAttribute.</td>
</tr>
<tr>
<td>Referral</td>
<td>Guides the request to a domain controller in the correct domain</td>
</tr>
<tr>
<td>WriteGroups</td>
<td>Specifies whether groups should be written to user store.</td>
</tr>
<tr>
<td>EmptyRolesAllowed</td>
<td>Specifies whether the underlying user store allows empty groups to be created. In the case of LDAP in Carbon, the schema is modified such that empty groups are allowed to be created. Usually LDAP servers do not allow you to create empty groups.</td>
</tr>
<tr>
<td>GroupSearchBase</td>
<td>DN of the context under which user entries are stored in the user store.</td>
</tr>
<tr>
<td>GroupSearchFilter</td>
<td>The query used to search for groups.</td>
</tr>
<tr>
<td>GroupNameListFilter</td>
<td>Filtering criteria for listing all the group entries in the user store. Groups are created in LDAP using the &quot;groupOfName&quot; class. The group search operation only returns objects created from this class.</td>
</tr>
<tr>
<td>GroupEntryObjectClass</td>
<td>Object class used to construct group entries.</td>
</tr>
<tr>
<td>GroupNameSearchFilter</td>
<td>Filtering criteria used to search for a particular group entry.</td>
</tr>
<tr>
<td>GroupNameAttribute</td>
<td>Attribute used for uniquely identifying a user entry. This attribute is to be treated as the group name.</td>
</tr>
<tr>
<td>MembershipAttribute</td>
<td>Attribute used to define members of groups.</td>
</tr>
<tr>
<td>MembershipAttributeRange</td>
<td>Attribute used by Active Directories where they need limit membership attributes. The default value for this is 1500.</td>
</tr>
<tr>
<td>UserRolesCacheEnabled</td>
<td>This is to indicate whether to cache the role list of a user. By default this is set to true. Set it to false if the user roles are changed by external means and those changes should be instantly reflected in the Carbon instance.</td>
</tr>
<tr>
<td>UserDNPattern</td>
<td>(LDAP) The pattern for the user's DN, which can be defined to improve the search. When there are many user entries in the LDAP user store, defining a UserDNPattern provides more impact on performances as the LDAP does not have to travel through the entire tree to find users.</td>
</tr>
</tbody>
</table>
ReplaceEscapeCharactersAtUserLogin (LDAP) If the user name has special characters it replaces it to validate the user logging in. Only "\" and "\" are identified as escape characters.

TenantManager Includes the location of the tenant manager.

ReadOnly (LDAP and JDBC) Indicates whether the user store of this realm operates in the user read only mode or not.

IsEmailUserName (JDBC) Indicates whether the user's email is used as their username (apply when realm operates in read-only mode).

DomainCalculation (JDBC) Can be either default or custom (this applies when the realm operates in read-only mode).

PasswordDigest (JDBC) Digesting algorithm of the password. Has values such as, PLAIN_TEXT, SHA etc.

StoreSaltedPassword (JDBC) Indicates whether to salt the password.

Tip: Make sure you secure the password with salt and key.

UserNameUniqueAcrossTenants (JDBC) An attribute used for multi-tenancy.

PasswordJavaRegEx (LDAP and JDBC) A regular expression to validate passwords. By default, strings having a length between 5 to 30 with non-empty characters are allowed.

PasswordJavaScriptRegEx The regular expression used by the front-end components for password validation.

UsernameJavaRegEx A regular expression to validate usernames. By default, strings having a length 5 to 30 between with non-empty characters are allowed.

UsernameJavaScriptRegEx The regular expression used by the front-end components for username validation.

RolenameJavaRegEx A regular expression to validate role names. By default, strings having a length between 5 to 30 with non-empty characters are allowed.

RolenameJavaScriptRegEx The regular expression used by the front-end components for rolename validation.

MultiTenantRealmConfigBuilder Tenant Manager specific realm config parameter. Can be used to build different types of realms for the tenant.

SharedGroupEnabled This property is used to enable/disable the shared role functionality.

SharedGroupSearchBase Shared roles are created for other tenants to access under the mentioned DN.

SharedTenantObjectClass Object class for the shared groups created.

SharedTenantNameAttribute Name attribute for the shared group.

SharedTenantNameListFilter This is currently not used.

LDAPConnectionTimeout If the connection to an LDAP is inactive for the length of time (in milliseconds) specified by this property, the connection will be terminated.

Writing a Custom User Store Manager

This page demonstrates the process of writing a simple custom user store manager for WSO2 products.

In enterprise systems, some key components are centralized for painless management. User management is one such component that is centralized and carefully monitored. There may be user management systems that use a database or LDAP as the datasources. Any WSO2 product based on WSO2 Carbon can be configured to use these existing centralized user management systems as the user store. This topic demonstrates how to integrate an existing JDBC user store with a WSO2 product.

The following sections provide information that you need to be aware of when writing a custom user store manager:

- AbstractUserStoreManager and implementations
  - Important methods
  - Read-write methods
  - Read methods
  - Implementations
- Implementing a custom JDBC user store manager
  - Setting up the implementation
  - Writing a custom user store manager for a sample scenario
  - Deploying and configuring the custom user store manager
AbstractUserStoreManager and implementations

There are a set of methods available in the AbstractUserStoreManager class. These methods are used when interacting with user stores. When we implement a custom user store manager, it is important to identify the methods that must be implemented or overridden.

**Tip about overriding methods:**

You must select the methods to override based on your requirement. For example, if you want to change the way you encrypt the password, you only need to implement the preparePassword method. If you want to implement a completely new read/write user store manager, you must implement all the methods listed in the tables given below. If the user store is read-only, you can implement only the important methods and read methods (if you extend from AbstractUserStoreManager you have to keep unrelated methods empty).

There are a few other methods used for internal purposes. You do not need to override those methods.

The following list briefly explains the use of the available methods in the AbstractUserStoreManager class. Most of the methods provide a configuration option through properties. It is recommended to use these methods with the provided customization options.

**Important methods**

<table>
<thead>
<tr>
<th>Available methods</th>
<th>Default behaviour</th>
<th>Reasons for overriding</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean doAuthenticate(String userName, Object credential)</td>
<td>This method returns details on whether the given username and password is matched or not. Credential is usually a String literal.</td>
<td>If you want to change the authentication logic you can override this method and write your own implementation. The default task of this method is to compare the given password with the stored password. The given credentials are passed to the preparePassword method to do the salting or encryption before the comparison takes place.</td>
</tr>
<tr>
<td>String preparePassword(Object password, String saltValue)</td>
<td>This returns the encrypted or plain-text password based on the configurations.</td>
<td>You can override this method if you need to change the way you encrypt the password. If you want to change the algorithm that is used for encryption, you can configure it.</td>
</tr>
<tr>
<td>Properties getDefaultUserStoreProperties()</td>
<td>The default properties of the user store are returned using this method. These properties are used in user store related operations.</td>
<td>By overriding this method, you can programmatically change the configuration of the user store manager implementation.</td>
</tr>
</tbody>
</table>

**Read-write methods**

<table>
<thead>
<tr>
<th>Available methods</th>
<th>Default behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>void doAddUser(String userName, Object credential, String[] roleList, Map&lt;String, String&gt; claims, String profileName, boolean requirePasswordChange)</td>
<td>This method is responsible to create a new user based on the given values. We can change the JDBC query or LDAP attribute name with the user store configuration.</td>
</tr>
</tbody>
</table>

---

**Properties**

- Be sure to manually add the following property when you implement the class:
  
  ```java
  setOptionalProperty("Disabled", "false", "Whether user store is disabled");
  ```

  This property is what controls whether the user store is enabled or disabled.

- The criteria used for defining a valid username can be configured as a regex in user store configurations. If you want to change the way user name validation is done, override this method.

- Returns whether the given password is compatible with the defined criteria. This is invoked when creating a user, updating a password and authorization. Similar to the user name, you can configure the format of a valid password in configuration. If you want to change that behavior you can override this method.
void doDeleteUser(String userName)  
This removes the user store record related to the given username.

void doUpdateCredential(String userName, Object newCredential, Object oldCredential)  
Responsible to update the credential of the given username after authenticating with the existing password.

void doUpdateCredentialByAdmin(String userName, Object newCredential)  
Admin users can use this method to update the credentials of a given user. This can be done without validating the existing password.

void doAddRole(String roleName, String[] userList, boolean shared)  
Creates a new user role with given roleName and maps the given users to newly created role. Shared parameter indicate whether this role is shared among tenant or not.

void doDeleteRole(String roleName)  
This method removes the given role and related mappings from the user store.

void doUpdateRoleName(String roleName, String newRoleName)  
This method is used to update the name of the existing roles.

void doUpdateRoleListOfUser(String userName, String[] deletedRoles, String[] newRoles)  
This is used to delete the existing mappings between the given user and the deletedRoles while creating mappings to newRoles.

void doUpdateUserListOfRole(String roleName, String[] deletedUsers, String[] newUsers)  
Used to delete the existing mappings between the given role and the deletedUsers while creating mappings to newUsers.

void doSetUserClaimValue(String userName, String claimURI, String claimValue, String profileName)  
This is responsible for creating a new claim for a given user and profile, with the given claim URI and value.

void doSetUserClaimValues(String userName, Map<String, String> claims, String profileName)  
This is responsible for creating a new claim for a given user and profile, with the given list of claim URIs and values.

void doDeleteUserClaimValue(String userName, String claimURI, String profileName)  
Remove the existing claim details mapped with the given user and profile.

void doDeleteUserClaimValues(String userName, String[] claims, String profileName)  
Remove the given list of claims from a given user.

void addRememberMe(String userName, String token)  
This method is used to persist tokens in the user store.

### Read methods

<table>
<thead>
<tr>
<th>Available methods</th>
<th>Default behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean doCheckExistingUser(String userName)</td>
<td>Returns whether the provided userName already exists in the user store.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>boolean doCheckExistingRole(String roleName)</td>
<td>Returns whether the provided roleName already exists in the user store.</td>
</tr>
<tr>
<td>String[] doListUsers(String filter, int maxItemLimit)</td>
<td>This method returns a list of usernames that match with the given filter string.</td>
</tr>
<tr>
<td>String[] doGetRoleNames(String filter, int maxItemLimit)</td>
<td>Returns a list of role names that match with the given filter string.</td>
</tr>
<tr>
<td>String[] doGetExternalRoleListOfUser(String userName, String filter)</td>
<td>Returns a list of external role names of a given user that match with the given filter string.</td>
</tr>
<tr>
<td>String[] doGetSharedRoleListOfUser(String userName, String tenantDomain, String filter)</td>
<td>This method returns a list of shared role names of a given user that match with the given filter string.</td>
</tr>
<tr>
<td>Map&lt;String, String&gt; getUserPropertyValues(String userName, String[] propertyNames, String profileName)</td>
<td>This method returns values for the given propertyNames for a given userName and profileName.</td>
</tr>
<tr>
<td>String[] getUserListFromProperties(String property, String value, String profileName)</td>
<td>This returns a list of usernames that match with the given value of the given property and profileName.</td>
</tr>
<tr>
<td>String[] doGetDisplayNamesForInternalRole(String[] userNames)</td>
<td>Returns names to display in the UI for given usernames.</td>
</tr>
<tr>
<td>Date getPasswordExpirationTime(String userName)</td>
<td>Returns the password expiry date of a given user. The default value is null.</td>
</tr>
<tr>
<td>int getUserId(String username)</td>
<td>This method returns the identifier of a given user name. Default value is 0.</td>
</tr>
<tr>
<td>boolean doCheckIsUserInRole(String userName, String roleName)</td>
<td>True is returned if the given user is already mapped to the given role name.</td>
</tr>
<tr>
<td>String[] getProfileNames(String userName)</td>
<td>Returns a list of profile names mapped with a given user name.</td>
</tr>
<tr>
<td>String[] doGetSharedRoleNames(String tenantDomain, String filter, int maxItemLimit)</td>
<td>This returns a list of role names that are associated with the given tenant domain and match with the filter.</td>
</tr>
<tr>
<td>String[] doGetUserListOfRole(String roleName, String filter)</td>
<td>This method returns a list of usernames that are mapped with the given rolename.</td>
</tr>
<tr>
<td>String[] getAllProfileNames()</td>
<td>All the profile names are returned including the default profile.</td>
</tr>
<tr>
<td>boolean isValidRememberMeToken(String userName, String token)</td>
<td>This method is used to check if the given token exists for the given user.</td>
</tr>
<tr>
<td>boolean isMultipleProfilesAllowed()</td>
<td>Returns whether this user store is allowed to have multiple profiles per user. The default value is false.</td>
</tr>
</tbody>
</table>
### Implementations

In WSO2 Carbon-based products, there are four user store manager classes that implement the `AbstractUserStoreManager` class. You can select one of those classes according to the user store that you have in your environment.

<table>
<thead>
<tr>
<th>User store manager class</th>
<th>When you would use it</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</td>
<td>If your user details are stored in a database, you must use this user store manager implementation. With the abstraction provided in this implementation, most of the JDBC-based scenarios can be handled without writing a custom user store manager.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</td>
<td>You can use this class if you have an LDAP user store. This implementation does not allow you to insert or update users from the WSO2 product side. Instead you can only read and use them in the product.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldapReadWriteLDAPUserStoreManager</td>
<td>If you want to allow the WSO2 product to manipulate user store data, you need to use this implementation.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ActiveDirectoryLDAPUserStoreManager</td>
<td>Active Directory also can be used as the user store of a WSO2 product and you can configure it using this user store manager implementation.</td>
</tr>
</tbody>
</table>

#### Implementing a custom JDBC user store manager

The instructions in this section are focused on implementing a sample JDBC user store manager. For this sample, the following tools are used to implement the custom user store manager.

- Java 1.6.0
- IDE (Eclipse, IntelliJ IDEA, etc.)
- Apache Maven

**Setting up the implementation**

To set up this implementation, do the following.

1. Create a new Apache Maven project with the help of the IDE that you are using. The project should be a simple Apache Maven project and you can use any desired artifact and group ID.
2. Add the WSO2 user store management .jar file as a dependency of our project. Since this .jar file is stored in WSO2’s Maven repository, you must add the WSO2 repository to your POM file. Please see the below sample POM file.

```xml
<?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.wso2.sample</groupId>
    <artifactId>CustomReadOnlyJDBCUserStoreManager</artifactId>
    <version>1.0.0</version>
    <packaging>bundle</packaging>
    <repositories>
        <repository>
            <id>ws02-nexus</id>
            <name>WSO2 internal Repository</name>
            <url>http://maven.wso2.org/nexus/content/groups/wso2-public/</url>
            <releases>
                <enabled>true</enabled>
                <updatePolicy>daily</updatePolicy>
                <checksumPolicy>ignore</checksumPolicy>
            </releases>
        </repository>
    </repositories>
</project>
```

**Note:** Note that the version number of the carbon dependencies seen below have to be updated according to the carbon kernel version that the particular product version is compatible with. For example, WSO2 IS 5.3.0 is built on top of carbon kernel version 4.4.11 therefore, the version given in the sample pom file below is 4.4.11. Change this value accordingly based on the relevant carbon kernel version of the product you are using by referring to this release matrix.
</repositories>
<dependencies>
  <dependency>
    <groupId>org.wso2.carbon</groupId>
    <artifactId>org.wso2.carbon.user.core</artifactId>
    <version>4.4.11</version>
  </dependency>
  <dependency>
    <groupId>org.wso2.carbon</groupId>
    <artifactId>org.wso2.carbon.utils</artifactId>
    <version>4.4.11</version>
  </dependency>
  <dependency>
    <groupId>org.wso2.carbon</groupId>
    <artifactId>org.wso2.carbon.user.api</artifactId>
    <version>4.4.11</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-scr-plugin</artifactId>
      <version>1.7.2</version>
      <executions>
        <execution>
          <id>generate-scr-scrdescriptor</id>
          <goals>
            <goal>scr</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <version>2.3.5</version>
      <configurations>
        <bundle>
          <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
          <Bundle-Name>${project.artifactId}</Bundle-Name>
          <Private-Package>
            org.wso2.sample.user.store.manager.internal
          </Private-Package>
          <Export-Package>
            !org.wso2.sample.user.store.manager.internal,
            org.wso2.sample.user.store.manager.*,
          </Export-Package>
          <Import-Package>
            org.wso2.carbon.*,
            org.apache.commons.logging.*,
            org.osgi.framework.*,
            org.osgi.service.component.*
          </Import-Package>
        </bundle>
      </configurations>
    </plugin>
  </plugins>
</build>
</plugin>
Now your basic implementation is ready.

Writing a custom user store manager for a sample scenario

As a sample of how this can be done, consider a scenario where you want to use a custom hashing method using a 3rd party library such as Jasypt. So, in order to do this, you must override the `doAuthentication` and `preparePassword` methods as an example.

Do the following steps to write the custom user store manager.

1. Include the required dependencies in your development environment. To do that, include the relevant Apache Maven dependency details or manually add the .jar files to your classpath. For example, add the following XML snippet under the dependencies tag in your pom.xml file to include the Jasypt dependency.

```xml
<dependency>
  <groupId>org.jasypt</groupId>
  <artifactId>jasypt</artifactId>
  <version>1.9.2</version>
</dependency>
```

2. Create a new class by extending the existing `JDBCUserStoreManager` implementation. The following code is an example of how this would look.

```java
package com.wso2.custom.usermgt;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.jasypt.util.password.StrongPasswordEncryptor;
import org.wso2.carbon.user.api.RealmConfiguration;
import org.wso2.carbon.user.core.UserRealm;
import org.wso2.carbon.user.core.UserStoreException;
import org.wso2.carbon.user.core.claim.ClaimManager;
import org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager;
import org.wso2.carbon.user.core.profile.ProfileConfigurationManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Map;
public class CustomUserStoreManager extends JDBCUserStoreManager {
    private static Log log = LogFactory.getLog(StarkUserStoreManager.class);
    // This instance is used to generate the hash values
    private static StrongPasswordEncryptor passwordEncryptor = new
    StrongPasswordEncryptor();
    // You must implement at least one constructor
    public CustomUserStoreManager(RealmConfiguration realmConfig,
    Map<String, Object> properties, ClaimManager
    claimManager, ProfileConfigurationManager profileManager,
    UserRealm realm, Integer tenantId)
    throws UserStoreException {
        super(realmConfig, properties, claimManager,
        realm, tenantId);
        log.info("CustomUserStoreManager initialized...");
    }
```
```java
@Override
public boolean doAuthenticate(String userName, Object credential) throws UserStoreException {
    boolean isAuthenticated = false;
    if (userName != null && credential != null) {
        try {
            String candidatePassword = String.copyValueOf(((Secret) credential).getChars());

            Connection dbConnection = null;
            ResultSet rs = null;
            PreparedStatement prepStmt = null;
            String sql = null;
            dbConnection = this.getDBConnection();
            dbConnection.setAutoCommit(false);
            // get the SQL statement used to select user details
            sql = this.realmConfig.getUserStoreProperty("SelectUserSQL");
            if (log.isDebugEnabled()) {
                log.debug(sql);
            }
            prepStmt = dbConnection.prepareStatement(sql);
            prepStmt.setString(1, userName);
            // check whether tenant id is used
            if (sql.contains("UM_TENANT_ID")) {
                prepStmt.setInt(2, this.tenantId);
            }
            rs = prepStmt.executeQuery();
            if (rs.next()) {
                String storedPassword = rs.getString(3);
                // check whether password is expired or not
                boolean requireChange = rs.getBoolean(5);
                Timestamp changedTime = rs.getTimestamp(6);
                GregorianCalendar gc = new GregorianCalendar();
                gc.add(GregorianCalendar.HOUR, -24);
                Date date = gc.getTime();
                if (!(requireChange && changedTime.before(date))) {
                    // compare the given password with stored password using jasypt
                    isAuthenticated = passwordEncryptor.checkPassword(candidatePassword, storedPassword);
                }
            }
            log.info(userName + " is authenticated? " + isAuthenticated);
        } catch (SQLException exp) {
            log.error("Error occurred while retrieving user authentication info.", exp);
            throw new UserStoreException("Authentication Failure");
        }
    }
    return isAuthenticated;
}

@Override
protected String preparePassword(Object password, String saltValue) throws UserStoreException {
    if (password != null) {
        String candidatePassword = String.copyValueOf(((Secret) password).getChars());
        // ignore saltValue for the time being
        log.info("Generating hash value using jasypt...");
        return passwordEncryptor.encryptPassword(candidatePassword, saltValue);
    } else {
        log.error("Password cannot be null");
        throw new UserStoreException("Authentication Failure");
    }
```
Deploying and configuring the custom user store manager

Do the following to deploy and configure the custom user store manager in your WSO2 product.

1. Copy the artifact of your project (custom-userstore.jar, in this case) to the `<PRODUCT_HOME>/repository/components/dropins` directory. Also copy all OSGi bundles to this location. If you have any dependency .jar files, copy them to the `<PRODUCT_HOME>/repository/components/lib` directory.

2. Change the configuration of the WSO2 product to use our custom implementation for user store management. To do this, open the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file and change the `UserStoreManager` class.

```
<UserStoreManager
  class="com.wso2.custom.usermgt.CustomUserStoreManager"/>
```

**Tip:** This step provides instructions on configuring your custom user store manager as a primary user store. Alternatively, you can configure this as a secondary user store if you already have a different primary user store configured. For more information configuring user stores in WSO2 products, see Configuring User Stores.

You do not need to change anything else since you extend the JDBCUserStoreManager class, so the configurations will remain the same.

You have now implemented a custom user store manager for a WSO2 product. Once you have done this, start the product and see the log messages that you have placed inside overridden methods when you create a new user or login. This ensures that all your configurations work as intended.
Security

See the following topics for information on the security configurations that are applicable to WSO2 products:

- Maintaining Logins and Passwords
- Securing Passwords in Configuration Files
- Configuring Transport Level Security
- Enabling Java Security Manager
- Using Asymmetric Encryption
- Using Symmetric Encryption
- Mitigating Cross Site Request Forgery Attacks
- Mitigating Cross Site Scripting Attacks
- Enabling HostName Verification
- Configuring TLS Termination

See the topics given below for information on applying security patches to your Carbon product.

- WSO2 Security Patch Releases: Find the list of security patches that are applicable to each WSO2 product version from this web site.
- WSO2 Patch Application Process: Find the instructions on applying patches to WSO2 products.

Maintaining Logins and Passwords

This section covers the following topics:

- Changing the super admin password
- Unlocking the admin user
- Recovering a password
- Setting up an email login

Changing the super admin password

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 credentials, change the same in the

```xml
<UserManager>
  <Realm>
    <Configuration>
      ...
      <AdminUser>
        <UserName>admin</UserName>
        <Password>admin</Password>
      </AdminUser>
      ...
    </Realm>
  </UserManager>
</UserManager>
```

Do you have any special characters in passwords?

For usernames and passwords inside XML files, take care when giving special characters. 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?±Jv855&A4a,¾MBB®h]]>
</Password>
```
Unlocking the admin user

To unlock an admin user who is locked due to an exceeding number of login failures, restart the server using the `-unlockAdmin` system property.

Recovering a password

Use `<PRODUCT_HOME>/bin/chpasswd.sh` script.

Setting up an email login

You can configure WSO2 products to authenticate users using an email or mobile number instead of a username.

The '@' is a special character in usernames of WSO2 products as it is used in multi-tenant environments to build the user's fully-qualified name. For example, user daniel from the tenant domain WSO2.com has the fully-qualified name daniel@WSO2.com. Before using an email as the username, configure the WSO2 product to differentiate between the '@' symbol in the user's emails and usernames as follows:

1. Open `<PRODUCT_HOME>/repository/conf/carbon.xml`
2. Uncomment the commented out configuration `<EnableEmailUserName>true</EnableEmailUserName>`. This enables email authentication.

```
<EnableEmailUserName>true</EnableEmailUserName>
```

**Tip:** When you do this configuration, the email becomes the admin username and you cannot configure your email address as an attribute in your user profile.

3. Next, edit `<PRODUCT_HOME>/repository/conf/user-mgt.xml`. You might be connected to an LDAP, Active Directory, or a JDBC-based user store. Regardless of the user store manager, change the following:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserNameAttribute</td>
<td>Set the mail attribute of the user.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Property name=&quot;UserNameAttribute&quot;&gt;mail&lt;/Property&gt;</code></td>
</tr>
<tr>
<td>UserNameSearchFilter</td>
<td>Use the mail attribute of the user instead of cn or uid.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Property name=&quot;UserNameSearchFilter&quot;&gt;(&amp;(objectClass=identityPerson)(mail=?))&lt;/Property&gt;</code></td>
</tr>
<tr>
<td>UserNameListFilter</td>
<td>Use the mail attribute of the user in the user name filter list as well.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Property name=&quot;UserNameListFilter&quot;&gt;(&amp;(objectClass=identityPerson)(mail=*))&lt;/Property&gt;</code></td>
</tr>
</tbody>
</table>
Securing Passwords in Configuration Files

All WSO2 Carbon products contain some configuration files with sensitive information such as passwords. This documentation explains how such plain text passwords in configuration files can be secured using the Secure Vault implementation that is built into Carbon products.

See the following topics:

- Encrypting Passwords with Cipher Tool
- Resolving Encrypted Passwords
- Carbon Secure Vault Implementation

Encrypting Passwords with Cipher Tool

The instructions on this page explain how plain text passwords in configuration files can be encrypted using the secure vault implementation that is built into WSO2 products. Note that you can customize the default secure vault configurations in the product by implementing a new secret repository, call back handler etc. Read more about the Secure Vault implementation in WSO2 products.

In any WSO2 product that is based on Carbon 4.4.0 or a later version, the Cipher Tool feature will be installed by default. You can use this tool to easily encrypt passwords or other elements in configuration files.

- If you are a developer who is building a Carbon product, see the topic on enabling Cipher Tool for password encryption for instructions on how to include the Cipher Tool as a feature in your product build.
- The default keystore that is shipped with your WSO2 product (i.e. wso2carbon.jks) is used for password encryption by default. See this link for details on how to set up and configure new keystores for encrypting plain text passwords.

Follow the topics given below for instructions.

- Before you begin
- Encrypting passwords using the automated process
- Encrypting passwords manually
- Changing encrypted passwords

**Before you begin**

If you are using Windows, you need to have Ant (http://ant.apache.org/) installed before using the Cipher Tool.
Encrypting passwords using the automated process

This automated process can only be used for passwords that can be given as an XPath. If you cannot give an XPath for the password that you want to encrypt, you must use the manual encryption process explained in the next section.

Follow the steps given below to have passwords encrypted using the automated process:

1. The first step is to update the `cipher-tool.properties` file and the `cipher-text.properties` file with information of the passwords that you want to encrypt.

   By default, the `cipher-tool.properties` and `cipher-text.properties` files that are shipped with your product will contain information on the most common passwords that require encryption. If a required password is missing in the default files, you can add them manually.

Follow the steps given below.

a. Open the `cipher-tool.properties` file stored in the `<PRODUCT_HOME>/repository/conf/security` folder. This file should contain information about the configuration files in which the passwords (that require encryption) are located. The following format is used:

   ```
   <alias>=<file_name>//<xpath>,<true/false>
   ```

   By default, the `cipher-tool.properties` and `cipher-text.properties` files that are shipped with your product will contain information on the most common passwords that require encryption. If a required password is missing in the default files, you can add them manually.

Important!

- The `<alias>` should be the same value that is hard-coded in the relevant Carbon component.
- The `<file_name>` specifies the path to the XML file that contains the password. This can be the relative file path, or the absolute file path (starting from `<PRODUCT_HOME>`).
- The `<xpath>` specifies the XPath to the XML `element/attribute/tag` that should be encrypted. See the examples given below.
- The flag that follows the XPath should be set to 'false' if you are encrypting the value of an `XML element`, or the value of an `XML attribute's tag`. The flag should be 'true' if you are encrypting the `tag` of an `XML attribute`. See the examples given below.
- When using Secure Vault, as you use the password aliases in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file, make sure to define these aliases in the following files, which are in the `<PRODUCT_HOME>/repository/conf/security` directory as follows:

   • Define your password in the `cipher-text.properties` file.

     ```
     Carbon.Security.InternalKeyStore.KeyPassword=[your_password]
     ```

   • Define the XPath of your password in the `cipher-tool.properties` file.

     ```
     ```

Only applicable when using WSO2 API Manager Analytics

When using Secure Vault with WSO2 API Manager Analytics (WSO2 API-M Analytics), make sure to define the password aliases in the following files, which are in the `<PRODUCT_HOME>/repository/conf/security` directory as follows:

• Define your password in the `cipher-text.properties` file.

```
Example 1: Consider the admin user’s password in the `user-mgt.xml` file shown below.

```xml
<UserManager>
  <Realm>
    <Configuration>
      <AddAdmin>true</AddAdmin>
      <AdminRole>admin</AdminRole>
      <AdminUser>
        <UserName>admin</UserName>
        <Password>admin</Password>
      </AdminUser>
    </Configuration>
    ........
  </Realm>
</UserManager>
```

To encrypt this password, the `cipher-tool.properties` file should contain the details shown below. Note that this password is a value given to an XML element (which is 'Password'). Therefore, the XPath ends with the element name, and the flag that follows the XPath is set to 'false'.

```properties
DataBridge.Config.keyStorePassword=[your_password]
Analytics.DASPassword=[your_password]
Analytics.DASRestApiPassword=[your_password]
```

- Define the XPath of your password in the `cipher-tool.properties` file.

```properties
DataBridge.Config.keyStorePassword=repository/conf/data-bridge/data-bridge-config.xml//dataBridgeConfiguration/keyStorePassword,false
Analytics.DASPassword=repository/conf/api-manager.xml//APIManager/Analytics/DASPassword,true
Analytics.DASRestApiPassword=repository/conf/api-manager.xml//APIManager/Analytics/DASRestApiPassword,true
```

Example 2: Consider the password that is used to connect to an LDAP user store (configured in the `user-mgt.xml` file) shown below.

```xml
<Property name="ConnectionPassword">admin</Property>
```

To encrypt this password, the `cipher-tool.properties` file should be updated as shown below. Note that there are two possible alias values you can use for this attribute. In this example, the 'Property' element of the XML file uses the 'name' attribute with the
"ConnectionPassword" tag. The password we are encrypting is the value of this "ConnectionPassword" tag. This is denoted in the XPath as `Property[@name='ConnectionPassword']`, and the flag that follows the XPath is set to 'false'.

- Using the `UserStoreManager.Property.ConnectionPassword` alias:

```
UserStoreManager.Property.ConnectionPassword=repository/conf/user-mgt.xml//UserManager/Realm/UserStoreManager/Property[@name='ConnectionPassword'],false
```

- Using the `UserManager.Configuration.Property.ConnectionPassword` alias:

```
UserManager.Configuration.Property.ConnectionPassword=repository/conf/user-mgt.xml//UserManager/Realm/UserStoreManager/Property[@name='ConnectionPassword'],false
```

If you are trying the above example, be sure that only the relevant user store manager is enabled in the `user-mgt.xml` file.

**Example 3:** Consider the keystore password specified in the `catalina-server.xml` file shown below.

```xml
...
  keystorePass="wso2carbon"
  ...
>
```

To encrypt this password, the `cipher-tool.properties` file should contain the details shown below. In this example, 'Connector' is the XML element, and 'keystorePass' is an attribute of that element. The password value that we are encrypting is the value of the XML attribute. This is denoted in the XPath as 'Connector[@keystorePass]', and the flag that follows the XPath is set to 'true'.

```
```

b. Open the `cipher-text.properties` file stored in the `<PRODUCT_HOME>/repository/conf/security` folder. This file should contain the secret alias names and the corresponding plaintext passwords (enclosed within square brackets) as shown below.

```
(alias)=[plain_text_password]
```

Shown below are the records in the `cipher-text.properties` file for the three examples discussed above.
Encrypting passwords manually

This manual process can be used for encrypting any password in a configuration file. However, if you want to encrypt any elements that cannot use an xpath to specify the location in a configuration file, you must use manual encryption. It is not possible to use the automated encryption process if an xpath is not specified for the element.

For example, consider the log4j.properties file given below, which does not use xpath notations. As shown below, the password of the LOGEVENT append is set to admin:

```properties
UserManager.AdminUser.Password=[admin]
```
# LOGEVENT is set to be a LogEventAppender using a PatternLayout to send logs to LOGEVENT
log4j.appender.LOGEVENT=org.wso2.carbon.logging.service.appender.LogEventAppender
log4j.appender.LOGEVENT.url=tcp://localhost:7611
log4j.appender.LOGEVENT.layout=org.wso2.carbon.utils.logging.TenantAwarePatternLayout
log4j.appender.LOGEVENT.columnList=%T,%S,%A,%d,%c,%p,%m,%I,%Stacktrace
log4j.appender.LOGEVENT.userName=admin
log4j.appender.LOGEVENT.password=admin
log4j.appender.LOGEVENT.processingLimit=1000
log4j.appender.LOGEVENT.maxTolerableConsecutiveFailure=20

Since we cannot use the automated process to encrypt the admin password shown above, follow the steps given below to encrypt it manually.

1. Download and install a WSO2 product.
2. Open a command prompt and go to the <PRODUCT_HOME>/bin directory, where the cipher tool scripts (for Windows and Linux) are stored.
3. You must first enable the Cipher tool for the product by executing the -Dconfigure command with the cipher tool script as shown below.
   - On Linux: ./ciphertool.sh -Dconfigure
   - On Windows: .\ciphertool.bat -Dconfigure

   If you are using the cipher tool for the first time, this command will first initialize the tool for your product. The tool will then encrypt any plain text passwords that are specified in the cipher-text.properties file. See the automated encryption process for more information.

4. Now, you can start encrypting the admin password manually. Execute the Cipher tool using the relevant command for your OS:
   - On Linux: ./ciphertool.sh
   - On Windows: .\ciphertool.bat

5. You will be asked to enter the primary key password, which is by default 'wso2carbon'. Enter the password and proceed.
6. You will now be asked to enter the plain text password that you want to encrypt. Enter the following element as the password and proceed:

```
Enter Plain Text Value : admin
```

Note that in certain configuration files, the password that requires encryption may not be specified as a single value as it is in the log4j.properties file. For example, the jndi.properties file used in WSO2 ESB contains the password in the connection URL. In such cases, you need to encrypt the entire connection URL as explained here.

7. You will receive the encrypted value. For example:

```
Encrypted value is:
gamPtzAcMScBaH1i2LXsmp1i4HL1OM/srL5pB8jyknRKQ2z7T7NuCvtl+qEkE1rLgw1ro9z
3ikuE0KFuapXrCBe5pxfGMOl4n4/k7dn2S1wbsG8C++/
2fUuft1o6cqvDRM55fQw2CPfYb1713HvVu3oDaJ9I%KgSbvh1Qj6zqzg=
```

8. Open the cipher-text.properties file, stored in the <PRODUCT_HOME>/repository/conf/security folder.
9. Add the encrypted password against the secret alias as shown below.
9. log4j.appender.LOGEVENT.password=cpw74SGeBNqAVpqyqj5/xshSyW5BDW9d1W0xM2
DxVea6RjyAlJRUtZ4SzfSgSzy2GQj/2jQiw701eT5EQEAR8XLGaqlsE5ILNoe9dhyLiPEXPRGk/BgUQD
YiB0nU7wRsbRByXvrf+ak8ulX2yGv0Sf8=

10. Now, open the log4j.properties file, stored in the <PRODUCT_HOME>/repository/conf folder and replace the plain text element with the alias of the encrypted value as shown below.

```properties
# LOGEVENT is set to be a LogEventAppender using a PatternLayout to send logs to LOGEVENT
....
log4j.appender.LOGEVENT.password=secretAlias:log4j.appender.LOGEVENT.password
....
```

11. If you are encrypting a password in the <PRODUCT_HOME>/repository/conf/identity/EndpointConfig.properties file, you need to add the encrypted values of the keys in the EndpointConfig.properties file itself.

```properties
```

Another example of a configuration file that uses passwords without an XPath notation is the jndi.properties file. This file is used in WSO2 Enterprise Service Bus (WSO2 ESB) for the purpose of connecting to a message broker. You can read more about this functionality from here. As shown below, this file contains a password value (admin) in the connection URL (amqp://admin:admin@clientID/carbon?brokerlist=tcp://localhost:5673). To encrypt this password, you can follow the same manual process explained above. However, you must encrypt the entire connection URL (amqp://admin:admin@clientID/carbon?brokerlist=tcp://localhost:5673) and not just the password value given in the URL.
# register some connection factories
# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.QueueConnectionFactory =
ampq://admin:admin@localhost/carbon?brokerlist='tcp://localhost:5672'

# register some queues in JNDI using the form
# queue.[jndiName] = [physicalName]
queue.MyQueue = example.MyQueue

# register some topics in JNDI using the form
# topic.[jndiName] = [physicalName]
topic.MyTopic = example.MyTopic

## Changing encrypted passwords

To change any password which we have encrypted already, follow the below steps:

1. Be sure to shut down the server.
2. Open a command prompt and go to the `<PRODUCT_HOME>/bin` directory, where the cipher tool scripts (for Windows and Linux) are stored.
3. Execute the following command for your OS:
   - On Linux: `.ciphertool.sh -Dchange`
   - On Windows: `.ciphertool.bat -Dchange`
   
   If you are using the cipher tool for the first time, this command will first initialize the tool for your product. The tool will then encrypt any plain text passwords that are specified in the `cipher-text.properties` file for automatic encryption.

4. It will prompt for the primary keystore password. Enter the keystore password (which is "wso2carbon" for the default keystore).
5. The alias values of all the passwords that you encrypted will now be shown in a numbered list.
6. The system will then prompt you to select the alias of the password which you want to change. Enter the list number of the password alias.
7. The system will then prompt you (twice) to enter the new password. Enter your new password.

If you have encrypted passwords as explained above, note that these passwords have to be decrypted again for the server to be usable. That is, the passwords have to be resolved by a system administrator during server startup. The Resolving Passwords topic explains how encrypted passwords are resolved.

## Resolving Encrypted Passwords

If you have secured the plain text passwords in configuration files using Secure Vault, the keystore password and private key password of the product's primary keystore will serve as the root passwords for Secure Vault. This is because the keystore passwords are needed to initialise the values encrypted by the Secret Manager in the Secret Repository. Therefore, the Secret Callback handler is used to resolve these passwords. Read about the Secure Vault implementation in WSO2 products. Also, see how passwords in configuration files are encrypted using Secure Vault.

The default secret CallbackHandler in a WSO2 product provides two options for reading these encrypted passwords when you start the server:

- **Enter password in command-line**
- **Start server as a background job**

### Enter password in command-line

1. Start the server by running the product start up script from the `<PRODUCT_HOME>/bin/` directory as shown below.
2. When you run the startup script, the following message will be prompted before starting the server: “[Enter KeyStore and Private Key Password :]”. This is because, in order to connect to the default user store, the encrypted passwords should be decrypted. The administrator starting the server must provide the private key and keystore passwords using the command-line. Note that passwords are hidden from the terminal and log files.

Start server as a background job

If you start the WSO2 Carbon server as a background job, you will not be able to provide password values on the command line. Therefore, you must start the server in “daemon” mode as explained below.

1. Create a new file in the `<PRODUCT_HOME>` directory. The file should be named according to your OS as explained below.
   - For Linux: The file name should be `password-tmp`
   - For Windows: The file name should be `password-tmp.txt`

   When you start the server (see step 3 below), the keystore password will be picked from this new file. Note that this file is automatically deleted from the file system after the server starts. Therefore, the admin has to create a new text file every time the server starts. Alternatively, if you want to retain the password file after the server starts, the file should be named as follows:
   - For Linux: The file name should be `password-persist`
   - For Windows: The file name should be `password-persist.txt`

2. Add “wso2carbon” (the primary keystore password) to the new file and save. By default, the password provider assumes that both private key and keystore passwords are the same. If not, the private key password must be entered in the second line of the file.
3. Now, start the server as a background process by running the following command.

   
   ```sh
   ./wso2server.sh start
   ```

4. Start the server by running the product start-up script from the `<PRODUCT_HOME>/bin` directory by executing the following command:

   ```sh
   daemon.sh wso2server.sh -start
   ```

Carbon Secure Vault Implementation

WSO2 Carbon is shipped with a Secure Vault implementation, which is a modified version of synapse Secure Vault. This allows you to store encrypted passwords that are mapped to aliases. That is, you can use the aliases instead of the actual passwords in your configuration files for better security. For example, some configurations require the admin username and password. If the admin user password is “admin”, you could use the alias `UserManager.AadminUser.Password` in your configuration file. You would then map that alias to the actual password “admin”. At runtime, the product will look up this alias in the secure vault and then decrypt and use its password.

The Cipher Tool is used in WSO2 products to create encrypted values for passwords. See the following sections in the documentation for more information:

- Encrypting Passwords with Cipher Tool
- Resolving Encrypted Passwords

See the following topics:

- Elements of the Secure Vault implementation
- Customizing the Secure Vault configuration
  - Creating a Secret Callback Handler
  - Creating a custom Secret Repository
**Elements of the Secure Vault implementation**

Some of the important elements in the secure vault implementation, which are used in Carbon products for encrypting plain text passwords are as follows:

- **Secret Repository:** This is used to store the secret values (encrypted values). The `cipher-text.properties` file, located in the `<PRODUCT_HOME>/repository/conf/security` folder is the default file based secret repository used by the Secret Manager in your Carbon product. Note that, currently, Secure Vault only implements file based secret repositories. The Secret Repository stores aliases vs. their actual secrets in encrypted format (encrypted via a key in keystore). Any secret repositories can be written by implementing the `SecretRepositoryProvider` classes. See the topic on customizing the Secure Vault configuration.

- **Secret Manager:** The Secret Manager initializes the Secret Repository and the keystore configured for the Carbon server. The secrets stored in the Secret Repository are accessed using the aliases indicated in the `cipher-text.properties` file. The keystore is required to create the decryption crypto, which can be used to resolve encrypted secret values. The keystore and Secret Repository are configurable through the `secret-conf.properties` file, which is created in the `<PRODUCT_HOME>/repository/conf/security` folder when you execute the Cipher Tool.

- **Secret Callback:** This provides the actual password for a given alias. There is a `SecretManagerSecretCallbackHandler`, which is combined with Secret Manager to resolve the secret. Any callback can be written by implementing the `SecretCallbackHandler` class. See the topic on customizing the Secure Vault configuration.

- **Secret Resolver:** Any configuration builder that uses secret information within its own configuration file needs to initialize the Secret Resolver when building its own configuration. The Secret Resolver keeps a list of secured elements that need to be defined in the configuration file with secret aliases. Secret Resolver initializes the Secret Callback handler class, which is defined in the configuration file.

**Customizing the Secure Vault configuration**

You can implement your own Secure Vault configurations by changing the default Secret Repository and the Secret Callback Handler. See the following for topics for instructions:

**Creating a Secret Callback Handler**

Let's see how we can write a new Secret Callback Handler class to secure the user management and registry database connection password. In this sample, you do not need to configure a Secret Repository or keystore (cipher-text.properties) as you are not going to store the secret or encrypted values.

1. Write a Secret Callback class. You need to implement the `SecretCallbackHandler` interface or extend the `AbstractSecretCallbackHandler` abstract class. For example,

```java
public class HardCodedSecretCallbackHandler extends AbstractSecretCallbackHandler {
    protected void handleSingleSecretCallback(SingleSecretCallback singleSecretCallback) {
        singleSecretCallback.setSecret("password");
    }
}
```

2. We can set multiple password-based as follows:

```java
public class HardCodedSecretCallbackHandler extends AbstractSecretCallbackHandler {
    protected void handleSingleSecretCallback(SingleSecretCallback singleSecretCallback) {
        if("foo".equals(singleSecretCallback.getId())){
            singleSecretCallback.setSecret("foo_password");
        } else if("bar".equals(singleSecretCallback.getId())){
            singleSecretCallback.setSecret("bar_password");
        }
    }
}
```

3. Create a JAR or an OSGI bundle and copy the JAR file to the `<PRODUCT_HOME>/repository/component/lib/` directory or the OSGI bundle to the `<PRODUCT_HOME>/repository/component/dropins/` directory.

4. Configure the `master-datasources.xml` file with an alias name and your Secret Callback handler class name. For example,
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000</url>
      <username>wsocarbon</username>
      <passwordsvns:secretAlias="Datasources.WSO2_CARBON_DB.Configuration.Password">password</password>
      <driverClassName>org.h2.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
    </definition>
</datasource>

Also, replace the secret callback handler class name in <PRODUCT_HOME>/repository/conf/security/secret-conf.properties file with your Secret Callback handler class name.

5. Restart the server.

Creating a custom Secret Repository

To create a custom secret repository, you need to implement the SecretRepository and SecretRepositoryProvider interfaces:

1. Create your custom secret repository by implementing the org.wso2.securevault.secret.SecretRepository interface:

   public class CustomSecretRepositoryImpl extends SecretRepository {
     public void init(Properties properties, String s) {
     }
     public String getSecret(String s) {
       return null;
     }
     public String getEncryptedData(String s) {
       return null;
     }
     public void setParent(SecretRepository secretRepository) {
     }
     public SecretRepository getParent() {
       return null;
     }
   }

2. Then you need to implement the org.wso2.securevault.secret.SecretRepositoryProvider class as shown below. This class returns an instance of the custom SecretRepository that you implemented above.
public class CustomSecretRepositoryProvider implements SecretRepositoryProvider {
    public SecretRepository getSecretRepository(IdentityKeyStoreWrapper identityKeyStoreWrapper,
            TrustKeyStoreWrapper trustKeyStoreWrapper) {
        return new CustomSecretRepositoryImpl(identityKeyStoreWrapper, trustKeyStoreWrapper);
    }
}

3. Create a JAR or an OSGI bundle.
4. Then, copy the JAR file to the `<PRODUCT_HOME>/repository/component/lib/directory` or the OSGI bundle to the `<PRODUCT_HOME>/repository/component/dropins` directory.
5. Replace the `secretRepositories.file.provider` entry in the `secret-conf.properties` file (stored in the `<PRODUCT_HOME>/repository/conf/security/directory`) with your secret repository class name.

Configuring Transport Level Security

Given below are the various transport-level security configurations that are required for WSO2 products. See the following topics for instructions.

- Enabling TLS and disabling SSL support
- Enabling SSL protocols and ciphers in ThriftAuthenticationService
- Enabling TLSv1.1/TLSv1.2 for products with JDK 1.7
- Disabling weak ciphers
- Changing the server name in HTTP response headers

Enabling TLS and disabling SSL support

The transport-level security protocol of the Tomcat server is configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file. By default, "TLS" is configured as the SSL protocol for HTTPS communication by setting the `sslProtocol="TLS"` attribute in the `catalina-server.xml` file. Specifying TLS as the SSL protocol ensures that all TLS versions, as well as SSL protocol versions, are supported. However, due to the Poodle Attack, it is necessary to make sure that only TLS protocol versions are enabled.

Note that in some WSO2 products, such as WSO2 Enterprise Integrator (ESB profile) and WSO2 API Manager, pass-thru transports are enabled. Therefore, to disable SSL in such products, the `axis2.xml` file stored in the `axis2/conf/axis2/` directory should also be configured.

**Poodle Attack:**

It is necessary to disable SSL in Carbon servers because of a bug (Poodle Attack) in the SSL protocol that could expose critical data encrypted between clients and servers. The Poodle Attack makes the system vulnerable by telling the client that the server does not support the more secure TLS (Transport Layer Security) protocol, and thereby forces it to connect via SSL. The effect of this bug can be mitigated by disabling the SSL protocol for your server.

**Disabling SSL support for products with JDK 1.8**

Follow the steps given below to disable SSL support for the Tomcat layer.

1. Open the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.
2. Make a backup of the `catalina-server.xml` file and stop the product server.
3. Find the Connector configuration corresponding to TLS (usually, this connector has the port set to 9443 and the `sslProtocol` as TLS). Remove the `sslProtocol="TLS"` attribute and replace it with `sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"` as shown below.

```
    port="9443"
    bindOnInit="false"
    sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"
```

4. Start the server.
5. To verify that the configurations are all set correctly, download and run the TestSSLServer.jar.

```java
java -jar TestSSLServer.jar localhost 9443
```

The output of the command after disabling SSL is shown below.

Supported versions: TLSv1.0
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):

- TLSv1.0
- RSA_EXPORT_WITH_RC4_40_MD5
- RSA_WITH_RC4_128_MD5
- RSA_WITH_RC4_128_SHA
- RSA_EXPORT_WITH_DES40_CBC_SHA
- RSA_WITH_DES_CBC_SHA
- RSA_WITH_3DES_EDE_CBC_SHA
- DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- DHE_RSA_WITH_DES_CBC_SHA
- DHE_RSA_WITH_3DES_EDE_CBC_SHA
- RSA_WITH_3DES_EDE_CBC_SHA
- RSA_WITH_AES_128_CBC_SHA
- DHE_RSA_WITH_AES_128_CBC_SHA
- DHE_RSA_WITH_AES_256_CBC_SHA
- RSA_WITH_AES_256_CBC_SHA
- DHE_RSA_WITH_AES_256_CBC_SHA

If you have enabled the PassThrough transport, do the following:

This is applicable for WSO2 Enterprise Integrator (ESB profile) and WSO2 API Manager products.

1. Stop the server.
2. Open the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file and add the specified parameter under the `<transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener">` element as well as under the `<transportSender name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLSender">` element. If you are using JDK 1.8, you can add the following parameter:

```xml
<parameter name="HttpsProtocols">TLSv1,TLSv1.1,TLSv1.2</parameter>
```

- If you are using JDK 1.6, add the following parameter:

```xml
<parameter name="HttpsProtocols">TLSv1</parameter>
```

- If you are using JDK 1.7, add the following parameter:

```xml
<parameter name="HttpsProtocols">TLSv1,TLSv1.1,TLSv1.2</parameter>
```
2. Start the server.
4. Test the pass-through transport using the following command with the corresponding port:

```bash
$ java -jar TestSSLServer.jar localhost 8243
```

Enabling SSL protocols and ciphers in ThriftAuthenticationService

Do the following to enable SSL protocols and ciphers in the ThriftAuthenticationService.

1. Add the following configurations in the `<CARBON_SERVER>/repository/conf/identity/thrift-authentication.xml` file as sub-elements of the root `<Server>` element.

```xml
<SSLEnabledProtocols>TLSv1,TLSv1.1,TLSv1.2</SSLEnabledProtocols>
<Ciphers>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256</Ciphers>
```

**Tip:** You can also add the following additional cipher suites to the `<Ciphers>` property if JCE Unlimited Strength Jurisdiction Policy is enabled in Java.

```xml
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_DHE_RSA_WITH_AES_256_CBC_SHA384,TLS_DHE_RSA_WITH_AES_256_CBC_SHA
```

If you wish to remove TLSv1 or TLSv1.1, you can do so by removing them as values from the `<SSLEnabledProtocols>` property.

2. Restart the server.

Enabling TLSv1.1/TLSv1.2 for products with JDK 1.7

The TLS protocol is set to TLSv1.0 (by default), in WSO2 products running on JDK 1.7. You cannot configure this using the `catalina-server.xml` file or the `axis2.xml` file as we do with products based on JDK 1.7. Therefore, you need to enable TLSv1.1 and TLSv1.2 globally by setting a system property.

1. Download the following artifacts:
   - `wso2-ssl-socket-factory-provider-1.0.0.jar`
   - `wso2-ssl-security`
2. Copy the `wso2-ssl-socket-factory-provider-1.0.0.jar` file to the `<PRODUCT_HOME>/lib/endorsed` directory.
3. Copy the `wso2-ssl-security` file to the `<PRODUCT_HOME>/repository/conf/` directory.
4. Open the product startup script (`wso2server.sh` for Linux, or `wso2server.bat` for Windows), which is stored in the `<PRODUCT_HOME>/bin` directory.
5. Add the following system properties to the script:

   ```bash
   <parameter name="HttpsProtocols">TLSv1,TLSv1.1,TLSv1.2</parameter>
   ```
Disabling weak ciphers

A cipher is an algorithm for performing encryption or decryption. When you set the `sslprotocol` of your server to TLS, the TLS and the default ciphers get enabled without considering the strength of the ciphers. This is a security risk as weak ciphers, also known as EXPORT ciphers, can make your system vulnerable to attacks such as the Logjam attack on Diffie-Hellman key exchange. The Logjam attack is also called the Man-in-the-Middle attack. It downgrades your connection’s encryption to a less-secured level (e.g., 512 bit) that can be decrypted with sufficient processing power.

To prevent these types of security attacks, it is encouraged to disable the weak ciphers. You can enable only the ciphers that you want the server to support in a comma-separated list in the `ciphers` attribute. Also, if you do not add this cipher attribute or keep it blank, the browser will support all the SSL ciphers by JSSE. This will enable the weak ciphers.

Disabling weak ciphers for the Tomcat transport

1. Open the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.
2. Make a backup of the `catalina-server.xml` file and stop the WSO2 product server.
3. Add the `ciphers` attribute to the existing configuration in the `catalina-server.xml` file by adding the list of ciphers that you want your server to support as follows: `ciphers="<cipher-name>,<cipher-name>"`. See the example given below.

   For Tomcat version 7.0.59 and JDK version 1.7:
   ```
   ciphers="TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA"
   ```
   For Tomcat version 7.0.59 and JDK version 1.8:
   ```
   ciphers="TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256"
   ```

See the list of supported cipher suites.
4. Start the server.
5. To verify that the configurations are all set correctly, download and run the `TestSSLServer.jar`.

   ```
   $ java -jar TestSSLServer.jar localhost 9443
   ```

Note the following when you run `TestSSLServer.jar`:

- The "Supported cipher suites" section in the output does not contain any EXPORT ciphers.
- When you use the supported cipher suites listed here, the BEAST attack status will be shown as vulnerable. Note that this is a client-side vulnerability caused by the TLSv1 protocol. You can make the BEAST status protected by removing TLSv1, which will make clients with TLSv1 unusable. Therefore, it is recommended to fix this from the client side.

Disables weak ciphers for the PassThrough transport

Remove any weak ciphers from the PassThrough transport and ensure that the server does not accept connections using those weak ciphers. The
PassThrough transport is configured using the axis2.xml file (stored in the `<PRODUCT_HOME>/repository/conf/axis2/` directory).

1. Open the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file.
2. Make a backup of the axis2.xml file and stop the WSO2 product server.
3. You need to add the `PreferredCiphers` parameter under the "Transport Ins (Listeners)" section along with the list of relevant cipher suites.

   ```xml
   <parameter
       name="PreferredCiphers">TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256</parameter>
   ``

4. Start the server.
5. Test the pass-through transport using the following command with the corresponding port:

   ```bash
   $ java -jar TestSSLServer.jar localhost 8243
   ```

From Firefox 39.0 onwards, the browser does not allow to access Web sites that support DHE with keys less than 1023 bits (not just DHE_EXPORT). 768/1024 bits are considered to be too small and vulnerable to attacks if the hacker has enough computing resources.

To use AES-256, the Java JCE Unlimited Strength Jurisdiction Policy files need to be installed. Download them from [http://www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html).

From Java 7, you must set the `jdk.certpath.disabledAlgorithms` property in the `<JAVA_HOME>/jre/lib/security/java.security` file to `jdk.certpath.disabledAlgorithms=MD2, DSA, RSA keySize < 2048`. It rejects all algorithms that have key sizes less than 2048 for MD2, DSA and RSA.

**Note:** This tip is not applicable when you are disabling weak ciphers in WSO2 Identity Server.

### Changing the server name in HTTP response headers

By default, all WSO2 products pass "WSO2 Carbon Server" as the server value in HTTP headers when sending HTTP responses. This means that information about the WSO2 product stack will be exposed through HTTP responses. It is recommended to change this by configuring the server name in the catalina-server.xml file.

1. Open the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.
2. Add a new server name using the `server` property (under the relevant Tomcat connector configuration):

   ```xml
   server="WSO2 Carbon Server"
   ```

See the [Security Guidelines for Production Deployment](#) for the full list of security-related recommendations for WSO2 products.

### 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>/repository/conf/sec.policy` file. You modify this file to change the Java security permissions as required.

Before you begin

- Ensure that you have Java 1.8 installed.
- Note that you need to use a keystore for signing JARs using the Java security manager. In this example, you will be using the default keystore in your WSO2 product (`<PRODUCT_HOME>/repository/resources/security/wso2carbon.jks`). You can read about the recommendations for using keystores from [here](#).

The steps below show how to enable the Java Security Manager for WSO2 products.

1. Download the WSO2 product to any location (e.g., `<HOME>/user/<product-pack>`).
2. To sign the JARs in your product, you need a key. You can generate a new keystore (with a new key) by executing the `keytool` command given below. Note that the new keystore is created in the directory from which you execute the `keytool` command.

   ```bash
   keytool -genkey -alias signFiles -keyalg RSA -keystore signkeystore.jks -validity 3650 -dname "CN=Sanjeewa,OU=Engineering, O=WSO2, L=Colombo, ST=Western, C=LK"
   Enter keystore password:
   Re-enter new password:
   Enter key password for (RETURN if same as keystore password)
   ```

   Now you have a new keystore (`signkeystore.jks`) with a new public key certificate (`signFiles`).

3. By default, WSO2 products use the default `wso2carbon.jks` keystore for signing JARs. This keystore is stored in the `<PRODUCT_HOME>/repository/resources/security` directory. Therefore, you need to add the `signFiles` public key certificate that you created earlier into the `wso2carbon.jks` keystore.

   First, export the `signFiles` public key certificate from the `signkeystore.jks` keystore by executing the following command:

   ```bash
   $ keytool -export -keystore signkeystore.jks -alias signFiles -file sign-cert.cer
   ```

   Then, import the same `signFiles` certificate to the `wso2carbon.jks` keystore by executing the command given below. Be sure to specify the correct directory path to the `wso2carbon.jks` file of your product.

Please note that this document is currently under review. Currently, WSO2 does not recommend Java security manager for production environments due to some known issues.
$ keytool -import -alias signFiles -file sign-cert.cer -keystore
<PATH_to_PRODUCT_HOME>/repository/resources/security/wso2carbon.jks
    Enter keystore password:
    Owner: CN=Sanjeewa, OU=Engineering, O=WSO2, L=Colombo, ST=Western,
C=LK
    Issuer: CN=Sanjeewa, OU=Engineering, O=WSO2, L=Colombo, ST=Western,
C=LK
    Serial number: 5486f3b0
    Valid from: Tue Dec 09 18:35:52 IST 2014 until: Fri Dec 06 18:35:52 IST 2024
    Certificate fingerprints:
    Signature algorithm name: SHA1withRSA
    Version: 3
    Trust this certificate? [no]: yes
    Certificate was added to keystore

Note that WSO2 no longer recommends MD5 for JAR signing due to cryptographic limitations.

4. Open the security policy file, and update the "grant signedBy" value in the with the new signFiles alias key, as shown below.

        grant signedBy "signFiles" {
            // permission java.util.PropertyPermission "+", "read";
            // permission java.lang.RuntimePermission "+", "+";
            // permission java.io.FilePermission "+", "+";
            permission java.security.AllPermission;
        };

5. Prepare the scripts to sign the JARs and grant them the required permission. For example, the signJar.sh script given below can be used to sign each JAR file separately or you can use the signJars.sh script, which runs a loop to read all JARs and sign them.
signJar.sh script

```bash
#!/bin/bash
set -e
jarfile=$1
keystore_file="signkeystore.jks"
keystore_keyalias='signFiles'
keystore_storepass='wso2123'
keystore_keypass='wso2123'
signjar="${JAVA_HOME}/bin/jarsigner -keystore $keystore_file
-storepass $keystore_storepass -keypass $keystore_keypass"
verifyjar="${JAVA_HOME}/bin/jarsigner -keystore $keystore_file
-verify"

echo "Signing $jarfile"
$signjar $jarfile $keystore_keyalias
echo "Verifying $jarfile"
$verifyjar $jarfile
# Check whether the verification is successful.
if [ $? -eq 1 ]
then
    echo "Verification failed for $jarfile"
fi
```

signJars.sh script

```bash
#!/bin/bash
if [[ ! -d $1 ]]; then
    echo "Please specify a target directory"
    exit 1
fi
for jarfile in `find . -type f -iname ".jar"`
do
    ./signJar.sh $jarfile
done
```

6. Execute the following commands to sign the JARs in your product:

   ./signJars.sh /HOME/user/<product-pack>

Every time you add an external JAR to the WSO2 product, sign them manually using the above instructions for the Java Security Manager to be effective. You add external JARs to the server when extending the product, applying patches etc.

7. Open the startup script in the <PRODUCT_HOME>/bin folder. For Linux, it is wso2server.sh.
8. Add the following system properties to the startup script and save the file:
9. Create a `sec.policy` file with the required security policies in the `<PRODUCT_HOME>/repository/conf` folder and start the server. Starting the server makes the Java permissions defined in the `sec.policy` file to take effect.

An example of a `sec.policy` file is given below. It includes mostly WSO2 Carbon-level permissions.

```java
grant {
    // Allow socket connections for any host
    permission java.net.SocketPermission "*:1-65535", "connect,resolve";

    // Allow to read all properties. Use -Ddenied.system.properties in
    // wso2server.sh to restrict properties
    permission java.util.PropertyPermission "*", "read";
    permission java.lang.RuntimePermission "getClassLoader";

    // CarbonContext APIs require this permission
    permission java.lang.management.ManagementPermission "control";

    // Required by any component reading XMLs. For example:
    org.wso2.carbon.databridge.agent.thrift:4.2.1.
    permission java.lang.RuntimePermission
    "accessClassInPackage.com.sun.xml.internal.bind.v2.runtime.reflect";

    // Required by org.wso2.carbon.ndatasource.core:4.2.0. This is only
    // necessary after adding above permission.
    permission java.lang.RuntimePermission
    "accessClassInPackage.com.sun.xml.internal.bind";
};
```

#### Using Asymmetric Encryption

WSO2 products use asymmetric encryption by default for the authentication and protection of data. In asymmetric encryption, keystores (with private keys and public key certificates) and truststores (with only public key certificates) are created and stored for a 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 and truststores, and how they are used in WSO2 products.

- Understanding keystores and truststores
- Usage of keystores in WSO2 products
- Recommendations for setting up keystores in WSO2 products

**Understanding keystores and truststores**

A keystore is a repository (protected by a password) that holds the keys and certificates of a trust chain. There may be multiple trust chains (i.e., multiple keys with corresponding certificates) in one keystore. You use these artifacts for security purposes such as protecting sensitive information and establishing trust between your server and outside parties that connect to the server. The usage of keys and certificates contained in a keystore are explained below.

**Keys:** According to public-key cryptography, the concept of a key pair (public key and the corresponding private key) is used for protecting sensitive information and for authenticating the identity of external parties that communicate with your server. For example, the information that is encrypted in your server using the public key can only be decrypted using the corresponding private key. Therefore, if any party wants to decrypt this encrypted data, they should have the corresponding private key, which is usually kept as a secret (not publicly shared).
Digital certificate: When there is a key pair, it is also necessary to have a digital certificate to verify the identity of the keys. Typically, the public key of a key pair is embedded in this digital certificate, which also contains additional information such as the owner, validity, etc. of the keys. For example, if an external party wants to verify the integrity of data or validate the identity of the signer (by validating the digital signature), it is necessary for them to have this digital certificate of the signer.

Trusted certificates and certificate signing authorities: To establish trust, the digital certificate containing the public key should be signed by a trusted certificate signing authority (CA). You can generate self-signed certificates for the public key (thereby creating your own certifying authority), or you can get the certificates signed by the digital certificate of an external CA. When the certificate is signed by a reputed CA, all the parties that trust this CA will also trust the certificates signed by them. To establish maximum trust, it is important to have a root CA directly sign your public key certificate, or else, you can have an intermediate CA certificate (which is already signed by a root CA) sign your certificate. Therefore, in the later case, there can be a chain of CAs involved in signing your public key certificate. However, note that both types of public key certificates (self-signed or CA-signed) can be effectively used depending on the sensitivity of the information that is protected by the keys.

In summary, each trust chain entry in a keystore contains the following:

- A private key protected by a password.
- A digital certificate in which the public key (corresponding to the private key) is embedded.
- Additionally, if this public key certificate is not self-signed but signed by a Certificate Signing Authority (CA), an additional set of certificates (of the CAs involved in the signing process) will also be included. This may be just one additional certificate if the immediate CA certificate that was used to sign the public key certificate is of a Root CA. If the immediate certificate is not of a root CA, all the certificates of the intermediate CAs should also be included in the keystore.

The usage of a truststore in WSO2 products aligns with this concept of trust explained above. A truststore is just another repository that is protected by a password (similar to a keystore), which stores digital certificates. These certificates can be either of the following:

- Certificates of trusted third parties with which a software system intends to communicate directly.
- Certificates of reputed certificate signing authorities (CA) that can be used to validate the identity of untrusted third parties that are being contacted. For example, consider a scenario where the exact certificate of the third party that the WSO2 server is attempting to contact is not in the truststore. In this scenario, if the third party has a CA-signed certificate and one of the certificates of its trust chain is already included in the WSO2 server's truststore, the trust is automatically granted and a successful SSL connection is established between the WSO2 server and the third party.

Default keystore and truststore in WSO2 products

All WSO2 products are by default shipped with a keystore file and truststore file (stored in the `<PRODUCT_HOME>/repository/resources/security/directory`):

- **wso2carbon.jks**: This is the default keystore, which contains a private key and the self-signed public key certificate.
- **client-truststore.jks**: This is the default truststore, which contains certificates of reputed CAs that can validate the identity of third party systems. This truststore also contains the self-signed certificate of the default wso2carbon.jks keystore.

Usage of keystores in WSO2 products

In WSO2 products, asymmetric encryption is used by default for the following purposes:

- Authenticating the communication over Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocols.
- Encrypting sensitive data such as plain-text passwords found in both product-level and product feature-level configurations/configuration files using secure vault.
- Encrypting and signing SOAP messages using WS-Security.

The default keystore that is shipped with a WSO2 product (`wso2carbon.jks`) is by default configured for all of the above purposes. However, in a production environment, it is advised to set up several different keystores with separate trust chains for the above use cases.

Recommendations for setting up keystores in WSO2 products

Follow the recommendations given below when you set up your keystores.

- Maintain one primary keystore for encrypting sensitive internal data such as admin passwords and any other sensitive information found at both product-level and product feature-level configurations/configuration files. Note that the primary keystore will also be used for signing messages when the product communicates with external parties (such SAML, OIDC id_token signing).

In new versions of WSO2 products, you can have separate keystores for encrypting sensitive information for internal data encryption as a recommended practice. See Configuring Keystores in WSO2 Products for details.

- Maintain another secondary keystore, containing the server's public key certificate for authenticating communication over SSL/TLS (for both Tomcat and Axis2 level HTTP connections).
- If your deployment contains multiple products, instances of the same product must use the same keystore for SSL. Different products can use different keystores for SSL, but it is not mandatory.
- It is recommended to use a CA-signed keystore for SSL communication; however, this is not mandatory. Even a self-signed certificate may suffice if it can be trusted by the clients.
- The keystore used for SSL must contain the same password for the Keystore and private key due to a Tomcat limitation.
The primary keystore used for admin passwords and other data encryption requirements can be a self-signed one. There is no value added by using a CA-signed keystore for this purpose as it is not used for any external communication.

The primary keystore’s public key certificate must have the **Data Encryption** key usage to allow direct encipherment of raw data using its public key. This key usage is already included in the self-signed certificate that is included in the default `wso2carbon.jks` keystore. If the **Data Encryption** key usage is not included in your public key certificate, the following error can occur when you attempt data encryption:

```
Exception in thread "main" org.wso2.ciphertool.CipherToolException:
    Error initializing Cipher at
    org.wso2.ciphertool.CipherTool.handleException(CipherTool.java:861) at
    org.wso2.ciphertool.CipherTool.initCipher(CipherTool.java:202) at
    org.wso2.ciphertool.CipherTool.main(CipherTool.java:80) Caused by:
    java.security.InvalidKeyException: Wrong key usage at
    javax.crypto.Cipher.init(DashoA13..) at
    javax.crypto.Cipher.init(DashoA13..) at
    org.wso2.ciphertool.CipherTool.initCipher(CipherTool.java:200) ... 1
    more
```

Optionally, you can set up separate keystores for message-level data encryption in WS-Security.

If you already have the required keystores for your product, you can generate CA-signed certificates and import them into the keystores. It is not recommended to create new keystores for the purpose of replacing the certificates in the keystore. See **Adding CA-signed certificates to keystores** for instructions.

For information on creating new keystores with the required certificates, see **Creating New Keystores**, and for information on how to update configuration files in your product with keystore information, see **Configuring Keystores in WSO2 Products**.

### Creating New Keystores

WSO2 Carbon-based products are shipped with a default keystore named `wso2carbon.jks`, which is stored in the `<PRODUCT_HOME>/repository/resources/security` directory. This keystore comes with a private/public key pair that is used for all purposes, such as encrypting sensitive information, communicating over SSL and for message encryption/signing purposes in WS-Security. You can either use one new keystore for all purposes in your product, or you can create multiple keystores for each purpose.

Let's get started with creating new keystores.

- **Before you begin**
- **Creating a new keystore**
  - Creating a keystore using an existing certificate
  - Creating a keystore using a new certificate
- **Adding CA-signed certificates to keystores**
  - Step 1: Generating a CA-signed certificate
  - Step 2: Importing certificates to the keystore
  - Step 3: Importing certificates to the truststore
- **What's next?**

#### Before you begin

Be sure to go through the recommendations for setting up keystores in WSO2 products to understand the types of keystores you need.

#### Creating a new keystore

There are two ways to create keystores for a WSO2 product. You can either generate a keystore using an already existing public key certificate (CA-signed), or you can create the public key certificate at the time of generating the keystore. See the instructions given below.

**Important!**

If you are creating a new keystore for data encryption, be sure to acquire a public key certificate that contains the **Data Encryption** key usage as explained in the keystore recommendations. Otherwise, the following error can occur when you attempt data encryption:
Creating a keystore using an existing certificate

Secure Sockets Layer (SSL) is a protocol that is used to secure communication between systems. This protocol uses a public key, a private key and a random symmetric key to encrypt data. As SSL is widely used in many systems, certificates may already exist that can be reused. In such situations, you can use an already existing CA-signed certificate to generate your keystore for SSL by using OpenSSL and Java keytool.

1. First, you must export certificates to the **PKCS12/PFX** format. Give strong passwords whenever required.

   In WSO2 products, it is a must to have the same password for both the keystore and private key.

   Execute the following command to export the entries of a trust chain into a keystore of .pfx format:

   ```
   openssl pkcs12 -export -in <certificate file>.crt -inkey <private>.key -name "<alias>" -certfile <additional certificate file> -out <pfx keystore name>.pfx
   ```

   2. Convert the **PKCS12/PFX** formatted keystore to a Java keystore using the following command:

   ```
   keytool -importkeystore -srckeystore <pkcs12 file name>.pfx -srcstoretype pkcs12 -destkeystore <JKS name>.jks -deststoretype JKS
   ```

   Now you have a keystore with a CA-signed certificate.

Creating a keystore using a new certificate

You can follow the steps in this section to create a new keystore with a private key and a new public key certificate. We will be using the keytool that is available with your JDK installation. Note that the public key certificate we generate for the keystore is **self-signed**. Therefore, if you need a public key certificate that is CA-signed, you need to generate a CA-signed certificate and import it to the keystore as explained in the next section. Alternatively, you can choose the option of generating a new keystore using a CA-signed public certificate as explained previously.

1. Open a command prompt and go to the `<PRODUCT_HOME>/repository/resources/security` directory. All keystores should be stored here.
2. Create the keystore that includes the private key by executing the following command:
keytool -genkey -alias newcert -keyalg RSA -keysize 2048 -keystore newkeystore.jks -dname "CN=<testdomain.org>,
OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass mypassword -keypass mypassword

This command will create a keystore with the following details:
- Keystore name: newkeystore.jks
- Alias of public certificate: newcert
- Keystore password: mypassword
- Private key password: mypassword (this is required to be the same as keystore password)

Note that if you did not specify values for the `keypass` and the `storepass` in the above command, you will be asked to give a value for the `storepass` (password of the keystore). As a best practice, use a password generator to generate a strong password. You will then be asked to enter a value for `keypass`. Click Enter because we need the same password for both the keystore and the key. Also, if you did not specify values for `-dname`, you will be asked to provide those details individually.

3. Open the <PRODUCT_HOME>/repository/resources/security/ directory and check if the new keystore file is created. Make a backup of it and move it to a secure location. This is important as it is the only place with your private key.

You now have a keystore (.jks file) with a private key and a self-signed public key certificate.

**Adding CA-signed certificates to keystores**

Now, let's look at how you can get a CA-signed certificate for your keystores. **Note** that you do not need to create a new keystore everytime you need add a CA-signed certificate.

**Step 1: Generating a CA-signed certificate**

First, you need to generate a certificate signing request (CSR) for your keystore (.jks file). This CSR file can then be certified by a certification authority (CA), which is an entity that issues digital certificates. These certificates certify the ownership of a public key.

1. Execute the following command to generate the CSR:

   keytool -certreq -alias certalias -file newcertreq.csr -keystore newkeystore.jks

   As mentioned before, use the same alias that you used during the keystore creation process.

   You will be asked to give the keystore password. Once the password is given, the command will output the newcertreq.csr file to the <PRODUCT_HOME>/repository/resources/security/ directory. This is the CSR that you must submit to a CA.

   2. You must provide this CSR file to the CA. For testing purposes, try the 90 days trial SSL certificate from Comodo.

   It is preferable to have a wildcard certificate or multiple domain certificates if you wish to have multiple subdomains like gateway.sample domain.org , publisher.sampledomain.org , identity.sampledomain.org , etc., for the deployment. For such requirements, you must modify the CSR request by adding subject alternative names. Most of the SSL providers give instructions to generate the CSR in such cases.

   3. After accepting the request, a signed certificate is provided along with a root certificate and several intermediate certificates (depending on the CA) as a bundle (.zip file).

   **Sample certificates provided by the CA (Comodo)**

   | The Root certificate of the CA: AddTrustExternalCARoot.crt |
   | Intermediate certificates: COMODORSAddTrustCA.crt, COMODORSDomainValidationSecureServerCA.crt |
   | SSL Certificate signed by CA: test_sampleapp_org.crt |

**Step 2: Importing certificates to the keystore**
Follow the steps given below to import the CA-signed certificate to your keystore.

1. Before importing the CA-signed certificate to the keystore, you must add the root CA certificate and the two (related) intermediate certificates by executing the commands given below. Note that the sample certificates given above are used as examples.

```
keytool -import -v -trustcacerts -alias ExternalCARoot -file AddTrustExternalCARoot.crt -keystore newkeystore.jks -storepass mypassword
keytool -import -v -trustcacerts -alias TrustCA -file COMODORSAddTrustCA.crt -keystore newkeystore.jks -storepass mypassword
keytool -import -v -trustcacerts -alias SecureServerCA -file COMODORSADomainValidationSecureServerCA.crt -keystore newkeystore.jks -storepass mypassword
```

2. After you add the root certificate and all other intermediate certificates, add the CA-signed SSL certificate to the keystore by executing the following command:

```
keytool -import -v -alias newcert -file <test_sampleapp_org.crt> -keystore newkeystore.jks -keypass mypassword -storepass mypassword
```

Now you have a Java keystore, which includes a CA-signed public key certificate that can be used for SSL in a production environment. Next, you may need to add the same CA-signed public key certificate to the `client-truststore.jks` file. This will provide security and trust for backend communication/inter-system communication of WSO2 products via SSL.

**Step 3: Importing certificates to the truststore**

In SSL handshake, the client needs to verify the certificate presented by the server. For this purpose, the client usually stores the certificates it trusts, in a trust store. To enable secure and trusted backend communication, all WSO2 products are shipped with a trust store named `client-truststore.jks`, which resides in the same directory as the default keystore (`<PRODUCT_HOME>/repository/resources/security/`). Follow the steps given below to import the same CA-signed public key certificate (which you obtained in the previous step) into your WSO2 product's default truststore (`client-truststore.jks`).

1. Get a copy of the `client-truststore.jks` file from the `<PRODUCT_HOME>/repository/resources/security/` directory.
2. Export the public key from your `.jks` file using the following command.

```
keytool -export -alias certalias -keystore newkeystore.jks -file <public key name>.pem
```

3. Import the public key you extracted in the previous step to the `client-truststore.jks` file using the following command.

```
keytool -import -alias certalias -file <public key name>.pem -keystore client-truststore.jks -storepass wso2carbon
```

Note that 'wso2carbon' is the keystore password of the default client-truststore.jks file.
Now, you have an SSL certificate stored in a Java keystore and a public key added to the `client-truststore.jks` file. Note that both these files should be in the `<PRODUCT_HOME>/repository/resources/security/` directory. You can now replace the default `wso2carbon.jks` keystore in your product with the newly created keystore by updating the relevant configuration files in your product.

For information on the concepts of keystores and about how keystores are used in WSO2 products, see Using Asymmetric Encryption.

### What’s next?

Once you have created a new keystore in your product as explained above, update the relevant configuration files as explained in Configuring Keystores in WSO2 Products.

### Renewing a CA-Signed Certificate in a Keystore

A digital certificate has a validity period, after which the certificate expires. Once a certificate expires, it is no longer valid, and it can cause the client-server communication to fail at the SSL handshake level. Therefore, it is important to plan certificate renewal ahead of time. Neglecting certificate renewal can eventually lead to a catastrophic situation such as major service outage.

Following are the high level steps you need to follow to renew an expired certificate in a keystore.

- **Step 1: Check the validity period of the certificate**
- **Step 2: Generate a certificate signing request**
- **Step 3: Import the new certificate to a keystore**

### Following are a few important point to keep in mind when you are renewing an expired certificate:

- Use the same certificate authority that you used when you first got the public certificate. If you use a different certificate authority for certificate renewal, you will have to import the new CA-certificate as well as the intermediate certificates to the keystore and the client’s trust store.
- If the certificate authority’s certificate is not in the keystore, you will get the following error when you try to import the CA-signed certificate to the keystore:

  ```
  keytool error: java.lang.Exception: Failed to establish chain from reply
  ```

  To overcome the above error, be sure to first import the CA-signed certificate as well as the intermediate certificates to the keystore in the correct order.

Now let’s take a look at each high level step in detail.

### Step 1: Check the validity period of the certificate

Follow one of the steps below to view the validity period of a certificate:

- If you have a public hostname, go to [https://www.sslshopper.com/ssl-checker.html](https://www.sslshopper.com/ssl-checker.html) and specify the hostname of your server. SSL hopper lists all the information about the server certificate.
- If you have a java keystore, execute the following keytool command to view the certificate information:

  ```
  keytool -list -keystore <keystore_name.jks> -alias <cert_alias> -v
  ```

This prompts for the keystore password. Once you specify the password, you can view the certificate information in a human readable format where the validity period is displayed as follows:

```
```
If you have the certificate file, execute the following OpenSSL command:

```bash
x509 -in <certname.cer> -text -noout
```

This displays the validity as follows:

```
Validity
Not Before: Jun 18 13:56:25 2017 GMT
Not After : Jun 19 13:56:25 2027 GMT
```

If it is a website, you can view the certificate information via the browser. All major browsers provide the capability to view certificate information.

Once you view the validity period of a certificate and if it says that the certificate is about to expire or has already expired, the next step you should generate a Certificate Signing Request (CSR) and get a new certificate generated from the CA.

**Step 2: Generate a certificate signing request**

Depending on the type of keystore you have, follow one of the steps below to generate a CSR:

- If you have a Java keystore, execute the following command:

  ```bash
  keytool -certreq -alias <cert_alias> -file <CSR.csr> -keystore <keystore_name.jks>
  ```

  If you have the private key and public key, execute the following command:

  ```bash
  openssl x509 -x509toreq -in <cert_name.crt> -out <CSR.csr> -signkey <private_key.key>
  ```

  Once you generate the CSR, you need to submit the CSR to your certificate authority to get a new CA-signed certificate. For testing purposes you can go to [http://www.getacert.com/signacert.html](http://www.getacert.com/signacert.html) and submit your CSR to obtain a new CA-signed certificate for free.

  After you obtain a new certificate, you have to import the new certificate to a keystore if you are using a Java keystore.

**Step 3: Import the new certificate to a keystore**
Execute the following command to import a new certificate to a keystore:

```bash
keytool -import -v -trustcacerts -alias <current_alias> -file <ca_signed_cert.cer> -keystore <keystore_name.jks>
```

If you want to view information related to the renewed certificate, execute the following keytool command:

```bash
keytool -list -keystore <keystore_name.jks> -alias <cert_alias> -v
```

Configuring Keystores in WSO2 Products

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 keystores work. Note that keystores are used for multiple functions in WSO2 products, which includes authenticating communication over SSL/TLS, encrypting passwords and other confidential information in configuration files etc. Therefore, you must update the specific configuration files with the updated keystore information. For example, you may have separate keystores for the purpose of encrypting passwords in configuration files, and for authenticating communication over SSL/TLS.

The `wso2carbon.jks` keystore file, which is shipped with all WSO2 products, is used as the default keystore for all functions. However, in a production environment, it is recommended to create new keystores with new keys and certificates.

Please note that in WSO2 IoT Server and WSO2 Enterprise Integrator the `<PRODUCT_HOME>/repository/conf` directory is in the following location: `<PRODUCT_HOME>/conf`

If you want an easy way to locate all the configuration files that have references to keystores, you can use the `grep` command as follows:

1. Open a command prompt and navigate to the `<PRODUCT_HOME>/repository/conf/` directory where your product stores all configuration files.
2. Execute the following command: `grep -nr ".jks"`.

The configuration files and the keystore files referred to in each file are listed out. See an example of this below.

```
./axis2/axis2.xml:260:
<Location>repository/resources/security/wso2carbon.jks</Location>
./axis2/axis2.xml:431:
<Location>repository/resources/security/wso2carbon.jks</Location>
./carbon.xml:316:
<Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
./carbon.xml:332:
<Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
./identity.xml:180:
<Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
./security/secret-conf.properties:21:#keystore.identity.location/repository/resources/security/wso2carbon.jks
```

See the following for details:

- Before you begin
- Configuring the primary keystore
- Configuring a separate keystore for encrypting data in internal data stores
• Configuring a secondary keystore (for SSL connections)
• Configuring a keystore for Java permissions
• Configuring keystores for WS-Security
• What's next?

**Before you begin**

• Be sure to go through the recommendations for setting up keystores in WSO2 products to understand the various keystores you will need.
• If you haven’t already created the keystores required for your system, see the instructions for creating new keystores.

**Configuring the primary keystore**

The primary keystore of WSO2 products are configured by the `<KeyStore>` element in the `carbon.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/` directory). This keystore is used for the following functions in WSO2 products by default.

- Encrypting/decrypting passwords and other confidential information, which are maintained in various configuration files as well as internal data stores. Note that you also have the option of separating the keystore for encrypting information in internal data stores.
- Signing messages when the WSO2 product communicates with external parties (such as SAML, OIDC id_token signing).

The default configuration is shown below.

```xml
<KeyStore>
  <Location>${carbon.home}/resources/security/wso2carbon.jks</Location>
  <Type>JKS</Type>
  <Password>wso2carbon</Password>
  <KeyAlias>wso2carbon</KeyAlias>
  <KeyPassword>wso2carbon</KeyPassword>
</KeyStore>

<TrustStore>
  <!-- trust-store file location -->
  <Location>${carbon.home}/repository/resources/security/client-truststore.jks</Location>
  <!-- trust-store type (JKS/PKCS12 etc.) -->
  <Type>JKS</Type>
  <!-- trust-store password -->
  <Password>wso2carbon</Password>
</TrustStore>
```

**Configuring a separate keystore for encrypting data in internal data stores**

This feature is available via the WUM update 2792 released on the 8th of July 2018 for the following product versions:

- WSO2 Identity Server 5.5.0
- WSO2 API Manager 2.2.0
- WSO2 Data Analytics Server 3.2.0
- WSO2 Enterprise Integrator 6.2.0

This is available as part of the newly introduced Crypto Service. It is an extensible framework that facilitates the cryptography needs of WSO2 products.

Currently, the primary keystore configured by the `<Security>/<KeyStore>` element in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file is used for internal data encryption (encrypting data in internal data stores and configuration files) as well as for signing messages that are communicated with external parties. However, it is sometimes a common requirement to have separate keystores for communicating messages with external parties (such as SAML, OIDC id_token signing) and for encrypting information in internal data stores. This is because, for the first scenario of signing messages, the keystore certificates need to be frequently renewed. However, for encrypting information in internal data stores, the keystore certificates should not be changed frequently because the data that is already encrypted will become unusable every time the certificate changes.

This feature allows you to create a separate keystore for encrypting data in internal data stores. Follow the instructions given below.

**Warning:** Using a totally new keystore for internal data encryption in an existing deployment will make already encrypted data unusable. In such cases, an appropriate data migration effort is needed.

1. Enable the Crypto Service by adding the following configuration block to the `<PRODUCT_HOME>/repository/conf/carbon.xml` file.
1. Configure the new keystore by adding the following configuration block inside the `<CryptoService>` tag in the `<Security>` tag in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file.

```xml
<CryptoService>
  <Enabled>true</Enabled>
  <InternalCryptoProviderClassName>org.wso2.carbon.crypto.provider.KeyStoreBasedInternalCryptoProvider</InternalCryptoProviderClassName>
  <ExternalCryptoProviderClassName>org.wso2.carbon.core.encryption.KeyStoreBasedExternalCryptoProvider</ExternalCryptoProviderClassName>
  <KeyResolvers>
    <KeyResolver className="org.wso2.carbon.crypto.defaultProvider.resolver.ContextIndependentKeyResolver" priority="-1"/>
  </KeyResolvers>
</CryptoService>
```

2. The values of the properties such as passwords must be changed based on the keystore.

Note: The values of the properties such as passwords must be changed based on the keystore.

```xml
<InternalKeyStore>
  <Location>${carbon.home}/repository/resources/security/internal.jks</Location>
  <Type>JKS</Type>
  <Password>wso2carbon</Password>
  <KeyAlias>wso2carbon</KeyAlias>
  <KeyPassword>wso2carbon</KeyPassword>
</InternalKeyStore>
```

**Configuring a secondary keystore (for SSL connections)**

The `catalina-server.xml` file stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory should be updated with the keystore used for certifying SSL connections to Carbon servers. Given below is the default configuration in the `catalina-server.xml` file, which points to the default keystore in your product.

```xml
<KeyStore>
  <keystoreFile>${carbon.home}/repository/resources/security/wso2carbon.jks</keystoreFile>
  <keystorePass>wso2carbon</keystorePass>
</KeyStore>
```

If you are using WSO2 API Manager (WSO2 APIM) or the ESB of WSO2 Enterprise Integrator (WSO2 EI), you need to update the keystore information in the `axis2.xml` file (in addition to the `catalina-server.xml` file explained above). **Note** that the `axis2.xml` file is stored in the `<APIM_HOME>/repository/conf/axis2/` directory for APIM, and the `<EI_HOME>/conf/axis2/` directory for the ESB of WSO2 EI.
Configuring a keystore for Java permissions

The Java Security Manager is used for defining various security policies that prevent untrusted code from manipulating your system. Enabling the Java Security Manager for WSO2 products will activate the Java permissions that are in the <PRODUCT_HOME>/repository/conf/sec.policy file. Administrators can modify this file to change the Java security permissions as required and grant various application-level permissions to the signed and trusted code using Java.

If you are granting specific Java-level permissions to some signed code, you should import the public key certificate of the signer as a trusted certificate to one of your keystores. You must then update the sec.policy file with the keystore path and the alias of the certificate as shown below.

```
keystore
"file:$user.dir/repository/resources/security/wso2carbon.jks", "JKS";
```

Following is the default keystore configuration in the sec.policy file, which grants all Java-level permissions to the code signed by the certificate that uses the 'wso2carbon' alias.

```
grant signedBy "wso2carbon" {
    permission java.security.AllPermission;
};
```

Configuring keystores for WS-Security
If there are WS-Security scenarios implemented in your WSO2 product, you can use separate keystores for these scenarios. WS-Security is used for proxy services and data services in the ESB. See the documentation of WSO2 Enterprise Integrator (WSO2 EI) for instructions on applying security policies for proxy services, and data services.

**What’s next?**

Some WSO2 products will use keystore for more use cases than the ones listed above. See the documentation for your WSO2 product for instructions.

**Managing Keystores with the UI**

**Important!**

In WSO2 products (based on Carbon 4.4.0 and later versions), you can use the management console to view details of your keystores. To do this, you must first upload the relevant keystore from the UI, and then view details such as the certificates available in the keystore. See details below.

**Prerequisites**

All the required keystore files should first be created and stored in the `<PRODUCT_HOME>/repository/resources/security/` directory. For information on how to create new keystore files see Creating New Keystores, and for information on how to update configuration files in your product with keystore information, see Configuring Keystores in WSO2 Products.

The default `wso2carbon.jks` keystore cannot be deleted.

**Viewing keystore details from the UI**

Follow the instructions below to upload a keystore file to the management console.

1. Log in to the WSO2 product with your username and password.
2. Go to the `Configure` tab and click `Key Stores`.
3. The `Key Store Management` page appears. Click the `Add New Key store` link.
4. Specify the `Provider` and the `Keystore Password`, which points to the password required to access the private key.
5. In the `Keystore Type` field, specify whether the keystore file you are uploading is JKS or PKCS12.
   - **JKS** (Java Key Store): Allows you to read and store key entries and certificate entries. However, the key entries can store only private keys.
   - **PKCS12** (Public Key Cryptography Standards): Allows you to read a keystore in this format and export the information from that keystore. However, you cannot modify the keystore. This is used to import certificates from different browsers into your Java Key store.
6. Click `Next` and provide the `Private Key Password`.
7. Click `Finish` to add the new keystore to the list.
8. The keystore file is saved to the registry of your product. To see the registry path click `Registry Browse` on the navigator, and go to `_system governance repository security key-stores`. The keystores added from the UI will be listed here.

Follow the instructions below to view details of the keystore you uploaded.

1. Log in to the WSO2 product with your user name and password.
2. Go to the `Configure` tab and click `Key Stores`.
3. The `Key Store Management` page appears. All the keystores that are currently added to the product will be listed here.
4. Click `View` in the list of actions. The `View Key Store` screen shows information about the available certificates.
4. It also displays information about private key certificates:

Certificate of the Private Key

<table>
<thead>
<tr>
<th>Alias</th>
<th>IssuerDN</th>
<th>NotAfter</th>
<th>NotBefore</th>
<th>SerialNumber</th>
<th>SubjectDN</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>wso2carbon</td>
<td>13/02/2005</td>
<td>19/02/2010</td>
<td>C=US</td>
<td>1286633496</td>
<td>CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US</td>
<td>3</td>
</tr>
</tbody>
</table>

Import File Finish

5. Click Finish to go back to the Key Store Management screen.

Using Symmetric Encryption

The capability of using symmetric encryption was introduced by the Carbon 4.4.3 release. Therefore, note that this feature is only applicable to products that are based on Carbon 4.4.3 or later versions.

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.

Follow the steps given below to enable symmetric encryption.

1. Open the carbon.xml file from the <PRODUCT_HOME>/repository/conf directory.
2. Add the following properties:

   ```xml
   <SymmetricEncryption>
   <IsEnabled>true</IsEnabled>
   <Algorithm>AES</Algorithm>
   <SecureVaultAlias>symmetric.key.value</SecureVaultAlias>
   </SymmetricEncryption>
   ```

   - The IsEnabled property is used to set symmetric encryption to 'true' or 'false'.
   - The Algorithm property specifies the symmetric key algorithm used.
   - The SecureVaultAlias property is used to specify the secret alias if secure vault has been used to encrypt the secret key.

3. Create a file named 'symmetric.key.properties' in the <PRODUCT_HOME>/repository/resources/security folder and enter the symmetric key using the symmetric.key property. See the following example where a plain text key is entered in the symmetric.key.properties file:
If Secure Vault has been used for encrypting the symmetric key, this value will be replaced by the secret alias as shown below. For detailed instructions on how the secret key can be encrypted using Secure Vault, see Encrypting Passwords with Cipher Tool.

```
symmetric.key=samplekeyvalue
```

```
symmetric.key=secretAlias:symmetric.key.value
```

Mitigating Cross Site Request Forgery Attacks

The following sections 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
- Configuring applications in WSO2 product to mitigate CSRF attacks
  - Securing web applications
  - Securing Jaggery applications

How can CSRF attacks be harmful?

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. The session in which you logged in to the web application on the browser is used to bypass the authentication step during this attack. If you are already authenticated on the website, the site or application cannot distinguish between a forged request and a legitimate request. Therefore, it is also known as "session riding".

The attack includes maliciously tricking you to click a URL or HTML content that will consequently send a request to the website. For example:

- You send a request to an online banking application to transfer $100 to another bank account.
- An example URL including the parameters (i.e. account number and transfer amount) of a request is similar to the following: `https://bank.com/transfer.do?acct=10220048&amount=100`
- The attacker uses this same URL by replacing the actual account number with a malicious account number. Then the attacker disguises this URL by including it in a clickable image and sends it to you in an email with other content.
- You may unknowingly click on this URL, which will send a transfer request to the bank to transfer money to the malicious bank account.

Mitigating CSRF attacks

**OWASP CSRFGuard** is an OWASP flagship project that provides synchronizer token pattern-based CSRF protection in a comprehensive and customizable manner. You can use the best practices and configuration recommendations of OWASP CSRFGuard to mitigate CSRF attacks in applications hosted on the WSO2 platform. Fine-tuned configuration values of CSRFGuard increases security, based on the security requirements of the specific application.

CSRFGuard offers complete protection over CSRF scenarios by covering HTTP POST, HTTP GET as well as AJAX-based requests. You can protect forms based on HTTP POST and HTTP GET methods by injecting CSRF tokens into the “action” of the form, or by embedding a token in a hidden field. Include these tokens manually using provided JSP tag library or by using a JavaScript based automated injection mechanism. AJAX requests are protected by injecting an additional header, which contains a CSRF token.

Configuring applications in WSO2 product to mitigate CSRF attacks

**Before you begin**, note the following:

- If your WSO2 product is based on Carbon 4.4.6 or a later version, the configurations for mitigating CSRF attacks are enabled by default for all the applications that are built into the product. Therefore, you need to apply these configurations manually, only if you have any custom applications deployed in your product.
- If your WSO2 product is based on a Carbon version prior to version 4.4.6, the configurations for mitigating CSRF attacks should be applied to all applications manually.
- **Important!** Some updates of JDK 1.8 (for example, JDK1.8.0.151) are affected by a known issue related to GZIP decoding, which may prevent these CSRF-related configurations from working for your product. Therefore, until this issue is fixed, we recommend one of the following approaches:
  - Be sure that your product is running on JDK1.8.0.144 or JDK1.8.0.077. We have verified that these JDK versions are not...
follow the steps below to secure web applications.

1. Add the following configurations in the `web.xml` file of your application.

```
<param name="org.apache.coyote.http11.V1_1" value="false"/>
```

See the following for instructions on manually updating CSRF configurations in WSO2 products:

- **Securing web applications**
- **Securing Jaggery applications**
2. Include the following JavaScriptServlet as the first JavaScript inclusion of the `<head>` element, in the HTML template of all pages of the application that you need to protect.
3. Create a CSRF configuration properties file (e.g. abc.properties) within your application, and copy the content in the `<CARBON_HOME>repository/conf/security/Owasp.CsrfGuard.Carbon.properties` file to it.

4. Use the `org.owasp.csrfguard.unprotected.prefix` in the configuration property keys, for the relevant patterns that you need to exclude from CSRF protection. For example:

   ```
   org.owasp.csrfguard.unprotected.Default=%servletContext%/exampleAction
   org.owasp.csrfguard.unprotected.Default_1=%servletContext%/exampleAction
   org.owasp.csrfguard.unprotected.Example=%servletContext%/exampleAction/*
   org.owasp.csrfguard.unprotected.ExampleRegEx=^%servletContext%/.*Public\.do$
   ```

5. Change the following configuration properties, to further enhance security. You may need justifiable application level requirements to change them since they will affect performance or user experience.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.owasp.csrfguard.PRNG=SHA1PRNG</td>
<td>Defines the hashing algorithm used to generate the CSRF token.</td>
</tr>
<tr>
<td>org.owasp.csrfguard.TokenLength=32</td>
<td>Defines the length of the CSRF token.</td>
</tr>
<tr>
<td>org.owasp.csrfguard.action.Invalidate=org.owasp.csrfguard.action.Invalidate</td>
<td>Invalidates the user session, if a CSRF attack attempt was blocked by CSRFGuard.</td>
</tr>
</tbody>
</table>

### Securing Jaggery applications

Follow the steps below to secure Jaggery applications.

1. Add the following configurations in the `jaggery.conf` file of your application.

    - `<html>
      - <head>
      - `<script type="text/javascript" src="/csrf.js"></script>
      - <!-- other JavaScript inclusions should follow "csrf.js" inclusion -->
      - `<script type="text/javascript" src="/main.js"></script>
      - </head>
      - <body>
      - ...
      - </body>
      - </html>`
"listeners" : [
  {
    "class" : "org.owasp.csrfguard.CsrfGuardServletContextListener"
  },
  {
    "class" : "org.owasp.csrfguard.CsrfGuardHttpSessionListener"
  }
],
"servlets" : [
  {
    "name" : "JavaScriptServlet",
    "class" : "org.owasp.csrfguard.servlet.JavaScriptServlet"
  }
],
"servletMappings" : [
  {
    "name" : "JavaScriptServlet",
    "url" : "/csrf.js"
  }
],
"contextParams" : [
  {
    "name" : "Owasp.CsrfGuard.Config",
    "value" : "/repository/conf/security/Owasp.CsrfGuard.dashboard.properties"
  }
]

2. Include the following JavaScriptServlet as the first JavaScript inclusion of the <head> element in the HTML template of all pages of the application that you need to protect.

```html
<html>
<head>
  ...
  <script type="text/javascript" src="/csrf.js"></script>
  <!-- other JavaScript inclusions should follow "csrf.js" inclusion -->
  <script type="text/javascript" src="/main.js"></script>
  ...
</head>
<body>
  ...
</body>
</html>
```

3. Create a CSRF configuration properties file (e.g. abc.properties) within your application, and copy the content in the `<CARBON_HOME>repository/conf/security/Owasp.CsrfGuard.Carbon.properties` file to it.

4. Use the `org.owasp.csrfguard.unprotected.` prefix in the configuration property keys, for the relevant patterns that you need to exclude from CSRF protection. For example:
5. Change the following configuration properties, to further enhance security. You may need justifiable application level requirements to change them since they will affect performance or user experience.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.owasp.csrfguard.PRNG=SHA1PRNG</td>
<td>Defines the hashing algorithm used to generate the CSRF token.</td>
</tr>
<tr>
<td>org.owasp.csrfguard.TokenLength=32</td>
<td>Defines the length of the CSRF token.</td>
</tr>
<tr>
<td>org.owasp.csrfguard&amp;action.Invalidate=org.owasp.csrfguard.action.Invalidate</td>
<td>Invalidates the user session, if a CSRF attack attempt was blocked by CSRFGuard.</td>
</tr>
</tbody>
</table>

**Mitigating Cross Site Scripting Attacks**

The following sections describe the impact of the XSS attack and the approaches you can use to mitigate it. Note that XSS attacks are prevented on the latest WSO2 products by default. This is due to output encoding of the displaying values. However, if additional protection is required, an input validation valve can be configured as explained below.

- How can XSS attacks be harmful?
- Mitigating XSS attacks

**How can XSS attacks be harmful?**

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. XSS Attackers can gain elevated access privileges to sensitive page content, session cookies, and a variety of other information with respect to web applications that are maintained by the web browser on behalf of the user.

**Mitigating XSS attacks**

You can use the following approach to mitigate XSS attacks.

**Mitigating using the XSS Valve**

The XSS Valve acts as a filter to differentiate between the malicious scripts from the legitimate scripts by carrying out a specific validation on the URL patterns.

**Configuring the XSS Valve**

1. Open the `<PRODUCT_HOME>/repository/conf/carbon.xml` file and add the following code snippet under the `<Security>` tag.
1. Add the following configuration within the `<Hosts>` element of the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.

```xml
<Valve className="org.wso2.carbon.ui.valve.XSSValve"/>
```

3. Restart the product server.

### Enabling HostName Verification

The possibility to configure hostname verification is available for WSO2 products from Carbon 4.4.6 onwards. If hostname verification is disabled for your product, the hostnames (that are accessed by a particular client) will not be verified against the hostnames specified in the product’s SSL certificate. If hostname verification is enabled, you will be able to configure how hostnames are filtered during the verification process.

See the following topics for details:

- Prerequisites
- Configuring hostname verification (Carbon 4.4.6 to Carbon 4.4.16)
- Configuring hostname verification (Carbon 4.4.17 onwards)

#### Important!

Be sure to enable hostname verification for your production environment. This is necessary to avoid security issues in production. See the full list of security recommendations for production environments.

### Prerequisites

Note that this configuration is only available as a WUM update for the following product versions:

- WSO2 Enterprise Integrator (WSO2 EI) version 6.1.1 can use this feature with WUM updates obtained after the 4th of February, 2018.
- WSO2 API Manager (WSO2 AM) versions 2.0.0 and 2.1.0 can use this feature with WUM updates obtained after the 28th of July, 2017.
- WSO2 Identity Server (WSO2 IS) version 5.2.0 can use this feature with WUM updates obtained after the 28th of July, 2017.
- WSO2 Enterprise Service Bus version 5.0.0 can use this feature with WUM updates obtained after the 2nd of October, 2017.

See [Getting Started with WUM](#) for more instructions on how to get updates. If you have already got the WUM updates, see the topic below on Configuring hostname verification (Carbon 4.4.17 onwards) for instructions on how to apply this configuration.

For all other product versions (that are not listed above), this configuration will apply based on the Carbon version. Therefore, check the Carbon version of your WSO2 product, and then following the instructions given below.

### Configuring hostname verification (Carbon 4.4.6 to Carbon 4.4.16)

Before you begin, be sure to check the prerequisites.

If you are using a WSO2 product that is based on a Carbon version between 4.4.6 and 4.4.16, hostname verification is disabled by default. This setting is disabled using the `org.wso2.ignoreHostnameVerification` property in the product startup script (`wso2server.sh` for Linux, or `wso2server.bat` for Windows) that is stored in the `<PRODUCT_HOME>/bin/` directory.

```bash
-Dorg.wso2.ignoreHostnameVerification=true
```

### Configuring hostname verification (Carbon 4.4.17 onwards)

Before you begin, be sure to check the prerequisites.

If you are using a WSO2 product that is based on Carbon 4.4.17 or a later version, hostname verification is enabled by default. This is done using the `httpclient.hostnameVerifier` property in the product’s startup script (`wso2server.sh` for Linux and `wso2server.bat` for Windows) as shown below.
The product startup script is stored in the `<PRODUCT_HOME>/bin` directory. This property will be effective during server startup.

```
-Dhttpclient.hostnameVerifier="DefaultAndLocalhost"
```

The values you can use with this property are explained below. Note that these values will behave the same as synapse hostname verification options.

- **DefaultAndLocalhost**: This is the value that is enabled, by default. This means that all hostnames, except the ones listed below, will be verified against the hostnames specified in the product's SSL certificate. That is, the following hostnames will be allowed regardless of the server's certificate.
  - localhost
  - localhost.localdomain
  - 127.0.0.1
  - ::1

  Note that if the wildcard symbol is used to specify a hostname in the SSL certificate (such as `*.foo.com`), all the subdomains of `*.foo.com` are also included. That is, a hostname that matches a subdomain of `*.foo.com` will also be allowed access.

- **Strict**: When this mode is enabled, hostnames will be strictly verified against the hostname specified in the product's SSL certificate. For example, if `"*.foo.com"` is specified as the hostname in the certificate, only the hostnames at the same level will be authorized by the server. That is, subdomains such as `"a.b.foo.com"` will **not** be authorized.

- **AllowAll**: This option turns off hostname verification for the server. Note that this is not recommended in a production setup and should only be used for demonstrations and testing.

---

**Important!**

If you are disabling hostname verification for **WSO2 AM 2.0.0** or **2.1.0**, you need to use both system properties listed below.

```
-Dorg.wso2.ignoreHostnameVerification=true \  
-Dhttpclient.hostnameVerifier=AllowAll \  
```

If you are disabling hostname verification for **WSO2 EI 6.1.1**, you need to use both system properties listed below.

```
-Dhttpclient.hostnameVerifier=AllowAll \  
-Dorg.opensaml.httpclient.https.disableHostnameVerification=true \  
```

If you are disabling hostname verification for **WSO2 IS**, use the following system properties.

```
-Dorg.opensaml.httpclient.https.disableHostnameVerification=true \  
-Dhttpclient.hostnameVerifier="AllowAll"  
```

---

**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.

When you work with Carbon servers, this will allow you to access admin services and the admin console of your product using HTTP (without SSL).

Given below are the steps you need to follow:

- **Step 1**: Configuring the load balancer with TLS termination
- **Step 2**: Enabling RemoteIpValve for Carbon servers
Step 1: Configuring the load balancer with TLS termination

See the documentation of the load balancer that you are using for instructions on how to enable TLS termination. For example, see NGINX SSL Termination.

Step 2: Enabling RemoteIpValve for Carbon servers

You can enable Tomcat's RemoteIpValve for your Carbon server by simply adding the valve to the catalina-sever.xml file (stored in the <PRODUCT_HOME>/repository/conf/tomcat directory). This valve should be specified under the <Host> element (shown below) in the catalina-sever.xml file. See the Tomcat documentation for more information about RemoteIpValve.

```xml
<Host name="localhost" unpackWARs="true" deployOnStartup="false"
    autoDeploy="false"
    appBase="${carbon.home}/repository/deployment/server/webapps/">
    ............
    <Valve className=""/>
</Host>
```
Carbon Applications

See the following topics for information on how to set up and configure various components in WSO2 products:

- Adding a Custom Proxy Path
- Changing the Default Ports
- Configuring SAML2 Single-Sign-On Across Different WSO2 Products
- Configuring the Task Scheduling Component
- Customizing Error Pages
- Customizing the Management Console
- Installing WSO2 Products as a Service
- Managing Datasources
- Managing Users, Roles and Permissions
- Securing Carbon Applications
- Working with Composite Applications
- Working with Features
- Working with Multiple Tenants
- Working with Transports

Adding a Custom Proxy Path

Adding a custom proxy path is useful when you have a proxy server fronting your Carbon server. In this scenario, the "custom proxy path" is used for mapping a proxy url with the actual url of your Carbon server, which allows clients to access the Carbon server with the proxy url.

This feature is particularly useful when multiple WSO2 products are hosted under the same domain name. For example, consider that you have three WSO2 products; Application Server, API Manager and ESB, deployed in your production environment and you want all of them to be hosted with the "wso2test.com" domain. By using a reverse proxy and by configuring your servers with 'custom proxy paths', you can host all products under a single domain and assign proxy paths for each product separately as shown below:

![Reverse Proxy Server](https://wso2test.com)

Proxy URLs mapped to Carbon server URLs:


Note the following:

- This functionality is only available for WSO2 products that are based on Carbon 4.3.0 or a later Carbon version. See the WSO2 product release matrix for more information about WSO2 Carbon platform releases.
- Once you have configured your products with a proxy server, it will no longer be possible to access the product behind the proxy. See the section given below on configuring products to use the proxy server for more information.

In the above example, “apimanager”, “esb” and “appserver” are the “proxy context paths” of the respective products, which are configured in the carbon.xml file (stored in `<PRODUCT_HOME>/repository/conf` directory) for each product. When a client sends a request to the proxy entry url path, e.g. https://wso2test.com/apimanager, the request is directed to the back-end service url (https://10.100.1.1:<PortNumber>/carbon) where the original service lies. Eventually, the client has to be served via the requested proxy entry url path. The mapping between the proxy url path and the back-end service url path is resolved by the reverse proxy server fronting the back-end service.

Prior to this solution, it was necessary to host these products as sub domains of the "wso2.com" domain as: https://apim.wso2.com, https://esb.wso2.com, https://as.wso2.com.

Access WSO2 products through a custom proxy path
This functionality will be demonstrated in this documentation using two WSO2 product servers as examples; WSO2 Application Server and WSO2 ESB as the back-end servers, and nginx as the reverse proxy.

Follow the steps given below.
- Step 1: Install and configure a reverse proxy
- Step 2: Configure products with proxy context path
- Step 3: Start the Product

**Step 1: Install and configure a reverse proxy**

1. Download nginx server.
2. Install the nginx server in your deployment server by executing the following command:

   ```bash
   sudo apt-get install nginx
   ```

3. Create a folder called “ssl” inside /etc/nginx, and create the ssl certificates inside this folder by executing the following commands:

   ```bash
   sudo mkdir /etc/nginx/ssl
   cd /etc/nginx/ssl
   ```

4. The next step is to create the server key and certificates. First create the private key as shown below. Note that a pass phrase is prompted when creating the private key.

   ```bash
   sudo openssl genrsa -des3 -out server.key 1024
   ```

5. Next, create the certificate signing request as shown below.

   ```bash
   sudo openssl req -new -key server.key -out server.csr
   ```

   Fill in the required details. Most important entry is the Common Name. Enter the domain name or the ip address if there is no domain name.

6. Next step is to sign the SSL certificate using the following command:

   ```bash
   sudo openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
   ```

   The certificate is now created.

7. The last step is to set up the virtual host displaying the new certificate. Create a copy of the default, " sites-enabled" configuration using the following command:

   ```bash
   sudo cp /etc/nginx/sites-available/default /etc/nginx/sites-available/wso2
   ```
8. Now, create a symbolic link between the "sites-enabled" directory and the "sites-available" directory using the following command:

```
sudo ln -s /etc/nginx/sites-available/wso2 /etc/nginx/sites-enabled/wso2
```

The host is now activated.

9. Open the `/etc/nginx/sites-enabled/wso2` file and enter the following configurations.
#Configurations for listener 8243.
server {
    listen 8243;
    server_name wso2test.com;
    client_max_body_size 100M;
    root /usr/share/nginx/www;
    index index.html index.htm;
    
    ssl on;
    ssl_certificate /etc/nginx/ssl/server.crt;
    ssl_certificate_key /etc/nginx/ssl/server.key;
    
    #with portOffset 0 running AS
    location /appserver/ {
        proxy_pass https://wso2test.com:9443/;
        proxy_redirect https://wso2test.com:8243/;
        proxy_cookie_path / /appserver/;
    }
    
    #with portOffset 10 running ESB
    location /esb/ {
        proxy_pass https://wso2test.com:9453/;
        proxy_redirect https://wso2test.com:8243/;
        proxy_cookie_path / /esb/;
    }
}

#Configurations for listener 8280.
server {
    listen 8280;
    server_name wso2test.com;
    client_max_body_size 100M;
    root /usr/share/nginx/www;
    index index.html index.htm;
    
    #with portOffset 0 running AS
    location /appserver/ {
        proxy_pass http://wso2test.com:9763/;
        proxy_redirect http://wso2test.com:8280/;
        proxy_cookie_path / /appserver/;
    }
    
    #with portOffset 10 running ESB
    location /esb/ {
        proxy_pass http://wso2test.com:9773/;
        proxy_redirect http://wso2test.com:8280/;
        proxy_cookie_path / /esb/;
    }
}

According to the nginx configuration, https requests with the /appserver/* pattern are directed to the /* pattern and then when the service is served to the client, it resolves the url pattern to /appserver/*. This works the same for http requests.

10. Save the file and restart the nginx server using the following command to complete the nginx configuration:
11. In the above configuration, the https and http requests are listening on 8243 and 8280 ports respectively. Server name is set to wso2test.com. To test this in a local machine, you need to add wso2test.com and as.wso2.com to the /etc/hosts file as shown below.

    127.0.0.1  wso2test.com
    127.0.0.1  as.wso2test.com
    127.0.0.1  esb.wso2test.com

**Step 2: Configure products with proxy context path**

1. Download WSO2 Application Server and WSO2 ESB.
2. Open the carbon.xml file stored in the `<PRODUCT_HOME>/repository/conf/` directory and set the HostName to what you defined in the nginx configuration as shown below (for both products):

    <HostName>wso2test.com</HostName>

3. Now, set the MgtHostName as shown below.
   - For Application Server:
     
     <MgtHostName>as.wso2test.com</MgtHostName>

   - For ESB:
     
     <MgtHostName>esb.wso2test.com</MgtHostName>

4. Set the "ProxyContextPath" as shown below. This is the proxy path string, which will appear in the management console, web apps and services urls.
   - For Application Server:
     
     <ProxyContextPath>appserver</ProxyContextPath>

   - For ESB:
     
     <ProxyContextPath>esb</ProxyContextPath>
5. Since you need to run both products (AS and ESB) simultaneously, set port offsets as shown below.
   - For Application Server: `<Offset>0</Offset>`
   - For ESB: `<Offset>10</Offset>`

6. According to the nginx configuration, the https, http requests are listening on 8243 and 8280 ports. However, by default WSO2 products are listening on 9443 (WSO2 Application Server) and 9453 (WSO2 ESB). Therefore, the listening ports of the reverse proxy should be configured as proxy ports in Application Server and ESB respectively. To enable proxy ports, open the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file and add the "proxyPort" entries.

   Note that after you define proxy ports (8243 and 8280) in the catalina-server.xml file, it will no longer be possible to access the products using the normal ports (9443 and 9453).

For example, the "proxyPort" entries for Application Server are as follows:

```xml
             port="9763"
            proxyPort="8280"
             redirectPort="9443"
             bindOnInit="false"
             maxHttpHeaderSize="8192"
             acceptorThreadCount="2"
             maxThreads="250"
             minSpareThreads="50"
             disableUploadTimeout="false"
             connectionUploadTimeout="false"
             maxKeepAliveRequests="200"
             acceptCount="200"
             server="WSO2 Carbon Server"
             compression="on"
             compressionMinSize="2048"
             noCompressionUserAgents="mozilla,traviata"
             compressableMimeType="text/html,text/javascript,application/xjavascript,application/javascript,application/xml,text/css,application/xslt+xml,text/xsl,image/gif,image/jpg,image/jpeg"
             URIEncoding="UTF-8" />
```

Optional attributes:
- `proxyPort="443"`

Step 3: Start the Product

1. Start the server and enter the following url in a browser:
   - For Application Server: https://wso2test.com:8243/appserver/carbon/
   - For ESB: https://wso2test.com:8243/esb/carbon/
2. Give the admin credentials and log in to the server. You’ll find the proxy path for admin console, services, webapps changed for each product as shown below.

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. Port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be changed. For example, if the default HTTP port is 9763 and the port 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.

- Default ports of WSO2 products
Setting a port offset for the server

Default ports of WSO2 products

See this link for the list of ports that are used in all WSO2 products.

Setting a port offset for the server

The default port offset value is 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 Ports section of `<PRODUCT_HOME>/repository/conf/carbon.xml` as follows:
  ```
  <Offset>3</Offset>
  ```

When you set the server-level port offset in WSO2 AS as shown above, all the ports used by the server will change automatically. However, this may not be the case with some WSO2 products such as WSO2 APIM and WSO2 AM. See the product documentation for instructions that are specific to your product.

Default Ports of WSO2 Products

This page describes the default ports that are used for each WSO2 product when the port offset is 0.

- **Common ports**
  - Management console ports
  - LDAP server ports
  - KDC ports
  - JMX monitoring ports
  - Clustering ports
  - Random ports

- **Product-specific ports**
  - API Manager
  - Data Analytics Server
    - Ports inherited from WSO2 BAM
    - Ports used by the Spark Analytics Engine
  - Business Process Server
  - Complex Event Processor
  - Elastic Load Balancer
  - Enterprise Service Bus
  - Enterprise Integrator
    - ESB ports
    - EI-Analytics ports
    - EI-Business Process ports
    - EI-Broker ports
    - EI-Micro Integrator ports
    - EI-MSF4J ports
  - Identity Server
  - Message Broker
  - Machine Learner
  - Storage Server
  - Enterprise Mobility Manager
  - IoT Server
    - Default ports
    - Ports required for mobile devices to communicate with the server and the respective notification servers.

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 (except WSO2 Enterprise Integrator) 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.

• 45664 - 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 WSO2 products will have additional ports as explained below.

*API Manager* | *Data Analytics Server* | *Business Process Server* | *Complex Event Processor* | *Elastic Load Balancer* | *Enterprise Service Bus* | *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+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.

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 *API Manager documentation*.  

**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/docs/latest/configuration.html).

• **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>
<tr>
<td>spark.worker.port</td>
<td>11000</td>
</tr>
<tr>
<td>spark.worker.webui.port</td>
<td>11500</td>
</tr>
</tbody>
</table>

**Business Process Server**

- 2199 - RMI registry port (datasources provider port)

**Complex Event Processor**

- 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

**Enterprise Service Bus**

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**

Listed below are the default ports that are used in WSO2 Enterprise Integrator (WSO2 EI) when the port offset is 0.

**ESB 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. For example, if you want to send a request to an API or a proxy service, you must use these ports: `<EI_HOME>/conf/axis2/axis2.xml`

- 8243 - Passthrough or NIO HTTPS transport
- 8280 - Passthrough or NIO HTTP transport

**EI-Analytics ports**

- 9643 - The port on which the Analytics dashboard opens
- 9161 - Cassandra port on which Thrift listens to clients
- 7712 - Thrift SSL port for secure transport, where the client is authenticated for the Analytics profile
- 7612 - Thrift TCP port to receive events from clients to DAS

**EI-Business Process ports**

- 9445 - HTTPS servlet transport (the default URL of the management console is `https://localhost:9445/carbon`)
- 9765 - HTTP servlet transport

**EI-Broker 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.

**EI-Micro Integrator ports**

- 8290 - HTTP servlet transport
- 8253 - HTTPS servlet transport

**EI-MSF4J ports**

- 9090 - HTTP servlet transport

**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 MapReduce job tracker
- 50470 - Name node secure HTTP server port
- 50471 - 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:**

The following WSO2 API Manager ports are only applicable to WSO2 EMM 1.1.0 onwards.

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports

**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.**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5228</td>
<td>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.</td>
</tr>
</tbody>
</table>
1. Transmission Control Protocol (TCP) port used by devices to communicate to APNs servers.

2. TCP port used to send notifications to APNs.

3. TCP port used by the APNs feedback service.

4. 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.

## Configuring SAML2 Single-Sign-On Across Different WSO2 Products

With the SAML2 relying party capabilities of Carbon, it is possible to set up Single Sign-On between different WSO2 Product instances where WSO2 Identity Server acts as the identity provider while other WSO2 products act as the relying party. This topic provides instructions on how to set up Single Sign-On between different WSO2 products.

### Step 1 - Installing the SAML2 relying party (SAML2 SSO Authenticator) feature in a Carbon Server

You only need to do this step if "SAML2 based Single Sign-On authenticator" is not installed in your WSO2 product.

SAML2 relying party components are not shipped with the default Carbon distribution. However, these bundles are packages that can be installed as a feature, which is available in the online-hosted P2 repository. Therefore, it is possible to install this feature with minimal effort through the Carbon Feature Manager.

1. Add the WSO2 online P2 repository as a new repository. Usually, the hosted P2 repository is available at this URL: `http://dist.wso2.org/p2/carbon/releases/Carbon-Release-Version`. Learn how to add a repository to the Identity Server here.


### Step 2 - Configuring the Carbon Server to use the SAML2-based authentication instead of default username/password-based authentication

After installing the SAML2 relying party components (SAML2 SSO authenticator), it is necessary to configure the SAML2 SSO authentication component to communicate with the Identity Server for user authentication. This can be configured in the `<PRODUCT_HOME>/repository/conf/security/authenticators.xml` file. This file will contain configurations for different authenticators. By default, it is shipped with a sample configuration for SAML2 SSO authenticator and requires minor modifications to prior setup.

```xml
<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:9443/samlsso</Parameter>
        <Parameter name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
        <Parameter name="IdPCertAlias">wso2carbon</Parameter>
    </Config>
</Authenticator>
```

- **Authenticator disabled** - This should be set to `false`.
- **Priority** - This is the priority level of the authenticator. In the Carbon Runtime, the authenticator with the highest priority will be picked up. This value should be greater than 5 in order to supersede the default username/password-based authenticator.
- **Parameter LoginPage** - This is the default login page URL of Carbon. All requests coming to this page will be intercepted for authentication. It is not necessary to change this value from the value given in the sample configuration.
- **Parameter ServiceProviderID** - This is the unique identifier for the Carbon Server in an SSO setup. This value should be used as the value of the issuer in the Identity Server configuration.
- **Parameter IdentityProviderSSOServiceURL** - This is the Identity Server URL to which the users will be redirected for authentication.
Step 3 - Configuring the Identity Server as the Single Sign-On provider

Finally, you need to configure the Identity Server to act as the Single Sign-on provider. Each relying party should be registered as a service provider at the Identity Server-end. The following is a sample configuration for registering a Carbon server as a service provider.

1. Sign in. Enter your username and password to log on to the Management Console.
2. Navigate to the Main menu to access the Identity menu. Click Add under Service Providers.
3. Fill in the Service Provider Name and provide a brief Description of the service provider. Only Service Provider Name is a required field.
4. Click Register to add the new service provider.
5. Expand the Inbound Authentication Configuration section, followed by the SAML2 Web SSO Configuration section and click Configure.
6. Fill in the form that appears.
   - Specify the Issuer. This should be equal to the ServiceProviderID value mentioned in the authenticators.xml of the relying party Carbon server.
   - Specify the Assertion Consumer URL. This is the URL to which the browser should be redirected after the authentication is successful. It should have this format: https://(host-name):(port)/acs.
   - Select Use fully qualified username in SAML Response if that feature is required.
   - Select Enable Response Signing to sign the SAML2 Responses returned after the authentication.
   - Select Enable Assertion Signing to sign the SAML2 Assertions returned after the authentication. SAML2 relying party components expect these assertions to be signed by the Identity Server.
   - Select Enable Signature Validation in Authentication Requests and Logout Requests if you need this feature configured.
   - Select Enable Single Logout so that all sessions are terminated once the user signs out from one server. You can enter a Custom Logout URL if required.
   - Select Enable Attribute Profile to enable this and add a claim by entering the claim link and clicking the Add Claim button.
   - Select Enable Audience Restriction to restrict the audience. You may add audience members using the Audience text box and clicking the Add Audience button.
7. Expand the Local & Outbound Authentication Configuration section and do the following.
   - Select Use tenant domain in local subject identifier to append the tenant domain to the local subject identifier.
   - Select Use user store domain in local subject identifier to append the user store domain that the user resides in the local subject identifier.
8. Click the Update button to update the details of the service provider.

Configuring the Task Scheduling Component

The Task Scheduling component in a WSO2 product allows you to define specific tasks and to invoke them periodically. This functionality is used by WSO2 products such as WSO2 Enterprise Integrator (WSO2 EI), WSO2 Enterprise Service Bus (WSO2 ESB), and WSO2 Data Services Server (WSO2 DSS).

Follow the instructions given on this page to configure and set up this component. Then, see the respective product’s documentation for details on how to use task scheduling.

The task scheduling component is configured in the tasks-config.xml file (stored in the <PRODUCT_HOME>/conf/etc/ directory for WSO2 EI, and in the <PRODUCT_HOME>/repository/conf/etc/ directory for other products).

Given below are the settings that you can configure in the tasks-config.xml file.

- Step 1: Setting the task server mode
- Step 2: Configuring a cluster of task servers
  - Setting the task server count
  - Setting the default location resolver
Step 1: Setting the task server mode

Select one of the following values for the `<taskServerMode>` element in the `tasks-config.xml` file:

- **AUTO**: This is the default task handling mode. This setting detects if clustering is enabled in the server and automatically switches to CLUSTERED mode.
- **STANDALONE**: This setting is used when a single instance of the product server is installed. That is, tasks will be managed locally within the server.
- **CLUSTERED**: This setting is used when a cluster of product servers are put together. This requires Axis2 clustering to work. With this setting, if one of the servers in the cluster fail, the tasks will be rescheduled in one of the remaining server nodes.

Find out more about clustering WSO2 products.

Step 2: Configuring a cluster of task servers

If you have enabled the CLUSTERED task server mode in step 1, the following configuration elements in the `tasks-config.xml` file will be effective:

### Setting the task server count

Use the parameter shown below to specify the number of servers (in the cluster) that can handle tasks. The task server count is set to "1" by default, which indicates that at least one node in the cluster is required for task handling.

```xml
<taskServerCount>1</taskServerCount>
```

Note that a product cluster begins the process of scheduling tasks only after the given number of servers are activated. For example, consider a situation where ten tasks are saved and scheduled in your product and there are five task-handling servers. As the individual servers become active, we do not want the first active server to schedule all the tasks. Instead, all five servers should become active and share the ten tasks between them.

### Setting the default location resolver

The default location resolver controls how the scheduled tasks are shared by multiple server nodes of a cluster. Note that the task server count should be a value larger than "1" for this location resolver setting to be effective. For example, if there are 5 task-handling servers in the cluster, the location resolver determines how the tasks get shared among the 5 servers.

The location resolver parameter also controls how task failover is handled in a clustered environment. A scheduled task will only run in one of the nodes (at a given time) in a clustered environment. The task will failover to another node only if the first node fails. This parameter determines how nodes are selected for task failover.

One of the following options can be used for the `location resolver`.

- **RoundRobinTaskLocationResolver**: Cluster nodes are selected on a round-robin basis and the tasks are allocated. This location resolver is enabled in the `tasks-config.xml` file by default as shown below.

  ```xml
  <defaultLocationResolver>
  <locationResolverClass>org.wso2.carbon.ntask.core.impl.RoundRobinTaskLocationResolver</locationResolverClass>
  </defaultLocationResolver>
  ```

- **RandomTaskLocationResolver**: Cluster nodes are randomly selected and the tasks are allocated. If you want to enable this location resolver, you need to change the default configuration (shown above) in the `tasks-config.xml` file as shown below.

  ```xml
  <defaultLocationResolver>
  <locationResolverClass>org.wso2.carbon.ntask.core.impl.RandomTaskLocationResolver</locationResolverClass>
  </defaultLocationResolver>
  ```
RuleBasedLocationResolver: This allows you to set the criteria for selecting the cluster nodes to which the tasks should be allocated. The [task-type-pattern], [task-name-pattern], and [address-pattern of the server node] can be used as criteria. For example, with this setting, a scheduled task that matches a particular [task-type-pattern], and [task-name-pattern] will be allocated to the server node with a particular [address-pattern]. If multiple server nodes in the cluster match the [address-pattern], the nodes are selected on a round robin basis. This criteria is specified in the configuration using the <property> element. Therefore, you can define multiple properties containing different criteria. Before you enable this location resolver, you need to first comment out the default location resolver that is already enabled in the task-config.xml file. You can then uncomment the following code block, and update the property values as required.

As shown above, the property names (rule-1, rule-2 and rule-5) define a sequence for the list of properties in the configuration. Therefore, scheduled tasks will evaluate the criteria specified in each property according to the sequence. That is, rule-1 is checked before rule-2. In other words, the scheduled task will first check if it matches the criteria in rule-1, and if it does not, it will check rule-2.

Customizing Error Pages

WSO2 Carbon servers display errors, exceptions, and HTTP status codes in full detail. These are known as Verbose error messages. These error messages contain technical details such as stack traces. There may also disclose other sensitive details. Attackers may fingerprint the server, based on the information disclosed in error messages. Alternatively, attackers may attempt to trigger specific error messages to obtain technical information about the server. You can avoid these situations by configuring the server to display generic, non-detailed error messages in Apache Tomcat.

From Carbon 4.4.6 onwards, the pages that should be displayed on a certain throwaway exception, error or an HTTP status code are specified in the CARBON_HOME>repository/conf/tomcat/carbon/WEB-INF/web.xml file. You can customize those error pages as preferred. For example, if you try to access a resource that is not available in the Carbon server (e.g., https://10.100.5.72:9443/abc), you will view the error page as follows: "Error 404 - Not Found".

You can customize the above error message by following the instructions given below.

1. Download and install Apache Maven.
2. Create a Maven project using your IDE.
3. Create a directory named resources inside the <PROJECT_HOME>/src/main/ directory, and then create another directory named web inside it.

   <PROJECT_HOME> denotes the top-level directory of your Maven project.

4. Create a new HTML error page (e.g. new_error_404.html file) as shown below. This contains the customized error page.
5. Add the new_error_404.html file inside the <PROJECT_HOME>/src/main/web directory.
6. Add the following property below the <version> element in the <PROJECT_HOME>/pom.xml file: <packaging>bundle</packaging>
7. Add the following configurations inside the <plugins> element in the <PROJECT_HOME>/pom.xml file.

```xml
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <instructions>
      <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
      <Bundle-Name>${project.artifactId}</Bundle-Name>
      <Import-Package>
        org.osgi.framework,
        org.osgi.service.http,
        org.wso2.carbon.ui,
        javax.servlet.*;version="2.4.0",
        *;resolution:=optional
      </Import-Package>
      <Fragment-Host>org.wso2.carbon.ui</Fragment-Host>
      <Carbon-Component>UIBundle</Carbon-Component>
    </instructions>
  </configuration>
</plugin>
```

8. Add the following configurations inside the <dependencies> element in the <PROJECT_HOME>/pom.xml file:

```xml
<dependency>
  <groupId>org.apache.felix</groupId>
  <artifactId>org.apache.felix.framework</artifactId>
  <version>1.0.3</version>
</dependency>
```

9. Build the Maven project by executing the following command: mvn clean install
10. Once the project is built, copy the JAR file (from the <PROJECT_HOME>/target directory) to the <CARBON_HOME>/repository/components/dropins directory.
11. Restart the WSO2 Carbon server.
12. Access the following URL again, to test the error page you customized: https://10.100.5.72:9443/abc.
You will view the new error page with the following content: "Sorry, this resource is not found."

Customizing the Management Console
See the following topics to do customizations to the management console:

- Changing the management console's URL
- Configuring the session time-out
- Changing the management console's interface

Changing the management console's URL
When you start a WSO2 server, the URL of the management console will be printed on the terminal. The URL format is as follows: https://<Server Host>:<Management Console Port>/carbon/. When accessing the management console from the same server where it is installed, you can type localhost instead of the IP address.

You can change the URL by modifying the value of the <MgtHostName> property in the <EI_HOME>/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>

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.

```xml
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

Changing the management console's interface
The user interfaces of every Carbon product allows you to configure, monitor, tune, and maintain the product. The components that formulate the design and style of these user interfaces are defined in resource (JAR) files.

The user interface of every Carbon product consists of two layers:

- The common product layout/design inherited from the Carbon platform: All the common templates, styles (CSS files), and images are stored in the
Carbon core UI bundle, which is named \texttt{org.wso2.carbon.ui-<version-number>.jar} (\texttt{<version-number>} is the particular version of the bundle). This bundle is responsible for the overall look and feel of the entire Carbon platform.

- The styles/images unique to each product: Each Carbon product (that is built on Carbon kernel) has another style bundle, which contains all the overriding style sheets and images: \texttt{org.wso2.<product-name>.styles-<version-number>.jar}.

You can customize the user interfaces by modifying these resource files. You need to create a fragment bundle for the original resource file. Then, you can pack the modified resource files in the required bundle. The files in the required bundle will get precedence and will override the files in the original bundle.

You can use this same technique to customize any aspect of the user interfaces. The advantage of this technique is that you will not lose your customizations when you apply official patches to the product by replacing the original bundles.

For example, when you access the Management Console using the following URL, by default, it has the WSO2 product logo as shown below: \url{https://10.10.0.5.72:9443/carbon/}

Note that the images and instructions given on this page are valid for WSO2 products that are based on Carbon 4.4.x.

Follow the steps below to customize the above management console by changing the logo.

1. Open the \texttt{<PRODUCT_HOME>/repository/components/plugins/} directory. You need to find the bundle that contains the resource files that you want to modify. In this case, the logo and the related CSS files are contained in the \texttt{org.wso2.carbon.ui_<version-number>.jar} file. Copy the \texttt{org.wso2.carbon.ui_<version-number>.jar} file to a separate location on your computer, and extract the file. Note the symbolic name of this bundle, which is \texttt{org.wso2.carbon.ui_<version-number>}.

2. Create a new Maven project using your IDE. Be sure to include the symbolic name of the original bundle that you extracted in the previous step (which is \texttt{org.wso2.carbon.ui_<version-number>}) in the Maven project name. For example, you can use \texttt{org.wso2.carbon.ui_<version-number>_patch} as the Maven project name.

3. Add the following content to the \texttt{pom.xml} file of the \texttt{org.wso2.carbon.ui_<version-number>_patch} project. In this \texttt{pom.xml} file, be sure to replace the \texttt{<version-number>} of \texttt{org.wso2.carbon.ui_<version-number>_patch} with the correct version value.
<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.wso2.carbon</groupId>
  <artifactId>org.wso2.carbon.ui_<version-number>_patch</artifactId>
  <version>1.0.0</version>
  <packaging>bundle</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>3.0.1</version>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
            <Bundle-Name>${project.artifactId}</Bundle-Name>
            <Export-Package>web.admin.*</Export-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>

4. Create directories in your Maven project as explained below.
   a. Create the /web folder under the /src/main/resources directory of the org.wso2.carbon.ui_<version-number>_patch project.
   b. Then, create the /admin directory under /web.
   c. Finally, create the /css, /images, and /layout directories under /admin.

Your org.wso2.carbon.ui_<version-number>_patch project should now look as shown below.
5. Create a new CSS file (e.g. customizations.css) with the following content.

```css
#header div#header-div div.left-logo {
  background-image: url( ../images/new-logo.png );
  background-repeat: no-repeat;
  background-position: left top;
  background-size: contain;
  height: 40px;
  width: 300px;
  margin-top: 23px;
  margin-left: 20px;
  float: left;
}
```

This file includes the logo customization styles.


7. Locate the template.jsp file that is in the `<org.wso2.carbon.ui_<version-number>.jar` bundle, which you extracted in step 1 above. You will find template.jsp file inside the `<org.wso2.carbon.ui_<version-number>.jar>/web/admin/layout/` directory. Then, copy this file to the `<org.wso2.carbon.ui_<version-number>_patch>/src/main/resources/web/admin/layout/` directory.

8. Locate the following line in the `<org.wso2.carbon.ui_<version-number>_patch>/src/main/resources/web/admin/layout/template.jsp` file, which you added in the previous step:

```html
<link href="<%=globalCSS%>" rel="stylesheet" type="text/css" media="all"/>
```

9. Replace the above line with the following:
10. Add the below image as the new logo (e.g. new-logo.png) to the `<org.wso2.carbon.ui_<version-number>_patch>/src/main/resources/web/admin/images/` directory.

![Lorem Ipsum Online](image)

11. Create another Maven project using your IDE. Be sure to include the symbolic name of the original bundle that you extracted in step 1 above (which is `org.wso2.carbon.ui_<version-number>`) in the project name. For example, you can use `org.wso2.carbon.ui_<version-number>_fragment` as the Maven project name.

   This creates a project for the fragment bundle. Since the symbolic name of the original bundle is `org.wso2.carbon.ui`, the fragment host value of this bundle should be the same (e.g. `org.wso2.carbon.ui_<version-number>_fragment`). This fragment bundle will not contain anything (expect the `pom.xml` file) when it is built.

12. Add the following content to the `pom.xml` file of the `org.wso2.carbon.ui_<version-number>_fragment` project. In this `pom.xml` file, replace the `<version-number>` of `org.wso2.carbon.ui_<version-number>_patch` and `org.wso2.carbon.ui_<version-number>_fragment` with the correct version value.

   This `pom.xml` file of the fragment bundle defines properties, which includes the required bundle value (i.e. `org.wso2.carbon.ui_<version-number>_patch`).
13. Now you can build the two projects. Open a terminal, navigate to the relevant project directory (listed above), and execute the following command: `mvn clean install.
   * org.wso2.carbon.ui_<version-number>_fragment
   * org.wso2.carbon.ui_<version-number>_patch
14. Once the project is built, copy the two JAR files listed below (from the `<PROJECT_HOME>/target/` directory) to the `<PRODUCT_HOME>/repository/components/dropins/` directory.
   * org.wso2.carbon.ui_<version-number>_fragment-1.0.0.jar
   * org.wso2.carbon.ui_<version-number>_patch-1.0.0.jar
15. Restart the WSO2 product server.
16. Access the management console of your WSO2 product using the following URL: https://10.100.5.12:9443/carbon/. You view the new logo, which the patch bundle contains as shown below.
The following topics will guide you on how a WSO2 product can be installed as a Windows/Linux service:

- Installing as a Linux Service
- Installing as a Windows Service

## 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

### Prerequisites

#### System requirements

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 2 GB minimum</td>
</tr>
<tr>
<td>~ 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 500 MB, excluding space allocated for log files and databases.</td>
</tr>
</tbody>
</table>

#### Environment compatibility

- All WSO2 Carbon-based products are Java applications that can be run on **any platform that is Oracle JDK 7/8 compliant**.
- All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see [Working with Databases](#).
- It is not recommended to use Apache DS in a production environment due to issues with scalability. Instead, it is recommended to use an LDAP like OpenLDAP for user management.
- 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 product and its samples or for building from the source code. Mandatory installations are marked with an asterisk *.

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Version</th>
<th>Download Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Java SE Development Kit (JDK)*</td>
<td>To launch the product as each product is a Java application.</td>
<td>JDK 7 or 8.</td>
<td><a href="http://java.sun.com/javase/downloads/index.jsp">http://java.sun.com/javase/downloads/index.jsp</a></td>
</tr>
<tr>
<td></td>
<td>To <strong>build the product from the source distribution</strong> (both JDK and Apache Maven are required).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>To run Apache Ant.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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:
#!/bin/bash

case "\$1 in
  start)
    echo "Starting the Service"
    ;;
  stop)
    echo "Stopping the Service"
    ;;
  restart)
    echo "Restarting the Service"
    ;;
  *)
    echo "$Usage: $0 {start|stop|restart}"
    exit 1
esac

Given below is a sample startup script. <PRODUCT_HOME> can vary depending on the WSO2 product's directory.

#!/ /bin/sh
export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_07"

startcmd='<PRODUCT_HOME>/bin/wso2server.sh start > /dev/null &'
restartcmd='<PRODUCT_HOME>/bin/wso2server.sh restart > /dev/null &'
stopcmd='<PRODUCT_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 "Usage: $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. Install the startup script to respective runlevels using the command update-rc.d. For example, give the following command for the sample script shown in step1:

   - 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.

```
sudo update-rc.d prodserver defaults
```

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 WSO2 product can be run as a Windows service. It is also possible to install and run multiple WSO2 products as Windows services simultaneously. See the following topics for instructions:

### Prerequisites

- **System requirements**
  - **Memory**: ~ 2 GB minimum
  - ~ 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.
  - **Disk**: ~ 500 MB, excluding space allocated for log files and databases.

- **Environment compatibility**
  - **Operating Systems / Databases**: All WSO2 Carbon-based products are Java applications that can be run on any platform that is Oracle JDK 7/8 compliant.
  - All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see Working with Databases.
  - It is not recommended to use Apache DS in a production environment due to issues with scalability. Instead, it is recommended to use an LDAP like OpenLDAP for user management.
  - 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 product and its samples or for building from the source code. Mandatory installations are marked with an asterisk `*`:
    - **Oracle Java SE Development Kit (JDK)**
      - To launch the product as each product is a Java application.
      - To build the product from the source distribution (both JDK and Apache Maven are required).
      - To run Apache Ant.
      - **Version**: JDK 7 or 8.
        - Oracle and IBM JRE 1.7 are also supported when running (not building) WSO2 products.
      - Download and install a service wrapper library to use for running your WSO2 product as a Windows service. WSO2 recommends Yet Another Java Service Wrapper (YAJSW), and several WSO2 products provide a default `wrapper.conf` file in their `<PRODUCT_HOME>/bin/yajsw` directory.
Installing a single product as a Windows service

Given below are the steps for installing a single WSO2 server as a windows service.

1. Setting up the YAJSW wrapper: The configuration file used for wrapping Java Applications by YAJSW is `wrapper.conf`, which is located in the `<YAJSW_HOME>/conf/` directory and in the `<PRODUCT_HOME>/bin/yajsw/` 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 the `<YAJSW_HOME>/conf/` directory.

Sample `wrapper.conf` file

```
#********************************************************************
# 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
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}\repository\endorsed\${java_home}\jre\lib\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root="/\nwrapper.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=${carbon_home}\repository\conf\log4j.properties
wrapper.java.additional.16 = -Dcarbon.config.dir.path=${carbon_home}\repository\conf
wrapper.java.additional.17 = -Dcarbon.logs.path=${carbon_home}\repository\logs
wrapper.java.additional.18 = -Dcomponents.repo=${carbon_home}\repository\components\plugins
wrapper.java.additional.19 = -Dconf.location=${carbon_home}\repository\conf
wrapper.java.additional.20 = -Dcom.atomikos.icatch.file=${carbon_home}\lib\transactions.properties
wrapper.java.additional.21 = -Dcom.atomikos.icatch.hide_init_file_path=true
wrapper.java.additional.22 = -Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true
wrapper.java.additional.23 = -Dcom.sun.jndi.ldap.connect.pool.authentication=simple
wrapper.java.additional.24 =
Setting up Carbon Home:
Extract the Carbon-based product that you want to run as a Windows service, and then set the Windows environment variable `CARBON_HOME` to the extracted product directory location. For example, if you want to run ESB 4.5.0 as a Windows service, you would set `CARBON_HOME` to the extracted `wso2esb-4.5.0` directory.

Test the service in console mode:
You can verify that YAJSW is configured correctly for running the Carbon-based product as a Windows service.

- Open a Windows command prompt and execute the `runConsole.bat` script from the `<YAJSW_HOME>/bat/` directory as shown below.

```bash
cd C:\DocumentsandSettings\yajsw_home\bat\runConsole.bat
```

If the configurations are set properly for YAJSW, you will see console output similar to the following:

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.

Install and run the product as a service:
Execute the relevant script as explained below.

- First, install the WSO2 product as a Windows service, by executing the following command in the `<YAJSW_HOME>/bat/` directory:

```bash
installService.bat
```

The console will display a message confirming that the WSO2CARBON service was installed.
4. a. Start the service by executing 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" -Xmx300m -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"
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
****************************************************************************
Service "WS02CARBON" installed
Press any key to continue...
```

b. Stop and uninstall service: Execute the relevant scripts as shown below.

5. **Stop and uninstall service**: Execute the relevant scripts as shown below.

a. 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:

```
C:\Documents and Settings\yajsw_home\bat>stopService.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" -Xmx300m -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"
Yajsw: yajsw-stable-11.03
OS : Windows XP/5.2/ amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 12:09:00 PM org.apache.commons.vfs2.VfsLog info
INFO: Using "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\vfs_cache" as temporary files store.
platform null
****************************************************************************
Service "WS02CARBON" started
Press any key to continue . . .
```
b. To uninstall the service, execute the following command in the same console window:

```
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 -Djna.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
Dec 30, 2012 11:11:31 AM org.apache.commons.vfs2.VfsLog info INFO: Using "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\vfs_cache" as temporary files store.
platform null
************************** STOPPING "WSO2CARBON" **************************
Service "WSO2CARBON" stopped
Press any key to continue . . .
```

### Installing multiple products as Windows services

The following instructions explain how to install multiple WSO2 servers in a single computer. In this scenario, you simple have to maintain separate YAJSW configuration files for each product. For example, consider that you need to install WSO2 ESB and WSO2 DSS as Windows services and follow the instructions given below.

1. Download and unzip WSO2 ESB and WSO2 DSS.
2. Download and unzip yajsw.
3. Create two directories ‘esb_service’ and ‘dss_service’.
4. Copy the `<YAJSW_HOME>` directory to ‘esb_service’ and ‘dss_service’ separately. Now you will have two separate yajsw directories for the two products.
5. Now, update the wrapper.conf file for each of the products, which is stored in the esb_service/<ESB_YAJSW_HOME>/conf/ and dss_serv
ice/<DSS_YAJSW_HOME>/conf/ directories. You simply have to replace `carbon_home` with `esb_home` and `dss_home` respectively.
6. Copy the `<ESB_HOME>` directory to ‘esb_service’ and the `<DSS_HOME>` directory to ‘dss_service’.
7. Set port offset for DSS to ‘1’ in the `<DSS_HOME>/repository/conf/carbon.xml` file. This will ensure that the DSS service will run on https port 9444 (default 9443 + 1). WSO2 ESB will be running on the default port 9443.
8. Set the ESB_HOME, DSS_HOME and JAVA_HOME environment variables, which points to the extracted folders of each service.
9. Now, update the wrapper.conf file for each of the products, which is stored in the esb_service/<ESB_YAJSW_HOME>/conf/ and dss_serv
ice/<DSS_YAJSW_HOME>/conf/ directories. You simply have to replace `carbon_home` with `esb_home` and `dss_home` respectively.
Only applicable to WSO2 ESB 4.9.0 and WSO2 EI 6.x.x versions!

You must manually add the following property to the `wrapper.conf` file to avoid errors in the management console:

```java
wrapper.java.additional.26 = -Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false
```

If this property is not added, you may come across an exception that will result in blank pages when you log in to the ESB’s management console.

10. Navigate to the `esb_service/<ESB_YAJSW_HOME>/bin` directory and execute the scripts as shown below.
   a. Run `installService.bat` to install the Windows service.
   b. Run `startService.bat` to start the service.

11. Do the same above for the ‘dss_service’ as well.

12. Right click on My Computer -> Manage. Then click Services and Applications -> Services. You can see both ESB and DSS services running.

You can stop or restart the services from the UI as shown in the diagram above. Alternatively, you can go to the `<YAJSW_HOME>/bin` directory for each product and execute the `stopService.bat` and `uninstallService.bat` scripts to stop and uninstall Windows services.

13. You can now open the management consoles of the two products with the following urls:
   - For DSS: https://localhost:9444/carbon.

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.
See the following for more details:

- Adding Datasources
- Configuring an RDBMS Datasource
- Configuring a Custom 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 an 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:

**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:
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:

![Add New Data Source](image-url)
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</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_NONE</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_READ_COMMITTED</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_READ_UNCOMMITTED</td>
</tr>
<tr>
<td></td>
<td>- TRANSACTION_REPEATABLE_READ</td>
</tr>
<tr>
<td></td>
<td>- 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>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>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>For a true value to have any effect, the validationQuery parameter must be set to a non-null string.</td>
<td></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>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 [propertyName=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 thisLinux 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>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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 to the removeAbandonedTimeout value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if log Abandoned 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>
<tr>
<td>Alternate User Name Allowed (boolean)</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Configuring a Custom Datasource

When adding a datasource, if you select the custom datasource type, the following screen will appear:

![New Data Source](image)

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 InMemoryDataSource.

A tabular datasource is typically associated with a SQL data services query. WSO2 products use an internal SQL parser to execute SQL against the custom datasource. For more information, see a sample data service descriptor at InMemoryDSSample. Carbon datasources also support tabular data with the org.wso2.carbon.dataservices.core.custom.datasource.CustomTabularDataSourceReader 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.

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>:

```
<configuration>
  <customDatasourceProps>
    <property name="inmemory_data_source_schema">["Vehicle":
      ["ID", "make", "model", "year", "mileage"]
      , "Vehicle":
      ["ID", "make", "model", "year", "mileage"]
      , "Vehicle":
      ["ID", "make", "model", "year", "mileage"]
      , "Vehicle":
      ["ID", "make", "model", "year", "mileage"]
      , "Vehicle":
      ["ID", "make", "model", "year", "mileage"]
      , "Vehicle":
      ["ID", "make", "model", "year", "mileage"]
    ]
  </property>
</customDatasourceProps>
```

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.

Managing Users, Roles and Permissions

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the user-mgt.xml file found in the <PRODUCT_HOME>/repository/conf/ directory. The following documentation explains how users, roles and permissions can be managed using the management console of WSO2 products.

WSO2 products support the role-based authentication model where privileges of a user are based on the role attached. Each role is configured with zero or more permissions. Therefore, the set of permissions owned by a user is determined by the roles assigned to that user. If a user has several roles assigned, their permissions are added together.

By default, all WSO2 products have the following roles configured:

- **Admin** - Provides full access to all features and controls. By default, the admin user is assigned to both the Admin and the Everyone roles.
- **Internal/Everyone** - Every new user is assigned to this role by default. It does not include any permissions.
- **Internal/System** - This role is not visible in the Management Console.

More roles may be configured by default, depending on the type of features installed in your product. For example, in WSO2 Storage Server (which has the Cassandra feature and RSS Manager feature installed), the following roles will also be defined by default: Internal/Cassandra and Internal/RSSManager.
Before you begin your configurations, note the following:

- Only system administrators or other users with Security level permissions can add, modify and remove users and roles. For more information on permissions, see Role-based Permissions.
- Your product has a primary user store where the users/roles that you create using the management console are stored by default. The default RegEx configurations for this user store are as follows. RegEx configurations ensure that parameters like the length of a user name/password meet the requirements of the user store.

<table>
<thead>
<tr>
<th>Java RegEx</th>
<th>JavaScript RegEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>PasswordJavaRegEx--------</td>
<td>^[\S]{5,30}$</td>
</tr>
<tr>
<td>PasswordJavaScriptRegEx--</td>
<td>^[\S]{5,30}$</td>
</tr>
<tr>
<td>UsernameJavaRegEx--------</td>
<td>^~!#$;%*+={}{3,30}$</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx--</td>
<td>^[\S]{3,30}$</td>
</tr>
<tr>
<td>RolenameJavaRegEx--------</td>
<td>^~!#$;%*+={}{3,30}$</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx--</td>
<td>^[\S]{3,30}$</td>
</tr>
</tbody>
</table>

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 <PRODUCT_HOME>/repository/conf/user-mgt.xml.

- The permission model of WSO2 products is hierarchical. Permissions can be assigned to a role in a fine-grained or a coarse-grained manner.

See the following topics for instructions:

- Changing a Password
- Configuring Roles
- Configuring Users
- Role-based Permissions
- Managing User Attributes
- Removing References to Deleted User Identities in WSO2 Products

### Changing a Password

If you are a user with admin privileges, you can change your own password or reset another user's password using the the management console as explained below.

To change a user's password:

1. Log in to the management console of your product.
2. On the Main tab, click List under Users and Roles.

3. To change your own password, click Change My Password, enter your current password and new password, and click Change.

4. If you are an admin user and need to change another user's password (such as if they have forgotten their current password and need you to...
reset it), do the following:
   a. Click Users.
   b. Find the user's account on the Users screen and click Change Password in the Actions column.
   c. Enter a new temporary password and click Change.
   d. Inform the user of their new temporary password and instruct them to log in and change it as soon as possible.

**Configuring Roles**

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the user-mgt.xml file found in the <PRODUCT_HOME>/repository/conf/ directory. The instructions given in this topic explain how you can add and manage user roles from the management console.

Roles contain permissions for users to manage the server. You can create different roles with various combinations of permissions and assign them to a user or a group of users. Through the Management Console, you can also edit and delete an existing user role.

WSO2 supports the role-based authentication model where privileges of a user are based on the role to which it is attached. By default, WSO2 products come with the following roles:

- **Admin** - Provides full access to all features and controls. By default, the admin user is assigned to both the Admin and the Everyone roles.
- **Everyone** - Every new user is assigned to this role by default. It does not include any permissions by default.
- **System** - This role is not visible in the Management Console.

If a user has several assigned roles, their permissions are added together.

- Adding a user role
- Editing or deleting a role
- Updating role names

**Adding a user role**

Follow the instructions below to add a user role.

1. On the Main tab in the management console, click Add under Users and Roles.
2. Click Roles. This link is only visible to users with Security level permissions role. By default, the admin user has this permission enabled. For more information on permissions, see Role-based Permissions.
3. Click Add New Role. The following screen appears:

   **Add Role**

   **Step 1: Enter role details**

   ![Enter role details form]

   4. Do the following:
      a. In the Domain list, specify the user store where you want to create this role. This list includes the primary user store and any other secondary user stores that are configured for your product. For information on ow user stores (which are repositories storing information about users and roles) are set up and configured, see Configuring User Stores.
      b. Enter a unique name for this role.
      c. Click Next.
   5. Select the permissions that you want users with this role to have. For more information on permissions, see Role-based Permissions. Note that when you assign this role to a user, you can override the role’s permissions and customize them for the user.
   6. Select the existing users to whom this role should be assigned. You can also assign this role to users later, but if you are creating this role in an external user store that does not allow empty roles, you must assign it to at least one user. You can search for a user by name, or view all users by entering * in the search field.
   7. Click Finish.

The role is created and is listed on the Roles page. You can now edit the role as needed.

**Editing or deleting a role**

If you need to do modifications to a role, select the domain (user store) where the role resides, and then use the relevant links in the Actions column on the Roles screen:
1. Rename the role
2. Change the default permissions associated with this role
3. Assign this role to users
4. View the users who are assigned this role
5. Delete the role if you no longer need it

If the role is in an external user store to which you are connected in read-only mode, you will be able to view the existing roles but not edit or delete them. However, you can still create new editable roles.

Updating role names

If you need to do modifications to the role names, you need to do one of the following:

- Update before the first startup (recommended)
- Update after the product is used for some time

**Update before the first startup (recommended)**

The default role names (admin and everyone) can be changed before starting the WSO2 product by editing `<PRODUCT_HOME>/repository/conf/user-mgt.xml`. For more information on configuring the system administrator, see Configuring the System Administrator.

```xml
<Configuration>
  <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/WSO2CarbonDB</Property>
  <Property name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
</Configuration>
```

The following are the changes that need to be made in the configurations above:

- Change `<AdminRole>admin</AdminRole>` to `<AdminRole>administrator</AdminRole>`.
- Change `<EveryOneRoleName>everyone</EveryOneRoleName>` to `<EveryOneRoleName>Your role</EveryOneRoleName>`.

**Update after the product is used for some time**

You do not have to do this when updating before the first startup. The following steps guide you through updating the role names:

1. Do the configuration changes indicated in the above section.
2. You need to do the following user store level changes for existing users if you have changed the role names as mentioned earlier.
   - If you are connected to JDBCUUserStoreManager you need to update the UM_USER_ROLE table with the existing users after changing the admin and everyone role names. Also if you have changed the permission of everyone role, the UM_ROLE_PERMISSION has to be updated with the permissions to the new role.
   
   The schema can be located by referring to the data source defined in the user-mgt.xml file. The data source definition can be found under `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml`.

   - If you are connected to ReadWriteLdapUserStoreManager, you need to populate the members of the previous admin role to the new role under the Groups. For more information, see Configuring User Stores.
3. After the changes, restart the server.

**Configuring Users**

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. The instructions given in this topic explain how you can add and manage users from the management console.

To enable users to log into the management console, you create user accounts and assign them roles, which are sets of permissions. You can add individual users or import users in bulk.

- **Adding a new user and assigning roles**
Adding a new user and assigning roles

Add the GetRoleListOfInternalUserSQL property within the <Realm> section in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file as shown below, to avoid case sensitivity issues when creating users.

```xml
<Realm>
  <Configuration>
    <Property name="GetRoleListOfInternalUserSQL">SELECT UM_ROLE_NAME FROM UM_HYBRID_USER_ROLE, UM_HYBRID_ROLE WHERE UPPER(UM_USER_NAME)=UPPER(? ) AND UM_HYBRID_USER_ROLE.UM_ROLE_ID=UM_HYBRID_ROLE.UM_ID AND UM_HYBRID_USER_ROLE.UM_TENANT_ID=? AND UM_HYBRID_ROLE.UM_TENANT_ID=? AND UM_HYBRID_USER_ROLE.UM_DOMAIN_ID=(SELECT UM_DOMAIN_ID FROM UM_DOMAIN WHERE UM_TENANT_ID=? AND UM_DOMAIN_NAME=?)</Property>
  </Configuration>
</Realm>
```

Follow the instructions below to add a new user account and configure its role.

1. On the Main tab in the management console, and click Add under Users and Roles.
2. Click Users. This link is only visible to users with the Admin role.
3. Click Add New User. The following screen will open:

   Home > Configure > Users and Roles > Users > Add User

Add User

Step 1: Enter user name

<table>
<thead>
<tr>
<th>Enter user name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
</tr>
<tr>
<td>User Name*</td>
</tr>
<tr>
<td>Password*</td>
</tr>
<tr>
<td>Password Repeat*</td>
</tr>
</tbody>
</table>

4. Do the following:
   a. In the Domain list, specify the user store where you want to create this user account. This list includes the primary user store and any other secondary user stores that are configured for your product. For information on configuring user stores, see Configuring User Stores.
   b. Enter a unique username and the password that the person will use to log in. By default, the password must be at least five characters and should have at least one character with a capital letter, characters, numbers and special characters.
   c. Click Next.
5. Optionally, select the role(s) you want this user to have. If you have many roles in your system, you can search for them by name.
6. Click Finish.

A new user account is created with the specified roles and is listed on the Users page.

Importing users
In addition to creating users manually, user information stored in a CSV or Excel file can be imported in bulk to a user store configured in your WSO2 product. This possibility is only available if you have a JDBC user store configured for your product.

Note the following before you use this feature:

- The option to import users in bulk is enabled in your product by default. If not, you can enable it by adding the following property to the JDBC user store configured in the user-mgt.xml file (stored in the <PRODUCT_HOME>/repository/conf directory).

  ```xml
  <Property name="IsBulkImportSupported">true</Property>
  ```

- It is recommended to upload a maximum of 500,000 users at a time. If you need to upload more users, you can upload them in separate batches of 50,000 each.
- You can also specify the size of the file that you can upload to the product in the <PRODUCT_HOME>/repository/conf/carbon.xml file using the TotalFileSizeLimit element as shown below. This value is in MB.

  ```xml
  <TotalFileSizeLimit>100</TotalFileSizeLimit>
  ```

Creating a file with users

You must first create a CSV file or an Excel file with the user information. It is possible to import the **username** and **password** directly from the CSV/Excel to the product. You can also assign each user to multiple roles. Here's an example CSV file:

```
UserName,password,role
user1,password123, role=admin:developer
user2,password123, role=admin:tester
user3,password123, role=admin:developer:tester
user4,password123, role=devops
user5,password123, role=devops:tester
```

Make sure you have the roles that you assign to the users available in the system. If not, the server will throw an error. See Configuring Roles for information on adding user roles to the server.

In addition to importing users with their passwords and roles, you can import other user attributes such as **email address**, **full name**, **last name**, **mobile**, **given name** etc. using claim URIs that are defined for attributes. Here's an example of claim URIs that you can defined for your product:

```
http://wso2.org/claims/givenname
http://wso2.org/claims/lastname
http://wso2.org/claims/mobile
http://wso2.org/claims/role
```

To import users with username, password, roles, and other attributes (as claim URIs), create a CSV file as shown in the example below:
Importing users from the CSV/Excel file

To import users in bulk:

1. Log in to the management console of your WSO2 product.
2. In the Configure menu, Under Users and Roles, click Add.
3. Click Bulk Import Users.

4. The user stores configured for your product will be listed in the Domain field. Select the user store to which you want to import the users from the

User Name, password, Claims

If you are using WSO2 Identity Server, you can choose to leave the password empty as shown by the third line in the below sample. To use this option, you need to first enable the Ask Password option for the server.

User Name, Password, Claims
name1, password1, http://wso2.org/claims/emailaddress=name1@wso2.com
name2, password2, http://wso2.org/claims/emailaddress=name2@wso2.com
name3, http://wso2.org/claims/emailaddress=name3@wso2.com
list, upload the CSV or excel sheet, and click Finish.

Home > Configure > Users and Roles > Add > System User Store

### Bulk Import Users

**Bulk import details**

- **Domain**: PRIMARY
- **File containing users**: Choose File users.csv
- **Finish**

The default password of the imported users is valid only for 24 hours. As the system administrator, you can resolve issues of expired passwords by logging in as the Admin and changing the user's password from the User Management -> Users page. The 'Everyone' role will be assigned to the users by default.

---

**Customizing the user’s roles and permissions**

Each role specifies a set of permissions that the user will have when assigned to that role. After creating a user, you can assign and remove roles for that user by clicking Assign Roles in the Actions column. To see which users a role is already assigned to, click View Users next to the role.

You can also customize which permissions apply to this user by clicking View Roles in the Actions column of the Users screen and then selecting the permissions from each role that you want this user to have. For information on permissions, see Role-based Permissions.

---

**Customizing a user’s profile**

Each individual user has a profile that can be updated to include various details. To do this, click User Profile on the Users screen. Make the changes required and click Update. You can also add multiple profiles for a user.

**Note**: You can only add new profiles if you are connected to a JDBC user store. You also need to have administrator privileges.

---

Do the following in order to add new profiles.

1. On the Main tab in the Management Console, click List under Users and Roles.
2. Click Users. This link is only visible to users with the Admin role.
3. Click the User Profile link.
4. You can add multiple profiles using the Add New Profile link and create any number of profiles for your user as long as the user is located in a JDBC user store.

---

**Deleting an existing user**

Follow the instructions below to delete a user.

A user cannot be undone.

---

1. On the Main tab in the management console click List under Users and Roles.
2. Click Users. This link is only visible to users with User Management level permissions. For information on permissions, see Role-based Permissions.
3. In the Users list, click Delete next to the user you want to delete, and then click Yes to confirm the operation.

---

Once a user is deleted, you can remove all references to the deleted user’s identity via the Identity Anonymization tool. For information on building and running the tool, see Removing References to Deleted User Identities in WSO2 Products.

---

**Role-based Permissions**
The User Management module in WSO2 products enable role-based access. With this functionality, the permissions enabled for a particular role determines what that user can do using the management console of a WSO2 product. Permissions can be granted to a role at two levels:

- **Super tenant level**: A role with super tenant permissions is used for managing all the tenants in the system and also for managing the key features in the system, which are applicable to all the tenants.
- **Tenant level**: A role with tenant level permissions is only applicable to individual tenant spaces.

The permissions navigator that you use to enable permissions for a role is divided into these two categories (Super Admin permissions and Admin permissions) as shown below. However, note that there may be other categories of permissions enabled for a WSO2 product, depending on the type of features that are installed in the product.

You can access the permissions navigator for a particular role by clicking Permissions as shown below.

By default, every WSO2 product comes with the following User, Role and Permissions configured:

- The Admin user and Admin role is defined and linked to each other in the user-mgt.xml file, stored in the <PRODUCT_HOME>/repository/conf directory as shown below.

  ```xml
  <AddAdmin>true</AddAdmin>
  <AdminRole>admin</AdminRole>
  <AdminUser>
    <UserName>admin</UserName>
    <Password>admin</Password>
  </AdminUser>
  ```

- The Admin role has all the permissions in the system enabled by default. Therefore, this is a super tenant, with all permissions enabled.

You will be able to log in to the management console of the product with the Admin user defined in the user-mgt.xml file. You can then create new users and roles and configure permissions for the roles using the management console. However, note that you cannot modify the permissions of the Admin role. The possibility of managing users, roles and permissions is granted by the User Management permission. For more information, see Configuring
the User Realm.

Description of role-based permissions

Note that the descriptions given in this document only explains how permissions control access to operations available on the management console.

The descriptions of permissions in the Permissions navigator are as follows:

- The Login permission defined under Admin permissions allows users to log in to the management console of the product. Therefore, this is the primary permission required for using the management console.

- The following table describes the permissions at Super Tenant level. These are also referred to as Super Admin permissions.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration permissions:</td>
<td>The Super Admin/Configuration permissions are used to grant permission to the key functions in a product server, which are common to all the tenants. In each WSO2 product, several configuration permissions will be available depending on the type of features that are installed in the product.</td>
</tr>
<tr>
<td></td>
<td>- Feature Management permission ensures that a user can control the features installed in the product using the management console. That is, the Features option will be enabled under the Configure menu.</td>
</tr>
<tr>
<td></td>
<td>- Logging permission enables the possibility to configure server logging from the management console. That is, the Logging option will be enabled under the Configure menu.</td>
</tr>
<tr>
<td>Management permissions:</td>
<td>The Super Admin/Manage permissions are used for adding new tenants and monitoring them.</td>
</tr>
<tr>
<td></td>
<td>- Modify/Tenants permission enables the Add New Tenant option in the Configure menu of the management console, which allows users to add new tenants.</td>
</tr>
<tr>
<td></td>
<td>- Monitor/Tenants permission enables the View Tenants option in the Configure menu of the management console.</td>
</tr>
<tr>
<td>Server Admin permissions:</td>
<td>Selecting the Server Admin permission enables the Shutdown/Rerst art option in the Main menu of the management console.</td>
</tr>
</tbody>
</table>

- The following table describes the permissions at Tenant level. These are also referred to as Admin permissions.

Note that when you select a node in the Permissions navigator, all the subordinate permissions that are listed under the selected node are also automatically enabled.

<table>
<thead>
<tr>
<th>Permission level</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>When the Admin permission node is selected, the following menus are enabled in the management console:</td>
</tr>
<tr>
<td></td>
<td>- User Store Management: This permission allows users to add new user stores and manage them with the management console. Note that only secondary user stores can be added using this option. See the topic on user store management for more details.</td>
</tr>
<tr>
<td></td>
<td>- Identity Providers: See the topic on working with identity providers for details on how to use this option.</td>
</tr>
<tr>
<td></td>
<td>- Additionally, all permissions listed under Admin in the permissions navigator are selected automatically.</td>
</tr>
<tr>
<td>Admin/Configure</td>
<td>When the Admin/Configure permission node is selected, the following menus are enabled in the management console:</td>
</tr>
<tr>
<td></td>
<td>- Main menu/PAP: See the topic on working with entitlement for details on how to use this option.</td>
</tr>
<tr>
<td></td>
<td>- Main menu/PDP: See the topic on working with entitlement for details on how to use this option.</td>
</tr>
<tr>
<td></td>
<td>- Configure menu/Server Roles: See the topic on server roles for more details.</td>
</tr>
<tr>
<td></td>
<td>- Tools menu/Try it (XACML): See the topic on working with the Try it tool for details on how to use this option.</td>
</tr>
<tr>
<td></td>
<td>- Additionally, all permissions listed under Configure in the permissions navigator are selected automatically.</td>
</tr>
<tr>
<td>Admin/Configure/Security</td>
<td>When the Admin/Configure/Security permission node is selected, the following menus are enabled in the Configure menu of the management console:</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>- Claim Management: See the topic on claim management for details on how to use this option.</td>
<td>- Keystores: See the topic on keystores for details on how to use this option.</td>
</tr>
<tr>
<td>- Service Principle (Kerberos KDC): See the topic on kerberos security for details on how to use this option.</td>
<td>- Email Templates: See the topics on email templates for details on how to use this option.</td>
</tr>
<tr>
<td>- This permission will also enable the Roles option under Configure/Users and Roles. See the topic on configuring users, roles and permissions for more information.</td>
<td>- Additionally, all permissions listed under Security in the permissions navigator are selected automatically.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin/Configure/Security/Identity Management/User Management</th>
<th>This permission enables the possibility to add users from the management console. That is, the Users option will be enabled under Configure/Users and Roles.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin/Configure/Security/Identity Management/Password Management</td>
<td>This permission enables the Change Password option for the users listed in the User Management/Users and Roles/Users screen, which allows the log in user to change the passwords.</td>
</tr>
<tr>
<td>Admin/Configure/Security/Identity Management/Profile Management</td>
<td>This permission enables the User Profile option for the users listed in the User Management/Users and Roles/Users screen, which allows the log in user to update user profiles.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin/Manage</th>
<th>When the Admin/Manage permission is selected, the following menus will be enabled in the management console:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Main menu/Service Providers: See the topic on working with service providers for details on how to use this option.</td>
<td>- Tools menu/SAML: See the topic on working with the SAML tool kit for more details.</td>
</tr>
<tr>
<td>- Additionally, all permissions listed under Admin/Manage in the permissions navigator will be enabled automatically.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin/Manage/Resources/Browse</th>
<th>This permission enables the Browse option under the Registry menu in the main navigator. This option allows users to browse the resources stored in the registry by using the Registry tree navigator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin/Manage/Search</td>
<td>This permission enables the Search option under the Registry sub menu in the Main menu. This option allows users to search for specific resources stored in the registry by filling in the search criteria.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin/Monitor</th>
<th>When the Admin/Monitor permission node is selected, the following menus are enabled in the management console:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Monitor menu/System Statistics: See the topic on system statistics for information on how to use this option.</td>
<td>- Monitor menu/soap Message Tracer: See the topic on the SOAP tracer for information on how to use this option.</td>
</tr>
<tr>
<td>- Additionally, all permissions listed under Admin/Monitor in the permissions navigator will be enabled automatically.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin/Monitor/Logs</th>
<th>When the Admin/Monitor/Logs permission node is selected, the following menus are enabled in the management console:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Monitor menu/Application Logs</td>
<td>- Monitor menu/System Logs</td>
</tr>
<tr>
<td>See the topic on system logs for information on how to use these options.</td>
<td></td>
</tr>
</tbody>
</table>

### Managing User Attributes

For user and role management in WSO2 products, it is important to understand how to manage the attributes of users within it. In the products, each user store attribute can be mapped as a claim. Therefore, you need to use the claim management functionality available in the product and properly map your LDAP/AD/JDBC user store attributes with the claim URIs defined by the product. You can also add different claim URIs and manage them using claim management.

The following topics provide instructions on how to manage user attributes in WSO2 products:

- Managing the attributes of a user
- Claim mapping when using multiple user stores
- Attributes with multiple values
- Writing custom attributes
- Authentication using multiple attributes
- Customizing the claim for the user attribute
Managing the attributes of a user

The following are the three main ways to view, add, edit and delete attributes of a user in the product.

1. By accessing the profile of the user and changing the attributes using the Management Console.
   a. Log into the product's Management Console.
   b. On the Configure tab in the Management Console, click Users and Roles.
   c. Click Users. This link is only visible to users with the Admin role.
   d. From the list of users that appear in the resulting page, identify the user whose attributes you want to modify and click User Profile.
   e. Click Update to save changes to the attributes.

2. You can use the RemoteUserStoreManagerService API. This is a SOAP-based API and is very easy to use. Supposing you want to set a user attribute, you can call the following method.

   ```java
   setUserClaimValue("username", "http://wso2.org/claims/emailaddress", "asela@soasecurity.org", null)
   ```

   Here "http://wso2.org/claims/emailaddress" is the claim URI that has been mapped with the user store's email attribute. The last parameter is profile, we can just pass "null", as there is only one profile. You can retrieve the user attribute value as follows.

   ```java
   getUserClaimValue("username", "http://wso2.org/claims/emailaddress", null)
   ```

3. You can use the REST Web service according to the SCIM provisioning specification.

Claim mapping when using multiple user stores

When you are using more than one user store, you must map the attributes correctly using claim management. Under "Mapped Attribute(s)" you need to follow the pattern.
However, for the default user store, you do not need to provide the domain name. As an example, if you have two user stores, one is default and other one with domain “LDAP” then the pattern would be as follows for http://wso2.org/claims/emailaddress.

email;LDAP/mail

**Attributes with multiple values**

If your user store supports having multiple values for attributes, the WSO2 product can view, add, update or delete them (normally LDAP/AD offer support for this). The following are the different ways you can do this.

1. In the product's Management Console, multiple attribute values are separated by comma. If you want to update two email addresses using the user profile UI, you must provide it as follows.

   asela@soasecurity.com,aselapathberiya@soasecurity.com

See the following screen for how this will look in the user interface of the product's Management Console.
2. When using the `RemoteUserStoreManagerService` API, call it as follows.

```java
setUserClaimValue("username", "http://wso2.org/claims/emailaddress", "asela@soasecurity.org,aselapathberiya@gmail.com", null)
```

The GET results are returned in the form of comma separated values for the attribute.

```
"asela@soasecurity.org,aselapathberiya@gmail.com"
```

The following screen shows how this looks in the LDAP.
Writing custom attributes

Supposing the attributes of a user are stored in both the user store (LDAP) and another location (JDBC table), the product needs to retrieve/add the user’s attribute in both these places. In scenarios like this, some customization must be done. To customize this, you can simply extend the current user store manager implementation and write a custom implementation to do it. In the custom user store implementation, you only need to extend the following three methods that help to retrieve/add a user attribute. Other methods can be kept as they are.

- **Method 1.**

  ```java
  public Map<String, String> getUserPropertyValues(String userName, String[] propertyNames, String profileName) throws UserStoreException
  ```

- **Method 2.**

  ```java
  protected abstract void doSetUserClaimValue(String userName, String claimURI, String claimValue, String profileName) throws UserStoreException;
  ```

- **Method 3.**

  ```java
  protected abstract void doSetUserClaimValues(String userName, Map<String, String> claims, String profileName) throws UserStoreException;
  ```
Authentication using multiple attributes

In a user store, each user has different attributes such as uid, cn, email and so on. Some of the attributes can be unique. As an example, normally uid and mail can be unique attributes for user.

Once you connect your LDAP with an application, the application can use one of the unique attributes in LDAP to authenticate the user (as the user name of the user in that application). Considering our example, it can be the uid or mail attribute. Additionally, in some cases, the application can use both attributes. So end users can be authenticated in the application using both their uid or mail.

WSO2 products can be deployed with any LDAP based server and it can expose authentication via a Web Service API, SAML, OAuth, OpenID, etc. By default, WSO2 products configured to authenticate with only one user attribute in the LDAP. This topic provides instructions on how the products can be extended to authenticate users using more than one attribute.

For the purposes of this example, we assume that users need to be authenticated using both their uid and mail attributes in the LDAP.

1. Configure the LDAP user store related configurations using the user-mgt.xml file found in the <PRODUCT_HOME>/repository/conf directory.
   a. Configure UserNameSearchFilter that helps to search for the user object in the LDAP using both mail and uid attributes.
      <Property name="UserNameSearchFilter">(&amp;{(objectClass=person)(|(mail=?)(uid=?)))</Property>
   b. Disable UserDNPattern property, if it is currently enabled.
      <Property name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</Property>
   c. The mail attribute has requirements that are unique. If you are using the mail attribute, you need to open the carbon.xml file found in the <PRODUCT_HOME>/repository/conf directory and uncomment the following.
      <EnableEmailUserName>true</EnableEmailUserName>

2. If you want to work with multiple attributes (basically to retrieve internal roles with multiple attributes), you must add following property in the <PRODUCT_HOME>/repository/conf/user-mgt.xml file.
   <Property name="MultipleAttributeEnable">true</Property>

3. To test this, restart the product and try to log in to the Management Console by providing both the mail and uid with the same password.

Customizing the claim for the user attribute

If you are using multiple attribute authentication and want to customize the claim to be used for user name attribute, do the following.

Edit the following element in the <PRODUCT_HOME>/repository/conf/security/application-authentication.xml file.

```xml
<AuthenticatorConfig name="BasicAuthenticator" enabled="true">
  <Parameter name="UserNameAttributeClaimUri">http://wso2.org/claims/emailaddress</Parameter>
</AuthenticatorConfig>
```

This will return the email address of the authenticated user. It can be configured to return any attribute by changing the 'UserNameAttributeClaimUri' parameter.

Removing References to Deleted User Identities in WSO2 Products

When it comes to data protection in enterprise systems, a key requirement outlined in the General Data Protection Regulation (GDPR) is the right of the data subject to be forgotten, which gives users the right to request an organization to remove their personal data when requested.

To comply with the GDPR right to be forgotten requirement, all products in the WSO2 platform support removing references to personally identifiable information (PII) of a user when a user is deleted from the system. You can use the Identity Anonymization tool to remove references to a user’s PII that can generally be stored in metadata database tables, access logs, audit logs as well as any other log files in any WSO2 product.

The tool is designed to replace all references to a deleted user’s identity with either a randomly generated UUID value, or a pseudonym that you specify when you run the tool.

Note

The following WSO2 products have the Identity Anonymization tool packaged in the product distribution by default. Therefore, if you are using any of the following products, see the relevant product documentation page for product specific instructions on how to run the tool.

- WSO2 Identity Server 5.5.0
  For information on how to run the tool, see Removing References to Deleted User Identities.
- WSO2 Enterprise Integrator 6.2.0
  For information on how to run the tool, see General Data Protection Regulation (GDPR) for WSO2 Enterprise Integrator.
- WSO2 API Manager 2.2.0
  For information on how to run the tool, see General Data Protection Regulation (GDPR) for WSO2 API Manager.
- WSO2 Data Analytics Server 3.2.0
  For information on how to run the tool, see General Data Protection Regulations (GDPR) for WSO2 Data Analytics Server.
- WSO2 Stream Processor 4.1.0
  For information on how to run the tool, see General Data Protection Regulations (GDPR) for WSO2 Stream Processor.
WSO2 also supports extending this tool to include additional modules from which you would want to remove deleted user identities. For information on how to extend the tool, see extending the Identity Anonymization Tool.

The following topics walk you through the process of building the tool and configuring the appropriate files and directories so that you can run the tool in standalone mode to successfully remove references to a deleted user's identity from one or more WSO2 products.

- Building the Identity Anonymization tool
- Configuring the master configuration file
- Running the tool

### Prerequisites

- Download and install Apache Maven.

### Building the Identity Anonymization tool

Follow the steps below to build the tool:

2. In the source that you checked out, navigate to identity-anonymization-tool, and run mvn clean install. This downloads all dependencies and builds the tool in your local repository. You can find the org.wso2.carbon.privacy.forgetme.tool-SNAPSHOT.zip file created in the identity-anonymization-tool/components/org.wso2.carbon.privacy.forgetme.tool/target directory.
3. Unzip the org.wso2.carbon.privacy.forgetme.tool-SNAPSHOT.zip file. This creates the identity-anonymization-tool-SNAPSHOT directory with a directory. The path to the identity-anonymization-tool-SNAPSHOT directory will be referred to as <TOOL_HOME> throughout this section.

The following table describes the purpose of the most important configuration related directories and files of the tool, which are in the <TOOL_HOME>/conf directory:

<table>
<thead>
<tr>
<th>Directory/File name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>config.json</td>
<td>This is the master configuration file. You can configure this file depending on the metadata database tables, access logs, audit logs, or any other log files on which you want the Identity Anonymization tool to run. For information on how to configure this file, see Configuring the master configuration file.</td>
</tr>
<tr>
<td>datasources</td>
<td>This is the default directory where configured datasources are searched for when you run the Identity Anonymization tool. If necessary, you can define your own datasource configurations depending on the databases that you want to connect to, and specify the defined datasource configuration location using command line arguments.</td>
</tr>
<tr>
<td>log-config/patterns.xml</td>
<td>This file should contain all the regex patterns that can be used to find and replace references to deleted user identities in log file entries.</td>
</tr>
<tr>
<td>sql</td>
<td>This directory should include all the SQL files that contain required queries to replace or delete references to deleted user identities.</td>
</tr>
</tbody>
</table>

### Configuring the master configuration file

The master configuration file of the Identity Anonymization tool is the config.json file. Following is a sample config.json file:
You can configure the following in the config.json file based on your requirement:

- **processors** - A list of processors on which you want the tool run. The processors that you can specify are pre-defined. Possible values are RDBMS and log-file.
- **directories** - The definitions of directories on which you want the tool to run. When you specify a directory definition, be sure to either specify the directory path relative to the location of the config.json file, or specify the absolute path to the directory.
- **processor** - The type of processor to use to process instructions in the corresponding directory.
- **extensions** - The extensions to be initialized prior to starting a processor.

**Running the tool**

Navigate to the <TOOL_HOME>/bin directory, and 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>

**Note**

The commands specified above use 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.

Following are details of all possible command line options that you can use when you run the tool:

<table>
<thead>
<tr>
<th>Command Line Option</th>
<th>Description</th>
<th>Required</th>
<th>Sample Command</th>
</tr>
</thead>
</table>
| U                   | The user name of the user whose identity references you want to remove. | Yes | On Linux/Mac OS: ./forgetme.sh -U Sam
<p>|                     |             |          | On Windows: forgetme.bat -U Sam |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default Value</th>
<th>Example Linux</th>
<th>Example Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d</code></td>
<td>The configuration directory to use when the tool is run. If you do not specify a value for this option, the default configuration directory of the tool will be used.</td>
<td>No</td>
<td>On Linux/Mac OS: <code>./forgetme.sh -U Sam -d &lt;TOOL_HOME&gt;/conf</code></td>
<td>On Windows: <code>forgetme.bat -U Sam -d &lt;TOOL_HOME&gt;/conf</code></td>
</tr>
<tr>
<td><code>T</code></td>
<td>The tenant domain of the user whose identity references you want to remove. The default value is <code>carbon.super</code>. For information on working with tenants in WSO2 products, see Working with Multiple Tenants.</td>
<td>No</td>
<td>On Linux/Mac OS: <code>./forgetme.sh -U Sam -T sam.com -TID 1</code></td>
<td>On Windows: <code>forgetme.bat -U Sam -T sam.com -TID 1</code></td>
</tr>
<tr>
<td><code>TID</code></td>
<td>The tenant ID of the user whose identity references you want to remove. The default value is <code>1</code>.</td>
<td>No</td>
<td>On Linux/Mac OS: <code>./forgetme.sh -U Sam -T sam.com -TID 1</code></td>
<td>On Windows: <code>forgetme.bat -U Sam -T sam.com -TID 1</code></td>
</tr>
<tr>
<td><code>D</code></td>
<td>The user store domain name of the user whose identity references you want to remove. The default value is <code>PRIMARY</code>.</td>
<td>No</td>
<td>On Linux/Mac OS: <code>./forgetme.sh -U Sam -D Finance-Domain</code></td>
<td>On Windows: <code>forgetme.bat -U Sam -D Finance-Domain</code></td>
</tr>
</tbody>
</table>
| pu | The pseudonym with which you want to replace references to a deleted user's identity. If you do not specify a pseudonym when you run the tool, a random UUID value is generated as the pseudonym by default to anonymize references to the deleted user's identity. | No | On Linux/Mac OS: .\forgetme.sh -U Sam -pu 123-343-453-545-dfd-4  
On Windows: forgetme.bat -U Sam -pu 123-343-453-545-dfd-4 |
| carbon | The CARBON HOME directory path on which you want to run the tool. You should replace this with the variable $CARBON_HOME used in directories you have configured in the master configuration file. | No | On Linux/Mac OS: .\forgetme.sh -U Sam -carbon /usr/bin/wso2is/wso2is5.5.0  
On Windows: forgetme.bat -U Sam -carbon /usr/bin/wso2is/wso2is5.5.0 |

When you specify the required command line options and run the tool, it generates relevant execution reports with the `Report-<PROCESSOR>-<TIMESTAMP>.txt` naming convention in your current working directory.

### Extending the Identity Anonymization Tool

**THE CONTENT ON THIS PAGE IS STILL A WORK IN PROGRESS.**

If you want to include additional relational databases and log files from which you want to remove references to deleted user identities, you can extend the Identity Anonymization tool to include required relational databases and log files. It is also possible to extend the tool to remove references to deleted user identities from additional modules other than relational databases or log files.

The following topics provide detailed instructions on how you can extend the tool depending on your requirement:

- **Before you begin,**
  - Check out the source of the Identity Anonymization tool from [here](#), and build the tool. For detailed instructions on how to build the tool, see [Building the Identity Anonymization tool](#). Once you build the source you can extract the `identity-anonymization-tool-SNAPSHOT` directory. The path to this directory will be referred to as `<TOOL_HOME>` throughout this section.
  - Extending the tool to remove references from additional relational databases
  - Extending the tool to remove references from additional log files
Extending the tool to remove references from additional relational databases

Follow the steps below if you want to extend the Identity Anonymization tool to include an additional relational database from which you want to remove references to deleted user identities:

1. Create a new directory in `<TOOL_HOME>/conf/sql/`, with an appropriate name based on the relational database table from which you want to remove references.

   If there is an existing directory that serves this purpose, you should use the existing directory instead of creating a new directory.

   For example, if you want to remove references to deleted user identities in a relational database where customer information is stored, you can create a directory named customer.

2. Create an SQL file that includes the required commands, and save the file with an appropriate name in the directory that you created in step 1.

   Ensure that the file is saved with the `.sql` extension.

The SQL statements should either be UPDATE or DELETE statements. The following variables can be used to replace respective values at the time of execution:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value to replace</th>
</tr>
</thead>
<tbody>
<tr>
<td>pseudonym</td>
<td>The pseudonym that should be used to replace a deleted user's identity.</td>
</tr>
<tr>
<td>username</td>
<td>The user name that should be replaced with the pseudonym.</td>
</tr>
<tr>
<td>user_store_domain</td>
<td>The user store domain.</td>
</tr>
<tr>
<td>tenant_domain</td>
<td>The tenant domain.</td>
</tr>
</tbody>
</table>

Following is a sample SQL statement that can be used:

```sql
UPDATE IDN_ASSOCIATED_ID
SET USER_NAME = `pseudonym`
WHERE USER_NAME = `username`
AND DOMAIN_NAME = `user_store_domain`
AND TENANT_ID = (SELECT UM_ID
    FROM UM_TENANT
    WHERE UM_DOMAIN_NAME = `tenant_domain`
    ORDER BY UM_ID DESC LIMIT 1)
```

Note

When you run the Identity Anonymization tool to remove references to a deleted user's identity from relational databases, there can be instances where a user name is stored in different forms in databases. For example, a user name can be stored as `user1` or `user1@ws02.com` or `PRIMARY/user1`. To handle such scenarios, you need to assign a query type to each SQL query used with the Identity Anonymization tool, create a `sql.properties` file corresponding to the SQL query, and then specify the query type in the properties file as follows:

For example, if the SQL file is `customer-access-token.sql`, then the corresponding properties file would be `customer-access-token.sql.properties`.

```
type=<Query-Type>
```

For example, `type=DOMAIN_APPENDED`

Following are the query types that you can use together with the description of each query type:

<table>
<thead>
<tr>
<th>Query type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
</tr>
<tr>
<td>PRIMARY</td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>STORE</td>
<td></td>
</tr>
<tr>
<td>TENANT</td>
<td></td>
</tr>
<tr>
<td>ACCESS</td>
<td></td>
</tr>
<tr>
<td>TOKEN</td>
<td></td>
</tr>
</tbody>
</table>
2. Provide a datasource definition in the `<TOOL_HOME>/conf/datasources` directory to map the directory that you created (i.e., `<TOOL_HOME>/conf/sql/customer`) in step 1. Following is a sample datasource definition:

```xml
<datasources-configuration>
  <datasources>
    <datasource>
      <name>customer</name>
      <description>The datasource used for customer</description>
      <definition type="RDBMS">
        <configuration>
          <url>jdbc:h2:file:WSO2CARBON_DB</url>
          <url>jdbc:mysql://localhost:3306/userdb</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>
          <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
      </definition>
    </datasource>
  </datasources>
</datasources-configuration>
```

Here, the datasource name `<name>customer</name>` maps to the directory that you created in step 1. This datasource can be reused by any of the scripts provided within the `<TOOL_HOME>/conf/sql/customer` directory.

3. The Identity Anonymization tool does not support JNDI. Therefore, if there are JNDI configuration sections, be sure to remove those.

4. Add the corresponding JDBC driver into the `<TOOL_HOME>/lib` directory if it is not already added.

Extending the tool to remove references from additional log files

Follow the steps below if you want to extend the Identity Anonymization tool to include an additional log file from which you want to remove references to deleted user identities:

1. Open the `<TOOL_HOME>/conf/config.json` file, go to the `directories` section, and add a new directory entry. You need to specify the following details when adding a new directory entry:
   - `dir`: The directory where the regex replacement patterns are defined. The specified directory can have multiple RegEx pattern files.
   - `type`: `<deprecated>`
   - `processor`: The name of the processor.
1. log-file-path: The directory where the log files are located.
2. log-file-name-regex: The regular expression to filter log files. Specify this when there are rolling log files, where you have to define which files need processing with the specified replacement logic.

2. Open the `<TOOL_HOME>/conf/patterns.xml` file, and define required patterns using regular expressions to find and replace references to a deleted user’s identity. Following are the variable that you can use in a regular expression:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>${userstoreDomain}</code></td>
<td>Replaces the user store domain name of a deleted user with a specified pseudonym.</td>
</tr>
<tr>
<td><code>${username}</code></td>
<td>Replaces the user name of a deleted user with a specified pseudonym.</td>
</tr>
<tr>
<td><code>${tenantId}</code></td>
<td>Replaces the tenant ID of a deleted user with a specified pseudonym.</td>
</tr>
<tr>
<td><code>${tenantDomain}</code></td>
<td>Replaces the tenant domain of a deleted user.</td>
</tr>
</tbody>
</table>

Following is a sample pattern that you can define in the `<TOOL_HOME>/conf/patterns.xml` file:

```xml
<patterns xmlns="patterns.xsd"
xsi:schemaLocation="patterns.xsd">
  <pattern key="pattern4">
    <detectPattern> (.)*(Initiator : )(.)*${username}</detectPattern>
    <replacePattern>${username}</replacePattern>
  </pattern>
  ...
</patterns>
```

Here, the `<detectPattern>` element contains the pattern to detect in log file entries. The `<replacementPattern>` element contains the variable that should be replaced with the pseudonym.

Extending the tool to remove references from additional modules other than relational databases or log files

Follow the steps below if you want to extend the tool to include additional modules other than relational databases or log files:

1. Import the following maven dependency:

```xml
<dependency>
  <groupId>org.wso2.carbon.privacy</groupId>
  <artifactId>org.wso2.carbon.privacy.forgetme.api</artifactId>
</dependency>
```

2. Implement the `ForgetMeInstruction` interface and the `InstructionReader` interface. Keep the following in mind when you implement the interfaces:

- The `ForgetMeInstruction` interface that you implement should contain a single execution of a single artifact. For example, a single log file or a single datasource that can be processed with a single instruction.
- The `InstructionReader` interface should generate a list of instructions related to a given artifact type in the `config` directory. With regard to RDBMS, the `InstructionReader` interface is responsible for treating a single sql file as a single instruction.
- Make sure you specify a distinctive name for the result of `getType()`. This will be the name of the processor.

```java
public String getType() {return MY_PROCESOR_NAME;}
```

An instruction is considered to be atomic. i.e., An instruction should either be completely processed or should not be processed at all.
3. Follow the steps below to register the `InstructionReader` interface with the Java 8 SPI (Service Provider Interface).
   a. Create the `META-INF/services/org.wso2.carbon.privacy.forgetme.api.runtime.InstructionReader` in the resource directory, which will be packed inside the JAR.
   b. Add the fully qualified class name of the implementation of `InstructionReader` inside the SPI file.

```
org.wso2.carbon.privacy.forgetme.logs.instructions.LogFileInstructionReader
```

4. Compile and build the JAR.
5. Add the pre-built jar to the `<TOOL_HOME>/lib` directory.
   Alternatively, you can build your own distribution of the tool and include it in the product itself.

### Securing Carbon Applications

When you work with WSO2 products, you may host multiple applications (Web applications and Jaggery applications) and expose them to external users. The management console, which is shipped with every WSO2 product is also an application that is deployed in the Carbon server. Before you expose the applications to the external network, be sure to configure the security settings for all your applications.

See the following topics for instructions:

- Enabling HTTPS access to the management console
- Enabling HTTP access to the management console
- Starting the server without the management console
- Enabling role-based permissions for the management console
- Restricting access to Carbon applications
  - For the management console only
  - For Jaggery Apps only
  - For all web applications
  - For web application servlets
- Enabling HTTP Strict Transport Security (HSTS) Headers
  - For the management console
  - For web applications
  - For Jaggery applications
- Preventing browser caching
  - For the management console
  - For web applications
  - For Jaggery applications

#### Enabling HTTPS access to the management console

All WSO2 products expose the management console through the default HTTPS transport, which is configured in the `catalina-server.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat` directory). This transport must be properly configured in this file for the management console to be accessible.

See [HTTPS Servlet Transport](#) for instructions.

#### Enabling HTTP access to the management console

If you are exposing the management console to a secure network, you may sometimes allow HTTP access to the management console. Note that HTTPS access is enabled by default. You can follow the steps given below to enable HTTP access to the management console:

1. See [HTTP Servlet Transport](#) and make sure that the HTTP transport connector is configured for your product. Note that 9763 is the default port that will be used.
2. Open the `carbon.xml` file stored in the `<PRODUCT_HOME>/repository/conf/tomcat` directory and uncomment the following element:

```
<EnableHTTPAdminConsole>true</EnableHTTPAdminConsole>
```

3. Disable secure cookies for the management console. To do this, open the `web.xml` file from the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/WEB-INF` directory and set the `<secure>` property to `false`.


<session-config>
  <cookie-config>
    <secure>false</secure>
  </cookie-config>
</session-config>

You can now start the product server and access the management console through HTTP. Use the following URL: http://localhost:<port>/carbon/admin/login.jsp, where <port> corresponds to the HTTP port configured for the server. The default HTTP port for all WSO2 servers is 9763. However, this may change if a port offset is applied to your server as explained here.

### Starting the server without the management console

If you want to provide access to the production environment without allowing any user group (including admin) to log into the management console, execute the following command:

```
sh <PRODUCT_HOME>/bin/wso2server.sh -DworkerNode
```

If you want to check any additional options available to be used with the startup commands, type `--help` after the command, such as:

```
sh <PRODUCT_HOME>/bin/wso2server.sh --help.
```

### Enabling role-based permissions for the management console

You can grant management console access to selected users by configuring role-based permissions.

### Restricting access to Carbon applications

When hosting your products in production, it's imperative that you restrict the access to the management console from the external network. Additionally, you may also need to restrict access to other applications. Accessing Carbon applications in your server (including the management console) can be restricted to selected IP addresses. You can use the Tomcat servlet filter (`org.apache.catalina.filters.RemoteAddrFilter`) for the purpose of restricting access.

**Note** that you can either restrict access to the management console exclusively, or you can restrict access to all web applications in your server (which includes the management console) at the same time.

#### For the management console only

If you want only selected IP addresses to be able to access the management console, add the required IP addresses to the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml` file as follows:

```
<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.

#### For jaggery Apps only

If you want only selected IP addresses to be able to access the jaggery apps like publisher, store, admin portal which resides in WSO2 API Manager, go to `<PRODUCT_HOME>/repository/deployment/server/jaggeryapps/<jaggeryapp-name>/jaggery.conf` file and add the configuration as follows.
The value of the "allow" key that we defined in this file will apply to the publisher/store/admin portal as the the particular jagger.conf file you add this configuration and, thereby, all outside requests to the publisher/store/admin portal will be blocked.

For all web applications

To control access to all web applications deployed in your server (including the management console), 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.

For web application servlets

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 `web.xml` file illustrates how access to the `login.jsp` of the management console (`/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>
```

Enabling HTTP Strict Transport Security (HSTS) Headers

Enable "HTTP Strict Transport Security headers" (HSTS) for the applications deployed in your server, to confirm that the relevant headers are present in the HTTP response. HSTS is not enabled for applications in WSO2 products by default.
Note that HSTS should not be enabled in development environments because transport security validations can interrupt the development processes by validating signatures of self-signed certificates.

**For the management console**

If the `HttpHeaderSecurityFilter` element is available in the web.xml file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/WEB-INF/` directory) as shown below, it implies that security headers are by default configured for the management consoles of all of your profiles. However, in a production deployment, 'Strict-Transport-Security' needs to be explicitly enabled by replacing the default `<init-param>` values of the `HttpHeaderSecurityFilter` filter.

Shown below is the default filter configuration.

```xml
<!-- Tomcat http header security filter -->
<filter>
    <filter-name>HttpHeaderSecurityFilter</filter-name>
    <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
    <init-param>
        <param-name>hstsEnabled</param-name>
        <param-value>false</param-value>
    </init-param>
</filter>
```

Shown below is how you should explicitly enable HSTS.

```xml
<!-- Tomcat http header security filter -->
<filter>
    <filter-name>HttpHeaderSecurityFilter</filter-name>
    <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
    <init-param>
        <param-name>hstsMaxAgeSeconds</param-name>
        <param-value>15768000</param-value>
    </init-param>
</filter>
```

**For web applications**

Similar to the management console, check whether the `HttpHeaderSecurityFilter` (stored in the `<PRODUCT_HOME>/repository/deployment/server/webapps/` directory) is available in the web.xml file of that particular web application. If the filter is available, enable HSTS as shown below.

```xml
<filter>
    <filter-name>HttpHeaderSecurityFilter</filter-name>
    <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>HttpHeaderSecurityFilter</filter-name>
    <url-pattern>*</url-pattern>
</filter-mapping>
```
For Jaggery applications

For Jaggery applications, the HttpSessionSecurityFilter element should be configured in the jaggery.conf file (stored in the `<PRODUCT_HOME>`/repository/deployment/server/jaggeryapps/ directory). This filter configuration is applicable to the /dashboard jaggery applications in this location. To enable HSTS for a Jaggery application, change the default filter configuration as shown below.

The default filter configuration:

```
"params" : ["name" : "hstsEnabled", "value" : "false"]
```

The filter configuration after enabling HSTS:

```
"params" : ["name" : "hstsMaxAgeSeconds", "value" : "15768000"]
```

NOTE: Returning HTTP security headers could also be achieved by configuring those headers from the Proxy/LB configuration.

Preventing browser caching

If there are dynamic pages in your application, which also include sensitive information, you need to prevent caching. This can be done by making sure that the applications return the following HTTP security headers in HTTP responses.

```
Expires:0
Pragma:no-cache
Cache-Control:no-store, no-cache, must-revalidate
```

The following topics explain how you can configure these security headers for different types of applications used in WSO2 products.

For the management console

You can enable these headers for the management console by adding the following configuration to the web.xml file (stored in the `<PRODUCT_HOME>`/repository/conf/tomcat/carbon/WEB-INF/ directory).

```
<filter>
  <filter-name>URLBasedCachePreventionFilter</filter-name>
  <filter-class>org.wso2.carbon.ui.filters.cache.URLBasedCachePreventionFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>URLBasedCachePreventionFilter</filter-name>
  <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

For web applications

If your web application (stored in the `<PRODUCT_HOME>`/repository/deployment/server/webapps/ directory) serves dynamic pages/content, then make sure that either URLBasedCachePreventionFilter or ContentTypeBasedCachePreventionFilter is available in the web.xml file of the particular application.

Note that the applications that are included in the /webapps directory by default in a WSO2 product do not serve sensitive content that requires cache prevention. However, if you are adding any new applications, you need to be mindful of this requirement.
For Jaggery applications

For Jaggery-based applications (stored in the `<PRODUCT_HOME>/repository/deployment/server/jaggeryapps/` directory), either `URLBasedCachePreventionFilter` or `ContentTypeBasedCachePreventionFilter` should be available in the `jaggery.conf` file as shown below.

```
"filters":
[{
"name": "ContentTypeBasedCachePreventionFilter","class":
"org.wso2.carbon.ui.filters.cache.ContentTypeBasedCachePreventionFilter",
"params":
[{
"name": "patterns","value": "text/html",application/json",plain/text"},
{"name": "filterAction","value": "enforce"},
{"name": "httpHeaders","value": "Cache-Control: no-store, no-cache,
must-revalidate, private"}
]
},
```

Working with Composite Applications

The topics below explain Composite Applications (C-Apps) and about how to deploy them:

- Introduction to Composite Applications
- Packaging Artifacts into Composite Applications
- Deploying Composite Applications in the Server

Introduction to Composite Applications

Any WSO2 product can have numerous artifacts such as Axis2 services, data services, endpoints, mediators, registry resources, BPEL workflows etc. Usually, these artifacts are created in a development environment and then moved one by one to staging/production environments. This manual process is time-consuming. Instead, you can bundle the configuration files and artifacts that are in one environment to a Composite Application (C-App) and migrate configurations across environments by deploying the C-App in the new environments.

A C-App is a collection of artifacts deployable on different WSO2 product runtimes. Composite Application Archive (CAR) files, which have the extension .car are used in some deployment options. A C-App and CAR file can have multiple artifacts bundled in it but the runtime that you choose to deploy it in deploys only the artifacts that match its server role. For example, an ESB runtime does not deploy a data service that is bundled in the CAR file unless the default configuration is altered. Therefore, when you deploy a C-App or CAR in a particular WSO2 product, but all its artifacts might not be deployed in that particular product instance.

The structure of a C-App

A typical C-App contains individual directories for its artifacts, along with a file named `artifacts.xml`, which contains metadata about the artifacts that are inside the C-App. The diagram below depicts the structure of a sample C-App:
Given below is a sample `artifacts.xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<artifacts>
  <artifact name="sampleCApp" version="1.0.0" type="carbon/application">
    <dependency artifact="testArtifact01" version="1.0.0" serverRole="ApplicationServer"/>
    <dependency artifact="testArtifact02" version="1.0.0" include="true" serverRole="ApplicationServer"/>
    <dependency artifact="testArtifact03" version="1.0.0" include="true" serverRole="ApplicationServer"/>
  </artifact>
</artifacts>
```

The sample file contains the name of the C-App, its version and the artifact type according to which the deployer for the artifact is identified. For C-Apps, the artifact type of the sample is "carbon/application". In addition, it also contains details about the artifacts that are bundled in the CAR file. If it's an Axis2 service, the file extension used is .aar, if it is a web app, it is .war etc. The artifact type changes accordingly. For example, if it's an Axis2 service, the type is "service/axis2" and if it's a web app, the type is "web/application" or "webapp/jaxws".

**What is the serverRole property**

Although a C-App can have a collection of different artifacts, the runtime that you choose to deploy it in deploys only the artifacts that match its serverRole property. For example, you do not deploy a data service to an ESB runtime. When a C-App is being deployed, it reads the serverRole property that is in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file and deploys only the artifacts that match the serverRole value in the file.

Each product has a different default serverRoles property as follows:

<table>
<thead>
<tr>
<th>WSO2 product</th>
<th>serverRole value</th>
<th>Sample artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSO2 Application Server</td>
<td>ApplicationServer</td>
<td>foo.aar, jax-wx.war</td>
</tr>
<tr>
<td>WSO2 Enterprise Service Bus</td>
<td>EnterpriseServiceBus</td>
<td>proxy.xml</td>
</tr>
<tr>
<td>WSO2 Identity Server</td>
<td>IdentityServer</td>
<td></td>
</tr>
<tr>
<td>WSO2 Data Services Server</td>
<td>DataServicesServer</td>
<td></td>
</tr>
<tr>
<td>WSO2 Governance Registry</td>
<td>GovernanceRegistry</td>
<td></td>
</tr>
<tr>
<td>WSO2 Business Activity Monitor</td>
<td>BusinessActivityMonitor</td>
<td></td>
</tr>
<tr>
<td>WSO2 Business Process Server</td>
<td>BusinessProcessServer</td>
<td>my_bpel.zip</td>
</tr>
<tr>
<td>WSO2 Business Rules Server</td>
<td>BusinessRulesServer</td>
<td></td>
</tr>
<tr>
<td>WSO2 Gadget Server</td>
<td>GadgetServer</td>
<td></td>
</tr>
</tbody>
</table>
You can set the serverRole property in several ways as follows:

- Using the Management Console to set the serverRole property
- Using the carbon.xml file to set the serverRole property
- Using a system property to set the serverRole property

Using the Management Console to set the serverRole property

This is the easiest and the most recommended way to configure your server roles.

1. Log in to the Management Console of your product and click Server Roles in the Configure tab.
2. Click Add New Server Role, enter the role name and click Add. You can add any textual name as a server role without special characters except underscore.
3. Note that the newly added server role is displayed in the list.

![Add Custom Server Role](image)

Note that the newly added server role is displayed in the list.

You can delete the server role by clicking Delete.

**Tip:** You cannot undo a deletion once performed. Users can even delete a default server role. Once deleted, the server role manager will not pick up the deleted server role from the carbon.xml file, next time the server starts.

**Tip:** The server roles that you set through the Management Console cannot be changed using other methods. Server roles are stored in the registry when they are configured through the Management Console. Values in the Registry are always given priority over others.

Using the carbon.xml file to set the serverRole property

Find the serverRoles element in <PRODUCT_HOME>/repository/conf/carbon.xml file. For example,

```xml
<ServerRoles>
  <Role>DataServicesServer</Role>
</ServerRoles>
```

You can also set multiple server roles. For example, if you want the server to deploy both Web services and data services, you can assign both roles to it as follows:

```xml
<ServerRoles>
  <Role>appserver1</Role>
  <Role>dataservices1</Role>
</ServerRoles>
```

Also, ensure that the current server has capability to deploy Axis2 services and data services. When you deploy a C-App on this server, all artifacts that
have the above two server roles get deployed.

**Using a system property to set the serverRole property**

You can use the system property `ServerRoles` to specify the server roles that can be acted by the current product instance. When you start the server, pass the server roles as a comma-separated list. For example,

```
sh wso2server.sh -DserverRoles=appserver1,dataservices1
```

Packaging Artifacts into Composite Applications

After working on your artifacts using the Tooling environment, you bundle them into Composite Applications (C-Apps) that can later be deployed in the server. The steps below describe how to bundle your artifacts into a C-App. When deploying via the management console, you will need to create a Composite Application Archive (CAR) file. See Creating a Composite Application Archive (CAR) file.

Also, you can package individual artifacts into separate C-Apps as well. See Packaging individual artifacts into separate Composite Applications.

1. Open the Tooling interface with all the artifacts/projects that you created. For example, shown below is a proxy service artifact (`StockQuoteProxy.xml file`) created using Tooling.

2. Right-click the **Project Explorer** and click **New -> Project**.

3. From the window that opens, click **Composite Application Project**.
4. Give a name to the **Composite Application** project and select the projects that you need to group into your C-App from the list of available projects below. For example,
5. In the **Composite Application Project POM Editor** that opens, under **Dependencies**, note the information for each of the projects you selected earlier. You can also change the project details here.

Creating a Composite Application Archive (CAR) file
To deploy a C-App via the product's management console, you will need to first create a Composite Application Archive (CAR) file of that C-App.

1. To create the CAR file, do one of the following:
   - Right-click the C-App project and select Export Composite Application Project from the pop-up menu.
   - OR
   - Open the pom.xml file in the Composite Application Project POM Editor and click the button for creating an archive in the upper-right corner.
2. Give the location of the CAR file and the artifacts you want to include in it.

   **Tip:** When you create a CAR file with artifacts, ensure that each artifact name is the same as the relevant artifact file name.

You have now exported all your project's artifacts into a single CAR file. Next, deploy the Composite Application in the server.

**Note**
- In a CAR file, if a particular artifact name is different from the relevant artifact file name, re-deploying the CAR file fails with an error.
- If a CAR file has one or more artifacts that have a artifact name different from the relevant artifact file name, removing those artifacts from memory fails when you delete the CAR file.

**Packaging individual artifacts into separate Composite Applications**

You can also create separate deployable artifacts for each individual artifact in your project. For example, suppose you created an Apache Axis2 Service. When you right-click the Axis2 Service Project, there is an option called Export Project as Deployable Archive. It creates the relevant deployable archive in a location you specify.

Expand to see the archive file types for different app types...

Following are the deployable archives that will be generated for each artifact type.

<table>
<thead>
<tr>
<th>Artifact Type</th>
<th>Archive Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Axis2 Artifact</td>
<td>.aar</td>
</tr>
<tr>
<td>Endpoint Artifact</td>
<td>.xml</td>
</tr>
<tr>
<td>Sequence Artifact</td>
<td>.xml</td>
</tr>
<tr>
<td>Proxy Service Artifact</td>
<td>.xml</td>
</tr>
<tr>
<td>Local Entry Artifact</td>
<td>.xml</td>
</tr>
<tr>
<td>Synapse Configuration</td>
<td>.xml</td>
</tr>
<tr>
<td>ESB Mediator</td>
<td>.jar</td>
</tr>
<tr>
<td>Registry Resource</td>
<td>Registry Resource with necessary metadata</td>
</tr>
<tr>
<td>Third Party Library Artifact</td>
<td>.jar (OSGi Bundle)</td>
</tr>
</tbody>
</table>
Deploying Composite Applications in the Server

After packaging artifacts into a Composite Application, you can deploy it to the server using the Tooling interface.

Alternatively, you can also do this using one of the following options:

- Deploying using the Tooling interface
- Deploying via the product's Management Console
- Deploying via hot deployment
- Deploying using the Maven plug-in

Before you begin, package your artifacts into a Composite Application. Note that the examples here are given using WSO2 ESB, but the steps are the same for any WSO2 server.

You need to give the same name, which you gave for the artifact in the artifacts.xml file of the Composite Application.

Deploying using the Tooling interface

If you have already added a WSO2 product server instance to Tooling, right click on it in the Servers tab, and click Add and Remove...
Then, select the Composite Application you want to deploy from the Available list, click Add to move it into the Configured list, and then click Finish.
1. In the Tooling interface, navigate to **Developer Studio Dashboard** and click **Server** under **Add Server**.

2. In the **Define a New Server** dialog box, expand the WSO2 folder and select the version of your server. In this case, it is **WSO2 ESB Server 5.0.0**.
3. Click **Next**. In the CARBON_HOME field, provide the path to your product's home directory and then click **Next** again. For example,
4. Review the default port details for your server and click **Next**.
   Typically, you can leave these unchanged but if you are already running another server on these ports, give unused ports here.

   **Tip:** See Default Ports of WSO2 Products for more information.
4. To deploy the C-App project to your server, select SampleServicesCompositeApplication from the list, click Add to move it into the Configured list, and then click Finish.
6. Note that your server is now added to the Tooling interface.

On the tab, note that the server is currently stopped. Click the "start the server" icon on the toolbar. If prompted to save changes to any of the artifact files you created earlier, click Yes.

As the server starts, the Console tab appears. Note messages indicating that the Composite app was successfully deployed. The C-App is now available in the product's Management Console, under Manage -> Carbon Applications -> List.
If you do not use the Tooling interface to deploy your artifacts to the server, you can alternatively do that using the product's Management Console or via hot deployment.

**Deploying via the product’s Management Console**

You can also deploy a C-App via the product's Management Console. To do this you will first need to create a CAR file and then deploy the created CAR file as follows:

1. Click the **Main** tab on the Management Console, go to **Manage** -> **Carbon Applications** and then click **Add**.
   
   The **Add Carbon Applications** screen appears.

2. Click **Choose File**, select your CAR file and click **Upload**.
   
   The CAR files that you upload are dropped to the `<PRODUCT_HOME>/tmp/carbonapps/{tenant-ID}/` directory.

3. Refresh the browser to see that the CAR file has been deployed.

4. Click the **Main** tab on the Management Console, go to **Manage** -> **Carbon Applications** and then click **List**. If successfully deployed, the CAR file appears here.

   It is not recommended to use the Management Console to edit the artifacts that have been deployed to your server using a CAR file.

**Deploying via hot deployment**

You can deploy a C-App by directly saving it to the `<PRODUCT_HOME>/repository/deployment/server/carbonapps/` directory. If you are running products in a cluster, use the Deployment Synchronizer to keep the configurations on all nodes of the cluster in sync.

**Deploying using the Maven plug-in**

1. Open the `pom.xml` file of the Composite Application Project. In the **Source** view, search for `maven-car-deploy-plugin` under the `<plugins>` element and edit the `<trustStorePath>` so that it points to the actual location within the product folder structure. See below for an example:
<plugin>
  <groupId>org.wso2.maven</groupId>
  <artifactId>maven-car-deploy-plugin</artifactId>
  <version>1.1.1</version>
  <extensions>true</extensions>
  <configuration>
    <carbonServers>
      <CarbonServer>
        <trustStorePath>/Users/Gillian/ESB/wso2esb-5.0.0/repository/resources/security/wso2carbon.jks</trustStorePath>
        <trustStorePassword>wso2carbon</trustStorePassword>
        <trustStoreType>JKS</trustStoreType>
        <serverUrl>https://localhost:9443</serverUrl>
        <userName>admin</userName>
        <password>admin</password>
        <operation>deploy</operation>
      </CarbonServer>
    </carbonServers>
  </configuration>
</plugin>

2. Using command prompt, navigate to the **ESB Config Project** folder and build the project using the following command:

```bash
mvn clean install
```

If you have more than one project bundled in your C-App project, you need to build each of those projects using the above command.

3. Build and deploy the **Composite Application Project** using one of the following:
   - Open the **pom.xml** file of the **Composite Application Project**. In the **Source** view, add the following line in the **<properties>** section:

```xml
<maven.car.deploy.skip>true</maven.car.deploy.skip>
```

   OR

   - Using command prompt, navigate to the **Composite Application Project** folder and use the following command:

```bash
mvn clean deploy -Dmaven.deploy.skip=true
-Dmaven.car.deploy.skip=false
```

When you access the ESB management console, you can see the artifacts in your Composite Application Project are deployed to your ESB server.

When you deploy a Composite Application, the ESB artifacts are deployed in the below order.

- a. Local entries
- b. Endpoints
- c. Sequences
- d. Message stores
- e. Templates
Working with Features

**Important!**

Note that **WSO2 does not recommend** installing new features on standard products as this practice is not supported by WSO2 Update Manager (WUM). Use the instructions below only for the purpose of demo or test.

Each enterprise middleware product is a collection of reusable software units called features. Similarly, WSO2 Carbon consists of a collection of features where a single feature is a list of components and/or other features. A component of the Carbon platform is a single OSGi bundle or a collection of bundles. Similar to a standard JAR file in Java, a bundle is the modularization unit in OSGi.

Components in the Carbon platform add functionality to Carbon-based products. For example, the statistics component enables users to monitor system and service-level statistics. This component contains two bundles: One is the back-end bundle that collects, summarizes and stores statistics. The other is the front-end bundle, that presents the data to the user through a user-friendly interface.

Equinox P2 is integrated with WSO2 Carbon, which allows users to download WSO2 Carbon or any other WSO2 product and simply extend them by installing various features. The WSO2 Feature Repository consists of features that are bundled into WSO2 products (based on a particular Carbon release). The feature repository for WSO2 products based on Carbon 4.4.x versions is http://product-dist.wso2.com/p2/carbon/releases/wilkes/.

- Installing features from the management console
- Installing features using pom files
- Uninstalling features (using the management console)
- Recovering from unsuccessful feature installation
  - Reverting using the management console
  - Reverting using the command line

Installing features from the management console

Note the following, when you use the **management console** to install a new feature:

- Features will only be installed in the default profile. Features can be installed into other profiles, only using the **POM-based approach**.
- You need to start the server after installing new features from the management console, which will create logs, local indices (solr) and entries in the database. Further, any available webapps will also get deployed. If you want to deploy the installed feature in a clustered environment, these data should first be cleared. It is not required to restart the server if you use the **POM-based approach**. You can use maven to install the feature, then directly take the pack (in which the required feature is installed), and deploy it in the cluster.
Follow the instructions below:

1. Log in to the product's management console.
3. Click Available Features.
4. Select a relevant repository. You can add the repository to get the WSO2 product features of the Carbon 4.4.x platform (Wilkes). If no repositories have been added, or the required repository is not available, add a new repository.

To add a new repository:

- a. Go to the Configure menu on the management console, and click Features.
- b. Go to the Repository Management tab, and click Add Repository.
- c. Provide a convenient name for the repository being added.
- d. Enter the repository location using one of the following methods:
  - i. URL: This option is used when you are adding an external repository.
    - Select the URL option.
    - Enter the Equinox P2 repository URL.
  - ii. Location: This option is used when you are adding a repository that you have downloaded to your computer.
    - Select the Local option.
    - Enter the directory path of the repository on your local machine.
- e. Click Add. The newly added repository will appear in the list of available repositories.

Once you have added the new repository, go back to the Available Features tab to find the required features.

5. Some repositories contain multiple versions of features. If you are only interested in the latest versions, click Show only the latest versions.
6. A feature category is a logical grouping of the features that constitute a particular Carbon-based product. Selecting the Group features by category option enables you to easily view and select the entire list of features of a particular product at once. If you do not select this option when looking for features, you will see an uncategorized, flat feature list from which individual features can be selected separately.

7. Click Find Features. The available features will be listed.
8. Select the features you wish to install.

To find a particular feature, you can use the search function. Enter the name of a feature (or a part of the name) and press Enter.

   ![Filter by feature name](mediation)

This search will return only available features; excluding the ones already installed.

9. Click Install.
10. Verify the feature to be installed and click Next.
11. Read and accept the terms of the license agreement.
12. Click Next. The installation process starts. It may take a few minutes to download the necessary components.
13. Once the installation process is complete, click Finish.
14. Restart the server for the changes to take effect. Based on the newly added features, you will be able to see the additional functionalities.

### Installing features using pom files

When you are using the pom-based approach, be sure that Maven3 and Ant are installed in your system.

Following are the steps to create a new feature installed distribution using a POM file:

1. Copy the sample pom.xml file given below to a directory on your machine.

   ```xml
   <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/maven-v4_0_0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>org.wso2.sample</groupId>
      <artifactId>sample-feature-installation</artifactId>
      <version>1.0.0</version>
   </project>
   ```
<packaging>pom</packaging>
<name>New feature</name>
<url>http://wso2.org</url>

<build>
  <plugins>
    <plugin>
      <groupId>org.wso2.maven</groupId>
      <artifactId>carbon-p2-plugin</artifactId>
      <version>1.5.4</version>
      <executions>
        <execution>
          <id>feature-install</id>
          <phase>package</phase>
          <goals>
            <goal>p2-profile-gen</goal>
          </goals>
          <configuration>
            <profile>default</profile>
            <metadataRepository>file:p2-repo</metadataRepository>
            <artifactRepository>file:p2-repo</artifactRepository>
            <destination>$distribution_name/repository/components</destination>
            <deleteOldProfileFiles>false</deleteOldProfileFiles>
            <features>
              <feature>
                <groupId>org.wso2.carbon.tryit.feature.group</groupId>
                <version>4.3.0</version>
              </feature>
            </features>
            <metadataRepository>file:p2-repo</metadataRepository>
            <artifactRepository>file:p2-repo</artifactRepository>
            <destination>$distribution_name/repository/components</destination>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-antrun-plugin</artifactId>
      <version>1.1</version>
      <executions>
        <execution>
          <phase>package</phase>
          <configuration>
            <tasks>
              <replace token="false" value="true">
                <include name="**/bundles.info"/>
              </replace>
            </tasks>
          </configuration>
          <goals>
            <goal>run</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

<repositories>
  <repository>
    <id>wso2-nexus</id>
    <name>WSO2 internal Repository</name>
    <url>http://maven.wso2.org/nexus/content/groups/wso2-public/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
  </repository>
</repositories>
<releases>
  <repository>
    <pluginRepositories>
      <pluginRepository>
        <id>wso2-maven-releases-repository</id>
        <url>http://maven.wso2.org/nexus/content/repositories/releases/</url>
      </pluginRepository>
      <pluginRepository>
        <id>wso2-maven-snapshots-repository</id>
        <url>http://maven.wso2.org/nexus/content/repositories/snapshots/</url>
      </pluginRepository>
    </pluginRepositories>
  </repository>
</releases>
2. The above sample pom.xml file specifies the default product profile \(<\text{profile}>default</\text{profile}>\) and the corresponding directory path \(\text{dir}="\$\text{distribution}_\text{name}/repository/components/default/configuration/org.eclipse.equinox.simpleconfigurator */\)
   ). If your product is running on a different profile, you need to update the profile name. Read more about profiles from here.

3. Unzip the original product distribution (e.g., wso2carbon-<version>.zip) and copy it to the same location as your pom.xml file.

4. Replace \$\text{distribution}_\text{name} in the pom.xml file with the name of the unzipped product distribution (e.g., wso2carbon-<version>). Note that if your product distribution is not in the same location as your pom.xml file, you can replace \$\text{distribution}_\text{name} with the complete path to your product distribution.

5. Now you need to specify the p2 repository from which the required features should be installed. This can be done in two ways:
   a. Copy the relevant p2 repository to the same directory location as the pom.xml file. Note that your p2 repository can be a local repository, or a download of the WSO2 feature repository (e.g., http://product-dist.wso2.com/p2/carbon/releases/wilkes/).
   b. Replace \text{file}:p2-repo in the pom.xml file with the direct link to the required p2 repository. For example, shown below is how the direct link to the Wilkes p2 repository of WSO2 is given in the pom.xml file:

   ```xml
   ```

6. In the pom.xml file, list down the features you want to install into your product. For example, consider the Try It feature in the Wilkes repository of WSO2. The feature name given in the Wilkes repository is org.wso2.carbon.tryit_4.5.4. Therefore, you can add the feature ID and version to your pom.xml file as shown below. Note that the feature ID should end with 'feature.group'.

   ```xml
   <feature>
     <id>org.wso2.carbon.tryit.feature.group</id>
     <version>4.5.4</version>
   </feature>
   ```

7. Now let's add the feature to the product distribution: Open a terminal, navigate to the location of your pom.xml file and execute the following command:

   ```bash
   mvn clean install
   ```

   Upon successful invocation of the build, the product distribution is provisioned with the new features. This approach is scriptable.

**Uninstalling features (using the management console)**

You can uninstall features from the management console, by following the steps given below.

1. Log in to the product's management console.
2. Go to the Configure menu, and click Features. The Feature Management page will appear.
3. Click Installed Features. The Installed Features page allows you to browse through the list of installed features.
4. Select the features that you need to uninstall. If you wish to uninstall all the features, click Select all in this page.
5. Click Uninstall. A page will appear containing details of the features to be uninstalled.
6. Verify the information and click Next. If the feature is successfully removed, a success message will appear.
If there are other features that depend on the feature that needs to be uninstalled, those dependent sub features need to be uninstalled first, before attempting to uninstall the main feature.

7. Click Finish and restart the server to apply the changes.

Recovering from unsuccessful feature installation

After installing features, if you encounter server issues or startup failures, you can revert the current configuration by restoring a previous one using either the management console or the command line. The latter is recommended if you cannot start the server.

Use the following steps to check your feature installation history and revert the server back to a previous installation. In this recovery process, some features might get installed and some uninstalled.

Reverting using the management console

1. Log in to the management console.
2. Go to the Configure menu, and click Features.
3. Click on the Installation History tab. The Installation History page appears. See the example below.

<table>
<thead>
<tr>
<th>Installation History</th>
</tr>
</thead>
<tbody>
<tr>
<td>This page lists the history of provisioning operations performed on the system. Click on a configuration to view more details.</td>
</tr>
<tr>
<td>Previous Configurations</td>
</tr>
<tr>
<td>Current Configuration</td>
</tr>
<tr>
<td>August 26, 2013 at 13:42:54 IST</td>
</tr>
<tr>
<td>August 25, 2013 at 13:28:34 IST</td>
</tr>
</tbody>
</table>

4. Select the configuration to which you wish to revert.
5. Click Revert, to revert the current configuration to a previous configuration.

Reverting using the command line

If you cannot start the server after an unsuccessful feature installation, use the following steps to revert to a previous installation:

1. Start your product with the -DosgiConsole system property.
2. Once the server is started, type the following command:

```
osgi> getInstallationHistory
```

You will get the following list of statuses:

```
1376883697814 August 19, 2013 at 09:11:37 IST
1376883697957 August 19, 2013 at 09:11:37 IST
1376883700725 August 19, 2013 at 09:11:40 IST
1376883701385 August 19, 2013 at 09:11:41 IST
1376883704884 August 19, 2013 at 09:11:44 IST
1376883712770 August 19, 2013 at 09:11:52 IST
1376883715952 August 19, 2013 at 09:11:55 IST
1376933879416 August 19, 2013 at 23:07:59 IST
1376940017503 August 20, 2013 at 00:50:17 IST
```

3. Check what features are installed and uninstalled them, in a given state, by entering the following command:

```
osgi> getInstallationHistory <timestamp>
```

For example:

```
osgi> getInstallationHistory 1376933879416
```

The output will be as follows:
4. Decide the status to which you need to revert the system, and thereafter use the following command:

```
 osgi> revert <timestamp>
```

For example:

```
 osgi> revert 1376933879416
```

The output will be as follows:

```
Successfully reverted to 1376933879416
Changes will get applied once you restart the server.
```

Working with Multiple Tenants

See the following topics for information on how multitenancy works in WSO2 products.

- Introduction to Multitenancy
- Configuring the Tenant Loading Policy
- Adding New Tenants

Introduction to Multitenancy

The goal of multitenancy 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. Multitenancy 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.

When multitenancy is enabled and a tenant becomes inactive for a long period of time, the tenant is unloaded from the server's memory. By default, the time period is 30 minutes. After that, the tenant has to log in again before sending requests to the server.

You change the default time period allowed for tenant inactiveness by adding `-Dtenant.idle.time=<time_in_minutes>` java property to the product's startup script (`./wso2server.sh` file for Linux and `wso2server.bat` for Windows) as shown below:

```
JAVA_OPTS \n  -Dtenant.idle.time=30 \n```

- Architecture
- Resource sharing
- Tenant loading policy
- Restrictions
- Request dispatching
- Scaling
**Architecture**

The multi-tenant architecture of WSO2 products allows you to deploy Web applications, Web services, ESB mediators, mashups etc. in an environment that supports the following:

- **Tenant isolation:** Each tenant has its own domain, which the other tenants cannot access.
- **Data isolation:** Each tenant can manage its data securely in an isolated manner.
- **Execution isolation:** Each tenant can carry out business processes and workflows independent of the other tenants. No action of a tenant is triggered or inhibited by another tenant.
- **Performance Isolation:** No tenant has an impact on the performance of another tenant.

A tenant is an isolated domain. The users within this domain can manage their own data and perform their own transactions without being affected by actions carried out in other domains.

These domains are allocated server space from the complete server space of a WSO2 product instance which is referred to as the *super tenant*.

The super tenant as well as each individual tenant has its own configuration and context module.

Each tenant has its own security domain. A domain has a set of users, and permissions for those users to access resources. Thus, a tenant is restricted by the users and permissions of the domain assigned to it. The artifact repositories of the tenants are separated from each other.

An individual tenant can carry out the following activities within the boundaries of its own configuration and context module:

- Deploying artifacts
- Applying security
- User management
- Data management
- Request throttling
- Response caching

WSO2 Carbon provides a number of Admin services which have special privileges to manage the server. These admin services are deployed in the super tenant. Other tenants can make use of these admin services to manage their deployment. The admin services operate in a tenant aware fashion. Thus, privileges and restrictions that apply to any client using an admin service are taken into account.

**Resource sharing**

WSO2 Carbon supports the following methods for sharing resources among tenants:

- **Private Jet mode:** This method allows the load of a tenant ID to be deployed in a single tenant mode. A single tenant is allocated an entire service cluster. The purpose of this approach is to allow special privileges (such as priority processing and improved performance) to a tenant.
- **Separation at hardware level:** This method allows different tenants to share a common set of resources, but each tenant has to run its own operating system. This approach helps to achieve a high level of isolation, but it also incurs a high overhead cost.
- **Separation at JVM level:** This method allows tenants to share the same operating system. This is done by enabling each tenant to run a separate JVM instance in the operating system.
- **Native multi-tenancy:** This method involves allowing all the tenants to share a single JVM instance. This method minimises the overhead cost.

**Tenant loading policy**

Lazy loading is a design pattern used specifically in cloud deployments to prolong the initialization of an object or artifact until it is requested by a tenant or an internal process. In WSO2 products based on Carbon 4.4.0 or later versions, you have the option of setting the required tenant loading policy by enabling either **Lazy Loading** or **Eager Loading** of tenants. Additionally, you can separately control the loading policy for web applications and axis2.
services deployed in your tenants using the **GhostDeployment** setting.

See [Configuring the Tenant Loading Policy](#) for more information.

**Restrictions**

The following restrictions are imposed to ensure that each individual tenant has the required level of isolation and maintains fine-grained security control over its own services without affecting the other tenants.

- Only the super tenant can modify its own configuration. In addition, it can add, view and delete tenants.
- When a tenant logs into the system, it can only access artifacts deployed under its own configuration. One tenant cannot manipulate the code of another tenant.
- The super admin or tenant admin can add user stores to their own domain. Dynamic configurations are possible only for secondary user stores and the primary user store is not configurable at run time. This is because primary user stores are available for all tenants and allowing changes to the configuration at run time can lead to instability of the system. Therefore, the primary user store is treated as a static property in the implementation and it should be configured prior to run time.
- A tenants code cannot invoke sensitive server side functionality. This is achieved via Java security.
- Tenants share the transports provided by the system. They are not allowed to create their own transports.

**Request dispatching**

This section describes how the multi tenancy architecture described above works in a request dispatching scenario.

When a Carbon server receives a request, the message is first received by the handlers and dispatchers defined for the server configuration (i.e. super tenant). The server configuration may include handlers that implement cross tenant policies and Service Level Agreement (SLA) management. For example, a priority based dispatcher can be applied at this stage to offer differentiated qualities of service to different clients. Once the relevant handlers and dispatchers are applied, the request is sent to the tenant to which it is addressed. Then the message dispatchers and handlers specific to that tenant will be applied.

The following example further illustrates how message dispatching is carried out in a multi tenant server.

For example, two tenants named foo.com and bar.com may deploy a service named MyService. When this service is hosted on the two tenants, they would have the following URLs.

http://example.com/t/foo.com/services/MyService
http://example.com/t/bar.com/services/MyService

The name of the tenant in the URL allows the tenant to be identified when the Carbon server receives a message which is addressed to a specific client. Alternatively, you may configure a CNAME record in DNS (Domain Name System) as an alias for this information.

If a request is addressed to the MyService service hosted by foo.com, the message handlers and dispatchers of the super tenant will be applied and the tenant foo.com will be identified by the tenant name in the URL. Then the request will be sent to foo.com where it will be processed.

**Scaling**

The multi tenancy architecture described above mainly refers to a scenario where a single instance of a Carbon server acts as a single multi tenant node. In a situation where a very high load of requests are handled, you may need multiple multi tenant nodes. In order to operate with multiple multi tenant nodes, you need load balancing. The load balancer you use also needs to be tenant-aware.

**Configuring the Tenant Loading Policy**

In WSO2 products based on Carbon 4.4.0 or later versions, you have the option of setting the required tenant loading policy by enabling either **Lazy Loading** or **Eager Loading** of tenants. Additionally, you can separately control the loading policy for web applications and axis2 services deployed in your tenants using the **GhostDeployment** setting.

**By default, Lazy Loading is enabled for tenants in all WSO2 products.**

First, read the following descriptions to understand how Lazy loading and Eager loading work:

- **Lazy Loading**: Lazy loading of tenants ensures that all tenants are not loaded at the time the server starts. Instead, the tenants are loaded on demand (upon a request sent to a particular tenant). When a tenant receives a request, the particular tenant and all tenant-specific artifacts, except web applications and axis2 services, get loaded. Therefore, if you have these artifacts (web applications and axis2 services) deployed in your tenants, you will need to separately enable lazy loading for artifacts using the GhostDeployment setting. If lazy loading is not enabled for artifacts, by default, these artifacts will comply with the Eager loading behavior that is explained below.

Tenants (including the tenant-specific artifacts) are **unloaded** from memory if the tenant remains idle for a specified amount of time. You can configure the allowed tenant idle time. See [Configuring Lazy Loading](#) for instructions.

- **Eager Loading**: Unlike lazy loading, eager loading ensures that tenants will be initialized when the server starts (without any delays). You can switch to eager loading if required. Note that you also have the option of enabling eager loading for specific tenants so that only the required tenants will be loaded when the server starts. If you have web applications and axis2 services deployed for your tenants, you will want these artifacts to behave according to the eager loading policy. Therefore, it is recommended that you disable lazy loading for artifacts (GhostDeployment) setting when eager loading is enabled.

Copyright © WSO2 Inc. 2015-2019
When eager loading is enabled, tenants (and the tenant-specific artifacts) are expected to remain in memory without unloading until the server shuts down. However, due to a known issue in the system, tenants (and the tenant-specific artifacts) will unload from the system if the tenants are idle for more than 30 minutes. You can overcome this condition by applying a system property to change the allowed tenant idle time. See Configuring Eager Loading for instructions.

Now, see the instructions given below to configure the required tenant loading policy for your system.

- Configuring Lazy Loading
  - Enabling lazy loading of artifacts (web applications and axis2 services)
  - Configuring the tenant unloading time (for Lazy Loading)
  - Configuring Eager Loading

**Configuring Lazy Loading**

As explained above, Lazy Loading (for tenants) is enabled as the loading policy in WSO2 products, by default. To verify that lazy loading is enabled, open the `carbon.xml` file (stored in the `<PRODUCT_HOME>/repository/conf` directory) and see that `<LazyLoading>` is uncommented as shown below.

```xml
<Tenant>
  <LoadingPolicy>
    <LazyLoading>
      <IdleTime>30</IdleTime>
    </LazyLoading>
    <!--<EagerLoading>
      <Include>*</Include>
    </EagerLoading>-->  
  </LoadingPolicy>
</Tenant>
```

Enabling lazy loading of artifacts (web applications and axis2 services)

If you have lazy loading enabled, and if you have web applications and axis2 services deployed as artifacts in your tenants, you need to separately enable lazy loading for artifacts. This will ensure that these artifacts are first loaded in ghost form. The actual artifacts are deployed only when the artifact is requested.

**Before you enable lazy loading of artifacts**, note the following:

- This setting is only applicable to the following artifacts: web applications and axis2 services.
- This setting only applies if the artifacts (web applications and axis2 services) are using the HTTP/S transport. However, sometimes your axis2 services may be using other transport types such as JMS. In such situations, it is not recommended to enable lazy loading for your artifacts.
- When this setting is enabled for PaaS deployments, lazy loading applies for tenants as well as the tenant artifacts. As a result, for a tenant in a cloud environment, lazy loading is applicable on both levels.
- Also, if an artifact has not been utilized for a certain period of time (tenant idle time), it will be unloaded from memory.

Follow the steps given below.

1. Open the `carbon.xml` file (stored in the `<PRODUCT_HOME>/repository/conf` directory).

```xml
<GhostDeployment>
  <Enabled>false</Enabled>
</GhostDeployment>
```

2. Set the `<Enabled>` property to true.

**Configuring the tenant unloading time (for Lazy Loading)**

If you have Lazy loading enabled, you can configure the allowed tenant idle time. For example, if you set the idle time to 30 minutes, tenants that are idle for more than 30 minutes will be unloaded automatically in your system. You can configure this value using two methods.

- Specify the tenant idle time when you configure the tenant loading policy:
  1. Open the `carbon.xml` file.
  2. Be sure that the `<LazyLoading>` element is enabled and `<EagerLoading>` is commented out as per the default setting.
  3. Set the tenant idle time using the `<IdleTime>` element as shown below.
Alternatively, you can specify the tenant idle time when you start the server:

1. Open the product startup script (./wso2server.sh file for Linux and wso2server.bat for Windows), which is stored in the <PRODUCT_HOME>/bin directory.
2. Add the following system property.

```bash
$JAVA_OPTS \
-Dtenant.idle.time=<value_in_minutes>. \
```

3. Restart the server.

### Configuring Eager Loading

Follow the instructions given below to change the tenant loading policy to eager loading.

**Before you enable eager loading**, note the following:
- Web applications and axis2 services that are deployed as artifacts in your tenants follow the eager loading behavior by default unless the `GhostDeployment` setting is enabled, as explained above under Enabling lazy loading of artifacts. Therefore, when you enable eager loading for your tenants, make sure that `GhostDeployment` is disabled.
- The server startup time may increase depending on the number of tenants and artifacts you have.
- The server’s memory footprint will increase depending on the number of tenants and artifacts that are loaded.

1. Open the `carbon.xml` file from the `<PRODUCT_HOME>/repository/conf` directory.
2. Enable the `<EagerLoading>` element and comment out `<LazyLoading>` as shown below.

```xml
<Tenant>
  <LoadingPolicy>
    <!---<LazyLoading>
    <IdleTime>30</IdleTime>
  </LazyLoading-->  
    <EagerLoading>
      <Include>*,!foo.com,!bar.com</Include>
    </EagerLoading>
  </LoadingPolicy>
</Tenant>
```

3. You can then list the specific tenant domains to which eager loading should apply, by using the `<Include>` element. See the following examples:
   - If the setting should apply to all tenants, add `<Include>*</Include>`.
   - If the setting should apply to all tenants, except `foo.com` and `bar.com`, add `<Include>*,!foo.com,!bar.com</Include>`.
   - If the setting should apply only to `foo.com` and `bar.com`, add `<Include>foo.com,bar.com</Include>`.

Due to a known issue, when eager loading is enabled, tenants will be unloaded from memory if the tenant remains idle for more than 30 minutes. This issue will be resolved in the Carbon 4.5.0 release. In current product releases, you have the option of changing the allowed idle
Adding New Tenants

See the topics given below for instructions.

- Adding tenants using the management console
- Managing tenants using Admin Services

Adding tenants using the management console

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 your product’s 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 the number of users, bandwidth etc.) for the tenant.</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>

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. If you want to view only tenants of a specific domain, enter the domain name in the Enter the Tenant Domain parameter and click Find.

Managing tenants using Admin Services

Other tenant management operations such as activating, deactivating, and updating, which are not available in the management console UI, can be done through one of the following admin services:

1. Open the product startup script (/wso2server.sh file for Linux and wso2server.bat for Windows), which is stored in the `<PRODUCT T_HOME>/bin` directory.
2. Add the following system property.

```
$JAVA_OPTS \n  -Dtenant.idle.time=<value_in_minutes>
```

3. Restart the server.
You can invoke these operations using a SOAP client like SOAP UI as follows:

1. Open the `<PRODUCT_HOME>/repository/conf/carbon.xml` file and set the `HideAdminServiceWSDLs` parameter to false.
2. Start the product server by executing the product startup script from the `<PRODUCT_HOME>/bin` directory:

   **In Linux**
   
   ```
   sh wso2server.sh
   ```

   **In Windows**
   
   ```
   wso2server.bat
   ```

3. Start the SOAP UI client, and import the WSDL of the admin service that you are using:

   - **For TenantMgtAdminService**: `https://localhost:9443/services/TenantMgtAdminService?wsdl`
   - **For RemoteTenantManagerService**: `https://localhost:9443/services/RemoteTenantManagerService?wsdl`

   This assumes that you are running the SOAP UI client from the same machine as the product instance. Note that there are several operations shown in the SOAP UI after importing the `wsdl` file:

Get the list of available admin services

If you want to discover the admin services that are exposed by your product:

a. Execute the following command:

   **In Linux**
   
   ```
   sh wso2server.sh -DosgiConsole
   ```

   **In Windows**
   
   ```
   wso2server.bat -DosgiConsole
   ```

   - When the server is started, hit the enter/return key several times to get the OSGI shell in the console.
   - In the OSGI shell, enter the following: `listAdminServices`

   This will give the list of admin services for your product.
3. Click on the operation to open the request view. For example, to activate a tenant use the `activateTenant` operation. If your tenant domain is abc.com, invoke the `activateTenant` operation with the following request:

```xml
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
  xmlns:ser="http://services.mgt.tenant.carbon.wso2.org">
  <soapenv:Header/>
  <soapenv:Body>
    <ser:activateTenant>
      <!--Optional:-->
      <ser:tenantDomain>abc.com</ser:tenantDomain>
    </ser:activateTenant>
  </soapenv:Body>
</soapenv:Envelope>
```

4. Click on the operation to open the request view. For example, to activate a tenant use the `activateTenant` operation.

5. If your tenant domain is abc.com, invoke the `activateTenant` operation with the following request:

```xml
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
  xmlns:ser="http://services.mgt.tenant.carbon.wso2.org">
  <soapenv:Header/>
  <soapenv:Body>
    <ser:activateTenant>
      <!--Optional:-->
      <ser:tenantDomain>abc.com</ser:tenantDomain>
    </ser:activateTenant>
  </soapenv:Body>
</soapenv:Envelope>
```

### Working with Transports

This chapter contains the following information:
- Introduction to Transports
- Carbon Transports

The screenshots used in this section may vary depending on the product you are using and the specific configuration options.

### Introduction to Transports

WSO2 Carbon is the base platform on which all WSO2 Java products are developed. Built on OSGi, WSO2 Carbon encapsulates all major SOA functionality. It supports a variety of transports, which make Carbon-based products capable of receiving and sending messages over a multitude of transport and application-level protocols. This functionality is implemented mainly in the Carbon core, which combines a set of transport-specific components to load, enable, manage and persist transport related functionality and configurations.

All transports currently supported by WSO2 Carbon are directly or indirectly based on the Apache Axis2 transports framework. This framework provides two main interfaces for each transport implementation.

- `org.apache.axis2.transport.TransportListener` - Implementations of this interface should 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 should specify how a message can be sent out from the Axis2 engine.

Each transport implementation generally contains a transport receiver/listener and a transport sender, since they use the interfaces above. The Axis2
transport framework enables the user to configure, enable and manage transport listeners and senders independent to each other, without having to restart the server. For example, one may enable only the JMS transport sender without having to enable JMS transport listener.

There are two main types of transports: blocking and non-blocking. In a blocking transport, the I/O threads get blocked since the same worker thread that sends the request to the server will remain open to receive the response, until messages are completely processed by the underlying Axis2 engine. However, in non-blocking transports the worker thread that sends the request will not wait for the response and another thread will receive the response. Thereby, non-blocking transports increase the performance of the server.

The transport management capability of WSO2 Carbon is provided by the following feature in the WSO2 feature repository:

**Name:** WSO2 Carbon - Transport Management Feature  
**Identifier:** org.wso2.carbon.transport.mgt.feature.group

If transport management capability is not included in your product by default, you can add it by installing the above feature using the instructions given in the Feature Management section.

**Carbon Transports**

The following transport implementations are supported by default in the WSO2 Carbon platform:

- HTTP Servlet Transport
- HTTPS Servlet Transport
- HTTP-NIO Transport
- HTTPS-NIO Transport
- VFS Transport
- JMS Transport
- MailTo Transport
- TCP Transport
- Local Transport
- UDP Transport
- FIX Transport

**HTTP Servlet Transport**

The HTTP and HTTPS transports in WSO2 products are based on Apache Tomcat's connector implementation. The connector configurations for both HTTP and HTTPS are available in the `catalina-server.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory). The transport class that should be specified for each connector configuration in the `catalina-server.xml` file is as follows:

```
```

See the following topics for instructions on configuring this transport:

- Configuring the HTTP Connector Parameters
- Defining multiple tomcat connectors

**Configuring the HTTP Connector Parameters**

The following table lists the parameters that you can configure for the HTTP connector. Note that these are only a subset of the supported parameters. The servlet HTTP transport uses the `org.apache.catalina.connector.Connector` implementation from Apache Tomcat. So the servlet HTTP transport actually accepts any parameter accepted by the connector implementation. For the complete list of supported parameters, see Apache Tomcat's connector configuration reference.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The port over which this transport receiver listens for incoming messages.</td>
<td>A positive integer less than 65535</td>
<td>9763 for HTTP Connector, 9443 for HTTPS Connector</td>
</tr>
<tr>
<td>redirectPort</td>
<td>If this Connector is supporting non-SSL requests, and a request is received for which a matching <code>&lt;security-constraint&gt;</code> requires SSL transport, Catalina will automatically redirect the request to the port number specified here.</td>
<td>A positive integer less than 65535</td>
<td>9443</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default Value</td>
<td>Constraints</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bindOnInit</td>
<td>Controls when the socket used by the connector is bound. By default it is bound when the connector is initiated and unbound when the connector is destroyed. If set to false, the socket will be bound when the connector is started and unbound when it is stopped.</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>proxyPort</td>
<td>When used, this transport listener will accept messages arriving through a HTTP proxy server which listens on the specified proxy port. Apache mod_proxy should be enabled on the proxy server. All the WSDLs generated will contain the proxy port value as the listener port.</td>
<td>A positive integer less than 65535</td>
<td></td>
</tr>
<tr>
<td>maxHttpHeaderSize</td>
<td>The maximum size of the HTTP request and response header in bytes.</td>
<td>A positive integer</td>
<td>4096</td>
</tr>
<tr>
<td>acceptorThreadCount</td>
<td>The number of threads to be used to accept connections. Increase this value on a multi CPU machine, although you would never really need more than 2. Also, with a lot of non keep alive connections, you might want to increase this value as well.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>maxThreads</td>
<td>The maximum number of worker threads created by the receiver to handle incoming requests. This parameter largely determines the number of concurrent connections that can be handled by the transport.</td>
<td>A positive integer</td>
<td>40</td>
</tr>
<tr>
<td>minSpareThreads</td>
<td>The minimum number of threads always kept running. If not specified, the default will be used.</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>enableLookups</td>
<td>Use this parameter to enable DNS lookups in order to return the actual host name of the remote client. Disabling DNS lookups at transport level generally improves performance. By default, DNS lookups are disabled.</td>
<td>true, false</td>
<td>false</td>
</tr>
<tr>
<td>disableUploadTimeout</td>
<td>This flag allows the servlet container to use a different, longer connection timeout while a servlet is being executed, which in the end allows either the servlet a longer amount of time to complete its execution, or a longer timeout during data upload.</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>connectionUploadTimeout</td>
<td>Specifies the timeout, in milliseconds, to use while a data upload is in progress. This only takes effect if disableUploadTimeout is set to false.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clientAuth</td>
<td>Set to true if you want the SSL stack to require a valid certificate chain from the client before accepting a connection. Set to want if you want the SSL stack to request a client Certificate, but not fail if one is not present. A false value (which is the default) will not require a certificate chain unless the client requests a resource protected by a security constraint that uses CLIENT-CERT authentication.</td>
<td>true, false, want</td>
<td>false</td>
</tr>
</tbody>
</table>
### maxKeepAliveRequests
The maximum number of HTTP requests which can be pipelined until the connection is closed by the server. Setting this attribute to 1 will disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. Setting this to -1 will allow an unlimited amount of pipelined or keep-alive HTTP requests.

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 or any positive integer</td>
<td>100</td>
</tr>
</tbody>
</table>

### acceptCount
The maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full will be refused.

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A positive integer</td>
<td>10</td>
</tr>
</tbody>
</table>

### server
Overrides the Server header for the http response. If set, the value for this attribute overrides the Tomcat default and any Server header set by a web application. If not set, any value specified by the application is used.

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any string</td>
<td>WSO2 Carbon Server</td>
</tr>
</tbody>
</table>

### compression
The Connector may use HTTP/1.1 GZIP compression in an attempt to save server bandwidth.

The acceptable values for the parameter is "off" (disable compression), "on" (allow compression, which causes text data to be compressed), "force" (forces compression in all cases), or a numerical integer value (which is equivalent to "on", but specifies the minimum amount of data before the output is compressed). If the content-length is not known and compression is set to "on" or more aggressive, the output will also be compressed. If not specified, this attribute is set to "off".

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>on, off, force</td>
<td>off</td>
</tr>
</tbody>
</table>

### compressionMinSize
If compression is set to "on" then this attribute may be used to specify the minimum amount of data before the output is compressed.

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A positive integer</td>
<td>2048</td>
</tr>
</tbody>
</table>

### noCompressionUserAgents
Indicate a list of regular expressions matching user-agents of HTTP clients for which compression should not be used, because these clients, although they do advertise support for the feature, have a broken implementation.

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A comma-separated list of regular expressions</td>
<td>empty string</td>
</tr>
</tbody>
</table>

### compressableMimeType
Use this parameter to indicate a list of MIME types for which HTTP compression may be used.

<table>
<thead>
<tr>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A comma-separated list of valid mime types</td>
<td>text/html, text/xml, text/plain</td>
</tr>
</tbody>
</table>

### URIEncoding
This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL.

<table>
<thead>
<tr>
<th>URI encoding</th>
<th>Character set name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO-8859-1</td>
<td></td>
</tr>
</tbody>
</table>

This servlet transport implementation can be further tuned up using the following parameters for **outbound connections**.

<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>PROTOCOL</td>
<td>The version of HTTP protocol to be used for outgoing messages.</td>
<td>No</td>
<td>HTTP/1.0, HTTP/1.1</td>
<td>HTTP/1.1</td>
</tr>
</tbody>
</table>
Transfer-Encoding: Effective only when the HTTP version is 1.1 (i.e. the value of the PROTOCOL parameter should be HTTP/1.1). Use this parameter to enable chunking support for the transport sender.

No | chunked | Not Chunked

SocketTimeout: The socket timeout value in milliseconds, for outbound connections.

No | A positive integer | 60000 ms

ConnectionTimeout: The connection timeout value in milliseconds, for outbound connections.

No | A positive integer | 60000 ms

OmitSOAP12Action: Set this parameter to "true" if you need to disable the soap action for SOAP 1.2 messages.

No | true, false | false

Defining multiple tomcat connectors

You have the option of defining multiple tomcat connectors in the `catalina-server.xml` file. Note that when you define multiple connectors, all the endpoints of the applications deployed in your WSO2 server will still be exposed through all the connector ports. However, you can configure your load balancer to ensure that only the relevant applications are exposed through the required connector port.

Therefore, you can use multiple connectors to strictly separate the applications deployed in your server as explained below.

1. See the example given below where two connectors are defined in the `catalina-server.xml` file.

   ```xml
   <!-- Connector using port 9763 -->
               port="9763"
               ...... />
   <!-- Connector using port 9764 -->
               port="9764"
               ...... />
   ```

2. Configure your load balancer so that the relevant applications are exposed through the required connector port.

HTTPS Servlet Transport

Similar to the HTTP transport, the HTTPS transport is also based on Apache Tomcat's connector implementation. Both the HTTP and HTTPS connector configurations are available in the `catalina-server.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory). The transport class that should be specified for each connector configuration in the `catalina-server.xml` file is as follows:

```
```

See the following topics for instructions on configuring this transport:

- Configuring the HTTPS connector parameters
- Defining multiple tomcat connectors

Configuring the HTTPS connector parameters

In addition to the configuration parameters supported by the HTTP servlet transport, the HTTPS servlet transport supports the configuration parameters listed below. You can configure these parameters in the `catalina-server.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory). For a complete list of supported parameters, see Apache Tomcat's connector configuration reference.

In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

<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>sslProtocol</td>
<td>Transport level security protocol to be used.</td>
<td>No</td>
<td>TLS, SSL</td>
<td>TLS</td>
</tr>
</tbody>
</table>
keystore | Path to the keystore which should be used for encryption/decryption. | Yes | A valid file path to a keystore file

keypass | Password to access the specified keystore. | Yes | A valid password

Defining multiple tomcat connectors

You have the option of defining multiple tomcat connectors in the `catalina-server.xml` file. Note that when you define multiple connectors, all the endpoints of the applications deployed in your WSO2 server will still be exposed through all the connector ports. However, you can configure your load balancer to ensure that only the relevant applications are exposed through the required connector port.

Therefore, you can use multiple connectors to strictly separate the applications deployed in your server as explained below.

1. See the example given below where two connectors are defined in the `catalina-server.xml` file.

   ```xml
   <!-- Connector using port 9763 -->
       port="9763"
       ......
       ....../>

   <!-- Connector using port 9764 -->
       port="9764"
       ......
       ....../>
   ```

2. Configure your load balancer so that the relevant applications are exposed through the required connector port.

HTTP-NIO Transport

HTTP-NIO transport is a module of the Apache Synapse project. Apache Synapse (as well as the WSO2 ESB) ships the HTTP-NIO transport as the default HTTP transport implementation; however, other products can install the feature that has this transport if needed. The two classes that implement the receiver and sender APIs are `org.apache.synapse.transport.nhttp.HttpCoreNIOListener` and `org.apache.synapse.transport.nhttp.HttpCoreNIOSender` respectively. These classes are available in the JAR file named `synapse-nhttp-transport.jar`. This non-blocking transport implementation is one of the secrets behind the superior performance figures of the WSO2 ESB. 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.

HTTP relay transport

Message Relay in older versions of Carbon was simply a message builder-formatter pair. You engage it on a per-content basis. Once engaged for a given content type, messages with that content type are streamed through Carbon. It ran on same old NHTTP transport.

The Relay transport in newer versions of Carbon, is an entire HTTP transport implementation based on HTTP Core NIO. This can be used as an alternative to the NHTTP transport. It doesn't really care about the content type and simply streams all received messages through. It's as if the old Message Relay was engaged on all possible content types. The new transport also has a simpler and cleaner model for forwarding messages back and forth.

To enable this, remove the comment of the relevant HTTP transport entries in the `axis2.xml`. Also, comment out the usual settings for NHTTP transport receiver and sender.

Transport receiver parameters

In transport parameter tables, literals displayed in italic mode under the “Possible Values” column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

<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>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>
</tbody>
</table>
non-blocking  Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.  Yes  true

bind-address  The address of the interface to which the transport listener should bind.  No  A host name or an IP address  127.0.0.1

hostname  The host name of the server to be displayed in service EPRs, WSDLs etc. This parameter takes effect only when the WSDLEPRPrefix parameter is not set.  No  A host name or an IP address  localhost

WSDLEPRPrefix  A URL prefix which will be added to all service EPRs and EPRs in WSDLs etc.  No  A URL of the form <protocol://<hostname>:<port>/

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>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 '</td>
<td>'</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</td>
<td></td>
</tr>
</tbody>
</table>

HTTPS-NIO Transport

HTTPS-NIO transport is also a module that comes from the Apache Synapse code base. Apache Synapse (as well as the WSO2 ESB) ships the HTTPS-NIO transport as the default HTTPS transport implementation; however, other products can install the feature that has this transport if needed. The receiver class is named as follows:

org.apache.synapse.transport.nhttp.HttpCoreNIOSSLLlistener

The sender class is named as follows:

org.apache.synapse.transport.nhttp.HttpCoreNIOSSLSender

As far as the actual implementation of the transport is concerned, these two classes simply extend the HTTP-NIO implementation by adding SSL support to it. Therefore, they support all the configuration parameters supported by the HTTP-NIO receiver and sender. In addition to that, both HTTPS-NIO listener and the HTTPS-NIO sender support the following two parameters. The above mentioned classes are available in the synapse-nhttp-transport.jar bundle.

Transport Parameters (Common to both receiver and the sender):

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Requred</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<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 '</td>
<td>'</td>
</tr>
</tbody>
</table>
keystore

The default keystore to be used by the receiver or the sender should be specified here along with its related parameters as an XML fragment. The path to the keystore file, its type and the passwords to access the keystore should be stated in the XML. The keystore would be used by the transport to initialize a set of key managers.

```xml
<parameter name="keystore">
  <KeyStore>
    <Location>lib/identity.jks</Location>
    <Type>JKS</Type>
    <Password>password</Password>
    <KeyPassword>password</KeyPassword>
  </KeyStore>
</parameter>
```

truststore

The default trust store to be used by the receiver or the sender should be specified here along with its related parameters as an XML fragment. The location of the trust store file, its type and the password should be stated in the XML body. The truststore is used by the transport to initialize a set of trust managers.

```xml
<parameter name="truststore">
  <TrustStore>
    <Location>lib/identity.jks</Location>
    <Type>JKS</Type>
    <Password>password</Password>
  </TrustStore>
</parameter>
```

The HTTPS NIO transport sender supports the concept of custom SSL profiles. An SSL profile is a user defined keystore-truststore pair. Such an SSL profile can be associated with one or more target servers. When the HTTPS sender connects to a target server, it will use the SSL profile associated with the target server. If no custom SSL profiles are configured for the target server, the default keystore-truststore pair will be used. Using this feature the NIO HTTPS sender can connect to different target servers using different certificates and identities. The following table shows how to configure custom SSL profiles. The given example only contains a single SSL profile, but there can be as many profiles as required.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Required</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
</table>
| customSSLProfiles        | Define one or more custom SSL profiles and associate them with target servers. Each profile must be associated with at least one target server. If a profile should be associated with multiple target servers, the server list should be specified as a comma separated list. A target server is identified by a host-port pair. | No       | <parameter name="customSSLProfiles">
|                          |                                                                            |          | <profile>                                                                        |                        |
|                          |                                                                            |          | <servers>www.test.org:80,                                                      |                        |
|                          |                                                                            |          | www.test2.com:9763</servers>                                                  |                        |
|                          |                                                                            |          | <KeyStore>                                                                      |                        |
|                          |                                                                            |          | <Location>/path/to/identity/store                                               |                        |
|                          |                                                                            |          | <Type>JKS</Type>                                                               |                        |
|                          |                                                                            |          | <Password>password</Password>                                                  |                        |
|                          |                                                                            |          | <KeyPassword>password</KeyPassword>                                            |                        |
|                          |                                                                            |          | <TrustStore>                                                                    |                        |
|                          |                                                                            |          | <Location>/path/to/trust/store                                                  |                        |
|                          |                                                                            |          | <Type>JKS</Type>                                                               |                        |
|                          |                                                                            |          | <Password>password</Password>                                                  |                        |
|                          |                                                                            |          | </TrustStore>                                                                  |                        |
|                          |                                                                            |          | </profile>                                                                    |                        |
|                          |                                                                            |          | </parameter>                                                                   |                        |

VFS Transport

**VFS (Virtual File System) transport** implementation is a module which belongs to the Apache Synapse project. The following classes implement the listener and sender APIs.

- org.apache.synapse.transport.vfs.VFSTransportListener
- org.apache.synapse.transport.vfs.VFSTransportSender

The necessary classes can be found in the `synapse-vfs-transport.jar` file. Unlike the transports described previously, VFS transport does not have any global parameters to be configured. Rather, it has a set of service level parameters that needs to be specified for each service. VFS transport
implementation is mainly used and mostly effective in the WSO2 ESB.

Carbon VFS transport supports the **FTPS protocol**. Configuration is identical to other protocols with the only difference being the URL prefixes.

The VFS transport implementation is based on Apache Commons VFS implementation. Therefore `commons-vfs.jar` file should be included in the Carbon classpath to enable the VFS transport.

Since VFS transport deals with file operations, there are instances that these can fail due to unavailability of some resource. In such an instance, the VFS transport is equipped with the following fault-handling mechanism.

When a failure occurs in a file object, it will be marked as a failed record and will be moved to a location (configured by the user) where error file objects are kept. The failed record will be maintained inside a text file (file name is configurable) and the location of that file will be provided by the user. When the next polling iteration is going on, it will check the file against the failed record and if the file is a failed record, it will skip processing and schedule a move task to move that file (the retry duration of the file move task can be configured). It's handled this way because it is a random failure in the move operation.

**VFS service level 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>transport.vfs.FileURI</td>
<td>The file URL from where the input files should be fetched.</td>
<td>Yes</td>
<td>A valid file URL of the form file://file://&lt;path&gt;</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.ContentType</td>
<td>Content type of the files transferred over the transport.</td>
<td>Yes</td>
<td>A valid content type for the files (e.g., text/xml)</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.FileNamePattern</td>
<td>If the VFS listener should read only a subset of all the files available in the specified file URI location, this parameter can be used to select those files by name using a regular expression.</td>
<td>No</td>
<td>A regular expression to select files by name (e.g., *.xml)</td>
<td></td>
</tr>
<tr>
<td>transport.PollInterval</td>
<td>The polling interval in milliseconds for the transport receiver to poll the file URI location.</td>
<td>No</td>
<td>A positive integer</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.ActionAfterProcess</td>
<td>Action to perform over the files after processed by the transport.</td>
<td>No</td>
<td>MOVE, DELETE</td>
<td>DELETE</td>
</tr>
<tr>
<td>transport.vfs.ActionAfterFailure</td>
<td>Action to perform over the files after processed by the transport.</td>
<td>No</td>
<td>MOVE, DELETE</td>
<td>DELETE</td>
</tr>
<tr>
<td>transport.vfs.MoveAfterProcess</td>
<td>The location to move the files after processing.</td>
<td>Required if ActionAfterProcess is MOVE</td>
<td>A valid file URI</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.MoveAfterFailure</td>
<td>The location to move the files after a failure occurs.</td>
<td>Required if ActionAfterFailure is MOVE</td>
<td>A valid file URI</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.ReplyFileURI</td>
<td>The location to which reply files should be written by the transport.</td>
<td>No</td>
<td>A valid file URI</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.ReplyFileName</td>
<td>The name for reply files written by the transport.</td>
<td>No</td>
<td>A valid file name</td>
<td>response.xml</td>
</tr>
</tbody>
</table>

In transport parameter tables, literals displayed in italic mode under the “Possible Values” column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>transport.vfs.Move</td>
<td>The pattern/format of the timestamps added to file names as prefixes when moving files (See the API documentation of java.text.SimpleDateFormat for details).</td>
<td>No</td>
<td>A valid timestamp pattern (e.g., yyyy-MM-dd'T'HH:mm:ss.SSSZ)</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>No</td>
<td>No</td>
<td>true, false</td>
<td>false</td>
</tr>
<tr>
<td>transport.vfs.Streaming</td>
<td>If files should be transferred in streaming mode or not.</td>
<td>No</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.Reconnect Timeout</td>
<td>Reconnect timeout value in seconds to be used in case of an error when transferring files.</td>
<td>No</td>
<td>A positive integer</td>
<td>30 sec</td>
</tr>
<tr>
<td>transport.vfs.MaxRetry Count</td>
<td>Maximum number of retry attempts to carry out in case of errors.</td>
<td>No</td>
<td>A positive integer</td>
<td>3</td>
</tr>
<tr>
<td>transport.vfs.Append</td>
<td>When writing the response to a file, if the response should be appended to the response file this parameter should be set to true. By default the response file will be completely overwritten.</td>
<td>No</td>
<td>true, false</td>
<td>false</td>
</tr>
<tr>
<td>transport.vfs.MoveAfter FailedMove</td>
<td>New destination to move the failed file.</td>
<td>No</td>
<td>A valid file URI</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.Failed Records File</td>
<td>The file name to maintain the list of failure files.</td>
<td>No</td>
<td>A valid file name</td>
<td>vfs-move-failed-records.properties</td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.vfs.Failed Records File</td>
<td>The destination of the failed file.</td>
<td>No</td>
<td>A folder URI</td>
<td>repository/conf/</td>
</tr>
<tr>
<td>Destination</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.vfs.Move</td>
<td>When adding a record to the failed file, entries are logged as: file_name.time_stamp. This will configure the time stamp format.</td>
<td>No</td>
<td>A valid timestamp pattern (e.g: yyyy-MM-dd'T'HH:mm:ss.SSSZ)</td>
<td></td>
</tr>
<tr>
<td>Failed Record Timestamp Format</td>
<td>No</td>
<td>No</td>
<td>dd-MM-yyyy HH:mm:ss</td>
<td></td>
</tr>
<tr>
<td>transport.vfs.Failed Record Next</td>
<td>The time in milli second for the move task to wait until next retry.</td>
<td>No</td>
<td>A positive integer</td>
<td>3000 milliseconds</td>
</tr>
<tr>
<td>Next Retry Duration</td>
<td>No</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


**JMS Transport**

JMS (Java Message Service) transport implementation also comes from the WS-Commons Transports project. All the relevant classes are packed into the
The JMS transport implementation requires an active JMS server instance to be able to receive and send messages. We recommend using Apache ActiveMQ JMS server, but other implementations such as Apache Qpid and Tibco are also supported. You also need to put the client JARs for your JMS server in Carbon classpath. In case of Apache ActiveMQ, you need to put the following JARs in the classpath:

- `activemq-core.jar`
- `geronimo-j2ee-management_1.0_spec-1.0.jar`
- `geronimo-jms_1.1_spec-1.1.1.jar`

These JAR files can be obtained by downloading the latest version of Apache ActiveMQ (version 5.2.0 is recommended). Extract the downloaded archive and find the required dependencies in the `$ACTIVEMQ_HOME/lib` directory. You need to copy these JAR files over to `$CARBON_HOME/repository/components/lib` directory for Carbon to be able to pick them up at run-time.

Configuration parameters for JMS receiver and the sender are XML fragments that represent JMS connection factories. A typical JMS parameter configuration would look like this:

```xml
<parameter name="myTopicConnectionFactory">
    <parameter name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url">tcp://localhost:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</parameter>
    <parameter name="transport.jms.ConnectionFactoryType">topic</parameter>
</parameter>
```

This is a bare minimal JMS connection factory configuration, which consists of four connection factory parameters. JMS connection factory parameters are described in detail below.

### JMS connection factory parameters

In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

<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>java.naming.factory.initial</td>
<td>JNDI initial context factory class. The class must implement the java.naming.spi.InitialContextFactory interface.</td>
<td>Yes</td>
<td>A valid class name</td>
<td></td>
</tr>
<tr>
<td>java.naming.provider.url</td>
<td>URL of the JNDI provider.</td>
<td>Yes</td>
<td>A valid URL</td>
<td></td>
</tr>
<tr>
<td>java.naming.security.principal</td>
<td>JNDI Username.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.naming.security.credentials</td>
<td>JNDI password.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.Transactionality</td>
<td>Desired mode of transactionality.</td>
<td>No</td>
<td>none, local, jta</td>
<td>none</td>
</tr>
<tr>
<td>transport.UserTxnJNDIName</td>
<td>JNDI name to be used to require user transaction.</td>
<td>No</td>
<td>java:comp/UserTransaction</td>
<td></td>
</tr>
<tr>
<td>transport.CacheUserTxn</td>
<td>Whether caching for user transactions should be enabled or not.</td>
<td>No</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>transport.jms.SessionTransacted</td>
<td>Whether the JMS session should be transacted or not.</td>
<td>No</td>
<td>true, false</td>
<td>true if transactionality is &quot;local&quot;</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default Value</td>
<td>Possible Values</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>---------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>transport.jms.SessionAcknowledgment</td>
<td>JMS session acknowledgment mode.</td>
<td>No</td>
<td>AUTO_ACKNOWLEDGE, CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE, SESSION_TRANSACTED</td>
<td></td>
</tr>
<tr>
<td>transport.jms.ConnectionFactoryJNDIName</td>
<td>The JNDI name of the connection factory.</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.jms.ConnectionFactoryType</td>
<td>Type of the connection factory.</td>
<td>No</td>
<td>queue, topic</td>
<td></td>
</tr>
<tr>
<td>transport.jms.JMSSpecVersion</td>
<td>JMS API version.</td>
<td>No</td>
<td>1.1, 1.0.2b</td>
<td></td>
</tr>
<tr>
<td>transport.jms.UserName</td>
<td>The JMS connection username.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.jms.Password</td>
<td>The JMS connection password.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.jms.Destination</td>
<td>The JNDI name of the destination.</td>
<td>No</td>
<td>Defaults to service name</td>
<td></td>
</tr>
<tr>
<td>transport.jms.DestinationType</td>
<td>Type of the destination.</td>
<td>No</td>
<td>queue, topic</td>
<td></td>
</tr>
<tr>
<td>transport.jms.DefaultReplyDestinationJNDIName</td>
<td>JNDI name of the default reply destination.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.jms.DefaultReplyDestinationType</td>
<td>Type of the reply destination.</td>
<td>No</td>
<td>Defaults to the type of the destination</td>
<td></td>
</tr>
<tr>
<td>transport.jms.MessageSelector</td>
<td>Message selector implementation.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.jms.SubscriptionDurable</td>
<td>Whether the connection factory is subscription durable or not.</td>
<td>No</td>
<td>true, false</td>
<td></td>
</tr>
<tr>
<td>transport.jms.DurableSubscriberName</td>
<td>Name of the durable subscriber.</td>
<td>Yes if the subscription durable is turned on</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.jms.PubSubNoLocal</td>
<td>Whether the messages should be published by the same connection they were received.</td>
<td>No</td>
<td>true, false</td>
<td></td>
</tr>
<tr>
<td>transport.jms.CacheLevel</td>
<td>JMS resource cache level.</td>
<td>No</td>
<td>none, connection, session, consumer, producer, auto</td>
<td></td>
</tr>
<tr>
<td>transport.jms.ReceiveTimeout</td>
<td>Time to wait for a JMS message during polling. Set this parameter value to a negative integer to wait indefinitely. Set to zero to prevent waiting.</td>
<td>No</td>
<td>Number of milliseconds to wait</td>
<td></td>
</tr>
<tr>
<td>transport.jms.ConcurrentConsumers</td>
<td>Number of concurrent threads to be started to consume messages when polling.</td>
<td>No</td>
<td>Any positive integer - For topics this must be always 1</td>
<td></td>
</tr>
<tr>
<td>transport.jms.MaxConcurrentConsumers</td>
<td>Maximum number of concurrent threads to use during polling.</td>
<td>No</td>
<td>Any positive integer - For topics this must be always 1</td>
<td></td>
</tr>
<tr>
<td>transport.jms.IdleTaskLimit</td>
<td>The number of idle runs per thread before it dies out.</td>
<td>No</td>
<td>Any positive integer</td>
<td></td>
</tr>
<tr>
<td>transport.jms.MaxMessagesPerTask</td>
<td>The maximum number of successful message receipts per thread.</td>
<td>No</td>
<td>Any positive integer - Use -1 to indicate infinity</td>
<td></td>
</tr>
<tr>
<td>transport.jms.InitialReconnectDuration</td>
<td>Initial reconnection attempts duration in milliseconds.</td>
<td>No</td>
<td>Any positive integer</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2015-2019
JMS transport implementation has some parameters that should be configured at service level, in other words in service XML files of individual services.

### Service level JMS configuration 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>transport.jms.ConnectionFactory</td>
<td>Name of the JMS connection factory the service should use.</td>
<td>No</td>
<td>A name of an already defined connection factory</td>
<td>default</td>
</tr>
<tr>
<td>transport.jms.PublishEPR</td>
<td>JMS EPR to be published in the WSDL.</td>
<td>No</td>
<td>A JMS EPR</td>
<td></td>
</tr>
</tbody>
</table>

**MailTo Transport**

The polling MailTo transport supports sending messages (E-Mail) over SMTP and receiving messages over POP3 or IMAP. This transport implementation is available as a module of the WS-Commons Transports project. The receiver and sender classes that should be included in the Carbon configuration to enable the MailTo transport are `org.apache.axis2.transport.mail.MailTransportListener` and `org.apache.axis2.transport.mail.MailTransportSender` respectively. The JAR consisting of the transport implementation is named `axis2-transport-mail.jar`.

The mail transport receiver should be configured at service level. That is each service configuration should explicitly state the mail transport receiver configuration. This is required to enable different services to receive mails over different mail accounts and configurations. However, transport sender is generally configured globally so that all services can share the same transport sender configuration.

**Service Level Transport Receiver Parameters**

The MailTo transport listener implementation can be configured by setting the parameters described in JavaMail API documentation. For IMAP related properties, see [Package Summary - IMAP](#). For POP3 properties, see [Package Summary - POP3](#). In addition to the parameters described in the JavaMail API documentation, the MailTo transport listener also supports the following transport parameters.

#### Tip

In the following transport parameter tables, the literals displayed in italics under the **Possible Values** column should be considered as fixed literal constant values. Those values can be directly specified in the transport configuration.

<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>transport.mail.Address</td>
<td>The mail address from which this service should fetch incoming mails.</td>
<td>Yes</td>
<td>A valid e-mail address</td>
<td></td>
</tr>
<tr>
<td>transport.mail.Folder</td>
<td>The mail folder in the server from which the listener should fetch incoming mails.</td>
<td>No</td>
<td>A valid mail folder name (e.g., inbox)</td>
<td>inbox folder if that is available or else the root folder</td>
</tr>
<tr>
<td>transport.mail.Protocol</td>
<td>The mail protocol to be used to receive messages.</td>
<td>No</td>
<td>pop3, imap</td>
<td>imap</td>
</tr>
<tr>
<td>transport.mail PreserveHeaders</td>
<td>A comma separated list of mail header names that this receiver should preserve in all incoming messages.</td>
<td>No</td>
<td>A comma separated list</td>
<td></td>
</tr>
<tr>
<td>transport.mail.RemoveHeaders</td>
<td>A comma separated list of mail header names that this receiver should remove from incoming messages.</td>
<td>No</td>
<td>A comma separated list</td>
<td></td>
</tr>
<tr>
<td>transport.mail.ActionAfterProcess</td>
<td>Action to perform on the mails after processing them.</td>
<td>No</td>
<td>MOVE, DELETE</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
transport.mail.ActionAfterFailure | Action to perform on the mails after a failure occurs while processing them. | No | MOVE, DELETE | DELETE

transport.mail.MoveAfterProcess | Folder to move the mails after processing them. | Required if ActionAfterProcess is MOVE | A valid mail folder name

transport.mail.MoveAfterFailure | Folder to move the mails after encountering a failure. | Required if ActionAfterFailure is MOVE | A valid mail folder name

transport.mail.ProcessInParallel | Whether the receiver should incoming mails in parallel or not (works only if the mail protocol supports that - for example, IMAP). | No | true, false | false

transport.ConcurrentPollingAllowed | Whether the receiver should poll for multiple messages concurrently. | No | true, false | false

transport.mail.MaxRetryCount | Maximum number of retry operations to be performed when fetching incoming mails. | Yes | A positive integer

transport.mail.ReconnectTimeout | The reconnect timeout in milliseconds to be used when fetching incoming mails. | Yes | A positive integer

Global Transport Sender Parameters

For a list of parameters supported by the MailTo transport sender, see Package Summary - SMTP. In addition to the parameters described there, the MailTo transport sender supports the following 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>transport.mail.SMTPBccAddresses</td>
<td>If one or more e-mail addresses need to be specified as BCC addresses for outgoing mails, this parameter can be used.</td>
<td>No</td>
<td>A comma separated list of e-mail addresses</td>
<td></td>
</tr>
<tr>
<td>transport.mail.Format</td>
<td>Format of the outgoing mail.</td>
<td>No</td>
<td>Text, Multipart</td>
<td>Text</td>
</tr>
</tbody>
</table>

TCP Transport

The TCP transport implementation is in the Apache WS-Commons Transports project. The two classes that act as the transport listener and the sender are org.apache.axis2.transport.tcp.TCPServer and org.apache.axis2.transport.tcp.TCPTransportSender respectively. In order to use the transport axis2-transport-tcp.jar should be added to the Carbon classpath.

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>port</td>
<td>The port on which the TCP server should listen for incoming messages</td>
<td>No</td>
<td>A positive integer less than 65535</td>
<td>8000</td>
</tr>
<tr>
<td>hostname</td>
<td>The host name of the server to be displayed in WSDLs etc</td>
<td>No</td>
<td>A valid host name or an IP address</td>
<td></td>
</tr>
</tbody>
</table>

Transport sender parameters

The TCP transport sender does not accept any configuration parameters as of now.

To enable the TCP transport for samples, simply open the repository/conf/axis2.xml file in a text editor and add the following transport receiver configuration and sender configuration. TCP transport module is shipped with WSO2 ESB by default.
<transportReceiver name="tcp">
  <parameter name="transport.tcp.port">6060</parameter>
</transportReceiver>
<transportSender name="tcp"/>

If you wish to use the sample Axis2 client to send TCP messages, you have to remove the comment of the TCP transport sender configuration in the following file:

dsamples/axis2Client/client_repo/conf/axis2.xml

Local Transport

Apache Axis2's local transport implementation is used to make internal service calls and transfer data within the Axis2 instance. The following class implements the sender API:

   org.apache.axis2.transport.local.LocalTransportSender

The transport does not have a receiver implementation as of now.

It provides an opportunity for fast in-VM service call.

To use this transport, configure an endpoints with the local:// prefix. For example, to make an in-VM call to the HelloService, use local://HelloService.

Configuring a local transport with WSO2 products

Shown below is how to configure a local transport with any WSO2 Carbon-based product.

1. In the carbon.xml file at location <PRODUCT_HOME>/repository/conf, an endpoint is available as follows by default.

   <ServerURL>local://services/&lt;/ServerURL>

2. In the axis2.xml file at location <PRODUCT_HOME>/repository/conf/axis2, there is a transport sender named 'local' specified as follows:

   <transportSender name="local"
   class="org.apache.axis2.transport.local.LocalTransportSender"/>

It has to be replaced with the following sender/receiver pair.

   <transportReceiver name="local"
   class="org.wso2.carbon.core.transports.local.CarbonLocalTransportReceiver"/>
   <transportSender name="local"
   class="org.wso2.carbon.core.transports.local.CarbonLocalTransportSender"/>

For more information about transports, refer to Transport Management.

UDP Transport

The UDP transport implementation is in the Apache WS-Commons Transports project. The following classes implement the Axis2 transport listener and
sender APIs respectively.

- org.apache.axis2.transport.udp.UDPListener
- org.apache.axis2.transport.udp.UDPSender

The axis2-transport-udp.jar archive file contains the above implementation classes.

To enable the UDP transport for samples, simply open the file repository/conf/axis2.xml in a text editor and add the following transport configurations. UDP transport component is shipped with the WSO2 ESB by default.

```xml
<transportReceiver name="udp"/>
<transportSender name="udp"/>
```

If you wish to use the sample Axis2 client to send UDP messages, you have to uncomment the UDP transport sender configuration in the samples/axis2Client/client_repo/conf/axis2.xml file.

**FIX Transport**

**FIX (Financial Information eXchang) transport** implementation is a module developed under the Apache Synapse project. This transport is mainly used with WSO2 ESB in conjunction with proxy services. The following class acts as the transport receiver:

- org.apache.synapse.transport.fix.FIXTransportListener

The org.apache.synapse.transport.fix.FIXTransportSender acts as the transport sender implementation. These classes can be found in the synapse-fix-transport.jar file. The transport implementation is based on Quickfix/J open source FIX engine. Therefore, the following additional dependencies are required to enable the FIX transport.

- mina-core.jar
- quickfixj-core.jar
- quickfixj-msg-fix40.jar
- quickfixj-msg-fix41.jar
- quickfixj-msg-fix42.jar
- quickfixj-msg-fix43.jar
- quickfixj-msg-fix44.jar
- slf4j-api.jar
- slf4j-log4j12.jar

This transport supports JMX. Download Quickfix/J from here: [http://www.quickfixj.com/downloads](http://www.quickfixj.com/downloads). In the distribution archive, you will find all the dependencies listed above. Also, refer to Quickfix/J documentation on configuring FIX acceptors and initiators.

The FIX transport does not support any global parameters. All the FIX configuration parameters should be specified at service level.

**Service level FIX parameters**

In transport parameter tables, literals displayed in italic mode under the “Possible Values” column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

<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>transport.fix.AcceptorConfigURL</td>
<td>URL to the Quickfix/J acceptor configuration file (see notes below).</td>
<td>Required for receiving messages over FIX</td>
<td>A valid URL</td>
<td></td>
</tr>
<tr>
<td>transport.fix.InitiatorConfigURL</td>
<td>URL to the Quickfix/J initiator configuration file (see notes below).</td>
<td>Required for sending messages over FIX</td>
<td>A valid URL</td>
<td></td>
</tr>
<tr>
<td>transport.fix.AcceptorLogFactory</td>
<td>Log factory implementation to be used for the FIX acceptor (Determines how logging is done at the acceptor level).</td>
<td>No</td>
<td>console, file, jdbc</td>
<td>Logging disabled</td>
</tr>
<tr>
<td>transport.fix.InitiatorLogFactory</td>
<td>Log factory implementation to be used for the FIX acceptor (Determines how logging is done at the acceptor level).</td>
<td>No</td>
<td>console, file, jdbc</td>
<td>Logging disabled</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
<td>Values</td>
<td>Notes</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>transport.fix. AcceptorMessage Store</td>
<td>Message store mechanism to be used with the acceptor (Determines how the FIX message store is maintained).</td>
<td>No</td>
<td>memory, file, sleepycat, jdbc</td>
<td>memory</td>
</tr>
<tr>
<td>transport.fix. InitiatorMessage Store</td>
<td>Message store mechanism to be used with the initiator (Determines how the FIX message store is maintained).</td>
<td>No</td>
<td>memory, file, sleepycat, jdbc</td>
<td>memory</td>
</tr>
<tr>
<td>transport.fix. ResponseDeliverTo CompID</td>
<td>If the response FIX messages should be delivered to a location different from the location the request was originated, use this property to set the DeliverToCompID field of the FIX messages.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.fix. ResponseDeliverTo SubID</td>
<td>If the response FIX messages should be delivered to a location different from the location the request was originated, use this property to set the DeliverToSubID field of the FIX messages.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.fix. ResponseDeliverTo LocationID</td>
<td>If the response FIX messages should be delivered to a location different from the location the request was originated use this property to set the DeliverToLocationID field of the FIX messages.</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transport.fix. SendAllToInSequence</td>
<td>By default, all received FIX messages (including responses) will be directed to the in sequence of the proxy service. Use this property to override that behavior.</td>
<td>No</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>transport.fix. BeginStringValidation</td>
<td>Whether the transport should validate BeginString values when forwarding FIX messages across sessions.</td>
<td>No</td>
<td>true, false</td>
<td>true</td>
</tr>
<tr>
<td>transport.fix. DropExtraResponses</td>
<td>In situation where the FIX recipient sends multiple responses per request use this parameter to drop excessive responses and use only the first one.</td>
<td>No</td>
<td>true, false</td>
<td>false</td>
</tr>
</tbody>
</table>
Performance Tuning

This section describes some recommended performance tuning configurations to optimize WSO2 products. It assumes that you have set up the product on a server running Unix/Linux, which is recommended for production deployment.

- OS-Level settings
- JVM settings
- JDBC pool configuration
- Caching configuration
- Setting the thread execution limit
- Pooling LDAPS connections
- Product-specific performance tuning

### Important

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to get familiar with these files using Unix/Linux documentation before editing them.
- The parameter values we discuss below are just examples. They might not be the optimal values for the specific hardware configurations in your environment. We recommend that you carry out load tests on your environment to tune the product accordingly.

### OS-Level settings

When it comes to performance, the OS that the server runs plays an important role.

If you are running MacOS Sierra and experience 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 in this blog post.

Following are the configurations you can apply to optimize OS-level performance:

1. To optimize network and OS performance, configure the following settings in `/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_reuse = 1
   net.ipv4.tcp_tw_recycle = 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 we have the localhost port range configuration lower bound to 1024, there is a possibility that some processes may pick the ports which are already used by WSO2 servers. Therefore, it's good to increase the lower bound as sufficient for production, e.g., 10,000.

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.

   ```
   * 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 server (both hard and soft).
JVM settings

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` attribute that is in the `<PRODUCT_HOME>/bin/wso2server.bat` file (for Windows) or the `<PRODUCT_HOME>/bin/wso2server.sh` file (for Linux/Solaris). The default entity expansion limit is 64000.

```bash
-DentityExpansionLimit=100000
```

In a clustered environment, the entity expansion limit has no dependency on the number of worker nodes.

JDBC pool configuration

Within the WSO2 platform, we use Tomcat JDBC pooling as the default pooling framework due to its production-ready stability and high performance. The goal of tuning the pool properties is to maintain a pool that is large enough to handle peak load without unnecessarily utilizing resources. These pooling configurations can be tuned for your production server in general in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file.

The following parameters should be considered when tuning the connection pool:

- The application's concurrency requirement.
- The average time used for running a database query.
- The maximum number of connections the database server can support.

The table below indicates some recommendations on how to configure the JDBC pool. For more details about recommended JDBC configurations, see Tomcat JDBC Connection Pool.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Tuning Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated from the connection pool at the same time. The default value is 100.</td>
<td>The maximum latency (approximately) = ( P \div M ) * T, where,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- M = maxActive value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- P = Peak concurrency value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- T = Time (average) taken to process a query.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Therefore, by increasing the maxActive value (up to the expected highest number of concurrency), the time that requests wait in the queue for a connection to be released will decrease. But before increasing the Max. Active value, consult the database administrator, as it will create up to maxActive connections from a single node during peak times, and it may not be possible for the DBMS to handle the accumulated count of these active connections. Note that this value should not exceed the maximum number of requests allowed for your database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Formula</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| maxWait    | The maximum time that requests are expected to wait in the queue for a connection to be released. This property comes into effect when the maximum number of active connections allowed in the connection pool (see maxActive property) is used up. | If the maximum latency (approximately) = (P / M) * T , where,  
- M = maxActive value,  
- P = Peak concurrency value,  
- T = Time (average) taken to process a query,  
then, the maxWait = (P / M) * T + buffer time. |
| minIdle    | The minimum number of connections that can remain idle in the pool, without extra ones being created. The connection pool can shrink below this number if validation queries fail. Default value is 0. | This value should be similar or near to the average number of requests that will be received by the server at the same time. With this setting, you can avoid having to open and close new connections every time a request is received by the server. |
| maxIdle    | The maximum number of connections that can remain idle in the pool. | The value should be less than the maxActive value. For high performance, tune maxIdle to match the number of average, concurrent requests to the pool. If this value is set to a large value, the pool will contain unnecessary idle connections. |
| testOnBorrow | The indication of whether connection objects will be validated before they are borrowed from the pool. If the object validation fails, the connection is dropped from the pool, and there will be an attempt to borrow another connection. | When the connection to the database is broken, the connection pool does not know that the connection has been lost. As a result, the connection pool will continue to distribute connections to the application until the application actually tries to use the connection. To resolve this problem, set "Test On Borrow" to "true" and make sure that the "ValidationQuery" property is set. To increase the efficiency of connection validation and to improve performance, validationInterval property should also be used. |
| validationInterval | This parameter controls how frequently a given validation query is executed (time in milliseconds). The default value is 30000 (30 seconds). That is, if a connection is due for validation, but has been validated previously within this interval, it will not be validated again. | Deciding the value for the "validationInterval" depends on the target application's behavior. Therefore, selecting a value for this property is a trade-off and ultimately depends on what is acceptable for the application. |
| validationQuery | 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 an SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server). | Specify an SQL query, which will validate the availability of a connection in the pool. This query is necessary when testOnBorrow property is true. |
| MaxPermSize | The memory size allocated for the WSO2 product. | The default memory allocated for the product via this parameter is as follows:  
- -Xms256m -Xmx512m  
- -XX:MaxPermSize=256m  
You can increase the performance by increasing this value in the <PRODUCT_HOME>/bin/wso2server.sh file as follows:  
- -Xms2048m -Xmx2048m  
- -XX:MaxPermSize=1024m |
Caching configuration

The `<Cache>` element configured in the `carbon.xml` file (stored in the `<PRODUCT_HOME>/repository/conf` directory) sets the global caching timeout in minutes for your server. This value specifies the time period after which, the cache will refresh. If the components in your product do not have specific caching timeout values configured, the global caching timeout will be applicable by default. Be sure to restart the server if you change the default caching timeout in the `carbon.xml` file shown below.

```xml
<Cache>
  <!-- Default cache timeout in minutes -->
  <DefaultCacheTimeout>15</DefaultCacheTimeout>
</Cache>
```

Setting the thread execution limit

The 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.

```xml
<Valve
  className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve"
  threshold="600"/>
```

- The `className` is the Java class name used for the implementation. This must be set to `org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve`.
- The `threshold` gives the minimum duration in seconds after which a thread is considered stuck. Default value is 600 seconds.

This configuration works only for the servlet transport.

Pooling LDAPS connections

Connection pooling does not apply to LDAPS connections (SSL-enabled LDAP connections) by default. Therefore, you need to enable connection pooling for LDAPS connections at the time of starting your server:

1. Stop the server.
2. Open a command terminal, navigate to the `<PRODUCT_HOME>/bin` directory, and execute the relevant startup script:
   - On Linux: `wso2server.sh`
   - On Windows: `wso2server.bat`
3. Add the following system property to the script:
   ```bash
   -Dcom.sun.jndi.ldap.connect.pool.protocol=ssl
   ```
4. Start the server.

Product-specific performance tuning
For performance tuning guidelines that are specific to each product, go to the product documentation for each product listed below and search for performance tuning guidelines.

Listed below are the main WSO2 products:

- API Manager
- Data Analytics Server
- Enterprise Integrator profiles
  - ESB profile
  - Business Process Management profile
  - Message Broker profile
  - Analytics profile
- IOT
- Identity Server
- Enterprise Service Bus

The following are now legacy products of WSO2:

- Application Server
- Business Rules Server
- Enterprise Mobility Manager
- Enterprise Store
- Governance Registry
Monitoring

See the following topics for information on the various monitoring capabilities available for WSO2 servers:

- Capturing System Data in Error Situations
- JMX-Based Monitoring
- Monitoring Logs
- Monitoring Message Flows
- Monitoring Performance Statistics
- Monitoring SOAP Messages
- Monitoring TCP-Based Messages
- Monitoring with WSO2 Carbon Metrics
- Monitoring Server Health

Capturing System Data in Error Situations

Carbon Dump is a tool for collecting all the necessary data (i.e., heap and thread dumps) from a running Carbon instance at the time of an error for a head dump and thread stack analysis. The Carbon Dump generates a ZIP archive with the collected data, which helps the WSO2 support team to analyze your system and determine the problem which caused the error. Therefore, it is recommended that you run this tool as soon as an error occurs in the Carbon instance.

As with any other java product, if your WSO2 product cluster fails due to a resource exhaustion, the heap and thread dumps will always point you towards the cause of the leak. Therefore, it is important to be able to retrieve heap and thread dumps from an environment at the point when an error occurs. This will avoid the necessity of reproducing the exact issue again (specially, in the case of production issues). A resource exhaustion can happen for two reasons:

- Due to a bug in the system.
- An actual limitation of resources based on low configuration values.

You can easily create a heap dump and thread dump using the CarbonDump tool that is shipped with your product. These will also provide information about the product version and any patch inconsistencies.

If you are using an Ubuntu version 10.10 or above and if you get an error on being unable to attach the process, execute the following command to rectify it:

```
$ echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope
```

This changes the yama/ptrace_scope variable of the kernel temporarily (i.e., until the next reboot). For more information, go to Oracle documentation.

When using the tool, you have to provide the process ID (pid) of the Carbon instance and the <PRODUCT_HOME> which is where your unzipped Carbon distribution files reside. The command takes the following format:

```
sh carbondump.sh [-carbonHome path] [-pid of the carbon instance]
```

For example,

```
In Linux: sh carbondump.sh -carbonHome /home/user/wso2carbon-3.0.0/ -pid 5151
```
```
In Windows: carbondump.bat -carbonHome c:\wso2carbon-3.0.0\ -pid 5151
```

The tool captures the following information about the system:

- Operating system information** (OS/kernel) version
- Installed modules lists and their information
- List of running tasks in the system
- Memory information of the Java process** (Java heap memory dump)
- Histogram of the heap
- Objects waiting for finalization
- Java heap summary. GC algo used, etc.
- Statistics on permgen space of Java heap
- Information about the running Carbon instance** (Product name and version)
- Carbon framework version (This includes the patched version)
JMX-Based Monitoring

Java Management Extensions (JMX) is a technology that lets you implement management interfaces for Java applications. A management interface, as defined by JMX, is composed of named objects called MBeans (Management Beans). MBeans are registered with a name (an ObjectName) in an MBeanServer. To manage a resource or many resources in your application, you can write an MBean defining its management interface and register that MBean in your MBeanServer. The content of the MBeanServer can then be exposed through various protocols, implemented by protocol connectors, or protocol adaptors.

- Configuring JMX in a WSO2 product
  - Configuring JMX ports for the server
  - Disabling JMX for the server
  - Enabling JMX for a datasource
- Monitoring a WSO2 product with JConsole
  - Starting the WSO2 product with JMX
  - Using the ServerAdmin MBean
  - Using the ServiceAdmin MBean
  - Using the StatisticsAdmin MBean
  - Using product-specific MBeans
- Monitoring a WSO2 product with Jolokia

Configuring JMX in a WSO2 product

JMX is enabled in WSO2 products by default, which ensures that the JMX server starts automatically when you start a particular product. Additionally, you can enable JMX separately for the various datasources that are used by the product. Once JMX is enabled, you can log in to the JConsole tool and monitor your product as explained in the next section.

Configuring JMX ports for the server

The default JMX ports (RMIRegistryPort and the RMIServerPort) are configured in the carbon.xml file (stored in the <PRODUCT_HOME>/repository/conf directory) as shown below. If required, you can update these default values.

```xml
<JMX>
  <!--The port RMI registry is exposed-->
  <RMIRegistryPort>9999</RMIRegistryPort>
  <!--The port RMI server should be exposed-->
  <RMIServerPort>11111</RMIServerPort>
</JMX>
```

Disabling JMX for the server

The JMX configuration is available in the jmx.xml file (stored in the <PRODUCT_HOME>/repository/conf//etc directory) as shown below. You can disable the JMX server for your product by setting the <StartRMIServer> property to false. Note that this configuration refers to the JMX ports configured in the carbon.xml file.
Enabling JMX for a datasource

You can enable JMX for a datasource by adding the `<jmxEnabled>true</jmxEnabled>` element to the datasource configuration file. For example, to enable JMX for the default Carbon datasource in your product, add the following property to the `master-datasources.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/datasources` directory).

```xml
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:h2:./repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000</url>
      <username>wso2carbon</username>
      <password>wso2carbon</password>
      <driverClassName>org.h2.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
      <jmxEnabled>true</jmxEnabled>
    </configuration>
  </definition>
</datasource>
```

Monitoring a WSO2 product with JConsole

Jconsole is a JMX-compliant monitoring tool, which comes with the Java Development Kit (JDK) 1.5 and newer versions. You can find this tool inside your `<JDK_HOME>/bin` directory. See the instructions on Installing the JDK for more information.

Starting the WSO2 product with JMX

First, start the WSO2 product:

1. Open a command prompt and navigate to the `<PRODUCT_HOME>/bin` directory.
2. Execute the product startup script (`wso2server.sh` for Linux and `wso2server.bat` for Windows) to start the server.

If JMX is enabled, the JMX server URL will be published on the console when the server starts as shown below.
Once the product server is started, you can start the JConsole tool as follows:

1. Open a command prompt and navigate to the `<JDK_HOME>/bin` directory.
2. Execute the `jconsole` command to open the log-in screen of the Java Monitoring & Management Console as shown below.

3. Enter the connection details in the above screen as follows:
   a. Enter the JMX server URL in the Remote Process field. This URL is published on the command prompt when you start the WSO2 server as explained above.
   b. Enter values for the Username and Password fields to log in. If you are logging in as the administrator, you can use the same administrator account that is used to log in to the product's management console: admin/admin.

Tip

If you are connecting with a remote IP address instead of localhost, you need to bind the JMX service to the externally accessible IP address by adding the following system property to the product startup script stored in the `<PRODUCT_HOME>/bin` directory (ws2server.sh for Linux and ws2server.bat for Windows). For more information, read Troubleshooting Connection Problems in JConsole.

```
-Djava.rmi.server.hostname=<IP_ADDRESS_WHICH_YOU_USE_TO_CONNECT_TO_SERVER>
```

Be sure to restart the server after adding the above property.
Make sure that the user ID you are using for JMX monitoring is assigned a role that has the Server Admin permission. See Configuring Roles for further information about configuring roles assigned to users. Any user assigned to the admin role can log in to JMX.

4. Click Connect to open the Java Monitoring & Management Console. The following tabs will be available:

   Overview Memory Threads Classes VM Summary MBeans

See the Oracle documentation on using JConsole for more information on these tabs.
See the Oracle documentation on using JConsole for more information on these tabs.
See the Oracle documentation on using JConsole for more information on these tabs.
Using the ServerAdmin MBean

When you go to the MBeans tab in the JConsole, the ServerAdmin MBean will be listed under the "org.wso2.carbon" domain as shown below.

The ServerAdmin MBean is used for administering the product server instance. There are several server attributes such as "ServerStatus", "ServerData" and "ServerVersion". The "ServerStatus" attribute can take any of the following values:

- RUNNING
- SHUTTING_DOWN
- RESTARTING
- IN_MAINTENANCE
The **ServerAdmin** MBean has the following operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shutdown</td>
<td>Forcefully shut down the server.</td>
</tr>
<tr>
<td>restart</td>
<td>Forcefully restart the server.</td>
</tr>
<tr>
<td>restartGracefully</td>
<td>Wait till all current requests are served and then restart.</td>
</tr>
<tr>
<td>shutdownGracefully</td>
<td>Wait till all current requests are served and then shutdown.</td>
</tr>
<tr>
<td>startMaintenance</td>
<td>Switch the server to maintenance mode. No new requests will be accepted while the server is in maintenance.</td>
</tr>
<tr>
<td>endMaintenance</td>
<td>Switch the server to normal mode if it was switched to maintenance mode earlier.</td>
</tr>
</tbody>
</table>

Using the **ServiceAdmin** MBean
This MBean is used for administering services deployed in your product. Its attributes are as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumberOfActiveServices</td>
<td>The number of services which can currently serve requests.</td>
</tr>
<tr>
<td>NumberOfInactiveServices</td>
<td>The number of services which have been disabled by an administrator.</td>
</tr>
<tr>
<td>NumberOfFaultyServices</td>
<td>The number of services which are faulty.</td>
</tr>
</tbody>
</table>

The operations available in the ServiceAdmin MBean:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startService(p1:string)</td>
<td>The p1 parameter is the service name. You can activate a service using this operation.</td>
</tr>
<tr>
<td>stopService(p1:string)</td>
<td>The p1 parameter is the service name. You can deactivate/disable a service using this operation.</td>
</tr>
</tbody>
</table>
Using the StatisticsAdmin MBean

This MBean is used for monitoring system and server statistics. Its attributes are as follows:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvgSystemResponseTime</td>
<td>The average response time for all the services deployed in the system. The beginning of the measurement is the time at which the server started.</td>
</tr>
<tr>
<td>MaxSystemResponseTime</td>
<td>The maximum response time for all the services deployed in the system. The beginning of the measurement is the time at which the server started.</td>
</tr>
<tr>
<td>MinSystemResponseTime</td>
<td>The minimum time for all the services deployed in the system. The beginning of the measurement is the time at which the server started.</td>
</tr>
<tr>
<td>SystemFaultCount</td>
<td>The total number of faults that occurred in the system since the server was started.</td>
</tr>
<tr>
<td>SystemRequestCount</td>
<td>The total number of requests that has been served by the system since the server was started.</td>
</tr>
<tr>
<td>SystemResponseCount</td>
<td>The total number of response that has been sent by the system since the server was started.</td>
</tr>
</tbody>
</table>
Operations available in the **Statistics** MBean:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getServiceImplCount(p1:string)</code></td>
<td>The p1 parameter is the service name. You can get the total number of requests received by this service since the time it was deployed, using this operation.</td>
</tr>
<tr>
<td><code>getServiceResponseCount(p1:string)</code></td>
<td>The p1 parameter is the service name. You can get the total number of responses sent by this service since the time it was deployed, using this operation.</td>
</tr>
<tr>
<td><code>getServiceFaultCount(p1:string)</code></td>
<td>The p1 parameter is the service name. You can get the total number of fault responses sent by this service since the time it was deployed, using this operation.</td>
</tr>
<tr>
<td><code>getMaxServiceResponseTime(p1:string)</code></td>
<td>The p1 parameter is the service name. You can get the maximum response time of this service since deployment.</td>
</tr>
<tr>
<td><code>getMinServiceResponseTime(p1:string)</code></td>
<td>The p1 parameter is the service name. You can get the minimum response time of this service since deployment.</td>
</tr>
<tr>
<td><code>getAvgServiceResponseTime(p1:string)</code></td>
<td>The p1 parameter is the service name. You can get the average response time of this service since deployment.</td>
</tr>
<tr>
<td><code>getServiceImplCount(p1:string, p2:string)</code></td>
<td>The p1 parameter is the service name. The p2 parameter is the operation name. You can get the total number of requests received by this operation since the time its service was deployed, using this operation.</td>
</tr>
<tr>
<td><code>getServiceImplCount(p1:string, p2:string)</code></td>
<td>The p1 parameter is the service name. The p2 parameter is the operation name. You can get the total number of responses sent by this operation since the time its service was deployed, using this operation.</td>
</tr>
<tr>
<td><code>getServiceFaultCount(p1:string, p2:string)</code></td>
<td>The p1 parameter is the service name. The p2 parameter is the operation name. You can get the total number of fault responses sent by this operation since the time its service was deployed, using this operation.</td>
</tr>
<tr>
<td><code>getMaxOperationResponseTime(p1:string, p2:string)</code></td>
<td>The p1 parameter is the service name. The p2 parameter is the operation name. You can get the maximum response time of this operation since deployment.</td>
</tr>
<tr>
<td><code>getMinOperationResponseTime(p1:string, p2:string)</code></td>
<td>The p1 parameter is the service name. The p2 parameter is the operation name. You can get the minimum response time of this operation since deployment.</td>
</tr>
<tr>
<td><code>getAvgOperationResponseTime(p1:string, p2:string)</code></td>
<td>The p1 parameter is the service name. The p2 parameter is the operation name. You can get the average response time of this operation since deployment.</td>
</tr>
</tbody>
</table>
Using the DataSource MBean

If you have JMX enabled for a datasource connected to the product, you can monitor the performance of the datasource using this MBean. The **DataSource** MBean will be listed as shown below.

**Example:** If you have JMX enabled for the default Carbon datasource in the `master-datasources.xml` file, the JDBC connection pool parameters that are configured for the Carbon datasource will be listed as attributes as shown below. See the performance tuning guide for instructions on how these parameters are configured for a datasource.
Using product-specific MBeans

The WSO2 product that you are using may have product-specific MBeans enabled for monitoring and managing specific functions. See the documentation for your product for detailed instructions on such product-specific MBeans.

Monitoring a WSO2 product with Jolokia

Jolokia is a JMX-HTTP bridge, which is an alternative to JSR-160 connectors. It is an agent-based approach that supports many platforms. In addition to basic JMX operations, it enhances JMX monitoring with unique features like bulk requests and fine-grained security policies.

Follow the steps below to use Jolokia to monitor a WSO2 product.

1. Download Jolokia OSGi Agent. (These instructions are tested with the Jolokia OSGi Agent version 1.3.6 by downloading the jolokia-osgi-1.3.6.jar file.)
2. Add it to the <PRODUCT-HOME>/repository/components/dropins/ directory.
   
   In WSO2 EI, add it to the <EI-HOME>/dropins/ directory.
3. Start the WSO2 product server.

Once the server starts, you can read MBeans using Jolokia APIs. Following are a few examples.

- List all available MBeans: http://localhost:9763/jolokia/list (Change the appropriate hostname and port accordingly.)
- WSO2 ESB MBean: http://localhost:9763/jolokia/read/org.apache.synapse:Name=https-sender,Type=PassThroughConnections/ActiveConnections
- Reading Heap Memory: http://localhost:9763/jolokia/read/java.lang:type=Memory/HeapMemoryUsage

For more information on the JMX MBeans that are available in WSO2 products, see Monitoring a WSO2 product with JConsole.

Monitoring Logs

Logging is one of the most important aspects of a production-grade server. A properly configured logging system is vital for identifying errors, security threats, and usage patterns.

See the following topics for details:

- Log types in WSO2 products
- Configuring products for log monitoring
  - Setting the Log4j log level
Managing log growth
- Managing the growth of Carbon logs
- Limiting the size of Carbon log files
- Limiting the size of audit log files
- Monitoring logs

Log types in WSO2 products

Listed below are the various log types that are used in WSO2 products.

Separate log files are created for each of the log types given below in the `<PRODUCT_HOME>/repository/logs` directory.

- **Carbon logs**: All WSO2 products are shipped with log4j logging capabilities that generate administrative activities and server side logs. The Carbon log (`wso2carbon.log`) is a log file that covers all the management features of a product. Carbon logs are configured in the `log4j.properties` file (stored in the `<PRODUCT_HOME>/repository/conf` directory).

Java logging and Log4j integration: In addition to the logs from libraries that use Log4j, all logs from libraries (such as, Tomcat, Hazelcast and more) that use Java logging framework are also visible in the same log files. That is, when Java logging is enabled in Carbon, only the Log4j appenders will write to the log files. If the Java Logging Handlers have logs, these logs will be delegated to the log events of the corresponding Log4j appenders. A Pub/Sub registry pattern implementation has been used in the latter mentioned scenario to plug the handlers and appenders. The following default Log4j appenders in the `log4j.properties` file are used for this implementation:

- `org.wso2.carbon.logging.appenders.CarbonConsoleAppender`
- `org.wso2.carbon.logging.appenders.CarbonDailyRollingFileAppender`

- **Audit logs**: Audit logs are used for tracking the sequence of actions that affect a particular task carried out on the server. These are also configured in the `log4j.properties` file.
- **HTTP access logs**: HTTP requests/responses are logged in access logs to monitor the activities related to an application’s usage. These logs are configured in the `catalina-server.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat` directory).
- **Patch logs**: These logs contain details related to patches applied to the product. Patch logs cannot be customized. See WSO2 Patch Application Process for more information.
- **Service/Event logs**: These are logs that are enabled in some WSO2 products for tracing services and events using a separate log file (`wso2-*trace.log`). If server/event tracing logs are used in your WSO2 product, you can configure them in the `log4j.properties` file.
- **Product-specific logs**: Each WSO2 product may generate other log files in addition to the Carbon logs, Audit logs, HTTP access logs, Patch logs and Service/Event logs. See the product's documentation for descriptions of these log files and instructions on how to configure and use them.

Configuring products for log monitoring

See the following information on configuring Carbon logs, Audit logs, HTTP access logs, and Service/Event logs for your WSO2 product.

- **Configuring Carbon logs**

  You can easily configure Carbon logs using the management console of your product, or you can manually edit the `log4j.properties` file. It is recommended to use the management console to configure logging because all changes made to log4j through the management console persists in the WSO2 Registry. Therefore, those changes will be available after the server restarts and will get priority over what is defined in the log4j.properties file. Also, note that the logging configuration you define using the management console will apply at run time. However, if you modify the `log4j.properties` file and restart the server, the earlier log4j configuration that persisted in the registry will be overwritten. There is also an option in the management console to restore the original log4j configuration from the `log4j.properties` file. The log levels that can be configured are listed below.

  **Identifying forged messages:**

  The log pattern defines the output format of the log file. From Carbon 4.4.3 onwards, the conversion character 'K' can be used in the pattern layout to log a UUID. For example, the log pattern can be `%K [%T] [%s] [%d] %p%5p (%c) - %x %m (%c)%n, where [%K] is the UUID.

  The UUID can be used for identifying forged messages in the log. By default, the UUID will be generated every time the server starts. If required, you can configure the UUID regeneration period by manually adding the following property to the `log4j.properties` file (stored in the `<PRODUCT_HOME>/repository/conf` directory):

```properties
log4j.appender.CARBON_LOGFILE.layout.LogUUIDUpdateInterval=<number_of_hours>
```
Carbon logs in WSO2 Data Analytics Server (WSO2 DAS)

Carbon logs are configured in the `log4j.properties` file (stored in the `<PRODUCT_HOME>/repository/conf` directory) for all WSO2 products. However, WSO2 DAS generates some additional Carbon logs (which will be stored in the same Carbon log file) that should be separately configured by creating a new `log4j.properties` file in the `<DAS_HOME>/repository/conf/analytics/spark` directory. Note: To create this file, you need to rename the `log4j.properties.template` file that is available in the `<DAS_HOME>/repository/conf/analytics/spark` directory to `log4j.properties`.

See the following topics for instructions:

- Configuring Log4j Properties
- Configuring the Log Provider

* Configuring Audit logs

Audit logs are enabled in WSO2 products by default. You can change the following default configuration by manually updating the `log4j.properties` file. The log levels that can be configured are listed below.

```
log4j.logger.AUDIT_LOG=INFO, AUDIT_LOGFILE
# Appender config to AUDIT_LOGFILE
log4j.appender.AUDIT_LOGFILE=org.wso2.carbon.utils.logging.appenders.CarbonDailyRollingFileAppender
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 {%c}-%x %m %n
log4j.appender.AUDIT_LOGFILE.layout.TenantPattern=%U%@%D [%T] [%S]
log4j.appender.AUDIT_LOGFILE.threshold=INFO
log4j.additivity.AUDIT_LOG=false
```

* Configuring HTTP access logs

See [HTTP Access Logging](#) for instructions on how to configure and use HTTP access logs.

* Configuring Service/Event tracing logs

A separate log file for tracing services/events are enabled for certain WSO2 products in the `log4j.properties` file using a specific appender. These logs are published to a file named `wso2-<product>-trace.log`. See the table given below for instructions relevant to your product:

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carbon logs in (WSO2 DAS)</td>
<td>WSO2 Data Analytics Server</td>
</tr>
<tr>
<td>Carbon logs are configured in the <code>log4j.properties</code> file (stored in the <code>&lt;PRODUCT_HOME&gt;/repository/conf</code> directory) for all WSO2 products. However, WSO2 DAS generates some additional Carbon logs (which will be stored in the same Carbon log file) that should be separately configured by creating a new <code>log4j.properties</code> file in the <code>&lt;DAS_HOME&gt;/repository/conf/analytics/spark</code> directory. Note: To create this file, you need to rename the <code>log4j.properties.template</code> file that is available in the <code>&lt;DAS_HOME&gt;/repository/conf/analytics/spark</code> directory to <code>log4j.properties</code>.</td>
<td></td>
</tr>
<tr>
<td>Audit logs are enabled in WSO2 products by default. You can change the following default configuration by manually updating the <code>log4j.properties</code> file. The log levels that can be configured are listed below.</td>
<td></td>
</tr>
<tr>
<td>Configuring Log4j Properties</td>
<td>Configuring the Log Provider</td>
</tr>
<tr>
<td>Configuring Audit logs</td>
<td>Configuring HTTP access logs</td>
</tr>
<tr>
<td>Configuring Service/Event tracing logs</td>
<td>See <a href="#">HTTP Access Logging</a> for instructions on how to configure and use HTTP access logs.</td>
</tr>
</tbody>
</table>

---
WSO2 DAS Event tracing logs are enabled in WSO2 DAS using the `EVENT_TRACE_LOGGER` appender as shown below (Click Message tracing log configuration). This log file stores logs related to events in WSO2 DAS. By default, this appender uses the root log level, which is INFO. You can override the root log level by giving a specific log level for the appender as explained here.

```java
log4j.category.EVENT_TRACE_LOGGER=INFO, EVENT_TRACE_APPENDER,
EVENT_TRACE_MEMORYAPPENDER
log4j.additivity.EVENT_TRACE_LOGGER=false
log4j.appender.EVENT_TRACE_APPENDER=org.apache.log4j.DailyRollingFileAppender
log4j.appender.EVENT_TRACE_APPENDER.File=${carbon.home}/repository/logs/
$instance.log}/wso2-das-trace${instance.log}.log
log4j.appender.EVENT_TRACE_APPENDER.Append=true
log4j.appender.EVENT_TRACE_APPENDER.layout=org.apache.log4j.PatternLayout
log4j.appender.EVENT_TRACE_APPENDER.layout.ConversionPattern=%d{HH:mm:ss,SSS} [%X{ip}-%X{host}] [%t] %5p %c{1} %m%n
# The memory appender for trace logger
log4j.appender.EVENT_TRACE_MEMORYAPPENDER=org.wso2.carbon.utils.logging.appenders.MemoryAppender
log4j.appender.EVENT_TRACE_MEMORYAPPENDER.bufferSize=2000
log4j.appender.EVENT_TRACE_MEMORYAPPENDER.layout=org.apache.log4j.PatternLayout
log4j.appender.EVENT_TRACE_MEMORYAPPENDER.layout.ConversionPattern=%d
%H:mm:ss,SSS] [%X{ip}-%X{host}] [%t] %5p %m%n
```

### Setting the Log4j log level

The log level can be set specifically for each appender in the `log4j.properties` file by setting the threshold value. If a log level is not specifically given for an appender as explained below, the root log level (INFO) will apply to all appenders by default.

For example, shown below is how the log level is set to DEBUG for the `CARBON_LOGFILE` appender (Carbon log):

```java
log4j.appender.CARBON_LOGFILE.threshold=DEBUG
```

Listed below are the log levels that can be configured:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>The highest possible log level. This is intended for disabling logging.</td>
</tr>
<tr>
<td>FATAL</td>
<td>Indicates server errors that cause premature termination. These logs are expected to be immediately visible on the command line that you used for starting the server.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Indicates other runtime errors or unexpected conditions. These logs are expected to be immediately visible on the command line that you used for starting the server.</td>
</tr>
<tr>
<td>WARN</td>
<td>Indicates the use of deprecated APIs, poor use of API, possible errors, and other runtime situations that are undesirable or unexpected but not necessarily wrong. These logs are expected to be immediately visible on the command line that you used for starting the server.</td>
</tr>
<tr>
<td>INFO</td>
<td>Indicates important runtime events, such as server startup/shutdown. These logs are expected to be immediately visible on the command line that you used for starting the server. It is recommended to keep these logs to a minimum.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Provides detailed information on the flow through the system. This information is expected to be written to logs only. Generally, most lines logged by your application should be written as DEBUG logs.</td>
</tr>
</tbody>
</table>
Managing log growth

See the following content on managing the growth of Carbon logs and Audit logs:

Managing the growth of Carbon logs

Log growth (in Carbon logs) can be managed by the following configurations in the `<PRODUCT_HOME>/repository/conf/log4j.properties` file.

- Configurable log rotation: By default, log rotation is on a daily basis.
- Log rotation based on time as opposed to size: This helps to inspect the events that occurred during a specific time.
- Log files are archived to maximise the use of space.

The `log4j`-based logging mechanism uses appenders to append all the log messages into a file. That is, at the end of the log rotation period, a new file will be created with the appended logs and archived. The name of the archived log file will always contain the date on which the file is archived.

Limiting the size of Carbon log files

You can limit the size of the `<PRODUCT_HOME>/repository/logs/wso2carbon.log` file by following the steps given below. This is useful if you want to archive the logs and get backups periodically.

1. Change the `log4j.appender.CARBON_LOGFILE=org.wso2.carbon.utils.logging.appenders.CarbonDailyRollingFileAppender` appender in the `<PRODUCT_HOME>/repository/conf/log4j.properties` file as follows:

   ```
   log4j.appender.CARBON_LOGFILE=org.apache.log4j.RollingFileAppender
   ```

2. Add the following two properties under `RollingFileAppender`:
   - `log4j.appender.CARBON_LOGFILE.MaxFileSize=10MB`
   - `log4j.appender.CARBON_LOGFILE.MaxBackupIndex=20`

   If the size of the log file is exceeding the value defined in the `MaxFileSize` property, the content is copied to a backup file and the logs are continued to be added to a new empty log file. The `MaxBackupIndex` property makes the Log4j maintain a maximum number of backup files for the logs.

Limiting the size of audit log files

In WSO2 servers, audit logs are enabled by default. We can limit the audit log files with the following configuration:

1. Change the `log4j.appender.AUDIT_LOGFILE=org.wso2.carbon.logging.appenders.CarbonDailyRollingFileAppender` appender in the `<PRODUCT_HOME>/repository/conf/log4j.properties` file as follows:

   ```
   log4j.appender.AUDIT_LOGFILE=org.apache.log4j.RollingFileAppender
   ```

2. Add the following two properties under `RollingFileAppender`:
   - `log4j.appender.AUDIT_LOGFILE.MaxFileSize=10MB`
   - `log4j.appender.AUDIT_LOGFILE.MaxBackupIndex=20`

Monitoring logs

In each WSO2 product, users can configure and adjust the logging levels for each type of activity/transaction. There are several ways to view and monitor the logs:

- Carbon logs (system logs and application logs) of a running Carbon instance can be monitoring using the management console.
- Carbon logs, as well as HTTP access logs will be printed on the command terminal that open when you execute the product startup script.
- Alternatively, all log files can be viewed from the `<PRODUCT_HOME>/repository/logs` folder. This folder contains Audit logs, HTTP access logs as well as the Carbon logs in separate log files with time stamps. Note that older Carbon logs are archived in the `wso2carbon.log` file.

Monitoring Logs using Management Console

Monitoring logs using the management console of your product is possible with the Logging Management feature. If this feature is not bundled in your Carbon product by default, you can install it following the instructions in the Working with Features section.

Feature Name: WSO2 Carbon - Logging Management Feature
Feature Identifier: org.wso2.carbon.logging.mgt.feature.group

This feature enables the log viewer on the management console. You can click **System Logs** or **Application Logs** on the **Monitor** tab to access the log viewer as shown below.

![Log Viewer Screenshot](image_url)

To use this feature in your Carbon server, see the following topics:
- Configuring Log4j Properties
- Configuring the Log Provider
- View and Download Logs

**Configuring Log4j Properties**

All WSO2 products are shipped with the **log4j** logging capabilities, which generates administrative activities and server side logs. The `log4j.properties` file, which governs how logging is performed by the server can be found in the `<PRODUCT_HOME>/repository/conf` directory. If the Logging Management feature is installed, log4j properties can be configured using the management console.

There are three main components when configuring log4j. They are Loggers, Appenders, and Layouts. Using the management console allows you to change these parameters globally as well as individually at run time. First, the server stores new values in the database and then changes the appropriate components in the logging framework, enabling the logging properties to be updated immediately.

In most systems, logging properties should be specified before the server starts and cannot be changed while it is running. However, as shown here, the logging properties of a running Carbon instance can be changed through its management console, while the server is up and running.

To configure logging properties of the system and application logs of a Carbon server at run time:

1. Log in to the management console of your product and go to **Configure** -> **Logging** in the navigator. The **Logging Configuration** window appears as shown below.
2. If you select the **Persist All Configuration Changes** check box, all the modifications will persist and they will be available even after the server restarts.

3. The **Logging Configuration** window consists of three sections, which you can use to configure the layout and the amount of information you want to receive about system activity. The following topics describes each of these settings:

   - **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** - Severity of the message. Reflects a minimum level that the logger requires. See descriptions of the available log levels.
- **Log Pattern** - Defines the output format of the log file. This is the layout pattern which describes the log message format.

If you click **Restore Defaults**, the Registry will be overwritten by logging configurations specified in the `log4j.properties` file.

### Configure Log4J Appenders
This section allows you to configure appenders individually. 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, a WSO2 product server is entered in this field with the following log appenders configured:
  - **CARBON_CONSOLE** - Logs to the console when the server is running.
  - **CARBON_LOGFILE** - Writes the logs to `<PRODUCT_HOME>/repository/logs/wso2carbon.log`.

Some WSO2 products do not ship the following appenders by default.

- **SERVICE_APPENDER** - Writes service invocations to `<PRODUCT_HOME>/repository/logs/wso2-<PRODUCT_NAME>-service.log`.
- **ERROR_LOGFILE** - Writes warning/error messages to `<ESB_HOME>/repository/logs/wso2-<PRODUCT_NAME>-service.log`.

- **TRACE_APPENDER** - Writes tracing/debug messages to the `<PRODUCT_HOME>/repository/logs/wso2-<PRODUCT_NAME>-trace.log` for tracing enabled services.
- **CARBON_MEMORY**
- **CARBON_SYS_LOG** - Allows separating the software that generates messages, from the system that stores them and the software that reports and analyzes them.
- **CARBON_TRACE_LOGFILE**

- **Log pattern** - Defines the output format of the log file. From Carbon 4.4.3 onwards, the conversion character 'K' can be used in the pattern layout to log a UUID. For example, the log pattern can be `[%K] [%T] [%S] [%d] %P%5p {%c} - %x %m {%c}%n`, where `%K` is the UUID.

Note that the following capability was introduced by the Carbon 4.4.3 release. Therefore, it is only applicable to products that a based on Carbon 4.4.3 or later versions.

From Carbon 4.4.3 onwards, the UUID can be used for identifying forged messages in the log. By default, the UUID will be generated every time the server starts. If required, you can configure the UUID regeneration period by manually adding the following property to the `log4j.properties` file (stored in the `<PRODUCT_HOME>/repository/conf` directory):

```
log4j.appender.CARBON_LOGFILE.layout.LogUUIDUpdateInterval=<number_of_hours>
```

- **Sys Log Host** - The IP address of the system log server. The syslog server is a dedicated log server for many applications. It runs in a particular TCP port in a separate machine, which can be identified by an IP address.
- **Facility** - The log message type sent to the system log server.
- **Threshold** - Filters log entries based on their level. For example, threshold set to ‘WARN’ will allow the log entry to pass into appender. If its level is ‘WARN’, ‘ERROR’ or ‘FATAL’, other entries will be discarded. This is the minimum log level at which you can log a message. See descriptions of the available log levels.

Configure Log4j Loggers
A Logger is an object 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'.

When the server starts for the first time, all the loggers initially listed in the log4j.properties file appear on the logger name list. This section allows you to browse through all these loggers, define a log level and switch on/off additivity to any of them. After editing, the logging properties are read only from the database.

- **Logger** - The name of a logger.
- **Parent Logger** - The name of a parent logger.
- **Level** - Allows to select level (threshold) from the drop-down menu. After you specify the level for a certain logger, a log request for that logger will only be enabled if its level is equal or higher to the logger’s level. If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. Refer to the hierarchy of levels given above. See descriptions of the available log levels.
- **Additivity** - Allows to inherit all the appenders of the parent Logger if set as ‘True’.

In this section, loggers can be filtered by the first characters (use the Starts With button) or by a combination of characters (use the Contains button).

Configuring the Log Provider

Logs of a system can be stored in many ways. For example, they can be stored in a file system, an SQL server such as MySQL, a no-SQL server like Cassandra, etc. According to the default configurations in a Carbon product, the logs are stored in the directory `<PRODUCT_HOME>/repository/logs/` as .log files.

To view and download the logs using the management console, the following configurations are required: the Logging Management feature should be installed, the log4j properties should be configured and the LogProvider and LogFileProvider interfaces should be implemented and configured for the server as described below.

- **Implementing the LogProvider interface**
- **Implementing the LogFileProvider interface**
- **Configuring Carbon to plug the log provider**

Implementing the LogProvider interface

This org.wso2.carbon.logging.service.provider.api.LogProvider interface is used for viewing logs in the management console. It is introduced as an extension point to provide logs to the "Log Viewer" (in the management console). Any log provider can implement this interface to fetch logs from any mechanism, and the Log Viewer will use this interface to retrieve and show logs in the management console.

The LogProvider interface has the following methods:

- **init (LoggingConfig loggingConfig)** - Initialize the log provider by reading the properties defined in the logging configuration file. This will be called immediately after creating a new instance of LogProvider.
- **getApplicationNames(String tenantDomain, String serverKey)** - Return list of all application names deployed under provided tenant domain and server key.
- **getSystemLogs()** - Return a list of system LogEvents.
- **getAllLogs(String tenantDomain, String serverKey)** - Return list of all the logs available under given domain and server key.
- **getLogsByAppName(String appName, String tenantDomain, String serverKey)** - Return list of all the LogEvents belonging to the application, which is deployed under given tenant domain and server key.
- **getLogs(String type, String keyword, String appName, String tenantDomain, String serverKey)** - Returns list of all LogEvents related to the given application, which match to given type and LogEvent message has given key word with it. User can use this api for search operations.
Implementing the LogFileProvider interface

The org.wso2.carbon.logging.service.provider.api.LogFileProvider interface is used to list and download the archived log files using the management console. It is introduced as an extension point providing the list of log file names and the ability to download these logs to the "Log Viewer".

The LogFileProvider interface has the following methods:

- init(LoggingConfig loggingConfig): Initialize the file log provider by reading the properties defined in the logging configuration file. This will be called immediately after creating a new instance of LogFileProvider.
- getLogFileInfoList(String tenantDomain, String serviceName): Return information about the log files, which is available under given tenant domain and serviceName. For example, info about logs: log name, log date, log size.
- downloadLogFile(String logFile, String tenantDomain, String serviceName): Download the file.

Default log provider in Carbon products

A default "in memory" log provider, which implements the LogProvider interface has been created both as a sample and as the default log provider option in carbon. Main task of this class is to read the carbon logs available in the <PRODUCT_HOME>/repository/logs/ directory to a buffer stored in memory and enable the LogViewer to fetch and view these logs in the management console.

A default log file provider that implements the LogFileProvider interface has also been implemented as a sample and as the default log file provider option in carbon. The main task of this class is to read the log file names (including the size and date of these files) from the <PRODUCT_HOME>/repository/logs/ directory and to enable the download of these logs.

Configuring Carbon to plug the log provider

After implementing the above interfaces, update the logging-config.xml file stored in the <PRODUCT_HOME>/repository/conf/etc/ directory.

- Shown below is the configuration for the the default log provider and the default log file provider of a Carbon product:

```
<loggingConfig xmlns="http://wso2.org/projects/carbon/carbon.xml">
  <!-- Default log provider -->
  <logProviderConfig
   class="org.wso2.carbon.logging.service.provider.InMemoryLogProvider">
    <properties/>
  </logProviderConfig>

  <!-- Default log file provider -->
  <logFileProviderConfig
   class="org.wso2.carbon.logging.service.provider.FileLogProvider">
    <properties/>
  </logFileProviderConfig>
</loggingConfig>
```

The default "InMemoryLogProvider" uses the CarbonMemoryAppender. Therefore the log4j.properties file stored in <PRODUCT_HOME>/repository/conf/ directory should be updated with the following log4j.appender.CARBON_MEMORY property:

```
log4j.appender.CARBON_MEMORY=org.wso2.carbon.logging.service.appender.CarbonMemoryAppender]
```

If the implemented class requires additional properties to initialise the class, the <properties> element in the logging-config.xml file can be used. For example, a cassandra based log provider may need information on keyspace, column family, etc. You can configure these details in the logging-config.xml file and access them at runtime using the LoggingConfig class, which contains all configuration parameters. For a Cassandra based log provider, the following properties can be defined in the logging-config.xml file and later used in the implementation using the LoggingConfig class, which is assigned when initializing the class.
• The following properties can be configured in the `logging-config.xml` file for a Cassandra based log provider:

```xml
  <properties>
    <property name="userName" value="admin"/>
    <property name="password" value="admin"/>
    <property name="archivedHost" value="http://127.0.0.1/logs/stratos/0/WSO2%20Stratos%20Manager"/>
    <property name="archivedHDFSPath" value="/stratos/logs"/>
    <property name="archivedPort" value="80"/>
    <property name="archivedUser" value="admin"/>
    <property name="archivedPassword" value="admin"/>
    <property name="archivedRealm" value="Stratos"/>
    <property name="cassandraHost" value="localhost:9160"/>
    <property name="isDataFromCassandra" value="false"/>
    <property name="cassandraConsistencyLevel" value="ONE"/>
    <property name="cassandraAutoDiscovery.enable" value="false"/>
    <property name="cassandraAutoDiscovery.delay" value="1000"/>
    <property name="retryDownedHosts.enable" value="true"/>
    <property name="retryDownedHosts.queueSize" value="10"/>
    <property name="columnFamily" value="log"/>
    <property name="cluster" value="admin"/>
    <property name="keyspace" value="EVENT_KS"/>
  </properties>
</logProviderConfig>
```

### View and Download Logs

It is possible to monitor system logs and application logs in your server using the management console, if the Logging Management feature is installed. Further, when you monitor system logs, you can also download the archived log files using the management console. See Configuring Log4j Properties and Configuring Log Providers for details on how the server can be configured for log monitoring.

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 syslog-ng server.
- If syslog-ng is not configured, log files are taken from the local file system (super-tenant or Stand-alone apps).

The location of the log files on disk is specified in the `log4j.configuration` file.

The log messages displayed on this page are obtained from a memory appender. Therefore, the severity (log level) of the displayed log messages are equal to or higher than the threshold of the memory appender. For more information on appenders, loggers, their log levels and logging, go to [http://logging.apache.org/log4j](http://logging.apache.org/log4j).

See the following topics on system logs and application logs:
- View and download system logs
- View application logs

### View and download system logs

The **System Logs** page on the management console displays all events of a running Carbon instance. Further, it facilitates downloading and viewing log files according to user preferences. Follow the instructions below to access statistics on system logs:

1. Log on to the product's management console and select **Monitor -> System Logs**. The **System Logs** page appears and displays logs in a bottom-up manner.
2. To view old archived logs, click **Show archived logs** tab at the bottom of the **System Logs** page.
The **Download** link can be used to download the log files. For example, if the server is configured to use the default log provider, the "wso2carbon.log" file stored in the `<PRODUCT_HOME>/repository/logs/` directory can be downloaded.

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.

### System Logs

4. You can also find a specific log using the search function. Enter a keyword (or part of a keyword) and click **Search**.

---

**View application logs**

Events invoked by an application or a program running in a system are recorded as application logs. Similarly, the application logs of a running Carbon instance displays 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 you to view the logs relevant to a particular application.

Follow the instructions given below to access statistics on application logs:

1. Log on to the product’s management console and select **Monitor -> Application Logs**.
2. The "Application Logs" page appears. This page displays logs of a selected application in a bottom-up manner.
3. You can see a drop-down list from which a deployed web services or a web applications can be selected to view its log files.

---

**Application Logs**

4. 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,
HTTP Access Logging

HTTP access logs help you monitor your application's usage with information such as the persons who access it, how many hits it 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.

Using WSO2 ESB, WSO2 EI, or WSO2 APIM?

In products such as WSO2 Enterprise Service Bus (WSO2 ESB), WSO2 Enterprise Integrator (WSO2 EI), and WSO2 API Manager (WSO2 APIM), access logs can be generated for the passthrough transport in addition to the HTTP servlet transport. The passthrough 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 <PRODUCT_HOME>/repository/logs/ directory.

See the documentation for these specific products for instructions on how to use access logs.

Note that access logs for the HTTP servlet transport logs details of the request as well as the response. However, the access logs for the passthrough transport only logs the request details.

See the topics given below to configure the default behaviour of HTTP access logs in WSO2 products.

- Configuring access logs for the HTTP servlet transport
- Customizing access logs by pattern
  - Example 1: Logging request headers
  - Example 2: Logging response headers
  - Example 3: Logging other variable values
  - Example 4: Logging URL encoded parameters

Configuring access logs for the HTTP servlet transport

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 to the HTTP servlet transport:

1. Open the <PRODUCT_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.
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.</td>
</tr>
<tr>
<td>suffix</td>
<td>The suffix added to the log file's name. By default, this is .log for all WSO2 products.</td>
</tr>
</tbody>
</table>
| 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. 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 represents 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"
  ```

3. Restart the server. According to the default configurations, a log file named `localhost_access_log_sample.[DATE].log` is created inside the `<PRODUCT_HOME>/repository/logs` directory. The log is rotated on a daily basis.

**Customizing access logs by pattern**

Given below are a few sample configurations for customizing the `pattern` attribute:

- **Example 1: Logging request headers**
- **Example 2: Logging response headers**
- **Example 3: Logging other variable values**
- **Example 4: Logging URL encoded parameters**

**Example 1: Logging request headers**

The configuration is as follows:
This sample configuration logs the Content-type, Accept and Accept-encoding headers of every request coming to the server. For example, in the following example, we use the RequestInfoExample to send the HTTP request:

```
GET
http://<IP>:<PORT>/example/servlets/servlet/RequestInfoExample?abc=xyz
```

The following log entry is recorded in the localhost_access_log_sample.(DATE).log file.

```
text/plain; charset=utf-8 */* gzip, deflate, sdch
```

Example 2: Logging response headers

The configuration is as follows:

```
<Valve className="org.apache.catalina.valves.AccessLogValve"
directory="${carbon.home}/repository/logs"
prefix="localhost_access_log_test."
suffix=".log"
pattern="%{Content-Type}o %{Content-Length}o %{Date}o %{Server}o"
/>
```

The above configuration sample logs the Content-type, Content-Length, Date, and Server headers of every response coming from the server as follows:

```
text/html;charset=ISO-8859-1 662 Tue, 09 Jul 2013 11:21:50 GMT WSO2 Carbon
```
The above sample configuration logs the first line of the request (method and request URI), query string (prepended with a "?" if it exists), and a remote hostname (or IP) of every request coming to the server as follows:

```
"GET /example/servlets/servlet/RequestInfoExample?abc=xyz HTTP/1.1" ?abc=xyz 10.100.0.67
```

Example 4: Logging URL encoded parameters

You cannot use the AccessLogValve to log URL encoded parameters. However, you can use the ExtendedAccessLogValve attribute for this purpose. In this example only two values (namely, className, and pattern) are modified from the previous configuration.

The configuration is as follows:

```
<Valve className="org.apache.catalina.valves.ExtendedAccessLogValve"
directory="${carbon.home}/repository/logs"
prefix="localhost_access_log_extended."
suffix=".log"
pattern="x-P(param1) x-P(param2)"
/>
```

Send the POST request together with the URL encoded values such as `param1=value1` and `param2=value2` as follows:

```
POST http://<IP>:<PORT>/example/servlets/servlet/RequestInfoExample
```

The above sample configuration logs the following:

```
'value1'   'value2'
```

**Masking Sensitive Information in Logs**

There can be business sensitive information that are added to logs in the WSO2 product console and/or WSO2 Carbon log files. When these logs are analyzed, the information is exposed to those who check this.

To avoid this potential security pitfall, users can mask sensitive information from the log file at the time of logging. In this feature, you can define patterns that need to be masked from the logs. This is particularly useful in the case of credit card numbers, access tokens, etc.

To configure this feature, follow the instructions below.

**Enabling log masking**
1. Open the `<PRODUCT_HOME>/repository/conf/log4j.properties` file in a text editor.
2. Uncomment or add the following property under `CarbonConsoleAppender` or `CarbonDailyRollingFileAppender`.

```java
log4j.appender.CARBON_CONSOLE.maskingPatternFile=path-to-masking-patterns
```

The `path-to-masking-patterns` value must be an absolute path to the masking patterns file. In this file, each pattern is defined as key, value pairs (pattern-name=pattern). Please refer to the next section for information on this file.

The following is a sample configuration for the above property.

```java
log4j.appender.CARBON_CONSOLE.maskingPatternFile=/home/conf/masking-patterns.properties
```

For the `DailyRollingFileAppender` value, the above property would be similar to the following.

```java
log4j.appender.CARBON_LOGFILE.maskingPatternFile=path-to-masking-patterns
```

In this case, you can define separate masking pattern files for console appender and file appender. (or configure only one property.)

**The masking pattern file**

The masking pattern file is a property file that can contain one or more masking patterns. The following is a sample configuration that showcases how to mask the credit card numbers from the logs.

```java
masking.pattern.sample.CREDIT_CARD_VISA=4[0-9]{6,}$
masking.pattern.sample.CREDIT_CARD_MASTER=(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]{12}|[23-6][0-9]{12}|2?7[01][0-9]|2720)[0-9]{12}
masking.pattern.sample.CREDIT_CARD_AMEX=[34|37][0-9]{14}$
```

With this configuration, each log line is checked for all the configured patterns. If any match is found, it is masked with ‘*****’.

**Important**: There can be a performance impact when using this feature with many masking patterns since each log line is matched with each of the patterns. So it is highly advisable to only use the most necessary patterns.

**Monitoring Message Flows**

Message Flows provide graphical or textual views of the globally-engaged handlers in the system at a given time. This functionality is provided by the following feature:

**Name**: WSO2 Carbon - Message Flows Feature
Identifier: org.wso2.carbon.message.flows.feature.group

Modules use handlers to engage in different message flows at defined phases. You can observe the handlers invoked in each phase of each flow in real time. For example, the Apache/Rampart module defines handlers in the security phase of each flow, which handles the security aspects of the messages that are transferred through these flows. Therefore, if the Rampart module is engaged, you can see the Apache/Rampart handlers in the message flows in real time.

Follow the instructions below to access the Message Flows.

1. Log in to the management console and select Monitor → Message Flows.
2. The Message Flows page displays a graphical view of the message flows. There are four different flows defined in the system:
   a. In Flow: A correct message coming into the system.
   b. Out Flow: A correct message going out of the system.
   c. In Fault Flow: A faulty message coming into the system.
   d. Out Fault Flow: A faulty message going out of the system.
3. In each flow, a message passes through a set of phases to reach the service. These phases vary according to the currently engaged modules within the system. The interface displays the current phases in each and every flow as shown in the diagram below.

4. In the graphical view of the message flows, click the links to get a view of the engaged handlers in each phase. For example, the figure below shows the handlers engaged in the Addressing phase at system start up.

5. You can see the text view of message flows by clicking Show Text View.
5.

6. The page with the text view of message flows appears. The textual view provides the name and the fully qualified classes of all handlers within each and every phase.

Monitoring Performance Statistics

WSO2 products provide a range of performance statistics on a running Carbon instance. These statistics include information about memory availability, request count, server name, server start time, system up time, active services, total memory, average, minimum, maximum response times etc. Statistics are accessible through the management console of a running Carbon instance.

Access and performance statistics are available in system level as follows:

System-Level Statistics

Follow the instructions given below to access system-level statistics.

1. Log in to the management console and click **System Statistics** in the **Monitor** tab:
2. The System Statistics page appears as follows:

The following information is available:
- Service Summary
- Server Information
- Response Time Graph
- Memory Graph
- Statistics Configuration Panel

Service Summary
This panel provides the following information:

- **Average Response Time** - The average amount of time taken by the mediation channel to mediate a message (in milliseconds).
- **Minimum Response Time** - The least amount of time taken by the mediation channel to mediate a message (in milliseconds).
- **Maximum Response Time** - The most amount of time taken by the mediation channel to mediate a message (in milliseconds).
- **Total Request Count** - The total number of messages received and mediated through the mediation channel.
- **Total Response Count** - The total number of messages sent and mediated through the mediation channel.
- **Total Fault Count** - The number of messages that triggered faults while being mediated through the channel.
- **Active Services** - The number of currently active services.

### Server Information

<table>
<thead>
<tr>
<th>Server</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>192.168.1.5</td>
</tr>
<tr>
<td>Server Start Time</td>
<td>2011-10-10 17:55:46</td>
</tr>
<tr>
<td>System Up Time</td>
<td>0 day(s) 0 hr(s) 36 min(s) 9 sec(s)</td>
</tr>
<tr>
<td>Memory Allocated</td>
<td>494.94 MB</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>326.07 MB</td>
</tr>
</tbody>
</table>

This panel provides the following information:

- **Host** - Shows the IP address of the server.
- **Server Start Time** - Shows the time when the server started.
- **System Up Time** - Shows the time the server has been up and running.
- **Memory Allocated** - Shows the memory capacity of the server.
- **Memory Usage** - Shows the used memory of the server.

### Response Time Graph

This graph shows the temporal variation of the Average Response time.
Memory Graph

This graph shows the temporal variation of the server Memory.

Statistics Configuration Panel

The Statistics Configuration panel is provided to customize the System Statistics display by configuring the level information that can be viewed on the panel.

The following information can be configured:

- **Statistics Refresh Interval (milliseconds)**: Allows you to specify the interval of the statistics refresh. A smaller value refreshes the statistics display faster.
- **Response Time Graph**: Allows you to specify the X and Y parameters of the Response Time graph.
  - X-SCALE (units)
  - X-Width (px)
- **Memory Graph**: Allows you to specify the X and Y parameters of the Memory graph.
  - X-SCALE (units)
  - X-Width (px)

If you want to reset the previous values before submitting the page after editing, click **Reset**. The **Restore Defaults** button sets the default values in the corresponding fields.

Monitoring SOAP Messages

The SOAP Tracer provided by WSO2 is a tool that displays all the SOAP messages, including the message requests and responses that goes through your system, when serviced are invoked.

This functionality is provided by the following feature:

**Name**: WSO2 Carbon - SOAP Tracer Feature
**Identifier**: org.wso2.carbon.soaptracer.feature.group

Note that by default this feature is turned off. You have to enable tracing when you use it as explained in the following section. Turning on the SOAP Tracer feature may impose a significant performance overhead. This is because all the SOAP messages will always be completely built (deferred building will not be done) and stored in the database by WSO2 Data Services. Hence this option should be used with caution.
Enabling the SOAP Tracer

Follow the instructions below to access the SOAP Tracer.

1. Log in to the management console and select Monitor -> SOAP Tracer.

2. In the drop-down menu, select Yes.

3. The tracer will show the messages of the operations that were invoked. For example,
By using the SOAP Tracer, you can see the SOAP messages with their time-stamps, service name, operation invoked and the number of requests to the server. The most recent SOAP messages are listed at the top. When a particular SOAP message is selected, its "Request" and "Response" can be viewed.

**Note**

This tracer does not apply to operations invoked in the admin services. They are filtered out.

4. If you want to find a message, fill in the Filter field with a word (or a part of word) in the message and click Search.

5. You will see the message in the Messages list and its full description will be shown in the Request or Response text area. For example,
Monitoring TCP-Based Messages

Users can view and monitor the messages passed along a TCP-based conversation using the TCPMon utility. Therefore, this is a convenient tool, particularly useful for debugging when you develop Web services. TCPMon is an Apache project distributed under Apache 2.0 License.

TCPMon is not dependent on any third party libraries. Its user interface is based on a swing UI and works on almost all platforms that support Java.

- Starting TCPMon
- Message Monitoring with TCPMon
- Other Usages of TCPMon

Starting TCPMon

TCPMon is available in the `<PRODUCT_HOME>/bin` directory of any WSO2 Carbon based product distribution. Alternatively, you can download TCPMon from Apache and run the tool.

- Running TCPMon (from Carbon product pack)
- Running TCPMon (downloaded from Apache)

Running TCPMon (from Carbon product pack)

Ensure that the following prerequisites are fulfilled in order to run TCPMon.

- Install JDK 1.4 or later version.
- Set the JAVA_HOME variable. This setting is required only if you are using the TCPMon available in the WSO2 Carbon based product pack.

For information on how to set the JAVA_HOME variable, go to Installing the Product, select the instructions relevant to your operating system and refer the 'Setting JAVA_HOME' section.

To run the TCPMon available with your WSO2 Carbon product pack:

1. Go to `<PRODUCT_HOME>/bin` directory of your product pack.
2. Execute the following command to run the tool.

   For Windows

   tcpmon.bat

   For Linux
Running TCPMon (downloaded from Apache)

To download TCPMon from Apache and run the tool:

1. Download TCPMon from the following location: http://archive.apache.org/dist/ws/tcpmon/1.0/tcpmon-1.0-bin.zip.
2. Extract tcpmon-1.0-bin.zip archive.
3. Go to the build of the extracted directory to find the execution script.
4. Execute the following command to run the tool.

   For Windows
  (tcpmon.bat
   
   For Linux
   ./tcpmon.sh

Message Monitoring with TCPMon

The most common usage of TCPMon is as an intermediary, which monitors the communication between the client (front end) and the back end server. That is, the messages sent from the client are received by the intermediary instead of the back end server. These messages are then forwarded to the back end server from the intermediary.

Monitoring Messages between Client and Server

The following diagram depicts a typical communication between the front end client and the back end server. 80 is the listening port of the back end server which receives the messages from the client:

The following diagram depicts how TCPMon is placed between the client and the server in order to monitor the messages. 8081 is the listening port in TCPMon which receives the messages from the client instead of the back end server:
To monitor messages from client to server using TCPMon:
1. Start TCPMon. Follow the instructions on Starting TCPMon.
2. Give 8081 (the listening port of TCPMon) in the Listen Port field (This could be any unused port in your local machine).
3. Give the address of the back end server as the target hostname. For example, if you are monitoring messages sent to www.apache.org, enter this web address as the hostname.
4. Give 80 as the target port, which is the listening port of www.apache.org.

5. Click Add to save the setting.

7. A new tab in TCPMon will indicate the 8081 port. You can view the requests and responses passing through TCPMon as shown below.

As an intermediary, TCPMon only receives messages and forwards them to the back end server. Therefore, it is a safe tool to be used for debugging purposes.

Note that TCPMon cannot be used to view messages transferred over https protocol.
The options at the bottom of the screen can be used to have the messages in XML format (useful in debugging Web services), to save and resend the messages and also to switch the layout of the message windows.

### Other Usages of TCPMon

TCPMon is primarily used for message monitoring. Additionally, TCPMon can also be used for sending requests to web services and as a proxy service. Refer [Starting TCPMon](#) for details on how to start the tool.

- **Sending Requests for Web Services**
- **As a Proxy**
- **Advanced Settings**

**Sending Requests for Web Services**

TCPMon can also be used as a request sender for Web services. The request SOAP message can be pasted on the send screen and sent directly to the server.
As a Proxy

TCPMon can act as a proxy. To start it in proxy mode, select the Proxy option. When acting as a proxy, TCPMon only needs the listener port to be configured.

Advanced Settings

TCPMon can simulate a slow connection, in which case the delay and the bytes to be dropped can be configured. This is useful when testing Web services.

Monitoring with WSO2 Carbon Metrics

WSO2 products based on Carbon 4.4.x Kernel versions are shipped with JVM Metrics, which allows you to monitor statistics of your product server using
Java Metrics. The Java Metrics library consists of a variety of metrics that can be used for monitoring. With the WSO2 Carbon Metrics API, we have enabled all the metrics that are required for effectively monitoring WSO2 products.

The following sections explain how the Carbon Metrics functionality is used in WSO2 products:

- Setting Up Carbon Metrics
- Using JVM Metrics
- Using Messaging Metrics

### Setting Up Carbon Metrics

See the following topics for details on how to configure the metrics functionality for your product:

- Enabling Metrics and Storage Types
- Configuring Metrics Properties

### Enabling Metrics and Storage Types

Given below are the configurations that should be in place for your WSO2 product to use the metrics feature. You need to first enable metrics for your server and then enable the required storage types (reporters) that will be 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 `<PRODUCT_HOME>/repository/conf/metrics.xml` file. Alternatively, you can enable metrics at the time of starting the server by using the following command:

```
-Dmetrics.enabled=true
```

Once metrics are enabled, the metrics dashboard will be updated for your product.

#### Configuring the storage of metrics

WSO2 products (based on Carbon 4.4.x Kernel versions) are 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 `<PRODUCT_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 will be 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 or not metrics monitoring is enabled for JMX.</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>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>This parameter specifies whether or not metrics monitoring is enabled for CSV.</td>
<td>Boolean</td>
<td>false</td>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>The location where the CSV files are stored.</td>
<td>String</td>
<td><code>${carbon.home}/repository/logs/metrics/</code></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 milliseconds.</td>
<td>Integer</td>
<td>60</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**JDBC**

**H2 is not recommended in production**

The H2 database is NOT recommended in enterprise testing and production environments. It has lower performance, clustering limitations, and can cause file corruption failures. Please use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, or MS SQL instead. See the instructions on setting up an RDBMS.

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 or not metrics monitoring is enabled for JDBC.</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/WSO2Metric sDB</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 milliseconds.</td>
<td>Integer</td>
<td>60</td>
<td></td>
<td>This value is specified in milliseconds.</td>
</tr>
<tr>
<td>ScheduledCleanup</td>
<td>This element contains parameters relating to scheduled cleanup. The possible values are Enabled, Schedule dCleanupPeriod and DaysToKeep. Scheduled cleanup involves scheduling a task to clear metric data in the database after a specified time interval. This is done to avoid excessive memory usage.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Data type</td>
<td>Default value</td>
<td>Mandatory/Optional</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td>---------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>ScheduledCleanup/Enabled</td>
<td>This parameter specifies whether or not scheduled cleanup is enabled.</td>
<td>Boolean</td>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScheduledCleanup/ScheduledCleanupPeriod</td>
<td>The number of milliseconds that should elapse after a clean-up task before the next clean-up task is carried out.</td>
<td>Integer</td>
<td>86400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScheduledCleanup/DaysToKeep</td>
<td>The number of days during which the scheduled clean-up task should be carried out.</td>
<td>Integer</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you have enabled JDBC, then you also need to specify a datasource configuration, which will be used to create the connection between the WSO2 product 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>XML element</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>Mandatory</td>
</tr>
<tr>
<td>&lt;provider&gt;</td>
<td></td>
<td>The datasource provider, which should implement org.wso2.carbon.datasource.common.DataSource Reader. 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>Mandatory</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 will be bound.</td>
<td>String</td>
<td></td>
<td>Mandatory if specifying JNDI configuration</td>
</tr>
<tr>
<td>Environment</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;environment&gt;</td>
<td>The container element in which you specify the following JNDI properties:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* java.naming.factory.initial: Selects the registry service provider as the initial context.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* java.naming.provider.url: Specifies the location of the registry when the registry is being used as the initial context.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition</th>
<th>type</th>
<th>String</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;definition&gt;</td>
<td>The container element for the data source definition. Set the type attribute to &quot;RDBMS&quot;, or to &quot;custom&quot; if you’re creating a custom type. The &quot;RDBMS&quot; datasource reader expects a configuration element with the sub elements listed below.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>The container element for the RDBMS properties.</th>
<th>String</th>
<th>Mandatory if definition type is RDBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The connection URL that passes the JDBC driver to establish the connection.</td>
<td>URL</td>
<td>Mandatory</td>
</tr>
<tr>
<td>username</td>
<td>The connection user name that passes the JDBC driver to establish the connection.</td>
<td>String</td>
<td>Optional</td>
</tr>
<tr>
<td>password</td>
<td>The connection password that passes the JDBC driver to establish the connection.</td>
<td>String</td>
<td>Optional</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the JDBC driver.</td>
<td>Fully qualified Java class</td>
<td>Mandatory</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated from this pool at the same time.</td>
<td>Integer</td>
<td>Optional</td>
</tr>
</tbody>
</table>
### <maxWait>
- **Description:** Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception.
- **Type:** Integer
- **Value:** 30000 (30 seconds)
- **Optional:** Yes

### <testOnBorrow>
- **Description:** Specifies whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be 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.
- **Type:** Boolean
- **Value:** false
- **Optional:** Yes

### <validationQuery>
- **Description:** The SQL query that is used for validating connections from this pool before returning them to the caller. It specified, this query does not have to return any data, as it cannot throw an SQLException. The default value is "null". Example values are `SELECT 1 (mysql), select 1 from dual (oracle), SELECT 1 (MS Sql Server)`.
- **Type:** String
- **Value:** null
- **Optional:** No

### <validationInterval>
- **Description:** To avoid excess validation, a connection will be validated at this frequency, at most (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).
- **Type:** Long
- **Value:** 30000 (30 seconds)
- **Optional:** Yes

### 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
This is the main configuration file for metrics

<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>
    <!-- Polling Period in seconds -->
    <PollingPeriod>60</PollingPeriod>
  </CSV>

  <JDBC>
    <Enabled>true</Enabled>
    <!-- Source of Metrics, which will be used to identify each metric in database -->
    <!-- Commented to use the hostname -->
    <Source>Carbon</Source>
    <!-- JNDI name of the data source to be used by the JDBC Reporter. -->
    This data source should be defined in a *
    *-datasources.xml
    file in conf/datasources directory.
    <!-- SourceName>jdb/WSO2MetricsDB</DataSourceName>
    <!-- Polling Period in seconds -->
    <PollingPeriod>60</PollingPeriod>
    <ScheduledCleanup>
      <!-- Schedule regular deletion of metrics data older than a set number of days. -->
      It is strongly recommended that you enable this job to ensure your metrics tables do not get extremely large. Deleting data older than seven days should be sufficient.
      <!-- This is the period for each cleanup operation in seconds -->
      <ScheduledCleanupPeriod>86400</ScheduledCleanupPeriod>
      <!-- The scheduled job will clean up all data older than the specified days -->
      <DaysToKeep>7</DaysToKeep>
    </ScheduledCleanup>
  </JDBC>
</Reporting>
</Metrics>
Once you have enabled Metrics as explained above, proceed to the section on configuring metric properties for information on how to configure the gauges on the metrics dashboard.

### Configuring Metrics Properties

When the monitoring capability with metrics is enabled in your product, the metrics dashboard will be available when you log into the management console. The metrics dashboard will provide a page for viewing the statistics that use JVM metrics. Further, depending on the WSO2 product that you are using, there will be a separate page for viewing product-specific statistics using metrics.

The `<PRODUCT_HOME>/repository/conf/metrics.properties` file specifies the properties that correspond to the gauges for JVM metrics in the metrics dashboard. 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 will apply. The root level is defined as shown in the following example in the `metrics.properties` file.

```plaintext
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 the `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 will overrule that of the parent. For example, if we have `metric.level.jvm.memory=INFO` in the `<PRODUCT_HOME>/repository/conf/metrics.properties` file, all metrics under `jvm.memory` will 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

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metric.level.jvm.buffers</td>
<td>OFF</td>
<td>The gauge showing the current number of distinct buffers.</td>
</tr>
</tbody>
</table>

#### 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 that will 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 MB.</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 MB.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.non-heap</td>
<td>INFO</td>
<td>The gauge for showing the used code cache and used CMS Perm Gen in WSO2 MB.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.total</td>
<td>INFO</td>
<td>The gauge for showing the total memory currently available for the JVM.</td>
</tr>
<tr>
<td>metric.level.jvm.memory.pools</td>
<td>OFF</td>
<td>The gauge for showing the used and available memory for JVM in the memory pool.</td>
</tr>
</tbody>
</table>

### 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 will apply to all the remaining 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 in deadlock 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.threads.runnable.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 MB 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 with Timed_Waiting status.</td>
</tr>
<tr>
<td>metric.level.jvm.threads.waiting.count</td>
<td>OFF</td>
<td>The gauge for showing the number of threads with Waiting status 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>
Using JVM Metrics

JVM metrics are Java metrics enabled by default in WSO2 products for the purpose of monitoring general statistics related to server performance. Follow the steps given below to use the JVM metrics dashboard in a WSO2 product.

For detailed instructions on enabling/disabling JVM metrics and configuring the metric gauges, see Setting up Carbon Metrics.

1. Log in to the management console of your 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.

![CPU Load Graph](image-url)

Legend:
- Blue line: Process CPU Load
- Green line: System CPU Load

![System Load Average Graph](image-url)

Legend:
- Blue line: System Load Average
• **Memory**
  Click **Memory** to view statistics relating to the memory as shown below.

![Memory Statistics](image)

• **Threading**
  Click **Threading** to view statistics relating to threading as shown below.

![Threading Statistics](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.

Using Messaging Metrics
Messaging metrics are the Java metrics enabled in the WSO2 Message Broker (MB) or the Broker Profile of WSO2 EI for the purpose of monitoring broker-specific statistics.

The metrics feature is enabled in the broker by default. See the topic on setting up WSO2 Carbon metrics for details on how metrics are enabled.

Follow the steps given below for instructions on using the Messaging Metrics dashboard.

1. Log in to the management console of the broker and click Monitor -> Metrics -> Messaging Metrics.

2. The View Metrics page will open. At the top of this page, you will find the following panel:
3. First, select the Source from the drop-down list. In a clustered setup, you must specify the broker node that you want to monitor.
4. You can specify the time interval for which the statistics displayed are valid. By default, you will see statistics from the last 5 minutes.
5. In the Views section, you will find buttons corresponding to the different types of metrics that you want to view. You can click the relevant button to view the statistics. Given below are the statistics corresponding to each button:

- **Disruptor**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Messages in Inbound</td>
<td>The Disruptor is a new open-source concurrency framework, designed as a high-performance mechanism for inter-thread messaging. The current number of messages in the inbound disruptor can be viewed here.</td>
</tr>
<tr>
<td>Disruptor</td>
<td></td>
</tr>
<tr>
<td>Total Acks in Inbound Disruptor</td>
<td>The current number of acknowledgments in the inbound disruptor.</td>
</tr>
<tr>
<td>Total Messages in Outbound</td>
<td>The current number of messages in the outbound disruptor.</td>
</tr>
<tr>
<td>Disruptor</td>
<td></td>
</tr>
</tbody>
</table>

- **Publish & Subscribe**
5. Metric Description

Total Queue Subscribers
This metric shows the total number of active queue subscribers for a particular broker node. This is an INFO level metric.

Total Topic Subscribers
The total number of active topic subscribers.

Total Channels
The total number of active channels.

- Messages & Acknowledgements

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages Received/sec</td>
<td>This metric provides the number of messages received per second by a particular broker node. This metric is calculated when a message reaches the server.</td>
</tr>
<tr>
<td>Messages Published/sec</td>
<td>This metric provides the number of messages published per second. This metric is calculated when the server publishes a message to a subscriber.</td>
</tr>
<tr>
<td>Acknowledges Received /sec</td>
<td>This metric provides the number of acknowledgments received from publishers per second.</td>
</tr>
<tr>
<td>Acknowledges Sent /sec</td>
<td>This metric provides the number of acknowledgments sent to publishers per second.</td>
</tr>
</tbody>
</table>
• Database

Database Read Time (milliseconds)

Database Read Rate (reads/second)

Database Write Time (milliseconds)
### Monitoring Server Health

This document is work in progress!

The Carbon Health Check API can be used to check the health of a Carbon server. The sections below guide you through using this API.

- Deploying the API
- Configuring the API
- Invoking the API
- Error responses
- Adding new health checkers

**Note:** This API is only supported for WSO2 Carbon products that are running on Java 8 or a later version.

There are three health checkers available by default:

1. **Data sources health checker** - This checker goes through the data sources that are configured in the `master-datasources.xml` file and checks if the active connection count surpasses a healthy percentage limit (e.g., 80%) of the maximum allowed connections count. This checker also tests the connection from each data source to see whether the connection is successful.

2. **Server startup health checker** - This checker uses the ServerAdmin service to check if the server status is RUNNING.

3. **Super tenant user store health checker** - This checker iterates through configured user stores of the super tenant domain and attempts to invoke the `isExistingUser` method to check whether a failure occurs.

### Deploying the API

This API is supported by default from WSO2 Identity Server 5.7.0 onwards. It is available by default for WSO2 IS versions 5.5.0 and 5.6.0 only as a WUM update. For more information on how to update using WUM, see the [Getting WSO2 Updates documentation](#).

If you are using a WSO2 product version that supports this feature by default (either in a fresh pack or via a WUM update), skip the instructions in this section and proceed to the **configuring the API** section.

This section guides you through deploying the Carbon Health Check components in a WSO2 Carbon product that does not support this feature by default.

---

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database write latency</td>
<td>The average time taken for database write calls.</td>
</tr>
<tr>
<td>Database read latency</td>
<td>The average time taken for database read calls.</td>
</tr>
<tr>
<td>Database Method latency</td>
<td>The average time taken by message store implementation methods.</td>
</tr>
</tbody>
</table>

---

#### Metric Description

- **Database write latency**: The average time taken for database write calls.
- **Database read latency**: The average time taken for database read calls.
- **Database Method latency**: The average time taken by message store implementation methods.
1. Download the `org.wso2.carbon.healthcheck.server.feature-{version-number}.zip` and extract it. This folder is referred to as `<API_HOME>` in this document.
2. Copy the `org.wso2.carbon.healthcheck.api.core-{version-number}.jar` found in the `<API_HOME>/plugins` directory and paste it in the `<PRODUCT_HOME>/repository/components/dropins` directory.
3. Copy the war file `api#health-check#v1.0.war` found in the `<API_HOME>/features/org.wso2.carbon.healthcheck.server_1.0.0` directory and paste it in the `<PRODUCT_HOME>/repository/deployment/server/webapps` directory.
4. (Optional step) Copy the `health-check.config.xml` configuration file found in the `<API_HOME>/features/org.wso2.carbon.healthcheck.server_1.0.0` directory to your `<PRODUCT_HOME>/repository/conf/` directory.

### Configuring the API

This feature is disabled by default. To enable the API, set the `<Enable>` property in the `health-check-config.xml` file to `true`.

If the feature has not been enabled successfully, a request to the API will only return a 200 OK response.

#### Sample health-check-config.xml file

```
<CarbonHealthCheckConfigs>
  <Enable>true</Enable>
  <HealthCheckers>
    <HealthChecker name="DataSourceHealthChecker" orderId="97" enable="true">
      <!-- Property name="monitored.datasources">jdbc/WSO2CarbonDB,jdbc/WSO2MetricsDB,jdbc/WSO2UMDB</Property>-->  
      <Property name="pool.usage.limit.percentage">80</Property>
    </HealthChecker>
    <HealthChecker name="SuperTenantUSHealthChecker" orderId="98" enable="true">
      <!-- Property name="monitored.user.stores">primary,sec</Property-->  
    </HealthChecker>
  </HealthCheckers>
</CarbonHealthCheckConfigs>
```

- A health checker can be enabled or disabled using the `enable` attribute.
- The execution order in which the health checkers are executed can be configured using the `orderId` attribute.
- The properties configured under each health checker will be available for each health checker at runtime.

### Invoking the API

This is an open API which should ideally be blocked at the load balancer level. To invoke it, start the WSO2 product and send a GET request to the health check API. A sample cURL command is shown below.

```
curl -k -v https://{hostname}:{port}/api/health-check/v1.0/health
```

If the request is successful, you will receive a 200 OK response (similar to the one shown below) with a list of health check results.
Error responses

The following responses are possible error responses that you may receive.

The code block below shows a sample 503 Unavailable response with an array of errors.

```
{
    "errors": [
        {
            "code": "HC_00001",
            "message": "Error while getting database connection for datasource: jdbc/DISCONNECTED",
            "description": "Network is unreachable (connect failed)"
        },
        {
            "code": "HC_00003",
            "message": "Error while checking health of USM with domain: SEC",
            "description": "Access denied for user 'root'@'localhost' (using password: YES)"
        }
    ]
}
```

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC_00001</td>
<td>Data source connectivity error.</td>
</tr>
<tr>
<td>HC_00002</td>
<td>Number of connections in data source exceeds the healthy percentage.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HC_00003</td>
<td>Error while testing connectivity to the user store using the <code>isExistingUser</code> operation.</td>
</tr>
<tr>
<td>HC_00004</td>
<td>Server status is not running.</td>
</tr>
<tr>
<td>HC_00005</td>
<td>Error listing user stores.</td>
</tr>
</tbody>
</table>

### Adding new health checkers

To add a new health checker, you can implement the HealthChecker API and register the health checker as an OSGI service. To deploy it, copy the `.jar` file to the `<PRODUCT_HOME>/repository/component/lib/` directory or the OSGI bundle and paste it in the `<PRODUCT_HOME>/repository/component/dropins/` directory.
Getting WSO2 Updates

**WSO2 Updates** include any improvements (e.g., bug fixes, security fixes) that are released by WSO2 on top of a released WSO2 product version. With updates, you do not have to wait until the next product version is released to get the fixes you want. Find out more about WSO2 updates at [https://wso2.com/updates](https://wso2.com/updates).

You can get updates using the WSO2 in-place updates tool or WSO2 Update Manager (WUM tool). For more information, see [WSO2 Updates](https://wso2.com/updates).
Reference Guide

The following topics provide reference information for working with WSO2 Carbon:

- Configuration Files
- Database Upgrade Guide
- Product Startup Options
- Supported Cipher Suites
- Directory Structure of WSO2 Products
- WSO2 Patch Application Process

Configuration Files

Given below is an overview of the configuration files that are shipped with WSO2 Carbon Kernel. All WSO2 products that are based on Carbon Kernel 4.4.x will inherit the configurations given below. While most of the configurations in these files are common across the Carbon 4.4.x platform, some configurations may only be used for specific features used by certain products.

We have listed below all the configuration files that are shipped with Carbon 4.4.x. You can follow the link to view the contents in each configuration file from the Carbon 4.4.7 project in github. You will also find descriptions of the elements included in the files.

If you are using a WSO2 product that is based on Carbon 4.4.x, you will find all these configuration files stored in the `<PRODUCT_HOME>/repository/conf` directory and sub-directories of your product pack.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>carbon.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf</code></td>
</tr>
<tr>
<td>registry.xml</td>
<td></td>
</tr>
<tr>
<td>user-mgt.xml</td>
<td></td>
</tr>
<tr>
<td>log4j.properties</td>
<td></td>
</tr>
<tr>
<td>catalina-server.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/tomcat</code></td>
</tr>
<tr>
<td>tomcat-users.xml</td>
<td></td>
</tr>
<tr>
<td>web.xml</td>
<td></td>
</tr>
<tr>
<td>context.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/tomcat/carbon/META-INF</code></td>
</tr>
<tr>
<td>web.xml</td>
<td></td>
</tr>
<tr>
<td>Owasp.CsrfGuard.Carbon.properties</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/security</code></td>
</tr>
<tr>
<td>authenticators.xml</td>
<td></td>
</tr>
<tr>
<td>xss-patterns.properties</td>
<td></td>
</tr>
<tr>
<td>master-datasources.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/datasources</code></td>
</tr>
<tr>
<td>axis2.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/axis2</code></td>
</tr>
<tr>
<td>axis2_client.xml</td>
<td></td>
</tr>
<tr>
<td>tenant-axis2.xml</td>
<td></td>
</tr>
<tr>
<td>config-validation.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/etc</code></td>
</tr>
<tr>
<td>email-admin-config.xml</td>
<td><code>&lt;PRODUCT_HOME&gt;/repository/conf/email</code></td>
</tr>
</tbody>
</table>

See the following topics for more information on configuration files:

- Configuring catalina-server.xml
- Configuring master-datasources.xml
- Configuring registry.xml
- Configuring user-mgt.xml
- Configuring config-validation.xml

Configuring catalina-server.xml

Users can change the default configurations by editing the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file using the information given below.
XML Elements

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
<th>Default value</th>
<th>Mandatory/Opt</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Server&gt;</td>
<td></td>
<td>A Server element represents the entire Catalina servlet container. Therefore, it must be the single outermost element in the conf/server.xml configuration file. Its attributes represent the characteristics of the servlet container as a whole.</td>
<td></td>
<td></td>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>shutdown</td>
<td></td>
<td>The command string that must be received via a TCP/IP connection to the specified port number, in order to shut down Tomcat.</td>
<td>String</td>
<td>SHUTDOWN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>port</td>
<td></td>
<td>The TCP/IP port number on which this server waits for a shutdown command. Set to -1 to disable the shutdown port. Note: Disabling the shutdown port works well when Tomcat is started using Apache Commons Daemon (running as a service on Windows or with jsvc on un*xes). It cannot be used when running Tomcat with the standard shell scripts though, as it will prevent shutdown.bat</td>
<td>sh and catalina.bat</td>
<td>sh from stopping it gracefully.</td>
<td>Int</td>
<td>8005</td>
</tr>
</tbody>
</table>

Sample: `<Server shutdown="SHUTDOWN" port="8005">`
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Service&gt;</td>
<td>A Service element represents the combination of one or more Connector components that share a single Engine component for processing incoming requests. One or more Service elements may be nested inside a Server element.</td>
<td>Mandatory</td>
<td>&lt;Service name=&quot;Catalina&quot; className=&quot;org.wso2.carbon.tomcat.ext.service.ExtendedStandardService&quot;&gt;</td>
</tr>
<tr>
<td>name</td>
<td>The display name of this Service, which will be included in log messages if you utilize standard Catalina components. The name of each Service that is associated with a particular Server must be unique.</td>
<td>String</td>
<td>Catalina</td>
</tr>
<tr>
<td>className</td>
<td>Java class name of the implementation to use. This class must implement the org.apache.catalina.Service interface. If no class name is specified, the standard implementation will be used.</td>
<td>String</td>
<td>org.wso2.carbon.tomcat.ext.service.ExtendedStandardService</td>
</tr>
<tr>
<td>&lt;Connector&gt;</td>
<td></td>
<td>Optional</td>
<td>Connect, accept, compress, maxKeep, disable, accept, redirect, protocol</td>
</tr>
</tbody>
</table>

Connect, accept, compress, maxKeep, disable, accept, redirect, protocol
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The TCP port number on which this Connector will create a server socket and await incoming connections. Your operating system will allow only one server application to listen to a particular port number on a particular IP address. If the special value of 0 (zero) is used, then Tomcat will select a free port at random to use for this connector. This is typically only useful in embedded and testing applications.</td>
<td>Int</td>
<td>9763</td>
</tr>
<tr>
<td>URIEncoding</td>
<td>This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL.</td>
<td>Int</td>
<td>UTF-8</td>
</tr>
<tr>
<td>compressableMimeType</td>
<td>The value is a comma separated list of MIME types for which HTTP compression may be used.</td>
<td>String</td>
<td>text/html,text/javascript,application/javascript,application/xml,text/css,application/octet-stream,text/xml, application/x-java, text/javascript, text/x-javascript, text/javascript, application/javascript, application/xml, text/css, application/ x-javascript, text/javascript, image/gif, image/jpeg, image/png, image/png, image/jpeg</td>
</tr>
<tr>
<td>noCompressionUserAgents</td>
<td>The value is a regular expression (using java.util.regex) matching the user-agent header of HTTP clients for which compression should not be used, because these clients, although they do advertise support for the feature, have a broken implementation.</td>
<td>String</td>
<td>gozilla, traviata</td>
</tr>
<tr>
<td>compressionMinSize</td>
<td>If compression is set to &quot;on&quot; then this attribute may be used to specify the minimum amount of data before the output is compressed.</td>
<td>Int</td>
<td>2048</td>
</tr>
<tr>
<td>compression</td>
<td></td>
<td>String</td>
<td>on</td>
</tr>
</tbody>
</table>
The Connector may use HTTP/1.1 GZIP compression in an attempt to save server bandwidth. The acceptable values for the parameter is “off” (disable compression), “on” (allow compression, which causes text data to be compressed), “force” (forces compression in all cases), or a numerical integer value (which is equivalent to “on”, but specifies the minimum amount of data before the output is compressed). If the content-length is not known and compression is set to “on” or more aggressive, the output will also be compressed. If not specified, this attribute is set to “off”.
Note: There is a tradeoff between using compression (saving your bandwidth) and using the sendfile feature (saving your CPU cycles). If the connector supports the sendfile feature, e.g. the NIO connector, using sendfile will take precedence over compression. The symptoms will be that static files greater that 48 Kb will be sent uncompressed. You can turn off sendfile by setting useSendfile attribute of the connector, as documented below, or change the sendfile usage threshold in the configuration of the DefaultServlet in the default conf/web.xml or in the web.xml of your web application.

<table>
<thead>
<tr>
<th>server</th>
<th>Overrides the Server header for the http response. If set, the value for this attribute overrides the Tomcat default and any Server header set by a web application. If not set, any value specified by the application is used. Most often, this feature is not required.</th>
<th>String</th>
<th>WSO2 Carbon Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptCount</td>
<td>The maximum queue length for incoming connection requests when all possible request processing threads are in use. Any requests received when the queue is full will be refused.</td>
<td>Int</td>
<td>200</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>maxKeepAliveRequests</td>
<td>The maximum number of HTTP requests which can be pipelined until the connection is closed by the server. Setting this attribute to 1 will disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. Setting this to -1 will allow an unlimited amount of pipelined or keep-alive HTTP requests.</td>
<td>Int</td>
<td>200</td>
</tr>
<tr>
<td>connectionUploadTimeout</td>
<td>Specifies the timeout, in milliseconds, to use while a data upload is in progress. This only takes effect if disableUploadTimeout is set to false.</td>
<td>Int</td>
<td>120000</td>
</tr>
<tr>
<td>disableUploadTimeout</td>
<td>This flag allows the servlet container to use a different, usually longer connection timeout during data upload.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>minSpareThreads</td>
<td>The minimum number of threads always kept running.</td>
<td>Int</td>
<td>50</td>
</tr>
<tr>
<td>maxThreads</td>
<td>The maximum number of request processing threads to be created by this Connector, which therefore determines the maximum number of simultaneous requests that can be handled. If an executor is associated with this connector, this attribute is ignored as the connector will execute tasks using the executor rather than an internal thread pool.</td>
<td>Int</td>
<td>250</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>acceptorThreadCount</td>
<td>The number of threads to be used to accept connections. Increase this value on a multi CPU machine, although you would never really need more than 2. Also, with a lot of non keep alive connections, you might want to increase this value as well.</td>
<td>Int</td>
<td>2</td>
</tr>
<tr>
<td>maxHttpHeaderSize</td>
<td>The maximum size of the request and response HTTP header, specified in bytes.</td>
<td>Int</td>
<td>8192</td>
</tr>
<tr>
<td>bindOnInit</td>
<td>Controls when the socket used by the connector is bound. By default it is bound when the connector is initiated and unbound when the connector is destroyed. If set to false, the socket will be bound when the connector is started and unbound when it is stopped.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>redirectPort</td>
<td>If this Connector is supporting non-SSL requests, and a request is received for which a matching &lt;security-constraint&gt; requires SSL transport, Catalina will automatically redirect the request to the port number specified here.</td>
<td>Int</td>
<td>9443</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>SSLEnabled</td>
<td>Use this attribute to enable SSL traffic on a connector. To turn on SSL handshake/encryption/decryption on a connector set this value to true. The default value is false. When turning this value to true you will want to set the scheme and the secure attributes as well to pass the correct request.getScheme() and request.isSecure() values to the servlets. See SSL Support for more information.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>secure</td>
<td>Set this attribute to true if you wish to have calls to request.isSecure() to return true for requests received by this Connector. You would want this on an SSL Connector or a non-SSL connector that is receiving data from a SSL accelerator, like a crypto card, a SSL appliance or even a webserver.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>scheme</td>
<td>Set this attribute to the name of the protocol you wish to have returned by calls to request.getScheme(). For example, you would set this attribute to “https” for an SSL Connector.</td>
<td>String</td>
<td>https</td>
</tr>
<tr>
<td>config</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>clientAuth</td>
<td>Set to true if you want the SSL stack to require a valid certificate chain from the client before accepting a connection. Set to false if you want the SSL stack to request a client Certificate, but not fail if one isn't presented. A false value will not require a certificate chain unless the client requests a resource protected by a security constraint that uses CLIENT-CERT authentication.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>enableLookups</td>
<td>Set to true if you want calls to request.getRemoteHost() to perform DNS lookups in order to return the actual host name of the remote client. Set to false to skip the DNS lookup and return the IP address in String form instead (thereby improving performance). By default, DNS lookups are disabled.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>sslProtocol</td>
<td>The SSL protocol(s) to use (a single value may enable multiple protocols - see the JVM documentation for details). The permitted values may be obtained from the JVM documentation for the allowed values for algorithm when creating an SSLContext instance e.g. Oracle Java 6 and Oracle Java 7. Note: There is overlap between this attribute and sslEnabledProtocols.</td>
<td>String</td>
<td>TLS</td>
</tr>
</tbody>
</table>
### keystoreFile

This setting allows you to use separate keystore and security certificates for SSL connections. The location of the keystore file and the keystore password can be given for these parameters. Note that by default, these parameters point to the location and password of the default keystore in the Carbon server.

### <Engine>

The Engine element represents the entire request processing machinery associated with a particular Catalina Service. It receives and processes all requests from one or more Connectors, and returns the completed response to the Connector for ultimate transmission back to the client. Exactly one Engine element MUST be nested inside a Service element, following all of the corresponding Connector elements associated with this Service.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Mandatory</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Logical name of this Engine, used in log and error messages. When using multiple Service elements in the same Server, each Engine MUST be assigned a unique name.</td>
<td>String</td>
<td>Catalina</td>
<td><code>&lt;Engine name=&quot;Catalina&quot; defaultHost=&quot;localhost&quot;&gt;</code></td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td>Default Value</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>defaultHost</td>
<td>The default host name, which identifies the Host that will process requests directed to host names on this server, but which are not configured in this configuration file. This name MUST match the name attributes of one of the Host elements nested immediately inside.</td>
<td>String</td>
<td>localhost</td>
<td></td>
</tr>
<tr>
<td>&lt;Realm&gt;</td>
<td>A Realm element represents a &quot;database&quot; of usernames, passwords, and roles (similar to Unix groups) assigned to those users. Different implementations of Realm allow Catalina to be integrated into environments where such authentication information is already being created and maintained, and then utilize that information to implement Container Managed Security as described in the Servlet Specification. You may nest a Realm inside any Catalina container Engine, Host, or Context. In addition, Realms associated with an Engine or a Host are automatically inherited by lower-level containers, unless explicitly overridden.</td>
<td>Mandatory</td>
<td>&lt;Realm className=&quot;org.wso2.carbon.tomcat.ext.realms.CarbonTomcatRealm&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td>className</td>
<td>Java class name of the implementation to use. This class must implement the org.apache.catalina.realmManagerInterface.</td>
<td>String</td>
<td>org.wso2.carbon.tomcat.ext.realms.CarbonTomcatRealm</td>
<td></td>
</tr>
<tr>
<td>&lt;Host&gt;</td>
<td>The Host element represents a</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
virtual host, which is an association of a network name for a server (such as "www.mycompany.com") with the particular server on which Tomcat is running. For clients to be able to connect to a Tomcat server using its network name, this name must be registered in the Domain Name Service (DNS) server that manages the Internet domain you belong to - contact your Network Administrator for more information.

In many cases, System Administrators wish to associate more than one network name (such as www.mycompany.com and company.com) with the same virtual host and applications. This can be accomplished using the Host Name Aliases feature discussed below.

One or more Host elements are nested inside an Engine element. Inside the Host element, you can nest Context elements for the web applications associated with this virtual host. Exactly one of the Hosts associated with each Engine MUST have a name matching the defaultHost attribute of that Engine.

Clients normally use host names to identify the server they wish to connect to. This host name is also included in the HTTP request.
headers. Tomcat extracts the host name from the HTTP headers and looks for a Host with a matching name. If no match is found, the request is routed to the default host. The name of the default host does not have to match a DNS name (although it can) since any request where the DNS name does not match the name of a Host element will be routed to the default host.

<table>
<thead>
<tr>
<th>name</th>
<th>Usually the network name of this virtual host, as registered in your Domain Name Service server. Regardless of the case used to specify the host name, Tomcat will convert it to lower case internally. One of the Hosts nested within an Engine MUST have a name that matches the defaultHost setting for that Engine. See Host Name Aliases for information on how to assign more than one network name to the same virtual host.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>localhost</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>appBase</td>
<td>The Application Base directory for this virtual host. This is the pathname of a directory that may contain web applications to be deployed on this virtual host. You may specify an absolute pathname, or a pathname that is relative to the CATALINA_BASE directory. See Automatic Application Deployment for more information on automatic recognition and deployment of web applications. If not specified, the default of webapps will be used.</td>
</tr>
<tr>
<td>autoDeploy</td>
<td>This flag value indicates if Tomcat should check periodically for new or updated web applications while Tomcat is running. If true, Tomcat periodically checks the appBase and xmlBase directories and deploys any new web applications or context XML descriptors found. Updated web applications or context XML descriptors will trigger a reload of the web application. See Automatic Application Deployment for more information.</td>
</tr>
<tr>
<td>deployOnStartup</td>
<td>This flag value indicates if web applications from this host should be automatically deployed when Tomcat starts. See Automatic Application Deployment for more information.</td>
</tr>
<tr>
<td>unpackWARs</td>
<td>Set to true if you want web applications that are placed in the appBase directory as web application archive (WAR) files to be unpacked into a corresponding disk directory structure, false to run such web applications directly from a WAR file. WAR files located outside of the Host's appBase will not be expanded.</td>
</tr>
</tbody>
</table>

```xml
<Valve>
  The Access Log Valve creates log files in the same format as those created by standard web servers. These logs can later be analyzed by standard log analysis tools to track page hit counts, user session activity, and so on. The files produces by this Valve are rolled over nightly at midnight. This Valve may be associated with any Catalina container (Context, Host, or Engine), and will record ALL requests processed by that container. Some requests may be handled by Tomcat before they are passed to a container. These include redirects from /foo to /foo/ and the rejection of invalid requests. Where Tomcat can identify the Context that would have handled the request, the request/response will be logged in the AccessLog(s) associated Context, Host and Engine. Where Tomcat cannot identify the Context that
</Valve>
```
would have handled the request, e.g. in cases where the URL is invalid, Tomcat will look first in the Engine, then the default Host for the Engine and finally the ROOT (or default) Context for the default Host for an AccessLog implementation. Tomcat will use the first AccessLog implementation found to log those requests that are rejected before they are passed to a container.

The output file will be placed in the directory given by the directory attribute. The name of the file is composed by concatenation of the configured prefix, timestamp and suffix. The format of the timestamp in the file name can be set using the fileDateFormat attribute. This timestamp will be omitted if the file rotation is switched off by setting rotatable to false.

Warning: If multiple AccessLogValve instances are used, they should be configured to use different output files.

If sendfile is used, the response bytes will be written asynchronously in a separate thread and the access log valve will not know how many bytes were actually written. In this case, the number of bytes that was passed to the sendfile thread for writing
will be recorded in the access log valve.

<table>
<thead>
<tr>
<th>attributeName</th>
<th>description</th>
<th>type</th>
<th>java class name of the implementation to use.</th>
<th>org.wso2.carbon.tomcat.ext.valves.CarbonContextCreatorValve, org.apache.catalina.valves.AccessLogValve</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>A formatting layout identifying the various information fields from the request and response to be logged, or the word common or combined to select a standard format.</td>
<td>String</td>
<td>combined</td>
<td></td>
</tr>
<tr>
<td>suffix</td>
<td>The suffix added to the end of each log file name.</td>
<td>String</td>
<td>.log</td>
<td></td>
</tr>
<tr>
<td>prefix</td>
<td>The prefix added to the start of each log file name.</td>
<td>String</td>
<td>http_access_</td>
<td></td>
</tr>
<tr>
<td>directory</td>
<td>Absolute or relative path name of a directory in which log files created by this valve will be placed. If a relative path is specified, it is interpreted as relative to $CATALINA_BASE. If no directory attribute is specified, the default value is &quot;logs&quot; (relative to $CATALINA_BASE).</td>
<td>String</td>
<td>${carbon.home}/repository/logs</td>
<td></td>
</tr>
</tbody>
</table>
threshold | Minimum duration in seconds after which a thread is considered stuck. If set to 0, the detection is disabled. Note: since the detection is done in the background thread of the Container (Engine, Host or Context) declaring this Valve, the threshold should be higher than the backgroundProcessorDelay of this Container.

| Int | 600 |

### Configuring master-datasources.xml

Users can change the default configurations by editing the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file using the information in the following table.

#### XML Elements

Click on the table and use the left and right arrow keys to scroll horizontally. For sample values, see the Example below the table.

<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>Mandatory</td>
</tr>
<tr>
<td>&lt;provider&gt;</td>
<td></td>
<td>The datasource provider, which should implement or g.wso2.carbon.datasource.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>Mandatory</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>Description of the datasource.</td>
<td>String</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jndiConfig</td>
<td>The container element that allows you to expose this datasource as a JNDI datasource.</td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>The JNDI resource name to which this datasource will be bound.</td>
<td>String</td>
<td>Mandatory if specifying JNDI configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>environment</td>
<td>The container element in which you specify the following JNDI properties:</td>
<td>Fully qualified Java class</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java.naming.factory.initial: Selects the registry service provider as the initial context.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- java.naming.provider.url: Specifies the location of the registry when the registry is being used as the initial context.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>type 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>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>configuration</td>
<td>The container element for the RDBMS properties.</td>
<td>Mandatory if definition type is RDBMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>url</td>
<td>The connection URL to pass to the JDBC driver to establish the connection.</td>
<td>URL</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>The connection user name to pass to the JDBC driver to establish the connection.</td>
<td>String</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>The connection password to pass to the JDBC driver to establish the connection.</td>
<td>String</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the JDBC driver to use.</td>
<td>Fully qualified Java class</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td><code>&lt;maxActive&gt;</code></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>
<td></td>
</tr>
<tr>
<td><code>&lt;maxWait&gt;</code></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>
<td></td>
</tr>
<tr>
<td><code>&lt;testOnBorrow&gt;</code></td>
<td>Specifies whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. When set to true, the <code>validationQuery</code> parameter must be set to a non-null string.</td>
<td>Boolean</td>
<td>false</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td><code>&lt;validationQuery&gt;</code></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 <code>SELECT 1(mysql)</code>, <code>select 1 from dual(oracle)</code>, <code>SELECT 1(MS Sql Server)</code>.</td>
<td>String</td>
<td>null</td>
<td>Mandatory when <code>testOnBorrow</code> is set to true</td>
<td></td>
</tr>
<tr>
<td><code>&lt;validationInterval&gt;</code></td>
<td>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).</td>
<td>Long</td>
<td>30000 (30 seconds)</td>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>

Example
<datasources-configuration
xmlns:svns="http://org.wso2.securevault/configuration">
  <providers>
    <provider>
      org.wso2.carbon.ndatasource.rdbms.RDBMSDataSourceReader
    </provider>
  </providers>
  <datasources>
    <datasource>
      <name>WSO2_CARBON_DB</name>
      <description>The datasource used for registry and user
manager</description>
      <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
      </jndiConfig>
      <definition type="RDBMS">
        <configuration>
          <url>
            jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000
          </url>
          <username>wso2carbon</username>
          <password>wso2carbon</password>
          <driverClassName>org.h2.Driver</driverClassName>
          <maxActive>50</maxActive>
          <maxWait>60000</maxWait>
          <testOnBorrow>true</testOnBorrow>
          <validationQuery>SELECT 1</validationQuery>
          <validationInterval>30000</validationInterval>
        </configuration>
      </definition>
    </datasource>
  </datasources>
</datasources-configuration>

Configuring registry.xml

Users can change the default configurations by editing the <PRODUCT_HOME>/repository/conf/registry.xml file using the information given below. Click on the table and use the left and right arrow keys to scroll horizontally.

**XML Elements**

<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>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wso2registry&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mandatory</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2015-2019
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cacheConfig&gt;</code></td>
<td>When an application requests to use a particular resource stored in the registry or creates/modifies a resource in the registry, the Carbon server will cache the resource for subsequent use. The <code>&lt;cacheConfig&gt;</code> element in this file is used to configure the time period for which a resource can be cached.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;lastAccessedExpirationMillis&gt;</code></td>
<td>Specifies the time period for which a resource in the registry will be cached after it is read by an application.</td>
<td>15</td>
<td>Optional</td>
</tr>
<tr>
<td><code>&lt;lastModifiedExpirationMillis&gt;</code></td>
<td>Specifies the time period for which a resource in the registry will be cached after it is created or modified.</td>
<td>15</td>
<td>Optional</td>
</tr>
<tr>
<td><code>&lt;currentDBConfig&gt;</code></td>
<td>The server can only handle one active configuration at a time. The currentDBConfig parameter defined in the registry.xml is used to specify the database configuration that is active at present. The value of the currentDBConfig parameter should be a valid name of a database configuration defined on the registry.xml file.</td>
<td>String</td>
<td>wso2registry</td>
</tr>
<tr>
<td><code>&lt;readOnly&gt;</code></td>
<td>To run the registry in read-only mode, set the readOnly element to true. Setting the read-only mode allows you to run an immutable instance of registry repository. This setting is valid on a global level.</td>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
<th>Mandatory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;enableCache&gt;</td>
<td>To enable registry caching, set the enableCache element to true. Once caching is enabled, repetitive read operations will be executed against the cache instead of the database. This setting is valid on a global level.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
<td>&lt;enableCache&gt;true&lt;/enableCache&gt;</td>
</tr>
<tr>
<td>&lt;registryRoot&gt;</td>
<td>The registryRoot parameter can be used to define the apparent root of the running instance of the server. This setting is valid on a global level.</td>
<td>String</td>
<td>/</td>
<td>Mandatory</td>
<td>&lt;registryRoot&gt;/&lt;/registryRoot&gt;</td>
</tr>
<tr>
<td>&lt;dbConfig&gt;</td>
<td></td>
<td>String</td>
<td></td>
<td>Mandatory</td>
<td>&lt;dbConfig name=&quot;wso2registry&quot;&gt;</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>String</td>
<td>wso2registry</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;validationQuery&gt;</code></td>
<td>N/A</td>
<td>You can configure this parameter under the <code>&lt;dbConfig&gt;</code> element of the <code>&lt;PRODUCT_HOME&gt;/repository/conf/registry.xml</code> file. When you are maintaining DB connections, it is always recommended to use a validation query to check the health of the TCP connection of the connections that stay in the DB connection pool. Since opening connections is an expensive and time consuming operation, after a connection is created, it will be kept open for a specific time in the pool. When re-using these connections from the pool, there can be situations that the TCP connection to the DB is interrupted and the connection consumer gets errors such as communication link failures etc. To avoid that you can use a validation query always as a best practice. It is an SQL statement specific to the DBMS type, which runs before using the connection.</td>
<td>String</td>
<td>N/A</td>
<td>Optional</td>
</tr>
<tr>
<td>Tag</td>
<td>Type</td>
<td>Description</td>
<td>Default</td>
<td>Status</td>
<td>Example</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;testOnBorrow&gt;</td>
<td>N/A</td>
<td>This validates the connection objects before borrowing them from the pool. If the object fails to validate, it will be dropped from the pool, and another will be attempted to borrow. This property can be added to the <code>&lt;dbConfig&gt;</code> element in the <code>&lt;PRODUCT_HOME&gt;/repository/conf/registry.xml</code> file.</td>
<td>Boolean</td>
<td>true</td>
<td>Optional &lt;/testOnBorrow&gt;true&lt;/testOnBorrow&gt;</td>
</tr>
<tr>
<td>&lt;dataSource&gt;</td>
<td>String</td>
<td>jdbc/WSO2CarbonDB</td>
<td>Mandatory</td>
<td></td>
<td>&lt;dataSource&gt;jdbc/WSO2CarbonDB&lt;/dataSource&gt;</td>
</tr>
<tr>
<td>&lt;handler&gt;</td>
<td>String</td>
<td>Handlers are pluggable components, that contain custom processing logic for handling resources. All handlers extend an abstract class named Handler, which provides default implementations for resource handling methods as well as a few utilities useful for concrete handler implementations. Handler implementations can provide alternative behaviors for basic resource related operations, by overwriting one or more methods in the Handler class.</td>
<td>Optional</td>
<td></td>
<td>&lt;handler class=&quot;org.wso2.carbon.registry.extensions.handlers.SynapseRepositoryHandler&quot;&gt;</td>
</tr>
<tr>
<td>class</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td>&lt;filter class=&quot;org.wso2.carbon.registry.core.jdbc.handlers.filters.MediaTypeMatcher&quot;&gt;</td>
</tr>
<tr>
<td>&lt;filter&gt;</td>
<td>String</td>
<td></td>
<td>Optional</td>
<td></td>
<td>&lt;filter class=&quot;org.wso2.carbon.registry.core.jdbc.handlers.filters.MediaTypeMatcher&quot;&gt;</td>
</tr>
</tbody>
</table>
**<remoteInstance>**

In order to mount an external registry, you have to define the remote instance. This could use either the JDBC-based configuration, the Atom-based configuration model or the WebService-based configuration model.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the remote instance.</td>
<td>String</td>
<td>Optional</td>
<td>remoteInstance url=&quot;<a href="https://localhost:9443/registry">https://localhost:9443/registry</a>&quot;</td>
</tr>
<tr>
<td>&lt;ID&gt;</td>
<td>Remote instance ID.</td>
<td>String</td>
<td>Optional</td>
<td>&lt;id&gt;instanceid&lt;/id&gt;</td>
</tr>
<tr>
<td>&lt;username&gt;</td>
<td>Username of the remote registry login.</td>
<td>String</td>
<td>Optional</td>
<td>&lt;username&gt;username&lt;/username&gt;</td>
</tr>
<tr>
<td>&lt;password&gt;</td>
<td>Password of the remote registry login.</td>
<td>String</td>
<td>Optional</td>
<td>&lt;password&gt;password&lt;/password&gt;</td>
</tr>
<tr>
<td>&lt;dbConfig&gt;</td>
<td>The database configuration to use.</td>
<td>String</td>
<td>Optional</td>
<td>wso2registry</td>
</tr>
<tr>
<td>&lt;readOnly&gt;</td>
<td>To run the registry in read-only mode set the readOnly element to true.</td>
<td>String</td>
<td>Optional</td>
<td>&lt;readOnly&gt;false&lt;/readOnly&gt;</td>
</tr>
<tr>
<td>&lt;enableCache&gt;</td>
<td>To enable registry caching, set the enableCache element to true. Once caching is enabled, repetitive read operations will be executed against the cache instead of the database. This setting is valid only for the specific remote instance.</td>
<td>String</td>
<td>Optional</td>
<td>&lt;enableCache&gt;true&lt;/enableCache&gt;</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td>Default</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>--------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;registryRoot&gt;</td>
<td>The registryRoot parameter can be used to define whether the apparent root of the running instance of the server. This setting is valid only for the specific remote instance.</td>
<td>String</td>
<td>/</td>
<td>Optional &lt;registryRoot/&gt;&lt;/registryRoot&gt;</td>
</tr>
<tr>
<td>&lt;mount&gt;</td>
<td>Once a remote instance has been defined, a collection on the remote registry can be mounted to the local instance.</td>
<td>Mandatory</td>
<td></td>
<td>&lt;mount path=&quot;/system/config&quot; overwrite=&quot;true</td>
</tr>
<tr>
<td>path</td>
<td>The path to which the mount will be added to.</td>
<td>String</td>
<td>/_system/config</td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Whether an existing collection at the given path would be overwritten or not.</td>
<td></td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>&lt;instanceID&gt;</td>
<td>Remote instance ID.</td>
<td>instanceid</td>
<td>Mandatory</td>
<td>&lt;instanceId&gt;:instanceid&lt;/instanceId&gt;</td>
</tr>
<tr>
<td>&lt;targetPath&gt;</td>
<td>The path on the remote registry.</td>
<td>String</td>
<td>/_system/nodes</td>
<td>Mandatory &lt;targetPath&gt;/_system/nodes&lt;/targetPath&gt;</td>
</tr>
<tr>
<td>&lt;versionResourcesOnChange&gt;</td>
<td>You can configure whether you want to auto-version the resources (non-collection) by setting versionResourcesOnChange element to true. In this configuration it will create a version for the resources whenever it is updated.</td>
<td>Boolean</td>
<td>false</td>
<td>Mandatory &lt;versionResourcesOnChange&gt;false&lt;/versionResourcesOnChange&gt;</td>
</tr>
<tr>
<td>&lt;staticConfiguration&gt;</td>
<td>You can configure whether you want to auto-version the resources (non-collection) by setting versionResourcesOnChange element to true. In this configuration it will create a version for the resources whenever it is updated.</td>
<td>Mandatory</td>
<td></td>
<td>&lt;staticConfiguration&gt;&lt;versioningProperties&gt;true&lt;/versioningProperties&gt;&lt;versioningComments&gt;true&lt;/versioningComments&gt;&lt;versioningTags&gt;true&lt;/versioningTags&gt;&lt;versioningRatings&gt;true&lt;/versioningRatings&gt;&lt;/staticConfiguration&gt;</td>
</tr>
</tbody>
</table>
While most configuration options can be changed after the first run of the WSO2 Governance Registry, changing the Static Configuration (configuration details under the staticConfiguration parameter), will not be fully effective. If you need to change any Static Configuration and expect it to take effect, you will have to erase the contents of the database, and restart the server passing the -Dsetup system property which will re-generate the database.

**Deprecation of -DSsetup**

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 -D Setup option is redundant and typically unusable. As a result, from January 2018 onwards 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.
You are supposed to change the static configuration section only before loading any data to the registry (That is before the first start-up).

<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>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;versioningProperties&gt;</td>
<td></td>
<td>Whether the properties are versioned when a snapshot is created.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>&lt;versioningComments&gt;</td>
<td></td>
<td>Whether the comments are versioned when a snapshot is created.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>&lt;versioningTags&gt;</td>
<td></td>
<td>Whether the tags are versioned when a snapshot is created.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>&lt;versioningRatings&gt;</td>
<td></td>
<td>Whether the ratings are versioned when a snapshot is created.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
<td></td>
</tr>
</tbody>
</table>

**Configuring user-mgt.xml**

Users can change the default user management functionality related configurations by editing the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file using the information given below.

Click on the table and use the left and right arrow keys to scroll horizontally.

**XML Elements**

<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>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;UserManager&gt;</td>
<td></td>
<td>User kernel configuration for Carbon server.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Realm&gt;</td>
<td></td>
<td>Realm configuration.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
<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/WSO2CarbonDB</Property>

</Configuration>

<AddAdmin> Specifies whether the admin user and admin role will be created in the primary user store. This element enables the user to create additional admin users in the user store. If the <AdminUser> element does not exist in the external user store, it will be automatically created only if this property is set to true. If the value is set to false, the given admin user and role should already exist in the external user store. Boolean true Mandatory
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;AdminRole&gt;</td>
<td>The role name that is used as an admin role for the Carbon server.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;AdminUser&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;UserName&gt;</td>
<td>User name that is used to represent an admin user for the Carbon server.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;Password&gt;</td>
<td>Password of the admin user. If the admin user needs to be created in the Carbon server.</td>
<td>String</td>
<td>N/A</td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;EveryOneRoleName&gt;</td>
<td>By default, every user in the user store is assigned to this role.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;Property&gt;</td>
<td>User realm configuration specific property values.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
### <UserStoreManager>

User Store manager implementation classes and their configurations for use realm.

- Use the `ReadOnlyLDAPUserStoreManager` to do read-only operations for external LDAP user stores.

- To do both read and write operations, use the `ReadWriteLDAPUserStoreManager` for external LDAP user stores.

- If you wish to use an Active Directory Domain Service (AD DS) or Active Directory Lightweight Directory Service (AD LDS), use the `ActiveDirectoryUserStoreManager`.

- Use `JDBCUserStoreManager` for both internal and external JDBC user stores.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadOnley</td>
<td>false</td>
</tr>
</tbody>
</table>

### <AuthorizationManager>

Authorization manager implementation class and its configuration for user realm.

- Use `JDBCAuthorizationManager` for both internal and external JDBC user stores.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthorizationCacheEnable</td>
<td>true</td>
</tr>
</tbody>
</table>

---

**Class:**

- `<UserStoreManager>`
- `<AuthorizationManager>`
Configuring config-validation.xml

The `<PRODUCT_HOME>/repository/conf/etc/config-validation.xml` file contains the recommended system configurations for a server. When you start the server, the system configurations will be validated against these recommendations, and warnings will be published if conflicts are found.

Given below are the default recommendations in the config-validation.xml file. If required, you may change some of these recommendations on this file according to the conditions in your production environment.

**System Validation**

Following are the system parameter values recommended for the purpose of running a WSO2 product server.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Description</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Required processor speed.</td>
<td>800</td>
</tr>
<tr>
<td>RAM</td>
<td>Required RAM in your environment.</td>
<td>2048</td>
</tr>
<tr>
<td>swap</td>
<td>Required space in hard disk to use for virtual memory.</td>
<td>2048</td>
</tr>
<tr>
<td>freeDisk</td>
<td>Free disk space required in your environment.</td>
<td>1024</td>
</tr>
</tbody>
</table>
| ulimit     | The limit of resources per user. This value indicates the limit on the number of file descriptors a process may have. This property is specified in the product startup script as shown below. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`.

```bash
#ulimit -n 100000
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Description</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initHeapSize</td>
<td>The initial heap size that applies if the JVM requires more memory than is allocated by default.</td>
<td>256</td>
</tr>
<tr>
<td>maxHeapSize</td>
<td>The maximum heap size that applies if the JVM requires more memory than is allocated by default.</td>
<td>512</td>
</tr>
<tr>
<td>maxPermGenSize</td>
<td>The maximum heap size of the permanent generation of heap.</td>
<td>256</td>
</tr>
</tbody>
</table>

These parameters are specified in the product startup script as shown below, where, "-Xms", "-Xmx" and "-XX*" correspond to "initHeapSize", "maxHeapSize" and "maxPermGenSize" respectively. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`.  

If the values set for these parameters in your environment are less than the recommendations, the following warnings will be published when you start your server.

- WARN - ValidationResultPrinter CPU speed (MHz): `<systemCPU>` of the system is below the recommended minimum speed: `<recommended value>`
- WARN - ValidationResultPrinter RAM size (MB): `<systemRam>` of the system is below the recommended minimum size: `<recommended value>`
- WARN - ValidationResultPrinter Swap Memory size (MB): `<systemSwap>` of the system is below the recommended minimum size `<recommended value>`
- WARN - ValidationResultPrinter Maximum free Disk Space (MB): `<systemDisk>` of the system is below the recommended minimum size: `<recommended value>`
- WARN - ValidationResultPrinter Open files limit: `<openFileLimit>` of the system is below the recommended minimum count: `<recommended value>`

**JVM Validation**

The following JVM heap size values are recommended by default in the config-validation.xml file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initHeapSize</td>
<td>The initial heap size that applies if the JVM requires more memory than is allocated by default.</td>
<td>256</td>
</tr>
<tr>
<td>maxHeapSize</td>
<td>The maximum heap size that applies if the JVM requires more memory than is allocated by default.</td>
<td>512</td>
</tr>
<tr>
<td>maxPermGenSize</td>
<td>The maximum heap size of the permanent generation of heap.</td>
<td>256</td>
</tr>
</tbody>
</table>

These parameters are specified in the product startup script as shown below, where, "-Xms", "-Xmx" and "-XX*" correspond to "initHeapSize", "maxHeapSize" and "maxPermGenSize" respectively. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`.  

If the values set for these parameters in your environment are less than the recommendations, the following warnings will be published when you start your server.

- WARN - ValidationResultPrinter CPU speed (MHz): `<systemCPU>` of the system is below the recommended minimum speed: `<recommended value>`
- WARN - ValidationResultPrinter RAM size (MB): `<systemRam>` of the system is below the recommended minimum size: `<recommended value>`
- WARN - ValidationResultPrinter Swap Memory size (MB): `<systemSwap>` of the system is below the recommended minimum size `<recommended value>`
- WARN - ValidationResultPrinter Maximum free Disk Space (MB): `<systemDisk>` of the system is below the recommended minimum size: `<recommended value>`
- WARN - ValidationResultPrinter Open files limit: `<openFileLimit>` of the system is below the recommended minimum count: `<recommended value>`
If these heap size values in your product startup script are less than the recommended values, the following warnings will be published when you start your server:

- WARN - ValidationResultPrinter Initial Heap Memory (MB) : <system value> of the running JVM is set below the recommended minimum size :<recommended value>
- WARN - ValidationResultPrinter Maximum Heap Memory (MB) : <system value> of the running JVM is set below the recommended minimum size :<recommended value>
- WARN - ValidationResultPrinter Maximum PermGen space (MB) :<system value> of the running JVM is set below the recommended minimum size :<recommended value>

**System Property Validation**

According to the config-validation.xml file, values are required to be specified for the following properties in your system. Note that it is not recommended to remove this validations as these are mandatory settings.

- carbon.home
- carbon.config.dir.path
- axis2.home

The carbon.home and carbon.config.dir.path properties are given in the product startup script as shown below. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`.

```
$JAVA_OPTS
-Dcarbon.home=<$CARBON_HOME>
-Dcarbon.config.dir.path=<$CARBON_HOME/repository/conf>
```

The axis2.home property is given in the product startup script as shown below. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`.

```
# Set AXIS2_HOME. Needed for One Click JAR Download
AXIS2_HOME=<$CARBON_HOME>
```

If the values for these properties are null in the product startup script, the following warning message will be published when you start the server: "Value is not set for the required system property : <property-value>".

**Supported OS Validation**

The product has been tested for compatibility with the following operating systems, which are listed in the config-validation.xml file. Therefore, by default, the system is validated against these operating systems.

- Linux
- Unix
- Mac OS
- Windows Server 2003
- Windows XP
- Windows Vista
- Windows 7
- Mac OS X
- Windows Server 2008
- Windows Server 2008 R2
- AIX

If the OS in your environment is not included in this list, the following warning message will be published when you start the server: "WARN - ValidationResultPrinter The running OS : <value> is not a tested Operating System for running WSO2 Carbon."
This page takes you through the general steps for upgrading product versions based on Carbon 4.4.6 to Carbon 4.4.7.

Preparing to Upgrade

The following are the specific prerequisites you must complete before an upgrade:

- Before you upgrade to the latest version of a product, you create a staging database, which is essentially an empty database. Note that you should NOT connect a new product version to an older database that has not been upgraded.
- Make a backup of the database and the <PRODUCT_HOME> directory prior to upgrading. The <PRODUCT_HOME> directory can simply be copied to the new directory.
- Stop all the Carbon servers connected to the database before running the migration scripts.

Note that the upgrade should be done during a period when there is low traffic on the system.

Limitations

- This upgrade can only be done if the database type is the same. For example, if you are using MySQL currently and you need to migrate to Oracle in the new version, these scripts will not work.
- You cannot roll back an upgrade. It is impossible to restore a backup of the previous server and retry the upgrade process.

Downtime

The downtime is limited to the time taken for switching databases when the staging database is promoted to the actual production status.

Upgrading the configurations

There are no database changes between Carbon 4.4.6 to Carbon 4.4.7. Therefore, only the new configuration options in Carbon 4.4.7 should be updated for the new environment as explained below.

1. Copy the data from the old database to the staging database you created. This becomes the new database for your new version of Carbon.
2. Download Carbon 4.4.7 and connect it to your staging database.
3. Update the configuration files in Carbon 4.4.7 as required.
4. Copy the following directories from the old database to the staging database.
   a. To migrate the super tenant settings, copy the <PRODUCT_HOME>/repository/deployment/server directory.
   b. If multitenancy is used, copy the <PRODUCT_HOME>/repository/tenants directory.

Note that configurations should not be copied directly between servers.

5. Start the server.

Going into production

The following are recommended tests to run on the staging system.

- Create multiple user stores and try adding users to different user stores.
- Create multiple tenants and add different user stores to the different tenants. Thereafter, add users to the various user stores.

Once the above tests are run successfully, it is safe to consider that the upgrade is ready for production. However, it is advised to test any features that are being used in production.

Product Startup Options

Given below are the options that are available when starting a WSO2 product. The product startup scripts are stored in the <PRODUCT_HOME>/bin/ directory. When you execute the startup script, you can pass a system property by appending it next to the start-up script as shown below.

```
sh wso2server.sh -<startup option>
```

For example:
Listed below are some general options that can be used for starting the server.

<table>
<thead>
<tr>
<th>Startup Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-start</td>
<td>Starts the Carbon server using &quot;nohup&quot; in the background. This option is not available for Windows.</td>
</tr>
<tr>
<td>-stop</td>
<td>Stops the Carbon server process. This option is not available for Windows.</td>
</tr>
<tr>
<td>-restart</td>
<td>Restarts the Carbon server process. This option is not available for windows.</td>
</tr>
<tr>
<td>-cleanRegistry</td>
<td>Cleans the registry space. <strong>Caution:</strong> All registry data will be lost.</td>
</tr>
<tr>
<td>-debug &lt;port&gt;</td>
<td>Starts the server in remote debugging mode. The remote debugging port should be specified.</td>
</tr>
<tr>
<td>-version</td>
<td>Shows the version of the product that you are running.</td>
</tr>
<tr>
<td>-help</td>
<td>Lists all the available commands and system properties.</td>
</tr>
</tbody>
</table>

Listed below are some system properties that can be used when starting the server.

<table>
<thead>
<tr>
<th>Startup Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-D osgiConsole=&lt;port&gt;</td>
<td>Starts the Carbon server with the Equinox OSGi console. If the optional 'port' parameter is provided, a telnet port will be opened.</td>
</tr>
<tr>
<td>-D osgiDebugOptions=&lt;options-file&gt;</td>
<td>Starts the Carbon server with OSGi debugging enabled. Debug options are loaded from the <code>&lt;PRODUCT_HOME&gt;/repository/conf/etc/osgi-debug.options</code>.</td>
</tr>
<tr>
<td>-D setup</td>
<td>Cleans the registry and other configurations, recreates DB, re-populates the configuration and starts the server. <strong>Note:</strong> It is not recommended to use this option in a production environment. Instead, you can manually run the DB scripts directly in the database.</td>
</tr>
<tr>
<td>-D workerNode</td>
<td>Starts the product as a worker node, which means the front-end features of your product will not be enabled.</td>
</tr>
<tr>
<td>-D serverRoles=&lt;roles&gt;</td>
<td>A comma separated list of roles used in deploying Carbon applications.</td>
</tr>
<tr>
<td>-D profile=&lt;profileName&gt;</td>
<td>Starts the server with the specified profile, e.g., worker profile.</td>
</tr>
<tr>
<td>-D tenant.idle.time=&lt;time&gt;</td>
<td>If a tenant is idle for the specified time, the tenant will be unloaded. The default tenant idle time is 30 minutes. This is required in clustered setups, which has master and worker nodes.</td>
</tr>
</tbody>
</table>

**Supported Cipher Suites**

Given below are the cipher suites that are functional in Tomcat (Tomcat version 7.0.59 with the JSSE providers 7/8) for the following SSL protocols: TLSv1, TLSv1.1 and TLSv1.2. See [Configuring Transport-Level Security](#) for instructions on how to enable the required ciphers and to disable the weak ciphers for your WSO2 server.

See the following topics:

- Cipher suites supported by Tomcat 7.0.59 and Oracle JDK 1.8
- Cipher suites supported by Tomcat 7.0.59 and Oracle JDK 1.7
Weak ciphers

Cipher suites supported by Tomcat 7.0.59 and Oracle JDK 1.8

The following cipher suites are supported by Tomcat version 7.0.59 and Oracle JDK 1.8:

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

The following additional cipher suites will be supported if JCE Unlimited Strength Jurisdiction Policy is used with Tomcat 7.0.59 and Oracle JDK 1.8:

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

Cipher suites supported by Tomcat 7.0.59 and Oracle JDK 1.7

The following cipher suites are supported by Tomcat version 7.0.59 and Oracle JDK 1.7:

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA

The following additional cipher suites will be supported if JCE Unlimited Strength Jurisdiction Policy is used with Tomcat version 7.0.59 and Oracle JDK 1.7:

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA

Weak ciphers

Listed below are the relatively weaker cipher suites (which use DES/3DES, RC4 and MD5). It is not recommended to use these cipher suites for the following reasons:

- DES/3DES are deprecated and should not be used.
- MD5 should not be used due to known collision attacks.
- RC4 should not be used due to crypto-analytical attacks.
- DSS is limited to 1024 bit key size.
- Cipher-suites that do not provide Perfect Forward Secrecy/ Forward Secrecy (PFS/FS).

The following cipher suites are weak for Tomcat version 7.0.59 when either JDK version (7/8) is used. The same applies if JCE Unlimited Strength Jurisdiction Policy is used.

- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA

The following cipher suites are weak for Tomcat version 7.0.59 and JDK version 1.7:
Directory Structure of WSO2 Products

All WSO2 products are built on top of the Carbon platform. The directory structure described below is the structure that is inherited by all Carbon-based WSO2 products. However, note that each product may contain folders and files that are specific to the product, in addition to what is described below.

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
<th>General Folder Path</th>
<th>Folder Path for Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Contains various scripts (.sh &amp; .bat scripts).</td>
<td>&lt;PRODUCT_HOME&gt;/bin/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/bin/</td>
</tr>
<tr>
<td>database</td>
<td>Contains the databases.</td>
<td>&lt;PRODUCT_HOME&gt;/repository/database/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/repository/database/</td>
</tr>
<tr>
<td>dbscripts</td>
<td>Contains all the database scripts.</td>
<td>&lt;PRODUCT_HOME&gt;/dbscripts/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/dbscripts/</td>
</tr>
<tr>
<td>lib</td>
<td>Contains the basic set of libraries required for starting a WSO2 product in standalone mode.</td>
<td>&lt;PRODUCT_HOME&gt;/lib/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/lib/</td>
</tr>
<tr>
<td>repository</td>
<td>The repository where services and modules deployed in a WSO2 product are stored. In addition to this, the repository/components directory contains the Carbon runtime and JAR files added by users (such as third party libraries).</td>
<td>&lt;PRODUCT_HOME&gt;/repository/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/repository/</td>
</tr>
<tr>
<td>conf</td>
<td>Contains configuration files.</td>
<td>&lt;PRODUCT_HOME&gt;/repository/conf/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/conf/</td>
</tr>
<tr>
<td>components</td>
<td>Contains different components (OSGI bundles, features etc.) that are related to the product.</td>
<td>&lt;PRODUCT_HOME&gt;/repository/components/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/components/</td>
</tr>
<tr>
<td>plugins</td>
<td>Contains plugins that are related to the product.</td>
<td>&lt;PRODUCT_HOME&gt;/repository/components/plugins/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/components/plugins/</td>
</tr>
<tr>
<td>patches</td>
<td>Contains patches that are issues with the product.</td>
<td>&lt;PRODUCT_HOME&gt;/patches/</td>
<td>&lt;PRODUCT_HOME&gt;/patches/</td>
</tr>
<tr>
<td>logs</td>
<td>Contains all log files created during execution.</td>
<td>&lt;PRODUCT_HOME&gt;/repository/logs/</td>
<td>&lt;PRODUCT_HOME&gt;/repository/logs/</td>
</tr>
<tr>
<td>resources</td>
<td>Contains additional resources that may be required, including sample configurations and sample resources.</td>
<td>&lt;PRODUCT_HOME&gt;/resources/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/resources/</td>
</tr>
<tr>
<td>samples</td>
<td>Contains sample services and client applications to demonstrate the functionality and capabilities of WSO2 products.</td>
<td>&lt;PRODUCT_HOME&gt;/samples/</td>
<td>&lt;PRODUCT_HOME&gt;/samples/&lt;PROFILE_HOME&gt;/</td>
</tr>
<tr>
<td>tmp</td>
<td>Used for storing temporary files, and is pointed to by the java.io.tmpdir system property.</td>
<td>&lt;PRODUCT_HOME&gt;/tmp/</td>
<td>&lt;PRODUCT_HOME&gt;/wso2/&lt;PROFILE_HOME&gt;/tmp/</td>
</tr>
<tr>
<td>LICENSE.txt</td>
<td>Apache License 2.0 and other relevant licenses under which the WSO2 product is distributed.</td>
<td>&lt;PRODUCT_HOME&gt;/LICENSE.txt</td>
<td>N/A</td>
</tr>
<tr>
<td>README.txt</td>
<td>This document.</td>
<td>&lt;PRODUCT_HOME&gt;/README.txt</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<Product_HOME> refers to the root folder of the WSO2 product distribution. <PROFILE_HOME> refers to the root directory of other profiles that are shipped as separate runtimes with a product.
WSO2 Patch Application Process

WSO2 has introduced the WSO2 Update Manger (WUM), which is a command-line tool that allows you to update your product with the latest available patches and enhancements. All WSO2 products based on Carbon 4.4.x will soon be supported by WUM. Go to the WUM website to see if your product version is currently supported. You can follow the instructions in Updating your WSO2 product to get the patch updates using WUM. Note that WUM does not allow you to pick and choose the patches that you apply. Therefore, if you want to apply a selected patch to your product, you must use the following patch application process.

The patch application process described below guides you on how to manually apply patches, such as security patches, to Carbon 4.4.x-based products (if your product version is currently not supported by WUM).

* What is a patch?
* Applying patches to the product
* Verifying the patch application
* Removing patches

What is a patch?

The following diagram depicts the contents of a patch archive that is provided by WSO2. The patch archive name indicates the WSO2 Carbon Kernel version and the patch number. In this example, the Kernel version is 4.4.0 and the patch number is 1341. Inside the patch archive there is a README.txt file that includes the guidelines on how to apply the patch. Some patches (as in the example given below) might have a resources directory that contain artifacts, such as web apps, library files, configurations, scripts and more.

**Example:**

```markdown
WSO2-CARBON-PATCH-4.4.0-1341
LICENSE.txt
NOT_A_CONTRIBUTION.txt
patch1341
  org.wso2.carbon.device.mgt.common_1.1.2.SNAPSHOT.jar
  org.wso2.carbon.device.mgt.core_1.1.2.SNAPSHOT.jar
README.txt
resources
webapps
  api#device-mgt#v1.0.war
```

Applying patches to the product

Note the following before you begin:

* It is mandatory to follow the steps specified in the README.txt of the patch before applying the patch.
* If the README.txt provides instructions to replace existing resources in the WSO2 server with any artifact in the resources directory of the patch archive, it is highly recommended to backup the existing resources. The original files might be required if you are reverting the patch later.
* As a precaution, make a backup of the server.

Follow the steps given below to apply patches to your server.

1. Shut down the server.
2. Copy the patches (patchxxx directory in the patch archive) to the `<PRODUCT_HOME>/repository/components/patches` directory. **Note** that you may sometimes need to apply the patches one by one in order to avoid conflicts. Check the README.txt files carefully for these instructions.
3. If the resources directory in the patch archive contains artifacts, copy them to the appropriate location in your server as instructed in the README.txt file. Note that this will replace the already existing artifacts.
4. Start the WSO2 server. The patches will then be incrementally applied to the `<PRODUCT_HOME>/repository/components/plugins/` directory.

When you start the server, the process first creates a backup folder named `patch0000` inside the `<PRODUCT_HOME>/repository/components/patches/` directory, which will contain the original content of the `<PRODUCT_HOME>/repository/components/plugins/` directory. This step enables you to revert back to the previous state if something goes wrong during operations.

Prior to Carbon 4.2.0, users were expected to apply patches by starting the server with `wso2server.sh -DapplyPatches`. Now, you do not have to issue a special command to trigger the patch application process. It starts automatically if there are changes in the `<PRODUCT_HOME>/repository/components/patches/` directory. It verifies all the latest JARs in the `patches` directory against the JARs in the `plugins` directory by comparing the MD5s of JARs.

Verifying the patch application

After the patch application process is completed, the patch verification process ensures that the latest patches are correctly applied to the `<PRODUCT_HOME>/repository/components/plugins/` folder.

1. Compare the md5sum of each component in the `patchXXXX` directory against the same component in the `<PRODUCT_HOME>/repository/components/plugins/` directory.
2. The artifacts (from the resources directory) of the latest patch version that were copied to various locations of the server must contain the same md5sum as the artifact in the resources directory of the patch.
3. You can use the following resources to track the patches:
   - All patch-related logs are recorded in the `<PRODUCT_HOME>/repository/logs/patches.log` file.
   - The `<PRODUCT_HOME>/repository/components/patches/.metadata/prePatchedJARs.txt` meta file contains the list of patched JARs and the md5 values.
   - The patch directory information of all the applied patched will be in the `<PRODUCT_HOME>/repository/components/default/configuration/prePatchedDir.txt` file.

Do not change the data in the `<PRODUCT_HOME>/repository/components/default/configuration/prePatchedDir.txt` file. The patch application process gets the pre-patched list from this file and compares the list with the patches available in the `patches` directory. If you change the data in this file, you will get a startup error when applying patches.

Removing patches

Patches installed in your WSO2 product using the above steps can also be removed when required. However, this needs to be done with caution as explained below.

**Before you begin** removing an installed patch:
- Shut down the server.
- Make a backup as a precaution.
- Read the `README.txt` file that is included in the patch ZIP. This file will specify if there are other patches that depend on the patch you are going to remove. You must also identify if there are manual steps to roll back.

To remove a patch:

1. Remove the patch from the `<PRODUCT_HOME>/repository/components/patches` directory.
2. If there were artifacts copied from the resources directory of the patch archive, you need to replace them with the original artifacts (from the backup that was created before applying the patch). Also, if there were exploded artifacts (such as `.war` files in the `<PRODUCT_HOME>/repository/deployment/server/webapps` directory), be sure to remove them as well.
3. Restart the server. The remaining patches will now be reinstalled.

The `patches.log` file in the `<PRODUCT_HOME>/repository/logs` directory indicates the patches that are applied to the server.