



# WSO2 Administration Guide

## Documentation

### Carbon 4.4.x Platform

# Table of Contents

1. WSO2 Administration Guide .....	6
1.1 Deployment and Clustering .....	6
1.1.1 Key Concepts .....	7
1.1.1.1 Clustering Overview .....	7
1.1.1.2 Load Balancing .....	9
1.1.1.3 Separating the Worker and Manager Nodes .....	11
1.1.1.4 Sharing Databases in a Cluster .....	14
1.1.1.5 Sticky Sessions with Manager Nodes .....	16
1.1.2 Production Deployment Guidelines .....	16
1.1.2.1 Troubleshooting in Production Environments .....	29
1.2 Data Stores .....	33
1.2.1 Working with Databases .....	34
1.2.1.1 Setting up the Physical Database .....	34
1.2.1.1.1 Setting up IBM DB2 .....	35
1.2.1.1.2 Setting up IBM Informix .....	38
1.2.1.1.3 Setting up H2 .....	40
1.2.1.1.4 Setting up MariaDB .....	42
1.2.1.1.5 Setting up Microsoft SQL .....	43
1.2.1.1.6 Setting up MySQL .....	44
1.2.1.1.7 Setting up a MySQL Cluster .....	46
1.2.1.1.8 Setting up Oracle .....	46
1.2.1.1.9 Setting up Oracle RAC .....	47
1.2.1.1.10 Setting up PostgreSQL .....	49
1.2.1.1.11 Setting up Derby .....	50
1.2.1.2 Changing the Carbon Database .....	51
1.2.1.2.1 Changing to Embedded Derby .....	52
1.2.1.2.2 Changing to Embedded H2 .....	55
1.2.1.2.3 Changing to IBM DB2 .....	59
1.2.1.2.4 Changing to IBM Informix .....	62
1.2.1.2.5 Changing to MariaDB .....	65
1.2.1.2.6 Changing to MSSQL .....	68
1.2.1.2.7 Changing to MySQL .....	71
1.2.1.2.8 Changing to Oracle .....	74
1.2.1.2.9 Changing to Oracle RAC .....	77
1.2.1.2.10 Changing to PostgreSQL .....	80
1.2.1.2.11 Changing to Remote H2 .....	83
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 .....	90
1.2.2.1.3 Entries and Content .....	91
1.2.2.1.4 Role Permissions .....	102
1.2.2.2 Searching the Registry .....	103
1.2.2.3 Using Remote Registry Instances for the Registry Partitions .....	105
1.2.2.3.1 All Partitions in a Single Server .....	105
1.2.2.3.2 Config and Governance Partitions in a Remote Registry .....	106
1.2.2.3.3 Governance Partition in a Remote Registry .....	111

1.2.2.3.4 Config and Governance Partitions in Separate Nodes .....	116
1.2.3 Working with Users, Roles and Permissions .....	126
1.2.3.1 Introduction to User Management .....	127
1.2.3.2 Configuring the User Realm .....	130
1.2.3.2.1 Configuring the System Administrator .....	131
1.2.3.2.2 Configuring the Authorization Manager .....	133
1.2.3.2.3 Configuring User Stores .....	135
1.3 Security .....	178
1.3.1 Maintaining Logins and Passwords .....	179
1.3.2 Securing Passwords in Configuration Files .....	181
1.3.2.1 Encrypting Passwords with Cipher Tool .....	181
1.3.2.2 Resolving Encrypted Passwords .....	188
1.3.2.3 Carbon Secure Vault Implementation .....	189
1.3.3 Configuring Transport Level Security .....	192
1.3.4 Enabling Java Security Manager .....	196
1.3.5 Using Asymmetric Encryption .....	199
1.3.5.1 Creating New Keystores .....	202
1.3.5.2 Configuring Keystores in WSO2 Products .....	206
1.3.5.3 Managing Keystores with the UI .....	209
1.3.6 Using Symmetric Encryption .....	212
1.3.7 Mitigating Cross Site Request Forgery Attacks .....	213
1.3.8 Mitigating Cross Site Scripting Attacks .....	219
1.3.9 Enabling HostName Verification .....	219
1.3.10 Configuring TLS Termination .....	220
1.4 Carbon Applications .....	221
1.4.1 Adding a Custom Proxy Path .....	222
1.4.2 Changing the Default Ports .....	228
1.4.2.1 Default Ports of WSO2 Products .....	229
1.4.3 Configuring SAML2 Single-Sign-On Across Different WSO2 Products .....	235
1.4.4 Configuring the Task Scheduling Component .....	237
1.4.5 Customizing Error Pages .....	239
1.4.6 Customizing the Management Console .....	241
1.4.7 Installing WSO2 Products as a Service .....	245
1.4.7.1 Installing as a Linux Service .....	245
1.4.7.2 Installing as a Windows Service .....	248
1.4.8 Managing Datasources .....	256
1.4.8.1 Adding Datasources .....	257
1.4.8.2 Configuring an RDBMS Datasource .....	257
1.4.8.2.1 Configuring the Datasource Provider .....	258
1.4.8.2.2 Configuring a JNDI Datasource .....	259
1.4.8.2.3 Configuring the Datasource Connection Pool Parameters .....	260
1.4.8.3 Configuring a Custom Datasource .....	265
1.4.9 Managing Users, Roles and Permissions .....	267
1.4.9.1 Changing a Password .....	268
1.4.9.2 Configuring Roles .....	268
1.4.9.3 Configuring Users .....	271
1.4.9.4 Role-based Permissions .....	274
1.4.9.5 Managing User Attributes .....	279
1.4.10 Working with Composite Applications .....	284
1.4.10.1 Introduction to Composite Applications .....	285

1.4.10.2 Packaging Artifacts into Composite Applications .....	288
1.4.10.3 Deploying Composite Applications in the Server .....	292
1.4.11 Working with Features .....	300
1.4.11.1 Introduction to Feature Management .....	300
1.4.11.2 Managing the Feature Repository .....	301
1.4.11.3 Installing Features .....	302
1.4.11.3.1 Installing Features using pom Files .....	302
1.4.11.3.2 Installing Features via the UI .....	306
1.4.11.4 Uninstalling Features .....	307
1.4.11.5 Recovering from Unsuccessful Feature Installation .....	308
1.4.12 Working with Multiple Tenants .....	310
1.4.12.1 Introduction to Multitenancy .....	310
1.4.12.2 Configuring the Tenant Loading Policy .....	313
1.4.12.3 Adding New Tenants .....	316
1.4.13 Working with Transports .....	317
1.4.13.1 Introduction to Transports .....	318
1.4.13.2 Carbon Transports .....	318
1.4.13.2.1 HTTP Servlet Transport .....	318
1.4.13.2.2 HTTPS Servlet Transport .....	323
1.4.13.2.3 HTTP-NIO Transport .....	324
1.4.13.2.4 HTTPS-NIO Transport .....	326
1.4.13.2.5 VFS Transport .....	327
1.4.13.2.6 JMS Transport .....	331
1.4.13.2.7 MailTo Transport .....	334
1.4.13.2.8 TCP Transport .....	336
1.4.13.2.9 Local Transport .....	337
1.4.13.2.10 UDP Transport .....	338
1.4.13.2.11 FIX Transport .....	338
1.5 Performance Tuning .....	341
1.6 Monitoring .....	346
1.6.1 Capturing System Data in Error Situations .....	347
1.6.2 JMX-Based Monitoring .....	348
1.6.3 Monitoring Logs .....	361
1.6.3.1 Monitoring Logs using Management Console .....	366
1.6.3.1.1 Configuring Log4j Properties .....	367
1.6.3.1.2 Configuring the Log Provider .....	370
1.6.3.1.3 View and Download Logs .....	373
1.6.3.2 HTTP Access Logging .....	375
1.6.4 Monitoring Message Flows .....	378
1.6.5 Monitoring Performance Statistics .....	380
1.6.6 Monitoring Product Deployment .....	384
1.6.6.1 About the Carbon Monitoring Tool .....	388
1.6.6.2 Configuring the Carbon Monitoring Tool .....	391
1.6.6.3 Using the Command Line .....	397
1.6.6.4 Defining Custom Monitoring Tasks .....	398
1.6.7 Monitoring SOAP Messages .....	398
1.6.8 Monitoring TCP-Based Messages .....	400
1.6.8.1 Starting TCPMon .....	400
1.6.8.2 Message Monitoring with TCPMon .....	401
1.6.8.3 Other Usages of TCPMon .....	404

1.6.9 Monitoring with AppDynamics .....	406
1.6.10 Monitoring with WSO2 Carbon Metrics .....	408
1.6.10.1 Setting Up Carbon Metrics .....	408
1.6.10.1.1 Enabling Metrics and Storage Types .....	408
1.6.10.1.2 Configuring Metrics Properties .....	414
1.6.10.2 Using JVM Metrics .....	416
1.7 Updating WSO2 Products .....	419
1.7.1 How the WSO2 Update Service Works .....	425
1.7.2 Moving Updates into Production .....	426
1.8 Reference Guide .....	426
1.8.1 Configuration Files .....	427
1.8.1.1 Configuring catalina-server.xml .....	428
1.8.1.2 Configuring master-datasources.xml .....	438
1.8.1.3 Configuring registry.xml .....	441
1.8.1.4 Configuring user-mgt.xml .....	445
1.8.1.5 Configuring config-validation.xml .....	447
1.8.2 Database Upgrade Guide .....	450
1.8.3 Product Startup Options .....	451
1.8.4 Supported Cipher Suites .....	452
1.8.5 Directory Structure of WSO2 Products .....	454
1.8.6 WSO2 Patch Application Process .....	455

# WSO2 Administration Guide

Welcome to the WSO2 Administration Guide! In this guide, you will find all you need to know about how to set up and configure a WSO2 product to meet your enterprise requirements. WSO2 products are built on top of a platform called [Carbon](#), which means that most of the configuration tasks are common to all products. This administration guide applies to all WSO2 products of the Carbon 4.4.x platform. 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, you can refer the [product documentation](#), for details on how to use each product.

To download a PDF of this document or a selected part of it, click [here](#) (only generate one PDF at a time). You can also use this link to export to HTML or XML.

# Deployment and Clustering

This section includes information on deployment and clustering of a WSO2 product, including guidelines and concepts. See the following topics for more information.

- [Key Concepts](#)
- [Production Deployment Guidelines](#)

## 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

---

[Introduction to clustering](#) | [About membership schemes](#) | [Clustering compatibility with WSO2 products](#) | [Deciding how to setup 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 act as if they are a single instance and divide up the work. This approach improves **performance**, because requests are distributed among several servers instead of just one, and it ensures **reliability**, because if one instance becomes unavailable or is experiencing high traffic, another instance will seamlessly handle the requests. Clustering also provides the following benefits.

- **High availability:** Some systems may require high availability percentages like two-nines (99%) availability. 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, and since downtime is costly to any business, clustering has a direct and positive impact on costs.
- **Simplified administration:** You can add and remove resources to meet the size and time requirements for your workloads. You can also launch compute jobs using simple APIs or management tools and automate workflows for maximum efficiency and scalability. Administration is simplified by using tools like the deployment synchronizer and log collector.
- **Increased scalability:** Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner, or its ability to be enlarged to accommodate that growth. Scalability enables us to leverage 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. If you are still in development mode, you do not need a cluster, but once you are ready to start testing and to go into production, where performance and reliability are critical, you should create a cluster.

For additional general information on clustering, see the following resources:

- Why Clustering: <http://publib.boulder.ibm.com/html/as400/v4r4/ic2924/info/RZAIGRZAIGA02.HTM>
- Why Would Anyone Need a Computer Cluster?: <http://obscuredclarity.blogspot.com/2008/10/why-would-anyone-need-computer-cluster.html>

WSO2 provides [Hazelcast Community Edition](#) as its default clustering engine. For clustering on a secure channel (i.e., secure Hazelcast), you have to use Hazelcast Enterprise and this is a commercial version of Hazelcast. 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, you must configure it to either of the available membership schemes.

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

All of these membership schemes are ready to use for production but the choice of which of these to use should depend on your production environment. The following table compares the two membership schemes:

Multicast	WKA	AWS
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 IaaS 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. If the WKA member is down, the cluster breaks, and the members will not be able to 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 setup your cluster***

When setting up your cluster, you must consider differed approaches that you need to take. You must decide how you want to setup 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 worker and manager concerns in the cluster.

## Load Balancing

In production environments, services are clustered in order to scale up applications, achieve high availability or to achieve both. By scaling up, the application supports a larger number of user requests and through high availability, the service is available even if few servers are down. To support balancing of load among these servers, a load

balancer is used to distribute requests among the nodes in the cluster. The nodes that receive this incoming traffic are a set of backend worker nodes in a [worker/manager separated cluster](#) or in a cluster that does not support worker/manager separation. This set of worker nodes can be either statically configured or dynamically discovered. In the static mode, new nodes cannot be added to the pre-defined set of worker nodes at runtime, while dynamic load balancers support addition and removal of worker nodes at runtime, without having to know the IP addresses and other connection details of the backend nodes beforehand.

Load balancers come in wide varieties and among them are hardware load balancers, DNS load balancers, transport level load balancers (e.g., HTTP level like Apache, Tomcat) and application level load balancers (like Synapse). High-level load balancers, like application level load balancers, operate with more information about the messages they route and hence provide more flexibility, but also incur more overhead. So the choice of a load balancer is a trade-off between performance and flexibility.

There are a wide variety of algorithms or methods of distributing the load between servers. Random or round-robin distribution of load are simple approaches, while more sophisticated algorithms consider runtime properties in the system like machine load or the number of pending requests into consideration. Furthermore, the distribution can also be controlled by application specific requirements like sticky sessions. However, it is worth noting that with a reasonably diverse set of users, simple approaches tend to perform on par with complex approaches and therefore, they should be given the considerations first.

In WSO2 Carbon-based products, cluster messages are used based on axis2 clustering to identify a node that is joining or leaving the cluster.

The following are some key aspects of load balancing.

- Session affinity
- Service-aware load balancing
- Tenant-aware load balancing

#### ***Session affinity***

Stateful applications inherently do not scale well. Therefore, architects minimize server-side state in order to gain better scalability. State replication induces huge performance overheads on the system. As a solution to the problem of deploying stateful applications in clusters, session-affinity-based load balancing can be used.

Session affinity ensures that, when a client sends a session ID, the load balancer forwards all requests containing a particular session ID to the same backend worker node, irrespective of the specified load balancing algorithm. This may look like defeating the purpose of load balancing. But, before the session is created, the request will be dispatched to the worker node which is next-in-line, and a session will be established with that worker node.

#### ***Service-aware load balancing***

Service-awareness provides a cost-effectiveness when used not only in the cloud but also on-premise. In addition, a single load balancer can balance incoming requests to clusters of different services such as Application Servers, Business Process Servers, Mashup Servers etc.

Most of the load balancing processes in a real production environment do not happen at the load balancer level, but the backend worker nodes. As a result, a typical load balancer is designed to front a large number of backend worker nodes. In a traditional deployment, one LB may front a cluster of homogenous worker nodes. One load balancer is generally capable of handling multiple such clusters, and route traffic to the correct cluster, while balancing load according to the algorithm specified for that cluster.

A cluster of homogeneous worker nodes is called a Cloud service, in Cloud deployments. A load balancer that fronts multiple Cloud services is typically called a service-aware load balancer.

#### ***Tenant-aware load balancing***

Tenant-awareness allows the load balancer to provide a scalable approach for balancing the load across a set of

tenants sharing a collection of worker nodes. Tenants can also be partitioned in various ways.

When a typical Cloud deployment scales, it requires tenant-partitioning. For a single Cloud service, there can be multiple clusters and each of these service clusters can handle a subset of the tenants in the system. In such a tenant-partitioned deployment, the load balancers themselves need to be tenant-aware, in order to be able to route the requests to the proper tenant clusters. In a Cloud environment, a tenant-aware load balancer should also be service-aware, since it is the service clusters that are partitioned according to the tenants.

## Separating the Worker and Manager Nodes

The process of separating the worker and manager nodes depends on the [worker/manager clustering pattern](#) you choose, so ensure that you follow the correct configurations where appropriate. Although this topic refers to [WSO2 Application Server \(AS\)](#) in most examples, these concepts apply to other WSO2 products as well.

[ [Why separate the worker and manager nodes?](#) ] [ [Worker/Manager separated clustering patterns](#) ]

### **Why separate the worker and manager nodes?**

When clustering, you must ensure that the manager node is well protected. Generally, you would deploy the manager in a secure location behind a firewall, which will only allow admin traffic from within your network to reach the manager node. This is why we recommend having the manager node separately. It is also best to separate the management and worker concerns in your deployment. To list them out, the advantages of this separation include:

1. **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.
2. **Specific worker node tasks:** Worker nodes can only deploy artifacts and read configuration. The separation enables you to have worker nodes with limited and yet specific tasks.
3. **Lower memory requirements:** There is a lower memory footprint in the worker nodes because the OSGi bundles related to the management console are not loaded.
4. **Improved security:** Management nodes can be behind the internal firewall and can be exposed to clients running within the organization only, while worker nodes can be exposed to external clients.

### **Worker/Manager separated clustering patterns**

Since all WSO2 products are built on the cluster-enabled Carbon platform, you can cluster most WSO2 products in a similar way, although there may be a few differences depending on the product that you use and the deployment pattern you have chosen. WSO2 Carbon version 4.0.0 onwards supports deployment models that consist of 'worker' nodes and 'management' nodes. A worker node is used to serve requests received by clients, whereas a management node is used to deploy and configure artifacts (web applications, services, proxy services, etc.).

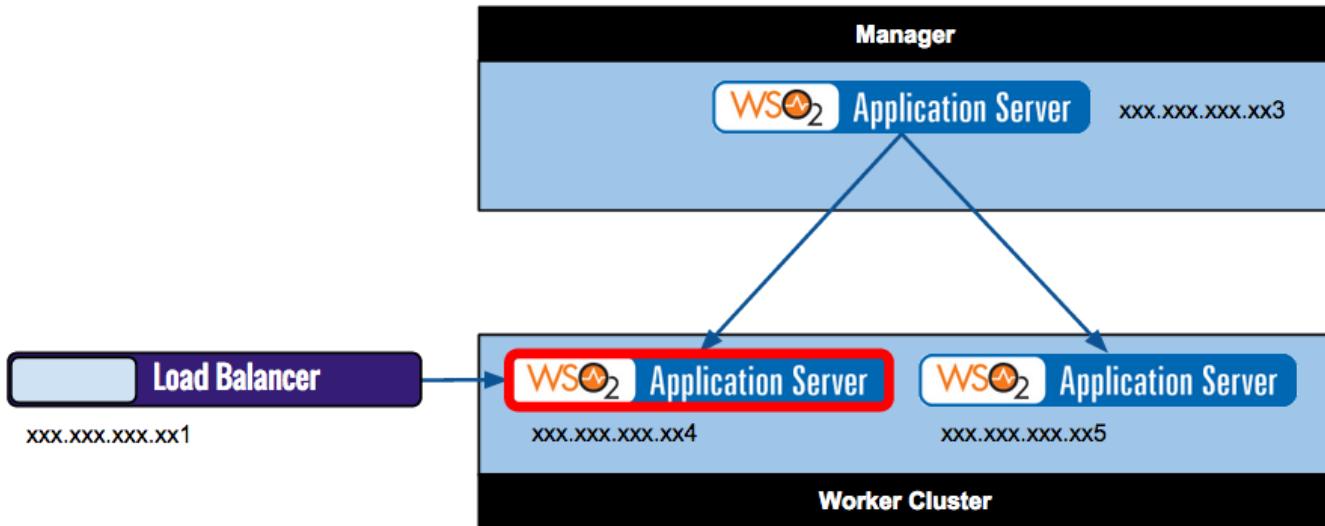
This worker/manager deployment setup provides proper separation of concerns between a Carbon-based product's UI components, management console, and related functionality with its internal framework serving requests to deployment artifacts. Typically, the management nodes are in read-write mode and authorized to add new artifacts or make configuration changes, whereas the worker nodes are in read-only mode, authorized only to deploy artifacts and read configurations. This deployment model provides improved security since its management nodes can be set up behind an internal firewall and only exposed to internal clients while only worker nodes can be exposed externally. Also, since the user interface is not loaded onto 'worker' nodes, the deployment model is more efficient in memory utilization.

You can select one of the following patterns based on your load coupled with your targeted expenditure.

The clustering deployment pattern you choose is very important, as your configurations will change based on the deployment pattern that you use. For the purposes of this worker/manager separation example, we have set the IP addresses of the WSO2 products in the clustering deployment patterns.

## Worker/Manager clustering deployment pattern 1

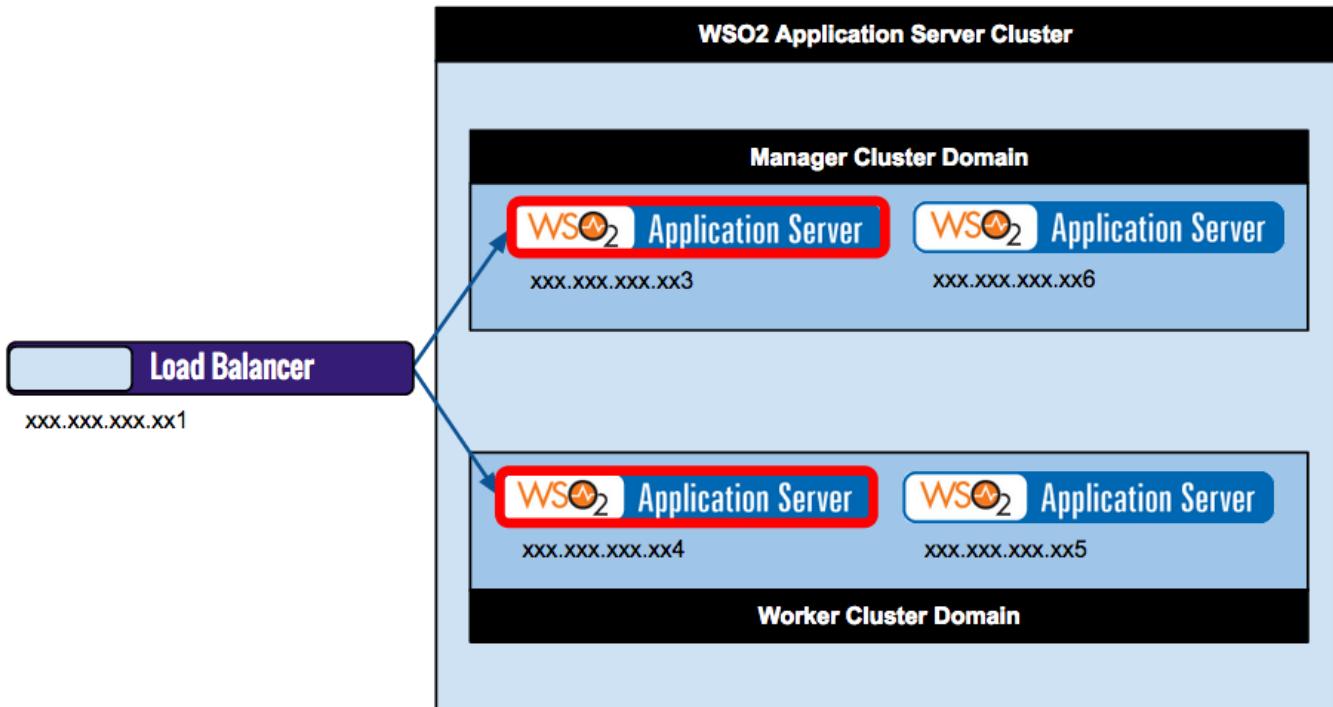
This pattern involves two worker nodes within a cluster. Here the worker is in high availability mode while the manager is not. This pattern is suitable in situations where it is rare to deploy applications or modify a running application (hence you would need only a single management node). However, the application should run continuously, hence the worker nodes are in a cluster.



This mode is rarely used. The preferred mode is two management nodes in Active/Passive mode (as is the case in deployment pattern 2).

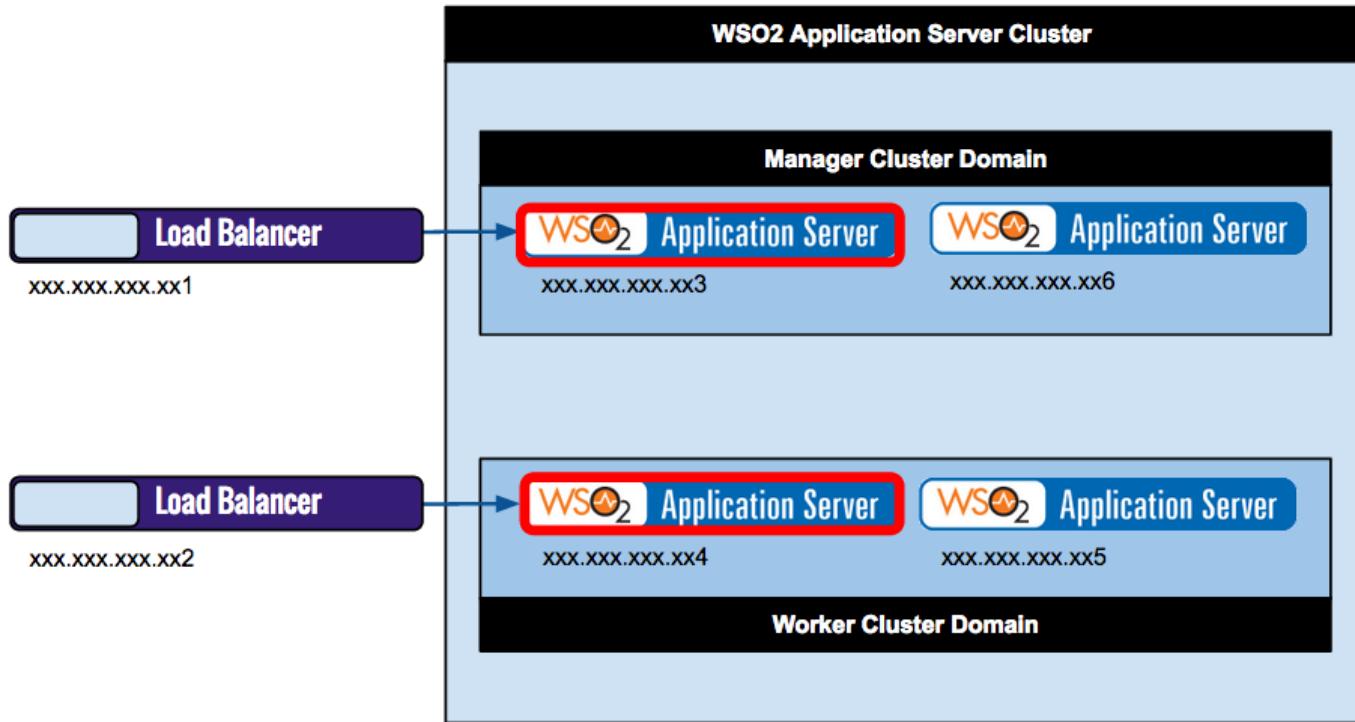
## Worker/Manager clustering deployment pattern 2

This pattern has two manager nodes in one cluster and two worker nodes in a separate cluster. This is similar to deployment pattern 1, but the management node is in Active/Passive mode for high availability. Active/Active mode for a management node is generally not recommended, but if you want high availability for your data center and location-based services, it is useful to set up Active/Active mode. This pattern is useful in scenarios where the application deployment/modification might be frequent (hence you would need a cluster for the management node). However, the load you may have is less, which is why you would 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 sub-domains are part of a single WSO2 product cluster domain. Both sub-domains use their own load balancer while existing within the same cluster. Multiple load balancers result in several unique configuration steps, so please ensure that you follow the relevant steps carefully. 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.

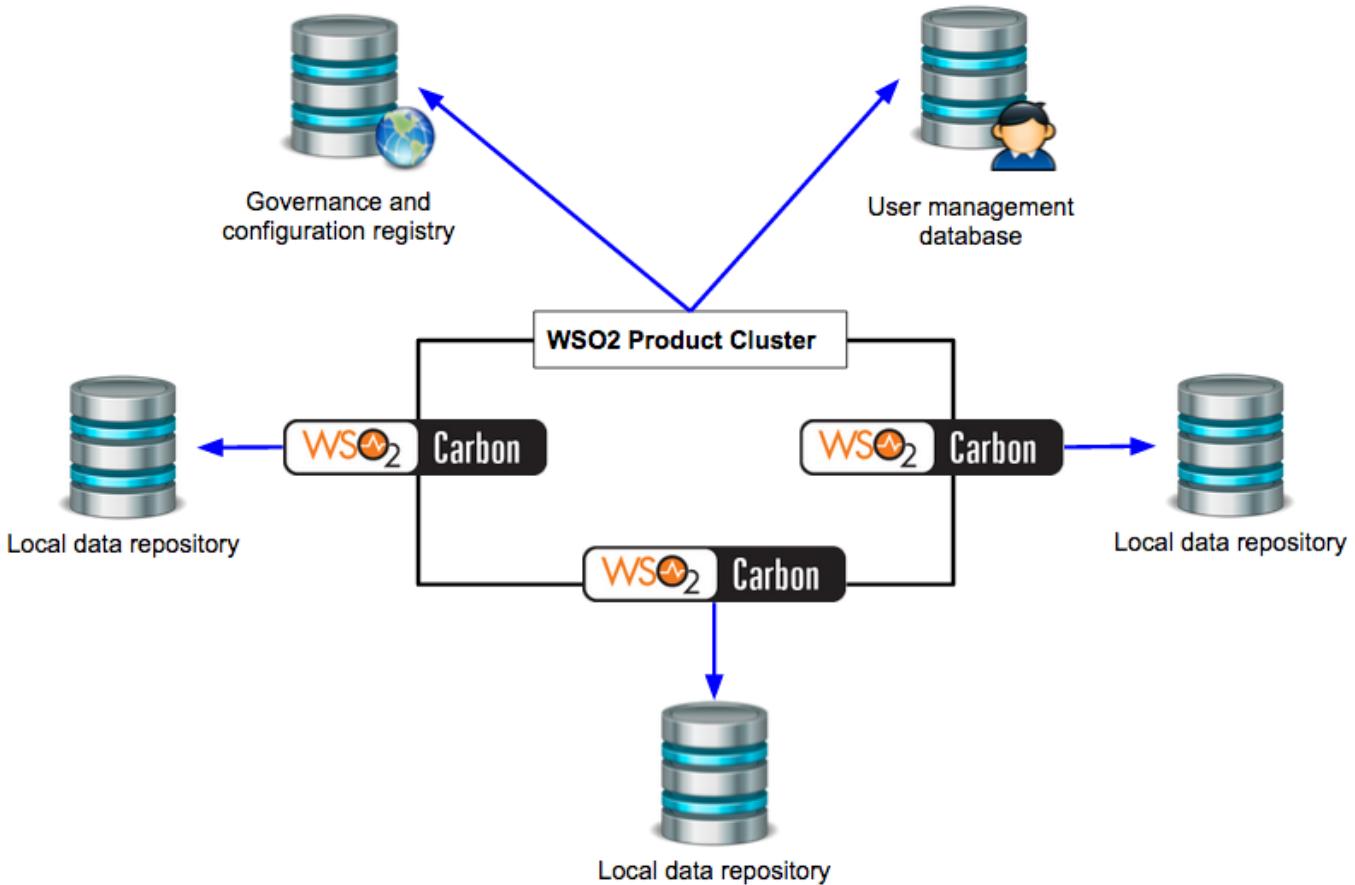
Partition	Description	Type of data
Local data repository	This partition of the registry space is specific to the node that the product resides in. This is not intended to be shared among multiple servers.	The local data repository contains system configuration as well as runtime data that is local to a single instance of a product. For example, the local data repository is used to store mount configurations that are local to each instance.
Configuration registry	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.	The configuration registry, which is the most widely used partition of the registry space, contains product specific configuration.

Governance registry	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.	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.
---------------------	---	---

#### 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 and 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 to balance the load among these products, the situation is a little different.

If the backend application is state-full, it may use sessions. Sessions can be managed 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 used. 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 larger 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](#) ] [ [Installing the WSO2 product](#) ] [ [Running the product](#) ] [ [Host machine security hardening](#) ] [ [Common guidelines and checklist](#) ] [ [Backup and recovery recommendations](#) ]

### **Installation prerequisites**

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

#### ***Hardware requirements (per WSO2 product instance)***

Physical	<ul style="list-style-type: none"> <li>• 3 GHz Dual-core Xeon/Opteron (or latest)</li> <li>• 4 GB RAM (2 GB for JVM and 2 GB for the operating system)</li> <li>• 10 GB free disk space</li> </ul> <p>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.</p>
Virtual Machine	<ul style="list-style-type: none"> <li>• 2 compute units minimum (each unit having 1.0-1.2 GHz Opteron/Xeon processor)</li> <li>• 4 GB RAM</li> <li>• 10 GB free disk space</li> <li>• One CPU unit for the operating system and one for JVM.</li> </ul> <p>Three WSO2 product instances running would require VM of 4 compute units, 8 GB RAM, and 30 GB free space.</p>
EC2	<ul style="list-style-type: none"> <li>• 1 c3.large instance to run one WSO2 product instance.</li> </ul> <p>Three WSO2 product instances can be run in 1 EC2 Extra-Large instance. Based on the I/O performance of the c3.large instance, it is recommended to run multiple instances in a larger instance (c3.xlarge or c3.2xlarge).</p>
Cassandra data nodes	<ul style="list-style-type: none"> <li>• 4 core processors</li> <li>• 8 GB RAM</li> </ul> <p>For more information, see the Cassandra documentation on hardware recommendations for enterprise implementations.</p>

#### ***Environment compatibility***

- All WSO2 Carbon-based products are Java applications that can be run on **any platform that is Oracle JDK 1.7/1.8 compliant. Also, OpenJDK is not recommended or supported.**
- 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.
- 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 [Environments Tested with WSO2 Products](#).

- If you have difficulty in setting up any WSO2 product on a specific platform or with a database, please [contact us](#).

### **Required applications**

The following applications are required for running the WSO2 product and its samples or for building from the source code. Mandatory installs are marked with \*.

Application	Purpose	Version	Download Link
<b>Oracle Java S E Development Kit (JDK)*</b>	<p>Required by all WSO2 products to:</p> <ul style="list-style-type: none"> <li>• To launch the product as each product is a Java application.</li> <li>• To build the product from the source distribution (both JDK and Apache Maven are required).</li> <li>• To run Apache Ant.</li> </ul>	<p>JDK 1.7 or 1.8</p> <ul style="list-style-type: none"> <li>• <b>Oracle and IBM JRE 1.7 are also supported</b> when running (not building) WSO2 products.</li> <li>• If you are using <b>JDK 1.7 on Mac OS or Solaris</b>, install the snappy-java library using the following steps:           <ol style="list-style-type: none"> <li>1. Download the <a href="#">snappy-java JAR</a> and extract it to a preferred location. This folder will be referred to as &lt;SNAPPY_HOME&gt;.</li> <li>2. Copy the appropriate snappy-java library file <code>i386.jnilib</code> (32bit) or <code>x86_64.jnilib</code> (64bit), which is in the &lt;SNAPPY_HOME&gt;/org/xerial/snappy/native/Mac directory, to the &lt;PRODUCT_HOME&gt; directory.</li> </ol> <p>For more information on installing snappy-java library, see <a href="#">Snappy-java fails on Mac OS JDK 1.7</a>.</p> </li> </ul>	<a href="http://java.sun.com">http://java.sun.com</a>

You are now ready to install the WSO2 product.

### **Installing the WSO2 product**

Before you begin, [please see our compatibility matrix](#) to find out if this version of the product is fully tested on your operating system.

Do the following to install the WSO2 product.

1. If you have not done so already, download the latest version of the WSO2 product from the product section of [wso2.com](#).
2. Extract the archive file to a dedicated directory for the WSO2 product. This will hereafter be referred to as <PRODUCT\_HOME> in this guide.

### **Setting JAVA\_HOME**

You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer.

Environment variables are global system variables accessible by all the processes running under the

operating system.

1. In your home directory, open the BASHRC file (.bash\_profile file on OS X) in a Linux text editor, such as vi, emacs, pico, or mcedit.
2. Add the following two lines at the bottom of the file, replacing /usr/java/jdk1.7.0\_80 with the actual directory where the JDK is installed.

On Linux:

```
export JAVA_HOME=/usr/java/jdk1.7.0_80
export PATH=${JAVA_HOME}/bin:${PATH}
```

On OS X:

```
export
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_80/Contents/Home
```

3. Save the file.

If you do not know how to work with text editors in a Linux SSH session, run the following command:

```
cat >> .bashrc
```

Paste the string from the clipboard and press "Ctrl+D".

4. To verify that the JAVA\_HOME variable is set correctly, execute the following command:

On Linux:

```
echo $JAVA_HOME
```

On OS X:

```
which java
```

If the above command gives you a path like /usr/bin/java, then it is a symbolic link to the real location. To get the real location, run the following:

```
ls -l `which java`
```

The system returns the JDK installation path.

#### **Setting system properties**

If you need to set additional system properties when the server starts, you can take the following approaches:

- **Set the properties from a script.** Setting your system properties in the startup script (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 properties 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:**

- 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.

### **Starting the server**

To start the server, you run the script `wso2server.sh` (on Linux) from the `bin` folder.

To start and stop the server in the background mode of Linux, run `wso2server.sh start` and `wso2server.sh stop` commands.

1. Open the command line on Linux and establish an SSH connection to the server or log into the text Linux console.
2. Execute the following command, where `<PRODUCT_HOME>` is the directory where you installed the product distribution:

```
sh <PRODUCT_HOME>/bin/wso2server.sh
```

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`.

Alternatively, you can restrict access to the management console of your product by binding the management console with selected IP addresses. Note that you can either restrict access to the management console only, or you can restrict access to all web applications in your server as explained below.

- To control access only to the management console, add the IP addresses to the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml` file as follows:

```
<Valve
  className="org.apache.catalina.valves.RemoteAddrValve"
  allow="|<IP-address-02>|<IP-address-03>"/>
```

The `RemoteAddrValve` Tomcat valve defined in this file will only apply to the Carbon management console, and thereby all outside requests to the management console will be blocked.

- To control access to all web applications deployed in your server, add the IP addresses to the `<PRODUCT_HOME>/repository/conf/context.xml` file as follows:

```
<Valve
  className="org.apache.catalina.valves.RemoteAddrValve"
  allow="|<IP-address-02>|<IP-address-03>"/>
```

The `RemoteAddrValve` Tomcat valve defined in this file will apply to each web application hosted on the Carbon server. Therefore, all outside requests to any web application will be blocked.

- You can also restrict access to particular servlets in a web application by adding a Remote Address Filter to the `web.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory), and by mapping that filter to the servlet URL. In the Remote Address Filter that you add, you can specify the IP addresses that should be allowed to access the servlet.

The following example from a `web.xml` file illustrates how access to the management page (`/carbon/admin/login.jsp`) is granted only to one IP address:

```

<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>

```

**Note:** Any configurations (including values) defined in the <PRODUCT\_HOME>/repository/conf/tomcat/catalina-server.xml file applies to all web applications and is globally available across server, regardless of host or cluster. See the official Tomcat documentation for more information about using **remote host filters**.

The operation log appears. When the product server is running, the log displays the message "WSO2 Carbon started in 'n' seconds."

#### **Stopping the server**

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

#### **Tuning parameters for latency**

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. The following tuning parameters can be used as a starting point for your system.

The following configurations must be placed in <PRODUCT\_HOME>/repository/conf/hazelcast.properties file. Create this file if it does not exist.

```

hazelcast.shutdownhook.enabled=false
hazelcast.heartbeat.interval.seconds=1
hazelcast.master.confirmation.interval.seconds=5
hazelcast.max.no.heartbeat.seconds=45
hazelcast.max.no.master.confirmation.seconds=60

```

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 that all the members are added to the well-known members list that can be found in the <PRODUCT\_HOME>/repository/conf/axis2/axis2.xml file.

#### **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]
```

### Host machine security hardening

Integrate the newly added server to your organizational backup policy of critical services. This will help you to recover a critical system in case of a compromise. We recommend you apply all the critical security patches from the respective OS vendor. If you have RedHat Enterprise Linux installed you must have an RHN subscription to receive updates. Turning off all services that you're not using is also critical. You can further add a firewall rule to close all ports other than the ports the WSO2 product is running on.

The following links give some sample guidelines on server hardening:

- <http://www.puschitz.com/SecuringLinux.shtml>
- <http://www.tecmint.com/linux-server-hardening-security-tips/>

### 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.

| Guideline                                                                                                                                                                                                                                                                                      | Details                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |               |                                                                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Default credentials                                                                                                                                                                                                                                                                            | <p>Once you have installed the product, it is a good practice to change the default passwords. All WSO2 products come with a default administrator user named 'admin' as username and password. You can change it in the user management section of the Management console.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: right; padding: 5px;">Related links</th></tr> </thead> <tbody> <tr> <td style="padding: 5px;"><a href="#">See Changing a Password</a> for more information on how to change the password of the administrator.</td></tr> </tbody> </table>                                                         | Related links | <a href="#">See Changing a Password</a> for more information on how to change the password of the administrator.                                                                                                                                                                               |
| Related links                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |               |                                                                                                                                                                                                                                                                                                |
| <a href="#">See Changing a Password</a> for more information on how to change the password of the administrator.                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |               |                                                                                                                                                                                                                                                                                                |
| Secure Sockets Layer (SSL) keys                                                                                                                                                                                                                                                                | <p>WSO2 products by default come with a self-signed SSL key. Since these keys are public, it is recommended to replace them with your own keystore.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: right; padding: 5px;">Related links</th></tr> </thead> <tbody> <tr> <td style="padding: 5px;"> <ul style="list-style-type: none"> <li>• See <a href="#">Using Asymmetric Encryption</a> for more information on keystores and their settings in WSO2 products.</li> <li>• See <a href="#">Creating New Keystores</a> for information on how to create and configure your own keys.</li> </ul> </td></tr> </tbody> </table> | Related links | <ul style="list-style-type: none"> <li>• See <a href="#">Using Asymmetric Encryption</a> for more information on keystores and their settings in WSO2 products.</li> <li>• See <a href="#">Creating New Keystores</a> for information on how to create and configure your own keys.</li> </ul> |
| Related links                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |               |                                                                                                                                                                                                                                                                                                |
| <ul style="list-style-type: none"> <li>• See <a href="#">Using Asymmetric Encryption</a> for more information on keystores and their settings in WSO2 products.</li> <li>• See <a href="#">Creating New Keystores</a> for information on how to create and configure your own keys.</li> </ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |               |                                                                                                                                                                                                                                                                                                |

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JVM version   | <p>The recommended version is JDK 1.7 or 1.8.</p> <p>For JDK 1.7, set the appropriate Heap and Permgen values for the JVM based on your deployment/wso2server.sh file. You do not need to set this in JDK 1.8 as the MaxPermSize value has been removed.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p><b>For example</b></p> <pre>-Xms512m -Xmx2048m -XX:MaxPermSize=1024m</pre> </div> <div style="border: 1px solid #ccc; padding: 10px; background-color: #e0f2e0; margin-top: 10px;"> <p><b>Tip:</b> To run the JVM with 2 GB (-Xmx2048m), you should ideally have about 4GB of memory available.</p> </div>                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Host name     | <p>By default, WSO2 products identify the host name of the current machine through the Java API. In some environments, users are recommended to configure the host name by setting the host name in the repository/conf/carbon.xml file.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <pre>&lt;HostName&gt;your.host.name&lt;/HostName&gt;</pre> </div> <p>To configure host names for WSDLs and endpoints, users are recommended to add the following parameter to the axis2/axis2.xml file as shown below.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <pre>&lt;parameter name="WSDLEPRPrefix" locked="false"&gt;[http]://your.host.name&lt;/parameter&gt;</pre> </div> <div style="border: 1px solid #ccc; padding: 10px; background-color: #e0f2e0; margin-top: 10px;"> <p><b>Related links</b></p> <ul style="list-style-type: none"> <li>See <a href="#">Setting up host names and ports</a> for more information.</li> <li>See <a href="#">Working with Transports</a> for information on transports in WSO2 products.</li> </ul> </div> |
| Managing logs | <p>It is recommended to configure logging using the management console.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p><b>Related links</b></p> <p><a href="#">See Monitoring Logs</a> for details on how to configure logging details in WSO2 products.</p> </div>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Registry and governance | <p>All WSO2 products make use of an instance of a registry to store configurations. The registry uses an embedded H2 database.</p> <p>This embedded database might yield a lower performance and is less reliable compared to a standard deployed artifacts. Hence, you should look at associated trade-offs, and we recommend that you switch to a remote database.</p> <p>Moreover, it is worth noting that the default setup does not include database backup procedures. This is because the database backup procedures are not configured by default.</p> <p>When the registry database is pointed to a remote database, multiple running instances of the product can share the same configuration stored in the registry. This, in turn, helps with governance.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p><b>Related links</b></p> <p><a href="#">See here</a> for more information on sharing a registry space across multiple WSO2 product instances.</p> </div> |
| User stores             | <p>WSO2 products offer three choices to store user details:</p> <ul style="list-style-type: none"> <li>• Using a database</li> <li>• Using an LDAP server</li> <li>• Using an Active Directory service</li> </ul> <p>The default is to use the embedded H2 database, with the user store schema. Like in the case of Oracle, MySQL or MSSQL. You can point to an existing LDAP or an Active Directory to make use of those userstores.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p><b>Related links</b></p> <p><a href="#">See Configuring User Stores</a> for more information on userstores, how they work, and how to configure them.</p> </div>                                                                                                                                                                                                                                                                                                                         |
| Monitoring with JMX     | <p>WSO2 Products support JMX for monitoring. By default, JMX uses port 9999. You can configure this in the &lt;PRODUCT_HOME&gt;/repository/conf/carbon.xml file.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <pre>&lt;Ports&gt;   &lt;JMX&gt;9999&lt;/JMX&gt; &lt;/Ports&gt;</pre> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p><b>Related links</b></p> <p><a href="#">See JMX-Based Monitoring</a> for information on monitoring WSO2 products using JMX.</p> </div>                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| Tuning WSO2 products                                                                                                                                                                                                                                                                                                                                                                                                                                  | <p>There are performance tuning guidelines for all products. Check these in their respective product documentation.</p> <table border="1"> <thead> <tr> <th colspan="2"><b>Performance tuning guidelines by product</b></th> </tr> </thead> <tbody> <tr> <td colspan="2"> <ul style="list-style-type: none"> <li>• API Manager</li> <li>• Application Server</li> <li>• Business Process Server</li> <li>• Business Rules Server</li> <li>• Data Analytics Server</li> <li>• Data Services Server</li> <li>• Enterprise Mobility Manager</li> <li>• Enterprise Service Bus</li> <li>• Enterprise Store</li> <li>• Governance Registry</li> <li>• Identity Server</li> <li>• Message Broker</li> <li>• Storage Server</li> </ul> </td></tr> </tbody> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                | <b>Performance tuning guidelines by product</b> |  | <ul style="list-style-type: none"> <li>• API Manager</li> <li>• Application Server</li> <li>• Business Process Server</li> <li>• Business Rules Server</li> <li>• Data Analytics Server</li> <li>• Data Services Server</li> <li>• Enterprise Mobility Manager</li> <li>• Enterprise Service Bus</li> <li>• Enterprise Store</li> <li>• Governance Registry</li> <li>• Identity Server</li> <li>• Message Broker</li> <li>• Storage Server</li> </ul> |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <b>Performance tuning guidelines by product</b>                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                 |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| <ul style="list-style-type: none"> <li>• API Manager</li> <li>• Application Server</li> <li>• Business Process Server</li> <li>• Business Rules Server</li> <li>• Data Analytics Server</li> <li>• Data Services Server</li> <li>• Enterprise Mobility Manager</li> <li>• Enterprise Service Bus</li> <li>• Enterprise Store</li> <li>• Governance Registry</li> <li>• Identity Server</li> <li>• Message Broker</li> <li>• Storage Server</li> </ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                 |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| Securing plain-text passwords                                                                                                                                                                                                                                                                                                                                                                                                                         | <p>WSO2 products use a tool called “Secure Vault” to secure the plain-text passwords in configuration files. Use Secure Vault to encrypt all the passwords in the following configuration files.</p> <ul style="list-style-type: none"> <li>• &lt;PRODUCT_HOME&gt;/repository/conf/user-mgt.xml</li> <li>• &lt;PRODUCT_HOME&gt;/repository/conf/carbon.xml</li> <li>• &lt;PRODUCT_HOME&gt;/repository/conf/axis2/axis2.xml</li> <li>• &lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml</li> <li>• &lt;PRODUCT_HOME&gt;/repository/conf/tomcat/catalina-server.xml</li> </ul> <table border="1"> <thead> <tr> <th colspan="2"><b>Related links</b></th> </tr> </thead> <tbody> <tr> <td colspan="2"> <a href="#">See Carbon Secure Vault Implementation</a> for more information on how to secure plain-text passwords.         </td></tr> </tbody> </table>                                                                                                                                                                                                                                                                                                                          | <b>Related links</b>                            |  | <a href="#">See Carbon Secure Vault Implementation</a> for more information on how to secure plain-text passwords.                                                                                                                                                                                                                                                                                                                                    |  |
| <b>Related links</b>                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                 |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| <a href="#">See Carbon Secure Vault Implementation</a> for more information on how to secure plain-text passwords.                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                 |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| Firewalls                                                                                                                                                                                                                                                                                                                                                                                                                                             | <p>The following ports must be accessed when operating within a firewall.</p> <ul style="list-style-type: none"> <li>• 9443 - Used by the management console and services that use the servlet transport, and is defined in the &lt;PRODUCT_HOME&gt;/repository/conf/tomcat/catalina-server.xml file.</li> <li>• 9763 - Used by the services that use servlet transport, and is defined in the &lt;PRODUCT_HOME&gt;/repository/conf/axis2/axis2.xml file.</li> <li>• 9999 - Used for JMX monitoring, defined in the &lt;PRODUCT_HOME&gt;/repository/conf/carbon.xml file.</li> <li>• 8280 - Default HTTP port used by ESB for proxy services, and is defined in the &lt;PRODUCT_HOME&gt;/repository/conf/tomcat/catalina-server.xml file.</li> <li>• 8243 - Default HTTPS port used by ESB for proxy services, and is defined in the &lt;PRODUCT_HOME&gt;/repository/conf/tomcat/catalina-server.xml file.</li> </ul> <table border="1"> <thead> <tr> <th colspan="2"><b>Related links</b></th> </tr> </thead> <tbody> <tr> <td colspan="2"> <a href="#">See Default Ports of WSO2 Products</a> for a list of common and product-specific ports used by WSO2 products.         </td></tr> </tbody> </table> | <b>Related links</b>                            |  | <a href="#">See Default Ports of WSO2 Products</a> for a list of common and product-specific ports used by WSO2 products.                                                                                                                                                                                                                                                                                                                             |  |
| <b>Related links</b>                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                 |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
| <a href="#">See Default Ports of WSO2 Products</a> for a list of common and product-specific ports used by WSO2 products.                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                 |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |

| Proxy servers                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <p>If the product is hosted behind a proxy such as Apache HTTPD, users can configure products to handle proxy properties at the start-up.</p> <pre data-bbox="409 270 752 333"> -Dhttp.proxyHost=xxxx -Dhttp.proxyPort=xxxx </pre> <p>Alternatively, this can be done by adding the following configurations in the &lt;PRODUCT_HOME&gt;/repository/conf/axis2/axis2.xml file:</p> <pre data-bbox="409 502 1086 692"> &lt;parameter name="Proxy"&gt;     &lt;Configuration&gt;         &lt;proxyhost&gt;you.proxy.host&lt;/proxyhost&gt;         &lt;proxyport&gt;your.proxy.port&lt;/proxyport&gt;     &lt;/configuration&gt; &lt;/parameter&gt; </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| High availability                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <p>For high availability, WSO2 products must run on a <a href="#">cluster</a>. This enables the WSO2 products to share a single IP address. Configuration, repository, user management, and business activity monitoring can also be configured in a cluster. Clustering is supported for all WSO2 products except MySQL.</p> <table border="1" data-bbox="368 910 1503 1227"> <thead> <tr> <th data-bbox="1171 931 1351 963">Related links</th></tr> </thead> <tbody> <tr> <td data-bbox="376 1005 1503 1195"> <ul style="list-style-type: none"> <li>See <a href="#">Overview</a> for more details on clustering, what it is, how it helps and other related information.</li> <li>See <a href="#">Separating the Worker and Manager Nodes</a> for information on clustering WSO2 products between the nodes.</li> <li>See <a href="#">Configuring Clustering for Specific Products</a> for more information on product-specific clustering.</li> <li>See <a href="#">Setting up a Cluster</a> for more information on clustering WSO2 products when fronted by a load balancer.</li> <li>See <a href="#">Setting up a MySQL cluster</a> for more information on configuring a cluster for a database.</li> </ul> </td></tr> </tbody> </table> | Related links | <ul style="list-style-type: none"> <li>See <a href="#">Overview</a> for more details on clustering, what it is, how it helps and other related information.</li> <li>See <a href="#">Separating the Worker and Manager Nodes</a> for information on clustering WSO2 products between the nodes.</li> <li>See <a href="#">Configuring Clustering for Specific Products</a> for more information on product-specific clustering.</li> <li>See <a href="#">Setting up a Cluster</a> for more information on clustering WSO2 products when fronted by a load balancer.</li> <li>See <a href="#">Setting up a MySQL cluster</a> for more information on configuring a cluster for a database.</li> </ul> |
| Related links                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <ul style="list-style-type: none"> <li>See <a href="#">Overview</a> for more details on clustering, what it is, how it helps and other related information.</li> <li>See <a href="#">Separating the Worker and Manager Nodes</a> for information on clustering WSO2 products between the nodes.</li> <li>See <a href="#">Configuring Clustering for Specific Products</a> for more information on product-specific clustering.</li> <li>See <a href="#">Setting up a Cluster</a> for more information on clustering WSO2 products when fronted by a load balancer.</li> <li>See <a href="#">Setting up a MySQL cluster</a> for more information on configuring a cluster for a database.</li> </ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Data backup and archiving                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <p>For data backup and for archiving of data, use the functionality provided by the RDBMS.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Feature management                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <p>Compatible features can be installed on any WSO2 product using the <a href="#">WSO2 Carbon Feature Manager</a>. The Feature Manager is a feature of the WSO2 Carbon runtime and is powered by Equinox P2. It allows you to connect to a remote or local P2 repository and get a list of compatible features for your product. You can then install them using the Feature Manager.</p> <table border="1" data-bbox="368 1501 1503 1691"> <thead> <tr> <th data-bbox="1171 1522 1351 1554">Related links</th></tr> </thead> <tbody> <tr> <td data-bbox="376 1586 1503 1670"> <ul style="list-style-type: none"> <li>See <a href="#">Working with Features</a> for more information on features and how to use them.</li> <li>See <a href="#">Provisioning WSO2 Carbon based SOA Products with Equinox P2</a> for more information on provisioning WSO2 products with the Feature Manager.</li> </ul> </td></tr> </tbody> </table>                                                                                                                                                                                                                                                                                                              | Related links | <ul style="list-style-type: none"> <li>See <a href="#">Working with Features</a> for more information on features and how to use them.</li> <li>See <a href="#">Provisioning WSO2 Carbon based SOA Products with Equinox P2</a> for more information on provisioning WSO2 products with the Feature Manager.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                             |
| Related links                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <ul style="list-style-type: none"> <li>See <a href="#">Working with Features</a> for more information on features and how to use them.</li> <li>See <a href="#">Provisioning WSO2 Carbon based SOA Products with Equinox P2</a> for more information on provisioning WSO2 products with the Feature Manager.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

|                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Spooling and log rotation   | <p>Ensure that you have a relevant log rotation scheme to manage logs. Log4J properties for WSO2 repository/conf/log4j.properties file.</p> <p>To roll the <b>wso2carbon.log</b> based on size, the following configurations can be used.</p> <pre>log4j.appenders.CARBON_LOGFILE=org.apache.log4j.RollingFileAppender log4j.appenders.CARBON_LOGFILE=\${carbon.home}/repository/logs/\${instanceId} log4j.appenders.CARBON_LOGFILE.MaxFileSize=1000KB log4j.appenders.CARBON_LOGFILE.MaxBackupIndex=10</pre>                                                                                                                                                                                                                                                                                                                           |
| Configure the UUID for logs | <p>The log pattern defines the output format of the log file. From Carbon 4.4.3 onwards, the conversations will include a UUID. For example, the log pattern can be [%K] [%T] [%S] [%d] %P%5p {%c} - %x %m {%c}%n, where %x is the UUID.</p> <p>The UUID can be used for identifying forged messages in the log. By default, the UUID will be generated every 10 minutes. You can configure the UUID regeneration period by manually adding the following property to the <code>log4j.properties</code> file (in the <code>repository/conf</code> directory):</p> <pre>log4j.appenders.CARBON_LOGFILE.layout.LogUUIDUpdateInterval=&lt;number_of_milliseconds&gt;</pre> <p><b>Related links</b></p> <p><a href="#">See Configuring Log4j Properties</a> for more information on configuring the <code>log4j.properties</code> file.</p> |

## 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

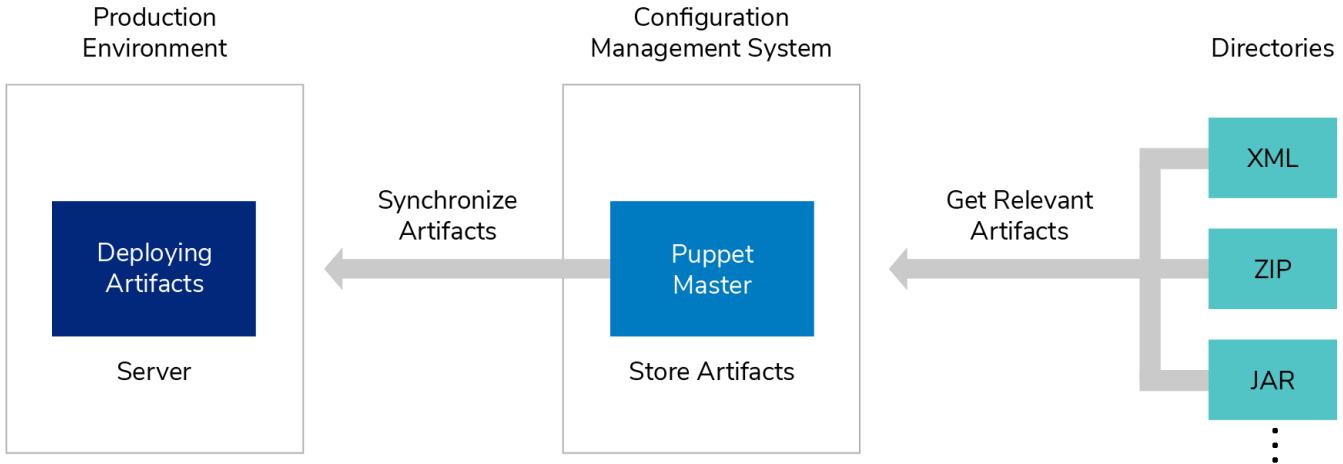
1. **Database backups:**
  - Back up of all the databases defined in `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml`.
  - Back up any other databases configured in any files in the `<PRODUCT_HOME>/repository/conf/datasources` directory.
2. **Artifact backups:**

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.



**Diagram:** managing your artifacts using a configuration management system

#### **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 what the latest, knows, good point is.
- **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](#).

## **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  | -   | 0  | 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.

```

"HTTPS-Sender I/O dispatcher-1" prio=10 tid=0x00007fb54c010000
nid=0x644 runnable [0x00007fb534e20000]
    java.lang.Thread.State: RUNNABLE
        at
org.apache.http.impl.nio.reactor.IOSessionImpl.getEventMask(IOSession
Impl.java:139)
    - locked <0x00000006cd91fef8> (a
org.apache.http.impl.nio.reactor.IOSessionImpl)
        at
org.apache.http.nio.reactor.ssl.SSLIOSession.updateEventMask(SSLIOSe
sion.java:300)
        at
org.apache.http.nio.reactor.ssl.SSLIOSession.inboundTransport(SSLIOSe
sion.java:402)
    - locked <0x00000006cd471df8> (a
org.apache.http.nio.reactor.ssl.SSLIOSession)
        at
org.apache.http.impl.nio.reactor.AbstractIODispatch.inputReady(Abstra
ctIODispatch.java:121)
        at
org.apache.http.impl.nio.reactor.BaseIOReactor.readable(BaseIOReactor
.java:160)
        at
org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvent(Abstr
actIOReactor.java:342)
        at
org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvents(Abst
ractIOReactor.java:320)
        at
org.apache.http.impl.nio.reactor.AbstractIOReactor.execute(AbstractIO
Reactor.java:280)
        at
org.apache.http.impl.nio.reactor.BaseIOReactor.execute(BaseIOReactor.
java:106)
        at
org.apache.http.impl.nio.reactor.AbstractMultiworkerIOReactor$Worker.
run(AbstractMultiworkerIOReactor.java:604)
        at java.lang.Thread.run(Thread.java:722)

```

### **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.

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

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.

**Warning:** Reducing the value of this property to a lower value can result in nodes being considered as dead even if they are not. This results in multiple messages indicating that a node is leaving and rejoining the cluster.

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.

## 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](#)

**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 18091980
      Database Connection Information
      Database server      = DB2/LINUX 9.7.4
      SQL authorization ID = GREG
      Local database alias = REGDB
db2 => GRANT DBADM, CREATETAB, BINDADD, CONNECT, CREATE_NOT_FENCED,IMPLICIT_SCHEMA, LOAD ON DATABASE TO USER user
DB20000I  The SQL command completed successfully.
db2 => ■
```

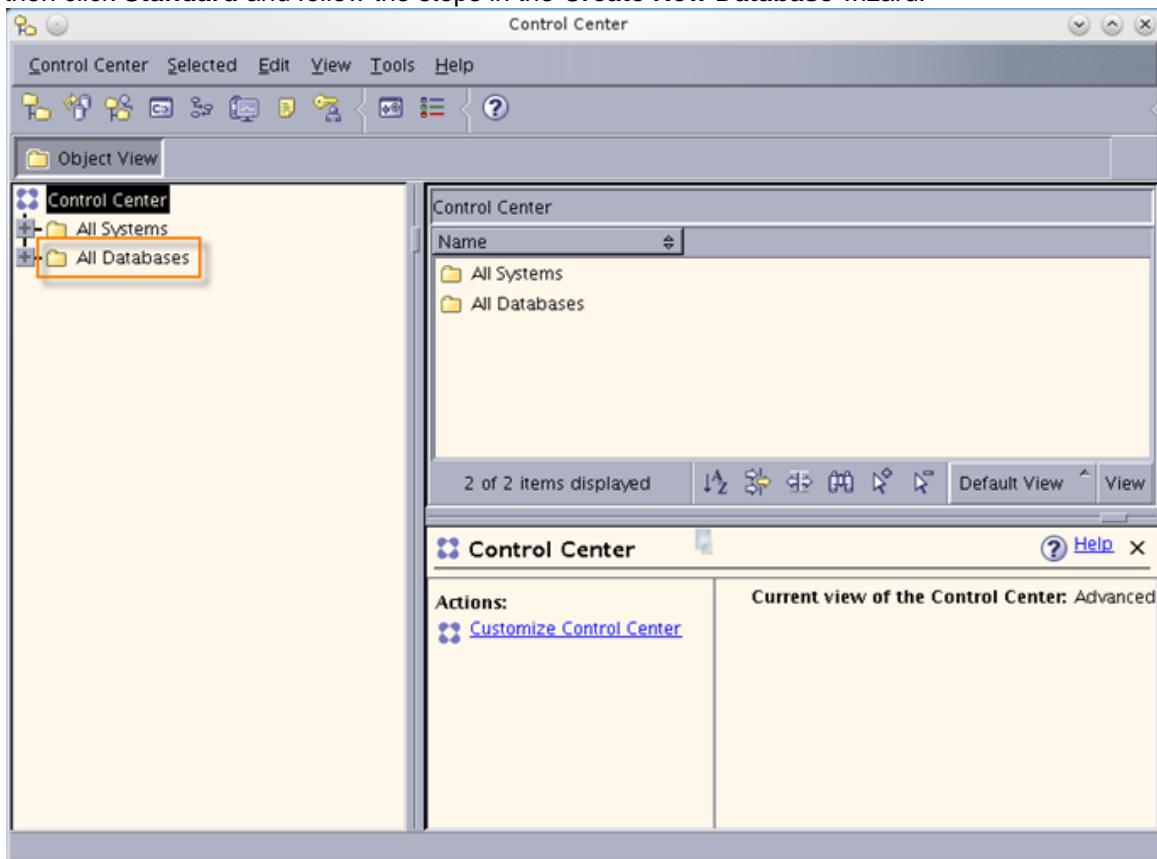
For more information on DB2 commands, see the [DB2 Express-C Guide](#).

### **Using the DB2 control center**

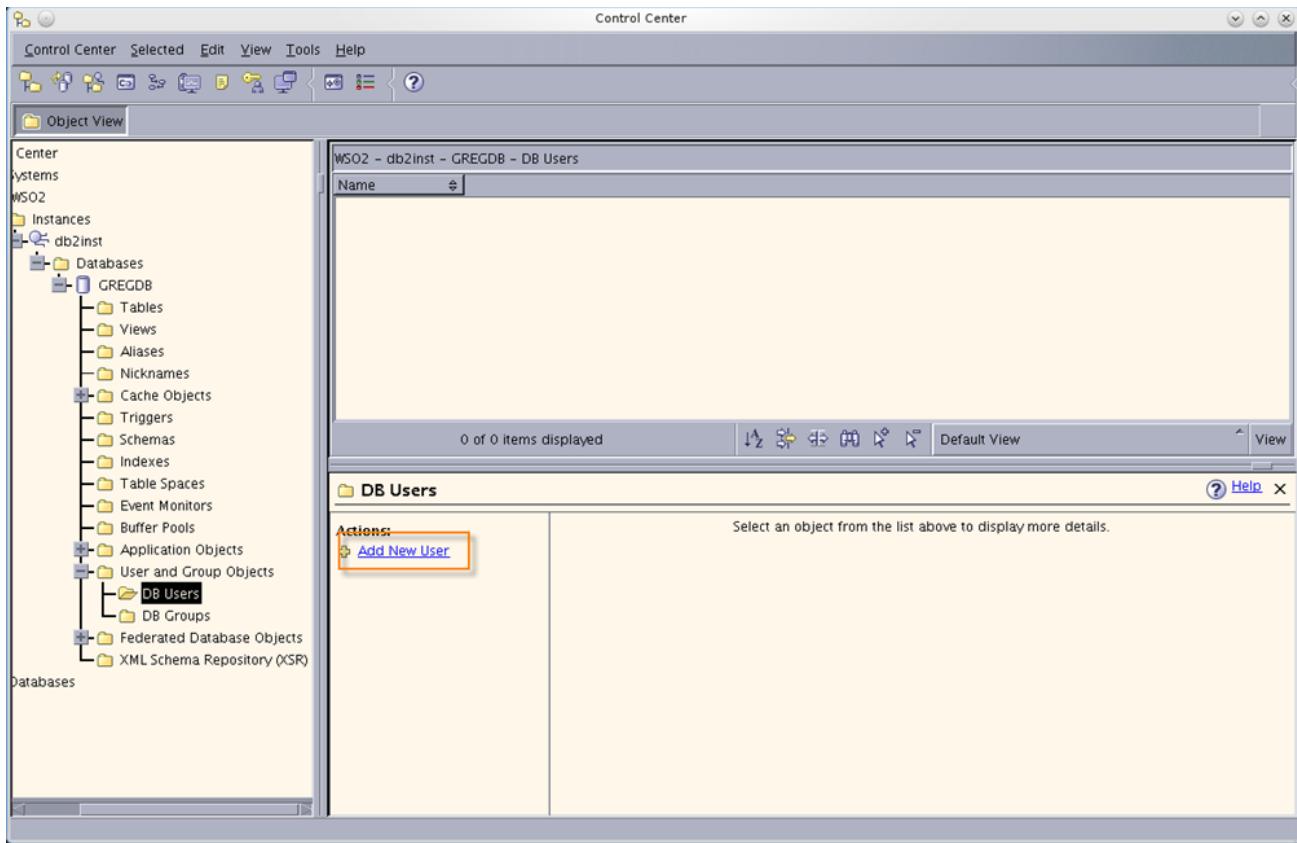
1. Open the DB2 control center using the db2cc command as follows:

```
greg@wso2:~/sqllib/bin$ ./db2cc
```

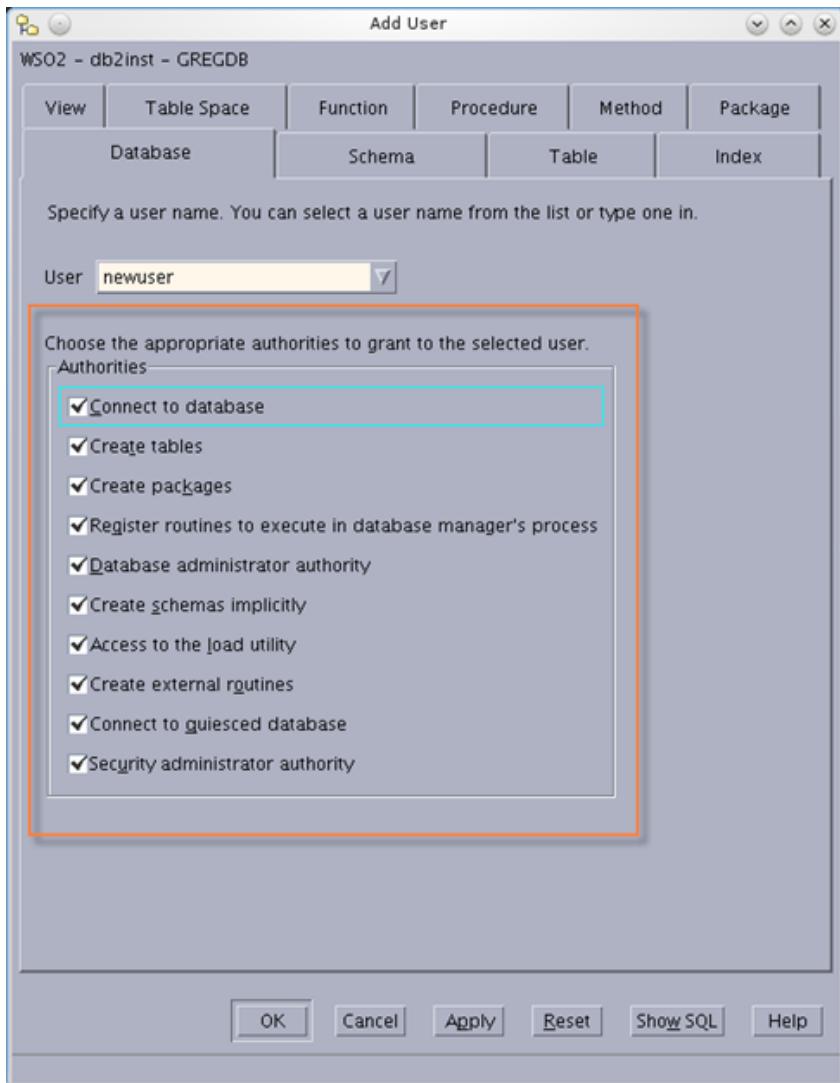
2. Right-click **All Databases** in the control center tree (inside the object browser), click **Create Database**, and then click **Standard** and follow the steps in the **Create New Database** wizard.



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.

```
user@wso2:~/sql1lib/java$ cp db2jcc.jar db2jcc_license_cu.jar /home/user/wso2/greg/repository/components/lib/
user@wso2:~/sql1lib/java$
```

`<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. testspace4) 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](#)

### Setting up Embedded 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

## 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 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](#).

```
client@wso2:~/dtb$ wget -c http://www.h2database.com/h2-2011-09-11.zip
--2011-09-30 00:28:27-- http://www.h2database.com/h2-2011-09-11.zip
Resolving www.h2database.com... 80.74.147.171
Connecting to www.h2database.com|80.74.147.171|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6007851 (5.7M) [application/zip]
Saving to: "h2-2011-09-11.zip"

15% [=====] 923,384      111K/s  eta 45s
```

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@wso2:~/dtb/h2/bin$ chmod 0744 h2.sh
```

3. Run the H2 database server with the following commands:

- For Linux:

```
$ ./h2.sh
```

- For Windows:

```
$ h2.bat
```

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

- 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:

- 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`
- 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.

- 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](#).

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

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

- 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:

```
create database regdb;
```

5. Give authorization to the regadmin user as follows:

```
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:

```
FLUSH PRIVILEGES;
```

7. 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.

**Note** that you must use the MySQL connector that is compatible with your MariaDB version. For example, mysql-connector-java-5.1.36-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.

The best practice is to use port 1433, because you can use it in order processing services.

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.

### 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

### 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
```

- Start the MySQL service using the following command:

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

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

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

- 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.

- 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>;
```

- Give authorization to the regadmin user as follows:

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

- 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.

## 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 listner.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:

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

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

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

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

```
$ 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_OPT` 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 `orcl1`) 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=orcl1
```

2. Connect to Oracle using SQL\*Plus as SYSDBA.

```
[oracle@node1 ~]$ sqlplus SYSDBA/1 as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Fri Nov 18 18:10:42 2011

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

SQL> select 2+2 from dual;

  2+2
-----
     4

SQL> create user dbgreg identified by dbgreg account unlock;

User created.

SQL> grant connect to dbgreg;

Grant succeeded.

SQL> grant create session, dba to dbgreg;

Grant succeeded.

SQL> commit;

Commit complete.

SQL> quit
Disconnected from Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options
[oracle@node1 ~]$ █
```

3. 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;
```

4. 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.

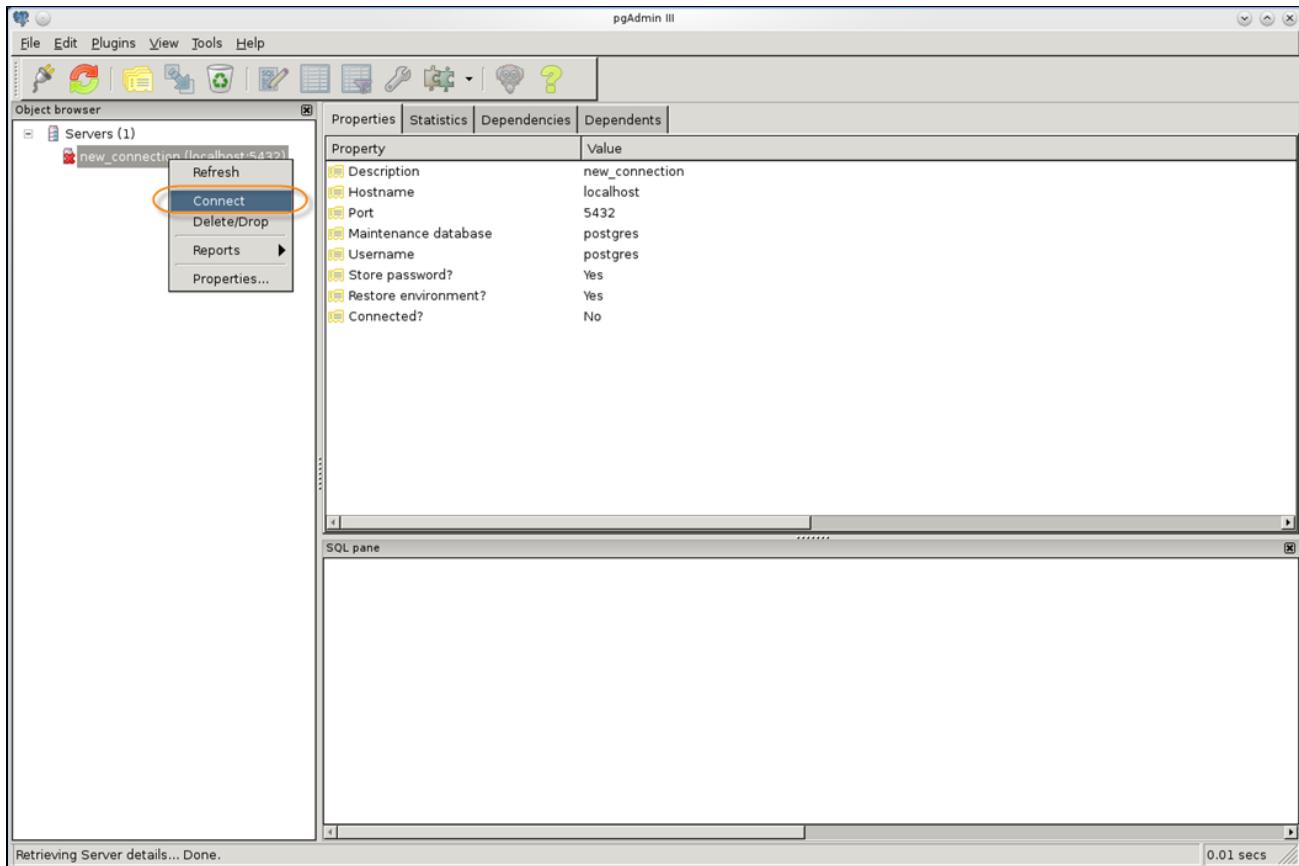
1. Install PostgreSQL on your computer as follows:

```
client@wso2:~/dtb$ sudo apt-get install postgresql
```

2. Start the PostgreSQL service using the following command:

```
client@wso2:~$ sudo /etc/init.d/postgresql start
Starting PostgreSQL 8.4 database server: main.
client@wso2:~$
```

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:



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.

## 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 `derbynnet.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 Oracle
- Changing to Oracle RAC
- Changing to PostgreSQL
- Changing to Remote H2

## 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.

1. Edit the default datasource configuration 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:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>

            <driverClassName>org.apache.derby.jdbc.EmbeddedDriver</driverClassNam
e>
                <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:

| Element                              | Description                                                                                                                                                                                                     |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>url</b>                           | The URL of the database. The default port for a DB2 instance is 50000.                                                                                                                                          |
| <b>username</b> and <b>pass word</b> | The name and password of the database user.                                                                                                                                                                     |
| <b>driverClassName</b>               | The class name of the database driver.                                                                                                                                                                          |
| <b>maxActive</b>                     | 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.                                   |
| <b>maxWait</b>                       | 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. |
| <b>minIdle</b>                       | The minimum number of active connections that can remain idle in the pool without extra ones being created. Enter zero to create none.                                                                          |
| <b>testOnBorrow</b>                  | 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.  |

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>validationQuery</b>    | The SQL query that will be used to validate connections from this pool before returning them to the caller.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>validationInterval</b> | 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.                                                                                                                                                                                                                                                                                                 |
| <b>defaultAutoCommit</b>  | <p>This property is <b>not</b> 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.</p> <p>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.</p> |

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](#).

### 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](#).
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

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@wso2:~/dtb/db-derby-10.8.1.2-bin/bin$ ./ij
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`

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

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.

## 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.

1. Edit the default datasource configuration 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: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>
                <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:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>pass word</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.

<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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](#).
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 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.

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 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***

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.

```

<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:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.

<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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 the DB2 Express-C command editor.

1. To create tables in the registry and user manager database (WSO2CARBON\_DB), use the below script:

```
<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 `-D setup` 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.

```

<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>wso2carbon</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:

Element	Description
<b>url</b>	<p>The URL of the database. The default port for a DB2 instance is 50000.</p> <p>You need to add the following configuration when specifying the connection URL as shown example above:</p> <div style="background-color: #e0f2e0; padding: 10px;">           Add the following configuration to the connection URL when specifying it as shown example above: <code>CLIENT_LOCALE=en_US.utf8;DB_LOCALE=en_us.utf8;IFX_USE_STRENC=true;</code> </div>
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from the pool. Enter any negative value to denote an unlimited number of active connections.

<b>maxWait</b>	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 a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without being created, or enter zero to create none.
<b>testOnBorrow</b>	The indication of whether objects will be validated before being borrowed from the pool. If an object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> applicable to the Carbon database in WSO2 products because 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. Try committing is enabled for RDBMS drivers by default.</p> <p>When auto committing is enabled, each SQL statement will be committed to the database in individual transaction, as opposed to committing multiple statements as a single transaction.</p>

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](#).

### 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.

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.

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 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.

Do not change the `WSO2_CARBON_DB` datasource name in the below 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://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:

Element	Description
<b>url</b>	The URL of the database. The default port for MariaDB is 3306
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.

<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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/

```
<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:

You may have to enter the password for each command when prompted.

```
mysql -u regadmin -p -Dregdb < '<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`

## 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_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 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_CARBON_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\_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.

```

<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</url>
                <username>regadmin</username>
                <password>regadmin</password>

            <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver
ClassName>
                <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:

Element	Description
<b>url</b>	The URL of the database. Change the <IP> with the IP of the server. The best practice is to use port 1433, because you can use it in order processing services.
<b>username</b> and <b>pass word</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.

<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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 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.

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 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.

Element	Description
url	The URL of the database. The default port for MySQL is 3306
username and password	The name and password of the database user

<b>driverClassName</b>	The class name of the database driver
------------------------	---------------------------------------

```

<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.

Element	Description
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	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.

**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.

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](#).

## 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:

```
<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:

```
<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
- To create tables in the registry and user manager database (`WSO2CARBON_DB`), execute the relevant script as shown below.

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

## For MySQL 5.7:

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

- `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**:

1. First, change the existing `mysql.sql` file to a different filename.
2. Change the `<PRODUCT_HOME>/dbscripts/mysql5.7.sql` script to `mysql.sql`.
3. Change the `<PRODUCT_HOME>/dbscripts/identity/mysql5.7.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.

## 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` data source](#), 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.

```

<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/DB_NAME</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:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.

<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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 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> @$<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.

```

<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:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.

<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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 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> @$<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 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.

Be sure to **remove the element <validationQuery>SELECT 1</validationQuery> and add <defaultAutoCommit>false</defaultAutoCommit>**. This change is not required in other datasources such as REGISTRY and USER.

```

<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>
            <defaultAutoCommit>false</defaultAutoCommit>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
<b>url</b>	The URL of the database. The default port for a PostgreSQL instance is 5432.
<b>username</b> and <b>password</b>	The name and password of the database user.
<b>driverClassName</b>	The class name of the database driver.
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.
<b>defaultAutoCommit</b>	Whether to commit database changes automatically or not.

<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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](#).

Be sure to remove the element <validationQuery>SELECT 1</validationQuery> and add <defaultAutoCommit>false</defaultAutoCommit>.

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.

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

```
<PRODUCT_HOME>/dbscripts/postgresql.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 Remote H2

The following sections describe how to replace the default H2 databases with Remote 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 Remote H2 before following the steps to configure your product with it. For more information, see [Setting up Remote 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 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.

```

<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:

Element	Description
<b>url</b>	The URL of the database. The default port for a DB2 instance is 50000.
<b>username</b> and <b>password</b>	The name and password of the database user
<b>driverClassName</b>	The class name of the database driver
<b>maxActive</b>	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.
<b>maxWait</b>	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.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
<b>testOnBorrow</b>	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.
<b>validationQuery</b>	The SQL query that will be used to validate connections from this pool before returning them to the caller.

<b>validationInterval</b>	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.
<b>defaultAutoCommit</b>	<p>This property is <b>not</b> 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.</p> <p>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.</p>

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](#).

### 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**.

H2 Console - Iceweasel

File Edit View History Bookmarks Tools Help

http://127.0.1.1:8082/login.do?sessionid=0cd14a9efc1a047a86dea3d67b330b11

Most Visited ▾ Getting Started Latest Headlines ▾

developerworks : Information... H2 Console WS2 Governance Registry +

Auto commit Max rows: 1000 Auto complete Normal

Run (Ctrl+Enter) clear SQL statement:

```
jdbc:h2:~/REGISTRATIONDB_1
INFORMATION_SCHEMA
Users
H2 1.3.160 (2011-09-11)

CREATE TABLE IF NOT EXISTS REG_CLUSTER_LOCK (
    REG_LOCK_NAME VARCHAR(20),
    REG_LOCK_STATUS VARCHAR(20),
    REG_LOCKED_TIME TIMESTAMP,
    REG_TENANT_ID INTEGER DEFAULT 0,
    PRIMARY KEY (REG_LOCK_NAME)
);

CREATE TABLE IF NOT EXISTS REG_LOG (
    REG_LOG_ID INTEGER AUTO_INCREMENT,
    REG_PATH VARCHAR(2000),
    REG_USER_ID VARCHAR(31) NOT NULL,
    REG_LOGGED_TIME TIMESTAMP NOT NULL,
    REG_ACTION_INTEGER NOT NULL,
    REG_ACTION_DATA VARCHAR(500),
    REG_TENANT_ID INTEGER DEFAULT 0,
    PRIMARY KEY (REG_LOG_ID, REG_TENANT_ID)
);

CREATE TABLE IF NOT EXISTS REG_PATH(
    REG_PATH_ID INTEGER NOT NULL AUTO_INCREMENT,
    REG_PATH_VALUE VARCHAR(2000) NOT NULL,
    REG_PATH_PARENT_ID INT,
    REG_TENANT_ID INTEGER DEFAULT 0,
    CONSTRAINT PK_REG_PATH PRIMARY KEY(REG_PATH_ID, REG_TENANT_ID)
);
CREATE INDEX IF NOT EXISTS REG_PATH_IND_BY_NAME ON REG_PATH(REG_PATH_VALUE, REG_TENANT_ID);
CREATE INDEX IF NOT EXISTS REG_PATH_IND_BY_PARENT_ID ON REG_PATH(REG_PATH_PARENT_ID, REG_TENANT_ID);

CREATE TABLE IF NOT EXISTS REG_CONTENT (
    REG_CONTENT_ID INTEGER NOT NULL AUTO_INCREMENT,
    REG_CONTENT_DATA LONGBLOB,
    REG_TENANT_ID INTEGER DEFAULT 0,
    CONSTRAINT PK_REG_CONTENT PRIMARY KEY(REG_CONTENT_ID, REG_TENANT_ID)
);
```

**Important Commands**

- Displays this Help Page
- Shows the Command History
- Executes the current SQL statement
- Disconnects from the database

- ## 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

# 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 theis 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.



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

A screenshot of the 'Browse' page in the WSO2 Management Console. The page title is 'Browse'. At the top, it shows the 'Root /' node. Below that is a 'Location:' input field containing a '/' symbol and a 'Go' button. Underneath the location field are two buttons: 'Tree view' (which is selected) and 'Detail view'. The main area displays a tree view of the registry structure. It shows a single node named '\_system' which is expanded, revealing its sub-nodes: '\_system/local' and '\_system/config'.

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

The screenshot shows the 'Browse' interface of the WSO2 Message Broker. At the top, it displays 'Root /\_system'. Below this is a search bar with 'Location: /\_system' and a 'Go' button. Underneath are two navigation buttons: 'Tree view' and 'Detail view', with 'Detail view' being selected. The main area contains four panels: 'Metadata' (highlighted with a red border), 'Properties', 'Entries', and 'Permissions', each with a '+' icon to add new items.

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,

Metadata	
<a href="#"> Feed</a>	
Created:	By admin on 03 Aug 11:08:28
Last Updated:	By admin on 03 Aug 11:08:05
Media Type:	text/plain <a href="#"> Edit</a>
Checkpoint:	<a href="#"> Create Checkpoint</a>
Versions:	<a href="#"> View versions</a>
Permalink:	<a href="#"> HTTP</a> <a href="#"> HTTPS</a>
Description:	<a href="#"> Edit</a>

### ***Creating a checkpoint***

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

Metadata	
<a href="#"> Feed</a>	
Created:	By wso2.system.user on 20 May 14:05:26
Last Updated:	By wso2.system.user on 20 May 14:05:26
Media Type:	application/vnd+wso2.sequence
Checkpoint:	<a href="#"> Create Checkpoint</a>
Versions:	<a href="#"> View versions</a>
Description:	<a href="#"> Edit</a>

**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:

Metadata	
<a href="#"> Feed</a>	
Created:	By wso2.system.user 20 May 14:05:26
Last Updated:	By wso2.system.user 20 May 14:05:26
Media Type:	application/vnd+wso2.sequence
Checkpoint:	<a href="#"> Create Checkpoint</a>
Versions:	<a href="#"> View versions</a>
Description:	<a href="#"> Edit</a>

It opens the versions. For example,

Versions of /_system/governance/trunk/services/test			
Version	Last Modified Date	Last Modified By	Actions
1	18m ago	admin	<a href="#"> Details</a> <a href="#"> Restore</a> <a href="#"> Delete Version History</a>

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.

Name	Value	Action
prop1	val	Edit  Delete

2. Enter a unique name for the property and a value and click **Add**.

Name	Value	Action
prop1	val	Edit  Delete

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,

Name	Created On	Author
_system	28 Sep	wso2.system..

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

Links in the Entries panel	Description
<b>Add Collection</b>	See <a href="#">Adding child collections</a> .
<b>Add Resource</b>	See <a href="#">Adding and Editing resources</a> .
<b>Create Link</b>	See <a href="#">Entries and Content#Adding links</a> .

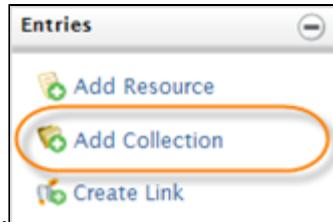
<b>Info</b>	<ul style="list-style-type: none"> <li><b>Feed</b> - Generate a RSS feed for the resource</li> <li><b>Rating</b> - Ratings of the resource</li> <li><b>Media type</b> - Each collection or resource created and stored on the repository has an associated media type. If you leave the media type unspecified, it takes the default value. There are two main ways to configure media types for resources.           <ol style="list-style-type: none"> <li>1. Use a one-time configuration, by modifying the <code>mime.types</code> 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.</li> <li>2. Use the management console as described when adding a collection and a resource.</li> </ol> </li> </ul> <p>Initially, the system contains media types defined in <code>mime.types</code> file. They are available for resources and a set of default media types is available for collections.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>Human-readable media types have shorter names in place of MIME names (for example, <code>WSDL</code> is used instead of <code>application/wsdl+xml</code>). This is achieved by introducing a new file as <code>&lt;PRODUCT_HOME&gt;/repository/conf/mime.mappings</code>. For more information, see <a href="#">Configuring Registry Files</a> section in WSO2 Governance Registry documentation.</p> </div> <p>You can manage media types for resources by editing the properties of <code>/system/mime.types/index</code> collection. This collection contains two resources: <code>collection</code> and <code>custom.ui</code>. To manage media types of collections and custom user interfaces, edit the properties of these two resources.</p>
<b>Actions</b>	<p>Allows you to <b>rename</b>, <b>move</b>, <b>copy</b> or <b>delete</b> a resource/child collection.</p>  <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>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: <a href="#">Remote Instance and Mount Configuration Details</a>.</p> </div> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>These options are not available for all resources/collections.</p> </div>

- Adding child collections
- Adding resources to a collection
- Editing resources
- Adding links

## 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.

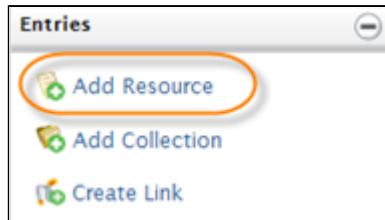


2. Click **Add Collection**.
3. Specify the following options:
  - A unique name for the collection and a description
  - Select a [media type](#) from the drop-down menu
4. Click **Add**.

### 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,

The image shows a screenshot of the 'Add Resource' form. At the top, there is a dropdown menu labeled 'Method' with the option 'Upload content from file' selected, which is highlighted with an orange oval. Below the dropdown, there is a section titled 'Upload Content From File' with fields for 'File \*' (containing 'Choose File No file chosen' and a note 'Give the path of a file to fetch content (xml,wsdl,jar etc..)'), 'Name \*' (empty), 'Media type' (empty), and 'Description' (empty). At the bottom of the form are 'Add' and 'Cancel' buttons.

#### ***Importing content from URL***

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

**Add Resource**

Method	<input type="button" value="Import content from URL ▾"/>
<b>Import Content From URL</b>	
URL *	<input type="text"/>
Give the full url of the resource to fetch content from URL	
Name *	<input type="text"/>
Media type	<input type="text"/>
Description	<input type="text"/>
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

### ***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,

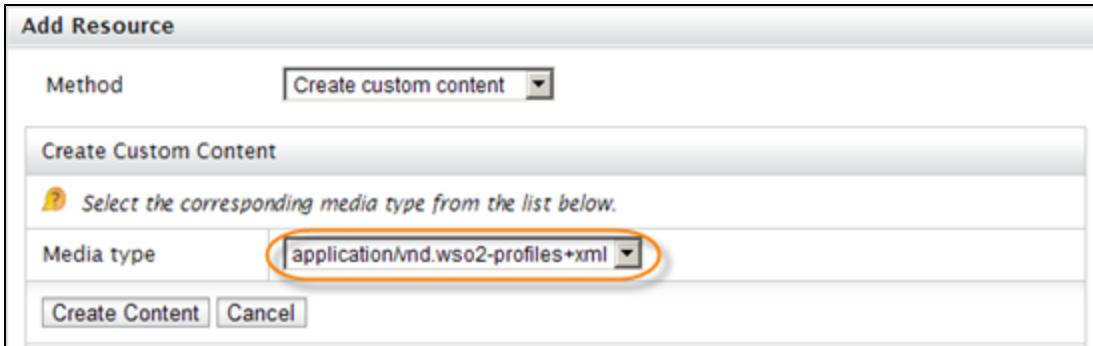
**Create Text Content**

Name *	<input type="text"/>
Media type	<input type="text" value="text/plain"/>
Description	<input type="text"/>
Content	<input checked="" type="radio"/> Plain Text Editor <input type="radio"/> Rich Text Editor Font Name and Size      Font Style      Lists      Insert Item Arial      13 <b>B</b> <i>I</i> <u>U</u> 
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

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.

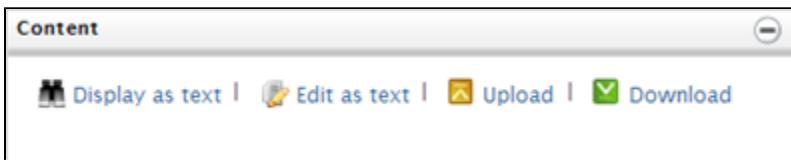
### ***Creating custom content***

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,

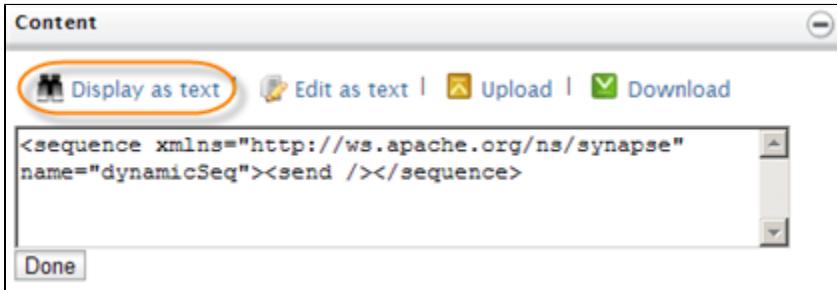


## Editing resources

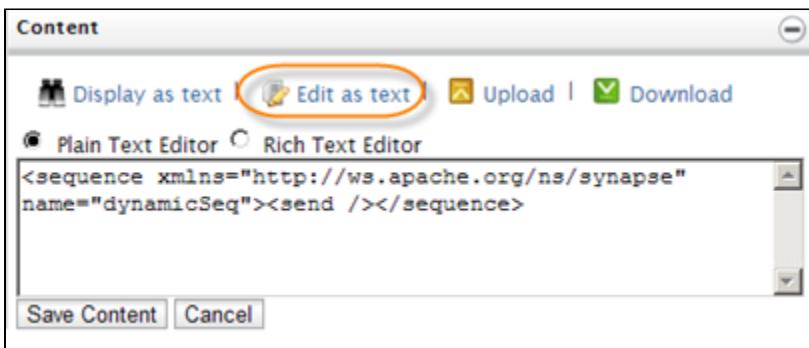
If you select a resource, in its detailed view, you can see the **Content** panel , which provides a UI to edit, upload, and download the content as follows:



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



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

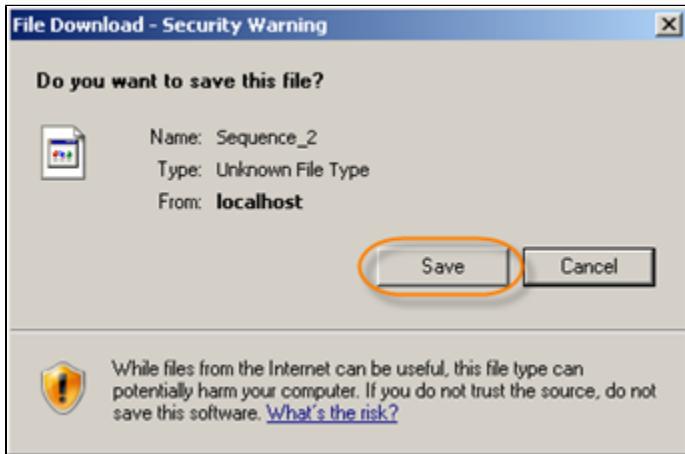


- **Upload** : Allows to upload a file to the resource. The existing content of the resource will be replaced by the you upload.

**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.



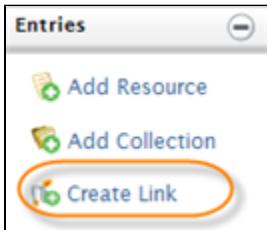
## Adding links

**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.

**Create Link**

Method	Add Symbolic Link						
Add Symbolic Link <table border="1"> <tr> <td>Name *</td> <td>Symb1</td> </tr> <tr> <td>Path*</td> <td>/_system/config</td> </tr> <tr> <td colspan="2"> <input type="button" value="Add"/> <input type="button" value="Cancel"/> </td> </tr> </table>		Name *	Symb1	Path*	/_system/config	<input type="button" value="Add"/> <input type="button" value="Cancel"/>	
Name *	Symb1						
Path*	/_system/config						
<input type="button" value="Add"/> <input type="button" value="Cancel"/>							

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

Name	Created On	Author
permission	13 May	wso2.system..
repository	13 May	wso2.system..
Sequence_2	20 May	wso2.system..
 Symb1	13 May	admin
users	13 May	wso2.system..

### 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.

**Add Remote Link**

Name *	Remote1
Remote Instance*	No instances available
Path	
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

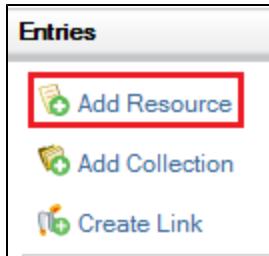
After mounting the Remote collection, you can access and work on that collection from your local instance.

### 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.

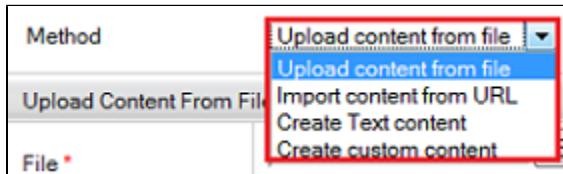
1. To add a new resource, click on the *Add Resource* link.



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.

The image shows a screenshot of the 'Add Resource' panel for 'Upload content from file'. The form fields are as follows:

File *	<input type="text" value="C:\wso2\Chad.wsdl"/> <input type="button" value="Browse..."/>	Give the path of a file to fetch content (xml,wSDL,jar etc..)
Name *	<input type="text" value="Chad.wsdl"/>	
Media type	<input type="text" value="application/wsdl+xml"/>	
Description	<input type="text" value="test"/>	

At the bottom are two buttons: 'Add' (highlighted with a red box) and 'Cancel'.

### ***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.

Import Content From URL	
URL *	http://services/Calculator?wsdl2 Give the full url of the resource to fetch content from URL
Name *	Calculator.wsdl2
Media type	
Description	

**Add** **Cancel**

### ***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.

**Add Resource**

Method	Create Text content
Create Text Content	
Name *	TestResource
Media type	text/plain
Description	
Content	<input checked="" type="radio"/> Plain Text Editor <input type="radio"/> Rich Text Editor <pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;!-- ~ Licensed to the Apache Software Foundation (ASF) under one ~ or more contributor license agreements. See the NOTICE file ~ distributed with this work for additional information ~ regarding copyright ownership. The ASF licenses this file ~ to you under the Apache License, Version 2.0 (the ~ "License"); you may not use this file except in compliance ~ with the License. You may obtain a copy of the License at ~ ~ http://www.apache.org/licenses/LICENSE-2.0 ~ ~ Unless required by applicable law or agreed to in writing,</pre>
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

### Custom Content Creation

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

**Add Resource**

Method	Create custom content
Create Custom Content	
 Select the corresponding media type from the list below.	
Media type	application/vnd.wso2-profiles+xml
<input type="button" value="Create Content"/> <input type="button" value="Cancel"/>	

### 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>\repository\conf\etc 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.

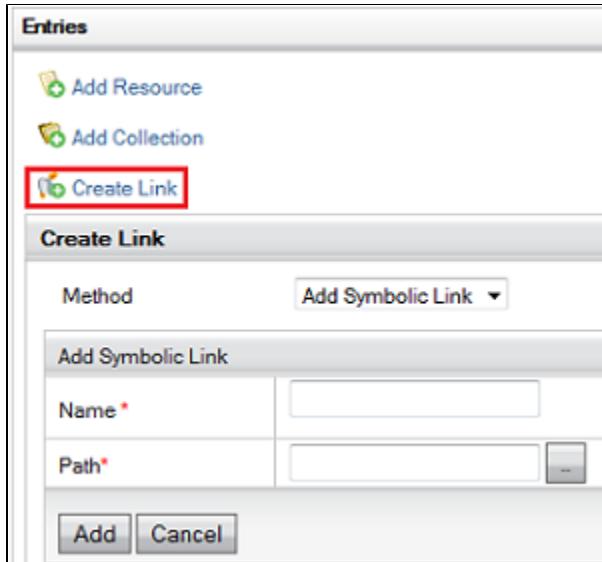
Initially the system contains the media types defined in the mime.types file will be available for resources and a set of default media types will be available for collections.

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.

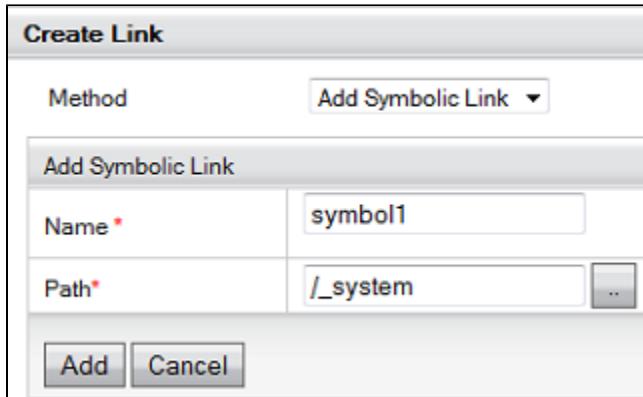
1. Symbolic links and Remote links can be created in a similar way to adding a normal resource. To add a link, click *Create Link* in the *Entries* panel.



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.

**Create Link**

Method: Add Remote Link

Name *	remote1
Remote Instance*	No instances available
Path	

Add Cancel

### Editing collections using the Entries panel

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:

**Entries**

Name	Created On	Author
_system	28 Sep	wso2.system..

Add Resource  
Add Collection  
Create Link

- 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](#).

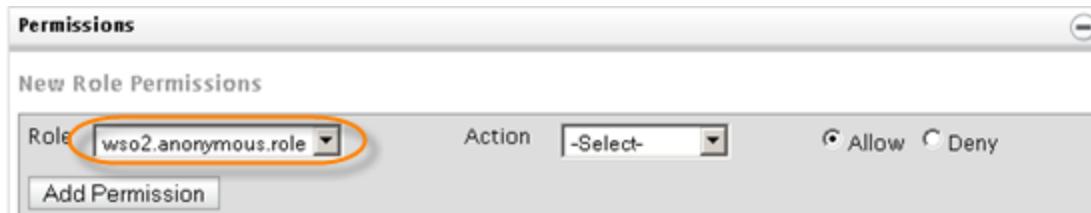
These options are not available for all resources/collections.

### 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.



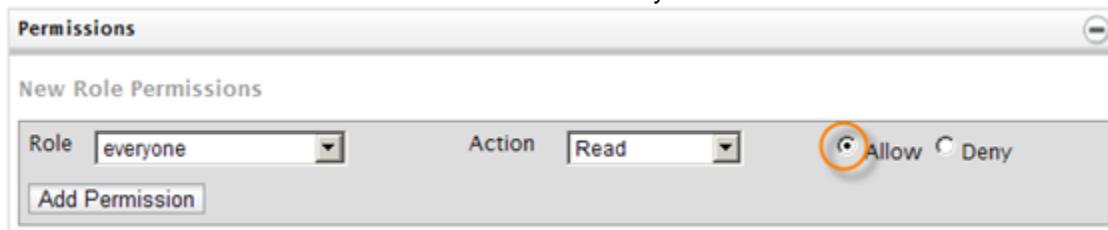
The **wso2.anonymous.role** is a special role that represents a user who is not logged in to the management console. Granting Read access to this role means that you do not require authentication to access resources using the respective Permalinks.

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



**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.

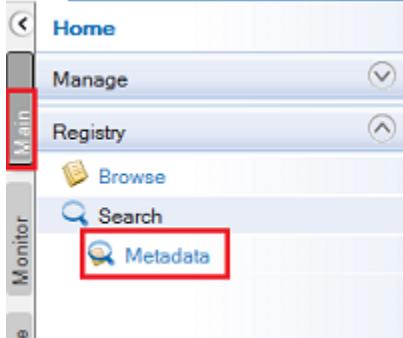
Role	Read		Write		Delete		Authorize	
	Allow	Deny	Allow	Deny	Allow	Deny	Allow	Deny
everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<b>Apply All Permissions</b> <b>Reset</b>								

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.

**Search**

**Search Registry**

Load a Saved Search  
Fill the *Search Options* from a previous search.

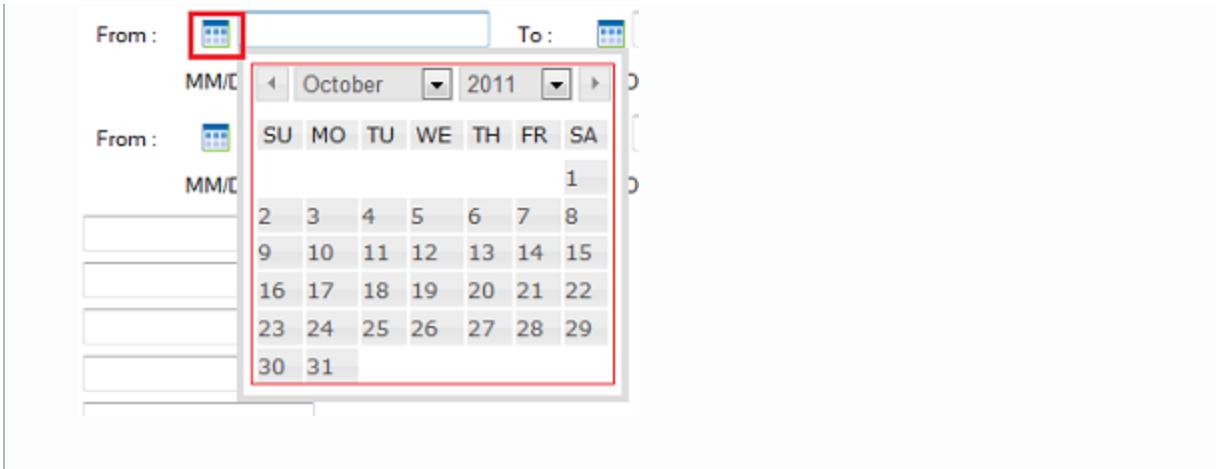
Filter Name	<input type="button" value="None"/>	<input type="button" value="Load"/>
<b>Search Options</b>		
Resource Name	<input type="text"/>	
Created	From : <input type="button" value="Calendar"/>	To : <input type="button" value="Calendar"/>
	MM/DD/YYYY	
Updated	From : <input type="button" value="Calendar"/>	To : <input type="button" value="Calendar"/>
	MM/DD/YYYY	
Created by	<input type="text"/>	
Updated by	<input type="text"/>	
Tags	<input type="text"/>	
Comments	<input type="text"/>	
Property Name	<input type="text"/>	
Property Value	<input type="text"/>	
Media Type	<input type="text"/>	
<input type="button" value="Search"/> <input type="button" value="Clear"/>		

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



- **Created/updated author** - The person who created/updated the resource
- Tags and comments
- Property Name
- Property Value
- Media Type

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**

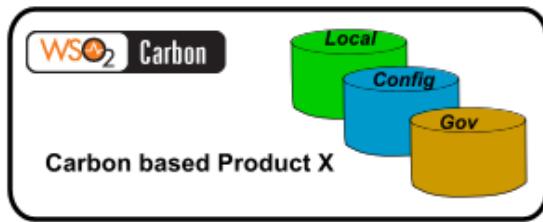
**Pattern #1 : Local Registry**

Figure 1: All registry partitions in a single server instance.

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.

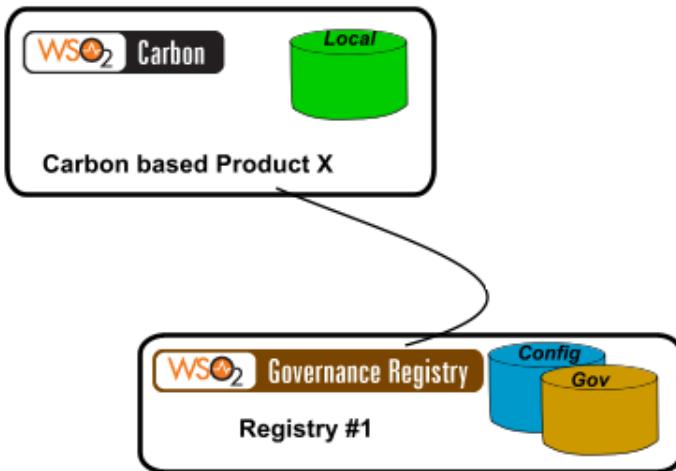
**Pattern #2 : Remote Conf, Go.Reg**

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.

1. Download and extract WSO2 Governance Registry distribution from <http://wso2.com/products/governance-registry>.
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.

4. Copy the 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) 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 wso2server.sh -Dsetup
```

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.
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.

```

<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,

```

<dbConfig name="remote_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>

```

Specify the remote Governance Registry instance with the following configuration:

```

<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.
- <cacheld> : 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 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.

```
<mount path="/_system/config" overwrite="true">
    <instanceId>instanceid</instanceId>
    <targetPath>/_system/config</targetPath>
</mount>
<mount path="/_system/governance" overwrite="true">
    <instanceId>instanceid</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

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:

```
<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' (<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 14:02:31,309] INFO - CarbonCoreActivator Java Home : /home/gillian/Products/Mount/Node1
[2012-12-14 14:02:31,309] INFO - CarbonCoreActivator Java Temp Dir : /home/gillian/Products/Mount/Node1/tmp
[2012-12-14 14:02:31,310] INFO - CarbonCoreActivator User : gillian, en-US, Asia/Colombo
[2012-12-14 14:02:31,383] INFO - Agents Successfully deployed Agent Client
[2012-12-14 14:02:35,145] INFO - EmbeddedRegistryService Configured Registry in 54ms
[2012-12-14 14:02:35,182] INFO - EmbeddedRegistryService Connected to mount at remote_registry in 4ms
[2012-12-14 14:02:35,460] INFO - EmbeddedRegistryService Connected to mount at remote_registry in 1ms
[2012-12-14 14:02:35,495] INFO - RegistryCoreServiceComponent Registry Mode : READ-WRITE
[2012-12-14 14:02:38,739] INFO - ClusterBuilder Clustering has been disabled
[2012-12-14 14:02:39,646] INFO - LandingPageWebappDeployer Deployed product landing page webapp: StandardEngine[Catalina].StandardContext[/home]
[2012-12-14 14:02:39,729] INFO - HttpCoreNIOSSLSSender Loading Identity Keystore from : repository/resources/security/wso2carbon
[2012-12-14 14:02:39,783] INFO - HttpCoreNIOSSLSSender Loading Trust Keystore from : repository/resources/security/client-tru
```

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,

The screenshot shows the 'Browse' interface of the WSO2 Carbon management console. At the top, there is a 'Location' input field with a '/' placeholder and a 'Go' button. Below it are two tabs: 'Tree view' (which is selected) and 'Detail view'. The main area displays a tree structure under 'Root /'. The root node has a red box around it. Underneath, there is a node for '\_system'. Within '\_system', there are two nodes: 'config' and 'governance', both of which are also highlighted with a red box. Other nodes like 'local' are visible but not highlighted.

### 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.

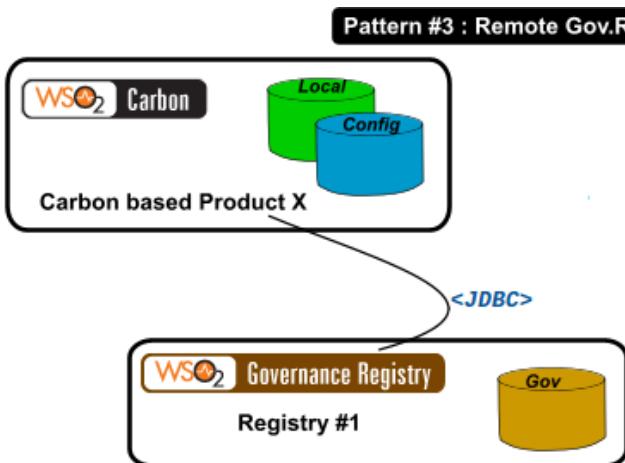


Figure 3: Governance partition in the remote Governance Registry instance.

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
```

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 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.

1. Download and extract WSO2 Governance Registry distribution from <http://wso2.com/products/governance-registry>.
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 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' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) 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 wso2server.sh -Dsetup
```

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,

```
<dbConfig name="remote_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
```

Specify the remote Governance Registry instance with the following configuration:

```
<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="/_system/governance" overwrite="true">
    <instanceId>instanceid</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**

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:

```
<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.

6. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \$G-REG\_HOME/repository/components/lib in both Carbon server instances.

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.

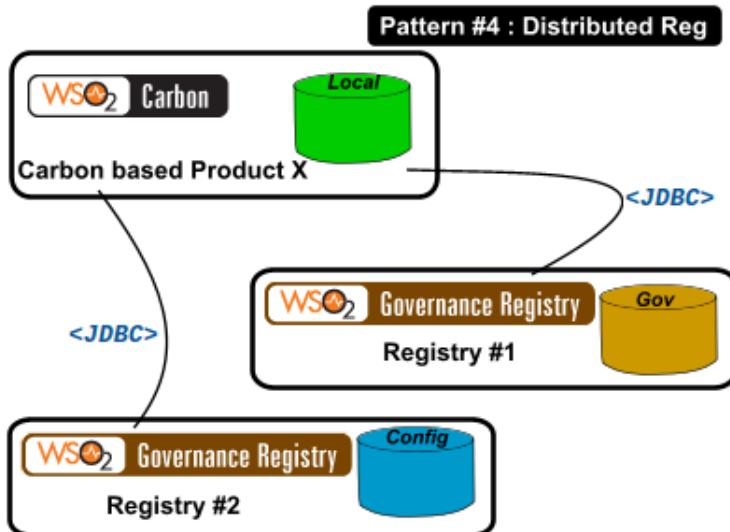


Figure 4: Config and governance partitions in separate registry instances.

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.

1. Download and extract WSO2 Governance Registry distribution from <http://wso2.com/products/governance-registry>.
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>
            <url>jdbc:mysql://10.20.30.42:3306/registrydb2</url>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

4. Repeat the same for G-Reg 3 as follows.

```

<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://10.20.30.43:3306/registrydb3</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>

```

5. Navigate to \$G-REG\_HOME/repository/conf/axis2/axis2.xml file in all instances 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.

6. Copy the 'mySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to G-REG\_HOME/repository/components/lib directory in G-Reg 1, G-Reg 2 and G-Reg 3.

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
```

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.

```

<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>
            <url>jdbc:mysql://10.20.30.42:3306/registrydb2</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 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,

```

<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://10.20.30.42:9443/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.

```

<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:

```

<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.

4. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \$G-REG\_HOME/repository/components/lib in Carbon server instances of Foo product cluster.

### **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.

```

<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>
            <url>jdbc:mysql://10.20.30.43:3306/registrydb2</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 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,

```

<dbConfig name="governance_registry">
    <dataSource>jdbc/WSO2CarbonDB_GREG</dataSource>
</dbConfig>
<dbConfig name="config_registry">
    <dataSource>jdbc/WSO2CarbonDB_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.

## 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.

```

<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 target path 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:

```

<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.

4. Copy 'MySQL JDBC connector jar' (<http://dev.mysql.com/downloads/connector/j/5.1.html>) to \$G-REG\_HOME/repository/components/lib in Carbon server instances of Bar product cluster.
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](#)

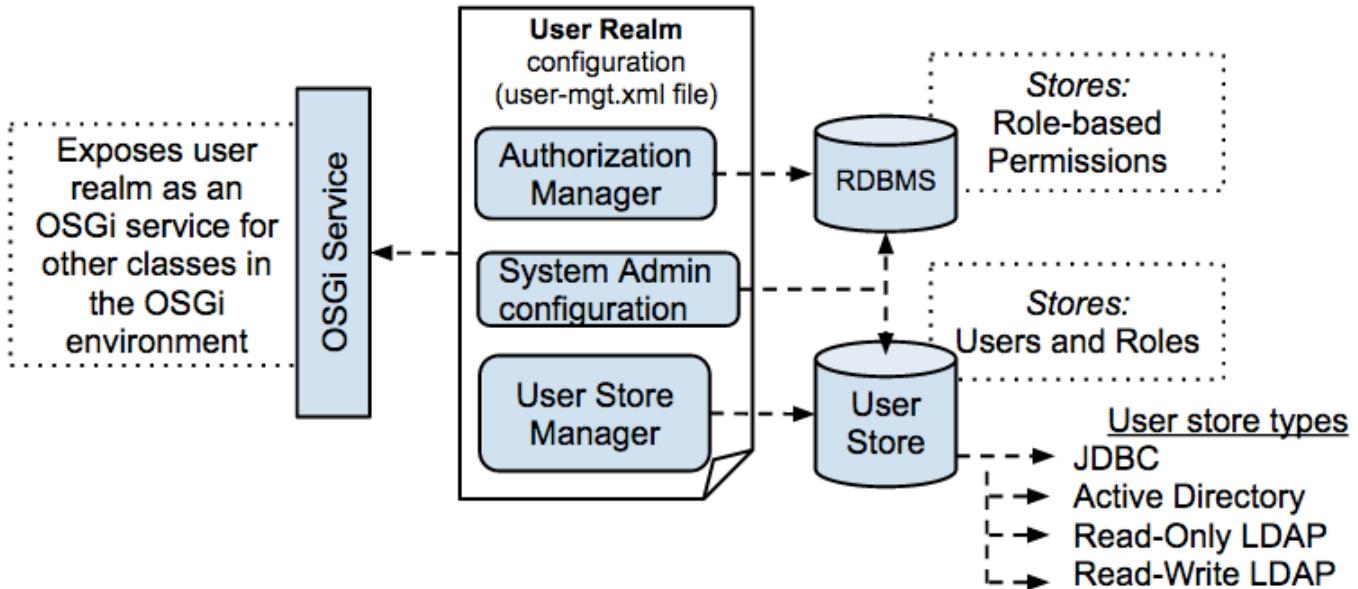
## Introduction to User Management

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.

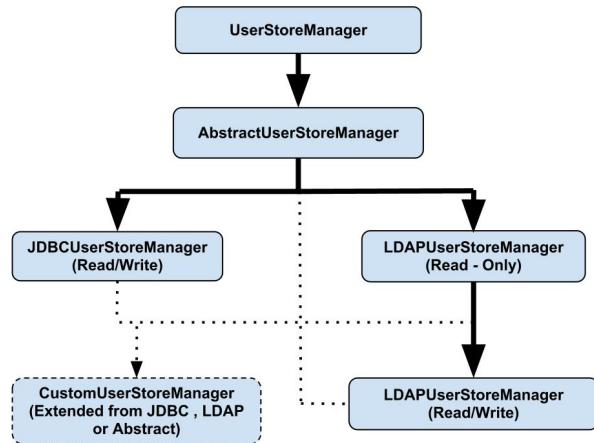
According to the default configuration in WSO2 products, the embedded H2 RDBMS that is shipped with the product is used as the user store as well as the RDBMS for storing information related to permissions.

- **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.

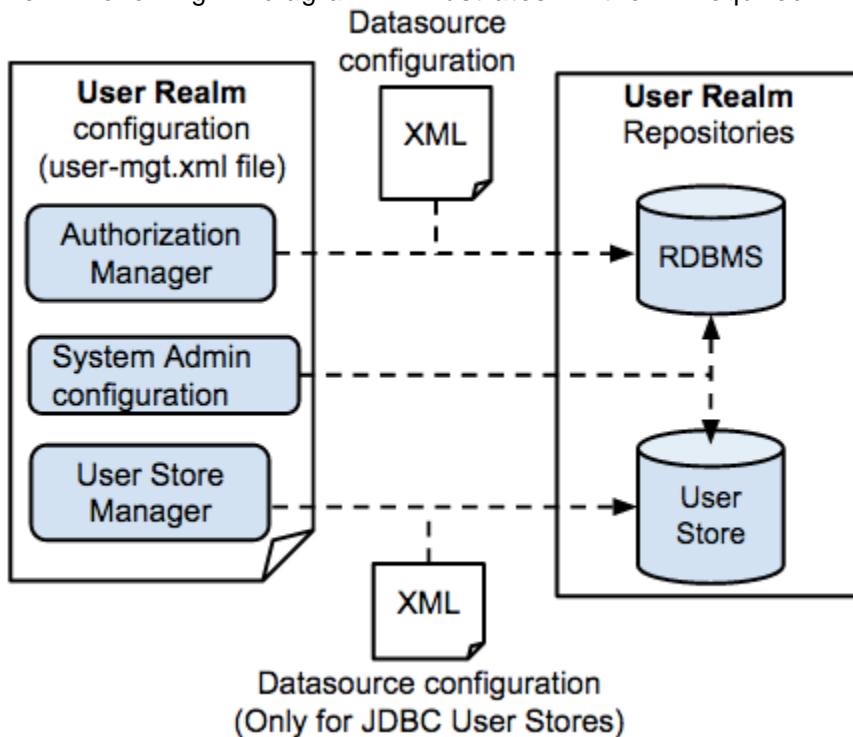
For information on how you can set up and configure the user management realm, see [Configuring the User Realm](#), and for information on how you can manage the users, roles and permissions using the Management Console, see [Managing Users, Roles and Permissions](#).

## 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.

If the primary user store is read-only, you will be using a user ID and role that already exists in the user store, for the administrator. If the user store is read/write, you have the option of creating the administrator user in the user store as explained below. By default, the embedded H2 database (with read/write enabled) is used for both these purposes in WSO2 products.

Note the following key facts about the system administrator in your system:

- The admin user and role is always stored in the primary user store in your system.
- An administrator is configured for your system by default. This **admin** user is assigned to the **admin** role, which has all permissions enabled.
- The permissions assigned to the default **admin** role cannot be modified.

### 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](#): This topic explains how the repository (RDBMS) for storing authorization information (role-based permissions) is configured for your product.
- [Changing a Password](#): This topic explains how you can change the admin password using the management console of the product.

### Updating the administrator

The `<Configuration>` section at the top of the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file allows you to configure the administrator user in your system as well as the RDBMS that will be used for storing information related to user authentication (i.e. role-based permissions).

```

<Realm>
    <Configuration>
        <AddAdmin>true</AddAdmin>
        <AdminRole>admin</AdminRole>
        <AdminUser>
            <UserName>admin</UserName>
            <Password>admin</Password>
        </AdminUser>
        <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in
this role see the registry root -->
        <Property name=""></Property>
        .....
    </Configuration>
    ...
</Realm>

```

Note the following regarding the configuration above.

Element	Description
<AddAdmin>	When <code>true</code> , this element creates the admin user based on the <code>AdminUser</code> 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 <code>false</code> 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 <code>true</code> , 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 <code>AddAdmin</code> value is set to <code>true</code> in this case, it will generate an exception.
<AdminRole>wso2admin</AdminRole>	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.
<AdminUser>	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 <code>&lt;AdminUser&gt;</code> element. If the external user store is in read/write mode, and you set <code>&lt;AddAdmin&gt;</code> to <code>true</code> , the user you specify will be automatically created.
<UserName>	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.

<Password>	<p>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 <code>AddAdmin</code> value is set to <code>true</code>.</p> <p>Note that the password in the <code>user-mgt.xml</code> 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 <code>user-mgt.xml</code> file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the <code>user-mgt.xml</code> file. To change the admin password, you must use the <b>Change Password</b> option from the management console as explained <a href="#">here</a>.</p>
<EveryOneRoleName>	The name of the "everyone" role. All users in the system belong to this role.

## Configuring the Authorization Manager

According to the default configuration in WSO2 products, the Users, Roles and Permissions are stored in the same repository (i.e., the default, embedded H2 database). However, you can change this configuration in such a way that the Users and Roles are stored in one repository (User Store) and the Permissions are stored in a separate repository. A user store can be a typical RDBMS, an LDAP or an external Active Directory. 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.

Property Name	Description
dataSource	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 <code>&lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml</code> file linked from the <code>user-mgt.xml</code> file.
isCascadeDeleteEnabled	This property is set to 'true' by default, which enables cascade delete for the <code>UM_USER_PERMISSION</code> and <code>UM_ROLE_PERMISSION</code> tables when a permission gets deleted from the <code>UM_PERMISSION</code> table. That is, if a record in the parent table is deleted the corresponding records in the child table will be automatically deleted.

You can add more optional configurations using the `<Property>` element:

Property Name	Description	Mandatory
testOnBorrow	It is recommended to set this property to 'true' so that object connections will be validated before being borrowed from the JDBC pool. For this property to be effective, the <code>validationQuery</code> parameter in the <code>&lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml</code> file should be a non-string value. This setting will avoid connection failures. See the section on performance tuning of WSO2 products for more information.	Optional

### **Configuring the Authorization Manager**

Shown below is how the Authorization Manager is enabled in the `user-mgt.xml` file.

```

<AuthorizationManager
    class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
    <Property
        name="AdminRoleManagementPermissions">/permission</Property>
        <Property name="AuthorizationCacheEnabled">true</Property>
</AuthorizationManager>

```

- The `org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager` class enables the Authorization Manager for your product.
- The `AdminRoleManagementPermissions` property sets the registry path where the authorization information (role-based permissions) are stored. Note that this links to the repository that you defined in **Step 1**.
- It is recommended to enable the `GetAllRolesOfUserEnabled` property in the `AuthorizationManager` as follows:

```
<Property name="GetAllRolesOfUserEnabled">true</Property>
```

Although using the user store manager does not depend on this property, you must consider enabling this if there are any performance issues in your production environment. Enabling this property affects the performance when the user logs in. This depends on the users, roles and permission stats.

- By default, the rules linked to a permission (role name, action, resource) are not case sensitive. If you want to make them case sensitive, enable the following property:

```
<Property name="CaseSensitiveAuthorizationRules">true</Property>
```

## 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/use r-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 in the same `<PRODUCT_HOME>/repository/conf/` directory. Alternatively, you can manually create the configuration file and store it in this directory without using the management console.

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 a 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 located in the `<PRODUCT_HOME>/repository/conf/` directory. This file contains the configuration for the primary user store.

`ser-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.

User store manager class	Description
<code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code>	Use <code>ReadOnlyLDAPUsers</code> external LDAP user stores.
<code>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</code>	Use <code>ReadWriteLDAPUsers</code> both read and write operations un-commented in the code in
<code>org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager</code>	Use <code>ActiveDirectoryUser</code> Domain Service (AD DS) or LDS). This can be used <b>only</b> read-only, you must use <code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code> .
<code>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</code>	Use <code>JDBCUserStoreManager</code> stores. This is the user store in the <code>user-mgt.xml</code> file for (which uses the <code>ReadWriteLDAPUserStoreManager</code> ).

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
- Step 2: Updating the system administrator
- Step 3: Updating the datasources
- Step 4: Starting the server

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:

```
<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
- External JDBC User Store

The following sample shows how to configure the internal RDBMS as the JDBC user store:

```
<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">^\[^~!#\$;%^*+=\}\\\\|\\\\\&lt;\&gt;,\\\'\\"]{3,30}\$</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
        name="RolenameJavaRegEx">^\[^~!@#\$;%^*+=\}\\\\|\\\\\&lt;\&gt;,\\\'\\"]{3,30}\$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
</UserStoreManager>
```

The following sample shows how to configure an external RDBMS as the JDBC user store:

```
<UserStoreManager
    class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <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="UserRolesCacheEnabled">true</Property>
            <Property name="PasswordDigest">SHA-256</Property>
            <Property name="ReadGroups">true</Property>
            <Property name="ReadOnly">false</Property>
            <Property name="IsEmailUserName">false</Property>
```

```

<Property name="DomainCalculation">default</Property>
<Property name="StoreSaltedPassword">true</Property>
<Property name="WriteGroups">false</Property>
<Property name="UserNameUniqueAcrossTenants">false</Property>
<Property name="PasswordJavaRegEx">^[\S]{5,30}\$</Property>
<Property name="PasswordJavaScriptRegEx">^[\S]{5,30}\$</Property>
<Property name="UsernameJavaRegEx">^[\S]{5,30}\$</Property>
<Property name="UsernameJavaScriptRegEx">^[\S]{5,30}\$</Property>
<Property name="RolenameJavaRegEx">^[\S]{5,30}\$</Property>
<Property name="RolenameJavaScriptRegEx">^[\S]{5,30}\$</Property>
<Property name="SCIMEabled">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="UserRoleSQL">SELECT UM_ROLE_NAME FROM
UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_USER.UM_USER_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND
UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID AND UM_USER_ROLE.UM_TENANT_ID=?
AND UM_ROLE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
<Property name="UserSharedRoleSQL">SELECT UM_ROLE_NAME,
UM_ROLE.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="IsRoleExistingSQL">SELECT UM_ID FROM UM_ROLE
WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>
<Property name=" GetUserListOfRoleSQL">SELECT UM_USER_NAME FROM
UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_ROLE.UM_ROLE_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND
UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID AND UM_USER_ROLE.UM_TENANT_ID=?
AND UM_ROLE.UM_TENANT_ID=? AND UM_USER.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_ROLE.UM_ROLE_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</Property>
<Property name="IsUserExistingSQL">SELECT UM_ID FROM UM_USER
WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
<Property name=" GetUserPropertiesForProfileSQL">SELECT

```

```

UM_ATTR_NAME, 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=" 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_ATTR_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
    <Property name=" GetUserListForPropertySQL ">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_ATTR_VALUE LIKE ? 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_USER_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=" AddRoleToUserSQL ">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=" AddSharedRoleToUserSQL ">INSERT INTO
UM_SHARED_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_USER_TENANT_ID,
UM_ROLE_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?)</Property>
    <Property name=" RemoveUserFromSharedRoleSQL ">DELETE FROM
UM_SHARED_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE
WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID
FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?) AND
UM_USER_TENANT_ID=? AND UM_ROLE_TENANT_ID = ?</Property>

```

```

<Property name="RemoveUserFromRoleSQL">DELETE FROM UM_USER_ROLE
WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND
UM_TENANT_ID=?) AND UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
<Property name="RemoveRoleFromUserSQL">DELETE FROM UM_USER_ROLE
WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND
UM_TENANT_ID=?) AND UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
<Property name="DeleteRoleSQL">DELETE FROM UM_ROLE WHERE
UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>
<Property name="OnDeleteRoleRemoveUserRoleMappingSQL">DELETE
FROM UM_USER_ROLE WHERE UM_ROLE_ID=(SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
<Property name="DeleteUserSQL">DELETE FROM UM_USER WHERE
UM_USER_NAME = ? AND UM_TENANT_ID=?</Property>
<Property name="OnDeleteUserRemoveUserRoleMappingSQL">DELETE
FROM UM_USER_ROLE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_TENANT_ID=?</Property>
<Property name="OnDeleteUserRemoveUserAttributeSQL">DELETE FROM
UM_USER_ATTRIBUTE 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="UpdateUserPasswordSQL">UPDATE UM_USER SET
UM_USER_PASSWORD= ?, UM_SALT_VALUE=?, UM_REQUIRE_CHANGE=?,
UM_CHANGED_TIME=? WHERE UM_USER_NAME= ? AND UM_TENANT_ID=?</Property>
<Property name="UpdateRoleNameSQL">UPDATE UM_ROLE set
UM_ROLE_NAME=? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>
<Property name="AddUserPropertySQL">INSERT INTO
UM_USER_ATTRIBUTE (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE,
UM_PROFILE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?, ?, ?)</Property>
<Property name="UpdateUserPropertySQL">UPDATE UM_USER_ATTRIBUTE
SET UM_ATTR_VALUE=? WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_ATTR_NAME=? AND
UM_PROFILE_ID=? AND UM_TENANT_ID=?</Property>
<Property name="DeleteUserPropertySQL">DELETE FROM
UM_USER_ATTRIBUTE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?) AND UM_ATTR_NAME=? AND
UM_PROFILE_ID=? 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="AddUserPropertySQL-mssql">INSERT INTO

```

```
UM_USER_ATTRIBUTE (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE,
UM_PROFILE_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), (?), (?), (?), (?)</Property>
<Property name="AddUserToRoleSQL-openedge">INSERT INTO
UM_USER_ROLE (UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT UU.UM_ID,
UR.UM_ID, ? FROM UM_USER UU, UM_ROLE UR WHERE UU.UM_USER_NAME=? AND
UU.UM_TENANT_ID=? AND UR.UM_ROLE_NAME=? AND
UR.UM_TENANT_ID=?</Property>
<Property name="AddRoleToUserSQL-openedge">INSERT INTO
UM_USER_ROLE (UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT UR.UM_ID,
UU.UM_ID, ? FROM UM_ROLE UR, UM_USER UU WHERE UR.UM_ROLE_NAME=? AND
UR.UM_TENANT_ID=? AND UU.UM_USER_NAME=? AND
UU.UM_TENANT_ID=?</Property>
<Property name="AddUserPropertySQL-openedge">INSERT INTO
UM_USER_ATTRIBUTE (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE,
UM_PROFILE_ID, 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:

```
<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 MD 5 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/Mess ageDigest.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.

```
<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.

```
<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>true` should be set to true as shown below.

```
<AddAdmin>true</AddAdmin>
<AdminRole>admin</AdminRole>
<AdminUser>
    <UserName>admin</UserName>
    <Password>admin</Password>
</AdminUser>
```

In the realm configuration section, set the value of the `MultiTenantRealmConfigBuilder` property to `org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder`. For example:

```
<Property
name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
```

### 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](#).

- There are two possible methods for updating datasources:

- 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-datasource.xml` file with the relevant information.

```

<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>

```

- Alternatively, instead of using the master-datasource.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](#).

```

<UserManager>
  <Realm>
    ...
      <UserStoreManager
        class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">
          <Property
            name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDA
PTenantManager</Property>
          <Property
            name="ConnectionURL">ldap://localhost:10389</Property>
          <Property
            name="ConnectionName">uid=admin,ou=system</Property>
            <Property name="ConnectionPassword">admin</Property>
            <Property name="AnonymousBind">false</Property>
            <Property name="UserSearchBase">ou=system</Property>
            <Property name="UserNameAttribute">uid</Property>
            <Property
              name="UserNameSearchFilter">(&& (objectClass=person) (uid=?))</Prope
rty>
            <Property
              name="UserNameListFilter">(objectClass=person)</Property>
              <Property name="DisplayNameAttribute"/>
              <Property name="ReadGroups">true</Property>
              <Property name="GroupSearchBase">ou=system</Property>
              <Property name="GroupNameAttribute">cn</Property>
              <Property
                name="GroupNameSearchFilter">(&& (objectClass=groupOfNames) (cn=?))<
/Property>
              <Property
                name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
                <Property name="MembershipAttribute">member</Property>
                <Property name="BackLinksEnabled">false</Property>
                <Property
                  name="UsernameJavaRegEx">[a-zA-Z0-9._-|//]{3,30} ${}</Property>
                  <Property name="PasswordJavaRegEx">^[\S]{5,30} ${}</Property>
                  <Property
                    name="RolenameJavaRegEx">[a-zA-Z0-9._-|//]{3,30} ${}</Property>
                    <Property name="SCIMEncoded">false</Property>
                    <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
                    <Property name="MultiAttributeSeparator">,</Property>
                    <Property name="MaxUserNameListLength">100</Property>
                    <Property name="MaxRoleNameListLength">100</Property>
                    <Property name="UserRolesCacheEnabled">true</Property>
                    <Property name="ConnectionPoolingEnabled">true</Property>
                    <Property name="LDAPConnectionTimeout">5000</Property>
                    <Property name="ReadTimeout"/>
                    <Property name="RetryAttempts"/>
                    <Property
                      name="ReplaceEscapeCharactersAtUserLogin">true</Property>
                    </UserStoreManager>
                  </Realm>
                </UserManager>
  
```

1. 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:

```
<Property name="ConnectionURL">ldap://<AD
host-ip>:<AD_listen_port></Property>
```

If you are using ldaps (secured) to connect to the Active Directory as shown below, you need to import the certificate of Active Directory to the `client-truststore.jks` of the WSO2 product. See the topic on configuring [keystores](#) for information on how to add certificates to the trust-store.

```
<Property
name="ConnectionURL">ldaps://10.100.1.100:636</Property>
```

2. 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](#).

```
<Property
name="ConnectionName">uid=AdminLDAP,ou=system</Property>
<Property name="ConnectionPassword">2010#Avrudu</Property>
```

3. Update `<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.

```
<Property name="UserSearchBase">ou=system</Property>
```

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.

For example:

```
<Property name="UserNameAttribute">uid</Property>
```

5. 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.

```
<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:

```
<Property
name="GroupSearchBase ">ou=system,CN=Users,DC=wso2,DC=test</
Property>
```

- Set the GroupSearchFilter and GroupNameAttributes. For example:

```
<Property
name="GroupSearchFilter">(objectClass=groupOfNames)</Proper
ty>
<Property name="GroupNameAttribute">cn</Property>
```

- 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>
```

6. 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.

7. 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.

#### 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.

```
<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 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](#).

#### Step 3: Starting the server

Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

#### [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.CommonHybridLDA
        PTenantManager</Property>
    <Property
        name="ConnectionURL">ldaps://10.100.1.100:636</Property>
    <Property
        name="ConnectionName">CN=admin,CN=Users,DC=WSO2,DC=Com</Property>
        <Property name="ConnectionPassword">A1b2c3d4</Property>
        <Property name="AnonymousBind">false</Property>
    <Property
        name="UserSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
        <Property name="UserEntryObjectClass">user</Property>
        <Property name="UserNameAttribute">cn</Property>
    <Property
        name="UserNameSearchFilter">(&amp;(objectClass=user)(cn=?))</Property>
    <Property
        name="UserNameListFilter">(objectClass=user)</Property>
        <Property name="DisplayNameAttribute"/>

```

```

<Property name="ReadGroups">true</Property>
<Property name="WriteGroups">true</Property>
<Property
name="GroupSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
    <Property name="GroupEntryObjectClass">group</Property>
    <Property name="GroupNameAttribute">cn</Property>
    <Property
name="GroupNameSearchFilter">(&& (objectClass=group) (cn=?))</Property>
    <Property
name="GroupNameListFilter">(objectcategory=group)</Property>
        <Property name="MembershipAttribute">member</Property>
        <Property name="MemberOfAttribute">memberOf</Property>
        <Property name="BackLinksEnabled">true</Property>
        <Property name="Referral">follow</Property>
        <Property
name="UsernameJavaRegEx">[a-zA-Z0-9._-|//]{3,30}\$</Property>
        <Property
name="UsernameJavaScriptRegEx">^[\S]{3,30}\$</Property>
        <Property
name="UsernameJavaRegExViolationErrorMsg">Username pattern policy
violated</Property>
        <Property name="PasswordJavaRegEx">^[\S]{5,30}\$</Property>
        <Property
name="PasswordJavaScriptRegEx">^[\S]{5,30}\$</Property>
        <Property
name="PasswordJavaRegExViolationErrorMsg">Password length should be
within 5 to 30 characters</Property>
        <Property
name="RolenameJavaRegEx">[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="EmptyRolesAllowed">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="ReadTimeout" />
<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 Authorization Manager for more information.

```

<AuthorizationManager
class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
<Property
name= "AdminRoleManagementPermissions" >/permission</Property>
<Property name= "AuthorizationCacheEnabled" >true</Property>
<Property name= "GetAllRolesOfUserEnabled" >true</Property>
</AuthorizationManager>

```

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) 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](#).

```

<Property
name= "ConnectionURL" >ldaps://10.100.1.100:636</Property>

```

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.

```
<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.

```
<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:

```
<Property  
name="GroupSearchBase">ou=system,CN=Users,DC=wso2,DC=test</Property>
```

- Set the `GroupSearchFilter` and `GroupNameAttributes`. For example:

```
<Property  
name="GroupSearchFilter">(objectClass=groupOfNames)</Property>  
<Property name="GroupNameAttribute">cn</Property>
```

- 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 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.

```

<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-store.xml` file.

`ser-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

## 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](#).

```

<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDA
PTenantManager</Property>
    <Property
name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServer
Port}</Property>
    <Property name="ConnectionName">uid=admin,ou=system</Property>
    <Property name="ConnectionPassword">admin</Property>
    <Property name="PasswordHashMethod">SHA</Property>
    <Property name="UserNameListFilter">(objectClass=person)</Property>
    <Property name="UserEntryObjectClass">wso2Person</Property>
    <Property name="UserSearchBase">ou=Users,dc=wso2,dc=org</Property>
    <Property
name="UserNameSearchFilter">(&& (objectClass=person)(uid=?))</Prope
rty>
    <Property name="UserNameAttribute">uid</Property>
    <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="UsernameJavaRegEx">^ [ ~!@#$%^*+={}\\|\\\\\\&lt;&gt;, \\\" ]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="RolenameJavaRegEx">^ [ ~!@#$%^*+={}\\|\\\\\\&lt;&gt;, \\\" ]{3,30}$</Property>
    <Property name="ReadGroups">true</Property>
    <Property name="WriteGroups">true</Property>
    <Property name="EmptyRolesAllowed">true</Property>
    <Property
name="GroupSearchBase">ou=Groups,dc=wso2,dc=org</Property>
    <Property
name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
    <Property name="GroupEntryObjectClass">groupOfNames</Property>
    <Property
name="GroupNameSearchFilter">(&& (objectClass=groupOfNames)(cn=?))</Property>
    <Property name="GroupNameAttribute">cn</Property>
    <Property name="SharedGroupNameAttribute">cn</Property>
    <Property
name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property
>
    <Property name="SharedGroupEntryObjectClass">groups</Property>
    <Property
name="SharedTenantNameListFilter">(object=organizationalUnit)</Proper
ty>
    <Property name="SharedTenantNameAttribute">ou</Property>
    <Property
name="SharedTenantObjectClass">organizationalUnit</Property>
    <Property name="MembershipAttribute">member</Property>
    <Property name="LDAPConnectionTimeout">5000</Property>
    <Property name="UserRolesCacheEnabled">true</Property>

```

```
<Property
name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</Property>
</UserStoreManager>
```

1. To read and write to an LDAP user store, 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>
```

2. 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 administrator.

```
<Property name="UserNameAttribute">uid</Property>
```

3. 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>
```

4. 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.
5. 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.

```
<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:

```
<Property
  name="GroupSearchBase">ou=system,CN=Users,DC=wso2,DC=test</
  Property>
```

- Set the GroupSearchFilter and GroupNameAttributes. For example:

```
<Property
  name="GroupSearchFilter">(objectClass=groupOfNames)</Proper
  ty>
<Property name="GroupNameAttribute">cn</Property>
```

- 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>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.

```
<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** sub menu in the **Main** menu.
2. The **Add New User Store** page opens.

**Note:** You cannot update the PRIMARY user store at run time, 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.

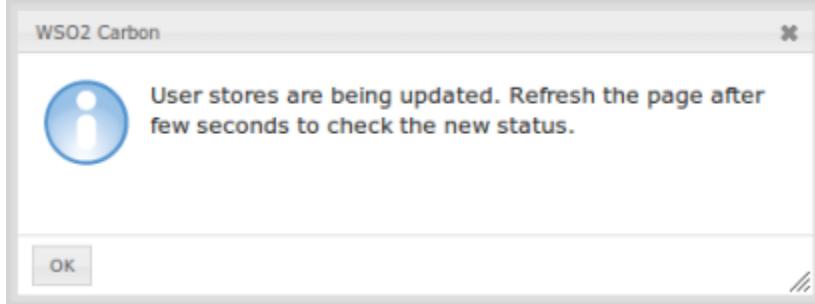
User store manager	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadC user stores
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadW read and w
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use Activ Domain Se LDS). This read-only, : DAPUserS
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBC! can be cor property: <:
org.wso2.carbon.identity.user.store.remote.CarbonRemoteUserStoreManger	Use Carb based on V
	<p><b>Note</b> prod WSC</p>

You can also populate this drop-down list with custom user store manager implementations by adding them to the server. A sample custom user store manager can be found in [the repository](#).

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 an Optional or Advanced section at the bottom of the screen. For information on the properties that are used when defining user stores, see [Properties of User Stores](#).

Property Name	Property Value	Description
ConnectionName*	uid=admin,ou=system	This should be the DN (Distinguish Name) of the admin user in LDAP
ConnectionURL*	localhost:\${Ports.EmbeddedLDAP.LDAPServerPort}	Connection URL for the user store
ConnectionPassword*	*****	Password of the admin user
UserSearchBase*	ou=Users,dc=wso2,dc=org	DN of the context under which user entries are stored in LDAP
Disabled*	<input type="checkbox"/>	Whether user store is disabled
UserNameListFilter*	(objectClass=person)	Filtering criteria for listing all the user entries in LDAP
UserNameAttribute*	uid	Attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, uid and etc
UserNameSearchFilter*	(&(objectClass=person)(uid=?))	Filtering criteria for searching a particular user entry
UserEntryObjectClass*	wso2Person	Object Class used to construct user entries

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 start up.

- 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 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.
- One file only 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.

Property name	Description
MaxUserNameListLength	Controls the number of users listed in the user store of a WSO2 product. Setting this property to 0 displays all users.
ConnectionURL	Connection URL to the user store server. In the case of default LdapUserStore, this URL is included in this configuration.
ConnectionName	The username used to connect to the database and perform various operations on the user store. This value is used by the administrator role in the WSO2 product that you are using, but the ConnectionName user performs search operations on the user store. The value you specify is used by the ConnectionName user.
ConnectionPassword	Password for the ConnectionName user.
DisplayNameAttribute	This is an optional property. The Display Name Attribute is the name of the attribute used to display the user's name in the WSO2 Management Console (Configuration -> Users tab).
PasswordHashMethod	Password hash method to use when storing user entries in the user store.
UserNameListFilter	Filtering criteria for listing all the user entries in the user store. This operation only provides the objects created from the specified class.
UserEntryObjectClass	Object class used to construct user entries. By default, it is a custom class.
UserSearchBase	DN of the context or object under which the user entries are stored. If no base is specified, users, it will start from this location of the directory.  Different databases have different search bases.
UserNameSearchFilter	Filtering criteria used to search for a particular user entry.
UserNameAttribute	The attribute used for uniquely identifying a user entry. Users can be identified by this attribute.  The name of the attribute is considered as the username.

UsernameWithEmailJavaScriptRegEx	This property defines the JavaScript regular expression pattern which you need to support both email as a user name and normal user names.
	<pre>&lt;Property name="UsernameWithEmailJavaScriptRegEx"&gt;^([a-zA-Z0-9_\.-]+ [^@]+@[a-zA-Z0-9\.\-]+\.[a-zA-Z]{2,})&lt;/Property&gt;</pre>
PasswordJavaScriptRegEx	Policy that defines the password format.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user names.
UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to contain at least one character from the set of alphabets, numbers and also ranges of ASCII values in the RegEx.
	<pre>&lt;Property name="UsernameJavaRegEx"&gt;[a-zA-z0-9._- !#\$%`^&amp;~&lt;/Property&gt;</pre>
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role names.
RolenameJavaRegEx	A regular expression used to validate role names. By default, strings have to contain at least one character from the set of alphabets, numbers and also ranges of ASCII values in the RegEx.
ReadGroups	Specifies whether groups should be read from the user store. If this is set to true, the following group configurations are NOT mandatory: GroupSearchBase, GroupSearchFilter, GroupNameListFilter, GroupEntryObjectClass, GroupNameSearchFilter, GroupNameAttribute, MembershipAttribute.
WriteGroups	Specifies whether groups should be written to user store.
EmptyRolesAllowed	Specifies whether the underlying user store allows empty groups to be created. By default, empty groups are allowed to be created. Usually LDAP servers do not allow you to create empty groups.
GroupSearchBase	DN of the context under which user entries are stored in the user store.
GroupSearchFilter	The query used to search for groups.
GroupNameListFilter	Filtering criteria for listing all the group entries in the user store. GroupEntryObjectClass returns objects created from this class.
GroupEntryObjectClass	Object class used to construct group entries.
GroupNameSearchFilter	Filtering criteria used to search for a particular group entry.
GroupNameAttribute	Attribute used for uniquely identifying a user entry. This attribute is required.
MembershipAttribute	Attribute used to define members of groups.
UserRolesCacheEnabled	This is to indicate whether to cache the role list of a user. By default, those changes should be instantly reflected in the Carbon instance.
UserDNPattern	(LDAP) The pattern for the user's DN, which can be defined to improve performances. NPattern provides more impact on performances as the LDAP directory server does not have to perform a search operation.
ReplaceEscapeCharactersAtUserLogin	(LDAP) If the user name has special characters it replaces it to validate the user name.
TenantManager	Includes the location of the tenant manager.

ReadOnly	(LDAP and JDBC) Indicates whether the user store of this realm is read-only.
IsEmailUserName	(JDBC) Indicates whether the user's email is used as their username.
DomainCalculation	(JDBC) Can be either default or custom (this applies when the realm is using JDBC).
PasswordDigest	(JDBC) Digesting algorithm of the password. Has values such as, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, F24, F25, F26, F27, F28, F29, F30, F31, F32, F33, F34, F35, F36, F37, F38, F39, F40, F41, F42, F43, F44, F45, F46, F47, F48, F49, F50, F51, F52, F53, F54, F55, F56, F57, F58, F59, F500.
StoreSaltedPassword	(JDBC) Indicates whether to salt the password.  <b>Tip:</b> 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 have to match the regular expression.
PasswordJavaScriptRegEx	The regular expression used by the front-end components for password validation.
UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to match the regular expression.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for username validation.
RolenameJavaRegEx	A regular expression to validate role names. By default, strings have to match the regular expression.
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role name validation.
MultiTenantRealmConfigBuilder	Tenant Manager specific realm config parameter. Can be used to build the configuration 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 search base.
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 minutes), the connection is closed.

## 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:** Decide on which methods you want to use based on what you want to do. For example, if you want to change the way you authorize users, you only need to implement the relevant method for that purpose. If you want to integrate with a completely different datasource by extending the `AbstractUserStoreManager` class in read/write mode, you must implement all the methods. If your user store is a read-only user store, you must implement the important methods and read 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

Available methods	Default behaviour	Read only

<pre>boolean doAuthenticate(String                       userName, Object credential)</pre>	<p>This method returns details on whether the given username and password is matched or not. Credential is usually a String literal.</p>	<p>If ch auth you this writ imp The this con pas stor The cre pas are hoc salt bef con plac</p>
<pre>String preparePassword(String                       password, String saltValue)</pre>	<p>This returns the encrypted or plain-text password based on the configurations.</p>	<p>You this nee way pas war algo use you</p>
<p><b>Properties</b> <code>getDefaultUserStoreProperties()</code></p>	<p>The default properties of the user store are returned using this method. These properties are used in user store related operations.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>Be sure to manually add the following property when you implement the class:</p> <div style="border: 1px solid #0070C0; padding: 5px; margin-top: 10px;"> <pre>setOptionalProperty("Disabled",                     "false", "Whether user store                     is disabled");</pre> </div> <p>This property is what controls whether the user store is enabled or disabled.</p> </div>	<p>By met pro cha con use imp</p>

<pre>boolean checkUserNameValid(String userName)</pre>	Returns whether the given username is compatible with the defined criteria.	The defi use con regi con you the vali ove met
<pre>boolean checkUserPasswordValid(Object credential)</pre>	Returns whether the given password is compatible with the defined criteria. This is invoked when creating a user, updating a password and authorization.	Sim nan con of ε in you that can met

Read-write methods

Available methods	Default behaviour
<pre>void doAddUser(String userName, Object credential, String[] roleList, Map&lt;String, String&gt; claims, String profileName, boolean requirePasswordChange)</pre>	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.
<pre>void doDeleteUser(String userName)</pre>	This removes the user store record related to the given username.
<pre>void doUpdateCredential(String userName, Object newCredential, Object oldCredential)</pre>	Responsible to update the credential of the given username after authenticating with the existing password.
<pre>void doUpdateCredentialByAdmin(String userName, Object newCredential)</pre>	Admin users can use this method to update the credentials of a given user. This can be done without validating the existing password.
<pre>void doAddRole(String roleName, String[] userList, boolean shared)</pre>	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.
<pre>void doDeleteRole(String roleName)</pre>	This method removes the given role and related mappings from the user store.

<pre>void doUpdateRoleName(String roleName, String newRoleName)</pre>	This method is used to update the name of the existing roles.
<pre>void doUpdateRoleListOfUser(String userName, String[] deletedRoles, String[] newRoles)</pre>	This is used to delete the existing mappings between the given user and the deletedRoles while creating mappings to newRoles.
<pre>void doUpdateUserListOfRole(String roleName, String[] deletedUsers, String[] newUsers)</pre>	Used to delete the existing mappings between the given role and the deletedUsers while creating mappings to newUsers.
<pre>void doSetUserClaimValue(String userName, String claimURI, String claimValue, String profileName)</pre>	This is responsible for creating a new claim for a given user and profile, with the given claim URI and value.
<pre>void doSetUserClaimValues(String userName, Map&lt;String, String&gt; claims, String profileName)</pre>	This is responsible for creating a new claim for a given user and profile, with the given list of claim URIs and values.
<pre>void doDeleteUserClaimValue(String userName, String claimURI, String profileName)</pre>	Remove the existing claim details mapped with the given user and profile.
<pre>void doDeleteUserClaimValues(String userName, String[] claims, String profileName)</pre>	Remove the given list of claims from a given user.
<pre>void addRememberMe(String userName, String token)</pre>	This method is used to persist tokens in the user store.

Read methods

Available methods	Default behaviour
<pre>boolean doCheckExistingUser(String userName)</pre>	Returns whether the provided userName already exists in the user store.
<pre>boolean doCheckExistingRole(String roleName)</pre>	Returns whether the provided roleName already exists in the user store.
<pre>String[] doListUsers(String filter, int maxItemLimit)</pre>	This method returns a list of usernames that match with the given filter string.
<pre>String[] doGetRoleNames(String filter, int maxItemLimit)</pre>	Returns a list of role names that match with the given filter string.
<pre>String[] doGetExternalRoleListOfUser(String userName, String filter)</pre>	Returns a list of external role names of a given user that match with the given filter string.

<pre>String[] doGetSharedRoleListOfUser(String userName, String tenantDomain, String filter)</pre>	This method returns a list of shared role names of a given user that match with the given filter string.
<pre>Map&lt;String, String&gt; getUserPropertyValues(String userName, String[] propertyNames, String profileName)</pre>	This method returns values for the given propertyNames for a given userName and profile Name.
<pre>String[] getUserListFromProperties(String property, String value, String profileName)</pre>	This returns a list of usernames that match with the given value of the given property and profileName.
<pre>String[] doGetDisplayNamesForInternalRole(String[] userNames)</pre>	Returns names to display in the UI for given usernames.
<pre>Date getPasswordExpirationTime(String userName)</pre>	Returns the password expiry date of a given user. The default value is null.
<pre>int getUserId(String username)</pre>	This method returns the identifier of a given user name. Default value is 0.
<pre>boolean doCheckIsUserInRole(String userName, String roleName)</pre>	True is returned if the given user is already mapped to the given role name.
<pre>String[] getProfileNames(String userName)</pre>	Returns a list of profile names mapped with a given user name.
<pre>String[] doGetSharedRoleNames(String tenantDomain, String filter, int maxItemLimit)</pre>	This returns a list of role names that are associated with the given tenant domain and match with the filter.
<pre>String[] doGetUserListOfRole(String roleName, String filter)</pre>	This method returns a list of usernames that are mapped with the given rolename.
<pre>String[] getAllProfileNames()</pre>	All the profile names are returned including the default profile.
<pre>boolean isValidRememberMeToken(String userName, String token)</pre>	This method is used to check if the given token exists for the given user.
<pre>boolean isMultipleProfilesAllowed()</pre>	Returns whether this user store is allowed to have multiple profiles per user. The default value is false.
<pre>boolean isBulkImportSupported()</pre>	This method returns whether this user store allows bulk transactions or not.

## 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 above tables. 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.

#### 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.

User store manager class	When you would use it
<code>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</code>	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.
<code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code>	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.
<code>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</code>	If you want to allow the WSO2 product to manipulate user store data, you need to use this implementation.
<code>org.wso2.carbon.user.core.ldap.ActiveDirectoryLDAPUserStoreManager</code>	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.

## **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 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>wso2-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>
    <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>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.1</version>
            <inherited>true</inherited>
            <configuration>
                <encoding>UTF-8</encoding>
                <source>1.7</source>
                <target>1.7</target>
            </configuration>
        </plugin>
        <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>
            <extensions>true</extensions>
            <configuration>
                <instructions>
                    <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>
</instructions>
</configuration>
</plugin>
```

```

        </plugins>
    </build>
</project>

```

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.

```

<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.

```

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, profileManager,
realm, tenantId);
    log.info("CustomUserStoreManager initialized...");}
}

@Override
public boolean doAuthenticate(String userName, Object credential)
throws UserStoreException {
    boolean isAuthenticated = false;
    if (userName != null && credential != null) {
        try {
            String candidatePassword = (String) credential;

            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(String password, String saltValue)
throws UserStoreException {
    if (password != null) {
        // ignore saltValue for the time being
        log.info("Generating hash value using jasypt...\"");
        return passwordEncryptor.encryptPassword(password);
    } else {
        log.error("Password cannot be null");
        throw new UserStoreException("Authentication Failure");
    }
}
```

}

**Note:** The default constructor is not enough when you implement a custom user store manager, and you must implement a constructor with relevant arguments.

## 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 <PRODUCT\_HOME>/repository/conf/user-mgt.xml file:

```
<UserManager>
    <Realm>
        <Configuration>
            ...
            <AdminUser>
                <UserName>admin</UserName>
                <Password>admin</Password>
            </AdminUser>
            ...
        </Realm>
    </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](http://www.w3schools.com/xml/xml_cdata.asp)) as shown below or remove the character:

```
<Password>
<! [ CDATA[ xnvYh?@VHAkc?qZ%Jv855&A4a , %M8B@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](http://WSO2.com) has the fully-qualified name [daniel@WSO2.com](mailto: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`. 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:

Parameter	Description
UserNameAttribute	<p>Set the mail attribute of the user.</p> <pre>&lt;Property name="UserNameAttribute"&gt;mail&lt;/Property&gt;</pre>

UserNameSearchFilter	Use the mail attribute of the user instead of cn or uid.
UserNamesListFilter	Use the mail attribute of the user in the user name filter list as well.
UsernameJavaRegEx	Use the following email regex.
UserDNPattern	This parameter is used to speed up the LDAP search operations. You can comment out this parameter if you do not want to use it.
Realm configurations	The AdminUser username should use the email attribute of the admin user.

## 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](#).

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.
  1. Open the `cipher-tool.properties` file stored in the `<PRODUCT_HOME>/repository/conf/security` folder. The file should contain information about the files in which the passwords (that require encryption) are located as shown below:

```
<alias>=<file_name>//<xpath>,<true/false>
```

For example, if you want to encrypt the admin user password in the `user-mgt.xml` file, the following should be added to the `cipher-tool.properties` file:

```
UserManager.AdminUser.Password=repository/conf/user-mgt.xml//User
rManager/Realm/Configuration/AdminUser/Password,false
```

Either the relative path or the absolute path of each file starting from `<PRODUCT_HOME>` should be given. The last value that follows the file path is set to 'true' or 'false' indicating whether or not the value to be encrypted is an attribute.

By default, the file that is shipped with your product pack will contain information on the most common passwords that require encryption.

2. 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 plain text passwords (enclosed within square brackets) as shown below.

```
<alias>=[plain_text_password]
```

For example, if you want to encrypt the admin user password in the `user-mgt.xml` file, the following should be added to the `cipher-text.properties` file:

```
UserManager.AdminUser.Password=[admin]
```

By default, the `cipher-tool.properties` and `cipher-text.properties` files that are shipped with your product pack 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. For example, if you want to encrypt the password that is used to [connect to an LDAP user store](#) (configured in the `user-mgt.xml` file), add the following:

1. Add a new entry to the `cipher-tool.properties` file as shown below. Note that the `<alias>` value should be the same value that is hard coded in the relevant Carbon component. In this example (LDAP user store connection password), there are two possible alias values you can use as shown below.

- Using the `UserStoreManager.Property.ConnectionPassword` alias:

```
UserStoreManager.Property.ConnectionPassword=user-mgt.xml//UserManager/Realm/UserStoreManager/Property[@name='ConnectionPassword'],true
```

- Using the `UserManager.Configuration.Property.ConnectionPassword` alias:

```
UserManager.Configuration.Property.ConnectionPassword=user-mgt.xml//UserManager/Realm/UserStoreManager/Property[@name='ConnectionPassword'],true
```

2. Add a new entry to the `cipher-text.properties` file corresponding to the new entry in the `cipher-tool.properties` file:

```
//Use the alias corresponding to the value in the cipher-tool.properties file.
#UserStoreManager.Property.ConnectionPassword=[password]
#UserManager.Configuration.Property.ConnectionPassword=[password]
```

3. Save the information.
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 cipher tool script from the command prompt using the command relevant to your OS:
    - On Linux: ./ciphertool.sh -Dconfigure
    - On Windows: ./ciphertool.bat -Dconfigure
  4. The following message will be prompted: "[Please Enter Primary KeyStore Password of Carbon Server : ]". Enter the keystore password (which is "wso2carbon" for the default keystore) and proceed. If the script execution is successful, you will see the following message: "Secret Configurations are written to the property file successfully".

If you are using the cipher tool for the first time, the -Dconfigure command will first initialize the tool for your product. The tool will then start encrypting the plain text passwords you specified in the cipher-text.properties file.

Shown below is an example of an alias and the corresponding plain text password (in square brackets) in the cipher-text.properties file:

```
UserManager.AdminUser.Password=[admin]
```

If a password is not specified in the cipher-text.properties file for an alias, the user needs to provide it through the command line. Check whether the alias is a known password alias in Carbon configurations. If the tool modifies the configuration element and file, you must replace the configuration element with the alias name. Define a Secret Callback in the configuration file and add proper namespaces for defining the Secure Vault.

5. Now, to verify the password encryption:
  - Open the cipher-text.properties file and see that the plain text passwords are replaced by a cipher value.
  - Open the secret-conf.properties file from the <PRODUCT\_HOME>/repository/conf/security/ folder and see that the default configurations are changed.

#### ***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 appender is set to admin:

```
# LOGEVENT is set to be a LogEventAppender using a PatternLayout to send
logs to LOGEVENT
log4j.appenders.LOGEVENT=org.wso2.carbon.logging.service.appenders.LogEventA
ppender
log4j.appenders.LOGEVENT.url=tcp://localhost:7611
log4j.appenders.LOGEVENT.layout=org.wso2.carbon.utils.logging.TenantAwarePa
tternLayout
log4j.appenders.LOGEVENT.columnList=%T,%S,%A,%d,%c,%p,%m,%I,%Stacktrace
log4j.appenders.LOGEVENT.userName=admin
log4j.appenders.LOGEVENT.password=admin
log4j.appenders.LOGEVENT.processingLimit=1000
log4j.appenders.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:  
gaMpTzAccMScaH1lsZLXspml4HLI0M/srL5pB8jyknRKQ2zT7NuCvt1+qEkElRLgwlro  
hz31kuE0KFuapXrCSS5pxfGMOLn4/k7dNs2SlwbsG8C++/  
zfUuft1S16cqvDRM55fQwzCPfyb1713HvKu3oDaJ9VKgSbvHlQj6zqzg=
```

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.

```
log4j.appenders.LOGEVENT.password=cpw74SGeBNgAVpryqj5/xshSyW5BDW9d1UW0  
xMZ  
DxVeoa6RjyA1JRHutz4SfzfSgSzy2GQJ/2jQIw70IeT5EQEAR8XLGaqlsE5IlNoe9dhyL  
iPXEPRGq4k/BgUQD  
YiBg0nU7wRsR8YXrvf+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.

```
# LOGEVENT is set to be a LogEventAppender using a PatternLayout to  
send logs to LOGEVENT  
....  
log4j.appenders.LOGEVENT.password=secretAlias:log4j.appenders.LOGEVENT.  
password  
....
```

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 ([a](#) `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 =
amqp://admin:admin@clientID/carbon?brokerlist='tcp://localhost:5673'

# 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
```

Note the following if you have special characters in the passwords on your `jndi.properties` file:

- It is not possible to use the @ symbol in the username or password.
- It is also not possible to use the percentage (%) sign in the password. When building the connection URL, the URL is parsed. This parsing exception happens because the percentage (%) sign acts as the escape character in URL parsing. If using the percentage (%) sign in the connection string is required, use the respective encoding character for the percentage (%) sign in the connection string. For example, if you need to pass `adm%in` as the password, then the % symbol should be encoded with its respective URL encoding character. Therefore, you have to send it as `adm%25in`.

For a list of possible URL parsing patterns, see [URL encoding reference](#).

### ***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.

- 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***

- Start the server by running the product start up script from the <PRODUCT\_HOME>/bin/ directory as shown below.

```
./wso2server.sh
```

- 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.

- 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.

```
./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:

```
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.AdminUser.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 SecretRepository and 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,

```
public class HardCodedSecretCallbackHandler extends
AbstractSecretCallbackHandler {
    protected void handleSingleSecretCallback(SingleSecretCallback
singleSecretCallback) {
        singleSecretCallback.setSecret("password");
    }
}
```

2. We can set multiple password-based as follows:

```
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>wso2carbon</username>

            <passwordsvns:secretAlias="Datasources.WSO2_CARBON_DB.Configuration.P
assword">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

The transport level security protocol of the Tomcat server is configured in the `<PRODUCT_HOME>/conf/tomcat/catalina-server.xml` file. Note that the `sslProtocol` attribute is set to "TLS" by default. See the following topics for detailed configuration options:

- Disabling SSL
- Disabling weak ciphers
- Configuring the PassThrough transport

### Disabling SSL

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 SSL protocol for your server.

Follow the steps given below to disable SSL support on WSO2 Carbon based servers.

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 Carbon 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.

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9443"
           bindOnInit="false"
           sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"
```

4. Start the server.

In some Carbon products, such as WSO2 ESB and WSO2 API Manager, pass-thru transports are enabled. Therefore, to disable SSL in such products, the axis2.xml file stored in the <PRODUCT\_HOME>/repository/conf/axis2/ directory should also be configured.

To test if SSL is disabled:

1. Download TestSSLServer.jar from [here](#).
2. Execute the following command to test the transport:

```
java -jar TestSSLServer.jar localhost 9443
```

3. 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_AES_128_CBC_SHA
DHE_RSA_WITH_AES_128_CBC_SHA
RSA_WITH_AES_256_CBC_SHA
DHE_RSA_WITH_AES_256_CBC_SHA

```

## 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.

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 `cipher` 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_A
ES_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"
```

For Tomcat version 7.0.59 and JDK version 1.8:

```
ciphers="TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_A
ES_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_ECDH
E_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_128_GCM_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.

Firefox 39.0 onwards 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.

**Tip:** To use AES-256, the Java JCE Unlimited Strength Jurisdiction Policy files need to be installed. Downloaded them from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

**Tip:** 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 that this tip is not applicable when you are disabling weak ciphers in [WSO2 Identity Server](#).

## Configuring the PassThrough transport

If you have enabled the PassThrough transport, do the following:

This is applicable for WSO2 Enterprise Service Bus and WSO2 API Manager products.

1. Stop the server.
2. Open the <ESB\_HOME>/repository/conf/axis2/axis2.xml file and based on the JDK version you are using 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.PassThroughHttpSSLListener"> element.
  - If you are using JDK 1.6, add the following parameter:
 

```
<parameter name="HttpsProtocols">TLSv1</parameter>
```
  - If you are using JDK 1.7, add the following parameter:
 

```
<parameter
name="HttpsProtocols">TLSv1,TLSv1.1,TLSv1.2</parameter>
```
3. Start the server.
4. Test the pass-through transport using the following command with the corresponding port:
 

```
$ java -jar TestSSLServer.jar localhost 8243
```

## 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.

The steps below show how to enable the Java Security Manager for WSO2 products.

**Before you begin**, ensure that you have Java 1.8 installed.

1. Download the WSO2 product to any location (e.g., <HOME>/user/<product-pack> folder).
2. To sign the JARs in your product, you need a key. Generate it using the `keytool` command as follows:

```

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)

```

The default keystore of the WSO2 products is `wso2carbon.jks`, which is in the `<PRODUCT_HOME>/repository/resources/security` folder. It is used for signing JARs.

- Import the `signFiles` public key certificate that you created earlier to `wso2carbon.jks`. The sample below shows the security policy file referring the signer certificate from the `wso2carbon.jks` file:

```

$ keytool -export -keystore signkeystore.jks -alias signFiles -file
sign-cert.cer

$ keytool -import -alias signFiles -file sign-cert.cer -keystore
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:
MD5: 54:13:FD:06:6F:C9:A6:BC:EE:DF:73:A9:88:CC:02:EC
SHA1: AE:37:2A:9E:66:86:12:68:28:88:12:A0:85:50:B1:D1:21:BD:49:52
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.

- Update the "grant signedBy" value in the security policy file with the signed alias key. See the following sample security policy file:

```

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

```
#!/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

```
#!/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`.

- Add the following system properties to the startup script and save the file:

```
-Djava.security.manager=org.wso2.carbon.bootstrap.CarbonSecurityManager \
-Djava.security.policy=$CARBON_HOME/repository/conf/sec.policy \
-Drestricted.packages=sun.,com.sun.xml.internal.ws.,com.sun.xml.internal.bind.,com.sun.imageio.,org.wso2.carbon. \
-Ddenied.system.properties=javax.net.ssl.trustStore,javax.net.ssl.trustStorePassword,denied.system.properties \
```

- 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.

```
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
- Managing keystores

### 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>/r

pository/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 key stores 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.
- 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 Encipherment** 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 Encipherment** 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.

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](#).

### Managing keystores

WSO2 products provide the facility to add keystores using the Management Console or using an XML configuration, and to import certificates to the keystore using the Management Console. The WSO2 keystore management feature provides a UI and an API to add and manage keystores. For example, when you apply WS-Security to Web services using the Management Console, you can select a keystore for encryption/signing processes from the uploaded keystores. The Management Console also allows you to view/delete keystores.

All the functions of keystore management are exposed via APIs. As a result, if you are writing a custom extension to a WSO2 product (e.g., for WSO2 ESB mediators), you can directly access the configured keystores using the API. The API hides the underlying complexity, allowing you to easily use it in third-party applications to manage their keystores as well.

This functionality is bundled with the following feature that is installed in your product.

<b>Name:</b>	WSO2 Carbon	-	Security	Management	Feature
<b>Identifier:</b>	org.wso2.carbon.security.mgt.feature.group				

### 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, for communication 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.

**Before you start** creating new keystores and replacing the default keystore configurations with new ones, be sure to go through the [recommendations for setting up keystores in WSO2 products](#).

Let's start creating a new keystore:

- Creating a keystore using an existing certificate

- Creating a keystore using a new certificate
  - Step 1: Creating keystore with private key and public certificate
  - Step 2: Creating CA-signed certificates for public key
  - Step 3: Importing CA-signed certificates to keystore
- Adding a public key to client-truststore.jks
- What's next?

If you are creating a new keystore for data encryption, be sure to acquire a public key certificate that contains the **Data Encipherment** key usage. See the [keystore recommendations](#) for more information.

#### ***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 thekeystoreand 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***

If there are no certificates signed by a Certification Authority, you can follow the steps in this section to create a keystore for SSL using a new certificate. We will be using the keytool that is available with your JDK installation.

#### **Step 1: Creating keystore with private key and public certificate**

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.

## Step 2: Creating CA-signed certificates for public key

Now we have a .jks file. This keystore (.jks file) can be used to generate a certificate signing request (CSR). This CSR file must be certified by a certificate authority or certification authority (CA), which is an entity that issues digital certificates. These certificates can 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.sampledomain.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.

- 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: COMODORSAAAddTrustCA.crt , COMODORSADomainValidationSecureServerCA.crt
SSL Certificate signed by CA: test_sampleapp_org.crt
```

### Step 3: Importing CA-signed certificates to keystore

- 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 COMODORSAAAddTrustCA.crt -keystore newkeystore.jks -storepass mypassword
keytool -import -v -trustcacerts -alias SecureServerCA -file COMODORSADomainValidationSecureServerCA.crt -keystore newkeystore.jks -storepass mypassword
```

Optionally we can append the `-storepass <keystore password>` option to avoid having to enter the password when prompted later in the interactive mode.

- 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
```

In this command, use the same alias (i.e., 'newcert') that you used while creating the keystore

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.

#### ***Adding a public key to client-truststore.jks***

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/`). In case we need to import a public key certificate into the trust store, you can achieve this using keytool as explained below. In this example, we are importing the same CA-signed public key certificate (which we obtained above) into `client-truststore.jks`.

Note that we are using the default `client-truststore.jks` file in your WSO2 product as the trust store in this example.

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](#).

## **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.

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:

- Configuring the primary keystore (for internal data encryption)
- Configuring a secondary keystore (for SSL connections)
- Configuring a keystore for Java permissions
- Configuring keystores for WS-Security
- What's next?

#### ***Configuring the primary keystore (for internal data encryption)***

Encrypting administrator passwords as well as other confidential, internal information that are maintained in various product-level and product feature-level configurations/configuration files can be classified as internal data encryption. The Keystore element in the carbon.xml file, stored in the <PRODUCT\_HOME>/repository/conf/ directory should be updated with details of the keystore serving this purpose.

The default configuration is shown below.

```

<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 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.

```

keystoreFile="${carbon.home}/repository/resources/security/wso2carbon.jks"
keystorePass="wso2carbon"

```

#### **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.

#### **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**

If the WSO2 Security Management feature is installed in your product, you can manage the keystores using the Management Console. In order to do this, 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.

### **Adding keystores**

Keystores allow you to manage the keys that are stored in a database. WSO2 Carbon keystore management provides the ability to manage multiple keystores. Follow the instructions below to add a new keystore to your product using the Management Console.

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. Click the **Add New Key store** link to open the following screen:

## Add New KeyStore

### Step 1: Upload KeyStore File

**KeyStore File**

KeyStore File*	<input type="button" value="Choose File"/> no file selected
KeyStore Password*	*****
Provider	admin
KeyStore Type	JKS

**Next >** **Cancel**

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 on the next page, provide the **Private Key Password**.
7. Click **Finish** to add the new keystore to the list.

### Viewing keystores

Keystores allow you to manage the keys that are stored in a database. WSO2 Carbon keystore management provides the ability to view keystores using the Management Console. Follow the instructions below to view a keystore.

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 as follows:

## KeyStore Management

**Search**

Enter Keystore name pattern (* for all)		
*	<input type="button" value="Search"/>	
Name	Type	Actions
wso2carbon.jks	JKS	<a href="#"></a> <a href="#"></a> <a href="#"></a>

[Add New KeyStore](#)

4. Click **View** in the list of actions. The **View Key Store** screen shows information about the available certificates.

#### Available Certificates

Search							
Enter Certificate alias pattern (* for all)							
Alias	IssuerDN	NotAfter	NotBefore	SerialNumber	SubjectDN	Version	Actions
wso2carbon.cert	CN=wso2carbon, OU=None, L=Seattle, ST=Washington, O=WSO2, C=LK	25/09/2282	11/12/2008	1228997318	CN=wso2carbon, OU=None, L=Seattle, ST=Washington, O=WSO2, C=LK	1	
entrustclientca	CN=Entrust.net Client Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/Client_CA_Info/CPS incorp. by ref. limits lab., O=Entrust.net, C=US	13/10/2019	13/10/1999	939758062	CN=Entrust.net Client Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/Client_CA_Info/CPS incorp. by ref. limits lab., O=Entrust.net, C=US	3	
verisignclass3g2ca	OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 3 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US	02/08/2028	18/05/1998	167285380242319648451154478808036881606	OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 3 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US	1	
thawtepersonalbasicca	EMAILADDRESS=personal-basic@thawte.com, CN=Thawte Personal Basic CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	01/01/2021	01/01/1996	0	EMAILADDRESS=personal-basic@thawte.com, CN=Thawte Personal Basic CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	3	
verisignclass2g3ca	CN=VeriSign Class 2 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US	17/07/2036	01/10/1999	129520775995541613599859419027715677050	CN=VeriSign Class 2 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc. - For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US	1	
thawtepersonalpremiumca	EMAILADDRESS=personal-premium@thawte.com, CN=Thawte Personal Premium CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	01/01/2021	01/01/1996	0	EMAILADDRESS=personal-premium@thawte.com, CN=Thawte Personal Premium CA, OU=Certification Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA	3	
valicertclass2ca	EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValCert Class 2 Policy Validation Authority, O=ValCert, Inc., L=ValCert Validation Network	26/06/2019	26/06/1999	1	EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValCert Class 2 Policy Validation Authority, O=ValCert, Inc., L=ValCert Validation Network	1	
entrustssica	CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/CPS incorp. by ref. (limits lab.), O=Entrust.net, C=US	25/05/2019	25/05/1999	927650371	CN=Entrust.net Secure Server Certification Authority, OU=(c) 1999 Entrust.net Limited, OU=www.entrust.net/CPS incorp. by ref. (limits lab.), O=Entrust.net, C=US	3	
equifaxsecureebusinessca2	OU=Equifax Secure eBusiness CA-2, O=Equifax Secure, C=US	23/06/2019	23/06/1999	930140085	OU=Equifax Secure eBusiness CA-2, O=Equifax Secure, C=US	3	
equifaxsecureebusinessca1	CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US	21/06/2020	21/06/1999	4	CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US	3	

<< first <Prev 1 2 3 ... Next > last >

It also displays information about private key certificates:

#### Certificate of the Private Key

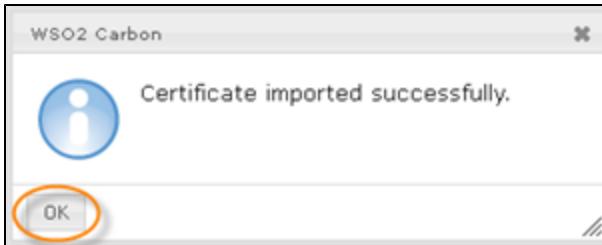
Alias	IssuerDN	NotAfter	NotBefore	SerialNumber	SubjectDN	Version
wso2carbon	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	13/02/2035	19/02/2010	1266562946	CN=localhost, O=WSO2, L=Mountain View, ST=CA, C=US	3
<a href="#">Import Cert</a> <a href="#">Finish</a>						

5. Click **Finish** to go back to the **Key Store Management** screen.

#### Importing certificates to keystore

Keystores allow you to manage the keys that are stored in a database. WSO2 Carbon keystore management provides the ability to import certificates for keystores. Follow the instructions below to import a certificate for a keystore.

1. Log in to the WSO2 product with your user name and password.
2. Go to the **Configure** tab and click **Keystores**.
3. The **Keystore Management** page appears. All the keystores that are currently added to the product will be listed here as follows:
4. Click **Import Cert** associated with the keystore for which you want to import a certificate.
5. The available certificates are already listed on the **Import Certificates** screen. Click **Browse** to find the location of the new certificate that you want to import.
6. Once you have selected the certificate, click **Import**.
7. Once a certificate is imported successfully, you will see the following confirmation:



Click **OK**.

8. The imported certificate appears in the list of **Available Certificates**. In the example shown below, the "GeoTrust\_Global\_CA" certificate was imported.

verisignclass3ca
entrustgsslc
geotrustglobalca
verisignclass1g2ca
mycert

## 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:

```
<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:

```
symmetric.key=samplekeyvalue
```

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=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 CSRGGuard](#) 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 CSRGGuard to mitigate CSRF attacks in applications hosted on the WSO2 platform. Fine-tuned configuration values of CSRGGuard increases security, based on the security requirements of the specific application.

CSRGGuard 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.

You can protect HTTP GET requests sent as a result of resource inclusions and links can be appended a relevant token in the "href" or "src" attributes. 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.

See the following for instructions on manually updating CSRF configurations in WSO2 products:

- Securing web applications
- Securing Jaggery applications

#### ***Securing web applications***

Follow the steps below to secure web applications.

1. Add the following configurations in the `web.xml` file of your application.

```

<!-- OWASP CSRFGuard context listener used to read CSRF configuration
-->
<listener>

<listener-class>org.owasp.csrfguard.CsrfGuardServletContextListener</
listener-class>
</listener>
<!-- OWASP CSRFGuard session listener used to generate per-session
CSRF token -->
<listener>

<listener-class>org.owasp.csrfguard.CsrfGuardHttpSessionListener</lis
tener-class>
</listener>
<!-- OWASP CSRFGuard per-application configuration property file
location-->
<context-param>
    <param-name>Owasp.CsrfGuard.Config</param-name>

    <param-value>/repository/conf/security/Owasp.CsrfGuard.properties</pa
ram-value>
</context-param>
<!-- OWASP CSRFGuard filter used to validate CSRF token-->
<filter>
    <filter-name>CSRFGuard</filter-name>
    <filter-class>org.owasp.csrfguard.CsrfGuardFilter</filter-class>
</filter>
<!-- OWASP CSRFGuard filter mapping used to validate CSRF token-->
<filter-mapping>
    <filter-name>CSRFGuard</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- OWASP CSRFGuard servlet that serves dynamic token injection
JavaScript (application can customize the URL pattern as required)-->
<servlet>
    <servlet-name>JavaScriptServlet</servlet-name>

    <servlet-class>org.owasp.csrfguard.servlet.JavaScriptServlet</servlet
-class>
</servlet>
<servlet-mapping>
    <servlet-name>JavaScriptServlet</servlet-name>
    <url-pattern>/csrf.js</url-pattern>
</servlet-mapping>

```

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>
  <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;

```

org.owasp.csrfguard.unprotected.Default=%servletContext%/exampleAction
org.owasp.csrfguard.unprotected.Default_1=%servletContext%/exampleAct
ion
org.owasp.csrfguard.unprotected.Example=%servletContext%/exampleActio
n/*
org.owasp.csrfguard.unprotected.ExampleRegEx=^%servletContext%/.+Publ
ic\.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.

Property	Description
org.owasp.csrfguard.PRNG=SHA1PRNG	Defines the hash algorithm used to generate CSRF tokens.
org.owasp.csrfguard.TokenLength=32	Defines the length of the generated CSRF tokens.

org.owasp.csrfguard.action.Invalidate=org.owasp.csrfguard.action.Invalidate	Inva the u sess CSR atter bloc CSR
-----------------------------------------------------------------------------	------------------------------------------------------

## **Securing Jaggery applications**

Follow the steps below to secure Jaggery applications.

1. Add the following configurations in the `jaggery.conf` file of your application.

```
"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>
  <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;

```

org.owasp.csrfguard.unprotected.Default=%servletContext%/exampleAction
org.owasp.csrfguard.unprotected.Default_1=%servletContext%/exampleActi
on
org.owasp.csrfguard.unprotected.Example=%servletContext%/exampleActio
n/*
org.owasp.csrfguard.unprotected.ExampleRegEx=^%servletContext%/.*Publ
ic\.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.

Property	Description
org.owasp.csrfguard.PRNG=SHA1PRNG	Defines the hash algorithm used for generating CSRF tokens.
org.owasp.csrfguard.TokenLength=32	Defines the length of the generated CSRF tokens.

org.owasp.csrfguard.action.Invalidate=org.owasp.csrfguard.action.Invalidate	Inva the u sess CSR atter block CSR
-----------------------------------------------------------------------------	-------------------------------------------------------

## Enabling HostName Verification

Hostname verification is enabled in WSO2 products by default, which means that when a hostname is being accessed by a particular client, it will be verified against the hostname specified in the product's SSL certificate.

Carbon 4.4.10 introduced a new property (`httpclient.hostnameVerifier`) for the purpose of configuring the default hostname verification setting in a Carbon server. Therefore this possibility is available for all WSO2 products that are based on Carbon Kernel 4.4.10 or later versions. You can add this property to the product's startup script (`wso2server.sh` for Linux and `wso2server.bat` for Windows), which is stored in the `<PRODUCT_HOME>/bin` directory and specify a value as shown below. The property will be effective during server startup.

```
-Dhttpclient.hostnameVerifier=<property_value>
```

The values you can use with this property are explained below. If none of these values are specified, the default mode will be effective:

- **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.
- **DefaultAndlocalhost:** This option works the same as the default mode, except that the following hostnames **will not** be verified against the hostname in the certificate: "`localhost`", "`localhost.localdomain`", "`127.0.0.1`", "`::1`". That is, these hostnames will be allowed regardless of the server's certificate.

Note that the above values will behave the same as synapse hostname verification options.

## 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 `RemotelpValve` 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 `RemotelpValve` 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 `RemotelpValve` for Carbon servers

You can enable Tomcat's `RemotelpValve` for your Carbon server by simply adding the valve to the `catalina-server.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.

```
<Host name="localhost" unpackWARs="true" deployOnStartup="false"
autoDeploy="false"
appBase="\${carbon.home}/repository/deployment/server/webapps/">
.....
<Valve className="/" />
</Host>
```

## Carbon Applications

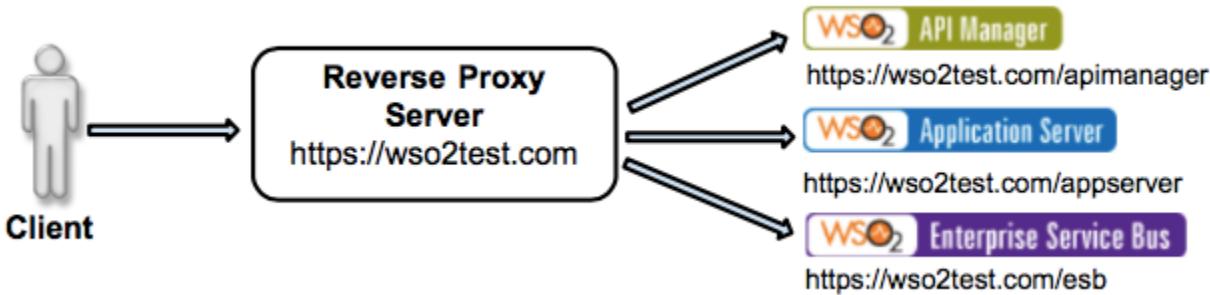
See the following topics for information on how to set up and configure various components in WSO2 products:

- 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
- 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:



#### Proxy URLs mapped to Carbon server URLs:

- `https://10.100.1.1:<ListeningPort-apimanager>/carbon` mapped to `https://wso2test.com/apimanager`.
- `https://10.100.1.1:<ListeningPort-esb>/carbon` mapped to `https://wso2test.com/esb`.
- `https://10.100.1.1:<ListeningPort-appserver>/carbon` mapped to `https://wso2test.com/appserver`.

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://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:

```
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:

```
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.

```
sudo openssl genrsa -des3 -out server.key 1024
```

5. Next, create the certificate signing request as shown below.

```
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:

```
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:

```
sudo cp /etc/nginx/sites-available/default
/etc/nginx/sites-available/wso2
```

8. Now, create a symbolic 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/
https://wso2test.com:8243/appserver/;
        proxy_cookie_path / /appserver;
    }

    #with portOffset 10 running ESB
```

```
location /esb/ {
    proxy_pass https://wso2test.com:9453/;
    proxy_redirect https://wso2test.com:8243/
https://wso2test.com:8243/esb/;
    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/
http://wso2test.com:8280/appserver/;
        proxy_cookie_path / /appserver;
    }

    #with portOffset 10 running ESB
    location /esb/ {
        proxy_pass http://wso2test.com:9773/;
        proxy_redirect http://wso2test.com:8280/
http://wso2test.com:8280/esb/;
```

```
    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:

```
sudo service nginx restart
```

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 `.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>

- 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:

```

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9763"
           proxyPort="8280"
           redirectPort="9443"
           bindOnInit="false"
           maxHttpHeaderSize="8192"
           acceptorThreadCount="2"
           maxThreads="250"
           minSpareThreads="50"
           disableUploadTimeout="false"
           connectionUploadTimeout="120000"
           maxKeepAliveRequests="200"
           acceptCount="200"
           server="WSO2 Carbon Server"
           compression="on"
           compressionMinSize="2048"
           noCompressionUserAgents="ozilla, 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"
-->
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9443"
           proxyPort="8243"
           bindOnInit="false"
           sslProtocol="TLS"
           maxHttpHeaderSize="8192"

```

### 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.
  - For "/appserver" proxy path: <https://wso2test.com:8243/appserver/carbon/admin/index.jsp>.
  - For "/esb" proxy path: <https://wso2test.com:8243/esb/carbon/admin/index.jsp>.

## 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
    - Integration runtime ports
    - EI-Analytics runtime ports
    - EI-Business Process runtime ports
    - EI-Broker runtime 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 (EI) uses the following ports to access the management console:

- 8243 - HTTPS servlet transport for the **Integrator** runtime (the default URL of the management console is [http://localhost:8243/carbon](https://localhost:8243/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 - RMIServer port. Used to monitor Carbon remotely
- 9999 - RMIServer port. Used along with the RMIServer port when Carbon is monitored from a JMX client that is behind a firewall

### Clustering ports

To cluster any running Carbon instance, either one of the following ports must be opened.

- 45564 - Opened if the membership scheme is multicast
- 4000 - Opened if the membership scheme is wka

### Random ports

Certain ports are randomly opened during server startup. This is due to specific properties and configurations that become effective when the product is started. Note that the IDs of these random ports will change every time the server is started.

- A random TCP port will open at server startup because of the `-Dcom.sun.management.jmxremote` property set in the server startup script. This property is used for the JMX monitoring facility in JVM.
- A random UDP port is opened at server startup due to the log4j appender (`SyslogAppender`), which is configured in the <PRODUCT\_HOME>/repository/conf/log4j.properties file.

### Product-specific ports

Some 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**

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to BAM/CEP: stat pub

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](#).

- **P o r t s              a v a i l a b l e              f o r              a l l              m o d e s**  
The following ports are available for all three modes explained above.

Description	Port number
spark.ui.port	4040
spark.history.ui.port	18080
spark.blockManager.port	12000
spark.broadcast.port	12500
spark.driver.port	13000

spark.executor.port	13500
spark.fileservice.port	14000
spark.repliclassserver.port	14500

- **Ports available for the cluster mode**  
The following ports are available only for the cluster mode.

Description	Port number
spark.master.port	7077
spark.master.rest.port	6066
spark.master.webui.port	8081
spark.worker.port	11000
spark.worker.webui.port	11500

#### ***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***

##### ***Integration runtime ports***

- 8243 - HTTPS servlet transport (the default URL of the management console is <https://localhost:8243/carbon>)

Non-blocking HTTP/S transport ports: Used to accept message mediation requests. If you want to send a request to an API or a proxy service for example, you must use these ports: <EI\_HOME>/conf/axis2/axis2.xml file.

- 8243 - Passthrough or NIO HTTPS transport
- 8280 - Passthrough or NIO HTTP transport

#### ***EI-Analytics runtime ports***

- 9444 - Management console port

- 9161 - Cassandra port on which Thrift listens to clients
- 7712 - Thrift SSL port for secure transport, where the client is authenticated to DAS
- 7612 - Thrift TCP port to receive events from clients to DAS

#### ***EI-Business Process runtime ports***

- 9445 - HTTPS servlet transport (the default URL of the management console is <https://localhost:9445/carbon>)
- 9765 - HTTP servlet transport

#### ***EI-Broker runtime ports***

- 9446 - HTTPS servlet transport (the default URL of the management console is <https://localhost:9446/carbon>)
- 9766 - HTTP servlet transport

EI-Broker uses the following JMS ports to communicate with external clients over the JMS transport.

- 5675 - Port for listening for messages on TCP when the AMQP transport is used.
- 8675 - Port for listening for messages on TCP/SSL when the AMQP Transport is used.
- 1886 - Port for listening for messages on TCP when the MQTT transport is used.
- 8836 - Port for listening for messages on TCP/SSL when the MQTT Transport is used.
- 7614 - The port for Apache Thrift Server.

#### ***Identity Server***

- 8000 - KDCServerPort. Port which KDC (Kerberos Key Distribution Center) server runs
- 10500 - ThriftEntitlementReceivePort

#### ***Message Broker***

Message Broker uses the following JMS ports to communicate with external clients over the JMS transport.

- 5672 - Port for listening for messages on TCP when the AMQP transport is used.
- 8672 - Port for listening for messages on TCP/SSL when the AMQP Transport is used.
- 1883 - Port for listening for messages on TCP when the MQTT transport is used.
- 8833 - Port for listening for messages on TCP/SSL when the MQTT Transport is used.
- 7611 - The port for Apache Thrift Server.

#### ***Machine Learner***

- 7077 - The default port for Apache Spark.
- 54321 - The default port for H2O.
- 4040 - The default port for Spark UI.

#### ***Storage Server***

Cassandra:

- 7000 - For Inter node communication within cluster nodes
- 7001 - For inter node communication within cluster nodes via SSL
- 9160 - For Thrift client connections
- 7199 - For JMX

HDFS:

- 54310 - Port used to connect to the default file system.
- 54311 - Port used by the MapRed job tracker
- 50470 - Name node secure HTTP server port
- 50475 - Data node secure HTTP server port
- 50010 - Data node server port for data transferring
- 50075 - Data node HTTP server port

- 50020 - Data node IPC server port

### **Enterprise Mobility Manager**

The following ports need to be opened for Android and iOS devices so that it can connect to Google Cloud Messaging (GCM)/Firebase Cloud Messaging (FCM) and APNS (Apple Push Notification Service), and enroll to WSO2 EMM.

A n d r o i d :

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**

<b>8243</b>	HTTPS gateway port.
<b>8280</b>	HTTP gateway port.
<b>1886</b>	Default MQTT port.
<b>9445</b>	HTTPS port for the analytics profile.
<b>9765</b>	HTTP port for the analytics profile.
<b>1039</b>	HTTP port for the analytics profile

### **Ports required for mobile devices to communicate with the server and the respective notification servers.**

	<b>Android</b>
--	----------------

<b>5228</b>	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.
<b>5229</b>	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.
<b>5230</b>	
	<b>iOS</b>
<b>5223</b>	Transmission Control Protocol (TCP) port used by devices to communicate to APNs servers.
<b>2195</b>	TCP port used to send notifications to APNs.
<b>2196</b>	TCP port used by the APNs feedback service.
<b>443</b>	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/\(Carbon-Release-Version\)](http://dist.wso2.org/p2/carbon/releases/(Carbon-Release-Version)). Learn how to add a repository to the Identity Server [here](#).
2. Search for the word "authenticator". Select "SAML2 based Single Sign-On authenticator" from the result and click "Install." See [Installing Features](#).

### ***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 to setup.

```

<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/samlss</Param
eter>
        <Parameter
name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspec
ified</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. It should have this format: [https://\(host-name\):\(port\)/samlss](https://(host-name):(port)/samlss).
- Parameter NameIDPolicyFormat - This specifies the name identifier format that the Carbon server wants to receive in the subject of an assertion from a particular identity provider.
- Parameter IdPCertAlias - This is uncommented by default. This is the alias of the identity provider certificate. This is specifically used whenever a Carbon server uses IS as the identity provider. The configuration needs to be done at the relying party server's <PRODUCT\_HOME>/repository/conf/secur
ity/authenticators.xml file.

### **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](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.

Local & Outbound Authentication Configuration

Authentication Type:  
 Default  
 Local Authentication  
 Federated Authentication  
 Advanced Configuration

Assert identity using mapped local subject identifier  
 Always send back the authenticated list of identity providers  
 Use tenant domain in local subject identifier  
 Use user store domain in local subject identifier

Request Path Authentication Configuration

8. Click the **Update** button to update the details of the service provider.

## Configuring the Task Scheduling Component

This **Task Scheduling** component allows you to define specific tasks that can be invoked periodically. This functionality is used by the following WSO2 products such as WSO2 Enterprise Integrator, WSO2 Enterprise Service Bus and WSO2 Data Services Server.

Follow the instructions given on this page to configure and set up this component for your server. You can find details on how to use this component in the respective product documentation.

The task scheduling component is configured in the `tasks-config.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/etc/` directory). The default values in the `tasks-config.xml` file ensures that minimal changes are required when running in both standalone and clustered modes. Given below are the settings that you can configure in this file.

- Step 1: Setting the task server mode
- Step 2: Configuring a clustered task server

### Step 1: Setting the task server mode

You can set the task server mode by using 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** task handling mode.

- **STANDALONE:** This mode is used when the Carbon server is used as a single installation. That is, tasks will be managed locally within the server.
- **CLUSTERED:** This mode is used when a cluster of Carbon 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 clustered task server**

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:

- `<taskServerCount>`: This value specifies the number of nodes in the server cluster, which basically represents the number of servers that will share the task scheduling. Task scheduling will only begin 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 servers in the cluster. When individual servers become active, we do not want the first active server to schedule all the tasks. Instead, we will want all five servers to become active and share the ten tasks between them.

The task server count is set to "2" by default, which indicates that at least two nodes will be there in a clustered setup.

- `<defaultLocationResolver>`: The default location resolver basically controls how the scheduled tasks are allocated among multiple nodes of a cluster. The possible options are as follows:
  - `RoundRobinTaskLocationResolver`: Cluster nodes are selected on a round robin basis and the tasks are allocated.
  - `RandomTaskLocationResolver`: Cluster nodes are randomly selected and the tasks are allocated.
  - `RuleBasedLocationResolver`: This allows you to set a 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 values.

For example, see the details of the `RuleBasedLocationResolver` configuration given below.

```

<defaultLocationResolver>

    <locationResolverClass>org.wso2.carbon.ntask.core.impl.RuleBased
    LocationResolver</locationResolverClass>
        <properties>
            <property
name="rule-1">HIVE_TASK,HTTP_SCRIPT*,192.168.1.*</property>
            <property
name="rule-2">HIVE_TASK,.* ,192.168.2.*</property>
            <property name="rule-5">.* ,.* ,.*</property>
        </properties>
    </defaultLocationResolver>

```

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 order; i.e., 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.

The `RuleBasedLocationResolver` allows you to address scenarios where tasks are required to be executed in specific server nodes first. Then, it can fail-over to another set of server nodes if the first (preferred) one is not available.

## 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 throwable 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. Create a Maven project using your IDE.
2. 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.

3. Create a new HTML error page (e.g. `new_error_404.html`) as shown below. This contains the customized error page.

```

<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
charset=ISO-8859-1">
    <title>404 - Error not found</title>
  </head>
  <body>
    <h1>Sorry, this resource is not found.</h1>
  </body>
</html>

```

4. Add the `new_error_404.html` file inside the `<PROJECT_HOME>/src/main/resources/web` directory.
5. Add the following property below the `<version>` element in the `<PROJECT_HOME>/pom.xml` file: `<packaging>bundle</packaging>`

6. Add the following configurations inside the `<plugins>` element in the `<PROJECT_HOME>/pom.xml` file.

```

<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>

            <Private-Package>org.vanji.styles.internal</Private-Package>
            <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>

```

7. Add the following configurations inside the `<dependencies>` element in the `<PROJECT_HOME>/pom.xml` file:

```

<dependency>
    <groupId>org.apache.felix</groupId>
    <artifactId>org.apache.felix.framework</artifactId>
    <version>1.0.3</version>
</dependency>

```

8. Build the Maven project by executing the following command: `mvn clean install`
9. Once the project is built, copy the JAR file (from the `<PROJECT_HOME>/target/` directory) to the `<CARBON_HOME>/repository/components/dropins/` directory.
10. Change the following configurations in the `<CARBON_HOME>/repository/conf/tomcat/carbon/WEB-INF/web.xml` file.

```

<error-page>
    <error-code>404</error-code>
    <location>/carbon/errors/new_error_404.html</location>
</error-page>

```

You need to replicate this configuration, and change the values of the <error-code> and <location> elements accordingly for each new HTML error page you add.

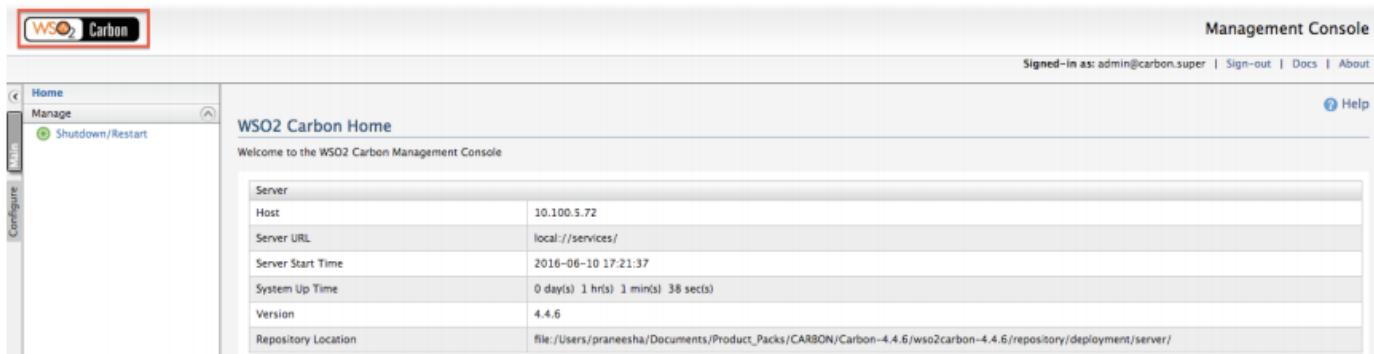
11. Restart the WSO2 Carbon server.
12. Access the following URL again, to test the error page you customized: <https://10.100.5.72:9443/abc>. You will view the new error page with the following content: "Sorry, this resource is not found."

## Customizing the Management Console

Some of the WSO2 products consist of a web user interface named the Management Console. It allows administrators to configure, monitor, tune, and maintain WSO2 products. The components that formulate the design and style of the Management Console are defined in resource (JAR) files. You can customize the Management Console by modifying these resource files. You need to create a fragment bundle for the original resource file that specifies a required bundle for it. 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 Management Console. 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: <https://10.100.5.72:9443/carbon/>



Follow the steps below to change this logo.

1. Find the <PRODUCT\_HOME>/repository/component/patches/org.wso2.carbon.ui\_4.4.6.jar file, and extract it.

Find the bundle that contains the resource files you need to modify. In this case, the logo and the related CSS files are contained in the org.wso2.carbon.ui\_4.4.6.jar file.

2. Create a new Maven project using your IDE (e.g. org.wso2.carbon.ui\_4.4.6\_patch).

This creates a project for a new bundle to pack the files you modify. Since the symbolic name of the original bundle is 'org.wso2.carbon.ui', this patch bundle should have the symbolic name as part of its name (e.g. 'org.wso2.carbon.ui\_4.4.6\_patch'). Also, you need to use this name with the symbolic name as part of it, as the required bundle in the fragment bundle, which you will create later.

3. Add the following content to the pom.xml file of the org.wso2.carbon.ui\_4.4.6\_patch project.

```

<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_4.4.6_patch</artifactId>
    <version>1.0.0</version>
    <packaging>bundle</packaging>

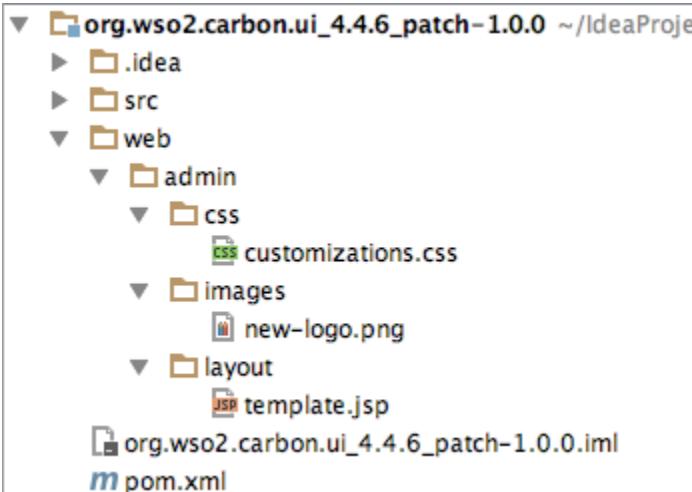
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-bundle-plugin</artifactId>
                <version>3.0.1</version>
                <extensions>true</extensions>
                <configuration>
                    <instructions>

                        <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 the web/, admin/, css/, images/ and layout/ directory locations as they were in the original bundle as shown in the file structure below.



5. Create a new CSS file (e.g. customizations.css) with the following content.

This file includes the logo customization styles.

```
#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;
}
```

6. Add the `customizations.css` file to the `<org.wso2.carbon.ui_4.4.6_patch>/web/admin/css/` directory.

This file includes the logo customization styles.

7. Copy the content of the `<org.wso2.carbon.ui_4.4.6.jar>/ web/admin/layout/template.jsp` file to the `<org.wso2.carbon.ui_4.4.6_patch>/web/admin/layout/template.jsp` file.
8. Add the following line to the `<org.wso2.carbon.ui_4.4.6_patch> web/admin/layout/template.jsp` file:

```
<link href="../../admin/css/customizations.css" rel="stylesheet"
      type="text/css" media="all"/>
```

9. Add the below image as the new logo (e.g. `new-logo.png`) to the `<org.wso2.carbon.ui_4.4.6_patch>/ web/admin/images/` directory.



**Lorem Ipsum Online**

10. Create another new Maven project using your IDE (e.g. `org.wso2.carbon.ui_4.4.6_fragment`).

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_4.4.6_fragment`). This fragment bundle will not contain anything (expect the `pom.xml` file) when it is built.

11. Add the following content to the `pom.xml` file of the `org.wso2.carbon.ui_4.4.6_fragment` project.

This pom.xml file of the fragment bundle define properties including the required bundle value (i.e. 'org.wso2.carbon.ui\_4.4.6\_patch').

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xs="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_4.4.6_fragment</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>

<Require-Bundle>org.wso2.carbon.ui_4.4.6_patch</Require-Bundle>

<Fragment-Host>org.wso2.carbon.ui</Fragment-Host>
                    </instructions>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

12. Build the following two projects by executing the following command: mvn clean install
  - org.wso2.carbon.ui\_4.4.6\_fragment
  - org.wso2.carbon.ui\_4.4.6\_patch
13. Once the project is built, copy the two JAR files (from the <PROJECT\_HOME>/target/ directory) to the <CARBON\_HOME>/repository/components/dropins/ directory.
  - org.wso2.carbon.ui\_4.4.6\_fragment-1.0.0.jar
  - org.wso2.carbon.ui\_4.4.6\_patch-1.0.0.jar
14. Restart the WSO2 Carbon server.
15. Access the Management Console of WSO2 Carbon using the following URL:  
You view the new logo, which the patch bundle contains as shown below.

## Installing WSO2 Products as a Service

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

<b>Memory</b>	<ul style="list-style-type: none"> <li>• ~ 2 GB minimum</li> <li>• ~ 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.</li> </ul>
<b>Disk</b>	<ul style="list-style-type: none"> <li>• ~ 500 MB, excluding space allocated for log files and databases.</li> </ul>

- Environment compatibility

<b>Operating Systems / Databases</b>	<ul style="list-style-type: none"> <li>• All WSO2 Carbon-based products are Java applications that can be run on <b>any platform that is Oracle JDK 7/8 compliant</b>. Also, we <b>do not recommend or support OpenJDK</b>.</li> <li>• All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see <a href="#">Working with Databases</a>.</li> <li>• 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.</li> <li>• If you have difficulty in setting up any WSO2 product in a specific platform or database, please <a href="#">contact us</a>.</li> </ul>
--------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- **R e q u i r e d**

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 \*.

- **a p p l i c a t i o n s**

Application	Purpose	Version	Download Links
-------------	---------	---------	----------------

<b>Oracle Java S E Development Kit (JDK)*</b>	<ul style="list-style-type: none"> <li>• To launch the product as each product is a Java application.</li> <li>• To build the product from the source distribution (both JDK and Apache Maven are required).</li> <li>• To run Apache Ant.</li> </ul>	JDK 7 or 8. Oracle and IBM JRE 1.7 are also supported when running (not building) WSO2 products.	<a href="http://java.sun.com/javase/downloads/index.jsp">http://java.sun.com/javase/downloads/index.jsp</a>
-----------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------

## 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.

If you want to keep the scripts in a location other than `/etc/init.d/` folder, you can add a symbolic link to the script in `/etc/init.d/` and keep the actual script in a separate location. Say your script name is prodserver and it is in `/opt/WSO2/` folder, then the commands for adding a link to `/etc/init.d/` is as follows:

- Make executable: `sudo chmod a+x /opt/WSO2/prodserver`
- Add a link to `/etc/init.d/`: `sudo ln -snf /opt/WSO2/prodserver /etc/init.d/prodserver`

## 3. Install the startup script to respective runlevels using the command `update-rc.d`. For example, give the following command for the sample script shown in step1:

```
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
- Installing a single product as a Windows service
- Installing multiple products as Windows services

### *Prerequisites*

- **System requirements**

<b>Memory</b>	<ul style="list-style-type: none"> <li>• ~ 2 GB minimum</li> <li>• ~ 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.</li> </ul>
<b>Disk</b>	<ul style="list-style-type: none"> <li>• ~ 500 MB, excluding space allocated for log files and databases.</li> </ul>

- **Environment compatibility**

<b>Operating Systems / Databases</b>	<ul style="list-style-type: none"> <li>• All WSO2 Carbon-based products are Java applications that can be run on <b>any platform that is Oracle JDK 7/8 compliant</b>. Also, we <b>do not recommend or support OpenJDK</b>.</li> <li>• All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see <a href="#">Working with Databases</a>.</li> <li>• 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.</li> <li>• If you have difficulty in setting up any WSO2 product in a specific platform or database, please <a href="#">contact us</a>.</li> </ul>
--------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### **R e q u i r e d**

### **a p p l i c a t i o n s**

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 \*.

Application	Purpose	Version	Download Links
-------------	---------	---------	----------------

<b>Oracle Java S E Development Kit (JDK)*</b>	<ul style="list-style-type: none"> <li>• To launch the product as each product is a Java application.</li> <li>• To build the product from the source distribution (both JDK and Apache Maven are required).</li> <li>• To run Apache Ant.</li> </ul>	JDK 7 or 8. Oracle and IBM JRE 1.7 are also supported when running (not building) WSO2 products.	<a href="http://java.sun.com/javase/downloads/index.jsp">http://java.sun.com/javase/downloads/index.jsp</a>
-----------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------

- 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

###### Minimal `wrapper.conf` configuration

```
#*****
# 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 ROLLOUT it will be
automatically added as suffix of the file name
wrapper.logfile.maxsize=10m
# Maximum number of rolled log files which will be allowed before old
# files are deleted. The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
*****
# Wrapper Windows Service and Posix Daemon Properties
*****
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
*****
# Wrapper System Tray Properties
*****
# enable system tray
wrapper.tray = true
# TCP/IP port. If none is defined multicast discovery is used to find
the port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002
*****
# Exit Code Properties
# Restart on non zero exit code
*****
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
*****
# Trigger actions on console output
*****
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\trayMessage.gv
wrapper.filter.script.0.args=Exception
*****
# genConfig: further Properties generated by genConfig
*****
placeHolderSoGenPropsComeHere=

```

```

wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 =
-Xbootclasspath\:/a:${carbon_home}\lib\xboot\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 =
-XX:HeapDumpPath=${carbon_home}\repository\logs\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 =
-Djava.endorsed.dirs=${carbon_home}\lib\endorsed;${java_home}\jre\
\lib\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=\
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwsos2.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\log
4j.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 =

```

```
-Dcom.sun.jndi.ldap.connect.pool.timeout=3000
wrapper.java.additional.25 =
-Dorg.terracotta.quartz.skipUpdateCheck=true
```

## Only applicable to ESB 4.9.0:

If you are using the ESB 4.9.0 version, you must manually add the following property to the `wrapper.conf` file to avoid errors in the management console:

```
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.

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](#).

2. **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.
3. **Test the service in console mode:** You can verify that YAJSW is configured correctly for running the Carbon-based product as a Windows service.
  1. Open a Windows command prompt and execute the `runConsole.bat` script from the `<YAJSW_HOME>/bat/` directory as shown below.

```
cd C:\Documents and Settings\yajsw_home\bat\runConsole.bat
```

If the configurations are set properly for YAJSW, you will see console output similar to the following:

```
C:\Documents and Settings\yajsw_home\bat>runConsole.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" -c "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 11:12:27 AM org.apache.commons.vfs2.VfsLog info
INFO: Using "C:\DOCUMENTS\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store
[...]
```

2. You can now access the management console from your web browser via <https://localhost:9443/carbo>

n.

**4. Install and run the product as a service:** Execute the relevant script as explained below.

1. First, install the WSO2 product as a Windows service, by executing the following command in the <YAJSW\_HOME>/bat/ directory:

```
installService.bat
```

The console will display a message confirming that the WSO2CARBON service was installed:

```
C:\Documents and Settings\yajsw_home\bat>installService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
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" -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
Dec 30, 2012 12:51:42 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** INSTALLING "WSO2CARBON" *****
Service "WSO2CARBON" installed
Press any key to continue . . .
```

2. 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\
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" -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 1:09:00 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
platform null
***** STARTING "WSO2CARBON" *****
Service "WSO2CARBON" started
Press any key to continue . . .
C:\Documents and Settings\yajsw_home\bat>
```

**5. Stop and uninstall service:** Execute the relevant scripts as shown below.

1. 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\
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.UfsLog info
INFO: Using "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\ufs_cache" as temporary files store.
platform null
***** STOPPING "WSO2CARBON" *****
Service "WSO2CARBON" stopped
Press any key to continue . . .
```

2. 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" -r "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 1:19:14 PM org.apache.commons.vfs2.UfsLog info
INFO: Using "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\ufs_cache" as temporary files store.
platform null
***** REMOVING "WSO2CARBON" *****
Service "WSO2CARBON" removed
Press any key to continue . . .
C:\Documents and Settings\yajsw_home\bat>
```

### ***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\_service/<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\_service/<DSS\_YAJSW\_HOME>/conf/ directories. You simply have to replace carbon\_home with esb\_home and dss\_home respectively.

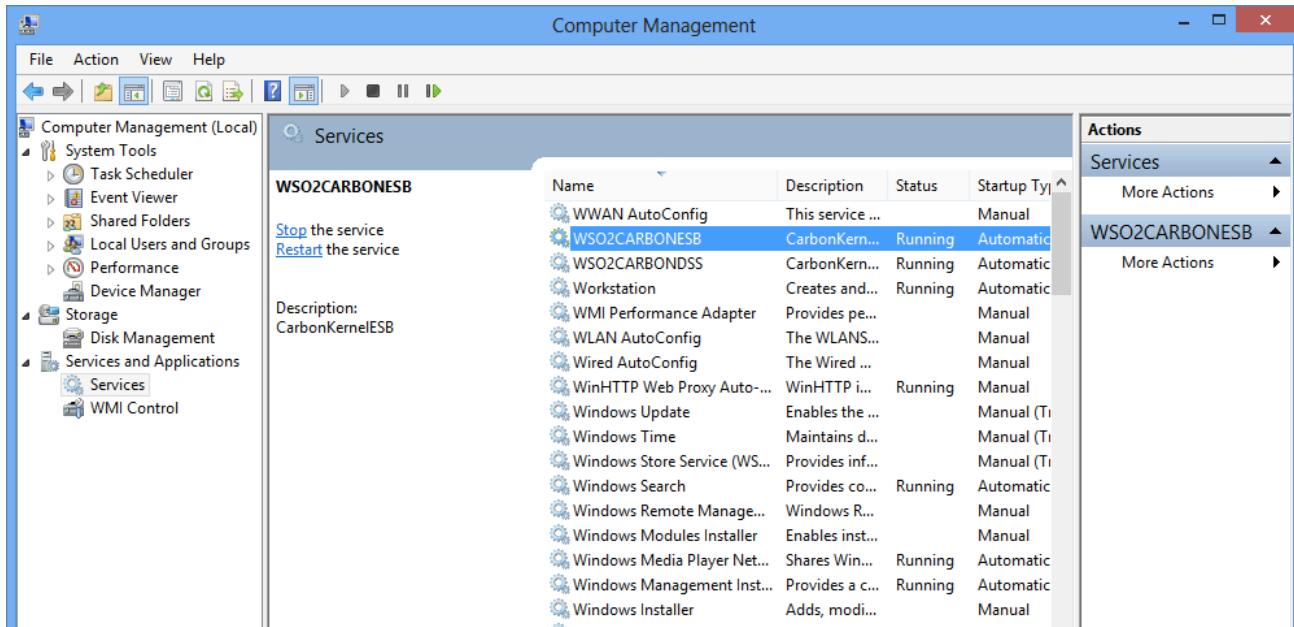
## Only applicable to ESB 4.9.0:

If you are using the ESB 4.9.0 version, you must manually add the following property to the wrapper.conf file to avoid errors in the management console:

```
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.
  1. Run installService.bat to install the Windows service.
  2. 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 ESB: <https://localhost:9443/carbon>.
- 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 <PROD UCT\_HOME>/repository/components/lib.

## Configuring an RDBMS Datasource

When adding a datasource, if you select RDBMS as the datasource type, the following screen appears:

### New Data Source

New Data Source

Data Source Type*	<input type="button" value="RDBMS"/>	
Name*	<input type="text"/>	
Description	<input type="text"/>	
Data Source Provider*	<input type="button" value="default"/>	
Driver*	<input type="text"/>	
URL*	<input type="text"/>	
User Name	<input type="text"/>	
Password	<input type="text"/>	
<input type="checkbox"/> Expose as a JNDI Data Source		
<input type="checkbox"/> Data Source Configuration Parameters		
<input type="button" value="Test Connection"/>	<input type="button" value="Save"/>	<input type="button" value="Cancel"/>

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-5.5.1-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 Souce:** 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 [ApacheTomcat JDBC Connection Pool guide](#).

After creating datasources, they appear on the **Data Sources** page. You can edit and delete them as needed by clicking **Edit** or **Delete** links.

### Configuring the Datasource Provider

A datasource provider connects to a source of data such as a database, accesses its data, and returns the results of the access queries. When creating a RDBMS datasource, use the default provider or link to an external provider.

#### Default datasource provider

To use the default datasource provider, select **default**, and then enter the Driver, URL, User Name, and Password connection properties as follows:

### New Data Source

New Data Source

Data Source Type*	RDBMS
Name*	rdbmsdatasource
Description	RDBMS Data Source
Data Source Provider*	default
Driver*	com.mysql.jdbc.Driver
URL*	jdbc:mysql://localhost:3306/test
User Name	root
Password	●●●●
<input type="checkbox"/> Expose as a JNDI Data Source	
<input type="checkbox"/> Data Source Configuration Parameters	
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>	

### External datasource provider

If you need to add a datasource supported by an external provider class such as `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource`, select **External Data Source**, click **Add Property**, and then enter the name and value of each connection property you need to configure. Following is an example datasource for an external datasource provider:

### New Data Source

New Data Source

Data Source Type*	RDBMS												
Name*	rdbmsdatasource												
Description	RDBMS Data Source												
Data Source Provider*	External Data Source												
Data Source Class Name*	<code>lbc.jdbc2.optional.MysqlXADataSource</code>												
<input type="button" value="Add Property"/> <table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>url</td> <td>:mysql://localhost:3306/test</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>user</td> <td>root</td> <td><input type="button" value="Delete"/></td> </tr> <tr> <td>password</td> <td>root</td> <td><input type="button" value="Delete"/></td> </tr> </tbody> </table>		Name	Value	Action	url	:mysql://localhost:3306/test	<input type="button" value="Delete"/>	user	root	<input type="button" value="Delete"/>	password	root	<input type="button" value="Delete"/>
Name	Value	Action											
url	:mysql://localhost:3306/test	<input type="button" value="Delete"/>											
user	root	<input type="button" value="Delete"/>											
password	root	<input type="button" value="Delete"/>											
<input type="checkbox"/> Expose as a JNDI Data Source													
<input type="checkbox"/> Data Source Configuration Parameters													
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>													

### 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:

## New Data Source

New Data Source

Data Source Type\* RDBMS

Name\*

Description

Data Source Provider\* default

Driver\*

URL\*

User Name admin

Password \*\*\*\*\*

Expose as a JNDI Data Source

Name

Use Data Source Factory

JNDI Properties

Data Source Configuration Parameters

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.

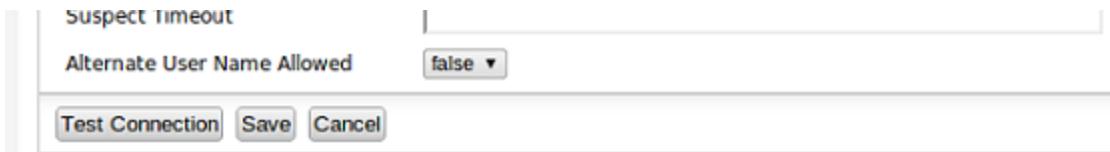
RDBMS datasources in WSO2 products use Tomcat JDBC connection pool (`org.apache.tomcat.jdbc.pool`). It is common to all components that access databases for data persistence, such as the registry, user management (if configured against a JDBC userstore), etc.

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

New Data Source

DataSource Id*	oracle-ds
Data Source Type*	RDBMS
Database Engine*	Oracle
Driver Class*	oracle.jdbc.driver.OracleDriver
URL*	jdbc:oracle:[drivertype]:[username/password]@[host]:[port]/[database]
User Name	
Password	
<input checked="" type="checkbox"/> Data source configuration parameters	
Transaction Isolation	TRANSACTION_UNKNOWN
Initial Size	
Max. Active	
Max. Idle	
Min. Idle	
Max. Wait	
Validation Query	
Test On Return	false
Test On Borrow	true
Test While Idle	false
Time Between Eviction Runs Mills	
Minimum Evictable Idle Time	
Remove Abandoned	false
Remove Abandoned Timeout	
Log Abandoned	false
Default Auto Commit	false
Default Read Only	false
Default Catalog	
Validator Class Name	
Connection Properties	
Init SQL	
JDBC Interceptors	
Validation Interval	
JMX Enabled	false
Fair Queue	false
Abandon When Percentage Full	
Max Age	
Use Equals	false



Following are descriptions of the parameters you can configure. For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

Parameter name	Description
Transaction isolation	The default TransactionIsolation state of connections created by this pool are as follows: <ul style="list-style-type: none"> <li>• TRANSACTION_UNKNOWN</li> <li>• TRANSACTION_NONE</li> <li>• TRANSACTION_READ_COMMITTED</li> <li>• TRANSACTION_READ_UNCOMMITTED</li> <li>• TRANSACTION_REPEATABLE_READ</li> <li>• TRANSACTION_SERIALIZABLE</li> </ul>
Initial Size (int)	The initial number of connections created, when the pool is started. Default value is zero.
Max. Active (int)	Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.
Max. Idle (int)	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 <code>minEvictableIdleTimeMillis</code> will be released. (also see <code>testWhileIdle</code> )
Min. Idle (int)	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 <code>testWhileIdle</code> .
Max. Wait (int)	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).
Validation Query (String)	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).
Test On Return (boolean)	Used to indicate if objects will be validated before returned to the pool. The default value is false. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</div>

Test On Borrow (boolean)	<p>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.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. In order to have a more efficient validation, see <code>validationInterval</code>.</p> </div>
Test While Idle (boolean)	<p>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 <code>timeBetweenEvictionRunsMillis</code>.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</p> </div>
Time Between Eviction Runs Mills (int)	<p>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).</p>
Minimum Evictable Idle Time (int)	<p>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).</p>
Remove Abandoned (boolean)	<p>Flag to remove abandoned connections if they exceed the <code>removeAbandonedTimeout</code>. If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the <code>removeAbandonedTimeout</code>. Setting this to true can recover database connections from applications that fail to close a connection. For more information, see <code>logAbandoned</code>. The default value is false.</p>
Remove Abandoned Timeout (int)	<p>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.</p>
Log Abandoned (boolean)	<p>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.</p>
Auto Commit (boolean)	<p>The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default. If not set, then the <code>setAutoCommit</code> method will not be called.</p>
Default Read Only (boolean)	<p>The default read-only state of connections created by this pool. If not set then the <code>setReadOnly</code> method will not be called. (Some drivers don't support read only mode. For example: Informix)</p>
Default Catalog (String)	<p>The default catalog of connections created by this pool.</p>

Validator Class Name (String)	The name of a class which implements the <code>org.apache.tomcat.jdbc.pool.Validator</code> . Validates the interface and provides a no-arg constructor (may be implicit). If specified, the class will be used to create a <code>Validator</code> instance, which is then used instead of any validation query to validate connections. The default value is null. An example value is <code>com.mycompany.project.SimpleValidator</code> .
Connection Properties (String)	Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be <code>[propertyName=property;]*</code> . The default value is null.  The <code>user</code> and <code>password</code> properties will be passed explicitly, so that they do not need to be included here.
Init SQL	Ability to run a SQL statement exactly once, when the connection is created.
JDBC Interceptors	Flexible and pluggable interceptors to create any customizations around the pool, the query execution and the result set handling.
Validation Interval (long)	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).
JMX Enabled (boolean)	Register the pool with JMX or not. The default value is true.
Fair Queue (boolean)	Set to true, if you wish that calls to <code>getConnection</code> should be treated fairly in a true FIFO fashion. This uses the <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue</code> 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 <code>fairQueue=true</code> , there is a decision making process based on what operating system the system is running. If the system is running on Linux ( <code>property os.name=Linux</code> ), then to disable this Linux specific behavior and still use the fair queue, simply add the property <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue.ignoreOS=true</code> to your system properties, before the connection pool classes are loaded.
Abandon When Percentage Full (int)	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 <code>abandonWhenPercentageFull</code> . The value should be between 0-100. The default value is zero, which implies that connections are eligible for closure as soon as <code>removeAbandonedTimeout</code> has been reached.
Max Age (long)	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 <code>maxAge</code> 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.
Use Equals (boolean)	Set to true, if you wish the <code>ProxyConnection</code> class to use <code>String.equals</code> , and set to false when you wish to use <code>==</code> when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.

Suspect Timeout (int)	Timeout value in seconds. Default value is zero. Similar to the <code>removeAbandonedTimeout</code> value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if <code>logAbandoned</code> 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.
Alternate User Name Allowed (boolean)	<p>By default, the <code>jdbc-pool</code> will ignore the <code>DataSource.getConnection(username,password)</code> call, and simply return a previously pooled connection under the globally configured properties <code>username</code> and <code>password</code>, for performance reasons.</p> <p>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 <code>DataSource.getConnection(username,password)</code> call, simply set the property <code>alternateUsernameAllowed</code>, to true. If you request a connection with the credentials <code>user1/password1</code>, and the connection was previously connected using different <code>user2/password2</code>, 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.</p>

## Configuring a Custom Datasource

When adding a datasource, if you select the custom datasource type, the following screen will appear:

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>
    <customDataSourceClass>org.wso2.carbon.dataservices.core.custom.ds.</customDataSourceClass>
    <customDataSourceProps>
        <property name="inmemory_datasource_schema">{Vehicles:[ID,N
        <property name="inmemory_datasource_records">
            {Vehicles:[["S10_1678","Harley Davidson Ultimate Chopp
            ["S10_1949","Alpine Renault 1300","Classic Cars","1952"
            ["S10_2016","Moto Guzzi 1100i","Motorcycles","1996"],
            ["S10_4698","Harley-Davidson Eagle Drag Bike","Motorcy
            ["S10_4757","Alfa Romeo GTA","Classic Cars","1972"],
            ["S10_4962","Lancia A Delta 16V","Classic Cars","1962"],
            ["S12_1099","Ford Mustang","Classic Cars","1968"],
            ["S12_1108","Ferrari Enzo","Classic Cars","2001"]]}
        </property>
    </customDataSourceProps>
</configuration>

```

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**.

Permissions assigned to the Admin role cannot be modified.

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.

```
PasswordJavaRegEx----- ^[\s]{5,30}$
PasswordJavaScriptRegEx-- ^[\s]{5,30}$
UsernameJavaRegEx----- ^~!#$;%*+={}\\{3,30}$
UsernameJavaScriptRegEx-- ^[\s]{3,30}$
RolenameJavaRegEx----- ^~!#$;%*+={}\\{3,30}$
RolenameJavaScriptRegEx-- ^[\s]{3,30}$
```

When creating users/roles, if you enter a username, password etc. that does not conform to the RegEx configurations, the system throws an exception. You can either change the RegEx configuration or enter values that conform to the RegEx. If you change the default user store or set up a secondary user store, configure the RegEx accordingly under the user store manager configurations in `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.

- 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:

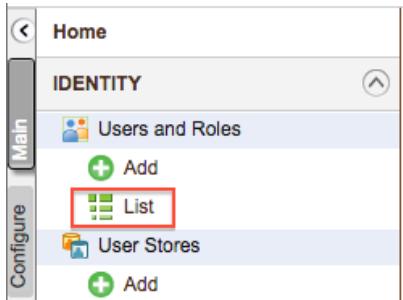
- Changing a Password
- Configuring Roles
- Configuring Users
- Role-based Permissions
- Managing User Attributes

## 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 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**.

### Users and Roles



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:
  1. Click **Users**.
  2. Find the user's account on the **Users** screen and click **Change Password** in the **Actions** column.
  3. Enter a new temporary password and click **Change**.
  4. 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 will open:

[Home > Configure > Users and Roles > Roles > Add Role](#)

## Add Role

### Step 1 : Enter role details

The screenshot shows a web-based form titled "Add Role". At the top, it says "Step 1 : Enter role details". Below this is a section titled "Enter role details" containing a "Domain" dropdown set to "PRIMARY" and a "Role Name\*" input field which is empty. At the bottom of this section are three buttons: "Next >", "Finish", and "Cancel".

4. Do the following:
  1. 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 user stores (which are repositories storing information about users and roles) are set up and configured, see [Configuring User Stores](#).
  2. Enter a unique name for this role.
  3. 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:

- Rename the role
- Change the default permissions associated with this role
- Assign this role to users
- View the users who are assigned this role
- 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](#).

```
<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.mult
itenancy.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 JDBCUserStoreManager 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/data-sources/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
- Importing users
  - Creating a file with users
  - Importing users from the CSV/Excel file
- Customizing the user's roles and permissions
- Customizing a user's profile
- Deleting an existing user

### ***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.

```
<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

The screenshot shows a web-based form titled "Enter user name". At the top is a "Domain" dropdown menu set to "PRIMARY". Below it are four input fields: "User Name\*" (empty), "Password\*" (empty), and "Password Repeat\*" (empty). At the bottom of the form are three buttons: "Next >" (disabled), "Finish" (disabled), and "Cancel".

4. Do the following:
  1. 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](#).
  2. 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.
  3. 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:

- If the option to import users in bulk is not enabled in your product by default, 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).

```
<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 500,000 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.

```
<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. Other user attributes can be imported only if [claim URLs are defined for such attributes](#) in the product. For example, consider that you have claim URLs defined for your product as shown below. These will allow you to import the user's **email address**, **full name**, **last name**, **given name** and **role** in addition to the **username** and **password**.

Select

```
http://wso2.org/claims/emailaddress
http://wso2.org/claims/fullname
http://wso2.org/claims/givenname
http://wso2.org/claims/lastname
http://wso2.org/claims/role
```

The username, password and other attributes (claim URLs) that you import should be given in a CSV file as shown below. Note that the first line of the file will not be imported considering that it is not a username.

```
UserName,Password,Claims
name1,password1,http://wso2.org/claims/emailaddress=name1@gmail.com,http://wso2.org/claims/fullname=fullname1
name2,password2,http://wso2.org/claims/emailaddress=name2@gmail.com,http://wso2.org/claims/fullname=fullname2
name3,password3,http://wso2.org/claims/emailaddress=name3@gmail.com,http://wso2.org/claims/fullname=fullname3
```

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 file. To use this option, you need to first enable the [Ask Password](#) option for the server.

```
UserName,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
```

## Importing users from the CSV/Excel file

To import users in bulk:

1. Log in to the management console of your WSO2 product.
2. Click **Add** under **Users and Roles** in the **Configure** menu.
3. In the **Add Users and Roles** screen, 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 list.
5. Click **Choose File** to give the path to the CSV/Excel file that contains the users that you want to import.
6. Click **Finish** to start importing.

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.

Deleting a user cannot be undone.

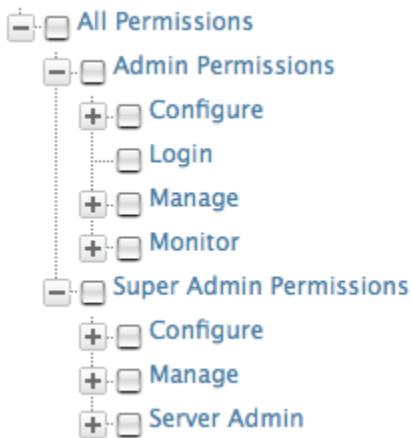
1. On the **Main** tab in the management console click **List** under **Users and Roles**.
2. Click **Users**. This link is only visible to users with **User Management** level permissions. For information on permissions, see [Role-based Permissions](#).
3. In the **Users** list, click **Delete** next to the user you want to delete, and then click **Yes** to confirm the operation.

## **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.

## Roles

The screenshot shows a search interface for roles. It includes a 'Search' bar, a 'Select Domain' dropdown set to 'ALL-USER-STORE-DOMAINS', and a search input field with a placeholder '\*' and a 'Search' button. Below this is a table listing roles:

Name	Actions
admin	<a href="#">Assign Users</a> <a href="#">View Users</a>
Internal/everyone	<a href="#">Permissions</a>

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.

```

<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.

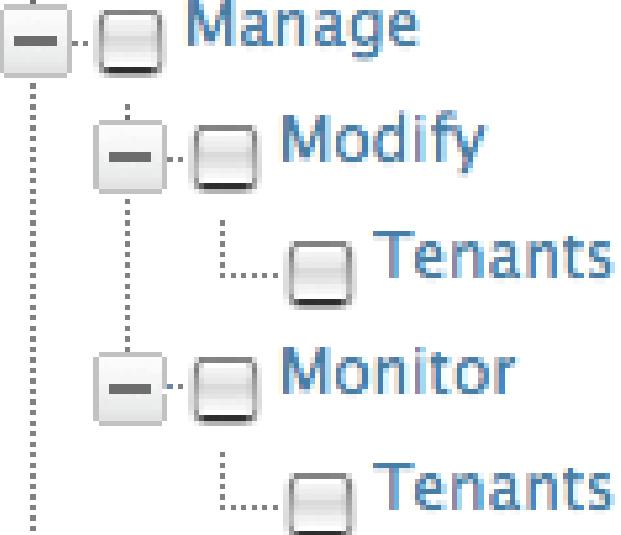
You will be able to log in to the management console of the product with the **Admin** user defined in the `user-mgt.xml` file. You can then create new users and roles and configure permissions for the roles using the management console. However, note that you cannot modify the permissions of the **Admin** 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.

Permission	Description of UI menus enabled
<b>Configuration</b> permissions: 	<p>The <b>Super Admin/Configuration</b> 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.</p> <ul style="list-style-type: none"> <li>- <b>Feature Management</b> permission ensures that a user can control the features installed in the product using the management console. That is, the <b>Features</b> option will be enabled under the <b>Configure</b> menu.</li> <li>- <b>Logging</b> permission enables the possibility to configure server logging from the management console. That is, the <b>Logging</b> option will be enabled under the <b>Configure</b> menu.</li> </ul>
<b>Management</b> permissions: 	<p>The <b>Super Admin/Manage</b> permissions are used for adding new tenants and monitoring them.</p> <ul style="list-style-type: none"> <li>- <b>Modify/Tenants</b> permission enables the <b>Add New Tenant</b> option in the <b>Configure</b> menu of the management console, which allows users to add new tenants.</li> <li>- <b>Monitor/Tenants</b> permission enables the <b>View Tenants</b> option in the <b>Configure</b> menu of the management console.</li> </ul>

**Server Admin** permissions:



Selecting the **Server Admin** permission enables the **Shutdown/Restart** option in the **Main** menu of the management console.

- 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.

Permission level	Description of UI menus enabled
Admin	<p>When the <b>Admin</b> permission node is selected, the following menus are enabled in the management console:</p> <ul style="list-style-type: none"> <li>- <b>User Store Management:</b> 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 <a href="#">user store management</a> for more details.</li> <li>- <b>Identity Providers:</b> See the topic on <a href="#">working with identity providers</a> for details on how to use this option.</li> <li>- Additionally, all permissions listed under <b>Admin</b> in the permissions navigator are selected automatically.</li> </ul>
Admin/Configure	<p>When the <b>Admin/Configure</b> permission node is selected, the following menus are enabled in the management console:</p> <ul style="list-style-type: none"> <li>- <b>Main</b> menu/<b>PAP:</b> See the topic on <a href="#">working with entitlement</a> for details on how to use this option.</li> <li>- <b>Main</b> menu/<b>PDP:</b> See the topic on <a href="#">working with entitlement</a> for details on how to use this option.</li> <li>- <b>Configure</b> menu/<b>Server Roles:</b> See the topic on <a href="#">server roles</a> for more details.</li> <li>- <b>Tools</b> menu/<b>Tryit (XACML):</b> See the topic on <a href="#">working with the TryIt tool</a> for details on how to use this option.</li> <li>- Additionally, all permissions listed under <b>Configure</b> in the permissions navigator are selected automatically.</li> </ul>

Admin/Configure/Security	<p>When the <b>Admin/Configure/Security</b> permission node is selected, the following menus are enabled in the <b>Configure</b> menu of the management console:</p> <ul style="list-style-type: none"> <li>- <b>Claim Management:</b> See the topic on <a href="#">claim management</a> for details on how to use this option.</li> <li>- <b>Keystores:</b> See the topic on <a href="#">keystores</a> for details on how to use this option.</li> <li>- <b>Service Principle (Kerberos KDC):</b> See the topic on <a href="#">kerberos security</a> for details on how to use this option.</li> <li>- <b>Email Templates:</b> See the topics on <a href="#">email templates</a> for details on how to use this option.</li> </ul> <p>- This permission will also enable the <b>Roles</b> option under <b>Configure/ Users and Roles</b>. See the topic on <a href="#">configuring users, roles and permissions</a> for more information.</p> <p>- Additionally, all permissions listed under <b>Security</b> in the permissions navigator are selected automatically.</p>
Admin/Configure/Security/Identity Management/User Management	This permission enables the possibility to add users from the management console. That is, the <b>Users</b> option will be enabled under <b>Configure/Users and Roles</b> .
Admin/Configure/Security/Identity Management/Password Management	This permission enables the <b>Change Password</b> option for the users listed in the <b>User Management/Users and Roles/Users</b> screen, which allows the log in user to change the passwords
Admin/Configure/Security/Identity Management/Profile Management	This permission enables the <b>User Profile</b> option for the users listed in the <b>User Management/Users and Roles/Users</b> screen, which allows the log in user to update user profiles.
Admin/Manage	<p>When the <b>Admin/Manage</b> permission is selected, the following menus will be enabled in the management console:</p> <ul style="list-style-type: none"> <li>- <b>Main menu/Service Providers:</b> See the topic on <a href="#">working with service providers</a> for details on how to use this option.</li> <li>- <b>Tools menu/SAML:</b> See the topic on <a href="#">working with the SAML tool kit</a> for more details.</li> </ul> <p>- Additionally, all permissions listed under <b>Admin/Manage</b> in the permissions navigator will be enabled automatically.</p>
Admin/Manage/Resources/Browse	This permission enables the <b>Browse</b> option under the <b>Registry</b> menu in the main navigator. This option allows users to browse the resources stored in the registry by using the <b>Registry</b> tree navigator.
Admin/Manage/Search	This permission enables the <b>Search</b> option under the <b>Registry</b> sub menu in the <b>Main</b> menu. This option allows users to search for specific resources stored in the registry by filling in the search criteria.

Admin/Monitor	When the <b>Admin/Monitor</b> permission node is selected, the following menus are enabled in the management console:  - <b>Monitor menu/System Statistics:</b> See the topic on <a href="#">system statistics</a> for information on how to use this option. - <b>Monitor menu/SOAP Message Tracer:</b> See the topic on the <a href="#">SOAP tracer</a> for information on how to use this option.  - Additionally, all permissions listed under <b>Admin/Monitor</b> in the permissions navigator will be enabled automatically.
Admin/Monitor/Logs	When the <b>Admin/Monitor/Logs</b> permission node is selected, the following menus are enabled in the management console:  - <b>Monitor menu/Application Logs</b> - <b>Monitor menu/System Logs</b>  See the topic on <a href="#">system logs</a> for information on how to use these options.

## 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.
  1. Log into the product's Management Console.
  2. On the **Configure** tab in the Management Console, click **Users and Roles**.
  3. Click **Users**. This link is only visible to users with the Admin role.
  4. From the list of users that appear in the resulting page, identify the user whose attributes you want to modify and click **User Profile**.

## Update Profile : JasonH

User Profile

Profile Name *	default
First Name *	<input type="text"/>
Last Name *	JasonH
Organization	<input type="text"/>
Address	<input type="text"/>
Country	<input type="text"/>
Email *	<input type="text"/>
Telephone	<input type="text"/>
Mobile	<input type="text"/>
IM	<input type="text"/>
URL	<input type="text"/>
Role	Internal/everyone

5. 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.

```
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.

```
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.

```
{domain_name/attribute_Name} ; {domain_name/attribute_Name} ;
{domain_name/attribute_Name} ;
```

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://ws.o2.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.

```
asel@soasecurity.com,aselapathberiya@soasecurity.com
```

See the following screen for how this will look in the user interface of the product's Management Console.

[Home](#) > [Configure](#) > [Users and Roles](#) > [Users](#) > [Update Profile](#)

## Update Profile : asela

User Profile

Profile Name *	default
First Name *	Asela
Last Name *	pathberiya
Organization	
Address	
Country	
Email *	asela@soasecurity.com,aselapathberiya@
Telephone	
Mobile	
IM	
URL	
Role	Internal/everyone

**Update** **Cancel**

- When using the `RemoteUserStoreManagerService` API, call it as follows.

```
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.

The screenshot shows an LDAP browser window with the title bar "uid=asela,ou=Users,dc=WSO2,dc=ORG". The "DN" field also contains "uid=asela,ou=Users,dc=WSO2,dc=ORG". Below is a table of attributes and their values:

Attribute Description	Value
<b>objectClass</b>	<i>identityPerson (structural)</i>
<b>objectClass</b>	<i>inetOrgPerson (structural)</i>
<b>objectClass</b>	<i>organizationalPerson (structural)</i>
<b>objectClass</b>	<i>person (structural)</i>
<b>objectClass</b>	<i>scimPerson (structural)</i>
<b>objectClass</b>	<i>top (abstract)</i>
<b>objectClass</b>	<i>wso2Person (structural)</i>
<b>cn</b>	asela
<b>sn</b>	pathberiya
<b>createdDate</b>	2014-09-25T04:20:35
<b>givenName</b>	Asela
<b>lastModifiedDate</b>	2014-09-25T04:21:25
<b>mail</b>	asela@soasecurity.com
<b>mail</b>	aselapathberiya@gmail.com
<b>scimId</b>	82706898-6874-4242-b903-c9b94f76be8f
<b>uid</b>	asela
<b>userPassword</b>	Plain text password

### 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.

```
public Map<String, String> getUserPropertyValues(String userName,
String[] propertyNames, String profileName) throws UserStoreException
```

- Method 2.

```
protected abstract void doSetUserClaimValue(String userName, String
claimURI, String claimValue, String profileName) throws
UserStoreException;
```

- Method 3.

```
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.
  1. Configure **UserNameSearchFilter** that helps to search for the user object in the LDAP using both **mail** and **uid** attributes.
 

```
<Property
name="UserNameSearchFilter">(&&(objectClass=person)(|(mail=?)(uid=?)))</
Property>
```
  2. Disable **UserDNPattern** property, if it is currently enabled.
 

```
<!--Property
name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</Property-->
```
  3. 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.

```
<AuthenticatorConfig name="BasicAuthenticator" enabled="true">
<Parameter
name="UserNameAttributeClaimUri">http://wso2.org/claims/emailaddress</Para
meter>
</AuthenticatorConfig>
```

This will return the email address of the authenticated user. It can be configured to return any attribute by changing the 'UserNameAttributeClaimUri' parameter.

## 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](#)

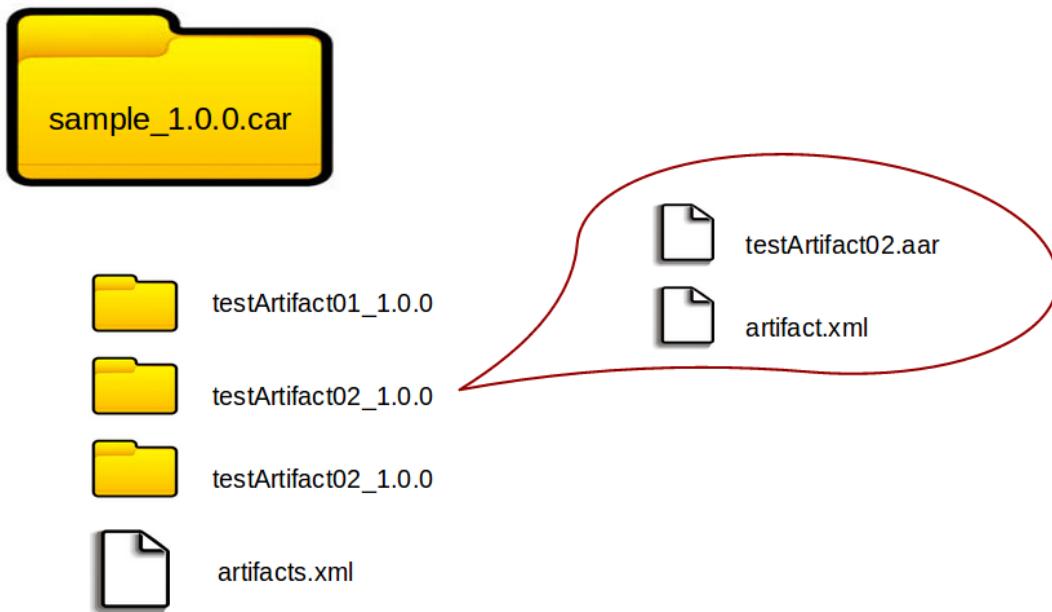
### Introduction to Composite Applications

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 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:

WSO2 product	serverRole value	Sample artifacts
WSO2 Application Server	ApplicationServer	foo.aar, jax-wx.war
WSO2 Enterprise Service Bus	EnterpriseServiceBus	proxy.xml
WSO2 Identity Server	IdentityServer	
WSO2 Data Services Server	DataServicesServer	
WSO2 Governance Registry	GovernanceRegistry	
WSO2 Business Activity Monitor	BusinessActivityMonitor	
WSO2 Business Process Server	BusinessProcessServer	my_bpel.zip
WSO2 Business Rules Server	BusinessRulesServer	
WSO2 Gadget Server	GadgetServer	

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.

The screenshot shows a modal dialog titled "Add Custom Server Role". Inside, there is a "Role Name" input field with the value "TestRole" highlighted. Below the input field are two buttons: "Add" and "Cancel".

3. Note that the newly added server role is displayed in the list.

The screenshot shows a table titled "Server Roles". The columns are "Role Name", "Type", and "Actions". There are two rows: "CarbonServer" (Type: Default) and "TestRole" (Type: Custom). The "TestRole" row is highlighted with a red border. At the bottom left, there is a green button labeled "+ Add New Server Role".

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,

```
<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:

```
<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,

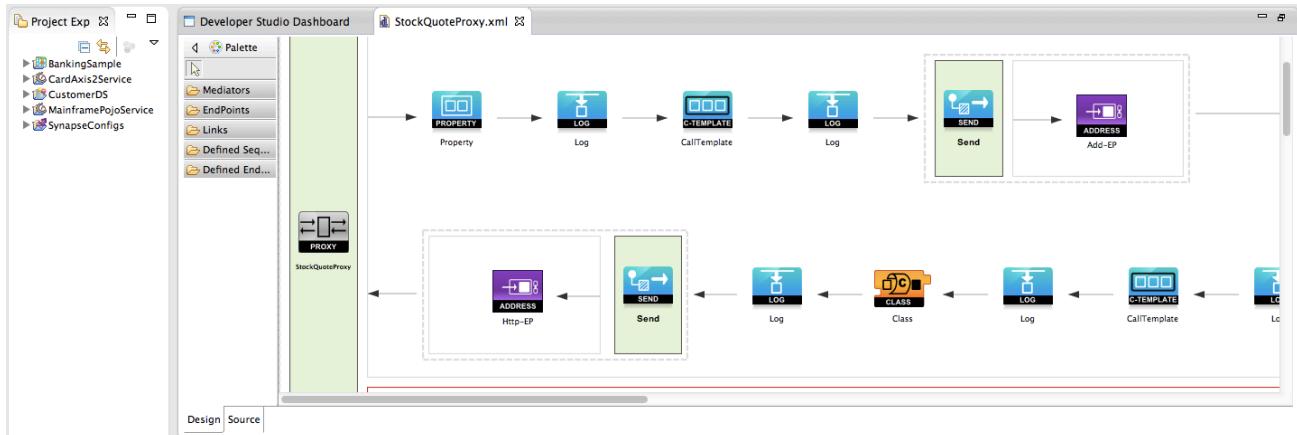
```
sh wso2server.sh -DserverRoles=appserver1,dataservices1
```

## Packaging Artifacts into Composite Applications

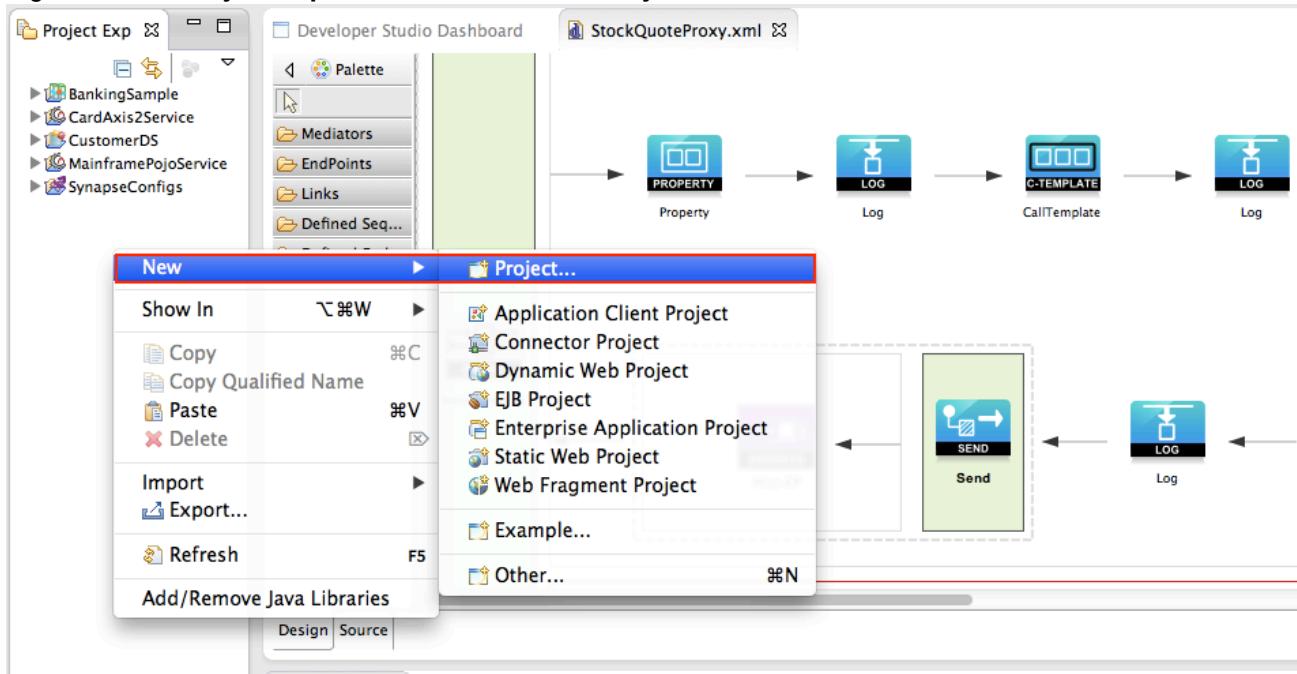
After working on your artifacts using the Tooling environment, you bundle them into Composite Applications (C-Apps) that can later be deployed in the server. The steps below describe how to bundle your artifacts into a C-App. When deploying via the management console, you will need to create a Composite 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](#).

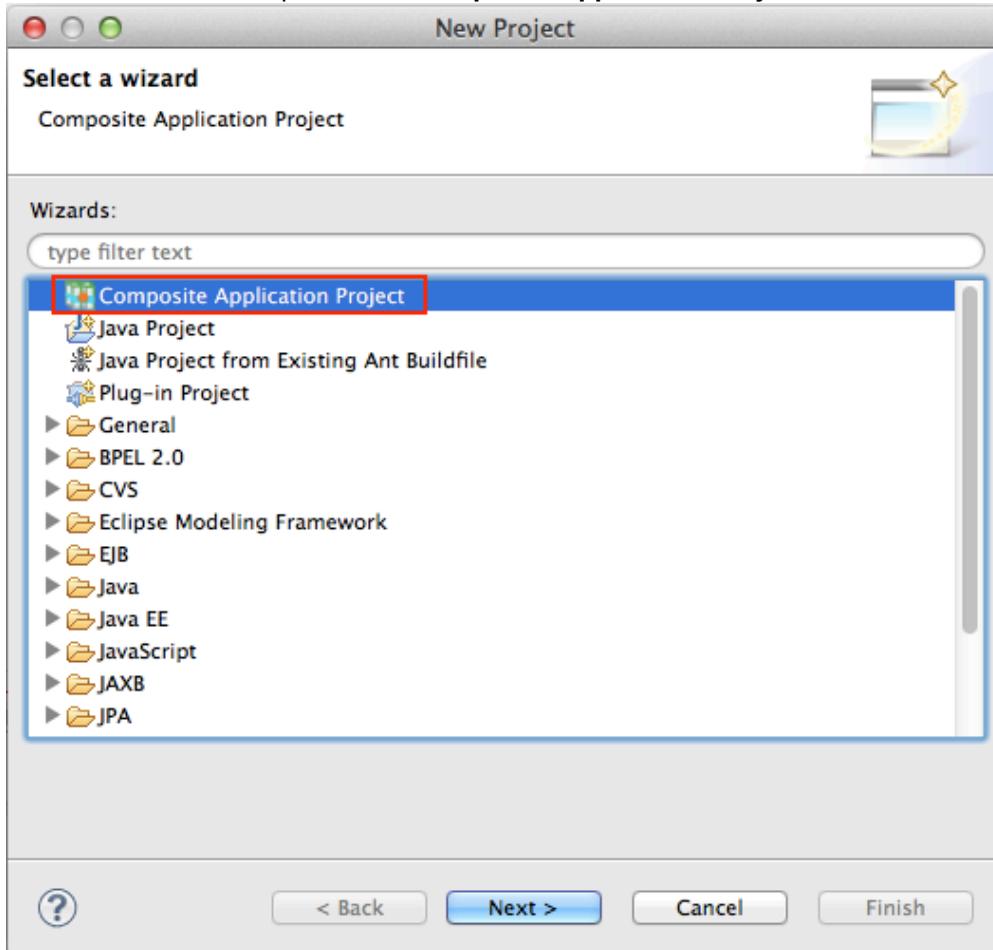
1. Open the Tooling interface with all the artifacts/projects that you created.



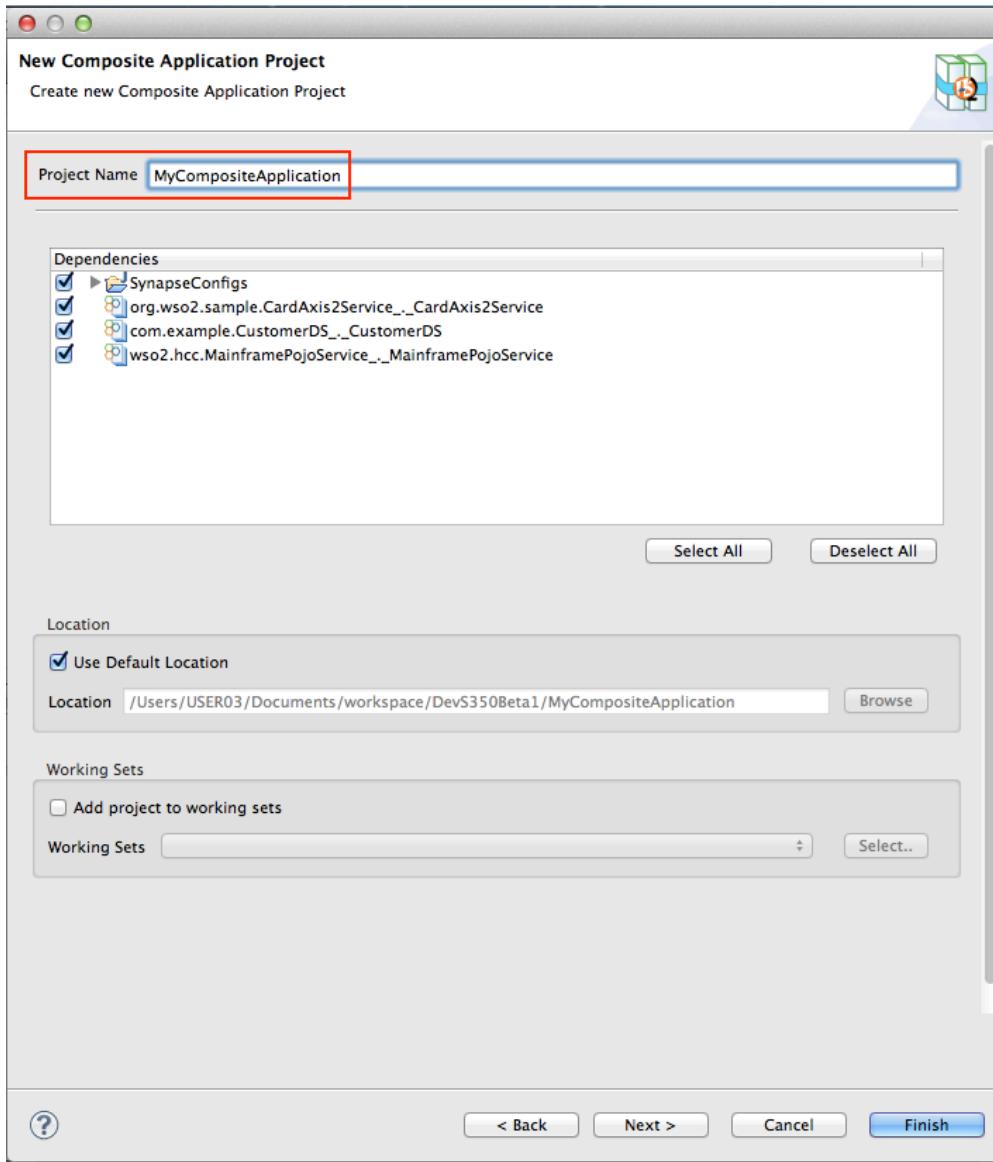
2. Right-click the Project Explorer and click New -> Project.



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.

Artifact	Server Role	Version
SynapseConfigs	--	1.0.0
com.example.CustomerDS..._CustomerDS	DataServicesServer	1.0.0
org.wso2.sample.CardAxis2Service..._CardAxis2Service	ApplicationServer	1.0.0
wso2.hcc.MainframePojoService..._MainframePojoService	ApplicationServer	1.0.0

## 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

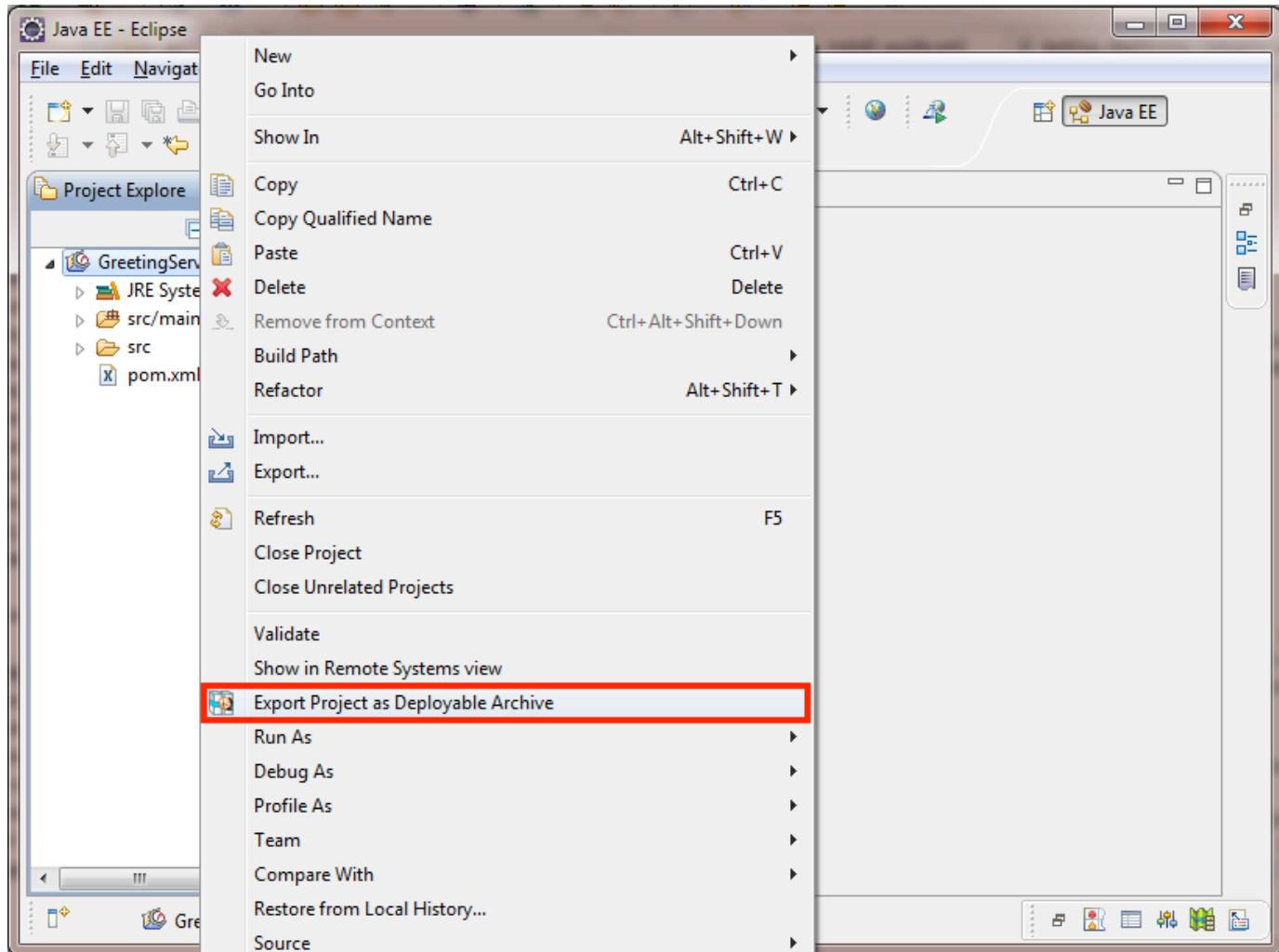
You can also create separate deployable artifacts for each individual artifact in your project. For example, suppose you created an Apache Axis2 Service. When you right-click the Axis2 Service Project, there is an option called **Export Project as Deployable Archive**. It creates the relevant deployable 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.

Artifact Type	Archive Type
Apache Axis2 Artifact	.aar
Endpoint Artifact	.xml
Sequence Artifact	.xml
Proxy Service Artifact	.xml
Local Entry Artifact	.xml
Synapse Configuration	.xml
ESB Mediator	.jar

Registry Resource	Registry Resource with necessary metadata
Third Party Library Artifact	.jar (OSGI Bundle)



## Deploying Composite Applications in the Server

After packaging artifacts into a Composite Application, you can deploy it to the server using the Tooling interface.

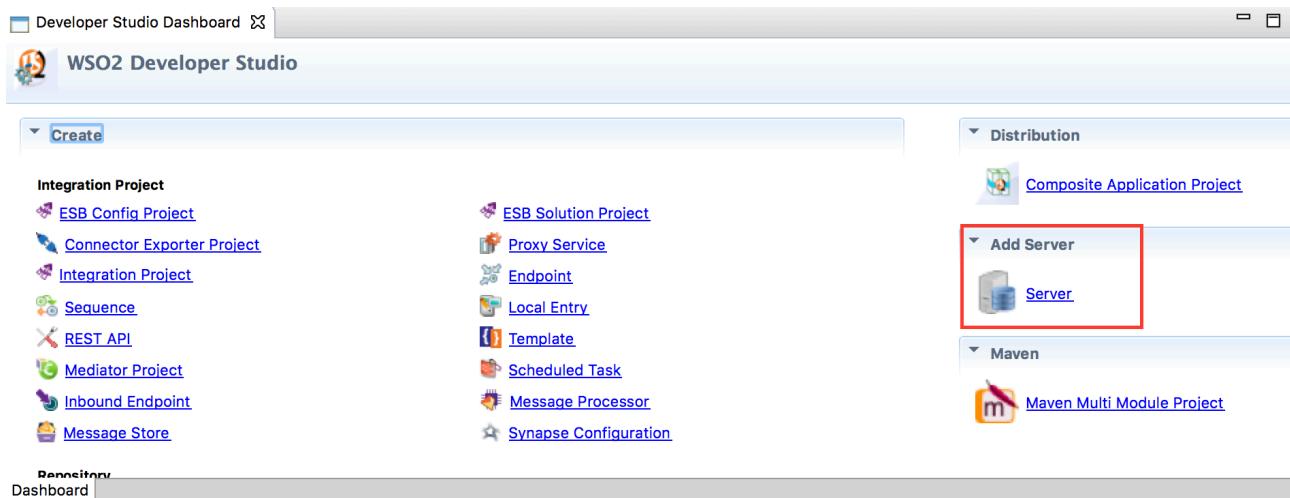
Alternatively, you can also do this using one of the following options:

- Deploying via the product's Management Console
- Deploying via hot deployment
- Deploying using the Maven plug-in

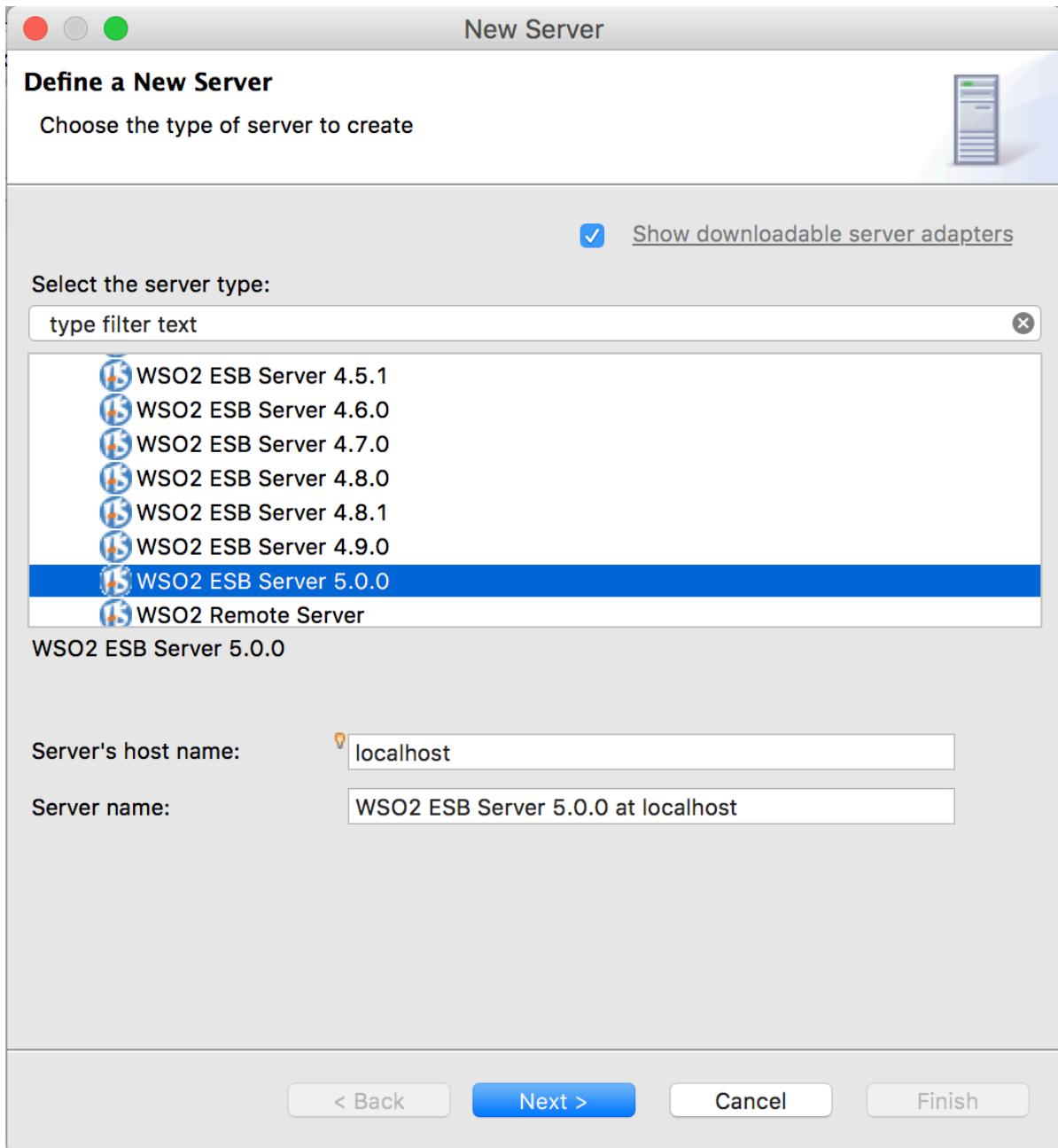
**Before you begin**, package your artifacts into a Composite Application. Note that the examples here are given using WSO2 ESB, but the steps are the same for any WSO2 server.

### Deploying using the Tooling interface

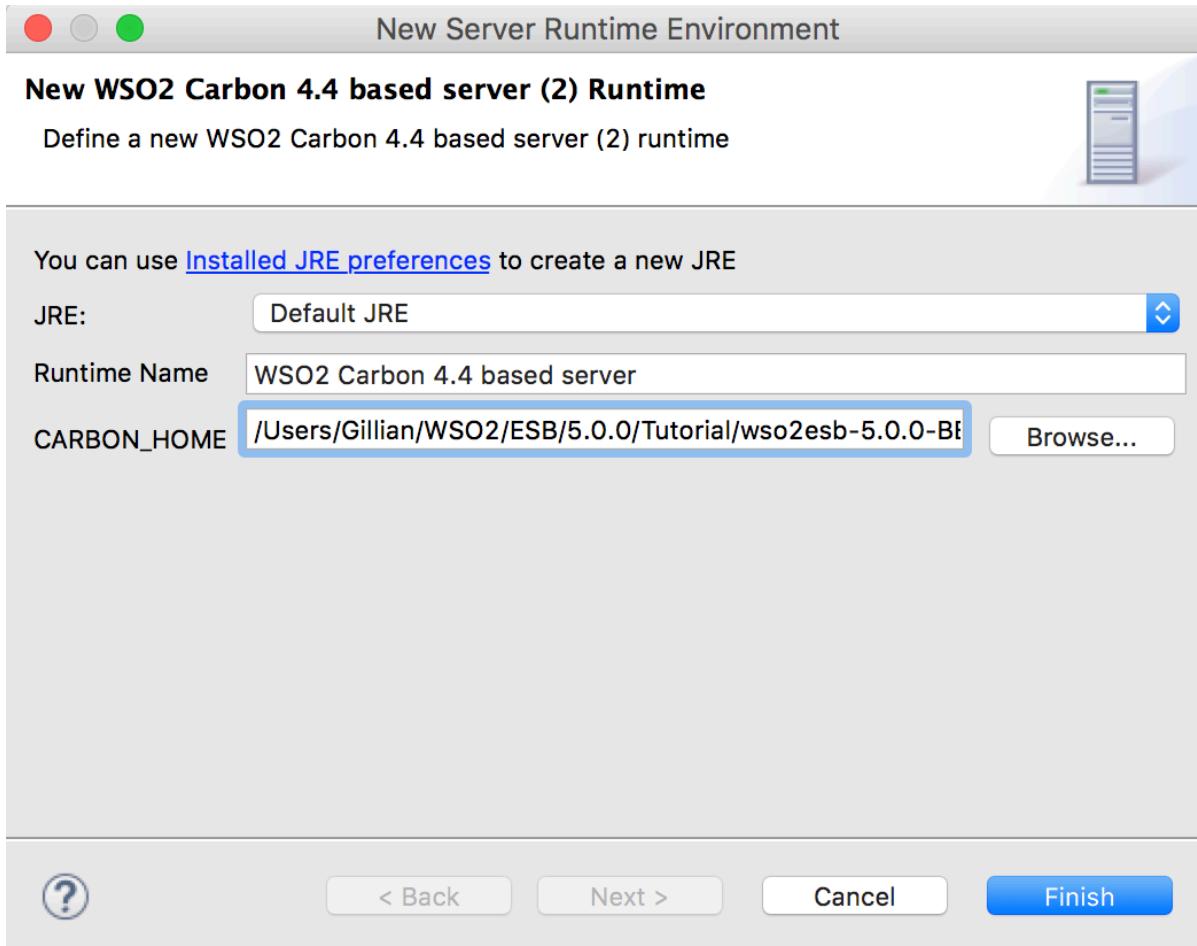
1. In the Tooling interface, navigate to **Developer Studio Dashboard** and click **Server** under **Add Server**.



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.

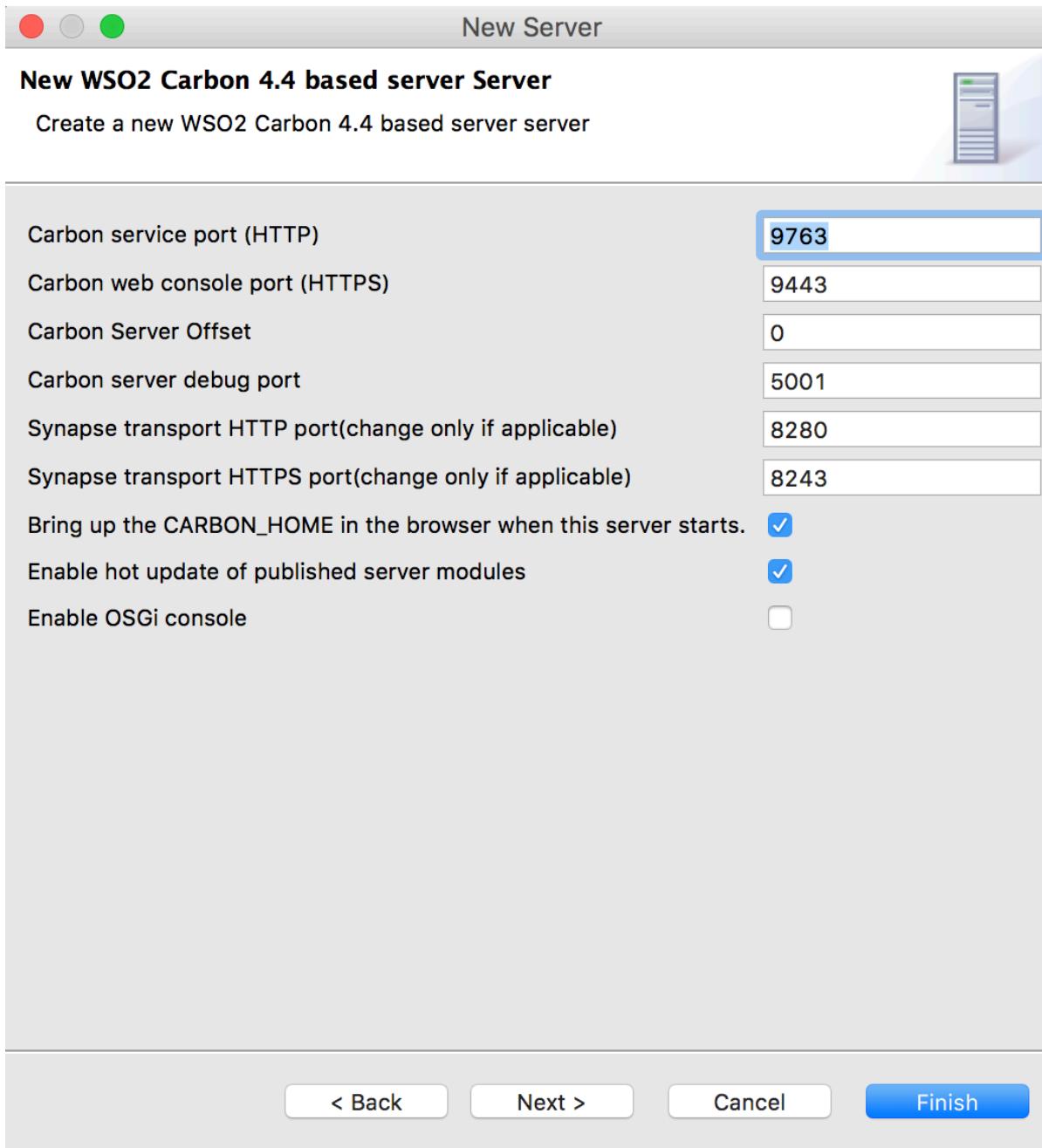


3. Click **Next**. In the CARBON\_HOME field, provide the path to your product's home directory and then click **Next** again. For example,

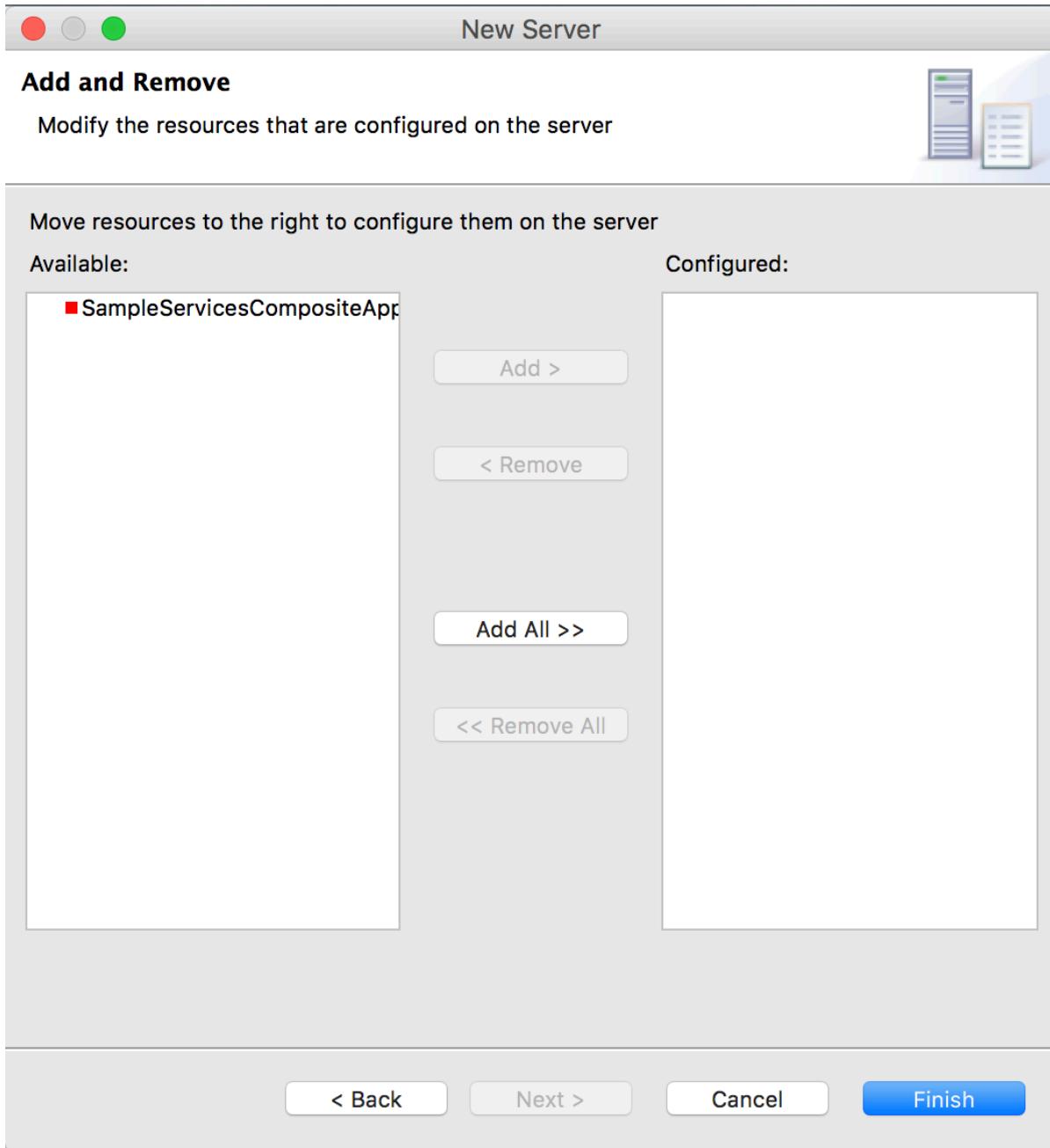


4. Review the default port details for your server and click **Next**. Typically, you can leave these unchanged but if you are already running another server on these ports, give unused ports here.

**Tip:** See [Default Ports of WSO2 Products](#) for more information.



5. 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.

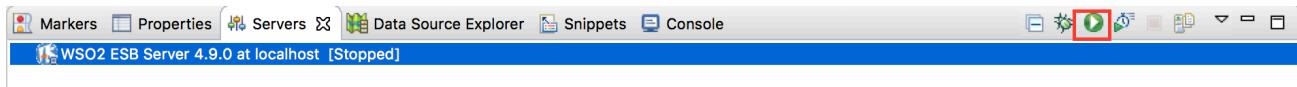
[Home](#) > Manage > Carbon Applications > List

### Carbon Applications List

1 Running Carbon Applications.

Carbon Applications	Version	Actions
SampleCApp	1.0.0	Delete  Download

7. On the **Servers** tab, note that the server is currently stopped. Click the "start the server" icon on the **Servers** tab's toolbar. If prompted to save changes to any of the artifact files you created earlier, click **Yes**.



8. 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**.

**Tip:** Note that when **deleting a C-App**, it is not recommended to delete individual artifacts of the CAR file. The recommended practice is to delete the entire C-App.

You can delete a C-App from the Management Console or by deleting the CAR archive from <PRODUCT\_HOME>/repository/carbonapps/{tenant-ID}. {tenant-ID} is 0 in a single-tenant environment as 0 is the super tenant ID. Manual undeployment in a multi-tenanted environment is not recommended if you are unaware of the tenant ID.

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.

**Tip:** 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/ deployment/ 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>

```

- Using command prompt, navigate to the **ESB Config Project** folder and build the project using the following command:

```
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.

- 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:

```
<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:

```
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.

Services	Service Type	Status	WSDL1.1	WSDL2.0	Action	Action	Action	Action
echo	axis2	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download	Design View	Source View
ProxySample	axis2	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download		
Version	axis2	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download		
wso2carbon-sts	sts	Unsecured	WSDL1.1	WSDL2.0				

## Working with Features

This chapter contains the following information:

- [Introduction to Feature Management](#)
- [Managing the Feature Repository](#)
- [Installing Features](#)
- [Uninstalling Features](#)
- [Recovering from Unsuccessful Feature Installation](#)

## Introduction to Feature Management

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 feature. A component in the Carbon platform is a single or a collection of OSGi 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.

This component-based architecture of the WSO2 Carbon platform gives developers flexibility to build efficient and lean products that best suit their unique business needs, simply by adding and removing components.

### **Provisioning WSO2 Carbon**

Provisioning software is the act of placing an individual software application or a complete software stack onto a target system. What we mean by provisioning WSO2 Carbon is installing/updating/uninstalling features to/from the Carbon base platform on top of which the entire WSO2 product stack is developed. It is also possible to easily revert to a previous feature configuration using the provisioning support.

Features can be easily installed to any WSO2 product using the *WSO2 Carbon Feature Manager* that comes with the product. Feature manager is powered by Equinox P2 and allows you to connect to a remote or local P2 repository and get any feature installed into the product's runtime.

P2 can be used as a provisioning platform for any OSGi-based application. P2 has enabled easy provisioning capabilities in WSO2 Carbon, thereby increasing the user-friendliness in building customized SOA products using the Carbon platform.

## Managing the Feature Repository

### WSO2 Feature Repository:

The WSO2 Feature Repository consists of features that are bundled into WSO2 products that are 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/>.

- Accessing the available repository list
- Adding a repository
- Editing a repository
- Enabling/disabling a repository
- Deleting a repository

#### *Accessing the available repository list*

1. Log on to the product's Management Console.
2. On the **Configure** menu, click **Features**.
3. Click **Repository Management**.  
The **Manage Repositories** page appears, which allows you to view or modify the feature management settings.

#### *Adding a repository*

1. [Access the available repositories list](#).
2. Click **Add Repository**.
3. Provide a convenient name for the repository being added.
4. Enter the repository location using one of the following methods:
  1. URL - this option is used when you are adding an external repository.
    - Select the **URL** option.
    - Enter the Equinox P2 repository URL.
  2. 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.
5. Click **Add**. The newly added repository will appear in the available repositories list.

#### *Editing a repository*

1. [Access the available repositories list](#).
2. Click **Edit** corresponding to the repository that you wish to modify.
3. You can specify a new repository name.

This page allows you to change the repository name only. If you want to change the URL, you need to remove the old repository and then add the new repository.

#### *Enabling/disabling a repository*

1. [Access the available repositories list](#).
2. Click on the **Enable** or **Disable** link associated with the repository.

By default, all the repositories are enabled. When you perform a provisioning operation, metadata and artifacts are searched **only** from the enabled repositories.

### **Deleting a repository**

1. Click **Remove** respective to the repository to be removed.
2. Click **Yes** to confirm your deletion request.

## **Installing Features**

The following are the ways in which features can be installed in WSO2 products:

- [Installing Features using pom Files](#)
- [Installing Features via the UI](#)

### **Installing Features using pom Files**

**Before you begin**, 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.

```
<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>
  <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>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>

```

```

<metadataRepository>file:p2-repo</metadataRepository>

<artifactRepository>file:p2-repo</artifactRepository>

<destination>$distribution_name/repository/components</destination>

<deleteOldProfileFiles>false</deleteOldProfileFiles>
    <features>
        <feature>

<id>org.wso2.carbon.tryit.feature.group</id>
            <version>4.3.0</version>
        </feature>
    </features>
</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"
dir="$distribution_name/repository/components/default/configuration/o
rg.eclipse.equinox.simpleconfigurator">
                        <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>
        <updatePolicy>daily</updatePolicy>
        <checksumPolicy>ignore</checksumPolicy>
    </releases>

```

```
</repository>
</repositories>
<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>
</project>

```

Read more about the following plugins:

- [maven-antrun-plugin](#)
- [Carbon P2 plugin](#)

2. The above sample `pom.xml` file specifies the default product profile (`<profile>default</profile>`) and the corresponding directory path (`dir="$distribution_name/repository/components/default/configuration/org.eclipse.equinox.simpleconfigurator"`). If your product is running on a different profile, you need to update the profile name. Read more about profiles from [here](#).
3. Unzip the original product distribution (e.g., `wso2carbon-<version>.zip`) and copy it to the same location as your `pom.xml` file.
4. Replace `$distribution_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 `$distribution_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 of two ways:
  1. 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/>).
  2. Replace `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:

```

<metadataRepository>http://product-dist.wso2.com/p2/carbon/releases/wilkes/</metadataRepository>
<artifactRepository>http://product-dist.wso2.com/p2/carbon/releases/wilkes/</artifactRepository>

```

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'.

```

<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:

```
mvn clean install
```

Upon successful invocation of the build, the product distribution is provisioned with the new features. This

approach is scriptable.

## Installing Features via the UI

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/>.

The Feature Management UI is available in WSO2 products to help users discover and try out new features. Developers can use this UI to install features in their local machine and try it out. It is not recommended to use this UI in production environments and UAT environments. In such environments, it is recommended to use the [POM-based approach](#).

Note the following when you use the Feature Management UI:

- 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 UI, which creates 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 and then directly take the pack (in which the required feature is installed) and deploy it in the cluster.
- In the [POM-based approach](#), you will have the list of features and the corresponding versions of the features that are installed. You can easily use this to refer what has been installed in the pack.

## Before you begin

If you are on Windows, be sure to point the `-Dcarbon.home` property in the product's startup script (`wso2server.bat`) to the product's distribution home (e.g., `-Dcarbon.home=C:\Users\VM\Desktop\wso2as-5.2.1`). Alternatively, you can also set the `carbon.home` as a system property in Windows. Then, restart the server. Without this setting, you might not be able to install features through the management console.

## Installing features from the UI

Follow the instructions below to install new features to a WSO2 product:

1. Log into the product's Management Console.
2. On the **Configure** menu, click **Features**.  
The **Feature Management** page will appear.
3. Click **Available Features**.
4. Select a relevant repository. You can add the <http://product-dist.wso2.com/p2/carbon/releases/wilkes/> repository to get 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. For more information see, [Adding a Repository](#).
5. Some repositories contain multiple versions of features. If you are only interested in the latest versions, select the **Show only the latest versions** option.
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. For more information, see [Feature Categorization](#). 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.  
 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.

#### ***Viewing feature information***

When features of a repository are loaded, you can install them or view the corresponding feature details by clicking the **More Info** link associated with the respective feature. The **Feature Information** page will then appear with the following information:

- Name
- Identifier
- Version
- Provider
- Description
- Copyright
- License Agreement

Familiarize yourself with the information and click **Back** to load the previous page.

#### **Feature Categorization**

### **WSO2 Feature Repository:**

The WSO2 Feature Repository consists of features that are bundled into WSO2 products that are 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/>.

Feature repository provided by WSO2 contains a number of features that can be installed in products based on WSO2 Carbon. The logically related features have been categorized in the feature repository, to make it easier for the user to search and install the required features more effectively. These feature categories are grouped under products. There are high-level feature categories such as Application Server and Governance Registry. Under these product-based feature categories, there is another set of feature categories, which are based on the product features. The user has the option of installing either the whole product or only the required features of a product. The feature categorization can be seen on the Available Features page after selecting/adding the repository and thereafter, searching for features that are grouped by category.

### **Uninstalling Features**

1. Log in to the product's Management Console.

2. On the **Configure** menu, 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, select the **Select all in this page** option.
5. Click **Uninstall**.  
A page will appear containing details of the features to be uninstalled.
6. Verify the information and click **Next**.  
If the uninstallation is successful, 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

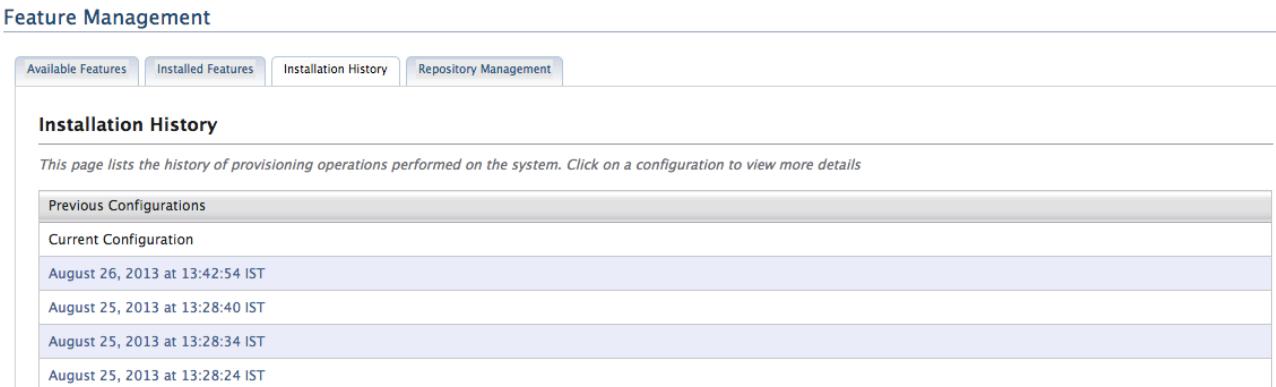
After installing features, if you encounter server issues or startup failures, you can revert the current configuration by restoring a previous one using either the management console or the command line. The latter is recommended if you cannot start the server.

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
- Reverting using the command line

### *Reverting using the management console*

1. Log in to the management console.
2. On the **Configure** menu, click **Features**.  
The **Feature Management** page appears.
3. Click on the **Installation History** tab.  
The **Installation History** page appears.



The screenshot shows the WSO2 Management Console interface. At the top, there is a navigation bar with tabs: Available Features, Installed Features, Installation History (which is currently selected and highlighted in blue), and Repository Management. Below the navigation bar, the title 'Installation History' is displayed. A sub-instruction below the title reads: 'This page lists the history of provisioning operations performed on the system. Click on a configuration to view more details'. A table follows, with columns for 'Previous Configurations' and 'Current Configuration'. The table contains five rows, each representing a configuration point in time:

Previous Configurations	Current Configuration
August 26, 2013 at 13:42:54 IST	August 26, 2013 at 13:42:54 IST
August 25, 2013 at 13:28:40 IST	August 25, 2013 at 13:28:40 IST
August 25, 2013 at 13:28:34 IST	August 25, 2013 at 13:28:34 IST
August 25, 2013 at 13:28:24 IST	August 25, 2013 at 13:28:24 IST

4. Click on the configuration to which you wish to revert.  
The actions that take place when reverting to the selected configuration is listed below.

## Feature Management

The screenshot shows the 'Feature Management' interface with the 'Repository Management' tab selected. A modal window titled 'Reverting to Previous Configuration - August 26, 2013 at 13:42:54 IST' is open. It contains a message stating 'Reverting to the Selected Configuration, would results in performing following actions on the Current Configuration.' Below this, it says 'Following features will be uninstalled' and lists two features in a table:

Name	Version	ID	Provider
WSO2 Carbon - Doc Request Processor Feature	4.2.0	org.wso2.carbon.docrequestprocessor.feature.group	WSO2 Inc.
WSO2 Carbon - API Store Feature	4.2.0	org.wso2.carbon.apimgt.store.feature.group	WSO2 Inc.

At the bottom of the modal are '< Back' and 'Revert' buttons.

- 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:

- Start your product with the `-D osgiConsole` system property.
- Once the server is started, type the following command:  
`osgi> getInstallationHistory`

You will get the following list of states :

```
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
1376883743493 August 19, 2013 at 09:12:23 IST
1376933879416 August 19, 2013 at 23:07:59 IST
1376940017503 August 20, 2013 at 00:50:17 IST
```

- Check what features are installed and uninstalled in a given state, by entering the following command:  
`osgi> getInstallationHistory <timestamp>`

```
F o r
osgi> getInstallationHistory
```

```
e x a m p l e :
1376933879416
```

The output will be as follows:

```
-- Installed features in this configuration

-- Uninstalled features in this configuration
WSO2 Carbon - Service Management Feature 4.2.0
WSO2 Stratos - Deployment Features 2.2.0
WSO2 Stratos - Common Composite Feature 2.2.0
WSO2 Stratos - Usage Agent Feature 2.2.0
WSO2 Stratos - Throttling Agent Feature 2.2.0
WSO2 Stratos AppServer - Dashboard UI Features 2.2.0
WSO2 Stratos AppServer - Dashboard UI Features 2.2.0
```

4. Decide the state to which you need to revert the system, and thereafter use the following command:

```
osgi> revert <timestamp>
```

For

```
osgi>
```

example:

```
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 sever/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 inactivity 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 \
-Dtenant.idle.time=30 \
```

## 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:

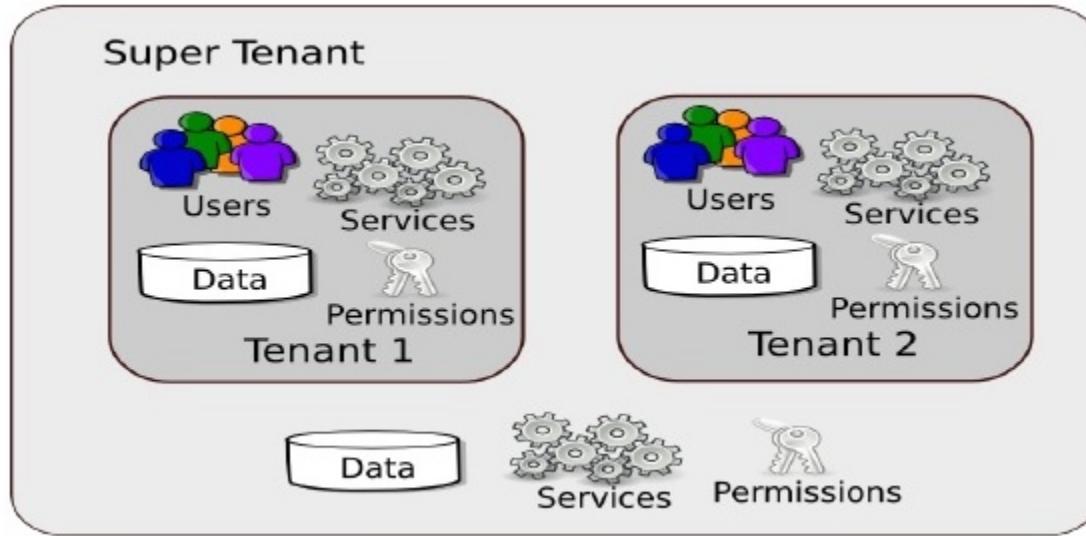
- **Tenant isolation:** Each tenant has its own domain, which the other tenants cannot access.
- **Data isolation:** Each tenant can manage its data securely in an isolated manner.
- **Execution isolation:** Each tenant can carry out business processes and workflows independent of the other tenants. 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 multitenancy:** This method involves allowing all the tenants to share a single JVM instance. This method minimises the overhead cost.

## Lazy loading

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.

### Tenants

Lazy loading of tenants is a feature that is built into all WSO2 products. This feature ensures that all the tenants are not loaded at the time the server starts in an environment with multiple tenants. Instead, they are loaded only when a request is made to a particular tenant. If a tenant is not utilized for a certain period of time (30 minutes by default), it will be unloaded from the memory.

You can change the default time period allowed for tenant inactivity by adding `-Dtenant.idle.time=<time_in_minutes>` java property to the startup script of the product (`./wso2server.sh` file for Linux and `wso2server.bat` for Windows) as shown below.

```
JAVA_OPTS \
-Dtenant.idle.time=30 \
```

## Artifacts

Lazy loading of artifacts is a feature that is used by some WSO2 products, which can be enabled in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file. The deployer that handles lazy loading of artifacts is called the `GhostDeployer`. A flag to enable or disable the `Ghost Deployer` is shown below. This is set to `false` by default because the `Ghost Deployer` works only with the `HTTP/S` transports. Therefore, if other transports are used, the `Ghost Deployer` does not have to be enabled.

```
<GhostDeployment>
<Enabled>false</Enabled>
<PartialUpdate>false</PartialUpdate>
</GhostDeployment>
```

When a stand-alone WSO2 product instance is started with lazy loading enabled, its services, applications and other artifacts are not deployed immediately. They are first loaded in the Ghost form and the actual artifact is deployed only when a request for the artifact is made. In addition, if an artifact has not been utilized for a certain period of time, it will be unloaded from the memory.

When lazy loading of artifacts is enabled for PaaS deployments, lazy loading applies both for tenants as well as a tenant artifacts. As a result, lazy loading is applicable on both levels for a tenant in a cloud environment. Therefore, the associated performance improvements and resource utilization efficiencies are optimal.

## **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.

- **Lazy Loading:** Lazy loading is a design pattern used specifically in cloud deployments to prolong the initialisation of an object until it is requested by a tenant or an internal process. Lazy loading of tenants is a feature that is built into all WSO2 products, which ensures that in an environment with multiple tenants, all tenants are not loaded at the time the server starts. Instead, they are loaded only when a request is made to a particular tenant. If a tenant is not utilised for a certain period of time, it will be unloaded from memory.
- **Eager Loading:** Eager loading is a design pattern commonly used in computer programming. As opposed to lazy loading, eager loading initialises an object upon creation. If Eager loading is enabled in WSO2 products, existing tenants will be loaded without any delays when the Carbon server starts. You can also enable this feature for specific tenants, so that only the required tenants will be loaded when the server starts.

Lazy Loading is enabled as the tenant loading policy in WSO2 products by default. You can either configure the default setting (Lazy Loading) or replace it with Eager Loading.

See the following topics for instructions:

- Enabling eager loading
- Configuring lazy loading
  - Configuring the lazy loading idle time
  - Lazy loading of artifacts

### *Enabling eager loading*

By default, lazy loading is enabled as the tenant loading policy. Note that eager loading only applies to tenants and not to artifacts. You can change this to eager loading as follows:

1. Open the `carbon.xml` file from the `<PRODUCT_HOME>/repository/conf/` directory. Enable the `<EagerLoading>` element and comment out `<LazyLoading>`.
2. Enable the `<EagerLoading>` element and comment out `<LazyLoading>` as shown below.

```

<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>.

### **Configuring lazy loading**

If you are using the default tenant loading policy (lazy loading), you can configure the following:

- Configuring the lazy loading idle time
- Lazy loading of artifacts

### **Configuring the lazy loading idle time**

You have the option of setting the required tenant idle time for lazy loading as explained below. 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. Note that configuration only applies to lazy loading of tenants and not for artifacts.

Follow the steps given below.

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.

```
<Tenant>
    <LoadingPolicy>
        <LazyLoading>
            <IdleTime>30</IdleTime>
        </LazyLoading>!-->
        <!--<EagerLoading>
            <Include>*,!foo.com,!bar.com</Include>
        </EagerLoading>-->
    </LoadingPolicy>
</Tenant>
```

### **Lazy loading of artifacts**

If required, you can enable lazy loading for artifacts by using the following code segment in the `carbon.xml` file. Note that you cannot enable `eager loading` for artifacts.

```
<GhostDeployment>
    <Enabled>false</Enabled>
</GhostDeployment>
```

Lazy loading of artifacts is a feature that is used by some Carbon products. The deployer that handles lazy loading is called the GhostDeployer. By default, this is set to 'false' because the Ghost Deployer works only with the HTTP/S transports. Therefore, if other transports are used, we do not have to enable the Ghost Deployer. When lazy loading of artifacts is enabled for PaaS deployments, lazy loading applies both for tenants as well as a tenant's artifacts. As a result, for a tenant in a cloud environment, lazy loading is applicable on both levels. The services,

applications and other artifacts are first loaded in ghost form. The actual artifact is deployed only when a request for the artifact is made. Also, if an artifact has not been utilized for a certain period of time, it will be unloaded from memory. Therefore, the associated performance improvement and resource utilization efficiencies are optimal.

## Lazy Loading Implementation

Out of four popular variants (Lazy Initialization, Virtual Proxy, Value Holder, Ghost) of the lazy loading design pattern, the "Value Holder" and "Ghost" variants are adopted and used in the lazy loading implementation in WSO2 Carbon Framework.

- [Use of Value Holder](#)
- [Use of GhostDeployer and GhostDispatcher/GhostWebappDeployerValve](#)

### Use of Value Holder

A value holder is an object, usually (but not necessarily) with a method by the name `getValue`, which the clients invoke to obtain a reference to the real object corresponding to a parameter passed in the method. The `ValueHolder` variant is used at the event of loading a tenant. The called method is as follow:

```
TenantAxisUtils.getTenantConfigurationContext(String tenantDomain)
```

The `getTenantConfigurationContext` method is the equivalent of the `getValue` method in the `ValueHolder` variant. It takes a `tenantDomain` as a reference and returns the `ConfigurationContext`, if the tenant is already loaded. If the tenant domain does not have an Axis2 `ConfigurationContext`, the method will return a newly created Axis2 `ConfigurationContext`. In both the latter mentioned methods, when loading an entire `AxisConfiguration`, all artifacts belonging to that particular tenant are also loaded. A more efficient practice is to lazy load those artifacts, by loading them only when it is requested by the system or a tenant. This prevents processing overhead and unnecessary memory usage. This is where the `Ghost` variant of the lazy loading pattern is used.

### Use of GhostDeployer and GhostDispatcher/GhostWebappDeployerValve

When the server starts, you can see all the previously deployed services and web applications listed in its management console. However, these artifacts are only the `Ghost` instances that maintain references to the actual instances. `GhostDeployer` that handles lazy loading of artifacts, holds a map of deployers for all artifact types.

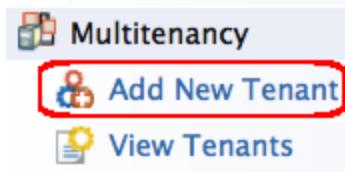
When a new artifact is deployed, the `GhostDeployer` creates a `Ghost Axis2 Service` containing a special parameter (`Ghost Parameter`) to distinguish the service as a `Ghost` service, and then registers that service into the `AxisConfig`. When a user requests for a particular service artifact, the `GhostDispatcher`, which is an Axis2 dispatching handler, checks the `Ghost` parameter added to the service to determine if it is the actual artifact or the ghost form. If it is the latter, the actual deployer is called to load the actual service, the rest of the metadata from the registry, service policies and other information regarding the artifact.

`GhostWebappDeployerValve`, on the other hand, is used with lazy loading for web applications. It is a Tomcat valve that intercepts the incoming http requests, and does the same work as the `GhostDispatcher` in Axis2. It intercepts the incoming web app request, checks for the `Ghost` parameter in the webapp and loads the actual web application if it is found to be in ghost form.

## Adding New Tenants

You can add a new tenant in the management console and then view it by following the procedure below. In order to add a new tenant, you should be logged in as a super user.

1. Click **Add New Tenant** in the **Configure** tab of your product's management console.



2. Enter the tenant information in **Register A New Organization** screen as follows, and click **Save**.

Parameter Name	Description
<b>Domain</b>	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.
<b>Select Usage Plan for Tenant</b>	The usage plan defines limitations (such as number of users, bandwidth etc.) for the tenant.
<b>First Name/Last Name</b>	The name of the tenant admin.
<b>Admin Username</b>	The login username of the tenant admin. The username always ends with the domain name (e.g., admin@abc.com)
<b>Admin Password</b>	The password used to log in using the admin username specified.
<b>Admin Password (Repeat)</b>	Repeat the password to confirm.
<b>Email</b>	The email address of the admin.

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**.

Tenants List				
Domain	Email	Created Date	Active	Edit
wso2.com	frankie.avalon@gmail.com	2014/11/17 12:03:06	<input checked="" type="checkbox"/>	Edit
abc.com	dean.martin@gmail.com	2014/11/17 13:43:46	<input checked="" type="checkbox"/>	Edit

## 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:

<b>Name:</b>	WSO2	Carbon	-	Transport	Management	Feature
<b>Identifier:</b>	<code>org.wso2.carbon.transport.mgt.feature.group</code>					

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 transport receiver implementation of the HTTP transport is available in the Carbon core component. The transport sender implementation comes from the Tomcat http connector. This transport is shipped with WSO2 Carbon and all WSO2 Carbon-based products, which use this transport as the default transport, except WSO2 ESB. By default, we use non-blocking Tomcat Java connector, `org.apache.coyote.http11.Http11NioProtocol`.

- This is a non-blocking HTTP transport implementation, which means that I/O the threads does not get blocked while received messages are processed.
- Although the `axis2.xml` file contains configurations for HTTP/S transports by default, they are not used by WSO2 products. Instead, the products use the HTTP/S transport configurations in Tomcat-level; therefore, changing the HTTP/S configurations in the `axis2.xml` file has no effect.

In the 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 put into the transport configurations.

### Transport Connector Parameters

Parameter Name	Description	Required	Possible Values	Default Value
port	The port number on which this transport receiver should listen for incoming messages.	Yes	A positive integer less than 65535	
proxyPort	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 in the proxy server. All the WSDLs generated will contain the proxy port value as the listener port.	No	A positive integer less than 65535	

HTTP servlet transport should be configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file. The transport class that should be specified in the `catalina-server.xml` file is as follows:

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol" />
```

Parameter Name	Description	Possible Values	Default Value

port	The port over which this transport receiver listens for incoming messages.	A positive integer less than 65535	9763 for HTTP Connector 9443 for HTTPS Connector
redirectPort	If this Connector is supporting non-SSL requests, and a request is received for which a matching <security-constraint> requires SSL transport, Catalina will automatically redirect the request to the port number specified here.	A positive integer less than 65535	9443
bindOnInit	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 <code>false</code> , the socket will be bound when the connector is started and unbound when it is stopped.		false
proxyPort	When used, this transport listener will accept messages arriving through a HTTP proxy server which listens on the specified proxy port. Apache <code>mod_proxy</code> should be enabled on the proxy server. All the WSDLs generated will contain the proxy port value as the listener port.	A positive integer less than 65535	
maxHttpHeaderSize	The maximum size of the HTTP request and response header in bytes.	A positive integer	4096
acceptorThreadCount	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.		2
maxThreads	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.	A positive integer	40
minSpareThreads	The minimum number of threads always kept running. If not specified, the default will be used.		50
enableLookups	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.	<code>true, false</code>	false

disableUploadTimeout	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.	<i>true, false</i>	true
connectionUploadTimeout	Specifies the timeout, in milliseconds, to use while a data upload is in progress. This only takes effect if <code>disableUploadTimeout</code> is set to <code>false</code> .		
clientAuth	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.	<i>true, false, want</i>	false
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.	-1 or any positive integer	100
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.	A positive integer	10
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.	Any string	WSO2 Carbon Server
compression	<p>The <b>Connector</b> may use HTTP/1.1 GZIP compression in an attempt to save server bandwidth.</p> <p>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".</p>	<i>on, off, force</i>	off
compressionMinSize	If <b>compression</b> is set to "on" then this attribute may be used to specify the minimum amount of data before the output is compressed.	A positive integer	2048

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.	A comma separated list of regular expressions	empty string
compressableMimeType	Use this parameter to indicate a list of MIME types for which HTTP compression may be used.	A comma separated list of valid mime types	text/html, text/xml, text/plain
URIEncoding	This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL.	URI encoding Character set name	ISO-8859-1

This servlet transport implementation can be further tuned up using the following parameters.

This is only a subset of all 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 a complete list of supported parameters, see [Apache Tomcat's connector configuration reference](#).

### Transport Sender Parameters

Parameter Name	Description	Required	Possible Values	Default Value
PROTOCOL	The version of HTTP protocol to be used for outgoing messages.	No	HTTP/1.0, HTTP/1.1	HTTP/1.1
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 out bound connections.	No	A positive integer	60000 ms
ConnectionTimeout	The connection timeout value in milliseconds, for out bound connections.	No	A positive integer	60000 ms
OmitSOAP12Action	Set this parameter to "true" if you need to disable the soapaction 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.

- See the example given below where two connectors are defined in the `catalina-server.xml` file.

```
<!-- Connector using port 9763 -->
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
    port="9763"
    .....
    ....../>
<!-- Connector using port 9764 -->
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
    port="9764"
    .....
    ....../>
```

- 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 consists of a receiver implementation which comes from the Carbon core component and a sender implementation which comes from the Apache Axis2 transport module. In fact, this transport uses exactly the same transport sender implementation as the HTTP transport. So the two classes that should be specified in the configuration are `org.wso2.carbon.core.transports.http.HttpsTransportListener` and `org.apache.axis2.transport.http.CommonsHTTPTransportSender` for the receiver and sender in the specified order. The configuration parameters associated with the receiver and the sender are the same as in HTTP transport. This is also a blocking transport implementation.

However, when using the following class as the receiver implementation, we need to specify the servlet HTTPS transport configuration in the transport's XML file.

- `org.wso2.carbon.core.transports.http.HttpsTransportListener`

The class that should be specified as the transport implementation is `org.wso2.carbon.server.transports.http.HttpsTransport`.

### Transport connector parameters

In addition to the configuration parameters supported by the [HTTP servlet transport](#), HTTPS servlet transport supports the following configuration 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.

Parameter Name	Description	Required	Possible Values	Default Value
<code>sslProtocol</code>	Transport level security protocol to be used.	No	<i>TLS, SSL</i>	<i>TLS</i>

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	

Similar to the servlet HTTP transport, this transport is also based on Apache Tomcat's connector implementation. Please refer Tomcat *connector configuration reference* for a complete list of supported parameters.

### 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.

- See the example given below where two connectors are defined in the `catalina-server.xml` file.

```
<!-- Connector using port 9763 -->
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9763"
           .....
           ....../>
<!-- Connector using port 9764 -->
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9764"
           .....
           ....../>
```

- 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.HttpCoreNIOListener` 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.

Parameter Name	Description	Required	Possible Values	Default Value
port	The port on which this transport receiver should listen for incoming messages.	No	A positive integer less than 65535	8280
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	<i>true</i>	
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 <code>WSDLEPRPrefix</code> 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 <code>&lt;protocol&gt;://&lt;hostname&gt;:&lt;port&gt;/</code>	

### Transport sender parameters

Parameter Name	Description	Required	Possible Values	Default Value
http.proxyHost	If the outgoing messages should be sent through an HTTP proxy server, use this parameter to specify the target proxy.	No	A host name or an IP address	

http.proxyPort	The port through which the target proxy accepts HTTP traffic.	No	A positive integer less than 65535	
http.nonProxyHosts	The list of hosts to which the HTTP traffic should be sent directly without going through the proxy.	No	A list of host names or IP addresses separated by ' '	
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	<i>true</i>	

## 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.HttpCoreNIOSSLListener
```

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):

Parameter Name	Description	Required	Possible Values	Default Value
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.	Yes	<pre>&lt;parameter name="keystore"&gt; &lt;KeyStore&gt; &lt;Location&gt;lib/identity.jks&lt;/Location&gt; &lt;Type&gt;JKS&lt;/Type&gt; &lt;Password&gt;password&lt;/Password&gt; &lt;KeyPassword&gt;password&lt;/KeyPassword&gt;  &lt;/KeyStore&gt; &lt;/parameter&gt;</pre>	

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.	Yes	<pre>&lt;parameter name="truststore"&gt; &lt;TrustStore&gt; &lt;Location&gt;lib/identity.jks&lt;/Location&gt; &lt;Type&gt;JKS&lt;/Type&gt; &lt;Password&gt;password&lt;/Password&gt; &lt;/TrustStore&gt; &lt;/parameter&gt;</pre>	
------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

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.

Parameter Name	Description	Required	Possible Values	Default Value
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	<pre>&lt;parameter name="customSSLProfiles"&gt; &lt;profile&gt; &lt;servers&gt;www.test.org:80, www.test2.com:9763&lt;/servers&gt; &lt;KeyStore&gt; &lt;Location&gt;/path/to/identity/store &lt;/Location&gt; &lt;Type&gt;JKS&lt;/Type&gt; &lt;Password&gt;password&lt;/Password&gt;  &lt;KeyPassword&gt;password &lt;/KeyPassword&gt; &lt;/KeyStore&gt; &lt;TrustStore&gt; &lt;Location&gt;/path/to/trust/store &lt;/Location&gt; &lt;Type&gt;JKS&lt;/Type&gt; &lt;Password&gt;password&lt;/Password&gt;  &lt;/TrustStore&gt; &lt;/profile&gt; &lt;/parameter&gt;</pre>	

## 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

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.

Parameter Name	Description	Required	Possible Values	Default Value
transport.vfs.FileURI	The file URL from where the input files should be fetched.	Yes	A valid file URL of the form <code>file://file://&lt;path&gt;</code>	
transport.vfs.ContentType	Content type of the files transferred over the transport.	Yes	A valid content type for the files (e.g., <code>text/xml</code> )	
transport.vfs.FileNamePattern	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.	No	A regular expression to select files by name (e.g., <code>*.xml</code> )	
transport.PollInterval	The polling interval in milliseconds for the transport receiver to poll the file URI location.	No	A positive integer	

transport.vfs.ActionAfterProcess	Action to perform over the files after processed by the transport.	No	<i>MOVE, DELETE</i>	DELETE
transport.vfs.ActionAfterFailure	Action to perform over the files after processed by the transport.	No	<i>MOVE, DELETE</i>	DELETE
transport.vfs.MoveAfterProcess	The location to move the files after processing.	Required if ActionAfterProcess is MOVE	A valid file URI	
transport.vfs.MoveAfterFailure	The location to move the files after a failure occurs.	Required if ActionAfterFailure is MOVE	A valid file URI	
transport.vfsReplyFileURI	The location to which reply files should be written by the transport.	No	A valid file URI	
transport.vfsReplyFileName	The name for reply files written by the transport.	No	A valid file name	response.xml
transport.vfs.MoveTimestampFormat	The pattern/format of the timestamps added to file names as prefixes when moving files (See the API documentation of <code>java.text.SimpleDateFormat</code> for details).	No	A valid timestamp pattern (e.g., <code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code> )	
transport.vfsStreaming	If files should be transferred in streaming mode or not.	No	true, false	false
transport.vfsReconnectTimeout	Reconnect timeout value in seconds to be used in case of an error when transferring files.	No	A positive integer	30 sec

transport.vfs.MaxRetryCount	Maximum number of retry attempts to carry out in case of errors.	No	A positive integer	3
transport.vfs.Append	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.	No	true, false	false
transport.vfs.MoveAfterFailedMove	New destination to move the failed file.	No	A valid file URI	
transport.vfs.FailedRecordsFileName	The file name to maintain the list of failure files.	No	A valid file name	vfs-move-failed-records.properties
transport.vfs.FailedRecordsFileDestination	The destination of the failed file.	No	A folder URI	repository/conf/
transport.vfs.MoveFailedRecordTimestampFormat	When adding a record to the failed file, entries are logged as: <code>file_name time_stamp</code> . This will configure the time stamp format.	No	A valid timestamp pattern (eg: yyyy-MM-dd'T'HH:mm:ss.SSSZ)	dd-MM-yyyy HH:mm:ss
transport.vfs.FailedRecordNextRetryDuration	The time in millisecond for the move task to wait until next retry.	No	A positive integer	3000 milliseconds

You can find a VFS transport implementation in this tutorial: <http://wso2.org/library/tutorials/2011/01/sftp-file-transer-wso2-esb>

## JMS Transport

JMS (Java Message Service) transport implementation also comes from the WS-Commons Transports project. All the relevant classes are packed into the `axis2-transport-jms-<version>.jar` and the following classes act as the transport receiver and the sender respectively.

- `org.apache.axis2.transport.jms.JMSListener`
- `org.apache.axis2.transport.jms.JMSSender`

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:

```
<parameter name="myTopicConnectionFactory">
    <parameter
        name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitia
        lContextFactory</parameter>
    <parameter
        name="java.naming.provider.url">tcp://localhost:61616</parameter>
    <parameter
        name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</par
        ameter>
    <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.

Parameter Name	Description	Required	Possible Values
----------------	-------------	----------	-----------------

java.naming.factory.initial	JNDI initial context factory class. The class must implement the <code>java.naming.spi.InitialContextFactory</code> interface.	Yes	A \
java.naming.provider.url	URL of the JNDI provider.	Yes	A \
java.naming.security.principal	JNDI Username.	No	
java.naming.security.credentials	JNDI password.	No	
transport.Transactionality	Desired mode of transactionality.	No	<i>no</i>
transport.UserTxnJNDIName	JNDI name to be used to require user transaction.	No	
transport.CacheUserTxn	Whether caching for user transactions should be enabled or not.	No	<i>true</i>
transport.jms.SessionTransacted	Whether the JMS session should be transacted or not.	No	<i>true</i>
transport.jms.SessionAcknowledgement	JMS session acknowledgment mode.	No	<i>AL CL DL SE</i>
transport.jms.ConnectionFactoryJNDIName	The JNDI name of the connection factory.	Yes	
transport.jms.ConnectionFactoryType	Type of the connection factory.	No	<i>que</i>
transport.jms.JMSSpecVersion	JMS API version.	No	1.1
transport.jms.UserName	The JMS connection username.	No	
transport.jms.Password	The JMS connection password.	No	
transport.jms.Destination	The JNDI name of the destination.	No	
transport.jms.DestinationType	Type of the destination.	No	<i>que</i>
transport.jms.DefaultReplyDestination	JNDI name of the default reply destination.	No	
transport.jms.DefaultReplyDestinationType	Type of the reply destination.	No	<i>que</i>

transport.jms.MessageSelector	Message selector implementation.	No	
transport.jms.SubscriptionDurable	Whether the connection factory is subscription durable or not.	No	true
transport.jms.DurableSubscriberName	Name of the durable subscriber.	Yes if the subscription durable is turned on	
transport.jms.PubSubNoLocal	Whether the messages should be published by the same connection they were received.	No	true
transport.jms.CacheLevel	JMS resource cache level.	No	no correlation
transport.jms.ReceiveTimeout	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.	No	None waiting
transport.jms.ConcurrentConsumers	Number of concurrent threads to be started to consume messages when polling.	No	Any thread
transport.jms.MaxConcurrentConsumers	Maximum number of concurrent threads to use during polling.	No	Any thread
transport.jms.IdleTaskLimit	The number of idle runs per thread before it dies out.	No	Any thread
transport.jms.MaxMessagesPerTask	The maximum number of successful message receipts per thread.	No	Any thread
transport.jms.InitialReconnectDuration	Initial reconnection attempts duration in milliseconds.	No	Any thread
transport.jms.ReconnectProgressFactor	Factor by which the reconnection duration will be increased.	No	Any thread
transport.jms.MaxReconnectDuration	Maximum reconnection duration in milliseconds.	No	

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

Parameter Name	Description	Required	Possible Values	Default Value
transport.jms.ConnectionFactory	Name of the JMS connection factory the service should use.	No	A name of an already defined connection factory	default
transport.jms.PublishEPR	JMS EPR to be published in the WSDL.	No	A JMS EPR	

## 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.

Parameter Name	Description	Required	Possible Values	Default Value
transport.mail.Address	The mail address from which this service should fetch incoming mails.	Yes	A valid e-mail address	

transport.mail.Folder	The mail folder in the server from which the listener should fetch incoming mails.	No	A valid mail folder name (e.g., inbox)	inbox folder if that is available or else the root folder
transport.mail.Protocol	The mail protocol to be used to receive messages.	No	<i>pop3, imap</i>	imap
transport.mail.PreserveHeaders	A comma separated list of mail header names that this receiver should preserve in all incoming messages.	No	A comma separated list	
transport.mail.RemoveHeaders	A comma separated list of mail header names that this receiver should remove from incoming messages.	No	A comma separated list	
transport.mail.ActionAfterProcess	Action to perform on the mails after processing them.	No	<i>MOVE, DELETE</i>	DELETE
transport.mail.ActionAfterFailure	Action to perform on the mails after a failure occurs while processing them.	No	<i>MOVE, DELETE</i>	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	<i>true, false</i>	false
transport.ConcurrentPollingAllowed	Whether the receiver should poll for multiple messages concurrently.	No	<i>true, false</i>	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.

Parameter Name	Description	Required	Possible Values	Default Value
transport.mail.SMTPBccAddresses	If one or more e-mail addresses need to be specified as BCC addresses for outgoing mails, this parameter can be used.	No	A comma separated list of e-mail addresses	
transport.mail.Format	Format of the outgoing mail.	No	<i>Text, Multi part</i>	Text

## 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

Parameter Name	Description	Required	Possible Values	Default Value
port	The port on which the TCP server should listen for incoming messages	No	A positive integer less than 65535	8000
hostname	The host name of the server to be displayed in WSDLs etc	No	A valid host name or an IP address	

### 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:

`samples/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, simple 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.

```
<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>. 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.

Parameter Name	Description	Required	Possible Values	Default Value
transport.fix.AcceptorConfigURL	URL to the Quickfix/J acceptor configuration file (see notes below).	Required for receiving messages over FIX	A valid URL	
transport.fix.InitiatorConfigURL	URL to the Quickfix/J initiator configuration file (see notes below).	Required for sending messages over FIX	A valid URL	
transport.fix.AcceptorLogFactory	Log factory implementation to be used for the FIX acceptor (Determines how logging is done at the acceptor level).	No	<i>console, file, jdbc</i>	Logging disabled
transport.fix.InitiatorLogFactory	Log factory implementation to be used for the FIX acceptor (Determines how logging is done at the acceptor level).	No	<i>console, file, jdbc</i>	Logging disabled
transport.fix.AcceptorMessageStore	Message store mechanism to be used with the acceptor (Determines how the FIX message store is maintained).	No	<i>memory, file, sleepycat, jdbc</i>	memory
transport.fix.InitiatorMessageStore	Message store mechanism to be used with the initiator (Determines how the FIX message store is maintained).	No	<i>memory, file, sleepycat, jdbc</i>	memory

transport.fix.ResponseDeliverToCompID	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.	No		
transport.fix.ResponseDeliverToSubID	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.	No		
transport.fix.ResponseDeliverToLocationID	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.	No		
transport.fix.SendAllToInSequence	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.	No	<i>true, false</i>	true
transport.fix.BeginStringValidation	Whether the transport should validate BeginString values when forwarding FIX messages across sessions.	No	<i>true, false</i>	true

transport.fix. DropExtraResponses	In situation where the FIX recipient sends multiple responses per request use this parameter to drop excessive responses and use only the first one.	No	<i>true, false</i>	false
--------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------	----	--------------------	-------

## 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

### 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_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.core.rmem_default = 524288
net.core.wmem_default = 524288
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.ip_local_port_range = 1024 65535
```

When 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).

```
* soft nproc 20000
* hard nproc 20000
```

## 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 expansion limit` 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.

```
-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](#).

Property	Description	Tuning Recommendations
maxActive	<p>The maximum number of active connections that can be allocated from the connection pool at the same time. The default value is 100.</p>	<p>The maximum latency (approximately) = <math>(P / M) * T</math> , where,</p> <ul style="list-style-type: none"> <li>• M = maxActive value</li> <li>• P = Peak concurrency value</li> <li>• T = Time (average) taken to process a query.</li> </ul> <p>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.</p> <p>Note that this value should not exceed the maximum number of requests allowed for your database.</p>
maxWait	<p>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.</p>	<p>Adjust this to a value slightly higher than the maximum latency for a request, so that a buffer time is added to the maximum latency. That is,</p> <p>If the maximum latency (approximately) = <math>(P / M) * T</math> , where,</p> <ul style="list-style-type: none"> <li>• M = maxActive value,</li> <li>• P = Peak concurrency value,</li> <li>• T = Time (average) taken to process a query,</li> </ul> <p>then, the maxWait = <math>(P / M) * T + \text{buffer time}</math>.</p>
minIdle	<p>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.</p>	<p>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.</p>
maxIdle	<p>The maximum number of connections that can remain idle in the pool.</p>	<p>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.</p>

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.	<p>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.</p> <p>If a larger value is set, the frequency of executing the Validation Query is low, which results in better performance. Note that this value can be as high as the time it takes for your DBMS to declare a connection as stale. For example, MySQL will keep a connection open for as long as 8 hours, which requires the validation interval to be within that range. However, note that the validation query execution is usually fast. Therefore, even if this value is only large by a few seconds, there will not be a big penalty on performance. Also, specially when the database requests have a high throughput, the negative impact on performance is negligible. For example, a single extra validation query run every 30 seconds is usually negligible.</p> <p>If a smaller value is set, a stale connection will be identified quickly when it is presented. This maybe important if you need connections repaired instantly, e.g. during a database server restart.</p>
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.

When it comes to web applications, users are free to experiment and package their own pooling framework such BoneCP.

### Caching configuration

The <Cache> element configured in the carbon.xml file (stored in the <PRODUCT\_HOME>/repository/conf dir

actory) 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.

```
<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.

```
<Valve
    className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionVal
    ve" threshold="600" />
```

- The `className` is the Java class name used for the implementation. This must be set to `org.wso2.carbo
n.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.

# 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 Product Deployment](#)
- [Monitoring SOAP Messages](#)
- [Monitoring TCP-Based Messages](#)
- [Monitoring with AppDynamics](#)
- [Monitoring with WSO2 Carbon Metrics](#)

## 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)
  - <PRODUCT\_HOME>, <JAVA\_HOME>
  - configuration files
  - log files
  - H2 database files
- Thread dump
- Checksum values of all the files found in the \$CARBON\_HOME

## 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 the DataSource MBean
  - Using product-specific MBeans

### 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 (RMIServerPort 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.

```
<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**.

```
<JMX xmlns="http://wso2.org/projects/carbon/jmx.xml">
    <StartRMIServer>true</StartRMIServer>
    <!-- HostName, or Network interface to which this RMI server should be
        bound -->
    <HostName>localhost</HostName>
    <!-- ${Ports.JMX.RMIRegistryPort} is defined in the Ports section of
        the carbon.xml-->
    <RMIRegistryPort>${Ports.JMX.RMIRegistryPort}</RMIRegistryPort>
    <!-- ${Ports.JMX.RMIServerPort} is defined in the Ports section of
        the carbon.xml-->
    <RMIServerPort>${Ports.JMX.RMIServerPort}</RMIServerPort>
</JMX>
```

#### ***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).

```

<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;LO
CK_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.

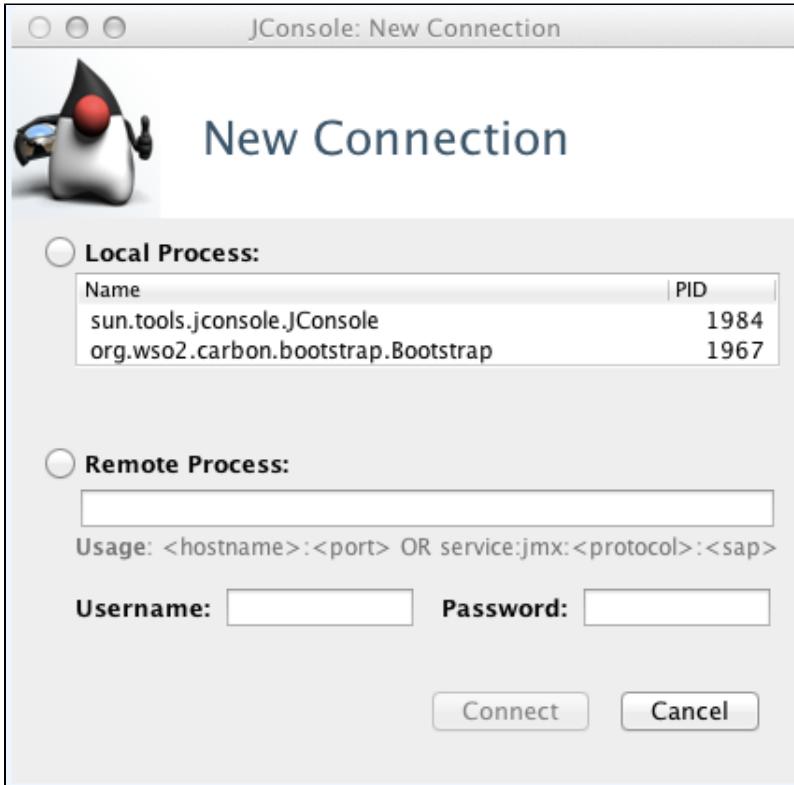
```

INFO {org.wso2.carbon.core.init.CarbonServerManager} - JMX
Service URL :
service:jmx:rmi://<your-ip>:11111/jndi/rmi://<your-ip>:9999/jmx
rmi

```

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:

1. 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](#).

### 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 (wso2server.sh for Linux and wso2server.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.

2. 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.

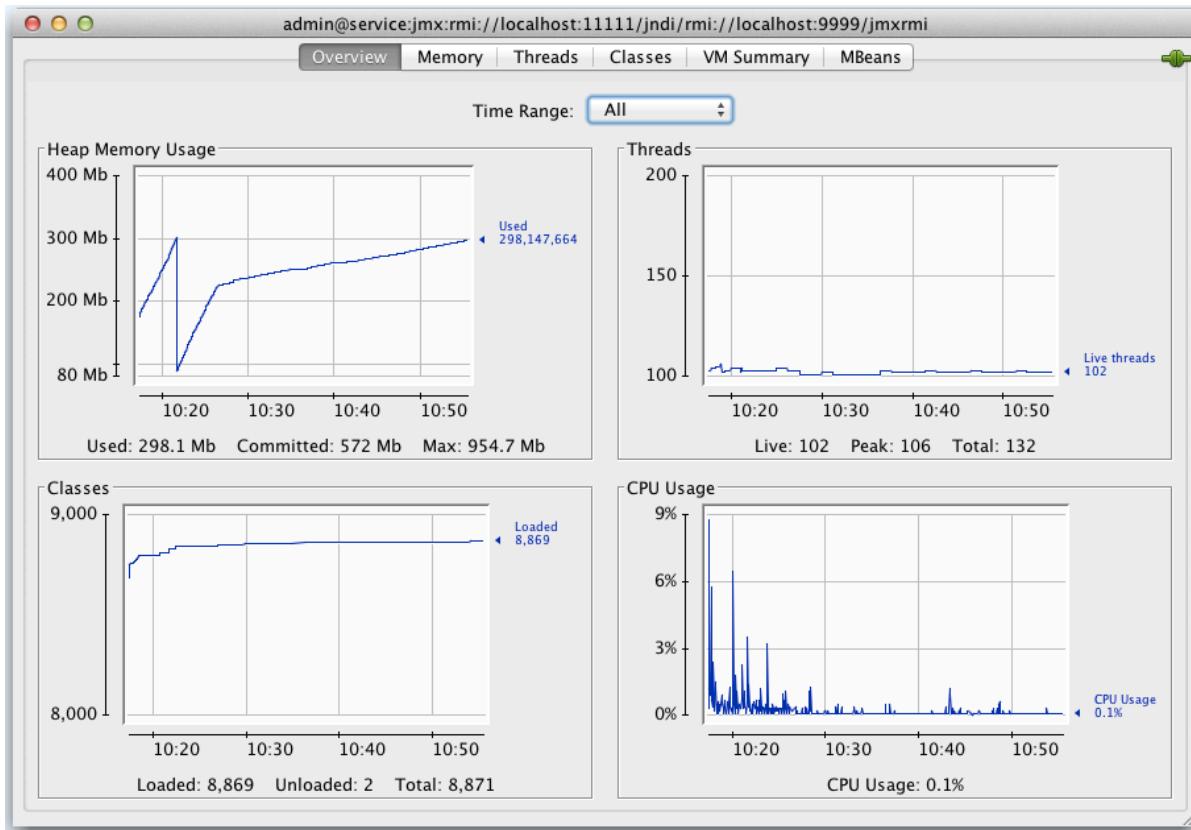
Make sure that the user ID you are using for JMX monitoring is assigned a role that has the **Se**

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.

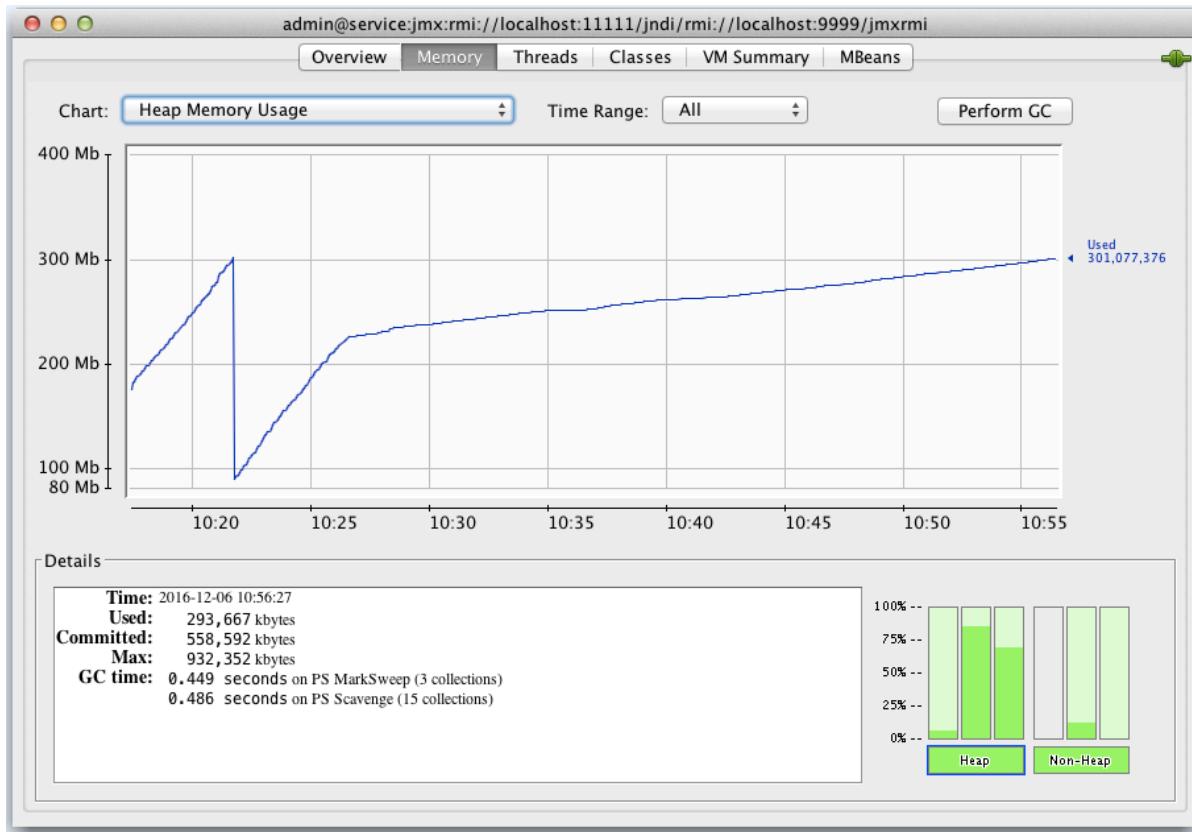
4. Click **Connect** to open the **Java Monitoring & Management Console**. The following tabs will be available:

- [Overview](#)
- [Memory](#)
- [Threads](#)
- [Classes](#)
- [VM](#)
- [MBeans](#)

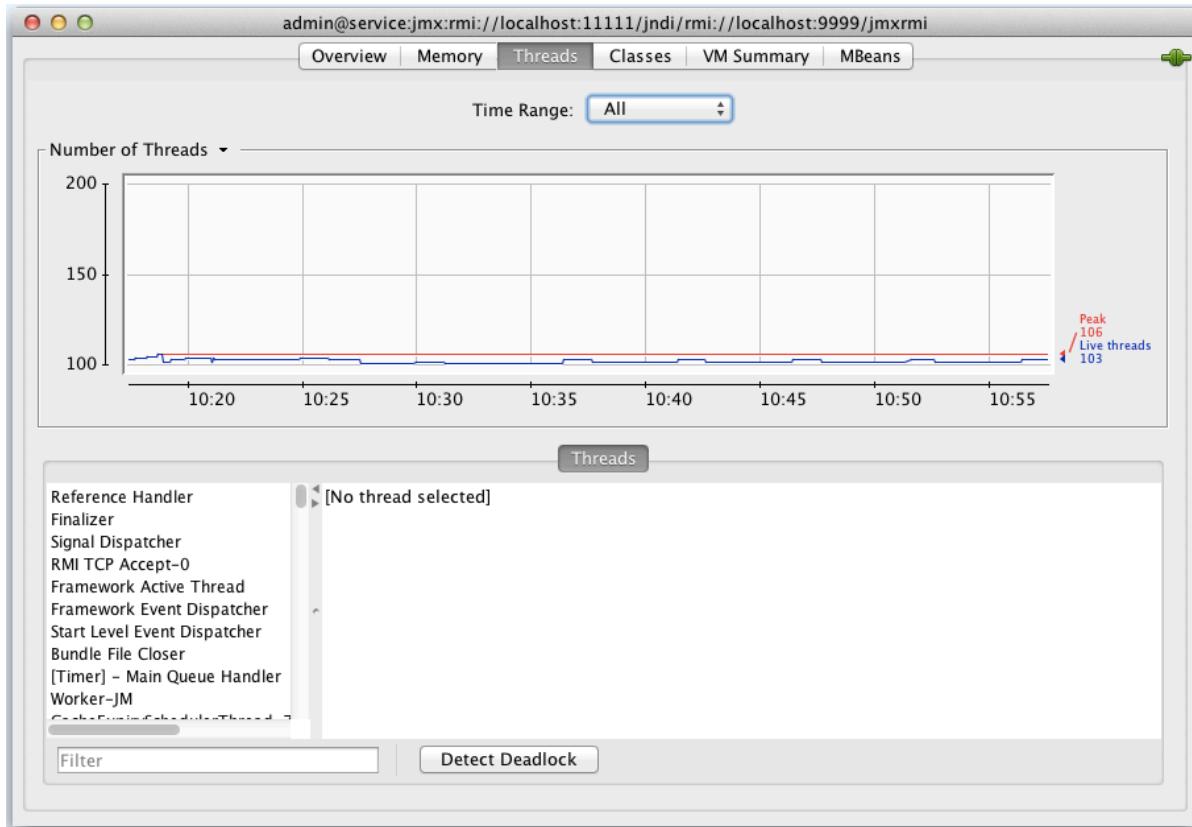
See the Oracle documentation on [using JConsole](#) for more information on these tabs.



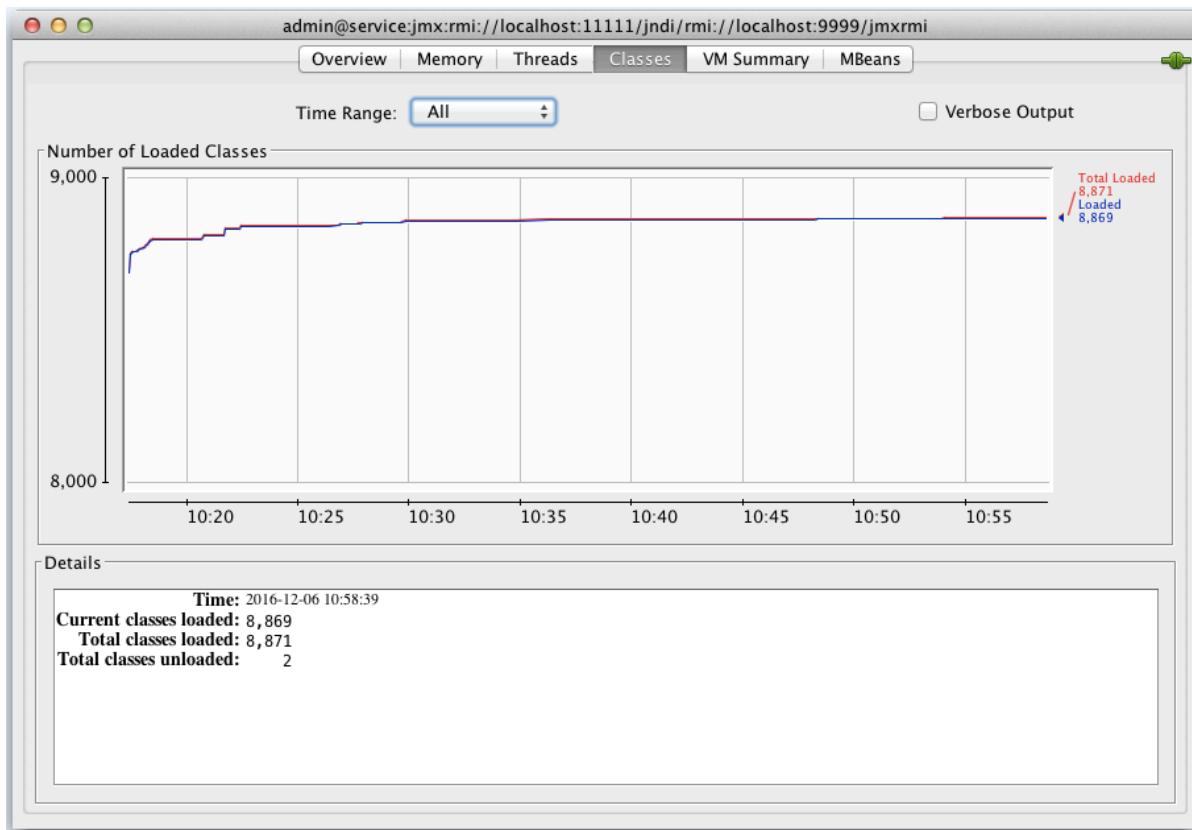
See the Oracle documentation on [using JConsole](#) for more information on these tabs.



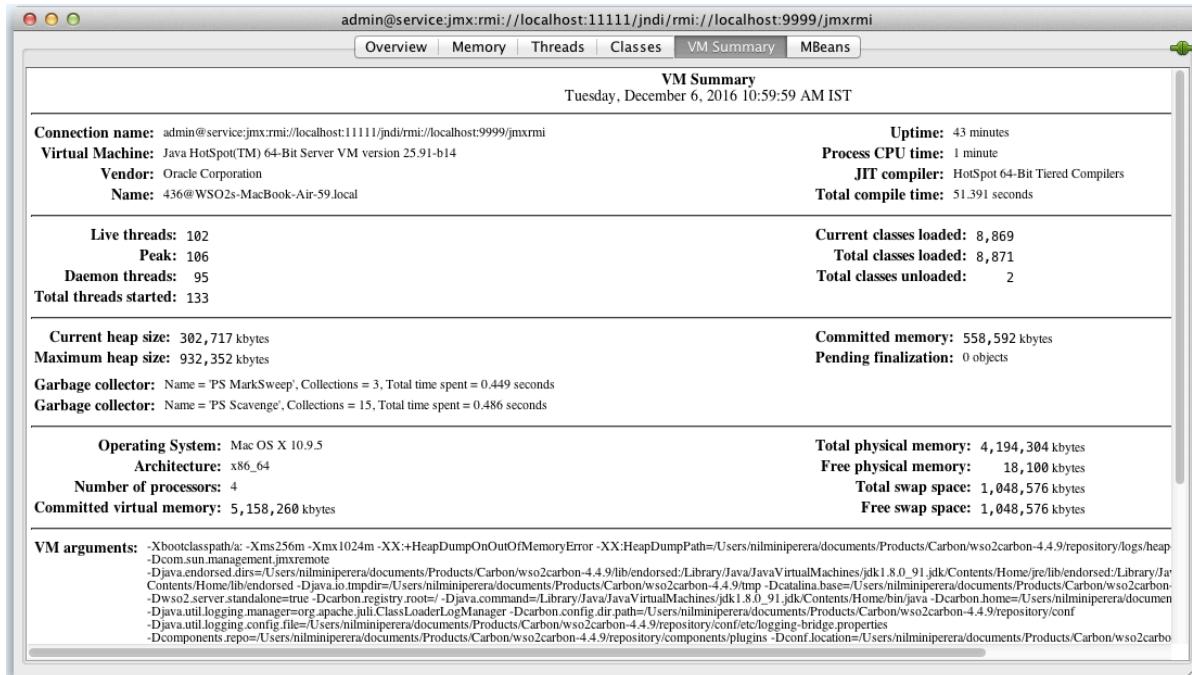
See the Oracle documentation on [using JConsole](#) for more information on these tabs.



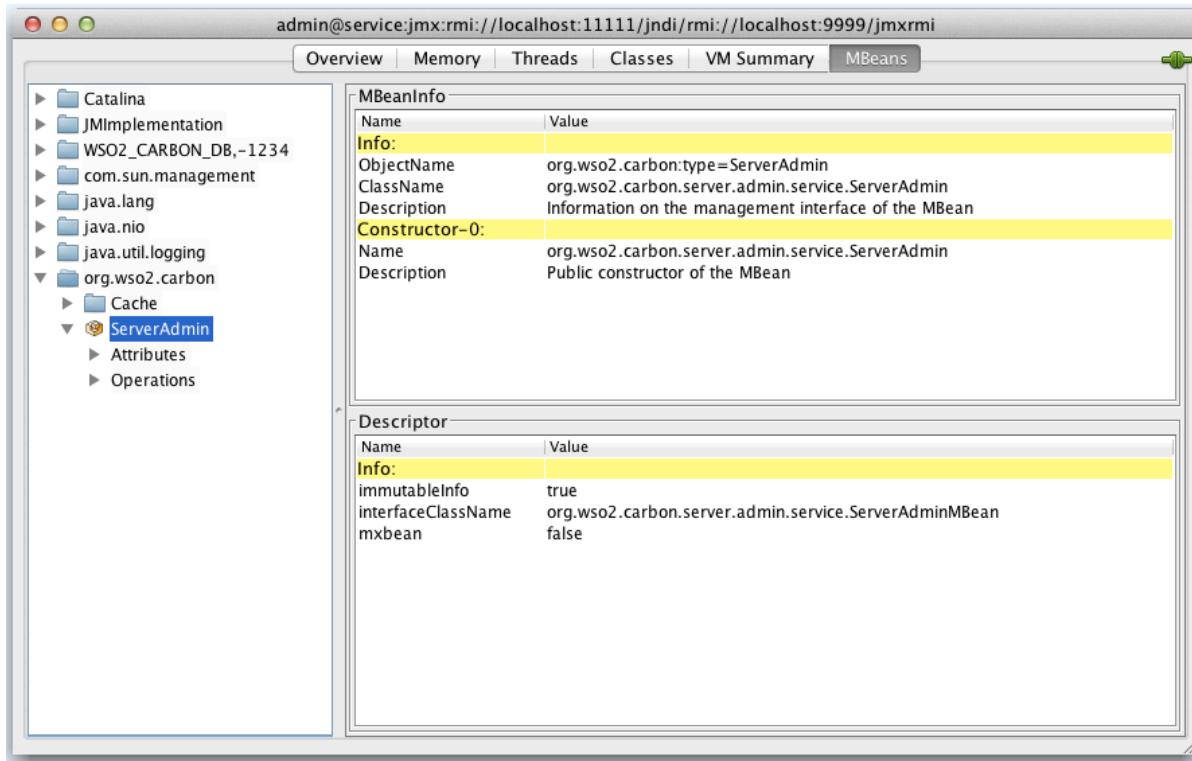
See the Oracle documentation on [using JConsole](#) for more information on these tabs.



See the Oracle documentation on [using JConsole](#) for more information on these tabs.

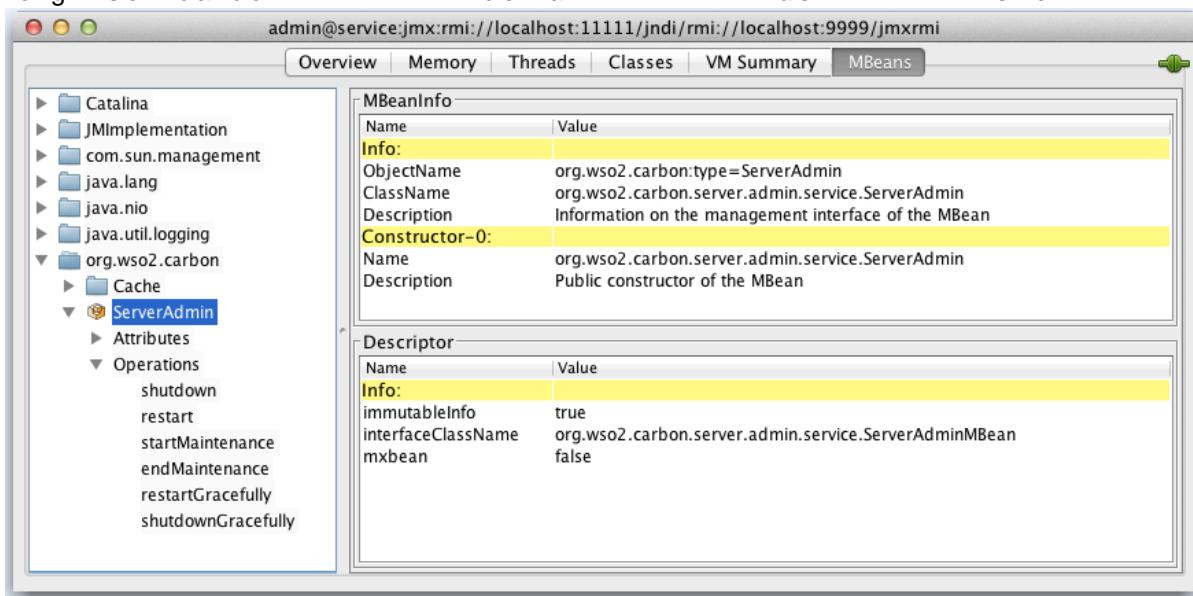


See the Oracle documentation on [using JConsole](#) for more information on these tabs.



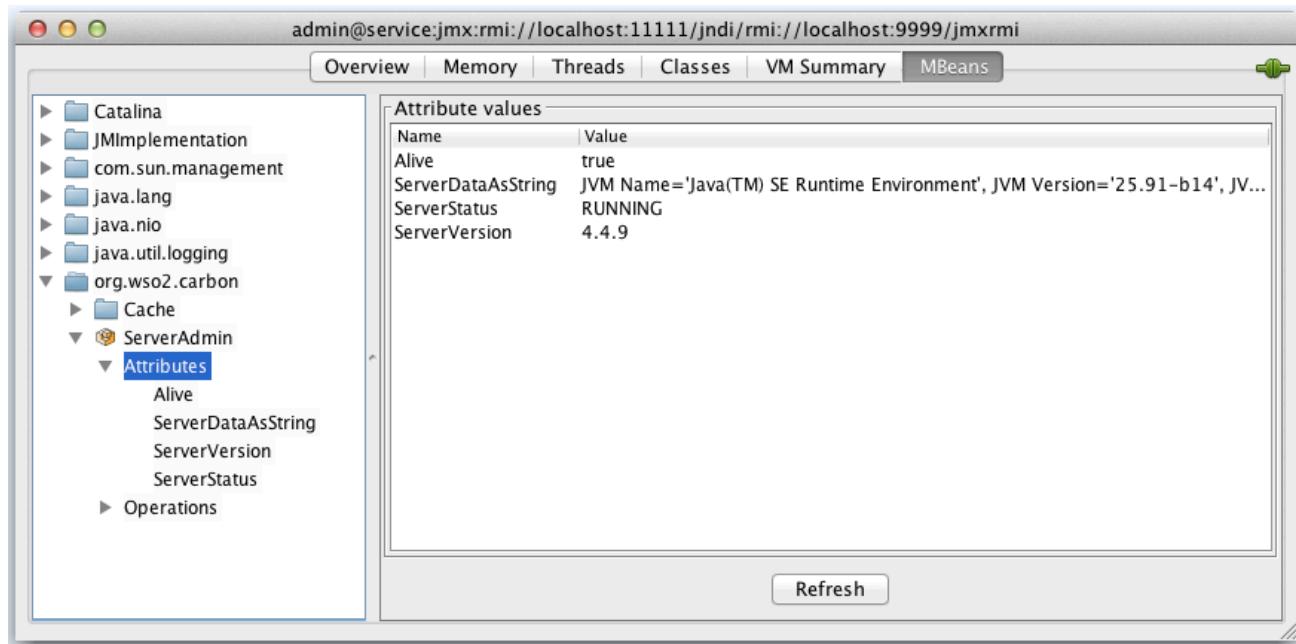
### Using the ServerAdmin MBean

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.



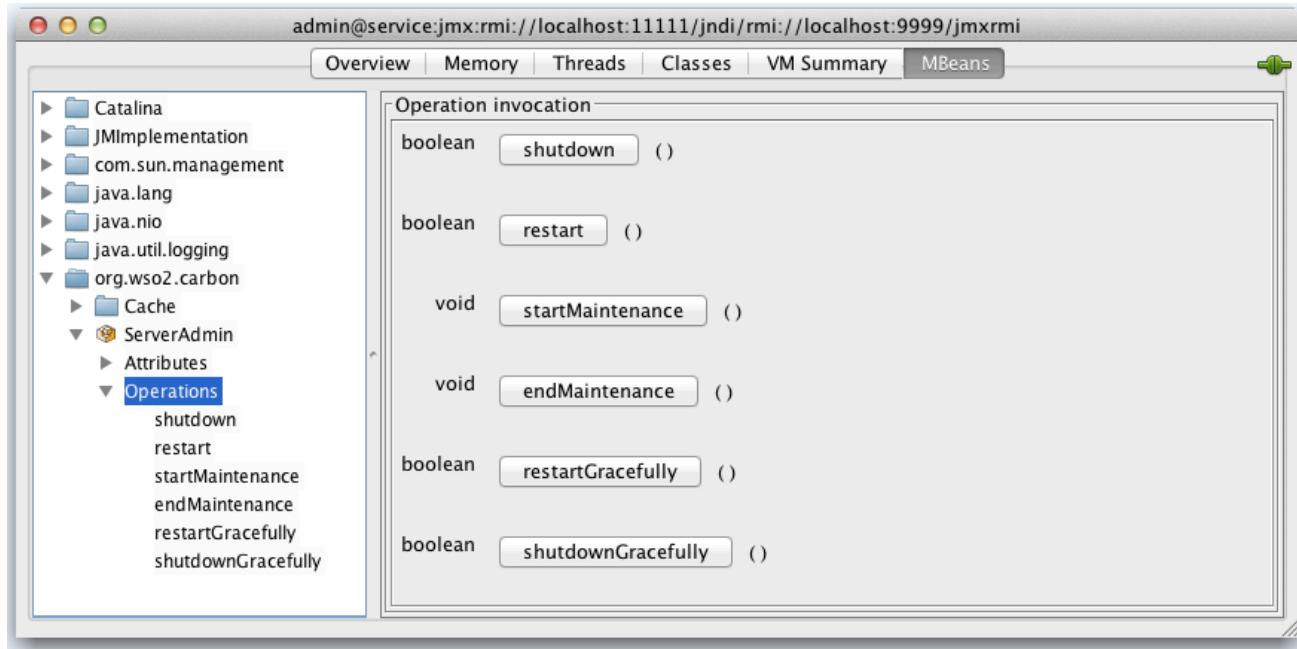
The **ServerAdmin** MBean is used for administering the product server instance. There are several server attributes such as "ServerStatus", "ServerData" and "ServerVersion". The "ServerStatus" attribute can take any of the following values:

- RUNNING
- SHUTTING\_DOWN
- RESTARTING
- IN\_MAINTENANCE



The **ServerAdmin** MBean has the following operations:

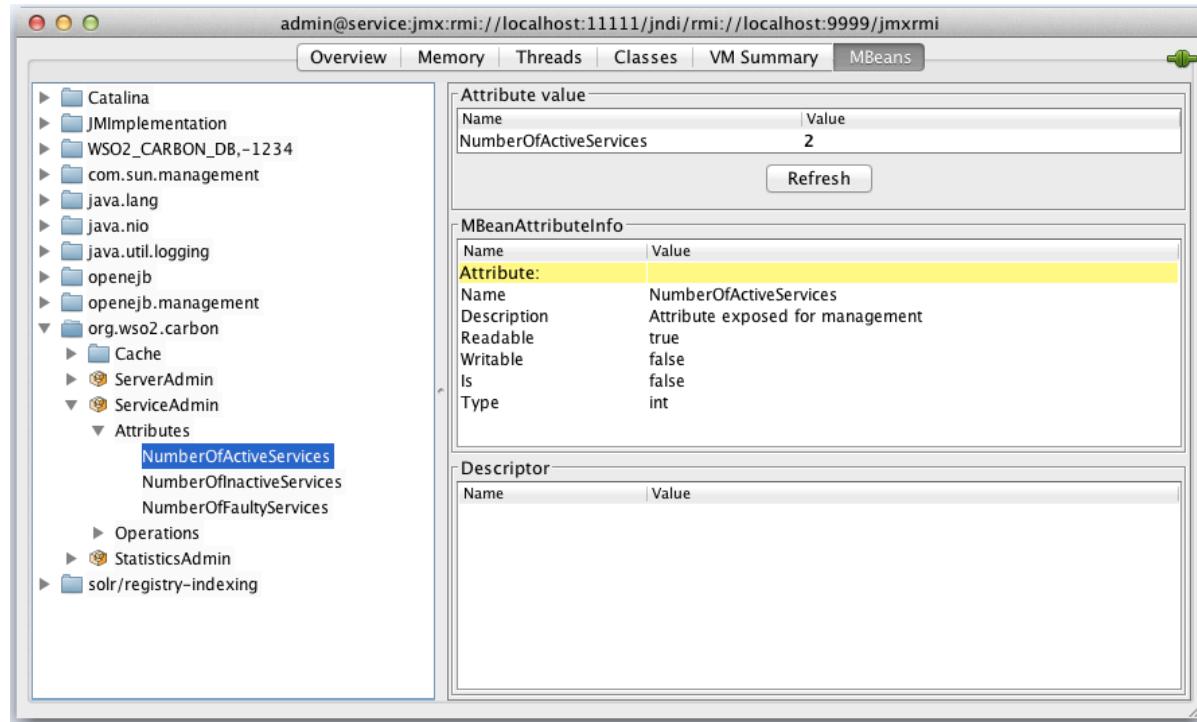
Operation	Description
<b>shutdown</b>	Forcefully shut down the server.
<b>restart</b>	Forcefully restart the server.
<b>restartGracefully</b>	Wait till all current requests are served and then restart.
<b>shutdownGracefully</b>	Wait till all current requests are served and then shutdown.
<b>startMaintenance</b>	Switch the server to maintenance mode. No new requests will be accepted while the server is in maintenance.
<b>endMaintenance</b>	Switch the server to normal mode if it was switched to maintenance mode earlier.



### Using the ServiceAdmin MBean

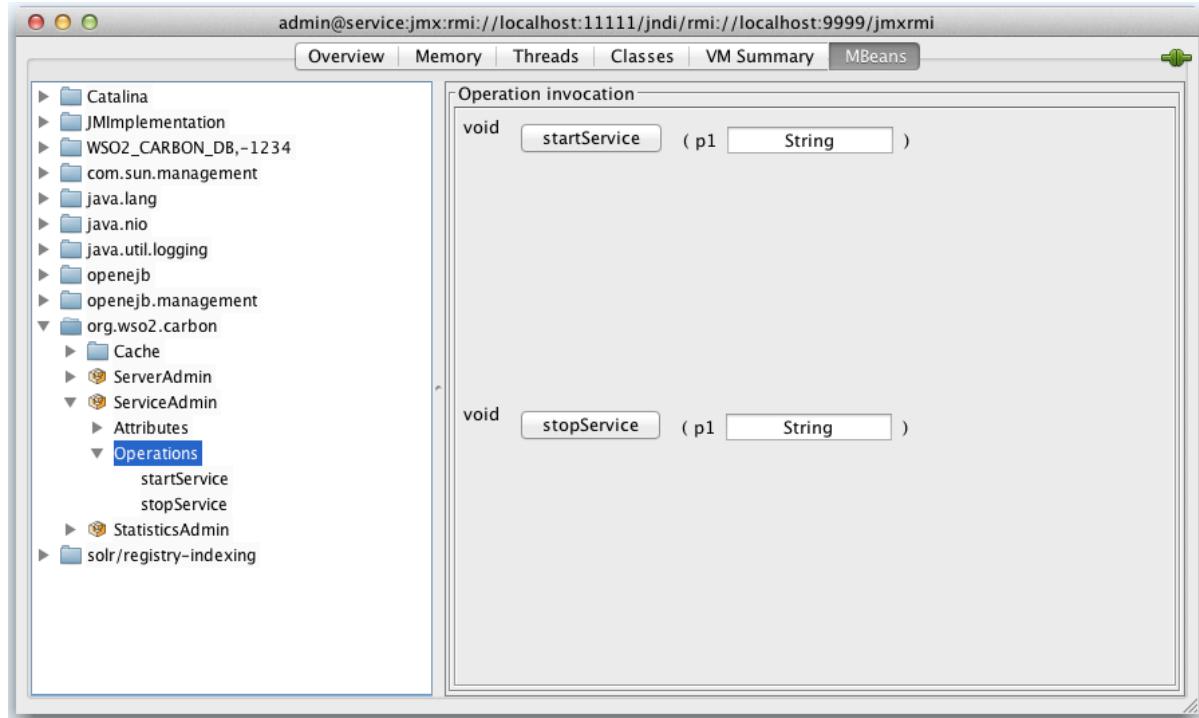
This MBean is used for administering services deployed in your product. Its attributes are as follows:

Attribute	Description
<b>NumberOfActiveServices</b>	The number of services which can currently serve requests.
<b>NumberofInactiveServices</b>	The number of services which have been disabled by an administrator.
<b>NumberOfFaultyServices</b>	The number of services which are faulty.



The operations available in the ServiceAdmin MBean:

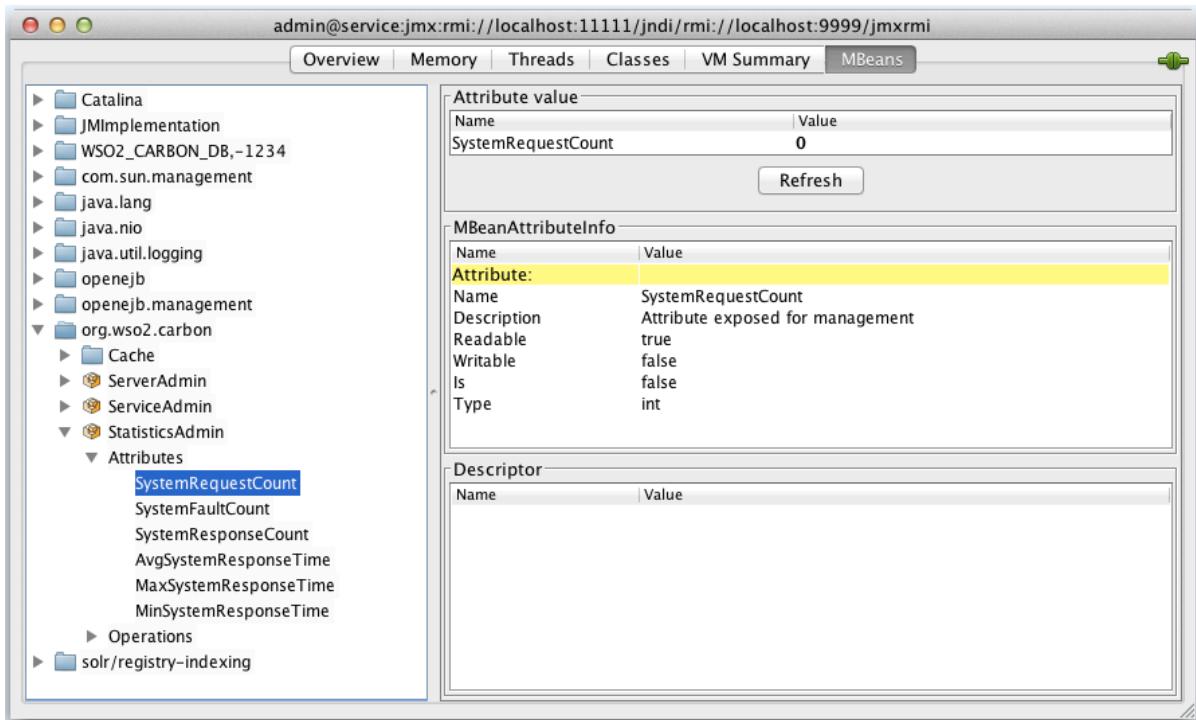
Operation	Description
<b>startService(p1:string)</b>	The p1 parameter is the service name. You can activate a service using this operation.
<b>stopService(p1:string)</b>	The p1 parameter is the service name. You can deactivate/disable a service using this operation.



### Using the StatisticsAdmin MBean

This MBean is used for monitoring system and server statistics. Its attributes are as follows:

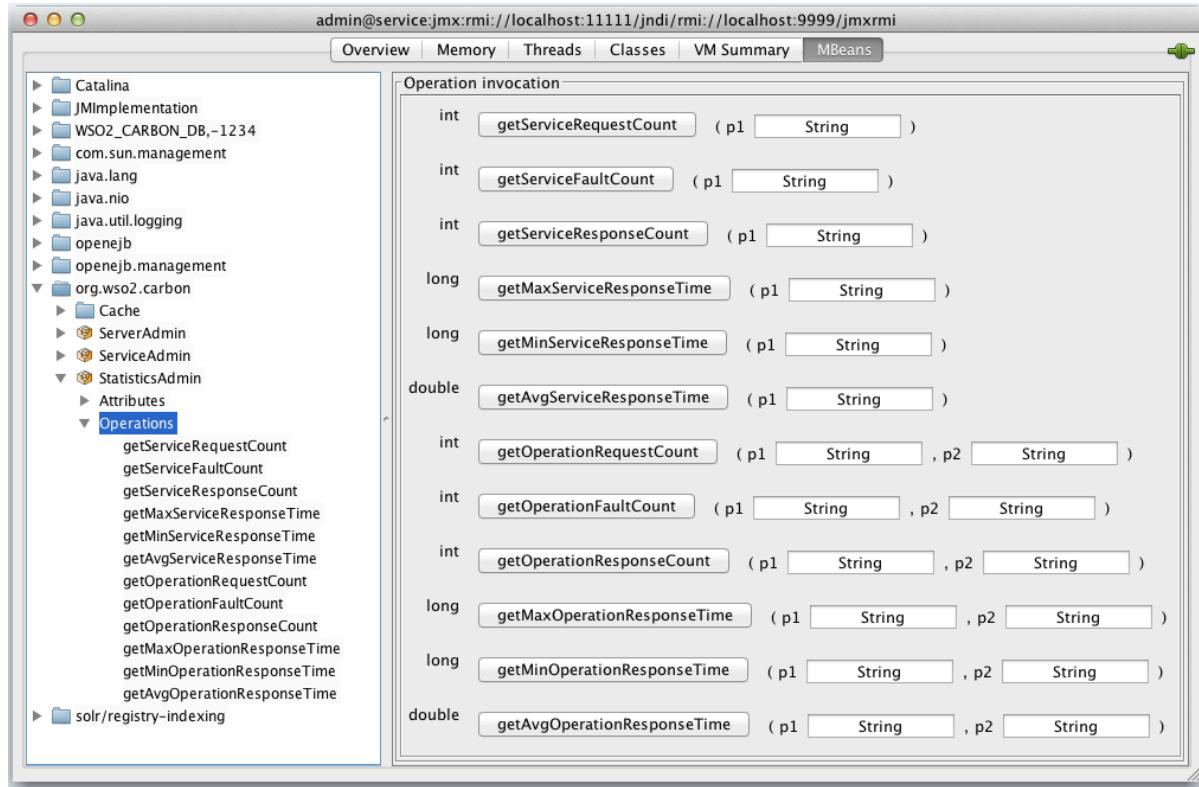
Attributes	Description
<b>AvgSystemResponseTime</b>	The average response time for all the services deployed in the system. The beginning of the measurement is the time at which the server started.
<b>MaxSystemResponseTime</b>	The maximum response time for all the services deployed in the system. The beginning of the measurement is the time at which the server started.
<b>MinSystemResponseTime</b>	The minimum time for all the services deployed in the system. The beginning of the measurement is the time at which the server started.
<b>SystemFaultCount</b>	The total number of faults that occurred in the system since the server was started.
<b>SystemRequestCount</b>	The total number of requests that has been served by the system since the server was started.
<b>SystemResponseCount</b>	The total number of response that has been sent by the system since the server was started.



Operations available in the **Statistics** MBean:

Operation	Description
<code>getServiceRequestCount(p1:string)</code>	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.
<code>getServiceResponseCount(p1:string)</code>	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.
<code>getServiceFaultCount(p1:string)</code>	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.
<code>getMaxServiceResponseTime(p1:string)</code>	The p1 parameter is the service name. You can get the maximum response time of this service since deployment.
<code>getMinServiceResponseTime(p1:string)</code>	The p1 parameter is the service name. You can get the minimum response time of this service since deployment.
<code>getAvgServiceResponseTime(p1:string)</code>	The p1 parameter is the service name. You can get the average response time of this service since deployment.
<code>getOperationRequestCount(p1:string, p2:string)</code>	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.
<code>getOperationResponseCount(p1:string, p2:string)</code>	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.

<b>getOperationFaultCount(p1:string, p2:string)</b>	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.
<b>getMaxOperationResponseTime(p1:string, p2:string)</b>	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.
<b>getMinOperationResponseTime(p1:string, p2:string)</b>	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.
<b>getAvgOperationResponseTime(p1:string, p2:string)</b>	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.



### 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.

**MBeanInfo**

Name	Value
Info:	WSO2_CARBON_DB,-1234:type=DataSource org.apache.tomcat.jdbc.pool.ConnectionPool
Constructor-0:	Name: org.apache.tomcat.jdbc.pool.ConnectionPool Description: Public constructor of the MBean
Parameter-0-0:	Name: p1 Type: org.apache.tomcat.jdbc.pool.ConnectionPool

**Descriptor**

Name	Value
Info:	immutableInfo: false interfaceClassName: org.apache.tomcat.jdbc.pool.ConnectionPoolMBean mxbean: false

**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.

**Attribute values**

Name	Value
AbandonWhenPercentageFull	0
AccessToUnderlyingConnectionAllowed	true
Active	0
AlternateUsernameAllowed	false
CommitOnReturn	false
ConnectionProperties	
DataSource	
DataSourceJNDI	
DbProperties	{}
DefaultAutoCommit	false
DefaultCatalog	
DefaultReadOnly	-1
DefaultTransactionIsolation	org.h2.Driver
DriverClassName	true
FairQueue	0
Idle	false
IgnoreExceptionOnPreLoad	
InitSQL	0
InitialSize	ConnectionState;StatementFinalizer;org.wso2.carbon.ndata...
JdbcInterceptors	Unavailable
JdbcInterceptorsAsArray	
JmxEnabled	true
LogAbandoned	false
LogValidationErrors	false
MaxActive	50
MaxAge	0
MaxIdle	8
MaxWait	60000

### 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 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-<product>-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:** From Carbon 4.4.3 onwards, it is possible to use a UUID in log messages so that any forged messages can be easily identified. The UUID is logged using a new conversion character 'K' in the log pattern layout. By default, the UUID will be generated every time the server starts. However, you can configure your server to generate the UUID more or less frequently, by specifying an exact time interval in the `log4j.properties` file.

#### 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 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.appenders.AUDIT_LOGFILE=org.wso2.carbon.utils.logging.appenders.
CarbonDailyRollingFileAppender
log4j.appenders.AUDIT_LOGFILE.File=${carbon.home}/repository/logs/audi
t.log
log4j.appenders.AUDIT_LOGFILE.Append=true
log4j.appenders.AUDIT_LOGFILE.layout=org.wso2.carbon.utils.logging.Ten
antAwarePatternLayout
log4j.appenders.AUDIT_LOGFILE.layout.ConversionPattern=[%d] %P%5p {%
c}-
%x %m %n
log4j.appenders.AUDIT_LOGFILE.layout.TenantPattern=%U%@%D [%T] [%S]
log4j.appenders.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:

Product	Description
WSO2 DAS	<p>Event tracing logs are enabled in WSO2 DAS using the <code>EVENT_TRACE_LOGGER</code> appender as shown related to events in WSO2 DAS. By default, this appender uses the root log level, which is <code>INFO</code>. <a href="#">as explained here</a>.</p> <p>▼ Message tracing log configuration</p> <pre> log4j.category.EVENT_TRACE_LOGGER=INFO, EVENT_TRACE_APPENDER, EVENT log4j.additivity.EVENT_TRACE_LOGGER=false log4j.appenders.EVENT_TRACE_APPENDER=org.apache.log4j.DailyRollingFi log4j.appenders.EVENT_TRACE_APPENDER.File=\${carbon.home}/repository/ log4j.appenders.EVENT_TRACE_APPENDER.Append=true log4j.appenders.EVENT_TRACE_APPENDER.layout=org.apache.log4j.PatternL log4j.appenders.EVENT_TRACE_APPENDER.layout.ConversionPattern=%d{HH: # The memory appender for trace logger log4j.appenders.EVENT_TRACE_MEMORYAPPENDER=org.wso2.carbon.utils.lo log4j.appenders.EVENT_TRACE_MEMORYAPPENDER.bufferSize=2000 log4j.appenders.EVENT_TRACE_MEMORYAPPENDER.layout=org.apache.log4j.I log4j.appenders.EVENT_TRACE_MEMORYAPPENDER.layout.ConversionPattern= </pre>

#### 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](#)):

```
log4j.appender.CARBON_LOGFILE.threshold=DEBUG
```

Listed below are the log levels that can be configured:

Level	Description
OFF	The highest possible log level. This is intended for disabling logging.
FATAL	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.
ERROR	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.
WARN	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.
INFO	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.
DEBUG	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.
TRACE	Provides additional details on the behavior of events and services. This information is expected to be written to logs only.

## 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.

#### **Limits 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.appenders.CARBON\_LOGFILE=org.wso2.carbon.utils.logging.appenders.CarbonDailyRollingFileAppender appender in the <PRODUCT\_HOME>/repository/conf/log4j.properties file as follows:

```
log4j.appenders.CARBON_LOGFILE=org.apache.log4j.RollingFileAppender
```

2. Add the following two properties under RollingFileAppender

- log4j.appenders.CARBON\_LOGFILE.MaxFileSize=10MB
- log4j.appenders.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.

#### **Limits 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.appenders.AUDIT\_LOGFILE=org.wso2.carbon.logging.appenders.CarbonDailyRollingFileAppender appender in the <PRODUCT\_HOME>/repository/conf/log4j.properties file as follows: log4j.appenders.AUDIT\_LOGFILE=org.apache.log4j.RollingFileAppender
2. Add the following two properties under RollingFileAppender:
  - log4j.appenders.AUDIT\_LOGFILE.MaxFileSize=10MB
  - log4j.appenders.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.

Type	Date	Log Message
Info	2014-10-15 10:51:15,832	'admin@carbon.super [-1234]' logged in at [2014-10-15 10:51:15,831+0530]
Info	2014-10-15 10:48:21,136	Mgt Console URL : https://10.100.5.65:9445/carbon/
Info	2014-10-15 10:48:20,916	WSO2 Carbon started in 12 sec

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

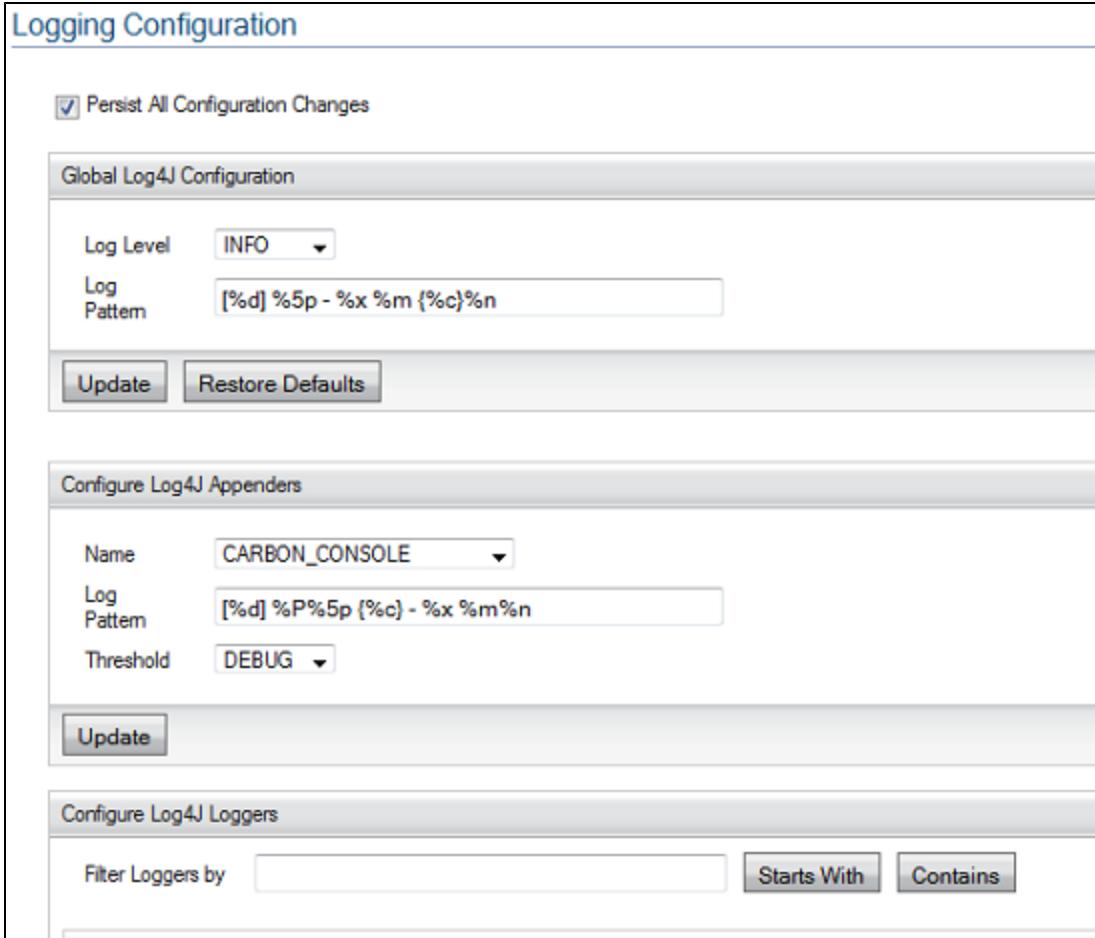
All WSO2 products are shipped with the [log4j](#) logging capabilities, which generates administrative activities and server side logs. The `log4j.properties` file, which governs how logging is performed by the server can be found in the `<PRODUCT_HOME>/repository/conf` directory. If the [Logging Management 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.



- 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.
- 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

Global Log4J Configuration	
Log Level	INFO
Log Pattern	[%d] %5p - %x %m {%c}%n
<input type="button" value="Update"/> <input type="button" value="Restore Defaults"/>	

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

Name	CARBON_CONSOLE
Log Pattern	[%d] %P%5p {%c} - %x %m%n
Threshold	DEBUG
<b>Update</b>	

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 are 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.appenders.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

Configure Log4J Loggers				
Filter Loggers by		Starts With	Contains	
Logger	Parent Logger	Level	Additivity	
root	-	INFO	True	
JdbmIndex	root	INFO	True	
JdbmTable	root	INFO	True	
LOG_CHANGES	root	INFO	True	
atomikos	root	WARN	True	
class org.apache.axiom.om.impl.llom.OMSourcedElementImpl.forceExpand	root	INFO	True	
httpclient.wire.content	root	INFO	True	
httpclient.wire.header	root	INFO	True	
java.lang.Class	root	INFO	True	
org.apache	root	INFO	True	
org.apache.abdera.protocol.server.ServiceManager	org.apache	INFO	True	
org.apache.abdera.protocol.server.impl.AbstractEntityCollectionAdapter	org.apache	INFO	True	
org.apache.abdera.protocol.server.impl.AbstractProvider	org.apache	INFO	True	

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 <PRODUCT\_HOME>/repository/logs/ directory 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.
- `logsCount(String tenantDomain, String serverKey)` - Return LogEvent count
- `clearLogs()` - Clear operation. For example, if it is an "in memory" log provider, this method can be used to clear the memory.

## 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.appenders.CARBON_MEMORY` property:

```
log4j.appenders.CARBON_MEMORY=org.wso2.carbon.logging.service.appenders.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:

```

<logProviderConfig xmlns="http://wso2.org/projects/carbon/carbon.xml"
class="org.wso2.carbon.logging.service.provider.CassandraLogProvider">
<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="archivedUser" value="admin"/>
    <property name="archivedPassword" value="admin"/>
    <property name="archivedPort" value="80"/>
    <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>.

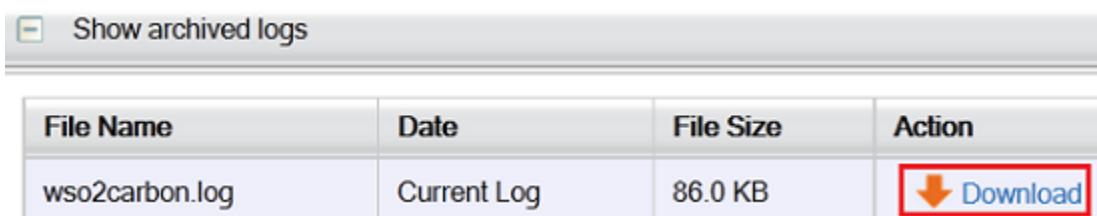
See the following topics on system logs and application logs:

- [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.



A screenshot of a web-based interface titled "Show archived logs". Below the title is a table with four columns: "File Name", "Date", "File Size", and "Action". There is one row in the table. The "File Name" column contains "wso2carbon.log", the "Date" column contains "Current Log", the "File Size" column contains "86.0 KB", and the "Action" column contains a red-bordered "Download" button with a downward arrow icon.

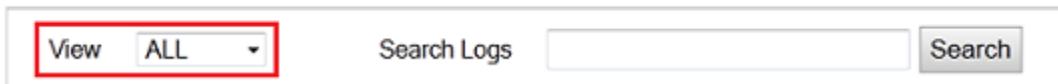
File Name	Date	File Size	Action
wso2carbon.log	Current Log	86.0 KB	 Download

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.

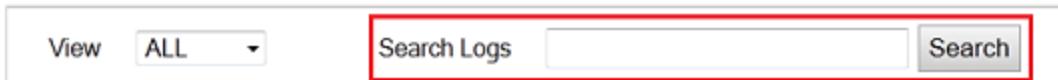
### **System Logs**



A screenshot of a web-based interface titled "System Logs". At the top, there is a search bar with a dropdown menu set to "ALL". To the right of the search bar is a "Search" button.

4. You can also find a specific log using the search function. Enter a keyword (or part of a keyword) and click **Search**.

### **System Logs**



A screenshot of a web-based interface titled "System Logs". At the top, there is a search bar with a red box around it. To the right of the search bar is a "Search" button.

## View application logs

Event 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.

The screenshot shows a user interface for viewing application logs. At the top, there is a title bar with the text 'Application Logs'. Below it is a search bar with a placeholder 'Search Logs' and a 'Search' button. To the left of the search bar is a dropdown menu labeled 'View' with the option 'ALL' selected. Next to it is another dropdown menu with the value 'echo' highlighted and surrounded by a red box. The rest of the page is currently blank.

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,

The screenshot shows the 'Application Logs' page again. The 'View' dropdown is now set to 'ERROR' (highlighted with a red box). The table below lists two log entries. Both entries have a red 'X' icon in the 'Type' column and show the same timestamp: '2012-08-20 12:39:19,757'. The log message for both is 'Exception occurred while trying to invoke service method echoInt'. There are 'More' links next to each entry.

Type	Date	Log Message	
✖	2012-08-20 12:39:19,757	Exception occurred while trying to invoke service method echoInt	<a href="#">More</a>
✖	2012-08-20 12:39:19,757	Exception occurred while trying to invoke service method echoInt	<a href="#">More</a>

5. You can also find a certain log using the search function. Enter a keyword (or part of a keyword) and click **Search**. When a search criterion is given, the value in the **View** field is displayed as **Custom**. For example,

The screenshot shows the 'Application Logs' page with the 'View' dropdown set to 'Custom' (highlighted with a red box). The search bar contains the keyword 'echoint' (also highlighted with a red box). The table below shows two log entries matching this search term. Both entries have a red 'X' icon in the 'Type' column and show the same timestamp: '2012-08-17 12:20:42,102'. The log message for both is 'Exception occurred while trying to invoke service method echoInt'. There are 'More' links next to each entry.

Type	Date	Log Message	
✖	2012-08-17 12:20:42,102	Exception occurred while trying to invoke service method echoInt	<a href="#">More</a>
✖	2012-08-17 12:20:42,102	Exception occurred while trying to invoke service method echoInt	<a href="#">More</a>

## HTTP Access Logging

HTTP Requests/Responses are logged in the access log(s) and are helpful to monitor your application's usage activities, such as the persons who access it, how many hits it receives, what the errors are etc. This information is useful for troubleshooting. As the runtime of WSO2 products are based on Apache Tomcat, you can use the `Access_Log_Valve` variable in Tomcat 7 as explained below to configure HTTP access logs in WSO2 products. In addition, you can customize the access logs based on the supported `Access Log Valve` attributes.

### **Configuring HTTP access logging**

To configure HTTP access logs:

1. Edit the `Access_Log_Valve` variable as follows in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file, which is the server descriptor file for the embedded Tomcat integration.

```
<Valve className="org.apache.catalina.valves.AccessLogValve"
directory="${carbon.home}/repository/logs"
prefix="localhost_access_log_sample."
suffix=".log"
pattern="%{xxx}i %{xxx}o"
resolveHosts="false"/>
```

The pattern attribute defines the formatting layout, which consists of the information fields from the requests and responses that should be logged. The description of the formatting layout specified in the above pattern attribute is given below. For more example configurations, see [Customizing access logs by pattern](#).

- `xxx` - header name we need to log
- `%{xxx}i` - incoming request headers
- `%{xxx}o` - outgoing response headers

For more information on all possible values that can be given in the pattern parameter, go to: [http://tomcat.apache.org/tomcat-7.0-doc/config/valve.html#Access\\_Log\\_Valve](http://tomcat.apache.org/tomcat-7.0-doc/config/valve.html#Access_Log_Valve)

If you do not wish to customize the formatting layout, you can simply use the word 'common' or 'combined' for the pattern attribute to select a standard format.

- common - %h %l %u %t "%r" %s %b
- combined - %h %l %u %t "%r" %s %b "%{Referer}i" "%{User-Agent}i"

The optimized access only supports 'common' and 'combined' as the value for this attribute.

2. Restart the server. This will create a log file named `localhost_access_log_sample.{DATE}.log` inside the `<PRODUCT_HOME>/repository/logs` directory.

### **Customizing access logs by pattern**

Given below are a few sample configurations, for ways in which the access logs can be customized by 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:

```
<Valve className="org.apache.catalina.valves.AccessLogValve"
directory="${carbon.home}/repository/logs"
prefix="localhost_access_log_test."
suffix=".log"
pattern="%{Content-Type}i %{Accept}i %{Accept-Encoding}i"
resolveHosts="false"/>
```

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
```

It records the following log entry in the `localhost_access_log_sample.{DATE}.log` file.

<code>text/plain; charset=utf-8</code>	<code>*/*</code>	<code>gzip,deflate,sdch</code>
----------------------------------------	------------------	--------------------------------

### 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"
resolveHosts="false"/>
```

The above configuration sample logs the Content-type, Content-Length, Date and Server headers of every response coming from the server as follows:

<code>text/html;charset=ISO-8859-1</code>	<code>662</code>	<code>Tue, 09 Jul 2013 11:21:50 GMT</code>
<code>WSO2 Carbon</code>		

### Example 3: Logging other variable values

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="%r %q %h"
resolveHosts="false"/>
```

The above sample configuration logs the first line of the request (method and request URI), query string (prepended with a '?' if it exists) and 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)"
resolveHosts="false"/>
```

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'
```

## Monitoring Message Flows

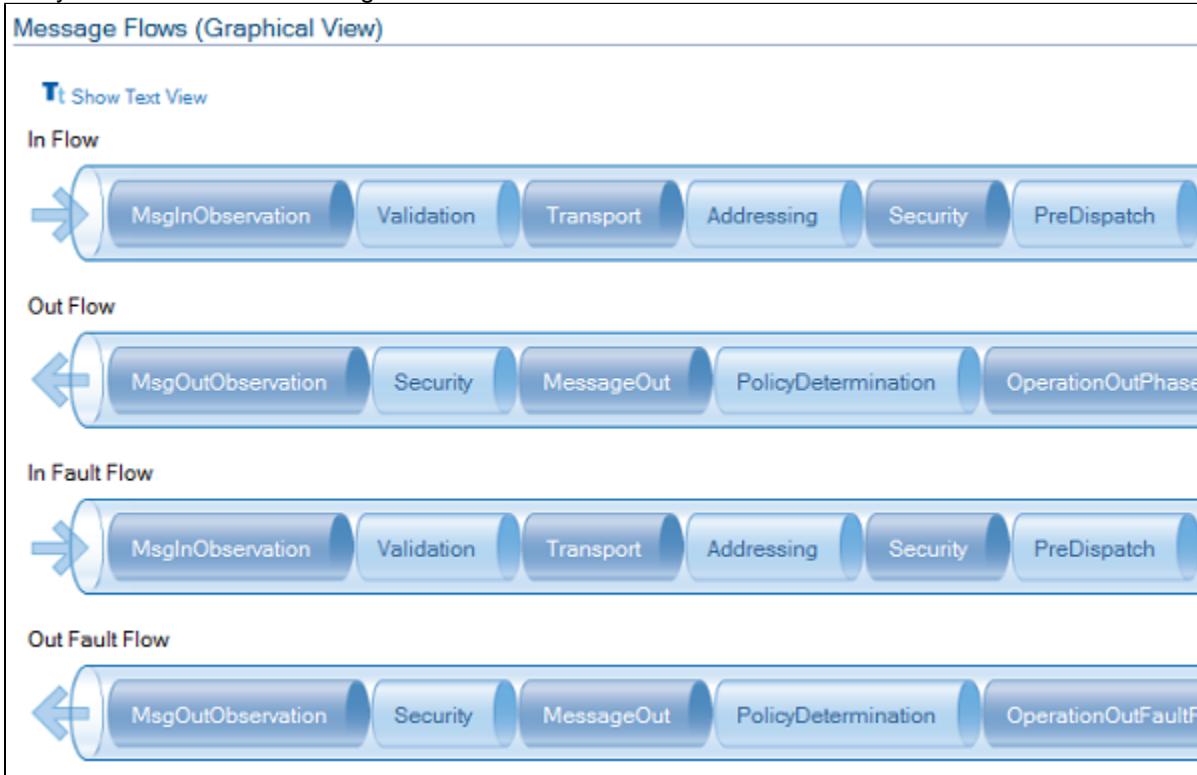
Message Flows provide graphical or textual views of the globally-engaged handlers in the system at a given time. This functionality is provided by the following feature:

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.

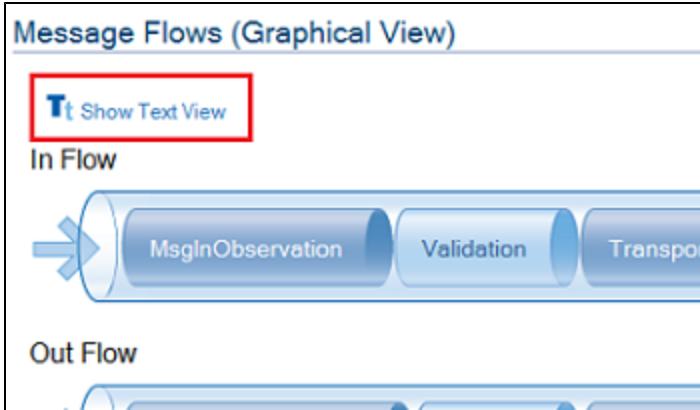
1. Log in to the management console and select **Monitor -> Message Flows**.
2. The **Message Flows** page displays a graphical view of the message flows. There are four different flows defined in the system:
  1. **In Flow** : A correct message coming into the system.
  2. **Out Flow** : A correct message going out of the system.
  3. **In Fault Flow** : A faulty message coming into the system.
  4. **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**.



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.

Message Flows (Text View)	
<a href="#">Show Graphical View</a>	
<b>In Flow</b>	
MsgInObservation	
InOnlyMEPHandler	org.wso2.carbon.statistics.module.InOnlyMEPHandler
Validation	
No handlers present.	
Transport	
TenantDomainHandler	org.wso2.carbon.core.multitenancy.TenantDomainHandler
RequestURIBasedDispatcher	org.apache.axis2.dispatchers.RequestURIBasedDispatcher
SOAPActionBasedDispatcher	org.apache.axis2.dispatchers.SOAPActionBasedDispatcher
REST/POX Security handler	org.wso2.carbon.security.pox.POXSecurityHandler
Addressing	
AddressingInHandler	org.apache.axis2.handlers.addressing.AddressingInHandler
AddressingBasedDispatcher	org.wso2.carbon.core.multitenancy.MultitenantAddressingBase

## 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.

This functionality is provided by the following feature:

<b>Name:</b>	WSO2	Carbon	-	Statistics	Feature
<b>Identifier:</b>	org.wso2.carbon.system.statistics.feature.group				

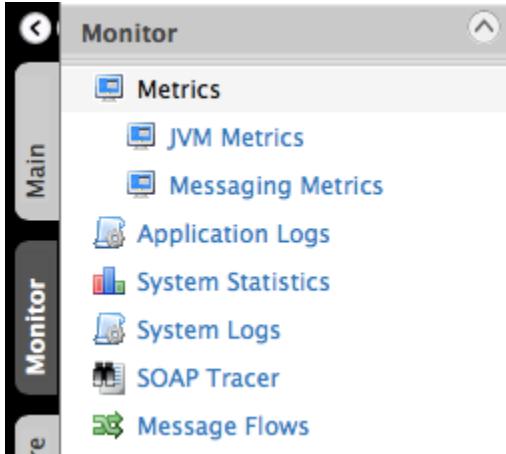
If this feature is not bundled with the WSO2 product by default, you can install it using the instructions provided in section [Feature Management](#).

Access and performance statistics are available in system level as follows:

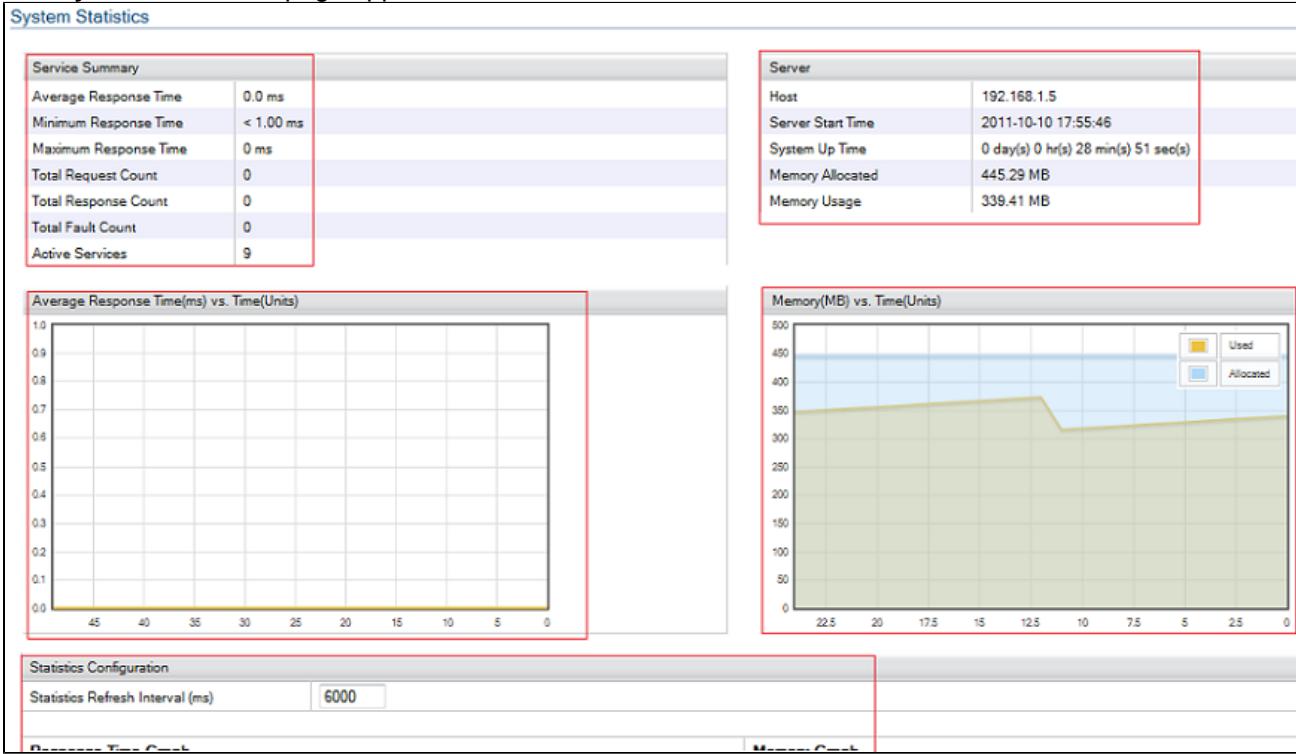
### System-Level Statistics

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

Service Summary	
Average Response Time	0.0 ms
Minimum Response Time	< 1.00 ms
Maximum Response Time	0 ms
Total Request Count	0
Total Response Count	0
Total Fault Count	0
Active Services	9

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.

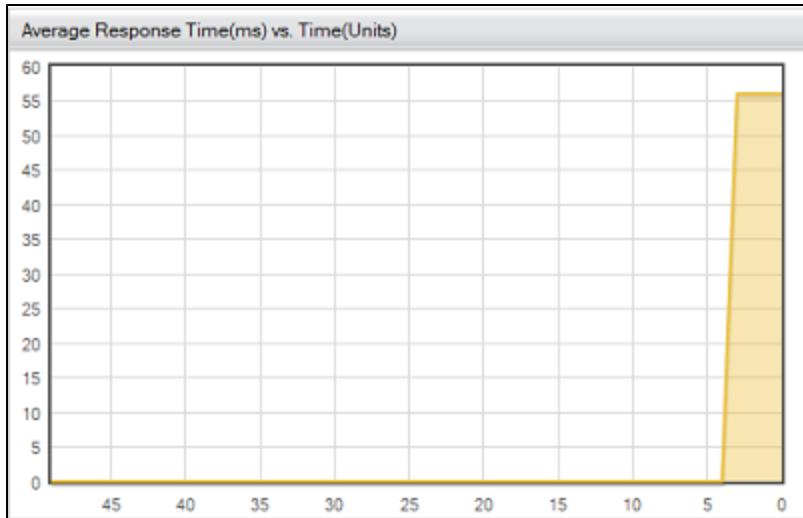
#### **Server Information**

Server	
Host	192.168.1.5
Server Start Time	2011-10-10 17:55:46
System Up Time	0 day(s) 0 hr(s) 36 min(s) 9 sec(s)
Memory Allocated	494.94 MB
Memory Usage	326.07 MB

This panel provides the following information:

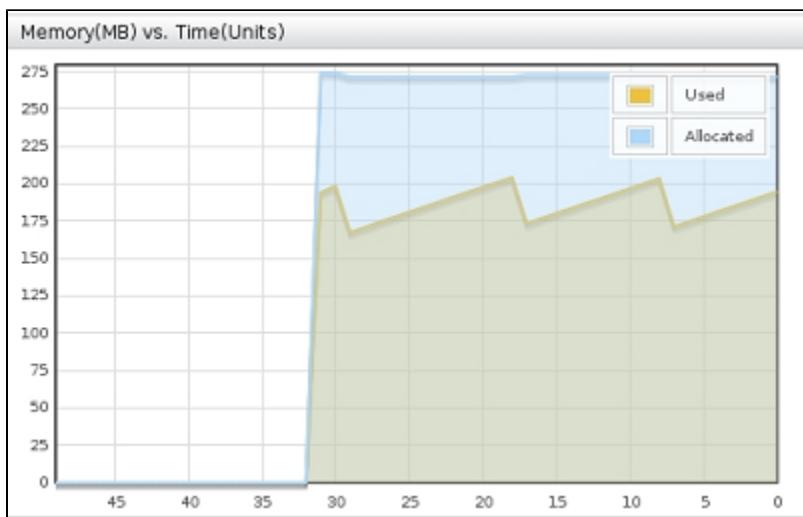
- **Host** - Shows the IP address of the server.
- **Server Start Time** - Shows the time when the server started.
- **System Up Time** - Shows the time the server has been up and running.
- **Memory Allocated** - Shows the memory capacity of the server.
- **Memory Usage** - Shows the used memory of 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.

Statistics Configuration	
Statistics Refresh Interval (ms)	<input type="text" value="6000"/>
<b>Response Time Graph</b>	
X-Scale (units)	<input type="text" value="50"/>
X-Width (px)	<input type="text" value="500"/>
<b>Memory Graph</b>	
X-Scale (units)	<input type="text" value="25"/>
X-Width (px)	<input type="text" value="500"/>
<input type="button" value="Update"/>	<input type="button" value="Reset"/>
<input type="button" value="Restore Defaults"/>	

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 Product Deployment

Page viewing is restricted to WSO2 as it is being edited.

The following instructions will guide you through the steps for monitoring the deployment of a WSO2 product, or a group of WSO2 product servers using the [WSO2 Carbon Monitoring Tool](#). The Carbon monitoring tool can connect to each of the product instances and provide information related to various aspects of your deployment.

The following instructions will help you get started with deployment monitoring of WSO2 products. We will apply minimal changes to the default settings in the Carbon Monitoring Tool as well as the WSO2 product servers that are being monitored to demonstrate how easy it is to use this tool.

- Go to [About the Carbon Monitoring Tool](#), to understand how the tool works.
- Go to [Configuring the Carbon Monitoring Tool](#) for instructions on how to update the tool's default configurations to fit your deployment patterns.

Let's get started:

- Before you begin
- Download and setup the WSO2 product servers
- Download and setup the WSO2 Carbon Monitoring tool
- Start monitoring

### Before you begin

Be sure that Java 1.8 or a higher JDK is installed in your system.

### Download and setup the WSO2 product servers

We will now create a server group consisting of two WSO2 Identity Server instances:

1. Go to <http://wso2.com/identity-and-access-management> and download the WSO2 Identity Server ZIP file.
2. Extract the downloaded ZIP file. We will refer to this extracted directory as <IS\_HOME\_1>.
3. Create a copy of <IS\_HOME\_1>. We will refer to this new directory as <IS\_HOME\_2>.
4. By default, most WSO2 servers will start on port 9443. Therefore, you need to set a port offset in one of the IS instances so that they can be started simultaneously without any port conflicts.
  1. Open the `carbon.xml` file in the <IS\_HOME\_2>/repository/conf directory.
  2. Note that by default, <Offset> is set to 0, which corresponds to the default port 9443. Change this

value to 1 as shown below, so that this IS instance can start in port 9444.

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

3. Save the information.

#### Download and setup the WSO2 Carbon Monitoring tool

1. Download the **wso2monitor.war** from here and copy it to a location on your computer.
2. Extract the WAR file and copy the **deployment-monitor-home** directory to a separate location on your computer. We will refer to this directory as <DEPLOYMENT\_MONITOR\_HOME>.
3. Open the <DEPLOYMENT\_MONITOR\_HOME>/conf/netty-transport.yaml file and change the keystore file path to <DEPLOYMENT\_MONITOR\_HOME>/resources/wso2carbon.jks.
4. Save the information.
5. Open the <DEPLOYMENT\_MONITOR\_HOME>/conf/deployment-monitor.yaml file and define the server group that you are going to monitor. In this example, we will be monitoring a server group consisting of two WSO2 Identity Server (IS) instances, which are hosted on two ports (9442 and 9444).

Specify a name for the server group and the hosts of the IS instances as shown below.

```
name: IS
hosts:
  - https://localhost:9443
  - https://localhost:9444
```

6. Copy the wso2-monitor-agent-x.x.x.jar file (stored in the <DEPLOYMENT\_MONITOR\_HOME> directory) to the /repository/components/dropins directory in <IS\_HOME\_1> and <IS\_HOME\_2> servers.
7. Save the information.

You are now ready to start using the Carbon monitoring tool to monitor the server group.

#### Start monitoring

Let's start the product servers in the server group.

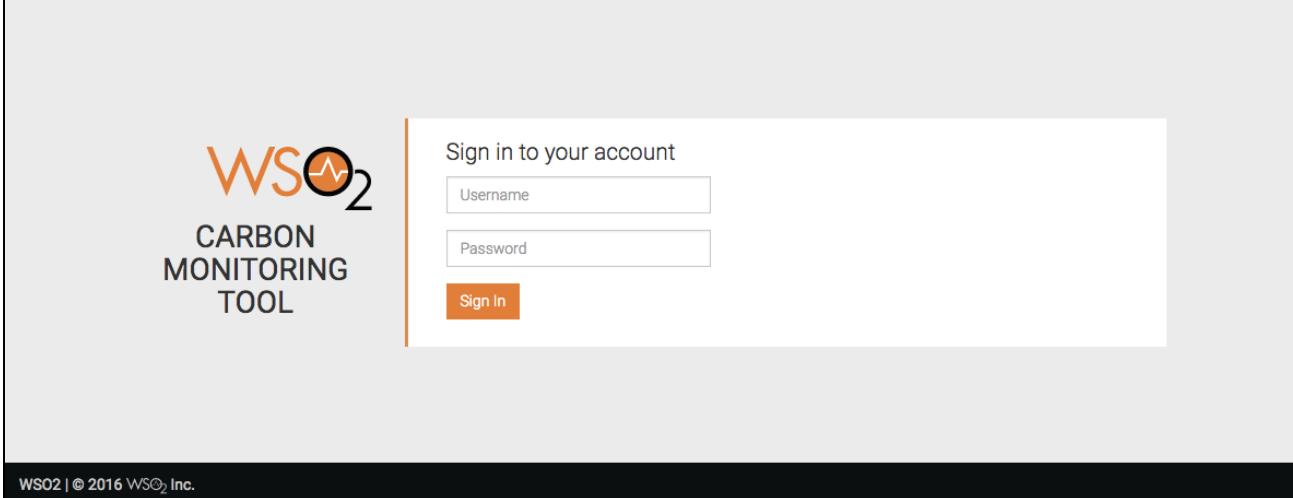
1. Open a command prompt and navigate to the <IS\_HOME\_1>/bin directory.
2. Execute the product startup script using one of the following commands:
  - On Windows: ./wso2server.bat
  - On Linux: ./wso2server.sh
3. The server will start on <https://localhost:9443>.
4. Repeat the above steps to start <IS\_HOME\_2>. This server instance will start on <https://localhost:9444>.

Now, let's start the Carbon monitoring tool as a web application.

1. Open a command prompt and navigator to the directory that contains the wso2monitor.war file that you downloaded.
2. Execute the following command:

```
java -Ddeployment.monitor.home=DEPLOYMENT_MONITOR_HOME -jar
wso2monitor.war
```

3. Open a web browser and go to <https://localhost:8888/monitor/> shown below:



4. Log in using the default username and password: admin/admin.

You can add new users and passwords using the users.xml file <DEPLOYMENT\_MONITORING\_HOME>/conf directory. Be sure to restart the tool in order to log in using new credentials. See Configuring the Deployment Monitoring Tool for instructions.

5. The tasks defined in the tool will be listed as shown below.

Task	Run	Schedule
Server Info		
Patch Verification		
Deployment Synchronizer Test		
WUM Updates		
Tenant Login		
Ping		
Analytics DbQuery		

WSO2 | © 2016 WSO2 Inc.

6. Click on a particular task and you will see the server group listed as shown below.

## Tasks

The screenshot shows a list of tasks under the heading 'Tasks'. The tasks listed are 'Server Info' and 'IS'. There are buttons for 'RUN' and 'SCHEDULE' next to each task entry.

7. Click **RUN** to execute any particular task. For example, run the **Server Info** task. If the task is executed successfully, you will see the following:

The screenshot shows the 'Server-Info' task details page. It displays a success message: 'Success! Server-Info task succeeded for IS'. The WSO2 logo and the text 'WSO2 CARBON MONITORING TOOL' are visible at the top. The navigation bar includes 'Home / Tasks' and 'SERVER-INFO'.

Note that the ten most recent task executions will be listed here.

8. Click on the green bar to see details. You will see server information from the two IS instances in the server group: r o u p :

The screenshot shows the WSO2 Message Broker Home page with the 'Tasks' section selected. A specific task, 'Server-Info', is highlighted. The task configuration includes a 'Product Server URL' of https://localhost:9443/carbon, 'Product Host Name' of localhost, 'Product' of WSO2 Identity Server - 5.1.0, 'Product Profile' of default, and 'Patches' of [0]. The task status is 'Completed' with a green success message: 'Success! Server-Info task succeeded for IS'. The task details show two sections of server information for ports 9443 and 9444, both indicating disabled clustering and deployment artifact synchronization. The task was completed successfully.

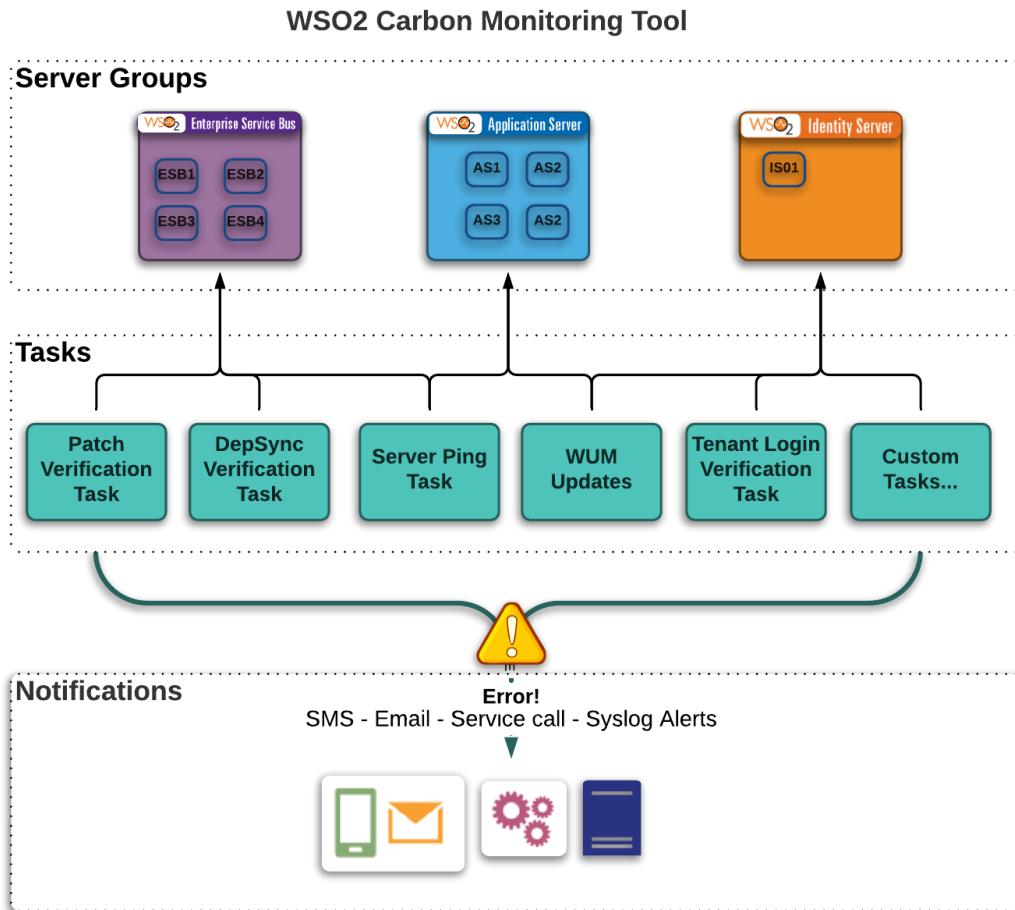
- Click **Schedule** on the Home page if you want the task to be executed periodically. By default, the schedule frequency is set to 60 seconds for all the default tasks. That is, when a task is on schedule mode as shown below, the task will be executed every 60 seconds. See [Configuring the Carbon Monitoring Tool](#) for instructions on changing the default settings.

The screenshot shows the 'Tasks' section on the WSO2 Message Broker Home page. It lists the 'Server Info' task, which is currently scheduled. There are buttons to 'RUN' or 'UNSCHEDULE' the task.

## About the Carbon Monitoring Tool

WSO2 provides a comprehensive monitoring tool (Carbon Monitoring Tool) for monitoring the deployment of WSO2 products. You can run this tool as a web application and execute various monitoring tasks on your product servers. The diagram given below illustrates how deployment monitoring works. The concepts and elements of deployment monitoring are also described.

- To get started with deployment monitoring, see [Monitoring Product Deployment](#).
- To use the tool from the command line instead of using the web application, see [Using the Command Line](#).
- Configuring the Carbon Monitoring Tool is as simple as updating the deployment-monitor.yaml file in the tool's distribution. See [Configuring the Carbon Monitoring Tool](#) for information on how to configure the tool according to your deployment scenario.



See the following topics for more information:

- [Server groups](#)
- [Tasks](#)
- [Notifications](#)
- [Permissions](#)

#### **Server groups**

The configurations in the monitoring tool should be updated with the details of the **server groups** that it is required to monitor. Server groups contain all the WSO2 product instances in your deployment. One server group may denote one individual server (as in the case of the WSO2 Identity Server group shown above), or it may be a cluster of multiple product servers (as in the case of the WSO2 ESB and WSO2 AS server groups shown above). See [Deployment and Clustering](#) for more information on how to define product clusters.

To add new server groups or configure server information, see [Updating Server Groups](#).

#### **Tasks**

The tool works by running **tasks** that are configured to monitor a specific aspect of deployment. The monitoring tool is shipped with **default tasks** that are configured to monitor specific areas such as patch verification and deployment synchronization.

Default tasks	Description
Server Info	Server information from all the connected servers will be checked.
Patch Verification	This task checks the details of patches that are installed in the servers that are being monitored. In a typical deployment, all the server instances of the same product should have the same patches installed.
Deployment Synchronizer Test	This task verifies whether or not the deployment synchronizer is working.
WUM Updates	This task verifies the updates that have been applied to the servers using <a href="#">WUM (WSO2 update manager)</a> .
Tenant Login	When this task is executed, the tool will log in to the super tenant of the servers to verify the tenant details.
Ping	When this task is executed, the tool connects to the servers to verify the connection.
AnalyticsDB Query	<b>This task can be used to connect to WSO2 DAS instances in your product deployment in order to verify if the latest details from products are published to DAS.</b>

Additionally, you have the option of defining **custom tasks**, which is as simple as implementing a Java interface, dropping the JAR into the tool's distribution directory and applying a minor update to one configuration file as explained below. Any of the monitoring tasks can also be **scheduled** so that they are executed periodically. By default, the tool is configured to run the default tasks every 60 seconds when scheduled mode is enabled.

For information on adding custom tasks or configuring an existing task, see [Updating Tasks](#).

#### Notifications

You can also set up notifications so that the operations team can be informed of task results through various channels (email, SMS etc.). By default, the deployment tool can be configured to send notifications through email and SMS (**Clickatell** and **bulkSMS** SMS providers are supported by default) channels. Additionally, you can write your custom tasks with the ability to send notifications through other channels. If a monitoring task results in an error, the tool can be configured to send notifications across multiple channels.

For information on configuring notifications, see [Updating notifications](#).

#### Permissions

The Carbon Monitoring Tool accesses the server groups through a monitoring agent. This agent, which is shipped with the WSO2 Carbon Monitoring Tool (`wso2-monitor-agent-x.x.x.jar` file), needs to be installed in each server that is being monitored. See the example in [Monitoring Product Deployment](#) for instructions.

The monitoring tool connects to a product server using a username and password that is set up for the super tenant of the product server. This user should be granted certain user permissions in the WSO2 server, in order to access the monitoring agent. Since this monitoring agent divulges information such as patch details, the required permission should be at the super-admin level. For example, if the Carbon Monitoring Tool accesses the server with a user called 'MonitoringUser', this user should be granted deployment monitoring permissions. The following diagram illustrates the **DeploymentMonitor** permission. See [Managing Users, Roles and Permissions](#) for instructions on editing permissions.

## Permissions of the Role : mc

The screenshot shows a tree view of permissions. At the top are 'All Permissions', 'Admin Permissions', and 'Super Admin Permissions'. Under 'Super Admin Permissions', there are four items: 'Configure', 'Manage', 'DeploymentMonitor' (which has a checked checkbox), and 'Server Admin'. Below the tree are two buttons: 'Update' and 'Cancel'.

Note that by default, the Carbon Monitoring Tool connects to a WSO2 server from the [default admin user](#), which will have the **DeploymentMonitor** permission granted by default (when the monitoring agent is installed in the server). To change the default settings for connecting from the Carbon Monitoring Tool to the WSO2 servers, see [Updating the Global Configurations](#).

## Configuring the Carbon Monitoring Tool

The WSO2 Carbon Monitoring Tool has a single configuration file called `deployment-monitor.yaml` (located in the `<DEPLOYMENT_MONITORING_HOME>/conf`), which specifies the main configurations related to monitoring WSO2 servers. Additionally, you can use the `log4j2.xml` file to configure logging for the tool and `users.xml` file to add users that can access the web application of the Carbon Monitoring Tool.

- To get started with deployment monitoring using the default settings, see [Monitoring Product Deployment](#).
- To understand how the Carbon Monitoring Tool works, go to [About the Carbon Monitoring Tool](#).
- To use the tool from the command line instead of using the web application, see [Using the Command Line](#).

See the following topics for instructions:

- [Updating server groups](#)
- [Updating tasks](#)
- [Updating Notifications](#)
- [Updating the global configurations](#)
- [Updating services](#)
- [Adding users to the Carbon monitoring tool](#)
- [Configuring logging for the Carbon monitoring tool](#)

### ***Updating server groups***

The server groups should be configured in the `deployment-monitor.yaml` file as shown in the following example. This example shows two server groups configured for deployment:

```

serverGroups:
  -
    name: IdentityServerGroup
    hosts:
      - https://is001.wso2.com:9443/
      - https://is002.wso2.com:9453/
  -
    name: ESBGroup
    hosts:
      - https://esb001.wso2.com:9443/
      - https://esb002.wso2.com:9453/

```

The parameters used above are as follows:

Parameter	Description
name	A descriptive name to identify the server group.
hosts	The hosts correspond to the individual servers in the server group. The hosts should have the following syntax: <code>https://&lt;hostname&gt;/&lt;port&gt;/&lt;web-context-root&gt;</code> . Usually, the <code>&lt;web-context-root&gt;</code> part is empty. For example <a href="https://localhost:9443/">https://localhost:9443/</a> .

Optional parameters

Parameter	Description
allowParallelExecution	Allows you to execute a given task in parallel for all hosts defined in this server group. When tasks are scheduled for this server group, the task will be executed separately for each host. An index will be added to the server group, which maps to the order defining the hosts. By default this parameter is set to <code>false</code> .

#### Updating tasks

Tasks are configured in the deployment-monitor.yaml file as shown below. Let's look at the configuration of one of the default tasks, for example, **Patch Verification**:

```

tasks:
  -
    name: PatchVerificationTask
    enable: true
    className:
      org.wso2.deployment.monitor.impl.task.patch.PatchVerificationTask
    onResult: org.wso2.deployment.monitor.impl.callback.DefaultCallback
    triggerType: simple
    trigger: 60m
    servers: [$ALL]
    taskParams:
      severity: 2

```

The parameters used above to configure the task are explained below.

Task Parameter	Description
name	The name of the task.
enable	This property enables or disables the task. You can set the values to true or false.
className	An implementation of the DeploymentMonitorTask interface, which defines the actual task execution logic.
onResult	
triggerType	Possible values: [simple, cron, onetime]. This configures how the trigger element will be interpreted. cron allows you to define a crontab for the trigger. The default value is simple.
trigger	This parameter applies when the task is run on <b>Scheduled</b> mode. That is, the value given here specifies the frequency of task execution in minutes. To find out how to schedule tasks, see <a href="#">Monitoring Product Deployment</a> (which uses the web application) or see <a href="#">Using the Command Line</a> for instructions relevant to the command line tool.  The default value is 60m.
servers	This parameter allows you to enable the task only for specific users or for all servers. Use \$ALL to run it against all server groups or give a specific server group name. Configuring server groups were explained previously.  Note that all the default tasks are set to \$ALL.
taskParams	These are custom task parameters that can be linked to the DeploymentMonitorTask as indicated by the className element.

To implement custom monitoring tasks see [Defining Custom Monitoring Tasks](#).

#### Updating Notifications

Just like server groups and tasks, configuring notifications is done in the deployment-monitor.yaml file. Let's look at the default configurations for email and SMS notifications:

```

notifications:
  email:
    enabled: false
    tlsEnabled: true
    authentication: true
    username: username
    password: password
    smtpServer: smtp.server
    smtpPort: 25
    fromAddress: from@wso2.com
    toAddresses: [to@wso2.com]
    subjectPrefix: "[WSO2DM]"
  sms:
    enabled: false
    provider: provider
    endpoint: endpoint
    apiID: id
    username: username
    password: password
    recipients: [+9471XXXXXXX,+9477XXXXXXX]
    messagePrefix: WSO2DM

```

The parameters used for **Email** configurations:

Config name	Description
enabled	This property enables or disables email notifications. You can set the values to <code>true</code> or <code>false</code> . By default, email notifications are disabled.
tlsEnabled	Specifies whether or not TLS is enabled for the SMTP server. The default value is <code>true</code> .
authentication	Specifies whether or not authentication is enabled for the SMTP server. The default value is <code>true</code> .
username	If authentication is enabled, this username is used to connect to the server.
password	If authentication is enabled, this password is used to connect to the server.
smtpServer	The hostname of the SMTP server used for sending emails.
smtpPort	The port of the SMTPserver used to send emails.
fromAddress	The address from which the emails will be sent.
toAddresses	The list of addresses to send emails. The addresses should be comma separated in the following format: <code>[address1@mail.com,address2@mail.com,address3@mail.com]</code>
subjectPrefix	You can define a prefix for the Subject of the Mail sent. This is useful to create Filtersformails. A sample prefix " <code>[WSO2 Deployment Monitor]</code> "

The parameters used for **SMS** configurations:

Config name	Description
-------------	-------------

enabled	This property enables or disables SMS notifications. You can set the values to true or false. By default, SMS notifications are disabled.
provider	Specifies the SMS provider being used ( <b>Clickatell</b> and <b>bulkSMS</b> ).
endpoint	The HTTP endpoint of the SMS service.
apiID	If the SMS provider isclickatell, provide the API ID. Forbulksms, leave the default value.
username	Username for the SMS service.
password	Password for the SMS service.
recipients	The list of phone numbers to send SMS to. The numbers should be comma separated in the following format: [+9471XXXXXXX,+9477XXXXXXX,+1650XXXXXXX]
messagePrefix	You can define a prefix for the SMS being sent. A sample prefix is WSO2DM.

### ***Updating the global configurations***

The global configurations section defines a set of default configuration values that will be common to all the tasks and server groups.

```
global:
  onResult: org.wso2.deployment.monitor.impl.callback.SimpleLoggingCallBack
  keyStore: resources/wso2carbon.jks
  keyStorePassword: wso2carbon
  trustStore: resources/client-truststore.jks
  trustStorePassword: wso2carbon
  tomcatPort: 8888
  servicesPort: 8889
  tenant:
    domain: carbon.super
    tenantID: -1234
    username: admin
    password: admin
```

The following table describes the above elements:

Config name	Description
onResult	Default callback implementation. This can be over-ridden at the task/server-group level if required. The defualt value is org.wso2.deployment.monitor.impl.callback.SimpleLoggingCallBack.
keyStore	The path to the keystore used by the Carbon monitoring tool, which is resources/wso2carbon.jks.
keyStorePassword	The password to access the keystore. Both the keystore and private key password should be the same. The default password is wso2carbon.
trustStore	The path to the trust store used by the tool, which is resources/client-truststore.jks.

trustStorePassword	The password to access the truststore. The default password is wso2carbon.
tomcatPort	The tomcat port on which the web application of the Carbon Monitoring Tool will open. The default value is 8888.
servicesPort	The port on which the Carbon Monitoring Tool service is exposed. This is only applicable when you are using the command line tool. The default values is 8889.
tenant	This specifies the super tenant domain of the WSO2 product servers. Note that the Carbon Monitoring Tool can only be used for monitoring the super tenant domain. The default super tenant details are specified: <ul style="list-style-type: none"> <li>• domain: carbon.super</li> <li>• tenantID: -1234</li> <li>• username: admin</li> <li>• password: admin</li> </ul>

### Updating services

A running instance of the Carbon Monitoring tool can be exposed as a service using the following configuration. **This service is only applicable when the product is run in scheduled mode. You can enable/disable tasks during runtime using this service.** Note that this service is written on top of WSO2 MSF4J framework. The api definition can be found by pointing your browser to <http://localhost:8090/swagger>.

```
service:
  enabled: true
  port: 8090
  serviceClasses:
    - org.wso2.deployment.monitor.service.DeploymentMonitorService
```

The following table describes the above elements:

Config name	Description
enabled	Specifies whether or not the service is enabled. The default value is true.
port	The http port this service will get exposed under
serviceClasses	The name of the MSF4J service's class. The default class is org.wso2.deployment.monitor.service.DeploymentMonitorService. You can replace this with your own implementation.

### Adding users to the Carbon monitoring tool

You can add users with permission to access the web application of the Carbon Monitoring Tool from the users.xml file. By default, the admin user is defined as shown below.

```
<tomcat-users>
  <role rolename="admin"/>
  <user username="admin" password="admin" roles="admin"/>
</tomcat-users>
```

***Configuring logging for the Carbon monitoring tool***

## Using the Command Line

The Carbon Monitoring tool is typically used as a web application as explained in [Monitoring Product Deployment](#). However, you also have the option of using this tool on a command line as explained below.

**Before you begin:**

- Get an understanding of how the Carbon Monitoring tool works.
- Be sure that the WSO2 server groups and monitoring tasks are set up as explained [here](#).
- Be sure that the [required permissions](#) are configured.
- Be sure that Java 1.8 or a higher JDK is installed in your system.

To run the WSO2 Carbon Monitoring Tool as a simple command-line tool:

1. Download the WSO2 product that you would like to monitor via <http://wso2.com>.
2. Unzip the product distribution. We will refer to this directory as <PRODUCT\_HOME> .
3. Download the WSO2 Deployment Monitoring tool ZIP from here.
4. Extract the ZIP file. We will refer to this extracted directory as <DEPLOYMENT\_MONITOR\_HOME> .
5. Copy the wso2-monitor-agent-x.x.x.jar file (stored in the <DEPLOYMENT\_MONITOR\_HOME> directory) to the <PRODUCT\_HOME>/repository/components/dropins directory.
6. Now, start the WSO2 product by executing the following command:
  - On Windows: <PRODUCT\_HOME>/bin/wso2server.bat
  - On Linux: <PRODUCT\_HOME>/bin/wso2server.sh
7. Now, let's start the deployment monitoring tool and execute the relevant monitoring tasks simultaneously:
  1. Open a command prompt and navigate to the <DEPLOYMENT\_MONITOR\_HOME>/bin directory.
  2. Execute one of the following commands depending on the default deployment task you want to execute:
 

Deployment Task	Command
Execute all deployment tasks.	run -all
Deployment Synchronizer verification task	run DeploymentSynchronizerTestTask
Patch verification task	run PatchVerificationTask
Server availability verification task	run server-info
Service availability verification task	run PingTask
Tenant login verification task	run TenantLoginTask

8. Analyze the log output that is published on your terminal.

Now, let's go a step further and add another server to the server group.

1. Unzip the product distribution to a new location. Keep the previous one running. Let's refer it as <PRODUCT\_HOME\_2>.
2. Start the second product server using the following command. As you see, we've set a port offset to avoid port conflicts.

```
<PRODUCT_HOME_2>/bin/wso2server.sh -DportOffset=10
```

3. By default, the server group configured in the deployment-monitor.yaml file handles a server at <https://localhost:9443/>. Add the new server to the server group using the following command:

```
DEPLOYMENT_MONITOR_HOME/bin/wso2monitor.sh server add -group
esb-cluster -host https://localhost:9443/
DEPLOYMENT_MONITOR_HOME/bin/wso2monitor.sh server add -group
esb-cluster -host https://localhost:9453/
```

4. Finally, execute all the task in the monitoring tool by executing the following command:

```
DEPLOYMENT_MONITOR_HOME/bin/wso2monitor.sh run -all
```

5. Analyze the log output

## Defining Custom Monitoring Tasks

WSO2 Carbon Monitoring tool is shipped with a set of default tasks as explained here. You easily write custom tasks and plug it to the Carbon Monitoring tool using the following resources:

- A good starting point is the `org.wso2.deployment.monitor.api` package. A sample task can be found at [org.wso2.deployment.monitor.impl.task.SampleLoggingTask](#).
- The Java docs are hosted [here](#).

## 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:

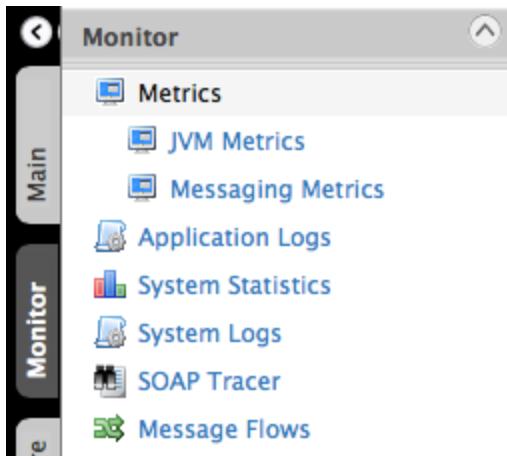
<b>Name:</b>	WSO2	Carbon	-	SOAP	Tracer	Feature
<b>Identifier:</b>	<code>org.wso2.carbon.soaptracer.feature.group</code>					

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.

This screenshot shows the 'SOAP Message Tracer' configuration page. A dropdown menu labeled 'Enable SOAP Tracing?' has 'Yes' selected, which is highlighted with a red box. A warning message at the bottom states: 'WARNING: Turning on SOAP tracing may have impose a significant performance overhead. Use with caution.'

3. The tracer will show the messages of the operations that were invoked. For example,

This screenshot displays the SOAP Message Tracer interface. At the top, there's a 'Configuration' section with a 'Filter' input field, a 'Search' button, and a 'Clear' button. Below it is a 'Messages' section showing a list of captured operations:

- Version.getVersion [0] - Mon, 23 Jan 2012 14:41:59 IST
- echo.echoInt [1] - Mon, 23 Jan 2012 14:41:34 IST
- echo.echoInt [0] - Mon, 23 Jan 2012 14:41:22 IST

At the bottom, there are 'Refresh' and 'Clear all SOAP messages' buttons. The main area is divided into 'Request' and 'Response' panes, each containing XML code. The 'Request' pane shows the XML for a Version.getVersion call, and the 'Response' pane shows the XML for a getVersionResponse.

```

Request:
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
    <wsa:To>http://10.100.3.56:9763/services/Version</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>urn:uuid:4a5a5a5a-4a5a-4a5a-a5a4-4a5a5a5a5a5a</wsa:MessageID>
    <wsa:Action>urn:getVersion</wsa:Action>
  </s:Header>
  <s:Body></s:Body>
</s:Envelope>

Response:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <wsa:Action>urn:getVersionResponse</wsa:Action>
    <wsa:RelatesTo>http://identifiers.org/urn:uuid:4a5a5a5a-4a5a-4a5a-a5a4-4a5a5a5a5a5a</wsa:RelatesTo>
  </soapenv:Header>
  <soapenv:Body>
    <ns:getVersionResponse xmlns:ns="http://identifiers.org/urn:uuid:4a5a5a5a-4a5a-4a5a-a5a4-4a5a5a5a5a5a">
      <return>WSO2 Data Services Server</return>
    </ns:getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>

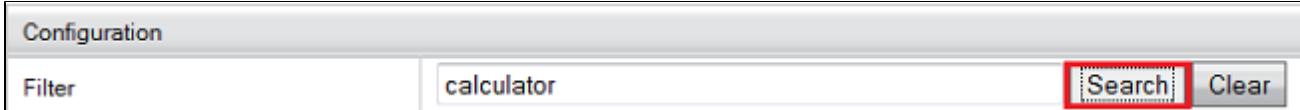
```

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,

The screenshot shows the SOAP Tracer interface with a 'Messages' list containing a single item: 'Calculator.add [2] - Mon, 10 Oct 2011 21:46:00 IST'. This item is highlighted with a red box. Below the list are 'Refresh' and 'Clear all SOAP messages' buttons. The main area is divided into 'Request' and 'Response' tabs, each showing the XML code of the selected message. The 'Request' tab contains the client-side XML, and the 'Response' tab contains the server-side XML, both with line numbers.

```

Request:
1 <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
2   <s:Header>
3     <wsa:To>http://192.168.1.5:9763/services/Calculator.Calc</wsa:To>
4     <wsa:ReplyTo>
5       <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
6     </wsa:ReplyTo>
7     <wsa:MessageID>http://identifiers.wso2.com/messageid/131</wsa:MessageID>
8     <wsa:Action>add</wsa:Action>
9   </s:Header>
10  <s:Body>
11    <p:add xmlns:p="http://calculator.jaxws.sample.appserver">
12      <!--Exactly 1 occurrence-->
13        <p:value1>?</p:value1>
14      <!--Exactly 1 occurrence-->

```

```

Response:
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
2   <soapenv:Header>
3     <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
4       <wsa:Action>http://www.w3.org/2005/08/addressing/anonymous</wsa:Action>
5     </soapenv:Header>
6     <soapenv:Body>
7       <soapenv:Fault>
8         <faultcode>soapenv:Server</faultcode>
9         <faultstring>java.lang.IllegalAccessError</faultstring>
10        <detail />
11      </soapenv:Fault>
12    </soapenv:Body>
13  </soapenv:Envelope>

```

## 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

```
./tcpmon.sh
```

#### ***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

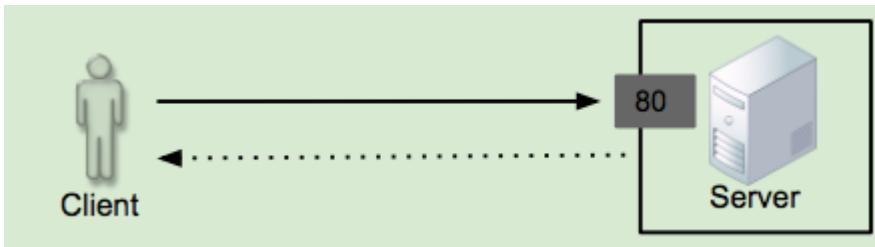
```
./tcpmon.sh
```

## Message Monitoring with TCPMon

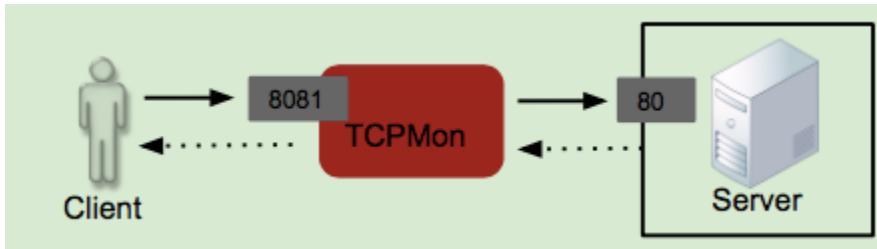
The most common usage of TCPMon is as an intermediary, which monitors the communication between the client (front end) and the back end server. That is, the messages sent from the client are received by the intermediary instead of the back end server. These messages are then forwarded to the back end 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:

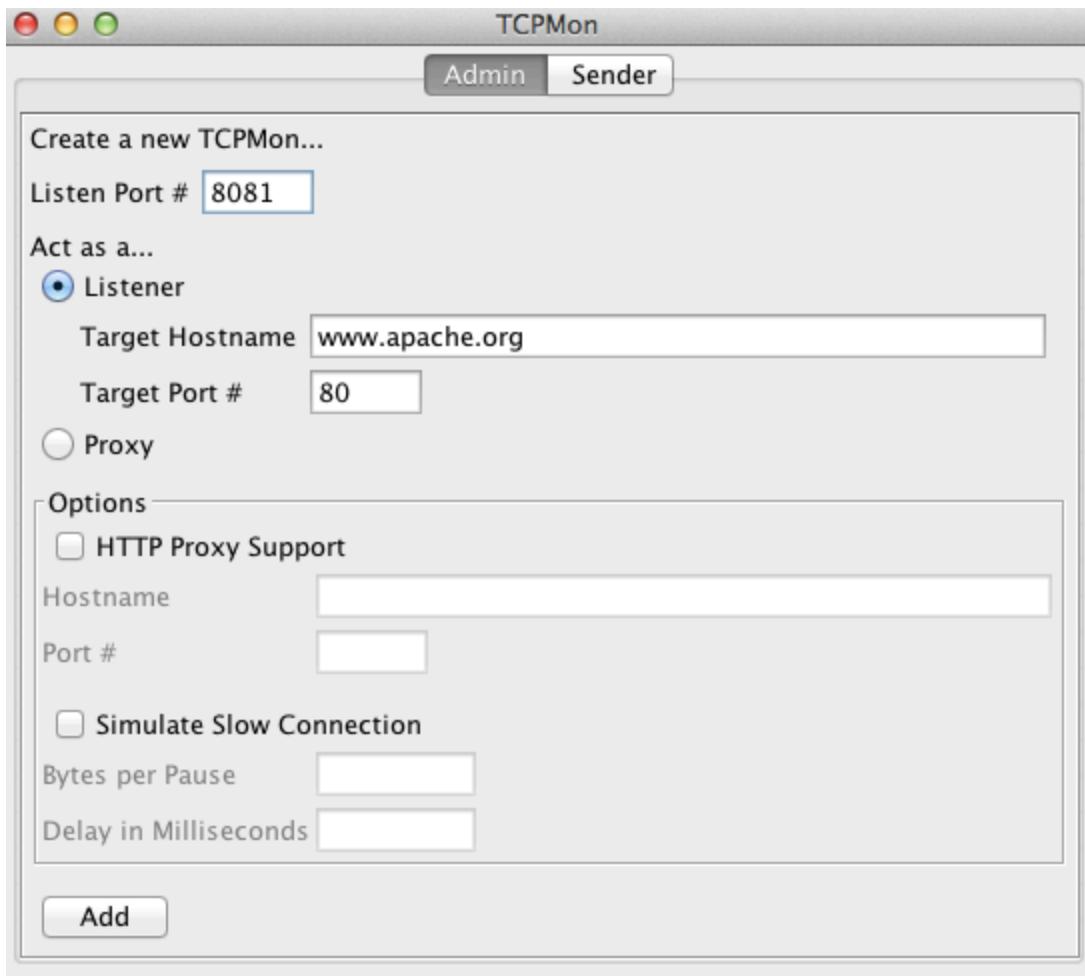


As an intermediary, TCPMon only receives messages and forwards them to the back end server. Therefore, it is a safe tool to be used for debugging purposes.

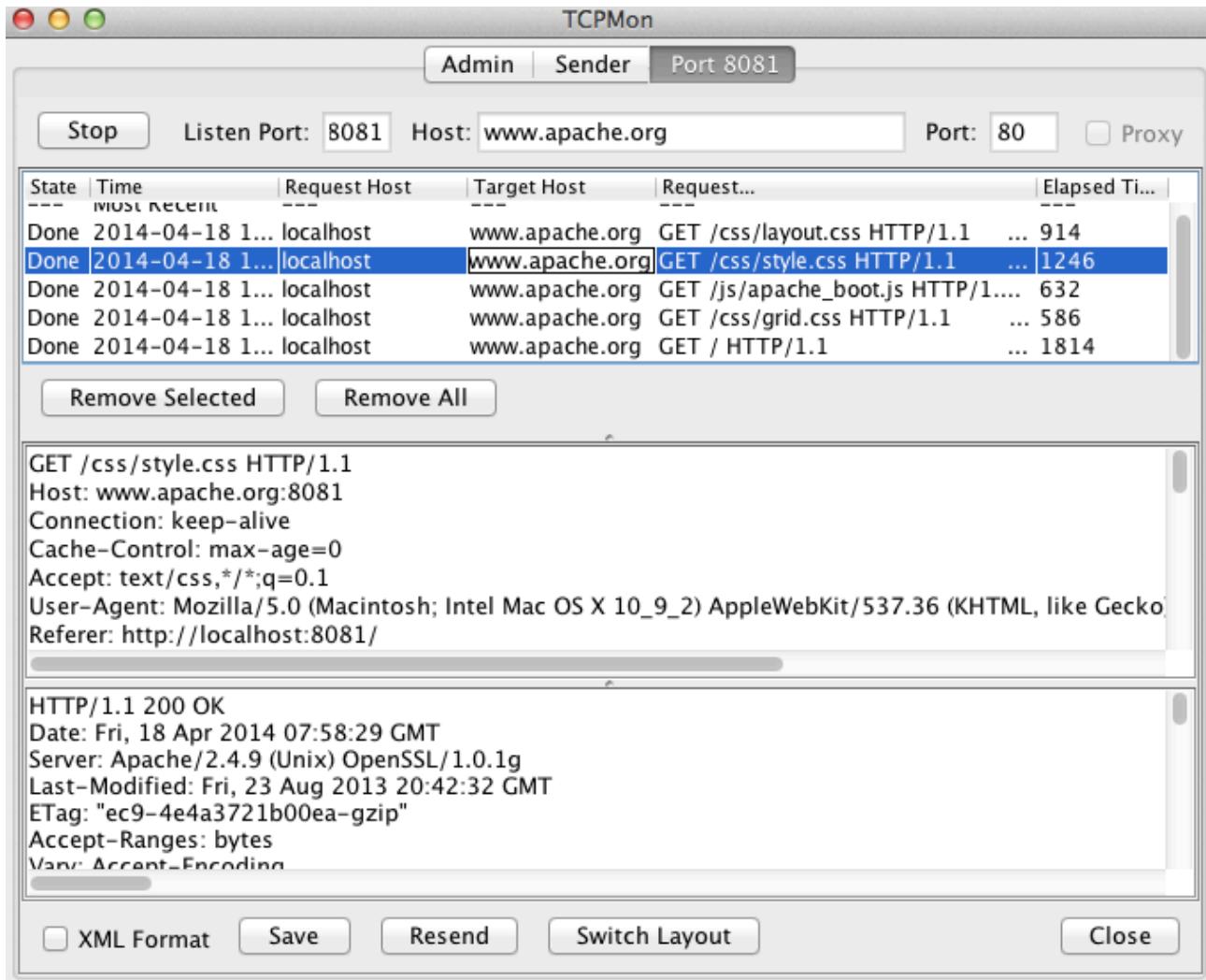
Note that TCPMon cannot be used to view messages transferred over https protocol.

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](http://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](http://www.apache.org).



5. Click **Add** to save the setting.
  6. Now, point the browser to 'localhost:8081' instead of [www.apache.org](http://www.apache.org).
- A screenshot of a web browser's address bar. The URL 'http://localhost:8081' is entered, and the browser is ready to load the page.
7. A new tab in TCPMon will indicate the 8081 port. You can view the requests and responses passing through TCPMon as shown below.



8. 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.



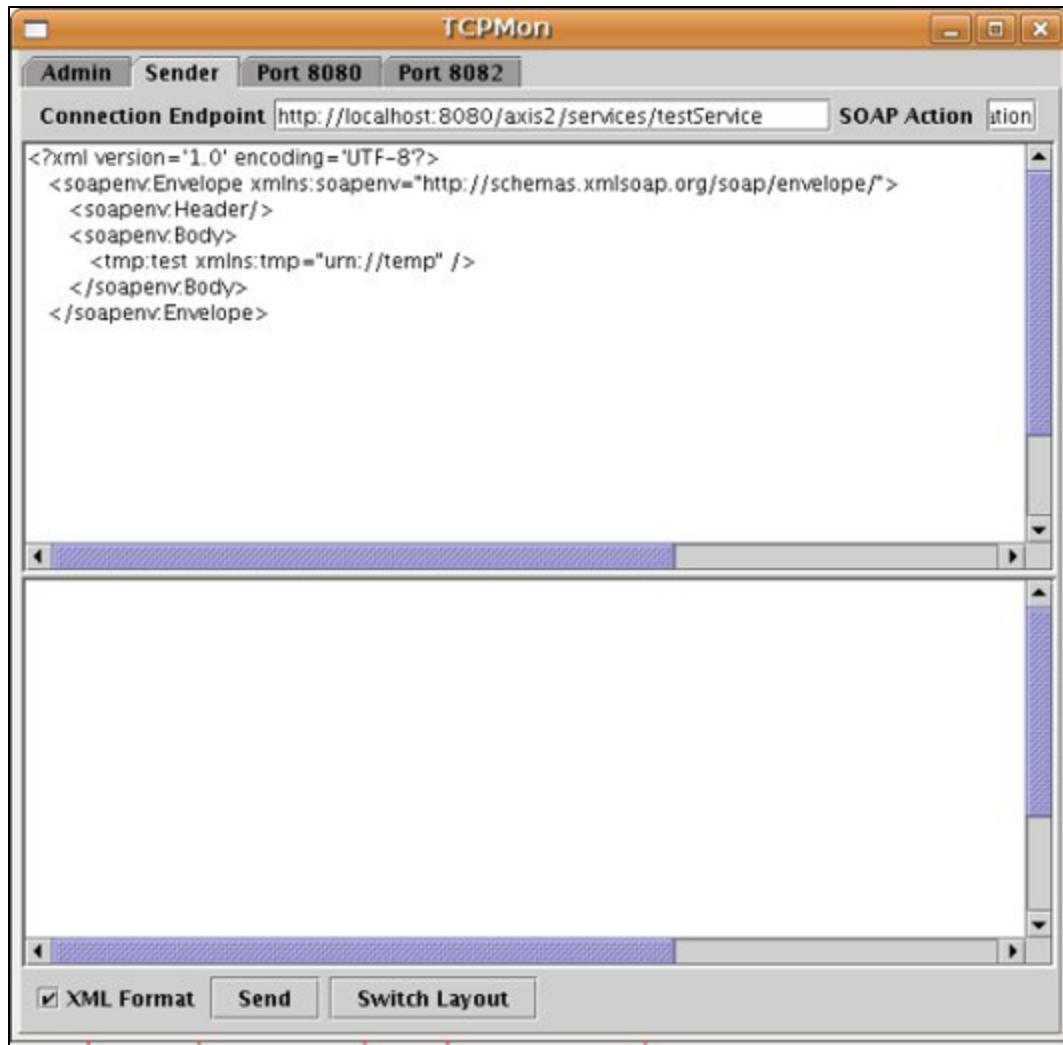
## Other Usages of TCPMon

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.



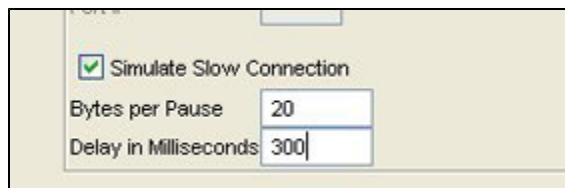
## As a Proxy

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.



Also, if HTTP proxy support is required, that can also be set on the admin screen.

## Monitoring with AppDynamics

This page is currently under review and restricted to WSO2 users.

AppDynamics is a product that can be used to monitor the runtime performance of an application. This includes end-user monitoring as well as business performance monitoring. For example, you can monitor various aspects of an application's performance, such as CPU usage, memory, response times, etc.

The steps below will guide you through the process of monitoring a WSO2 product using AppDynamics. We will use WSO2 API Manager (WSO2 APIM) as the product in this example.

- Step 1: Download and set up the AppDynamics agent
- Step 2: Download and setup the WSO2 product
- Step 3: Start the WSO2 product and view statistics

### Step 1: Download and set up the AppDynamics agent

Follow the steps given below.

1. Go to <https://www.appdynamics.com> and sign up for the free trial.
2. Start your free trial on AppDynamics. You will now be signing into the AppDynamics portal.
3. Click **Subscriptions** and select **SAAS** to open the AppDynamics management platform.
4. Click **Launch Appdynamics** to open the portal for AppDynamics SAAS. Proceed to download the app agent as follows:
  1. On the **Home** tab of the portal, select **Java**. This specifies the type of server-side application that we are going to monitor (which is WSO2 APIM).
  2. You will now be able to download the app agent for your Java application.
5. Unzip the downloaded app agent, which we will refer to as <APP\_AGENT\_HOME> from hereon.
6. To configure the app agent, open the `controller-info.xml` file located in the <APP\_AGENT\_HOME>/conf directory and add values for the **application name**, **tier name**, and **node name** as shown below. Note that 'application' corresponds to the WSO2 product.

```
<application-name>AppName</application-name>
<tier-name>TierName</tier-name>
<node-name>NodeName</node-name>
```

### Step 2: Download and setup the WSO2 product

Proceed with the following steps:

1. Download WSO2 API Manager from [here](#).
2. Open the product startup script (`wso2server.sh` or `wso2server.bat`) located in the <APIM\_HOME>/bin directory and insert the following, where <APP\_AGENT\_HOME\_PATH> is the directory path to the location of your <APP\_AGENT\_HOME> directory.

```
-javaagent:<APP_AGENT_HOME_PATH>/javaagent.jar" \
```

### Step 3: Start the WSO2 product and view statistics

Proceed with the following steps:

1. Start the WSO2 APIM server by executing one of the following scripts from the <APIM\_HOME>/bin directory.
  1. On Linux: wso2server.sh
  2. On Windows: wso2server.bat
2. Go back to the AppDynamics portal and go to the **Applications** tab. See that your WSO2 APIM server is listed as an application:

The screenshot shows the AppDynamics Applications dashboard. At the top, there are tabs for Home, Applications (which is selected and highlighted in blue), and User Experience. Below the tabs, the title 'Applications' is displayed. Underneath, there's a search bar with a magnifying glass icon and a 'Create Application' button. To the right of the search bar are icons for Details, Actions, View (grid, list, network), Load, and Sort. A dropdown menu labeled 'Load' is open. The main content area displays a summary for an application named 'AppName'. It includes metrics: Response Time (0 ms), Calls (0), Calls / min (0), Error % (0.0 %), Errors (0), and Errors / min (0). Below these metrics are two progress bars: 'Business Transaction Health' (grey) and 'Node Health' (green).

3. Select the application, and then open the product node (which is listed under **Tiers & Nodes**). You will now see the statistics for the WSO2 APIM node that we have started.

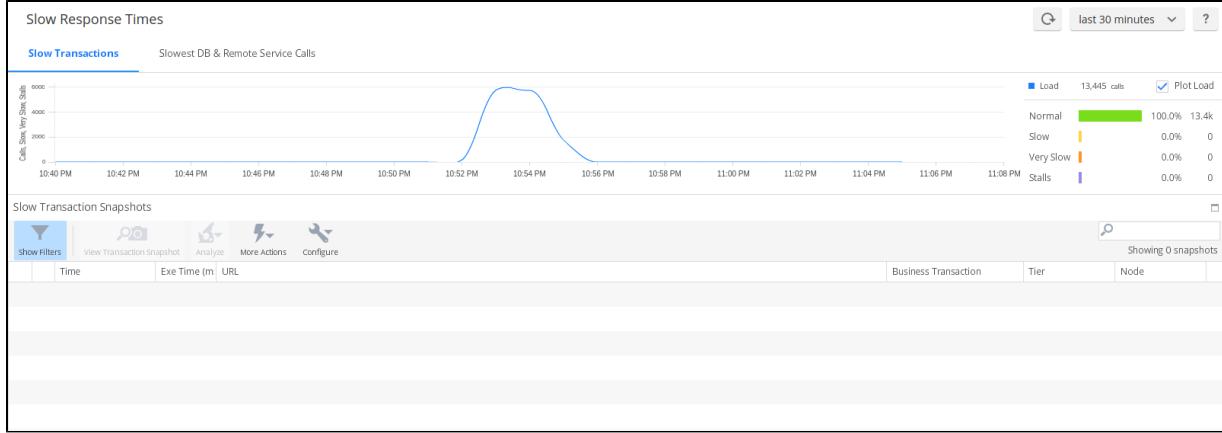
Shown below are some of the statistics that can be viewed for a particular node of the product server.

The screenshot shows the AppDynamics Node Statistics dashboard for the 'NodeName' node. The top navigation bar includes tabs for Home, Applications (selected), User Experience, Databases, Servers, Analytics, Dashboards & Reports, Alert & Respond, and a settings gear icon. Below the tabs, there are dropdown menus for 'NodeName' and time periods ('Daily Trend - Last 30 days', 'last 3 days', '?'). The main content area has tabs for Dashboard, My Dashboards, Container, Server, Memory (selected), JVM, JMX, Events, Slow Response Times, and '>>'. The 'Memory' tab is active, showing sub-options: Heap & Garbage Collection (selected), Automatic Leak Detection, Object Instance Tracking, and Custom Memory Structures.

Some of the above statistics are explained below.

- **Memory Usage:** From this view, we can find the memory utilization, GC cycles and timings, etc.
- **Object Tracking:** This view shows the number of objects created during the runtime of the server.
- **Automatic Leak Detection:** This tab shows us the memory leak objects and its stack traces. As shown by the graph below, if the size of a specific object does not decrease within 15 minutes, it will be indicated as a memory leak object. Further, the path through which the object is created and

accessed will be detected. This graph can be used to drill down to the reasons behind response time slowness.



## 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](#)

## 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.

Element Name	Description	Type	Default Value	Mandatory/Optional
<b>Enabled</b>	This parameter specifies whether or not metrics monitoring is enabled for JMX.	Boolean	true	Mandatory

### CSV

The following parameters in the `metrics.xml` file can be used to configure a CSV storage for metrics data.

Element Name	Description	Type	Default Value	Mandatory/Optional
<b>Enabled</b>	This parameter specifies whether or not metrics monitoring is enabled for CSV.	Boolean	false	Mandatory
<b>Location</b>	The location where the CSV files are stored.	String	<code> \${carbon.home}/repository/logs/metrics/</code>	

<b>PollingPeriod</b>	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.	Integer	60	
----------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------	----	--

## JDBC

The following parameters in the `metrics.xml` file can be used to configure a JDBC storage for metrics data.

Element Name	Description	Type	Default Value
<b>Enabled</b>	This parameter specifies whether or not metrics monitoring is enabled for JDBC.	Boolean	true
<b>DataSourceName</b>	The name of the datasource used.	String	jdbc/WSO2MetricsI
<b>PollingPeriod</b>	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.	Integer	60
<b>ScheduledCleanup</b>	This element contains parameters relating to scheduled cleanup. The possible values are <code>Enabled</code> , <code>ScheduledCleanupPeriod</code> and <code>DaysToKeep</code> . 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.		
<b>ScheduledCleanup/Enabled</b>	This parameter specifies whether or not scheduled cleanup is enabled.	Boolean	true
<b>ScheduledCleanup/ScheduledCleanupPeriod</b>	The number of milliseconds that should elapse after a clean-up task before the next clean-up task is carried out.	Integer	86400000

<b>ScheduledCleanup/DaysToKeep</b>	The number of days during which the scheduled clean-up task should be carried out.	Integer	
------------------------------------	------------------------------------------------------------------------------------	---------	--

If you have enabled JDBC, then you also need to specify a datasource configuration, which will be used to create the connection between WSO2 Message Broker 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:

XML element	Attribute	Description	
<code>&lt;datasources-configuration&gt;</code>	<code>xmlns</code>	The root element. The namespace is specified as: <code>xmlns:svns:p://org.wso2.securevault/configuration"</code>	
<code>&lt;providers&gt;</code>		The container element for the datasource providers.	
<code>&lt;provider&gt;</code>		The datasource provider, which should implement <code>org.wso2.carbon.datasource.common.spi.DataSourceReader</code> . The datasources follow a pluggable model in providing datasource type implementations using this approach.	
<code>&lt;datasources&gt;</code>		The container element for the datasources.	
<code>&lt;datasource&gt;</code>		The root element of a datasource.	
<code>&lt;name&gt;</code>		Name of the datasource.	
<code>&lt;description&gt;</code>		Description of the datasource.	
<code>&lt;jndiConfig&gt;</code>		The container element that allows you to expose this datasource JNDI datasource.	
<code>&lt;name&gt;</code>		The JNDI resource name to which this datasource will be bound.	
<code>&lt;environment&gt;</code>		<p>The container element in which you specify the following properties:</p> <ul style="list-style-type: none"> <li><code>java.naming.factory.initial</code>: Selects the registry service provider as the initial context.</li> <li><code>java.naming.provider.url</code>: Specifies the location of the registry when the registry is being used as the initial context.</li> </ul>	
<code>&lt;definition&gt;</code>	<code>type</code>	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 "RDBMS" datasource reader expects a <code>configuration</code> element and the sub elements listed below.	
<code>&lt;configuration&gt;</code>		The container element for the RDBMS properties.	
<code>&lt;url&gt;</code>		The connection URL that passes the JDBC driver to establish the connection.	

<username>		The connection user name that passes the JDBC driver to establish connection.
<password>		The connection password that passes the JDBC driver to establish connection.
<driverClassName>		The class name of the JDBC driver.
<maxActive>		The maximum number of active connections that can be allocated from this pool at the same time.
<maxWait>		Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception.
<testOnBorrow>		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.
<validationQuery>		The SQL query that is used for validating connections from this pool before returning them to the caller. If 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).
<validationInterval>		To avoid excess validation, a connection will be validated at this frequency, at most (interval time in milliseconds). If a connection has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).

## Sample configuration

Shown below is a sample `metrics.xml` file with the default configurations specifying the types of storages enabled for metrics data. See the above topics for instructions.

### ▼ The default configurations in the metrics.xml file

```
-->

<!--
    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.
        -->
        <DataSourceName>jdbc/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.
            -->
            <Enabled>true</Enabled>
            <!-- This is the period for each cleanup operation in
        seconds -->
            <ScheduledCleanupPeriod>86400</ScheduledCleanupPeriod>
            <!-- The scheduled job will cleanup 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:

Level	Description
<b>Off</b>	Designates no informational events.
<b>Info</b>	Designates informational metric events that highlight the progress of the application at coarse-grained level.
<b>Debug</b>	Designates fine-grained informational events that are most useful to debug an application.
<b>Trace</b>	Designates finer-grained informational events than the DEBUG.
<b>All</b>	Designates all the informational events.

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.

```
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

Property	Default Level	Description
metric.level.jvm.buffers	OFF	The gauge showing the current number of distinct buffers.

### Class loading

Property	Default Level	Description
metric.level.jvm.class-loading	INFO	The gauge showing the number of classes currently loaded for the JVM.

### GC

Property	Default Level	Description
metric.level.jvm.gc	DEBUG	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.

### Memory

Property	Default Level	Description
metric.level.jvm.memory	INFO	The gauge for showing the used and committed memory in WSO2 MB.
metric.level.jvm.memory.heap	INFO	The gauge for showing the used and committed heap in WSO2 MB.
metric.level.jvm.memory.non-heap	INFO	The gauge for showing the used code cache and used CMS Perm Gen in WSO2 MB.
metric.level.jvm.memory.total	INFO	The gauge for showing the total memory currently available for the JVM.
metric.level.jvm.memory.pools	OFF	The gauge for showing the used and available memory for JVM in the memory pool.

### Operating system load

Property	Default Level	Description

metric.level.jvm.os	INFO	The gauge for showing the current load imposed by the JVM on the operating system.
---------------------	------	------------------------------------------------------------------------------------

## Threads

Property	Default Level	Description
metric.level.jvm.threads	OFF	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.
metric.level.jvm.threads.count	DEBUG	The gauge for showing the number of active and idle threads currently available in the JVM thread pool.
metric.level.jvm.threads.daemon.count	DEBUG	The gauge for showing the number of active daemon threads currently available in the JVM thread pool.
metric.level.jvm.threads.blocked.count	OFF	The gauge for showing the number of threads that are currently blocked in the JVM thread pool.
metric.level.jvm.threads.deadlock.count	OFF	The gauge for showing the number of threads that are currently in deadlock in the JVM thread pool.
metric.level.jvm.threads.new.count	OFF	The gauge for showing the number of new threads generated in the JVM thread pool.
metric.level.jvm.threads.runnable.count	OFF	The gauge for showing the number of runnable threads currently available in the JVM thread pool.
metric.level.jvm.threads.terminated.count	OFF	The gauge for showing the number of threads terminated from the JVM thread pool since you started running the WSO2 MB instance.
metric.level.jvm.threads.timed_waiting.count	OFF	The gauge for showing the number of threads with <code>Time d_Waiting</code> status.
metric.level.jvm.threads.waiting.count	OFF	The gauge for showing the number of threads with <code>Waiting</code> 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.

## Using JVM Metrics

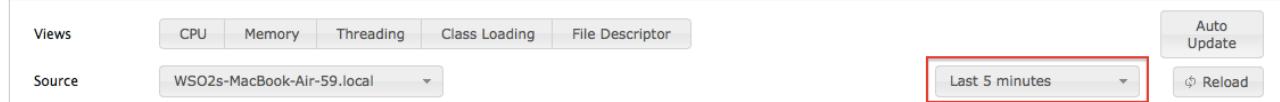
JVM metrics are Java metrics enabled by default in WSO2 products for the purpose of monitoring general statistics related to server performance. Follow the steps given below to use the JVM metrics dashboard in a WSO2 product.

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.



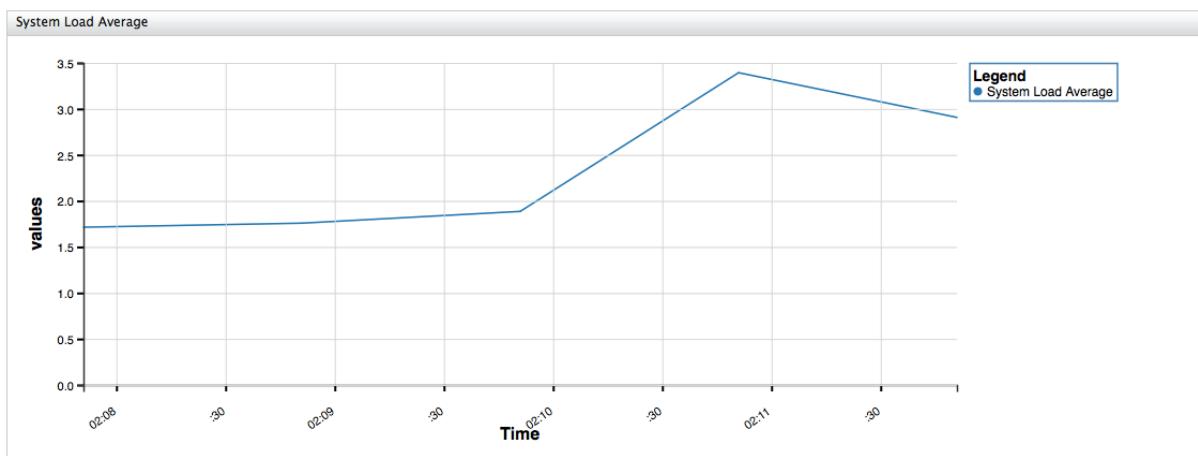
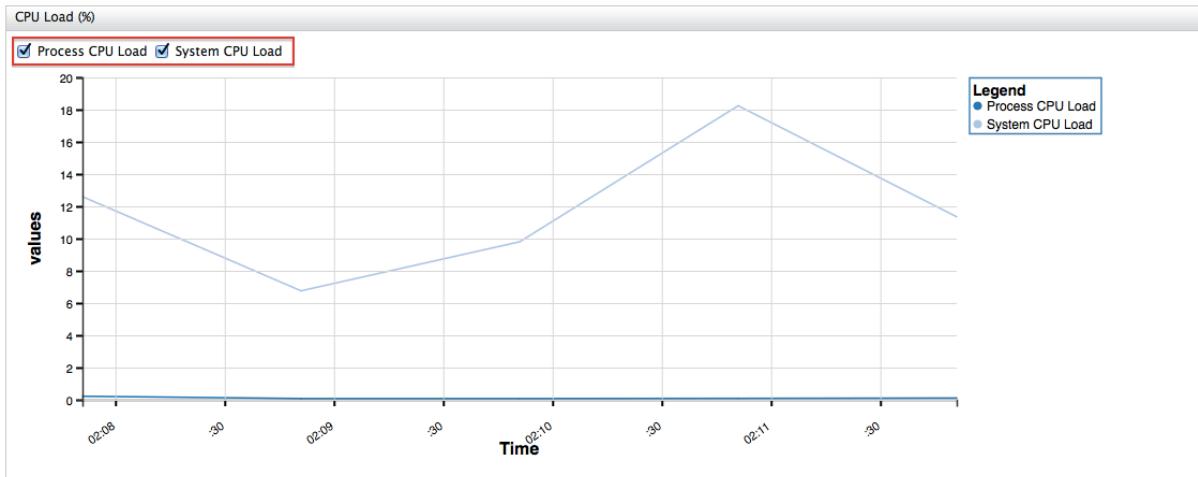
4. Click the required buttons opposite **Views** in the top panel to select the types of information you want to view in the dashboard and refresh the web page.



Statistics corresponding to each button can be viewed as follows:

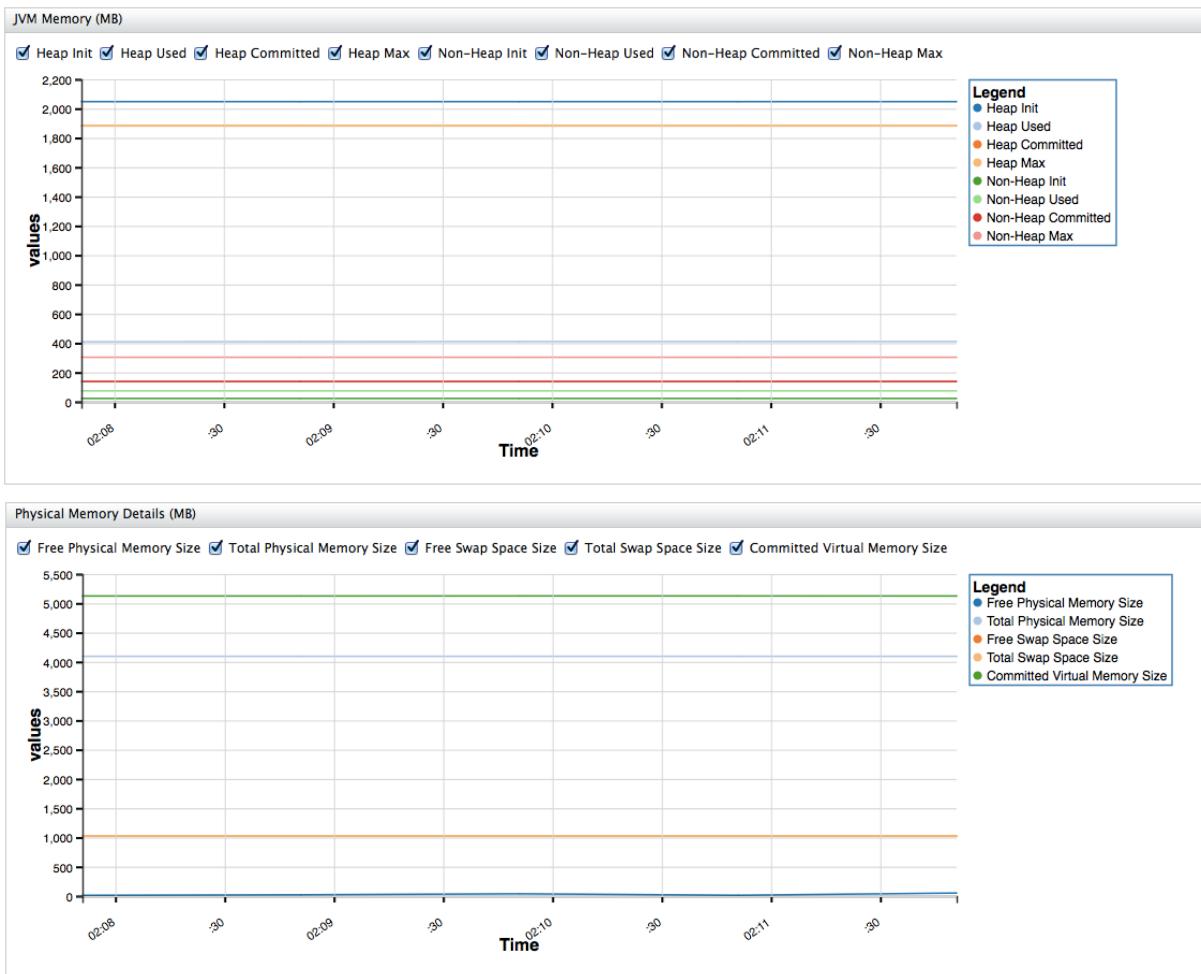
- **CPU**

Click this button to view statistics relating to the CPU as shown below.



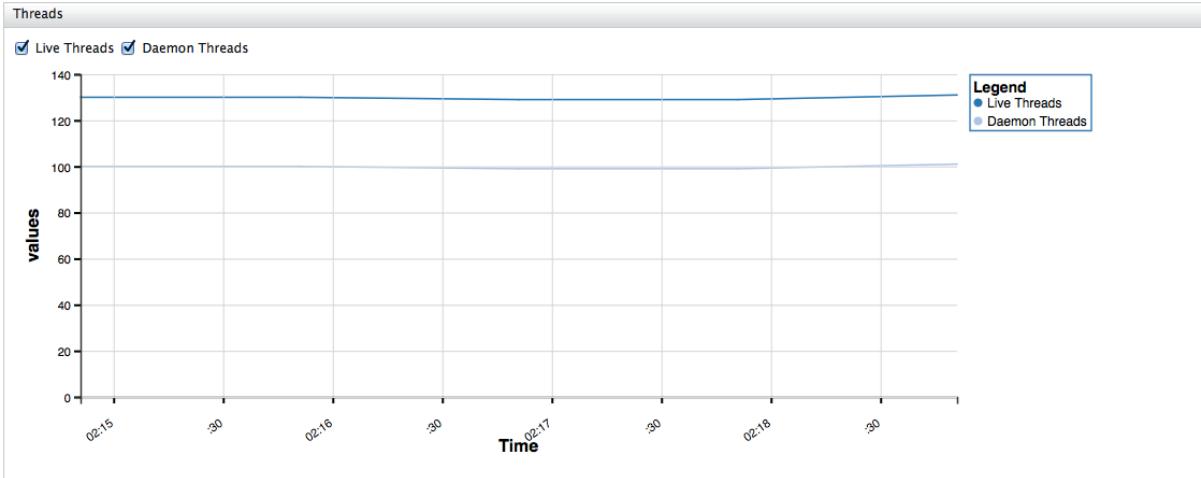
- **Memory**

Click **Memory** to view statistics relating to the memory as shown below.



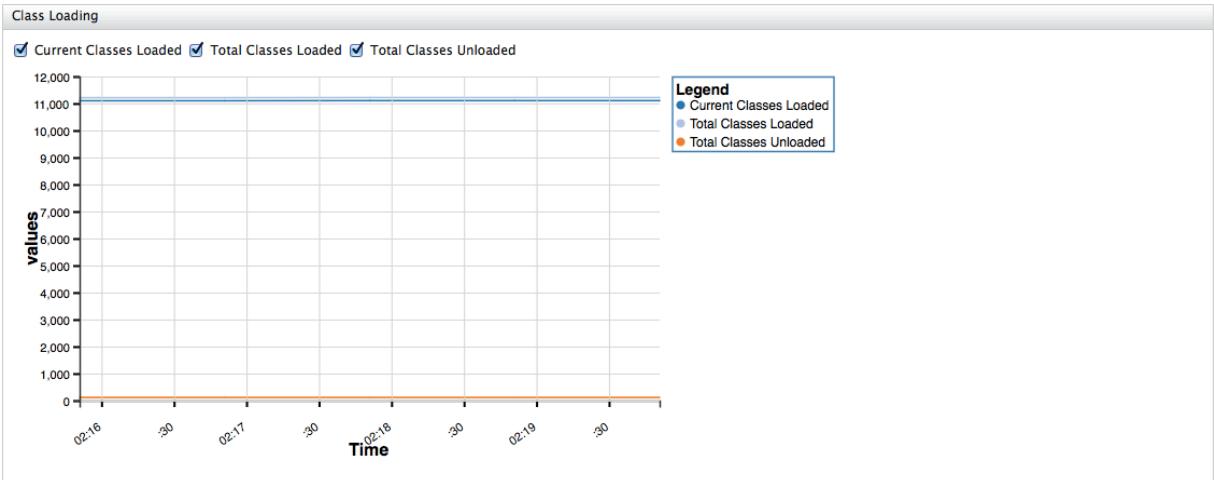
- **Threading**

Click **Threading** to view statistics relating to threading as shown below.



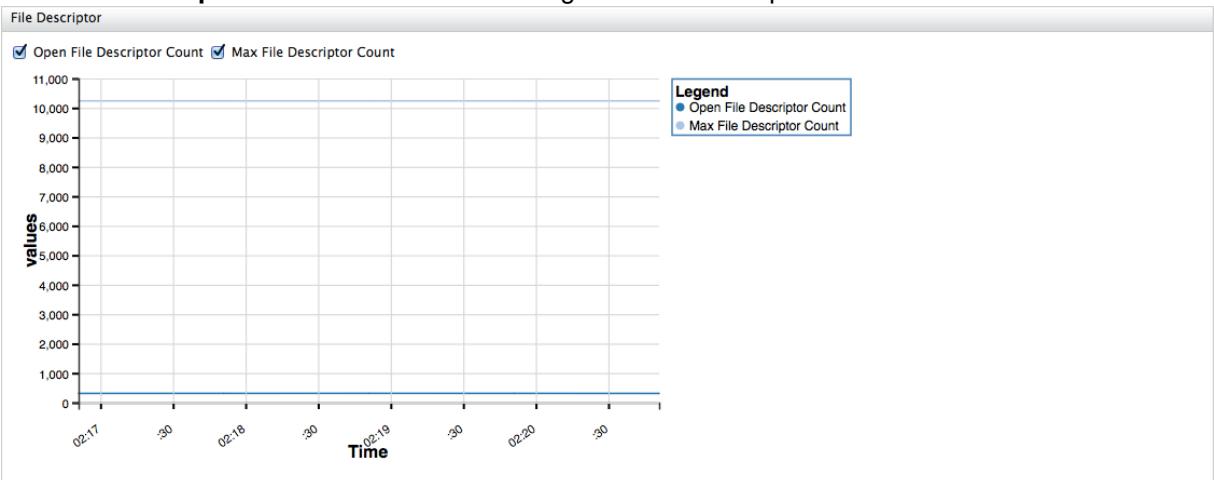
- **Class Loading**

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.



# Updating WSO2 Products

WSO2 introduces the [WSO2 Update Manager \(WUM\)](#), which is a command-line utility that allows you to get the latest updates that are available for a particular product release. These updates include the latest bug fixes and security fixes that are released by WSO2 after a particular product version is released. Therefore, you do not need to wait and upgrade to the next product release to get these bug fixes.

WUM works in conjunction with the WSO2 Update service to provide you with a new product distribution (ZIP file) that includes the updates. You can run WUM once to update the software to the latest set of released updates, or schedule it to run periodically so that you maintain the latest set of updates continuously.

Note that the updates you receive through WUM are licensed for free use in non-production environments only. To use updates in production, you are required to have a WSO2 Support Subscription. Find out more about the [license terms for WUM](#).

- Before you begin
- Download and get started with WUM
- Get the latest updates using WUM
  - Step 1: Initialize WUM
  - Step 2: Change the update repository location (Optional)
  - Step 3: Search for WSO2 products (Optional)
  - Step 4: Add the products that require updates
  - Step 5: Check for available updates (Optional)
  - Step 6: Update the products
- Manage products updated by WUM
  - Get details of products managed by WUM
  - Delete WSO2 products managed by WUM
- Verify the updates
- What's next?

## Before you begin

Note the following:

- To use the WSO2 Update Manager (WUM), you need to have a WSO2 account. If you don't already have one, you can register from here: <https://wso2.com/user/register>.

If you hold a production account with WSO2, please make sure you register for the service using your corporate email address so that we can match your account to your WSO2 support subscription. Find out more about the [license terms for WUM](#).

- WUM applies updates only to fresh product packs or to product packs that were previously updated using WUM. That is, you cannot apply updates to product packs that contain configuration changes, customizations or any other modifications. After each successful **wum update**, a new product pack will be created based on the timestamp of the update. You can then migrate all configurations and customizations from your existing production environment to the updated product. See [Moving Updates into Production](#) for more information.

## Download and get started with WUM

[Download](#) and install WSO2 Update Manager (WUM) to a location on your computer. Open a command prompt and get started with WUM.

Execute the following command to see the list of available WUM commands and flags:

```
wum
```

▼ WUM commands

<b>WUM command</b>	<b>Description</b>
init	Initialize WUM with your WSO2 credentials.
search	Search for WSO2 Products containing specific keyword(s).
add	Add or download a WSO2 product.
check-update	Check for new updates.
update	Update WSO2 products.
list	List the WSO2 products that are already managed by wum.
describe	Show details of WSO2 products managed by wum.
delete	Delete WSO2 products managed by wum.
config	Change the location of the local repository that was created when wum is initialized ( <code>init</code> command).
version	Display wum version information

▼ WUM command flags

<b>Flag</b>	<b>Description</b>
-h	Get help for a WUM command.
--help	
-v	Enable verbose mode.
--verbose	

You can execute the WUM commands as follows:

```
wum [ command ]
```

You can use flags with the WUM commands as follows:

```
wum [command] [flag]
```

## Get the latest updates using WUM

Once you have downloaded WUM, get the updates for your WSO2 product by executing the commands in the following order:

### Step 1: Initialize WUM

Start by initializing WUM with your WSO2 credentials.

1. Open a command prompt.
2. Execute the following command:

```
wum init
```

3. You will be asked to give your credentials. Enter your WSO2 credentials.

## Update Repository:

Initializing WUM creates a local directory on your computer called `wum-wso2`. Its path on Linux is `~/.wum-wso2`. This directory contains the Update Repository (`~/.wum-wso2/update` folder), which stores all the subsequent product updates that are done using WUM. As this directory is valid for 30 days only, initialize the WUM tool again after 30 days.

### Step 2: Change the update repository location (Optional)

If required, you can copy the contents of your update repository to a separate folder on your computer, which will allow you more convenient access to its contents. Execute the following command with the new directory location:

```
wum config local.product.repo ~/<new_directory_path>
```

### Step 3: Search for WSO2 products (Optional)

If required, you can do a search to find all the WSO2 products and versions that are supported by WUM. Execute the following command by adding a keyword.

```
wum search <keywords>
```

For example, if you want to search for products with 'esb', replace `<keyword>` with esb.

### Step 4: Add the products that require updates

You need to add a product distribution (ZIP file of the product that should be updated) to the update repository, which is the `~/.wum-wso2/update` folder that was created previously. The product can be added to the update

repository using one of the following methods:

- If you already have a downloaded product pack on your computer, you can specify the path to the distribution as follows:

```
wum add --file <Path_to_product_download>/<product>.zip
```

- If you are not using a product pack that is already on your computer, execute the following command to download a fresh pack to your update repository.

```
wum add <product>
```

For example, replace `<product>` with `wso2esb-4.9.0`.

Note that the product distribution that you specify should be an original ZIP file that has not been changed.

Once the product is added, the directory path to the product pack in the update repository will be published on your terminal.

#### **Step 5: Check for available updates (Optional)**

Before you start updating your product, you can check the updates that are available for the product by executing the following command:

```
wum check-update
```

The above command will check for available updates for all the product packs that you have added to your update repository using WUM.

Alternatively, you can specify the product for which you need to check available updates:

```
wum check-update <product>
```

For example, replace `<product>` with `wso2esb-4.9.0`.

#### **Step 6: Update the products**

Once you have the required product pack added to the update repository, you can execute the following command to start updating:

```
wum update <product>
```

For example, replace `<product>` with `wso2esb-4.9.0`.

Note that your credentials will be verified to check if you have a production support account with WSO2. If you do not have one, you will receive a warning before downloading the updates. Please remember that you are not allowed to use these updates in production if you do not have a production account. See [troubleshooting license warnings](#) for more details.

WUM will then download the updates and create a new product distribution in your update repository. The directory path to the new product pack will be published on your terminal. The details of the updates included in the new product pack will be emailed to you. The same information will also be included in your product pack. See [Verify the updates](#) for information.

## Manage products updated by WUM

You can manage the products in your update repository as explained below.

### Get details of products managed by WUM

You can get the details of products managed by the WUM tool as follows:

- To get the list of WSO2 products that are currently managed by WUM, execute the following command:

```
wum list
```

- To get details of a particular product that is managed by WUM, execute the following command:

```
wum describe <product>
```

For example, replace `<product>` with `wso2esb-4.9.0`.

### Delete WSO2 products managed by WUM

The product versions that are managed in your update repository can be deleted by executing the following command:

```
wum delete <product>
```

For example, replace `<product>` with `wso2esb-4.9.0`.

## Verify the updates

Once your product is updated, you can verify the updates.

- You will receive an email with details of all the updates that are included in the updated product distribution.
- The details of updates included in the product distribution will be included in a PDF inside the new product distribution. The location of the PDF will be as follows:

```
(product-archive)/updates/summary-2016-09-05T15-23-33/update-summary-1472889183206.pdf
```

- If there are configuration changes in the updated product, they will be listed in the `change-log.txt` file inside the new product distribution. The location of the change log file is as follows:

```
(product-archive)/updates/summary-2016-09-05T15-23-33/change-log-1472889183206.txt
```

## What's next?

See [Moving Updates into Production](#) for instructions on how to apply the WUM updates to your production environment.

## How the WSO2 Update Service Works

**This page is work in progress and restricted to internal users.**

WSO2 introduces the [WSO2 Update service](#), which allows you to get the latest updates that are available for a particular product release. These updates include the latest bug fixes and security fixes that are released by WSO2 after a particular product version is released. Therefore, you do not need to wait and upgrade to the next product release to get the latest updates that are available for your product version. You can simply update a product by subscribing to the WSO2 Update service.

Note that the updates you receive through WUM are licensed for free use in non-production environments only. To use updates in production, you are required to have a WSO2 Support Subscription. See [WSO2 Update license terms](#) for more information.

Once you have subscribed to the WSO2 update service, you can use the command-line utility called [WSO2 Update Manager \(WUM\)](#) for downloading updates. WUM works in conjunction with WSO2 Update to provide you with a new product distribution (ZIP file) that will include all the latest bug and security fixes for the product. For example, if you update WSO2ESB 4.9.0 with the latest updates, you will receive a new updated product distribution called `WSO2ESB 4.9.0.0000`, which will include the latest updates for WSO2 ESB. You can run WUM once to update the software to the latest set of released updates, or schedule it to run periodically so you maintain the latest set of updates continuously.

The following diagram depicts how the WSO2 Update service works:

.....  
**WSO2 Update service**

.....  
**WSO2 Update Manager**

.....  
**Local update repository**

.....  
**User authentication**

## Moving Updates into Production

If you have successfully followed the instructions [here](#) to get the latest updates for your product, you will now have a new product pack with all the required updates. The next step is to apply the new product pack in production. That is, you need to move all the configurations and customizations from your previous product pack to the new one. This process can be easily automated using one of the following approaches:

- Use a deployment automation tool
- Develop a script

Note the following:

- If you are upgrading to the updated product pack from a previous version of the product, see the upgrading instructions in the relevant [product documentation](#) for more information on the product version change.
- Third party JARs may have been introduced by enhancements to product features.
- The new updates in your product may have an impact on performance.

### ***Use a deployment automation tool***

You can use a tool such as **Puppet** or **Chef** for this purpose. With this approach, your deployment will be completely managed by the automation tool you use.

### ***Develop a script***

If you are managing the deployment manually without using a deployment automation tool, you can develop a script to get the required configuration changes and customizations, and apply them to the WUM-updated product pack. Before running the script, note that you need to separately maintain your customizations under proper revision control (using a system such as SVN or Git). This allows you to easily rollback to a previous revision if you find errors in the updated pack.

# 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.

File Name	Location
carbon.xml	<PRODUCT_HOME>/repository/conf
registry.xml	
user-mgt.xml	
log4j.properties	
catalina-server.xml	<PRODUCT_HOME>/repository/conf/tomcat
tomcat-users.xml	
web.xml	
context.xml	<PRODUCT_HOME>/repository/conf/tomcat/carbon/META
web.xml	
Owasp.CsrfGuard.Carbon.properties	<PRODUCT_HOME>/repository/conf/security
authenticators.xml	
xss-patterns.properties	
master-datasources.xml	<PRODUCT_HOME>/repository/conf/datasources
axis2.xml	<PRODUCT_HOME>/repository/conf/axis2
axis2_client.xml	
tenant-axis2.xml	

config-validation.xml	<PRODUCT_HOME>/repository/conf/etc
email-admin-config.xml	<PRODUCT_HOME>/repository/conf/email

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.

Click on the table and use the left and right arrow keys to scroll horizontally.

### XML Elements

XML element	Attribute	Description	Data type	Default value
<Server>		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.		
	shutdown	The command string that must be received via a TCP/IP connection to the specified port number, in order to shut down Tomcat.	String	SHUTDOWN
	port	<p>The TCP/IP port number on which this server waits for a shutdown command. Set to -1 to disable the shutdown port.</p> <p>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 .sh and catalina.bat .sh from stopping it gracefully.</p>	Int	8005

<Service>		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.		
	name	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.	String	Catalina
	className	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.	String	org.wso2.carbon.tomca
<Connect or>				
	port	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.	Int	9763
	URIEncoding	This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL.	Int	UTF-8
	compressableMimeType	The value is a comma separated list of MIME types for which HTTP compression may be used.	String	text/html,text/javascript,

	noCompressionUserAgents	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.	String	gozilla, traviata
	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.	Int	2048
	compression	<p>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".</p> <p>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 than 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.</p>	String	on
	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. Most often, this feature is not required.	String	WSO2 Carbon Server

	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.	Int	200
	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.	Int	200
	connectionUploadTimeout	Specifies the timeout, in milliseconds, to use while a data upload is in progress. This only takes effect if disableUploadTimeout is set to false.	Int	120000
	disableUploadTimeout	This flag allows the servlet container to use a different, usually longer connection timeout during data upload.	Boolean	false
	minSpareThreads	The minimum number of threads always kept running.	Int	50
	maxThreads	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.	Int	250
	acceptorThreadCount	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.	Int	2
	maxHttpHeaderSize	The maximum size of the request and response HTTP header, specified in bytes.	Int	8192

	bindOnInit	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.	Boolean	false
	redirectPort	If this Connector is supporting non-SSL requests, and a request is received for which a matching <security-constraint> requires SSL transport, Catalina will automatically redirect the request to the port number specified here.	Int	9443
	protocol	Sets the protocol to handle incoming traffic.	String	org.apache.coyote.http'
	SSLEnabled	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.	Boolean	true
	secure	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.	Boolean	true
	scheme	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.	String	https

	clientAuth	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.	Boolean	false
	enableLookups	Set to true if you want calls to <code>request.getRemoteHost()</code> 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.	Boolean	false
	sslProtocol	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.	String	TLS
	keystoreFile keystorePass	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.		
	name	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.	String	Catalina
	defaultHost	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.	String	localhost
<Realm>		A Realm element represents a "database" 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.		
	className	Java class name of the implementation to use. This class must implement the org.apache.catalina.Realminterface.	String	org.wso2.carbon.tomca

<Host>	<p>The Host element represents a virtual host, which is an association of a network name for a server (such as "<a href="#">www.mycompany.com</a>" 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.</p> <p>In many cases, System Administrators wish to associate more than one network name (such as <a href="#">www.mycompany.com</a> and company.com) with the same virtual host and applications. This can be accomplished using the Host Name Aliases feature discussed below.</p> <p>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.</p> <p>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.</p>		
--------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--	--

	<code>name</code>	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.	String	localhost
	<code>appBase</code>	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.	String	<code> \${carbon.home}/repository</code>
	<code>autoDeploy</code>	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.	Boolean	false
	<code>deployOnStartup</code>	This flag value indicates if web applications from this host should be automatically deployed when Tomcat starts. See Automatic Application Deployment for more information.	Boolean	false
	<code>unpackWARs</code>	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.	Boolean	true
<Valve		The Access Log Valve creates log		

&gt;

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 produced by this Valve are rolled over nightly at midnight. This Valve may be associated with any Catalina container (Context, Host, orEngine), 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 theContext that 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.		
	className	Java class name of the implementation to use.	String	org.wso2.carbon.tomca
	pattern	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.	String	combined
	suffix	The suffix added to the end of each log file name.	String	.log
	prefix	The prefix added to the start of each log file name.	String	http_access_
	directory	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 "logs" (relative to \$CATALINA_BASE).	String	\${carbon.home}/repository
	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.

XML element	Attribute	Description	Data type
<datasources-configuration>	xmlns	The root element. The namespace is specified as: xmlns:svns="http://org.wso2.securevault/configuration"	

<providers>		The container element for the datasource providers.	
<provider>		The datasource provider, which should implement <code>org.wso2.carbon.ndatasource.common.spi.DataSourceReader</code> . The datasources follow a pluggable model in providing datasource type implementations using this approach.	Fully qualified Java class
<datasources>		The container element for the datasources.	
<datasource>		The root element of a datasource.	
<name>		Name of the datasource.	String
<description>		Description of the datasource.	String
<jndiConfig>		The container element that allows you to expose this datasource as a JNDI datasource.	
<name>		The JNDI resource name to which this datasource will be bound.	String
<environment>		<p>The container element in which you specify the following JNDI properties:</p> <ul style="list-style-type: none"> <li>• <code>java.naming.factory.initial</code>: Selects the registry service provider as the initial context.</li> <li>• <code>java.naming.provider.url</code>: Specifies the location of the registry when the registry is being used as the initial context.</li> </ul>	Fully qualified Java class
<definition>	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 "RDBMS" data source reader expects a "configuration" element with the sub-elements listed below.	String
<configuration>		The container element for the RDBMS properties.	
<url>		The connection URL to pass to the JDBC driver to establish the connection.	URL
<username>		The connection user name to pass to the JDBC driver to establish the connection.	String
<password>		The connection password to pass to the JDBC driver to establish the connection.	String
<driverClassName>		The class name of the JDBC driver to use.	Fully qualified Java class
<maxActive>		The maximum number of active connections that can be allocated from this pool at the same time.	Integer

<maxWait>		Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception.	Integer
<testOnBorrow>		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.	Boolean
<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 a SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).	String
<validationInterval>		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).	Long

**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/WSOCARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIME
                        OUT=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**

XML element	Attribute	Description	Data type	Default value
<wso2registry>				

<cacheConfig>		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 <cacheConfig> element in this file is used to configure the time period for which a resource can be cached.		
<lastAccessedExpirationMillis>		Specifies the time period for which a resource in the registry will be cached after it is <b>read</b> by an application.		15
<lastModifiedExpirationMillis>		Specifies the time period for which a resource in the registry will be cached after it is <b>created or modified</b> .		15
<currentDBConfig>		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.	String	wso2registry
<readOnly>		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.	Boolean	false
<enableCache>		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.	Boolean	true

<registryRoot>		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.	String	/
<dbConfig>			String	
	name		String	wso2registry
<dataSource>			String	jdbc/WSO2CarbonDB
<handler>		<p>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.</p> <p>Handler implementations can provide alternative behaviors for basic resource related operations, by overwriting one or more methods in the Handler class.</p>	String	
	class		String	
<filter>			String	
	class		String	
<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.		
	url	The URL of the remote instance.	String	<a href="https://localhost:9443/registry">https://localhost:9443/registry</a>

<ID>		Remote instance ID.	String	instanceid
<username>		Username of the remote registry login.	String	username
<password>		Password of the remote registry login.	String	password
<dbConfig>		The database configuration to use.	String	wso2registry
<readOnly>		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 only for the specific remote instance.	String	false
<enableCache>		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.	String	true
<registryRoot>		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.	String	/
<mount>		Once a remote instance has been defined, a collection on the remote registry can be mounted to the local instance.		
	path	The path to which the mount will be added to.	String	/_system/config
	overwrite	Whether an existing collection at the given path would be overwritten or not.		true false virtual
<instanceID>		Remote instance ID.		instanceid
<targetPath>		The path on the remote registry.	String	/_system/nodes

<versionResourcesOnChange>		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.	Boolean	false
<staticConfiguration>		<p>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.</p> <p>You are supposed to change the static configuration section only before loading any data to the registry (That is before the first start-up).</p>		
<versioningProperties>		Whether the properties are versioned when a snapshot is created.	Boolean	true
<versioningComments>		Whether the comments are versioned when a snapshot is created.	Boolean	true
<versioningTags>		Whether the tags are versioned when a snapshot is created.	Boolean	true
<versioningRatings>		Whether the ratings are versioned when a snapshot is created.	Boolean	true

## 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

XML element	Attribute	Description	Data type	Default value	Mandatory/Optional
<UserManager>		User kernel configuration for Carbon server.			
<Realm>		Realm configuration.			
<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
<AdminRole>		The role name that is used as an admin role for the Carbon server.	String	N/A	Mandatory
<AdminUser>					
> <UserName>		User name that is used to represent an admin user for the Carbon server.	String	N/A	Mandatory
<Password>		Password of the admin user, If the admin user needs to be created in the Carbon server.	String	N/A	Optional
<EveryOneRoleName>		By default, every user in the user store is assigned to this role.	String	N/A	Mandatory
<Property>		User realm configuration specific property values.	String	N/A	Mandatory

r>	<UserStoreManager>	<p>User Store manager implementation classes and their configurations for use realm. Use the <code>ReadOnlyLDAPUserStoreManager</code> to do read-only operations for external LDAP user stores.</p> <p>To do both read and write operations, use the <code>ReadWriteLDAPUserStoreManager</code> for external LDAP user stores.</p> <p>If you wish to use an Active Directory Domain Service (AD DS) or Active Directory Lightweight Directory Service (AD LDS), use the <code>ActiveDirectoryUserStoreManager</code>. This can be used for both read-only and read/write operations.</p> <p>Use <code>JDBCUserStoreManager</code> for both internal and external JDBC user stores.</p>	String	N/A	Mandatory
	class				
	<Property>	User store configuration specific property values. See <a href="#">working with primary user store properties</a> for more information.	String	N/A	Optional
	<AuthorizationManager>	Authorization manager implementation class and its configuration for user realm.	String	N/A	Mandatory
	class				
	<Property>	Authorization manager configuration specific property values.	String	N/A	Optional

## 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. See more details on system requirements for your product on [Installation Prerequisites](#), and the procedure for starting a server in [Running the Product](#).

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.

Parameter	Parameter Description	Parameter Value
CPU	Required processor speed.	800
RAM	Required RAM in your environment.	2048
swap	Required space in hard disk to use for virtual memory.	2048
freeDisk	Free disk space required in your environment.	1024
ulimit	<p>The limit of resources per user. This value indicates the limit on the number of file descriptors a process may have.</p> <p>This property is specified in the product startup script as shown below. For example, see the product startup script for Linux: &lt;PRODUCT_HOME&gt;/bin/wso2server.sh:</p> <pre>#ulimit -n 100000</pre>	4096

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.

Parameter	Description	Parameter Value
initHeapSize	The initial heap size that applies if the JVM requires more memory than is allocated by default.	256
maxHeapSize	The maximum heap size that applies if the JVM requires more memory than is allocated by default.	512
maxPermGenSize	The maximum heap size of the permanent generation of heap.	256

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.

```
$JAVACMD \
-Xms256m -Xmx1024m -XX:MaxPermSize=256m \
```

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.patch 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."

## Database Upgrade Guide

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.
  1. To migrate the super tenant settings, copy the <PRODUCT\_HOME>/repository/deployment/server directory.
  2. 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:

```
./wso2server.sh -Dsetup (In Linux)
wso2server.bat -Dsetup (In Windows)
```

Listed below are some general options that can be used for starting the server.

Startup Option	Description
-start	Starts the Carbon server using "nohup" in the background. This option is not available for Windows.
-stop	Stops the Carbon server process. This option is not available for Windows.
-restart	Restarts the Carbon server process. This option is not available for windows.
-cleanRegistry	Cleans the registry space. <b>Caution:</b> All registry data will be lost.
-debug <port>	Starts the server in remote debugging mode. The remote debugging port should be specified.
-version	Shows the version of the product that you are running.
-help	Lists all the available commands and system properties.

Listed below are some system properties that can be used when starting the server.

Startup Options	Description
-DosgiConsole=[port]	Starts the Carbon server with the Equinox OSGi console. If the optional 'port' parameter is provided, a telnet port will be opened.
-DosgiDebugOptions=[options-file]	Starts the Carbon server with OSGi debugging enabled. Debug options are loaded from the <PRODUCT_HOME>/repository/conf/etc/osgi-debug.options.
-Dsetup	Cleans the registry and other configurations, recreates DB, re-populates the configuration and starts the server. <b>Note:</b> It is not recommended to use this option in a production environment. Instead, you can manually run the DB scripts directly in the database.
-DworkerNode	<p>Starts the product as a worker node, which means the front-end features of your product will not be enabled.</p> <p>Note that from Carbon 4.4.1 onwards, you can also start the worker profile by setting the following system property to 'true' in the product startup script before the script is executed.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>-DworkerNode=false</p> </div>
-DserverRoles=<roles>	A comma separated list of roles used in deploying Carbon applications.
-Dprofile=<profileName>	Starts the server with the specified profile, e.g., worker profile.
-Dtenant.idle.time=<time>	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.

## 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

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\_ECDH\_ECDSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_ECDH\_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:

- TLS\_ECDHE\_ECDSA\_WITH\_RC4\_128\_SHA
- TLS\_ECDHE\_RSA\_WITH\_RC4\_128\_SHA
- SSL\_RSA\_WITH\_RC4\_128\_SHA
- TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA
- TLS\_ECDH\_RSA\_WITH\_RC4\_128\_SHA
- SSL\_RSA\_WITH\_RC4\_128\_MD5

## 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.

<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.

Folder	Description	General Folder Path
bin	Contains various scripts (.sh & .bat scripts).	<PRODUCT_HOME>/bin/
database	Contains the databases.	<PRODUCT_HOME>/repository/database/
dbscripts	Contains all the database scripts.	<PRODUCT_HOME>/dbscripts/
lib	Contains the basic set of libraries required for starting a WSO2 product in standalone mode.	<PRODUCT_HOME>/lib/
repository	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).	<PRODUCT_HOME>/repository/
conf	Contains configuration files.	<PRODUCT_HOME>/repository/conf/
components	Contains different components (OSGI bundles, features etc.) that are related to the product.	<PRODUCT_HOME>/repository/components/

plugins	Contains plugins that are related to the product.	<PRODUCT_HOME>/repository/components/plugins/	◀
patches	Contains patches that are issues with the product.	<PRODUCT_HOME>/patches/	◀
logs	Contains all log files created during execution.	<PRODUCT_HOME>/repository/logs/	◀
resources	Contains additional resources that may be required, including sample configurations and sample resources.	<PRODUCT_HOME>/resources/	◀
samples	Contains sample services and client applications to demonstrate the functionality and capabilities of WSO2 products.	<PRODUCT_HOME>/samples/	◀
tmp	Used for storing temporary files, and is pointed to by the <code>java.io.tmpdir</code> system property.	<PRODUCT_HOME>/tmp/	◀
LICENSE.txt	Apache License 2.0 and other relevant licenses under which the WSO2 product is distributed.	<PRODUCT_HOME>/LICENSE.txt	▶
README.txt	This document.	<PRODUCT_HOME>/README.txt	▶
release-notes.html	Release information for the WSO2 product.	<PRODUCT_HOME>/release-notes.html	▶
wso2	Contains the directories of other profiles that are shipped as separate runtimes.	<PRODUCT_HOME>/wso2/	▶

## WSO2 Patch Application Process

This page is currently getting updated!

WSO2 has introduced the [WSO2 Update Manager \(WUM\)](#), 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.

The patch application process described below guides you on how to manually apply [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
  - Verify the components in the patch directory of the patch archive
  - Verify the artifacts in the resources directory of the patch archive
- 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 the above 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 necessary guidelines on how to apply the patch. Some patches (as in the example given above) might have a resources directory that contains artifacts, such as web apps, library files, configurations, scripts and more.

#### Example:

```
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 when applying patches.
- 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 that are going to be replaced by the patch. The original files might be required if you are [reverting the patch](#).
- As a precaution, make a backup of the server.

You can apply patches to your product in two ways:

- Apply each patch individually.
- If you want to apply multiple patches, create a collective patch and apply it to the product. A collective patch is a single patch that includes all the JAR files from the individual patches that should be applied.

Follow the steps given below to apply an individual patch or a collective patch to the product:

1. Shut down the server.
2. Copy the patches (patchxxx directory in the patch directory) to the <PRODUCT\_HOME>/repository/components/patches directory.
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 be replacing the already existing artifacts.
4. Start the Carbon server. The patches will then be incrementally applied to the plugins directory.

Before applying any patches, 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 version, 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 directories 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.

- All patch related logs are recorded in the <PRODUCT\_HOME>/repository/logs/patches.logfile.
- 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 directories. If you change the data in this file, you will get a startup error when applying patches.

### **Verify the components in the patch directory of the patch archive**

Check and compare the md5sum of each component in the patchXXXX directory against the same component in the <PRODUCT\_HOME>/repository/components/plugins directory. Note that if a component is already patched by a newer patch version, the md5sum of the component in <PRODUCT\_HOME>/repository/components/plugins directory should be the same as the component in latest patchXXXX.

### **Verify the artifacts in the resources directory of the patch archive**

The artifacts that got replaced by the latest patch version must contain the same md5sum as the artifact in the resource directory of the patch.

### **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. Further, you must also identify if there are manual steps that you need to roll back for the patch removal to be effective.

To remove a patch:

1. Remove the patch from the `<PRODUCT_HOME>/repository/components/patches` directory.
2. If the patch has a resources directory, you need to replace the artifact from the backup that was created during the patch application as explained [here](#). Also if there is an exploded artifact in the resources directory (such as `*.war` files in the `<PRODUCT_HOME>/repository/deployment/server/webapps` directory), you need to delete those exploded artifacts. This is only required if the artifact is not already replaced by a newer patch version.
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.