# Table of Contents

1. WSO2 Storage Server Documentation ................................................................. 5  
   1.1 About Storage Server ....................................................................................... 5  
      1.1.1 Introduction to Storage Server .............................................................. 6  
      1.1.2 Features ................................................................................................. 7  
      1.1.3 Architecture ......................................................................................... 8  
      1.1.4 About this Release ................................................................................. 9  
   1.2 Getting Started ............................................................................................... 9  
      1.2.1 Downloading the Product ..................................................................... 10  
      1.2.2 Building from Source ....................................................................... 10  
      1.2.3 Installing the Product ....................................................................... 11  
         1.2.3.1 Installation Prerequisites ............................................................. 12  
         1.2.3.2 Installing on Linux or OS X ......................................................... 15  
         1.2.3.3 Installing on Solaris ...................................................................... 17  
         1.2.3.4 Installing on Windows ................................................................. 18  
         1.2.3.5 Installing as a Windows Service ................................................. 22  
         1.2.3.6 Installing as a Linux Service ....................................................... 29  
      1.2.4 Running the Product ............................................................................ 31  
      1.2.5 Upgrading from a Previous Release .................................................... 35  
      1.2.6 Get Involved ....................................................................................... 36  
         1.2.6.1 WSO2 GitHub Repositories ......................................................... 39  
   1.3 User Guide ..................................................................................................... 41  
      1.3.1 Relational Store Service ..................................................................... 42  
         1.3.1.1 Configuring Environments for RDBMS ........................................ 42  
            1.3.1.1.1 Configuring the RSS Metadata Repository ............................. 43  
            1.3.1.1.2 Configuring RSS Environments ......................................... 46  
            1.3.1.1.3 Adding RSS Instances to Environment ............................... 51  
         1.3.1.2 Provisioning an RDBMS .............................................................. 53  
            1.3.1.2.1 Creating Databases ................................................................. 53  
            1.3.1.2.2 Creating Database Users ...................................................... 56  
            1.3.1.2.3 Creating Privilege Templates ................................................. 57  
      1.3.1.3 Monitoring RSS Statistics using WSO2 BAM ................................. 59  
      1.3.1.4 RSS CLI Commands Guide ............................................................... 61  
      1.3.1.5 Using Created Databases/Database Users in Applications ............. 61  
   1.3.2 Cassandra Columnar Store Service ....................................................... 62  
      1.3.2.1 Configuring Cassandra Environments .......................................... 63  
         1.3.2.1.1 Configuring Default Cassandra Environment .......................... 63  
         1.3.2.1.2 Configuring Multiple Cassandra Environments .................... 67  
      1.3.2.2 Enabling Permissions for Cassandra .............................................. 73  
      1.3.2.3 Creating Keyspaces ...................................................................... 75  
      1.3.2.4 Creating Column Families ............................................................ 78  
      1.3.2.5 Explore Cassandra Cluster ............................................................ 80  
      1.3.2.6 Cassandra CLI/CQL Commands .................................................... 81  
      1.3.2.7 Monitor Cassandra Cluster ............................................................ 83  
         1.3.2.7.1 Cluster Node Operations ......................................................... 84  
         1.3.2.7.2 Monitoring Cluster Statistics ............................................... 86  
         1.3.2.7.3 Monitoring Cassandra Statistics using WSO2 BAM .............. 94  
   1.3.3 Distributed File System Service ............................................................... 97
1.4.6 Logging Management ................................................................. 255
  1.4.6.1 Introduction to Logging ...................................................... 256
  1.4.6.2 Logging Configuration ....................................................... 256
1.4.7 Monitoring the Server ............................................................... 260
  1.4.7.1 System Logs ........................................................................ 260
  1.4.7.2 Application Logs ................................................................. 261
1.4.8 Configuring Multiple Tenants ..................................................... 262
  1.4.8.1 Managing Tenants .............................................................. 262
  1.4.8.2 Multi Tenant Architecture ................................................ 263
  1.4.8.3 Tenant-aware Load Balancing Using the WSO2 Elastic Load Balancer ................................................................. 266
1.4.9 Performance Tuning ................................................................. 268
1.4.10 Calling Admin Service from Apps ............................................. 272
1.4.11 Customizing the Management Console ..................................... 276
1.4.12 Service Principal Management ............................................... 279
1.4.13 Cassandra Migration Guide .................................................... 280
1.4.14 WSO2 Patch Application Process ........................................... 284
1.5 Reference Guide ............................................................................. 286
  1.5.1 Configuration Files .................................................................. 287
    1.5.1.1 Configuring axis2.xml ....................................................... 287
    1.5.1.2 Configuring carbon.xml ................................................... 299
    1.5.1.3 Configuring catalina-server.xml ....................................... 305
    1.5.1.4 Configuring config-validation.xml .................................... 316
    1.5.1.5 Configuring identity.xml .................................................. 319
    1.5.1.6 Configuring master-datasources.xml ................................. 323
    1.5.1.7 Configuring registry.xml .................................................. 326
    1.5.1.8 Configuring user-mgt.xml ................................................. 328
  1.5.2 Default Ports of WSO2 Products ............................................. 330
1.6 Getting Support .............................................................................. 334
1.7 Site Map ......................................................................................... 335
WSO2 Storage Server Documentation

Welcome to the WSO2 Storage Server documentation! The WSO2 Storage Server (SS) is an open source, multi-tenanted server for system administrators and developers to easily create and manage relational databases, column stores and HDFS file systems. It conveniently delivers a multi-tenant structured and unstructured data storage to development teams to rapidly provision secure and scalable repositories using a consistent management process.

This documentation provides information for setting up, configuring and implementing WSO2 Storage Server. For a comprehensive, end-to-end coverage of the entire product, its important subject areas, and advanced configurations, we recommend that you follow the defined order of topics.

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.

<table>
<thead>
<tr>
<th>About Storage Server</th>
<th>Getting Started</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduces WSO2 Storage Server, including the business cases it solves, its features, architecture and how to get help or get involved.</td>
<td>Instructions to download, install, run and get started quickly with WSO2 Storage Server.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Guide</th>
<th>Admin Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explores in depth the features and functionality of the Storage Server, solution development, debugging and deployment.</td>
<td>Explores various product deployment scenarios and other topics for system administrators.</td>
</tr>
</tbody>
</table>
About Storage Server

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

- Introduction to Storage Server
- Features
- Architecture
- About this Release

Introduction to Storage Server

Consider a typical scenario of setting up an application development environment which needs to access a relational database. With respect to handling the data side of things, the basic steps that need to be done are:

- Create a database
- Create permission templates
- Create users with permission templates
- Provide access to the database for different users with different permissions

The above steps can be done manually using RDBMS tools. However, it is rare that one would simply set up a development environment for just one application with one type of data server. As various applications come through, the data requirements would differ. Also it would be rare that all applications created can be satisfied with a relational database - there may be instances where a unstructured data storage mechanism or a distributed file system is needed.

Whatever the data requirement is, the basic steps that need to be done remain the same, but the tools used may differ depending on the data storage mechanism used. Thus arises the requirement for an abstraction which would provide such services, providing the user with a consistent interface to use while also performing the underlying actions with the required data storage system. WSO2's Storage Server is the answer to this.

The WSO2 Storage Server (SS) is a multi-tenanted platform for system administrators and developers to easily provision and manage Relational Database Management Systems (RDBMS), column stores and Hadoop Distributed File Systems (HDFS). It delivers a multi-tenanted structured and unstructured data storage for developers who wish to obtain multi-tenanted storage for their applications, etc. You can rapidly provision and access secure and scalable relational, NoSQL Columnar, and Hadoop Distributed File System (HDFS) repositories using a consistent management process. WSO2 Storage Server's management console UI provides you with the ability to create databases, add users, and provision access based on the Web application’s database policies with ease. The Storage Server's management console also provides a consistent management process for provisioning and managing relational, NoSQL Columnar and Hadoop distributed file system repositories.

WSO2 Storage Server is cloud-aware and supports elastic scalability, tenant isolation, on-demand self-service provisioning, and consumption based pricing models. The server publishes usage data to WSO2 Business Activity Monitor and enables advanced analytics, monitoring, and billing.

The WSO2 Storage Server has three main components as follows:

- Column Storage Service
- Relational Storage Service
- Hadoop Distributed File System

Column Storage Service

Column Storage Service is a Cassandra-based database, which is shipped with three components named Cassandra Manager, Viewer and the Cluster Monitor. These three components facilitate the creation, management
and monitoring of a Cassandra cluster through a simplified management console UI.

The Column Store Service is plugged to WSO2 carbon authentication system and Keyspaces users are authenticated and authorized against pluggable WSO2 carbon user base. A column storage cluster can be accessed via any public API like Thrift or Hector.

See Cassandra Columnar Store Service.

Relational Storage Service

Relational Storage Service (RSS) is a management interface to a relational database cluster. RSS allows users to create and manage relational databases in a horizontally-scalable manner. RSS supports H2 and MySQL by default and is extensible.

See Relational Store Service.

Hadoop Distributed File System

Hadoop Distributed File System (HDFS) facilitates users to create cluster file systems across nodes. HDFS can be used to store large files and it is horizontally-scalable.

See Distributed File System Service.

The Storage Server is an on-going project. It undergoes continuous improvements and enhancements with each new release to address new business challenges and customer expectations.

Features

WSO2 Storage Server has the key features listed below. For the relevant versions of the applications used as features, see Compatibility of WSO2 Products.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| Storage technology agnostic provisioning     | • Wide range of storage systems supported:  
• NoSQL stores: Cassandra
• Unstructured stores (file systems): HDFS
• Relational stores: MySQL, SQL Server, PostgreSQL and Oracle
• Operations Supported:  
• Manage databases
• Manage users/permissions
• Choose from default set of layouts or create your own personalized layout
• Policy driven provisioning (Coming Soon)
• Based on QoS/SLA parameters |
| Complete Storage as a Service                | • Monitor storage capacity and bandwidth (Coming soon)  
• Predefined usage policies for billing (Coming soon)  
• Horizontal scaling with elasticity (using S2 DB cartridges) (Coming soon)  
• Database migration to larger clusters  
• Setting up of database clusters for vertical scaling |
Complete application lifecycle support

- Database initialization
- Data migration
- Data backup and recovery

Architecture

The diagram below depicts an overview of the WSO2 Storage Server architecture.

WSO2 Storage Server, which resides on top of WSO2 Carbon, supports provisioning of relational databases, NoSQL databases and File Systems. It provides APIs for provisioning. Those APIs communicate with IaaS layer to get the provisioning operations done. In addition to provisioning APIs, Storage Server exposes tools to monitor the status and statistics of provisioned database, etc. Administrators are also facilitated to configure role-based access control for provisioning databases. In addition to those functionalities, Storage Server provides tools to explore each database provisioned. Once the databases are provisioned, users can access them directly without going through the Storage Server.

When considering an example where an application uses a relational data store, the following explains how WSO2 Storage Server works internally.

- Storage Server needs to be setup with the database server instance which acts as the container for the provisioned databases. A cluster of such containers is provided to the Storage Server configuration documents.
- Based on a round robin algorithm, databases are created in each database server instance cluster.
- The Storage Server has its own central database (RSS management database) which will contain all details pertaining to databases created, users created, users allocated and permissions created and granted. It is through this central database that all management activities are carried out.
• If an application needs to have separate environments (e.g., development and production) the Storage Server will enable provisioning environments which are logical boundaries separating database server instance cluster sets.

• Once the databases have been created, the privileges created, and the users created and allocated to the databases with their respective privileges, tables can be created using the management console and the created databases can be viewed.

Thereafter applications can access these databases.

About this Release

What's new in this release

WSO2 Storage Server version 1.5.0 is the successor of version 1.1.0. It contains the following new features and enhancements:

• Possibility for tenant admins to add database server instances (RDBMS) to be provisions via the management console.
• Possibility to manage multiple Cassandra environments with multiple clusters.
• Role-based permission model for separate RDBMS provisioning environments.
• Role-based permission model for separate Cassandra environments.
• Possibility to enable CQL and Hector datasource readers. This allows Cassandra to be accessed through Hector-based or CQL-based clients.

Fixed issues

To explore the fixed issues go to:

• WSO2 Storage Server 1.5.0 - Fixed issues in SS.

Known issues

For the list of known issues go to:

• WSO2 Storage Server 1.5.0 - Known issues in SS.

Compatible WSO2 product versions

WSO2 Storage Server 1.5.0 is based on WSO2 Carbon 4.3.0 and is expected to be compatible with any other WSO2 product that is based on Carbon 4.3.0. If you get any compatibility issues, please contact team WSO2. For information on the third-party software required with SS 1.5.0, see Installation Prerequisites.
Getting Started

The following topics show how to download, install, run and get started quickly with WSO2 Storage Server.

- Downloading the Product
- Building from Source
- Installing the Product
- Running the Product
- Upgrading from a Previous Release
- Get Involved

Downloading the Product

Follow the instructions below to download the product. You can also download and build the source code.

1. In your Web browser, go to http://wso2.com/products/storage-server/.
2. Click the Download button in the upper right-hand corner of the page to download the latest version. To download an older version, click the Previous Releases link and then select the version that you want.
3. Enter the required details in the form, and click Download.

The binary distribution contains the binary files for both MS Windows and Linux-based operating systems, compressed into a single ZIP file. This distribution is recommended for many users.

After downloading the binary distribution, go to Installation Prerequisites for instructions on installing the necessary supporting applications.

Building from Source

WSO2 invites you to contribute by downloading the source code from the GitHub source control system, building the product and making changes, and then committing your changes back to the source repository. The following sections describe this process:

- Downloading the source
- Editing the source code
- Building the product
- Committing your changes

Building from source is optional. Users who do not want to make changes to the source code can simply download the binary distribution of the product and install it.

Downloading the source

WSO2 products are built on top of WSO2 Carbon Kernel, which contains the Kernel libraries used by all products. When there are changes in the Carbon Kernel, they are bundled and released in a new WSO2 Carbon version (for example, WSO2 Carbon 4.3.0). You can download the complete WSO2 Kernel release using the following repository: https://github.com/wso2/carbon4-kernel, which is recommended if you intend to modify the source. After downloading the source of the Carbon Kernel, execute the following command to download the source of the product:

```
git clone https://github.com/wso2/product-ss
```

After the source code is downloaded, you can start editing. However, it is recommended to run a build prior
Editing the source code

Now that you have downloaded the source code for the Carbon project from GitHub, you can prepare your development environment and do the required changes to the code.

1. To edit the source code in your IDE, set up your development environment by running one of the following commands:

<table>
<thead>
<tr>
<th>IDE</th>
<th>Command</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntelliJ IDEA</td>
<td>mvn idea:idea</td>
<td><a href="http://maven.apache.org/plugins/maven-idea-plugin">http://maven.apache.org/plugins/maven-idea-plugin</a></td>
</tr>
</tbody>
</table>

2. Add the required changes to the source code.

Building the product

Ensure that the following prerequisites are in place before you build:

1. Make sure the build server has an active Internet connection to download dependencies while building.
2. Install Maven and JDK. For compatible versions, see Installation Prerequisites.
3. Set the environment variable MAVEN_OPTS="-Xms1024m -Xmx4096m -XX:MaxPermSize=1024m" to avoid the Maven OutOfMemoryError.

Use the following Maven commands to build your product:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvn clean install</td>
<td>The binary and source distributions.</td>
</tr>
<tr>
<td>mvn clean install -Dmaven.test.skip=true</td>
<td>The binary and source distributions, without running any of the unit tests.</td>
</tr>
<tr>
<td>mvn clean install -Dmaven.test.skip=true -o</td>
<td>The binary and source distributions, without running any of the unit tests, in offline mode. This can be done only if you have already built the source at least once.</td>
</tr>
</tbody>
</table>

Committing your changes

You can contribute to WSO2 products by committing your changes to GitHub. Whether you are a committer or a non-committer, you can contribute with your code as explained in the Get Involved section.

Installing the Product

Installing WSO2 is very fast and easy. Before you begin, be sure you have met the installation prerequisites, and then follow the installation instructions for your platform.

- Installation Prerequisites
- Installing on Linux or OS X
- Installing on Solaris
- Installing on Windows
- Installing as a Windows Service
Installing as a Linux Service

Installation Prerequisites

Prior to installing any WSO2 Carbon based product, it is necessary to have the appropriate prerequisite software installed on your system. Verify that the computer has the supported operating system and development platforms before starting the installation.

System requirements

| Memory | ~ 2 GB minimum  
| ~ 512 MB heap size. This is generally sufficient to process typical SOAP messages but the requirements vary with larger message sizes and the number of messages processed concurrently. |
| Disk | ~ 1 GB, excluding space allocated for log files and databases. |

Environment compatibility

Operating Systems / Databases

| Operating Systems / Databases | All WSO2 Carbon-based products are Java applications that can be run on any platform that is Oracle JDK 1.6.*/1.7.* compliant. JDK 1.8 is not supported yet. Also, we do not recommend or support OpenJDK.  
| All WSO2 Carbon-based products are generally compatible with most common DBMSs. For more information, see Working with Databases.  
| For environments that WSO2 products are tested with, see Compatibility of WSO2 Products. |

Required applications

The following applications are required for running the product and its samples or for building from the source code. For details on installing these applications, see the installation section for Linux, Solaris, or Windows. Mandatory installs are marked with *.

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Version</th>
<th>Download</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Oracle Java SE Development Kit (JDK)</strong></th>
<th><strong>WSO2 Storage Server, version 1.5.0</strong></th>
<th><strong><a href="http://java.sun.com">http://java.sun.com</a></strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• To launch the product as each product is a Java application.</td>
<td>1.6.24 or later / 1.7.*</td>
<td></td>
</tr>
<tr>
<td>• To build the product from the source distribution (both JDK and Apache Maven are required).</td>
<td>If you are using JDK 1.6, you might need to install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files to avoid &quot;illegal key size&quot; errors when you try to invoke a secured Web service.</td>
<td></td>
</tr>
<tr>
<td>• To run Apache Ant.</td>
<td>To build the product from the source distribution, you must use JDK 1.6 instead of JDK 1.7.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oracle and IBM JRE 1.7 are also supported when running (not building) WSO2 products.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If you are using JDK 1.7 on a Mac OS, install the snappy-java library using the following steps:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Download the snappy-java JAR and extract it to a preferred location. This folder will be referred to as <code>&lt;SNAP PY_HOME&gt;</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Copy the appropriate snappy-java library file <code>i386.jnilib</code> (32bit) or <code>x86_64.jnilib</code> (64bit), which is in the <code>&lt;SN APPY_HOME&gt;/org/xerial/snappy/native/Mac/dire ctory</code>, to the <code>&lt;SS_HOME&gt;</code> directory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information on installing snappy-java library, see Snap py-java fails on Mac OS JDK 1.7.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>We do not recommend OpenJDK.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Apache ActiveMQ JMS Provider</strong></th>
<th><strong><a href="http://activemq.apache.org">http://activemq.apache.org</a></strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• To enable the product's JMS transport and try out JMS samples if any. The ActiveMQ client libraries must be installed in the product's classpath before you can enable the JMS transport.</td>
<td>5.5.0 or later</td>
</tr>
<tr>
<td></td>
<td>If you use any other JMS provider (e.g., Apache Qpid), install any necessary libraries and/or components.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Apache Ant</strong></th>
<th><strong><a href="http://ant.apache.org">http://ant.apache.org</a></strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• To compile and run the product samples.</td>
<td>1.7.0 or later</td>
</tr>
<tr>
<td>SVN Client</td>
<td>Apache Maven</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>• To check out the code to <strong>build the product from the source distribution</strong>.</td>
<td>• To <strong>build the product from the source distribution</strong> (both JDK and Apache Maven are required). If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do <strong>not</strong> need to install Maven.</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linux</strong></td>
<td><strong><a href="http://subversion.apache.org/packages.html">http://subversion.apache.org/packages.html</a></strong></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td><strong><a href="http://tortoisesvn.net/downloads.html">http://tortoisesvn.net/downloads.html</a></strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>3.0.x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong><a href="http://maven.org">http://maven.org</a></strong></td>
</tr>
<tr>
<td>Web Browser</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>• To access the product's Management Console. The Web Browser must be JavaScript enabled to take full advantage of the Management console.</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** On Windows Server 2003, you must not go below the medium security level in Internet Explorer 6.x.

---

**Installing on Linux or OS X**

**Before you begin,** please see our compatibility matrix to find out if this version of the product is fully tested on Linux or OS X.

Follow the instructions below to install the required applications and WSO2 SS on Linux or Mac OS X.

**Install the required applications**

1. Establish an SSH connection to the Linux machine or log in on the text Linux console.
2. Be sure your system meets the **Installation Prerequisites**. Java Development Kit (JDK) is essential to run the product.

**Installing the product**

1. If you have not done so already, download the latest version of SS as described in Downloading the Product.
2. Extract the archive file to a dedicated directory for the product, which will hereafter be referred to as `<SS_HOME>`.

**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 in your favourite 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/jdk1.6.0_25 with the actual directory where the JDK is installed.

   ```bash
   export JAVA_HOME=/usr/java/jdk1.6.0_25
   export PATH=${JAVA_HOME}/bin:${PATH}
   ```

   The file should now look like this:

   ```bash
   #.bashrc
   # Source global definitions
   if [-f /etc/bashrc ]; then
     . /etc/bashrc
   fi
   # User specific aliases and functions
   export JAVA_HOME=/usr/java/jdk1.6.0_25
   export PATH=${JAVA_HOME}/bin:${PATH}
   export M2_HOME=/opt/apache-maven-3.6.3
   export PATH=${M2_HOME}/bin:${PATH}
   ```

3. Save the file.

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

   ```bash
   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:

   ```bash
   echo $JAVA_HOME
   ```

   The system returns the JDK installation path.

**Setting system properties**

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

- **Set the properties from a script**
  Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
• **Set the properties from an external registry**
  If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as, username and password to connect to the registry in a properties file, instead of in the Java code and secure the properties file with the secure vault. Read more about secure vault in the section on **working with security**.

---

**SUSE Linux**

When using SUSE Linux, it ignores `/etc/resolv.conf` and only looks at the `/etc/hosts` file. This means that the server will throw an exception on startup if you have not specified anything besides localhost. To avoid this error, add the following line above 127.0.0.1 localhost in the `/etc/hosts` file:

```
<ip_address> <machine_name> localhost.
```

You are now ready to run the product.

**Installing on Solaris**

*Before you begin, please see our compatibility matrix* to find out if this version of the product is fully tested on Solaris.

Follow the instructions below to install the required applications and the product on Solaris.

**Installing the supporting applications**

1. Establish an SSH connection to the Solaris machine or log in on the text console.
2. Be sure your system meets the **Installation Prerequisites**, and then install the following applications:
   1. Install Java Development Kit (JDK)
   2. Install Apache Maven (required only if you want to build the product from the source code or run samples)

**Installing the product**

1. If you have not done so already, download the latest version of the product as described in **Downloading the Product**.
2. Extract the archive file to a dedicated directory for the product, which will hereafter be referred to as `<PRODICT_HOME>`.

**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 in your favorite text editor, such as vi, emacs, pico, or mcedit.
2. Assuming you have JDK 1.6.0_25 installed, add the following two lines at the bottom of the file, replacing `/usr/j ava/jdk1.6.0_25` with the actual directory where the JDK is installed.

```bash
export JAVA_HOME=/usr/java/jdk1.6.0_25
export PATH=${JAVA_HOME}/bin:${PATH}
```

The file should now look like this:

```bash
# .bashrc
# Source global definitions
if [-f /etc/bashrc ]; then
  . /etc/bashrc
fi
# User specific aliases and functions
export JAVA_HOME=/usr/java/jdk1.6.0_25
export PATH=${JAVA_HOME}/bin:${PATH}
export M2_HOME=/opt/apache-maven-3.8.3
export PATH=${M2_HOME}/bin:${PATH}
```

3. Save the file.

4. To verify that the `JAVA_HOME` variable is set correctly, execute the following command: `echo $JAVA_HOME`

The system returns the JDK installation path.

**Setting system properties**

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

- **Set the properties from a script.** Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.

- **Set the properties from an external registry.** If you want to access properties from an external registry, you could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the `secure vault`.

You are now ready to run the product.
Installing on Windows

Before you begin, please see our compatibility matrix to find out if this version of the product is fully tested on Windows.

Follow the instructions below to install the required applications and the product on Windows.

Installing the supporting applications

- Be sure your system meets the Installation Prerequisites, and then install the following applications:
  1. Install Java Development Kit (JDK)
  2. Install Apache Maven (required only if you want to build the product from the source code and run samples)

- Be sure that the PATH environment variable is set to "C:\Windows\System32", because the findstr windows exe is stored in this path.

Installing the product

1. If you have not done so already, download the latest version of the product as described in Downloading the Product.
2. Extract the archive file to a dedicated directory for the product, which will hereafter be referred to as <PROD_HOME>.

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. Typically, the JDK is installed in a directory under C:/Program Files/Java. For example, C:/Program Files/Java/jdk1.6.0_27. If you have multiple versions installed, choose the latest one that you find after sorting by date.

Environment variables are global system variables accessible by all the processes running under the operating system. You can define an environment variable as a system variable, which applies to all users, or as a user variable, which applies only to the user who is currently logged in.

You can set JAVA_HOME using the System Properties, as described below. Alternatively, if you just want to set JAVA_HOME temporarily in the current command prompt window, set it at the command prompt.

Setting JAVA_HOME using the System Properties

1. Right-click the "My Computer" icon on the desktop and choose Properties.
2. In the System Properties window, click the Advanced tab, and then click the Environment Variables button.

3. Click the New button under “System variables” (for all users) or under “User variables” (just for the user who is currently logged in).
4. Enter the following information:

- In the "Variable name" field, enter: JAVA_HOME
- In the "Variable value" field, enter the installation path of the Java Development Kit, such as: c:\Program Files\Java jdk1.6.0_27

5. Click OK.

The JAVA_HOME variable is now set and will apply to any subsequent command prompt windows you open. If you have existing command prompt windows running, you must close and reopen them for the JAVA_HOME variable to take effect, or manually set the JAVA_HOME variable in those command prompt windows as described in the next section. To verify that the JAVA_HOME variable is set correctly, open a command window (from the Start menu, click Run, and then type CMD and click Enter) and execute the following command:

`set JAVA_HOME`

The system returns the JDK installation path.

**Setting JAVA_HOME temporarily using the Windows command prompt (CMD)**

You can temporarily set the JAVA_HOME environment variable within a Windows command prompt window (CMD). This is useful when you have an existing command prompt window running and you do not want to restart it.

1. In the command prompt window, enter the following command where <JDK_INSTALLATION_PATH> is the JDK installation directory and press Enter: `set JAVA_HOME=<JDK_INSTALLATION_PATH>`

For example:

`set JAVA_HOME=c:\Program Files\java\jdk1.6.0_27`

The JAVA_HOME variable is now set for the current CMD session only.

2. To verify that the JAVA_HOME variable is set correctly, execute the following command:

`set JAVA_HOME`

The system returns the JDK installation path.

**Setting system properties**

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

- **Set the properties from a script.** Setting your system properties in the startup script is ideal, because it ensures that you set the properties every time you start the server. To avoid having to modify the script each time you upgrade, the best approach is to create your own startup script that wraps the WSO2 startup script and adds the properties you want to set, rather than editing the WSO2 startup script directly.
- **Set the properties from an external registry.** If you want to access properties from an external registry, you
could create Java code that reads the properties at runtime from that registry. Be sure to store sensitive data such as username and password to connect to the registry in a properties file instead of in the Java code and secure the properties file with the secure vault.

You are now ready to run the product.

Installing as a Windows Service

WSO2 Carbon and any Carbon-based product can be run as a Windows service as described in the following sections:

- Prerequisites
- Setting up the YAJSW wrapper configuration file
- Setting up CARBON_HOME
- Running the product in console mode
- Working with the WSO2CARBON service

Prerequisites

- Install JDK and set up the JAVA_HOME environment variable.
- 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) version 11.03, and several WSO2 products provide a default wrapper.conf file in their <PRODUCT_HOME>/bin/yajsw/ directory. The instructions below describe how to set up this file.

Setting up the YAJSW wrapper configuration file

The configuration file used for wrapping Java Applications by YAJSW is wrapper.conf, which is located in the <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 <YAJSW_HOME>/conf/ directory.

If you want to set additional properties from an external registry at runtime, store sensitive information like usernames and passwords for connecting to the registry in a properties file and secure it with secure vault.

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 sign into the product management console.

```
# working directory
wrapper.working.dir=${carbon_home}
```

Minimal wrapper.conf configuration
# Java Main class.
# YAJSW: default is "org.rzo.yajsw.app.WrapperJVMMain"
# DO NOT SET THIS PROPERTY UNLESS YOU HAVE YOUR OWN IMPLEMENTATION
# wrapper.java.mainclass=
#*****************************************************************************
# tmp folder
# yajsw creates temporary files named in_.. out_.. err_.. jna..
# per default these are placed in jna.tmpdir.
# jna.tmpdir is set in setenv batch file to <yajsw>/tmp
#*****************************************************************************
wrapper.tmp.path = ${jna_tmpdir}
#*****************************************************************************
# Application main class or native executable
# One of the following properties MUST be defined
#*****************************************************************************
# Java Application main class
wrapper.java.app.mainclass=org.wso2.carbon.bootstrap.Bootstrap
# Log Level for console output. (See docs for log levels)
wrapper.console.loglevel=INFO
# Log file to use for wrapper output logging.
wrapper.logfile=${wrapper_home}\log\wrapper.log
# Format of output for the log file. (See docs for formats)
#wrapper.logfile.format=LPTM
# Log Level for log file output. (See docs for log levels)
#wrapper.logfile.loglevel=INFO
# Maximum size that the log file will be allowed to grow to before
# the log is rolled. Size is specified in bytes. The default value
# of 0, disables log rolling by size. May abbreviate with the 'k' (kB) or
# 'm' (mB) suffix. For example: 10m = 10 megabytes.
# If wrapper.logfile does not contain the string ROLLNUM it will be
# automatically added as suffix of the file name
wrapper.logfile.maxsize=10m
# Maximum number of rolled log files which will be allowed before old
# files are deleted. The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
#*****************************************************************************
# Wrapper Windows Service and Posix Daemon Properties
#*****************************************************************************
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
#*****************************************************************************
# Wrapper System Tray Properties
#*****************************************************************************
# enable system tray
wrapper.tray = true
# TCP/IP port. If none is defined multicast discovery is used to find the port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002

# Exit Code Properties
#********************************************************************
# Restart on non zero exit code
#********************************************************************
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART

# Trigger actions on console output
#********************************************************************
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\trayMessage.gv
wrapper.filter.script.0.args=Exception

# genConfig: further Properties generated by genConfig
#********************************************************************
placeHolderSoGenPropsComeHere=

wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 = -Xbootclasspath/a:${carbon_home}\lib\xboot\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 = -XX:HeapDumpPath=${carbon_home}\repository\logs\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 = -Djava.endorsed.dirs=${carbon_home}\lib\endorsed;${java_home}\jre\lib\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=/
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\bin\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\tmp
wrapper.java.additional.14 = -Dcatalina.base=${carbon_home}\lib\tomcat
wrapper.java.additional.15 = -Djava.util.logging.config.file=${carbon_home}\repository\conf\log4j.properties
wrapper.java.additional.16 = -Dcarbon.config.dir.path=${carbon_home}\repository\conf
wrapper.java.additional.17 = -Dcarbon.logs.path=${carbon_home}\repository\logs
wrapper.java.additional.18 = -Dcomponents.repo=${carbon_home}\repository\components\plugins
wrapper.java.additional.19 = -Dconf.location=${carbon_home}\repository\conf
wrapper.java.additional.20 = -Dcom.atomikos.icatch.file=${carbon_home}\lib\transactions.properties
wrapper.java.additional.21 = -Dcom.atomikos.icatch.hide_init_file_path=true
wrapper.java.additional.22 = -Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true
wrapper.java.additional.23 =
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.

Running the product in console mode

You will now verify that YAJSW is configured correctly for running the Carbon-based product as a Windows service.

1. Open a Windows command prompt and go to the <YAJSW_HOME>/bat/ directory. For example:

   \cd C:\Documents and Settings\yajsw_home\bat

2. Start the wrapper in console mode using the following command:

   runConsole.bat

   For example:

   C:\Documents and Settings\yajsw_home\bat>runConsole.bat

If the configurations are set properly for YAJSW, you will see console output similar to the following and can now access the WSO2 management console from your web browser via https://localhost:9443/carbon.
Working with the WSO2CARBON service

To install the Carbon-based product as a Windows service, execute the following command in the `<YAJSW_HOME>/bat` directory:

```
installService.bat
```

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

To start the service, execute the following command in the same console window:

```
startService.bat
```

The console will display a message confirming that the WSO2CARBON service was started.
To stop the service, execute the following command in the same console window:

```
stopService.bat
```

The console will display a message confirming that the WSO2CARBON service has stopped.

To uninstall the service, execute the following command in the same console window:

```
uninstallService.bat
```

The console will display a message confirming that the WSO2CARBON service was removed.
Installing as a Linux Service

WSO2 Carbon and any Carbon-based product can be run as a Linux service as described in the following sections:

- Prerequisites
- Setting up CARBON_HOME
- Running the product as a Linux service

Prerequisites

Install JDK and set up the JAVA_HOME environment variable. For more information, see Installation Prerequisites.

Setting up CARBON_HOME

Extract the WSO2 product that you want to run as a Linux service and set the environment variable CARBON_HOME to the extracted product 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 Service"
    ;;
  stop)
    echo "Stopping Service"
    ;;
  restart)
    echo "Restarting Service"
    ;;
  *)
    echo "$\{$0 \{start|stop|restart\}\}"
  esac

exit 1

For example, given below is a startup script written for WSO2 Application Server 5.2.0:

#!/bin/sh
export JAVA_HOME="/usr/lib/jvm/jdk1.7.0_07"

startcmd='/opt/WSO2/wso2as-5.2.0/bin/wso2server.sh start > /dev/null &'
restartcmd='opt/WSO2/wso2as-5.2.0/bin/wso2server.sh restart > /dev/null &'
stopcmd='opt/WSO2/wso2as-5.2.0/bin/wso2server.sh stop > /dev/null &'

case "$1" in
  start)
    echo "Starting WSO2 Application Server ..."
    su -c "$\{$startcmd\}" user1
    ;;
  restart)
    echo "Re-starting WSO2 Application Server ..."
    su -c "$\{$restartcmd\}" user1
    ;;
  stop)
    echo "Stopping WSO2 Application Server ..."
    su -c "$\{$stopcmd\}" user1
    ;;
  *)
    echo "Usage: $0 \{start|stop|restart\}"
  esac

exit 1

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 appserver 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/appserver`
- Add a link to /etc/init.d/: `sudo ln -snf /opt/WSO2/appserver /etc/init.d/appserver`

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 appserver 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.

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. This page describes how to run the product in the following sections:

- Before you begin
- Changing the default configurations
- Starting the server
- Accessing the management console
- Stopping the server

Before you begin

Before you start the server, note the following:

- 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-serv er.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.

Changing the default configurations
After you download the product, the default configurations in the product have to be changed according to your requirement, before you start the server. See the following topics for details:

- Configuring Environments for RDBMS
- Configuring Default Cassandra Environment
- Changing the Default HDFS Configuration

Starting the server

To start the server, you run the script `wso2server.bat` (on Windows) or `wso2server.sh` (on Linux/Solaris) from the bin folder. Alternatively, you can install and run the server as a Windows service.

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

1. Open a command prompt:
   - On Windows, choose Start -> Run, type `cmd` at the prompt, and press Enter.
   - On Linux/Solaris, establish a SSH connection to the server or log in to the text Linux console.
2. Execute one of the following commands, where `<SS_HOME>` is the directory where you installed the product distribution:
   - On Windows: `<SS_HOME>/bin/wso2server.bat --run`
   - On Linux/Solaris: `sh <SS_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 one of the following commands.

   - On Windows: `<PRODUCT_HOME>/bin\wso2server.bat --run -DworkerNode`
   - On Linux/Solaris: `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`. The description of each startup option is given here.

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

Accessing the management console

Once the server has started, you can run the Management Console by opening a Web browser and typing in the management console's URL. The URL is displayed as the last line in the start script's console and log. For example:

```
https://<Server Host>:9443/carbon
```

The URL should be in the following format: `https://<Server Host>:9443/carbon`

You can use this URL to access the Management Console on this computer from any other computer connected to the Internet or LAN. When accessing the Management Console from the same server where it's installed, you can type "localhost" instead of the IP address: `https://localhost:9443/carbon`

At the sign-in screen, sign in to the Management Console using `admin` as both the username and password. You can then use the Management Console to manage the product. The tabs and menu items in the navigation pane on the left may vary depending on the features you have installed.
To view information about a particular page, click the Help link in the top right corner of that page, or click the Docs link to open this documentation for full information on managing the product.

When the Management Console Sign-in page appears, the web browser will typically display an “Insecure connection” message, which requires your confirmation before you can continue.

The Management Console is based on HTTPS protocol, which is a combination of HTTP and SSL protocols. This protocol is generally used to encrypt the traffic from the client to server for security reasons. The certificate it works with is used for encryption only, and does not prove the server identity, so when you try to access the Management Console, a warning of untrusted connection is usually displayed. To continue working with this certificate, some steps should be taken to “accept” the certificate before access to the site is permitted. If you are using the Mozilla Firefox browser, this usually occurs only on the first access to the server, after which the certificate is stored in the browser database and marked as trusted. With other browsers, the insecure connection warning might be displayed every time you access the server.

This scenario is suitable for testing purposes, or for running the program on the company’s internal networks. If you want to make the Management Console available to external users, your organization should obtain a certificate signed by a well-known certificate authority, which verifies that the server actually has the name it is accessed by and that this server belongs to the given organization.

If you leave the Management Console unattended, the session will time out. The default timeout value is 15 minutes, but you can change this in the `<SS_HOME>/repository/conf/tomcat/carbon/WEB-INF/web.xml` file as follows:

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

Restricting Access to the Management Console and Web Applications:

You can restrict access to the management console of your product by binding the management console with selected IP addresses. Note that you can either restrict access to the management console only, or you can restrict access to all web applications in your server as explained below.
To control access only to the management console, add the IP addresses to the `<PRODUCT_HOME>/repository/conf/tomcat/carbon/META-INF/context.xml` file as follows:

```xml
<Valve
    className="org.apache.catalina.valves.RemoteAddrValve"
    allow="<IP-address-01>|<IP-address-02>|<IP-address-03>"/>
```

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

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

```xml
<Valve
    className="org.apache.catalina.valves.RemoteAddrValve"
    allow="<IP-address-01>|<IP-address-02>|<IP-address-03>"/>
```

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

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

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

```xml
<filter>
    <filter-name>Remote Address Filter</filter-name>
    <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
    <init-param>
        <param-name>allow</param-name>
        <param-value>127.0.0.1</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>Remote Address Filter</filter-name>
    <url-pattern>/carbon/admin/login.jsp</url-pattern>
</filter-mapping>
```

**Note:** Any configurations (including valves) 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.
Stopping the server

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

Upgrading from a Previous Release

This page takes you through the steps for upgrading from SS 1.1.0 version to SS 1.5.0. For more information on release versions, see the Release Matrix.

- Preparing to upgrade
- Upgrading the database
- Migrating the configurations
- Testing the upgrade

Preparing to upgrade

The following prerequisites must be completed before upgrading:

- Make a backup of the SS 1.1.0 database and copy the <SS_HOME_1.1.0> directory in order to backup the product configurations.
- Download WSO2 Storage Server 1.5.0 from http://wso2.com/products/storage-server/.

The downtime is limited to the time taken for switching databases when in the production environment.

Upgrading the database

1. Before you upgrade to SS 1.5.0, create a new database and restore the backup of the old database in this new database.

   You should NOT connect a new version of WSO2 SS to an older database that has not been upgraded.

2. Execute the relevant migration script as explained below. This will ensure that the new database is updated with the new tables and schemas required for SS 1.5.0.
   - If your system is used for provisioning RDBMSs, download the migration script relevant to the RDBMS type (Mysql_Migration_Script or Postgres_Migration_Script) and run the script against the new database you will use as the RSS metadata repository. This will update the new tables and schemas required for meta data related to RSS environments in SS 1.5.0.
   - If your system is used for provisioning Cassandra, download the cassandra_permissions_migration.sql script and run it against the new database you will use for storing role-based permissions. This will update the database with changes relevant to role-based Cassandra permissions in SS 1.5.0.

Once you run the migration scripts on the new database, it becomes the upgraded database for SS 1.5.0.

Migrating the configurations

Once you have completed the database upgrade as explained in the previous steps, you must migrate the
configurations from SS 1.1.0 to AS 1.5.0 as explained below.

Updating the configuration files

The following are the updates that need to be done to the configuration files in SS 1.5.0.

Note that configuration files should not be copied directly between servers.

1. If your system is used for provisioning RDBMSs, see the topics on configuring the RSS metadata repository and configuring RSS environments and change the relevant configuration files accordingly.
2. If your system is used for provisioning Cassandra, see the topic on configuring Cassandra environments and change the relevant configuration files accordingly.
3. Check for any other configurations that were done for SS 1.1.0 (based on your solutions), and update the configuration files in SS 1.5.0 accordingly. For example, external user stores, caching, mounting, etc.
4. Start the server.

Testing the upgrade

Verify that all the required scenarios are working as expected with SS 1.5.0. This confirms that the upgrade is successful.

Get Involved

WSO2 products are 100% open source and released under the Apache License Version 2.0. WSO2 welcomes anyone who is interested in WSO2 products to become a contributor by getting involved in the WSO2 community and helping with the development of WSO2 projects.

How can I get involved in the community?

Contributing as a non-committer – anyone can do it!
- Overview of the WSO2 repository
- Contributing to the WSO2 code base
- WSO2 GitHub guidelines

How can I get involved in the community?

You can get involved in the WSO2 community in various ways:

- **Use WSO2 products**
  The latest binary packs that correspond to the WSO2 product releases can be downloaded freely via the respective product pages on the [WSO2 website](https://wso2.com). We recommend that you download and use WSO2 products so that you can discover the advantages of our lean middleware stack. Your feedback on our products is much appreciated, as it will help us to drive our product roadmaps and the underlying technology. For information on product releases, go to the [Release Matrix](https://wso2.com/). For tutorials, articles, white papers, webinars, WSO2 documentation, and other learning resources, look in the Resources menu on the [WSO2 website](https://wso2.com).

- **Join WSO2 mailing lists**
  Many WSO2 mailing lists are open to the public, so anyone interested in WSO2 products can monitor the mail threads. You can subscribe to the [dev@wso2.org](mailto:dev@wso2.org) and [architecture@wso2.org](mailto:architecture@wso2.org) mailing lists to get involved in the discussions on WSO2 development. For more information on subscribing to these mailing lists, see [WSO2 Mailing Lists](https://wso2.com/).

- **Participate in user forums**
  The WSO2 team monitors and participates in the discussions on [Stack Overflow](https://stackoverflow.com). If you have any technical or programming questions related to WSO2 products, post them on Stack Overflow. Be sure to tag your...
question with appropriate keywords such as WSO2 and the product name so that our team can easily find your questions and provide answers. If you cannot find an answer on the user forum, you can email the WSO2 development team directly using the relevant mailing lists described at WSO2 Mailing Lists. We also encourage you to contribute by answering your fellow users’ questions on Stack Overflow.

- **Report bugs**
  WSO2 has a public bug-tracking system that you can use to report issues, submit enhancement requests, and track and comment on issues. You can also use this system to report issues related to WSO2 product documentation. If you find a bug, first search the dev mailing list to see if someone has faced the same issue, and also check the bug-tracking system. If you can't find any information on the issue, create an issue in the bug-tracking system with as much information as possible, including the product version you were using, how to reproduce the issue, etc.

- **Contribute to the WSO2 code base**
  WSO2 invites you to contribute by providing patches for bug fixes or features. For this purpose you can check out the source of the relevant GitHub repository, build the product, and make changes. You can then contribute your changes by sending a pull request for review. For more information, see the next section.

**Contributing as a non-committer – anyone can do it!**

**Anyone** (not just committers) can share contributions to WSO2's open-source software products. Your work will be recognized: if your contribution – feature enhancement, bug fix, or other improvements – is accepted, your name will be included as an author in the official commit logs. Read on for details on how you can contribute.

**Overview of the WSO2 repository**

WSO2 uses Git as its source control management system. The WSO2 Git repository maintains the code repository and the active build for continuous delivery incorporated with integrated automation.

**Contributing to the WSO2 code base**

Follow these instructions to contribute to the WSO2 code base. Be sure to follow the WSO2 GitHub Guidelines.

1. **Fork** the respective code base to your Git account.
2. Clone the code base to your local machine.
   
   `git clone <GitHub-REPOSITORY-URL>`

   If you are not sure which repository needs to be cloned, send an email to dev@wso2.org.

3. Build the product using Maven.

**Prerequisites**

- Install Maven and JDK. See the Installation Prerequisites page for compatible versions.
- Set the environment variable `MAVEN_OPTS="-Xms768m -Xmx3072m -XX:MaxPermSize=1200m"` to avoid the maven OutOfMemoryError.
- Make sure the build server has an active Internet connection to download dependencies while building.

Use the following commands to create complete release artifacts of a WSO2 product, including the binary and source distributions.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mvn clean install</code></td>
<td>The binary and source distributions.</td>
</tr>
</tbody>
</table>
4. If you need to add a new file to the repository:

1. Add the new file.
   
   ```
   git add <FILE-NAME>
   
   For example:
   
   git add mycode.java
   ```

2. Commit the newly added file to your local repository.

   ```
   git commit -m "<COMMIT-MESSAGE>"
   
   For example:
   
   git commit -m "Adding a new file"
   ```

5. If you need to update an existing file in the repository:

1. Open the file that you want to update and make the necessary changes.

2. Commit the changes to your local repository.

   ```
   git commit -m "<COMMIT-MESSAGE>" -a
   
   For example:
   
   git commit -m "Updated the clauses in the terms and conditions file" -a
   ```

6. Sync your changes with the upstream repository.

   ```
   git remote add <TAG-NAME> <UPSTREAM-GIT-REPO-URL>
   git fetch <TAG-NAME>
   git merge <TAG-NAME>/<BRANCH-NAME>
   ```

   For example:

   ```
   git remote add wso2_upstream https://github.com/wso2/wso2-synapse.git
   git fetch wso2_upstream
   git merge wso2_upstream/master
   ```

7. Push the changes to your own Git repository.

   ```
   git push
   ```

8. Send a Git pull request to the WSO2 Git repository and add the URL of the Git pull request in the JIRA that corresponds to the patch. Your pull request will be authorized only after it is reviewed by the team lead or release manager or responsible person for the corresponding Git repository.

For more information on using GitHub, see the related help articles Fork A Repo and Using Pull Requests.

**WSO2 GitHub guidelines**

- **The respective WSO2 Git repository should be forked**
  
  When contributing to WSO2 code base by way of a patch, make sure you identify the correct Git repository that needs to be forked. For more information on WSO2 Git repositories, see WSO2 GitHub Repositories. If you still are not sure which repository needs to be cloned, send an email to dev@wso2.org so that a WSO2 team member can advise you.

- **Do not build any dependencies**
  
  You do not need to build any dependencies, as everything you need will be automatically fetched from the Maven repository (Nexus) when you are building the product on your machine. Make sure the build server
has an active Internet connection to download dependencies while building.

- **Always sync with the forked repository before issuing a pull request**

There is a high possibility that the forked repository may differ from the upstream repository (remote repository that was forked) that you initially forked. Therefore, always sync the repository to prevent pull requests from being rejected.

**WSO2 GitHub Repositories**

**Kernel level Git repositories**

<table>
<thead>
<tr>
<th>Repo URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>carbon-kernel</td>
<td>Carbon 5 kernel repo</td>
</tr>
<tr>
<td>carbon4-kernel</td>
<td>Carbon 4 kernel repo</td>
</tr>
</tbody>
</table>

**Platform level Git repositories**

<table>
<thead>
<tr>
<th>Repo URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>carbon-analytics</td>
<td>Contains components and features related to analytics services.</td>
</tr>
<tr>
<td>carbon-apimgt</td>
<td>Contains components and features related to API management.</td>
</tr>
<tr>
<td>carbon-appmgmt</td>
<td>Contains components and features related to application management.</td>
</tr>
<tr>
<td>carbon-business-messaging</td>
<td>Contains the components and features related to business messaging.</td>
</tr>
<tr>
<td>carbon-business-process</td>
<td>Contains components and features related to business processes.</td>
</tr>
<tr>
<td>carbon-commons</td>
<td>Contains common components and features shared across the platform projects.</td>
</tr>
<tr>
<td>carbon-data</td>
<td>Contains components and features related to data services.</td>
</tr>
<tr>
<td>carbon-deployment</td>
<td>Contains components and features related to web application and service development (i.e., JavaEE WebProfile support, JAX-WS/RS service deployment, Webapp monitoring dashboards etc. ).</td>
</tr>
<tr>
<td>carbon-event-processing</td>
<td>Contains components and features related to event processing services.</td>
</tr>
<tr>
<td>carbon-governance</td>
<td>Contains components and features related to governance services.</td>
</tr>
<tr>
<td>carbon- mediation</td>
<td>Contains components and features related to mediation services.</td>
</tr>
<tr>
<td>carbon-ml</td>
<td>Contains components and features related to machine learner.</td>
</tr>
<tr>
<td>carbon-multitenancy</td>
<td></td>
</tr>
<tr>
<td>carbon-parent</td>
<td></td>
</tr>
<tr>
<td>carbon-platform-automated-test-suite</td>
<td>Contains WSO2 product integration test suites and Platform test suites with ant based test executor.</td>
</tr>
<tr>
<td>carbon-platform-integration</td>
<td>Contains WSO2 test automation framework modules.</td>
</tr>
<tr>
<td>carbon-platform-integration-utils</td>
<td>Contains utilities related to WSO2 test automation framework which is common to the whole product platform.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>carbon-qos</td>
<td>Contains components and features related to quality of service.</td>
</tr>
<tr>
<td>carbon-registry</td>
<td>Contains components and features related to registry services.</td>
</tr>
<tr>
<td>carbon-rules</td>
<td>Contains components and features related to business rules.</td>
</tr>
<tr>
<td>carbon-storage-management</td>
<td>Contains sources corresponding to the components that are primarily being used for storage provisioning and management related tasks. Out of all the components being maintained within this particular repository some components (i.e., Cassandra, HDFS) are used across the platform. In addition, some of the tools developed for storage browsing (i.e., Cassandra-Explorer etc.) too are part of this repository.</td>
</tr>
<tr>
<td>carbon-store</td>
<td></td>
</tr>
<tr>
<td>carbon-utils</td>
<td>Contains ntask, remote-tasks, ndatasource etc.</td>
</tr>
</tbody>
</table>

**Mobile platform Git repositories**

<table>
<thead>
<tr>
<th>Repo URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emm-agent-android</td>
<td>Maintains the Android agent that is used to enroll the device to EMM server.</td>
</tr>
<tr>
<td>emm-agent-ios</td>
<td>Maintains the iOS agent that is used to enroll the device to EMM server.</td>
</tr>
</tbody>
</table>

**Product level Git repositories**

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Repo URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Manager</td>
<td>product-apim</td>
<td>Maintains sources corresponding to building and packaging of WSO2 API manager distribution.</td>
</tr>
<tr>
<td>App Factory</td>
<td>product-af</td>
<td>Maintains sources corresponding to building and packaging of WSO2 APP Factory distribution.</td>
</tr>
<tr>
<td>Application Server</td>
<td>product-as</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Application Server distribution.</td>
</tr>
<tr>
<td>Complex Event Processor</td>
<td>product-cep</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Complex Event Processor distribution.</td>
</tr>
<tr>
<td>Data Analytics Server</td>
<td>product-bam</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Data Analytics Server distribution.</td>
</tr>
<tr>
<td>Data Services Server</td>
<td>product-dss</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Data Services Server distribution.</td>
</tr>
<tr>
<td>Enterprise Mobility Manager</td>
<td>product-emm</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Enterprise Mobility Manager distribution.</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Enterprise Service Bus</td>
<td>product-esb</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Enterprise Service Bus distribution.</td>
</tr>
<tr>
<td>Enterprise Store</td>
<td>product-es</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Enterprise Store distribution.</td>
</tr>
<tr>
<td>Governance Registry</td>
<td>product-greg</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Governance Registry distribution.</td>
</tr>
<tr>
<td>Identity Server</td>
<td>product-identity</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Identity Server distribution.</td>
</tr>
<tr>
<td>Message Broker</td>
<td>product-mb</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Message Broker distribution.</td>
</tr>
<tr>
<td>Private PaaS</td>
<td>product-private-paas</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Private PaaS distribution.</td>
</tr>
<tr>
<td>Storage Server</td>
<td>product-ss</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Storage Server distribution.</td>
</tr>
<tr>
<td>Task Server</td>
<td>product-ts</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Task Server distribution.</td>
</tr>
<tr>
<td>Developer Studio</td>
<td>developer-studio</td>
<td>Maintains sources corresponding to building and packaging of WSO2 Developer Studio distribution.</td>
</tr>
</tbody>
</table>

**Other WSO2 Git repositories**

The following are GitHub repository URLs that correspond to independent projects managed by WSO2:

<table>
<thead>
<tr>
<th>Repo URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>andes</td>
<td>Message broker core engine implementation.</td>
</tr>
<tr>
<td>balana</td>
<td>XACML core engine implementation.</td>
</tr>
<tr>
<td>charon</td>
<td>SCIM core engine implementation.</td>
</tr>
<tr>
<td>esb-connectors</td>
<td>Collection of connectors that allows you to interact with WSO2 ESB's third-party product function.</td>
</tr>
<tr>
<td>jaggery</td>
<td>This repo contains Jaggeryjs. Jaggery is a framework used to write webapps and HTTP-focused web services for all aspects of the application: front-end, communication, Server-side logic and persistence in pure Javascript.</td>
</tr>
<tr>
<td>jaggery-extensions</td>
<td>This contains extensions for the Jaggery framework.</td>
</tr>
<tr>
<td>orbit</td>
<td>Used to create OSGi bungles out of third-part dependencies.</td>
</tr>
<tr>
<td>siddhi</td>
<td>Complex event processing core engine implementation.</td>
</tr>
</tbody>
</table>
User Guide

The user guide provides information about the features, functionality, solution development, and debugging options of WSO2 Storage Server.

- Relational Store Service
- Cassandra Columnar Store Service
- Distributed File System Service

Relational Store Service

The WSO2 Relational Store Service (RSS) functionality of the Storage Server allows you to create and manage Relational Database Management Systems (RDBMS) seamlessly through a simplified user interface (management console). It provides the facility for tenants to create separate RDS instances and to provision databases within them by creating the required relational databases, tables, users, privilege templates etc.

The following topics explain how to create RDS instances, connect to the RDS from WSO2 RSS, create and manage your databases through the RSS user interface and then connect to your backend RDS instances from applications:

- Configuring Environments for RDBMS
- Provisioning an RDBMS
- Monitoring RSS Statistics using WSO2 BAM
- RSS CLI Commands Guide
- Using Created Databases/Database Users in Applications

Configuring Environments for RDBMS

In order to use the RDBMS provisioning facility offered by WSO2 Storage Server, as the first step, you need to configure the RDBMS environments using a configuration file. This includes setting up separate environments with database server instances and configuring the RSS metadata repository that will be used to maintain user-created databases, database users and privilege templates in each of the environments.

The following diagram illustrates a set up of environments for provisioning databases:

![Diagram of environments for RDBMS provisioning](image)

See the following topics for details on how to do these configurations:

- Configuring the RSS Metadata Repository
Configuring the RSS Metadata Repository

RSS metadata repository is used to maintain the databases, database users and privilege templates in each of the RSS environments in your system. Instructions for configuring RSS environments for your system are explained here. By default, the RSS metadata repository is pointed to an embedded H2 database that is shipped as part of the WSO2 Storage Server installation archive. In a typical production setting, we recommend to use an enterprise-grade RDBMS such as MySQL, SQL SERVER or Oracle as the RSS metadata repository.

To set up and configure the RSS metadata repository:

1. Create a database as the RSS metadata repository and a database user with admin privileges equivalent to the DBA role (this grants access to the database used as the RSS metadata repository).

   The database used as the metadata repository as well as all the database instances that you are going to provision (in every environment of your system) should be of the same RDBMS type.

2. Select the SQL script appropriate to the RDBMS you have from <SS_HOME>/dbscripts directory and run it. This creates the RSS metadata repository database structure. For example, if your RDBMS is MySQL, the script you must run is <SS_HOME>/dbscripts/rss-manager/mysql/wso2_rss_mysql.sql directory.

3. Add the required JDBC drivers for the database you selected to <SS_HOME>/repository/components/lib folder.

4. Create a datasource in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file and point to that from the <PRODUCT_HOME>/repository/conf/etc/rss-config.xml file using its JNDI name. Shown below is how the default metadata repository ("rss_db") is configured in these files.

   1. The default master-datasources.xml file.

      • SQL SERVER
      • MySQL
      • PostgreSQL

      SQL SERVER configurations:
<datasource>
  <name>MetadataRepo</name>
  <jndiConfig>
    <name>MetadataRepoDS</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <dataSourceClassName>com.microsoft.sqlserver.jdbc.SQLServerXADatasource</dataSourceClassName>
      <dataSourceProps>
        <property name="serverName">192.168.17.21</property>
        <property name="portNumber">1433</property>
        <property name="databaseName">rss_db</property>
        <property name="user">RootUser</property>
        <property name="password">RootPassword</property>
      </dataSourceProps>
    </configuration>
  </definition>
</datasource>

In this example, it uses Microsoft JDBC driver to connect to the database. You have to configure SQL Server to use XA datasource before configuring Storage Server. Configuration instructions can be found at http://msdn.microsoft.com/en-us/library/aa342335.aspx.

You can also use JTDS JDBC driver instead of Microsoft driver. Then `dataSourceClassName` in the above configuration should be `net.sourceforge.jtds.jdbcx.JtdsDataSource`. The server configuration instructions for XA datasources can be found in the README.XA file in the jTDS driver distribution pack.

MySQL configurations:
<datasource>
    <name>MetadataRepo</name>
    <jndiConfig>
        <name>MetadataRepoDS</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <dataSourceClassName>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</dataSourceClassName>
            <dataSourceProps>
                <property name="URL">jdbc:mysql://localhost:3306/rss_db</property>
                <property name="user">RootUser</property>
                <property name="password">RootPassword</property>
            </dataSourceProps>
        </configuration>
    </definition>
</datasource>

PostgreSQL configurations:

<datasource>
    <name>MetadataRepo</name>
    <jndiConfig>
        <name>MetadataRepoDS</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <dataSourceClassName>org.postgresql.xa.PGXADatasource</dataSourceClassName>
            <dataSourceProps>
                <property name="serverName">localhost</property>
                <property name="portNumber">5432</property>
                <property name="databaseName">rss_db</property>
                <property name="user">RootUser</property>
                <property name="password">RootPassword</property>
            </dataSourceProps>
        </configuration>
    </definition>
</datasource>

When creating above database user, make sure to grant all privileges for the database before creating any tables in it (i.e. before you run the database script on database).
The elements of the sample configuration are explained below. You can use this information to edit the file according to your requirements. For more information please see configuring master datasources.

- **<dataSourceClassName>**: This is the XA Driver Class provided by the JDBC driver used to connect to the type of RDBMS in which the RSS metadata repository is configured. E.g., if RSS metadata repository database is configured on MySQL - "com.mysql.jdbc.jdbc2.optional.MysqlXADataSource" has to be used as the value of this particular attribute.

- **<dataSourceProps>**: Root element of the XA data source property collection that needs to be configured in order to initialize the XA datasources driver used in the RSS metadata repository configuration.

- **<property name="property_name">**: Element used to specify bean properties of the XA datasource driver used in RSS metadata repository configuration. (Note: The set of properties that are to be configured are different based on the XA driver class used). E.g., the sample configuration which can be used to populate the bean properties required for MySQL XA driver is as follows:

```xml
<dataSourceProps>
    <property name = "URL">xxxx</property>
    <property name = "user">xxxx</property>
    <property name = "password">xxxx</property>
</dataSourceProps>
```

2. The default rss-config.xml file configuration for the metadata repository is shown below. Please note that JNDI name of both config files should be the same. In this example, it is "MetadataRepoDS".

```xml
<ManagementRepository>
    <DataSourceConfiguration>
        <JndiLookupDefinition>
            <Name>MetadataRepoDS</Name>
        </JndiLookupDefinition>
    </DataSourceConfiguration>
</ManagementRepository>
```

### Configuring RSS Environments

The RSS provisioning environments in your system can be considered as logical boundaries separating a set of database server instances from another. One typical example of using multiple provisioning environment is, if you are running an application development infrastructure allowing users to develop applications in different environments within its life cycle, such as DEV, QA, STAGING, PROD, etc., you would ideally need to have separate database provider infrastructure per environment. The aforementioned feature can be a handful in provisioning all those environments with a single WSO2 Storage Server instance.

Note that the database used as the metadata repository as well as all the database instances that you are going to provision (in every environment of your system), should be of the same RDBMS type.

See the following topics for instructions:

- Configuring the default RSS environment
- Configuring additional RSS environments
Enabling snapshot creation for database instances

Configuring the default RSS environment

The RSS environments can be defined and configured for a system using the rss-config.xml file stored in the <SS_HOME>/repository/conf/etc/ directory. By default, WSO2 Storage Server comes with one environment, which contains one RSS instance. Shown below is the default configurations in the rss-config.xml file. Descriptions of the rss-config.xml elements are explained below, which you can use to edit the file according to your requirements.

- SQL SERVER
- MySQL
- PostgreSQL

SQL SERVER configuration:

```xml
<Provider>SQLSERVER</Provider>
<Environments>
   <Environment>
      <Name>DEFAULT</Name>
      <RSSInstances>
         <RSSInstance>
            <Name>WSO2RSS1</Name>
            <InstanceType>SYSTEM</InstanceType>
            <DbmsType>SQLSERVER</DbmsType>
            <ServerCategory>LOCAL</ServerCategory>
            <DataSourceConfiguration>
               <Definition>
                  <Url>jdbc:sqlserver://192.168.17.21</Url>
                  <Username>RootUser</Username>
                  <Password>RootPassword</Password>
               </Definition>
               <DriverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</DriverClassName>
            </DataSourceConfiguration>
         </RSSInstance>
      </RSSInstances>
   </Environment>
</Environments>
```

If you use jTDS JDBC driver, the DriverClassName should be net.sourceforge.jtds.jdbc.Driver.

MySQL configuration:
<Provider>MYSQL</Provider>

<Environments>
  <Environment>
    <Name>DEFAULT</Name>
    <RSSInstances>
      <RSSInstance>
        <Name>WSO2RSS1</Name>
        <InstanceType>SYSTEM</InstanceType>
        <DbmsType>MYSQL</DbmsType>
        <ServerCategory>LOCAL</ServerCategory>
        <DataSourceConfiguration>
          <Definition>
            <Url>jdbc:mysql://localhost:3306</Url>
            <Username>RootUser</Username>
            <Password>RootPassword</Password>
            <DriverClassName>com.mysql.jdbc.Driver</DriverClassName>
          </Definition>
        </DataSourceConfiguration>
      </RSSInstance>
    </RSSInstances>
  </Environment>
</Environments>

PostgreSQL configuration:
The elements of the sample configuration are explained below.

- `<Provider>`: Element used to configure the DBMS Type of the RSS instance. Fixed values are H2, MYSQL, SQLSERVER, POSTGRES.
- `<Environments>`: Root element that wraps the configurations for all RSS environments.
- `<Environment>`: Root element that wraps the configurations for an individual RSS environment. You can add new environments by duplicating this element in this file.
- `<Name>`: Element used to configure the name of the RSS environment.
- `<RSSInstances>`: Root element that wraps the RSS instances configured in the RSS environment.
- `<RSSInstance>`: Element used to configure one particular database server instance to be provisioned. You can add new RSS instances by duplicating this element in this file.
- `<InstanceType>`: Element used to configure the type of the RSS instance. Currently, the possible values of this particular attribute is SYSTEM and USER_DEFINED. SYSTEM implies that the configured RSS instance would be provisioned as part of the logical database cluster named SYSTEM. USER_DEFINED is for instances that are created by individual tenant users from the management console.
- `<DbmsType>`: Element used to configure the DBMS type of the RSS instance. Fixed values are H2, MYSQL, SQLSERVER, POSTGRES.
- `<DataSourceConfiguration>`: Root element of the datasource configuration of the RSS instance.
- `<Definition>`: Root element of the datasource configuration.
- `<Url>`: Element provided to configure the JDBC URL that is used to connect to the database server (configured as the RSS instance).
- `<Username>`: Element provided to configure the username of the database user. This user connects to the database server, which is configured as the RSS instance.
- `<Password>`: Element provided to configure the password of the database user.
- `<DriverClassName>`: Element provided to configure the JDBC driver class that is used to connect to the database server (which is configured as the RSS instance).

Configuring additional RSS environments
You can add any number of RSS environments to your system, by duplicating the <Environment> section in the rss-config.xml file (shown below) and updating the environment related details. Descriptions of the rss-config.xml elements are explained above, which you can use to edit the file according to your requirements.

```xml
<Provider/>

<Environments>
  <Environment>
    <Name/>
    <RSSInstances>
      <RSSInstance>
        <Name/>
        <InstanceType/>
        <DbmsType/>
        <ServerCategory/>
        <DataSourceConfiguration>
          <Definition>
            <Url/>
            <Username/>
            <Password/>
            <DriverClassName/>
          </Definition>
        </DataSourceConfiguration>
      </RSSInstance>
    </RSSInstances>
  </Environment>
</Environments>
```

### Enabling snapshot creation for database instances

When you define the RSS instances for an environments, you can enable the possibility to create snapshots of databases using the management console. A database snapshot is a complete copy of a source database, which is taken at the time of snapshot creation. You can enable this feature by adding the following configurations to the rss-config.xml file:

- To allow the Storage Server to create snapshots of databases in RSS instances, an authenticated connection has to be established between the Storage Server instance and the locations where the RSS instances are hosted. We will be using a keystore with a key pair for this purpose as follows:
  1. Create a key pair and add it to a keystore. Find more information about keystores in the working with security section.
  2. Update the rss-config.xml file with the path to the private key and the pass phrase as shown below. Note that the path to the private key created in step 1 above should be specified as the `<PrivateKeyPath>` in this configuration.
3. The public key of the keystore should be shared with the hosts of the RSS instances for the authentication to work.

- Details of where the RSS instance is hosted should be specified as shown below. Note that the below example shows details of the default RSS instance in SS.

```xml
<PrivateKeyConfiguration>
  <PrivateKeyPath>/data/private/my_keys/private.key</PrivateKeyPath>
  <PassPhrase>pass</PassPhrase>
</PrivateKeyConfiguration>
```

Note that this configuration is common to all RSS instances in all the RSS environments in your system.

- `<Host>`: The IP or host name of the location where the database server instance is hosted.
- `<Port>`: The port ID.
- `<Username>`: The user name required to log in to the host.
- `<TargetDirectory>`: The location of the database server instance in the hosting machine.

As explained above, you can add both environments as well the RSS instances directly using the rss-config.xml file. However, once you have configured all the RDBMS environments, you have the option of adding RSS instances to each environment using the management console of WSO2 Storage Server as explained here.

### Adding RSS Instances to Environment

In an enterprise system, there will be multiple environments (such as Development, QA, Production etc.) consisting of multiple database server instances (RSS instances). The information about the separate environments and the RSS instances contained in each environment can be defined in the rss-config.xml file stored in the `<SS_HOME>/repository/conf/etc/` folder as explained here. Alternatively, once you have already defined a particular environment in the rss-config.xml file, you can add RSS instances to that environment using the management console of WSO2 Storage Server as explained below.

There are two types of RSS instances:

- **SYSTEM**: Only the Super Tenant (System Administrator) is allowed to create RSS instances of this type.
- **USER_DEFINED**: Any tenant can create RSS instances of this type, provided that the required authority is
Tenant-specific RSS instances:

As a log in user, you need to be assigned to a role that is granted permission to create RSS instances in a particular environment. See more information about creating roles and granting permissions. Thereby, the possibility to define role-based permissions allow tenants to create and maintain separate RSS instances in the environments.

To create an RSS instance:

Note that all RSS instances in all the environments in your system, as well as the database used as the metadata repository should be of the same RDBMS type.

1. Start WSO2 Storage Server and log in to the management console.
2. Click RSS Instances under RSS Manager in the Main menu.
3. Click Add Server Instance and the following screen will open:

![New Database Server Instance](image)

4. Enter values in the relevant fields to define the RSS instance using the following descriptions:
   - **RSS Instance Name**: Give a name for the database server instance.
   - **Environment**: Select the environment in which the RSS instance should be created. Environments are defined in the rss-config.xml file as explained [here](#).
   - **Instance Type**: If you are logged in as the super tenant (admin user), you can select whether to create an RSS instance of type SYSTEM or USER_DEFINED. However, if you are not the super tenant, USER_DEFINED should be selected for this option.
   - **Type**: Specify the type of database instance. For example, H2, MySQL, Oracle etc.
• **JDBC URL**: The connection URL to pass to the JDBC driver to establish the connection.
• **Datasource Class Name**: The class name of the JDBC driver to use.
• **Administrative Username**: The user name that is used to access the database.
• **Administrative Password**: The password used to access the database.

5. In the section on **Snapshot Related Information**, enter the details of where the database server instance is hosted as shown below. You can use this information to **make snapshots of databases** created within the RSS instance at a given time.

   Note that snapshot creation cannot be used for database instances of "PostgreSQL" type.

   ![Snapshot Related Information](image)

   - **SSH Host**: The IP or host name of the location where the database server instance is hosted.
   - **SSH Port**: The port ID.
   - **SSH Username**: The user name required to log in to the host.
   - **Snapshot Target Directory**: The location of the database server instance in the hosting machine.

### Provisioning an RDBMS

When you have set up and **configured the RDBMS environments** for your system, users will be able to carry out the database provisioning activities in the relevant environments as explained below.

- **Creating Databases**
- **Creating Database Users**
- **Creating Privilege Templates**

Note that as a user, you need to be assigned to a role with the relevant authority in order to create or edit databases and users. The Admin user is by default granted authority to perform this task. See more information about managing users and roles.

### Creating Databases

In order to add a database user, you must first create a **database server instance**. A database server instance acts as the container for the provisioned databases/database users. By default, you have a database server cluster by the name of WSO2_RSS, which provides the storage provisioning infrastructure. However, individual tenants can also define their own. When databases are created, the Relational Storage Service of WSO2 Storage Server uses a round robin algorithm to select the next database server instance from the list pertaining to the WSO2_RSS database server cluster.

You can create server instances directly in `<PRODUCT_HOME>/repository/conf/etc/rss-config.xml` file as
1. Log in to the management console and select **Databases** under **RSS Manager**.

2. Select environment and click the **Add Database** link and fill in the database information.

3. The newly-created database appears in the **Databases** screen. For example, database URL allows you to point to the database from applications.

**Creating database snapshots:**

A database snapshot is a read-only, static view of a database. You can create snapshots of a database at any given time by clicking **Snapshot** in the above screen. Note that this function is only available if the RSS environments in your system are configured as explained here.
Attaching database users

1. Click the Manage link to attach a user and a privilege template to the newly-created database. For example,

Available Database User: This drop-down list is populated by the users created for the RDBMS. Also see Creating Database Users.

Database Privileges Template: This drop-down list is populated by the privilege templates created in RSS. It sets the appropriate permission scheme for the database user selected above. Also see Creating Privilege Templates.

2. After attached User to Database it will display like below.

The purpose of each of the links associated with the database user is described below.

Explore database:

This option opens the Database Console window, which displays the database tables and allows you to execute SQL queries on the data. For example,
Create datasources:

Datasource is the connection set up to a database from the server. A datasource is a resource for the user, who can use it to expose data as services across different cluster nodes of his/her tenant space.

Edit Privileges:

The Edit link allows you to change user privileges.

Detach User

Detach link allows you to detach user from database.

Creating Database Users

In order to add a database user, you must first create a database server instance. A database server instance acts as the container for the provisioned databases/database users. By default, you have a database server cluster by the name of WSO2_RSS, which provides the storage provisioning infrastructure. However, individual tenants can also define their own. When databases are created, the Relational Storage Service of WSO2 Storage Server uses a round robin algorithm to select the next database server instance from the list pertaining to the WSO2_RSS database server cluster.

You can create server instances directly in \(<PRODUCT_HOME>/repository/conf/etc/rss-config.xml\) file as explained in Configuring Environments for RDBMS. Alternatively, you can create the database server instances using the management console as explained in Creating RSS Instances.

Note that as a user, you need to be assigned to a role with the relevant authority in order to create or edit new database users from the management console. The Admin user is by default granted authority to perform this task. See more information about managing users and roles.

After a database server instance is created, follow the steps below to create a database user.

1. Log in to the management console and select Database Users under RSS Manager.

2. Select the environment and click Add Database User. Enter the new database user details as shown below.
3. Click **Save** to complete. You now have a database user, which you can connect to a database.

4. The created user can be connected to a database as explained [here](#).

**Creating Privilege Templates**

Storage Server gives you full control to define your own permission schemes, which you can assign to database users. Users are spared the overhead of writing and executing lengthy SQL queries to grant database privileges to database users.

Follow the steps given below to define privilege templates.

1. Log in to the management console and select **Privilege Templates** from **RSS Manager**.

2. Select an RSS provisioning environment in the **Environment** field. The privilege templates that are already defined for the selected environment will be listed.

3. Click **Add Database Privilege Template** to add a template to open the following screen:
4. Provide a name for the template and enable the permissions by selecting the appropriate check boxes. The permissions are described below.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>The SELECT privilege enables you to select rows from tables in a database</td>
</tr>
<tr>
<td>INSERT</td>
<td>The INSERT privilege enables rows to be inserted into tables in a database</td>
</tr>
<tr>
<td>UPDATE</td>
<td>The UPDATE privilege enables rows to be updated in tables in a database.</td>
</tr>
<tr>
<td>DELETE</td>
<td>The DELETE privilege enables rows to be deleted from tables in a database.</td>
</tr>
<tr>
<td>Privilege</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CREATE</td>
<td>The <strong>CREATE</strong> privilege enables creation of new databases and tables.</td>
</tr>
<tr>
<td>DROP</td>
<td>The <strong>DROP</strong> privilege enables you to drop (remove) existing databases, tables, and views.</td>
</tr>
<tr>
<td>GRANT</td>
<td>The <strong>GRANT</strong> privilege enables you to give privileges to a specific user or role, or to all users, to perform actions on database objects.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>The <strong>REFERENCES</strong> privilege Enables a user to create a foreign key dependency on a table or materialized view.</td>
</tr>
<tr>
<td>INDEX</td>
<td>The <strong>INDEX</strong> privilege enables you to create or drop (remove) indexes.</td>
</tr>
<tr>
<td>ALTER</td>
<td>The <strong>ALTER</strong> privilege enables use of altering the structures of available tables.</td>
</tr>
<tr>
<td>CREATE_TEMP_TABLE</td>
<td>The <strong>CREATE TEMPORARY TABLES</strong> privilege enables the creation of temporary tables.</td>
</tr>
<tr>
<td>LOCK_TABLES</td>
<td>The <strong>LOCK_TABLES</strong> explicitly acquires table locks for the current client session.</td>
</tr>
<tr>
<td>CREATE_VIEW</td>
<td>The <strong>CREATE VIEW</strong> privilege enables the user to create a view.</td>
</tr>
<tr>
<td>SHOW_VIEW</td>
<td>The <strong>SHOW VIEW</strong> privilege enables the user to view the created view.</td>
</tr>
<tr>
<td>CREATE_ROUTINE</td>
<td>The <strong>CREATE ROUTINE</strong> privilege is needed to create stored routines (procedures and functions)</td>
</tr>
<tr>
<td>ALTER_ROUTINE</td>
<td>The <strong>ALTER ROUTINE</strong> privilege is needed to alter or drop stored routines (procedures and functions).</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>The <strong>EXECUTE</strong> privilege is required to execute stored routines (procedures and functions).</td>
</tr>
<tr>
<td>EVENT</td>
<td>The <strong>EVENT</strong> privilege is required to create, alter, drop, or see events for the Event Scheduler</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>The <strong>TRIGGER</strong> privilege enables trigger operations.</td>
</tr>
</tbody>
</table>

5. Click **Save**.

### Monitoring RSS Statistics using WSO2 BAM

**WSO2 BAM** is a monitoring tool, which performs the task of aggregating, analyzing and presenting information about business activities. Thereby, BAM can be used with SS in order to effectively monitor RSS statistics.

To use BAM with SS for monitoring RSS statistics, you must first configure the SS to publish information to BAM. In BAM, the `.tbox` tool box should be installed. With this tool box, the BAM dashboard will graphically display the statistics published from Storage Server.

Note that monitoring RSS statistics using the BAM toolbox is only available for **MySQL** and **PostgreSQL** type databases.

Follow the instructions given below.

- **Enabling RSS statistics publisher in SS**
• Setting up the BAM tool box
• Viewing RSS statistics in BAM

**Enabling RSS statistics publisher in SS**

1. Stop Storage Server if it is already started.
2. To start the RSS statistics publisher, open the rss-monitor-config.xml file in the `<SS_HOME>/repository/conf/etc/` directory and set the `<monitoringEnable>` setting to true (default is false).
4. Give the cron expression using `<cronExpression>` to schedule the publisher.
5. Provide BAM admin credentials.
6. Start the Storage Server.

**Setting up the BAM tool box**

1. Stop the BAM server if it is already started.
2. Open the `master-datasources.xml` file in the `<BAM_HOME/repository/conf/datasources/` directory and add the `WSO2SS_STAT_DATASOURCE` data source as shown below. This datasource connects to the default H2 database in BAM. Therefore, the summarized RSS information published to BAM from SS will be stored in this database.

   The `master-datasources.xml` file, consists of the datasource connection to the default, internal database (H2) that is shipped with every WSO2 product. This database maybe replaced as explained in the Working with Databases section. Thereby, the datasource information in the `master-datasources.xml` file should be changed accordingly.

   ```xml
   <datasource>
       <name>WSO2SS_STAT_DATASOURCE</name>
       <description>The datasource used for ss stat analyzer</description>
       <definition type="RDBMS">
           <configuration>
               <url>jdbc:h2:repository/database/samples/WSO2SS_STAT_DB;AUTO_SERVER=TRUE</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>
   
   3. Start the BAM server to install the toolbox.

   Port offset may need to be changed. Open the `carbon.xml` file from the `<BAM_HOME>/repository/conf/` folder and change the `<offset>` to an appropriate value.
4. Go to the **Main** menu in the navigator and click **Add** in the **BAM Toolbox** menu.
5. You can now add the RSS statistics monitoring tool box (rss_stat_monitoring_toolbox.tbox), which is stored in the `<SS_HOME>/resources/bam-toolboxes` folder as a custom toolbox. Use the **ToolBox from File System** option as shown below. Click **Install**. The toolbox will now be added to BAM.

**Viewing RSS statistics in BAM**

1. In BAM, go to the **Dashboard** sub menu under the **Main** navigator menu.
2. Log in to the dashboard to view the statistics:

![Dashboard](image)

**RSS CLI Commands Guide**

After creating a database, you can connect to it, explore and perform database management operations through the Storage Server management console UI. See [Explore Databases](#).

The following guide gives you some of the common command-line operations used to connect to and query databases in different RDBMSs. From the list below, select the RSS CLI client based on the back end database server you have. If you do not find the RDBMS of your choice here, it is possibly not yet supported by WSO2 RSS.

**CLI commands for MySQL**

The following command can be used through a MySQL client application: To launch MySQL client: `mysql -u<user> -p -h <host name> <database name>`

- `<host name>`: RSS host name
- `<user>`: RSS Database user
- `<database name>`: the name of the database you want to connect to and manage

**Using Created Databases/Database Users in Applications**

- Introduction
  - Building sample application

**Introduction**

This sample demonstrates how to use databases and database users created through WSO2 Storage Server, in a
Building sample application

1. Extract the JDBC URL of the created database.
2. Set the user name and the password of a user who is attached to the database.
3. Select appropriate JDBC Driver class based on the backend RDBMS.

```java
Connection connection = null;
String driverClass = "com.mysql.jdbc.Driver";
String dbUser = "user123";
String dbPassword = "user123";
String jdbcUrl = "jdbc:mysql://localhost:3306/testDB";

BasicDataSource ds = new BasicDataSource();
ds.setDriverClassName(driverClass);
ds.setUrl(jdbcUrl);
ds.setUsername(dbUser);
ds.setPassword(dbPassword);

try {
    connection = ds.getConnection();
    //your code goes here
} catch (SQLException e) {
    log.error(e);
} finally {
    try {
        if (connection != null) {
            connection.close();
        }
    } catch (SQLException e) {
        log.error(e);
    }
}
```

Cassandra Columnar Store Service

WSO2 Column Store Service is based on the multi-tenanted Apache Cassandra. It offers highly-scalable, next-generation, distributed database management functionality, through the following feature in the WSO2 feature repository:

**Name:** WSO2 Carbon - Cassandra Feature

**Identifier:** org.wso2.carbon.cassandra.feature.group

Features can be installed or removed from the WSO2 Storage Server by following the instructions given in Installing Features.

Cassandra supports a Column Family based data model. The WSO2 Carbon Cassandra component uses the Carbon User Management component as the user base and helps users create and manage NoSQL data stores with authentication and authorization. The WSO2 Cassandra Explorer feature comes with a simplified management console UI. It allows users to add, view, manage and configure keyspaces, column families, columns and to explore and manipulate data.
Before you start:

- Get an understanding of Cassandra/NoSQL concepts and terminology by going through the Apache Cassandra Glossary: http://io.typepad.com/glossary.html.
- Get familiar with the basics of Cassandra, including the prerequisites for using Cassandra, by going through the following: https://wiki.apache.org/cassandra/GettingStarted.
- Get familiar with using the Cassandra CLI, by going through the following: http://wiki.apache.org/cassandra/CassandraCLI.
- Also, note that Cassandra requires the most stable version of Java 7 or 8 you can deploy; preferably the Oracle/Sun JVM.

See the following topics:

- Configuring Cassandra Environments
- Enabling Permissions for Cassandra
- Creating Keyspaces
- Creating Column Families
- Explore Cassandra Cluster
- Cassandra CLI/CQL Commands
- Monitor Cassandra Cluster

Clustering Cassandra

See here for information on how to cluster Cassandra.

Configuring Cassandra Environments

WSO2 Storage Server is shipped with a default Cassandra environment, which consists of one Cassandra cluster. When you start with WSO2 SS, you can change the configurations of the default environment and also add multiple environments to your system by following the instructions given in these topics:

- Configuring Default Cassandra Environment
- Configuring Multiple Cassandra Environments

Configuring Default Cassandra Environment

WSO2 Storage Server is shipped with a default Cassandra environment, which contains one Cassandra cluster. If required, you can set up additional Cassandra environments for your system, which can consist of multiple clusters. The information given on this page explains how to set up and configure the default Cassandra environment.

- Setting up a Cassandra cluster
- Configuring the SS nodes in the Cassandra cluster

Setting up a Cassandra cluster

WSO2 Storage Server consists of an embedded Cassandra instance. Therefore, the default Cassandra environment of Storage Server consists of a Cassandra cluster, which has one Cassandra node (the default embedded Cassandra instance) for provisioning NoSQL data stores. If you use this single node, default Cassandra cluster, tenant users can log in to the management console of SS and manage the NoSQL data stores for that single node Cassandra cluster.

However, in a production environment, you will need to have multiple Cassandra nodes for a Cassandra cluster. You can configure multiple Cassandra nodes for the default cluster by setting up multiple SS deployments as
Cassandra nodes.

As depicted above, you will be using one SS deployment as the provisioning SS instance, which will have the UI enabled for users to log in. All the other SS deployments in the system will be used as back-end Cassandra nodes connected to one another, thereby forming the Cassandra cluster.

Configuring the SS nodes in the Cassandra cluster

As explained in the above section, Storage Server comes with configurations suited for standalone Cassandra deployment. However, once you set up a multi-node cluster, you must configure each of the nodes as follows:

- **Step 1: Changing the default IP's and ports**
- **Step 2: Pointing to remote Cassandra nodes**
- **Step 3: Cassandra Cluster Configuration for Statistics and Node Operations**
- **Step 4: Exposing services to the public**

**Step 1: Changing the default IP's and ports**

You must change listening and ports accordingly in `<SS_HOME>/repository/conf/etc/cassandra.yaml` file for each of the SS nodes.

1. Cassandra listening IP is used for inter-node communications in a clustered environment: `listen_address: <Server listening IP or domain name>`
2. Storage port is used to exchange the data and the command between the cluster nodes: `storage_port: 7000`

   This port changes according to `<Offset>` value in `<Ports>` section in `carbon.xml`. Changing `storage_port` value in `cassandra.yaml` will not affect the server.

3. If encrypted communication is enabled, the cluster uses the port defined in `ssl_storage_port` for cluster-related commands and data communication: `ssl_storage_port: 7001`

RPC listen address is used for the thrift-based communication between the server and client:

```yaml
rpc_address: <IP_ADDRESS>
# port for Thrift to listen for clients on
rpc_port: 9160
```

RPC port changes according to `<Offset>` value in `<Ports>` section in `carbon.xml`. Changing `rpc_port` value in `cassandra.yaml` will not affect the server.
4. Native Transport Port is the port which is listening to CQL clients. Please note that the address on which the native transport is bound is the same as the rpc_address (to start the native transport server, start_native_transport should be equal to true, which is its default value). This needs to be set as follows:

```
start_native_transport: true
native_transport_port: 9042
```

For a full list of explanations of each configuration directive, refer to the file's code comments. Additionally, see [http://wiki.apache.org/cassandra/StorageConfiguration](http://wiki.apache.org/cassandra/StorageConfiguration).

**Step 2: Pointing to remote Cassandra nodes**

Each SS node in the cluster should be configured with the details of the other SS nodes in the cluster.

1. Open the `hector-config.xml` file stored in the `<SS_HOME>/repository/conf/etc/hector-config.xml` file.
2. List all the nodes in the cluster. Shown below is how the default Cassandra node is entered.

```
<HectorConfiguration>
  <Cluster>
    <Name>ClusterOne</Name>
    <Nodes>localhost:9160</Nodes>
    <AutoDiscovery disable="false" delay="1000"/>
  </Cluster>
</HectorConfiguration>
```

The XML elements are described below.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Name/&gt;</td>
<td>Hector reference name for the Cassandra cluster connection.</td>
</tr>
<tr>
<td>&lt;Nodes/&gt;</td>
<td>Comma separated is of Cassandra Cluster nodes.</td>
</tr>
<tr>
<td>&lt;AutoDiscovery disable=&quot;false&quot; delay=&quot;1000&quot;/&gt;</td>
<td>Enable Hector Auto node discovery service.</td>
</tr>
</tbody>
</table>

**Step 3: Cassandra Cluster Configuration for Statistics and Node Operations**

To view Cassandra cluster statistics and to perform cluster operations, the `cluster-config.xml` file stored in the `<SS_HOME>/repository/conf/etc/` directory needs to be configured. Here, all the SS nodes and their service URLs need to be configured.
<cluster>
  <configuration>
    <cluster_authentication>
      <username>admin</username>
      <password>admin</password>
    </cluster_authentication>
    <nodes>
      <node>
        <host>127.0.0.1</host>
        <backend_url>local://services/</backend_url>
      </node>
    </nodes>
  </configuration>
</cluster>

The XML elements are described below.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;cluster_authentication/&gt;</td>
<td>SS admin service authentication details</td>
</tr>
<tr>
<td>&lt;username/&gt;</td>
<td>Admin service username</td>
</tr>
<tr>
<td>&lt;password/&gt;</td>
<td>Admin service password</td>
</tr>
<tr>
<td>&lt;nodes/&gt;</td>
<td>Parent element of SS nodes</td>
</tr>
<tr>
<td>&lt;node/&gt;+</td>
<td>Node element for each SS node</td>
</tr>
<tr>
<td>&lt;host/&gt;</td>
<td>Host Name / IP of SS node</td>
</tr>
<tr>
<td>&lt;backend_url/&gt;</td>
<td>SS node's service URL</td>
</tr>
</tbody>
</table>

**Step 4: Exposing services to the public**

In a IaaS infrastructure, the services, public IP and domain names of the backend Cassandra cluster must be exposed via public addresses. This is done in the `cassandra-endpoint.xml` file stored in the `<SS_HOME>/repository/conf/etc/` directory. Given below is the default configuration, where the `<EndPoint>` and `<HostName>` elements represent each Cassandra node by its host name.
Configuring Multiple Cassandra Environments

In an enterprise system, you would typically have multiple environments, such as Development, QA, Production etc. Therefore, you can have scenarios where multiple Cassandra clusters are used in each of the separate environments. In such scenarios, WSO2 Storage Server can be configured to manage all the Cassandra clusters in all environments as depicted by the following diagram:

When you have multiple environments with multiple Cassandra clusters, you can configure them using the cassandra-environments.xml file stored in the `<SS_HOME>/repository/conf/etc/` directory. This file should be configured with details of all the environments and clusters that should be managed by your Storage Server. You should also configure datasources for each cluster in the master-datasources.xml file stored in `<SS_HOME>/repository/conf/datasources/` directory.

**Embedded Cassandra and External**
Cassandra Nodes:

When a Cassandra cluster is set up, you have the option of using a Storage Server instance as a cluster node, because Cassandra is by default embedded in WSO2 Storage Server. Otherwise, you have to use external Cassandra nodes. Note that all the cluster nodes in a particular environment should be either embedded Cassandra instances or external Cassandra nodes for the configurations in the cassandra-environments.xml file to be effective.

The following topics explain how to set up the Cassandra clusters in your environments, and how to configure the environments:

- Setting up Cassandra clusters
- Configuring the cassandra-environments.xml file
- Configuring datasources for Cassandra clusters

Setting up Cassandra clusters

A Cassandra cluster is a collection of Cassandra nodes. When you use WSO2 Storage Server, you have the option of setting up Cassandra clusters for an environment using two methods:

- **Using SS deployments as Cassandra nodes:** WSO2 Storage Server is shipped with an embedded Cassandra instance. This gives us the option of using SS deployments as the Cassandra nodes in a cluster. In this scenario, you will be using one SS deployment as the provisioning SS instance, which will have the UI enabled for users to log in. All the other SS deployments in the system will be used as back-end Cassandra nodes connected to one another, thereby forming the Cassandra cluster.
- **Using external Cassandra installations as Cassandra nodes:** You can install external (Vanilla) Cassandra nodes for your clusters as explained [here](#).

Note that all the Cassandra clusters in one environment should be of the same type for this configuration to work.

Configuring the cassandra-environments.xml file

After you set up all the clusters, you can partition them into separate environments by using the cassandra-environments.xml file. This file should be updated with the information of all environments in your system, including the default environment. The elements in this file are as follows:

- You should point to the default environment from the cassandra-environments.xml as shown below. Note that the default environment should be separately configured as explained [here](#).

```xml
<Environment>
    <Name>DEFAULT</Name>
    <IsExternal>false</IsExternal>
</Environment>
```

- The `<Environment>` section is used to add the details of environments. You can add any number of environments using this element.
- The `<IsExternal>` element is used to specify whether the clusters in an environment consist of embedded Cassandra nodes or external Cassandra nodes. Set this element to ‘true’ or ‘false’. If this value is set to ‘true’, you can only have external Cassandra nodes in the environment, and if the value is ‘false’, you can only have embedded Cassandra nodes as explained above.
• The `<Clusters>` element contains the details of the clusters within the environment. You can add any number of clusters using the `<Cluster>` element under `<Clusters>`.
• The `<DataSourceJndiName>` element specifies the datasource information corresponding to a Cassandra cluster. Therefore, the datasources for each cluster should be defined in the master-datasources.xml file stored in the `<SS_HOME>/repository/conf/datasources/` directory. See how datasources are configured for Cassandra clusters.

Shown below is the default configuration in the cassandra-environments.xml file.

```
<CassandraEnvironmentConfig>
  <CassandraEnvironments>
    <Environment>
      <Name>DEFAULT</Name>
      <IsExternal>false</IsExternal>
    </Environment>
    <Environment>
      <Name>DEV</Name>
      <IsExternal>true</IsExternal>
      <Clusters>
        <Cluster>
          <Name>DevCluster</Name>
          <DataSourceJndiName>DevDS</DataSourceJndiName>
        </Cluster>
        <Cluster>
          <Name>ProdCluster</Name>
          <DataSourceJndiName>ProdDS</DataSourceJndiName>
        </Cluster>
      </Clusters>
    </Environment>
  </CassandraEnvironments>
</CassandraEnvironmentConfig>
```

Configuring datasources for Cassandra clusters

Datasources should be defined for all Cassandra clusters (except for the default Cassandra cluster), in order to allow clients to access the cluster. In the datasource configuration (master-datasources.xml file) you can allow Hector-based clients as well as CQL-based clients to access a cluster by enabling the relevant datasource reader. Developers can refer to the datasource object from JNDi lookup.

The instructions for defining datasources are explained below.

1. Open the master-datasources.xml file from the `<SS_HOME>/repository/conf/` directory.
2. Enable the relevant datasource readers (Hector and CQL) in the file. The default configurations in the master-datasources.xml file for Hector and CQL are as follows:

   Note that the `DataSourceJndiName` given for the cluster in the cassandra-environments.xml file should be the same as the "jndiConfig/name" in master-datasources.xml.
• Enable the following for the Hector datasource reader:

```
<!-- Hector datasource -->
<provider>org.wso2.carbon.cassandra.datareader.hector.HectorBase
 dDataSourceReader</provider>
  <!-- datasource -->
  <name>HectorDS</name>
  <description>The datasource used for RSS metadata repository</description>
  <jndiConfig>
    <name>CassandraRepo</name>
  </jndiConfig>
  <definition type="HECTOR">
    <configuration>
      <hosts>10.100.101.3:9160,10.100.101.4:9160,10.100.101.5:9160</hosts>
      <username>admin</username>
      <password>admin</password>
      <clusterName>TestCluster</clusterName>
      <maxActive>200</maxActive>
      <enableSecurity>false</enableSecurity>
    </configuration>
  </definition>
</datasource-->
```

Note the following:

1. `<jndiConfig>`/<name> should be the same name given as the **DataSourceJNDiName** for the cluster in the master-datasources.xml file.

2. The IP addresses of the Cassandra nodes should be listed using the `<hosts>` element.

3. A separate datasource should be defined for each cluster by duplicating the `<datasource>` section shown above.

• Enable the following for CQL datasource reader:

```
<datasources>
  <!-- external Cassandra data source. please enable either one of datasource (CQL or Hector) based on your preference -->
  <!-- CQL datasource -->
  <provider>org.wso2.carbon.cassandra.datareader.cql.CassandraDataSourceReader</provider>
  <!-- datasource -->
  <name>WSO2_CASSANDRA_DB</name>
  <description>The datasource used for cassandra</description>
</datasources>
```
<jndiConfig>
  <name>CassandraRepo</name>
</jndiConfig>
<definition type="CASSANDRA">
  <configuration>
    <async>false</async>
    <clusterName>TestCluster</clusterName>
    <compression>SNAPPY</compression>
    <concurrency>100</concurrency>
    <username>admin</username>
    <password encrypted="true">admin</password>
    <port>9042</port>
    <maxConnections>100</maxConnections>
    <hosts>
      <host>127.0.0.1</host>
    </hosts>
    <loadBalancePolicy>
      <exclusionThreshold>2.5</exclusionThreshold>
      <latencyAware>true</latencyAware>
      <minMeasure>100</minMeasure>
      <policyName>RoundRobinPolicy</policyName>
      <retryPeriod>10</retryPeriod>
      <scale>2</scale>
    </loadBalancePolicy>
    <poolOptions>
      <coreConnectionsForLocal>8</coreConnectionsForLocal>
      <coreConnectionsForRemote>2</coreConnectionsForRemote>
      <maxConnectionsForLocal>10</maxConnectionsForLocal>
      <maxConnectionsForRemote>10</maxConnectionsForRemote>
      <maxSimultaneousRequestsForLocal>10</maxSimultaneousRequestsForLocal>
      <maxSimultaneousRequestsForRemote>10</maxSimultaneousRequestsForRemote>
      <minSimultaneousRequestsForLocal>10</minSimultaneousRequestsForLocal>
      <minSimultaneousRequestsForRemote>10</minSimultaneousRequestsForRemote>
    </poolOptions>
    <reconnectPolicy>
      <baseDelayMs>100</baseDelayMs>
    </reconnectPolicy>
  </configuration>
</definition>
<policyName>ConstantReconnectionPolicy</policyName>
  <reconnectPolicy>
    <socketOptions>
      <connectTimeoutMillis>10000</connectTimeoutMillis>
      <keepAlive>true</keepAlive>
      <readTimeoutMillis>15000</readTimeoutMillis>
      <tcpNoDelay>true</tcpNoDelay>
    </socketOptions>
  </reconnectPolicy>
Note the following:

1. `<jndiConfig><name>` should be the same name given as the `DataSourceJNDIName` for the cluster in the `master-datasources.xml` file.

2. The IP addresses of the Cassandra nodes should be listed using the `<hosts>` element.

3. A separate datasource should be defined for each cluster by duplicating the `<datasource>` section shown above.

### Enabling Permissions for Cassandra

With the User Management functionality that is built into WSO2 products, system administrators can define role-based permissions for tenant users. In WSO2 Storage Server, you can define role-based permissions for each of the Cassandra environments in your system. That is, as a tenant user, you will be allowed to add or edit the information in your Cassandra environment, only if the relevant permissions are granted. Therefore, this permission model provides the flexibility to maintain tenant-isolated Cassandra environments in your enterprise system.

The following topics explain role-based permissions and native Cassandra permissions in detail:

- Enabling role-based Cassandra permissions
- Enabling native Cassandra permissions

#### Enabling role-based Cassandra permissions

Role-based permissions are the general permissions that are granted to a role when the role is defined. When the Cassandra feature is installed in a product (as in WSO2 Storage Server), you can configure the permissions to manipulate the information in Cassandra environments. Note that, for SS embedded Cassandra, you need to configure the native Cassandra permissions, in addition to the role-based permissions described here.

**Before you begin**, all the Cassandra environments in your system should already be configured in the `cassandra-environments.xml` file (stored in the `<SS_HOME>/repository/conf/etc/` directory), to be able to grant permissions for a role.

Shown below is how you can enable Cassandra permission for the default Cassandra environment:
Enabling native Cassandra permissions

This feature is only applicable for environments with embedded Cassandra clusters. That is, when you use the embedded Cassandra instance of a WSO2 Storage Server, you have the in-built Cassandra authentication and authorization functionality, which can enable a separate set of permissions for Cassandra. You have the option of disabling this function in your server if required. However, if the feature is enabled, note that the in-built Cassandra permissions as well as the role-based Cassandra permissions should not conflict with one another.

Configuring Cassandra authentication and authorization

- In WSO2 Storage Server, to log in to embedded Cassandra, you need to be an authenticated Carbon user. The authentication class is configured in `<SS_HOME>/repository/conf/etc/cassandra.yaml` using the following authenticator: `org.wso2.carbon.cassandra.server.CarbonCassandraAuthenticator`. By default, Cassandra authentication is enabled for all users in WSO2 Storage Server. If required, you can set this configuration to 'disable authentication' as explained below.

```
# - AllowAllAuthenticator performs no checks - set it to disable authentication.
```

- In WSO2 Storage Server, Cassandra authorization can be managed in a fine-grained manner. The authorizer class is configured in `<SS_HOME>/repository/conf/etc/cassandra.yaml` using the following authorizer: `org.wso2.carbon.cassandra.server.CarbonCassandraAuthorizer`. By default, Cassandra authorization is enabled for all users in WSO2 Storage Server. If required, you can set this configuration to 'disable authorization' as shown below.

```
# - AllowAllAuthorizer allows any action to any user - set it to disable authorization
```
Enabling native Cassandra permissions

If the Cassandra authenticator and authorizer is enabled for the SS embedded Cassandra as explained in the previous step, you can enable or disable the native Cassandra permissions for a user role as explained below. Native Cassandra permissions can be enabled at three levels: all keyspaces, individual keyspace and column family. The permissions that can be enabled are Create, Alter, Drop, Select, Modify and Authorize.

Before you begin, note the following:

- You need to have authorization permissions to set permissions for other roles. See more information about roles and permissions.
- If a role has a particular permission for a resource in some level, that role implicitly gets the same permission for all lower levels of that resource.
- Cassandra caches its permissions. You can set cache expiry time in `<SS_HOME>/repository/conf/etc/cassandra.yaml` (the default is 2000 ms).

```
permissions_validity_in_ms: 2000
```

Be careful when you increase this value because Cassandra does not invalidate the permission cache when permissions are updated. It only invalidates this cache after the expiry time.

To enable keyspace level authorization:

- Go to Manage > Cassandra Keyspaces > List. You can see the following page which allows you to manage root level permissions for each role.

To enable keyspace/Column family level authorization:

- Go to Manage > Cassandra Keyspaces > List.
- Click Set Permissions in the list of actions for the keyspace. You can then manage the native permissions applicable to the column families in the keyspace.

Creating Keypaces

Conceptually similar to a database in relational database management systems, a keyspace in Cassandra is a logical namespace for a group of Cassandra Column Families. Just as a relational database contains multiple...
tables, a Cassandra Keyspace contains multiple Column Families. A keyspace typically has a name and a set of attributes to define its behavior as described below when creating a keyspace.

A single Cassandra cluster can contain multiple keyspace. The recommended best practice is to use one keyspace per application so that the applications in a single cluster are logically isolated through their respective keyspaces. However, this practice may have practical limitations when it comes to managing a bulk of applications.

Follow the instructions below to add/delete keyspaces using the Management Console.

1. Log in to the WSO2 Storage Server management console and select Add under **Cassandra Keyspaces** in the **Main** menu.

2. Fill in the details and save.

   **For Simple Strategy:**

For Network Topology Strategy:
The fields in the new keyspace window are described below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the keyspace. This field is mandatory. The name &quot;system&quot; is reserved for Cassandra internals.</td>
</tr>
<tr>
<td>Replication Placement</td>
<td>Replica placement strategy determines how the keyspace data copies are placed in the Cassandra cluster, which nodes carry copies of which keys etc. We have the 'Simple' Strategy, 'Old Network Topology' Strategy and 'Network Topology' Strategy.</td>
</tr>
<tr>
<td>Factor</td>
<td><strong>For Simple Strategy:</strong> Number of replicas (additional copies) of keyspace data stored in a Cassandra cluster. For example, if replication factor is set to 2, then 2 nodes in the cluster will have copies of keyspace data and this replication is transparent to the clients. The replication factor should not exceed the number of nodes in the cluster. <strong>For Network Topology Strategy:</strong> Number of replicas (additional copies) of keyspace data stored in each Data Center in a Cassandra cluster. For example, in a particular Data Center, if replication factor is set to 2, then 2 nodes in that Data Center will have copies of keyspace data and this replication is transparent to the clients. The replication factor should not exceed the number of nodes in that particular Data Center.</td>
</tr>
</tbody>
</table>

3. The added keyspace is listed under **Keyspaces**.
   Storage Server has the system keyspace by default. It is the metadata table used by Cassandra server, and users are not allowed to modify it.

4. You can perform the following operations on the newly-added custom keyspace.

**Keyspaces**

<table>
<thead>
<tr>
<th>Keyspace Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>N/A</td>
</tr>
<tr>
<td>system_auth</td>
<td>![Set Permissions], ![Edit], ![Delete]</td>
</tr>
<tr>
<td>system_traces</td>
<td>![Set Permissions], ![Edit], ![Delete]</td>
</tr>
</tbody>
</table>

- **Set Permissions**
- **Edit**: Takes you back to step 2 above to change keyspace information
- **Delete**: Allows to delete the keyspace. Once executed, this operation cannot be undone

Note that when a cassandra keyspace is created, followed by column families in that keyspace, you can use the tenant user credentials (user@tenant.com) to log in to that cassandra server.
View keyspace information

You can view information about the keyspace by clicking on its name. This displays general information of the keyspace such as name of the cluster, name of the keyspace, replication factor, placement strategy and endpoints.

Keyspace Information (system_auth)

<table>
<thead>
<tr>
<th>Keyspace Details</th>
<th>Endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Name</td>
<td>SSClusteradmin2f78951-f23e-41f9-bc74-03d0204f7f98</td>
</tr>
<tr>
<td>Keyspace Name</td>
<td>system_auth</td>
</tr>
<tr>
<td>Replication Factor</td>
<td>1</td>
</tr>
<tr>
<td>Replica Placement Strategy</td>
<td>Simple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Family Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>Set Permissions Edit Delete</td>
</tr>
</tbody>
</table>

Add New Column Family

Set permissions

Permissions can be provided for the roles available in the keyspace by selecting the relevant checkboxes under Permission.

Permissions for Keyspace : testKey

<table>
<thead>
<tr>
<th>Role</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal/everyone</td>
<td>[ ] Create [ ] Alter [ ] Drop [ ] Select [ ] Modify [ ] Authorize</td>
</tr>
</tbody>
</table>

Save Cancel

For more information on creating user roles, refer to Creating a User Role. The following roles are available by default:

- Admin: Tenant administrator role
- everyone: Default tenant user role

Creating Column Families

A Cassandra Column Family is similar to a table in a relational database management system. However, unlike relational tables, Column Families only define metadata of the columns as described below when creating a Column Family. The actual columns in a row are defined typically by the client application. The actual columns of each row can vary from each other depending on the data stored in a row. For an example of adding data to Cassandra Column Families through a client application, refer to Cassandra Keyspace and Column Family Operations sample.

Note that as a user, you need to be assigned to a role with the relevant authority in order to create or edit column families. The Admin user is by default granted authority to perform this task. See more information...
Follow the steps below to add column families to an existing keyspace. Also see, *Creating Keyspaces*.

1. Click **List** under **Cassandra Keyspaces** and select a keyspace to add column families.

![Cassandra Keyspaces](image)

2. Click **Add a New Column Family** to open the **New Column Family** window where you can enter column family details to the keyspace.

![New Column Family](image)

The fields of the new column family window are described below:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name*</td>
<td>Column Family (CF) Name is mandatory. Must not contain the character &quot;.&quot;.</td>
</tr>
<tr>
<td>Comment</td>
<td>Description of the column family. Used to attach additional human-readable information about the column family to its definition</td>
</tr>
</tbody>
</table>
| Column Type             | • Standard Column Family  
                          | • Super Column Family       |
| Key Validation Class    | Datatype of Row keys                                                      |
| Comparator              | Datatype of Column keys                                                   |
| Default Validation Class| Default datatype of Column values                                          |

3. Click **Save** after adding column family details. The **Keyspace Information** window opens with links to column family operations such as **Set Permissions**, **Edit** or **Delete**.
4. Click on the Column Family's name and then the **Add New Index/Non-Index Column** link to create an indexed or non-indexed column.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the new column</td>
<td>Required</td>
</tr>
<tr>
<td>Index Name</td>
<td>Index Name</td>
<td>Required (only in case of an index column)</td>
</tr>
<tr>
<td>Validation Class</td>
<td>Datatype of column value</td>
<td>Required</td>
</tr>
</tbody>
</table>

There are 2 ways of defining a column.

1) Create a (non-indexed) column with a Validation class: Do not specify any values for 'Index Name'.

2) Create an indexed column with a Validation class: Specify a name for the Index in 'Index Name'.

Note that when a **cassandra keyspace is created**, followed by column families in that keyspace, you can use the tenant user credentials (**user@tenant.com**) to log in to that cassandra server.

**Explore Cassandra Cluster**

The Cassandra explorer provides a simplified interface to explore data, view rows and columns residing in a Cassandra cluster. You can explore internal or external Cassandra keyspaces. Cassandra explorer also provides a comprehensive search mechanism.

Follow the instructions below to connect to and explore a Cassandra cluster.

1. Log in to the management console and click **Connect to Cluster**.

2. Enter the connection details and click **Connect**. You have to provide connection details each time because the Cassandra explorer is designed to connect to several different internal or external clusters.
2. Connection Url*: connection url of the Cassandra cluster with ip and port. e.g., localhost:9160, 10.8.99.101:9161. This field is mandatory.
5. Maximum Result Count: Maximum results retrieved from the Cassandra cluster. The default value is 1000.

3. The connected cluster opens, listing all available keyspaces and their column families.
4. Click on a selected column family to view its rows. You can search for rows with a given name and navigate easily through full-numbered pagination. For example,

Unlike relational databases where each table has a defined set of columns, Cassandra defines column families that only define the metadata of the columns. The actual columns that make up a row are determined by the client application. The actual columns of each row can vary from each other depending on the data stored in a row. Therefore, the above view does not carry the exact column names in table column headers. Instead we have named columns as "Column1", "Column2" etc, while actual column headers are shown inline with the column values themselves.

Data is typically added to these columns through an API or a data service. For example, refer to Cassandra Keyspace and Column Family Operations sample.

5. Click the View more link associated with each row in the column family to navigate to a comprehensive column explorer with facility to search column data across the row.

Cassandra CLI/CQL Commands
After creating a keyspace, column families and columns for data, you can connect to them and perform management operations through the Storage Server management console UI. Additionally you can also connect to Cassandra using clients such as Cassandra-CLI and CQL client.

**Cassandra CLI**

This `cassandra-cli` script is shipped with Storage Server. You can find it at `<SS_HOME>/bin`.

This guide gives you some of the common CLI operations used to connect to and query data in Cassandra clusters.

**CQL**

For this, you can use **Database Explorer** in **Tools**.

1. Before you start using Database Explorer for Cassandra you have to download and copy Cassandra JDBC driver to `<SS_HOME>/repository/components/lib`.
2. Go to **Tools > Database Explorer** and log in with the following information.

   ![Database Explorer Login Form]

   - Driver Class: `org.apache.cassandra.cql.jdbc.CassandraDriver`
   - JDBC URL: `jdbc:cassandra://localhost:9160`
   - User Name: `<User_Name>`
   - Password: `<Password>`

3. The following window appears.
Now you can use CQL to explore Cassandra keyspaces. This guide gives you some of the common CQL commands used to connect to and query data in a Cassandra cluster.

You may not be able to access data of column families which are created through Cassandra-CLI, Carbon UI or any other Thrift clients when using CQL. However, you can do that by making a small change (which is introducing a primary key) to the column family (i.e. CQL table). See here for more information.

Monitor Cassandra Cluster

The Cassandra Column Store Service provides tools for you to monitor and manage a Cassandra cluster through the management console, and to perform the full set of functions in the Cassandra Node Tools API. This feature is fully implemented using Web services, because exposing the JMX externally is a security vulnerability.

Note that this functionality is only applicable if you have Cassandra environments set up in your system using SS embedded Cassandra clusters.

The following sections show how to execute Cassandra operations, view statistics of cluster rings etc.

- Cluster Node Operations
- Monitoring Cluster Statistics
- Monitoring Cassandra Statistics using WSO2 BAM
Cluster Node Operations

When you have a Cassandra cluster with SS embedded Cassandra as the Cassandra nodes, you can perform certain operations on the nodes using the management console of the provisioning Storage Server.

Before you begin, ensure that all the nodes of the cluster are configured in the cluster-config.xml file stored in the <SS_HOME>/repository/conf/etc/ directory as explained here.

You may have multiple SS embedded Cassandra clusters in multiple environments of your system. Please note that the cluster-config.xml file should be updated in one SS node of each Cassandra cluster.

Follow the steps given below.

1. Log in to the WSO2 Storage Server management console.
2. Go to the Tools menu and click Cassandra Operations. The Cluster Nodes screen will open:

   ![Cluster Nodes](image)

   3. Click View Operations associated with each cluster node in the cluster. The Node Operations screen opens:

      ![Node Operations](image)

      It contains all the cluster-node-level, keyspace-level and column-family-level operations, such as backup, flush, scrub, shutting down RPC and Gossip servers etc.

The following topics explains each operation in detail:

- General Cassandra Operations
- Backup Operations
- Server Operations
- Cache Operations
- Other Operations
- Keyspace Operations

General Cassandra Operations
**Decommission**: This operation removes the node from the cluster. If there is only one node in the cluster, it cannot be removed.

**Drain**: This operation flushes the entire node. This causes the node to stop accepting write requests until it is restarted.

**Move**: This operation changes the node’s current token to a new one. This is important when load-balancing a cluster.

**Perform GC (Garbage Collector)**: This operation activates the garbage collector on the node. It affects the overall latency of activities on the node.

### Backup Operations

<table>
<thead>
<tr>
<th>Backup Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Take Snapshot</strong></td>
</tr>
<tr>
<td><strong>Clear Snapshot</strong></td>
</tr>
<tr>
<td><strong>Enable Incremental Backup</strong></td>
</tr>
</tbody>
</table>

**Take Snapshot**: Enables the user to backup data in the node.

**Clear Snapshot**: Clears backups taken by the Take Snapshot operation and frees the memory that was utilized for them.

**Enable Incremental Backup**: Allows to fully backup the data in a node. Also, data will be continuously written to the backup as and when additional data is received by the node, even after the backup is taken.

### Server Operations

<table>
<thead>
<tr>
<th>Server Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stop RPC</strong></td>
</tr>
<tr>
<td><strong>Stop Gossip</strong></td>
</tr>
</tbody>
</table>

**Stop RPC**: Allows to shutdown and restart the RPC server of the node.

**Stop Gossip**: Allows to shutdown and restart the Gossip server of the node.

### Cache Operations

<table>
<thead>
<tr>
<th>Cache Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Invalidate Row Cache</strong></td>
</tr>
<tr>
<td><strong>Set Cache Capacity</strong></td>
</tr>
</tbody>
</table>

**Invalidate Row Cache**

**Set Cache Capacity**

### Other Operations

<table>
<thead>
<tr>
<th>Other Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Remove Token</strong></td>
</tr>
<tr>
<td><strong>Stop Compaction</strong></td>
</tr>
<tr>
<td><strong>Rebuild</strong></td>
</tr>
<tr>
<td><strong>Set Stream Throughput</strong></td>
</tr>
<tr>
<td><strong>Set Compaction Throughput</strong></td>
</tr>
</tbody>
</table>

**Remove Token**: Removes a token in the cluster or forcefully removes completion operations listed under this operation. User gets a live update of the current token removal status in the management console.

**Stop Compaction**: Stops any listed, permitted compaction under this operation.

**Rebuild**: Allows to specify an available data center for re-building.

**Set Stream Throughput**: Set the stream throughput of the node.
**Set Compaction Throughput**: Sets the compaction throughput for the node.

**Keyspace Operations**

Click **Keyspace Operations** at the bottom of the **Node Operations** screen.

The **Keyspace Operations** screen opens with the list of operations associated with each keyspace as follows:

<table>
<thead>
<tr>
<th>Keyspace</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>Repair  Compact</td>
</tr>
<tr>
<td></td>
<td>Flush CleanUp Scrub</td>
</tr>
<tr>
<td>test</td>
<td>Repair  Compact</td>
</tr>
<tr>
<td></td>
<td>Flush CleanUp Scrub</td>
</tr>
</tbody>
</table>

The operations are explained below.

**Repair**: Compares the replicas for each key responsible for the node and all replicas are updated to the newest available version.

**Compact**: Triggers major compaction in all column families of the keyspace and performs compaction of multiple SS tables into a single SS table.

**Flush**: This flushes the memtables (in memory) to SSTables (on disk), which also enables CommitLog segments to be deleted.

**Cleanup**: Prompts Cassandra to cleanup keys, which no longer belong to the node.

**Scrub**: This operation rebuilds sstables with correct bloom filters, with no data loss. The nodetool scrub creates a snapshot of data files before rebuilding.

**UpgradeSSTables**: While the **Scrub** operation rebuilds your sstables, it also discards data it deems broken and creates a snapshot of it, which has to be removed manually. To rebuild your sstables without going through this process, it is best to use UpgradeSSTables operation, which is particularly useful when upgrading your server or changing compression options.

**Take Snapshot**: Makes a backup of the the keyspace data.

**Clear snapshot**: Clears the backup of a keyspace.

**Monitoring Cluster Statistics**

When you have a Cassandra cluster with SS embedded Cassandra as the Cassandra nodes, you can monitor cluster statistics using the management console of the provisioning Storage Server.
Before you begin, ensure that all the nodes of the cluster are configured in the cluster-config.xml file stored in the `<SS_HOME>/repository/conf/etc/` directory as explained here.

You may have multiple SS embedded Cassandra clusters in multiple environments of your system. Please note that the cluster-config.xml file should be updated in one SS node of each Cassandra cluster.

The following instructions explain how to monitor node statistics:

1. Log in to the WSO2 Storage Server management console.
2. Go to the and Monitor menu and click Cassandra Stats. The Cluster Ring screen opens showing cluster-wide, keyspace-level and column-family-level statistics, such as gossip info, column family histograms etc.

![Cluster Ring](image)

The elements of this table are described below.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Node IP</td>
</tr>
<tr>
<td>State</td>
<td>Current state of the node (Normal/Leaving/Joining/Moving)</td>
</tr>
<tr>
<td>Data Center</td>
<td>Data Center which this node belongs to</td>
</tr>
<tr>
<td>Rack</td>
<td>Rack which this node belongs to</td>
</tr>
<tr>
<td>Status</td>
<td>Current status of the node (Up/Down)</td>
</tr>
<tr>
<td>Load</td>
<td>Used disk storage</td>
</tr>
<tr>
<td>Token</td>
<td>Token of the node</td>
</tr>
<tr>
<td>Ownership</td>
<td>How much (%) data in the cluster is owned by this node</td>
</tr>
</tbody>
</table>

3. Click Node to access its node-level statistics. The Node Stats screen opens as follows:

![Node Stats](image)

Different types of statistics on the cluster node can be accessed by clicking on each of the following links:

- Node Info
- Node InfoCF Stats
- Version
Node Info

An example of the Node Info screen is shown below.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>Token of the node</td>
</tr>
<tr>
<td>GossipState</td>
<td>Gossip is enabled or not</td>
</tr>
<tr>
<td>Load</td>
<td>Used disk storage</td>
</tr>
<tr>
<td>Generation No</td>
<td>Times started</td>
</tr>
<tr>
<td>Up time(Seconds)</td>
<td>Server up time</td>
</tr>
<tr>
<td>Head Memory Usage(MB)</td>
<td>Head Memory Usage (MB)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Data center</td>
<td>Data Center which this node belongs to</td>
</tr>
<tr>
<td>Rack</td>
<td>Rack which this node belongs to</td>
</tr>
<tr>
<td>Exception Count</td>
<td>Exception Count</td>
</tr>
</tbody>
</table>

**CF Stats**

Shows the column family information. For example,

The following table describes information shown in the above window.

**Keyspace Information:**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Count</td>
<td>Number of read requests received since server startup</td>
</tr>
<tr>
<td>Read Latency(ms)</td>
<td><strong>total time spent for read requests since server started</strong> divided by number of read requests since server started</td>
</tr>
<tr>
<td>Write Count</td>
<td>Number of write requests received since server startup</td>
</tr>
</tbody>
</table>
Write Latency (ms) | total time spent for write requests since server started divided by number of write requests since server started
---|---
Pending Tasks | Tasks in the queue for reads, writes, and cluster operations of tables in the keyspace.

**Column Family Information:**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS Table Count</td>
<td>Number of SSTables containing data from the table</td>
</tr>
<tr>
<td>Space Used(live)</td>
<td>Total size of all live SS tables</td>
</tr>
<tr>
<td>Space Used(total)</td>
<td>Total size of all SS tables</td>
</tr>
<tr>
<td>Number of Keys (estimate)</td>
<td>Number of keys in the node</td>
</tr>
<tr>
<td>Memtable Column Count</td>
<td>Column Count in Memtable</td>
</tr>
<tr>
<td>Memtable Data Size</td>
<td>Data Size in Memtable</td>
</tr>
<tr>
<td>Memtable Switch Count</td>
<td>How many times a “full” memtable was swapped for an empty one</td>
</tr>
<tr>
<td>Read Count</td>
<td>Number of read requests received since server startup</td>
</tr>
<tr>
<td>Read Latency(ms)</td>
<td>total time spent for read requests since last call to jmx mbean divided by number of read requests since last call to jmx mbean</td>
</tr>
<tr>
<td>Write Count</td>
<td>Number of write requests received since server startup</td>
</tr>
<tr>
<td>Write Latency(ms)</td>
<td>total time spent for write requests since last call to jmx mbean divided by number of write requests since last call to jmx mbean</td>
</tr>
<tr>
<td>Pending Tasks</td>
<td>Approximate number of pending tasks thread pool has.</td>
</tr>
<tr>
<td>Bloom Filter False Positives</td>
<td>Number of false positives, which occur when the bloom filter said the row existed, but it actually did not exist in absolute numbers</td>
</tr>
<tr>
<td>Bloom Filter False Ratio</td>
<td>Fraction of all bloom filter checks resulting in a false positive</td>
</tr>
<tr>
<td>Bloom Filter Space Used</td>
<td>Bytes of bloom filter data</td>
</tr>
<tr>
<td>Compacted Row Minimum Size</td>
<td>Lower size limit in MB for table rows being compacted in memory</td>
</tr>
<tr>
<td>Compacted Row Maximum Size</td>
<td>Upper size limit in bytes for compacted table rows, configurable in the cassandra.yaml (in_memory_compaction_limit_in_mb)</td>
</tr>
<tr>
<td>Compacted Row Mean Size</td>
<td>The average size in bytes of compacted table rows</td>
</tr>
</tbody>
</table>

*Version*
Shows the Cassandra Server version in a pop-up dialog box as follows:

![Cassandra Version: 1.2.10](image)

**Thread Pool Stats**

Shows the thread pool statistics available for a node. Given below are descriptions of the information shown under general and dropped properties.

### General Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Active</th>
<th>Pending</th>
<th>Completed</th>
<th>Blocked</th>
<th>All time blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadStage</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RequestResponseStage</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReadRepairStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MutationStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ReplicateOnWriteStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GossipStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AntiEntropyStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MigrationStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>StreamStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MemtablePostFlusher</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FlushWriter</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MiscStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>commitlog_archiver</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>InternalResponseStage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Handoff</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadStage</td>
<td></td>
</tr>
<tr>
<td>RequestResponseStage</td>
<td></td>
</tr>
<tr>
<td>ReadRepairStage</td>
<td></td>
</tr>
<tr>
<td>MutationStage</td>
<td></td>
</tr>
</tbody>
</table>
ReplicateOnWriteStage
GossipStage
AntiEntropyStage
MigrationStage
StreamStage
MemtablePostFlusher
FlushWriter
MiscStage
commitlog_archiver
InternalResponseStage
HintedHandoff

Dropped Properties

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Dropped</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANGE_SLICE</td>
<td>0</td>
</tr>
<tr>
<td>READ_REPAIR</td>
<td>0</td>
</tr>
<tr>
<td>BINARY</td>
<td>0</td>
</tr>
<tr>
<td>READ</td>
<td>0</td>
</tr>
<tr>
<td>MUTATION</td>
<td>0</td>
</tr>
<tr>
<td>REQUEST_RESPONSE</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANGE_SLICE</td>
<td></td>
</tr>
<tr>
<td>READ_REPAIR</td>
<td></td>
</tr>
<tr>
<td>BINARY</td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td></td>
</tr>
<tr>
<td>MUTATION</td>
<td></td>
</tr>
<tr>
<td>REQUEST_RESPONSE</td>
<td></td>
</tr>
</tbody>
</table>

Compaction Stats

Shows the compaction statistics of the node as shown in the following example.
Compaction Stats- 127.0.0.1

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending Tasks</td>
<td>How many pending compaction tasks are pending</td>
</tr>
<tr>
<td>Active Remaining Time</td>
<td>How much time remaining to complete compact tasks</td>
</tr>
</tbody>
</table>

**Gossip Info**

Shows statistics of the Gossip server as shown in the following example.

**Network Stats**

**Range Key**

Click **Keyspace Stats** in the **Node Stats** screen to view statistics of the key spaces and associated column families.

**Keyspace Statistics**

Shows the available keyspaces with operations as depicted in the example below.
It shows the following statistics for each keyspace.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ownership</td>
<td></td>
</tr>
<tr>
<td>Describe Ring</td>
<td></td>
</tr>
</tbody>
</table>

Click on a selected keyspace to view its column-family statistics.

**Column Family Statistics**

Click on the selected keyspace to open the column family statistics associated with it. The **Column Family Stats** screen opens. For example,

The table below gives information of each type of statistical data that can be viewed by clicking on each link on the screen.

<table>
<thead>
<tr>
<th>Stats Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram</td>
<td></td>
</tr>
<tr>
<td>Endpoints</td>
<td>Shows which node has a given key</td>
</tr>
<tr>
<td>SSTables</td>
<td>Shows which SS table has a given key</td>
</tr>
<tr>
<td>Compaction Thresholds</td>
<td></td>
</tr>
</tbody>
</table>

**Monitoring Cassandra Statistics using WSO2 BAM**

WSO2 BAM is a monitoring tool, which performs the task of aggregating, analyzing and presenting information about business activities. Thereby, BAM can be used with SS in order to effectively monitor Cassandra statistics.

To use BAM with SS for Cassandra monitoring, you must first configure the SS to publish Cassandra information to BAM. In BAM, the cassandra_stat_monitoring_toolbox.tbox toolbox should be installed. With this toolbox, the BAM dashboard will graphically display the statistics published from Storage Server.
Follow the steps given below.

- Enabling the Cassandra statistics publisher in SS
- Setting up the BAM toolbox
- Viewing Cassandra statistics in BAM

**Enabling the Cassandra statistics publisher in SS**

1. Stop the Storage Server if it is already started.
2. To start the Cassandra statistics publisher, open the `cluster-monitor-config.xml` file in the `<SS_HOME>/repository/conf/etc/` directory and set the `<monitoring_enable>` setting to `true` (default setting is `false`).

   You may have multiple SS embedded Cassandra clusters in multiple environments of your system. Please note that this statistics publisher should be enabled in only one SS node of each Cassandra cluster. If you enable the publisher in multiple SS nodes, duplicate data will be published to BAM.

3. Set `<bam_secure_url>` and `<bam_receiver_url>` as well.
4. Provide the cron expression in `<cron_expression>` to schedule the publisher.
5. Provide the BAM admin credentials.
6. Start the Storage Server.

**Setting up the BAM toolbox**

1. Stop the BAM server if it is already started.
2. Open the `master-datasources.xml` in the `<BAM_HOME/repository/conf/datasources/` directory and add the `WSO2SS_STAT_DATASOURCE` data source as shown below. This datasource connects to the default H2 database in BAM. Therefore, the summarized Cassandra information published to BAM from SS will be stored in this database.

   The master-datasources.xml file, consists of the datasource connection to the default, internal database (H2) that is shipped with every WSO2 product. This database maybe replaced as explained in the Working with Databases section. Thereby, the datasource information in the master-datasources.xml file should be changed accordingly.
3. Start the BAM server to install the toolbox.

   Port offset may need to be changed. Open the carbon.xml file from the <BAM_HOME>/repository/conf/ folder and change the <offset>0</offset> to an appropriate value.

4. Go to the Main menu in the navigator and click Add in the BAM Toolbox menu.
5. You can now add the Cassandra statistics monitoring toolbox (cassandra_stat_monitoring_toolbox.tbox), which is stored in the <SS_HOME>/resources/bam-toolboxes/ folder as a custom toolbox. Use the Toolbox from File System option as shown below.

6. Click Install. The toolbox will now be added to BAM.

Viewing Cassandra statistics in BAM

1. In BAM, go to the Dashboard sub menu under the Main navigator menu.
2. Log in to the dashboard. The published and summarized Cassandra statistics are shown in the dashboard under the following types:
Distributed File System Service

WSO2 Distributed File System Service provides Apache HDFS-based, highly-scalable distributed file system management capability. It supports unstructured data (files) and helps users to create and manage a distributed file system with authentication and authorization. The Hadoop file system that comes with WSO2 Storage Server uses the Carbon user base for authentication and authorization.

See the following topics for details:

- Setting up Storage Server with HDFS
- Changing the Default HDFS Configuration
- HDFS Browser
- HDFS CLI Commands

Setting up Storage Server with HDFS

This topic explains the basic procedure that you can follow to set up Storage Server with HDFS.

Pre-requisites

1. Kerberos should be installed on the client and host machines. If not, you will need to install the following in UNIX:
   - `krb5-kdc`
   - `krb5-admin-server`


2. Open a terminal and type the following:
   - `sudo apt-get install krb5-kdc krb5-admin-server`
3. Set the realm as 'WSO2.ORG'.

Starting the Storage Server node

1. Follow the steps given below to create a keytab with the following service principals.
   1. `admin/carbon.super - password: admin`
   2. `datanode/carbon.super - password: node0`
   3. If you are starting a data node, add the data node principal as well.
   4. The keytabs are created.
2. Cache the principle key using the following command:

```
ktutil: addent -password -p <your principle> -k 1 -e <encryption algo>
```

The following is a sample for this:

```
deep@den:$ ktutil
ktutil: addent -password -p admin/carbon.super@WSO2.ORG -k 1 -e des-cbc-md5
Password for admin/carbon.super@WSO2.ORG: admin
ktutil: addent -password -p datanode/carbon.super@WSO2.ORG -k 1 -e des-cbc-md5
Password for datanode/carbon.super@WSO2.ORG: datanode
```

3. Write a keytab for the service principle using the following command:

```
ktutil: write_kt <keytab file name>
```

The following is a sample for this:

```
ktutil: wkt carbon.keytab
```

4. Copy the created keytab file to `[SS_HOME]/repository/conf/etc/hadoop/keytabs/` and rename it to `carbon.keytab`.

5. Start the server with HDFS enabled.

```
./wso2server.sh -enable.hdfs.startup
```

6. Access the Carbon configuration menu and create a new service principal for data nodes with relevant passwords.

When the namenode starts, the user should go to the Carbon console in the namenode and create a service principal for the datanodes. `datanode/carbon.super` should be added, plus any other datanodes the user is willing to start.

7. If your name node is up and HDFS is set up properly, you will notice the following lines in your console.

```
[2013-11-12 16:57:36,561] INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem - fsOwner=admin/node0@WSO2.ORG
```
[2013-11-12 16:57:36,565]  INFO {org.apache.hadoop.hdfs.server.namenode.FSNamesystem} - isAccessTokenEnabled=true accessKeyUpdateInterval=600 min(s), accessTokenLifetime=600 min(s)
[2013-11-12 16:57:36,571]  INFO {org.apache.hadoop.hdfs.server.namenode.FSNamesystem} - Registered FSNamesystemStateMBean and NameNodeMXBean
[2013-11-12 16:57:36,586]  INFO {org.apache.hadoop.hdfs.server.namenode.NameNode} - Caching file names occurring more than 10 times
[2013-11-12 16:57:36,593]  INFO {org.apache.hadoop.hdfs.server.common.Storage} - Number of files = 1
[2013-11-12 16:57:36,596]  INFO {org.apache.hadoop.hdfs.server.common.Storage} - Number of files under construction = 0
[2013-11-12 16:57:37,016]  INFO {org.apache.hadoop.hdfs.server.namenode.NameCache} - initialized with 0 entries 0 lookups
[2013-11-12 16:57:37,023]  INFO {org.apache.hadoop.hdfs.server.namenode.FSNamesystem} - Total number of blocks = 0
[2013-11-12 16:57:37,024]  INFO {org.apache.hadoop.hdfs.server.namenode.FSNamesystem} - Number of invalid blocks = 0
[2013-11-12 16:57:37,024]  INFO {org.apache.hadoop.hdfs.server.namenode.FSNamesystem} - Number of under-replicated blocks = 0
Starting multiple Data nodes pointing to one Namenode

1. Change the following property values in the hdfs-site.xml file to point to the namenode:

   dfs.http.address
   dfs.https.port
   dfs.https.address

2. Change the following properties in the core-site.xml file to point to the namenode:

   fs.default.name
   hadoop.security.group.mapping.service.url

3. Change the following in the hdfs-site.xml file to start the datanode.

   dfs.datanode.address
   dfs.datanode.https.address
   dfs.datanode.http.address
   dfs.datanode.ipc.address
   dfs.replication

4. Add the datanode IP and port to the slaves, one per line.
5. When starting multiple datanodes in the same machine, make sure you change the PID_DIR and the IDENT_STRING for the data node in the hadoop-env.sh file.

   • If starting as a secure datanode, then add the following line:

   ```
   # The directory where pid files are stored for secured datanode
   export HADOOP_SECURE_DN_PID_DIR=/tmp/2
   ```

   • Alternatively, add the following:

   ```
   # A string representing this instance of hadoop. $USER by default.
   export HADOOP_IDENT_STRING=$USER_02
   ```

Changing the Default HDFS Configuration
WSO2 Storage Server ships a pre-configured HDFS file system. In the default configuration, HDFS is disabled and the user has to enable it by disabling the system variable named `disable.hdfs.startup` in the startup script (wso2server.sh in Linux or wso2server.bat in Windows), which is stored in the `<SS_HOME>/bin/` directory.

```
enable.hdfs.startup=true
```

**HDFS Name Node**

To start an HDFS name node, open a command prompt (in Windows) or a text Linux console (in Linux) and go to the `<SS_HOME>/bin` directory and execute the startup script, i.e., `wso2server.bat` (in Windows) or `wso2server.sh` (in Linux).

**HDFS Data Node**

Do not format a running Hadoop file system as it will cause all your data to be erased. Before formatting, ensure that the `dfs.name.dir` directory exists. The name node can be formatted by running the following command from the `<SS_HOME>/bin` directory.

```
./hadoop namenode -format
```

To start the HDFS data node, change the directory to `$SS_HOME`.

```
HADOOP_SECURE_DN_USER=<user> sudo -E bin/hadoop datanode
```

The user should have sudo privileges.

**HDFS Browser**

The HDFS Browser provides the facility of viewing your HDFS Instance in a tree view, and a detailed view. HDFS instance is mapped to the tree view and the detailed view, which lists out the items in a path.

**Tree View**

This view shows the files and folders of a given path in the HDFS Instance of the user, in a tree structure.

HDFS Explorer

```
Location: /
```

```
Go
```

Tree view

```
Detail view
```

```
/
```

```
test
```

```
user
```

**Detail View**
This view shows the files and folders of a given path in the HDFS instance of the user, listed. The view is available in the following occasions:

1. When a user clicks on a tree node from the tree view.
2. When a user navigates to the Detail view.
3. Searching for a file path using the Location field.

Navigation via Location Bar

The location bar allows a user to navigate the tree and directs the user to the detailed view of the entered path.

Folder Management

HDFS browser provides functionalities to create/rename/delete folders so that a user can organize the HDFS instance easily.

Create Folder
Except for the super tenant admin, who is the owner of the file system, no other user can create a folder in the root folder path.

1. Go to the detail view of the parent folder.
2. Expand the actions.
3. Click **Create**.
4. Give a unique name and enter.

**Rename Folder**

1. Go to the detail view of the parent folder.
2. Expand the actions.
3. Click **Rename**.
4. Give a unique name and press enter.

**Delete Folder**

1. Go to the detail view of the parent folder.
2. Expand the actions.
3. Click **Delete**.
4. Confirm the action.
File Management

HDFS browser provides functionalities to rename/delete files so that a user can organize the HDFS instance easily. The user is given the facility to upload and download files to and from folders of choice.

Rename File

1. Go to the detail view of the parent folder.
2. Expand the actions.
3. Click Rename.
4. Give a unique name and enter.
Delete File

1. Go to the detail view of the parent folder.
2. Expand the actions.
3. Click **Delete**.
4. Confirm the action.

File Upload/Download

A user is given the facility to upload and download files to folders of a HDFS Instance via the HDFS Browser.

**File Upload**

1. Go to the detail view of the parent folder.
2. Expand the actions.
3. Click **Upload**.
4. Select the file location.
5. Give a name to the file.
6. Click Add.

HDFS Explorer

Location: /  Go

Name

File: 
Name:
Add Cancel

Home > Manage > Hadoop file System > HDFS Explorer

File Download

1. Go to the detail view of the parent folder.
2. Expand the actions of the file to download.
3. Click Download.
4. Confirm action.

HDFS CLI Commands

After creating a Hadoop file system, you can connect to it, explore and perform data management operations. Some of the common command-line operations used to connect to, explore and manage a Hadoop file system is explained here.

WSO2 Storage server ships a script to support Hadoop CLI commands. Also user has to install Kerberos tools to cache the Kerberos ticket from the KDC server hosted within Storage Server.

1. Cache the user kerberos ticket to the system:
if super tenant $kinit <username>/<domain Name>
if user or tenant user or tenant admin $kinit <username>_<domain Name>

2. List HDFS directories:
   • To list directories in the user's home directory, navigate to the user's home directory by changing the directory to <SS_HOME> created by admin. For example, if for tenant user tuser, /user/test.test_tuser. The home directory for a user is created in /user directory.
   • List directories: $bin/hadoop dfs -lsr /user/user1

3. Create HDFS directory: $bin/hadoop dfs -mkdir /user/user1/wso2.
Admin Guide

The following topics explore various product administration scenarios and other topics useful for system administrators.

- User Management
- Feature Management
- Registry Management
- Working with Databases
- Working with Security
- Logging Management
- Monitoring the Server
- Configuring Multiple Tenants
- Performance Tuning
- Calling Admin Service from Apps
- Customizing the Management Console
- Service Principal Management
- Cassandra Migration Guide
- WSO2 Patch Application Process

User Management

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `ser-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. For more information on user management, see the following topics:

- Introduction to User Management
- Configuring User Management
- Managing Users, Roles and Permissions

Screenshots in this section are taken using the WSO2 Carbon product. They may vary depending on the product and the configuration options you are using.

Introduction to User Management

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `ser-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. The following documentation introduces the main concepts in User Management, such as users, roles, permissions, user stores etc. and how they are used in WSO2 products.

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 log-in statuses, the privileges of each user, and their activity in the system, enabling 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, manual/automatic log-out, and resetting user 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, security-conscious 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 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 log-in name, password, first name, last name, and e-mail address.
- **RDBMS (for Authentication and Authorization)**: This RDBMS stores information of 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.
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.

Related Topics

- **Configuring User Management**: This topic explains how you can set up and configure the user management realm.
- **Managing Users, Roles and Permissions**: This topic explains how you can manage the Users, Roles and Permissions using the management console.

### Configuring User Management

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:
See the following topics for instructions:

- Configuring User Stores
- Configuring the System Administrator
- Configuring the Authorization Manager

Configuring User Stores

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 topic explains the different types of user stores that can be configured for a WSO2 product.

The user management feature in WSO2 products allows you to maintain multiple user stores for your system that are used to store the users and their roles. You can set up any of the following types of user stores:

- JDBC user stores
- Active Directory user stores
- Read-Only LDAP user stores
- Read-Write LDAP user stores

The Primary User Store in every WSO2 product is configured in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file. By default, the embedded H2 database (JDBC) that is shipped with WSO2 products is configured as the primary user store, except for WSO2 Identity Server, which has an embedded LDAP as its primary user store. You can change the default configuration by replacing the default database according to your requirement. The primary user store is shared among all the tenants in the system.

With the user management feature, any number of Secondary User Stores can be easily set up for your system using the management console. This will automatically create an XML file with the configurations corresponding to the secondary user store 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. See the section on configuring the authorization manager for information on how to set up the RDBMS for storing permissions.

**Related Topics**

See the following topics for instructions on setting up user stores:

- Configuring Primary User Stores
- Configuring Secondary User Stores
- Working with Properties of User Stores

See the topic on configuring the Authorization Manager for instructions on how to set up a repository for storing permissions.

**Configuring Primary User Stores**

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. This documentation explains the process of setting up a primary user store for your system.

### The default User Store

The primary user store that is configured by default in every WSO2 product is a JDBC user store, which reads/writes into the internal database of the product server. By default, the internal database is H2 (except for WSO2 IS, which uses an LDAP as the default user store). This database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles).

Instead of using the embedded database, you can set up a separate repository and configure it as your primary user store. Since the user store you want to connect to might have different schemas from the ones available in the embedded user store, it needs to go through an adaptation process. WSO2 products provide the following adapters, for connecting to LDAP, Active Directory and JDBC. Thereby, these adapters enable you to authenticate users from different types of user stores.

<table>
<thead>
<tr>
<th>User store manager class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</code></td>
<td>Use <code>ReadOnlyLDAPUserStoreManager</code> for external LDAP user stores.</td>
</tr>
<tr>
<td><code>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</code></td>
<td>Use <code>ReadWriteLDAPUserStoreManager</code> both read and write operations uncommented in the code in</td>
</tr>
</tbody>
</table>
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager

Use ActiveDirectoryUserStoreManager to configure an Active Directory Domain Service (AD DS) or Active Directory Lightweight Directory Service (AD LDS). This can be used only read-only, you must use org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager.

org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager

Use JDBCUserStoreManager for both internal and external JDBC user stores. This is the user store configuration which is uncommented in the code in the user-mgt.xml file for all WSO2 products, except WSO2 Identity Server (which uses the ReadWriteLDAPUserStoreManager).

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-Write Active Directory User Store
- Configuring a Read-Only LDAP/AD 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 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 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.

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 as instructed in the related documentation.

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:

   ```xml
   <UserStoreManager
       class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
   ```

2. Specify the connection to the RDBMS inside the JDBC user store manager according to your requirement. For more information about user store properties, see the related documentation.

   - Internal JDBC User Store
   - External JDBC User Store

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

```xml
<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
    <Property name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
    <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">^[^!#;\^*+={}\\|\&lt;&gt;:,\'\"]{5,30}$</Property>
    <Property name="UsernameJavaRegEx">^[^!#;\^*+={}\\|\&lt;&gt;:,\'\"]{3,30}$</Property>
    <Property name="RolenameJavaRegEx">^[^!#;\^*+={}\\|\&lt;&gt;:,\'\"]{3,30}$</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
</UserStoreManager>
```
<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="SCIMEnabled">false</Property>
<Property name="SelectUserSQL">SELECT * FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="GetRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_TENANT_ID=? AND UM_SHARED_ROLE='0' ORDER BY UM_ROLE_NAME</Property>
<Property name="GetSharedRoleListSQL">SELECT UM_ROLE_NAME, UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ? AND UM_SHARED_ROLE='1' ORDER BY UM_ROLE_NAME</Property>
<Property name="UserFilterSQL">SELECT UM_USER_NAME FROM UM_USER WHERE UM_USER_NAME LIKE ? AND UM_TENANT_ID=? ORDER BY UM_USER_NAME</Property>
<Property name="UserSharedRoleSQL">SELECT UM_ROLE_NAME, UM_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_USER.UM_TENANT_ID = ?</Property>
<Property name="IsRoleExistingSQL">SELECT UM_ID FROM UM_ROLE WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="GetUserListOfSharedRoleSQL">SELECT UM_USER_NAME FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE UM_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="GetPropertyForProfileSQL">SELECT


(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=?) 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="RemoveRoleFromUserSQL">DELETE FROM UM_USER_ROLE

<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="UpdateRoleNameSQL">UPDATE UM_ROLE set UM_ROLE_NAME=? WHERE UM_ROLE_NAME = ? AND UM_TENANT_ID=?</Property>

<Property name="AddUserPropertySQL">INSERT INTO UM_USER_ATTRIBUTE (UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) VALUES ((SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), ?, ?, ?, ?)</Property>


<Property name="UserNameUniqueAcrossTenantsSQL">SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=?</Property>

<Property name="IsDomainExistingSQL">SELECT UM_DOMAIN_ID FROM UM_DOMAIN WHERE UM_DOMAIN_NAME=? AND UM_TENANT_ID=?</Property>

<Property name="AddDomainSQL">INSERT INTO UM_DOMAIN (UM_DOMAIN_NAME, UM_TENANT_ID) VALUES (?, ?)</Property>

<Property name="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="AddUserRoleAttributeSQL-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>

<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>
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/MessageDigest.html](http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html).

4. If you are setting up an external JDBC user store, you need to set the following property to ‘true’ to be able to create roles in the primary user store.

```
<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. Refer the related topics for information about 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> 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. Refer the related topics for detailed information on setting up databases and configuring datasources.

1. 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-datasource.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.

Related Topics

- **Setting Up the Physical Database**: This section explains how you can set up a new RDBMS and configure it for your system.
- **Managing Datasources**: This section explains the purpose of defining datasources and how they are configured for a product.
- **Configuring the System Administrator**: This section provides information about the system administrator user.
- **Properties of Primary User Stores**: This section describes each of the properties used in the user-mgt.xml file for configuring the primary user store.

**Configuring a Read-Write Active Directory 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 topic provides instructions on how to configure an external Active Directory as the primary user store for the WSO2 server, by changing the default configuration given in this file.

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

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.
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 about user store properties, see the related documentation.

```
<UserStoreManager
  class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
  <Property
    name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
  <Property
    name="defaultRealmName">WSO2.ORG</Property>
  <Property
    name="Disabled">false</Property>
  <Property
    name="kdcEnabled">false</Property>
  <Property
    name="ConnectionURL">ldaps://10.100.1.100:636</Property>
  <Property
    name="ConnectionName">CN=admin,CN=Users,DC=WSO2,DC=Com</Property>
  <Property
    name="ConnectionPassword">A1b2c3d4</Property>
  <Property
    name="PasswordHashMethod">PLAIN_TEXT</Property>
  <Property
    name="UserSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
  <Property
    name="UserEntryObjectClass">user</Property>
  <Property
    name="UserNameAttribute">cn</Property>
  <Property
    name="isADLDSRole">false</Property>
  <Property
    name="UserNameListFilter">(objectClass=user)</Property>
  <Property
    name="UserNameSearchFilter">(&amp;{objectClass=user}(cn=?))</Property>
  <Property
    name="UsernameJavaRegEx">[a-zA-Z0-9._-]{3,30}$</Property>
  <Property
    name="UsernameJavaScriptRegEx">^[\S]{3,30}$</Property>
  <Property
    name="PasswordJavaScriptRegEx">^[\S]{5,30}$</Property>
  <Property
    name="RolenameJavaScriptRegEx">^[\S]{3,30}$</Property>
  <Property
    name="RolenameJavaRegEx">[a-zA-Z0-9._-]{3,30}$</Property>
  <Property
    name="ReadGroups">true</Property>
  <Property
    name="WriteGroups">true</Property>
  <Property
    name="EmptyRolesAllowed">true</Property>
  <Property
    name="GroupSearchBase">CN=Users,DC=WSO2,DC=Com</Property>
  <Property
    name="GroupEntryObjectClass">group</Property>
  <Property
    name="GroupNameAttribute">cn</Property>
</UserStoreManager>
```
<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)</Property>
<Property name="SharedTenantNameAttribute">ou</Property>
<Property name="SharedTenantObjectClass">organizationalUnit</Property>
<Property name="MembershipAttribute">member</Property>
<Property name="GroupNameListFilter">(objectcategory=group)</Property>
<Property name="GroupNameSearchFilter">(&amp;(objectClass=group)(cn=?))</Property>
<Property name="UserRolesCacheEnabled">true</Property>
<Property name="Referral">follow</Property>
<Property name="BackLinksEnabled">true</Property>
<Property name="MaxRoleNameListLength">100</Property>
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.

```xml
<AuthorizationManager
    class="org.wso2.carbon.user.core.authorization.JDBCAuthorizationManager">
    <Property
        name="AdminRoleManagementPermissions"/>
    <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 permissions 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. See the topic on configuring keystores for information on how to add certificates to the trust-store.

```xml
<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.
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.

```xml
<Property name="ReadGroups">true</Property>
```

- Set the GroupSearchBase property to the directory name where the Roles are stored. That is, the roles you create using the management console of your product will be stored in this directory location. Also, when LDAP searches for users, it will start from this location of the directory. For example:

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

- Set the GroupSearchFilter and GroupNameAttributes. For example:

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

- Set the MembershipAttribute property as shown below:

```xml
<Property name="MembershipAttribute">member</Property>
```

To read roles based on a backlink attribute, use the following code snippet instead of the above:
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. See the related documentation for more information about the system administrator.

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 <AddUser> 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.

Related Topics

- Configuring the System Administrator: This section provides information about the system administrator user.
- Working with Security: This section explains about configuring security, including keystores.
Properties of Primary User Stores: This section describes each of the properties used in the user-mgt.xml file for configuring the primary user store.

Configuring a Read-Only LDAP/AD 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 and read-write LDAP/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.

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:

1. Setting up the read-only LDAP/AD user store manager
2. Updating the system administrator
3. Starting the server

Step 1: Setting up the read-only LDAP/AD user store manager

Uncomment the following user store in the 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.

Step 2: Updating the system administrator

Step 3: Starting the server

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 of the user store properties, see the related documentation.

```
<UserManager>
<Realm>
```
...<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">
    <Property name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
    <Property name="ReadOnly">true</Property>
    <Property name="Disabled">false</Property>
    <Property name="MaxUserNameListLength">100</Property>
    <Property name="ConnectionURL">ldap://localhost:10389\</Property>
    <Property name="ConnectionName">uid=admin,ou=system</Property>
    <Property name="ConnectionPassword">admin</Property>
    <Property name="PasswordHashMethod">PLAIN_TEXT</Property>
    <Property name="UserNameSearchBase">ou=system</Property>
    <Property name="UserNameListFilter">(objectClass=person)</Property>
    <Property name="UserNameSearchFilter">(&(objectClass=person)(uid=?))</Property>
    <Property name="UserNameAttribute">uid</Property>
    <Property name="ReadGroups">true</Property>
    <Property name="GroupSearchBase">ou=system</Property>
    <Property name="GroupNameListFilter">(objectClass=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="SharedGroupNameListFilter">(objectClass=groupOfNames)</Property>
    <Property name="SharedTenantNameListFilter">(objectClass=organizationalUnit)</Property>
    <Property name="SharedTenantNameAttribute">ou</Property>
    <Property name="SharedTenantObjectClass">organizationalUnit</Property>
    <Property name="MembershipAttribute">member</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="ReplaceEscapeCharactersAtUserLogin">true</Property>
    <Property name="MaxRoleNameListLength">100</Property>
    <Property name="SCIMEnabled">false</Property>
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:
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.

  ```xml
  <Property name="ReadGroups">true</Property>
  ```

- Set the `GroupSearchBase` property to the directory name where the Roles are stored. That is, the roles you create using the management console of your product will be stored in this directory location. Also, when LDAP searches for users, it will start from this location of the directory. For example:

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

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

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

- Set the `MembershipAttribute` property as shown below:

  ```xml
  <Property name="MembershipAttribute">member</Property>
  ```

To read roles based on a backlink attribute, use the following code snippet instead of the above:

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

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.

See the related documentation for more information about the system administrator.

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.

Related Topics
• Configuring the System Administrator: This section provides information about the system administrator user.
• Properties of Primary User Stores: This section describes each of the properties used in the user-mgt.xml file for configuring the primary user store.

**Configuring a Read-Write LDAP User Store**

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the user-mgt.xml file found in the <PRODUCT_HOME>/repository/conf/ directory. This file is shipped with user store manager configurations for all possible user store types (JDBC, read-only LDAP/Active Directory, read-write LDAP and read-write Active directory). The instructions given below explains how to configure a read-write LDAP as the primary user store for the WSO2 server.

![The default User Store](image)

The primary user store that is configured by default in the user-mgt.xml file of WSO2 products is a JDBC user store, which reads/writes into the internal database of the product server. By default, the internal database is H2. This database is used by the Authorization Manager (for user authentication information) as well as the User Store Manager (for defining users and roles). In the case of the WSO2 Identity Server, the default user store is an LDAP (Apache DS) that is shipped with the product.

Note that the RDBMS used in the default configuration can remain as the database used for storing Authorization information.

Follow the given steps to configure a read-write LDAP as the primary user store:

• Step 1: Setting up the read-write LDAP user store manager
• Step 2: Updating the system administrator
• Step 3: Starting the server

**Step 1: Setting up the read-write LDAP user store manager**

Once the above points are made note of and completed, you can start configuring your external read-write LDAP as the primary user store.

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](image)

   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 user store properties, see the related documentation.

```xml
<UserStoreManager
    class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
    <Property
        name="TenantManager" org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
    <Property
        name="ConnectionURL" ldap://localhost:${Ports.EmbeddedLDAP.LDAPServerPort}
    </Property>
    <Property
        name="ConnectionName" uid=admin,ou=system</Property>
    <Property
        name="ConnectionPassword" admin</Property>
    <Property
        name="PasswordHashMethod" SHA</Property>
    <Property
        name="UserNameListFilter" (objectClass=person)</Property>
    <Property
        name="UserEntryObjectClass" wso2Person</Property>
    <Property
        name="UserSearchBase" ou=Users,dc=wso2,dc=org</Property>
    <Property
        name="UserNameSearchFilter" (&objectClass=person)(uid=?)</Property>
    <Property
        name="UserNameAttribute" uid</Property>
    <Property
        name="PasswordJavaScriptRegEx" \{\S\}{5,30}</Property>
    <Property
        name="UsernameJavaScriptRegEx" \{\S\}{3,30}</Property>
    <Property
        name="UsernameJavaRegEx" ^[^~!@#$;%^*+={}\|\<>,\''\"]{3,30}$</Property>
    <Property
        name="RolenameJavaScriptRegEx" \{\S\}{3,30}</Property>
    <Property
        name="RolenameJavaRegEx" ^[^~!@#$;%^*+={}\|\<>,\''\"]{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="GroupNameObjectClass" 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="SharedGroupNameListFilter" (object=organizationalUnit)</Property>
    </UserStoreManager>
```
<Property name="SharedTenantNameAttribute">ou</Property>
<Property name="SharedTenantObjectClass">organizationalUnit</Property>
<Property name="MembershipAttribute">member</Property>
<Property name="UserRolesCacheEnabled">true</Property>
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>
```

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>
```

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.LDAPS erverPort}</Property>
<Property name="ConnectionName">uid=admin,ou=system</Property>
<Property name="ConnectionPassword">admin</Property>
```

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:
Set the GroupSearchFilter and GroupNameAttributes. For example:

```
<Property name="GroupSearchBase">ou=system,CN=Users,DC=wso2,DC=test</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. See the related documentation for information about 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.
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.

Related Topics

- **Configuring the System Administrator**: This section provides information about the system administrator user.
- **Properties of Primary User Stores**: This section describes each of the properties used in the user-mgt.xml file for configuring the primary user store.

**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.
2. The **User Store Management** page opens. Initially, there are no secondary user stores.

   **Note:** You cannot update the PRIMARY user store at run time, so it is not visible on this page.

3. Click **Add Secondary User Store**.

4. In the User Store Manager Class list, select the type of user store you are creating:

<table>
<thead>
<tr>
<th>User store manager</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager</td>
<td>Use <strong>ReadOnlyLDAPUserManager</strong> user stores.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager</td>
<td>Use <strong>ReadWriteLDAPUserManager</strong> read and write operations.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager</td>
<td>Use <strong>ActiveDirectoryUserStoreManager</strong> for AD DS or AD LDS. This can be used for read/write operations. If you need to use AD as only read-only, you must use <strong>ReadOnlyLDAPUserStoreManager</strong>.</td>
</tr>
<tr>
<td>org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager</td>
<td>Use <strong>JDBCUserStoreManager</strong> can be configured for property: <code>&lt;Property&gt;</code></td>
</tr>
</tbody>
</table>

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.

5. Enter a unique domain name with no underscore (_) characters, and optionally enter a description for this user store.

6. 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. See the related topics for descriptions of user store properties.
7. Ensure that all the mandatory fields are filled and a valid domain name is given and click **Add**.

8. A message appears saying that the user stores are being added.

   ![User stores are being updated. Refresh the page after few seconds to check the new status.](image)

   **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.

9. Refresh the page after a few seconds to check the status.

10. If the new user store is successfully added, it will appear in the **User Store Management** page.

11. 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_HOM`
If it is a general tenant, save the configuration in `<PRODUCT_HOME>/repository/tenants/<tenantid>/userstores` directory.

The secondary user store configuration file must have the same name as the domain with an underscore (_) in place of the period. For example, if the domain is `wso2.com`, name the file as `wso2_com.xml`.

One file only contains the definition for one user store domain.

Related topics

- **Properties of User Stores**: This topic explains the properties that are used when defining user stores.

**Working with Properties of User Stores**

The following table provides descriptions of the key properties you use to configure primary user stores.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxUserNameListLength</td>
<td>Controls the number of users listed in the user store of a WSO2 product. Setting this property to 0 displays all users.</td>
</tr>
<tr>
<td>ConnectionURL</td>
<td>Connection URL to the user store server. In the case of default LDAP in Carbon, the port is specified in the <code>carbon.xml</code> file, and a reference to that port is included in this configuration.</td>
</tr>
<tr>
<td>ConnectionName</td>
<td>The username used to connect to the database and perform various operations. This user does not have to be an administrator in the user store or have an administrator role in the WSO2 product that you are using, but this user MUST have permissions to read the user list and users' attributes and to perform search operations on the user store. The value you specify is used as the DN (Distinguish Name) attribute of the user. This property is mandatory.</td>
</tr>
<tr>
<td>ConnectionPassword</td>
<td>Password for the <code>ConnectionName</code> user.</td>
</tr>
<tr>
<td>DisplayNameAttribute</td>
<td>This is an optional property. The Display Name Attribute is the name by which users will be listed when you search for users in the management console (Configuration -&gt; Users tab).</td>
</tr>
<tr>
<td>PasswordHashMethod</td>
<td>Password hash method to use when storing user entries in the user store.</td>
</tr>
<tr>
<td>UserNameListFilter</td>
<td>Filtering criteria for listing all the user entries in the user store. The search operation only provides the objects created from the specified class.</td>
</tr>
<tr>
<td>UserEntryObjectClass</td>
<td>Object class used to construct user entries. By default, it is a custom object class defined with the name <code>wso2Person</code>.</td>
</tr>
<tr>
<td>UserSearchBase</td>
<td>DN of the context or object under which the user entries are stored in the user store. When the user store searches for users, it will start from this location of the directory. Different databases have different search bases.</td>
</tr>
<tr>
<td>UserNameSearchFilter</td>
<td>Filtering criteria used to search for a particular user entry.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UserNameAttribute</td>
<td>The attribute used for uniquely identifying a user entry. Users can be authenticated using their email address, UID, etc. The name of the attribute is considered as the username.</td>
</tr>
<tr>
<td>UsernameWithEmailJavaScriptRegEx</td>
<td>This property defines the JavaScript regular expression pattern when you need to support both email as a user name and normal user names. You can set this property as follows.</td>
</tr>
<tr>
<td>PasswordJavaScriptRegEx</td>
<td>Policy that defines the password format.</td>
</tr>
<tr>
<td>UsernameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for username validation.</td>
</tr>
<tr>
<td>UsernameJavaRegEx</td>
<td>A regular expression to validate usernames. By default, strings have a length of 5 to 30. Only non-empty characters are allowed. You can provide ranges of alphabets, numbers and also ranges of ASCII values in the RegEx.</td>
</tr>
<tr>
<td>RolenameJavaScriptRegEx</td>
<td>The regular expression used by the front-end components for role name validation.</td>
</tr>
<tr>
<td>RolenameJavaRegEx</td>
<td>A regular expression used to validate role names. By default, strings have a length of 5 to 30. Only non-empty characters are allowed.</td>
</tr>
<tr>
<td>ReadGroups</td>
<td>Specifies whether groups should be read from the user store. If this is disabled by setting it to false, none of the groups in the user store can be read, and the following group configurations are NOT mandatory: GroupSearchBase, GroupSearchFilter, GroupNameListFilter.</td>
</tr>
<tr>
<td>WriteGroups</td>
<td>Specifies whether groups should be written to user store.</td>
</tr>
<tr>
<td>EmptyRolesAllowed</td>
<td>Specifies whether the underlying user store allows empty groups to be created. Usually LDAP servers do not allow you to create empty groups.</td>
</tr>
<tr>
<td>GroupSearchBase</td>
<td>DN of the context under which user entries are stored in the user store.</td>
</tr>
<tr>
<td>GroupSearchFilter</td>
<td>The query used to search for groups.</td>
</tr>
<tr>
<td>GroupNameListFilter</td>
<td>Filtering criteria for listing all the group entries in the user store. Groups are created in LDAP using the &quot;groupOfName&quot; class. The group search operation only returns objects created from this class.</td>
</tr>
<tr>
<td>GroupEntryObjectClass</td>
<td>Object class used to construct group entries.</td>
</tr>
<tr>
<td>GroupNameSearchFilter</td>
<td>Filtering criteria used to search for a particular group entry.</td>
</tr>
<tr>
<td>GroupNameAttribute</td>
<td>Attribute used for uniquely identifying a user entry. This attribute is the name of the attribute.</td>
</tr>
<tr>
<td>MembershipAttribute</td>
<td>Attribute used to define members of groups.</td>
</tr>
<tr>
<td>UserRolesCacheEnabled</td>
<td>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.</td>
</tr>
</tbody>
</table>
UserDNPattern

(LDAP) The pattern for the user's DN, which can be defined to improve search performance as the LDAP does not have to travel through the entire tree to find users.

ReplaceEscapeCharactersAtUserLogin

(LDAP) If the user name has special characters it replaces it to validate the user logging in. Only \" and \\ are identified as escape characters.

TenantManager

Includes the location of the tenant manager.

ReadOnly

(LDAP and JDBC) Indicates whether the user store of this realm operates in read-only mode or not.

IsEmailUserName

(JDBC) Indicates whether the user’s email is used as their username (applies when the realm operates in read-only mode).

DomainCalculation

(JDBC) Can be either default or custom (applies when the realm operates in read-only mode).

PasswordDigest

(JDBC) Digesting algorithm of the password. Has values such as, F...Find more.

StoreSaltedPassword

(JDBC) Indicates whether to salt the password.

UserNameUniqueAcrossTenants

(JDBC) An attribute used for multi-tenancy.

PasswordJavaRegEx

(LDAP and JDBC) A regular expression to validate passwords. By default, strings having a length between 5 to 30 with non-empty characters are allowed.

PasswordJavaScriptRegEx

The regular expression used by the front-end components for password validation.

UsernameJavaRegEx

A regular expression to validate usernames. By default, strings having a length of 5 to 30 with non-empty characters are allowed.

UsernameJavaScriptRegEx

The regular expression used by the front-end components for username validation.

RolenameJavaRegEx

A regular expression to validate role names. By default, strings having a length of 5 to 30 with non-empty characters are allowed.

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 different types of realms for tenants.

SharedGroupEnabled

This property is used to enable/disable the shared role functionality.

SharedGroupSearchBase

Shared roles are created for other tenants to access under the mentioned DN.

SharedTenantObjectClass

Object class for the shared groups created.

SharedTenantNameAttribute

Name attribute for the shared group.

SharedTenantNameListFilter

This is currently not used.

Configuring the System Administrator

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the user-mgt.xml file found in the <PRODUCT_HOME>/repository/conf/ directory. This documentation explains...Find more.
the main settings relevant to the system administrator.

The **admin** user is the super tenant that will be able to manage all other users, roles and permissions in the system by using the management console of the product. Therefore, the user that should have admin permissions is required to be stored in the primary user store when you start the system for the first time. The documentation on setting up primary user stores will explain how to configure the administrator while configuring the user store. The information under this topic will explain the main configurations that are relevant to setting up the system administrator.

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

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

**Before you begin:**

Ensure that you have a primary user store (for storing users and roles) and an RDBMS (for storing information related to permissions). See the related documentation for instructions on how to set up these repositories.

**Updating the administrator**

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

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

Note the following regarding the configuration above.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;AddAdmin&gt;true</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>&lt;AdminRole&gt;admin</code></td>
<td>The admin user and role is always stored in the primary user store in your system.</td>
</tr>
<tr>
<td><code>&lt;AdminUser&gt;</code></td>
<td>An administrator is configured for your system by default. This <strong>admin</strong> user is assigned to the <strong>admin</strong> role, which has all permissions enabled.</td>
</tr>
<tr>
<td><code>&lt;EveryOneRoleName&gt;everyone</code></td>
<td>The permissions assigned to the default <strong>admin</strong> role cannot be modified.</td>
</tr>
</tbody>
</table>
### <AddAdmin>

When `true`, this element creates the admin user based on the `adminUser` element. It also indicates whether to create the specified admin user if it doesn't already exist. When connecting to an external read-only LDAP or Active Directory user store, this property needs to be `false` if an admin user and admin role exist within the user store. If the admin user and admin role do not exist in the user store, this value should be `true`, so that the role is added to the user management database. However, if the admin user is not there in the user store, we must add that user to the user store manually. If the `AddAdmin` value is set to `true` in this case, it will generate an exception.

### <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 `<AdminUser>` element. If the external user store is in read/write mode, and you set `<Add Admin>` to `true`, 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>

Do NOT put the password here but leave the default value. If the user store is read-only, this element and its value are ignored. This password is used only if the user store is read-write and the `AddAdmin` value is set to `true`.

Note that the password in the `user-mgt.xml` file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the `user-mgt.xml` file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the `user-mgt.xml` file. To change the admin password, you must use the Change Password option from the management console. See the related topics for instructions.

### <EveryOneRoleName>

The name of the "everyone" role. All users in the system belong to this role.

---

**Related Topics**
Configuring the Authorization Manager

User management functionality is provided by default in all WSO2 Carbon-based products and is configured in the `user-mgt.xml` file found in the `<PRODUCT_HOME>/repository/conf/` directory. This documentation explains how to set up a repository for storing authorization information (role-based permissions) and how to change the relevant configurations.

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. See the related topics for information on how user stores are configured.

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.

Refer the related topics for detailed information on setting up databases and configuring datasources.

### Step 2: Updating the user realm configurations

Once you have set up a new RDBMS and configured the datasource, the `user-mgt.xml` file (user realm configuration) should be updated as explained below.

**Setting up the database connection**

Update the datasource information using the `<Property>` element under `<Configuration>`. Given below are the properties that are set by default.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataSource</td>
<td>This is the jndi name of the datasource that is used for referring to the datasource in the following example, the jndi name of the default datasource defined in the &lt;PRODUCT_HOME&gt;/repository/conf/datasources/master-datasources.xml file linked from the user-mgt.xml file.</td>
</tr>
<tr>
<td>isCascadeDeleteEnabled</td>
<td>This property is set to 'true' by default, which enables cascade delete for the UM_USER_PERMISSION and UM_ROLE_PERMISSION tables when a permission gets deleted from the UM_PERMISSION table. That is, if a record in the parent table is deleted the corresponding records in the child table will be automatically deleted.</td>
</tr>
</tbody>
</table>

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

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

**Configuring the Authorization Manager**

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

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

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

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

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

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

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

**Related Topics**

- **Configuring User Stores**: This topic explains how the repositories for storing information about Users and Roles are configured.
- **Setting Up the Physical Database**: This section explains how you can set up a new RDBMS and configure it for your system.
- **Managing Datasources**: This section explains the purpose of defining datasources and how they are configured for a product.

**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. See the related topics for details on 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.
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.

**Related Topics**
- Changing a Password
- Configuring Roles
- Configuring Users
- Role-Based Permissions

**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 **Configure** tab, click **Users and Roles**.

3. To change your own password, click **Change My Password**, enter your current password and new password, and click **Change**.
4. If you are an admin user and need to change another user’s password (such as if they have forgotten their current password and need you to reset it), do the following:
   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. Go to the **Configure** tab in the management console and click **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. See more information about permissions in the related topics.
3. Click **Add New Role**. The following screen will open:

   ![Add Role](image)

   **Step 1 : Enter role details**

   - **Domain**
   - **Role Name**

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. See the related topics for information on 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. See more information about permissions in the related topics. 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

- Adding a user role
- Editing or deleting a role
- Updating role names
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 sometime**

**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`. See more information about configuring the system administrator.

```xml
<Configuration>
  <AdminRole>admin</AdminRole>
  <AdminUser>
    <UserName>admin</UserName>
    <Password>admin</Password>
  </AdminUser>
  <EveryOneRoleName>everyone</EveryOneRoleName> <!-- By default users in this role sees the registry root -->
  <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
  <Property name="MultiTenantRealmConfigBuilder">org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder</Property>
</Configuration>
```

The following are the changes that need to be made in the configurations above:

- **Change** `<AdminRole>admin</AdminRole>` to `<AdminRole>administrator</AdminRole>`.  
- **Change** `<EveryOneRoleName>everyone</EveryOneRoleName>` to `<EveryOneRoleName>Your role</EveryOneRoleName>`.

**Update after the product is used for sometime**

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.

3. After the changes, restart the server.

Related Topics

• About Role-Based Permissions: This section describes all the permissions that can be enabled for a role using the management console.
• Configuring the System Administrator: This section describes how the system administrator user and role is set up and configured.
• Configuring User Stores: This section explains how user stores (which are repositories storing information about Users and Roles) are set up and configured.

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 manager 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
• Customizing the user’s roles and permissions
• Customizing a user’s profile
• Deleting an existing user
• Related Topics

Adding a new user and assigning roles

Follow the instructions below to add a new user account and configure its role.

1. Go to the Configure tab in the management console and click 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:
3. 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. See the related topics for information on configuring user stores.
   2. Enter a unique user name 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**.
   4. 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.
   5. Click **Finish**.

A new user account is created with the specified roles and is listed on the **Users** page.

**Importing users**

In addition to manually adding individual users, you can import multiple users in bulk if you have exported them to a comma-separated values (.csv) file or Microsoft Excel (.xls) file. 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**.


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.

<table>
<thead>
<tr>
<th>UserName,Password,Claims</th>
</tr>
</thead>
<tbody>
<tr>
<td>username1,password1,<a href="http://wso2.org/claims/emailaddress=username1@gmail.com,http://wso2.org/claims/fullname=fullname1">http://wso2.org/claims/emailaddress=username1@gmail.com,http://wso2.org/claims/fullname=fullname1</a></td>
</tr>
</tbody>
</table>
On the **Users** screen, click **Bulk Import Users**.

2. Browse and select the file that contains the user data.

3. Specify a default password to assign to all the users you are importing and click **Finish**. This password is valid for only 24 hours, so you should inform your users that they must log in and change their password within 24 hours.

### 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. See the related topics for information about 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 **Configure** tab in the Management Console, click **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. Go to the **Configure** tab in the management console and click **Users and Roles**.
2. Click **Users**. This link is only visible to users with **User Management** level permissions. See the related topics for more information about permissions.
3. In the **Users** list, click **Delete** next to the user you want to delete, and then click **Yes** to confirm the operation.
Related Topics

- About Role-Based Permissions: This section describes all the permissions that can be enabled for a role using the management console.
- Configuring the System Administrator: This section describes how the system administrator user and role is set up and configured.
- Configuring User Stores: This section explains how user stores (which are repositories storing information about Users and Roles) are set up and configured.

Role-Based Permissions

The User Management module in WSO2 products enables 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>Assign Users</td>
</tr>
<tr>
<td></td>
<td>View Users</td>
</tr>
<tr>
<td>Internal/everyone</td>
<td>Permissions</td>
</tr>
</tbody>
</table>
By default, every WSO2 product comes with the following User, Role and Permissions configured:

- The **Admin** user and **Admin** role is defined and linked to each other in the user-mgt.xml file, stored in the `<PRODUCT_HOME>/repository/conf/` directory as shown below.

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

- The **Admin** role has all the permissions in the system enabled by default. Therefore, this is a super tenant, with all permissions enabled.

You will be able to log in to the management console of the product with the **Admin** user defined in the user-mgt.xml file. You can then create new users and roles and configure permissions for the roles using the management console. However, note that you cannot modify the permissions of the **Admin** role. The possibility of managing users, roles and permissions is granted by the **User Management** permission. See the documentation on configuring the system administrator for more information.

See the following topics:

- **Description of role-based permissions**
- **Product-specific permissions**

**Description of role-based permissions**

Note that the descriptions given in this document only explains how permissions control access to operations available on the management console.

The descriptions of permissions in the **Permissions** navigator are as follows:

- The **Login** permission defined under **Admin** permissions allows users to log in to the management console of the product. Therefore, this is the primary permission required for using the management console.

- The following table describes the permissions at **Super Tenant** level. These are also referred to as **Super Admin** permissions.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
</table>


Configuration permissions:

- **Feature Management** permission ensures that a user can control the features installed in the product using the management console. That is, the Features option will be enabled under the Configure menu. See the topic on feature management for more information.

- **Logging** permission enables the possibility to configure server logging from the management console. That is, the Logging option will be enabled under the Configure menu. See the topic on logging management for more information.

Management permissions:

- **Modify/Tenants** permission enables the option in the Configure menu of the management console, which allows users to add new tenants.

- **Monitor/Tenants** permission enables the View Tenants option in the Configure menu of the management console.

See the topic on configuring multiple tenants for more information.

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.
<table>
<thead>
<tr>
<th>Permission level</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
</table>
| Admin            | When the Admin permission node is selected, the following menus are enabled in the management console:  
- **Configure** menu/User Store Management: This permission allows users to add new user stores and manage them with the management console. Note that only secondary user stores can be added using this option. See the topic on user store management for more details.  
- **Configure** menu/HDFC Role Management  
- Additionally, all permissions listed under Admin in the permissions navigator are selected automatically. |
| Admin/Configure  | When the Admin/Configure permission node is selected, the following menus are enabled in the management console:  
- **Configure** menu/Datasources: See the topic on managing datasources for information on how to use this option.  
- **Configure** menu/Server Roles  
- Additionally, all permissions listed under Configure in the permissions navigator are selected automatically. |
| Admin/Configure/Security | When the Admin/Configure/Security permission node is selected, the following menus are enabled in the Configure menu of the management console:  
- Claim Management  
- Keystores: See the topic on working with keystores for information on keystores.  
- Service Principle (Kerberos KDC): See the topic on service principal management for more information on how to use this option.  
- This permission will also enable the Roles option under Configure/Users and Roles. See the topic on configuring users, roles and permissions for more information.  
- Additionally, all permissions listed under Security in the permissions navigator are selected automatically. |
| Admin/Configure/Security/Identity Management/User Management | This permission enables the possibility to add users from the management console. That is, the Users option will be enabled under Configure/Users and Roles. |
| Admin/Configure/Security/Identity Management/Password Management | This permission enables the Change Password option for the users listed in the User Management/Users and Roles/Users screen, which allows the log in user to change the passwords. |
Admin/Manage

When the Admin/Manage permission is selected, the following menus will be enabled in the management console:

- **Main** menu/RSS Manager: See the topic on provisioning relational database management systems for information on how to use this option.
- **Monitor** menu/Cassandra Stats: See the topic on monitoring cluster statistics for information on how to use this option.
- **Tools** menu/Cassandra Operations: See the topic on performing cluster node operations for information on how to use this option.

- Additionally, all permissions listed under Admin/Manage in the permissions navigator will be enabled automatically.

Admin/Manage/Add

This permission enables the Cassandra Keystages menu under the Main navigator menu. This option allows users to add and manage keystspaces in a Cassandra cluster.

Admin/Manage/Resources/Browse

This permission enables the Browse option under the Registry menu in the main navigator. This option allows users to browse the resources stored in the registry by using the Registry tree navigator.

Admin/Manage/Search

This permission enables the Search option under the Registry sub menu in the Main menu. This option allows users to search for specific resources stored in the registry by filling in the search criteria.

Admin/Monitor/Logs

When the Admin/Monitor/Logs permission node is selected, the following menus are enabled in the management console:

- **Monitor** menu/System Logs: See the topic on system logs for information on how to use this option.
- **Monitor** menu/Application Logs: See the topic on application logs for information on how to use this option.

**Product-specific permissions**

In addition to the common role-based permissions that are used in all WSO2 products, the following permissions are available in WSO2 Storage Sever:

<table>
<thead>
<tr>
<th>Permission Category</th>
<th>Permission</th>
<th>Description of UI menus enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong> permissions</td>
<td></td>
<td>Application permissions are a new category of permissions available in WSO2 Storage Server, because of the Cassandra and RSS Manager features that are installed in the product:</td>
</tr>
<tr>
<td>Name: WSO2 Carbon - Cassandra Feature</td>
<td>Identifier: org.wso2.carbon.cassandra.feature.group</td>
<td></td>
</tr>
<tr>
<td>Name: RSS Manager Feature</td>
<td>Identifier: org.wso2.carbon.rssmanager.feature.group</td>
<td></td>
</tr>
</tbody>
</table>
### Feature Management

This chapter contains the following information:

- Introduction to Feature Management
- Managing the Feature Repository
- Feature Categorization
- Installing Features
- Installed Features
- Installation History
- Recovering from Unsuccessful Feature Installation

The screenshots in this section may vary depending on the product and configuration options you are using.

### 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**

The instructions on this page will explain how feature repositories can be added to a product and managed using the management console. All the features that are built into WSO2 products are available in the WSO2 Feature Repository. Therefore, if a specific feature is not installed in your product, you can install it using the WSO2 feature repository. Alternatively, if you have developed features, you can add them to the product by creating a new feature repository.

You can log in to the product's management console and click **Features** in the **Configure** menu as shown below.

![Feature Repository Management](image)

The **Repository Management** tab allows you to view or modify feature management settings.

**Feature Management**

See the following topics for more information.

- Adding a Repository
- Editing a Repository
- Enabling/Disabling a Repository
- Deleting a Repository

*Adding a Repository*
1. To add a feature repository, click **Add Repository** in the **Manage Repositories** screen.

2. Provide a convenient name and the URL of the repository. You can also add a repository that is downloaded to your computer by giving the file path of the directory in the **Local** field shown below.

   ![Add repository name](repository-name.png)

**Tip**

The official WSO2 Carbon features are available in a Equinox P2 repository at: [http://dist.wso2.org/p2/carbon/releases](http://dist.wso2.org/p2/carbon/releases). Equinox P2 provides provisioning technology for OSGi-based applications such as WSO2 Carbon.

**Editing a Repository**

1. Select a repository to modify and click **Edit** link:

2. You can specify a new name.

   ![Edit repository name](repository-name.png)

   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 add the new one.

**Enabling/Disabling a Repository**
To enable/disable a repository, click 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. Select a repository to delete and click **Remove**.

2. Confirm your request.

### Feature Categorization

The feature repository provided by WSO2 contains a number of features that can be installed into WSO2 Carbon based products. 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. 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. For more information, see **Installing Features**.

Shown below are the features that are installed in this version of WSO2 Storage Server.
Installing Features

The manual way of provisioning Carbon is to add the bundles and configuration files that belong to a feature in to the product directory. This method is not recommended since it can cause errors. Besides, finding the exact set of components and configuration files is a complex task. Components have inter-dependencies with other components. Some components depend on specific versions of other components.

In order to overcome these issues, WSO2 has Equinox P2 integrated with Carbon. It enables user-friendly provisioning capabilities by allowing users to download WSO2 Carbon or any other WSO2 product and simply extend them by installing various features.

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). Then, restart the server. If not, you might not be able to install features through the management console.

Follow the instructions below to install new features to any product of the WSO2 products stack.

1. Log in to the product's management console and click **Features** in the **Configure** menu as shown below.
2. The Feature Management screen opens. The Available Features tab allows you to search for available features in repositories. You can select a repository from the drop-down list. If there is no added repository, see the instructions on how to add a new repository.

The following options can be selected:

- **Show only the latest versions:** Some repositories contain multiple versions of features. If you are only interested in the latest versions, click **Show only the latest versions**.
- **Group features by category:** A feature category is a logical grouping of the features which constitute a particular Carbon-based product. 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 example, the features required to install WSO2 Data Services Server is grouped under the ‘Data Service Server’ feature category as shown below.
If you clear these check boxes when finding features, you will see an uncategorized, flat feature list from which, individual features can be selected separately.

3. Once the repository and options are selected, click **Find Features**. You will see a list of all features. Select the ones you want to add them by selecting the check boxes.

---

**Tip**

To find a particular feature, you can use the search function. Enter the name of a feature (or a part of the name) and click **Enter**.

This search will return only the available features, excluding the ones already installed.

4. Once done, click **Install**.

---

5. The **Install Details** screen appears. Verify the provided information and click **Next**.
6. Read and accept the terms of license agreement.

7. The installation process starts. It may take a few minutes to download the necessary components. Once the installation process is complete, click Finish and restart the server for the changes to take effect.

8. You can see the functionality added by new features in the management console. For example, the Databas
es menu appears on the navigator when the Admin Console feature is installed.

View Feature Information: When features of a repository are loaded, you can install them or view the details of particular ones. To view the details, click the More Info link associated with each feature as shown below.

When the link is selected, the Feature Information screen appears, with the following information: Name, Identifier, Version, Provider, Description, Copyright, License Agreement.

9. Once you are familiar with the information click Back to load the previous page.
**Installed Features**

The instructions on this page will explain how features can be installed in a product and managed using the management console. All the features that are built into WSO2 products are available in the WSO2 Feature Repository. Therefore, if a specific feature is not installed in your product, you can install it using the WSO2 feature repository. Alternatively, if you have developed features, you can add them to the product by creating a new feature repository.

1. Log on to the product’s management console and select **Features** from the **Configure** menu as shown below.

![Feature Management](image)

2. The **Feature Management** screen opens. The **Installed Features** tab allows you to browse through the list of installed features and select the ones you want to uninstall.

![Installed Features Table](image)

You can access additional information about a feature by clicking the **More info** link in the **Actions** column.

![More Info Link](image)

WSO2 Carbon supports back-end, front-end separation where you can manage multiple back-end servers using a single front-end server. You can convert a given product either to a back-end server or to a front-end server. This conversion can be performed using the **Feature Manager** user interface.
If you want to get only a back-end server, you have to uninstall all the front-end features. To do that, select "Front-end" from the drop down menu as displayed in the example below.

This will list all the front-end features that are currently installed in the system. Clicking **Uninstall** will uninstall all the front-end features in the current installation. The steps are similar if you want to convert your product into a front-end server.

**Uninstalling Features**

1. Select features to uninstall by selecting the relevant check boxes. You can also select all/none features using the appropriate links.

2. Once selected, click **Uninstall**.

3. A page will appear containing details of the features to be uninstalled. Verify the information and click **Next**.

4. If successful, a message appears as "Uninstallation Complete". Click **Finish** and restart the server to apply the changes.

**Note**

A feature cannot be uninstalled from the system without uninstalling other features that depend on it, if any.
Installation History

The following instructions explain how to view the history of feature installation in your product.

1. Log on to the product's Management Console and select "Features" from the "Configure" menu. For example,

2. The "Installation History" tab lists the history of provisioning operations performed on the system. Previous configurations can be identified as previous states of the system. A state/configuration can be simply identified by the set of installed features. When you perform a provisioning operation such as installing/uninstalling of features, a system state/configuration change occurs.

Feature Management

3. You can revert the current configuration to a previous configuration. In this process, some features might get installed and some uninstalled. When you click on a previous configuration, you will be able to see information as shown in the following example.
4. Click "Revert" to revert the current configuration to a previous configuration or Back to return to the previous page without any 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.

- Restoring using the management console
- Restoring using the command line

Restoring using the management console

1. Log in to the management console and select Features from the Configure menu.
2. In the Feature Management page, go to the Installation History tab.
3. This tab lists the history of provisioning operations performed on the system. For example,

   ![Feature Management](image)

4. Click on a configuration to view its details. For example,
Previous configurations can be identified as previous states of the system. It is a set of installed features. When you perform a provisioning operation such as installing/uninstalling of features, a system state/configuration change occurs.

5. Verify if the state is where you want to revert to and click **Revert**.

**Restoring using the command line**

If you cannot start the server after an unsuccessful feature installation, use the following steps to restore to a previous installation.

1. Start the product with `-DosgiConsole` system property.
2. Once the server is started, type the command `osgi> getInstallationHistory`.
3. A list of previous server states appears. For example,

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1376883697814</td>
<td>August 19, 2013 at 09:11:37 IST</td>
</tr>
<tr>
<td>1376883697957</td>
<td>August 19, 2013 at 09:11:37 IST</td>
</tr>
<tr>
<td>1376883700725</td>
<td>August 19, 2013 at 09:11:40 IST</td>
</tr>
<tr>
<td>1376883704884</td>
<td>August 19, 2013 at 09:11:44 IST</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

4. You can check what features are installed and uninstalled in a given state by entering the following command:

   `osgi> getInstallationHistory <timestamp>`  

For example:

   `osgi> getInstallationHistory 1376933879416`

The output gives you details similar to the following:

```
-- Installed 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
...```

5. Decide to which state you want to revert the system and enter the following command:

   `osgi> revert <timestamp>`

For example:

   `osgi> revert 1376933879416`

The output will be similar to the following:

```
Successfully reverted to 1376933879416
Changes will get applied once you restart the server.
```
Registry Management

This chapter contains the following information:

- Introduction to Registry
- Managing the Registry
- Searching the Registry

Introduction to Registry

A registry is a content store and a metadata repository. Various SOA artifacts such as services, WSDLs and configuration files can be stored in a registry, keyed by unique paths. 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 Carbon provides the basic registry and repository functionality. Products based on Carbon use the services provided by the Registry kernel to establish their own registry space, which is 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.
- AtomPub interfaces to publish, view and manage resources from remote or non-Java clients.

The Registry space provided to each Carbon product contains three major partitions.

- **Local Data Repository** - Used to store settings/metadata specific to the product. This registry is not intended to be shared among multiple servers.
- **Configuration Registry** - Used to store product-specific configurations. These configurations can be shared across multiple instances of the same product like a cluster.
- **Governance Registry** - Used to store user-specified metadata and resources and can be shared across an organization.

These registry instances are mounted to a single top level registry to provide a single, unified view. Mount points of the three registries are /_system/local, /_system/config and /_system/governance respectively. One could browse the contents of the registry used by the Carbon product through its management console.

Managing the Registry

Follow the instructions below to access the registry user interface.

1. Log on to the product's Management Console and select *Browse* under *Registry*. 
2. The *Browse* page appears.

![Browse page](image)

**Components of Registry User Interface**

- **Breadcrumb** - Shows the current directory hierarchy.
- **Metadata** - Shows metadata for the resource/collection.
- **Properties** - Shows properties for the resource/collection.
- **Entries** - Shows the contents of the resource/collection.
- **Permissions** - Shows the defined role permissions to use the resource/collection.

**Managing Breadcrumb**

Use the breadcrumb to navigate backward in the current branch of the directory path by clicking on a directory name in the breadcrumb.

1. In the *Browse* window, click the *Tree View* tab to see the branch.
2. Click on a particular directory name to see its details in the *Detail view* tab.

![metadata panel](image)

### Managing Metadata

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

- **Created** - Shows the time when a resource was created and the author of a resource/collection.
- **Last Updated** - Shows the time when a resource was updated and the author of alterations/collection.
- **Media Type** - An associated media type of the resource/collection.
- **Checkpoint** - Allows to create a checkpoint (URL for the permanent link) of a resource/collection.
- **Versions** - Allows to view versions of a resource/collection.
- **Description** - Description of the resource/collection.

For example,
Checkpoint Creation

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

2. If the checkpoint was successfully created, a message will be displayed. Click *OK*.

Viewing Versions

1. Click on the *View versions* link.
2. The *Versions* page appears.

- **Version** - Shows the number of a resource/collection version.
- **Last Modified Date** - Shows the last date of updating.
- **Last Modified By** - Shows the author of alterations.
- **Actions**
  - **Details** - Allows to get to the *Browse* page of a particular resource/collection version.
  - **Restore** - Allows to restore a resource/collection version.

**Editing Description**

1. To edit a description of a resource/collection, click on the *Edit* link.
2. Edit the description of a resource/collection in the text area and click Save.

---

Managing 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.

---

Adding a Property

1. To add a property, click on the Add New Property link.

2. In the Add New Property panel, enter a unique name of a property and its value. Click Add.
Editing a Property

1. Click on the *Edit* link of a particular property in the *Action* column.

2. Edit the name and the value of a property in the active fields and click *Save*.

Deleting a Property

1. To delete a property, click on the *Delete* link of a certain property in the *Action* column.

2. Confirm your request by clicking *Yes*.
Managing Entries and Content

If the currently selected entity is a collection, the Content panel is called the Entries panel and shows the child entries under that collection. It provides details of each entry. An entry can be either another collection or a resource. Here you can also add a new resource, add a new collection and create links.

If the currently selected entity is a resource, the Content panel provides a user interface through which one can display, edit, upload, and download the content.

The Entries Panel

The following information is given as shown in the example screenshot below.

- Add Resource
- Add Collection
- Create Link
- Child Resources - The list of child entries provides the following information:
  - Name - The name of a child resource.
  - Created On - The date when a child resource was created.
  - Author - The author who created a child resource.

You can also see the detailed information about the resource by clicking on the Info icon. The following information is available as shown in the example screenshot below.

- Media Type
- Feed
- Rating
To see the available actions over a resource, click on the Actions icon.

The following actions over the resources are available:

- **Rename** - Allows to rename a resource.
- **Move** - Allows to move a resource to a new directory.
- **Delete** - Allows to delete a resource.
- **Copy** - Allows to copy a resource to a specified directory.

**Tip**

All these options are available not for all the resources.

**Renaming a Resource**

1. To rename a resource, click **Rename** and enter a new name to the field.

2. Click on the **Rename** button to save a new name of a resource.

**Moving a Resource**

1. To move a resource to a new directory, click **Move** and specify **Destination Path**.

2. Click **Move**.

**Deleting a Resource**
1. To delete a resource, click **Delete** and confirm your request by clicking **Yes** in the message that appears if the resource is deleted successfully.

![Delete Resource](image)

**Copying a Resource**

1. To copy a resource to some directory, click **Copy** and specify **Destination Path**.

![Copy Resource](image)

2. Click **Copy**.

If the resource was successfully copied, a message appears. Click **OK**.

**Adding a Collection**

Follow the instructions below to add a new collection.

1. To add a new collection, click **Add Collection**.

![Add Collection](image)

2. Specify the following options:

   - **Name** - The unique name of a collection.
   - **Media Type** - Select media type of a collection from the drop-down menu:
     - application/vnd.wso2.esb
3. Click *Add*.

**Adding a Resource**

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

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

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

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

The following methods are available:

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

**Uploading Content from File**

1. If this method was selected, specify the following options:

   - File - The path of a file to fetch content (XML, WSDL, JAR etc.) Use the *Browse* button to upload a file.
   - Name - The unique name of a resource.
   - Media type - Can add a configured media type or leave this unspecified to enforce the default media type.
   - Description - The description of a resource.

2. Click *Add* once the information is added as shown in the example below.
Importing Content from URL

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

2. Click Add once the information is added.

Text Content Creation

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

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

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

Media Types

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

- The first method is by means of a one-time configuration, which can be done by modifying the "mime.types" file found in `<CARBON_HOME>/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.

   ![Create Link Panel](image)

   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.

   ![Create Symbolic Link Panel](image)

   **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.
Managing Role Permissions

The Permissions panel shows the defined role permissions, allows to add new role permissions and edit existing ones.

Adding New Role Permissions

1. In the New Role Permission pane, select a role to set a permission.

2. Select an action from the drop-down menu. The following actions are available:
   - Read
   - Write
   - Delete
   - Authorize
3. Select whether to allow the action the selected role or deny.

4. Click on the Add Permission button.

5. A new permission appears in the Defined Role Permissions list.

**Editing Role Permissions**

1. You can also edit the defined role permissions using the check boxes in the Defined Role Permissions list.

2. After editing the permissions, click on the Apply All Permissions button to save the alterations.

**Searching the Registry**

All resources found in the Registry can be searched through the product's Management Console. Search can be refined by resource name, created date range, updated date range, tags, comments, property name, property value, media type etc.
To search for matches containing a specific pattern, use the "%" symbol.

Follow the instructions below to find a necessary resource in the Registry.

1. Log on to the product's Management Console and select Metadata under Registry.

2. The Search window appears.
The search can be refined by:

- **Resource Name**
- **Created Date Range** - The date when a resource was created.
- **Updated Date Range** - The date when a resource was updated.
- **Update Author** - The author of a resource updating.
- **Create Author** - The author of a resource creation.
- **Tags**
- **Comments**
- **Property Name**
- **Property Value**
- **Media Type**

## Working with Databases

The default databases that WSO2 products use to store registry, user management and product-specific data are the following H2 databases stored in the `<PRODUCT_HOME>/repository/database` directory:

- **`WSO2CARBON_DB.h2.db`**: used to store registry and user management data.
- **`RSS_DATABASE.ht.db`**: used to store RSS specific data.

These embedded H2 databases are suitable for development, testing, and some production environments. For most production environments, however, we recommend to use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. You can use the scripts provided with WSO2 products to install and configure several other types of relational databases in place of the default H2 database. However, note that in WSO2 Storage Server, you can only use **H2**, **MySQL**, **MS SQL** and **PostgreSQL** as the database type for the RSS specific databases.

The following topics explain how to change the default databases with a new database type:

- **Setting up the Physical Database**

Tip

Created or updated dates can be either entered in the format of MM/DD/YYYY or picked from the calendar interface provided.

3. Fill the search criteria and click on the **Search** button. The results are displayed in the Search Results window.
Managing Datasources

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 Derby
- Setting up H2
- Setting up IBM DB2
- Setting up MariaDB
- Setting up MS SQL
- Setting up MySQL
- Setting up OpenEdge
- Setting up Oracle
- Setting up Oracle RAC
- Setting up with MySQL Cluster
- Setting up with PostgreSQL

Setting up Derby

You can set up either an embedded Derby database or a remote one according to the information given below:

- Setting up Embedded Derby
- Setting up Remote Derby

Setting up Embedded Derby

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

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

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.

Setting up the drivers

Copy `derby.jar`, `derbyclient.jar`, and `derbynet.jar` from the `<DERBY_HOME>/lib/` directory to the `<PRODUCT_HOME>/repository/components/extensions/` directory (the classpath of the WSO2 Carbon web application).

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 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/monster-datasources.xml file as shown below.

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections</td>
</tr>
</tbody>
</table>
   ```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

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

Configuring new datasources to manage registry or user management data

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

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

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

3. If you are setting up a separate database to store user management data, update the following configurations in the `<PRODUCT_HOME>/repository/conf/ user-mgt.xml` file.
<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 version 10.8
   i> 
   ```

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.
Setting up Remote Derby

The following sections describe how to replace the default H2 databases with a remote Derby database:

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

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

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 Derby 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:derby://localhost:1527/db;create=true</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.apache.derby.jdbc.ClientDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and pass word</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
</tbody>
</table>
**validationInterval**  
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 an 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](#).

In contrast to setting up with embedded Derby, in the remote registry you set the database driver name (the `driverName` element) to the value `org.apache.derby.jdbc.ClientDriver` and the database URL (the `url` element) to the database remote location.

Configuring new datasources to manage registry or user management data

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

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

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

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

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

**Creating database tables**

You can create database tables by executing the following script(s):

1. Run the `ij` tool located in the `<DERBY_HOME>/bin/` directory.
2. Create the database and connect to it using the following command inside the ij prompt:

   ```
   connect 'jdbc:derby://localhost:1527/db;user=regadmin;password=regadmin;create=true';
   ```

   Replace the database file path, user name, and password in the above command to suit your requirements.

3. Exit from the ij tool by typing the exit command as follows:

   ```
   exit;
   ```

4. Log in to the ij tool with the username and password you just used to create the database.

   ```
   connect 'jdbc:derby://localhost:1527/db' user 'regadmin' password 'regadmin';
   ```

5. You can create database tables manually by executing the following scripts.

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

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

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 now configured to run using a remote Apache Derby database.

### Setting up H2

You can set up either an embedded H2 database or a remote one using the instructions given below:

- Setting up Embedded H2
- Setting up Remote H2

### Setting up Embedded H2

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

- Setting up the database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
**Setting up the database**

Download and install the H2 database engine in 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.

**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:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

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

Configuring new datasources to manage registry or user management data

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

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

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

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

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

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

**Creating database tables**

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

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

Setting up Remote H2

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

- Setting up the remote H2 database
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

**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:$ cd bin
```

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:

   ```bash
   tcp://localhost:9092
   ```
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.

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

**Setting up the drivers**

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

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

**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:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
</tbody>
</table>
**validationInterval**
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 an 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**.

---

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

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

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

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

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

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

---

**Creating database tables**

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

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

  ```sql
  <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`

**Setting up IBM DB2**

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

- **Prerequisites**
- Setting up the database and users
- Setting up DB2 JDBC drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity databases

**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 in CLI to open DB2.
2. Create the database using the following command:
   ```
   create database <DB_NAME>
   ```
3. Before issuing a 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 Command Processor Example](image)

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:

   ![DB2 Control Center Command](image)

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_c0u.jar) from `<DB2_HOME>/SQLLIB/java` directory to the `<PRODUCT_HOME>/repository/components/lib/` directory.

```
user@ws02:~/sqlib/java export DB2_HOME=/home/user/ws02/greg/repository/components/lib/
```

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

Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, `WSO2_CYBER_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_CYBER_DB` datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default WSO2_CYBER_DB datasource**
Follow the steps below to change the type of the default WSO2_CARBON_DB datasource.

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

   ```xml
   <datasource>
     <name>WSO2_CARBON_DB</name>
     <description>The datasource used for registry and user manager</description>
     <jndiConfig>
       <name>jdbc/WSO2CarbonDB</name>
     </jndiConfig>
     <definition type="RDBMS">
       <configuration>
         <url>jdbc:db2://SERVER_NAME:PORT/DB_NAME</url>
         <username>regadmin</username>
         <password>regadmin</password>
         <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
         <maxActive>80</maxActive>
         <maxWait>360000</maxWait>
         <minIdle>5</minIdle>
         <testOnBorrow>true</testOnBorrow>
         <validationQuery>SELECT 1</validationQuery>
         <validationInterval>30000</validationInterval>
         <defaultAutoCommit>false</defaultAutoCommit>
       </configuration>
     </definition>
   </datasource>
   ```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
</tbody>
</table>
## Configuring new datasources to manage registry or user management data

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

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

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

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

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

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

## Creating database tables

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

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

```
<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 the product-specific/identity databases

The topics above show how to change the WSO2CARBON_DB, which is used to store registry and user manager information. If you changed the product-specific database (WSO2_RSS_DB) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the master-datasources.xml file. The datasource for the product-specific database is already there in the file by the name WSO2_RSS_DB. Change its elements with your custom values.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the product-specific database</th>
<th>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/rss-manager/</code> folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the identity database</td>
<td>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/</code> folder</td>
</tr>
</tbody>
</table>

Setting up MariaDB

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

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

Setting up the database and users

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

1. Download, install and start MariaDB on your computer. See https://downloads.mariadb.org/.
You can install MariaDB standalone or as a galera cluster for high availability. Database clustering is independent of WSO2 product clustering. For more information on setting up a galera cluster, see the MariaDB Galera Cluster documentation.

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

2. Log in to MariaDB as the root user (or any other user with database creation privileges).
   
   ```
   mysql -u root -p
   ```

3. Enter the password when prompted.

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

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

5. Give authorization of the database 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 which version of a WSO2 product has been tested for compatibility with which version of MariaDB and MySQL connector.

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 datasource name `WSO2_CARBON_DB` in the below configuration.

```xml
<datasource>
  <name>WSO2_CARBON_DB</name>
  <description>The datasource used for registry and user manager</description>
  <jndiConfig>
    <name>jdbc/WSO2CarbonDB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://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>
      <validationOnBorrow>SELECT 1</validationOnBorrow>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for MariaDB is 3306</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

For more information on other parameters that can be defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file, see [Tomcat JDBC Connection Pool](https://wso2.org/products/tomcat/docs/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/wso2registry` file.

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

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

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

**Creating database tables**

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

## Setting up MS SQL

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

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the product-specific/identity databases

### 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 2005**.
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 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, create tables, and insert, index, select, update, and delete data in tables in the newly created database, as 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.

**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.

```xml
<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:sqlserver://<IP>:1433;databaseName=wso2greg</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```
The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. Change the &lt;IP&gt; with the IP of the server. The best practice is to use port 1433, because you can use it in order processing services.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

Creating the database tables

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

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

```sh
<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 the product-specific/identity databases

The topics above show how to change the `WSO2CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2_RSS_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2_RSS_DB`. Change its elements with your custom values.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the product-specific database</th>
<th>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/rss-manager/</code> folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the identity database</td>
<td>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/</code> folder</td>
</tr>
</tbody>
</table>

### Setting up MySQL

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

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the registry/user management databases
- Changing the product-specific/identity databases

#### 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:

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

2. Start the MySQL service using the following command:

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

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

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

4. Enter the password when prompted.

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

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

   ```bash
   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:

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

   For users of other operating systems, the standard database creation commands will suffice. For
6. Give authorization of the database to the regadmin user as follows:

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

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

   FLUSH PRIVILEGES;

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

   quit;

### Setting up the drivers

Download the MySQL Java connector JAR file, and copy it to the \(<\text{PRODUCT_HOME}>/\text{repository/}\text{components/}\text{lib/}\) directory.

**Tip:** 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 \(<\text{PRODUCT_HOME}>/\text{repository/}\text{components/}\text{lib}\) and \(<\text{PRODUCT_HOME}>/\text{repository/}\text{components/}\text{dropins}\).
2. Download the connector JAR that is compatible with your current MySQL version.
3. Copy the JAR file only to \(<\text{PRODUCT_HOME}>/\text{repository/}\text{components/}\text{lib}\). Files will be copied automatically to the dropins folder at the server startup.
4. Start the server with the \(-\text{Dsetup}\) parameter as `sh wso2server.sh -Dsetup`.

### Setting up datasource configurations

A datasource is used to establish the connection to a database. By default, \texttt{WSO2\_CARBON\_DB} datasource is used to connect 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 \texttt{WSO2\_CARBON\_DB} datasource, or configure a new datasource to point it to the new database as explained below.

**Changing the default \texttt{WSO2\_CARBON\_DB} datasource**

Follow the steps below to change the type of the default \texttt{WSO2\_CARBON\_DB} datasource.

1. Edit the default datasource configuration in the \(<\text{PRODUCT_HOME}>/\text{repository/}\text{conf/}\text{datasources/}\text{master-datasources.xml}\) file as shown below.

   Do not change the datasource name \texttt{WSO2\_CARBON\_DB} in the below configurations.
<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>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for MySQL is 3306</td>
</tr>
<tr>
<td>username and pass word</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
</tbody>
</table>
### Validation Interval

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.

### Default AutoCommit

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 an 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/` file, see Tomcat JDBC Connection Pool.

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

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

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

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

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

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

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

### Creating database tables

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

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

   ```sh
   You may have to enter the password for each command when prompted.
   ```
Changing the registry/user management databases

If you change the database that comes by default or set up a separate database for registry or user management related data, follow the below instructions.

1. Add the datasource 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.

Changing the product-specific/identity databases

The topics above show how to change the `WSO2CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2_RSS_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2_RSS_DB`. Change its elements with your custom values.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the product-specific database</th>
<th>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/rss-manager/</code> folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the identity database</td>
<td>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/</code> folder</td>
</tr>
</tbody>
</table>

Setting up OpenEdge
The following sections describe how to replace the default H2 database with OpenEdge (OE):

- Setting up the database and users
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables

**Setting up the database and users**

Follow the steps below to set up an OpenEdge (OE) database.

1. Download and install OpenEdge on your computer.
2. Go to the `<OE_HOME>/bin/` directory and use the `proenv` script to set up the environment variables.
3. Add `<OE_HOME>/java/prosp.jar` to the `CLASSPATH` environment variable.
4. Create an empty database using the `prodb` script as follows. This script creates a database by copying an existing database provided with the installation.
   
   ```bash
   prodb CARBON_DB <OE-installation-directory>/empty8
   ```
5. Start the database using the `proserve` script as follows. Provide the database name and a port as arguments to this script using the `-db` and `-S` parameters.
   
   ```bash
   proserve -db CARBON_DB -S 6767
   ```
6. Use the `sqlexp` script to start the default SQL explorer that comes with the OpenEdge installation. Connect to the database you just created by using the `-db` and `-S` parameters as follows:
   
   ```bash
   sqlexp -db CARBON_DB -S 6767
   ```
7. Use the following commands to create a user and grant that user the required permissions to the database:
   
   ```sql
   CREATE USER 'wso2carbon','wso2carbon';
   GRANT dba,resource TO 'wso2carbon';
   COMMIT;
   ```
8. Log out from the SQL explorer by typing the following command: `exit`

**Setting up the drivers**

Copy the `<OE_HOME>/java/openedge.jar` file to your WSO2 product's `<PRODUCT_HOME>/repository/components/lib/` directory.

**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 OpenEdge 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:datadirect:openedge://localhost:6767;databaseName=CARBON_DB</url>
      <username>regadmin</username>
      <password>regadmin</password>
      <driverClassName>com.ddtek.jdbc.openedge.OpenEdgeDriver</driverClassName>
      <maxActive>80</maxActive>
      <maxWait>60000</maxWait>
      <minIdle>5</minIdle>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and password</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td><strong>validationQuery</strong></td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>validationInterval</strong></td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td><strong>defaultAutoCommit</strong></td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

### Creating database tables

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

- To create tables in the registry and user manager database `WSO2CARBON_DB`, use the below script:
Follow the steps below to create the database tables by executing the scripts.

1. Modify the OpenEdge script provided with the product to create the tables manually. Make a backup of the `<PRODUCT_HOME>/dbscripts/openedge.sql` script under the name `openedge_manual.sql`.
2. Replace all the "/" symbols in the `openedge_manual.sql` script with the ":." symbol.
3. At the end of the `openedge_manual.sql` script, add the following line and save the script:
   
   ```sql
   COMMIT;
   ```
4. Run the modified script using the SQL explorer as follows:
   
   ```bash
   sqlexp -db CARBON_DB -S 6767 -user wso2carbon -password wso2carbon <PRODUCT_HOME>/dbscripts/openedge_manual.sql
   ```
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`

### Setting up Oracle

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

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the product-specific/identity databases

### Setting up the database and users

Follow the steps below to set up a Oracle database.

1. Create a new database by using the Oracle database configuration assistant (dbca) or manually.
2. Make the necessary changes in the Oracle `tnsnames.ora` and `listener.ora` files in order to define addresses of the databases for establishing connections to the newly created database.
3. After configuring the `.ora` files, start the Oracle instance using the following command:
   
   ```bash
   $ sudo /etc/init.d/oracle-xe restart
   ```
4. Connect to Oracle using SQL*Plus as SYSDBA as follows:
   
   ```bash
   $ ./$<ORACLE_HOME>/config/scripts/sqlplus.sh sysadm/password as SYSDBA
   ```
5. Connect to the instance with the username and password using the following command:

   $ 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 with WSO2 servers, set the Java property as follows:

   ```
   export JAVA_OPTS="-Duser.timezone='+05:30'
   ```

   The value of this property should be the GMT difference of the country. If it is necessary to set this property permanently, define it inside the `wso2server.sh` as a new JAVA_OPTS property.

**Setting up datasource configurations**

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

**Changing the default WSO2_CARBON_DB datasource**

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

1. Edit the default datasource configuration in the `<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:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and pass word</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
</tbody>
</table>
validationInterval | 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 an 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.

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.

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

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

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

### Creating the database tables

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

1. To create tables in the registry and user manager database (`WSO2CARBON_DB`), use the below script:
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 the product-specific/identity databases

The topics above show how to change the `WSO2CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2_RSS_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2_RSS_DB`. Change its elements with your custom values.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the product-specific database</th>
<th>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/rss-manager/</code> folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the identity database</td>
<td>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/</code> folder</td>
</tr>
</tbody>
</table>

### Setting up Oracle RAC

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

- Setting up the database and users
- Setting up the JDBC driver
- Setting up datasource configurations
- Creating the database tables
- Changing the product-specific/identity databases

Oracle Real Application Clusters (RAC) is an option for the Oracle Database for 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`. 

```sql
SQL> @$<PRODUCT_HOME>/dbscripts/oracle.sql
```
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 18:10:02 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.

**Setting up datasource configurations**

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

**Changing the default `WSO2_CARBON_DB` datasource**

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

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

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL of the database. The default port for a DB2 instance is 50000.</td>
</tr>
<tr>
<td>username and pass</td>
<td>The name and password of the database user</td>
</tr>
<tr>
<td>word</td>
<td></td>
</tr>
<tr>
<td>driverClassName</td>
<td>The class name of the database driver</td>
</tr>
<tr>
<td>maxActive</td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td>maxWait</td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td>minIdle</td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td>testOnBorrow</td>
<td>The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>validationQuery</td>
<td>The SQL query that will be used to validate connections from this pool before returning them to the caller.</td>
</tr>
<tr>
<td>validationInterval</td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td>defaultAutoCommit</td>
<td>This property is not applicable to the Carbon database in WSO2 products because auto committing is usually handled at the code level, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Typically, auto committing is enabled for an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

Creating the database tables

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

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

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

2. Restart the server.

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

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

Changing the product-specific/identity databases

The topics above show how to change the `WSO2CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2_RSS_DB`) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2_RSS_DB`. Change its elements with your custom values.

2. Create the database tables using the following scripts:

   ```
   | For the product-specific database | Use the scripts in `<PRODUCT_HOME>/dbscripts/rss-manager/` folder |
   ```
For the identity database | Use the scripts in <PRODUCT_HOME>/dbscripts/identity/ folder

Setting up with MySQL Cluster

Find instruction on setting up any WSO2 Carbon-based product with MySQL cluster, refer to the following article published on WSO2 Oxygen Tank:

- [http://wso2.org/library/articles/2012/06/deploying-wso2-platform-mysql-cluster](http://wso2.org/library/articles/2012/06/deploying-wso2-platform-mysql-cluster)

Setting up with PostgreSQL

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

- Setting up the database and login role
- Setting up the drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity databases

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:
   ```
   sudo apt-get install postgresql
   ```

2. Start the PostgreSQL service using the following command:
   ```
   sudo /etc/init.d/postgresql start
   ```

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

4. To connect PGAdminIII to a PostgreSQL database server, locate the server from the object browser, right-click the client, and click Connect. This will show you the databases, table spaces, 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 (for example: 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.

**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.

```xml
<data source>
  <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>true</defaultAutoCommit>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</data source>
```

The elements in the above configuration are described below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>url</strong></td>
<td>The URL of the database. The default port for a PostgreSQL instance is 5432.</td>
</tr>
<tr>
<td><strong>username</strong> and <strong>password</strong></td>
<td>The name and password of the database user.</td>
</tr>
<tr>
<td><strong>driverClassName</strong></td>
<td>The class name of the database driver.</td>
</tr>
<tr>
<td><strong>maxActive</strong></td>
<td>The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.</td>
</tr>
<tr>
<td><strong>maxWait</strong></td>
<td>The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.</td>
</tr>
<tr>
<td><strong>minIdle</strong></td>
<td>The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.</td>
</tr>
<tr>
<td><strong>testOnBorrow</strong></td>
<td>Whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and another attempt will be made to borrow another.</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>validationInterval</strong></td>
<td>The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again.</td>
</tr>
<tr>
<td><strong>defaultAutoCommit</strong></td>
<td>Whether to commit database changes automatically or not. 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 an RDBMS drivers by default. When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

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

### Creating database tables

To create the database tables, connect to the database that you created earlier and run the following scripts.
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 the product-specific/identity databases**

The topics above show how to change the WSO2CARBON_DB, which is used to store registry and user manager information. If you changed the product-specific database (WSO2_RSS_DB) that comes by default or set up a separate database for identity related data, the instructions are the same. In summary:

1. Add the datasource to the `master-datasources.xml` file. The datasource for the product-specific database is already there in the file by the name `WSO2_RSS_DB`. Change its elements with your custom values.

2. Create the database tables using the following scripts:

<table>
<thead>
<tr>
<th>For the product-specific database</th>
<th>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/rss-manager/</code> folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the identity database</td>
<td>Use the scripts in <code>&lt;PRODUCT_HOME&gt;/dbscripts/identity/</code> folder</td>
</tr>
</tbody>
</table>

**Managing Datasources**

A datasource provides information that a server can use to connect to a database or to an external data store. The topics given below explain the ways in which new datasources can be created and edited in WSO2 products.

- Creating datasources using the management console
- Creating datasources using configuration files
- Editing datasource information

**Creating datasources using the management console**

The following feature in the WSO2 Feature Repository allows you to manage datasources using the management console of your product:

**Name:** WSO2 Carbon - datasource management feature

**Identifier:** org.wso2.carbon.datasource.feature.group

If this feature is not included in your product by default, you can install it using the Feature Management section of this documentation. With this feature, you can add any RDBMS datasource or your own custom datasources using
the management console, which can later be edited or deleted for your product instance if required. Note that all datasources created using the management console are saved to the registry of your product. The steps for creating datasources are discussed in the following topics:

- Configuring an RDBMS Datasource
- Configuring a Custom Datasource

**Creating datasources using configuration files**

In all WSO2 products, you also have the option of creating datasources by simply creating a configuration file with the datasource information and storing it in the `<PRODUCT_HOME>/repository/conf/datasources/` directory. For example, all WSO2 products are shipped with the default `WSO2_CARBON_DB` database, and the datasource that is used to connect to this database is defined in the `master-datasources.xml` file stored in the `datasources` folder. You can also create your own custom datasources by creating configuration files and storing them in the same folder. For example, see the `custom-datasources.xml` file. The datasources defined using configuration files can be viewed using the management console if the datasource management feature is installed.

When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to `<PRODUCT_HOME>/repository/components/lib` directory.

**Editing datasource information**

After datasources are created (either using the management console or by creating a separate configuration file), you can access them by clicking **Data Sources** on the **Configure** tab of the management console as shown below.

![Data Sources in Management Console](image)

You can also edit and delete the datasources that are saved in the registry (i.e. datasources created using the management console). However, you cannot edit or delete the datasources that are added using configuration files in the `<PRODUCT_HOME>/repository/conf/datasources/` directory using the management console. For example, to modify the default `<WSO2_CARBON_DB>` datasource, you have to change the `master-datasources.xml` file.

**Configuring an RDBMS Datasource**

Follow the steps given below to create RDBMS datasources for your product server. RDBMS datasources allow you to service enable data from relational databases.

1. Log in to the management console of your product and click **Data Sources** on the **Configure** tab. The datasources that are already created will be listed.
2. Click **Add Data Source** to open the following screen:

![New Data Source screen]

3. Select ‘RDBMS’ for the **Data Source Type** field to define an RDBMS configuration.

4. Enter values for the remaining fields as explained below.
   - **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. Be sure to copy the JDBC driver relevant to the database engine to the `<PRODUCT_HOME>/repository/components/dropins` and `<PRODUCT_HOME>/repository/components/lib` directories. For example, if you are using MySQL, you would specify `com.mysql.jdbc.Driver` as the driver and would copy `mysql-connector-java-5.XX-bin.jar` to these directories. If you do not copy the driver to these directories 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 to pass to the JDBC driver to establish the connection
   - **Password**: The connection password to pass to the JDBC driver to establish the connection
   - **Expose as a JNDI Data Source**: Allows you to specify the JNDI data source
   - **Data Source Configuration Parameters**: Allows you to specify the datasource connection pool parameters when creating an RDBMS datasource

5. After creating datasources, they appear on the **Data Sources** page. If required, you can edit or delete them by clicking **Edit** or **Delete**.

   ![Screenshot of Data Sources page]

   This is the default RDBMS datasource configuration provided by WSO2. You can also write your own RDBMS configuration by selecting the **custom datasource** option.

### 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 an RDBMS datasource, you can 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 connection properties **Driver**, **URL**, **User Name**, **Password**, **Driver**, **URL**, **User Name**, and **Password**.
Name, and Password as follows:

External datasource provider

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

Configuring a JNDI Datasource

Java Naming and Directory Interface (JNDI) is a Java application programming interface (API) that provides naming and directory functionality for Java software clients to discover and look up data and objects via a name. It helps decouple 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.

You can expose an **RDBMS datasource** as a JNDI datasource at the point of adding a datasource. Click **Expose as a JNDI Data Source** to display the JNDI fields as shown below.
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 data source factory. When this option is selected, a reference object will be created with the defined datasource properties. The data source 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 follows: `<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

**Datasource Configuration 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** and 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 management console when creating a datasource. Click and expand the option as shown below:
Each parameter is described below:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Isolation</td>
<td>The default TransactionIsolation state of connections created by this pool.</td>
</tr>
<tr>
<td>Initial Size</td>
<td>(int) The initial number of connections created when the pool is started. Default value is 0.</td>
</tr>
<tr>
<td>Max. Active</td>
<td>(int) The maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.</td>
</tr>
<tr>
<td>Max. Idle</td>
<td>(int) The maximum number of connections that can remain idle in the pool, without extra ones being released. Default value is 8. Put a negative value for unlimited. Idle connections are checked periodically (if enabled) and connections that have been idle for longer than minEvictableIdleTimeMillis will be released. (also see testWhileIdle)</td>
</tr>
<tr>
<td>Min. Idle</td>
<td>(int) 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. (also see testWhileIdle)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Max. Wait</td>
<td>(int)</td>
</tr>
<tr>
<td>Validation Query</td>
<td>(String)</td>
</tr>
<tr>
<td>Test On Return</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Test On Borrow</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Test While Idle</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Time Between Eviction Runs Mills</td>
<td>(int)</td>
</tr>
<tr>
<td>Number of Tests Per Eviction Run</td>
<td>(int)</td>
</tr>
<tr>
<td>Minimum Evictable Idle Time</td>
<td>(int)</td>
</tr>
<tr>
<td>Property</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Remove Abandoned</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Remove Abandoned Timeout</td>
<td>(int)</td>
</tr>
<tr>
<td>Log Abandoned</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Auto Commit</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Default Read Only</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Default Catalog</td>
<td>(String)</td>
</tr>
<tr>
<td>Validator Class Name</td>
<td>(String)</td>
</tr>
<tr>
<td>Connection Properties</td>
<td>(String)</td>
</tr>
<tr>
<td>Init SQL</td>
<td></td>
</tr>
<tr>
<td>JDBC Interceptors</td>
<td></td>
</tr>
<tr>
<td>Validation Interval</td>
<td>(long)</td>
</tr>
<tr>
<td>JMX Enabled</td>
<td>(boolean)</td>
</tr>
<tr>
<td>Fair Queue</td>
<td>(boolean)</td>
</tr>
</tbody>
</table>
Abandon When Percentage Full  
(int) Connections that have been abandoned (timed out) won't get closed and reported up unless the number of connections in use are above the percentage defined by abandonWhenPercentageFull. The value should be between 0-100. The default value is 0, which implies that connections are eligible for closure as soon as removeAbandonedTimeout 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 now - time-when-connected > maxAge has been reached, and if so, it closes the connection rather than returning it to the pool. The default value is 0, 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 ProxyConnection class to use String.equals and set to false when you wish to use == when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.

Suspect Timeout  
(int) Timeout value in seconds. Default value is 0. Similar to to the removeAbandonedTimeout value but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if logAbandoned is set to true. If this value is equal or less than 0, no suspect checking will be performed. Suspect checking only takes place if the timeout value is larger than 0 and the connection was not abandoned or if abandon check is disabled. If a connection is suspect a WARN message gets logged and a JMX notification gets sent once.

Alternate User Name Allowed  
(boolean) By default, the jdbc-pool will ignore the DataSource.getConnection(username,password) call, and simply return a previously pooled connection under the globally configured properties username and password, for performance reasons.

The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the DataSource.getConnection(username,password) call, simply set the property alternateUsernameAllowed to true. Should you request a connection with the credentials user1/password1 and the connection was previously connected using different user2/password2, the connection will be closed, and reopened with the requested credentials. This way, the pool size is still managed on a global level, and not on a per schema level. The default value is false.


Configuring a Custom Datasource

This topic explains the purpose of using custom datasources and how you can define custom datasource implementations using the management console. Alternatively, you can simply create datasources using the default RDBMS configuration provided in WSO2 products.

- About custom datasources
- Creating custom datasources

About custom datasources

Custom datasources allows you to define your own datasource implementation. There are two options for writing a custom datasource, and these two options cover most of the common business use cases as follows:

- Custom tabular datasources: Used to represent data in tables, where a set of named tables contain data rows that can be queried later. A tabular datasource is typically associated with an SQL data services query. This is done by internally using our own SQL parser to execute SQL against the custom datasource. You can use the org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS interface to implement tabular datasources. For a sample implementation of a tabular custom datasource, see org.wso2.carbon.data
Creating custom datasources

You can create custom data sources as shown below.

1. Go to the Configure tab on the management console and click Data Sources to open the Data Sources screen.
2. Then click Add Data Source. The following screen will open:

3. Enter "Custom" as the datasource type.
4. In the Custom Data Source Type field, enter "DS_CUSTOM_TABULAR" (to store data in tables) or "DS_CUSTOM_QUERY" (to store non-tabular data accessed through a query).
5. In the Name and Description fields, enter a unique name for the datasource.
6. In the Configuration section, specify the xml configuration of the datasource. See the examples given below.
   - XML configuration for a custom tabular datasource (DS_CUSTOM_TABULAR type):
<configuration>
  <customDataSourceClass>org.wso2.carbon.dataservices.core.custom.datasource.InMemoryDataSource</customDataSourceClass>
  <customDataSourceProps>
    <property name="inmemory_datasource_schema">{Vehicles:[ID,Model,Classification,Year]}
      <property name="inmemory_datasource_records">
        [Vehicles:["S10_1678","Harley Davidson Ultimate Chopper","Motorcycles","1969"],
        ["S10_1949","Alpine Renault 1300","Classic Cars","1952"],
        ["S10_2016","Moto Guzzi 1100i","Motorcycles","1996"],
        ["S10_4698","Harley-Davidson Eagle Drag Bike","Motorcycles","2003"],
        ["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>
    </property>
  </customDataSourceProps>
</configuration>

- XML configuration for a custom query datasource (DS_CUSTOM_QUERY):

<configuration>
  <customDataSourceClass>org.wso2.carbon.dataservices.core.custom.datasource.EchoDataSource</customDataSourceClass>
  <customDataSourceProps>
    <property name="p1">val1</property>
    <property name="p2">val2</property>
  </customDataSourceProps>
</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.

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

**Working with Security**
This section explains how you can set up the security configurations in your storage server.

After you install WSO2 SS, it is recommended to change the default security settings according to the requirements of your production environment. Since SS is built on top of the WSO2 Carbon Kernel (version 4.3.0), the main security configurations applicable to SS are inherited from the Carbon kernel.

You can find detailed information on how to configure security in your SS as explained below.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring transport-level security</td>
<td>WSO2 products support a variety of transports that make them capable of receiving and sending messages over a multitude of transport and application-level protocols. By default, most of the WSO2 products come with the HTTP transport. The transport receiver implementation of the HTTP transport is available in Carbon. The transport sender implementation comes from the Tomcat HTTP connector, which is configured in the <code>&lt;AS_HOME&gt;/repository/conf/tomcat/catalina-server.xml</code> file. This also covers how you can protect your system from common security attacks. For example, 'Poodle' is a bug in the SSL version 3 protocol, which exposes critical data encrypted between clients and servers. 'Logjam' is a security threat (man-in-the-middle attack), which is caused by weak ciphers. These security vulnerabilities can be avoided by configuring transport-level security. For more information on securing the HTTP transport, see the topic on configuring transport level security in the WSO2 Carbon documentation.</td>
</tr>
</tbody>
</table>
| Configuring keystores                | A keystore is a repository that stores the cryptographic keys and certificates. These artifacts are used for encrypting sensitive information, and establishing trust between your server and outside parties that connect to your server. All WSO2 products come with a default keystore (`wso2carbon.jks`). In a production environment, it is recommended to replace it with a new keystore. You can also configure multiple keystores for different purposes. See the following in the WSO2 Carbon documentation:
  - Learn how public key encryption and keystores are used.
  - Learn how to create new keystores and replace the default one.
  - Learn how configuration files should be updated to use the relevant keystore for different purposes. |
| Securing sensitive passwords         | As a secure vault implementation is available in all WSO2 products, you can encrypt the sensitive data such as passwords in configuration files using the Cipher tool. See the following in the WSO2 Carbon documentation:
  - How the secure vault is implemented in WSO2 products.
  - How to encrypt passwords using the Cipher tool.
  - How to resolve encrypted passwords. |
| Enabling JAVA security manager       | See the topic on enabling JAVA security manager in the WSO2 Carbon documentation for details on how to prevent untrusted code from manipulating your system. |

**Logging Management**
This section provides the following information:

- Introduction to Logging
- Logging Configuration

Introduction to Logging

Logging is one of the most important aspects of a production-grade server. A properly configured logging system is vital in identifying errors, security threats and usage patterns. There are several ways in which the system and application logs of a running Carbon instance can be viewed.

- Through the Management Console.
- Through the log files that are stored in <PRODUCT_HOME>/repository/logs folder. The folder contains current logs in a log file with a date stamp. Older logs are archived in "wso2carbon.log" file.
- Through the command prompt/shell terminal that opens when running the "wso2server.bat"/"wso2server.sh" files to start the Carbon server.

WSO2 products use a log4j-based logging mechanism through Apache Commons Logging facade library. The log4j.properties file which governs how logging is performed by the server can be found in <PRODUCT_HOME>/repository/conf directory.

There are two ways to configuring log4j.

- Manually editing the log4j.properties.
- Logging configuration through the management console. Changes apply at run time.

The latter is recommended because all changes made to Log4j through the management console are persisted in the WSO2 Registry. Therefore, those changes will be available after server restarts. Any changes to the logging configuration you make through the management console will get priority over what is defined in the actual log4j.properties file. However, if you modify the log4j.properties file and restart the server, the earlier log4j configuration which 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.

Logging functionality is provided by the following feature in the WSO2 feature repository:

- **Name**: WSO2 Carbon - Logging Management Feature
- **Identifier**: org.wso2.carbon.logging.mgt.feature.group

If the above feature is not bundled in your product by default, you can install it using the instructions given in section Feature Management.

Logging Configuration

In most systems, logging properties should be specified before the server starts and cannot be changed while it is running. However, logging properties of a running Carbon instance can be changed through its management console, while the server is up and running.

There are three main components important when configuring log4j. They are Loggers, Appenders, and Layouts. The Logging Management feature of WSO2 enables to change these parameters both globally and individually, at
run time. When you change the parameters using the Management Console, 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.

Follow the instructions below to configure logging properties of the system and application logs of a Carbon server at run time.

1. Log on to the product's management console and select "Configure -> Logging."

2. The "Logging Configuration" window appears.
3. The "Logging Configuration" window has three sections:

- **Global Log4J Configuration**
- **Configure Log4J Appenders**
- **Configure Log4J Loggers**

Using these sections, you can configure the layout and the amount of information about the system activity you want to receive.

**Global Log4J Configuration**

This section allows you to assign a single log level and log pattern to all loggers.

- **Log Level** - Severity of the message. Reflects a minimum level that this logger cares about. You can view the hierarchy of 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 the "Restore Defaults" button, the Registry will be overwritten by logging configurations specified in the log4j.properties file.

**Configure Log4J Appenders**

This section allows you to configure appenders individually. Log4j allows logging requests to print to multiple destinations. These output destinations are called "Appenders." You can attach several appenders to one logger.

- **Name** - The name of an appender. By default, WSO2 Application Server comes with the following log appenders configured:
  - **CARBON_CONSOLE** - Logs to the console when the server is running.
  - **CARBON_LOGFILE** - Writes the logs to AS_HOME/repository/logs/wso2carbon.log.
- **CARBON_MEMORY**
- **CARBON_SYS_LOG** - Allows separation of the software that generates messages from the system that stores them and the software that reports and analyzes them.
- **CARBON_TRACE_LOGFILE**

- **Log pattern** - Defines the output format of the log file.
- **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 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.

The Available Categories of Logs you can View are:

- **TRACE** - Designates fine-grained informational events than the DEBUG.
- **DEBUG** - Designates fine-grained informational events that are most useful to debug an application.
- **INFO** - Designates informational messages that highlight the progress of the application at coarse-grained level.
- **WARN** - Designates potentially harmful situations.
- **ERROR** - Designates error events that might still allow the application to continue running.
- **FATAL** - Designates very severe error events that will presumably lead the application to abort.

**Configure Log4J Loggers**

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 one. If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. Refer to hierarchy of levels above.
- **Additivity** - Allows to inherit all the appenders of the parent Logger if set as "True.”
Monitoring the Server

This section provides various dashboards and statistics for monitoring the server as follows:

- System Logs
- Application Logs

Note that some of the above statistics may not be available in some WSO2 products, depending on the availability of the relevant feature in its distribution. If you want a particular functionality which is not bundled with the distribution by default, you need to install the relevant feature using the Configure -> Features menu in your product's Management Console.

For information on monitoring Cassandra statistics, see here.

System Logs

The "System Logs" page displays all events of a running Carbon instance. Further, it facilitates downloading and viewing log files according to user preferences. 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).

You can refer to more information about logs and how to set log properties in section Logging. Statistics on system logs are provided by the following feature:

Name: WSO2 Carbon - Logging Management Feature
Identifier: org.wso2.carbon.logging.mgt.feature.group

If it is not bundled in your product by default, you can install the above feature using the instructions given in section Feature Management.

Follow the instructions below to access statistics on System Logs.

1. Log on to the product's management console and select "Monitor -> System Logs."
2. The "System Logs" page appears. This page displays logs in a bottom-up manner.

   The log messages displayed on this page are obtained from a memory appender. Hence, 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, refer to http://logging.apache.org/log4j

3. To view old archived logs, click the "Show archived logs" tab at the bottom of the "System Logs" page. The
"Download" link can be used to download the "wso2carbon.log" file at location <PRODUCT_HOME>/repository/logs.

4. In the “View” list, select the category of logs you want to view. The available categories are:
   - TRACE - Trace messages.
   - DEBUG - Debug messages.
   - INFO - Information messages.
   - WARN - Warning messages.
   - ERROR - Error messages.
   - FATAL - Fatal error messages.

5. You can also find a certain log using the search function. Enter a keyword (or part of a keyword) and click "Search."

The location of the log files on disk is specified in the log4j.configuration file.

Application Logs

Application logs are where events invoked by an application or a program running in a system are recorded. Similarly, the Application Logs of a running Carbon instance display the log events of its deployed Web applications and Web services. The "Application Logs" page has been introduced as a fine-grained view of system logs. While system logs display log events of the entire system holistically, the application logs page allows the user to select a particular application and view its logs only.

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).

For more information on logs and how to change log properties according to your preferences, refer to section Logging.

Statistics on application logs are provided by the following feature:

Name: WSO2 Carbon - Logging Management Feature
Identifier: org.wso2.carbon.logging.mgt.feature.group

If the above feature is not bundled in your product by default, you can install it using the instructions given in section, Feature Management.

Follow the instructions 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. For example,

   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, refer to http://logging.apache.org/log4j
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.

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.

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 criteria is given, the "View" is displayed as "Custom".

   The location of the log files on disk is specified in the log4j,configuration file.

Configuring Multiple Tenants

The goal of multitenancy is to maximize resource sharing by allowing multiple users (tenants) to log in and use a single server/cluster at the same time, in a tenant-isolated manner. That is, each user is given the experience of using his/her own server, rather than a shared environment. Multitenancy ensures optimal performance of the system's resources such as memory and hardware and also secures each tenant's personal data.

You can register tenant domains using the Management Console of WSO2 products.

This section covers the following topics:

- Managing Tenants
- Multi Tenant Architecture
- Tenant-aware Load Balancing Using the WSO2 Elastic Load Balancer

Managing Tenants

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

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

2. Enter the tenant information in Register A New Organization screen as follows, and click Save.
### Parameter Name | Description
---|---
**Domain** | 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.

**Select Usage Plan for Tenant** | The usage plan defines limitations (such as number of users, bandwidth etc.) for the tenant.

**First Name/Last Name** | The name of the tenant admin.

**Admin Username** | The login username of the tenant admin. The username always ends with the domain name (e.g., admin@abc.com)

**Admin Password** | The password used to log in using the admin username specified.

**Admin Password (Repeat)** | Repeat the password to confirm.

**Email** | 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**.

---

### Multi Tenant 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.

**Architecture**

A tenant is an isolated domain. The users within this domain can manage their own data and perform their own transactions without being affected by actions carried out in other domains.
These domains are allocated server space from the complete server space of a WSO2 product instance which is referred to as the super tenant.

The super tenant as well as each individual tenant has its own configuration and context module.

Each tenant has its own security domain. A domain has a set of users, and permissions for those users to access resources. Thus, a tenant is restricted by the users and permissions of the domain assigned to it. The artifact repositories of the tenants are separated from each other.

An individual tenant can carry out the following activities within the boundaries of its own configuration and context module:

- Deploying artifacts
- Applying security
- User management
- Data management
- Request throttling
- Response caching

WSO2 Carbon provides a number of Admin services which have special privileges to manage the server. These admin services are deployed in the super tenant. Other tenants can make use of these admin services to manage their deployment. The admin services operate in a tenant aware fashion. Thus, privileges and restrictions that apply to any client using an admin service are taken into account.

**Resource sharing**

WSO2 Carbon supports the following methods for sharing resources among tenants:

- **Private Jet mode**: This method allows the load of a tenant ID to be deployed in a single tenant mode. A single tenant is allocated an entire service cluster. The purpose of this approach is to allow special privileges (such as priority processing and improved performance) to a tenant.
- **Separation at hardware level**: This method allows different tenants to share a common set of resources, but each tenant has to run its own operating system. This approach helps to achieve a high level of isolation, but it also incurs a high overhead cost.
- **Separation at JVM level**: This method allows tenants to share the same operating system. This is done by enabling each tenant to run a separate JVM instance in the operating system.
- **Native multi-tenancy**: This method involves allowing all the tenants to share a single JVM instance. This method minimises the overhead cost.
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 inactiveness by adding `-Dtenant.idle.time=<time_in_minutes>` java property to the startup scrip 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 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.

**Tenant-aware Load Balancing Using the WSO2 Elastic Load Balancer**

WSO2 Elastic Load Balancer is currently retired.

Tenant partitioning is required in a clustered deployment to be able to scale to large numbers of tenants. There can be multiple clusters for a single service and each cluster would have a subset of tenants as illustrated in the diagram below. In such situations, the load balancers need to be tenant aware in order to route the requests to the required tenant clusters. They also need to be service aware since it is the service clusters which are partitioned according to the clients.

The following example further illustrates how this is achieved in WSO2 Elastic Load Balancer (ELB).
A request sent to a load balancer has the following host header to identify the cluster domain:

https://appserver.cloud-test.wso2.com/carbon.as1.domain/carbon/admin/login.jsp

In this URL:

- appserver.cloud-test.wso2.com is the service domain which allows the load balancer to identify the service.
- carbon.as1.domain.com is the tenant domain which allows the load balancer to identify the tenant.

Services are configured with their cluster domains and tenant ranges in the in ELB_HOME/repository/conf/loadbalancer.conf file. These cluster domains and tenant ranges are picked by the load balancer when it loads.

The following is a sample configuration of the loadbalancer.conf file.
In the above configuration, there is a host address which maps to the application server service. If required, you can enter multiple host addresses separated by commas.

There are two cluster domains defined in the configuration. The cluster domain named `carbon.as1.domain` is used to load the range of tenants with IDs 1-100. The other cluster domain named `carbon.as2.domain` is used to load the tenants with IDs 101-200.

If the tenant ID of `abc.com` is 22, the request will be directed to the `Carbon.AS1.domain` cluster.

## Performance Tuning

This section describes some recommended performance tuning configurations to optimize WSO2 Storage Servicer. It assumes that you have set up WSO2 SS on a server running Unix/Linux, which is recommended for production deployment.

- OS-Level Settings
- JVM Settings
- JDBC Pool Configuration
- Setting the thread execution limit for multitenant mode

### Important

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to familiarize yourself with these files using Unix/Linux documentation before editing them.
- The 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 SS accordingly.

### OS-Level Settings

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.
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 /etc/security/limits.conf file of Linux.

```bash
* 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 /etc/security/limits.conf file of Linux (be sure to include the leading * character). Each carbon server instance you run would require up to 1024 threads (with default thread pool configuration). Therefore, you need to increase the nproc value by 1024 per each carbon server (both hard and soft).

```bash
* soft nproc 20000
* hard nproc 20000
```

**JVM Settings**

If one or more worker nodes in a clustered deployment require access to the management console, you need to increase the entity expansion limit in the `<SS_HOME>/bin/wso2server.bat` file (for Windows) or the `<SS_HOME>/bin/wso2server.sh` file (for Linux/Solaris) as show below. The default entity expansion limit is 64000.

```bash
-DentityExpansionLimit=100000
```

**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 utilising resources.
The following parameters should be considered when tuning the connection pool:

- The application's concurrency requirement.
- The average time taken to run 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 Storage Server using the `master-datasources.xml` file stored in the `<SS_HOME>/repository/conf/datasource/` folder. For more details about recommended JDBC configurations, see [Tomcat JDBC Connection Pool](https://tomcat.apache.org/tomcat-9.0-doc/jdbc-config.html).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Tuning Recommendations</th>
</tr>
</thead>
</table>
| maxActive | The maximum number of active connections that can be allocated from the connection pool at the same time. The default value is 100. | The maximum latency (approximately) = \( \frac{P}{M} \times T \), where,  
- \( M \) = maxActive value  
- \( P \) = Peak concurrency value  
- \( T \) = Time (average) taken to process a query.  

Therefore, by increasing the maxActive value (up to the expected highest number of concurrency), the time that requests wait in the queue for a connection to be released will decrease. But before increasing the Max. Active value, consult the database administrator, as it will create up to maxActive connections from a single node during peak times, and it may not be possible for the DBMS to handle the accumulated count of these active connections.  

Note that this value should not exceed the maximum number of requests allowed for your database. |
| maxWait | The maximum time that requests are expected to wait in the queue for a connection to be released. This property comes into effect when the maximum number of active connections allowed in the connection pool (see maxActive property) is used up. | 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,  
If the maximum latency (approximately) = \( \frac{P}{M} \times T \), where,  
- \( M \) = maxActive value,  
- \( P \) = Peak concurrency value,  
- \( T \) = Time (average) taken to process a query,  

then, the maxWait = \( \frac{P}{M} \times T \) + buffer time. |
| minIdle | The minimum number of connections that can remain idle in the pool, without extra ones being created. The connection pool can shrink below this number if validation queries fail. Default value is 0. | This value should be similar or near to the average number of requests that will be received by the server at the same time. With this setting, you can avoid having to open and close new connections every time a request is received by the server. |
maxIdle | The maximum number of connections that can remain idle in the pool. | The value should be less than the maxActive value. For high performance, tune maxIdle to match the number of average, concurrent requests to the pool. If this value is set to a large value, the pool will contain unnecessary idle connections.

testOnBorrow | The indication of whether connection objects will be validated before they are borrowed from the pool. If the object validation fails, the connection is dropped from the pool, and there will be an attempt to borrow another connection. | When the connection to the database is broken, the connection pool does not know that the connection has been lost. As a result, the connection pool will continue to distribute connections to the application until the application actually tries to use the connection. To resolve this problem, set “Test On Borrow” to “true” and make sure that the “ValidationQuery” property is set. To increase the efficiency of connection validation and to improve performance, validationInterval property should also be used.

validationInterval | This parameter controls how frequently a given validation query is executed (time in milliseconds). The default value is 30000 (30 seconds). That is, if a connection is due for validation, but has been validated previously within this interval, it will not be validated again. | Deciding the value for the “validationInterval” depends on the target application’s behavior. Therefore, selecting a value for this property is a trade-off and ultimately depends on what is acceptable for the application.

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.

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.

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.
Setting the thread execution limit for multitenant mode

In multitenant mode, 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 `<SS_HOME>/repository/conf/tomcat/catalina-server.xml` file as shown below.

```xml
<Valve
    className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve"
    threshold="600"/>
```

- The `className` is the Java class name used for the implementation. This must be set to `org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve`.
- The `threshold` gives the minimum duration in seconds after which a thread is considered stuck. Default value is 600 seconds.

Calling Admin Service from Apps

WSO2 products are managed internally using SOAP Web services known as **admin services**. WSO2 products come with a management console UI, which communicates with these admin services to facilitate administration capabilities through the UI.

A service in WSO2 products is defined by the following components:

- Service component: provides the actual service
- UI component: provides the Web user interface to the service
- Service stub: provides the interface to invoke the service generated from the service WSDL

There can be instances where you want to call back-end Web services directly. For example, in test automation, to minimize the overhead of having to change automation scripts whenever a UI change happens, developers prefer to call the underlying services in scripts. The topics below explain how to discover and invoke these services from your applications.

Discovering the admin services

By default, the WSDLs of admin services are hidden from consumers. Given below is how to discover them.

1. Set the `<HideAdminServiceWSDLs>` element to `false` in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file.
2. Restart the server.
3. Start the WSO2 product with the `-DosgiConsole` option, such as `sh <PRODUCT_HOME>/bin/wso2server.sh -DosgiConsole` in Linux.
4. When the server is started, hit the enter/return key several times to get the OSGI shell in the console.
5. In the OSGI shell, type: `osgi> listAdminServices`
6. The list of admin services of your product are listed. For example:
6. To see the service contract of an admin service, select the admin service’s URL and then paste it in your browser with \(?wsdl\) at the end. For example:
https://localhost:9443/services/UserAdmin\?wsdl

In products like WSO2 ESB and WSO2 API Manager, the port is 8243 (assuming 0 port offset). However, you should be accessing the Admin Services via the management console port, which is 9443 when there is no port offset.

7. Note that the admin service's URL appears as follows in the list you discovered in step 6:


Invoking an admin service

Admin services are secured using common types of security protocols such as HTTP basic authentication, WS-Security username token, and session based authentication to prevent anonymous invocations. For example, the UserAdmin Web service is secured with the HTTP basic authentication. To invoke a service, you do the following:

1. Authenticate yourself and get the session cookie.
2. Generate the client stubs to access the back-end Web services.

To generate the stubs, you can write your own client program using the Axis2 client API or use an existing tool like SoapUI (4.5.1 or later) or wsdl2java.

The wsdl2java tool, which comes with WSO2 products by default hides all the complexity and presents you with a proxy to the back-end service. The stub generation happens during the project build process within the Maven POM files. It uses the Maven ant run plug-in to execute the wsdl2java tool.

You can also use the Java client program given here to invoke admin services. All dependency JAR files that you need to run this client are found in the /lib directory.

Authenticate the user

The example code below authenticates the user and gets the session cookie:
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
import org.wso2.carbon.authenticator.stub.AuthenticationAdminStub;
import org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.apache.axis2.context.ServiceContext;
import java.rmi.RemoteException;
public class LoginAdminServiceClient {
    private final String serviceName = "AuthenticationAdmin";
    private AuthenticationAdminStub authenticationAdminStub;
    private String endPoint;

    public LoginAdminServiceClient(String backEndUrl) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        authenticationAdminStub = new AuthenticationAdminStub(endPoint);
    }

    public String authenticate(String userName, String password) throws RemoteException,
    LoginAuthenticationExceptionException {
        String sessionCookie = null;
        if (authenticationAdminStub.login(userName, password, "localhost")) {
            System.out.println("Login Successful");
            ServiceContext serviceContext = authenticationAdminStub._getServiceClient().getLastOperationContext().getServiceContext();
            sessionCookie = (String) serviceContext.getProperty(HTTPConstants.COOKIE_STRING);
            System.out.println(sessionCookie);
        }
        return sessionCookie;
    }

    public void logOut() throws RemoteException,
    LogoutAuthenticationExceptionException {
        authenticationAdminStub.logout();
    }
}
Generate the client stubs

After authenticating the user, give the retrieved admin cookie with the service endpoint URL as shown in the sample below. The service management service name is ServiceAdmin. You can find its URL (e.g., https://localhost:9443/services/ServiceAdmin) in the service.xml file in the META-INF folder in the respective bundle that you find in <PRODUCT_HOME>/repository/components/plugins.

```java
import org.apache.axis2.AxisFault;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.wso2.carbon.service.mgt.stub.ServiceAdminStub;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaDataWrapper;
import java.rmi.RemoteException;

public class ServiceAdminClient {
    private final String serviceName = "ServiceAdmin";
    private ServiceAdminStub serviceAdminStub;
    private String endPoint;

    public ServiceAdminClient(String backEndUrl, String sessionCookie) throws AxisFault {
        this.endPoint = backEndUrl + "/services/" + serviceName;
        serviceAdminStub = new ServiceAdminStub(endPoint);
        //Authenticate Your stub from sessionCookie
        ServiceClient serviceClient;
        Options option;

        serviceClient = serviceAdminStub._getServiceClient();
        option = serviceClient.getOptions();
        option.setManageSession(true);
        option.setProperty(org.apache.axis2.transport.http.HTTPConstants.COOKIE_STRING, sessionCookie);

        option.setProperty(org.apache.axis2.transport.http.HTTPConstants.COOKIE_STRING, sessionCookie);
    }

    public void deleteService(String[] serviceGroup) throws RemoteException {
        serviceAdminStub.deleteServiceGroups(serviceGroup);
    }

    public ServiceMetaDataWrapper listServices() throws RemoteException {
        return serviceAdminStub.listServices("ALL", "*", 0);
    }
}
```

The AuthenticationAdminStub class requires org.apache.axis2.context.ConfigurationContext as a parameter. You can give a null value there.
The following sample code lists the back-end Web services:

```java
import
org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import
org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaData;
import org.wso2.carbon.service.mgt.stub.types.carbon.ServiceMetaDataWrapper;
import java.rmi.RemoteException;

public class ListServices {
    public static void main(String[] args)
        throws RemoteException, LoginAuthenticationExceptionException,
                  LogoutAuthenticationExceptionException {
        System.setProperty("javax.net.ssl.trustStore",
                          "$ESB_HOME/repository/resources/security/wso2carbon.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "wso2carbon");

        System.setProperty("javax.net.ssl.trustStoreType", "JKS");
        String backEndUrl = "https://localhost:9443";

        LoginAdminServiceClient login = new
                            LoginAdminServiceClient(backEndUrl);
        String session = login.authenticate("admin", "admin");
        ServiceAdminClient serviceAdminClient = new
                                    ServiceAdminClient(backEndUrl, session);
        ServiceMetaDataWrapper serviceList =
                              serviceAdminClient.listServices();
        System.out.println("Service Names:");
        for (ServiceMetaData serviceData : serviceList.getServices()) {
            System.out.println(serviceData.getName());
        }

        login.logOut();
    }
}
```

Customizing the Management Console

The Management Console user interface (https://localhost:9443/carbon) of a Carbon product consists of two layers:

1. **UI inherited from WSO2 Carbon platform** contains the templates, styles (css files), and images that are stored in the core Carbon UI bundle stored in `<PRODUCT_HOME>/repository/components/plugins/org.wso2.carbon.ui_<version-number>.jar` where `<version-number>` is the version of the Carbon kernel that the product is built on. This bundle is responsible for the overall look and feel of the entire Carbon platform.
2. **UI unique to each product** contains all the styles and images that override the ones in core Carbon platform. This file is in `<PRODUCT_HOME>/repository/components/plugins/org.wso2.<product-name>.styles_<version-number>.jar` where `<version-number>` is the version of the product.

The following topics explain how to download a Carbon product and customize its user interface.

- Setting up the development environment
- Customizing the user interface
- Starting the server

**Setting up the development environment**

To download and set up the product environment for editing, take the following steps.

1. Download your product.
2. Extract the ZIP file into a separate folder in your hard drive.
3. Go to the `<PRODUCT_HOME>/repository/components/plugins/` directory to find the required JAR files:
   - `org.wso2.carbon.ui_<version-number>.jar`
   - `org.wso2.<product-name>.styles_<version-number>.jar`
4. Copy the JAR files to a separate location on your hard drive. Since the JAR files are zipped, you must unzip them to make them editable.

You can now customize the look and feel of your product by modifying the contents of the JAR files as described in the next section.

**Customizing the user interface**

Customizing the product interface involves changing the layout/design of the Carbon framework as well as changing the styles and images specific to the product. The following topics explain how some of the main changes to the product interface can be done.

- Changing the layout
- Changing the styles on the Carbon framework
- Changing the product specific styles and images

**Changing the layout**

The layout of the Carbon framework is built using a tiles JSP tag library. The use of tiles allows us to break the presentation of the layout into small JSP snippets that perform a specific function. For example, `header.jsp` and `footer.jsp` are the tiles corresponding to the header and footer in the layout. The `template.jsp` file controls the main layout page of the Carbon framework, which holds all the tiles together. That is, the header part in the `template.jsp` file is replaced with the `<tiles:insertAttribute name="header"/>` tag, which refers to the `header.jsp` file. The `template.jsp` file as well as the JSP files corresponding to the tiles are located in the `org.wso2.<product-name>.styles_<version-name>.jar/web/admin/layout/` directory.

Therefore, changing the layout of your product primarily involves changing the `template.jsp` page (main layout page) and the JSP files of the relevant JSP tiles.

Ensure that you do not change or remove the ID attributes on the `.jsp` files.

**Changing the styles on the Carbon framework**

The `global.css` file, which determines the styles of the Carbon framework, is located in the `org.wso2.carbon.`
ui_<version-name>.jar/web/admin/css/ directory. You can edit this file as per your requirement. Alternatively, you can apply a completely new stylesheet to your framework instead of the default global.css stylesheet.

To apply a new style sheet to the carbon framework:

1. Copy your new CSS file to this same location.
2. Open the template.jsp file located in the org.wso2.carbon.ui_<version-name>.jar/web/admin/layout/ directory, which contains the main layout of the page and the default JavaScript libraries.
3. Replace global.css with the new style sheet by pointing the String globalCSS attribute to the new stylesheet file.

```java
//Customization of UI theming per tenant
String tenantDomain = null;
String globalCSS = "./admin/css/global.css";
String mainCSS = "";
```

**Changing the product specific styles and images**

The styles and images unique to your product are located in the org.wso2.<product-name>.styles_<version-number>.jar folder. To modify product specific styles and images, take the following steps.

1. Copy the necessary images to the org.wso2.<product-name>.styles_<version-number>.jar/web/styles/images/ directory. For example, if you want to change the product banner, add the new image file to this directory.
2. Open the main.css file located in the org.wso2.<product-name>.styles_<version-number>.jar/web/styles/css/ directory.
3. To specify a new product banner, change the background-image attribute of org.wso2.<product-name>.styles_<version-number>.jar/web/styles/css/main.css file as follows:

```css
/* ---------------- header styles ------------------ */
div#header-div {
    background-image: url( ../images/newproduct-header-bg.png);
    height:70px;
}
```

Note that the size of the images you use will affect the overall UI of your product. For example, if the height of the product logo image exceeds 28 pixels, you must adjust the main.css file in the org.wso2.<product-name>.styles_<version-name>.jar/web/styles/css/ directory to ensure that the other UI elements of your product aligns with the product logo.

**Starting the server**

In the preceding steps, you have done the changes to the product interface after copying the JAR files to a separate location on your hard drive. Therefore, before you start your production server, these files must be correctly copied back to your production environment as explained below.

2. Change the name of the ZIP file to org.wso2.carbon.ui_<version-number>.jar and org.wso2.<product-name>.styles_<version-number>.jar respectively.
3. Copy these two new JAR files to the `<PRODUCT_HOME>/repository/components/plugins/` directory in your product installation.
4. Start the server.

**Service Principal Management**

Kerberos is an authentication protocol used to secure communication and enable secure user credential exchange in Web services. It also provides mutual authentication where a server can authenticate itself to the client. Kerberos protocol uses a trusted third-party called 'Key Distribution Center' (KDC), which consists two logical parts as follows.

- Authentication Server
- Ticket Granting Server

Actual user credentials are stored in the 'authentication server' and the communicating parties retrieve Kerberos tickets from the 'ticket granting server', which validates ticket-requesting user credentials with the authentication server.

The Kerberos Security feature uses ApacheDS based LDAP server user store. It also has an ApacheDS based on KDC implementation. Users can use a running Carbon instance with this feature installed as a KDC and implement Kerberos security to secure communications between services and clients.

**Enabling key distribution center**

Embedded LDAP shipped by default with WSO2 Carbon-based products can be used for KDC, which is disabled by default. KDC can be enabled by changing configurations in two places as follows.

1. Locate the configuration section "UserStoreManager" in `$CARBON_HOME/repository/conf/user-mgt.xml` and set the property "kdcEnabled" to true.

```xml
<Property name="kdcEnabled">true</Property>
```

2. Locate the `<KDCServer/>` XML tag in `$CARBON_HOME/repository/conf/embedded-ldap.xml` and set "enabled" to "true".

```xml
<!-- KDC configurations -->
<KDCServer>
  <Property name="name">defaultKDC</Property>
  <Property name="enabled">true</Property>
  <Property name="protocol">UDP</Property>
  <Property name="host">localhost</Property>
  <Property name="port">8000</Property>
  <Property name="maximumTicketLifeTime">8640000</Property>
  <Property name="maximumRenewableLifeTime">604800000</Property>
  <Property name="preAuthenticationTimeStampEnabled">true</Property>
</KDCServer>
```

**Service principals**

The KDC Server issues tickets to access a particular service. Therefore, the services are associated with a service name. In Kerberos context, the service name is called as Service Principal Name (SPN). Before assigning a SPN to a service, define Service Principal Name in KDC. Follow the instructions below to add a new service principle.
1. Log on to the product's Management Console and select "Configure -> Kerberos KDC -> Service Principals."

2. The "Service Principal Management" page appears. Click on the "Add New Service Principal" link to begin.

3. In the "Add Service Principal" window that appears, enter the required details and click "Finish" to complete.

4. The newly-added service principle will be displayed in the 'Service Principle Management' window. You can change its password or delete it by clicking the respective links associated with each principle.
Cassandra Migration Guide

This guide explains the Cassandra migration steps from Cassandra version 1.1.3 to version 1.2.13.

- Setting up Parallel SSH tool
- Backing up data from Cassandra source
- Migrating schema
- Migrating backed-up data to new product nodes

Setting up Parallel SSH tool

You need to install Parallel SSH tool in a central point to execute the NodeTool commands in all product nodes. This is done as to take the backup in all product nodes at the same time, to avoid any data inconsistencies. Follow the steps below to set up the Parallel SSH tool.

1. Download pssh-2.3.1.tar.gz file and install Parallel SSH on the client machine.
2. Create a text file (for example: host.txt) in <PSSH_HOME>/bin/ directory, and add all the host names of the product nodes in it.

   <PSSH_HOME> refers to the installation directory of Parallel SSH.

3. Run the following command to check if Parallel SSH is functioning as expected:

   ./pssh-2.3.1/bin/pssh -h ./pssh-2.3.1/bin/host.txt -A -v -l <ssh_username> -o <output_dir> uptime

   <ssh_username> refers to the username of the user account, and <output_dir> refers to the folder, in which all the output text files will be saved.

4. Enter the password of your user account on the machine.

   For example, in a scenario with a setup of three product nodes, the following output will be displayed if Parallel SSH is functioning as expected:

   
   
   [1] 06:08:37 [SUCCESS] 192.168.5.20

Backing up data from Cassandra source

1. Download Apache Cassandra 1.1.3, and copy the installation directory to the same location of all the product nodes.
2. Navigate to one of the <CASSANDRA_HOME>/bin/ directories, and enter the following command to log in to Cassandra via NodeTool:

   /nodetool -h <host> -p 9999 -u <username> -pw <password> ring

   <CASSANDRA_HOME> refers to the extracted installation directory of Apache Cassandra 1.1.3.
The following output will be displayed if NodeTool is operating successfully:

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>Rack</th>
<th>Status</th>
<th>State</th>
<th>Load</th>
<th>Effective Ownership</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.57.244.224</td>
<td>datacenter1</td>
<td>rack1</td>
<td>Up</td>
<td>Normal</td>
<td>498.19MB</td>
<td>100.00%</td>
<td>0</td>
</tr>
<tr>
<td>10.57.244.226</td>
<td>datacenter1</td>
<td>rack1</td>
<td>Up</td>
<td>Normal</td>
<td>502.76MB</td>
<td>100.00%</td>
<td>0</td>
</tr>
<tr>
<td>56713727820156410577229101238628035242</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.57.244.229</td>
<td>datacenter1</td>
<td>rack1</td>
<td>Up</td>
<td>Normal</td>
<td>500.91MB</td>
<td>100.00%</td>
<td>0</td>
</tr>
<tr>
<td>113427455640312821154458202477256070485</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Navigate to the `<PSSH_HOME>` folder and enter the following command, to check if the above command works with the Parallel SSH tool.

```bash
./pssh-2.3.1/bin/pssh -h ./pssh-2.3.1/bin/host.txt -A -v -l <ssh_username> -o <output_directory> bash
<CASSANDRA_HOME>/bin/nodetool -h <host> -p 9999 -u <username> -pw <password> ring
```

The following output files will be created in the specified output directory if Parallel SSH tool is operating successfully:

<table>
<thead>
<tr>
<th>Address</th>
<th>DC</th>
<th>Rack</th>
<th>Status</th>
<th>State</th>
<th>Load</th>
<th>Effective Ownership</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>113427455640312821154458202477256070485</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.57.244.224</td>
<td>datacenter1</td>
<td>rack1</td>
<td>Up</td>
<td>Normal</td>
<td>498.19MB</td>
<td>100.00%</td>
<td>0</td>
</tr>
<tr>
<td>10.57.244.226</td>
<td>datacenter1</td>
<td>rack1</td>
<td>Up</td>
<td>Normal</td>
<td>502.76MB</td>
<td>100.00%</td>
<td>0</td>
</tr>
<tr>
<td>56713727820156410577229101238628035242</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.57.244.229</td>
<td>datacenter1</td>
<td>rack1</td>
<td>Up</td>
<td>Normal</td>
<td>500.91MB</td>
<td>100.00%</td>
<td>0</td>
</tr>
<tr>
<td>113427455640312821154458202477256070485</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Run the following commands in the given order on each and every product node performing a rolling restart:

```bash
./nodetool -h localhost -p 9999 -u user_name -pw password disablethrift
./nodetool -h localhost -p 9999 -u user_name -pw password disablegossip
./nodetool -h localhost -p 9999 -u user_name -pw password drain
./nodetool -h localhost -p 9999 -u user_name -pw password flush
```

5. Restart the product node and run the following commands:

```bash
./nodetool -h localhost -p 9999 -u user_name -pw password enablethrift
./nodetool -h localhost -p 9999 -u user_name -pw password enablegossip
```

6. Clean up any existing snapshots in the source Cassandra cluster, by running NodeTool `cleansnapshot` command via Parent SSH. This is done to avoid any inconsistencies that could potentially occur, if we mix up already existing snapshots with the new snapshots.
7. Run the below command from the `<PSSH_HOME>` folder to create new snapshots:

```bash
<PSSH_HOME>/bin/pssh -h <PSSH_HOME>/bin/host.txt -A -v -l <username> -o /tmp/foo "sh <CASSANDRA_HOME>/bin/nodetool -h localhost -p 9999 -u <username> -pw <password> snapshot"
```

8. Navigate to a cassandra.yaml file of one node and set `incremental_backups` to true and then restart the node. Perform this to all nodes in rolling manner.

   This is done to backup data that has been changed since the last snapshot. Each time a SSTable is flushed, a hard link is copied into a `/backups` sub-directory of the data directory (provided JNA is enabled).

**Migrating schema**

Before migrating Cassandra data to a new cluster, you need to migrate the schema, using the Cassandra CLI tool.

1. Navigate to `<CASSANDRA_HOME>/bin/` directory of one product node and run the following command:
   ```bash
   /cassandra-cli -u <username> -pw <password> -h <host>
   ```
2. Run the following command: `show schema;`
3. Copy all the schema creation commands of all non-system key spaces and save them for later use.
4. Navigate to each new product cluster node, and change `partitioner` property in the cassandra.yaml file as follows:
   ```yaml
   partitioner : org.apache.cassandra.dht.RandomPartitioner
   ```
   In older Cassandra versions, default partitioner is `RandomPartitioner`. Therefore, we change this property, because when migrating data created with that partitioner, you need to use the same partitioner in the new cluster as well.
5. Start the product nodes.
6. Download Apache Cassandra 1.2.13 and extract the installation directory to one new cluster node.
7. Navigate to the `<CASSANDRA_HOME>/bin/` directory and run the following command to log in:
   ```bash
   ./cassandra-cli -u <username> -pw <password> -h <host>
   ```
8. Run the commands you saved in step 3.
9. Repeat the above two steps for all new Cassandra nodes by appropriately changing the `<host>` element of the command in step 7.

**Migrating backed-up data to new product nodes**

1. Move all snapshot data and incremental backup data into the new product nodes.
2. Upgrade the existing Cassandra SSTables. Run the following command to rebuild SSTables that are not on the current version:

```
./pssh-2.3.1/bin/pssh -t -1 -h ./pssh-2.3.1/host.txt -v -l <username> -o <output_directory> bash
apache-cassandra-1.2.13/bin/nodetool -h localhost -p 9999 -u admin -pw admin
upgradesstables -a
```

3. Run the following command to reload the SSTables for all column families.

```
This loads newly placed SSTables on to the system without restarting the nodes. For example:

./pssh-2.3.1/bin/pssh -t -1 -h ./pssh-2.3.1/host.txt -v -l <username> -o <output_directory> bash
cha/apache-cassandra-1.2.13/bin/nodetool -h localhost -p 9999 -u admin -pw admin refresh EVENT_KS
org_wso2_bam_activity_monitoring
```

**WSO2 Patch Application Process**

You apply patches to WSO2 products either as individual patches or through a service pack. A service pack is recommended when the number of patches increase. The following sections explain the WSO2 patch application process:

- Applying service packs to the Kernel
- Applying individual patches to the Kernel
- Verifying the patch application
- Overview of the patch application process
Applying service packs to the product

Carbon 4.2.0 Kernel supports service packs. A service pack is a collection of patches in a single pack. It contains two elements:

- The `lib` directory: contains all the JARs relevant to the service pack.
- The `servicepack_patches.txt` text file: contains the list of JARs in the service pack.

Follow the steps below to apply service packs to your product.

1. Copy the service pack file to the `<PRODUCT_HOME>/repository/components/servicepacks/` directory. For example, the image below shows how a new service pack named `servicepack001` is added to this directory.

   ```
   servicepacks
   ├── servicepack001
   │   └── lib
   │       ├── org.wso2.carbon.base_4.2.0.SNAPSHOT.jar
   │       ├── org.wso2.carbon.bridge_4.2.0.SNAPSHOT.jar
   │       ├── org.wso2.carbon.core.bootup.validator_4.2.0.SNAPSHOT.jar
   │       ├── org.wso2.carbon.core.common_4.2.0.SNAPSHOT.jar
   │       └── org.wso2.carbon.registry.api_4.2.0.SNAPSHOT.jar
   └── servicepack_patches.txt
   ````

2. Start your product. The following steps will be executed:
   1. 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.
   2. The latest service pack in the `<PRODUCT_HOME>/repository/components/servicepacks/` directory will be applied. That is, the patches in the service pack will be applied to the `<PRODUCT_HOME>/repository/components/plugins/` directory.
   3. In addition to the service pack, if there are individual patches added to the `<PRODUCT_HOME>/repository/components/patches/` directory, those will also be incrementally applied to the plugins directory.

   The metadata file available in the service pack will maintain a list of the applied patches by service pack. Therefore, the metadata file information will be compared against the `<PRODUCT_HOME>/repository/components/patches/` directory, and only the patches that were not applied by the service pack will be incrementally applied to the plugins directory.

Applying individual patches to the product

You can apply each patch individually to your system as explained below. Alternatively, you can apply patches through service packs as explained above.
1. Copy the patches to the `<PRODUCT_HOME>/repository/components/patches/` directory.
2. 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, 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 either the `<PRODUCT_HOME>/repository/components/servicepacks/` directory or the `<PRODUCT_HOME>/repository/components/patches/` directory. It verifies all the latest JARs in the servicepacks and patches directories against the JARs in the plugins directory by comparing MD5s of JARs.

Verifying the patch application

After the patch application process is completed, the patch verification process ensures that the latest service pack and other existing 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.log` file.
- The `<PRODUCT_HOME>/repository/components/patches/` directory contains the list of patched JARs and the md5 values.
- A list of all the applied service packs and patches are in the `<PRODUCT_HOME>/repository/components/default/configuration/prePatched.txt` file.

Do not change the data in the `<PRODUCT_HOME>/repository/components/default/configuration/prePatched.txt` file. The patch application process gets the pre-patched list from this file and compares the list with the patches available in the servicepack and patches directories. If you change the data in this file, you will get a startup error when applying patches.

Overview of the patch application process

The diagram below shows how the patch application process is implemented when you start the server.
Reference Guide

The following topics provide reference information for working with WSO2 Storage Server:

- Configuration Files
- Default Ports of WSO2 Products

Configuration Files

The following sections provide detailed definitions for the main configuration files used in Carbon-based products.

- Configuring axis2.xml
- Configuring carbon.xml
- Configuring catalina-server.xml
- Configuring config-validation.xml
- Configuring identity.xml
- Configuring master-datasources.xml
- Configuring registry.xml
- Configuring user-mgt.xml

Configuring axis2.xml

Users can change the default functionality-related configurations by editing the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file using the information given below. This information is provided as reference for users who are already familiar with the product features and want to know how to configure them. If you need introductory information on a specific concept, such as message receivers and formatters, see the relevant topics in the User Guide.

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 Elements

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attributes</th>
<th>Description</th>
<th>Data type</th>
<th>Default value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;axisconfig&gt;</code></td>
<td>name</td>
<td>The root element. The name is defined as: name=&quot;AxisJava2.0&quot;</td>
<td></td>
<td></td>
<td>Mandatory</td>
</tr>
<tr>
<td>Tag</td>
<td>Attribute</td>
<td>Description</td>
<td>Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;module&gt;</td>
<td>ref</td>
<td>A globally engaged module. The ref attribute specifies the module name.</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;parameter&gt;</td>
<td>name</td>
<td>A parameter is a name-value pair. All top-level parameters (those that are direct sub-elements of the root element) will be transformed into properties in AxisConfiguration and can be accessed in the running system. The name attribute (required) specifies the parameter name. If you set the locked attribute to true (default is false), this parameter's value cannot be overridden by services and other configurations.</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>locked</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;listener&gt;</td>
<td>class</td>
<td>A registered listener that will be automatically informed whenever a change occurs in AxisConfiguration, such as when a service or module is deployed or removed. The class attribute specifies this listener's implementation class, which must implement the AxisObserver interface. Registering listeners is useful for additional features such as RSS feed generation, which will provide service information to subscribers.</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;messageReceivers&gt;</td>
<td></td>
<td>The container element for messages receiver definitions.</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Class</td>
<td>Description</td>
<td>Requirement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;messageReceiver&gt;</code></td>
<td>class</td>
<td>A message receiver definition. The <code>class</code> attribute (required) specifies the message receiver implementation class. The <code>mep</code> attribute (required) specifies the message exchange pattern supported by this message receiver. Each message receiver definition supports only one MEP.</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;messageFormatters&gt;</code></td>
<td></td>
<td>The container element for message formatter definitions, which are used to serialize outgoing messages to different formats (such as JSON). The format for a message can be specified by setting the &quot;messageType&quot; property in the MessageContext. It can also be specified as a parameter in service.xml (for service-based configuration) in addition to axis2.xml (for global configuration).</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;messageFormatter&gt;</code></td>
<td>contentType class</td>
<td>A message formatter definition. The <code>contentType</code> attribute specifies which message types are handled by this formatter, and the <code>class</code> attribute specifies the formatter implementation class.</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;messageBuilders&gt;</code></td>
<td></td>
<td>The container element for the message builder definitions, which are used to process the raw payload of incoming messages and convert them to SOAP.</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;messageBuilder&gt;</code></td>
<td>contentType class</td>
<td>A message builder definition. The <code>contentType</code> attribute specifies which message types are handled by this builder, and the <code>class</code> attribute specifies the builder implementation class.</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Attribute</td>
<td>Description</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;transportReceiver&gt;</td>
<td>name</td>
<td>A transport receiver definition, one for each transport type. The name attribute specifies the short name to use when referring to this transport in your configurations (http, tcp, etc.), and the class attribute specifies the receiver implementation class that provides the logic for receiving messages via this transport. You can specify &lt;parameter&gt; elements to pass any necessary information to the transport.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;transportSender&gt;</td>
<td></td>
<td>Just like &lt;transportReceiver&gt;, except &lt;transportSender&gt; allows you to define transport senders, which are used to send messages via the transport.</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;phaseOrder&gt;</td>
<td>type</td>
<td>Specifies the order of phases in the execution chain of a specific type of flow (specified by the type attribute), which can be one of the following:</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InFlow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OutFlow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InFaultFlow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OutFaultFlow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>You add phases using the &lt;phase&gt; sub-element. In the In phase orders, all phases before the Dispatch phase are global phases and after Dispatch are operation phases. In the Out phase orders, phases before the MessageOut phase are global phases and after MessageOut are operation phases.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;phase&gt;</td>
<td>name</td>
<td>The phase definition. The name attribute specifies the phase name. You can add the &lt;handler&gt; sub-element to execute a specific handler during this phase.</td>
<td>Mandatory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Optional Attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;handler&gt;</strong></td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>Optional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The handler (message processing functionality) to execute during this phase. Handlers are combined into chains and phases to provide customizable functionality such as security, reliability, etc. Handlers must be multi-thread safe and should keep all their state in Context objects (see the org.apache.axis2.context package).

| **<order>**   | Optional |
| phase        | Optional |

| **<clustering>** | Optional |
| class          | Optional |
| enable         | Optional |

Used to enable clustering. The `class` attribute specifies the clustering agent class. The `enable` attribute is false by default; set it to true to enable clustering.

| **<property>**  | Optional |
| name           | Optional |
| value          | Optional |

| **<members>**  | Optional |
| class          | Optional |

The list of static or well-known members. These entries will only be valid if the "membershipScheme" above is set to "wka"

| **<member>**    | Optional |
|                | N/A      |
| **<hostName>**  | Optional |
|                | N/A      |
| **<port>**      | Optional |
|                | N/A      |

| **<groupManagement>** | Optional |
| enable               | FALSE    |

Enable the groupManagement entry if you need to run this node as a cluster manager. Multiple application domains with different GroupManagementAgent implementations can be defined in this section.

| **<applicationDomain>** | Optional |
| name                   | N/A      |
Example

The following example shows excerpts from an `axis2.xml` file.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<axisconfig name="AxisJava2.0">
    <!-- ------------------------------- -->
    <!-- Parameters -->
    <!-- ------------------------------- -->
    ...

    <!-- If you want to enable file caching for attachments change this to true -->
    <parameter name="cacheAttachments" locked="false">false</parameter>
    <!-- Attachment file caching location relative to CARBON_HOME -->
    <parameter name="attachmentDIR" locked="false">work/mtom</parameter>
    <!-- Attachment file cache threshold size -->
    <parameter name="sizeThreshold" locked="false">4000</parameter>
    ...

    <!-- ------------------------------- -->
    <!-- Listeners -->
    <!-- ------------------------------- -->

    <!-- This deployment interceptor will be called whenever before a module is initialized or -->
    <!-- service is deployed -->
    <listener
        class="org.wso2.carbon.core.deployment.DeploymentInterceptor"/>

    <!-- ------------------------------- -->
    <!-- Deployers -->
    <!-- ------------------------------- -->

    <!-- Deployer for the dataservice. -->
    <!--<deployer extension="dbs" directory="dataservices"
    class="org.wso2.dataservices.DBDeployer"/>-->

    <!-- Axis1 deployer for Axis2 -->
    <!--<deployer extension="wsdd"
    class="org.wso2.carbon.axis1services.Axis1Deployer"-->
```
directory="axis1services"/-->  

...  

<!-- ================================================= -->  
<!--                Message Receivers                  -->  
<!-- ================================================= -->  

<!-- This is the set of default Message Receivers for the system, if you want to have -->  
<!-- message receivers for any of the other Message exchange Patterns (MEP) implement it -->  
<!-- and add the implementation class to here, so that you can refer from any operation -->  
<!-- Note : You can override this for particular service by adding this same element to the -->  
<!-- services.xml with your preferences -->  
<!-- messageReceivers  
<messageReceivers>  
  <messageReceiver mep="http://www.w3.org/ns/wsdl/in-only"  
  class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>  
  <messageReceiver mep="http://www.w3.org/ns/wsdl/robust-in-only"  
  class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>  
  <messageReceiver mep="http://www.w3.org/ns/wsdl/in-out"  
  class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>  
</messageReceivers>  

<!-- ================================================= -->  
<!--                Message Formatters                 -->  
<!-- ================================================= -->  

<!-- Following content type to message formatter mapping can be used to implement support -->  
<!-- for different message format serializations in Axis2. These message formats are -->  
<!-- expected to be resolved based on the content type. -->  
<!-- messageFormatters  
<messageFormatters>  
  <messageFormatter contentType="application/x-www-form-urlencoded"  
  class="org.apache.axis2.transport.http.XFormURLEncodedFormatter"/>  
  <messageFormatter contentType="multipart/form-data"  
  class="org.apache.axis2.transport.http.MultipartFormDataFormatter"/>  
  <messageFormatter contentType="application/xml"  
  class="org.apache.axis2.transport.http.ApplicationXMLFormatter"/>  
  <messageFormatter contentType="text/xml"  
  class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>  
  <messageFormatter contentType="application/soap+xml"  
  class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>  
</messageFormatters>
class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>
  <messageFormatter contentType="text/plain"

class="org.apache.axis2.format.PlainTextFormatter"/>

  ...

</messageFormatters>

<!-- ==============================================================
  <!--                Message Builders                           
  <!-- ============================================================== -->

<!-- Following content type to builder mapping can be used to implement
  support for -->
<!-- different message formats in Axis2. These message formats are
  expected to be -->
<!-- resolved based on the content type. -->

<messageBuilders>
  <messageBuilder contentType="application/xml"
    class="org.apache.axis2.builder.ApplicationXMLBuilder"/>
  <messageBuilder contentType="application/x-www-form-urlencoded"
    class="org.apache.synapse.commons.builders.XFormURLEncodedBuilder"/>
  <messageBuilder contentType="multipart/form-data"
    class="org.apache.axis2.builder.MultipartFormDataBuilder"/>
  <messageBuilder contentType="text/plain"
    class="org.apache.axis2.format.PlainTextBuilder"/>

  ...

</messageBuilders>

<!-- ==============================================================
  <!--             Transport Ins (Listeners)                     
  <!-- ============================================================== -->

<transportReceiver name="http" class="org.apache.synapse.transport.passthru.PassThroughHttpListener">
  <parameter name="port" locked="false">8280</parameter>
  <parameter name="non-blocking" locked="false">true</parameter>
  <!--parameter name="bind-address" locked="false">hostname or IP
  address</parameter-->
  <!--parameter name="WSDLEPRPrefix" locked="false">https://apachehost:port/somepath</parameter-->
  <parameter name="httpGetProcessor" locked="false">org.wso2.carbon.transport.nhttp.api.PassThroughNHttpGetProc
  essor</parameter>
  <!--<parameter name="priorityConfigFile" locked="false">location of
  priority configuration file</parameter-->-->
  </transportReceiver>
<!-- ================================================= -->
<!--             Transport Outs (Senders)               -->
<!-- ================================================= -->

<transportSender name="http"
class="org.apache.synapse.transport.passthru.PassThroughHttpSender">
  <parameter name="non-blocking" locked="false">true</parameter>
  <parameter name="warnOnHTTP500" locked="false"></parameter>
  <parameter name="http.proxyHost" locked="false">localhost</parameter>
  <parameter name="http.proxyPort" locked="false">3128</parameter>
  <parameter name="http.nonProxyHosts" locked="false">localhost|moon|sun</parameter>
</transportSender>

<!-- ================================================= -->
<!--             Global Engaged Modules                -->
<!-- ============================================================== -->

<!-- Comment this out to disable Addressing -->
<module ref="addressing"/>

<!-- ============================================================== -->
<!--                Clustering                           -->
<!-- ============================================================== -->

<!-- To enable clustering for this node, set the value of "enable" attribute of the "clustering" element to "true". The initialization of a node in the cluster is handled by the class corresponding to the "class" attribute of the "clustering" element. It is also responsible for getting this node to join the cluster. -->
<clustering
class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
enable="false">
  <parameter name="AvoidInitiation">true</parameter>
</clustering>

<!-- This parameter indicates whether the cluster has to be automatically initialized when the AxisConfiguration is built. If set to "true" the initialization will not be done at that stage, and some other party will have to explicitly initialize the cluster. -->
<parameter name="AvoidInitiation">true</parameter>
<members>
  <member>
    <hostName>127.0.0.1</hostName>
    <port>4000</port>
  </member>
</members>

<!-- Enable the groupManagement entry if you need to run this node as a cluster manager. -->
<groupManagement enable="false">
  <applicationDomain name="wso2.esb.domain"
    description="ESB group" 
    agent="org.wso2.carbon.core.clustering.hazelcast.HazelcastGroupManagementAgent"
    subDomain="worker"
    port="2222"/>
</groupManagement>

<!-- Uncomment and configure the following section to enable transactions support -->
<transaction timeout="30000">
  <parameter name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
  <parameter name="java.naming.provider.url">tcp://localhost:61616</parameter>
  <parameter name="UserTransactionJNDIName">UserTransaction</parameter>
  <parameter name="TransactionManagerJNDIName">TransactionManager</parameter>
</transaction>-->
<phaseOrder type="InFlow">
  <!-- System pre defined phases -->
  <phase name="MsgInObservation">
    <handler name="TraceMessageBuilderDispatchHandler" class="org.apache.synapse.transport.passthru.util.TraceMessageBuilderDispatchHandler"/>
  </phase>
  <phase name="Validation"/>
  <phase name="Transport">
    <handler name="RequestURIBasedDispatcher" class="org.apache.axis2.dispatchers.RequestURIBasedDispatcher"><order phase="Transport"/></handler>
    <handler name="CarbonContextConfigurator" class="org.wso2.carbon.mediation.initializer.handler.CarbonContextConfigurator"/>
    <handler name="RelaySecuirtyMessageBuilderDispatchandler" class="org.apache.synapse.transport.passthru.util.RelaySecuirtyMessageBuilderDispatchandler"/>
    <handler name="SOAPActionBasedDispatcher" class="org.apache.axis2.dispatchers.SOAPActionBasedDispatcher"><order phase="Transport"/></handler>
    <!--handler name="SMTPFaultHandler" class="org.wso2.carbon.core.transports.smtp.SMTPFaultHandler"><order phase="Transport"/></handler-->
    <handler name="CacheMessageBuilderDispatchandler" class="org.wso2.carbon.mediation.initializer.handler.CacheMessageBuilderDispatchandler"/>
  </phase>
  ...
</phaseOrder>
<phaseOrder type="OutFlow">
   <!-- Handlers related to unified-endpoint component are added to
   the UEPPhase -->
   <phase name="UEPPhase" />
   <!-- user can add his own phases to this area -->
   <phase name="RMPhase"/>
   ...
</phaseOrder>
Configuring carbon.xml

Users can change the configurations related to the default Carbon functionality by editing the `<PRODUCT_HOME>/repository/conf/carbon.xml` file using the information given below.

Click on the table and use the left and right arrow keys to scroll horizontally.

**XML Elements**

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Server&gt;</td>
<td>xmlns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Name&gt;</td>
<td></td>
<td>Product Name.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;ServerKey&gt;</td>
<td></td>
<td>Machine readable unique key to identify each product.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;Version&gt;</td>
<td></td>
<td>Product Version.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;HostName&gt;</td>
<td></td>
<td>Host name or IP address of the machine hosting this server e.g. <a href="http://www.wso2.org">www.wso2.org</a>, 192.168.1.10 This is will become part of the End Point Reference of the services deployed on this server instance.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;MgtHostName&gt;</td>
<td></td>
<td>Host name to be used for the Carbon management console.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;ServerURL&gt;</td>
<td></td>
<td>The URL of the back end server. This is where the admin services are hosted and will be used by the clients in the front end server. This is required only for the Front-end server. This is used when separating the BE server from the FE server.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;IndexPageURL&gt;</td>
<td></td>
<td>The URL of the index page. This is where the user will be redirected after signing in to the carbon server.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;ServerRoles&gt;</td>
<td></td>
<td>For cApp deployment, we have to identify the roles that can be acted by the current server. The following property is used for that purpose. Any number of roles can be defined here. Regular expressions can be used in the role. Ex : <code>&lt;Role&gt;.*&lt;/Role&gt;</code> means this server can act as any role.</td>
<td>String</td>
</tr>
<tr>
<td>&lt;Role&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;BamServerURL&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Package&gt;</td>
<td></td>
<td>The fully qualified name of the server.</td>
<td>String</td>
</tr>
<tr>
<td><strong>&lt;WebContextRoot&gt;</strong></td>
<td>Webapp context root of WSO2 Carbon.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;RegistryHttpPort&gt;</strong></td>
<td>In-order to get the registry http Port from the back-end when the default http transport is not the same.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;ItemsPerPage&gt;</strong></td>
<td>Number of items to be displayed on a management console page. This is used at the backend server for pagination of various items.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;InstanceMgtWSEndpoint&gt;</strong></td>
<td>The endpoint URL of the cloud instance management Web service.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;Ports&gt;</strong></td>
<td>Ports used by this server</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;Offset&gt;</strong></td>
<td>Ports offset. This entry will set the value of the ports defined below to the define value + Offset. e.g. Offset=2 and HTTPS port=9443 will set the effective HTTPS port to 9445.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;JMX&gt;</strong></td>
<td>The JMX Ports.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;RMIRegistryPort&gt;</strong></td>
<td>The port RMI registry is exposed.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;RMIServerPort&gt;</strong></td>
<td>The port RMI server should be exposed.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;EmbeddedLDAP&gt;</strong></td>
<td>Embedded LDAP server specific ports.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;LDAPServerPort&gt;</strong></td>
<td>Port which embedded LDAP server runs.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;KDCServerPort&gt;</strong></td>
<td>Port which KDC (Kerberos Key Distribution Center) server runs.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;EmbeddedQpid&gt;</strong></td>
<td>Embedded Qpid broker ports.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;BrokerPort&gt;</strong></td>
<td>Broker TCP Port.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;BrokerSSLPor&gt;</strong></td>
<td>SSL Port.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;JNDIProviderPort&gt;</strong></td>
<td>Override datasources JNDIproviderPort defined in bps.xml and datasources.properties files.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;ThriftEntitlementReceivePort&gt;</strong></td>
<td>Override receive port of thrift based entitlement service.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;JNDI&gt;</strong></td>
<td>JNDI Configuration.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;DefaultInitialContextFactory&gt;</strong></td>
<td>The fully qualified name of the default initial context factory.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;Restrictions&gt;</strong></td>
<td>The restrictions that are done to various JNDI Contexts in a Multi-tenant environment.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;AllTenants&gt;</strong></td>
<td>Contexts that are common to all tenants.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;SuperTenantOnly&gt;</strong></td>
<td>Contexts that will be available only to the super-tenant.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;IsCloudDeployment&gt;</strong></td>
<td>Property to determine if the server is running on a cloud deployment environment. This property should only be used to determine deployment specific details that are applicable only in a cloud deployment, i.e when the server is deployed &quot;as-a-service.&quot;</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;EnableMetering&gt;</strong></td>
<td>Property to determine whether usage data should be collected for metering purposes.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;MaxThreadExecutionTime&gt;</strong></td>
<td>The Max time a thread should take for execution in seconds.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;GhostDeployment&gt;</strong></td>
<td>A flag to enable or disable Ghost Deployer. By default this is set to false. That is because the Ghost Deployer works only with the HTTP/S transports. If you are using other transports, don't enable Ghost Deployer.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;Enabled&gt;</strong></td>
<td>When &lt;GhostDeployment&gt; is enabled, the lazy loading feature will apply to artifacts deployed. That is, when a tenant loads, only the specific artifact requested by the service will be loaded.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;PartialUpdate&gt;</strong></td>
<td>&lt;PartialUpdate&gt; is a further enhancement to lazy loading of artifacts, which applies when &lt;DeploymentSynchronizer&gt; is enabled in a clustered environment.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;Axis2Config&gt;</strong></td>
<td>Axis2 related configurations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>&lt;RepositoryLocation&gt;</strong></td>
<td>Location of the Axis2 Services &amp; Modules repository This can be a directory in the local file system, or a URL. e.g. 1. /home/wso2wsas/repository/ - An absolute path 2. repository - In this case, the path is relative to CARBON_HOME 3. file:///home/wso2wsas/repository/ 4. <a href="http://wso2wsas/repository/">http://wso2wsas/repository/</a>.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;DeploymentUpdateInterval&gt;</strong></td>
<td>Deployment update interval in seconds. This is the interval between repository listener executions.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;ConfigurationFile&gt;</strong></td>
<td>Location of the main Axis2 configuration descriptor file, a.k.a. axis2.xml file This can be a file on the local file system, or a URL e.g. 1. /home/repository/axis2.xml - An absolute path 2. conf/axis2.xml - In this case, the path is relative to CARBON_HOME 3. file:///home/carbon/repository/axis2.xml 4. <a href="http://repository/conf/axis2.xml">http://repository/conf/axis2.xml</a></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;ServiceGroupContextIdleTime&gt;</strong></td>
<td>ServiceGroupContextIdleTime, which will be set in ConfigurationContext for multiple clients which are going to access the same ServiceGroupContext Default Value is 30 Sec.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;ClientRepositoryLocation&gt;</strong></td>
<td>This repository location is used to create the client side configuration context used by the server when calling admin services.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;clientAxis2XmlLocation&gt;</strong></td>
<td>This axis2 xml is used in creating the configuration context by the FE server calling to BE server.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>HideAdminServiceWSDLs</td>
<td>If this parameter is set, the WSDL file on an admin service will not give the admin service WSDL. By default, this parameter is set to &quot;true&quot;. Note that setting this parameter to false will expose WSO2 Storage Server operations through a WSDL.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>HttpAdminServices</td>
<td>WARNING-Use With Care! Uncommenting bellow parameter would expose all AdminServices in HTTP transport. With HTTP transport your credentials and data routed in public channels are vulnerable for sniffing attacks. Use this parameter ONLY if your communication channels are confirmed to be secured by other means.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>ServiceUserRoles</td>
<td>The default user roles which will be created when the server is started up for the first time.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>EnableEmailUserName</td>
<td>Enable following config to allow Emails as usernames.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>Security configurations.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>KeyStore</td>
<td>KeyStore which will be used for encrypting/decrypting passwords and other sensitive information.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>Keystore file location.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Keystore type (JKS/PKCS12 etc.)</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Keystore password.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>KeyAlias</td>
<td>Private Key alias.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>KeyPassword</td>
<td>Private Key password.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>TrustStore</td>
<td>System wide trust-store which is used to maintain the certificates of all the trusted parties.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>Trust-store file location.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Trust-store type.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Trust-store password.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>NetworkAuthenticatorConfig</td>
<td>The Authenticator configuration to be used at the JVM level. We extend the java.net.Authenticator to make it possible to authenticate to given servers and proxies.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Credential</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Pattern</td>
<td>The pattern that would match a subset of URLs for which this authenticator would be used.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>The type of this authenticator. Allowed values are: 1. server 2. proxy.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Username</td>
<td>The username used to log in to server/proxy.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>The password used to log in to server/proxy.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td><code>&lt;TomcatRealm&gt;</code></td>
<td>The Tomcat realm to be used for hosted Web applications. Allowed values are: 1. UserManager 2. Memory If this is set to 'UserManager', the realm will pick users &amp; roles from the system's WSO2 User Manager. If it is set to 'memory', the realm will pick users &amp; roles from CARBON_HOME/repository/conf/tomcat/tomcat-users.xml.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><code>&lt;DisableTokenStore&gt;</code></td>
<td>Option to disable storing of tokens issued by STS.</td>
<td>Bool</td>
<td></td>
</tr>
<tr>
<td><code>&lt;TokenStoreClassName&gt;</code></td>
<td>Security token store class name. If this is not set, default class will be org.wso2.carbon.security.util.SecurityTokenStore</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><code>&lt;WorkDirectory&gt;</code></td>
<td>The temporary work directory.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;HouseKeeping&gt;</code></td>
<td>House-keeping configuration.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;AutoStart&gt;</code></td>
<td>True - Start House-keeping thread on server startup false - Do not start House-keeping thread on server startup. The user will run it manually as and when he wishes.</td>
<td>Boo</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Interval&gt;</code></td>
<td>The interval in <em>minutes</em>, between house-keeping runs.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><code>&lt;MaxTempFileLifetime&gt;</code></td>
<td>The maximum time in <em>minutes</em>, temp files are allowed to live in the system. Files/directories which were modified more than &quot;MaxTempFileLifetime&quot; minutes ago will be removed by the house-keeping task.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><code>&lt;FileUploadConfig&gt;</code></td>
<td>Configuration for handling different types of file upload and other file uploading related config parameters. To map all actions to a particular FileUploadExecutor, use <code>&lt;Action&gt;*&lt;/Action&gt;</code>.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;TotalFileSizeLimit&gt;</code></td>
<td>The total file upload size limit in MB.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Mapping&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Actions&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Action&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Class&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;HttpGetRequestProcessors&gt;</code></td>
<td>Processors which process special HTTP GET requests such as ?wsdl, ?policy etc. In order to plug in a processor to handle a special request, simply add an entry to this section. The value of the Item element is the first parameter in the query string(e.g. ?wsdl) which needs special processing The value of the Class element is a class which implements org.wso2.carbon.transport.HttpGetRequestProcessor</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Processor&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Item&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Class&gt;</code></td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>&lt;DeploymentSynchronizer&gt;</td>
<td>Deployment Synchronizer Configuration. Enabled when running with &quot;svn based&quot; dep sync. In master nodes you need to set both AutoCommit and AutoCheckout to true and in worker nodes set only AutoCheckout to true.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;Enabled&gt;</td>
<td></td>
<td>Boo</td>
<td></td>
</tr>
<tr>
<td>&lt;AutoCommit&gt;</td>
<td></td>
<td>Boo</td>
<td></td>
</tr>
<tr>
<td>&lt;AutoCheckout&gt;</td>
<td></td>
<td>Boo</td>
<td></td>
</tr>
<tr>
<td>&lt;RepositoryType&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;SvnUrl&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;SvnUser&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;SvnPassword&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;SvnUrlAppendTenantId&gt;</td>
<td></td>
<td>Boo</td>
<td></td>
</tr>
<tr>
<td>&lt;MediationConfig&gt;</td>
<td>Mediation persistence configurations. Only valid if mediation features are available i.e. ESB.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;LoadFromRegistry&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;SaveToFile&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;Persistence&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;RegistryPersistence&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;ServerInitializers&gt;</td>
<td>Server initializing code, specified as implementation classes of org.wso2.carbon.core.ServerInitializer. This code will be run when the Carbon server is initialized.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;Initializers&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;RequireCarbonServlet&gt;</td>
<td>Indicates whether the Carbon Servlet is required by the system, and whether it should be registered.</td>
<td>Boo</td>
<td></td>
</tr>
<tr>
<td>&lt;H2DatabaseConfiguration&gt;</td>
<td>Carbon H2 OSGI Configuration By default non of the servers start. name=&quot;web&quot; - Start the web server with the H2 Console name=&quot;webPort&quot; - The port (default: 8082) name=&quot;webAllowOthers&quot; - Allow other computers to connect name=&quot;webSSL&quot; - Use encrypted (HTTPS) connections name=&quot;tcp&quot; - Start the TCP server name=&quot;tcpPort&quot; - The port (default: 9092) name=&quot;tcpAllowOthers&quot; - Allow other computers to connect name=&quot;tcpSSL&quot; - Use encrypted (SSL) connections name=&quot;pg&quot; - Start the PG server name=&quot;pgPort&quot; - The port (default: 5435) name=&quot;pgAllowOthers&quot; - Allow other computers to connect name=&quot;trace&quot; - Print additional trace information; for all servers name=&quot;baseDir&quot; - The base directory for H2 databases; for all servers.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;StatisticsReporterDisabled&gt;</td>
<td>Disables the statistics reporter by default.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;EnableHTTPAdminConsole&gt;</td>
<td>Enables HTTP for WSO2 servers so that you can access the Admin Console via HTTP.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;FeatureRepository&gt;</td>
<td>Default Feature Repository of WSO2 Carbon.</td>
<td>Strir</td>
<td></td>
</tr>
<tr>
<td>&lt;RepositoryName&gt;</td>
<td></td>
<td>Strir</td>
<td></td>
</tr>
</tbody>
</table>
### 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**

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Server&gt;</td>
<td></td>
<td>A Server element represents the entire Catalina servlet container. Therefore, it must be the single outermost element in the conf/server.xml configuration file. Its attributes represent the characteristics of the servlet container as a whole.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shutdown</td>
<td></td>
<td>The command string that must be received via a TCP/IP connection to the specified port number, in order to shut down Tomcat.</td>
<td>String</td>
<td>SHUTDOWN</td>
</tr>
<tr>
<td>port</td>
<td>The TCP/IP port number on which this server waits for a shutdown command. Set to -1 to disable the shutdown port.</td>
<td>Int</td>
<td>8005</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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 *nixes). It cannot be used when running Tomcat with the standard shell scripts though, as it will prevent shutdown.bat</td>
<td>sh and catalina.bat</td>
<td>sh from stopping it gracefully.</td>
<td></td>
</tr>
</tbody>
</table>

| <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.tomcat.ext.service.ExtendedStandardService |

<p>| &lt;Connector&gt; | | | |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>The TCP port number on which this Connector will create a server socket and await incoming connections. Your operating system will allow only one server application to listen to a particular port number on a particular IP address. If the special value of 0 (zero) is used, then Tomcat will select a free port at random to use for this connector. This is typically only useful in embedded and testing applications.</td>
<td>Int</td>
<td>9763</td>
</tr>
<tr>
<td>URIEncoding</td>
<td>This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL.</td>
<td>Int</td>
<td>UTF-8</td>
</tr>
<tr>
<td>compressableMimeType</td>
<td>The value is a comma separated list of MIME types for which HTTP compression may be used.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>noCompressionUserAgents</td>
<td>The value is a regular expression (using java.util.regex) matching the user-agent header of HTTP clients for which compression should not be used, because these clients, although they do advertise support for the feature, have a broken implementation.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>compressionMinSize</td>
<td>If compression is set to &quot;on&quot; then this attribute may be used to specify the minimum amount of data before the output is compressed.</td>
<td>Int</td>
<td>2048</td>
</tr>
</tbody>
</table>
| **compression** | The Connector may use HTTP/1.1 GZIP compression in an attempt to save server bandwidth. The acceptable values for the parameter is "off" (disable compression), "on" (allow compression, which causes text data to be compressed), "force" (forces compression in all cases), or a numerical integer value (which is equivalent to "on", but specifies the minimum amount of data before the output is compressed). If the content-length is not known and compression is set to "on" or more aggressive, the output will also be compressed. If not specified, this attribute is set to "off".

Note: There is a tradeoff between using compression (saving your bandwidth) and using the sendfile feature (saving your CPU cycles). If the connector supports the sendfile feature, e.g. the NIO connector, using sendfile will take precedence over compression. The symptoms will be that static files greater that 48 Kb will be sent uncompressed. You can turn off sendfile by setting useSendfile attribute of the connector, as documented below, or change the sendfile usage threshold in the configuration of the DefaultServlet in the default conf/web.xml or in the web.xml of your web application. |
<p>| String | on |
| <strong>server</strong> | 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 |
| <strong>acceptCount</strong> | 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 |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxKeepAliveRequests</td>
<td>The maximum number of HTTP requests which can be pipelined until the connection is closed by the server. Setting this attribute to 1 will disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. Setting this to -1 will allow an unlimited amount of pipelined or keep-alive HTTP requests.</td>
<td>Int</td>
<td>200</td>
</tr>
<tr>
<td>connectionUploadTimeout</td>
<td>Specifies the timeout, in milliseconds, to use while a data upload is in progress. This only takes effect if disableUploadTimeout is set to false.</td>
<td>Int</td>
<td>120000</td>
</tr>
<tr>
<td>disableUploadTimeout</td>
<td>This flag allows the servlet container to use a different, usually longer connection timeout during data upload.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>minSpareThreads</td>
<td>The minimum number of threads always kept running.</td>
<td>Int</td>
<td>50</td>
</tr>
<tr>
<td>maxThreads</td>
<td>The maximum number of request processing threads to be created by this Connector, which therefore determines the maximum number of simultaneous requests that can be handled. If an executor is associated with this connector, this attribute is ignored as the connector will execute tasks using the executor rather than an internal thread pool.</td>
<td>Int</td>
<td>250</td>
</tr>
<tr>
<td>acceptorThreadCount</td>
<td>The number of threads to be used to accept connections. Increase this value on a multi CPU machine, although you would never really need more than 2. Also, with a lot of non keep alive connections, you might want to increase this value as well.</td>
<td>Int</td>
<td>2</td>
</tr>
<tr>
<td>maxHttpHeaderSize</td>
<td>The maximum size of the request and response HTTP header, specified in bytes.</td>
<td>Int</td>
<td>8192</td>
</tr>
<tr>
<td>bindOnInit</td>
<td>Controls when the socket used by the connector is bound. By default it is bound when the connector is initiated and unbound when the connector is destroyed. If set to false, the socket will be bound when the connector is started and unbound when it is stopped.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>redirectPort</td>
<td>If this Connector is supporting non-SSL requests, and a request is received for which a matching <code>&lt;security-constraint&gt;</code> requires SSL transport, Catalina will automatically redirect the request to the port number specified here.</td>
<td>Int</td>
<td>9443</td>
</tr>
<tr>
<td>SSLEnabled</td>
<td>Use this attribute to enable SSL traffic on a connector. To turn on SSL handshake/encryption/decryption on a connector set this value to true. The default value is false. When turning this value to true you will want to set the scheme and the secure attributes as well to pass the correct request.getScheme() and request.isSecure() values to the servlets. See SSL Support for more information.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>secure</td>
<td>Set this attribute to true if you wish to have calls to request.isSecure() to return true for requests received by this Connector. You would want this on an SSL Connector or a non-SSL connector that is receiving data from a SSL accelerator, like a crypto card, a SSL appliance or even a webserver.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>scheme</td>
<td>Set this attribute to the name of the protocol you wish to have returned by calls to request.getScheme(). For example, you would set this attribute to &quot;https&quot; for an SSL Connector.</td>
<td>String</td>
<td>https</td>
</tr>
<tr>
<td>clientAuth</td>
<td>Set to true if you want the SSL stack to require a valid certificate chain from the client before accepting a connection. Set to false if you want the SSL stack to request a client Certificate, but not fail if one isn't presented. A false value will not require a certificate chain unless the client requests a resource protected by a security constraint that uses CLIENT-CERT authentication.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>enableLookups</td>
<td>Set to true if you want calls to request.getRemoteHost() to perform DNS lookups in order to return the actual host name of the remote client. Set to false to skip the DNS lookup and return the IP address in String form instead (thereby improving performance). By default, DNS lookups are disabled.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>sslProtocol</td>
<td>The SSL protocol(s) to use (a single value may enable multiple protocols - see the JVM documentation for details). The permitted values may be obtained from the JVM documentation for the allowed values for algorithm when creating an SSLContext instance e.g. Oracle Java 6 and Oracle Java 7. Note: There is overlap between this attribute and sslEnabledProtocols.</td>
<td>String</td>
<td>TLS</td>
</tr>
<tr>
<td>keystoreFile</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keystorePass</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Engine&gt;</td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Logical name of this Engine, used in log and error messages. When using multiple Service elements in the same Server, each Engine MUST be assigned a unique name.</td>
<td>String</td>
<td>Catalina</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>-----------------</td>
</tr>
<tr>
<td>defaultHost</td>
<td>The default host name, which identifies the Host that will process requests directed to host names on this server, but which are not configured in this configuration file. This name MUST match the name attributes of one of the Host elements nested immediately inside.</td>
<td>String</td>
<td>localhost</td>
</tr>
<tr>
<td>&lt;Realm&gt;</td>
<td>A Realm element represents a &quot;database&quot; of usernames, passwords, and roles (similar to Unix groups) assigned to those users. Different implementations of Realm allow Catalina to be integrated into environments where such authentication information is already being created and maintained, and then utilize that information to implement Container Managed Security as described in the Servlet Specification. You may nest a Realm inside any Catalina container Engine, Host, or Context). In addition, Realms associated with an Engine or a Host are automatically inherited by lower-level containers, unless explicitly overridden.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>className</td>
<td>Java class name of the implementation to use. This class must implement the org.apache.catalina.Realminterface.</td>
<td>String</td>
<td>org.wso2.carbon.tomcat.ext.realms.CarbonTomcatRealm</td>
</tr>
</tbody>
</table>
| <Host> | The Host element represents a virtual host, which is an association of a network name for a server (such as "www.mycompany.com" with the particular server on which Tomcat is running. For clients to be able to connect to a Tomcat server using its network name, this name must be registered in the Domain Name Service (DNS) server that manages the Internet domain you belong to - contact your Network Administrator for more information.

In many cases, System Administrators wish to associate more than one network name (such as www.mycompany.com and company.com) with the same virtual host and applications. This can be accomplished using the Host Name Aliases feature discussed below.

One or more Host elements are nested inside an Engine element. Inside the Host element, you can nest Context elements for the web applications associated with this virtual host. Exactly one of the Hosts associated with each Engine MUST have a name matching the defaultHost attribute of that Engine.

Clients normally use host names to identify the server they wish to connect to. This host name is also included in the HTTP request headers. Tomcat extracts the host name from the HTTP headers and looks for a Host with a matching name. If no match is found, the request is routed to the default host. The name of the default host does not have to match a DNS name (although it can) since any request where the DNS name does not match the name of a Host element will be routed to the default host. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>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.</td>
<td>String</td>
<td>localhost</td>
</tr>
<tr>
<td>appBase</td>
<td>The Application Base directory for this virtual host. This is the pathname of a directory that may contain web applications to be deployed on this virtual host. You may specify an absolute pathname, or a pathname that is relative to the $CATALINA_BASE directory. See Automatic Application Deployment for more information on automatic recognition and deployment of web applications. If not specified, the default of webapps will be used.</td>
<td>String</td>
<td>${carbon.home}/repository/deployment/server/webapps/</td>
</tr>
<tr>
<td>autoDeploy</td>
<td>This flag value indicates if Tomcat should check periodically for new or updated web applications while Tomcat is running. If true, Tomcat periodically checks the appBase and xmlBase directories and deploys any new web applications or context XML descriptors found. Updated web applications or context XML descriptors will trigger a reload of the web application. See Automatic Application Deployment for more information.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>deployOnStartup</td>
<td>This flag value indicates if web applications from this host should be automatically deployed when Tomcat starts. See Automatic Application Deployment for more information.</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>unpackWARs</td>
<td>Set to true if you want web applications that are placed in the appBase directory as web application archive (WAR) files to be unpacked into a corresponding disk directory structure, false to run such web applications directly from a WAR file. WAR files located outside of the Host's appBase will not be expanded.</td>
<td>Boolean</td>
<td>true</td>
</tr>
<tr>
<td>&lt;&lt;Valve</td>
<td>The Access Log Valve creates log</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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, or Engine), and will record ALL requests processed by that container.

Some requests may be handled by Tomcat before they are passed to a container. These include redirects from /foo to /foo/ and the rejection of invalid requests. Where Tomcat can identify the Context that would have handled the request, the request/response will be logged in the AccessLog(s) associated Context, Host and Engine. Where Tomcat cannot identify the Context that 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,
case, the number of bytes that was passed to the sendfile thread for writing will be recorded in the access log valve.

<table>
<thead>
<tr>
<th>className</th>
<th>Java class name of the implementation to use.</th>
<th>String</th>
<th>org.wso2.carbon.tomcat.ext.valves.CarbonContextCreatorValve, org.apache.catalina.valves.AccessLogValve</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>A formatting layout identifying the various information fields from the request and response to be logged, or the word common or combined to select a standard format.</td>
<td>String</td>
<td>combined</td>
</tr>
<tr>
<td>suffix</td>
<td>The suffix added to the end of each log file name.</td>
<td>String</td>
<td>.log</td>
</tr>
<tr>
<td>prefix</td>
<td>The prefix added to the start of each log file name.</td>
<td>String</td>
<td>http_access_</td>
</tr>
<tr>
<td>directory</td>
<td>Absolute or relative path name of a directory in which log files created by this valve will be placed. If a relative path is specified, it is interpreted as relative to $CATALINA_BASE. If no directory attribute is specified, the default value is &quot;logs&quot; (relative to $CATALINA_BASE).</td>
<td>String</td>
<td>${carbon.home}/repository/logs</td>
</tr>
<tr>
<td>threshold</td>
<td>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.</td>
<td>Int</td>
<td>600</td>
</tr>
</tbody>
</table>

**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 Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Required processor speed.</td>
<td>800</td>
</tr>
<tr>
<td>RAM</td>
<td>Required RAM in your environment.</td>
<td>2048</td>
</tr>
<tr>
<td>swap</td>
<td>Required space in hard disk to use for virtual memory.</td>
<td>2048</td>
</tr>
<tr>
<td>freeDisk</td>
<td>Free disk space required in your environment.</td>
<td>1024</td>
</tr>
<tr>
<td>ulimit</td>
<td>The limit of resources per user. This value indicates the limit on the number of file descriptors a process may have.</td>
<td>4096</td>
</tr>
</tbody>
</table>

If the values set for these parameters in your environment are less than the recommendations, the following warnings will be published when you start your server.

- **WARN - ValidationResultPrinter CPU speed (MHz):** `<systemCPU>` of the system is below the recommended minimum speed : `<recommended value>`
- **WARN - ValidationResultPrinter RAM size (MB):** `<systemRam>` of the system is below the recommended minimum size : `<recommended value>`
- **WARN - ValidationResultPrinter Swap Memory size (MB):** `<systemSwap>` of the system is below the recommended minimum size : `<recommended value>`
- **WARN - ValidationResultPrinter Maximum free Disk Space (MB):** `<systemDisk>` of the system is below the recommended minimum size : `<recommended value>`
- **WARN - ValidationResultPrinter Open files limit:** `<openFileLimit>` of the system is below the recommended minimum count : `<recommended value>`

### JVM Validation

The following JVM heap size values are recommended by default in the config-validation.xml file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initHeapSize</td>
<td>The initial heap size that applies if the JVM requires more memory than is allocated by default.</td>
<td>256</td>
</tr>
<tr>
<td>maxHeapSize</td>
<td>The maximum heap size that applies if the JVM requires more memory than is allocated by default.</td>
<td>512</td>
</tr>
<tr>
<td>maxPermGenSize</td>
<td>The maximum heap size of the permanent generation of heap.</td>
<td>256</td>
</tr>
</tbody>
</table>

These parameters are specified in the product startup script as shown below, where, "-Xms", "-Xmx" and "-XX" correspond to "initHeapSize", "maxHeapSize" and "maxPermGenSize" respectively. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`. 

```bash
#ulimit -n 100000
```
If these heap size values in your product startup script are less than the recommended values, the following warnings will be published when you start your server:

- WARN - ValidationResultPrinter Initial Heap Memory (MB) : <system value> of the running JVM is set below the recommended minimum size :<recommended value>
- WARN - ValidationResultPrinter Maximum Heap Memory (MB) : <system value> of the running JVM is set below the recommended minimum size :<recommended value>
- WARN - ValidationResultPrinter Maximum PermGen space (MB) :<system value> of the running JVM is set below the recommended minimum size :<recommended value>

**System Property Validation**

According to the config-validation.xml file, values are required to be specified for the following properties in your system. Note that it is not recommended to remove this validations as these are mandatory settings.

- carbon.home
- carbon.config.dir.path
- axis2.home

The carbon.home and carbon.config.dir.path properties are given in the product startup script as shown below. For example, see the product startup script for Linux: `<PRODUCT_HOME>/bin/wso2server.sh`

```bash
$JAVA_OPTS
-Dcarbon.home="$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`

```bash
# 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
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."

Configuring identity.xml

Users can change the default configurations by editing the <PRODUCT_HOME>/repository/conf/identity.xml file using the information given below.

Click on the table and use the left and right arrow keys to scroll horizontally.

**XML Elements**

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Server&gt;</td>
<td>xmlns</td>
<td>Identity related data source configuration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;JDBCPersistenceManager&gt;</td>
<td></td>
<td>Include a data source name (jndiConfigName) from the set of data sources defined in master-datasources.xml.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;DataSource&gt;</td>
<td></td>
<td>If the identity database is created from another place and if it is required to skip schema initialization during the server start up, set the property to &quot;true&quot;.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;OpenID&gt;</td>
<td></td>
<td>OpenID related configurations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;OpenIDServerUrl&gt;</td>
<td></td>
<td>This is the URL that the OpenID server (servlet) is running in.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;OpenIDUserPattern&gt;</td>
<td></td>
<td>URL of the pattern that can be configured for the user's OpenID.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;OpenIDSkipUserConsent&gt;</td>
<td></td>
<td>Set to false if the users must be prompted for approval.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;OpenIDRememberMeExpiry&gt;</td>
<td></td>
<td>Expiry time of the OpenID RememberMe token in minutes.</td>
<td>Int</td>
<td>0 Minutes</td>
</tr>
<tr>
<td>&lt;UseMultifactorAuthentication&gt;</td>
<td></td>
<td>Multifactor authentication configuration.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;DisableOpenIDDumbMode&gt;</td>
<td></td>
<td>To enable or disable OpenID dumb mode.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;SessionTimeout&gt;</td>
<td></td>
<td>OpenID session timeout in seconds.</td>
<td>Int</td>
<td>36000 S</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>&lt;AcceptSAMLSSOLogin&gt;</td>
<td>Skips authentication if the valid SAML2 Web SSO browser session is available.</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>&lt;ClaimsRetrieverImplClass&gt;</td>
<td>User claim retrieving module for OpenID.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;OAuth&gt;</td>
<td>OAuth related configurations.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;AccessTokenDefaultValidityPeriod&gt;</td>
<td>Default validity period for Access Token in seconds.</td>
<td>Int</td>
<td>3600 Sec</td>
<td></td>
</tr>
<tr>
<td>&lt;TimestampSkew&gt;</td>
<td>Timestamp skew in seconds.</td>
<td>Int</td>
<td>300 Sec</td>
<td></td>
</tr>
<tr>
<td>&lt;EnableOAuthCache&gt;</td>
<td>Enable OAuth caching. This cache has the replication support.</td>
<td>Boolean</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>&lt;TokenPersistencePreprocessor&gt;</td>
<td>Configure the security measures needed to be done prior to storing the token in the database, such as hashing, encrypting, etc.</td>
<td>String</td>
<td>org.wso2:plainTokenPersistencePreprocessor</td>
<td></td>
</tr>
<tr>
<td>&lt;SupportedResponseTypes&gt;</td>
<td>Supported OAuth2.0 response types.</td>
<td>String</td>
<td>token, ci</td>
<td></td>
</tr>
<tr>
<td>&lt;SupportedGrantTypes&gt;</td>
<td>Supported OAuth2.0 grant types.</td>
<td>String</td>
<td>authoriz:grant-t</td>
<td></td>
</tr>
<tr>
<td>&lt;OAuthCallbackHandlers&gt;</td>
<td>OAuth callback handler module class name.</td>
<td>String</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>&lt;OAuthCallbackHandler&gt;</td>
<td>OAuth callback handler module class name.</td>
<td>String</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>&lt;EnableAssertions&gt;</td>
<td>Assertions can be used to embed parameters into the access token.</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>&lt;UserName&gt;</td>
<td>This enables you to add the user name as an additional parameter if you require it.</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>&lt;EnableAccessTokenPartitioning&gt;</td>
<td>This should be set to true when using multiple user stores and keys should be saved into different tables according to the user store. By default, all the application keys are saved into the same table. UserName Assertion should be ‘true’ to use this.</td>
<td>Boolean</td>
<td>FALSE</td>
<td></td>
</tr>
</tbody>
</table>
<AccessTokenPartitioningDomains>
This includes the user store domain names and mapping to the new table name. E.g., if you provide 'A:foo.com', foo.com should be the user store domain name and 'A' represents the relevant mapping of the token store table, i.e., tokens will be added to a table called IDN_OAUTH2_ACCESS_TOKEN_A.

<String values with Comma separated>N/A

<AuthorizationContextTokenGeneration>

<Enabled>
This mentions whether token generation is enabled or not.

<Boolean>FALSE

<TokenGeneratorImplClass>
Token generation class name.

<String>org.wso2.carbon.identity.oauth2.token.JWTTokenGenerator

<ClaimsRetrieverImplClass>
Claim retrieving class name for generating a token.

<String>org.wso2.carbon.identity.oauth2.token.DefaultClaimsRetriever

<ConsumerDialectURI>
Claim Dialect URI that is used for claim retrieving.

<String>http://ws

<SignatureAlgorithm>
Signature algorithm used for sign the token.

<String>SHA256

<AuthorizationContextTTL>
Token time to live value.

<Long>15 Minutes

<SAML2Grant>
Configuration related to SAML2 Grant type.

<OpenIDConnect>

<IDTokenBuilder>
IDToken generator implementation class name.

<String>org.wso2.carbon.identity.openidconnect.DefaultIDTokenBuilder

<IDTokenIssuerID>
The value of TokenIssuerID of the IDToken. This is a unique value and should be changed according to the deployment values.

<String>OIDCAuthzServer

<IDTokenSubjectClaim>
This is the claim used as the subject of the IDToken. You can use different claims such as http://wso2.org/claims/emailaddress.

<String>http://ws

<IDTokenCustomClaimsCallBackHandler>
Claim callback implementation class name. This is used to return custom claims with the IDToken.

<String>org.wso2.carbon.identity.openidconnect.SAMLAssertionClaimsCallback

<IDTokenExpiration>
The expiration value of the IDToken in seconds.

<Int>300 Seconds

<UserInfoEndpointClaimDialect>
Defines which claim dialect should be returned from the User Endpoint.

<String>http://ws
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;UserInfoEndpointClaimRetriever&gt;</td>
<td>Defines the implementation name of the class which builds the claims for the user info endpoint's response.</td>
<td>String</td>
<td>org.wso2:</td>
</tr>
<tr>
<td>&lt;UserInfoEndpointRequestValidator&gt;</td>
<td>Implementation name of the class that validates the user info request against the specification.</td>
<td>String</td>
<td>org.wso2:</td>
</tr>
<tr>
<td>&lt;UserInfoEndpointAccessTokenValidator&gt;</td>
<td>Implementation name of the class that validates the access token.</td>
<td>String</td>
<td>org.wso2:</td>
</tr>
<tr>
<td>&lt;UserInfoEndpointResponseBuilder&gt;</td>
<td>Implementation name of the class that builds the user info request.</td>
<td>String</td>
<td>org.wso2:</td>
</tr>
<tr>
<td>&lt;SkipUserConsent&gt;</td>
<td>Set to false if the users must be prompted for approval.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;MultifactorAuthentication&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;XMPPSettings&gt;</td>
<td>XMPP setting for multifactor authentication.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;XMPPConfig&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;XMPPProvider&gt;</td>
<td>XMPP provider name.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;XMPPServer&gt;</td>
<td>XMPP server name.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;XMPPPort&gt;</td>
<td>XMPP server's port.</td>
<td>Int</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;XMPPExt&gt;</td>
<td>XMPP domain.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;XMPPUserName&gt;</td>
<td>User name for login to XMPP server.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;XMPPPassword&gt;</td>
<td>Password for login to XMPP server.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;SSOService&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;IdentityProviderURL&gt;</td>
<td>Unique identifier for IDP. This would be passed as Issuer in SAML2 response.</td>
<td>String</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;SingleLogoutRetryCount&gt;</td>
<td>Number of retries that must be done if a single logout request is not received from the SP.</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td>&lt;SingleLogoutRetryInterval&gt;</td>
<td>Interval between two re-tries.</td>
<td>Int</td>
<td>60 Seconds</td>
</tr>
<tr>
<td>&lt;TenantPartitioningEnabled&gt;</td>
<td>This would add the tenant domain as parameter into the ACS URL.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;SessionTimeout&gt;</td>
<td>Remember me session timeout in seconds.</td>
<td>Int</td>
<td>36000 Seconds</td>
</tr>
<tr>
<td>&lt;AttributesClaimDialect&gt;</td>
<td>Claim Dialect URI that is used for claim retrieving.</td>
<td>String</td>
<td><a href="http://ws">http://ws</a></td>
</tr>
<tr>
<td>&lt;AcceptOpenIDLogin&gt;</td>
<td>Skips authentication if the valid OpenID login session is available.</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>&lt;ClaimsRetrieverImplClass&gt;</td>
<td>Claim retrieving class name for generating a token.</td>
<td>String</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Configuring master-datasources.xml

Users can change the default configurations by editing the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file using the information in the following table.

**XML Elements**

Click on the table and use the left and right arrow keys to scroll horizontally. For sample values, see the Example below the table.

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;datasources-configuration&gt;</code></td>
<td>xmlns</td>
<td>The root element. The namespace is specified as: xmlns:svns=&quot;<a href="http://org.wso2.securevault/configuration">http://org.wso2.securevault/configuration</a>&quot;</td>
<td></td>
</tr>
<tr>
<td><code>&lt;providers&gt;</code></td>
<td></td>
<td>The container element for the datasource providers.</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>&lt;provider&gt;</td>
<td>The datasource provider, which should implement org.wso2.carbon.ndatasource.common.spi.DataSourceReader. The datasources follow a pluggable model in providing datasource type implementations using this approach.</td>
<td>Fully qualified Java class</td>
<td></td>
</tr>
<tr>
<td>&lt;datasources&gt;</td>
<td>The container element for the datasources.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;datasource&gt;</td>
<td>The root element of a datasource.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>Name of the datasource.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>&lt;description&gt;</td>
<td>Description of the datasource.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>&lt;jndiConfig&gt;</td>
<td>The container element that allows you to expose this datasource as a JNDI datasource.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>The JNDI resource name to which this datasource will be bound.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>&lt;environment&gt;</td>
<td>The container element in which you specify the following JNDI properties:</td>
<td>Fully qualified Java class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• java.naming.factory.initial: Selects the registry service provider as the initial context.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• java.naming.provider.url: Specifies the location of the registry when the registry is being used as the initial context.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;definition&gt;</td>
<td>The container element for the data source definition.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;configuration&gt;</td>
<td>The container element for the RDBMS properties.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;url&gt;</td>
<td>The connection URL to pass to the JDBC driver to establish the connection.</td>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>&lt;username&gt;</td>
<td>The connection user name to pass to the JDBC driver to establish the connection.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>&lt;password&gt;</td>
<td>The connection password to pass to the JDBC driver to establish the connection.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>&lt;driverClassName&gt;</td>
<td>The class name of the JDBC driver to use.</td>
<td>Fully qualified Java class</td>
<td></td>
</tr>
<tr>
<td>&lt;maxActive&gt;</td>
<td>The maximum number of active connections that can be allocated from this pool at the same time.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>&lt;maxWait&gt;</td>
<td>Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>&lt;testOnBorrow&gt;</td>
<td>Specifies whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. When set to true, the validationQuery parameter must be set to a non-null string.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>&lt;validationQuery&gt;</td>
<td>The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>&lt;validationInterval&gt;</td>
<td>To avoid excess validation, only run validation at most at this frequency (interval time in milliseconds). If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).</td>
<td>Long</td>
<td></td>
</tr>
</tbody>
</table>

*Example*
<datasources-configuration
xmlns:svns="http://org.wso2.securevault/configuration">
  <providers>
    <provider>
      org.wso2.carbon.ndatasource.rdbms.RDBMSDataSourceReader
    </provider>
  </providers>
  <datasources>
    <datasource>
      <name>WSO2_CARBON_DB</name>
      <description>The datasource used for registry and user manager</description>
      <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
      </jndiConfig>
      <definition type="RDBMS">
        <configuration>
          <url>
            jdbc:h2:repository/database/WSO2CARBON_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000
          </url>
          <username>wso2carbon</username>
          <password>wso2carbon</password>
          <driverClassName>org.h2.Driver</driverClassName>
          <maxActive>50</maxActive>
          <maxWait>60000</maxWait>
          <testOnBorrow>true</testOnBorrow>
          <validationQuery>SELECT 1</validationQuery>
          <validationInterval>30000</validationInterval>
        </configuration>
      </definition>
    </datasource>
  </datasources>
</datasources-configuration>

Configuring registry.xml

Users can change the default configurations by editing the `<PRODUCT_HOME>/repository/conf/registry.xml` file using the information given below.

Click on the table and use the left and right arrow keys to scroll horizontally.

**XML Elements**

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;wso2registry&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © WSO2 Inc. 2014
<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;currentDBConfig&gt;</code></td>
<td>The server can only handle one active configuration at a time. The currentDBConfig parameter defined in the registry.xml is used to specify the database configuration that is active at present. The valid name of a database configuration defined on the registry.xml file. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Data">Database Configuration Details</a></td>
</tr>
<tr>
<td><code>&lt;readOnly&gt;</code></td>
<td>To run the registry in read-only mode, set the readOnly element to true. Setting the read-only mode allows you to run an immutable instance of registry repository. This setting is valid on a global level. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Registry+Configuration+Details">Registry Configuration Details</a></td>
</tr>
<tr>
<td><code>&lt;enableCache&gt;</code></td>
<td>To enable registry caching, set the enableCache element to true. Once caching is enabled, repetitive read operations will be executed against the cache instead of the database. This setting is valid on a global level.</td>
</tr>
<tr>
<td><code>&lt;registryRoot&gt;</code></td>
<td>The registryRoot parameter can be used to define the apparent root of the running instance of the server. This setting is valid on a global level. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Registry+Configuration+Details">Registry Configuration Details</a></td>
</tr>
<tr>
<td><code>&lt;dbConfig&gt;</code></td>
<td>name</td>
</tr>
<tr>
<td><code>&lt;dataSource&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;handler&gt;</code></td>
<td>Handlers are pluggable components, that contain custom processing logic for handling resources. All handlers extend an abstract class named Handler, which provides default implementations for resource handling methods as well as a few utilities useful for concrete Handler implementations can provide alternative behaviors for basic methods in the Handler class. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Handler+Configuration+Details">Handler Configuration Details</a></td>
</tr>
<tr>
<td><code>&lt;filter&gt;</code></td>
<td>class</td>
</tr>
<tr>
<td><code>&lt;remoteInstance&gt;</code></td>
<td>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. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Remote+Instance+and+Mount+Configuration+Details">Remote Instance and Mount Configuration Details</a></td>
</tr>
<tr>
<td><code>&lt;ID&gt;</code></td>
<td>url The URL of the remote instance.</td>
</tr>
<tr>
<td></td>
<td>Remote instance ID.</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;username&gt;</td>
<td>Username of the remote registry login.</td>
</tr>
<tr>
<td>&lt;password&gt;</td>
<td>Password of the remote registry login.</td>
</tr>
<tr>
<td>&lt;dbConfig&gt;</td>
<td>The database configuration to use.</td>
</tr>
<tr>
<td>&lt;readOnly&gt;</td>
<td>To run the registry in read-only mode set the readOnly element to true. This setting is valid only for the specific remote instance. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Registry+Configuration+Details">http://docs.wso2.org/display/Governance501/Registry+Configuration+Details</a></td>
</tr>
<tr>
<td>&lt;enableCache&gt;</td>
<td>To enable registry caching, set the enableCache element to true. Once caching is enabled, repetitive read operations will be executed against the cache instead of the database. This setting is valid only for the specific remote instance. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Registry+Configuration+Details">http://docs.wso2.org/display/Governance501/Registry+Configuration+Details</a></td>
</tr>
<tr>
<td>&lt;registryRoot&gt;</td>
<td>The registryRoot parameter can be used to define whether the app setting is valid only for the specific remote instance. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Registry+Configuration+Details">http://docs.wso2.org/display/Governance501/Registry+Configuration+Details</a></td>
</tr>
<tr>
<td>&lt;mount&gt;</td>
<td>Once a remote instance has been defined, a collection on the remote instance can be mounted to the local instance. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Remote+Instance+and+Mount+Configuration+Details">http://docs.wso2.org/display/Governance501/Remote+Instance+and+Mount+Configuration+Details</a></td>
</tr>
<tr>
<td>&lt;instanceID&gt;</td>
<td>Remote instance ID.</td>
</tr>
<tr>
<td>&lt;targetPath&gt;</td>
<td>The path on the remote registry.</td>
</tr>
<tr>
<td>&lt;path&gt;</td>
<td>The path to which the mount will be added to.</td>
</tr>
<tr>
<td>&lt;overwrite&gt;</td>
<td>Whether an existing collection at the given path would be overwritten or not.</td>
</tr>
<tr>
<td>&lt;versionResourcesOnChange&gt;</td>
<td>You can configure whether you want to auto-version the resources (non-collection) by setting the versionResourcesOnChange element to true. In this configuration it will create a version for the resource whenever it is updated. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources">http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources</a></td>
</tr>
<tr>
<td>&lt;staticConfiguration&gt;</td>
<td>While most configuration options can be changed after the first run of the server, 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. You are supposed to change the static configuration section only before the first start-up.) For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources">http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources</a></td>
</tr>
<tr>
<td>&lt;versioningProperties&gt;</td>
<td>Whether the properties are versioned when a snapshot is created. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources">http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources</a></td>
</tr>
<tr>
<td>&lt;versioningComments&gt;</td>
<td>Whether the comments are versioned when a snapshot is created. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources">http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources</a></td>
</tr>
<tr>
<td>&lt;versioningTags&gt;</td>
<td>Whether the tags are versioned when a snapshot is created.</td>
</tr>
<tr>
<td>&lt;versioningRatings&gt;</td>
<td>Whether the ratings are versioned when a snapshot is created. For more information, see the Governance Registry documentation here: <a href="http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources">http://docs.wso2.org/display/Governance501/Configuration+for+Static+%28One-time%29+and+Auto+Versioning+Resources</a></td>
</tr>
</tbody>
</table>

**Configuring user-mgt.xml**
Users can change the default user management functionality related configurations by editing the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file using the information given below.

Click on the table and use the left and right arrow keys to scroll horizontally.

**XML Elements**

<table>
<thead>
<tr>
<th>XML element</th>
<th>Attribute</th>
<th>Description</th>
<th>Data type</th>
<th>Default value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;UserManager&gt;</td>
<td></td>
<td>User kernel configuration for Carbon server.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Realm&gt;</td>
<td></td>
<td>Realm configuration.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;Configuration&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;AddAdmin&gt;</td>
<td></td>
<td>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 &lt;AdminUser&gt; 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.</td>
<td>Boolean</td>
<td>true</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;AdminRole&gt;</td>
<td></td>
<td>The role name that is used as an admin role for the Carbon server.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;AdminUser&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;UserName&gt;</td>
<td>User name that is used to represent an admin user for the Carbon server.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td>&lt;Password&gt;</td>
<td>Password of the admin user, If the admin user needs to be created in the Carbon server.</td>
<td>String</td>
<td>N/A</td>
<td>Optional</td>
</tr>
<tr>
<td>&lt;EveryOneRoleName&gt;</td>
<td></td>
<td>By default, every user in the user store is assigned to this role.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;Property&gt;</td>
<td></td>
<td>User realm configuration specific property values.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
User Store manager implementation classes and their configurations for use realm. Use the
ReadOnlyLDAPUserStoreManager to do read-only operations for external LDAP user stores.

To do both read and write operations, use the ReadWriteLDAPUserStoreManager for external LDAP user stores.

If you wish to use an Active Directory Domain Service (AD DS) or Active Directory Lightweight Directory Service (AD LDS), use the
ActiveDirectoryUserStoreManager. This can be used for both read-only and read/write operations.

Use JDBCUserStoreManager for both internal and external JDBC user stores.

<table>
<thead>
<tr>
<th>&lt;UserStoreManager&gt;</th>
<th>User store configuration specific property values. See working with primary user store properties for more information.</th>
<th>String</th>
<th>N/A</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Property&gt;</td>
<td>Authorization manager implementation class and its configuration for user realm.</td>
<td>String</td>
<td>N/A</td>
<td>Mandatory</td>
</tr>
<tr>
<td>&lt;AuthorizationManager&gt;</td>
<td>Authorization manager configuration specific property values.</td>
<td>String</td>
<td>N/A</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**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**
- **Product-specific ports**

**Common ports**

The following ports are common to all WSO2 products that provide the given feature. Some features are bundled in the WSO2 Carbon platform itself and therefore are available in all WSO2 products by default.

*Management console ports*
WSO2 products that provide a management console use the following servlet transport ports:

- 9443 - HTTPS servlet transport (the default URL of the management console is https://localhost:9443/carbon)
- 9763 - HTTP servlet transport

**LDAP server ports**

Provided by default in the WSO2 Carbon platform.

- 10389 - Used in WSO2 products that provide an embedded LDAP server

**KDC ports**

- 8000 - Used to expose the Kerberos key distribution center server

**JMX monitoring ports**

WSO2 Carbon platform uses TCP ports to monitor a running Carbon instance using a JMX client such as JConsole. By default, JMX is enabled in all products. You can disable it using <PRODUCT_HOME>/repository/conf/etc/jmx.xml file.

- 11111 - RMIRegistry port. Used to monitor Carbon remotely
- 9999 - RMIServer port. Used along with the RMIRegistry port when Carbon is monitored from a JMX client that is behind a firewall

**Clustering ports**

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

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

**Random ports**

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

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

**Product-specific ports**

Some products open additional ports.

**API Manager | BAM | BPS | Data Analytics Server | Complex Event Processor | Elastic Load Balancer | ESB | Identity Server | Message Broker | Machine Learner | Storage Server | Enterprise Mobility Manager**

**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.
BAM

- 9160 - Cassandra port using which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to BAM
- 7611 - Thrift TCP port to receive events from clients to BAM
- 21000 - Hive Thrift server starts on this port

BPS

- 2199 - RMI registry port (datasources provider port)

Data Analytics Server

- 9160 - Cassandra port on which Thrift listens to clients
- 7711 - Thrift SSL port for secure transport, where the client is authenticated to DAS
- 7611 - Thrift TCP port to receive events from clients to DAS
- For a list of Apache Spark related ports, see WSO2 Data Analytics Server Documentation - Spark Configurationss.

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

ESB

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

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

Identity Server

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

Message Broker

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

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

Machine Learner

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

**Storage Server**

Cassandra:

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

HDFS:

- 54310 - Port used to connect to the default file system.
- 54311 - Port used by the MapRed job tracker
- 50470 - Name node secure HTTP server port
- 50475 - Data node secure HTTP server port
- 50010 - Data node server port for data transferring
- 50075 - Data node HTTP server port
- 50020 - Data node IPC server port

**Enterprise Mobility Manager**

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

**Android:**
The ports to open are 5228, 5229 and 5230. GCM/FCM typically only uses 5228, but it sometimes uses 5229 and 5230. GCM/FCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google’s ASN of 15169.

**iOS:**

- 5223 - TCP port used by devices to communicate to APNs servers
- 2195 - TCP port used to send notifications to APNs
- 2196 - TCP port used by the APNs feedback service
- 443 - TCP port used as a fallback on Wi-Fi, only when devices are unable to communicate to APNs on port 5223

The APNs servers use load balancing. The devices will not always connect to the same public IP address for notifications. The entire 17.0.0.0/8 address block is assigned to Apple, so it is best to allow this range in the firewall settings.

**API Manager:**

```
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
## Getting Support

In addition to this documentation, there are several ways to get help as you work on WSO2 products.

<table>
<thead>
<tr>
<th><strong>Explore learning resources</strong>: For tutorials, articles, whitepapers, webinars, and other learning resources, look in the Resources menu on the WSO2 website. For training materials, click WSO2 Training on the Support &amp; Training menu. In products that have a visual user interface, click the Help link in the top right-hand corner to get help with your current task.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Try our support options</strong>: WSO2 offers a variety of development and production support programs, ranging from web-based support during normal business hours to premium 24x7 phone support. For support information, see <a href="http://wso2.com/support/">http://wso2.com/support/</a>.</td>
</tr>
<tr>
<td><strong>Ask questions in the user forums</strong> at <a href="http://stackoverflow.com">http://stackoverflow.com</a>. Ensure that you tag your question with appropriate keywords such as WSO2 and the product name so that our team can easily find your questions and provide answers. If you can't find an answer on the user forum, you can email the WSO2 development team directly using the relevant mailing lists described at <a href="http://wso2.org/mail">http://wso2.org/mail</a>.</td>
</tr>
<tr>
<td><strong>Report issues</strong>, submit enhancement requests, track and comment on issues using our public bug-tracking system, and contribute samples, patches, and tips &amp; tricks (see the WSO2 Contributor License Agreement).</td>
</tr>
</tbody>
</table>
Site Map

Use this site map to quickly find the topic you are looking for by searching for a title on this page using your browser's search feature. You can also use the search box in the upper right corner of this window to search for a word or phrase in all the pages in this documentation.