



API Manager

Documentation

WSO2 API Manager

Documentation

Version 1.8.0

Table of Contents

1. WSO2 API Manager Documentation	5
1.1 About API Manager	5
1.1.1 Introducing the API Manager	6
1.1.2 About this Release	6
1.2 Getting Started	7
1.2.1 Quick Start Guide	8
1.2.2 Downloading the Product	31
1.2.3 Installation Prerequisites	31
1.2.4 Installing the Product	34
1.2.4.1 Installing on Linux or OS X	35
1.2.4.2 Installing on Solaris	36
1.2.4.3 Installing on Windows	37
1.2.4.4 Installing as a Linux Service	40
1.2.4.5 Installing as a Windows Service	42
1.2.5 Building from Source	47
1.2.6 Running the Product	50
1.2.7 Upgrading from the Previous Release	51
1.2.8 Get Involved	53
1.2.8.1 WSO2 GitHub Repositories	56
1.3 User Guide	58
1.3.1 Key Concepts	59
1.3.2 API Developer Tutorials	78
1.3.2.1 Create and Publish an API	80
1.3.2.2 Edit an API from the Source Code	84
1.3.2.3 Add API Documentation	85
1.3.2.3.1 Add API Documentation In-line, using a URL or a File	85
1.3.2.3.2 Add Apache Solr-Based Indexing	91
1.3.2.4 Manage the API Lifecycle	93
1.3.2.4.1 Create a new API Version	93
1.3.2.4.2 Deploy and Test as a Prototype	96
1.3.2.4.3 Publish the new Version and Deprecate the old	100
1.3.2.5 Publish to multiple external API stores	102
1.3.2.6 Engage a new Throttling Policy	105
1.3.2.7 Block Subscription to an API	107
1.3.2.8 Enforce Throttling and Resource Access Policies	112
1.3.3 Application Developer Tutorials	116
1.3.3.1 Subscribe to an API	118
1.3.3.2 Invoke an API using the Integrated API Console	123
1.3.3.3 Invoke an API using the Integrated REST Client	123
1.3.3.4 Use the Community Features	130
1.3.3.5 Invoke an API using a SOAP Client	134
1.3.4 Configuring the API Manager	138
1.3.4.1 Customizing the API Store	138
1.3.4.2 Configuring Multiple Tenants	142
1.3.4.2.1 Multi Tenant Architecture	142

1.3.4.2.2 Managing Tenants	145
1.3.4.2.3 Tenant-Aware Load Balancing using WSO2 ELB	147
1.3.4.3 Adding Internationalization and Localization	148
1.3.4.4 Configuring Single Sign-on with SAML2	149
1.3.4.5 Changing the Default Transport	156
1.3.4.6 Configuring Caching	158
1.3.4.7 Working with Databases	161
1.3.4.7.1 Setting up the Physical Database	162
1.3.4.7.2 Managing Datasources	199
1.3.4.8 Managing Users and Roles	209
1.3.4.8.1 Adding User Roles	210
1.3.4.8.2 Adding Users	212
1.3.4.9 Configuring User Stores	215
1.3.4.9.1 Realm Configuration	216
1.3.4.9.2 Changing the RDBMS	217
1.3.4.9.3 Configuring Primary User Stores	218
1.3.4.9.4 Configuring Secondary User Stores	234
1.3.4.10 Directing the Root Context to the API Store	236
1.3.4.11 Adding Links to Navigate Between the Store and Publisher	236
1.3.4.12 Maintaining Separate Production and Sandbox Gateways	237
1.3.4.13 Configuring Transports	240
1.3.5 Extending the API Manager	240
1.3.5.1 Writing Custom Handlers	241
1.3.5.2 Integrating with WSO2 Governance Registry	245
1.3.5.3 Adding Mediation Extensions	246
1.3.5.4 Adding Workflow Extensions	248
1.3.5.4.1 Adding an Application Creation Workflow	248
1.3.5.4.2 Adding an Application Registration Workflow	251
1.3.5.4.3 Adding an API Subscription Workflow	254
1.3.5.4.4 Adding a User Signup Workflow	257
1.3.5.4.5 Invoking the API Manager from the BPEL Engine	260
1.3.5.4.6 Customizing a Workflow Extension	261
1.3.5.4.7 Configuring Workflows for Tenants	265
1.3.5.4.8 Configuring Workflows in a Cluster	272
1.3.5.4.9 Changing the Default User Role in Workflows	275
1.3.5.5 Adding new Throttling Tiers	275
1.3.5.6 Adding a Reverse Proxy Server	277
1.3.5.7 Adding a new API Store Theme	277
1.3.5.8 Transforming API Message Payload	281
1.3.6 Working with Security	289
1.3.6.1 Passing Enduser Attributes to the Backend Using JWT	290
1.3.6.2 Encrypting Passwords	294
1.3.6.3 Maintaining Logins and passwords	297
1.3.6.4 Saving Access Tokens in Separate Tables	298
1.3.6.5 Configuring WSO2 Identity Server as the Key Manager	300
1.3.6.6 Configuring Transport Level Security	300
1.3.6.7 Enabling the Java Security Manager	303
1.3.7 Working with Statistics	306

1.3.7.1 Publishing API Runtime Statistics	306
1.3.7.2 Integrating with Google Analytics	310
1.3.7.3 Viewing API Statistics	312
1.4 Samples	318
1.4.1 Setting up the Samples	319
1.4.2 Deploying and Testing YouTube API	320
1.4.3 Generating Billing Data	322
1.4.4 Invoking APIs using a Web App Deployed in WSO2 AS	325
1.4.5 Deploying and Testing a Wikipedia API	327
1.5 Published APIs	328
1.5.1 Publisher APIs	329
1.5.2 Store APIs	334
1.5.3 Token API	339
1.5.4 WSO2 Admin Services	350
1.6 Admin Guide	354
1.6.1 Migrating the API Manager	355
1.6.2 Deploying and Clustering the API Manager	356
1.6.3 Tuning Performance	356
1.7 Reference Guide	361
1.7.1 Product Profiles	362
1.7.2 Default Product Ports	363
1.7.3 Changing the Default Ports with Offset	365
1.7.4 Error Handling	367
1.7.5 WSO2 Patch Application Process	368
1.8 FAQ	370
1.9 Getting Support	377
1.10 Site Map	378

WSO2 API Manager Documentation

Welcome to WSO2 API Manager Documentation! [WSO2 API Manager \(APIM\)](#) is a fully open source, complete solution for creating, publishing and managing all aspects of an API and its lifecycle, and is ready for massively scalable deployments.

Use the descriptions below to find the section you need, and then browse the topics in the left navigation panel. You can also use the **Search** box on the left to find a term in this documentation, or use the search box in the top right-hand corner to search in all WSO2 product documentation.

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

About API Manager Introduces WSO2 API Manager, including the business cases it solves, its features, architecture and how to get help.	Getting Started Instructions to download, install, run and get started quickly with WSO2 API Manager.	User Guide Introduces the features and functionality of the API Manager, solution development, testing, debugging and deployment.
Admin Guide Introduces product deployment and other system administration tasks.	Samples Real-life business use cases of the product.	Published APIs APIs to be used in your applications.

About API Manager

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

- [Introducing the API Manager](#)
- [About this Release](#)

Introducing the API Manager

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

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

WSO2 API Manager overcomes these challenges with a set of features for API creation, publication, lifecycle management, versioning, monetization, governance, security etc. using proven WSO2 products such as [WSO2 Enterprise Service Bus](#), [WSO2 Identity Server](#), and [WSO2 Governance Registry](#). In addition, as it is also powered by the [WSO2 Business Activity Monitor](#) and is immediately ready for massively scalable deployments.

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

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

About this Release

What is new in this release

The WSO2 API Manager version **1.8.0** is the successor of version **1.7.0**. It contains the following new features and enhancements:

- Upgraded Swagger. You now have the facility to edit the Swagger definition of an API in the API design time.
- Improvements added to the ability to set up a reverse proxy server to pass server requests. See [Adding a Reverse Proxy Server](#).
- Auto populated access token in the Swagger API console. See [Invoke an API using Swagger](#).
- Performance improvements in the API Store by retrieving subscription details only for the selected application.
- Self sign-up option available for tenant stores. The super admin no longer has to manually add a tenant user to that store using the admin console.
- Ability to plug in a custom JWT generator with your own logic to generate claims. See [Passing Enduser Attributes to the Backend Using JWT](#).
- Ability to enable self sign up support for tenant API Stores. See [Enabling Self Sign-up](#).

Compatible WSO2 product versions

The following products were tested for compatibility with WSO2 APIM 1.8.0:

- WSO2 Governance Registry 4.6.0
- WSO2 Identity Server 5.0.0
- WSO2 Business Activity Monitor 2.5.0

WSO2 APIM 1.8.0 is based on WSO2 Carbon 4.2.0 and is expected to be compatible with any other WSO2 product

that is based on the same Carbon version. If you get any compatibility issues, please contact team WSO2. For information on the third-party software required with APIM 1.8.0, see [Installation Prerequisites](#).

Deprecated features

- From the 1.8.0, the [integrated WSO2 REST client](#) in the API Store is deprecated and it will be removed from the next release onwards. We encourage you to use the [API console](#) to invoke the APIs.

Fixed issues

For a list of fixed issues in this release, see [WSO2 API Manager 1.8.0- Fixed Issues](#).

Known issues

For a list of known issues, see [WSO2 API Manager- Known Issues](#).

Getting Started

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

- [Quick Start Guide](#)
- [Downloading the Product](#)
- [Installation Prerequisites](#)
- [Installing the Product](#)
- [Building from Source](#)
- [Running the Product](#)
- [Upgrading from the Previous Release](#)
- [Get Involved](#)

Quick Start Guide

WSO2 API Manager is a complete solution for publishing APIs, creating and managing a developer community and for routing API traffic in a scalable manner. It leverages the integration, security and governance components from the WSO2 Enterprise Service Bus, WSO2 Identity Server, and WSO2 Governance Registry. In addition, as it is powered by the WSO2 Business Activity Monitor (BAM), the WSO2 API Manager is ready for massively scalable deployments immediately.

This guide walks you thorough the basic usecases of the API Manager:

- [Introduction to basic concepts](#)
- [Starting the API Manager](#)
- [Creating users and roles](#)
- [Creating an API](#)
- [Adding API documentation](#)
- [Adding interactive documentation](#)
- [Versioning the API](#)
- [Publishing the API](#)
- [Subscribing to the API](#)
- [Invoking the API](#)
- [Monitoring APIs and viewing statistics](#)

Introduction to basic concepts

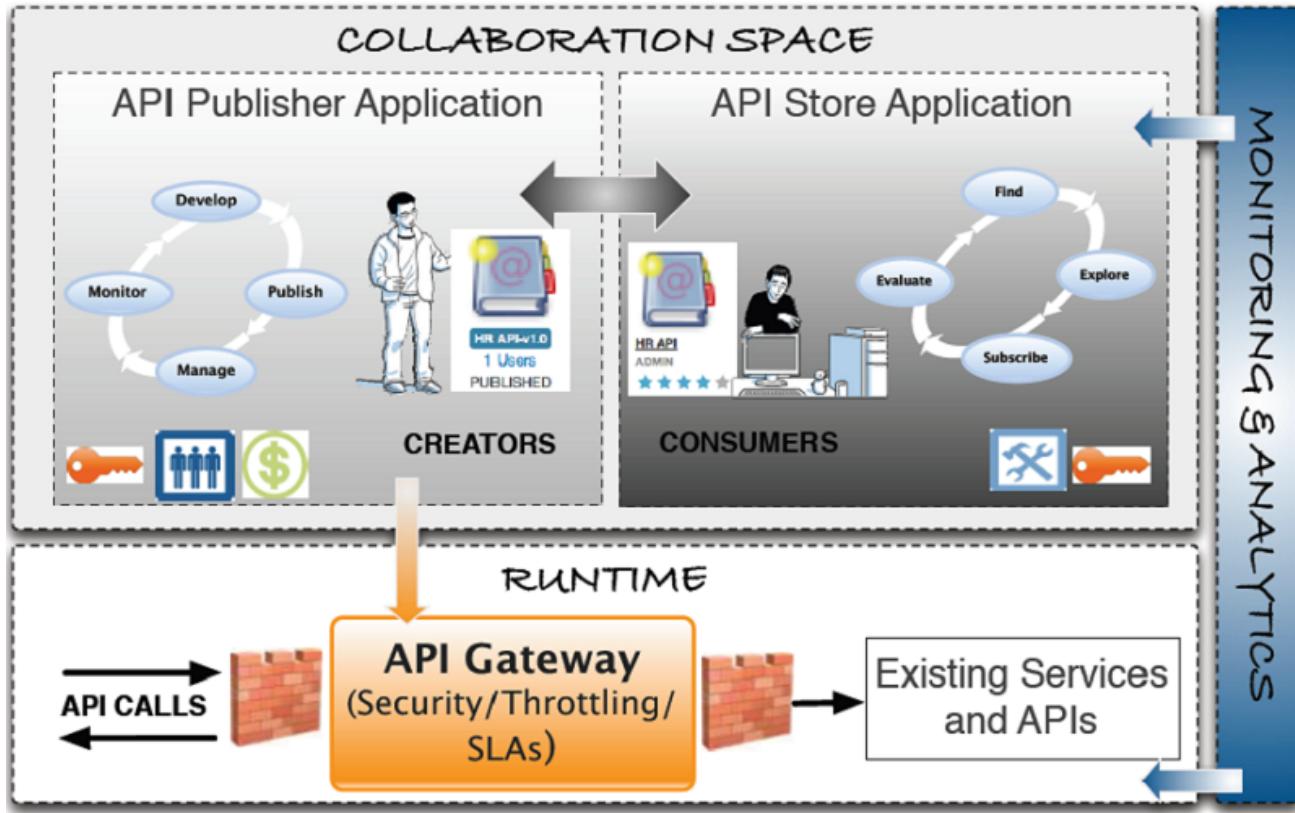
Let's take a look at the basic concepts that you need to know before using the API Manager.

[[Components](#)] [[Users and roles](#)] [[API life cycle](#)] [[Applications](#)] [[Throttling tiers](#)] [[API keys](#)] [[Application access tokens](#)] [[Application user access token](#)] [[API resources](#)]

Components

The API Manager comprises of the following components:

- **API Gateway:** Secures, protects, manages, and scales API calls. It is a simple API proxy that intercepts API requests and applies policies such as throttling and security checks. It is also instrumental in gathering API usage statistics. The Web interface can be accessed via `https://<Server Host>:9443/carbon`.
- **Key Manager:** Handles all security and key-related operations. API gateway connects with the Key Manager to check the validity of OAuth tokens when the APIs are invoked. The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway.
- **API Publisher:** Enables API providers to publish APIs, share documentation, provision API keys, and gather feedback on features, quality and usage. You access the Web interface via `https://<Server Host>:9443/publisher`.
- **API Store:** Enables API consumers to self register, discover and subscribe to APIs, evaluate them and interact with API publishers. You access the Web interface via `https://<Server Host>:9443/store`.
- Additionally, statistics are provided by the monitoring component, which integrates with WSO2 BAM.



Users and roles

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

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

API life cycle

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

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

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

the API Store anymore.

You can manage the API and service life cycles in the same governance registry/repository and automatically link them. This feature is available in WSO2 Governance Registry (version 4.5 onwards).

Applications

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

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

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

Throttling tiers

Throttling tiers are associated to an API at subscription time. They define the throttling limits enforced by the API Gateway. E.g., 10 TPS (transactions per second). You define the list of tiers that are available for a given API at the publisher level. The API Manager comes with three predefined tiers (Gold/Silver/Bronze) and a special tier called `Unlimited`, which you can disable by editing the `<TierManagement>` element of `<APIM_HOME>/repository/conf/api-manager.xml` file.

API keys

The API Manager supports two scenarios for authentication:

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

Application access tokens

Application access tokens are generated by the API consumer and must be passed in the incoming API requests. The API Manager uses the OAuth2 standard to provide key management. An API key is a simple string that you pass with an HTTP header (e.g., "Authorization: Bearer NtBQkXoKElu0H1a1fQ0DWf06IX4a") and it works equally well for SOAP and REST calls.

Application access tokens are generated at the application level and valid for all APIs that you associate to the application. These tokens have a fixed expiration time, which is set to 60 minutes by default. You can change this to a longer time, even for several weeks. Consumers can regenerate the access token directly from the API Store. To change the default expiration time, you open the `<APIM_HOME>/repository/conf/identity.xml` file and change the value of the element `<ApplicationAccessTokenDefaultValidityPeriod>`. If you set a negative value, the token never expires.

Application user access token

You generate access tokens on demand using the Token API. In case a token expires, you use the Token API to refresh it.

Application user access tokens have a fixed expiration time, which is 60 minutes by default. You can update it to a longer time by editing the `<ApplicationAccessTokenDefaultValidityPeriod>` element in the `<APIM_HOME>/repository/conf/identity.xml` file.

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

- Grant Type
- Username
- Password
- Scope

To generate a new access token, you issue a Token API call with the above parameters where `grant_type=pass`

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

API resources

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

- **verbs**: Specifies the HTTP verbs a particular resource accepts. Allowed values are GET, POST, PUT, OPTIONS, DELETE. You can give multiple values at once.
- **uri-template**: A URI template as defined in <http://tools.ietf.org/html/rfc6570> (e.g., /phoneregister/<phoneNumber>)
- **url-mapping**: A URL mapping defined as per the servlet specification (extension mappings, path mappings and exact mappings)
- **Throttling tiers**: Limits the number of hits to a resource during a given period of time.
- **Auth-Type**: Specifies the Resource level authentication along the HTTP verbs. Auth-type can be None, Application or Application User.
 - None : Can access the particular API resource without any access tokens
 - Application: An application access token is required to access the API resource
 - Application User: A user access token is required to access the API resource

Starting the API Manager

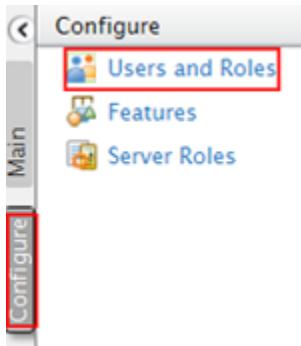
1. Download WSO2 API Manager from <http://wso2.com/api-management/try-it>.
2. Install [Oracle Java SE Development Kit \(JDK\)](#) version 1.6.24 or later or 1.7.*.
3. Set the `JAVA_HOME` environment variable.
4. Using the command line, go to `<APIM_HOME>/bin` and execute `wso2server.bat` (for Windows) or `wso2server.sh` (for Linux).
5. Wait until you see the message "WSO2 Carbon started in 'n' seconds" where 'n' can be any number of seconds.

The server started successfully. To stop the API Manager, simply hit Ctrl-C in the command window.

Creating users and roles

In [Users and roles](#), we introduced a set of users that are commonly found in many enterprises. Let's see how you can log in to the Management Console as an admin and create these roles.

1. Log in to the Management Console (<https://<hostname>:9443/carbon>) of the API Manager using admin/admin credentials.
2. Select the **Users and Roles** menu under the **Configure** menu.



3. Click the **Roles** link and then click **Add New Role**.

Roles

Search

Select Domain: ALL-USER-STORE-DOMAINS

Enter role name pattern (* for all): *

Actions

Name	Actions
admin	Assign Users View Users
Internal/everyone	Permissions
Internal/identity	Rename Permissions Assign Users View Users Delete
Internal/subscriber	Rename Permissions Assign Users View Users Delete

[Add New Role](#)

[Add New Internal Role](#)

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

Add Role

Step 1 : Enter role details

Enter role details

Domain: PRIMARY

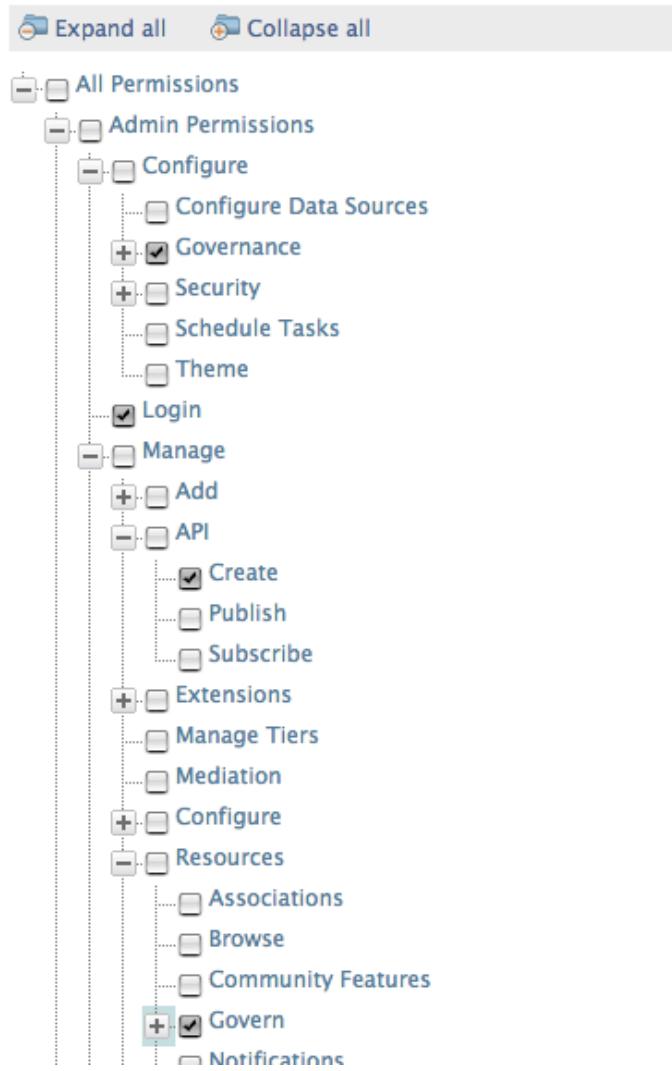
Role Name*: `creator`

Next > **Finish** **Cancel**

5. A list of permissions opens. Select the following and click **Finish**.

- Configure > Governance and all underlying permissions.
- Login
- Manage > API > Create
- Manage > Resources > Govern and all underlying permissions

Step 2 : Select permissions to add to Role



6. Similarly, create the publisher role with the following permissions.
 - Login
 - Manage > API > Publish
7. Note that the API Manager comes with the `subscriber` role available by default. It has the following permissions:
 - Login
 - Manage > API > Subscribe
8. Note that you have the following roles added:

Roles

Name	Actions
admin	Assign Users View Users
creator	Rename Permissions Assign Users View Users Delete
Internal/everyone	Permissions
Internal/identity	Rename Permissions Assign Users View Users Delete
Internal/subscriber	Rename Permissions Assign Users View Users Delete
publisher	Rename Permissions Assign Users View Users Delete

[Add New Role](#)

[Add New Internal Role](#)

Let's create users for each of the roles.

9. Click the **Users and Roles** menu under the **Configure** menu again.

10. Click the **Users** link and then click **Add New User**.

Users

Name	Actions
admin	Change Password Assign Roles View Roles User Profile
User1	Change Password Assign Roles View Roles Delete User Profile

[Add New User](#)

[Bulk Import Users](#)

11. Give the username/password and click **Next**. For example, lets create a new user by the name `apipublisher`.

Add User

Step 1 : Enter user name

Enter user name

Domain **PRIMARY**

User Name* **apipublisher**

Password* *********

Password Repeat* *********

Next > **Finish** **Cancel**

12. Select the role you want to assign to the user (e.g., publisher) and **Finish**. Given below is a list of usernames and the roles we assign to them in this guide.

Add User

Step 2 : Select roles of the user

Enter role name pattern (* for all) ***** **Search**

Users of Role

Select all on this page | Unselect all on this page

admin
 creator
 publisher
 Internal/everyone
 Internal/subscriber
 Internal/identity

Finish **Cancel**

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

Creating an API

An API creator uses the API Publisher to create and publish APIs to the API Store. Let's create an API and add interactive Swagger-based documentation to it.

1. Open the API Publisher (<https://<hostname>:9443/publisher>) and log in as **apicreator**.
2. Click the **Add** link and provide the information given in the table below. Click **Implement** once you are done.

Field	Sample value
Name	PhoneVerification
Context	/phoneverify
Version	1.0.0
Visibility	Public

Resources	URL pattern	CheckPhoneNumber
	Request types	GET, POST, OPTIONS

3. Give the following information in the **Implement** tab that opens and click **Manage** once you are done.

Field	Sample value
Implementation method	Backend
Endpoint type	HTTP
Production endpoint	<p>In this guide, we work with a service exposed by the Cdyne services provider. We use their phone validation service, which has SOAP and REST interfaces. Endpoint is http://ws.cdyne.com/phoneverify/phoneverify.asmx.</p> <p>This sample service has two operations as <code>CheckPhoneNumber</code> and <code>CheckPhoneNumbers</code>. Let's use <code>CheckPhoneNumber</code> here.</p>
Endpoint security scheme	Non Secured (If secured, user is asked for credentials of the backend service)



PhoneVerification : /phoneverify/1.0.0

Implementation Method Backend Endpoint Specify Inline

Endpoints

Endpoint Type:

Production Endpoint:
E.g.: <http://appserver/resource>

Sandbox Endpoint:
E.g.: <http://appserver/resource>

[Show More Options](#)

4. Click **Manage** to go to the Manage tab and provide the following information.

Field	Value	Description
Tier Availability	Bronze/Gold/Silver/Unlimited	The API can be available at different level of service; you can select multiple entries from the list. At subscription time, the consumer chooses which tier they are interested in.



PhoneVerification : /phoneverify/1.0.0

Configurations

Make this default version [?](#)
No default version defined for the current API

Tier Availability: **4 selected** [?](#)

Transports: HTTP HTTPS

Sequences: Check to select a custom sequence to be executed in the message flow

Response Caching: **Disabled** [?](#)

Subscriptions: **Available to current** [?](#)

Business Information >

Resources

Tip: For resources that have methods requiring authentication (i.e., Auth Type is not NONE), you set **None** as the Auth type of OPTIONS to support CORS (Cross Origin Resource Sharing) between the API Store and Gateway.

- Once you are done, click **Save**.

Adding API documentation

- After saving the API, click on its thumbnail in the API Publisher to open it.
- Click on the API's **Docs** tab and click the **Add New Document** link.

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Versions](#) **Docs** [Users](#)

[Add New Document](#)

Name	Type	Modified On	Actions
No documentation associated with the API			

- The document options appear. Note that you can create documentation inline, via a URL or as a file. For inline documentation, you can edit the content directly from the API publisher interface. You get several document types:
 - How To

- Samples and SDK
- Public forum / Support forum (external link only)
- API message formats
- Other

4. Create a 'How To,' using in-line content as the source. The document name is SimpleClient and click the **Add** button.

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

[Add New Document](#)

Name*
PhoneVerification

Summary
Check the validity of a phone number

Type
 How To
 Samples & SDK
 Public Forum
 Support Forum
 Other (specify)

Source
 In-line
 URL
 File

[Add Document](#) [Cancel](#)

5. Once the document is added, click **Edit Content** link associated with it to opens an embedded editor.

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

[Add New Document](#)

Name	Type	Modified On	Actions
SimpleClient	How To	4/7/2015, 4:54:28 PM	Edit Content Update Delete

6. Enter your API's documentation.

PhoneVerification

Save
Save and Close
Cancel

Determine whether a phone number is wireless or a landline - in real time. Even if the phone number has been ported between service providers, Phone Verification will return the latest information.

Phone Verification validates the ten digits of a U.S. telephone number and returns carrier information as well as the Location Routing Number (LRN) for telephone numbers administered by the North American Numbering Plan Administration. LRN is a unique 10-digit number that represents a telephone switch through which multiple phone numbers are routed. The LRN enables Local Number Portability (LNP), which allows phone numbers to be ported to different carriers.

The benefits of using the Phone Verification API include:

- Determine if a phone number is valid
- Remove dashes, parenthesis and spaces
- Receive latest SMTP email string data
- Define time zone to restrict calling times
- Validate wireless vs landline data for compliance

Save
Save and Close
Cancel

Adding interactive documentation

The API Manager provides facility to add interactive documentation support through the integration of Swagger. Swagger is a specification and a complete framework implementation for describing, producing, consuming, and visualizing RESTful Web services. You describe APIs in simple, static JSON representation through the Swagger API definition in the API Store. When an API is created, the JSON representation of that API is automatically generated and saved in the registry. This definition reflects the information you provide at the API creation stage. You can customize it as follows:

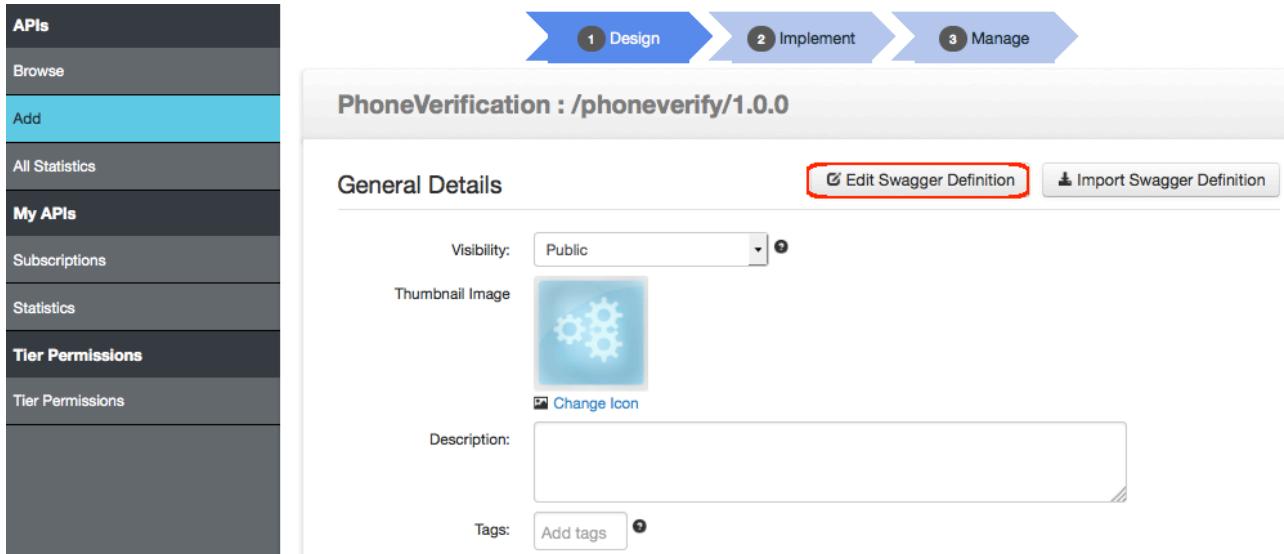
1. Open the API Publisher (<https://<hostname>:9443/publisher>) and log in as **apicreator** if you haven't done so already.
2. Click the PhoneVerification API to open it and then click the **Edit** link right next to the API's name. This opens the API in its edit mode.

PhoneVerification - 1.0.0

Edit

Overview
Versions
Docs
Users

3. Click the **Edit Swagger Definition** button.



The screenshot shows the WSO2 API Manager interface. On the left, a sidebar lists 'APIs', 'Browse', 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions', and 'Tier Permissions'. The main area is titled 'PhoneVerification : /phoneverify/1.0.0'. At the top, a blue ribbon navigation bar shows '1 Design', '2 Implement', and '3 Manage'. Below the ribbon, the 'General Details' section is visible, featuring a 'Visibility' dropdown set to 'Public', a 'Thumbnail Image' with a gear icon, a 'Description' text area, and a 'Tags' section with a 'Add tags' button. A red box highlights the 'Edit Swagger Definition' button.

- When the Swagger definition of the API opens, navigate to the GET method, add the following parameters to it and remove the existing body parameter. The code is given below:

```
parameters:
  - description: Give the phone number to be validated
    name: PhoneNumber
    type: string
    required: "True"
    paramType: query
  - description: "Give the license key. If you don't have any, enter 0"
    name: LicenseKey
    type: string
    required: "True"
    paramType: query
```

- Click **Save** once the changes are done. In a later section, we will see how these parameters appear to subscribers in the API Console of the API Store.

Versioning the API

Let's create a new version of this API.

- Log in to the API Publisher as **apicreator** if you are not logged in already.
- Click on the PhoneVerification API and then the **Copy** button that appears in its **Overview** tab.

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Versions](#) [Docs](#) [Users](#)

	Context	/phoneverify
0 Users	Production URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx
PUBLISHED	Date Last Updated	4/7/2015, 5:07:12 PM
1.0.0	Tier Availability	Bronze,Unlimited,Gold,Silver
Docs	Default API Version	None

[Copy](#)

3. Give a new version number (e.g., 2.0.0) and click **Done**.

To Version

2.0.0 Ex:v1.0.1

Default Version No default version defined for the current API

[Done](#) [Cancel](#)

 **Tip:** The **Default Version** option means that you make this version the default in a group of different versions of the API. A default API can be invoked without specifying the version number in the URL. For example, if you mark <http://host:port/youtube/2.0> as the default version when the API has 1.0 and 3.0 versions as well, requests made to <http://host:port/youtube/> get automatically routed to version 2.0.

If you mark any version of an API as the default, you get two API URLs in its **Overview** page in the API Store. One URL is with the version and the other is without. You can invoke a default version using both URLs.

If you mark an unpublished API as the default, the previous default, published API will still be used as the default until the new default API is published (or prototyped).

A new version of the API is created. It is a duplication of the original API, including its documentation. The PhoneVerification 2.0.0 API is now ready to be published. This is typically done by a user in the publisher role.

Publishing the API

1. Log in to the API Publisher as **apipublisher** that you created earlier in this guide.
2. Click on the PhoneVerification API version 2.0.0. Note that you now see a tab by the name **Lifecycle** in the API Publisher.
3. Go to the **Lifecycle** tab and select the state as **PUBLISHED** from the drop-down list.

PhoneVerification - 2.0.0

Overview **Lifecycle** Versions Docs Users

State: PUBLISHED

Propagate Changes to API Gateway
 Deprecate Old Versions
 Require Re-Subscription

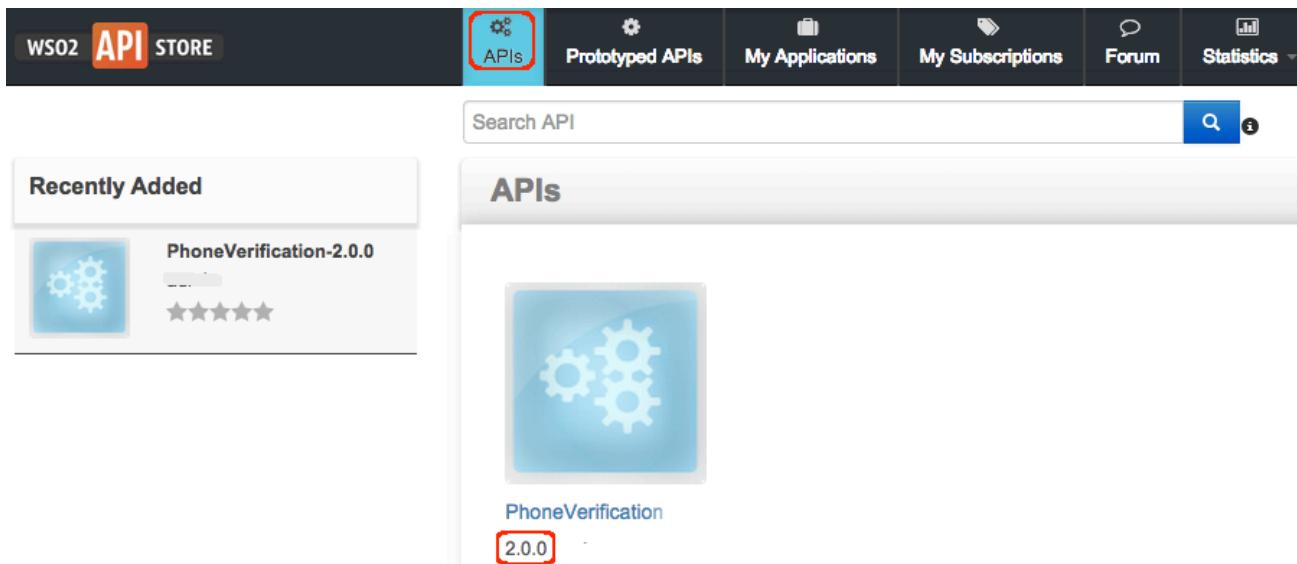
Update Reset

The

three checkboxes mean the following:

- **Propagate Changes to API Gateway:** Used to define an API proxy in the API Gateway runtime component, allowing the API to be exposed to the consumers via the API Gateway. If this option is left unselected, the API metadata will not change and you will have to manually configure the API Gateway according to the information published in the API Store.
- **Deprecate Old Versions:** If selected, any prior versions of the API that are published will be set to the DEPRECATED state automatically.
- **Require Re-Subscription:** Invalidates current user subscriptions, forcing users to subscribe again.

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



The screenshot shows the WSO2 API Store interface. At the top, there is a navigation bar with the WSO2 API STORE logo, a search bar, and several menu items: APIs (highlighted with a red box), Prototyped APIs, My Applications, My Subscriptions, Forum, and Statistics. Below the navigation bar, there are two sections: 'Recently Added' and 'APIs'. The 'Recently Added' section shows a thumbnail of the 'PhoneVerification-2.0.0' API and its rating. The 'APIs' section shows a large thumbnail for the 'PhoneVerification' API, version 2.0.0, with a red box highlighting the version number.

You have now published an API to the API Store. It is ready to be used by subscribers.

Subscribing to the API

You subscribe to APIs using the API Store.

1. Open the API Store (<https://<hostname>:9443/store>).
2. Self sign up to the API Store using the **Sign-up** link.

3. After signing up, log in to the API Store and click the API that you published earlier (PhoneVerification 2.0.0).
4. Note that you can now see the subscription options on the right hand side of the UI. Select the default application and Bronze tier, and click **Subscribe**.

Production and Sandbox URLs:

5. Once the subscription is successful, choose to go to the **My Subscriptions** page.
6. In the **My Subscriptions** page, click the **Generate** buttons to generate access tokens that you need to invoke the API.

Search API

Subscriptions

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple APIs and to subscribe to one API multiple times with different SLA levels.

DefaultApplication

Show Keys

Keys - Production

Production keys are not yet generated for this application.

Generate

Allowed Domains

ALL

The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.

Token Validity: 3600 Seconds

Keys - Sandbox

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

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

Invoking the API

Let's invoke the API using the integrated Swagger-based API Console.

1. Click the **APIs** menu in the API Store and then click on the API that you want to invoke. When the API opens, go to its **API Console** tab.

PhoneVerification - 2.0.0

admin



Rating: Your rating: N/A
★★★★★

Version: 2.0.0

Status: PUBLISHED

Updated: 18/Dec/2014 21:40:38 PM IST

Applications

Tiers

Allows 1 request(s) per minute.

Overview Documentation **API Console** Throttling Info

Try On Enviromant.

Authorization : Bearer 3aa11997caf103a5a44e395abb07a0

PhoneVerification

↗ Swagger Resource Listing (/api-docs)

checkphonenumber :		Show/Hide	List Operations	Expand Operations	Raw
<input type="button" value="GET"/>	/CheckPhoneNumber				
<input type="button" value="POST"/>	/CheckPhoneNumber				
<input type="button" value="OPTIONS"/>	/CheckPhoneNumber				

2. Expand the GET method of the resource CheckPhoneNumber. Note the parameters that you added in this step now appearing with their descriptions in the console.

/CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	(required)	Give the phone number to be validated	query	string
LicenseKey	(required)	Give the license key. If you don't have any, enter 0	query	string

3. Give sample values to the PhoneNumber and LicenseKey and click Try it Out to invoke the API.

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

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	6507452169	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key. If you don't have any, enter 0	query	string

Try it out!

4. Note the response for the API invocation. As we used a valid phone number in this example, the response is valid.

Response Body

```

<?xml version="1.0" encoding="utf-8"?>

<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query">
  <Company>TELEPORT COM SANFRAN</Company>

  <Valid>true</Valid>

  <Use>Assigned to a code holder for normal use.</Use>
  <State>CA</State>

  <RC>SSNFRNCSCO</RC>

  <OCN>7145</OCN>
  <OriginalNumber>6507452169</OriginalNumber>
  <CleanNumber>6507452169</CleanNumber>
  <SwitchName>SNFCCAFJDS0</SwitchName>
  <SwitchType>WECO 5ESS (Digital)</SwitchType>

  <Country>United States</Country>

```

You have invoked an API using the API Console.

Monitoring APIs and viewing statistics

Both the API publisher and store provide several statistical dashboards. Some of them are as follows:

- Number of subscriptions per API (across all versions of an API)
- Number of API calls being made per API (across all versions of an API)

- The subscribers who did the last 10 API invocations and the APIs/versions they invoked
- Usage of an API and from which resource path (per API version)
- Number of times a user has accessed an API
- The number of API invocations that failed to reach the endpoint per API per user
- API usage per application
- Users who make the most API invocations, per application
- API usage from resource path, per application

 If you are on **Windows**, note the following:

- If you install JDK in Program Files in the Windows environment, avoid the space by using PROGRA~1 when specifying environment variables for JAVA_HOME and PATH. Else, the server throws an exception.
- Install Cygwin (<http://www.cygwin.com>.) WSO2 BAM analytics framework depends on Apache Hadoop, which requires Cygwin in order to run on Windows. Install at least the basic net (OpenSSH,tcp_wrapper packages) and security related Cygwin packages. After Cygwin installation, update the PATH variable with C:/cygwin/bin and restart BAM.

Steps below explain how to configure **WSO2 BAM 2.4.1** with the API Manager. Let's do the configurations first.

1. Do the following changes in <APIM_HOME>/repository/conf/api-manager.xml file:

- Enable API usage tracking by setting the <APIUsageTracking> element to true
- Set the Thrift port to 7614
- Uncomments and set the data source used for getting BAM statistics in <DataSourceName> element.
- Set <BAMServerURL> to tcp://<BAM host IP>:7614/ where <BAM host IP> is the machine IP address. Do not use localhost unless you're in a disconnected mode.

```

<APIUsageTracking>
    <!-- Enable/Disable the API usage tracker. -->
    <Enabled>true</Enabled>

    <PublisherClass>org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataBridgeDataPublisher</PublisherClass>
        <ThriftPort>7614</ThriftPort>
        <BAMServerURL>tcp://<BAM host IP>:7614/</BAMServerURL>
        <BAMUsername>admin</BAMUsername>
        <BAMPassword>admin</BAMPassword>
        <!-- JNDI name of the data source to be used for getting BAM statistics. This data source should
            be defined in the master-datasources.xml file in conf/datasources
            directory. -->
        <DataSourceName>jdbc/WSO2AM_STATS_DB</DataSourceName>
</APIUsageTracking>

```

2. Specify the datasource definition in <APIM_HOME>/repository/conf/datasources/master-datasources.xml file as follows.

```

<datasource>
    <name>WSO2AM_STATS_DB</name>
    <description>The datasource used for getting statistics to API
Manager</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_STATS_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <!-- JDBC URL to query the database -->

<url>jdbc:h2:<BAM_HOME>/repository/database/APIMGTSTATS_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>

```

Next, prepare BAM to collect and analyze statistics from API manager.

3. Download WSO2 BAM 2.4.1 or later from location: <http://wso2.com/products/business-activity-monitor>.
4. Change port offset of BAM to **3** by editing the file `<BAM_HOME>/repository/conf/carbon.xml` file (search for the offset node).

```
<Offset>3</Offset>
```

This increments all ports used by the server by 3, which means the BAM server will run on port 9446. Port offset is used to increment the default port by a given value. It avoids possible port conflicts when multiple WSO2 products run in same host.

5. Do the following changes in `<BAM_HOME>/repository/conf/datasources/bam_datasources.xml` file:
 - Copy/paste `WSO2_AMSTATS_DB` definition from API Manager's `master-datasources.xml` file. You edited it in step 2.
 - Replace the port of `WSO2BAM_CASSANDRA_DATASOURCE` in URL (`jdbc:cassandra://localhost:9163/EVENT_KS`). Note that localhost is used here; not the machine IP.



- Do not edit the `WSO2BAM_UTIL_DATASOURCE`, which is using the offset
- Cassandra is bound by default on localhost, unless you change the `data-bridge/data-bridge-config.xml` file

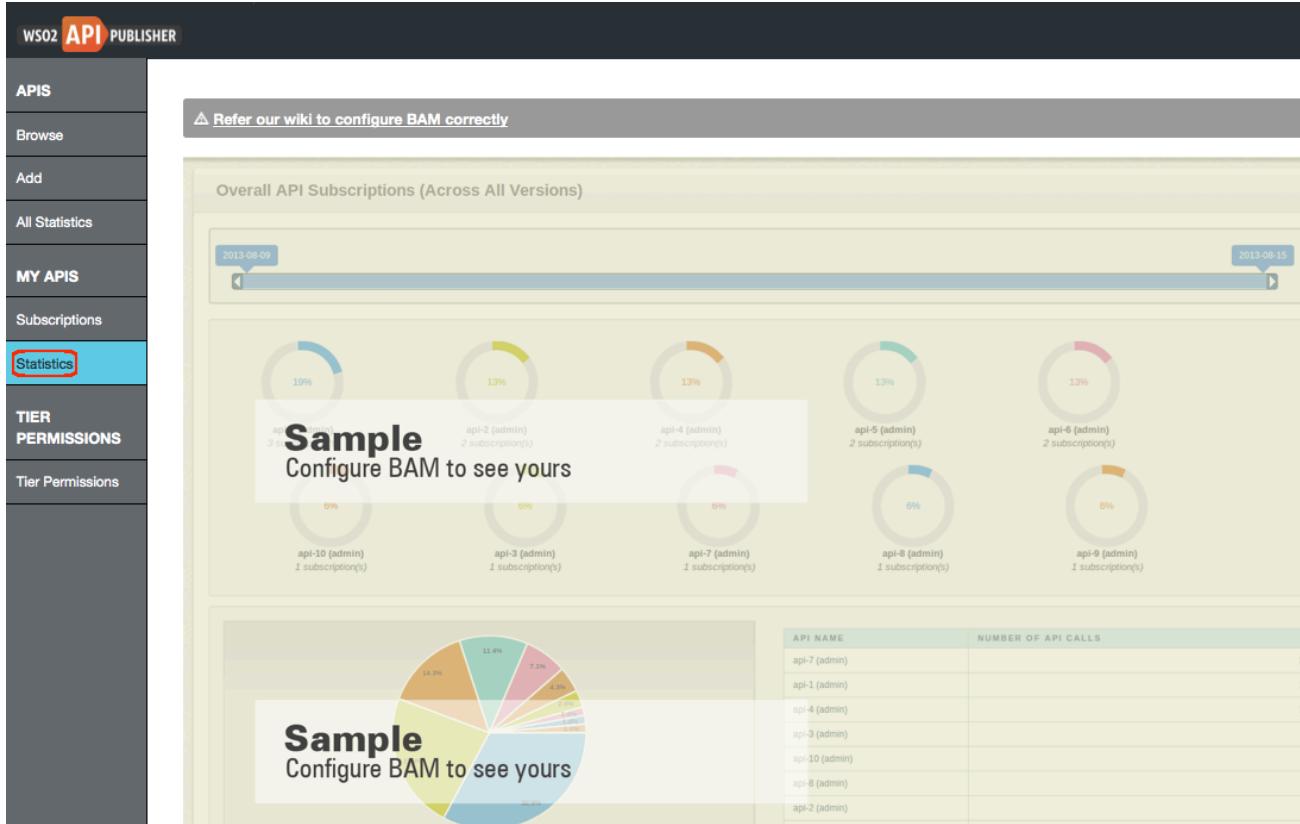
6. Copy the file `<APIM_HOME>/statistics/API_Manager_Analytics.tbox` to directory `<BAM_HOME>/repository/deployment/server/bam-toolbox`.

If this folder is not in the BAM installation directory by default, create it. The toolbox describes the information collected, how to analyze the data, as well as the location of the database where the analyzed data is stored.

7. Open `<BAM_HOME>/repository/conf/etc/hector-config.xml` file and change the port to `localhost:9163`. You must add the other nodes too when configuring a clustered setup.

```
<Nodes>localhost:9163</Nodes>
```

8. Restart the BAM server by running `<BAM_HOME>/bin/wso2server.[sh/bat]`. Let's see the statistics now.
9. Generate some traffic via the API Gateway (invoke the Cdyne API we use in this guide) and wait a few seconds.
10. Connect to the API Publisher as a creator or publisher. In the publisher role, you are able to see all stats and as creator, you see stats specific to the APIs you create.
11. Click the **Statistics** menu. We show the sample statistics here, but you will see graphs specific to your instance.



12. Similarly, API subscribers can also see statistics though the API Store. Click the **Statistics** menu as follows:

This concludes the API Manager quick start. You have set up the API Manager and gone through the basic usecases of the product. For more advanced usecases, please see the [User Guide](#) and the [Admin Guide](#) of the API Manager documentation.

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/api-manager>.
2. If you are a new user downloading WSO2 products for the first time, register and log in.
3. Once you are logged in, click the **Binary** button in the upper right corner of the page.

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.

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	<ul style="list-style-type: none"> • ~ 2 GB minimum • ~ 512 MB heap size. This is generally sufficient to process typical SOAP messages but the requirements vary with larger message sizes and the number of messages processed concurrently.
---------------	--

Disk

- ~ 500 MB for a fresh installation pack, excluding space allocated for log files and databases.

Environment compatibility

- 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. The embedded H2 database is suitable for development, testing, and some production environments. For most enterprise production environments, however, we recommend you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. For more information, see [Working with Databases](#). Additionally, we do not recommend the H2 database as a user store.
- It is **not recommended to use Apache DS** in a production environment due to scalability issues. Instead, use an LDAP like OpenLDAP for user management.
- For environments that WSO2 products are tested with, see [Compatibility of WSO2 Products](#).
- If you have difficulty in setting up any WSO2 product in a specific platform or database, please [contact us](#).

Required applications

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

Application	Purpose	Version	Download
Oracle Java S E Development Kit (JDK)*	<p>Required to,</p> <ul style="list-style-type: none"> • To launch the product as each product is a Java application. • To build the product from the source distribution (both JDK and Apache Maven are required). • To run Apache Ant. 	<p>1.6.27 or later / 1.7.*</p> <ul style="list-style-type: none"> • If you are using JDK 1.6, you might need to replace the Java Cryptography Extension (JCE) policy files in your JDK with the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy files. This will avoid "illegal key size" errors when you try to invoke a secured Web service. • To build the product from the source distribution, you must use JDK 1.6 instead of JDK 1.7. • Oracle and IBM JRE 1.7 are also supported when running (not building) WSO2 products. • If you are using JDK 1.7 on Mac OS or Solaris, install the snappy-java library using the following steps: <ol style="list-style-type: none"> 1. Download the snappy-java JAR and extract it to a preferred location. This folder will be referred to as <SNAPPY_HOME>. 2. Copy the appropriate snappy-java library file i386.jnilib (32bit) or x86_64.jnilib (64bit), which is in the <SNAPPY_HOME>/org/xerial/snappy/native/Mac/directory, to the <APIM_HOME> directory. For more information on installing snappy-java library, see Snappy-java fails on Mac OS JDK 1.7. • We do not recommend OpenJDK. 	http://java.oracle.com/

Apache ActiveMQ JMS Provider	<ul style="list-style-type: none"> To enable the product's JMS transport and try out JMS samples. The ActiveMQ client libraries must be installed in the product's classpath before you can enable the JMS transport. 	5.5.0 If you use any other JMS provider (e.g., Apache Qpid), install any necessary libraries and/or components.	or	later	http://activemq.apache.org
Apache Ant	<ul style="list-style-type: none"> To compile and run the product samples. 	1.7.0 or later			http://ant.apache.org
SVN Client	<ul style="list-style-type: none"> To check out the code to build the product from the source distribution. If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do not need to install SVN. 				<ul style="list-style-type: none"> Linux - ges.htm Windows - .html

Apache Maven	<ul style="list-style-type: none"> To build the product from the source distribution (both JDK and Apache Maven are required). If you are installing by downloading and extracting the binary distribution instead of building from the source code, you do not need to install Maven. 	3.0.*	http://maven.apache.org
Web Browser	<ul style="list-style-type: none"> Required by all WSO2 products to access each product's Management Console. The Web Browser must be JavaScript enabled to take full advantage of the Management console. <p>NOTE: On Windows Server 2003, you must not go below the medium security level in Internet Explorer 6.x.</p>		

You are now ready to install. Click one of the following links for instructions:

- [Installing on Linux or OS X](#)
- [Installing on Solaris](#)
- [Installing on Windows](#)
- [Installing as a Linux Service](#)

Installing the Product

Installing WSO2 is very fast and easy. Before you begin, be sure you have met the installation prerequisites, and

then follow the installation instructions for your platform. WSO2 also provides pre-configured packages for automated installation based on Puppet or similar solutions. For information, [contact team WSO2](#).

- [Installing on Linux or OS X](#)
- [Installing on Solaris](#)
- [Installing on Windows](#)
- [Installing as a Linux Service](#)
- [Installing as a Windows Service](#)

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 API Manager on Linux or Mac OS X.

Installing the required applications

1. Log in to the command line (Terminal on Mac) either as root or obtain root permissions after logging in via `su` or `sudo` command.

[Installation Prerequisites](#). Ensure that your system meets the Java Development Kit (JDK) is essential to run the product.

Installing the API Manager

[Download the Product](#) Download the latest version of the API Manager as described in .

2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API_HOME>`.

Setting up JAVA_HOME

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

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

1. In your home directory, open the `BASHRC` file (`.bash_profile` file on Mac) using editors such as `vi`, `emacs`, `pico`, or `mcedit`.
2. Assuming you have `JDK 1.6.0_25` in your system, 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.

On Linux:

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

On OS X:

```
export JAVA_HOME=/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
```

3. Save the file.

 If you do not know how to work with text editors in a Linux SSH session, run the following command:
`cat >> .bashrc`. Paste the string from the clipboard and press "Ctrl+D."

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

```
On Linux:
echo $JAVA_HOME
```

```
On OS X:
which java
```

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

5. 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](#).

i 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 Linux or OS X.

Follow the instructions below to install API Manager on Solaris.

Installing the required applications

1. Establish a SSH connection to the Solaris machine or log in on the text console. You should either log in as root or obtain root permissions after login via `su` or `sudo` command.

[Installation Prerequisites](#) Be sure your system meets the . Java Development Kit (JDK) is essential to run the product.

Installing the API Manager

[Downloading the Product](#) Download the latest version of the API Manager as described in .

2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as <APIM_HOME>.

Setting up JAVA_HOME

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

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

1. In your home directory, open the BASHRC file in your favorite text editor, such as vi, emacs, pico, or mcedit.
2. Assuming you have JDK 1.6.0_25 in your system, 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.

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

The file should now look like this:

```
# .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-3.0.3
export PATH=${M2_HOME}/bin:${PATH}
```

3. Save the file.

If you do not know how to work with text editors in an SSH session, run the following command: `cat >> .bashrc`
Paste the string from the clipboard and press "Ctrl+D."

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

```
echo $JAVA_HOME
```

```
[suncoma@wso2 ~]$ echo $JAVA_HOME
/usr/java/jdk1.6.0_25
[suncoma@wso2 ~]$
```

5. 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 API Manager on Windows.

Installing the required applications

Installation Prerequisites Be sure your system meets the. Java Development Kit (JDK) is essential to run the product.

2. Be sure that the `PATH` environment variable is set to "C:\Windows\System32", because the `findstr` window's exe is stored in this path.

Installing the API Manager

Downloading the Product Download the latest version of the API Manager as described in.

2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API_HOME>`.

Setting up JAVA_HOME

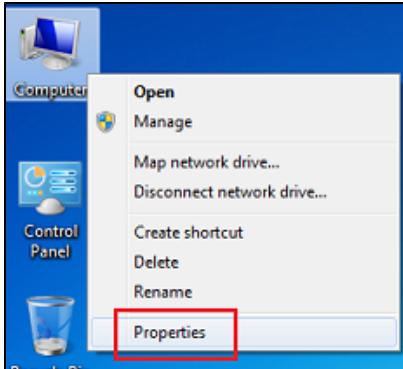
You must set your `JAVA_HOME` environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under `C:/Program Files/Java`, such as `C:/Program Files/Java/jdk1.6.0_27`. If you have multiple versions installed, choose the latest one, which you can find by sorting by date.

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

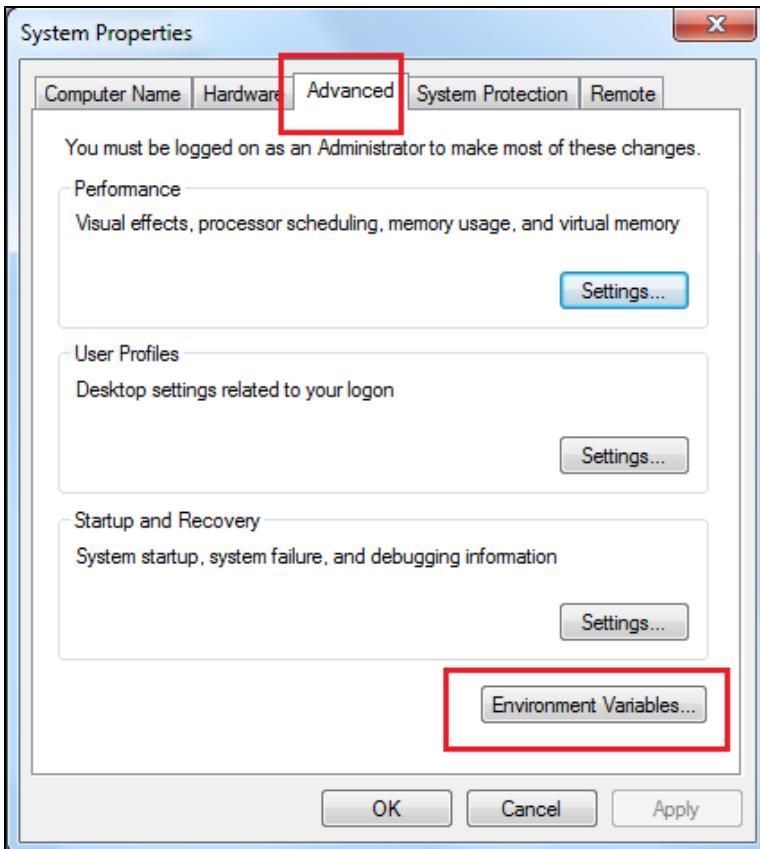
You set up `JAVA_HOME` using the System Properties, as described below. Alternatively, if you just want to set `JAVA_HOME` temporarily for the current command prompt window, set it at the command prompt.

Setting up JAVA_HOME using the system properties

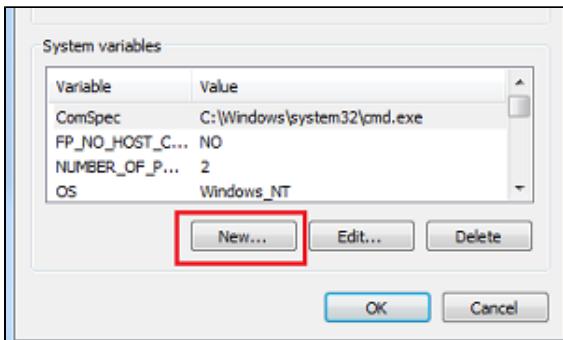
1. Right-click the **My Computer** icon on the desktop and choose **Properties**.



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

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

```
set JAVA_HOME
```

The system returns the JDK installation path. You are now ready to [run the product](#).

Setting JAVA_HOME temporarily using the Windows command prompt (CMD)

You can temporarily set the JAVA_HOME environment variable within a Windows command prompt window (CMD). This is useful when you have an existing 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
```

3. The system returns the JDK installation path.

Setting system properties

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

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

You are now ready to [run the product](#).

Installing as a Linux Service

Follow the sections below to run a WSO2 product as a Linux service:

- [Prerequisites](#)
- [Setting up CARBON_HOME](#)
- [Running the product as a Linux service](#)

Prerequisites

Install JDK 1.6.24 or later or 1.7.* and set up the JAVA_HOME environment variable.

Setting up CARBON_HOME

Extract the WSO2 product to a preferred directory in your machine and set the environment variable CARBON_HOME to the extracted directory location.

Running the product as a Linux service

1. To run the product as a service, create a startup script and add it to the boot sequence. The basic structure of the startup script has three parts (i.e., start, stop and restart) as follows:

```
#!/bin/bash

case "$1" in
start)
    echo "Starting the Service"
;;
stop)
    echo "Stopping the Service"
;;
restart)
    echo "Restarting the Service"
;;
*)
    echo $"Usage: $0 {start|stop|restart}"
exit 1
esac
```

Given below is a sample startup script. <PRODUCT_HOME> can vary depending on the WSO2 product's directory.

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

startcmd='<PRODUCT_HOME>/bin/wso2server.sh start > /dev/null &'
restartcmd='<PRODUCT_HOME>/bin/wso2server.sh restart > /dev/null &'
stopcmd='<PRODUCT_HOME>/bin/wso2server.sh stop > /dev/null &'

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

In the above script, the server is started as a user by the name user1 rather than the root user. For example,
`su -c "${startcmd}" user1`

2. Add the script to /etc/init.d/ directory.

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

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

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

```
sudo update-rc.d prodserver defaults
```

The `defaults` option in the above command makes the service to start in runlevels 2,3,4 and 5 and to stop in runlevels 0,1 and 6.

A **runlevel** is a mode of operation in Linux (or any Unix-style operating system). There are several runlevels in a Linux server and each of these runlevels is represented by a single digit integer. Each runlevel designates a different system configuration and allows access to a different combination of processes.

4. You can now start, stop and restart the server using `service <service name> {start|stop|restart}` command. You will be prompted for the password of the user (or root) who was used to start the service.

Installing as a Windows Service

WSO2 Carbon and any 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 1.6.24 or later or 1.7.* 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](#).

Minimal wrapper.conf configuration

```
#####
# working directory
#####
wrapper.working.dir=${carbon_home}\
# Java Main class.
```

```

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

```

```
*****
# Trigger actions on console output
*****
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\\trayMessage.gv
wrapper.filter.script.0.args=Exception
*****
# genConfig: further Properties generated by genConfig
*****
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\\bin\\java
wrapper.java.classpath.1 = ${java_home}\\lib\\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\\bin\\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 = -Xbootclasspath\\a:${carbon_home}\\lib\\xboot\\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError
wrapper.java.additional.6 =
-XX:HeapDumpPath=${carbon_home}\\repository\\logs\\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 =
-Djava.endorsed.dirs=${carbon_home}\\lib\\endorsed;${java_home}\\jre\\lib\\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=\
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\\bin\\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\\tmp
wrapper.java.additional.14 = -Dcatalina.base=${carbon_home}\\lib\\tomcat
wrapper.java.additional.15 =
-Djava.util.logging.config.file=${carbon_home}\\repository\\conf\\log4j.properties
wrapper.java.additional.16 = -Dcarbon.config.dir.path=${carbon_home}\\repository\\conf

wrapper.java.additional.17 = -Dcarbon.logs.path=${carbon_home}\\repository\\logs
wrapper.java.additional.18 =
-Dcomponents.repo=${carbon_home}\\repository\\components\\plugins
wrapper.java.additional.19 = -Dconf.location=${carbon_home}\\repository\\conf
wrapper.java.additional.20 =
-Dcom.atomikos.icatch.file=${carbon_home}\\lib\\transactions.properties
wrapper.java.additional.21 = -Dcom.atomikos.icatch.hide_init_file_path=true
wrapper.java.additional.22 =
-Dorg.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true
```

```
wrapper.java.additional.23 = -Dcom.sun.jndi.ldap.connect.pool.authentication=simple
wrapper.java.additional.24 = -Dcom.sun.jndi.ldap.connect.pool.timeout=3000
wrapper.java.additional.25 = -Dorg.terracotta.quartz.skipUpdateCheck=true
```

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>.

```
C:\Documents and Settings\yajsw_home\bat>runConsole.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -c "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 11:12:27 AM org.apache.commons.vfs2.VfsLog info
INFO: Using "C:\DOCUMENTS\ADMINI\LOCALS\Temp\ufs_cache" as temporary files store.
```

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.

```
C:\Documents and Settings\yajsw_home\bat>installService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -i "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 12:51:42 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\1\ADMINI\1\LOCALS\1\Temp\ vfs_cache" as temporary files store.
platform null
***** INSTALLING "WSO2CARBON" *****
Service "WSO2CARBON" installed
Press any key to continue . . .
```

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

```
startService.bat
```

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

```
C:\Documents and Settings\yajsw_home\bat>startService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\..\tmp" -jar "C:\Documents and Settings\yajsw_home\bat\..\wrapper.jar" -t "C:\Documents and Settings\yajsw_home\bat\..\conf\wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 1:09:00 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\1\ADMINI\1\LOCALS\1\Temp\ vfs_cache" as temporary files store.
platform null
***** STARTING "WSO2CARBON" *****
Service "WSO2CARBON" started
Press any key to continue . . .
C:\Documents and Settings\yajsw_home\bat>
```

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

```
stopService.bat
```

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

```
C:\Documents and Settings\yajsw_home\bat>stopService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\../tmp" -jar "C:\Documents and Settings\yajsw_home\bat\../wrapper.jar" -p "C:\Documents and Settings\yajsw_home\bat\../conf/wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 11:11:31 AM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\1\ADMINI\1\LOCALS\1\Temp\ufs_cache" as temporary files store.
platform null
***** STOPPING "WSO2CARBON" *****
Service "WSO2CARBON" stopped
Press any key to continue . . .
```

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

```
uninstallService.bat
```

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

```
C:\Documents and Settings\yajsw_home\bat>uninstallService.bat
C:\Documents and Settings\yajsw_home\bat>cd C:\Documents and Settings\yajsw_home\bat\
C:\Documents and Settings\yajsw_home\bat>call setenv.bat
"java" -Xmx30m -Djna_tmpdir="C:\Documents and Settings\yajsw_home\bat\../tmp" -jar "C:\Documents and Settings\yajsw_home\bat\../wrapper.jar" -r "C:\Documents and Settings\yajsw_home\bat\../conf/wrapper.conf"
YAJSW: yajsw-stable-11.03
OS : Windows XP/5.2/amd64
JVM : Oracle Corporation/1.7.0_06
Dec 30, 2012 1:19:14 PM org.apache.commons vfs2.UfsLog info
INFO: Using "C:\DOCUMENT\1\ADMINI\1\LOCALS\1\Temp\ufs_cache" as temporary files store.
platform null
***** REMOVING "WSO2CARBON" *****
Service "WSO2CARBON" removed
Press any key to continue . . .

C:\Documents and Settings\yajsw_home\bat>
```

Building from Source

WSO2 invites you to contribute by checking out the source from the Subversion (SVN) source control system, building the product and making changes, and then committing your changes back to the source repository. (For more information on Subversion, see <http://svnbook.red-bean.com>.) The following sections describe this process:

- Checking out the source
- Setting up your development environment
- Building the product
- Committing your changes

i 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.

Checking out 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 4.2.0).

A WSO2 platform release is a set of WSO2 products based on the same Carbon release. For example, Turing is the platform release name for WSO2 Carbon 4.2.0 and the WSO2 products that are based on it. Usually, not all products in a platform get released at the same time, so they are released in **chunks**, each of which contains the Carbon release and a subset of products. For example, the API Manager 1.8.0 comes in chunk 14 of the Turing platform.

Checking out the patches

Before checking out the product source, you need to checkout the patches related to the Carbon chunk using the following command.

```
$ svn checkout https://svn.wso2.org/repos/wso2/carbon/kernel/branches/4.2.0 <local-platform-directory-1>
```

 Replace <local-platform-directory-1> with a meaningful name, such as wso2carbon-platform.

Downloading the product source

For products based on WSO2 Carbon 4.2.0, use the below command to download the product source:

```
$ svn checkout https://svn.wso2.org/repos/wso2/carbon/platform/tags/turing-<release-chunk>/<local-platform-directory-2>
```

Replace <release-chunk> with the release chunk, on which the specific product version is based on. To find out the respective release chunk, see the [Release Matrix](#). For example, for products based on Chunk 14 of WSO2 Carbon 4.2.0, the command is as follows:

```
$ svn checkout https://svn.wso2.org/repos/wso2/carbon/platform/tags/turing-chunk14/<local-platform-directory-2>
```

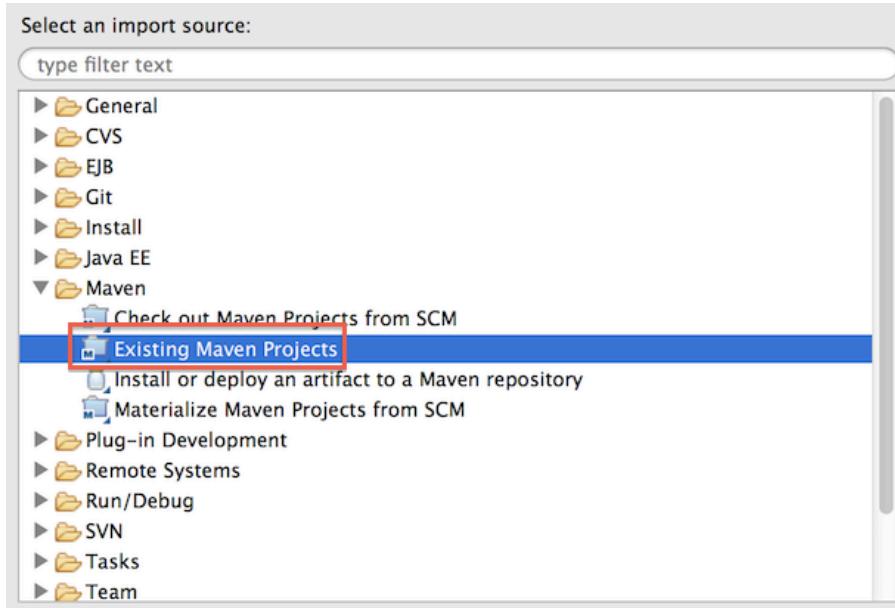
Setting up your development environment

Before you edit the source code in your IDE, set up your development environment by running one of the following commands:

If you are using this IDE...	Run this command...	Additional information
Eclipse	mvn eclipse:eclipse	http://maven.apache.org/plugins/maven-eclipse-plugin
IntelliJ IDEA	mvn idea:idea	http://maven.apache.org/plugins/maven-idea-plugin

If you are using a later Eclipse version and if you get errors (library path etc.) when trying to import the source code using the **Existing Projects into Workspace**, follow the steps below to solve them by importing the source code as a Maven project.

1. Build the source using the command: `mvn clean install`
2. Open Eclipse and click **Import** in the **File** menu and then click **Existing Maven Projects** as shown below:



Building the product

Follow the instructions below to build the product after editing the source code:

i Make sure the build server has an active Internet connection to download dependencies while building.

1. Install Maven and JDK. See [Installation Prerequisites](#) for compatible versions.
2. Set the environment variable `MAVEN_OPTS="-Xms1024m -Xmx4096m -XX:MaxPermSize=1024m"` to avoid the Maven `OutOfMemoryError`.
3. Navigate to each folder representing the patches within the `<local-platform-directory-1>` and run the following [Apache Maven](#) commands to build the patches. For information on the patches, which are applicable for the respective Carbon chunk release, go to [Release Matrix](#).

This command...	Creates...
<code>mvn clean install</code>	The binary and source distributions of the chunk release.
<code>mvn clean install -Dmaven.test.skip=true</code>	The binary and source distributions, without running any of the unit tests.
<code>mvn clean install -Dmaven.test.skip=true -o</code>	The binary and source distributions, without running any of the unit tests, in offline mode. This can be done only if you've already built the source at least once.

4. For products based on Carbon 4.2.0, to create complete release artifacts of the products released with this chunk version, including the binary and source distributions, go to `<local-platform-directory-2>/product-releases/ <release-chunk>/` directory and run the Apache Maven commands stated in the above step. **To build only a selected product/s**, open `<local-platform-directory-2>/product-releases/<release-chunk>/products/pom.xml` file, and comment out the products you do not want to build and run the relevant Maven command.

i After [building the source](#), you can find the artifacts/product binary distribution package of the product in the directory : `<local-platform-directory-2>/products/<product_name>/<product_release_version>/modules/distribution/target/` directory.

Committing your changes

If you are a committer, you can commit your changes using the following command (SVN will prompt you for your password):

```
$ svn commit --username your-username -m "A message"
```

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:

- Starting the server
- Running the management console
- Stopping the server

i **Before you begin**, note that the Management Console uses the default HTTP-NIO transport , which is configured in the `catalina-server.xml` file in the `<APIM_HOME>/repository/conf/tomcat` directory. This transport must be properly configured in this file for the Management Console to be accessible.

Starting the server

To start the server, you run the script `wso2server.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](#) .

i 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 `<APIM_HOME>` is the directory where you installed the product distribution:
 - On Windows: `<APIM_HOME>/bin/wso2server.bat --run`
 - On Linux/Solaris: `sh <APIM_HOME>/bin/wso2server.sh`

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

Running 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:

```
[2014-12-04 17:53:26,547] INFO {org.wso2.carbon.core.internal.StartupFinalizerServiceComponent} - WSO2 Carbon started in 45 sec
[2014-12-04 17:53:26,787] INFO {org.wso2.carbon.ui.internal.CarbonUIServiceComponent} - Mgt Console URL : https://localhost: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>.

 **Tip:** The Management Console URL can be changed by modifying the value of the `MgtHostName` in the `<PRODUCT_HOME>/repository/conf/carbon.xml` file.

```
<MgtHostName>localhost</MgtHostName>
```

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 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 `<APIM_HOME>/repository/conf/tomcat/carbon/WEB-INF/web.xml` file as follows:

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

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 the Previous Release

The following information describes how to upgrade your API Manager server from the previous release, which is APIM 1.7.0. To upgrade from a version older than 1.7.0, start from the doc that was released immediately after your

current release and upgrade incrementally.

- [Upgrading the product databases](#)
- [Migrating the configurations](#)
- [Upgrading APIM 1.7.0 to 1.8.0](#)

Migration scripts' location is https://svn.wso2.org/repos/wso2/carbon/platform/branches/turing/products/apimgt/1.8.0/modules/distribution/resources/migration-1.7.0_to_1.8.0.

Upgrading the product databases

1. Download the API Manager 1.8.0 from <http://wso2.com/products/api-manager>.
2. Stop all running API Manager server instances.
3. Back up the databases of your API Manager 1.7.0 server instance.
4. Download the migration scripts from the migration script location and execute the database upgrade scripts on your old database. You must select the script corresponding to your database type. For example, if your database is MySQL, execute `migration-1.7.0_to_1.8.0/mysql.sql` on it. The script adds all the schema changes done to API Manager tables in the 1.8.0 release.

Migrating the configurations

In this section, you move all existing API Manager configurations from the current environment to the new one.

1. Open `<APIM_1.8.0_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases. You can copy the configurations from the same file in the API Manager 1.7.0 instance.
 - User Store
 - Registry database
 - API Manager Databases
2. Edit the registry configurations in the `<APIM_HOME>/repository/config/registry.xml` and the user database in the `<APIM_HOME>/repository/conf/user-mgt.xml` file.
3. Move all your synapse configurations by copying and replacing `<APIM_1.7.0_HOME>/repository/deployment/server/synapse-config/default` directory to `<APIM_1.8.0_HOME>/repository/deployment/server/synapse-config/default` directory.



If you changed the default URLs in `AuthorizeAPI.xml` and `TokenAPI.xml` files, do not replace them when copying. They are application-specific APIs.

Upgrading APIM 1.7.0 to 1.8.0

1. Start the API Manager 1.8.0 and log in to its management console.
2. Copy the 'swagger-doc-migration' directory from the migration scripts location to `<APIM_1.8.0_HOME>`. The new directory path will now be `<APIM_1.8.0_HOME>/swagger-doc-migration`.
3. Configure `<APIM_1.8.0_HOME>/swagger-resource-migration/build.xml` file with the following properties:

Property	Description
registry.home	Path to the APIM distribution. In a distributed setup, give the API Publisher node's path.
username	Username for the server. For a tenant to log in, provide the tenant admin username.
password	Password for the server. For a tenant to log in, provide the tenant admin password.
host	IP of the running APIM server. In a distributed setup, give the host of the API Publisher node.
port	Port of the running APIM server. In a distributed setup, give the port of the APIM Publisher node.

version	Version of the server.
----------------	------------------------

4. Using the command line, go to <APIM_1.8.0_HOME>/swagger-resource-migration folder and execute **ant run**. If the above configuration is successful, you get a **BUILD SUCCESSFUL** message. It modifies the structure of Swagger content in the registry.
5. To re-index the artifacts in the registry, perform the two steps given below.
 - a) Rename the lastAccessTimeLocation in the <APIM_1.8.0_HOME>/repository/conf/registry.xml file.
Eg: Change `/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime` to `/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime_1`
 - b) Shutdown AM 1.8.0 and backup and delete the <APIM_1.8.0_HOME>/repository/conf/solr directory and restart the server.

Upgrading tenants

6. If you have **multiple tenants** added to your API Manager instance, follow the steps below to migrate tenant configurations:
 - a. Copy the contents from your previous <APIM_HOME>/repository/tenants directory to the same directory in the API Manager 1.8.0. Do not replace the `_TokenAPI_.xml`, `_RevokeAPI_.xml` and `_AuthorizeAPI_.xml` files in the `/default/api` sub directory.
 - b. Start the server
 - c. Execute steps 3 and 4 for all tenants in your system.
 - d. Execute steps 7 to 9 for all tenants in your system.

Upgrading external stores

7. If you have **external stores** configured in the registry, follow the steps below:
 - a. Log in to APIM 1.8.0 management console and click the **Resources -> Browse** menu.
 - b. Load the `/_system/governance/apimgt/externalstores/external-api-stores.xml` resource in the registry browser UI, [configure your external stores](#) and save.

Upgrading Google analytics

8. If you have **Google Analytics** configured in the registry, follow the steps below:
 - a. Log in to APIM 1.8.0 management console and go to **Resources -> Browse** menu.
 - b. Load the `/_system/governance/apimgt/statistics/ga-config.xml` resource in the registry browser UI, [configure the Google analytics](#) and save.

Upgrading workflows

9. If you have **Workflows** configured in the registry, follow the steps below:
 - a. Log in to APIM 1.8.0 management console and go to **Resources -> Browse** menu.
 - b. Load the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource in the registry browser UI, [configure your workflows](#) and save.

Get Involved

All 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](#). 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](#). For tutorials, articles, white papers, webinars, WSO2 documentation, and other learning resources, look in the Resources menu on the [WSO2 website](#).

- **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 and 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](#).

- **Participate in user forums**

The WSO2 team monitors and participates in the discussions on [Stack Overflow](#). 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 a 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.

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

4. If you need to add a new file to the repository:

- Add the new file.

```
git add <FILE-NAME>
```

For example:

```
git add mycode.java
```

- 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:

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

- 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

The following are the WSO2 GitHub repositories that need to be forked, so that you can contribute to the WSO2 community by offering patches for bug fixes or features for WSO2 products.

- Kernel level Git repositories
- Platform level Git repositories
- Mobile platform Git repositories
- Product level Git repositories
- Other WSO2 Git repositories

Kernel level Git repositories

Repo URL	Description
carbon-kernel	Carbon 5 kernel repo
carbon4-kernel	Carbon 4 kernel repo

Platform level Git repositories

Repo URL	Description
carbon-analytics	Contains components and features related to analytics services.
carbon-apimgt	Contains components and features related to API management.
carbon-appmgt	Contains components and features related to application management.
carbon-business-messaging	Contains the components and features related to business messaging.
carbon-business-process	Contains components and features related to business processes.
carbon-commons	Contains common components and features shared across the platform projects.
carbon-data	Contains components and features related to data services.
carbon-deployment	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.).
carbon-event-processing	Contains components and features related to event processing services.
carbon-governance	Contains components and features related to governance services.

carbon-mediation	Contains components and features related to mediation services.
carbon-multitenancy	
carbon-parent	
carbon-platform-automated-test-suite	Contains WSO2 product integration test suites and Platform test suites with ant based test executor.
carbon-platform-integration	Contains WSO2 test automation framework modules.
carbon-platform-integration-utils	Contains utilities related to WSO2 test automation framework which is common to the whole product platform.
carbon-qos	Contains components and features related to quality of service.
carbon-registry	Contains components and features related to registry services.
carbon-rules	Contains components and features related to business rules.
carbon-storage-management	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.
carbon-utils	Contains ntask, remote-tasks, ndatasource etc.

Mobile platform Git repositories

Repo URL	Description
emm-agent-android	Maintains the Android agent that is used to enroll the device to EMM server.
emm-agent-ios	Maintains the iOS agent that is used to enroll the device to EMM server.

Product level Git repositories

Product Name	Repo URL	Description
API Manager	product-apim	Maintains sources corresponding to building and packaging of WSO2 API manager distribution.
App Factory	product-af	Maintains sources corresponding to building and packaging of WSO2 APP Factory distribution.
Application Server	product-as	Maintains sources corresponding to building and packaging of WSO2 Application Server distribution.
Business Activity Monitor	product-bam	Maintains sources corresponding to building and packaging of WSO2 Business Activity Monitor distribution.
Business Process Server	product-bps	Maintains sources corresponding to building and packaging of WSO2 Business Process Server distribution.
Business Rules Server	product-brs	Maintains sources corresponding to building and packaging of WSO2 Business Rules Server distribution.
Complex Event Processor	product-cep	Maintains sources corresponding to building and packaging of WSO2 Complex Event Processor distribution.

Data Services Server	product-dss	Maintains sources corresponding to building and packaging of WSO2 Data Services Server distribution.
Enterprise Mobility Manager	product-emm	Maintains sources corresponding to building and packaging of WSO2 Enterprise Mobility Manager distribution.
Enterprise Service Bus	product-esb	Maintains sources corresponding to building and packaging of WSO2 Enterprise Service Bus distribution.
Enterprise Store	product-es	Maintains sources corresponding to building and packaging of WSO2 Enterprise Store distribution.
Governance Registry	product-greg	Maintains sources corresponding to building and packaging of WSO2 Governance Registry distribution.
Identity Server	product-identity	Maintains sources corresponding to building and packaging of WSO2 Identity Server distribution.
Message Broker	product-mb	Maintains sources corresponding to building and packaging of WSO2 Message Broker distribution.
Private PaaS	product-paas	Maintains sources corresponding to building and packaging of WSO2 Private PaaS distribution.
Storage Server	product-ss	Maintains sources corresponding to building and packaging of WSO2 Storage Server distribution.
Task Server	product-ts	Maintains sources corresponding to building and packaging of WSO2 Task Server distribution.
Developer Studio	developer-studio	Maintains sources corresponding to building and packaging of WSO2 Developer Studio distribution.

Other WSO2 Git repositories

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

Repo URL	Description
andes	Message broker core engine implementation.
balana	XACML core engine implementation.
charon	SCIM core engine implementation.
esb-connectors	Collection of connectors that allows you to interact with WSO2 ESB's third-party product function.
jaggery	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.
jaggery-extensions	This contains extensions for the Jaggery framework.
orbit	Used to create OSGi bungles out of third-part dependencies.
siddhi	Complex event processing core engine implementation.

User Guide

The user guide provides information about the features, functionality, solution development, testing and debugging options of WSO2 API Manager.

- [Key Concepts](#)
- [API Developer Tutorials](#)
- [Application Developer Tutorials](#)
- [Configuring the API Manager](#)
- [Extending the API Manager](#)
- [Working with Security](#)
- [Working with Statistics](#)

Key Concepts

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

[[API Manager components](#)] [[Users and roles](#)] [[API lifecycle](#)] [[Applications](#)] [[Access tokens](#)] [[Throttling tiers](#)] [[API visibility and subscription](#)] [[API documentation visibility](#)] [[API resources](#)] [[Cross-origin resource sharing](#)] [[OAuth scopes](#)] [[API templates](#)] [[Endpoints](#)] [[Sequences](#)] [[Caching](#)]

API Manager components

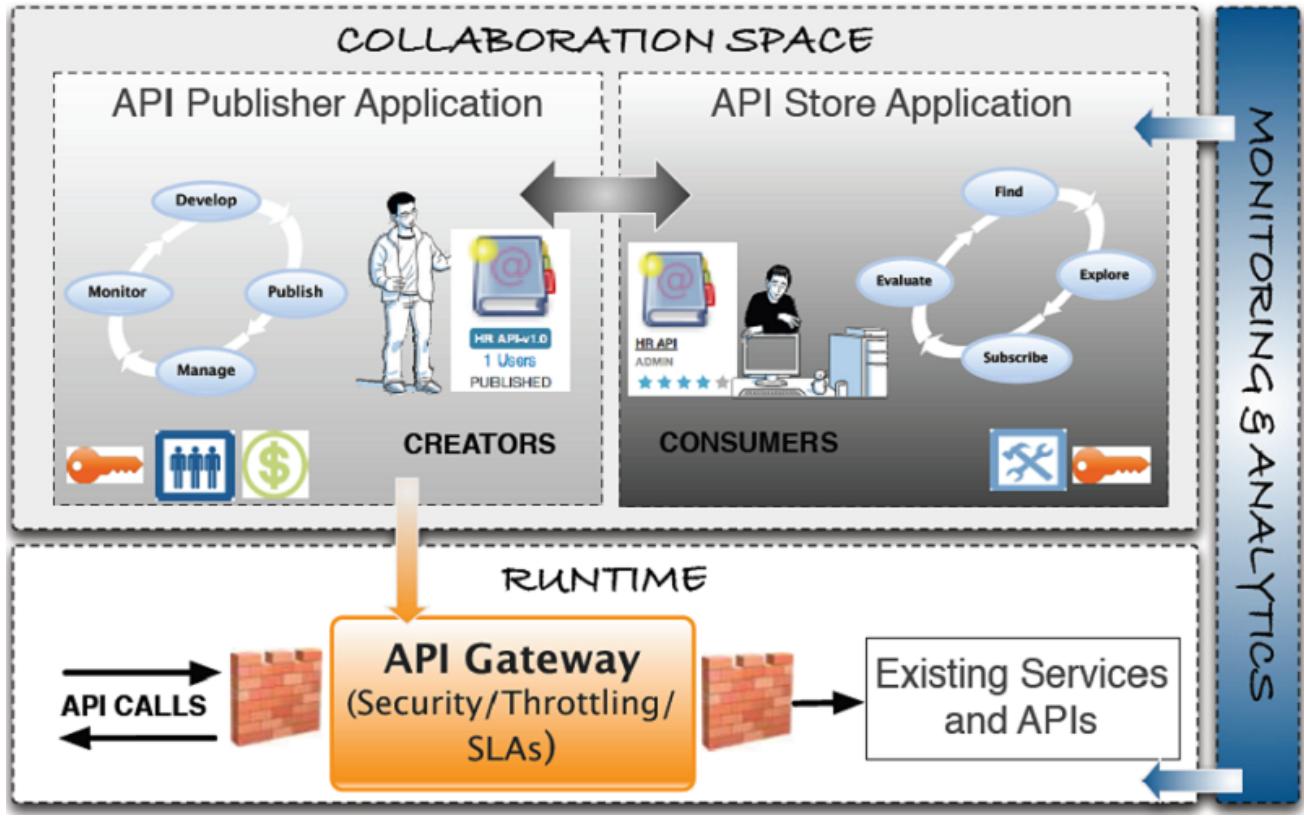
A component is made up of one or more [OSGi](#) bundles. A bundle is the modularization unit in OSGi, similar to a JAR file in Java. The component-based architecture of all WSO2 products gives developers flexibility to remove or add features with minimum dependencies.

The API Manager comprises the following high-level components:

Component	Description
API Publisher	Provides an end user, collaborative Web interface for API providers to publish APIs, share documentation keys, and gather feedback on API features, quality and usage. For API Publisher use cases, see API Publisher Use Cases .
API Store	Provides an end-user, collaborative Web interface for API consumers to self register, discover API functionality, evaluate them and interact with API publishers. For API Store use cases, see Application Development Use Cases .
API Gateway	<p>A runtime, back end component (an API proxy) developed using WSO2 ESB. API Gateway secures and scales API calls. It intercepts API requests, applies policies such as throttling and security, and manages API statistics. Upon validation of a policy, the Gateway passes Web service calls to the back end service. If the service call is a token request, the Gateway passes it directly to the token provider.</p> <p>When the API Manager is running, you can access the Gateway using the URL https://localhost:9443. You can also integrate a monitoring and statistics component to the API Manager without any additional configuration. The monitoring component integrates with WSO2 Business Activity Monitor, which can be deployed as a separate application. For more information, see Publishing API Runtime Statistics.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i Although the API Gateway contains ESB features, it is recommended not to use it for ESB-related tasks, such as message mediation. Instead, use the API Manager's mediation feature or a separate ESB cluster for that. </div>

Key Manager	<p>Handles all security and key-related operations. The Gateway connects with the key manager to check OAuth tokens when APIs are invoked. The key manager also provides a token API to generate OAuth tokens via the Gateway.</p> <p>All tokens used for validation are based on OAuth 2.0.0 protocol. Secure authorization of APIs is provided by the key management standard. The API Gateway supports API authentication with OAuth 2.0, allowing organizations to enforce rate limits and throttling policies.</p> <p>When the Gateway receives API invocation calls, it similarly contacts the Key Manager service for verification. If caching is not enabled at the Gateway level, this verification call happens every time the Gateway receives an API invocation call.</p> <ul style="list-style-type: none"> • Through a Web service call • Through a Thrift call (Thrift is the default communication protocol and is much faster than SOAP) <p>If your setup has a cluster of multiple Key Manager nodes that are fronted by a WSO2 ELB instance, change the key management protocol from Thrift to WSClient using the <code><KeyValidatorClientType> M_HOME>/repository/conf/api-manager.xml</code> file. Thrift uses TCP load balancing and the ELB</p>
Handlers	<p>You find the default handlers in any API's Synapse definition as shown below. When an API is created, a file with its synapse configuration is added to the API Gateway. You can find it at <code><repository>/deployment/server/synapse-configs/default/api</code>. It has a set of handlers which is executed on the APIs in the same order they appear in the configuration.</p> <pre> <handlers> <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/> <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler"/> <property name="id" value="A"/> <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/> </handler> <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/> <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/> <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/> </handlers> </pre>
Statistics	<p>Let's see what each handler does:</p> <ul style="list-style-type: none"> • APIAuthenticationHandler: Validates the OAuth2 bearer token used to invoke the API. It checks whether the token is of type Production or Sandbox and sets MessageContext variables as required to extend the default authentication handler, see Writing Custom Handlers. • APIThrottleHandler: Throttles requests based on the throttling policy specified by the policy. Throttling is applied both at the application level as well as subscription level. • APIMgtUsageHandler: Publishes events to BAM for collection and analysis of statistics. This has no effect if API usage tracking is enabled. See Publishing API Runtime Statistics for more information. • APIMgtGoogleAnalyticsTrackingHandler: Publishes events to Google Analytics. This has no effect if Google analytics tracking is enabled. See Integrating with Google Analytics for more information. • APIManagerExtensionHandler: Extends the mediation flow of messages passing through the API. Adding Mediation Extensions for more information. <p>Additionally, statistics are provided by the monitoring component, which integrates with WSO2 BAM.</p>

The components are depicted in the diagram below:



Users and roles

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

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

Tip: See [Managing Users and Roles](#) for more information.

API lifecycle

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

The following stages are available in the default API lifecycle:

- **CREATED:** API metadata is added to the API Store, but it is not deployed in the API gateway and therefore, is not visible to subscribers in the API Store.
- **PROTOTYPED:** the API is deployed and published in the API Store as a prototype. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. Users can invoke the

API without a subscription.

- **PUBLISHED:** The API is visible in the API Store and available for subscription.
- **DEPRECATED:** When an API is deprecated, new subscriptions are disabled. But the API is still deployed in the Gateway and is available at runtime to existing subscribers. Existing subscribers can continue to use it as usual until the API is retired.
- **RETIRED:** The API is unpublished from the API gateway and deleted from the store.
- **BLOCKED:** Access to the API is temporarily blocked. Runtime calls are blocked and the API is not shown in the API Store anymore.

Applications

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

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

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

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

Access tokens

An **access token** is a simple string that is passed as an HTTP header of a request. For example, "Authorization : Bearer NtBQkXoKELu0H1a1fQ0DWf06IX4a." Access tokens authenticate API users and applications, and ensure better security (e.g., prevent **DoS attacks**). If a token that is passed with a request is invalid, the request is discarded in the first stage of processing. Access tokens work equally well for SOAP and REST calls.

There are two types of access tokens:

- User access tokens
- Application access tokens

User access tokens

Tokens to authenticate the final user of an API. User access tokens allow you to invoke an API even from a third-party application like a mobile app. You generate/renew a user access token by calling the Login API through a REST client. For more information, see [Token API](#).

Application access tokens

Tokens to authenticate an application, which is a logical collection of APIs. You can access all APIs associated with an application using a single token, and also subscribe multiple times to a single API with different SLA levels. Application access tokens leverage OAuth2 to provide simple key management.

The steps below describe how to generate/renew application access tokens:

1. Log in to the API Store.
2. Click the **My Subscriptions** menu, select the application from the drop-down list and click the **Generate** or **Regenerate** buttons to create and renew access tokens.

The screenshot shows the 'My Subscriptions' section of the WSO2 API Manager interface. At the top, there are tabs for 'API Store', 'APIs', 'Prototyped APIs', 'My Applications', 'My Subscriptions' (which is highlighted with a red box), 'Statistics', and 'Tools'. Below the tabs is a search bar with the placeholder 'Search API' and a 'Go to API Publisher' link. The main content area is titled 'Subscriptions' and contains a message about generating keys for multiple APIs. It shows a table for 'Applications With Subscriptions' with a dropdown set to 'DefaultApplication' and a checked 'Show Keys' checkbox. The 'Keys - Production' section displays an 'Access Token' (K7va7zRj1EjS2PQuQ24TkR5Zqy8a), a 'Re-generate' button (highlighted with a red box), a 'Token Validity' of '3600 Seconds', and a 'Consumer Key' (IJKI4ogzbHJ1T_O2VjTiq_EQJf8a). The 'Allowed Domains' section shows 'ALL' and a 'Domains' button. The 'Keys - Sandbox' section shows a message that sandbox keys are not yet generated, a 'Generate' button (highlighted with a red box), and an 'Allowed Domains' section with 'ALL' and a 'Token Validity' of '3600 Seconds'.

Whenever an API call happens, the Gateway checks if the request originated from an allowed domain and grants access accordingly. You can specify these domains in the **Allowed Domains** text box. This ensures that clients from a restricted domain cannot access an API even if an application key is stolen (when the key is placed in client-side JS code).

Tip: When the client makes a request to an API that is only allowed to some domains, the request message must have an HTTP header to specify its domain name. Sending this header is mandatory only if the API is restricted to certain domains. An admin can configure this header name using `<ClientDomainHeader>` element under the `<APIGateway>` element in `<APIM_HOME>/repository/conf/api-manager.xml`.

For example, if the file contains `<ClientDomainHeader>domain</ClientDomainHeader>`, then the API invocation request must contain an HTTP header called `domain` with values as shown in the example below: `curl -v -H "Authorization: Bearer xxx" -H "domain: wso2.com" http://localhost:8280/twitter/1.0.0/search.atom?q=cat`

Throttling tiers

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

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

The following topics explain throttling:

- Different levels of throttling
- How throttling tiers work

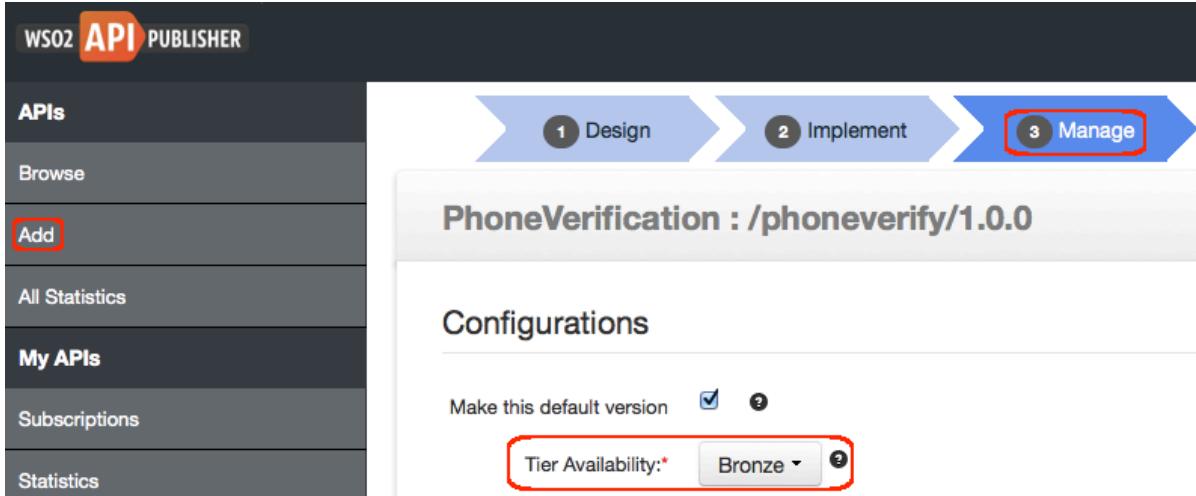
Different levels of throttling

Throttling applies in the following levels:

[[API-level throttling](#)] [[Application-level throttling](#)] [[Resource-level throttling](#)] [[IP-level throttling](#)]

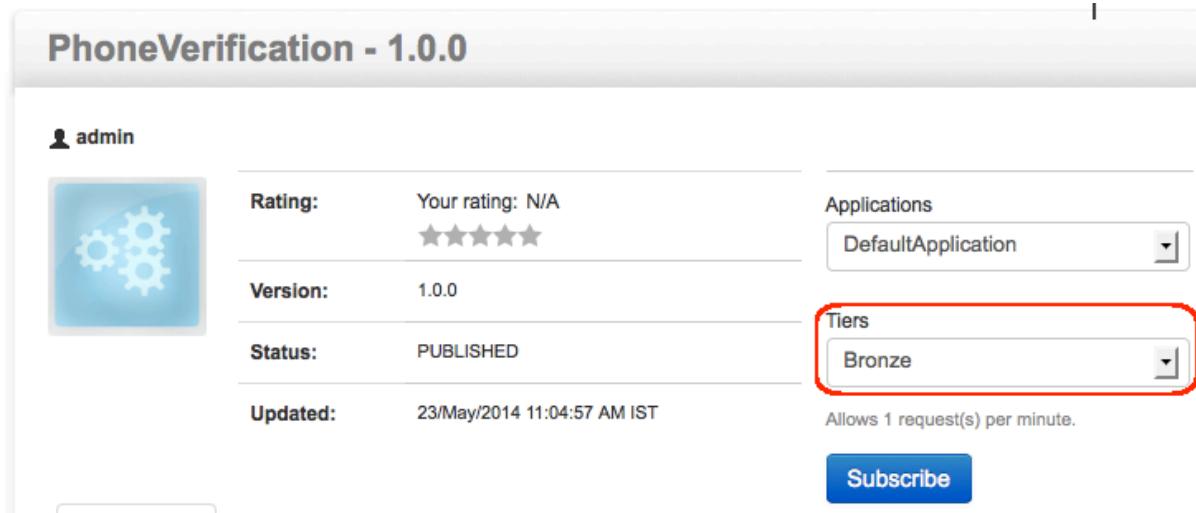
API-level throttling

API-level throttling tiers are defined when managing APIs using the API Publisher portal. The UI looks as follows:



The screenshot shows the WSO2 API Publisher interface. The top navigation bar says "WSO2 API PUBLISHER". On the left, a sidebar menu has "Add" highlighted with a red box. The main content area shows a progress bar with steps 1 (Design), 2 (Implement), and 3 (Manage), where "Manage" is also highlighted with a red box. Below the progress bar, the API details are shown: "PhoneVerification : /phoneverify/1.0.0". Under "Configurations", there is a checkbox "Make this default version" with a checked box and a question mark icon. Below it is a dropdown menu for "Tier Availability" set to "Bronze", which is also highlighted with a red box. The sidebar also includes "Browse", "All Statistics", "Subscriptions", and "Statistics" options.

After API-level throttling tiers are set and the API is published, at subscription time, the consumers of the API can log in to the **API Store** and select which tier they are interested in as follows:



The screenshot shows the API Store interface for the "PhoneVerification - 1.0.0" API. On the left, a user profile for "admin" is shown. The main details for the API are: "Rating: Your rating: N/A" with a 5-star icon, "Version: 1.0.0", "Status: PUBLISHED", and "Updated: 23/May/2014 11:04:57 AM IST". To the right, there are dropdown menus for "Applications" (set to "DefaultApplication") and "Tiers" (set to "Bronze", which is highlighted with a red box). Below the tiers dropdown, a note says "Allows 1 request(s) per minute." At the bottom right is a blue "Subscribe" button.

According to the tiers the subscriber selects, s/he is granted a maximum number of requests to the API. The default tiers are as follows:

- **Bronze:** 1 request per minute
- **Silver:** 5 requests per minute
- **Gold:** 20 requests per minute
- **Unlimited:** Allows unlimited access (you can disable the Unlimited tier by editing the <TierManagement> node of the <APIM_HOME>/repository/conf/api-manager.xml file)

Setting tier permissions: Users with Manage Tiers permission can set role-based permissions to API-level access throttling tiers. This is done using the **Tier Permissions** menu in the API Publisher as shown below. For each tier, you can specify a comma-separated list of roles and either Allow or Deny access to the list.

Tier	Permissions
Bronze	<input checked="" type="radio"/> Allow <input type="radio"/> Deny roles: Internal/everyone <small>Comma separated list (Ex: role1,role2,role3)</small>
Gold	<input checked="" type="radio"/> Allow <input type="radio"/> Deny roles: Internal/everyone

A subscriber logged into the API Store can consume APIs using a specific tier only if s/he is assigned to a role that is allowed access. In the API Store, the subscriber sees a list of tiers that is filtered based on the subscriber's role. Only the ALLOWED roles appear here. By default, all tiers are allowed to everyone.

Application-level throttling

Application-level throttling tiers are defined at the time an application is created in the API Store as shown below:

Use applications to subscribe to APIs and manage access keys. There is DefaultApplication pre-created to use and it can add more applications on this page.

Add New Application

Characters left: 70

Name:

Throttling Tier: **Unlimited** Allows unlimited requests

Callback URL:

Description:

Add

An application is a logical collection of one or more APIs and is required to subscribe to an API. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe to one API multiple times with different SLA levels.

An application is available to a consumer at different levels of service. For example, if you have infrastructure limitations in facilitating more than a certain number of requests to an application at a time, the throttling tiers can be set accordingly so that the application can have a maximum number of requests within a defined time. The default throttling levels are as follows:

- **Bronze:** 1 request per minute
- **Silver:** 5 requests per minute
- **Gold:** 20 requests per minute
- **Unlimited:** Unlimited access. The **Default Application**, which is provided out of the box has Unlimited tier set. You have the option to set it to a restricted limit.

Resource-level throttling

An API is made up of one or more resources. Each resource handles a particular type of request and is analogous to a method (function) in a larger API. Resource-level throttling tiers are set to HTTP verbs of an API's resources when Managing APIs using the API Publisher portal as shown below:

The screenshot shows the WSO2 API Publisher interface. The left sidebar has sections for APIs, Browse, Add, All Statistics, My APIs, Subscriptions, Statistics, and Tier Permissions. The 'Tier Permissions' section is currently selected. The main area shows the 'PhoneVerification : /phoneverify/1.0.0' configuration. The 'Manage' tab is highlighted with a red box. The 'Configurations' section includes fields for 'Make this default version' (checkbox), 'Tier Availability' (set to 'Bronze'), 'Transports' (HTTP and HTTPS checked), 'Sequences' (checkbox for custom sequences), and flow configuration tables for In Flow, Out Flow, and Fault Flow. The 'Resources' section shows a list with a 'path1' entry, which is highlighted with a red box. The 'path1' entry includes a 'GET' method, a URL pattern '/path1/resource', and a 'Bronze' tier level.

The default throttling levels are Gold, bronze, silver and unlimited, as explained in the previous sections.

When a subscriber views an API using the **API Store**, s/he can see the resource-level throttling tiers using the **Throttle Info** tab as follows:

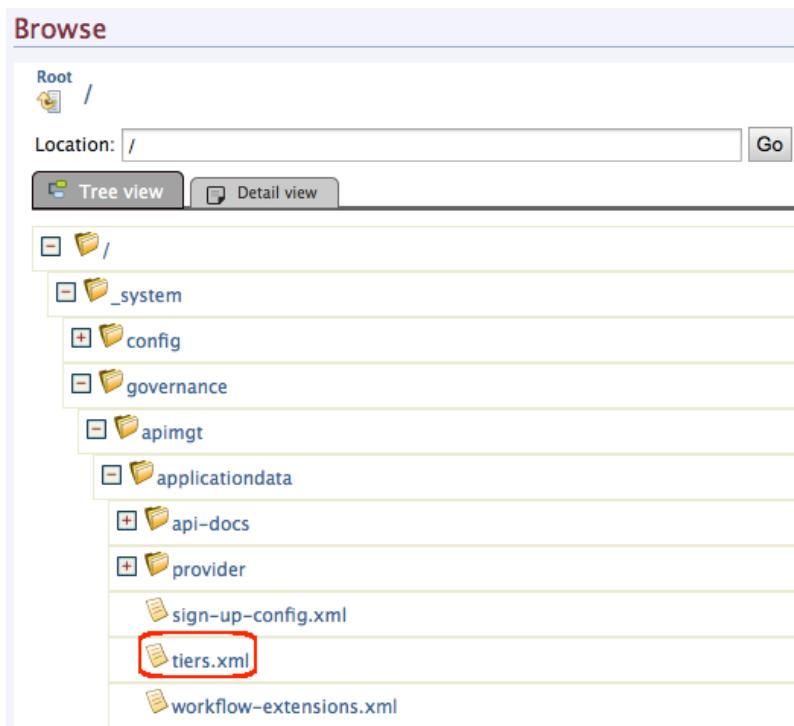
The screenshot shows the WSO2 API Store interface for the 'PhoneVerification - 1.0.0' API. The top navigation bar includes 'Overview', 'Documentation', 'API Console', 'Throttling Info' (which is highlighted with a red box), and 'Forum'. The main content area displays the API details: a blue gear icon, the rating (N/A), version (1.0.0), status (PUBLISHED), and last updated (23/May/2014 11:04:57 AM IST). The 'Throttling Info' tab shows a table with a single row: URL Prefix '/phoneverify/1.0.0', URL Pattern '/path1/resource', and Throttling Limit 'GET Allows unlimited requests'. A 'Subscribe' button is also present.

Subscribers are not allowed to change these throttling tiers. They are simply notified of the limitations.

IP-level throttling

In IP-based throttling, you can limit the number of requests sent by a client IP (e.g., 10 calls from single client).

1. Log in to the management console and click the **Resources -> Browse** menu.
2. Navigate to the `tiers.xml` file in the registry location `/_system/governance/apimgt/applicationdata`



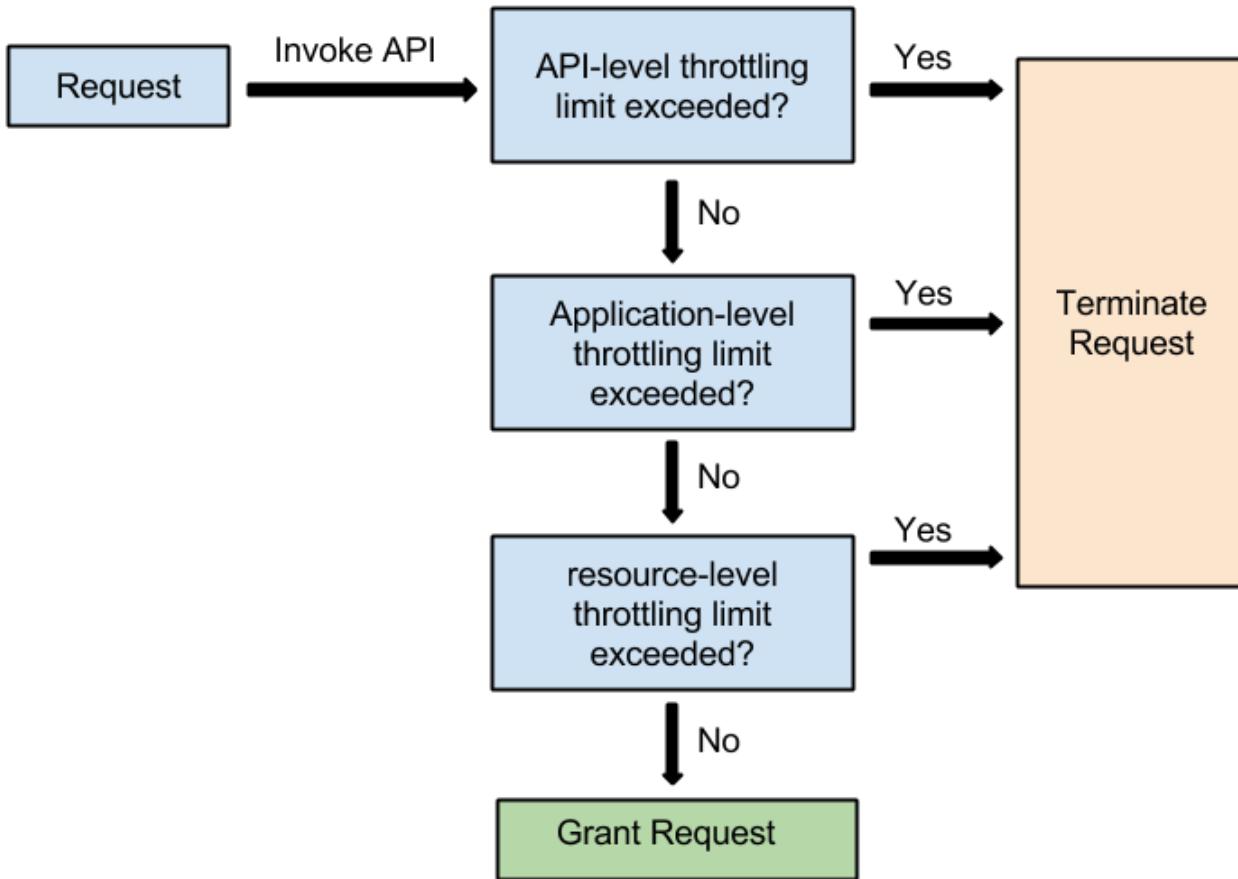
The screenshot shows the 'Browse' interface of the WSO2 API Manager. The left sidebar displays a tree view of the registry structure. The path 'applicationdata' is expanded, showing 'api-docs', 'provider', 'sign-up-config.xml', and 'tiers.xml'. The 'tiers.xml' file is highlighted with a red box. The top navigation bar includes 'Root /', a 'Location' input field with a 'Go' button, and tabs for 'Tree view' and 'Detail view'.

3. Add your policy. For example, the throttling policy shown below allows only 1 API call per minute for a client from 10.1.1.1 and 2 calls per minute for a client from any other IP address.

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">  
  <xmlns:throttle="http://www.wso2.org/products/wso2commons/throttle">  
    <throttle:MediatorThrottleAssertion>  
      <wsp:Policy>  
        <throttle:ID throttle:type="IP">10.1.1.1</throttle:ID>  
        <wsp:Policy>  
          <throttle:Control>  
            <wsp:Policy>  
              <throttle:MaximumCount>1</throttle:MaximumCount>  
              <throttle:UnitTime>60000</throttle:UnitTime>  
            </wsp:Policy>  
          </throttle:Control>  
        </wsp:Policy>  
      </throttle:MediatorThrottleAssertion>  
    </wsp:Policy>  
    <wsp:Policy>  
      <throttle:ID throttle:type="IP">other</throttle:ID>  
      <wsp:Policy>  
        <throttle:Control>  
          <wsp:Policy>  
            <throttle:MaximumCount>2</throttle:MaximumCount>  
            <throttle:UnitTime>60000</throttle:UnitTime>  
          </wsp:Policy>  
        </throttle:Control>  
      </wsp:Policy>  
    </wsp:Policy>  
  </throttle:MediatorThrottleAssertion></wsp:Policy>
```

How throttling tiers work

With capability to define throttling at three levels, the final request limit granted to a given user on a given API is ultimately defined by the consolidated output of all throttling tiers together.



Example: Lets say two users subscribed to an API using the Gold subscription, which allows 20 requests per minute. They both use the application App1 for this subscription, which again has a throttling tier set to 20 requests per minute. All resource level throttling tiers are unlimited. In this scenario, although both users are eligible for 20 requests per minute access to the API, each ideally has a limit of only 10 requests per minute. This is due to the application-level limitation of 20 requests per minute.

API visibility and subscription

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

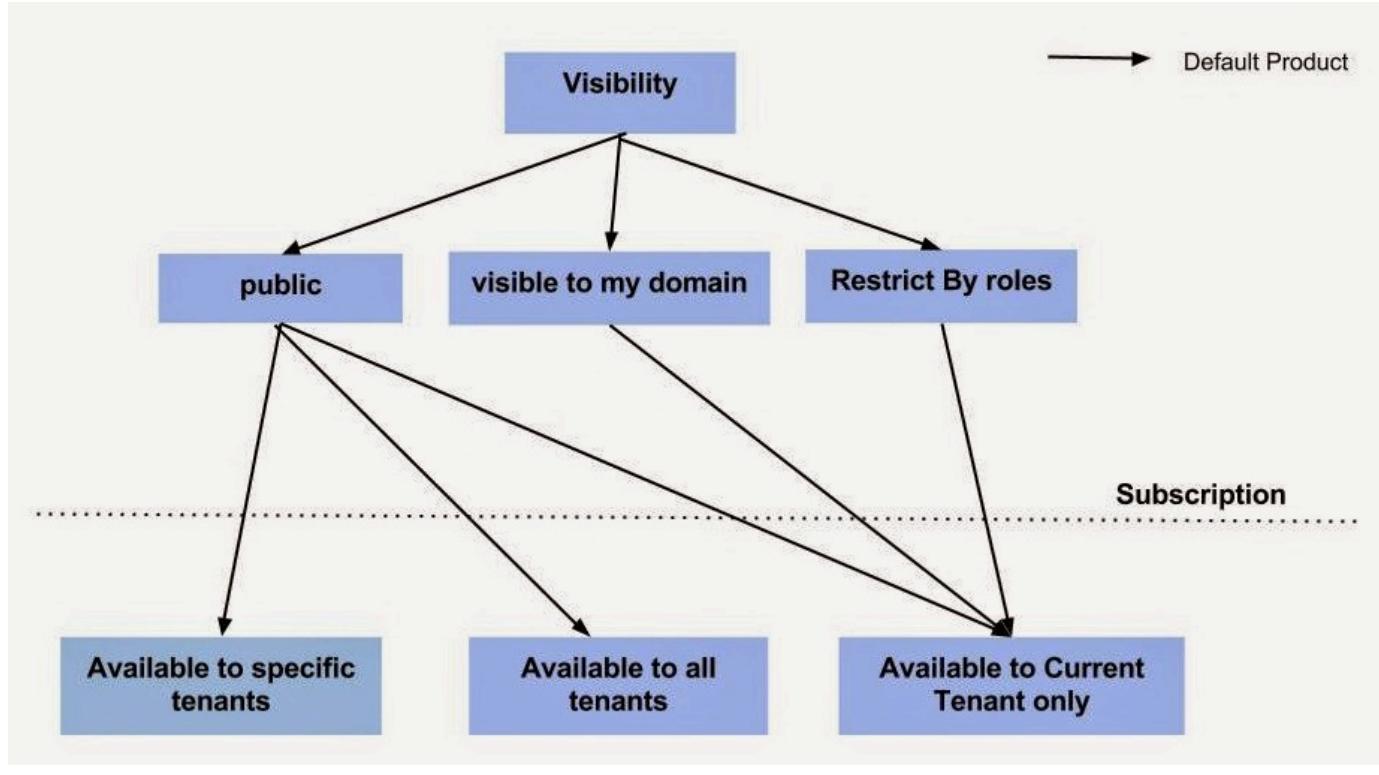
- **Public:** the API is visible to all users (registered and anonymous), and can be advertised in multiple stores (central and non-WSO2 stores).
- **Visible to my domain:** the API is visible to all users who are registered to the API's tenant domain.
- **Restricted by Roles:** The API is visible to its tenant domain and only to the user roles that you specify.

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

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

An API's visibility is directly related to the API's **subscription availability** because you cannot subscribe to

something you don't see in the store. The diagram below depicts this relationship:



API documentation visibility

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

```

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

Then, log in to the API Publisher, go to the **Doc** tab and click **Add new Document** to see a new drop-down list added to select visibility from:

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with options like 'APIs', 'Browse' (which is highlighted with a red box), 'Add', 'All Statistics', 'My APIs', 'Subscriptions', and 'Statistics'. The main area shows an API named 'PhoneVerify1 - 1.0.0'. At the top, there are tabs for 'Overview', 'Lifecycle', 'Versions', 'Docs' (which is highlighted with a red box), and 'Users'. Below these tabs, there's a 'Add New Document' button. The 'Docs' section contains fields for 'Name*' (with a red box around the input field), 'Summary' (with a red box around the input field), 'Type' (radio buttons for 'How To' (selected), 'Samples & SDK', 'Public Forum', 'Support Forum', and 'Other (specify)'), 'Source' (radio buttons for 'In-line' (selected), 'URL', and 'File'), and a 'Visibility' dropdown menu (which is highlighted with a red box) containing 'Visible to my domain'. At the bottom of the 'Docs' section are 'Add Document' and 'Cancel' buttons.

You set visibility in the following ways:

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

API resources

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

Resources

[+ Add Scopes](#)

/default

GET	<code>/* + Summary</code>	Application & Application User	Unlimited	+ Scope
POST	<code>/* + Summary</code>	Application & Application User	Unlimited	+ Scope
PUT	<code>/* + Summary</code>	Application & Application User	Unlimited	+ Scope
DELETE	<code>/* + Summary</code>	Application & Application User	Unlimited	+ Scope
OPTIONS	<code>/* + Summary</code>	None	Unlimited	+ Scope

API resources accept following attributes:

Attribute name	Description
OAuth Scopes	See OAuth scopes
URL Pattern	<p>A URL pattern can be one of the following types:</p> <ul style="list-style-type: none"> As a url-mapping. E.g., <code>/state/town/*</code> As a uri-template. E.g., <code>/{state}/{town}</code> <p>The terms url-mapping and uri-template come from synapse configuration language. When an API is published in the API Publisher, a corresponding XML definition is created in the API Gateway. This XML file has a dedicated section for defining resources. See examples below:</p> <pre><resource methods="POST GET" url-mapping="/state/town/*"> <resource methods="POST GET" uri-template="/{state}/{town}"></pre> <p>url-mapping performs a one-to-one mapping with the request URL, whereas the uri-template performs a pattern matching.</p> <p>Parametrizing the URL allows the API Manager to map the incoming requests to the defined resource templates based on the message content and request URI. Once a uri-template is matched, the parameters in the template are populated appropriately. As per the above example, a request made to http://gatewa_host:gateway_port/api/v1/texas/houston sets the value of state to <code>texas</code> and the value of town to <code>houston</code>. You can use these parameters within the synapse configuration for various purposes and gain access to these property values through the <code>uri.var.province</code> and <code>uri.var.district</code> properties. For more information on how to use these properties, see Introduction to REST API and the HTTP Endpoint of the WSO2 ESB documentation.</p> <p>Also see http://tools.ietf.org/html/rfc6570 on URI templates.</p>

HTTP Verb	The HTTP methods that specify the desired action to be performed on the resource. These methods can be GET, POST, PUT, DELETE or OPTIONS. Multiple methods can be selected.
Auth Type	<p>The authentication type of each HTTP method of the resource. You can give one of the following:</p> <ul style="list-style-type: none"> • None: No authentication is applied and the API Gateway skips the authentication process • Application: Authentication is done by the application. The resource accepts application access tokens. • Application User: Authentication is done by the application user. The resource accepts user access tokens. • Application and Application User: Both application and application user level authentication is applied. Note that if you select this option in the UI, it appears as Any in the API Manager's internal data storage and data representation, and Any will appear in the response messages as well. <p>Note that for the resources that have HTTP verbs (GET, POST etc.) requiring authentication (i.e., Auth Type is not NONE), set None as the Auth type of OPTIONS. This is to support CORS between the API Store and Gateway. (The above screenshot shows this).</p> <p>The auth type is cached in the API Manager for better performance. If you change the auth type through the UI, it takes about 15 minutes to refresh the cache. During that time, the server returns the old auth type from the cache. If you want the changes to be reflected immediately, please restart the server after changing the auth type.</p>

A resource's parameters are cached in the [resource cache](#) at the API Gateway.

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

Cross-origin resource sharing

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

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

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

```

<CORSConfiguration>
  <Enabled>true</Enabled>

  <Access-Control-Allow-Origin>https://localhost:9443,http://localhost:9763</Access-Control-Allow-Origin>

  <Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type</Access-Control-Allow-Headers>
</CORSConfiguration>

```

The elements are described below:

XML Elements	Values	Description
--------------	--------	-------------

<Enabled>	True/False	Used to enable/d CORS headers by default, CORS is needed for Swagger properly.
<Access-Control-Allow-Origin>	HTTP and HTTPS Store Address. Change the Host and Port for correct values of your store. For example, <code>https://localhost:9443,http://localhost:9763</code>	The value of the -Allow-Origin values are API Specification required for Swagger properly.
<Access-Control-Allow-Headers>	Header values you need to pass when invoking the API. For example, <code>authorization, Access-Control-Allow-Origin, Content-Type</code>	Default values are provided by Swagger to function

Change your code according to the sample given here.

If you try to invoke an API with inline endpoints, you add the CORS Handler in the <handlers> section of the API's configuration as follows. Find the API's configuration in the <APIM_HOME>/repository/deployment/server/synapse-configs/default/api folder.

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

OAuth scopes

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

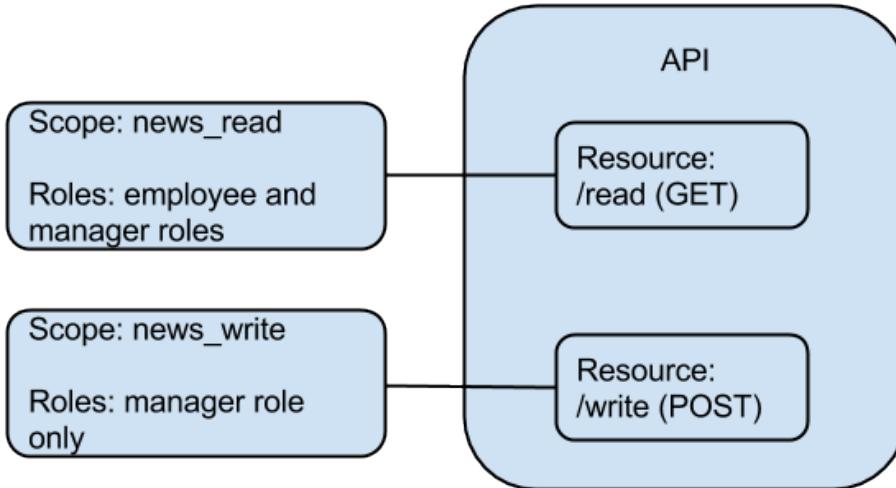
You can apply scopes to an API resource at the time the API is created or modified. In the API Publisher, click the **API -> Add** menu (to add a new API) or the **Edit** link next to an existing API. Then, navigate to the **Manage** tab and scroll down to see the **Add Scopes** button. A screen such as the following appears:

Define Scope

Scope Key	news_read
Scope Name	Read News
Roles	employee, manager
Description	Eg: This scope will group all the administration APIs

Scope Key	A unique key for identifying the scope. Typically, it is prefixed by part of the API's name for uniqueness, but is not necessarily reader-friendly.
Scope Name	A human-readable name for the scope. It typically says what the scope does.
Roles	The user role(s) that are allowed to obtain a token against this scope. E.g., manager, employee.

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



Assume that users named **Tom** and **John** are assigned the employee role and both the employee and manager roles respectively.

Tom requests a token through the Token API as `grant_type=password&username=nuwan&password=xxxx&scope=news_read news_write`. However, as Tom is not in the manager role, he will only be granted a token bearing the news_read scope. The response from the Token API will be similar to the following:

```

"scope": "news_read", "token_type": "bearer", "expires_in": 3299,
"refresh_token": "8579facb65d1d3eba74a395a2e78dd6",
"access_token": "eb51eff0b4d85cdaleb1d312c5b6a3b8"
  
```

Next, John requests a token as `grant_type=password&username=john&password=john123&scope=news_read news_write`. As john has both roles assigned, the token will bear both the requested scopes and the response will be similar to the following:

```
"scope": "news_read news_write", "token_type": "bearer", "expires_in": 3299,
"refresh_token": "4ca244fb321bd555bd3d555df39315",
"access_token": "42a377a0101877d1d9e29c5f30857e"
```

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

```
<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
    <ams:code>900910</ams:code>
    <ams:message>The access token does not allow you to access the requested resource</ams:message>
    <ams:description>Access failure for API: /orgnews, version: 1.0.0 with key: eb51eff0b4d85cdaleb1d312c5b6a3b8</ams:description>
</ams:fault>
```

 **Tip:** To invoke an API protected by scopes, you need to get an access token via the [Token API](#). Tokens generated from the **My Subscriptions** page in the API Store will not work.

API templates

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

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

Endpoints

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

WSO2 API Manager has support for a range of different endpoint types, allowing the API Gateway to connect with advanced types of backends. It supports [HTTP endpoints](#), [URL endpoints](#) (also termed as address endpoint), [WSDL endpoints](#), [Failover endpoints](#), [Load-balanced endpoints](#). For more information about endpoints and how to add, edit or delete them, see the [WSO2 ESB documentation](#).

Note the following:

- You can expose both REST and SOAP services to consumers through APIs.
- You cannot call backend services secured with OAuth through APIs created in the API Publisher. At the moment, you can call only services secured with username/password.
- The system reads gateway endpoints from the `<JAVA_HOME>/Repository/conf/api-manager.xml` file. When there are multiple gateway environments defined, it picks the gateway endpoint of the production environment. You can define both HTTP and HTTPS gateway endpoints as follows:

```
<GatewayEndpoint>http://${carbon.local.ip}:${http.nio.port},https://${carbon.local.ip}:${https.nio.port}</GatewayEndpoint>
```

- If both types of endpoints are defined, the HTTPS endpoint will be picked as the server endpoint.

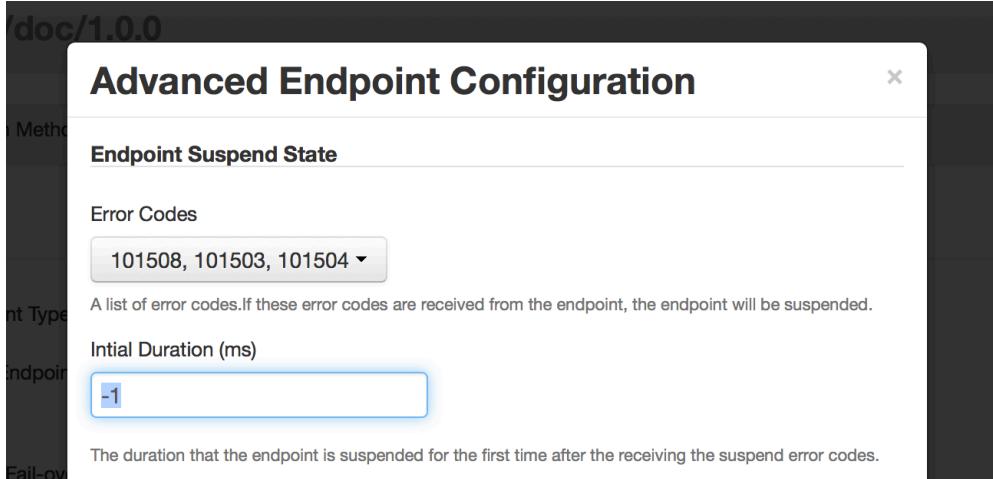


Tip: When you define secure (HTTPS) endpoints, set the `<parameter name="HostnameVerifier">AllowAll</parameter>` element to `AllowAll` in `<APIM_HOME>/repository/conf/axis2/axis2.xml` file's HTTPS transport sender configuration:

```
<parameter name="HostnameVerifier">AllowAll</parameter>
```

If not, the server throws an exception.

- When creating (or updating) Failover endpoints through the Publisher UI (in the Implement tab), you need to go into the **Advanced Options** of each endpoint and specify a set of Error Codes for the endpoint to fail over on and take off the Initial Duration by setting its value to -1.



Sequences

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

Caching

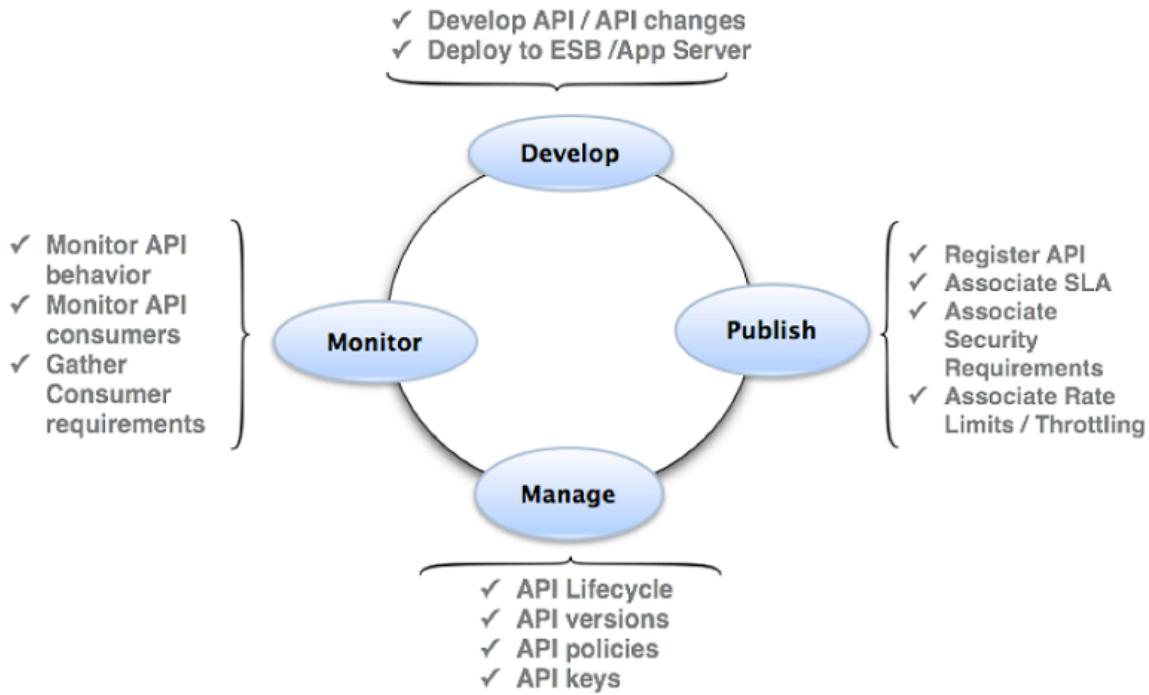
For information on configuring caching response messages and caching API calls at the Gateway and Key Manager server, see [Configuring Caching](#).

API Developer Tutorials

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

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

The diagram below shows the common lifecycle activities of an API developer/manager:



In this documentation, we use a role by the name **creator** to carry out API development-related tasks, and a role by the name **publisher** to carry out more management-related tasks. For instructions on adding the creator/publisher [Managing Users and Roles](#), see

To open the API Publisher, run the API Manager (see [Running the Product](#)) and access the following URL:

```
https://<YourHostName>:9443/publisher
```

 **Tip:** You cannot access the API Publisher using HTTP. It is exposed as HTTPS only.

The API Publisher log-in page opens.



You can log in as the administrator using `admin/admin` as the credentials or you can create users as described in [Managing Users and Roles](#). After logging in, see the following tutorials:

- Create and Publish an API
- Edit an API from the Source Code
- Add API Documentation
- Manage the API Lifecycle
- Publish to multiple external API stores
- Engage a new Throttling Policy
- Block Subscription to an API
- Enforce Throttling and Resource Access Policies

Create and Publish an API

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

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

- API visibility
- Resources
- Endpoints
- Throttling tiers
- Sequences
- Response caching

The steps below show how to create a new API.

1. Log in to the API Publisher.
2. Click the **Add** link and provide the information given in the table below.

Field		Sample value
Name		PhoneVerification
Context		/phoneverify
Version		1.0.0
Visibility		Public
Resources	URL pattern	CheckPhoneNumber
	Request types	GET, POST, OPTIONS
<p> Tip: Selecting the OPTIONS method is mandatory if you want to allow subscribers to invoke the API using the API Console in the store.</p> <p>For the resources that have methods requiring authentication (i.e., Auth Type is not NONE), you set None as the Auth type of OPTIONS to support CORS (Cross Origin Resource Sharing) between the API Store and Gateway.</p>		

3. Click **Add New Resource**. After the resource is added, expand its **GET** method, add the following parameters to it and click **Implement**. You add these parameters as they are required to invoke the API using our integrated API Console in later tutorials.

Parameter Name	Description	Parameter Type	Data Type	Required
PhoneNumber	Give the phone number to be validated	Query	String	True
LicenseKey	Give the license key as 0 for testing purpose	Query	String	True

/checkphonenumer

GET [/CheckPhoneNumber](#) [+ Summary](#) 

Implementation Notes :
[+ Add Implementation Notes](#)

Response Content Type : [application/json](#)

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required
body	Request Body	body	string	False
PhoneNumber	Give the phone number to be validated	query	string	True
LicenseKey	Give the license key as 0 for testing purpose	query	string	True

Parameter Name  [Add Parameter](#)

POST [/CheckPhoneNumber](#) [+ Summary](#) 

OPTIONS [/CheckPhoneNumber](#) [+ Summary](#) 

Save **Implement** [Cancel](#)

4. The **Implement** tab opens. Provide the information given in the table below. Click the **Show More Options** link to see the options that are not visible by default.

Field	Sample value
Implementation method	Backend
Endpoint type	HTTP endpoint
Production endpoint	<p>This sample service has two operations as <code>CheckPhoneNumber</code> and <code>CheckPhoneNumbers</code>. Let's use <code>CheckPhoneNumber</code> here.</p> <p>http://ws.cdyne.com/phoneverify/phoneverify.asmx</p> <p>To verify the URL, click the Test button next to it.</p>
Endpoint security scheme	Non Secured (If secured, user is asked for credentials of the backend service)



PhoneVerification : /phoneverify/1.0.0

Implementation Method Backend Endpoint Specify Inline

Endpoints

Endpoint Type:

Production Endpoint:
E.g.: http://appserver/resource

Sandbox Endpoint:
E.g.: http://appserver/resource

[Show More Options](#)

5. Click **Manage** to go to the Manage tab and provide the following information.

Field	Sample value
Tier Availability	Select all
Transports	HTTP/HTTPS

1 Design 2 Implement 3 Manage

PhoneVerification : /phoneverify/1.0.0

Configurations

Make this default version [?](#)
No default version defined for the current API

Tier Availability: **4 selected** [?](#)

Transports: HTTP HTTPS

Sequences: Check to select a custom sequence to be executed in the message flow

Response Caching: **Disabled** [?](#)

Subscriptions: **Available to current** [?](#)

Business Information

Resources

6. Click **Save & Publish**. This will publish the API that you just created in the API Store so that subscribers can use it.

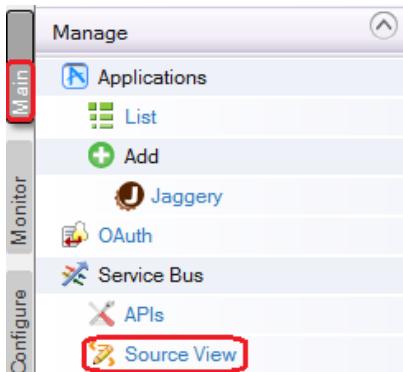
You have created an API.

Edit an API from the Source Code

Most common API configurations are facilitated through the API Publisher. You log in to the Publisher and click the **Edit** link next to the API's name to edit it using the UI. The **Edit** link is visible only to users with creator privileges.

To do more advanced configurations, you can go to the API's code-level using the management console as follows, or you can directly edit the file saved in `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api`.

1. Log in to the management console (<https://localhost:9443/carbon>) using admin/admin credentials
2. Select **Source View** sub menu under the **Service Bus** menu.



3. Source view contains the configuration of the API Gateway. You find sequences, filters, properties, APIs etc.

defined there. Search for the name of the API you want, and edit its content wrapped by the `<api></api>` elements.

You should not remove the default filter mediator and handler configurations in your API. They are needed for routing requests based on the throttling/security policies. If you want to add a custom mediator in the `insequence` path of a request, add that inside the filter mediator configuration as shown in the following example.

```
<filter source="$ctx:AM_KEY_TYPE" regex="PRODUCTION">
  <then>
    <class name="org.wso2.carbon.custommediator.CustomDataMediator"/>
    .....
  </then>
</filter>
```

4. Click **Update** to save your changes.

 and `<inSequence>` elements. The 'Update' button at the bottom left is also highlighted with a red box." data-bbox="145 345 825 705"/>

```

162 <outSequence>
163   <send/>
164 </outSequence>
165 </resource>
166 </api>
167 <api name="_WSO2AMAuthorizeAPI_" context="/authorize">
168   <resource methods="GET" url-mapping="/">
169     <inSequence>
170       <send>
171         <endpoint>
172           <address uri="https://localhost:9443/oauth2/authorize" />
173         </endpoint>
174       </send>
175     </inSequence>
176     <outSequence>
177       <send/>
178     </outSequence>
179   </resource>
180 </api>
181 </definitions>

```

Position: Ln 10, Ch 17 Total: Ln 181, Ch 8675

Toggle editor

Update **Reset**

5. Restart the server.

Add API Documentation

This section covers the following:

- [Add API Documentation In-line, using a URL or a File](#)
- [Add Apache Solr-Based Indexing](#)

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

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

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

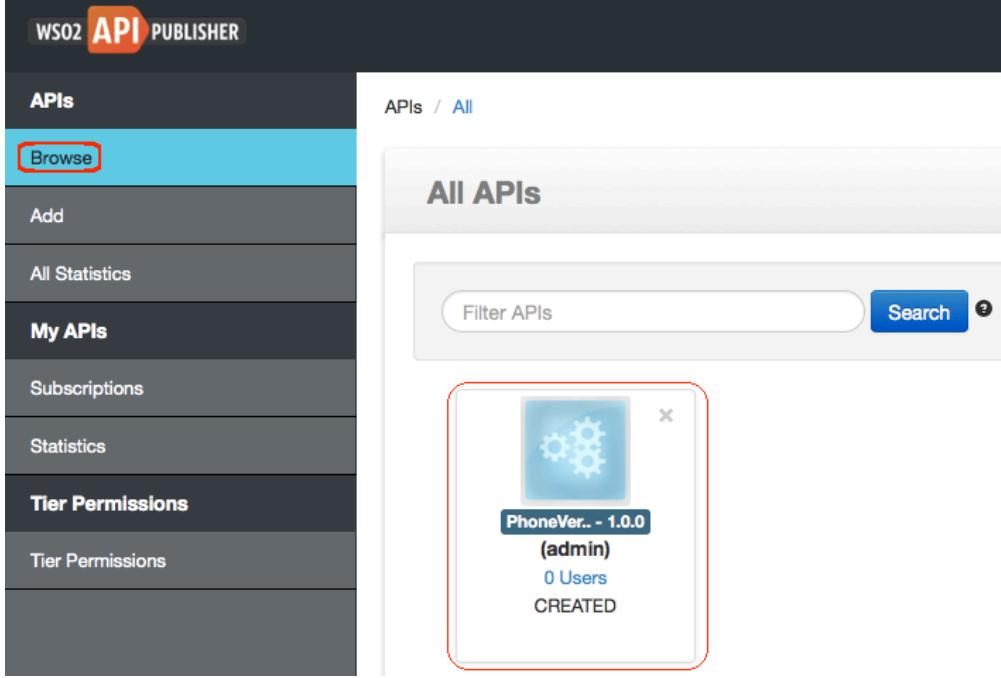
- **In-line:** Hosts documentation (How-tos, Samples, SDK, forums etc.) in the API Publisher itself and allows it to

be edited directly from the UI.

- **URL:** Links to file references (URLs) of an external configuration management system.
- **File:** Allows to upload the documentation directly to the server
- **Using the Swagger API Console**

 **Tip:** Do you want to set different visibility levels to the API documentation than the API? See [API documentation visibility](#).

1. Log in to WSO2 API Publisher.
2. Click the API to which you want to add documentation to (e.g., PhoneVerification 1.0.0).



The screenshot shows the WSO2 API Publisher interface. On the left, a sidebar menu is visible with options like 'APIs', 'Browse', 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions', and 'Tier Permissions'. The 'Browse' link is highlighted with a red box. The main content area is titled 'All APIs' and shows a list of APIs. One API, 'PhoneVer.. - 1.0.0', is highlighted with a red box. This API entry includes a gear icon, the name 'PhoneVer.. - 1.0.0', the status '(admin)', '0 Users', and the status 'CREATED'.

3. Select the **Docs** tab of the API and click the **Add New Document** link.

In-line documentation

4. Provide the following details to create a doc In-line.

Name	SimpleClient
Type	How To
Source	In-line
Summary	EXAMPLE REQUESTS TO PLACEFINDER WEBSERVICE

PhoneVerification - 1.0.0

[Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#)

[+ Add New Document](#)

Name*
SimpleClient

Summary
EXAMPLE REQUESTS TO
PLACEFINDER WEBSERVICE

Type

How To
 Samples & SDK
 Public Forum
 Support Forum
 Other (specify)

Source

In-line
 URL
 File

[Add Document](#) [Cancel](#)

5. Click the **Add Document** button.
6. After adding the document, click the **Edit Content** link associated with it.

PhoneVerification - 1.0.0 [Edit](#)

Overview	Lifecycle	Versions	Docs	Users
+ Add New Document				
Name	Type	Modified On	Actions	
SimpleClient	How To	23/07/2014 10:00:54	Edit Content Update Delete	

7. The embedded editor opens allowing you to edit the document's content in-line. Add in the content and click **S a v e** a n d **C l o s e**.

The image shows a screenshot of a web-based application interface. At the top, there is a toolbar with icons for file operations (New, Open, Save, Print, Find, Copy, Paste, Undo, Redo), text styling (Font Family, Font Size, Paragraph, Bold, Italic, Underline, etc.), and a search bar. Below the toolbar, there is a main content area with the following text:

EXAMPLE REQUESTS TO PLACEFINDER WEBSERVICE

Find the coordinates of a street address:

[http://where.yahooapis.com/geocode?q=1600+Pennsylvania+Avenue+Washington+DC&appid=\[yourappidhere\]](http://where.yahooapis.com/geocode?q=1600+Pennsylvania+Avenue+Washington+DC&appid=[yourappidhere])

Find the street address nearest to a point

[http://where.yahooapis.com/geocode?q=38.898717+-77.035974&gflags=R&appid=\[yourappidhere\]](http://where.yahooapis.com/geocode?q=38.898717+-77.035974&gflags=R&appid=[yourappidhere])

RATE LIMITS

Use of the Yahoo! Place Finder API web service should not exceed 50,000 requests per day.

If you believe your application will exceed such volume, please [contact us](#).

8. The API's **Doc** tab opens. Click the **Add New Document** link again.

Documentation using a URL

9. Then provide the following information to create another doc using a URL.

Name	CDYNE Wiki
Type	Other (Summary)
Source	URL http://api-portal.anypoint.mulesoft.com/cdyne/api/cdyne-phone-verification-api
Summary	<h1>CDYNE Phone Verification API</h1>

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#) [External API Stores](#)

[+ Add New Document](#)

Name*	<input type="text" value="CDYNE Wiki"/>
Summary	<input type="text" value="CDYNE Phone Verification API"/>
Type	<input type="radio"/> How To <input type="radio"/> Samples & SDK <input type="radio"/> Public Forum <input type="radio"/> Support Forum <input checked="" type="radio"/> Other (specify) <input type="text" value="Summary"/>
Source	<input type="radio"/> In-line <input checked="" type="radio"/> URL <input type="radio"/> File <input type="text" value="http://api-portal.anypoint.mulesoft.com/cdyne/api/cdyne-phone-verification-api"/>

[Add Document](#) [Cancel](#)

10. Click the **Add Document** button.
 11. The API's **Doc** tab opens again. Click the **Add New Document** link again to add yet another document using a file.

Documentation using a file

12. Enter the following information:

Name	API Manager Samples
Type	Samples & SDK
Source	You can provide any file format (common formats are PDF, HTML, .doc, text) of any size. For example, use the sample PDF file here .

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#) [External API Stores](#)

[+ Add New Document](#)

Name*

Summary

Type How To Samples & SDK Public Forum Support Forum Other (specify)

Source In-line URL File
 * WSO2 API Mana...

[Add Document](#) [Cancel](#)

13. After adding the details, click the **Add Document** button. You have now added three documents to the API: in-line, using a URL and a file.

PhoneVerification - 1.0.0 [Edit](#)

[Overview](#) [Lifecycle](#) [Versions](#) [Docs](#) [Users](#) [External API Stores](#)

[+ Add New Document](#)

Name	Type	Modified On	Actions
CDYNE Wiki	Other	01/09/2014 14:16:45	View Update Delete
API Manager Samples	Samples	01/09/2014 14:33:26	Open Update Delete
SimpleClient	How To	01/09/2014 14:34:09	Edit Content Update Delete

14. Log in to the API Store and click the PhoneVerification 1.0.0 API.

15. Go to the API's **Documentation** tab and see the documents listed by type. As a subscriber, you can read the doc and learn about the API.

PhoneVerification - 1.0.0

 admin

	Version: 1.0.0
	Status: PUBLISHED
	Updated: 26/Aug/2014 22:27:21 PM IST

[Overview](#) **Documentation** [API Console](#) [Throttling Info](#) [Forum](#)

[View Content](#) [Download](#)

[View Content](#) [Download](#)

[View Content](#) [Download](#)

16. Expand the categories and click the **View Content** or **Download** links to see the documentation content.

Other

Summary**Name:**

CDYNE Wiki

Summary:

CDYNE Phone Verification API

View Content**Samples****Name:**

API Manager Samples

Download**How To**

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

Add Apache Solr-Based Indexing

The API Manager has [Apache Solr](#) based indexing for API documentation content. It provides both the API Publisher and Store full-text search facility to search through API documentation, find documents and related APIs. The search syntax is **doc:keyword**. Search criteria looks for the keyword in any word/phrase in the documentation content and returns both the matching documents and associated APIs.

The following media types have Apache Solr based indexers by default, configured using the `<Indexers>` element in `<APIM_HOME>/repository/conf/registry.xml`.

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

Writing a custom index

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

1. Write a custom indexer. Given below is a sample indexer code.

```

package org.wso2.indexing.sample;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Arrays;
import org.apache.solr.common.SolrException;
import org.wso2.carbon.registry.core.exceptions.RegistryException;
import org.wso2.carbon.registry.core.utils.RegistryUtils;
import org.wso2.carbon.registry.indexing.IndexingConstants;
import org.wso2.carbon.registry.indexing.AsyncIndexer.File2Index;
import org.wso2.carbon.registry.indexing.Indexer;
import org.wso2.carbon.registry.indexing.solr.IndexDocument;

public class PlainTextIndexer implements Indexer {
    public IndexDocument getIndexedDocument(File2Index fileData) throws
SolrException,
    RegistryException {
        /* Create index document with resource path and raw content*/
        IndexDocument indexDoc = new IndexDocument(fileData.path,
RegistryUtils.decodeBytes(fileData.data), null);

        /* You can specify required field/value pairs for this indexing
document.
         * When searching we can query on these fields */
        Map<String, List<String>> fields = new HashMap<String,
List<String>>();
        fields.put("path", Arrays.asList(fileData.path));

        if (fileData.mediaType != null) {
            fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
Arrays.asList(fileData.mediaType));
        } else {
            fields.put(IndexingConstants.FIELD_MEDIA_TYPE,
Arrays.asList("text/plain"));
        }

        /* set fields for index document*/
        indexDoc.setFields(fields);
        return indexDoc;
    }
}

```

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

```

<indexers>
    <indexer class="org.wso2.indexing.sample.PlainTextIndexer"
mediaTypeRegEx="text/plain" profiles="default,api-store,api-publisher" />
</indexers>

```

The attributes of the above configuration are described below:

class	Java class name of the indexer
mefiaTypeRegEx	A regex pattern to match the media type
profiles	APIM profiles in which the indexer is available

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

Manage the API Lifecycle

In this section, we show you how to

- [Create a new API Version](#)
- [Deploy and Test as a Prototype](#)
- [Publish the new Version and Deprecate the old](#)

Create a new API Version

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

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



The examples here use the PhoneVerification API, which is created in section .

[Create and Publish an API](#)

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

1. Log in to the API Publisher as a user with the `publisher` role assigned.
2. Click the **Browse** menu and select the API that you want to create a version of (e.g., `PhoneVerification 1.0.0`).
3. The API's **Overview** page opens. Click the **Copy** button.

4. Give a version number, check the default version option and click **Done**.

To Version

2.0.0 Ex:v1.0.1

Default Version No default version defined for the current API

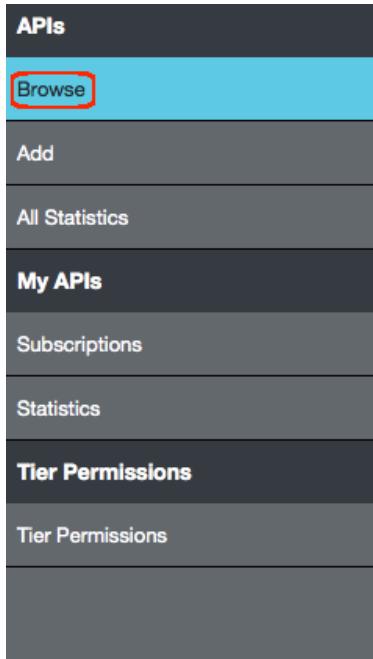
Done **Cancel**

 **Tip:** The **Default Version** option means that you make this version the default in a group of different versions of the API. A default API can be invoked without specifying the version number in the URL. For example, if you mark <http://host:port/youtube/2.0> as the default version when the API has 1.0 and 3.0 versions as well, requests made to <http://host:port/youtube/> get automatically routed to version 2.0.

If you mark any version of an API as the default, you get two API URLs in its **Overview** page in the API Store. One URL is with the version and the other is without. You can invoke a default version using both URLs.

If you mark an unpublished API as the default, the previous default, published API will still be used as the default until the new default API is published (or prototyped).

5. The **All APIs** page opens. Click on the new API version to open it.



APIs / All

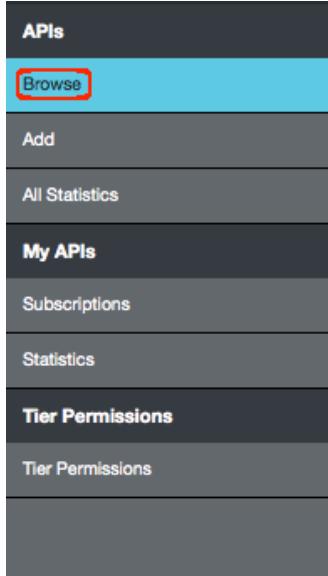
All APIs

Filter APIs

Search



6. Click the **Edit** link next to the API's name.



APIs / All / PhoneVerification-2.0.0

PhoneVerification - 2.0.0

[Edit](#)

Overview

Lifecycle

Versions

Docs

Users



CREATED

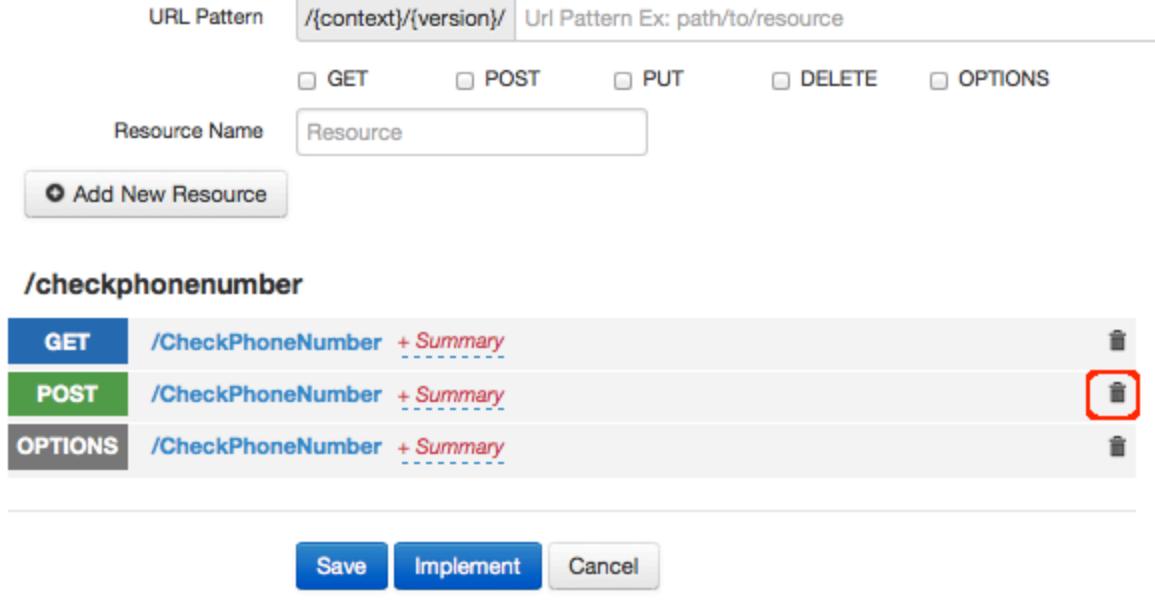
2.0.0

Docs

Context	/phoneverify
Production URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx
Sandbox URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx
Date Last Updated	04/12/2014 14:14:53
Tier Availability	Bronze,Gold,Unlimited,Silver
Default API Version	None

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

Resources



URL Pattern Url Pattern Ex: path/to/resource

GET POST PUT DELETE OPTIONS

Resource Name

/checkphonenumber

GET	/CheckPhoneNumber <small>+ Summary</small>	<input type="button" value="Delete"/>
POST	/CheckPhoneNumber <small>+ Summary</small>	<input style="border: 2px solid red;" type="button" value="Delete"/>
OPTIONS	/CheckPhoneNumber <small>+ Summary</small>	<input type="button" value="Delete"/>

8. Click **Save** once the edits are done.

 **Tip:** By default, only the latest version of an API is shown in the API Store. If you want to display multiple versions, set the `<DisplayMultipleVersions>` element to true in `<APIM_HOME>/repository/conf/api-manager.xml` file.

You have created a new version of an API. In the next tutorial, you [deploy this API as a prototype](#) and test it with its older versions.

Deploy and Test as a Prototype

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

 The examples here use the API `PhoneVerification 2.0.0`, which is created in the previous tutorial.

1. Log in to the API Publisher and select the API (e.g., `PhoneVerification 2.0.0`) that you want to `proto`.

The screenshot shows the 'All APIs' page in the WSO2 API Manager. On the left, a sidebar menu includes 'Browse' (which is highlighted with a red box), 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions', and 'Tier Permissions'. The main area displays two API versions: 'PhoneVer.. - 1.0.0' and 'PhoneVer.. - 2.0.0'. The 'PhoneVer.. - 2.0.0' entry is highlighted with a red box. It shows the icon, name, developer, user count (0), and status (PUBLISHED).

2. Click the **Lifecycle** tab of the API and change its state to **PROTOTYPED**. After creating a new version, you typically deploy it as a prototype for the purpose of testing and early promotion.

The screenshot shows the 'PhoneVerification - 2.0.0' API details page. The 'Lifecycle' tab is selected and highlighted with a red box. Below it, a 'State' dropdown is set to 'PROTOTYPED' and is also highlighted with a red box. There is a checked checkbox for 'Propagate Changes to API Gateway'. At the bottom are 'Update' and 'Reset' buttons, with 'Update' highlighted with a red box.

Tip: The **Propagate Changes to API Gateway** option is used to automatically change the API metadata in the API Gateway according to the information published in the API Store.

Tip: You can also deploy an API as a prototype using the **Implement** tab at the time it is created:

3. Log in to the API Store, click the **Prototyped APIs** menu and then click the newly prototyped API.

4. The APIs **Overview** page opens. Note that the subscription options are not available.
5. Note that you can read documentation, rate, comment and take part in the forum and social media. Also note that there are two URLs for both production and sandbox. This is because you marked the PhoneVerification 2.0.0 as the default version in step 4 of the previous tutorial.

PhoneVerification - 2.0.0

admin

Rating: 5.0 Your rating: 5/5

Version: 2.0.0

Status: PROTOTYPED

Updated: 05/Sep/2014 13:24:11 PM IST

Overview Documentation API Console Throttling Info Forum

Production and Sandbox URLs:

http://[REDACTED]:8280/phoneverify/2.0.0
http://[REDACTED]:8280/phoneverify

https://[REDACTED]:8243/phoneverify/2.0.0
https://[REDACTED]:8243/phoneverify

Share:

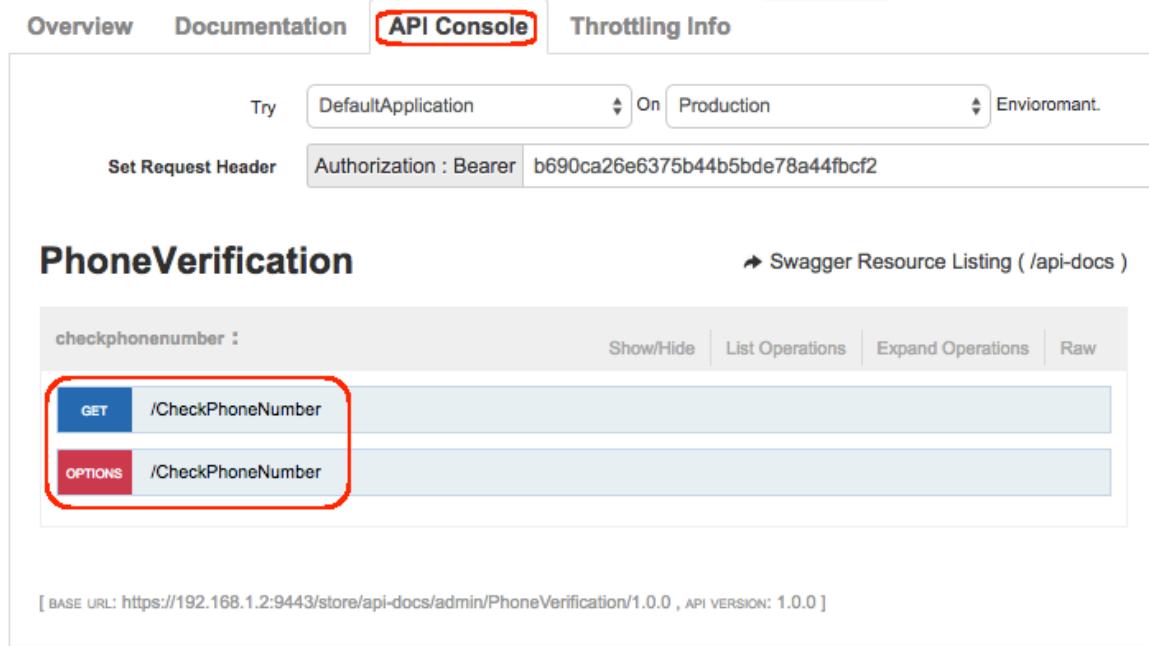
Social Sites Embed Email

f t g+ digg

Comments:

Characters left: 450

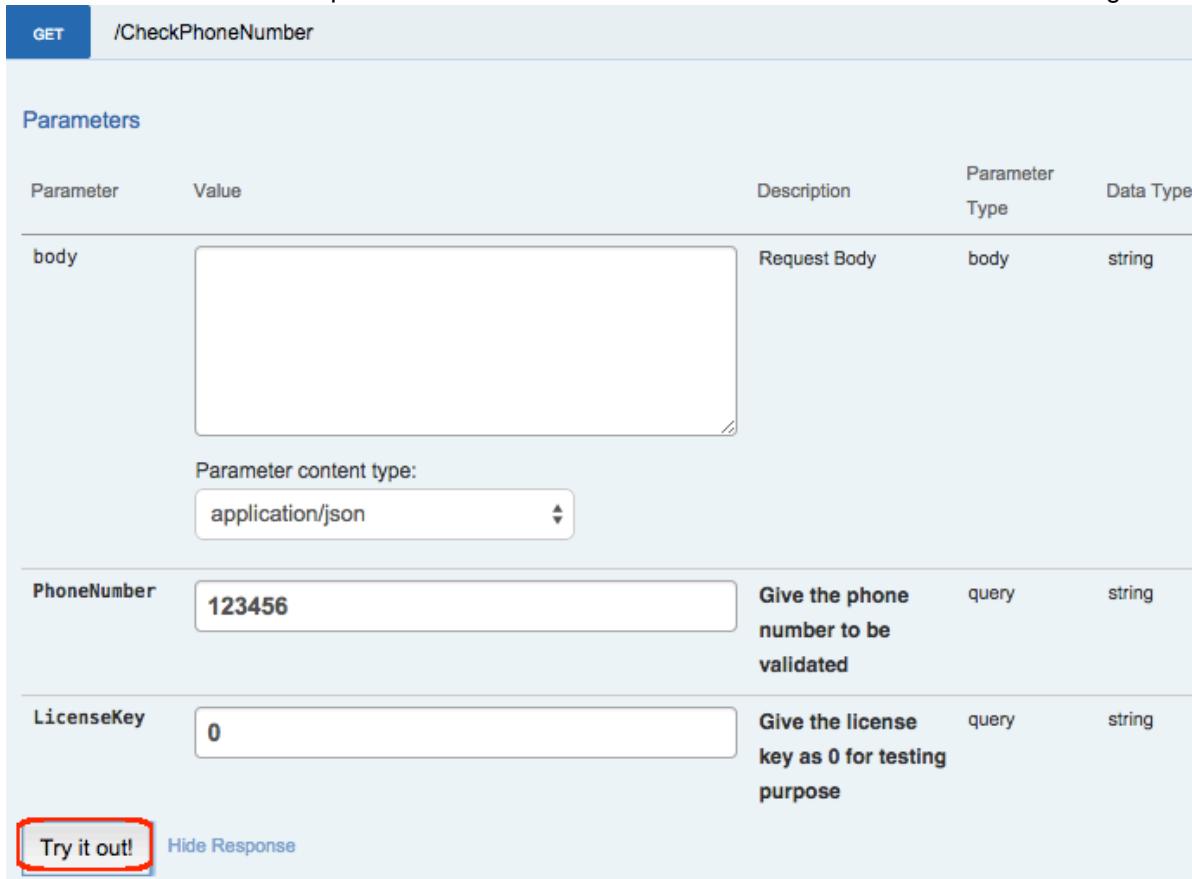
6. Click the **API Console** tab of the API and note that the **POST** method is not available as we removed that in the new version.



The screenshot shows the API Console interface for the 'PhoneVerification' API. The 'API Console' tab is active. The 'checkphonenumbers' operation is displayed with two methods: 'GET /CheckPhoneNumber' and 'OPTIONS /CheckPhoneNumber'. The 'GET' method is highlighted with a red box. The 'OPTIONS' method is also highlighted with a red box. The base URL is https://192.168.1.2:9443/store/api-docs/admin/PhoneVerification/1.0.0 and the API version is 1.0.0.

Let's invoke this API using the API Console.

7. Expand the **GET** method, give values to the **PhoneNumber** and **LicenseKey** parameters and invoke the API. You added these parameters to the API Console when creating the API.



The screenshot shows the 'Try it out!' button in the API Console interface, which is highlighted with a red box. The 'Try it out!' button is located at the bottom left of the interface.

8. Note that you get the expected result in the API Console. As this is a prototyped API, you do not need an access token and can leave the **Header** field blank.

Request URL

```
https://10.100.1.71:8243/phoneverify/2.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0
```

Response Body

```
<?xml version="1.0" encoding="utf-8"?>

<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query">
  <Valid>false</Valid>

  <OriginalNumber>123456</OriginalNumber>

  <CleanNumber>23456</CleanNumber>
  <Wireless>false</Wireless>
</PhoneReturn>
```

If you try to invoke the version 1.0.0 without an access token, you get an authentication error. This is because the older API version is published and requires an access token to be invoked.

You have prototyped an API and tested it along with its older and published versions. In the next tutorial, you [publish](#) the prototyped API and deprecate its older versions.

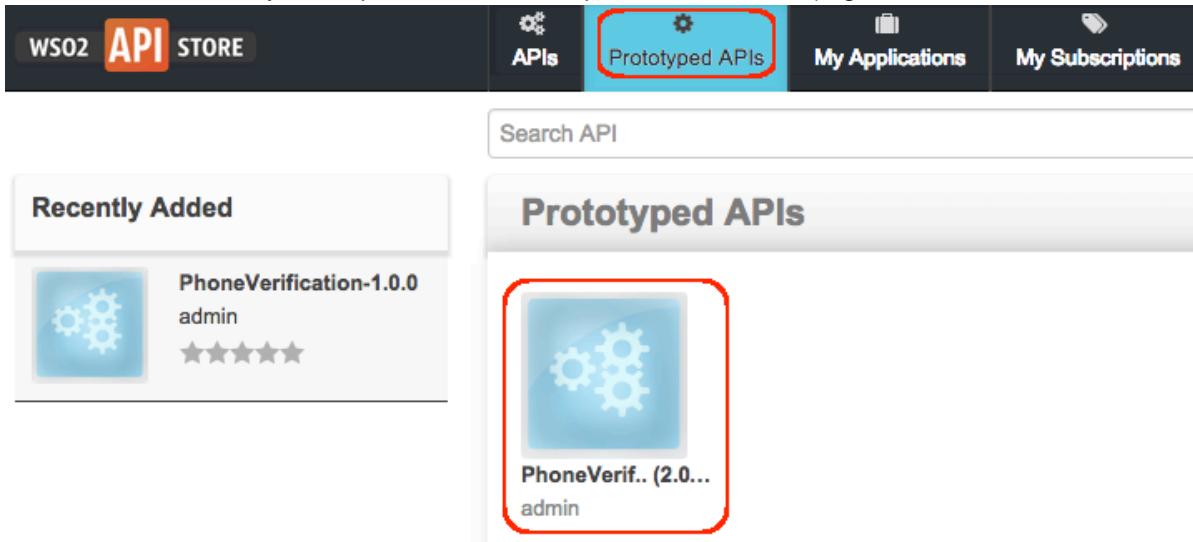
Publish the new Version and Deprecate the old

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

 For a description of the API lifecycle stages, see [API lifecycle](#).

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

1. Log in to the API Publisher as a user who has the [publisher](#) role assigned.
2. Click the API that you deprecated in the previous tutorial (e.g., PhoneVerification 2.0.0).



The screenshot shows the WSO2 API Store interface. At the top, there is a navigation bar with tabs: 'API STORE' (disabled), 'APIs' (disabled), 'Prototyped APIs' (selected and highlighted with a red box), 'My Applications', and 'My Subscriptions'. Below the navigation bar, there is a search bar labeled 'Search API'. On the left, there is a 'Recently Added' section showing a card for 'PhoneVerification-1.0.0' by 'admin' with a 5-star rating. On the right, there is a 'Prototyped APIs' section showing a card for 'PhoneVerif.. (2.0...)' by 'admin'. The 'Prototyped APIs' card is also highlighted with a red box.

3. Go to the API's **Lifecycle** tab and select the **PUBLISHED** state from the drop-down list. Then, select all the options and click **Update**.

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

The screenshot shows the WSO2 API Manager interface. On the left, there's a sidebar with options like 'Browse' (highlighted with a red box), 'Add', 'All Statistics', 'My APIs', 'Subscriptions', 'Statistics', 'Tier Permissions' (highlighted with a red box), and 'Tier Permissions'. The main area shows the 'PhoneVerification - 2.0.0' API details. The 'Lifecycle' tab is selected (highlighted with a red box). The 'State' is set to 'PUBLISHED'. Three checkboxes are checked and highlighted with a red box: 'Propagate Changes to API Gateway', 'Deprecate Old Versions', and 'Require Re-Subscription'. At the bottom, there are 'Update' and 'Reset' buttons, with 'Update' also highlighted with a red box.

The three options are described below:

Option	Description
Propagate Changes to API Gateway	Automatically changes the API metadata in the API Gateway according to the information published in the API Store. If unselected, you have to manually configure the Gateway.
Deprecate Old Versions	Automatically deprecates all prior versions of the API, if any.
Require Re-Subscription	Invalidates current user subscriptions, forcing the users to subscribe again.

The API is now published to the default API Store and all its previous versions are deprecated.

4. Log in to the default Store and click the **APIs** menu to see the API that you just published listed there. The older version (e.g., PhoneVerification 1.0.0) is no longer listed here as it is deprecated.

5. Click the **My Subscriptions** menu and look under the **Subscribed APIs** section. The subscriptions made to the older API versions must be deprecated now.

Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times different SLA levels.

Applications With Subscriptions

DefaultApplication ▾

[Keys - Production](#) ▾

[Keys - Sandbox](#) ▾

[Subscribed APIs](#) ▾

Deprecated



PhoneVerif.. - 1.0.0 X

admin

Bronze Subscription

 **Tip:** When an API is deprecated, new subscriptions are disabled (you cannot see the subscription options) and existing subscribers can continue to use the API as usual until it is eventually retired.

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

Publish to multiple external API stores

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

Capability to publish to external API Stores is not there by default. Follow the steps below to configure it.

1. Log in to APIM admin console (`https://<Server Host>:9443/carbon`) as admin and select **Browse** menu under **Resources** .



The screenshot shows the 'Resources' section of the WSO2 API Manager admin console. The 'Browse' menu option is highlighted with a red box. Below the menu, there is a search bar labeled 'Search'.

Below the search bar, the 'Browse' section is displayed. It shows a tree view of the registry structure. The root node is 'Root' (indicated by a folder icon and a slash). Under 'Root', there is a 'Location' input field with a value of '/'. Below the location field are two buttons: 'Tree view' (selected) and 'Detail view'. The tree view shows the following structure:

- Root /
- _system
- config
- governance
- apimgt
 - applicationdata
 - customsequences
 - externalstores
- external-api-stores.xml
- event

The 'external-api-stores.xml' file is highlighted with a red box.

2. The Registry opens. Go to `/_system/governance/apimgt/externalstores/external-api-store.xml`.

3. Click the **Edit as Text** link and change the `<ExternalAPIStores>` element of each external API store that you need to publish APIs to. For example,

```

<ExternalAPIStores>
    <StoreURL>http://localhost:9763/store</StoreURL>
    <ExternalAPIStore id="Store1" type="wso2">
        <DisplayName>Store1</DisplayName>
        <Endpoint>http://localhost:9763/store</Endpoint>
        <Username>xxxx</Username>
        <Password>xxxx</Password>
    </ExternalAPIStore>
    <ExternalAPIStore id="ProWeb" type="proWeb">
        <Name>ProgrammableWeb</Name>
        <Endpoint>xxxxxx</Endpoint>
    </ExternalAPIStore>
    <ExternalAPIStore id="Store2" type="wso2">
        <DisplayName>Store2</DisplayName>
        <Endpoint>http://localhost:9764/store</Endpoint>
        <Username>xxxx</Username>
        <Password>xxxx</Password>
    </ExternalAPIStore>
</ExternalAPIStores>

```

Note the following in the configuration above:

Element	Description
<ExternalAPIStore id="" type="">	id: The external store identifier, which is a unique value. type: Type of the Store. This can be a WSO2-specific API Store or an external one.
<StoreURL>	URL of the API store of the current APIM deployment. This is the URL to the API in the original publisher's store. APIs that are published to external stores will be redirected to this URL.
<DisplayName>	The name of the Store that is displayed in the publisher UI.
<Endpoint>	URL of the API Store.
<Username> & <Password>	Credentials of a user who has permissions to create and publish APIs.

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

4. Using the management console, [create an API](#).
5. Click on the newly created API to see a new tab called **External API Stores** added to the API Publisher console.

Display In External Stores: ProWeb
 Store1
 Store2

Save **Cancel**

Note the following:

- You can select multiple external API stores and click **Save** to publish your API to them.
- If the API creator updates the API after publication to external stores, either the creator or a publisher can simply push those changes to the published stores by selecting the stores and clicking **Save** again
- If the API creator deletes the API, each external store that it is published to will receive a request to delete the API.

6. Log in to an external API store where the API is published to and click it to open.
7. A link appears as **View Publisher Store** and it directs you to the original publisher's store through which you can subscribe to the API.

You have added multiple external stores to your registry and published your APIs to them.

Engage a new Throttling Policy

The steps below show how to engage a throttling policy to an API.

 **Tip:** For a description of throttling, see [Throttling Tiers](#).

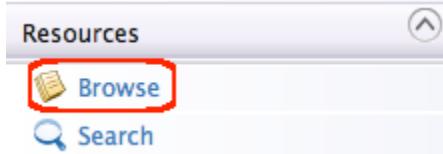
1. Write your throttling policy. For example, the following sample throttling policy points to a backend service and allows 1000 concurrent requests to a service.

```

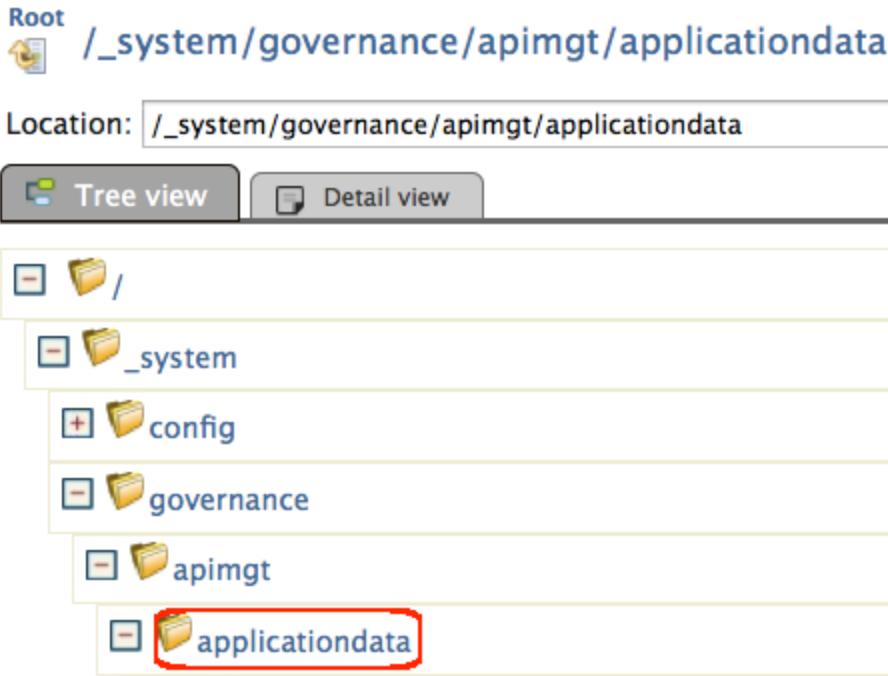
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-uti
  lity-1.0.xsd"
  xmlns:throttle="http://www.wso2.org/products/wso2commons/throttle"
    wsu:Id="WSO2MediatorThrottlingPolicy">
  <throttle:MediatorThrottleAssertion>
    <throttle:MaximumConcurrentAccess>1000</throttle:MaximumConcurrentAccess>
    <wsp:Policy>
      <throttle:ID throttle:type="IP">other</throttle:ID>
    </wsp:Policy>
  </throttle:MediatorThrottleAssertion>
</wsp:Policy>

```

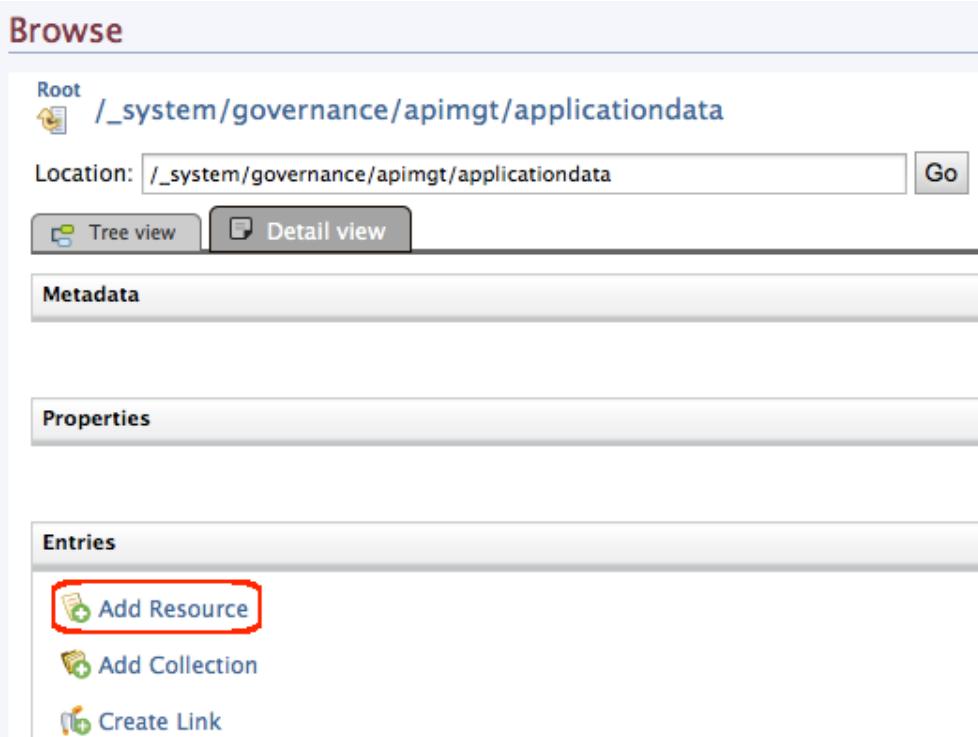
2. Log in to the API Manager's management console (<https://localhost:9443/carbon>) and click the **Resource > Browse** menu to view the registry.



3. Click the `/_system/goverence/apimgt/applicationdata` path to go to its detailed view.



4. In the detail view, click the **Add Resource** link.



5. Upload the policy file to the server as a registry resource.

6. In the management console, select the **Service Bus > Source View** menu.



7. The configurations of all APIs created in the API Manager instance opens. To engage the policy to a selected API, add it to your API definition. In this example, we add it to the login API.

```

<?xml version="1.0" encoding="UTF-8"?><api
  xmlns="http://ws.apache.org/ns/synapse" name="_WSO2AMLoginAPI_" context="/login">
  <resource methods="POST" url-mapping="/*">
    <inSequence>
      <send>
        <endpoint>
          <address uri="https://localhost:9493/oauth2/token" />
        </endpoint>
      </send>
    </inSequence>
    <outSequence>
      <send/>
    </outSequence>
  </resource>
  <handlers>
    <handler
      class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
      <property name="id" value="A"/>
      <property name="policyKey"
      value="gov:/apimgt/applicationodata/throttle.xml"/>
    </handler>
    <handler
      class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
    </handlers>
  </api>

```



Be sure to specify the same path used in step 5 in the policy key of your API definition.

You have successfully engaged a throttling policy to an API.

Block Subscription to an API

An API creator **blocks subscription** to an API as a way of disabling access to it and managing its usage and monetization. A blocking can be temporary or permanent. There is an unblocking facility to allow API invocations back.

You block APIs by subscriptions. That is, a given user is blocked access to a given API subscribed to using a given application. If a user is subscribed to two APIs using the same application and you block access to only one of the APIs, s/he can still continue to invoke the other APIs that s/he subscribed to using the same application. Also, s/he can continue to access the same API subscribed to using different applications.

Blocking can be done in two levels:

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

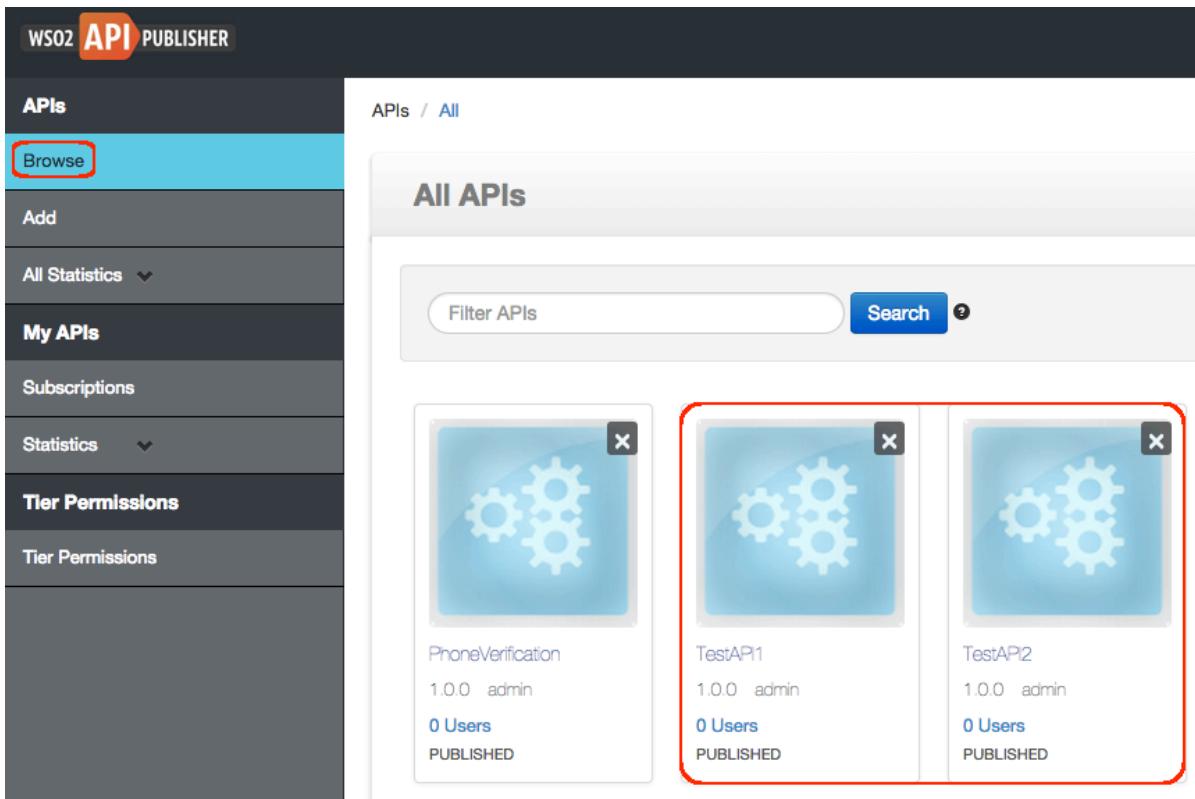
When [API Gateway caching](#) is enabled (it is enabled by default), even after blocking a subscription, consumers might still be able to access APIs until the cache expires, which happens approximately every 15 minutes.



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

- [Applications](#)
- [Throttling](#)
- [Access tokens](#)

1. Log in to the API Publisher.
2. Create two APIs by the names `TestAPI1` and `TestAPI2` and publish them to the API Store. In this example, the two APIs use the same backend and resources that were used when creating the `PhoneVerification` API in the first tutorial.



WSO2 API PUBLISHER

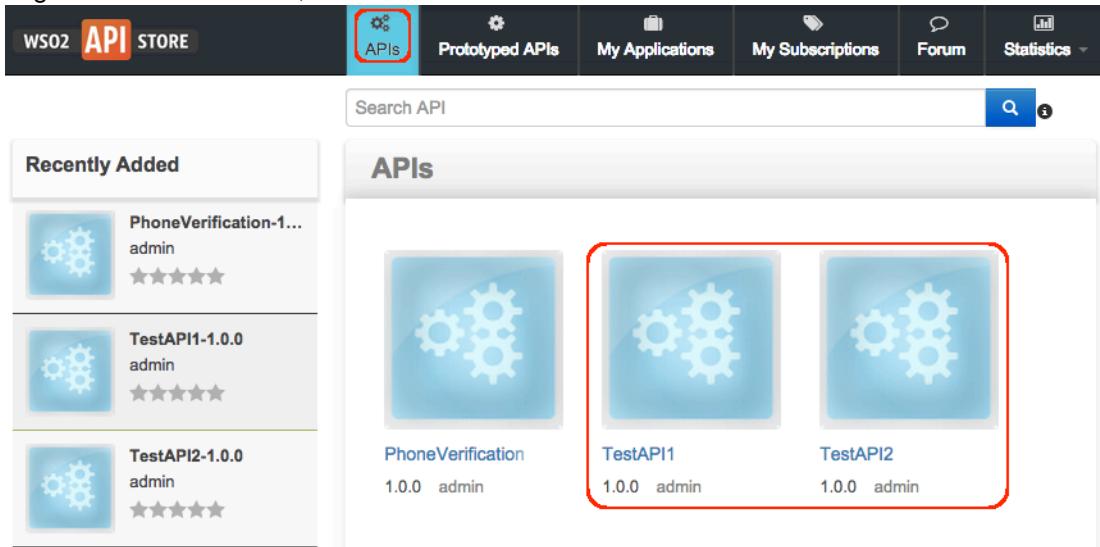
APIs / All

All APIs

Filter APIs Search ?

API	Version	Owner	Created	Published
PhoneVerification	1.0.0	admin	0 Users	PUBLISHED
TestAPI1	1.0.0	admin	0 Users	PUBLISHED
TestAPI2	1.0.0	admin	0 Users	PUBLISHED

3. Log in to the API Store, click the **APIs** menu and note that the two APIs are visible in the **APIs** page.



WSO2 API STORE

APIs ? Prototyped APIs My Applications My Subscriptions Forum Statistics

Search API ?

Recently Added

API	Version	Owner	Rating
PhoneVerification-1...	admin	★★★★★	
TestAPI1-1.0.0	admin	★★★★★	
TestAPI2-1.0.0	admin	★★★★★	

APIs

API	Version	Owner
PhoneVerification	1.0.0	admin
TestAPI1	1.0.0	admin
TestAPI2	1.0.0	admin

4. Subscribe to both APIs using the same application. You can use a default application or a new one.

TestAPI1 - 1.0.0

 nirdesha7.yahoo.com@nirdesha



Rating: Your rating: N/A

Version: 1.0.0
Status: PUBLISHED
Updated: 17/Aug/2014 23:33:28 PM PDT

Applications

Tiers

Allows 1 request(s) per minute.

Subscribe

5. Go to the **My Subscriptions** page and create an access token to the application.



Search API 🔍 ⓘ

Subscriptions

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple APIs and to subscribe to one API multiple times with different SLA levels.

Applications With Subscriptions

DefaultApplication Show Keys

Keys - Production ▼

Consumer Key : <input type="text" value="EqETBAkSPjvR6ValowsHkzYczLYa"/>	Allowed Domains <input type="text" value="ALL"/> <small>The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.</small>
Consumer Secret : <input type="text" value="vGEK3vbrZgTnFEEr7TAa9ZrrkFMa"/>	<input type="button" value="Update Domains"/>
Access Token: <input type="text" value="LoEjO7BkPupeL7VugsXw8Wx4UT8a"/>	<input type="button" value="Re-generate"/>
cURL ▼	Validity Time: <input type="text" value="3600"/> Seconds

6. Invoke both APIs using the access token you got in the previous step. In this example, we use the **API Console** tab of the APIs to invoke it.

Try DefaultApplication On Production Environment.

Set Request Header Authorization : Bearer 2c84df554dac1db6518e1fdb883185

Verify a phone number [Swagger Resource Listing \(/api-docs \)](#)

checkphonenumber :

Show/Hide List Operations Expand Operations Raw

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	123456	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

[Try it out!](#)

7. Note that you can successfully invoke both APIs.

Request URL

[https://\[REDACTED\]:8243/test1/1.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0](https://[REDACTED]:8243/test1/1.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0)

Response Body

```
<?xml version="1.0" encoding="utf-8"?>

<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.co
m/PhoneVerify/query">
<Valid>false</Valid>

<OriginalNumber>123456</OriginalNumber>

<CleanNumber>23456</CleanNumber>
<Wireless>false</Wireless>
</PhoneReturn>
```

You have subscribed to and invoked two APIs. Let's block one subscription and see the outcome.

8. Log in to the **API Publisher** and click the **Subscriptions** menu to open the **Subscriptions** page. It shows all APIs/applications that each user is subscribed to.

User	Application	Subscribed APIs	Actions
User1	DefaultApplication	TestAPI2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block
admin	DefaultApplication	Phone2-1.0.0 PhoneVerification-1.0.0 TestAPI1-1.0.0 TestAPI2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block

9. Block your previous subscription for TestAPI1. Select the production and sandbox option and click the **B l o c k** link.

admin	DefaultApplication	Phone2-1.0.0 PhoneVerification-1.0.0 TestAPI1-1.0.0 TestAPI2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block <input checked="" type="radio"/> Production Only <input checked="" type="radio"/> Production & Sandbox Block <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox Block
-------	--------------------	---	--

10. Note that the **Block** link immediately turns to **Unblock**, allowing you to activate the subscription back at any time.

11. Log back to the API Store and invoke the two APIs (TestAPI1 and TestAPI2) again.



You might have to **regenerate the access token** for DefaultApplication if the access token expiration time (1 hour by default) has passed since the last time you generated it. You can refresh the access token by going to the **My Subscriptions** page in the Store.

12. Note that you can invoke only TestAPI2 again. When you invoke TestAPI1, it gives a message that the requested API is temporarily blocked. Neither the API creator nor any subscriber can invoke the API until the **b l o c k** is **r e m o v e d**.

Response Body

```

<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
<ams:code>900907</ams:code>
<ams:message>The requested API is temporarily blocked</ams:message>
<ams:description>Access failure for API: /phoneneverify, version: 1.0.0 with key: 401a6a8284ef705638e433e2c717d4a</ams:description>
</ams:fault>

```

13. Go back to the API Publisher's **Subscriptions** page and **unblock** the subscription.

User	Application	Subscribed APIs	Actions
User1	DefaultApplication	TestAPI2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block
admin	DefaultApplication	Phone2-1.0.0 PhoneVerification-1.0.0 TestAPI1-1.0.0 TestAPI2-1.0.0	<input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block <input type="radio"/> Production Only <input checked="" type="radio"/> Production & Sandbox <input type="radio"/> Unblock <input type="radio"/> Production Only <input type="radio"/> Production & Sandbox <input checked="" type="radio"/> Block

14. Invoke **TestAPI1** again and note that you can invoke it as usual.

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

Enforce Throttling and Resource Access Policies

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

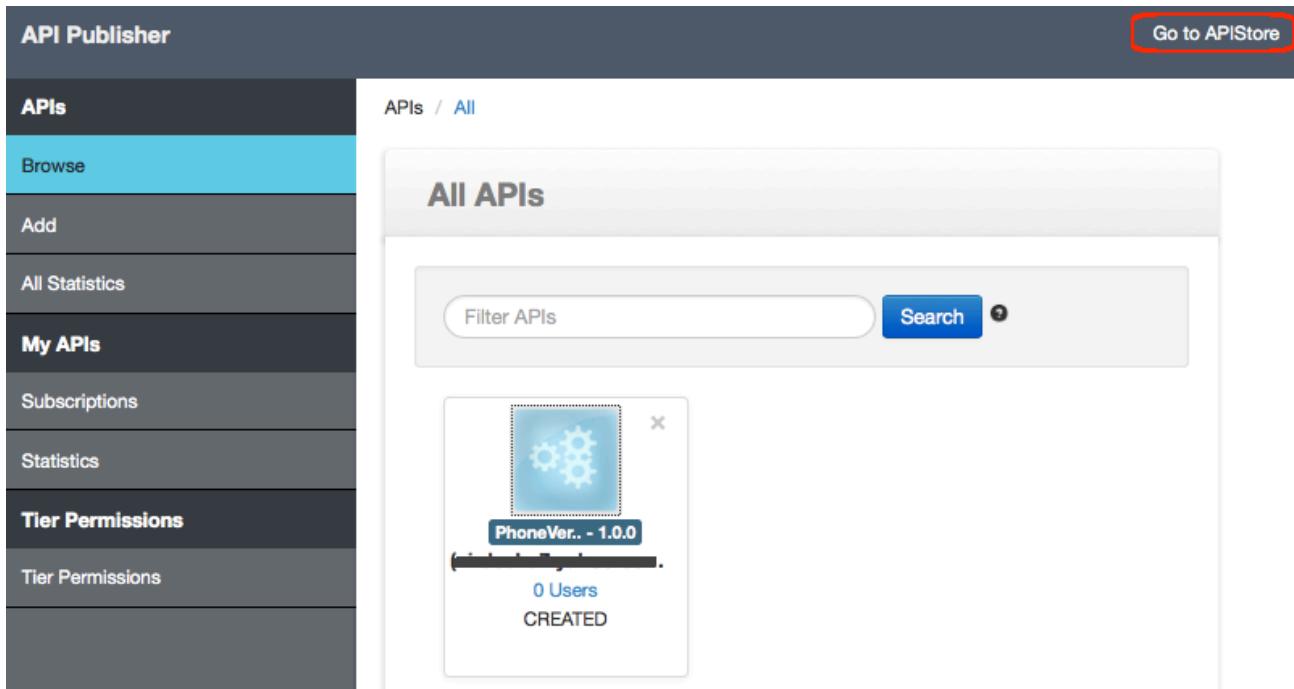
⚠ This tutorial uses the **PhoneVerification** API, which has one resource, GET and POST methods to access it and a throttling policy enforced.

Before you begin, follow the

[Create and Publish an API](#) and the [Subscribe to an API](#) tutorials to create, publish and subscribe to the **PhoneVerification** API using the **Bronze** throttling tier.

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

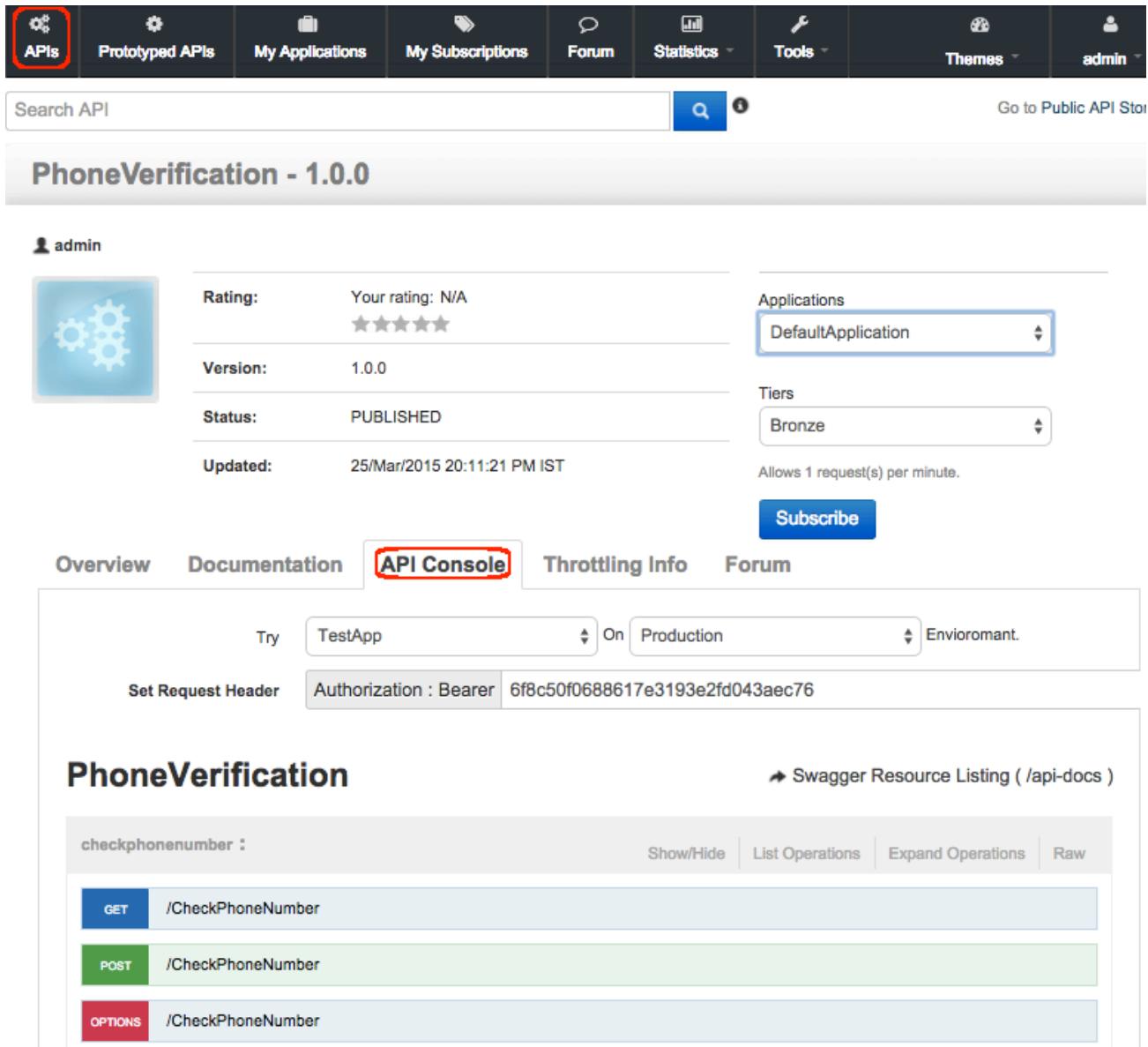
1. Log in to the API Cloud and the API Publisher will open automatically.
2. Click the **Go to API Store** link in the top right-hand corner of the API Publisher to open your default API Store.



The screenshot shows the WSO2 API Publisher interface. The left sidebar has a 'My APIs' section highlighted. The main area shows a list of APIs with one entry: 'PhoneVer.. - 1.0.0' which has 0 users and was created. A tip box is overlaid on the bottom left, containing the following text:

Tip: You can access any tenant's store using the URL `http://<hostname>/Store?tenant=<tenant_name>`.

3. Click the API Console tab.



The screenshot shows the WSO2 API Manager interface. At the top, there is a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Tools, Themes, and admin. The 'APIs' link is highlighted with a red box. Below the navigation bar is a search bar with the placeholder 'Search API' and a magnifying glass icon. To the right of the search bar is a link 'Go to Public API Store'.

The main content area displays the 'PhoneVerification - 1.0.0' API details. On the left, there is a profile icon for 'admin'. The API details include:

- Rating:** Your rating: N/A (with a 5-star icon)
- Version:** 1.0.0
- Status:** PUBLISHED
- Updated:** 25/Mar/2015 20:11:21 PM IST

On the right, there are dropdown menus for 'Applications' (set to 'DefaultApplication') and 'Tiers' (set to 'Bronze'). Below these is a note: 'Allows 1 request(s) per minute.' A 'Subscribe' button is located at the bottom right of this section.

Below the API details, there is a navigation bar with tabs: Overview, Documentation, API Console, Throttling Info, and Forum. The 'API Console' tab is highlighted with a red box. Under this tab, there is a 'Try' button with dropdown menus for 'TestApp' (selected), 'On' (selected), and 'Enviromant.' (selected). There is also a 'Set Request Header' button and a text input field showing 'Authorization : Bearer 6f8c50f0688617e3193e2fd043aec76'.

The 'PhoneVerification' API listing shows the following operations:

- checkphonenumber :** (GET) /CheckPhoneNumber
- checkphonenumber :** (POST) /CheckPhoneNumber
- checkphonenumber :** (OPTIONS) /CheckPhoneNumber

At the top right of the API listing, there is a link 'Swagger Resource Listing (/api-docs)'.

4. Expand the GET method, give the `PhoneNumber` and `LicenseKey` parameters and invoke the API.

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	(required)	Request Body	body	string

Parameter content type: application/json

PhoneNumber: 8004549078 (Give the phone number to be validated)

LicenseKey: 0 (Give the license key as 0 for testing purpose)

Try it out! Hide Response

5. The response appears in the console. As we used a valid phone number in this example, the response returns as valid.

Request URL: https://[REDACTED]:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=8004549078&LicenseKey=0

Response Body:

```

<?xml version="1.0" encoding="utf-8"?>

<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query">
  <Company>Toll Free</Company>
  <Valid>true</Valid>
  <Use>Assigned to a code holder for normal use.</Use>
  <State>TF</State>
  <RC />
  <OCN />
  <OriginalNumber>8004549078</OriginalNumber>
  <CleanNumber>8004549078</CleanNumber>

```

6. Within a minute after the first API invocation, make another attempt to invoke the API.

7. Note that you get a throttling error saying that you exceeded your quota. This is because you subscribed to the API on the Bronze throttling tier and the Bronze tier only allows you to make one call to the API per minute.

Response Body

```

<amt:Fault xmlns:amt="http://wso2.org/apimanager/throttling">
  <amt:Code>900800</amt:Code>
  <amt:Message>Message Throttled Out</amt:Message>
  <amt:Description>You have exceeded your quota</amt:Description>
</amt:Fault>

```

Let's try to invoke an invalid resource.

8. Install [cURL](#) if it is not there in your environment. Note that cURL comes by default in some operating systems. You can also use any other REST client.
9. Open the command line and execute the following cURL command with an invalid resource name (e.g., `CheckPhoneNum`.) Get the `<API URL>` from the API's **Overview** tab in the API Store.

```
curl -H "Authorization: Bearer <access token>" -v '<API URL>/CheckPhoneNum?PhoneNumber=123456&LicenseKey=0'
```

```
$ curl -H "Authorization: Bearer 6f8c50f0688617e3193e2fd043aec76" -v 'http://127.0.0.1:8280/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=123456&LicenseKey=0'
```

10. Note that you get a message as 'no matching resource.' This is because you are trying to access a REST resource that is not defined for the API.

```

<Access-Control-Allow-Headers: authorization,Access-Control-Allow-Origin,Content-Type
<Access-Control-Allow-Origin: *
<Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
<Content-Type: application/xml; charset=UTF-8
<Accept: */*
<Date:
<Server: WSO2-PassThrough-HTTP
<Transfer-Encoding: chunked
<
* Connection #0 to host left intact
<ams:fault xmlns:ams="http://wso2.org/apimanager/security"><ams:code>900906</ams:code><ams:message>No matching resource found in the API for the given request</ams:message><ams:description>Access failure for API: /phoneverify, version: 1.0.0 with key: 6f8c50f0688617e3193e2fd043aec76</ams:description></ams:fault>* Closing connection #0

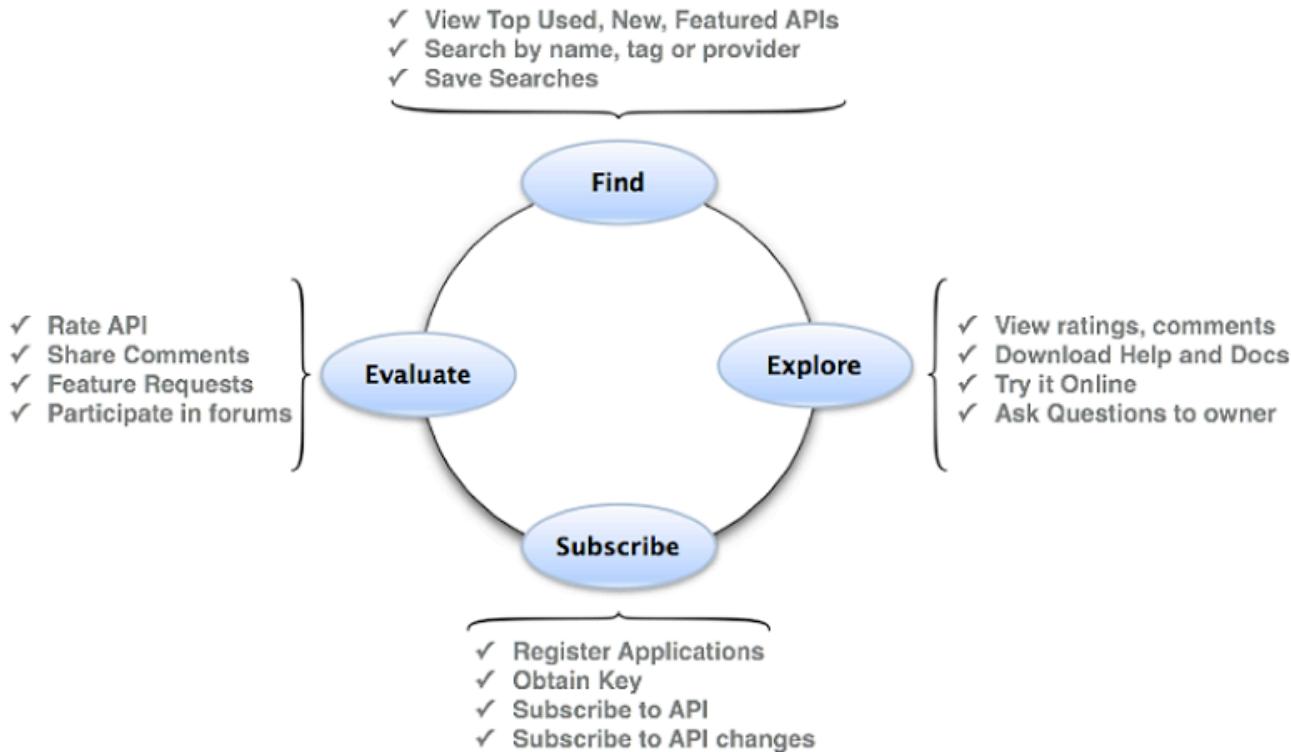
```

In this tutorial, you saw how the API Gateway enforces throttling and resource access policies to APIs.

Application Developer Tutorials

API Manager provides a Web interface called the **WSO2 API Store** to host and advertise published APIs. API consumers and partners can browse the store and subscribe to secured, protected, authenticated APIs.

The diagram below shows common API consumer lifecycle activities:



To open the API Store, run the API Manager (see [Running the Product](#)) and access the following URL:

`https://<HostName>:9443/store`

 **Tip:** You cannot access the API Store using HTTP. It is exposed as HTTPS only.

The API Store opens in the anonymous mode. You can see all public APIs without logging in, or self sign up using the [Sign-up](#) link to see all APIs.

After logging in, see the following tutorials:

- [Subscribe to an API](#)
- [Invoke an API using the Integrated API Console](#)
- [Invoke an API using the Integrated REST Client](#)

- Use the Community Features
- Invoke an API using a SOAP Client

Subscribe to an API

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

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

- API visibility and subscription availability
- Applications
- Application-level throttling
- Access tokens

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

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

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

2. Note the subscription options on the API's **Overview** page.

PhoneVerification - 1.0.0

admin



Rating: Your rating: N/A

Version: 1.0.0

Status: PUBLISHED

Updated: 23/May/2014 15:20:35 PM IST

Applications

Select Application...

Tiers

Bronze

Allows 1 request(s) per minute.

Subscribe

Overview

Documentation

API Console

Throttling Info

Forum

Production and Sandbox URLs:

[http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)
[http://\[REDACTED\]:8280/phoneverify](http://[REDACTED]:8280/phoneverify)

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)
[https://\[REDACTED\]:8243/phoneverify](https://[REDACTED]:8243/phoneverify)

Share:

[Social Sites](#)

[Embed](#)

[Email](#)



Comments:

Characters left: 450

3. Click the **My Applications** menu and create a new application.

Use applications to subscribe to APIs and manage access keys. There is DefaultApplication pre-created to use and it can add more applications on this page.

Add New Application

Characters left: 63

Name: (highlighted with a red box)

Throttling Tier: (highlighted with a red box) Allows unlimited requests

Callback URL:

Description:

Add (highlighted with a red box)

Tip: Instead of creating a new application, you can also use the default application.

4. Go back to the API's subscription options and select the application you just created, a tier and click **Subscribe**.

Applications: (highlighted with a red box)

Tiers: (highlighted with a red box)

Allows 1 request(s) per minute.

Subscribe (highlighted with a red box)

5. Click the **Go to My Subscriptions** button when prompted. The subscriptions page opens.

6. Select the application from the drop-down list and click **Generate** to create an application access token. You can use this token to invoke all APIs that you subscribe to using the same application.

WSO2 API Manager, version 1.8.0, WSO2 Inc.

APIs Prototyped APIs My Applications **My Subscriptions** Forum Statistics Tools Themes testme

Search API

Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times to a single API, with different SLA levels.

Applications With Subscriptions

TestApp

Show Keys

Keys - Production

Production keys are not yet generated for this application.

Allowed Domains

ALL

Leave empty or filling with "ALL" will allow all domains.

Token Validity: 3600 Seconds

Keys - Sandbox

Sandbox keys are not yet generated for this application.

Allowed Domains

ALL

Leave empty or filling with "ALL" will allow all domains.

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

7. Install **cURL** if it is not there in your environment. Note that **cURL** comes by default in some operating systems. You can also use any other REST client.
8. Open the command line and execute the following **cURL** command:

```
curl -H "Authorization: Bearer <access token>" -v '<API URL>'
```

Be sure to replace the placeholders as follows:

- **<access token>**: Give the token generated in step 8
- **<API URL>**: Go to the API's **Overview** tab in the API Store and copy the production URL and append the payload to it. E.g., <http://localhost:8280/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0>

Here's an example:

```
curl -H "Authorization: Bearer 8e64c4201d1c311c76a9c540856d1043" -v 'http://localhost:8280/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0'
```

9. Note the result that appears on the command line. In this example, the phone number is invalid.

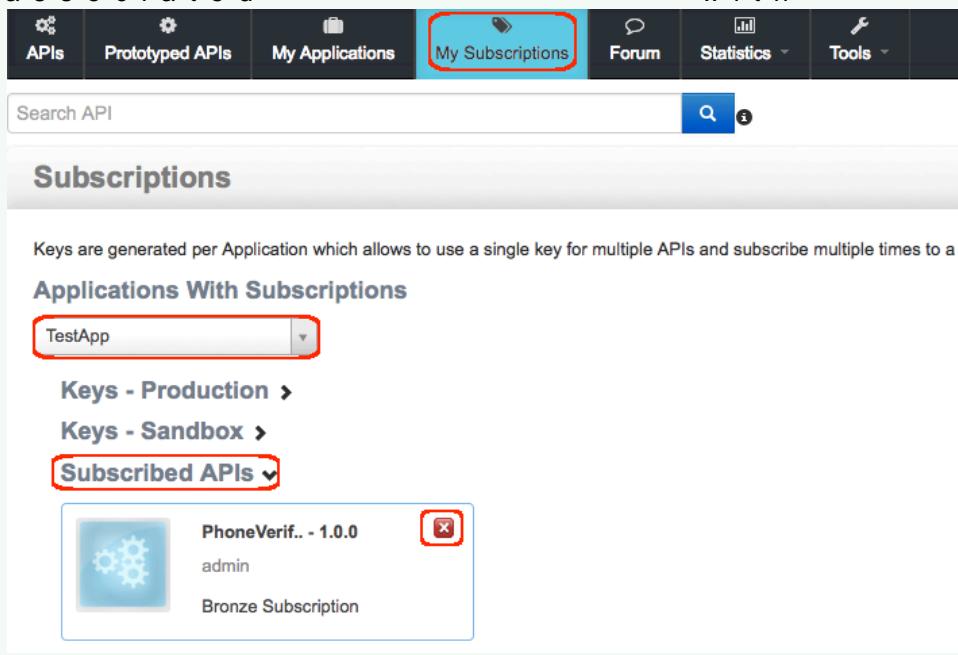
```
$ curl -H "Authorization :Bearer k29TzFjfB8UY7swhF2oIp5uSNfYa" -v 'http://localhost:8280/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0'
* About to connect() to localhost port 8280 (#0)
* Trying ::1...
* connected
* Connected to localhost (::1) port 8280 (#0)
> GET /phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=123456&LicenseKey=0 HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
> Host: localhost:8280
> Accept: */*
> Authorization :Bearer k29TzFjfB8UY7swhF2oIp5uSNfYa
>
< HTTP/1.1 200 OK
< X-AspNet-Version: 4.0.30319
< Access-Control-Allow-Headers: authorization,Access-Control-Allow-Origin,Content-Type
< Access-Control-Allow-Origin: *
< Expires: -1
< Content-Type: text/xml; charset=utf-8
< X-Powered-By: ASP.NET
< Pragma: no-cache
< Cache-Control: no-cache
< Date: Fri, 17 Oct 2014 08:00:24 GMT
< Server: WSO2-PassThrough-HTTP
< Transfer-Encoding: chunked
<
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query">
  <Valid>false</Valid>
  <OriginalNumber>123456</OriginalNumber>
  <CleanNumber>23456</CleanNumber>
  <Wireless>false</Wireless>
* Connection #0 to host localhost left intact
</PhoneReturn>* Closing connection #0
[...]
```

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

```
curl -H "Authorization :Bearer <your token here>" --data
"PhoneNumber=123456&LicenseKey=0"
http://localhost:8280/phoneverify/1.0.0/CheckPhoneNumber
```

You have subscribed to an API and invoked it.

 To unsubscribe from an API, go to the **My Subscriptions** menu in the API Store, select the application used for the subscription, find the API under the **Subscribed APIs** section and click the delete icon associated with it.



Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times to a single API.

Applications With Subscriptions

TestApp

Keys - Production >

Keys - Sandbox >

Subscribed APIs ▾

PhoneVerif.. - 1.0.0 

admin

Bronze Subscription

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

Invoke an API using the Integrated API Console

There are different ways to **invoke an API**: using the integrated API console, the integrated [WSO2 REST Client](#) or a third-party tool like cURL.

Swagger (<https://developers.heloreverb.com/swagger>) is a specification and a complete framework for describing, producing, consuming, and visualizing RESTful Web services as interactive documentation. For the Swagger specification of API declaration, see <https://github.com/wordnik/swagger-core/wiki/API-Declaration>.

The API Publisher has integrated Swagger to facilitate simple, interactive API documentation and invocation. The documentation is given in a Swagger API definition, which is the JSON representation of the API that is created using the information provided at the time the API is created. The Swagger JSON files are saved in the registry. You can edit the API definition using the JSONMate text editor in the API Publisher.

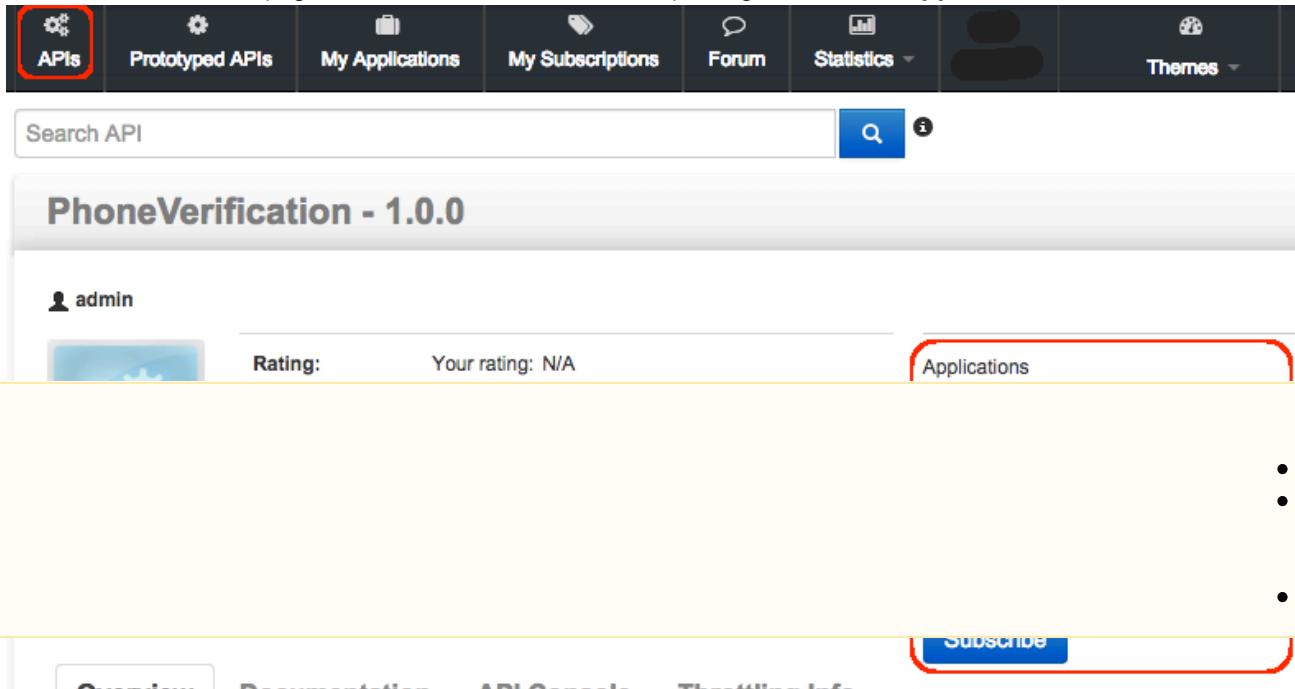
Let's see how to use the API Publisher to edit the Swagger API definition and then use the API Console in the Store to invoke the API.

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

- [Applications](#)
- [Throttling](#)
- [Access tokens](#)
- [Cross-origin resource sharing](#) if you have the **API Store and Gateway in different ports** or you want to invoke an API with **inline endpoints**.

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

1. Log in to the API Store and click an API (e.g., PhoneVerification).
2. Subscribe to the API (e.g., PhoneVerification 1.0.0) using the **default application** and an available tier



The screenshot shows the WSO2 API Manager API Store interface. The top navigation bar includes tabs for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, and Themes. The 'APIs' tab is highlighted with a red box. Below the navigation is a search bar with the placeholder 'Search API' and a magnifying glass icon. The main content area displays the 'PhoneVerification - 1.0.0' API details. The API title is 'PhoneVerification - 1.0.0'. Below it, the user 'admin' is listed. A warning icon is present. The 'Rating' section shows 'Rating: Your rating: N/A'. The 'Applications' section is highlighted with a red box. At the bottom of the page, there are tabs for Overview, Documentation, API Console, and Throttling Info. The 'Overview' tab is selected. A large red box highlights the 'Subscribe' button at the bottom right of the main content area.

3. Go to the **My Subscriptions** page and generate a production key to the default application using which you subscribed to the API

3.

Subscriptions

Create access tokens to applications. Because an application is a logical collection of APIs, you can use a single access token to invoke multiple APIs and to subscribe to one API multiple times with different SLA levels.

Applications With Subscriptions

DefaultApplication Show Keys

Keys - Production

Consumer Key :
G7jkZWkrKMQlof4XvcCWH9D561Ea

Consumer Secret :
hppj_c_7D1MUK00PamoQUrjf8la

Access Token:
1ca4534e9b04d6b33efcc462ba882a0

cURL

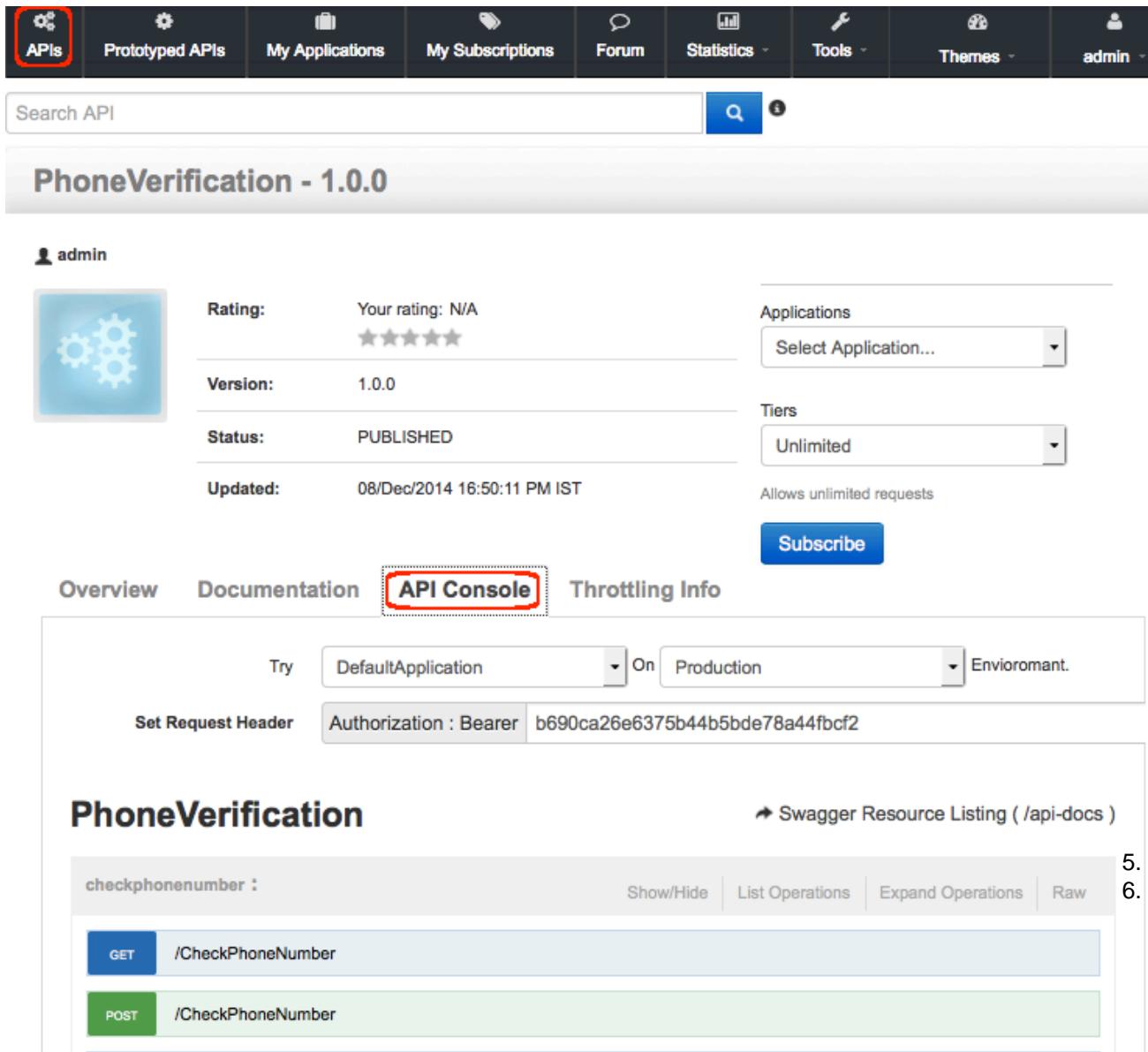
Allowed Domains
ALL

The domains from which requests are allowed to the APIs. Leave empty or enter "ALL" to allow all domains.

Keys - Sandbox

Sandbox keys are not yet generated for this application.

4: Click the **APIs** menu in the API Store and then click on the API that you want to invoke. When the API opens go to its **API Console** tab



The screenshot shows the WSO2 API Manager interface. The top navigation bar includes links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Tools, Themes, and admin. A search bar is at the top. The main content area displays the 'PhoneVerification - 1.0.0' API. On the left, there is a thumbnail image of three gears. The API details are as follows:

- Rating:** Your rating: N/A
- Version:** 1.0.0
- Status:** PUBLISHED
- Updated:** 08/Dec/2014 16:50:11 PM IST

On the right, there are dropdown menus for Applications (Select Application...), Tiers (Unlimited), and a note that says "Allows unlimited requests". A blue 'Subscribe' button is present. Below the API details, there are tabs: Overview, Documentation, API Console (which is selected and highlighted with a dashed border), and Throttling Info. Under the API Console tab, there are dropdowns for 'Try' (DefaultApplication) and 'On' (Production) environments, and an 'Enviromant.' dropdown. A 'Set Request Header' field contains 'Authorization : Bearer b690ca26e6375b44b5bde78a44fbef2'. The main content area shows the 'PhoneVerification' resource with two operations: a GET method for '/CheckPhoneNumber' and a POST method for '/CheckPhoneNumber'.

5. Expand the GET method, provide the required parameters and click **Try it Out**. For example,

PhoneNumber	E.g., 18006785432
LicenseKey	Give 0 for testing purpose
Authorization	<p>The API console is automatically populated by the access token that you generated in step 3 after subscribing to the API.</p> <p>The token is prefixed by the string "Bearer" as per the OAuth bearer token profile. OAuth security is enforced on all published APIs. If the application key is invalid, you get a 401 Unauthorized response in return.</p>

Base URL	<p>Appears at the bottom of the console. Using the base URL and the parameters, the system creates the API URL in the form <code>http://<host_name>:8280/<context>/<version>/<Resource, if any></code><backend service requirements included as parameters, if any>. For example, <code>http://localhost:8280/phoneverify/1.0.0/CheckPhoneNumber</code>. /phoneverify is the context, 1.0.0 is the version and CheckPhoneNumber is the resource.</p> <p> If you cannot invoke the API's HTTPS endpoint (causes the SSLPeerUnverified exception), it could be because the security certificate issued by the server is not trusted by your browser. To resolve this issue, access the HTTPS endpoint directly from your browser and accept the security certificate.</p> <p> Tip: If HTTP invocation is blocked in your corporate environment, you have to change the base path of the API to its HTTPS endpoint in the Swagger definition. This ensures that API invocations do not fail as a result of browsers blocking HTTP calls within HTTPS sessions.</p> <p> Tip: Your API's resource must have the OPTIONS method selected to allow subscribers to invoke the API using the API Console. For the resources that have methods requiring authentication (i.e., Auth Type is not NONE), you set None as the Auth type of OPTIONS to support CORS (Cross Origin Resource Sharing) between the API Store and Gateway.</p>
-----------------	---

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	(required)	Request Body	body	string

Parameter content type:

application/json

PhoneNumber	18006785432	Give the phone number to be validated
LicenseKey	0	Give the license key as 0 for testing purpose

Try it out!

POST /CheckPhoneNumber

[BASE URL: https://[REDACTED]:9443/store/api-docs/admin/PhoneVerification/1.0.0 , API VERSION: 1.0.0]

6. Note the response for the API invocation. As we used a valid phone number in this example, the response is valid.

Response Body

```

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

```

Response Code

200

Response Headers

Pragma: no-cache
 Content-Type: text/xml; charset=utf-8
 Cache-Control: no-cache
 Expires: -1

Create and Publish an API

API URL	<p>To get the URL, go to the API's Overview tab in the API Store. The URL takes the form <code>http://280/<context>/<version>/<Resource, if any><back end service requirement parameters, if any></code>. For example, <code>http://gateway.api.cloud.wso2.com:8280/t/yneverify/1.0.0/CheckPhoneNumber</code> where /phoneverify is the context, 1.0.0 is the version and CheckPhoneNumber is the resource.</p> <p>As you are going to make an HTTP GET call in this tutorial, append the payload to the URL. For example, <code>http://gateway.api.cloud.wso2.com:8280/t/yashiracom/phoneverify/1.0.0/CheckPhoneNumber=18006785432&LicenseKey=0</code>.</p>
Header	<p>Authorization:Bearer <give the access token that you generated E.g, Authorization:Bearer U9znDo4OSYPfzoW16S2puHmKahga</p> <p>OAuth security is enforced on all published APIs. Consumers must send the credentials (application and consumer) as per the OAuth bearer token profile. If not, you receive a 401 Unauthorized response in return.</p>

RESTClient**Request**

GET Send

Headers

Raw Form

Authorization: Bearer 3b470e8c41b3e6d68afea4d38f5e627.

8. Click the **Send** button to invoke the API. The response appears in the console. As we used a valid phone number in this example, the response returns as valid.

RESTClient**Request**

GET Send

Headers

Raw Form

Authorization: Bearer 3b470e8c41b3e6d68afea4d38f5e627.

Response

[Response Body](#) [Response Headers](#)

```
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://ws.cdyne.com/PhoneVerify/query">
<Company>Toll Free</Company>
<Valid>true</Valid>
<Use>Assigned to a code holder for normal use.</Use>
<State>TF</State>
<RC />
<OCN />
<OriginalNumber>8006333469</OriginalNumber>
<CleanNumber>8006333469</CleanNumber>
<SwitchName />
<SwitchType />
<Country>United States</Country>
<CLLI />
<PrefixType>Landline</PrefixType>
<LATA />
<sms>Landline</sms>
```

9. Within a minute after the first API invocation, make another attempt to invoke the API.

10. Note that you get a throttling error. This is because you applied a Bronze tier at the time you subscribed to the API and the Bronze tier only allows one API call per minute.

Response

[Response Body](#) [Response Headers](#)

```
<amt:Fault xmlns:amt="http://wso2.org/apimanager/throttling"><amt:Code>900800</amt:Code><amt:Message>Message Throttled Out</amt:Message><amt:Description>You have exceeded your quota</amt:Description></amt:Fault>
```

You have invoked an API using the REST client integrated in the API Store.

Use the Community Features

The API Store provides several useful features to build and nurture an active community of users for your APIs. This is required to advertise APIs, learn user requirements and market trends.

Let's see what community features are available in the API Store:

- [Use the search facility](#)
- [Rate and comment](#)
- [Share on social media/e-mail](#)
- [Embed an API widget](#)
- [Participate in the forum](#)

Use the search facility

You can search for APIs in the API Publisher or Store in the following ways:

Clause	Syntax
By the API's name	As this is the default option, simply enter the API's name and search.
By API the API provider	provider:xxxx. For example, provider:admin Provider is the user who created the API.
By the API version	version:xxxx. For example, version:1.0.0 A version is given to an API at the time it is created.
By the context	context:xxxx. For example, context:/phoneregister Context is the URL context of the API that is specified as /<context_name> at the time the API is created.
By the API's status	status:xxxx. For example, status:PUBLISHED A state is any stage of an API's lifecycle. The default lifecycle stages include created, prototyped, published, deprecated, retired and blocked.
By description	description:xxxx A description can be given to an API at the time it is created or later. There can be APIs without descriptions as this parameter is optional.
By the subcontext	subcontext:xxxx. For example, subcontext:/checkphonenumbers A subcontext is the URL pattern of any resource of the API. API resources are created at the time the API is created or later when it is modified. For example, if you create a resource by the name checkphonenumbers, then /checkphonenumbers becomes one subcontext of the API.

By the content of the API documentation

doc:xxxx

You can create API documentation in-line (using the API Publisher UI itself), by uploading a file or referring to an external URL. This search enables you to give a sentence or word phrase that is inside the in-line documentation and find the API that the documentation is added for.

Rate and comment

Rates and comments give useful insights to potential API consumers on the quality and usefulness of an API. You can rate and comment on APIs per each version.

1. Log in to the API Store and click on a published API.
2. The API's **Overview** page opens. Note the rating and commenting options there:

PhoneVerification - 1.0.0

 admin



Rating:		Your rating: 5/5
Version:	1.0.0	
Status:	PUBLISHED	
Updated:	04/Sep/2014 10:02:30 AM IST	

Overview **Documentation** **API Console** **Throttling Info** **Forum**

Production and Sandbox URLs:

http://[REDACTED]:8280/phoneverify/1.0.0

https://[REDACTED]:8243/phoneverify/1.0.0

Share:

[Social Sites](#) [Embed](#) [Email](#)



Comments:

Characters left: 450

[REDACTED]

3. Add a rating and a comment. Note that the comments appear sorted by the time they were entered, alongside the author's name.

Share on social media/e-mail

1. Log in to the API Store and click on a published API.
2. On the API's **Overview** page, you get the social media options using which you can share and advertize APIs.

PhoneVerification - 1.0.0

👤 admin



Rating:	Your rating: N/A
	★★★★★
Version:	1.0.0
Status:	PUBLISHED

Updated: 14/Aug/2014 14:19:47 PM IST

Overview

Documentation

API Console

Throttling Info

Forum

Production and Sandbox URLs:

[http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)

Share:

Social Sites Embed Email



Embed an API widget

A widget is an embeddable version of the API in HTML that you can share on your Website or other Web pages. This is similar to how Youtube videos can be embedded in a Web page.

1. Log in to the API Store and click on a published API.
2. Note the Embed tab under the API's sharing options.

PhoneVerification - 1.0.0

admin



Rating:	Your rating: N/A
	★★★★★
Version:	1.0.0
Status:	PUBLISHED
Updated:	14/Aug/2014 14:19:47 PM IST

Overview

Documentation

API Console

Throttling Info

Forum

Production and Sandbox URLs:

[http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

[https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)

Share:

Social Sites

Embed

Email

```
<iframe width="450" height="120" src="https://localhost:9443/store/apis/widget?name=PhoneVerification&version=1.0.0&provider=admin" frameborder="0" allowfullscreen></iframe>
```

Participate in the forum

1. Log in to the API Store.
2. Click the **Forum** tab or menu to go to the forum where you can initiate conversations and share your opinions with other users.

PhoneVerification - 1.0.0

admin



Rating:	Your rating: N/A
	★★★★★
Version:	1.0.0
Status:	PUBLISHED
Updated:	04/Sep/2014 13:03:10 PM IST

Applications

Select Application...

Tiers

Unlimited

Allows unlimited requests

[Subscribe](#)

[Overview](#)

[Documentation](#)

[API Console](#)

[Throttling Info](#)

[Forum](#)

[Create New Topic](#)



Topic

Replies

[When do we get the next version of this API?](#)

0

By : admin

Invoke an API using a SOAP Client

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



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

- Applications
- Throttling
- Access tokens

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

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

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

PhoneVerification - 1.0.0

admin

Rating: Your rating: N/A

Version: 1.0.0

Status: PUBLISHED

Updated: 14/Aug/2014 14:19:47 PM IST

Applications: DefaultApplication

Tiers: Bronze

Allows 1 request(s) per minute.

Subscribe

Overview Documentation API Console Throttling Info Forum

3. Go to the **My Subscriptions** page and generate a production key to the default application using which you subscribed to the API.

Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times

Applications With Subscriptions

DefaultApplication

Access Token: U9znDo4OSYPfzoW16S2puHmKahga

Re-generate Token Validity: 3600 Seconds

Consumer Key: LQuF3UNG5AlIJYoUCfKN8Gz_bWEa

Consumer Secret: Vn2hCdAOhRbbGspmnCKWlfcn08oa

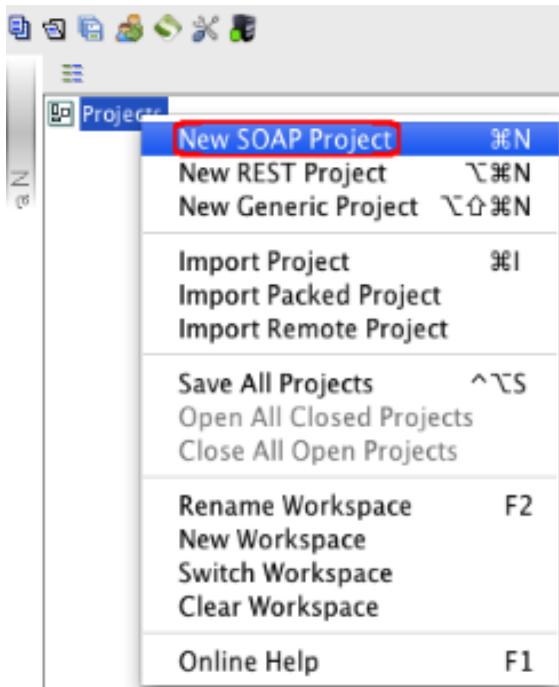
Allowed Domains: ALL

Leave empty or filling w
Update
Domains

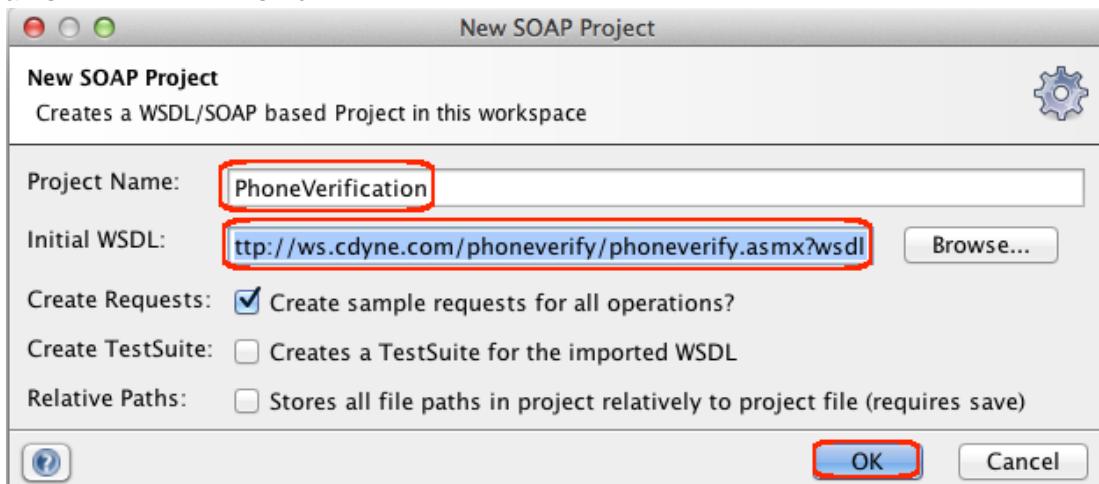
4. Copy the access token to the clipboard as you need it later to invoke the API.

5. Download the SOAP UI installation that suits your operating system from <http://www.soapui.org/> and open its console.

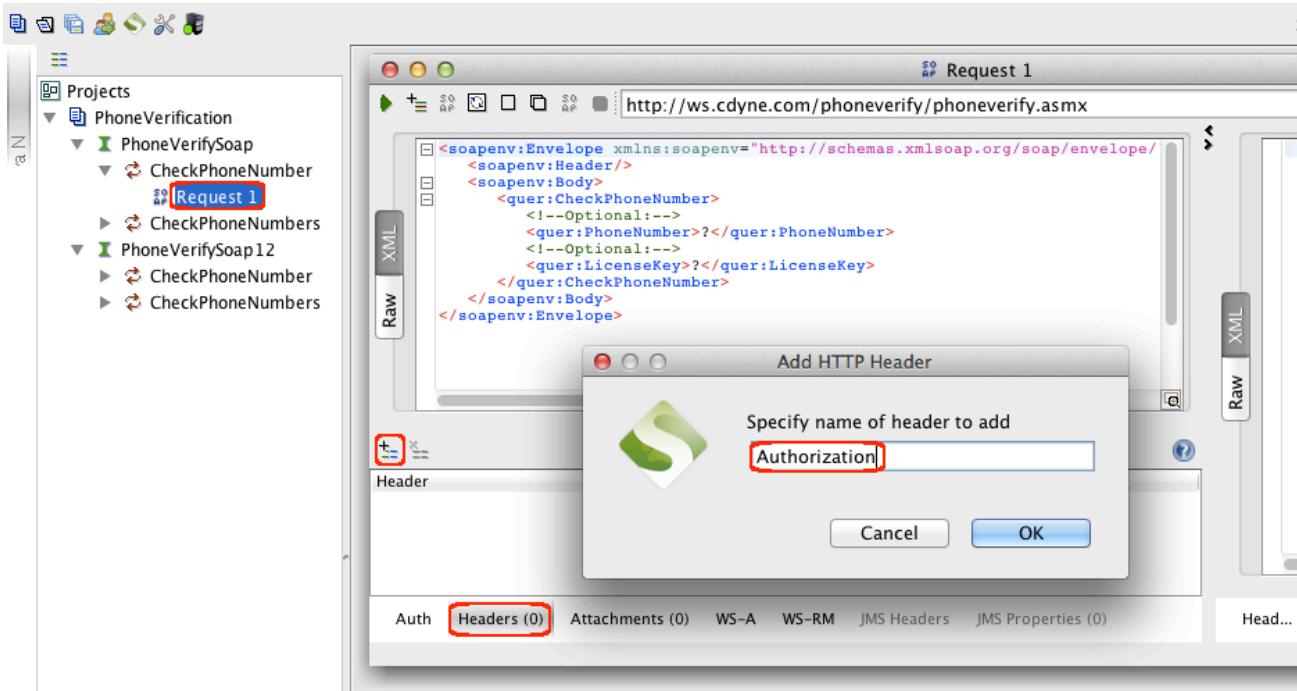
6. In the SOAP UI, right click on the **Projects** menu and create a new SOAP project.



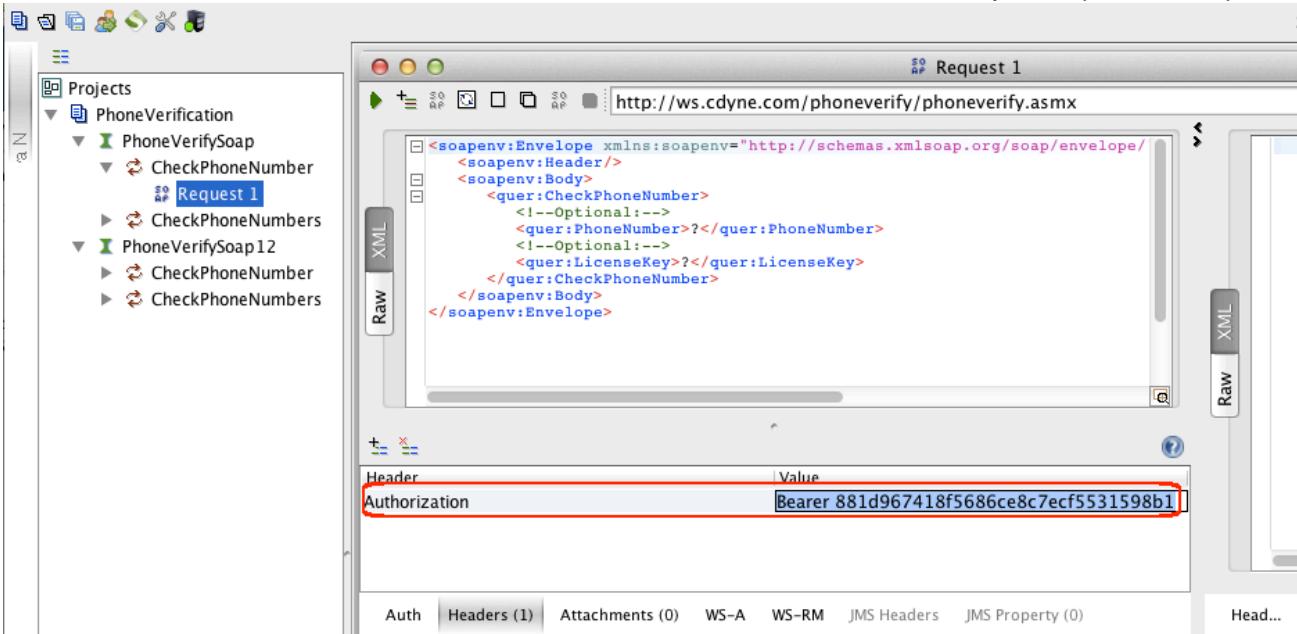
7. Give your API's WSDL and click **OK**. In this case, the WSDL is <http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl>.



8. The WSDL defines two operations. Let's work with **CheckPhoneNumber**. Double click on Request 1. Then, add an authorization header to your request by clicking the add sign on the **Header** tab of the console.

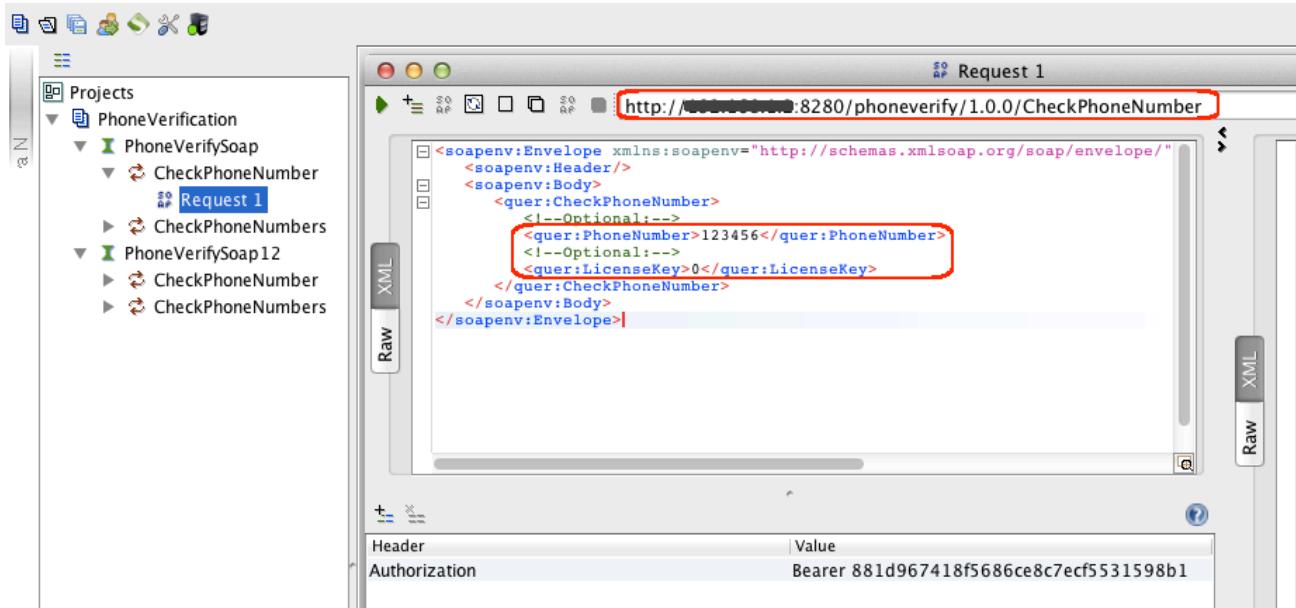


9. Give the value of the Authorization header as 'Bearer <the access token you copied in step 5>.

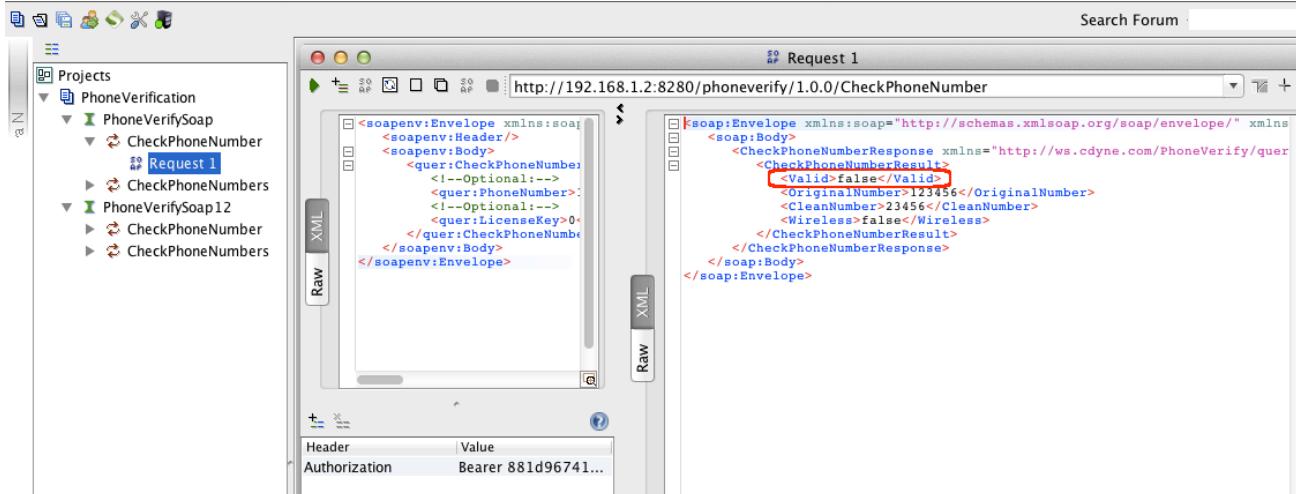


10. Add the following values and submit the request:

- Change the endpoint with the production URL of the API. You can copy the production URL from the API's **Overview** tab in the API Store. Append the resources to the end of the URL, if any. The resource is `/CheckPhoneNumber` for the PhoneVerification API that we use here.
- In the SOAP request, change the parameters, which are `PhoneNumber` and `LicenseKey`. Let's give any dummy phone number and 0 as the license key



11. Note the result on the right-hand side panel. As you gave a dummy phone number in this example, you get the result as invalid.



You have invoked an API using a SOAP client.

Configuring the API Manager

This section covers the following

- Customizing the API Store
- Configuring Multiple Tenants
- Adding Internationalization and Localization
- Configuring Single Sign-on with SAML2
- Changing the Default Transport
- Configuring Caching
- Working with Databases
- Managing Users and Roles
- Configuring User Stores
- Directing the Root Context to the API Store
- Adding Links to Navigate Between the Store and Publisher
- Maintaining Separate Production and Sandbox Gateways
- Configuring Transports

Customizing the API Store

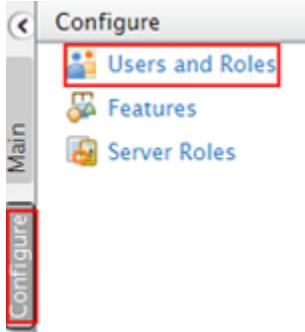
You can customize the API Store in the following ways:

- Enabling or disable self signup
- Changing the theme
- Changing language settings
- Setting single login for all apps
- Categorizing APIs

Enabling or disable self signup

In a multi-tenanted API Manager setup, self signup to the API Store is disabled by default to all tenants except the super tenant. A tenant admin can enable it as follows:

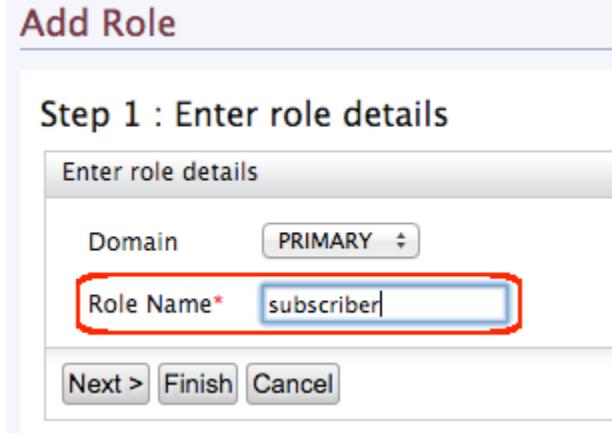
1. Log in to the management console (<https://<HostName>:9443/carbon>) as admin (or tenant admin).
2. Click the **Configure** -> **Users and Roles** menu.



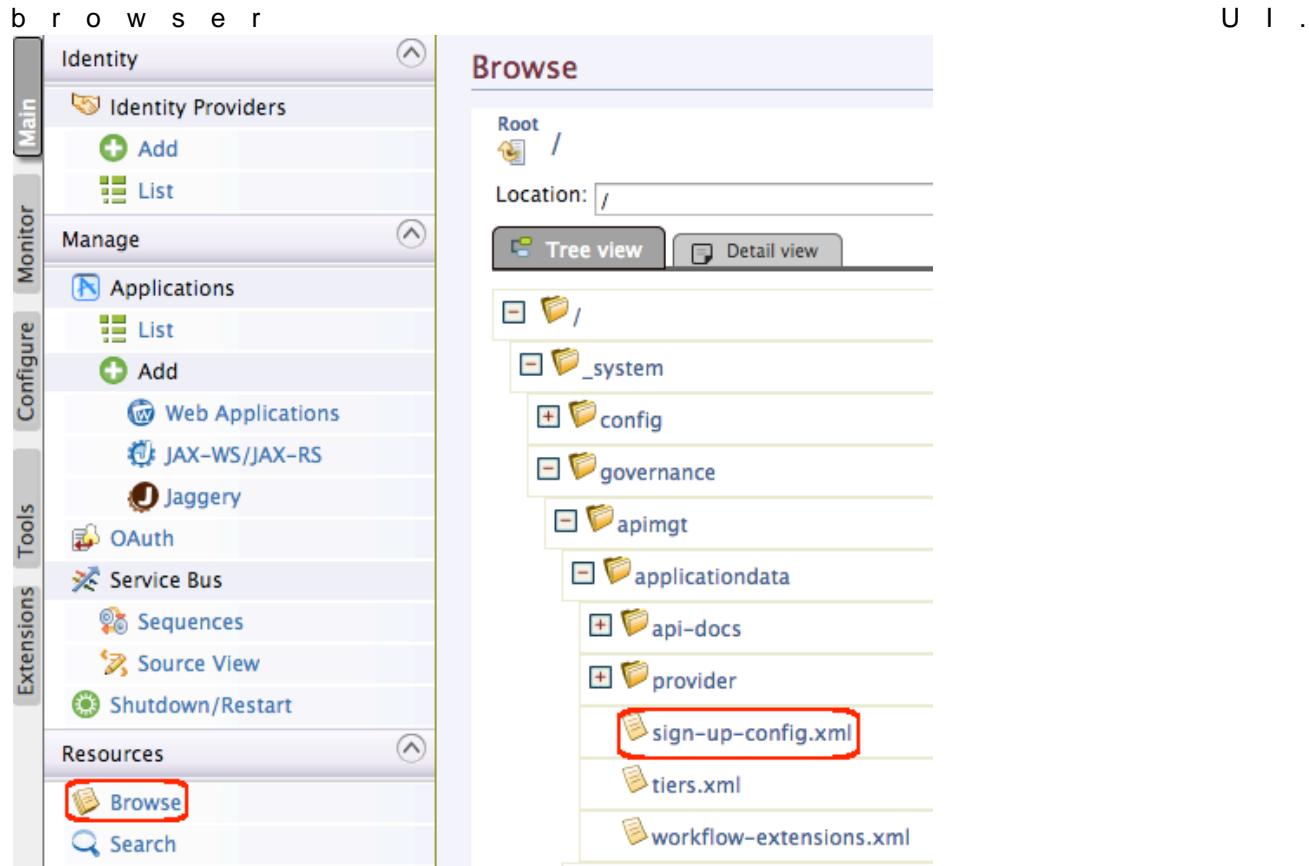
3. In the **User Management** page that opens, click **Roles**.



4. Add a role by the name **subscriber** (or any other name you prefer) and the following permissions:
 - Login
 - Manage > API > Subscribe



5. Go to the **Resources** -> **Browse** menu.
6. Load `/_system/governance/apimgt/applicationdata/sign-up-config` resource in the registry



7. Do the following changes in the signup configuration and save.

- Set <EnableSignup> to true
- Set <RoleName> to subscriber and <IsExternalRole> to true. Note that you should have the subscriber role created at this point.
- Set <AdminUserName> and password to the tenant admin's username and password.

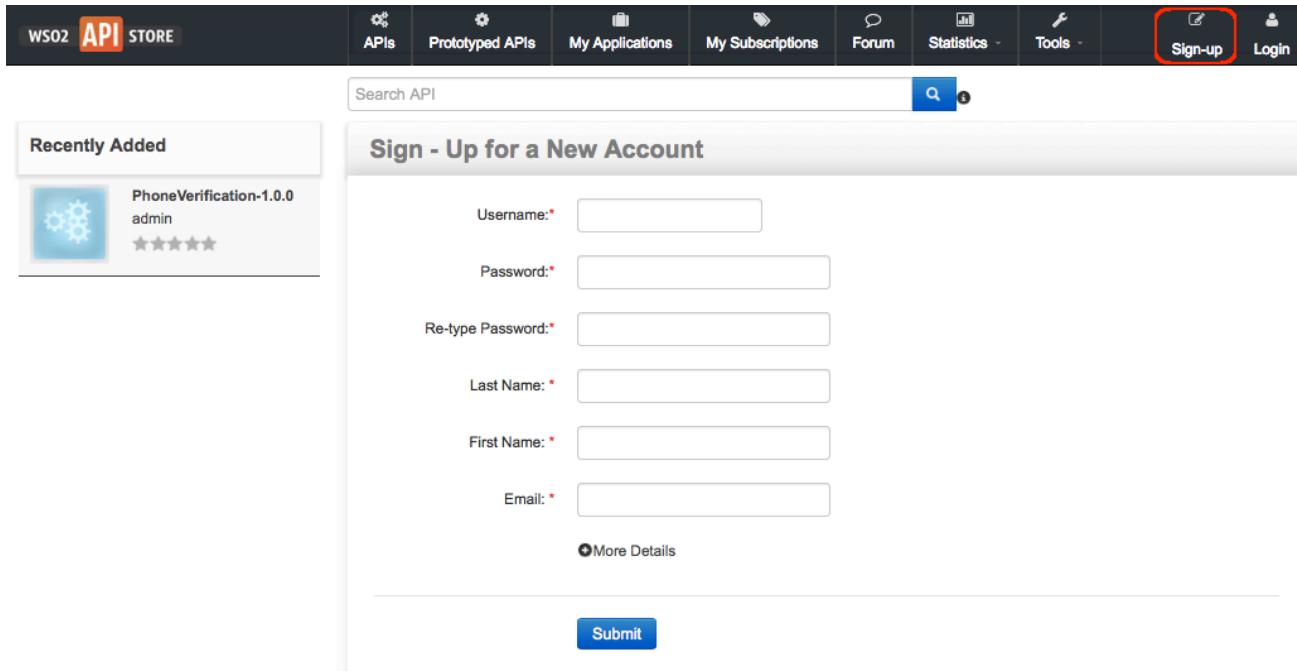
```

<SelfSignUp>
    <EnableSignup>true</EnableSignup>
    <!-- user storage to store users -->
    <SignUpDomain>PRIMARY</SignUpDomain>
    <!-- Tenant admin information. (for clustered setup credentials for
AuthManager) -->
    <AdminUserName>xxxx</AdminUserName>
    <AdminPassword>xxxx</AdminPassword>
    <!-- List of roles for the tenant user -->
    <SignUpRoles>
        <SignUpRole>
            <RoleName>subscriber</RoleName>
            <IsExternalRole>true</IsExternalRole>
        </SignUpRole>
    </SignUpRoles>
</SelfSignUp>

```

8. Open the API Store (<https://<HostName>:9443/store>.)

9. Note the **Sign-up** link that appears in the top, right-hand corner of the window.



The screenshot shows the WSO2 API Manager API Store interface. At the top, there is a navigation bar with links for APIs, Prototyped APIs, My Applications, My Subscriptions, Forum, Statistics, Tools, Sign-up (which is highlighted with a red box), and Login. Below the navigation bar is a search bar labeled 'Search API' with a magnifying glass icon. To the right of the search bar is a user icon. On the left, there is a sidebar titled 'Recently Added' showing a list of APIs: 'PhoneVerification-1.0.0' by 'admin' with a 5-star rating. The main content area is titled 'Sign - Up for a New Account' and contains fields for Username, Password, Re-type Password, Last Name, First Name, and Email. There is also a 'More Details' link and a 'Submit' button at the bottom.

10. To disable the self signup capability, navigate to `/_system/governance/apimgt/applicationdata/signup-config.xml` in the registry again and set the `<SelfSignUp><EnableSignup>` element to false.



Tip: To engage your own signup process, see [Adding a User Signup Workflow](#).

Changing the theme

See [Adding a new API Store Theme](#).

Changing language settings

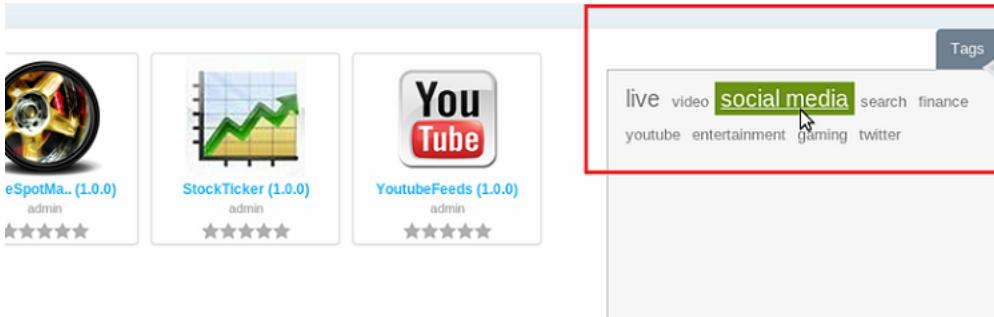
To change the language of the API Store, see [Adding Internationalization and Localization](#).

Setting single login for all apps

Single sign-on (SSO) allows users who are logged in to one application to automatically log in to multiple other applications using the same credentials. They do not have to repeatedly authenticate themselves. To configure, see [Configuring Single Sign-on with SAML2](#).

Categorizing APIs

API providers add tags to APIs when designing them using the API Publisher. Tags allow API providers to categorise APIs that have similar attributes. Once a tagged API gets published to the API Store, its tags appear as clickable links to the API consumers, who can use them to quickly jump to a category of interest.



The screenshot shows the WSO2 API Manager API Store interface. On the left, there is a list of APIs: 'eSpotMa.. (1.0.0)' by 'admin' with a 5-star rating, 'StockTicker (1.0.0)' by 'admin' with a 5-star rating, and 'YouTubeFeeds (1.0.0)' by 'admin' with a 5-star rating. The 'YouTubeFeeds' card is highlighted with a red box. In the 'Tags' section of this card, the 'social media' tag is highlighted with a red box. Other tags listed are live, video, search, finance, youtube, entertainment, gaming, and twitter.

If you want to see the APIs grouped according to different topics in the API Store, do the following:

1. Go to `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf` directory,

open the `site.json` file and set the `tagWiseMode` attribute as true.

2. Go to the API Publisher and add tags with the suffix "-group" to APIs (e.g., Workflow APIs-group, Integration APIs-group, Quote APIs-group.)
3. Restart the server.

After you publish the APIs, you see the APIs listed under their groups. You can click on a group to check what the APIs are inside it.

APIs groups

Integration APIs	Invoice APIs	Quote APIs	Workflow APIs
Lorem ipsum dolor sit amet, consectetur adipiscing elit.			

Configuring Multiple Tenants

The goal of multitenancy is to maximize resource sharing by allowing multiple users (tenants) to log in and use a single sever/cluster at the same time, in a tenant-isolated manner. That is, each user is given the experience of using his/her own server, rather than a shared environment. Multitenancy ensures optimal performance of the system's resources such as memory and hardware and also secures each tenant's personal data.

You can register tenant domains using the Management Console of WSO2 products.

This section covers the following topics:

- [Multi Tenant Architecture](#)
- [Managing Tenants](#)
- [Tenant-Aware Load Balancing using WSO2 ELB](#)

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:

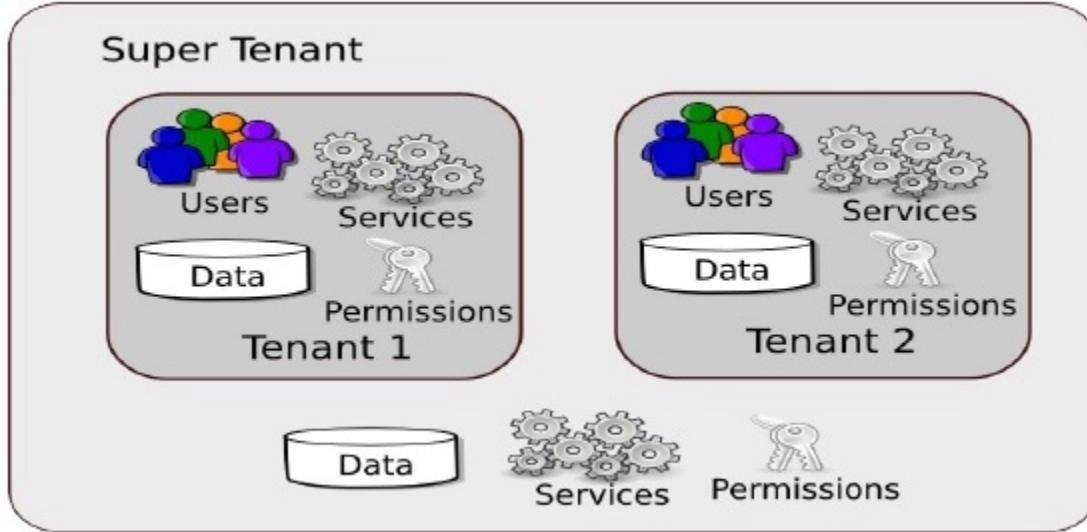
- **Tenant isolation:** Each tenant has its own domain, which the other tenants cannot access.
- **Data isolation:** Each tenant can manage its data securely, in an isolated manner.
- **Execution isolation:** Each tenant can carry out business processes and workflows independent of the other tenants. 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

The super tenant is the complete server space of a WSO2 product instance. Separate spaces within this server space are allocated to individual tenants.

The super tenant as well as each individual tenant has its own configuration and context module.

Each tenant has its own security domain. A domain has a set of users, and permissions for those users to access resources. Thus, a tenant is restricted by the users and permissions of the domain assigned to it. The artifact repositories of the tenants are separated from each other.



An individual tenant can carry out the following activities within the boundaries of its own configuration and context module:

- Deploying artifacts
- Applying security
- User management
- Data management
- Request throttling
- Response caching

WSO2 Carbon provides a number of Admin services which have special privileges to manage the server. These admin services are deployed in the super tenant. Other tenants can make use of these admin services to manage their deployment. The admin services operate in a tenant aware fashion. Thus, privileges and restrictions that apply to any client using an admin service are taken into account.

Resource sharing

WSO2 Carbon supports the following methods for sharing resources among tenants:

- **Private Jet mode:** This method allows the load of a tenant ID to be deployed in a single tenant mode. A single tenant is allocated an entire service cluster. The purpose of this approach is to allow special privileges (such as priority processing and improved performance) to a tenant.
- **Separation at hardware level:** This method allows different tenants to share a common set of resources, but each tenant has to run its own operating system. This approach helps to achieve a high level of isolation, but it also incurs a high overhead cost.
- **Separation at JVM level:** This method allows tenants to share the same operating system. This is done by enabling each tenant to run a separate JVM instance in the operating system.
- **Native multitenancy:** This method involves allowing all the tenants to share a single JVM instance. This method minimises the overhead cost.

Lazy loading

Lazy loading is a design pattern used specifically in cloud deployments to prolong the initialization of an object or artifact until it is requested by a tenant or an internal process.

Tenants

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

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

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

Artifacts

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

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

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

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

Restrictions

The following restrictions are imposed to ensure that each individual tenant has the required level of isolation and maintains fine grained security control over its own services without affecting the other tenants.

- Only the super tenant can modify its own configuration. In addition, it can add, view and delete tenants.
- When a tenant logs into the system, it can only access artifacts deployed under its own configuration. One tenant cannot manipulate the code of another tenant.
- The super admin or tenant admin can add user stores to their own domain. Dynamic configurations are possible only for secondary user stores and the primary user store is not configurable at run time. This is because primary user stores are available for all tenants and allowing changes to the configuration at run time can lead to instability of the system. Therefore, the primary user store is treated as a static property in the implementation and it should be configured prior to run time.
- A tenant's 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. See [Viewing Handlers in Message Flows](#) for further information on message handlers and dispatchers.

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/foocom/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. See [Tenant Aware Load Balancing Using the WSO2 Elastic Load Balancer](#) for further information.

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.

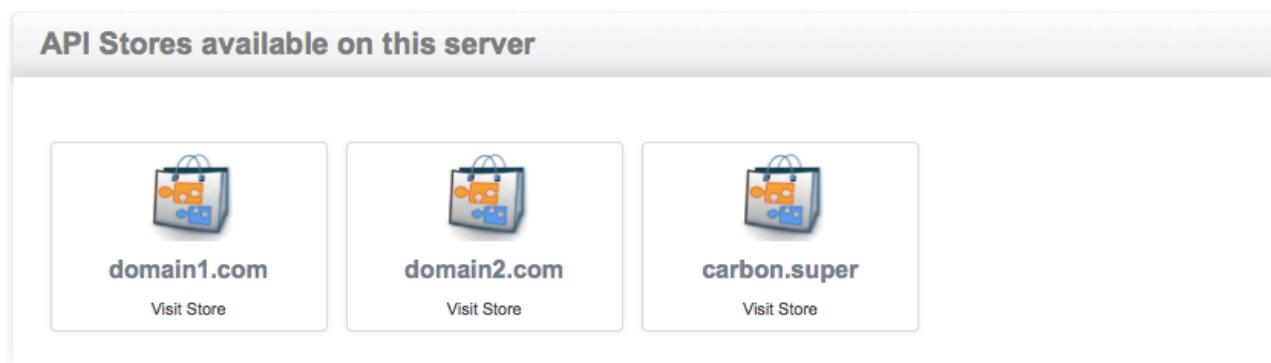
First Name/Last Name	The name of the tenant admin.
Admin Username	The login username of the tenant admin. The username always ends with the domain name (e.g., admin@abc.com)
Admin Password	The password used to log in using the admin username specified.
Admin Password (Repeat)	Repeat the password to confirm.
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**.

Enter the Tenant Domain	<input type="button" value="Find"/>															
Tenants List																
<table border="1"> <thead> <tr> <th>Domain</th> <th>Email</th> <th>Created Date</th> <th>Active</th> <th>Edit</th> </tr> </thead> <tbody> <tr> <td>wso2.com</td> <td>frankie.avalon@gmail.com</td> <td>2014/11/17 12:03:06</td> <td><input checked="" type="checkbox"/></td> <td>Edit</td> </tr> <tr> <td>abc.com</td> <td>dean.martin@gmail.com</td> <td>2014/11/17 13:43:46</td> <td><input checked="" type="checkbox"/></td> <td>Edit</td> </tr> </tbody> </table>		Domain	Email	Created Date	Active	Edit	wso2.com	frankie.avalon@gmail.com	2014/11/17 12:03:06	<input checked="" type="checkbox"/>	Edit	abc.com	dean.martin@gmail.com	2014/11/17 13:43:46	<input checked="" type="checkbox"/>	Edit
Domain	Email	Created Date	Active	Edit												
wso2.com	frankie.avalon@gmail.com	2014/11/17 12:03:06	<input checked="" type="checkbox"/>	Edit												
abc.com	dean.martin@gmail.com	2014/11/17 13:43:46	<input checked="" type="checkbox"/>	Edit												

When you create multiple tenants in an API Manager deployment, the API Stores of each tenant are displayed in a multi-tenanted view for all users to browse and permitted users to subscribe to as shown below:

1. Access the API Store URL (by default, `https://localhost:9443/store`) using a Web browser. You see the storefronts of all the registered tenant domains listed there. For example,



This is called the public store. Each icon here is linked to the API Store of a registered tenant, including the super tenant, which is `carbon.super`. That is, the super tenant is also considered a tenant.

2. Click the **Visit Store** link associated with a given store to open it.
 3. Anonymous users can browse all stores and all public APIs that are published to them. However, in order to subscribe to an API, the user must log in.

For example, if you are a user in the `domain1.com` tenant domain,

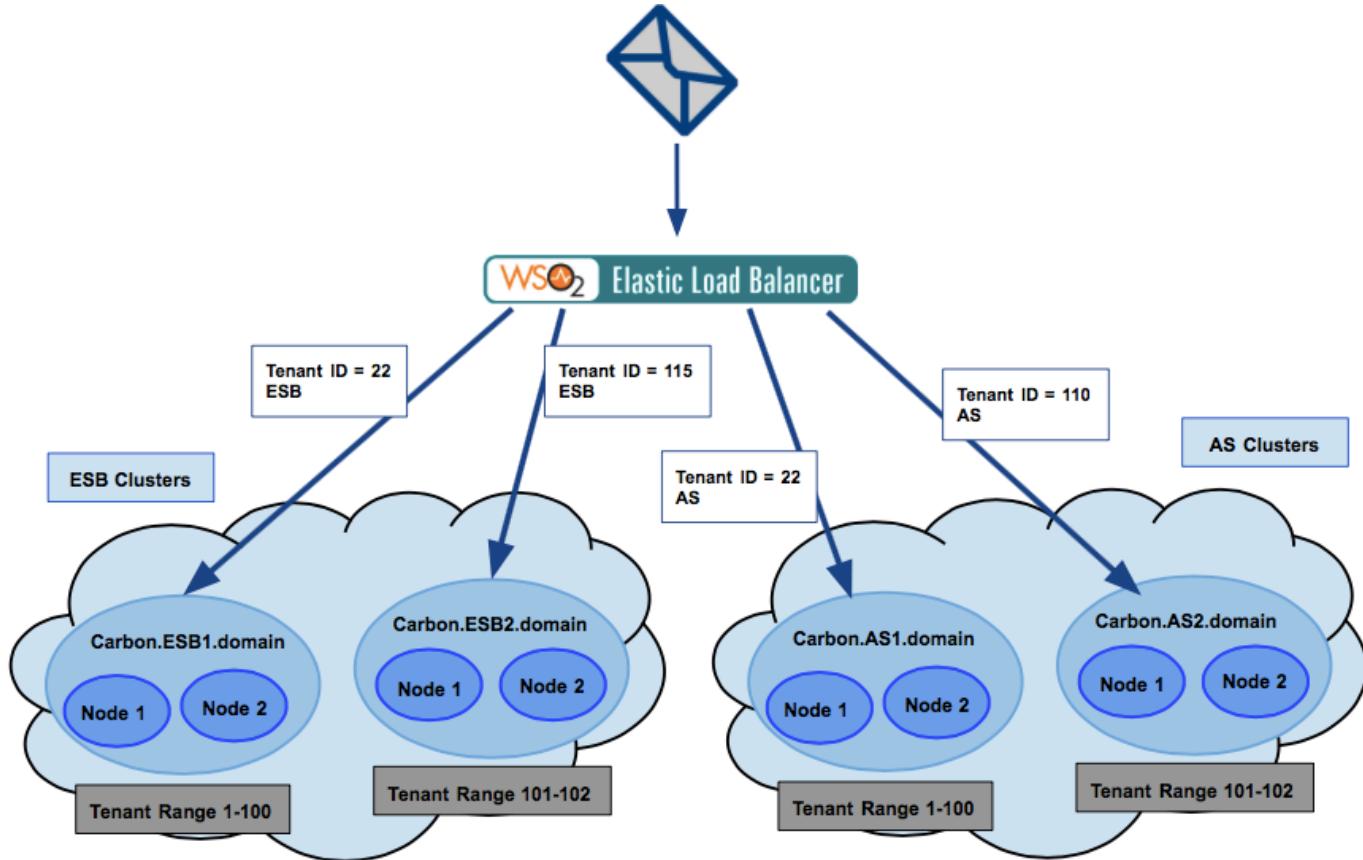
- You can access the public store (`https://localhost:9443/store`), go to the `domain1.com` store, log in to it and subscribe to its APIs.
- You can also browse the other tenant stores listed in the public store. But, within other tenant stores, you can only subscribe to the APIs to which your tenant domain is permitted to subscribe to. At the

time an API is created, the API creator can specify which tenants are allowed to subscribe to the API . For information, see [API Subscriptions](#).

Tenant-Aware Load Balancing using WSO2 ELB

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 `ELB_HOME/repository/conf/loa dbalancer.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.

```

appserver {
# multiple hosts should be separated by a comma.
hosts appserver.cloud-test.wso2.com;

domains {
carbon.as1.domain {
tenant_range 1-100;
}
carbon.as2.domain {
tenant_range 101-200;
}
}
}
}

```

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.

Adding Internationalization and Localization

The API Manager comes with two Web interfaces as API Publisher and API Store. The following steps show an example of how to localize the API Publisher UI. Same instructions apply to localize the API Store.

Changing the browser settings

1. Follow the instructions in your Web browser's user guide and set the browser's language to a preferred one. For example, in Google Chrome, you set the language using the **Settings -> Show advanced settings -> Languages** menu.
2. Set the browser's encoding type to UTF-8.

Introduction to resource files

3. Go to `<AM_HOME>/repository/deployment/server/jaggeryapps/publisher` directory where `<AM_HOME>` is the API Manager distribution's home.
4. There are two types of resource files used to define localization strings in the API Manager.
 - The resource file used to store the strings defined in `.jag` files according to browser locale (For example, `locale_en.json`) is located in `.../publisher/site/conf/locales/jaggery` folder.
 - The resource file `i18nResources.json`, which is used to store strings defined in client-side javascript files such as pop-up messages when a UI event is triggered, is located in `.../publisher/site/conf/locales/js` folder.

For example,

```

./locales/jaggery:
locale_en.json

./locales/js:
i18nResources.json

```

To implement localization support for jaggery, we use its in-built script module 'i18n'. For more information, refer to <http://jaggeryjs.org/apidocs/i18n.jag>.

Localizing strings in Jaggery files

- To localize the API publisher to Spanish, first localize the strings defined in jaggery files. Create a new file by the name `locale_{localeCode}.json` inside `...publisher/site/conf/locales/jaggery` folder. For example, if the language set in the browser is Spanish, the locale code is `es` and the file name should be `locale_es.json`.
- Add the key-value pairs to `locale_es.json` file. For an example on adding key value pairs, refer to `locale_en.json` file in `...publisher/site/conf/locales/jaggery` folder. It is the default resource file for jaggery.

In addition, a section of a sample `locale_es.json` file is shown below for your reference.

```
{
  "name" : "Nombre",
  "context" : "Contexto",
  "version" : "Versión",
  "description" : "Descripción",
  "visibility" : "Visibilidad",
  "thumbnail" : "Uña del pulgar",
  "endpoint" : "Producción URL",
  "sandbox" : "Cajón de arena URL",
  ...
}
```

Localizing strings in client-side Javascript files

- To localize the javascript UI messages, navigate to `publisher/site/conf/locales/js` folder and update `i18nResources.json` file with relevant values for the key strings.
- Once done, open the API Publisher web application in your browser (`https://<YourHostName>:9443/publisher`).
- Note that the UI is now changed to Spanish.

Configuring Single Sign-on with SAML2

Single sign-on (SSO) allows users, who are authenticated against one application, gain access to multiple other related applications as well without having to repeatedly authenticate themselves. It also allows the Web applications gain access to a set of back-end services with the logged-in user's access rights, and the back-end services can authorize the user based on different claims like user role.

WSO2 API Manager includes **Single Sign-On with SAML 2.0** feature, which is implemented according to the SAML 2.0 Web browser-based SSO support that is facilitated by WSO2 Identity Server (IS). This feature is available in any IS version from 4.1.0 onwards. We use **IS 5.0.0** in this guide. WSO2 Identity Server acts as an identity service provider of systems enabled with single sign-on, while the Web applications such as API Manager apps act as SSO service providers. Using this feature, you can configure SSO across the two API Manager Web applications, which are API Publisher and API Store as well as other Web applications in your organization. After configuring, you will be able to access API Store or API Publisher in a single authentication attempt.

i To learn more about Single Sign-On with WSO2 Identity Server, refer the following article on WSO2 library: <http://wso2.org/library/articles/2010/07/saml2-web-browser-based-sso-wso2-identity-server>

The topics below explain the configurations:

- Sharing the user store
- Sharing the registry space
- Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider
- Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

Sharing the user store

First, point both WSO2 IS and WSO2 API Manager to a single user store using the instructions given in section [Configuring User Stores](#). You do this to make sure that a user who tries to log in to the API Manager console, the API Store or the Publisher is authorized. When a user tries to log in to either of the three applications, s/he is redirected to the configured identity provider (WSO2 IS in this case) where s/he provides the login credentials to be authenticated. In addition to this, the user should also be authorized by the system as some user roles do not have permission to perform certain actions. For the purpose of authorization, the IS and API Manager need to have a

shared user store and user management database (by default, this is the H2 database in the <APIM_HOME>/repository/conf/user-mgt.xml file) where the user's role and permissions are stored.

For example, let's take a common JDBC user store (MySQL) for both IS and API Manager.

1. Create a MySQL database (e.g., 410_um_db) and run the <AM_HOME>/dbscripts/mysql.sql script on it to create the required tables. If you are using a different database type, find the relevant script from the <AM_HOME>/dbscripts directory.
2. Open <AM_HOME>/repository/conf/master-datasources.xml file and add the datasource configuration for the database that you use for the shared user store and user management information. For example,

```
<datasource>
    <name>WSO2_UM_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2UMDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/410_um_db</url>
            <username>username</username>
            <password>password</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

3. Add the same datasource configuration above to <IS_HOME>/repository/conf/master-datasources.xml file.
4. Copy the database driver JAR file to the <IS_HOME>/repository/components/lib and <AM_HOME>/repository/components/lib directories.
5. Open <AM_HOME>/repository/conf/user-mgt.xml file. The dataSource property points to the default H2 database. Change it to the jndiConfig name given above (i.e., jdbc/WSO2UMDB). This changes the datasource reference that is pointing to the default H2 database.

```
<Realm>
    <Configuration>
        ...
        <Property name="dataSource">jdbc/WSO2UMDB</Property>
    </Configuration>
    ...
</Realm>
```

6. Add the same configuration above to the <IS_HOME>/repository/conf/user-mgt.xml file.
7. The Identity Server has an embedded LDAP user store by default. As this is enabled by default, follow the instructions in [Internal JDBC User Store Configuration](#) to disable the default LDAP and enable the JDBC user store instead.

Sharing the registry space

In a multi-tenanted environment, by default, the Identity Server uses the key store of the super tenant to sign SAML responses. The API Store and Publishers are already registered as SPs in the super tenant. However, if you want the Identity Server to use the registry key store of the tenant that the user belongs to, you can create a common registry database and mount it on both the IS and the APIM.

1. Create a MySQL database (e.g., registry) and run the `<IS_HOME>/dbscripts/mysql.sql` script on it to create the required tables. If you are using a different database type, find the relevant script from the `<IS_HOME>/dbscripts` directory.
2. Add the following datasource configuration to both the `<IS_HOME>/repository/conf/datasources/master-datasources.xml` and `<AM_HOME>/repository/conf/datasources/master-datasources.xml` files.

```

<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used for registry</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>

            <url>jdbc:mysql://localhost:3306/registry?autoReconnect=true&relaxAutoCommit=true&lt;/url>
            <username>apiuser</username>
            <password>apimanager</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

3. Create the registry mounts by inserting the following sections into the `<IS_HOME>/repository/conf/registry.xml` file.

 When doing this change, do not replace the existing `<dbConfig>` for "wso2registry". Simply add the following configuration to the existing configurations.

```

<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>

<mount path="/_system/governance" overwrite="true">
    <instanceId>gov</instanceId>
    <targetPath>/_system/governance</targetPath>
</mount>

<mount path="/_system/config" overwrite="true">
    <instanceId>gov</instanceId>
    <targetPath>/_system/config</targetPath>
</mount>

```

4. Repeat the above step in the `<AM_HOME>/repository/conf/registry.xml` file as well.
Next, let us look at the SSO configurations.

Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider

1. Start the IS server and log in to its Management Console UI (<https://localhost:9443/carbon>).

 If you use login pages that are hosted externally to log in to the Identity Server, give the absolute URLs of those login pages in the `authenticators.xml` and `application-authenticators.xml` files in the `<IS_HOME>/repository/conf/security` directory.

2. Select **Add** under **Service Providers** menu.



3. Give a service provider name and click **Register**.

Add Service Provider

Basic Information

Service Provider Name:*	<input type="text" value="API_Manager"/>
<small>⑦ A unique name for the service provider</small>	
Description:	<input type="text"/>
<small>⑦ A meaningful description about the service provider</small>	
<input type="button" value="Register"/> <input type="button" value="Cancel"/>	

 In a multi tenanted environment, for all tenants to be able to log in to the APIM Web applications,

do the following:

- Click the **SaaS Application** option that appears after registering the service provider.

Basic Information

Service Provider Name:*	API_Manager
<small>⑦ A unique name for the service provider</small>	
Description:	
<small>⑦ A meaningful description about the service provider</small>	
<input type="checkbox"/> SaaS Application	

If not, only users in the current tenant domain (the one you are defining the service provider in) will be allowed to log in to the Web application and you have to register new service providers for all Web applications (API Store and API Publisher in this case) from each tenant space separately. For example, let's say you have three tenants as TA, TB and TC and you register the service provider in TA only. If you tick the **SaaS Application** option, all users in TA, TB, TC tenant domains will be able to log in. Else, only users in TA will be able to log in.

- Add the following inside the `<SSOService>` element in the `<IS_HOME>/repository/conf/identity.xml` file and restart the server.

```
<SSOService>
  <UseAuthenticatedUserDomainCrypto>true</UseAuthenticatedUserDomainCrypto>
  ...
</SSOService>
```

If not, you get an exception as SAML response signature verification fails.

- Because the servers in a multi-tenanted environment interact with all tenants, all nodes should share the same user store. Therefore, make sure you have a shared registry (JDBC mount, WSO2 Governance Registry etc.) instance across all nodes.

- You are navigated to the detailed configuration page. Expand **SAML2 Web SSO Configuration** inside the **Inbound Authentication Configuration** section.
- Provide the configurations to register the API Publisher as the SSO service provider. These sample values may change depending in your configuration.
 - Issuer : API_PUBLISHER
 - Assertion Consumer URL : https://localhost:9443/publisher/jagg/jaggery_acs.jag. Change the IP and port accordingly. This is the URL for the acs page in your running publisher app.
 - Select the following options:
 - Use fully qualified username in the NameID**
 - Enable Response Signing**
 - Enable Assertion Signing**
 - Enable Single Logout**
 - Click **Register** once done.

For

example :

Register New Service Provider

New Service Provider

Issuer *	API_PUBLISHER
Assertion Consumer URL *	localhost:9443/publisher/jagg/jaggery_acs.jag
NameID format	urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
<input checked="" type="checkbox"/> Use fully qualified username in the NameID <input type="checkbox"/> Define Claim Uri for NameID <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">http://wso2.org/claims/otherphone</div>	
<input checked="" type="checkbox"/> Enable Response Signing <input checked="" type="checkbox"/> Enable Assertion Signing <input type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests	
Certificate Alias	wso2carbon.cert
<input type="checkbox"/> Enable Assertion Encryption	
Certificate Alias	wso2carbon.cert
<input checked="" type="checkbox"/> Enable Single Logout	
Custom Logout URL	
<input type="checkbox"/> Enable Attribute Profile	

6. Similarly, provide the configurations to register the API Store as the SSO service provider. These sample values may change depending in your configuration.

- Issuer : API_STORE
- Assertion Consumer URL : https://localhost:9443/store/jagg/jaggery_acs.jag. Change the IP and port accordingly. This is the URL for the acs page in your running store app.
- Select the following options:
 - **Use fully qualified username in the NameID**
 - **Enable Response Signing**
 - **Enable Assertion Signing**
 - **Enable Single Logout**
- Click **Register** once done.

7. Make sure that the `<responseSigningEnabled>` element is set to `true` in both the following files:

- `<AM_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json`
- `<AM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json`

Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

1. Open `<AM_Home>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` and modify the following configurations found under **ssoConfiguration**.
 - **enabled**: Set this value to **true** to enable SSO in the application
 - **issuer**: API_PUBLISHER. This value can change depending on the **Issuer** value defined in WSO2 IS SSO configuration above.
 - **identityProviderURL**: <https://localhost:9444/samlss>. Change the IP and port accordingly. This is the redirecting SSO URL in your running WSO2 IS server instance.
 - **keyStoreName**: The keystore of the running IDP. As you use a remote instance of WSO2 IS here, you can import the public certificate of the IS keystore to the APIM and then point to the APIM keystore.

The default keystore of the APIM is <APIM_HOME>/repository/resources/security/wso2carbon.jks. **Be sure to give the full path of the keystore here.**

- **keyStorePassword:** Password for the above keystore
- **identityAlias:** wso2carbon

2. Similarly, configure the API Store with SSO. The only difference in API Store SSO configurations is setting **A PI_STORE** as the **issuer**.
3. Reduce the priority of the **SAML2SSOAuthenticator** configuration in the <APIM_HOME>/repository/conf/security/authenticators.xml file.

You do this as a workaround for a known issue that will be fixed in a future release. The **SAML2SSOAuthenticator** handler does not process only SAML authentication requests at the moment. If you set its priority higher than that of the **BasicAuthenticator** handler, the **SAML2SSOAuthenticator** tries to process the basic authentication requests as well. This causes login issues in the API Publisher/Store.

```
<Authenticator name="SAML2SSOAuthenticator" disabled="false">
    <Priority>0</Priority>
    ...
</Authenticator>
```

4. Access the API Publisher: <https://localhost:<Port number>/publisher> (e.g., <https://localhost:9443/publisher>). Observe the request redirect to WSO2 IS SAML2.0 based SSO login page. For example,

SAML 2.0 based Single Sign-On

Please enter your credentials to sign in to WSO2 Identity Server SAML SSO

User Name

Password

© 2008 - 2011 WSO2 Inc. All Rights Reserved.

5. Enter user credentials. If the user authentication is successful against WSO2 IS, it will redirect to the API Publisher Web application with the user already authenticated.
6. Access the API Store application, click its **Login** link (top, right-hand corner) and verify that the same user is already authenticated in API Store.

! Even with SSO enabled, if the user doesn't have sufficient privileges to access API Publisher/Store or any other application, s/he will not be authorized to access them.

i The steps above explain how to configure SSO between the API Publisher and Store Jagger applications, using WSO2 IS as the IDP. If there are many WSO2 products in your environment, you can configure SSO for the management consoles of those products by changing the **SAML2SSOAuthenticator** configuration in the <APIM_HOME>/repository/conf/security/authenticators.xml file as follows:

- Set disabled attributes in **<Authenticator>** element to **false**
- **ServiceProviderID** : In this example, it is the issuer name of the service provider created in step 1
- **IdentityProviderSSOServiceURL** : In this example, it is the Identity Server port

```

<Authenticator name="SAML2SSOAuthenticator" disabled="false">
    <Priority>10</Priority>
    <Config>
        <Parameter
name="LoginPage">/carbon/admin/login.jsp</Parameter>
        <Parameter name="ServiceProviderID">carbonserver1</Parameter>
        <Parameter
name="IdentityProviderSSOServiceURL">https://localhost:9444/samlsso</Parameter>
        <Parameter
name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
    </Config>

```

Make sure the `<priority>` of the `SAML2SSOAuthenticator` is less than that of the `BasicAuthenticator` handler. See [here](#) for more information.

Changing the Default Transport

APIs are synapse configurations in the back-end and API Manager accesses them using HTTP-NIO transport by default. You can switch to a different transport such as PassThrough. To change the default transport of API Manager, go to `<APIM_HOME>/repository/conf/axis2` folder and rename `axis2.xml_PT` file to `axis2.xml`. Similarly, you can switch back to NHTTP by simply renaming `axis2.xml_NHTTP` file to `axis2.xml`.

The following topics explain HTTP-NIO and PassThrough transports:

- [HTTP-NIO transport](#)
- [HTTP PassThrough transport](#)

HTTP-NIO transport

HTTP-NIO transport is a module of the Apache Synapse project. Apache Synapse as well as WSO2 APIM ship the HTTP-NIO transport as the default HTTP transport implementation. The two classes that implement the receiver and sender APIs are `org.apache.synapse.transport.nhttp.HttpCoreNIOListener` and `org.apache.synapse.transport.nhttp.HttpCoreNIOListener` respectively. These classes are available in the JAR file named `synapse-nhttp-transport.jar`. This non-blocking transport implementation improves performance. The transport implementation is based on Apache HTTP Core - NIO and uses a configurable pool of non-blocking worker threads to grab incoming HTTP messages off the wire.

Transport receiver parameters

Info In transport parameter tables, literals displayed in italic mode under the "Possible Values" column should be considered as fixed literal constant values. Those values can be directly put in transport configurations.

Parameter Name	Description	Required	Possible Values	Default Value
port	The port on which this transport receiver should listen for incoming messages.	No	A positive integer less than 65535	8280
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	<i>true</i>	

bind-address	The address of the interface to which the transport listener should bind.	No	A host name or an IP address	127.0.0.1
hostname	The host name of the server to be displayed in service EPRs, WSDLs etc. This parameter takes effect only when the WSDLEPRPrefix parameter is not set.	No	A host name or an IP address	localhost
WSDLEPRPrefix	A URL prefix which will be added to all service EPRs and EPRs in WSDLs etc.	No	A URL of the form <protocol>://<hostname>:<port>/	

Transport sender parameters

Parameter Name	Description	Required	Possible Values	Default Value
http.proxyHost	If the outgoing messages should be sent through an HTTP proxy server, use this parameter to specify the target proxy.	No	A host name or an IP address	
http.proxyPort	The port through which the target proxy accepts HTTP traffic.	No	A positive integer less than 65535	
http.nonProxyHosts	The list of hosts to which the HTTP traffic should be sent directly without going through the proxy.	No	A list of host names or IP addresses separated by ' '	
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	true	

HTTP PassThrough transport

HTTP PassThrough Transport is the default, non-blocking HTTP transport implementation based on HTTP Core NIO and is specially designed for streaming messages. It is similar to the old message relay transport, but it does not care about the content type and simply streams all received messages through. It also has a simpler and cleaner model for forw arding messages back and forth. It can be used as an alternative to the NHTTP transport.

The HTTP PassThrough Transport is enabled by default. If you want to use the NHTTP transport instead, uncomment the relevant NHTTP transport entries in `axis2.xml` and comment out the HTTP PassThrough transport entries. T he PassThrough Transport does not require the binary relay builder and expanding formatter.

Connection throttling

With the HTTP PassThrough and HTTP NIO transports, you can enable connection throttling to restrict the number of simultaneous open connections. To enable connection throttling, edit the `<PRODUCT_HOME>/repository/conf/nhttp.properties` (for the HTTP NIO transport) or `<PRODUCT_HOME>/repository/conf/passthru.properties` (for the PassThrough transport) and add the following line: `max_open_connections = 2`

This will restrict simultaneous open incoming connections to 2. To disable throttling, delete the `max_open_connections` setting or set it to -1.

 Connection throttling is never exact. For example, setting this property to 2 will result in roughly two

simultaneous open connections at any given time.

i WSO2 products do not use the HTTP/S servlet transport configurations that are in `axis2.xml` file. Instead, they use Tomcat-level servlet transports, which are used by the management console in `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file.

Configuring Caching

When an API call hits the API Gateway, the Gateway carries out security checks to verify if the token is valid. During these verifications, the API Gateway extracts parameters such as access token, API and API version that are passed on to it. Since the entire load of traffic to APIs goes through the API Gateway, this verification process needs to be fast and efficient in order to prevent overhead and delays. The API Manager uses caching for this purpose, where the validation information is cached with the token, API name and version, and the cache is stored in either the API Gateway or the key manager server.

This section covers the following:

- [Caching at API Gateway](#)
- [Resource caching](#)
- [Caching at Key Manager server](#)
- [Response caching](#)

Caching at API Gateway

When caching is enabled at the Gateway and a request hits the Gateway, it first populates the cached entry for a given token. If a cache entry does not exist in cache, it calls the key manager server. This process is carried out using Web service calls. Once the key manager server returns the validation information, it gets stored in the Gateway. Because the API Gateway issues a Web service call to the key manager server only if it does not have a cache entry, this method reduces the number of Web service calls to the key manager server. Therefore, it is faster than the alternative method.

By default, the API Gateway cache is enabled by setting the `<EnableGatewayKeyCache>` element to true in `<API_M_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayKeyCache>true</EnableGatewayKeyCache>
```

Clearing the API Gateway cache

To remove old tokens that might still remain active in the Gateway cache, you configure the `<RevokeAPIURL>` element in `api-manager.xml` file by providing the URL of the [Revoke API](#) that is deployed in the API Gateway node. The revoke API invokes the cache clear handler, which extracts information from transport headers of the revoke request and clears all associated cache entries. If there's a cluster of API Gateways in your setup, provide the URL of the revoke API deployed in one node in the cluster. This way, all revoke requests route to the OAuth service through the Revoke API.

Given below is how to configure this in a distributed API Manager setup.

1. In the `api-manager.xml` file of the key manager node, point the revoke endpoint as follows:

```
<RevokeAPIURL>https://{$carbon.local.ip}:{$https.nio.port}/revoke</RevokeAPIURL>
```

2. In the API Gateway, point the Revoke API to the OAuth application deployed in the key manager node. For example,

```

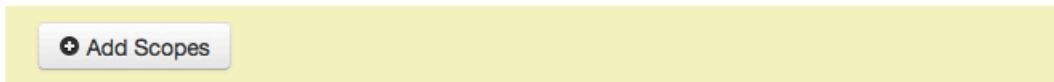
<api name="_WSO2AMRevokeAPI_" context="/revoke">
    <resource methods="POST" url-mapping="/*" faultSequence="_token_fault_">
        <inSequence>
            <send>
                <endpoint>
                    <address
uri="https://keymgt.wso2.com:9445/oauth2/revoke"/>
                </endpoint>
            </send>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerCacheExtensionHandle
r"/>
    </handlers>
</api>

```

Resource caching

An API's resources are HTTP methods that handle particular types of requests such as GET, POST etc. They are similar to methods of a particular class. Each resource has parameters such as its throttling level, Auth type etc.

Resources



/default

GET	/* + Summary	Application & Application User	Unlimited	+ Scope
POST	/* + Summary	Application & Application User	Unlimited	+ Scope
PUT	/* + Summary	Application & Application User	Unlimited	+ Scope
DELETE	/* + Summary	Application & Application User	Unlimited	+ Scope
OPTIONS	/* + Summary	None	Unlimited	+ Scope

Users can make requests to an API by calling any one of the HTTP methods of the API's resources. The API Manager uses the resource cache at the Gateway node to store the API's resource-level parameters (Auth type and throttling level). The cache entry is identified by a cache key, which is based on the API's context, version, request path and HTTP method. Caching avoids the need to do a separate back-end call to check the Auth type and throttling level of a resource, every time a request to the API comes. It improves performance.

Note that if you change a resource's parameters such as the Auth type through the UI, it takes about 15 minutes to refresh the resource cache. During that time, the server returns the old Auth type from the cache. If you want the changes to be reflected immediately, please restart the server after changing the value.

By default, the resource cache is enabled by setting the `<EnableGatewayResourceCache>` element to true in `<API_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayResourceCache>true</EnableGatewayResourceCache>
```

Caching at Key Manager server

In this method, the cache is maintained at the key manager server rather than the API Gateway. As a result, for each and every API call that hits the API Gateway, the Gateway issues a Web service call to the key manager server. If the cache entry is available in the key manager server, it is returned to the Gateway. Else, the database will be checked for the validity of the token.

This method has low performance compared to the earlier one, but the advantage of this method over the other is that we do not have to store any security-related information at the Gateway side.

By default, caching is enabled at the Gateway side as it is the faster method. If you want to change this default configuration, disable caching at the Gateway side and enable it at the key manager server side by using the instructions below.

1. Disable caching at API Gateway by adding the following entry to **APIGateway** section of `<APIM_HOME>/repository/conf/api-manager.xml` file.

```
<EnableGatewayKeyCache>false</EnableGatewayKeyCache>
```

2. Enable key manager server-side caching by adding the following entry to **APIKeyManager** section of the `api-manager.xml` file.

```
<EnableKeyMgtValidationInfoCache>true</EnableKeyMgtValidationInfoCache>
```

3. The API Manager generates JWT tokens for each validation information object. Usually, JWT tokens also get cached with the validation information object, but you might want to generate JWT per each call. You can do this by enabling JWT caching at key manager server. Add the following entry to **APIKeyManager** section of the `api-manager.xml` file.

```
<EnableJWTCache>true</EnableJWTCache>
```



Note that you must disable caching at the key manager server side in order to generate JWT per each call.

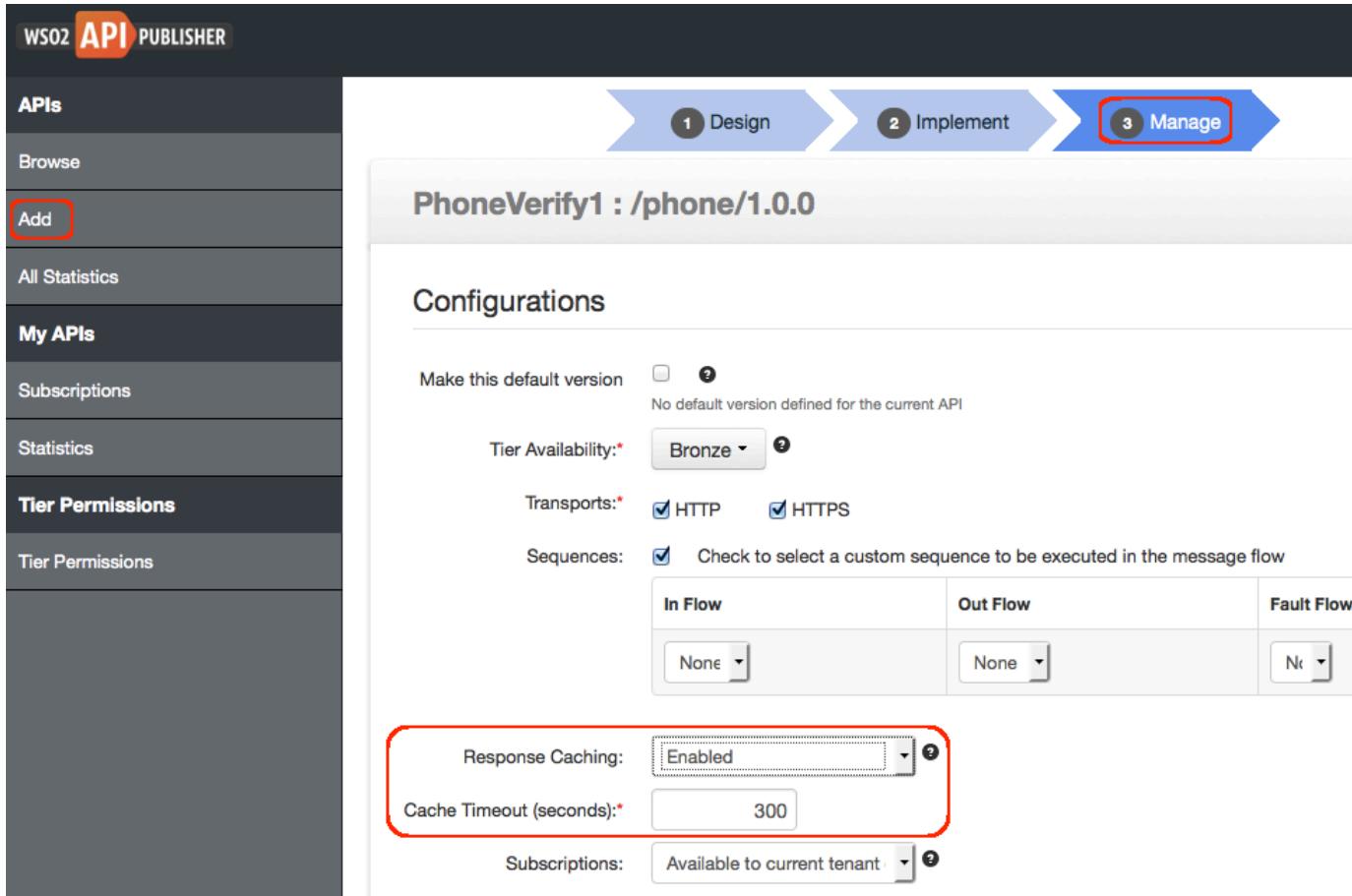
Also enable token generation by setting the following entry to `true` at the root level of the `api-manager.xml` file.

```
<APIConsumerAuthentication>
  <EnableTokenGeneration>true</EnableTokenGeneration>
  ...
</APIConsumerAuthentication>
```

Response caching

The API Manager uses [WSO2 ESB's cache mediator](#) to cache response messages per each API. Caching improves performance, because the backend server does not have to process the same data for a request multiple times. To offset the risk of stale data in the cache, you set an appropriate timeout period.

You enable response caching when creating a new API or editing an existing one using the API Publisher UI. Go to the API Publisher and click the **Add API** menu (to create a new API) or the **Edit** link associated with an existing API. Then, navigate to the **Manage** tab where you find the response caching section. You can set it to **Enabled** and give a timeout value. This enables the default response caching settings.



The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with 'APIs' and 'My APIs' sections. The 'Add' button in the 'APIs' section is highlighted with a red box. The main area shows an API named 'PhoneVerify1 : /phone/1.0.0' in the 'Manage' tab. The 'Configurations' section includes fields for 'Make this default version' (unchecked), 'Tier Availability' (set to 'Bronze'), 'Transports' (HTTP and HTTPS checked), 'Sequences' (checkbox checked), and a table for 'In Flow', 'Out Flow', and 'Fault Flow' with dropdowns for 'None'. At the bottom, there are fields for 'Response Caching' (set to 'Enabled') and 'Cache Timeout (seconds)' (set to '300'), which are also highlighted with a red box. The 'Subscriptions' field is set to 'Available to current tenant'.

To change the default response caching settings, edit the following cache mediator properties in <APIM_HOME>/repository/resources/api_templates/velocity_template.xml file:

Property	Description
collector	<ul style="list-style-type: none"> • <code>true</code> : specifies that the mediator instance is a response collection instance • <code>false</code>: specifies that the mediator instance is a cache serving instance
max Message Size	Specifies the maximum size of a message to be cached in bytes. An optional attribute, with the default value set to <code>unlimited</code> .
maxSize	Defines the maximum number of elements to be cached
hashGenerator	<p>Defines the hash generator class.</p> <p>When caching response messages, a hash value is generated based on the request's URI, transport headers and the payload (if available). WSO2 has a default REQUESTHASHGenerator class written to generate the hash value. See sample here.</p> <p>If you want to change this default implementation (for example, to exclude certain headers), you can write a new hash generator implementation by extending the REQUESTHASHGenerator and overriding its <code>getDigest()</code> method. Once done, add the new class as the <code>hashGenerator</code> attribute of the <code><cache></code> element in the <code>velocity_template.xml</code> file.</p>

Working with Databases

The default databases that WSO2 products uses to store registry, user manager and product-specific data are the H2 databases in <PRODUCT_Home>/repository/database as follows:

- **WSO2CARBON_DB.h2.db**: used to store registry and user manager data
- **WSO2AF_DB.h2.db**: used to store App Factory specific data

These embedded H2 databases are suitable for development, testing, and some production environments. For most production environments, however, we recommend you to use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. You can use the scripts provided with WSO2 products to install and configure several other types of relational databases, including MySQL, IBM DB2, Oracle, and more.

The following sections explain how to change the default databases:

- Setting up the Physical Database
- 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. After you set up the database, you create datasources to connect to it.

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

Setting up IBM DB2

The following sections describe how to replace the default H2 databases 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/storage 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 => connect to regdb user greg using 18091980
Database Connection Information
Database server      = DB2/LINUX 9.7.4
SQL authorization ID = GREG
Local database alias = REGDB
db2 => GRANT DBADM, CREATETAB, BINDADD, CONNECT, CREATE_NOT_FENCED_SCHEMA, LOAD ON DATABASE TO USER user
DB20000I  The SQL command completed successfully.
db2 => ■
```



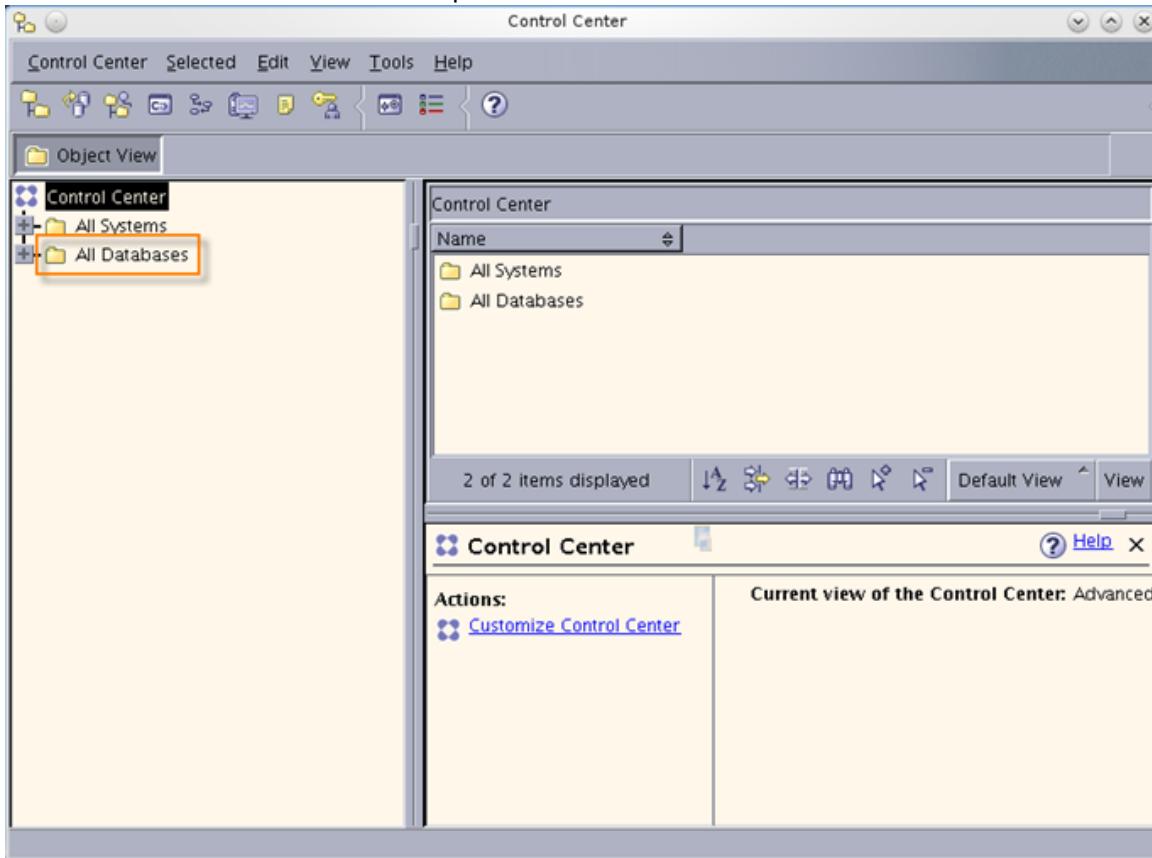
For more information on DB2 commands, see the [DB2 Express-C Guide](#).

Using the DB2 control center

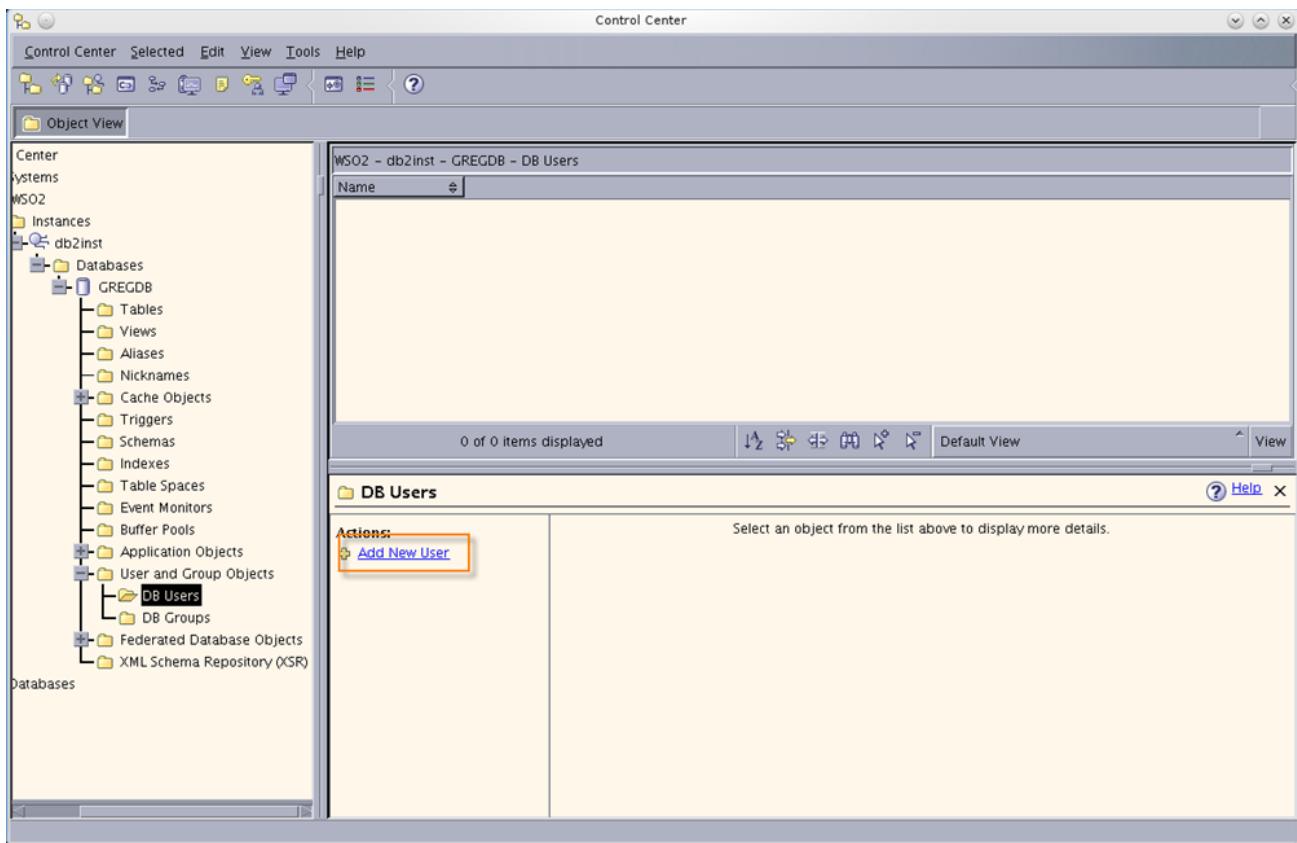
1. Open the DB2 control center using the db2cc command as follows:

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

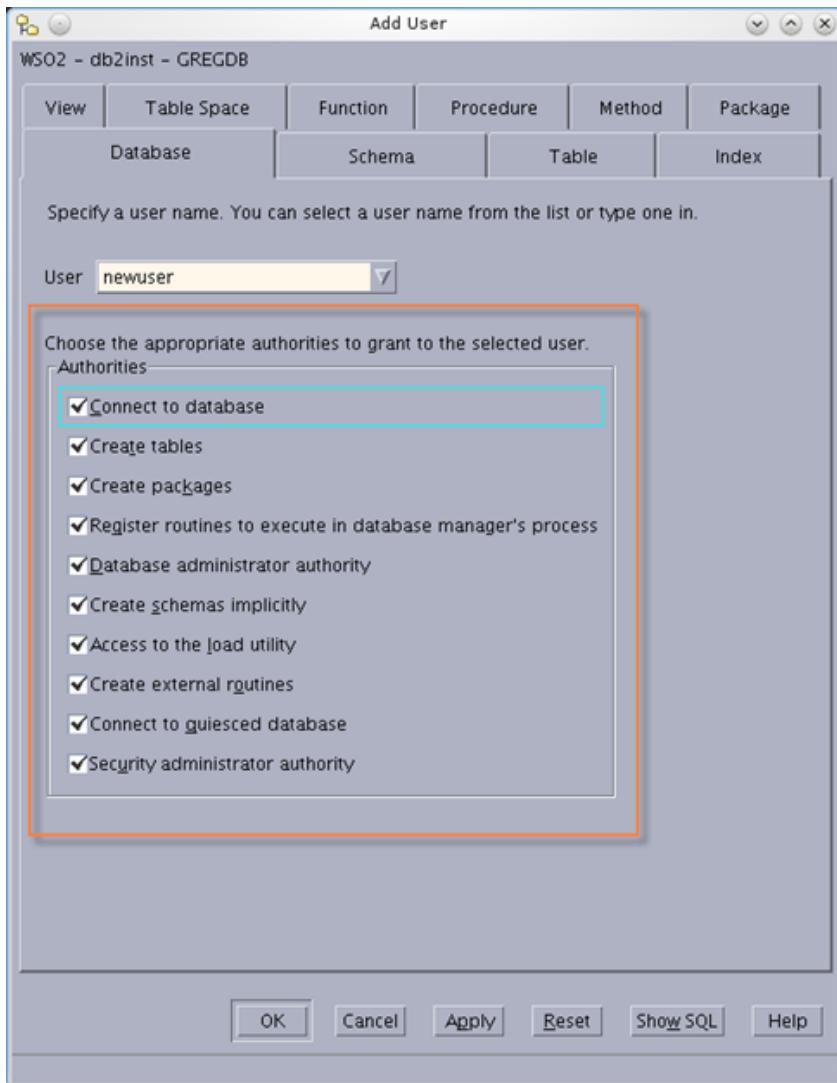
2. Right-click **All Databases** in the control center tree (inside the object browser), click **Create Database**, and then click **Standard** and follow the steps in the **Create New Database** wizard.



3. Click **User and Group Objects** in the control center tree to create users for the newly created database.



4. Give the required permissions to the newly created users.



Setting up DB2 JDBC drivers

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

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

i <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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly as shown below:

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>
            <url>jdbc:db2://SERVER_NAME:PORT/DB_NAME</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>360000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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

 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](#).

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.

(i) 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/storage databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database that come 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 `master-datasources.xml` file by the name `WSO2AM_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 <code><PRODUCT_HOME>/dbscripts/apimgt</code> folder
For the identity database	Use the scripts in <code><PRODUCT_HOME>/dbscripts/identity</code> folder

Setting up Derby

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

- [Setting up Embedded Derby](#)
- [Setting up Remote Derby](#)

Setting up Embedded Derby

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

- [Creating the database](#)
- [Setting up drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)
- [Changing the product-specific/identity/storage databases](#)

Creating the database

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

1. Download Apache Derby.

2. Install Apache Derby on your computer.

i For instructions on installing Apache Derby, see the [Apache Derby documentation](#).

Setting up drivers

Copy `derby.jar`, `derbyclient.jar`, and `derbynnet.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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Replace the `url`, `username`, `password` and `driverClassName` settings with your custom values and also the other values accordingly 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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
url	The URL of the database. The default port for a DB2 instance is 50000.
username and password	The name and password of the database user
driverClassName	The class name of the database driver
maxActive	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.
maxWait	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.

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

i 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](#).

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
ij> ■
```

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

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

i 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';
```

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

Restart the server.

i 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

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

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

Changing the product-specific/identity/storage databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database that come 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 `master-datasources.xml` file by the name `WSO2AM_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 <code><PRODUCT_HOME>/dbscripts/apimgt</code> folder
For the identity database	Use the scripts in <code><PRODUCT_HOME>/dbscripts/identity</code> folder

Setting up Remote Derby

The following sections describe how to replace the default H2 databases with a remote Derby database:

- [Creating the database](#)
- [Setting up drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)
- [Changing the product-specific/identity databases](#)

Creating the database

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

1. [Download Apache Derby](#).
2. [Install Apache Derby](#) on your computer.

i 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 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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Replace the `url`, `username`, `password` and `driverClassName` settings with your custom values and also the other values accordingly 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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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

 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](#).

i 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.

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.

```
client@wso2:~/dtb/db-derby-10.8.1.2-bin/bin$ ./ij
ij version 10.8
ij> ■
```

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

```
connect
'jdbc:derby://localhost:1527/db;user=regadmin;password=regadmin;create=true';
```

i 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.

i 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`

i The product is now configured to run using a remote Apache Derby database.

Changing the product-specific/identity databases

The topics above show how to change the `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_DB`) that come 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 `WSO2AM_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 <code><PRODUCT_HOME>/dbscripts/apimgt</code> folder
--	--

For the identity database

Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

Setting up H2

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

- Setting up Embedded H2
- Setting up Remote H2

Setting up Embedded H2

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

- Preparing the database
- Setting up drivers
- Setting up datasource configurations
- Creating database tables
- Changing the product-specific/identity databases

Preparing 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 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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly 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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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



For more information on other parameters that can be defined in the <PRODUCT_HOME>/repository

`y/conf/datasources/ master-datasources.xml file, see Tomcat JDBC Connection Pool.`

Creating database tables

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

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

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

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

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

i 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 `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_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 `WSO2AM_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 <code><PRODUCT_HOME>/dbscripts/apimgt</code> folder
For the identity database	Use the scripts in <code><PRODUCT_HOME>/dbscripts/identity</code> folder

Setting up Remote H2

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

- [Preparing the remote H2 database](#)
- [Setting up drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)
- [Changing the product-specific/identity databases](#)

Preparing 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.

i For instructions on installing, see the [H2 installation guide](#).

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

15% [=====] 923,304 111K/s eta 45s
```

2. Go to the <H2_HOME>/bin/ directory and run the H2 network server starting script as follows, where <H2_HOME> is the H2 installation directory:

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

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

- For Linux:

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

- For Windows:

```
$ h2.bat
```

i 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 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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly 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>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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

 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](#).

Creating database tables

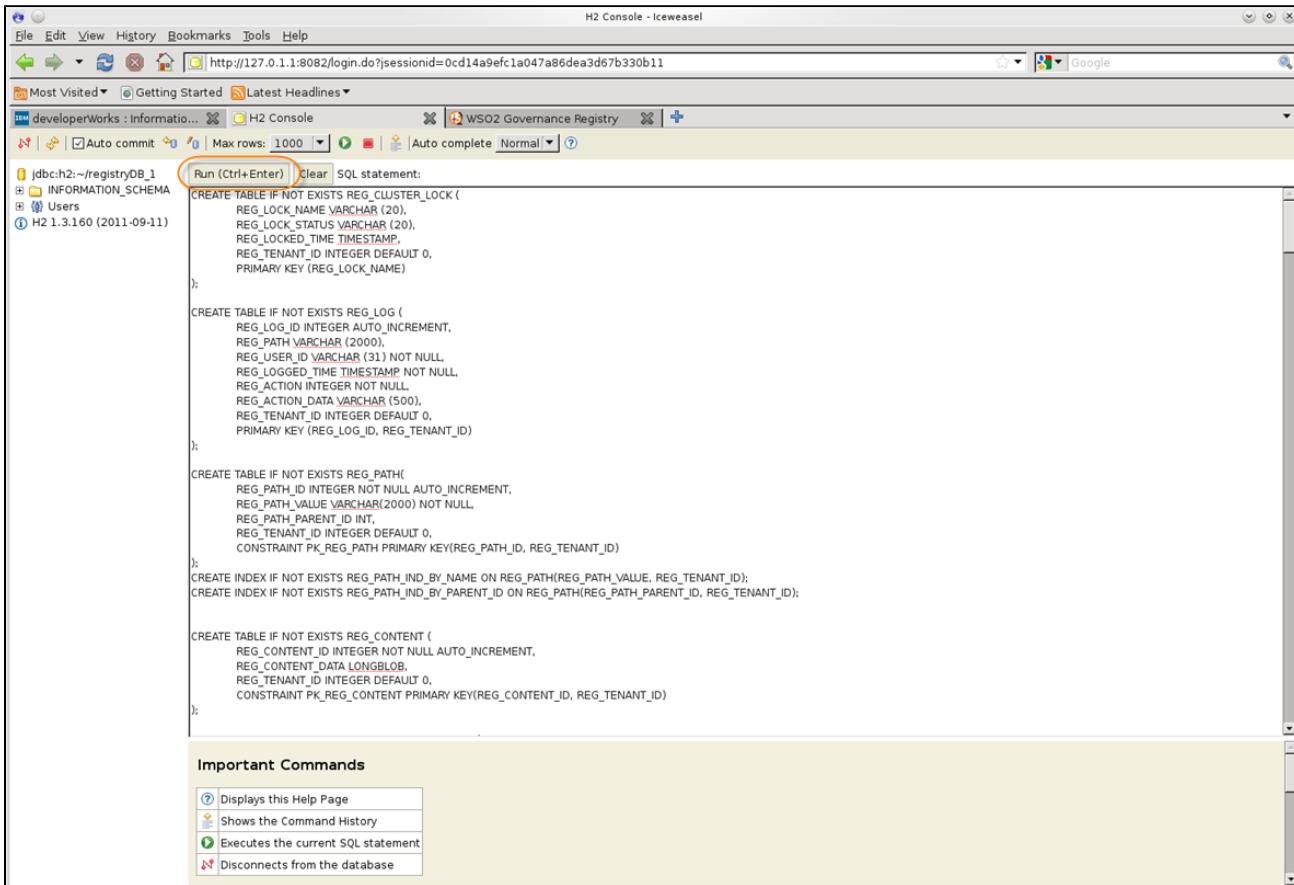
To create the database tables, connect to the database that you created earlier and run the following scripts in H2 shell or web console:

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

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

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

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



5. Restart the server.

i 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 `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_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 `WSO2AM_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 <code><PRODUCT_HOME>/dbscripts/apimgt</code> folder
For the identity database	Use the scripts in <code><PRODUCT_HOME>/dbscripts/identity</code> folder

Setting up IBM Informix

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

- [Prerequisites](#)
- [Creating the database](#)
- [Setting up Informix JDBC drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)
- [Changing the product-specific/identity databases](#)

Prerequisites

Download the latest version of [IBM Informix](#) and install it on your computer.

Creating the database

Create the database and users in Informix.

 For instructions on creating the database and users, see [Informix product documentation](#) .

Setting up Informix JDBC drivers

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

Setting up datasource configurations

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Replace the `url`, `username`, `password` and `driverClassName` settings with your custom values and also the other values accordingly as shown below:

```

<datasource>
    <name>WSO2AM_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <!-- IP ADDRESS AND PORT OF DB SERVER -->
            <url>jdbc:informix-sqli://localhost:1533/AM_DB</url>
            <username>wso2carbon</username>
            <password>wso2carbon</password>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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

 For more information on other parameters that can be defined in the < PRODUCT_HOME>/repository/conf/datasources/jndi-datasources.xml file, refer to the WSO2 API Manager Configuration Guide.

ry/conf/datasources/master-datasources.xml file, see Tomcat JDBC Connection Pool.

Creating database tables

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

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

<PRODUCT_HOME>/dbscripts/informix.sql

2. **R**estart **t**he **s**erver.

i 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 `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_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 `wso2AM_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/apimgt folder
For the identity database	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

Setting up Microsoft SQL

The following sections describe how to replace the default H2 database with MS SQL.

- Setting up the database and users
- Copying 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.

Follow the 3 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.

Copying 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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly as shown below:

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>
            <url>jdbc:sqlserver://<IP>:1433;databaseName=wso2greg</url>
            <username>regadmin</username>
            <password>regadmin</password>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

Element	Description
url	The URL of the database. Change the <IP> with the IP of the server. The best practice is to use port 1433, because you can use it in order processing services.

username and password	The name and password of the database user
driverClassName	The class name of the database driver
maxActive	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.
maxWait	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.
minIdle	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
testOnBorrow	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.
validationQuery	The SQL query that will be used to validate connections from this pool before returning them to the caller.
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.

i 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](#).

Creating the database tables

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

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

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

2. Restart the server.

i 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 WSO2_CARBON_DB, which is used to store registry and user manager information. If you changed the product-specific database (WSO2AM_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 **WSO2AM_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/apimgt folder
For the identity database	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

Setting up MySQL

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

 For instructions on installing MySQL on MAC OS, go to [Homebrew](#).

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

2. Start the MySQL service using the following command:

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

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

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

4. Enter the password when prompted.

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

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

```
create database regdb;
```

 For users of Microsoft Windows, when creating the database in MySQL, it is important to specify the character set as latin1. Failure to do this may result in an error (error code: 1709) when starting your cluster. This error occurs in certain versions of MySQL (5.6.x), and is related to the UTF-8 encoding. MySQL originally used the latin1 character set by default, which stored characters in a 2-byte sequence. However, in recent versions, MySQL defaults to UTF-8 to be friendlier to international users. Hence, you must use latin1 as the character set as indicated below in the database creation commands to avoid this problem. Note that this may result in issues with non-latin characters (like Hebrew, Japanese, etc.). The database creation command should be as follows:

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

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

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

6. Give authorization 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 `<PRODUCT_HOME>/repository/components/lib/` directory.

Setting up datasource configurations

The H2 database that comes by default stores registry and user management related data. Follow the steps below to change the type of the default database or create new databases to manage registry or/and user management related data separately.

Changing the default database

Follow the steps below to change the type of the default H2 database.

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Replace the `url`, `username`, `password` and `driverClassName` settings with your custom values and also the other values accordingly as shown below.



Do not change the datasource name `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>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

Element	Description
<code>url</code>	The URL of the database. The default port for MySQL is 3306

username and password	The name and password of the database user
driverClassName	The class name of the database driver
maxActive	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.
maxWait	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.
minIdle	The minimum number of active connections that can remain idle in the pool without extra ones being created, or enter zero to create none.
testOnBorrow	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.
validationQuery	The SQL query that will be used to validate connections from this pool before returning them to the caller.
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.

i 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](#).

Creating new databases to manage registry or user management related data

Follow the steps below to create new databases to manage registry or/and user management related data separately.

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

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/WSO2CarbonDB</dataSource>
</dbConfig>
```

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

```
<Configuration>
  <Property name="dataSource">jdbc/WSO2CarbonDB</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:

i 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.

i 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 registry/user management databases

If you change the database that come 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 `WSO2_CARBON_DB`, which is used to store registry and user manager information. If you changed the product-specific database (`WSO2AM_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 `WSO2AM_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/apimgt folder
For the identity database	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

Setting up MySQL Cluster

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

Setting up OpenEdge

The following sections describe how to set up the default H2 database with OpenEdge:

- [Setting up the database and user](#)
- [Setting up the drivers](#)
- [Setting up datasource configurations](#)
- [Creating database tables](#)

Setting up the database and user

Follow the steps below to set up an OpenEdge (OE) database.

1. Download and install OpenEdge on you 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.

```
prodb CARBON_DB <OE-installation-directory>/empty8
```

5. Start the database using the proserv script as follows. Provide the database name and a port as arguments to this script using the -db and -S parameters.

```
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:

```
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:

```
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

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly 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://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>
    </configuration>
</definition>
</datasource>

```

The elements in the above configuration are described below:

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

 For more information on other parameters that can be defined in the <PRODUCT_HOME>/repository

try/conf/datasources/ master-datasources.xml file, see [Tomcat JDBC Connection Pool](#).

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:

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

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:

```
COMMIT;
```

4. Run the modified script using the SQL explorer as follows:

```
sqlexp -db CARBON_DB -S 6767 -user wso2carbon -password wso2carbon
<PRODUCT_HOME>/dbscripts/openedge_manual.sql
```

5. Restart the server.

i 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 user](#)
- [Copying the JDBC driver](#)
- [Setting up datasource configurations](#)
- [Creating the database tables](#)
- [Changing the product-specific/identity databases](#)

Setting up the database and user

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 listner.ora files in order to define addresses of the databases for establishing connections to the newly created database.
3. After configuring the .ora files, start the Oracle instance using the following command:

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

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

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

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

```
$ connect
```

6. As SYSDBA, create a database user and grant privileges to the user as shown below:

```
Create user USER_NAME identified by password account unlock;
grant connect to USER_NAME;
grant create session, create table, create sequence, create trigger to USER_NAME;
commit;
```

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

Copying 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.

i 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_OPT` property.

Setting up datasource configurations

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file. Replace the `url`, `username`, `password` and `driverClassName` settings with your custom values and also the other values accordingly 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>
        </configuration>
    </definition>
</datasource>
```

The elements in the above configuration are described below:

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

 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](#).

Creating the database tables

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

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

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

2. Restart the server.

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

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

Changing the product-specific/identity databases

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager

information. If you changed the product-specific database (`WSO2AM_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 `WSO2AM_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 <code><PRODUCT_HOME>/dbscripts/apimgt</code> folder
For the identity database	Use the scripts in <code><PRODUCT_HOME>/dbscripts/identity</code> folder

Setting up Oracle RAC

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

- Setting up the database and user
- Copying 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`.

Setting up the database and user

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

1. Set environment variables `<ORACLE_HOME>`, `PATH`, and `ORACLE_SID` with the corresponding values `/oracle/app/oracle/product/11.2.0/dbhome_1`, `$PATH:<ORACLE_HOME>/bin`, and `orcl1` as follows:

```
[oracle@node1 ~]$ export ORACLE_HOME=/oracle/app/oracle/product/11.2.0/dbhome_1
[oracle@node1 ~]$ export PATH=$PATH:$ORACLE_HOME/bin
[oracle@node1 ~]$ export ORACLE_SID=orcl1
```

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

```
[oracle@node1 ~]$ sqlplus SYSDBA/1 as sysdba
SQL*Plus: Release 11.2.0.1.0 Production on Fri Nov 18 18:10:42 2011
Copyright (c) 1982, 2009, Oracle. All rights reserved.

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

SQL> select 2+2 from dual;

  2+2
-----
     4

SQL> create user dbgreg identified by dbgreg account unlock;
User created.

SQL> grant connect to dbgreg;
Grant succeeded.

SQL> grant create session, dba to dbgreg;
Grant succeeded.

SQL> commit;
Commit complete.

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

3. Create a database user and grant privileges to the user as shown below:

```
Create user USER_NAME identified by PASSWORD account unlock;
grant connect to USER_NAME;
grant create session, dba to USER_NAME;
commit;
```

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

Copying 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.

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

Setting up datasource configurations

After creating the database, you create a datasource to point to it in the following files:

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly as shown below:

```

<datasource>
    <name>WSO2_CARBON_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2CarbonDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on)
                (ADDRESS=(PROTOCOL=TCP)(HOST=racnode1) (PORT=1521))
                (ADDRESS=(PROTOCOL=TCP)(HOST=racnode2) (PORT=1521))
                (CONNECT_DATA=(SERVICE_NAME=rac)))</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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

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.



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](#).

Creating the database tables

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

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

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

2. Restart the server.



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

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

Changing the product-specific/identity databases

The topics above show how to change the WSO2_CARBON_DB, which is used to store registry and user manager information. If you changed the product-specific database (WSO2AM_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 **WSO2AM_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/apimgt folder
For the identity database	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

Setting up 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.

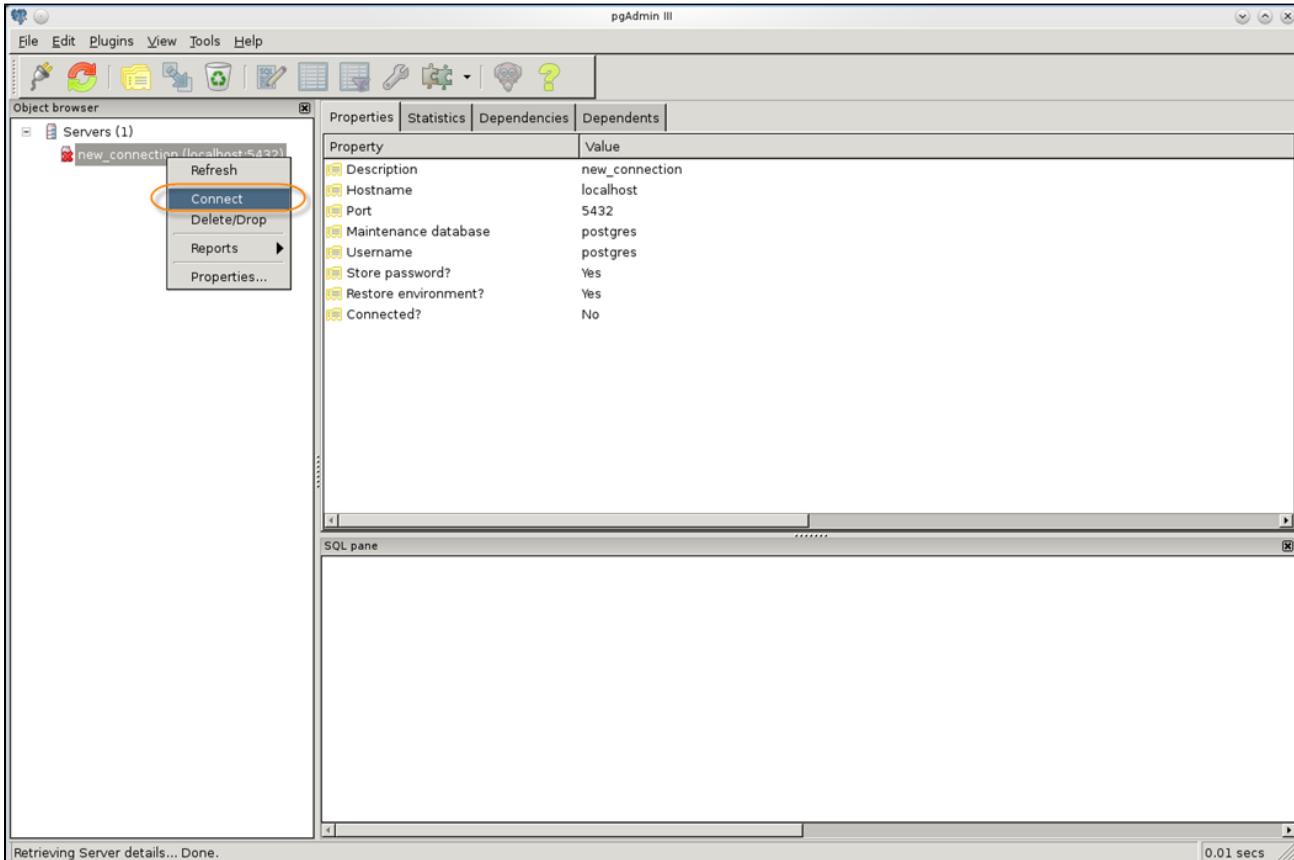
1. Install PostgreSQL on your computer as follows:

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

2. Start the PostgreSQL service using the following command:

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

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



5. To create a database, click **Databases** in the tree (inside the object browser), and click **New Database**.
6. In the **New Database** dialog box, give a name to the database (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.

Info 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

After creating the database, you create a datasource to point to it in the following files.

1. Edit the default datasource configuration in the <PRODUCT_HOME>/repository/conf/datasources/m a

ster-datasources.xml file. Replace the url, username, password and driverClassName settings with your custom values and also the other values accordingly 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:postgresql://localhost:5432/gregdb</url>
            <username>regadmin</username>
            <password>regadmin</password>
            <driverClassName>org.postgresql.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

The elements in the above configuration are described below:

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

i 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](#).

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.

i 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 WSO2_CARBON_DB, which is used to store registry and user manager information. If you changed the product-specific database (WSO2AM_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 **WSO2AM_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/apimgt folder
For the identity database	Use the scripts in <PRODUCT_HOME>/dbscripts/identity folder

Managing Datasources

A datasource provides information that a server can use to connect to a database. Datasource management is provided by the following feature in the WSO2 feature repository:

Name : WSO2 Carbon - datasource management feature
Identifier: org.wso2.carbon.datasource.feature.group

If datasource management capability is not included in your product by default, add it by installing the above feature, using the instructions given under the Feature Management section of this documentation.

Click **Data Sources** on the **Configure** tab of the product's management console to view, edit, and delete the datasources in your product instance.

i You can view, edit, and delete the datasources in your product instance by clicking **Data Sources** on the **Configure** tab of the product management console. However, you cannot edit or delete the default <WSO2_CARBON_DB> datasource.

Adding Datasources

If the datasource management feature is installed in your WSO2 product instance, you can add datasources that allow the server to connect to databases and other external data stores.

Use the following steps to add a datasource:

1. In the product management console, click **Data Sources** on the **Configure** tab.



2. Click **Add Data Source**.

3. Specify the required options for connecting to the database. The available options are based on the type of datasource you are creating:

- Configuring a RDBMS Datasource
- Configuring a Custom Datasource

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

! When adding an RDBMS datasource, be sure to copy the JDBC driver JAR file for your database to <PROD UCT_HOME>/repository/components/lib.

Configuring an RDBMS Datasource

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

New Data Source

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

This is the default RDBMS datasource configuration provided by WSO2. You can also write your own RDBMS configuration by selecting the custom datasource option. Enter values for the following fields when using the default RDBMS datasource configuration:

- **Data Source Type:** RDBMS
- **Name:** Name of the datasource (must be a unique value)
- **Data Source Provider:** Specify the datasource provider.
- **Driver:** The class name of the JDBC driver to use. Make sure to copy the JDBC driver relevant to the database engine to the <PRODUCT_HOME>/repository/components/lib/ directory. For example, if you are using MySQL, specify com.mysql.jdbc.Driver as the driver and copy mysql-connector-java-5.XX-bin.jar file to this directory. If you do not copy the driver to this directory when you create the datasource, you will get an exception similar to Cannot load JDBC driver class com.mysql.jdbc.Driver.
- **URL:** The connection URL to pass to the JDBC driver to establish the connection.
- **User Name:** The connection user name that will be passed to the JDBC driver to establish the connection.
- **Password:** The connection password that will be passed to the JDBC driver to establish the connection.
- **Expose as a JNDI Data Source:** Allows you to specify the JNDI datasource.
- **Data Source Configuration Parameters:** Allows you to specify the datasource connection pool parameters when creating a RDBMS datasource.

For more details on datasource configuration parameters, see [ApacheTomcat JDBC Connection Pool guide](#).

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

Configuring the Datasource Provider

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

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

New Data Source

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

External datasource provider

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

New Data Source

New Data Source

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

Configuring a JNDI Datasource

Java Naming and Directory Interface (JNDI) is a Java Application Programming Interface (API) that provides naming and directory functionality for Java software clients, to discover and look up data and objects via a name. It helps decoupling object creation from the object look-up. When you have registered a datasource with JNDI, others can discover it through a JNDI look-up and use it.

When adding a datasource, to expose a RDBMS datasource as a JNDI datasource, click **Expose as a JNDI Data Source** to display the JNDI fields as follows:

New Data Source

New Data Source							
Data Source Type*	RDBMS						
Name*	<input type="text"/>						
Description	<input type="text"/>						
Data Source Provider*	default						
Driver*	<input type="text"/>						
URL*	<input type="text"/>						
User Name	admin						
Password	*****						
<input type="checkbox"/> Expose as a JNDI Data Source							
<table border="1"> <tr> <td>Name</td> <td><input type="text"/></td> </tr> <tr> <td>Use Data Source Factory</td> <td><input type="checkbox"/></td> </tr> <tr> <td>JNDI Properties</td> <td>+ Add Property</td> </tr> </table>		Name	<input type="text"/>	Use Data Source Factory	<input type="checkbox"/>	JNDI Properties	+ Add Property
Name	<input type="text"/>						
Use Data Source Factory	<input type="checkbox"/>						
JNDI Properties	+ Add Property						
<input type="checkbox"/> Data Source Configuration Parameters							
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>							

Following are descriptions of the JNDI fields:

- **Name:** Name of the JNDI datasource that will be visible to others in object look-up.
- **Use Data Source Factory:** To make the datasource accessible from an external environment, you must use a datasource factory. When this option is selected, a reference object will be created with the defined datasource properties. The datasource factory will create the datasource instance based on the values of the reference object when accessing the datasource from an external environment. In the datasource configuration, this is set as: <jndiConfig useDataSourceFactory="true" >.
- **JNDI Properties:** Properties related to the JNDI datasource (such as password). When you select this option, set the following properties:
 - `java.naming.factory.initial`: Selects the registry service provider as the initial context.
 - `java.naming.provider.url`: Specifies the location of the registry when the registry is being used as the initial context.

Configuring the Datasource Connection Pool Parameters

When the server processes a database operation, it spawns a database connection from an associated datasource. After using this connection, the server returns it to the pool of connections. This is called datasource connection pooling. It is a recommended way to gain more performance/throughput in the system. In datasource connection pooling, the physical connection is not dropped with the database server, unless it becomes stale or the datasource connection is closed.

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

You can configure the datasource connection pool parameters, such as how long a connection is persisted in the pool, using the datasource configuration parameters section that appears in the product management console when creating a datasource. Click and expand the option as shown below:

Add New Data Source

New Data Source

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

Suspect Timeout

Alternate User Name Allowed

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

Parameter name	Description
Transaction isolation	<p>The default TransactionIsolation state of connections created by this pool are as follows:</p> <ul style="list-style-type: none"> • TRANSACTION_UNKNOWN • TRANSACTION_NONE • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE
Initial Size (int)	The initial number of connections created, when the pool is started. Default value is zero.
Max. Active (int)	Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.
Max. Idle (int)	Maximum number of connections that should be kept in the pool at all times. Default value is 8. Idle connections are checked periodically (if enabled), and connections that have been idle for longer than <code>minEvictableIdleTimeMillis</code> will be released. (also see testWhileIdle)
Min. Idle (int)	Minimum number of established connections that should be kept in the pool at all times. The connection pool can shrink below this number, if validation queries fail. Default value is zero. For more information, see testWhileIdle .
Max. Wait (int)	Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. Default value is 30000 (30 seconds).
Validation Query (String)	The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are <code>SELECT 1</code> (mysql), <code>select 1 from dual</code> (oracle), <code>SELECT 1</code> (MS Sql Server).
Test On Return (boolean)	Used to indicate if objects will be validated before returned to the pool. The default value is false. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. </div>
Test On Borrow (boolean)	Used to indicate if objects will be validated before borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. Default value is false. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. In order to have a more efficient validation, see validationInterval . </div>

Test While Idle (boolean)	<p>The indication of whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool. The default value is false and this property has to be set in order for the pool cleaner/test thread to run. For more information, see timeBetweenEvictionRunsMillis .</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. </div>
Time Between Eviction Runs Mills (int)	Number of milliseconds to sleep between runs of the idle connection validation/cleaner thread. This value should not be set under 1 second. It indicates how often we check for idle, abandoned connections, and how often we validate idle connections. The default value is 5000 (5 seconds).
Minimum Evictable Idle Time (int)	Minimum amount of time an object may sit idle in the pool before it is eligible for eviction. The default value is 60000 (60 seconds).
Remove Abandoned (boolean)	Flag to remove abandoned connections if they exceed the <code>removeAbandonedTimeout</code> . If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the <code>removeAbandonedTimeout</code> . Setting this to true can recover database connections from applications that fail to close a connection. For more information, see logAbandoned . The default value is false.
Remove Abandoned Timeout (int)	Timeout in seconds before an abandoned (in use) connection can be removed. The default value is 60 (60 seconds). The value should be set to the longest running query that your applications might have.
Log Abandoned (boolean)	Flag to log stack traces for application code which abandoned a connection. Logging of abandoned connections, adds overhead for every connection borrowing, because a stack trace has to be generated. The default value is false.
Auto Commit (boolean)	The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default. If not set, then the <code>setAutoCommit</code> method will not be called.
Default Read Only (boolean)	The default read-only state of connections created by this pool. If not set then the <code>setReadOnly</code> method will not be called. (Some drivers don't support read only mode. For example: Informix)
Default Catalog (String)	The default catalog of connections created by this pool.
Validator Class Name (String)	The name of a class which implements the <code>org.apache.tomcat.jdbc.pool.Validator</code> interface and provides a no-arg constructor (may be implicit). If specified, the class will be used to create a <code>Validator</code> instance, which is then used instead of any validation query to validate connections. The default value is null. An example value is <code>com.mycompany.project.SimpleValidator</code> .
Connection Properties (String)	Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be <code>[propertyName=property;]*</code> . The default value is null.
	i The <code>user</code> and <code>password</code> properties will be passed explicitly, so that they do not need to be included here.

Init SQL	Ability to run a SQL statement exactly once, when the connection is created.
JDBC Interceptors	Flexible and pluggable interceptors to create any customizations around the pool, the query execution and the result set handling.
Validation Interval (long)	To avoid excess validation, only run validation at most at this frequency - time in milliseconds. If a connection is due for validation, but has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).
JMX Enabled (boolean)	Register the pool with JMX or not. The default value is true.
Fair Queue (boolean)	Set to true, if you wish that calls to <code>getConnection</code> should be treated fairly in a true FIFO fashion. This uses the <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue</code> implementation for the list of the idle connections. The default value is true. This flag is required when you want to use asynchronous connection retrieval. Setting this flag ensures that threads receive connections in the order they arrive. During performance tests, there is a very large difference in how locks and lock waiting is implemented. When <code>fairQueue=true</code> , there is a decision making process based on what operating system the system is running. If the system is running on Linux (property <code>os.name=Linux</code>), then to disable this Linux specific behavior and still use the fair queue, simply add the property <code>org.apache.tomcat.jdbc.pool.FairBlockingQueue.ignoreOS=true</code> to your system properties, before the connection pool classes are loaded.
Abandon When Percentage Full (int)	Connections that have been abandoned (timed out) will not get closed and reported up, unless the number of connections in use are above the percentage defined by <code>abandonWhenPercentageFull</code> . The value should be between 0-100. The default value is zero, which implies that connections are eligible for closure as soon as <code>removeAbandonedTimeout</code> has been reached.
Max Age (long)	Time in milliseconds to keep this connection. When a connection is returned to the pool, the pool will check to see if the current time when connected, is greater than the <code>maxAge</code> that has been reached. If so, it closes the connection rather than returning it to the pool. The default value is zero, which implies that connections will be left open and no age check will be done upon returning the connection to the pool.
Use Equals (boolean)	Set to true, if you wish the <code>ProxyConnection</code> class to use <code>String.equals</code> , and set to false when you wish to use <code>==</code> when comparing method names. This property does not apply to added interceptors as those are configured individually. The default value is true.
Suspect Timeout (int)	Timeout value in seconds. Default value is zero. Similar to the <code>removeAbandonedTimeout</code> value, but instead of treating the connection as abandoned, and potentially closing the connection, this simply logs the warning if <code>logAbandoned</code> is set to true. If this value is equal or less than zero, no suspect checking will be performed. Suspect checking only takes place if the timeout value is larger than zero, and the connection was not abandoned, or if abandon check is disabled. If a connection is suspected, a warning message gets logged and a JMX notification will be sent.
Alternate User Name Allowed (boolean)	By default, the <code>jdbc-pool</code> will ignore the <code>DataSource.getConnection(username, password)</code> call, and simply return a previously pooled connection under the globally configured properties <code>username</code> and <code>password</code> , for performance reasons. The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the <code>DataSource.getConnection(username, password)</code> call, simply set the property <code>alternateUsernameAllowed</code> , to true. If you request a connection with the credentials <code>user1/password1</code> , and the connection was previously connected using different <code>user2/password2</code> , then the connection will be closed, and reopened with the requested credentials. This way, the pool size is still managed on a global level, and not on a per-schema level. The default value is false.

Configuring a Custom Datasource

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

Following are descriptions of the custom datasource fields:

- **Data Source Type:** Custom
- **Custom Data Source Type:** Specify whether the data is in a table or accessed through a query as described below.
- **Name:** Enter a unique name for this datasource
- **Description:** Description of the datasource
- **Configuration:** XML configuration of the datasource

Custom datasource type

When creating a custom datasource, specify whether the datasource type is DS_CUSTOM_TABULAR (the data is stored in tables), or DS_CUSTOM_QUERY (non-tabular data accessed through a query). More information about each type are explained below.

Custom tabular datasources

Tabular datasources are used for accessing tabular data, that is, the data is stored in rows in named tables that can be queried later. To implement tabular datasources, the interface `org.wso2.carbon.dataservices.core.custom.datasource.TabularDataBasedDS` is used. For more information, see a sample implementation of a tabular custom datasource at [InMemoryDataSource](#).

A tabular datasource is typically associated with a SQL data services query. WSO2 products use an internal SQL parser to execute SQL against the custom datasource. For more information, see a sample data service descriptor at [InMemoryDSSample](#). Carbon datasources also support tabular data with the `org.wso2.carbon.dataservices.core.custom.datasource.CustomTabularDataSourceReader` datasource reader implementation. If you have Data Services Server installed, for more information see the `<PRODUCT_HOME>\repository\conf\datasources\custom-datasources.xml` file, which is a sample Carbon datasource configuration.

Custom query datasources

Custom query-based datasources are used for accessing non-tabular data through a query expression. To implement query-based datasources, the `org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryBasedDS` interface is used. You can create any non-tabular datasource using the query-based approach. Even if the target datasource does not have a query expression format, you can create and use your own. For example, you can support any NoSQL type datasource using this type of a datasource.

For more information, see a sample implementation of a custom query-based datasource at [EchoDataSource](#), and a sample data service descriptor with custom query datasources in [InMemoryDSSample](#). Carbon datasources

also support query-based data with the `org.wso2.carbon.dataservices.core.custom.datasource.Cust` `omQueryDataSourceReader` datasource reader implementation. If you have Data Services Server installed, for more information, see the `<PRODUCT_HOME>\repository\conf\datasources\custom-datasources.xml` file, which is a sample Carbon datasource configuration.

In the `init` methods of all custom datasources, user-supplied properties will be parsed to initialize the datasource accordingly. Also, a property named `<__DATASOURCE_ID__>`, which contains a UUID to uniquely identify the current datasource, will be passed. This can be used by custom datasource authors to identify the datasources accordingly, such as datasource instances communicating within a server cluster for data synchronization.

Shown below is an example configuration of a custom datasource of type `<DS_CUSTOM_TABULAR>`:

New Data Source

New Data Source

Data Source Type*	Custom
Custom Data Source Type*	DS_CUSTOM_TABULAR
Name*	customdatasource
Description	Custom data source
Configuration	<pre> 1 <configuration> 2 <customDataSourceClass>org.wso2.carbon.dataservices.core.custom.da 3 <customDataSourceProps> 4 <property name="inmemory_datasource_schema">{Vehicles:[ID,M 5 <property name="inmemory_datasource_records"> 6 {Vehicles:[["S10_1678","Harley Davidson Ultimate Choppe 7 ["S10_1949","Alpine Renault 1300","Classic Cars","1952" 8 ["S10_2016","Moto Guzzi 1100i","Motorcycles","1996"],[9 ["S10_4698","Harley-Davidson Eagle Drag Bike","Motorcyc 10 ["S10_4757","Alfa Romeo GTA","Classic Cars","1972"], 11 ["S10_4962","Lancia A Delta 16V","Classic Cars","1962"], 12 ["S12_1099","Ford Mustang","Classic Cars","1968"], 13 ["S12_1108","Ferrari Enzo","Classic Cars","2001"]]} 14 </property> 15 </customDataSourceProps> 16 </configuration> </pre>

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

Managing Users and Roles

Before you begin, note the following:

- Only system administrators can add, modify and remove users and roles. To set up administrators, see [Real m Configuration](#).
- Your product has a primary user store where the users/roles that you create using the management console are stored by default. It's default RegEx configurations are as follows. RegEx configurations ensure that parameters like the length of a user name/password meet the requirements of the user store.

```

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

```

When creating users/roles, if you enter a username, password etc. that does not conform to the RegEx configurations, the system throws an exception. You can either change the RegEx configuration or enter values that

conform to the RegEx. If you change the default user store or set up a secondary user store, configure the RegEx accordingly under the user store manager configurations in <APIM_HOME>/repository/conf/user-mgt.xml file.

This chapter contains the following information:

- [Adding User Roles](#)
- [Adding Users](#)

Adding User Roles

Roles contain permissions for users to manage the server. They can be reused and they eliminate the overhead of granting permissions to users individually.

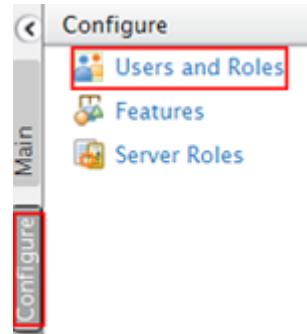
Throughout this documentation, we use the following roles that are typically used in many enterprises. You can also define different user roles depending on your requirements.

- **admin:** The API management provider who hosts and manages the [API Gateway](#). S/he is responsible for creating user roles in the system, assign them roles, managing databases, security etc. The Admin role is available by default with credentials admin/admin.
- **creator:** A creator is typically a person in a technical role who understands the technical aspects of the API (interfaces, documentation, versions etc.) and uses the [API publisher](#) to provision APIs into the API store. The creator uses the API Store to consult ratings and feedback provided by API users. Creator can add APIs to the store but cannot manage their lifecycle.
- **publisher:** A person in a managerial role and overlooks a set of APIs across the enterprise and controls the API lifecycle, subscriptions and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.
- **subscriber:** A user or an application developer who searches the [API store](#) to discover APIs and use them. S/he reads the documentation and forums, rates/comments on the APIs, subscribes to APIs, obtains access tokens and invokes the APIs.

Follow the instructions below to create the creator, publisher and subscriber roles in the API Manager.

Create user roles

1. Log in to the management console (<https://localhost:9443/carbon>) as admin (default credentials are admin/admin).



2. Select **Users and Roles** under the **Configure** menu.
3. In the **User Management** page that opens, click **Roles**.

User Management

System User Store

Users

Roles

Change Password

Change My Password

4. Click **Add** **New** **Role**.

Roles

Search

Select Domain: ALL-USER-STORE-DOMAINS

Enter role name pattern (* for all)

Name	Actions
admin	
Internal/everyone	
Internal/subscriber	
+ Add New Role	
+ Add New Internal Role	

5. Enter the name of the user role (e.g., creator) and click **Next**.

Step 1 : Enter role details

Enter role details

Domain: PRIMARY

Role Name*: creator

Next > Finish Cancel

Tip: The **Domain** drop-down list contains all user stores configured in the system. By default, you only have the PRIMARY user store. To configure secondary user stores, see [Configuring Secondary User Stores](#).

6. The permissions page opens. Select the permissions according to the role that you create. The table below lists the permissions of the `creator`, `publisher` and `subscriber` roles:

Roles	Permissions	UI
creator	<ul style="list-style-type: none"> Configure > Governance and all underlying permissions. Login Manage > API > Create Manage > Resources > Govern and all underlying permissions 	

publisher	<ul style="list-style-type: none"> • Login • Manage > API > Publish 	
subscriber	<ul style="list-style-type: none"> • Login • Manage > API > Subscribe 	

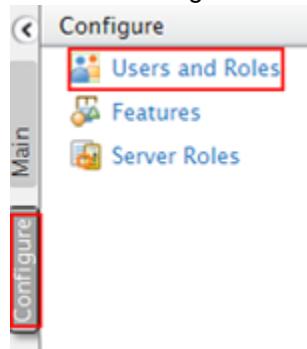
7. Click **Finish** once you are done adding permission.

Adding Users

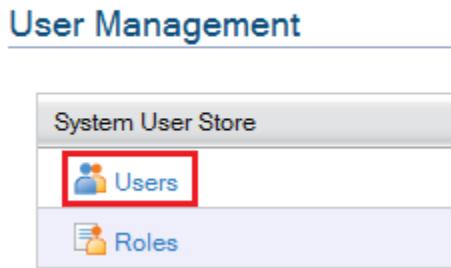
Users are consumers who interact with your enterprise's applications, databases or any other systems. These users can be persons, devices or applications/programs within or outside of the enterprise's network. Since these users interact with internal systems and access data, the need to define which user is allowed to do what is critical. This is called user management.

Follow the steps below to create users and assign them to roles. Also see how to add users via [e-mail](#) or a [social network](#) login.

to the Management Console and select **Users and Roles** from the **Configure** menu.



2. Click **Users** in the **User Management** window that opens.
[Home](#) > [Configure](#) > [Users and Roles](#)



⚠ The **Users** link is only visible to admins .

3. Click **Add** **New** **User** .

Users

Search

Select Domain

Enter user name pattern (* for all)

Name	Actions
admin	<input type="button" value="Change Password"/> <input type="button" value="Assign Roles"/> <input type="button" value="View Roles"/> <input type="button" value="User Profile"/>

[Add New User](#)

[Bulk Import Users](#)

4. The **Add User** page opens. Provide the username and password and click **Next**.

Add User

Step 1 : Enter user name

Enter user name

Domain	<input type="button" value="PRIMARY"/>
User Name*	<input type="text" value="user1"/>
Password*	<input type="password" value="*****"/>
Password Repeat*	<input type="password" value="*****"/>

Tip: The **Domain** drop-down list contains all user stores configured in the system. By default, you only have the **PRIMARY** user store. To configure secondary user stores, see [Configuring Secondary User Stores](#).

Tip: See how to create a user with the e-mail as the username:

5. Select the roles you want to assign to the user. In this example, we assign the `creator` role defined in the `previous` section.

Add User

Step 2 : Select roles of the user

Enter role name pattern (* for all)

Users of Role

[Select all on this page](#) | [Unselect all on this page](#)

- admin
- creator
- Internal/everyone
- Internal/subscriber

6. Click **Finish** to complete. The new user appears in the **Users** list.

Users

Enter user name pattern (* for all)

Name	Actions
admin	Change Password Roles
apicreator	Change Password Roles Delete

From here, you can change the user's password, assign different roles or delete it.

You cannot change the user name of an existing user.

Create a user with the e-mail as the username

When adding a user, if you provided an e-mail address as the username, modify the following files:

File	Modification
<AM_HOME>/repository/conf/carbon.xml	Set the <EnableEmailUserName> element to true
<AM_HOME>/repository/conf/api-manager.xml	<pre> <LoginConfig> <UserIdLogin primary="true"> <ClaimUri></ClaimUri> </UserIdLogin> <EmailLogin primary="false"> <ClaimUri>http://wso2.org/claim</ClaimUri> </EmailLogin> </LoginConfig> </pre>

<AM_HOME>/repository/conf/user-mgt.xml

Set two properties as:

```

<UserStoreManager class="org.wso2.carbon.us
...
<Property name="IsEmailUserName">true</I
<Property
name="UsernameJavaRegEx">>^[_~!#$;%^*+={}\\|_
...
</UserStoreManager>

```

Note that if you already have an element by the name define the regular expression, it takes priority over the <U

 E-mail login does not work for any tenant including the super tenant in a **multi-tenant environment**. This facility is currently only available in single tenant mode (i.e., users of the `carbon.super` tenant only).

Add users from a social network login

You can auto provision users based on a **social network login** by integrating the API Manager with WSO2 Identity Server. But, this is not supported in a **multi-tenant environment**.

In a multi-tenant environment, the system cannot identify the tenant domain in the login request that comes to API Manager's Publisher/Store. Therefore, the service provider is registered as a SaaS application within the super tenant's space. Configuring user provisioning is part of creating the service provider. In order to authenticate the user through a third party identity provider such as a social network login, you must enable identity federation. As the service provider is created in the super tenant's space, the provisioned user is also created within the super tenant's space. As a result, it is not possible to provision the user in the tenant's space.

To overcome this limitation, you can write a custom authenticator to retrieve the tenant domain of the user and write a custom login page where the user can enter the tenant domain, which is then added to the authenticator context. Then, write a custom provisioning handler to provision the user in the tenant domain that maintained in the context.

- For information on writing a custom authenticator, see [Creating Custom Authenticators in the WSO2 IS documentation](#).
- For information on writing a custom login page, see [Customizing Login Pages in the WSO2 IS documentation](#).

Configuring User Stores

A user store is the database where information of the users and/or user roles is stored. User information includes log-in name, password, first name, last name, e-mail etc.

All WSO2 products have an embedded H2 database except for WSO2 Identity Server, which has an embedded LDAP as its user store. Permission is stored in a separate database called the user management database, which by default is H2. However, users have the ability to connect to external user stores as well.

The user stores of Carbon products can be configured to operate in either one of the following modes.

- User store operates in read/write mode - In Read/Write mode, WSO2 Carbon reads/writes into the user store.
- User store operates in read only mode - In Read Only mode, WSO2 Carbon guarantees that it does not modify any data in the user store. Carbon maintains roles and permissions in the Carbon database but it can read users/roles from the configured user store.

The sections below provide configuration details:

- [Realm Configuration](#)
- [Changing the RDBMS](#)
- [Configuring Primary User Stores](#)

- Configuring Secondary User Stores

Realm Configuration

The `<Configuration>` section at the top of the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file allows you to specify basic configuration for connecting to this user store (also called a **realm**).

```

<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="dataSource">jdbc/WSO2CarbonDB</Property>
  </Configuration>
  ...
</Realm>

```

Note the following regarding the configuration above.

Element	Description
<code><AddAdmin></code>	When <code>true</code> , this element creates the admin user based on the <code>AdminUser</code> element. It also indicates whether to create the specified admin user if it doesn't already exist. When connecting to an external read-only LDAP or Active Directory user store, this property needs to be <code>false</code> if an admin user and admin role exist within the user store. If the admin user and admin role do not exist in the user store, this value should be <code>true</code> , so that the role is added to the user management database. However, if the admin user is not there in the user store, we must add that user to the user store manually. If the <code>AddAdmin</code> value is set to <code>true</code> in this case, it will generate an exception.
<code><AdminRole>wso2admin</AdminRole></code>	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.
<code><AdminUser></code>	Configures the default administrator for the WSO2 product. If the user store is read-only, the admin user must exist in the user store or the system will not start. If the external user store is read-only, you must select a user already existing in the external user store and add it as the admin user that is defined in the <code><AdminUser></code> element. If the external user store is in read/write mode, and you set <code><AddAdmin></code> to <code>true</code> , the user you specify will be automatically created.

<UserName>	This is the username of the default administrator or super tenant of the user store. If the user store is read-only, the admin user MUST exist in the user store for the process to work.
<Password>	Do NOT put the password here but leave the default value as it is if the user store is read-only as this element and its value are ignored. This password is used only if the user store is read-write and the <code>AddAdmin</code> value is set to <code>true</code> .
	<p> Note that the password in the <code>user-mgt.xml</code> file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the <code>user-mgt.xml</code> file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the <code>user-mgt.xml</code> file. To change the admin password, you must use the Change Password option from the management console.</p>

The main property given below contains details of the database connection.

Property Name	Description	Mandatory
dataSource	Data sources are configured in the <code><PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml</code> file. This property indicates the relevant data source configuration for the User Management database.	Mandatory

Given below are optional properties that can be used.

Property Name	Description
testOnBorrow	It is recommended to set this property to 'true' so that object is validated before being borrowed from the JDBC pool. For this property, the <code>validationQuery</code> parameter in the <code><PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml</code> file should be a non-null setting. This will avoid connection failures. See the section on performance products for more information.
CaseSensitiveAuthorizationRules	Permissions, and the rules (role name, action, resource) linked to each user are stored in the RDBMS of the server. By default, these rules are not case sensitive. This property can be used if you want to make the rules case sensitive.

Changing the RDBMS

The default database of user manager is the H2 database that comes with WSO2 products. You can configure it to point to databases by other vendors.

1. Add the JDBC driver to the classpath by dropping the JAR into `<PRODUCT_HOME>/repository/components/lib`.
2. Change values of properties given in on the [Realm Configuration](#) page appropriately.
3. Create the database by running the relevant script in `<PRODUCT_HOME>/dbscript` and restart the server.

- **For Linux:** sh wso2server.sh or sh wso2server.sh -Dsetup
- **For Windows:** wso2server.bat or wso2server.bat -Dsetup

Configuring Primary User Stores

Every WSO2 product comes with an embedded, internal user store, which is configured in <PRODUCT_HOME>/repository/conf/user-mgt.xml. In WSO2 Identity Server, the embedded user store is LDAP, and in other products it is JDBC. Because the domain name (unique identifier) of this default user store is set to PRIMARY by default, it is called the primary user store.

Instead of using the embedded user store, you can set your own user store as the primary user store. Since the user store you want to connect to might have different schemas from the ones available in the embedded user store, it needs to go through an adaptation process. WSO2 products provide the following adapters to enable you to authenticate users from different types of user stores and plug into LDAP, Active Directory, and JDBC to perform authentication:

User store manager class	Description
org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager	Use ReadOnlyLDAPUserManager to connect to external LDAP user stores.
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAPUserManager to connect to both read and write operations. Uncommented in the code in ReadOnlyLDAPUserManager.
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUserStoreManager to connect to Microsoft Domain Service (AD DS) or LDS. This can be used only for read-only operations; you must use org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager.
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreManager to connect to JDBC user stores.

The user-mgt.xml file already has sample configurations for all of the above user stores. To enable these configurations, you must uncomment them in the code and comment out the ones that you do not need.

The following topics provide details on the various primary user stores you can configure.

- Configuring an external LDAP or Active Directory user store
- Configuring an internal/external JDBC user store

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

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

Configuring an external LDAP or Active Directory user store

All WSO2 products can read and write users and roles from external Active Directory or LDAP user stores. You can configure WSO2 products to access these user stores in one of the following modes:

- Read-only mode
- Read/write mode

Read-only mode

Before you begin

- If you create the `user-mgt.xml` file yourself, be sure to save it in the `<PRODUCT_HOME>/repository/conf` directory.
- The class attribute for a read-only LDAP is `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager">`

When you configure a product to read users/roles from your company LDAP in read-only mode, it does not write any data into the LDAP.

1. Comment out the following user store which is enabled by default in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file.
`<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">`
2. Given below is a sample for the LDAP user store. This configuration is found in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file, however, you need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.

```

<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="UserSearchBase">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="MaxUserNameListLength">100</Property>
        <Property name="SCIMEnabled">false</Property>
      </UserStoreManager>
    </Realm>
  </UserManager>

```

a. Update the connection details to match your user store. For example:

```
<Property name="ConnectionURL">ldap://localhost:10389</Property>
```

b. Obtain a user who has permission to read all users/attributes and perform searches on the user store from your LDAP/Active Directory administrator. For example, if the privileged user is "AdminLDAP" and the password is "2010#Avrudu", update the following sections of the realm configuration as follows:

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

c. 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>
```

d. Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute" />` and `<Property name="UserNameSearchFilter" />` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

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

e. Set the `ReadGroups` property to 'true', if it should be allowed to read roles from this user store. When this property is 'true', you must also specify values for the `GroupSearchBase`, `GroupSearchFilter` and `GroupNameAttribute` properties as shown in the following example:

```
<Property name="ReadGroups">true</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
```

If the `ReadGroups` property is set to 'false', only Users can be read from the user store.

f. Optionally, configure the realm to read roles from the user store by reading the user/role mapping based on a membership (user list) or backlink attribute. The following code snippet represents reading roles based on a membership attribute. This is used by the ApacheDirectory server and OpenLDAP

```
<Property name="ReadLDAPGroups">false</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MembershipAttribute">member</Property>
```

g. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the user store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.

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

Read/write mode



Before you begin

- To read and write to an Active Directory user store, set the `WriteGroups` property to `true` instead of `false`.
- To write user entries to an LDAP user store (roles are not written, just user entries), you follow the steps in the [Read-only mode](#) section but specify the following class instead:

```
<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">
```

- Use the following class for Active Directory.

```
<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
```

The `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file has commented-out configurations for external LDAP/AD user stores.

1. Enable the `<ReadWriteLDAPUserStoreManager>` or the `<ActiveDirectoryUserStoreManager>` in the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file by uncommenting the code. When it is enabled, the user manager reads/writes into the LDAP/AD user store. Note that these configurations already exist in the `user-mgt.xml` file so you only need to uncomment them and make the appropriate adjustments. Also ensure that you comment out the configurations for other user stores which you are not using.
2. The default configuration for the external read/write user store in the `user-mgt.xml` file is as follows. Change the values according to your requirements.

LDAP User StoreActive Directory User Store

LDAP user store sample:

```

<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:${Ports.EmbeddedLDAP.LDAPServerPort}</Property>
    <Property name="ConnectionName">uid=admin,ou=system</Property>
    <Property name="ConnectionPassword">admin</Property>
    <Property name="PasswordHashMethod">SHA</Property>
    <Property name="UserNameListFilter">(objectClass=person)</Property>
    <Property name="UserEntryObjectClass">wso2Person</Property>
    <Property name="UserSearchBase">ou=Users,dc=wso2,dc=org</Property>
    <Property
name="UserNameSearchFilter">(&amp;(objectClass=person)(uid=?))</Property>
    <Property name="UserNameAttribute">uid</Property>
    <Property name="PasswordJavaScriptRegEx">[\S]{5,30}</Property>
    <Property name="UsernameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="UsernameJavaRegEx">^[\~!@#$%^+={}\\|\\\\\\&lt;,&gt;,\\"]{3,30}$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}</Property>
    <Property
name="RolenameJavaRegEx">^[\~!@#$%^+={}\\|\\\\\\&lt;,&gt;,\\"]{3,30}$</Property>
    <Property name="ReadLDAPGroups">true</Property>
    <Property name="WriteLDAPGroups">true</Property>
    <Property name="EmptyRolesAllowed">true</Property>
    <Property name="GroupSearchBase">ou=Groups,dc=wso2,dc=org</Property>
    <Property name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
    <Property name="GroupEntryObjectClass">groupOfNames</Property>
    <Property
name="GroupNameSearchFilter">(&amp;(objectClass=groupOfNames)(cn=?))</Property>
    <Property name="GroupNameAttribute">cn</Property>
    <Property name="SharedGroupNameAttribute">cn</Property>
    <Property
name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property>
    <Property name="SharedGroupEntryObjectClass">groups</Property>
    <Property
name="SharedTenantNameListFilter">(object=organizationalUnit)</Property>
    <Property name="SharedTenantNameAttribute">ou</Property>
    <Property name="SharedTenantObjectClass">organizationalUnit</Property>
    <Property name="MembershipAttribute">member</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
    <Property name="UserDNPattern">uid={0},ou=Users,dc=wso2,dc=org</Property>
</UserStoreManager>

```

 **Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up- `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

Active directory user store sample:

```

<UserStoreManager
class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager">
    <Property
name="TenantManager">org.wso2.carbon.user.core.tenant.CommonHybridLDAPTenantManager</Property>
        <Property name="defaultRealmName">WSO2.ORG</Property>
        <Property name="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="userAccountControl">512</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>
                <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>
                    <Property name="MaxUserNameListLength">100</Property>
                    <Property name="SCIMEnabled">false</Property>
            </UserStoreManager>

```

 **Tip:** Be sure to set the `EmptyRolesAllowed` property to true. If not, you will get the following error at start up- `APIManagementException: Error while creating subscriber role: subscriber - Self registration might not function properly.`

 When working with Active Directory it is best to enable the `GetAllRolesOfUserEnabled` property in the `AuthorizationManager` as follows.

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

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

 If you create the `user-mgt.xml` file yourself, be sure to save it in the `<PRODUCT_HOME>/repository/conf` directory.

The `class` attribute of the `UserStoreManager` element indicates whether it is an Active Directory or LDAP user store:

- Active Directory: `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager" />`
- Read-only LDAP: `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadOnlyLDAPUserStoreManager" />`

3. Set the attribute to use as the username, typically either `cn` or `uid` for LDAP. Ideally, `<Property name="UserNameAttribute">` and `<Property name="UserNameSearchFilter">` should refer to the same attribute. If you are not sure what attribute is available in your user store, check with your LDAP/Active Directory administrator.

For example:

LDAP Active Directory

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

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

4. The following code snippet represents reading roles based on a backlink attribute. This is used by the Active Directory.

```

<Property name="ReadLDAPGroups">true</Property>
<Property name="GroupSearchBase">cn=users,dc=wso2,dc=lk</Property>
<Property name="GroupSearchFilter">(objectcategory=group)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MemberOfAttribute">memberOf</Property>

```

5. For Active Directory, you can use `<Property name="Referral">follow</Property>` to enable referrals within the user store. The AD user store may be partitioned into multiple domains. However, according to the user 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.
6. Start your server and try to log in as the admin user you specified. The password is the admin user's password in the LDAP server.

i When configuring an external LDAP for Governance Registry or API Manager, the user name and password for the default admin will change to the LDAP admin. As a result, the `<PRODUCT_HOME>/repository/conf/api-manager.xml` file must be updated with the new LDAP admin credentials.

Configuring an internal/external JDBC user store

The default internal JDBC user store reads/writes into the internal database of the Carbon server. JDBC user stores can be configured using the `<PRODUCT_HOME>/repository/conf/user-mgt.xml` file's `JDBCUserStoreManager` configuration section. Additionally, all Carbon-based products can work with an external RDBMS. You can configure Carbon to read users/roles from your company RDBMS and even write to it. Therefore, in this scenario, the user core connects to two databases:

- The Carbon database where authorization information is stored internally.
- Your company database where users/roles reside.

Therefore, the `user-mgt.xml` file must contain details for two database connections. The connection details mentioned earlier are used by the authorization manager. If we specify another set of database connection details inside the `UserStoreManager`, it reads/writes users to that database. The following are step-by-step guidelines for connecting to an internal and external JDBC user store in read-only mode:

1. Uncomment the following section in `<PRODUCT_HOME>/repository/conf/user-mgmt.xml`:

```

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

```

The following are samples for the internal and external JDBC user store configuration:

[Internal JDBC User Store](#)[External JDBC User Store](#)

Internal JDBC user store configuration sample:

```

<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
  <Property
  name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
  <Property name="ReadOnly">false</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="IsEmailUserName">false</Property>
  <Property name="DomainCalculation">default</Property>
    <Property name="PasswordDigest">SHA-256</Property>
  <Property name="StoreSaltedPassword">true</Property>
  <Property name="UserNameUniqueAcrossTenants">false</Property>
  <Property name="PasswordJavaRegEx">[\S]{5,30}\$</Property>
  <Property name="PasswordJavaScriptRegEx">[\S]{5,30}\$</Property>
  <Property
  name="UsernameJavaRegEx">^\~!#%;^*+={}\\|\\\\&lt;&gt;,\\\"]{3,30}\$</Property>
  <Property name="UsernameJavaScriptRegEx">[\S]{3,30}\$</Property>
  <Property
  name="RolenameJavaRegEx">^\~!@#%;^*+={}\\|\\\\&lt;&gt;,\\\"]{3,30}\$</Property>
    <Property name="RolenameJavaScriptRegEx">[\S]{3,30}\$</Property>
    <Property name="UserRolesCacheEnabled">true</Property>
</UserStoreManager>

```

External JDBC user store configuration sample:

```

<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
  <Property
  name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManager</Property>
  <Property name="driverName">com.mysql.jdbc.Driver</Property>
  <Property name="url">jdbc:mysql://localhost:3306/tcsdev</Property>
  <Property name="userName">shavantha</Property>
  <Property name="password">welcome</Property>
  <Property name="Disabled">false</Property>
  <Property name="MaxUserNameListLength">100</Property>
  <Property name="MaxRoleNameListLength">100</Property>
  <Property name="UserRolesCacheEnabled">true</Property>
  <Property name="PasswordDigest">SHA-256</Property>
  <Property name="ReadGroups">true</Property>
  <Property name="ReadOnly">false</Property>
  <Property name="IsEmailUserName">false</Property>
  <Property name="DomainCalculation">default</Property>
  <Property name="StoreSaltedPassword">true</Property>
  <Property name="WriteGroups">false</Property>
  <Property name="UserNameUniqueAcrossTenants">false</Property>
  <Property name="PasswordJavaRegEx">^\S{5,30}\$</Property>
  <Property name="PasswordJavaScriptRegEx">^\S{5,30}\$</Property>
  <Property name="UsernameJavaRegEx">^\S{5,30}\$</Property>
  <Property name="UsernameJavaScriptRegEx">^\S{5,30}\$</Property>
  <Property name="RolenameJavaRegEx">^\S{5,30}\$</Property>
  <Property name="RolenameJavaScriptRegEx">^\S{5,30}\$</Property>
  <Property name="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="UserRoleSQL">SELECT UM_ROLE_NAME FROM UM_USER_ROLE,
UM_ROLE, UM_USER WHERE UM_USER.UM_USER_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID
AND UM_USER_ROLE.UM_TENANT_ID=? AND UM_ROLE.UM_TENANT_ID=? AND
UM_USER.UM_TENANT_ID=?</Property>
    <Property name="UserSharedRolesSQL">SELECT UM_ROLE_NAME,
UM_ROLE.UM_TENANT_ID, UM_SHARED_ROLE FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER
ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER JOIN UM_ROLE ON
UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE UM_USER.UM_USER_NAME = ? AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = ?</Property>
    <Property name="IsRoleExistingSQL">SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name="GetUserListOfRoleSQL">SELECT UM_USER_NAME FROM
UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_ROLE.UM_ROLE_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID
AND UM_USER_ROLE.UM_TENANT_ID=? AND UM_ROLE.UM_TENANT_ID=? AND
UM_USER.UM_TENANT_ID=?</Property>
    <Property name="GetUserListOfSharedRoleSQL">SELECT UM_USER_NAME FROM
UM_SHARED_USER_ROLE INNER JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID =
UM_USER.UM_ID INNER JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID =
UM_ROLE.UM_ID WHERE UM_ROLE.UM_ROLE_NAME= ? AND
UM_SHARED_USER_ROLE.UM_USER_TENANT_ID = UM_USER.UM_TENANT_ID AND
UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID = UM_ROLE.UM_TENANT_ID</Property>
    <Property name="IsUserExistingSQL">SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name=" GetUserPropertiesForProfileSQL">SELECT UM_ATTR_NAME,
UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID =
UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER.UM_USER_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
    <Property name=" GetUserPropertyForProfileSQL">SELECT UM_ATTR_VALUE FROM
UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID = UM_USER_ATTRIBUTE.UM_USER_ID AND
UM_USER.UM_USER_NAME=? AND UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
    <Property name=" GetUserListForPropertySQL">SELECT UM_USER_NAME FROM UM_USER,
UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID = UM_USER.UM_ID AND
UM_USER_ATTRIBUTE.UM_ATTR_NAME =? AND UM_USER_ATTRIBUTE.UM_ATTR_VALUE =? AND
UM_USER_ATTRIBUTE.UM_PROFILE_ID=? AND UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND
UM_USER.UM_TENANT_ID=?</Property>
    <Property name="GetProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM
UM_USER_ATTRIBUTE WHERE UM_TENANT_ID=?</Property>
    <Property name=" GetUserProfileNamesSQL">SELECT DISTINCT UM_PROFILE_ID FROM
UM_USER_ATTRIBUTE WHERE UM_USER_ID=(SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=? ) AND UM_TENANT_ID=?</Property>
    <Property name="GetUserIDFromUserNameSQL">SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
    <Property name="GetUserNameFromTenantIDSQl">SELECT UM_USER_NAME FROM
UM_USER WHERE UM_TENANT_ID=?</Property>
    <Property name="GetTenantIDFromUserNameSQL">SELECT UM_TENANT_ID FROM
UM_USER WHERE UM_USER_NAME=?</Property>
    <Property name="AddUserSQL">INSERT INTO UM_USER (UM_USER_NAME,
UM_USER_PASSWORD, UM_SALT_VALUE, UM_REQUIRE_CHANGE, UM_CHANGED_TIME,

```

```

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

```

```

<Property name="UserNameUniqueAcrossTenantsSQL">SELECT UM_ID FROM UM_USER
WHERE UM_USER_NAME=?</Property>
<Property name="IsDomainExistingSQL">SELECT UM_DOMAIN_ID FROM UM_DOMAIN
WHERE UM_DOMAIN_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="AddDomainSQL">INSERT INTO UM_DOMAIN (UM_DOMAIN_NAME,
UM_TENANT_ID) VALUES (?, ?)</Property>
<Property name="AddUserToRoleSQL-mssql">INSERT INTO UM_USER_ROLE
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?), (?)</Property>
<Property name="AddRoleToUserSQL-mssql">INSERT INTO UM_USER_ROLE
(UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT (SELECT UM_ID FROM UM_ROLE WHERE
UM_ROLE_NAME=? AND UM_TENANT_ID=?), (SELECT UM_ID FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?), (?)</Property>
<Property name="AddUserPropertySQL-mssql">INSERT INTO UM_USER_ATTRIBUTE
(UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) SELECT
(SELECT UM_ID FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?), (?), (?),
(?), (?)</Property>
<Property name="AddUserToRoleSQL-openedge">INSERT INTO UM_USER_ROLE
(UM_USER_ID, UM_ROLE_ID, UM_TENANT_ID) SELECT UU.UM_ID, UR.UM_ID, ? FROM UM_USER
UU, UM_ROLE UR WHERE UU.UM_USER_NAME=? AND UU.UM_TENANT_ID=? AND
UR.UM_ROLE_NAME=? AND UR.UM_TENANT_ID=?</Property>
<Property name="AddRoleToUserSQL-openedge">INSERT INTO UM_USER_ROLE
(UM_ROLE_ID, UM_USER_ID, UM_TENANT_ID) SELECT UR.UM_ID, UU.UM_ID, ? FROM UM_ROLE
UR, UM_USER UU WHERE UR.UM_ROLE_NAME=? AND UR.UM_TENANT_ID=? AND
UU.UM_USER_NAME=? AND UU.UM_TENANT_ID=?</Property>
<Property name="AddUserPropertySQL-openedge">INSERT INTO UM_USER_ATTRIBUTE
(UM_USER_ID, UM_ATTR_NAME, UM_ATTR_VALUE, UM_PROFILE_ID, UM_TENANT_ID) SELECT
UM_ID, ?, ?, ?, ? FROM UM_USER WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>

```

```

<Property name="DomainName">wso2.org</Property>
<Property name="Description"/>
</UserStoreManager>

```

i The sample for the external JDBC user store consists of properties pertaining to various SQL statements. This is because the schema may be different for an external user store, and these adjustments need to be made in order to streamline the configurations with WSO2 products.

i You can define a data source in <PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml and refer to it from the user-mgt.xml file. This takes the properties defined in the master-datasources.xml file and reuses them in the user-mgt.xml file. To do this, you need to define the following property:

```
<Property name = "dataSource">jdbc/WSO2CarbonDB</Property>
```

2. Find a valid user that resides in the RDBMS. For example, say a valid username is AdminSOA. Update the Admin user section of your configuration as follows. You do not have to update the password element; leave it as is.

```

<AdminUser>
  <UserName>AdminSOA</UserName>
  <Password>XXXXXX</Password>
</AdminUser>

```

3. Add the PasswordHashMethod property to the UserStoreManager configuration for JDBCUserStoreManager. For example:

```

<UserStoreManager class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
  <Property name="PasswordHashMethod">SHA</Property>
  ...
</UserStoreManager>

```

The PasswordHashMethod property specifies how the password should be stored. It usually has the following values:

- SHA - Uses SHA digest method.
- MD5 - Uses MD 5 digest method.
- PLAIN_TEXT - Plain text passwords.

In addition, it also supports all digest methods in <http://docs.oracle.com/javase/6/docs/api/java/security/Mess ageDigest.html>.

4. Update the connection details found within the <UserStoreManager> class based on your preferences.
5. 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>

```

6. Add the JDBC driver to the classpath by copying its JAR file into the <PRODUCT_HOME>/repository/components/lib directory.
7. Edit the SQLs in the user-mgt.xml file according to your requirements, and then start the server.

Related Links

[Properties of Primary User Stores](#) - For a comprehensive understanding on the configuration details.

Properties of Primary User Stores

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

Property name	Description
MaxUserNameListLength	Controls the number of users listed in the user store of a WSO2 product. If you want to list them all, Setting this property to 0 displays all users.
ConnectionURL	Connection URL to the user store server. In the case of default port, reference to that port is included in this configuration.
ConnectionName	The username used to connect to the database and perform various operations on the user store or have an administrator role in the WSO2 product that you can use to view users' attributes and to perform search operations on the user store. This attribute is mandatory.
ConnectionPassword	Password for the ConnectionName user.
PasswordHashMethod	Password hash method to use when storing user entries in the user store.
UserNameListFilter	Filtering criteria for listing all the user entries in the user store. In this case, the search operation only provides the objects created from the users in the management console.
UserEntryObjectClass	Object class used to construct user entries. By default, it is a custom class.
UserSearchBase	DN of the context or object under which the user entries are stored. When performing user store searches for users, it will start from this location of the directory.
	<i>Information</i> Different databases have different search bases.
UserNameSearchFilter	Filtering criteria used to search for a particular user entry.
UserNameAttribute	The attribute used for uniquely identifying a user entry. Users can be identified by this attribute.
	<i>Information</i> The name of the attribute is considered as the username.
UsernameWithEmailJavaScriptRegEx	This property defines the JavaScript regular expression pattern which is used to validate the user name in the configuration file. If you need to support both email as a user name and user name, you can define two regular expressions.
	<pre><Property name="UsernameWithEmailJavaScriptRegEx"></pre>
PasswordJavaScriptRegEx	Policy that defines the password format.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user name validation.

UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.
	<pre><Property name="UsernameJavaRegEx">[a-zA-Z0-9 ._- !#\$%"' *-=?^</pre>
RolenameJavaScriptRegEx	The regular expression used by the front-end components for role name validation.
RolenameJavaRegEx	A regular expression used to validate role names. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.
ReadGroups	Specifies whether groups should be read from the user store. If this is set to true, then the underlying user store can be read, and the following group configurations are NOT mandatory.
WriteGroups	Specifies whether groups should be written to user store.
EmptyRolesAllowed	Specifies whether the underlying user store allows empty groups to be created. By default, it is false, which means that empty groups are allowed to be created. Usually LDAP servers do not allow empty groups.
GroupSearchBase	DN of the context under which user entries are stored in the user store.
GroupSearchFilter	The query used to search for groups.
GroupNameListFilter	Filtering criteria for listing all the group entries in the user store. Group search operation only returns objects created from this class.
GroupEntryObjectClass	Object class used to construct group entries.
GroupNameSearchFilter	Filtering criteria used to search for a particular group entry.
GroupNameAttribute	Attribute used for uniquely identifying a user entry. This attribute is mandatory.
MembershipAttribute	Attribute used to define members of groups.
UserRolesCacheEnabled	This is to indicate whether to cache the role list of a user. By default, it is true. If this is false, then changes made to the user roles through external means and those changes should be instantly reflected in the user store.
UserDNPattern	(LDAP) The pattern for the user's DN, which can be defined to improve search performance. Note that defining a UserDNPattern provides more impact on performance than using a GroupSearchFilter.
ReplaceEscapeCharactersAtUserLogin	(LDAP) If the user name has special characters it replaces it to valid characters.
TenantManager	Includes the location of the tenant manager.
ReadOnly	(LDAP and JDBC) Indicates whether the user store of this realm is read-only.
IsEmailUserName	(JDBC) Indicates whether the user's email is used as their username.
DomainCalculation	(JDBC) Can be either default or custom (this applies when the realm is JDBC based).
PasswordDigest	(JDBC) Digesting algorithm of the password. Has values such as, MD5, SHA1, SHA256, SHA512, etc.
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 password. By default, strings have to provide ranges of alphabets, numbers and also ranges of ASCII values.

PasswordJavaScriptRegEx	The regular expression used by the front-end components for password validation.
UsernameJavaRegEx	A regular expression to validate usernames. By default, strings have to be alphanumeric.
UsernameJavaScriptRegEx	The regular expression used by the front-end components for user name validation.
RolenameJavaRegEx	A regular expression to validate role names. By default, strings have to be alphanumeric.
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 realm specific configurations.
SharedGroupEnabled	This property is used to enable/disable the shared role functionality.
SharedGroupSearchBase	Shared roles are created for other tenants to access under the merged search base.
SharedTenantObjectClass	Object class for the shared groups created.
SharedTenantNameAttribute	Name attribute for the shared group.
SharedTenantNameListFilter	This is currently not used.

Configuring Secondary User Stores

The default configurations of WSO2 products have a single, embedded 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](#)

 **Tip:** If you set 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 `<APIM_HOME>/dbscripts` folder and run it on your database. It creates the necessary tables.

 **Tip:** If your setup has multiple product clusters such as the API Manager cluster, WSO2 ESB cluster etc, when you add a secondary user store in one cluster, it will be unknown to the other product clusters. This is because the WSO2 deployment synchroniser does not facilitate cross product synchronisation of artefacts. Therefore, you need to create the same user store in one node of each product cluster.

For example, if your setup has a WSO2 Identity Server instance serving an APIM cluster and an ESB cluster, the user store needs to be created in one APIM node and one ESB node.

Configuring using the management console

1. Log in to the management console and click **User Store Management** sub menu under **Configure** menu.
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. The **User Store Manager** page opens. Enter a unique domain name and fill in the rest of the data.

 Domain names must be unique and must not include underscore character (_).

For details on each property, see the respective property description that is provided. Also, select the

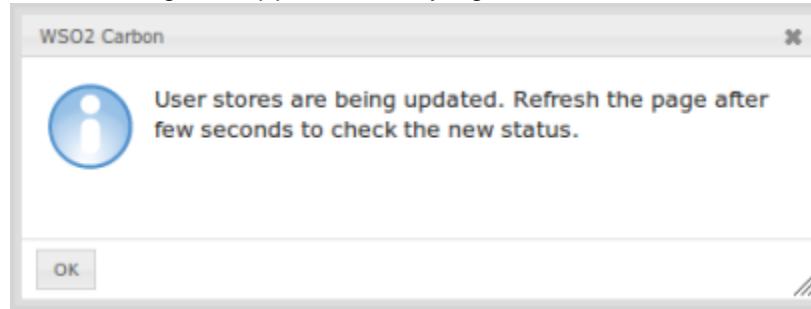
required implementation of user store manager from the **User Store Manager Class** drop-down list. The displayed property list varies depending on the selected user store manager implementation. By default, all WSO2 products come with four user store manager implementations as follows:

- ReadWriteLDAPUserStoreManager
- ReadOnlyLDAPUserStoreManager
- ActiveDirectoryUserStoreManager
- JDBCUserStoreManager

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](#).

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

5. Ensure that all the mandatory fields are filled and a valid domain name is given and click **Add**.
6. A message appears saying that the user stores are being added.



Note: The above message does not imply that the user store is added successfully. It simply means that the server is attempting to add the new user store to the end of the available chain of stores.

7. Refresh the page after a few seconds to check the status.
8. If the new user store is successfully added, it will appear in the **User Store Management** page.
9. After adding to the server, you can edit the properties of the new secondary user store and enable/disable it in a dynamic manner.

Configuring manually

By default, the configuration of the primary user store is saved in the `user-mgt.xml` file. When you create a secondary user store using the management console as explained above, its configuration is saved to an XML file with the same name as the domain name you specify. Alternatively, you can create this XML file manually and save it as follows:

- When you configure multiple user stores, you must **give a unique domain name to each user store** in the `<DomainName>` element. If you configure a user store without specifying a domain name, the server throws an exception at start up.
- If it is the configuration of a super tenant, save the secondary user store definitions in `<PRODUCT_HOME>/repository/deployment/server/userstores` directory.
- If it is a general tenant, save the configuration in `<PRODUCT_HOME>/repository/tenants/<tenantid>/userstores` directory.
- The 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.

Directing the Root Context to the API Store

WSO2 API Manager includes separate Web applications as the API Publisher and the API Store. The root context of the API Manager is set to go to the API Publisher by default. For example, assume that the API Manager is hosted on a domain named `apis.com` with default ports. The URLs of the API Store and API Publisher will be as follows:

- API Store - <https://apis.com:9443/store>
- API Publisher - <https://apis.com:9443/publisher>

If you open the root context, which is <https://apis.com:9443> in your browser, it directs to the API Publisher by default. You can set this to go to the API Store as follows:

1. Open the bundle `<AM_HOME>/repository/components/plugins/org.wso2.am.styles_1.x.x.jar`.
2. Open the `component.xml` file that is inside `META-INF` directory.
3. Change the `<context-name>` element, which points to publisher by default, to store:

```
<context>
    <context-id>default-context</context-id>
    <context-name>store</context-name>
    <protocol>http</protocol>
    <description>API Publisher Default Context</description>
</context>
```

4. Restart the server.
5. Open the default context (<https://apis.com:9443>) again in a browser and note that it directs to the API Store.

 **Tip:** If you want to configure the API Publisher and Store to pass proxy server requests, configure a [reverse proxy server](#).

Adding Links to Navigate Between the Store and Publisher

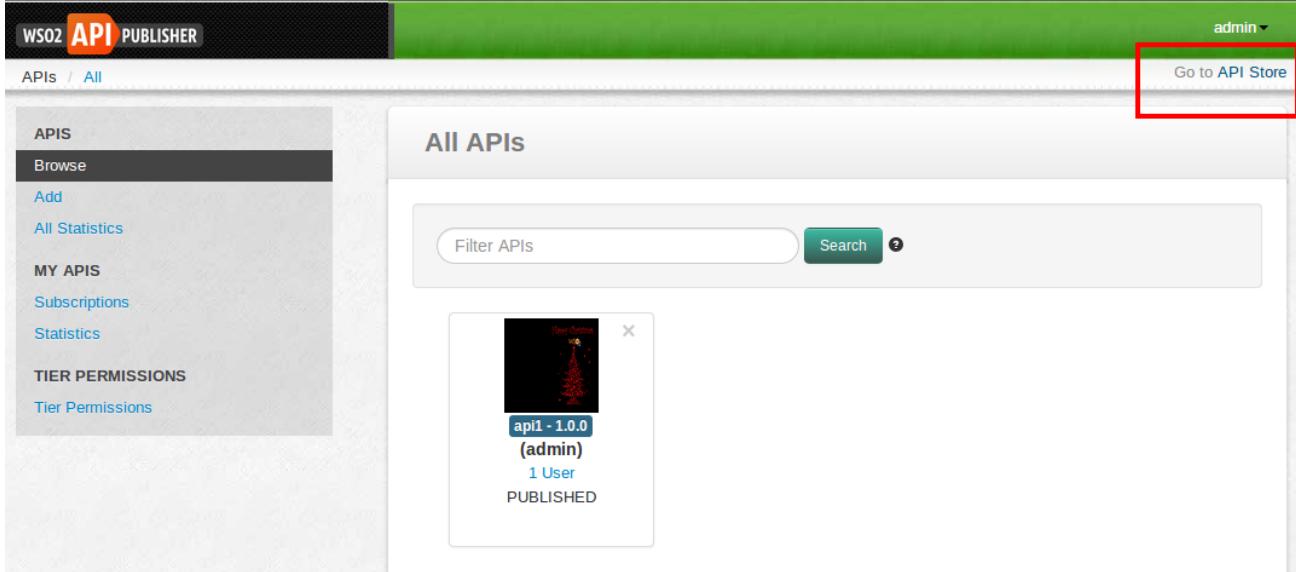
By default, there are no links in the UIs of the API Store and API Publisher applications to traverse between the two apps.

To add a link in API Publisher to API Store:

1. In `<AM_HOME>/repository/conf/api-manager.xml` file, set the `<DisplayURL>` to true and provide the URL of the Store.

```
<APIStore>
    <DisplayURL>true</DisplayURL>
    <URL>https://{$carbon.local.ip}:{$mgt.transport.https.port}/store</URL>
</APIStore>
```

2. Note a URL in the API Publisher that points to the API Store. For example,



The screenshot shows the WSO2 API Publisher interface. On the left, a sidebar menu includes 'APIS', 'Browse', 'Add', 'All Statistics', 'MY APIS', 'Subscriptions', 'Statistics', 'TIER PERMISSIONS', and 'Tier Permissions'. The main area is titled 'All APIs' and contains a search bar with 'Filter APIs' and a 'Search' button. Below the search bar is a list of APIs. One API is highlighted with a modal overlay: 'api1 - 1.0.0 (admin) 1 User PUBLISHED'. In the top right corner of the main area, there is a link 'Go to API Store' which is highlighted with a red box.

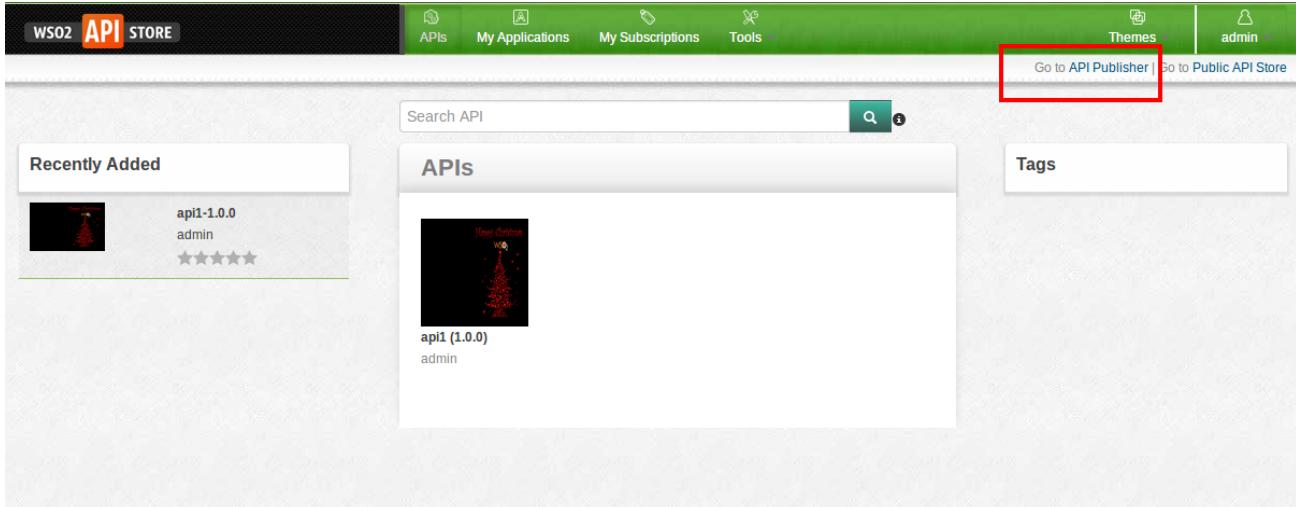
To add a link in API Store to API Publisher:

1. In <AM_HOME>/repository/conf/api-manager.xml file, set the <DisplayURL> to true and provide the URL of the Publisher.

```
<APIPublisher>
  <DisplayURL>true</DisplayURL>

  <URL>https://${carbon.local.ip}:${mgt.transport.https.port}/publisher</URL>
</APIPublisher>
```

2. Note a URL in the API Store that points to the API Publisher. For example,



The screenshot shows the WSO2 API Store interface. The top navigation bar includes 'APIs', 'My Applications', 'My Subscriptions', 'Tools', 'Themes', and 'admin'. In the top right, there are links 'Go to API Publisher' and 'Go to Public API Store', with 'Go to API Publisher' highlighted by a red box. The main area is titled 'APIs' and contains a search bar 'Search API'. Below the search bar is a list of APIs. One API is highlighted with a modal overlay: 'api1 (1.0.0) admin'. To the left, there is a 'Recently Added' section showing the same API.

Maintaining Separate Production and Sandbox Gateways

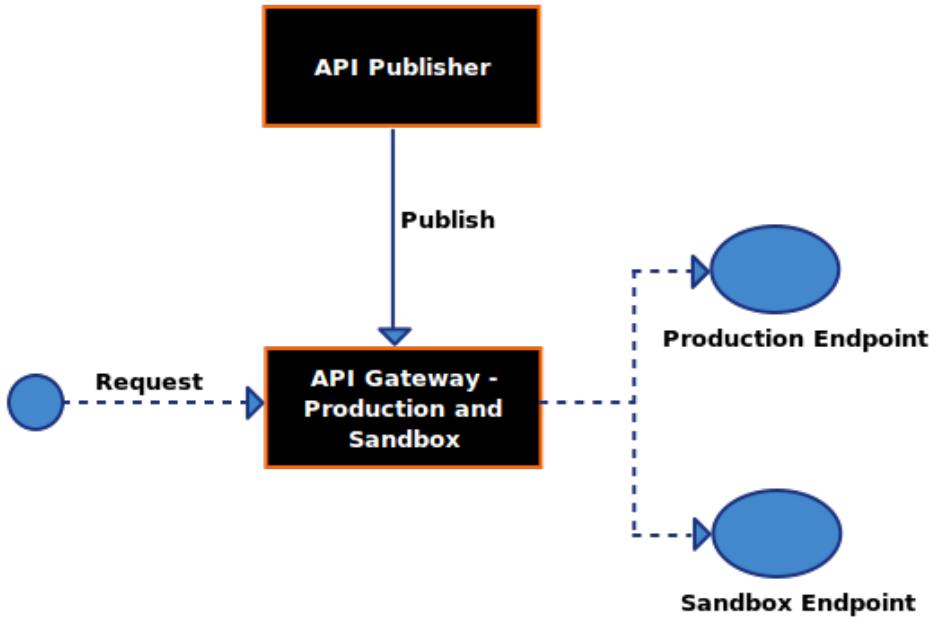
With WSO2 API Manager, you can maintain a production and a sandbox endpoint for a given API. The production endpoint is the actual location of the API, whereas the sandbox endpoint points to its testing/pre-production environment.

When you publish an API using the API Publisher, it gets deployed on the API Gateway. By default, there's a single Gateway instance (deployed either externally or embedded within the publisher), but you can also set up multiple Gateways:

- Single Gateway to handle both production and sandbox requests
- Multiple Gateways to handle production and sandbox requests separately

Single Gateway to handle both production and sandbox requests

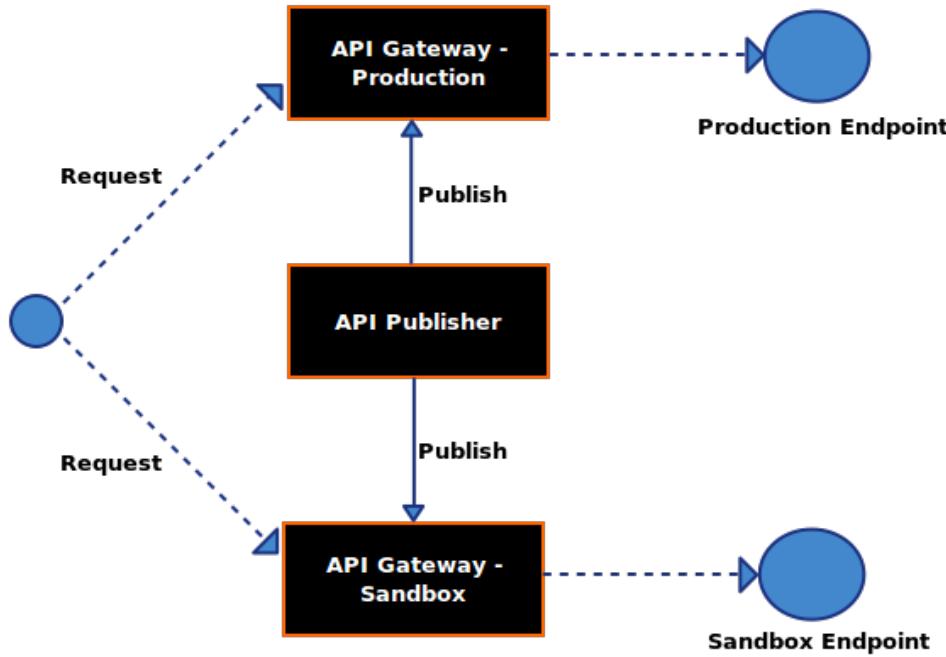
This is the default scenario. Because this Gateway instance handles both production and sandbox token traffic, it is called a hybrid API Gateway. When an API request comes to the API Gateway, it checks whether the requesting token is of type PRODUCTION or SANDBOX and forwards the request to the appropriate endpoint. The diagram below depicts this scenario.



Multiple Gateways to handle production and sandbox requests separately

Having a single gateway instance to pass through both types of requests can negatively impact the performance of the production server. To avoid this, you can set up separate API Gateways. The production API Gateway handles requests that are made using PRODUCTION type tokens and the sandbox API Gateway handles requests that are made using SANDBOX type tokens.

The diagram below depicts this using two Gateways:



In either of the two approaches, if an API Gateway receives an invalid token, it returns an error to the requesting client saying that the token is invalid.

You configure production and sandbox gateways using the `<Environments>` element in the `<AM_HOME>/repository/conf/api-manager.xml` file as shown in the following example:

```

<Environments>
    <Environment type="production">
        <Name>Production and Sandbox</Name>
        <ServerURL>https://localhost:9445/services/</ServerURL>
        <Username>admin</Username>
        <Password>admin</Password>
    </Environment>

    <Environment type="sandbox">
        <Name>Production and Sandbox</Name>
        <ServerURL>https://localhost:9448/services/</ServerURL>
        <Username>admin</Username>
        <Password>admin</Password>
    </Environment>
</Environments>
  
```

The `type` attribute of the `<Environment>` element can take the following values:

- **Production:** A production type Gateway
- **Sandbox:** A sandbox type Gateway
- **Hybrid:** The Gateway handles both types of tokens

If you work with Gateways in different geographical locations, configuring multiple environments using the `<APIGateway>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file is recommended. The diagram below depicts a sample setup:

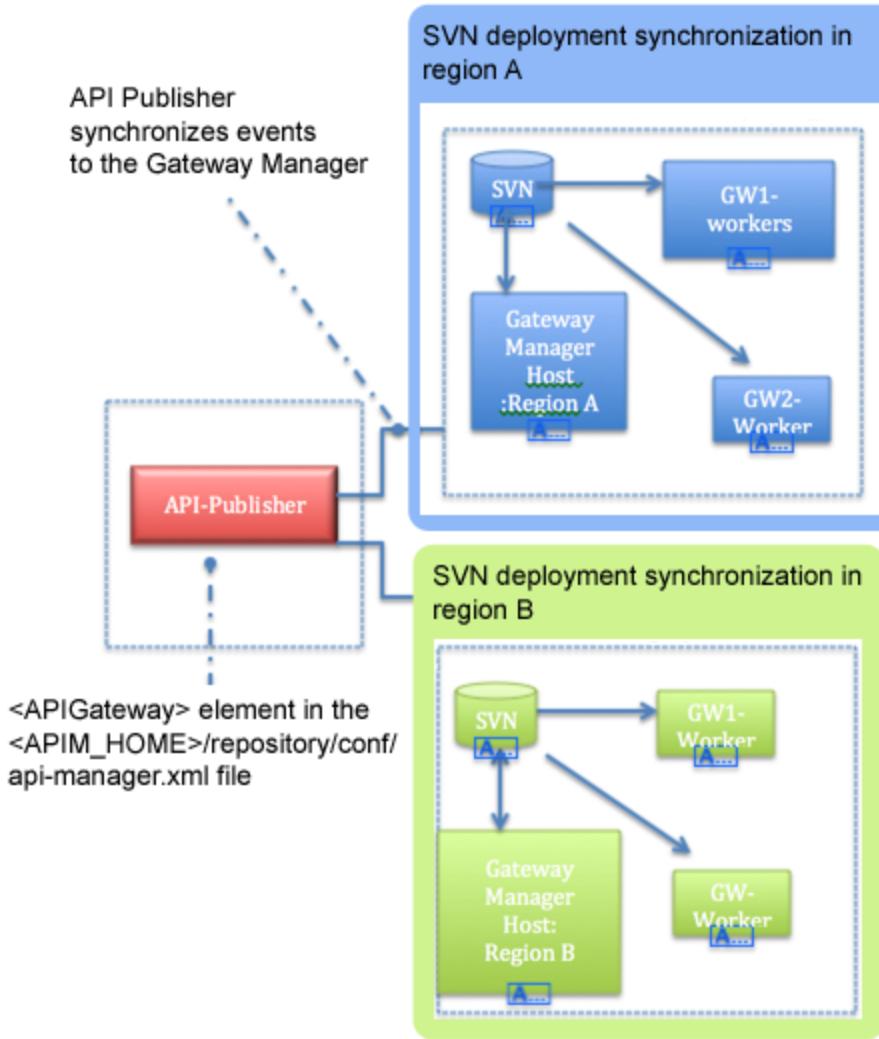


Figure: API Gateways in different geographical regions

Configuring Transports

A transport is responsible for carrying messages that are in a specific format. WSO2 API Manager supports all the widely used transports including HTTP/s, JMS, Pass-through and VFS, and domain-specific transports like FIX. All WSO2 transports are directly or indirectly based on the Apache Axis2 transports framework. This framework provides two main interfaces that each transport implementation has.

- **org.apache.axis2.transport.TransportListener:** Implementations of this interface specify how incoming messages are received and processed before handing them over to the Axis2 engine for further processing.
- **org.apache.axis2.transport.TransportSender:** Implementations of this interface specify how a message can be sent out from the Axis2 engine.

Because each transport has to implement the two interfaces above, each transport generally contains a transport receiver/listener and a transport sender. You configure, enable, and manage transport listeners and senders independently to each other. For example, you can enable just the JMS transport sender without having to enable the JMS transport listener.

For more information, see the following topics in the WSO2 ESB documentation:

- Available transports
- How to configure transports

Extending the API Manager

The following topics cover different ways in which you can extend the API Manager:

- [Writing Custom Handlers](#)
- [Integrating with WSO2 Governance Registry](#)
- [Adding Mediation Extensions](#)
- [Adding Workflow Extensions](#)
- [Adding new Throttling Tiers](#)
- [Adding a Reverse Proxy Server](#)
- [Adding a new API Store Theme](#)
- [Transforming API Message Payload](#)

Writing Custom Handlers

This section introduces handlers and using an example, explains how to write a custom handler:

- [Introducing Handlers](#)
- [Writing a custom handler](#)
- [Engaging the custom handler](#)

Introducing Handlers

W You find the default handlers in any API's Synapse definition as shown below.

hen an API is created, a file with its synapse configuration is added to the API Gateway. You can find it in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration.

```
class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
<handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
    <property name="id" value="A"/>
    <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/>
</handler>
<handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
<handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
<handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
</handlers>
```

Let's see what each handler does:

- **APIAuthenticationHandler:** Validates the OAuth2 bearer token used to invoke the API. It also determines whether the token is of type Production or Sandbox and sets MessageContext variables as appropriate.
- **APIThrottleHandler:** Throttles requests based on the throttling policy specified by the `policyKey` property. Throttling is applied both at the application level as well as subscription level.
- **APIMgtUsageHandler:** Publishes events to BAM for collection and analysis of statistics. This handler only comes to effect if API usage tracking is enabled. See [Publishing API Runtime Statistics](#) for more information.
- **APIMgtGoogleAnalyticsTrackingHandler:** Publishes events to Google Analytics. This handler only comes into effect if Google analytics tracking is enabled. See [Integrating with Google Analytics](#) for more information.
- **APIManagerExtensionHandler:** Triggers extension sequences. By default, the extension handler is listed at last in the handler chain, and therefore is executed last. You cannot change the order in which the handlers are executed, except the extension handler. To configure the API Gateway to execute extension handler first, uncomment the `<ExtensionHandlerPosition>` section in the `<APIM_HOME>/repository/conf/api-manager.xml` file and provide the value `top`. This is useful when you want to execute your own extensions before our default handlers in situations like doing additional security checks such as signature verification on access tokens before executing the default security handler. See [Adding Mediation Extensions](#).

Writing a custom handler

Let's see how you can write a custom handler and apply it to the API Manager. In this example, we extend the authentication handler. Make sure your custom handler name is not the same as the name of an existing handler.

WSO2 API Manager provides the OAuth2 bearer token as its default authentication mechanism. The source code of the implementation is [here](#). Similarly, you can extend the API Manager to support any custom authentication mechanism by writing your own authentication handler class. This custom handler must extend `org.apache.synapse.rest.AbstractHandler` class and implement the `handleRequest()` and `handleResponse()` methods.

Given below is an example implementation:

```
package org.wso2.carbon.test;

import org.apache.synapse.MessageContext;
import org.apache.synapse.core.axis2.Axis2MessageContext;
import org.apache.synapse.rest.AbstractHandler;

import java.util.Map;

public class CustomAPIAuthenticationHandler extends AbstractHandler {

    public boolean handleRequest(MessageContext messageContext) {
        try {
            if (authenticate(messageContext)) {
                return true;
            }
        } catch (APISecurityException e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean handleResponse(MessageContext messageContext) {
        return true;
    }

    public boolean authenticate(MessageContext synCtx) throws APISecurityException {
        Map headers = getTransportHeaders(synCtx);
        String authHeader = getAuthorizationHeader(headers);
        if (authHeader.startsWith("userName")) {
            return true;
        }
        return false;
    }

    private String getAuthorizationHeader(Map headers) {
        return (String) headers.get("Authorization");
    }

    private Map getTransportHeaders(MessageContext messageContext) {
        return ((Axis2MessageContext) messageContext).getAxis2MessageContext().
getProperty(org.apache.axis2.context.MessageContext.TRANSPORT_HEADERS);
    }
}
```

Engaging the custom handler

You can engage a custom handler to all APIs at once or only to selected APIs.

To engage to all APIs, the recommended approach is to add it to the `<APIM_HOME>/repository/resources/api_templates/velocity_template.xml` file. For example, the following code segment adds the custom authentication handler that you wrote earlier to the `velocity_template.xml` file while making sure that it skips the default `APIAuthenticationHandler` implementation:

```

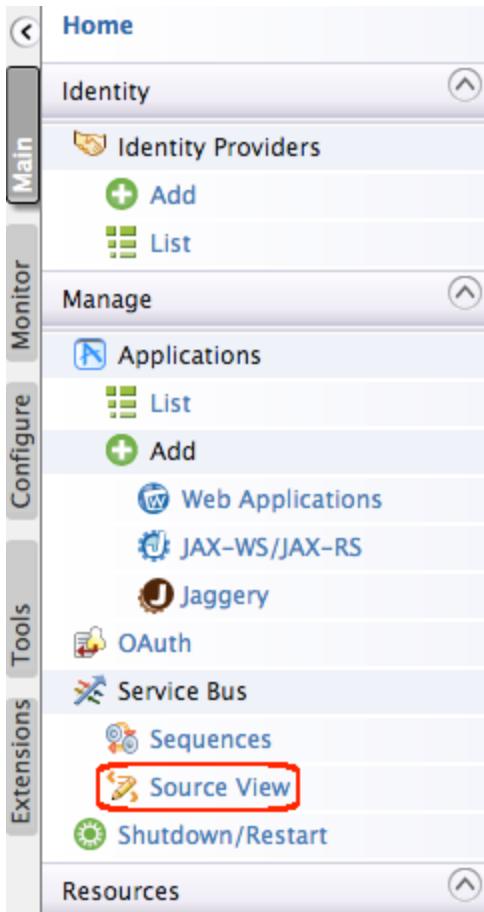
<handler
class="org.wso2.carbon.apimgt.custom.authentication.handler.CustomAPIAuthenticationHandler" />
    #foreach($handler in $handlers)
        #if(!($handler.className ==
"org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"))
            <handler xmlns="http://ws.apache.org/ns/synapse"
class="$handler.className">
                #if($handler.hasProperties())
                    #set ($map = $handler.getProperties())
                    #foreach($property in $map.entrySet())
                        <property name="$!property.key" value="$!property.value"/>
                    #end
                #end
            </handler>
        #end
    #end
</handlers>

```

Given below is how to engage handlers to a single API, by editing its source view.

 **Note** that when you engage a handler by editing the API's source view, your changes will be overwritten every time you save the API through the API Publisher.

1. Build the class and copy the JAR file to `<APIM_HOME>/repository/components/lib` folder.
2. Log in to the management console and select **Service Bus > Source View** in the **Main** menu.



3. In the configuration that opens, select an API and navigate to the <Handlers> section. The following line appears as the first handler. This is the current authentication handler used in the API Manager.

Service Bus Configuration

Make the required modifications to the configuration and click 'Update' to apply the changes to the server. Use 'Reset' button to undo your changes.

ESB Configuration

```

390 <send/>
391 </outSequence>
392 </resource>
393 <handlers>
394 <handler class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler" />
395 <handler class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
396 <property name="id" value="A"/>
397 <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml" />
398 </handler>
399 <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler" />
400 <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler" />
401 <property name="configKey" value="gov:/apimgt/statistics/ga-config.xml" />
402 </handler>
403 <handler class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler" />
404 </handlers>
405 </api>
406 <api name="admin--PhoneVerification"
407 context="/phoneverify"
408 version="2.0.0"
409 version-type="url">
410 <resource methods="OPTIONS GET" />

```

4. Replace the above line with the handler that you created. It will engage your custom handler to the API Manager instance. According to this example, it is as follows:

```

<handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.CustomAPIAuthenticationHa
ndler" />

```

Integrating with WSO2 Governance Registry

WSO2 Governance Registry is a registry-repository for storing and managing metadata related to services and other artifacts. Services in the Governance Registry are implemented as [configurable governance artifacts](#) (RXT files). Usually, APIs are created using the API Publisher Web interface. Instead, you can integrate the API Manager with the Governance Registry to directly create APIs in the API Publisher using the services deployed in the Governance Registry.

The steps below explain how to configure the two products to expose services in the Governance Registry as APIs.

- The following steps apply to WSO2 Governance Registry version 4.6.0 or after.
- In WSO2 Governance Registry 4.6.0, we do a simple POST to create APIs in the API Publisher. It does not involve registry mounting.

Follow the steps below to publish services on Governance Registry to the API Manager.

1. Download both WSO2 Governance Registry (G-Reg) and WSO2 API Manager.
2. Provide the API Manager credentials in `<GREG_HOME>/repository/resources/lifecycles/configurations.xml` file. For example, the following code block defines an `execution` element in `production` state. It provides the API Manager's endpoint, username and password as executor parameters.

```
<execution forEvent="Publish"
class="org.wso2.carbon.governance.registry.extensions.executors.apistore.ApiStore
Executor">
    <parameter name="apim.endpoint" value="http://localhost:9763" />
    <parameter name="apim.username" value="admin" />
    <parameter name="apim.password" value="admin" />
    <parameter name="default.tier" value="Unlimited" />
    <parameter name="throttlingTier"
value="Unlimited,Unlimited,Unlimited,Unlimited,Unlimited" />
</execution>
```

! **Note:** If you started the G-Reg server at least once before executing step 2, editing the `configurations.xml` file and restarting the server does not apply the configurations. You need to add the configurations using the G-Reg management console as follows:

- a. Log in to the G-Reg Management console and select **Extensions -> Configure -> Lifecycles** menu.
- b. Click the **Edit** link associated with `ServiceLifeCycle` .
- c. Add the configuration given in step 2 above and **Save**.

3. Run the G-Reg and the API Manager.

When running more than one WSO2 products on the same server, change the default port of one product to avoid port conflicts. You can do this by changing the `<offset>` value of one product in `<PRODUCT_HOME>/repository/conf/carbon.xml` file. In this example, we set the port offset value of Governance Registry to 1 as follows: `<Offset>1</Offset>`

! **Note:** If you offset the default API Manager port, you must also change the default API endpoints and the Thrift port accordingly. See [Changing the Default Ports with Offset](#).

4. Access the API Manager server using the following URL: <https://<HostName>:9443/carbon>. As you changed the default port of G-Reg, you can access the server using the following URL: <https://<HostName>:<9443+off set>/carbon>.
5. Log in to the G-Reg management console and create a new service in it and attach the default service

lifecycle to it. For instructions on how to add a new service and associate a new lifecycle, see <http://docs.wso2.org/governance-registry/Managing+Services> in the Governance Registry documentation.

6. Promote the service until it gets to the production state.
7. When it is in the production state, publish it using the **Publish** button. You should get a confirmation message once the API is successfully published.
8. You have now created an API using a service in the Governance Registry. Open the API Publisher to see that this service is successfully created as an API.

Adding Mediation Extensions

The API Gateway has a default mediation flow that is executed in each API invocation. You can do additional custom mediation for the messages in the API Gateway by extending its mediation flow. An extension is provided as a synapse mediation sequence.

Please do not use the API Manager's management console to create sequences as the functionality is not supported. You can design all sequences using a tool like WSO2 Developer Studio, and store the `sequence.xml` file in the governance registry. For information, see [Creating ESB Artifacts](#) in the Developer Studio documentation. The registry collection where sequences are stored is `customsequences`, which is available by default in `apimgt` governance registry location. Given below are the registry paths:

Sequence	Registry path
In	<code>/_system/governance/apimgt/customsequences/in</code>
Out	<code>/_system/governance/apimgt/customsequences/out</code>
Fault	<code>/_system/governance/apimgt/customsequences/fault</code>

For example, if you have an in sequence file as `testInSequence`, you must save it in `/_system/governance/apimgt/customsequences/in/testInSequence.xml`.

There are two ways to apply mediation extensions to messages:

- [Global Extensions](#) : Apply to all APIs
- [Per-API Extensions](#) : Apply only to an intended API

The difference between a global extension and a per-API extension is simply in the name given to the sequence that you use to create it.

Creating global extensions

Given below is the naming pattern of a global extension sequence.

`WSO2AM--Ext--<DIRECTION>`

The `<DIRECTION>` can be `In` or `Out`. To change the default fault sequence, you can either modify the default sequence or write a custom fault sequence and engage it to APIs through the API Publisher. When the direction of the sequence is `In`, the extension is triggered on the in-flow (request path). Similarly, when the direction of the sequence is `Out`, the extension is triggered on the out-flow (response path). Shown below is an example synapse configuration of a global extension sequence.

Global Extension Sequence Example

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="WSO2AM--Ext--In">
    <log level="custom">
        <property name="TRACE" value="Global Mediation Extension"/>
    </log>
</sequence>
```

To test the code, copy it to an XML file (e.g., `global_ext.xml`) and save the file in the `<APIM_HOME>/repository/`

deployment/server/synapse-configs/default/sequences directory. The above sequence prints a log message on the console on every API invocation.

Creating per-API extensions

Given below is the naming pattern of a per-API extension sequence.

<API_NAME> : v<VERSION>--<DIRECTION>

Shown below is an example synapse configuration of a per-API extension sequence. It is created for an API named admin--TwitterSearch with version 1.0.0.

API Extension Sequence Example

```
<sequence xmlns="http://ws.apache.org/ns/synapse"
name="admin--TwitterSearch:v1.0.0--In">
    <log level="custom">
        <property name="TRACE" value="API Mediation Extension"/>
    </log>
</sequence>
```

NOTE: The tenant username must be given as <username>-AT-<domain> in the configuration. For example, if the tenant username is testuser and the domain is wso2.com, then the name attribute in the above configuration must be testuser-AT-wso2.com--TwitterSearch:v1.0.0-In . The @ sign must be given as AT.

To test the code in super-tenant mode, copy it to an XML file (e.g., twittersearch_ext.xml) and save the file in the <APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences directory, if you are in single-tenant mode. In multi-tenant mode, copy the file to the tenant's synapse sequence folder. For example, if tenant id is 1, then copy it to <API_Gateway>/repository/tenants/1/synapse-configs/default/sequences folder.

The above sequence prints a log message on the console whenever the TwitterSearch API is invoked.

Alternatively, you can create the XML file and upload it to the registry using the management console UI.

1. Open the APIM management console (<https://localhost:9443/carbon> with admin/admin as the default credentials) and select **Resources -> Browse**.
2. Navigate to /_system/governance/apimgt/customsequences registry location.
3. Click **Add Resource** link to upload the XML file.

Selecting predefined APIs from the UI

You can attach pre-defined extension sequences to an API using the API Publisher Web interface, at the time the API is created. Log in to the API Publisher (<https://localhost:9443/publisher>) and click **Add** from the left panel. In the **Add New API** page that opens, navigate to the **Manager** section where you find **Sequences**. There, you can select In/Out/Fault sequences for the API from the drop-down lists. For example,

Sequences: Check to select a custom sequence to be executed in the message flow

In Flow	Out Flow	Fault Flow
log_in_me	log_out_me	json_1

To populate these drop-down lists, you must add mediation sequences as explained at the beginning.

Invoking the extension sequences

When an API is published, a file with its synapse configuration is created on the API Gateway. This synapse configuration has a set of handlers as shown in the following example:

```

API Configuration

<handlers>
    <handler
    class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
        <handler class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
        <handler
    class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
        <handler
    class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
        <property name="id" value="A"/>
        <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml"/>
    </handler>
    <handler
    class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
</handlers>

```

The handler by the name `APIManagerExtensionHandler` triggers both global as well as per-API extension sequences. It reads the sequence names and determines what APIs must be invoked. By default, the extension handler is listed at last in the handler chain, and therefore is executed last. You can configure the API Gateway to execute extension handlers first. To do that, open `<APIM_HOME>/repository/conf/api-manager.xml` file, uncomment the `<ExtensionHandlerPosition>` section and provide the value `top` as follows:

```
<ExtensionHandlerPosition>top</ExtensionHandlerPosition>
```

This is useful when you want to execute your own extensions before our default handlers. For example, if you want to have additional security checks such as signature verification on access tokens before executing the default security handler, you can define an extension and configure the Gateway to execute extension handlers first.

For more information on Handlers, see [API Manager Components](#).

Adding Workflow Extensions

Use workflow extensions to attach a workflow to the following API Store/API Publisher operations:

- [Adding an Application Creation Workflow](#)
- [Adding an Application Registration Workflow](#)
- [Adding an API Subscription Workflow](#)
- [Adding a User Signup Workflow](#)
- [Invoking the API Manager from the BPEL Engine](#)
- [Customizing a Workflow Extension](#)
- [Configuring Workflows for Tenants](#)
- [Configuring Workflows in a Cluster](#)
- [Changing the Default User Role in Workflows](#)

Adding an Application Creation Workflow

This section explains how to attach a custom workflow to the application creation operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

Configuring the Business Process Server

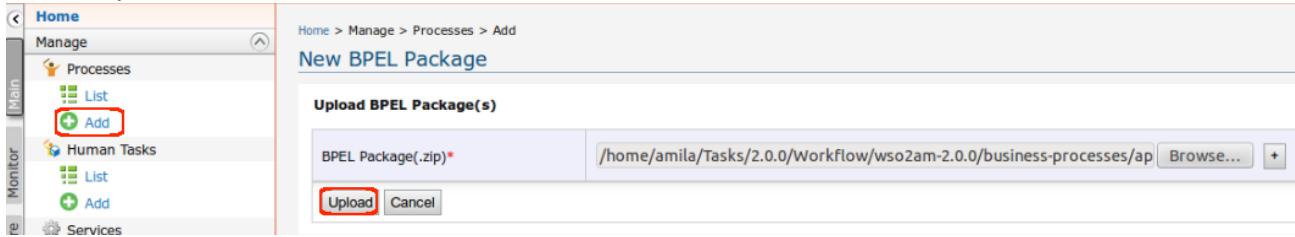
1. Download [WSO2 Business Process Server](#).
2. Set an offset of 2 to the default BPS port in `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

<Offset>2</Offset>

! If you change the port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the `hostname` to a different value than `localhost`), you must do the following:

- Search and replace the value 9765 in all the files (.epr, .wsdl files inside the ZIP archives) inside `<APIM_HOME>/business-processes` folder with the new port
- Zip the files you unzipped earlier and deploy the newly created zip file in BPS as explained in the steps below
- Search and replace port 9445 in `<AM_HOME>/repository/deployment/server/jagger/yapps/admin-dashboard/site/conf/site.json` file

3. Copy the following from `<APIM_HOME>/business-processes/epr` to `<BPS_HOME>/repository/conf/epr` folder. If the `<BPS_HOME>/repository/conf/epr` folder isn't there, please create it.
 - `ApplicationService.epr`
 - `ApplicationCallbackService.epr`
4. Start the BPS server and log in to its management console (`https://<Server Host>:9443+<port offset>/carbon`).
5. Select **Add** under **Processes** menu and upload the `<APIM_HOME>/business-processes/application-creation/BPEL/ApplicationApprovalWorkFlowProcess_1.0.0.zip` file to BPS. This is the business process archive file.

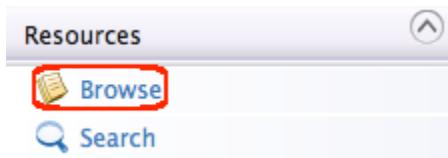


6. Select **Add** under the **Human Tasks** menu and upload `<APIM_HOME>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip` to BPS. This is the human task archived file.

Engaging the WS Workflow Executor in the API Manager

First, enable the application creation workflow.

1. Log in to APIM management console (`https://<Server Host>:9443/carbon`) and select **Browse** under **Resources**.



2. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```

<WorkFlowExtensions>
    <!--ApplicationCreation
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationCreationSimpleWorkflowE
xecutor"-->
    <ApplicationCreation
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationCreationWSWorkflowExecu
tor">
        <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationApprovalWorkFlow
Process/</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Prope
rty>
    </ApplicationCreation>
</WorkFlowExtensions>

```

The application creation WS Workflow Executor is now engaged.

3. Go to the API Store Web interface, open **My Applications** page and create a new application. It invokes the application creation process and creates a Human Task instance that holds the execution of the BPEL process until some action is performed on it.
4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
5. Log in to the workflow-admin app (<https://localhost:9443/workflow-admin>), list all the tasks for application creation and approve the task. It resumes the BPEL process and completes the application creation.
6. Go back to the **My Applications** page on the API Store and see the created application.

Whenever a user tries to create an application in the API Store, a request is sent to the workflow endpoint. Given below is a sample:

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
    <soapenv:Header />
    <soapenv:Body>
        <wor:createApplication
            xmlns:wor="http://workflow.application.apimgt.carbon.wso2.org">
            <wor:applicationName>application1</wor:applicationName>
            <wor:applicationTier>Gold</wor:applicationTier>

            <wor:applicationCallbackUrl>http://webapp/url</wor:applicationCallbackUrl>
                <wor:applicationDescription>Application 1</wor:applicationDescription>
                <wor:tenantDomain>wso2.com</wor:tenantDomain>
                <wor:userName>user1</wor:userName>

            <wor:workflowExternalRef>c0aad878-278c-4439-8d7e-712ee71d3f1c</wor:workflowExternalRef>
                <wor:callBackURL>https://localhost:8243/services/WorkflowCallbackService</wor:callBackURL>
                    </wor:createApplication>
                </soapenv:Body>
            </soapenv:Envelope>

```

Elements of the above configuration are described below:

Element	Description
applicationName	Name of the application the user creates.
applicationTier	Throttling tier of the application.
applicationCallbackUrl	When the OAuth2 Authorization Code grant type is applied, this is the endpoint on which the callback needs to happen after the user is authenticated. This is an attribute of the actual application registered on the API Store.
applicationDescription	Description of the application
tenantDomain	Tenant domain associated with the application (domain of the user creating the application).
userName	username of the user creating the application.
workflowExternalRef	The unique reference against which a workflow is tracked. This needs to be sent back from the workflow engine to the API Manager at the time of workflow completion.
callBackURL	At the time of workflow completion, the workflow-completion request is sent to this URL by the workflow engine. This property is configured in the <callBackURL> element in the api-manager.xml.

Adding an Application Registration Workflow

This section explains how to attach a custom workflow to the application registration operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

Introduction to the application registration workflow

Application creation and registration are different workflows. After an application is created, you can subscribe to available APIs, but you get the consumer key/secret and access tokens only after registering the application. There are two types of registrations that can be done to an application: production and sandbox. You change the default application registration workflow in situations such as the following:

1. To issue only sandbox keys when creating production keys is deferred until testing is complete.
2. To restrict untrusted applications from creating production keys. You allow only the creation of sandbox keys.
3. To make API subscribers go through an approval process before creating any type of access token.

Configuring the Business Process Server

1. Download WSO2 Business Process Server.
2. Set an offset of 2 to the default BPS port in `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

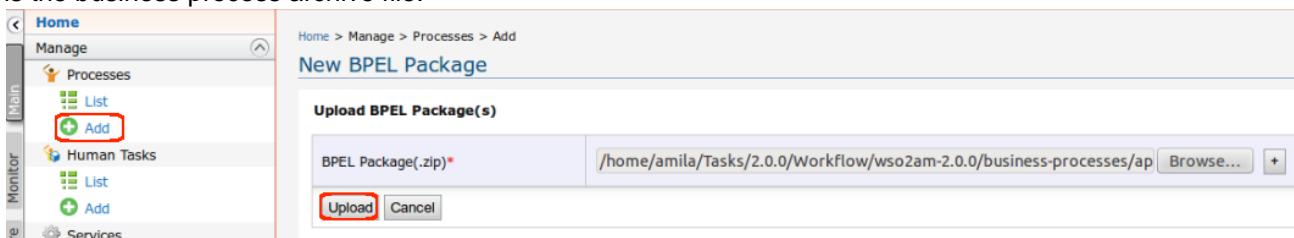
```
<Offset>2</Offset>
```



If you change the port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the `hostname` to a different value than `localhost`), you must do the following:

- Search and replace the value 9765 in all the files (`.epr`, `.wsdl` files inside the ZIP archives) inside `<APIM_HOME>/business-processes` folder with the new port.
- Zip the files you unzipped earlier and deploy the newly created zip file in BPS as explained in the steps below.
- Search and replace port 9445 in `<AM_HOME>/repository/deployment/server/jagger/yapps/admin-dashboard/site/conf/site.json` file.

3. Copy the following from `<APIM_HOME>/business-processes/epr` to `<BPS_HOME>/repository/conf/epr` folder. If the `<BPS_HOME>/repository/conf/epr` folder isn't there, please create it.
 - `RegistrationService.epr`
 - `RegistrationCallbackService.epr`
4. Start the BPS server and log in to its management console (`https://<Server Host>:9443+<port offset>/carbon`).
5. Select **Add** under **Processes** menu and upload the `<APIM_HOME>/business-processes/application-registration/BPEL/ApplicationRegistrationWorkflowProcess_1.0.0.zip` file to BPS. This is the business process archive file.



6. Select **Add** under the **Human Tasks** menu and upload `<APIM_HOME>/business-processes/application-registration/HumanTaskBPEL/ApplicationRegistrationTask-1.0.0.zip` to BPS. This is the human task archived file.

Engaging the WS Workflow Executor in the API Manager

First, enable the application registration workflow.

1. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



2. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor:

```

<WorkFlowExtensions>
    <!--ProductionApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationSimpleWorkflowExecutor"-->
    <ProductionApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
        <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
        </ProductionApplicationRegistration>
        <!--SandboxApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationSimpleWorkflowExecutor"-->
        <SandboxApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationWSWorkflowExecutor">
        <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationRegistrationWorkflowProcess/</Property>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
name="callbackURL">https://localhost:8248/services/WorkflowCallbackService</Property>
        </SandboxApplicationRegistration>
    </WorkFlowExtensions>

```



Note that all workflow process services of the BPS run on port 9765 as you changed its default port with an offset of 2.

3. Go to the API Store Web interface, open **My Subscriptions** page, select an application and click the **Generate** button associated with the production key. It invokes the `ApplicationRegistrationWorkflowProcess.bpel` that is bundled with `ApplicationRegistrationWorkflowProcess_1.0.0.zip` and creates a `HumanTask` instance that holds the execution of the BPEL process until some action is performed on it.
4. Note a message that appears saying that the request is successfully submitted if the BPEL was invoked correctly. For example,

Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times to a single API, with different SLA levels.

Applications With Subscriptions

SampleApp1

Keys - Production ▾

Request to register this application has been sent.

5. Log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>) and list all the tasks for application registrations. Click **Start** to start the Human Task and then change its state. Once you approve the task, it resumes the BPEL process and completes the registration.
6. Go back to the **My Subscriptions** page on the API Store and view your application.

It shows the application access token, consumer key and consumer secret. For example,

Subscriptions

Keys are generated per Application which allows to use a single key for multiple APIs and subscribe multiple times to a single API, with different SLA levels.

Applications With Subscriptions

SampleApp1

Show Keys

Keys - Production ▾

Access Token

W3e7f_f_43CxslciYC5jOSi58jga

[Re-generate](#)

Token Validity: 3600 Seconds

Allowed Domains

ALL

Leave empty or filling with "ALL" will allow all domains.

[Update](#)
[Domains](#)

After the registration request is approved, keys are generated by invoking the `APIKeyMgtSubscriber` service hosted in Key Manager nodes. Even when the request is approved, key generation can fail if this service becomes unavailable. To address such failures, you can configure to trigger key generation at a time Key Manager nodes become available again. Given below is the message used to invoke the BPEL process:

```

<applicationregistrationworkflowprocessrequest
  xmlns:wor="http://workflow.application.apimgt.carbon.wso2.org"
  xmlns="http://workflow.application.apimgt.carbon.wso2.org">
  <applicationname>NewApp5</applicationname>
  <applicationtier>Unlimited</applicationtier>
  <applicationcallbackurl></applicationcallbackurl>
  <applicationdescription></applicationdescription>
  <tenantdomain>carbon.super</tenantdomain>
  <username>admin</username>

  <workflowexternalref>4a20749b-a10d-4fa5-819b-4fae5f57ffaf</workflowexternalref>

  <callbackurl>https://localhost:8243/services/WorkflowCallbackService</callbackurl>
    <keytype>PRODUCTION</keytype>
  </applicationregistrationworkflowprocessrequest>

```

Adding an API Subscription Workflow

This section explains how to attach a custom workflow to the API subscription operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflows executors.

Attaching a custom workflow to API subscription enables you to add throttling tiers to an API that consumers cannot choose at the time of subscribing. Only admins can set these tiers to APIs. It also allows you to restrict API consumers to only subscribe to sandbox, and then go through an approval process to go to the next level of subscription.

Configuring the Business Process Server

1. Download [WSO2 Business Process Server](#).
2. Set an offset of 2 to the default BPS port in `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

```
<Offset>2</Offset>
```

 **Tip:** If you **change the BPS port offset to a value other than 2 or run the API Manager and BPS on different machines** (therefore, want to set the hostname to a different value than `localhost`), you do the following:

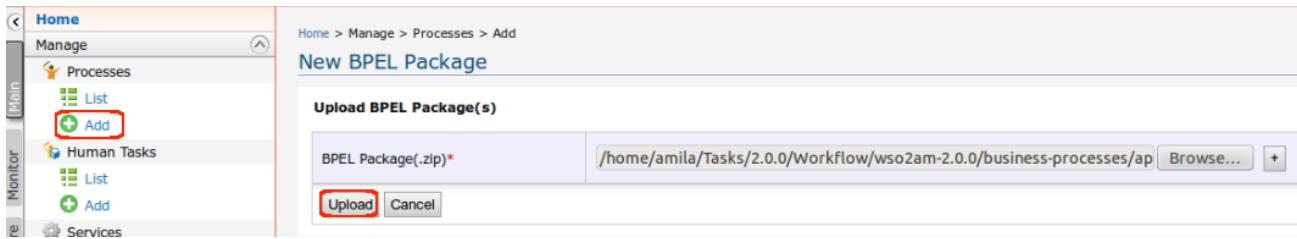
- Search and replace the value 9765 in all the files (`.epr`, `.wsdl` files inside the ZIP archives) inside `<APIM_HOME>/business-processes` folder with the new port.
- Zip the files you unzipped earlier and deploy the newly created zip file in BPS as explained in the steps below.
- Search and replace port 9445 in `<APIM_HOME>/repository/deployment/server/jaggerapps/admin-dashboard/site/conf/site.json` file.

 **Tip: Did you change the default port of the API Manager rather than the BPS?** If so, be sure to do the following:

- Search and replace the value 8243 in all the files (`.epr`, `.wsdl` etc.) inside the ZIP archives inside `<APIM_HOME>/business-processes` folder with the new port.
- Change the port in the following property in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/workflowcallbackService.xml` file.

```
<address
uri="https://localhost:9444/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag" format="rest"/>
```

3. Copy the following file from `<APIM_HOME>/business-processes/epr` to `<BPS_HOME>/repository/conf/epr` folder. If the `<BPS_HOME>/repository/conf/epr` folder isn't there, please create it.
 - `SubscriptionService.epr`
 - `SubscriptionCallbackService.epr`
4. Start the BPS server and log in to its management console (`https://<Server Host>:9443+<port offset>/carbon`).
5. Select **Add** under the **Processes** menu and upload the `<APIM_HOME>/business-processes/subscription-creation/BPEL/SubscriptionApprovalWorkFlowProcess_1.0.0.zip` file to BPS. This is the business process archive file.



6. Select **Add** under the **Human Tasks** menu and upload <APIM_HOME>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip to BPS. This is the human task archived file.

Engaging the WS Workflow Executor in the API Manager

First, enable the API subscription workflow.

1. Log in to APIM admin console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



2. Go to /_system/governance/apimgt/applicationdata/workflow-extensions.xml resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```

<WorkFlowExtensions>
    <!--SubscriptionCreation
executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationSimpleWorkflow
Executor"/><!--
    &lt;SubscriptionCreation
executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationWSWorkflowExec
utor"&gt;
        &lt;Property
name="serviceEndpoint"&gt;http://localhost:9765/services/SubscriptionApprovalWorkFlo
wProcess/&lt;/Property&gt;
        &lt;Property name="username"&gt;admin&lt;/Property&gt;
        &lt;Property name="password"&gt;admin&lt;/Property&gt;
        &lt;Property
name="callbackURL"&gt;https://localhost:8243/services/WorkflowCallbackService&lt;/Prope
rty&gt;
    &lt;/SubscriptionCreation&gt;
&lt;/WorkFlowExtensions&gt;
</pre>

```

The application creation WS Workflow Executor is now engaged.

3. Go to the API Store Web interface and subscribe to an API. It invokes the API subscription process and creates a Human Task instance that holds the execution of the BPEL until some action is performed on it.
4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
5. Log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>), list all the tasks for API subscription and approve the task. It resumes the BPEL process and completes the API subscription.
6. Go back to the API Store and see that the user is now subscribed to the API.

Whenever a user tries to subscribe to an API, a request of the following format is sent to the workflow

endpoint:

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wor="http://w
orkflow.subscription.apimgt.carbon.wso2.org">
  <soapenv:Header/>
  <soapenv:Body>
    <wor:createSubscription>
      <wor:apiName>sampleAPI</wor:apiName>
      <wor:apiVersion>1.0.0</wor:apiVersion>
      <wor:apiContext>/sample</wor:apiContext>
      <wor:apiProvider>admin</wor:apiProvider>
      <wor:subscriber>subscriber1</wor:subscriber>
      <wor:applicationName>application1</wor:applicationName>
      <wor:tierName>gold</wor:tierName>
      <wor:workflowExternalRef></wor:workflowExternalRef>
      <wor:callBackURL>?</wor:callBackURL>
    </wor:createSubscription>
  </soapenv:Body>
</soapenv:Envelope>

```

Elements of the above configuration are described below:

Element	Description
apiName	Name of the API to which subscription is requested.
apiVersion	Version of the API the user subscribes to.
apiContext	Context in which the requested API is to be accessed.
apiProvider	Provider of the API.
subscriber	Name of the user requesting subscription.
applicationName	Name of the application through which the user subscribes to the API.
tierName	Throttling tiers specified for the application.
workflowExternalRef	The unique reference against which a workflow is tracked. This needs to be sent back from the workflow engine to the API Manager at the time of workflow completion.
callBackURL	The URL to which the Workflow completion request is sent to by the workflow engine, at the time of workflow completion. This property is configured under the callBackURL property in the api-manager.xml.

Adding a User Signup Workflow

This section explains how to attach a custom workflow to the application creation operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

Configuring the Business Process Server

1. Download WSO2 Business Process Server.

2. Set an offset of 2 to the default BPS port in <BPS_HOME>/repository/conf/carbon.xml file. This prevents port conflicts that occur when you start more than one WSO2 product on the same server. Also see [Changing the Default Ports with Offset](#).

```
<Offset>2</Offset>
```

! If you change the port offset to a value other than 2 or run the API Manager and BPS on different machines (therefore, want to set the `hostname` to a different value than `localhost`), you must do the following:

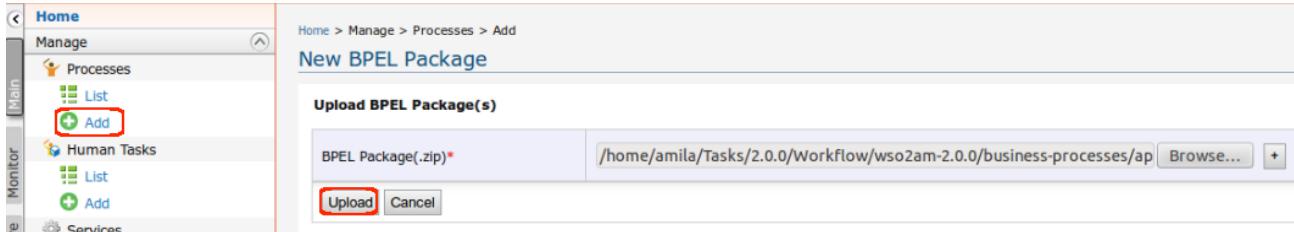
- Search and replace the value 9765 in all the files (.epr, .wsdl files inside the ZIP archives) inside <APIM_HOME>/business-processes folder with the new port.
- Zip the files you unzipped earlier and deploy the newly created zip file in BPS as explained in the steps below.
- Search and replace port 9445 in <AM_HOME>/repository/deployment/server/jagger/yapps/admin-dashboard/site/conf/site.json file.

3. Copy the following from <APIM_HOME>/business-processes/epr to <BPS_HOME>/repository/conf/epr folder. If the <BPS_HOME>/repository/conf/epr folder isn't there, please create it.

- `UserSignupService.epr`
- `UserSignupProcess.epr`

4. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).

5. Select **Add** under **Processes** menu and upload the <APIM_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip file to BPS. This is the business process archive file.

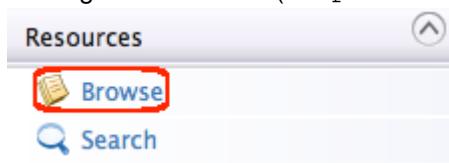


6. Select **Add** under the **Human Tasks** menu and upload <APIM_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip to BPS. This is the human task archived file.

Engaging the WS Workflow Executor in the API Manager

First, enable the user signup workflow.

1. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



2. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication).

```

<WorkFlowExtensions>
    <!--UserSignUp
    executor="org.wso2.carbon.apimgt.impl.workflow.UserSignUpSimpleWorkflowExecutor"/
-->
    <UserSignUp
    executor="org.wso2.carbon.apimgt.impl.workflow.UserSignUpWSWorkflowExecutor">
        <Property
        name="serviceEndpoint">http://localhost:9765/services/UserSignupProcess/</Propert
y>
        <Property name="username">admin</Property>
        <Property name="password">admin</Property>
        <Property
        name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Prope
rty>
    </UserSignUp>
</WorkFlowExtensions>

```

3. Go to the API Store Web interface and sign up. It invokes the signup process and creates a Human Task instance that holds the execution of the BPEL until some action is performed on it.
4. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
5. Log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>) and approve the user signup task. It resumes the BPEL process and completes the signup process.
6. Go back to the API Store and see that the user is now registered.

Whenever a user tries to sign up to the API Store, a request of the following format is sent to the workflow endpoint:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wor="http://workflow.subscription.apimgt.carbon.wso2.org">
    <soapenv:Header />
    <soapenv:Body>
        <wor:registerUser
            xmlns:wor="http://workflow.registeruser.apimgt.carbon.wso2.org">
            <wor:userName>sampleuser</wor:userName>
            <wor:tenantDomain>foo.com</wor:tenantDomain>

            <wor:workflowExternalRef>c0aad878-278c-4439-8d7e-712ee71d3f1c</wor:workflowExtern
alRef>

            <wor:callbackURL>https://localhost:8243/services/WorkflowCallbackService</wor:cal
lBackURL>
            </wor:registerUser>
        </soapenv:Body>
    </soapenv:Envelope>

```

Elements of the above configuration are described below:

Element	Description
userName	The user name requested by the user
tenantDomain	Domain to which the user belongs to

workflowExternalRef	The unique reference against which a workflow is tracked. This needs to be sent from the workflow engine to the API Manager at the time of workflow completion.
callBackURL	The URL to which the workflow completion request is sent by the workflow engine, at the time of workflow completion. This property is configured under the "callBackURL" property in the api-manager.xml.

Invoking the API Manager from the BPEL Engine

Once the workflow configurations are finalized at the BPEL, the call-back URL of the APIM, which is originally configured in the <APIM_HOME>/repository/conf/api-manager.xml file and sent to the BPEL engine in the outflow will be called to progress the workflow. In APIM, the endpoint is available in both SOAP and REST variants as follows:

Type	URI
SOAP	https://localhost:8243/services/WorkflowCallbackService WSDL Location : http://localhost:8280/services/WorkflowCallbackService?wsdl
REST	https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag

Both the endpoints are secured via basic authentication. Therefore, when you invoke either endpoint, you need to include an authorization header with a base64-encoded value of the username and password with the request. E.g., `Authorization: Basic <base64 encoded username:password>` .

The endpoint expects the following list of parameters:

Parameter	Description	Mandatory
workflowReference	The unique identifier sent to the BPEL against which the workflow is tracked in API Manager	YES
status	The next status to which the workflow needs to be promoted to.	YES
description	Notes, that may need to be persisted against a particular workflow.	NO

A sample curl request for invoking the REST endpoint is as follows:

```
curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -X POST
http://localhost:9763/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag -d
'workflowReference=b530be39-9174-43b3-acb3-2603a223b094&status=APPROVED&description=DESCRIPTION'
```

A sample SOAP request is given below:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:cal="http://callback.workflow.apimgt.carbon.wso2.org">
  <soapenv:Header/>
  <soapenv:Body>
    <cal:resumeEvent>

    <cal:workflowReference>b530be39-9174-43b3-acb3-2603a223b094</cal:workflowReference>
      <cal:status>APPROVED</cal:status>
      <cal:description>DESCRIPTION</cal:description>
    </cal:resumeEvent>
  </soapenv:Body>
</soapenv:Envelope>

```

Customizing a Workflow Extension

Each workflow executor in the WSO2 API Manager is inherited from the **org.wso2.carbon.apimgt.impl.workflow.WorkflowExecutor** abstract class, which has two abstract methods:

- **execute**: contains the implementation of the workflow execution
- **complete**: contains the implementation of the workflow completion
- **getWorkflowType**: abstract method that returns the type of the workflow as a String
- **getWorkflowDetails(String workflowStatus)**: abstract method that returns a list of **WorkflowDTO** objects. This method is not used at the moment and it returns null for the time being.

To customize the default workflow extension, you override the **execute()** and **complete()** methods with your custom implementation. For example, the following class is a sample implementation of the Subscription Creation workflow. It returns an email to an address provided through the configuration on each subscription creation:

```

package org.wso2.sample.workflow;

import java.util.List;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import org.wso2.carbon.apimgt.api.APIManagementException;
import org.wso2.carbon.apimgt.impl.APIConstants;
import org.wso2.carbon.apimgt.impl.dao.ApiMgtDAO;
import org.wso2.carbon.apimgt.impl.dto.SubscriptionWorkflowDTO;
import org.wso2.carbon.apimgt.impl.dto.WorkflowDTO;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowConstants;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowException;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowExecutor;
import org.wso2.carbon.apimgt.impl.workflow.WorkflowStatus;

public class SubsCreationEmailSender extends WorkflowExecutor {
    private String adminEmail;
    private String emailAddress;
    private String emailPassword;

    @Override

```

```

public List<WorkflowDTO> getWorkflowDetails(String arg0)
    throws WorkflowException {
    return null;
}

@Override
public String getWorkflowType() {
    return WorkflowConstants.WF_TYPE_AM_SUBSCRIPTION_CREATION;
}

@Override
public void execute(WorkflowDTO workflowDTO) throws WorkflowException{
    SubscriptionWorkflowDTO subsCreationWFDTO =
(SubscriptionWorkflowDTO)workflowDTO;

    Properties props = new Properties();
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.host", "smtp.gmail.com");
    props.put("mail.smtp.port", "587");

    Session session = Session.getInstance(props,
        new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(emailAddress,
                    emailPassword);
            }
        });
}

try {

    Message message = new MimeMessage(session);
    message.setFrom(new InternetAddress(emailAddress));
    message.setRecipients(Message.RecipientType.TO,
        InternetAddress.parse(adminEmail));
    message.setSubject("Subscription Creation");
    message.setText("Subscription created for API " +
subsCreationWFDTO.getApiName() +
                    " using Application " +
subsCreationWFDTO.getApplicationName() +
                    " by user " + subsCreationWFDTO.getSubscriber());

    Transport.send(message);
    System.out.println("Sent email to notify subscription creation");
    //Call the execute method of the parent class. This will create a
reference for the
    //workflow execution in the database.
    super.execute(workflowDTO);
    //Set the workflow Status to APPROVED and Immediately complete the
workflow since we
    //are not waiting for an external party to complete this.
    workflowDTO.setStatus(WorkflowStatus.APPROVED);
    complete(workflowDTO);

} catch (MessagingException e) {
    e.printStackTrace();
    throw new WorkflowException(e.getMessage());
} catch (Exception e){
    e.printStackTrace();
}

```

```
        throw new WorkflowException(e.getMessage());
    }
}

@Override
public void complete(WorkflowDTO workflowDTO) throws WorkflowException{
    workflowDTO.setUpdatedTime(System.currentTimeMillis());
    super.complete(workflowDTO);
    ApiMgtDAO apiMgtDAO = new ApiMgtDAO();
    try {
        apiMgtDAO.updateSubscriptionStatus(
            Integer.parseInt(workflowDTO.getWorkflowReference()),
            APIConstants.SubscriptionStatus.UNBLOCKED);
    } catch (APIManagementException e) {
        throw new WorkflowException(
            "Could not complete subscription creation workflow", e);
    }
}
public String getAdminEmail() {
    return adminEmail;
}
public void setAdminEmail(String adminEmail) {
    this.adminEmail = adminEmail;
}
public String getEmailAddress() {
    return emailAddress;
}
public void setEmailAddress(String emailAddress) {
    this.emailAddress = emailAddress;
}
public String getEmailPassword() {
    return emailPassword;
}
public void setEmailPassword(String emailPassword) {
```

```

        this.emailPassword = emailPassword;
    }
}

```

Note the following regarding the above sample:

- The `execute()` method takes in a `WorkflowDTO` object (**SubscriptionWorkflowDTO** class) that contains information about the subscription that is being created.
- The `adminEmail`, `emailAddress` and `emailPassword` are private String variables with public getter and setter methods. The values for these variables are populated through the server configuration.
- After sending the email, a call is made to the super class's `execute()` method in order to create a reference entry in the database. This entry is generally used to look up the workflow when the workflow happens asynchronously (via a human approval).
- The `complete()` method contains the code to mark the subscription active. Until then, the subscription is in `ON_HOLD` state.
- In this sample, the `complete()` method is called immediately to make the subscription active instantly. If the completion of your workflow happens asynchronously, you must not call the `complete()` method from the `execute()` method.
- The `WorkflowException` is thrown to roll back the subscription in case of a failure.

After the implementation of the class is done, follow the steps below to implement the new workflow extension in the API Manager:

1. Compile the class and export it as a JAR file. Make sure you have the following JARs in the classpath before compilation.
 - `<AM_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.impl_1.2.1.jar`
 - `<AM_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.api_1.2.1.jar`
 - `javax.mail.jar`: see <https://java.net/projects/javamail/pages/Home> to download the JAR
2. After exporting the JAR, copy it to `<AM_HOME>/repository/components/lib`.
3. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



4. Go to `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor. Also specify the service endpoint where the workflow engine is hosted and the credentials required to access the said service via basic authentication (i.e., username/password based authentication). For example:

```

<WorkFlowExtensions>
    <!--SubscriptionCreation
executor="org.wso2.carbon.apimgt.impl.workflow.SubscriptionCreationSimpleWorkflow
Executor"-->
    <SubscriptionCreation
executor="org.wso2.sample.workflow.SubsCreationEmailSender">
        <Property name="adminEmail">to_user@email.com</Property>
        <Property name="emailAddress">from _user@email.com</Property>
        <Property name="emailPassword">from_user_password</Property>
    </SubscriptionCreation>
</WorkFlowExtensions>

```

Note that the `adminEmail`, `emailAddress` and `emailPassword` properties will be assigned to the appropriate variables defined in the class through the public setter methods of those variables.

⚠ If you use the same or similar sample to return an email, you must remove the `org.jaggeryjs.hostobj
ects.email_0.9.0.ALPHA4_wso2v1.jar` file from `<AM_HOME>/repository/components/plugins` directory. Removing it results in a `ClassNotFoundException` thrown at server startup, but it does not affect the server's functionality.

Configuring Workflows for Tenants

Using the API Manager, you can configure custom workflows that get invoked at the event of a user signup, application creation, registration, subscription etc. You do these configurations in the `api-manager.xml` as described in the previous sections.

However, in a multi-tenant API Manager setup, not all tenants have access to the file system and not all tenants want to use the same workflow that the super admin has configured in the `api-manager.xml` file. For example, different departments in an enterprise can act as different tenants using the same API Manager instance and they can have different workflows. Also, an enterprise can combine WSO2 API Manager and WSO2 Business Process Server (BPS) to provide API Management As a Service to the clients. In this case, each client is a separate enterprise represented by a separate tenant. In both cases, the authority to approve business operations (workflows) resides within a tenant's space.

To allow different tenants to define their own custom workflows without editing configuration files, the API Manager provides configuration in tenant-specific locations in the registry, which you can access through the UI.

The topics below explain how to deploy a BPEL/human task using WSO2 BPS and how to point them to services deployed in the tenant spaces in the API Manager.

Deploying a BPEL and a HumanTask for a tenant

Only the users registered in the BPS can deploy BPELs and human tasks in it. Registration adds you to the user store in the BPS. In this guide, the API Manager and BPS use the same user store and all the users present in the BPS are visible to the API Manager as well. This is depicted by the diagram below:

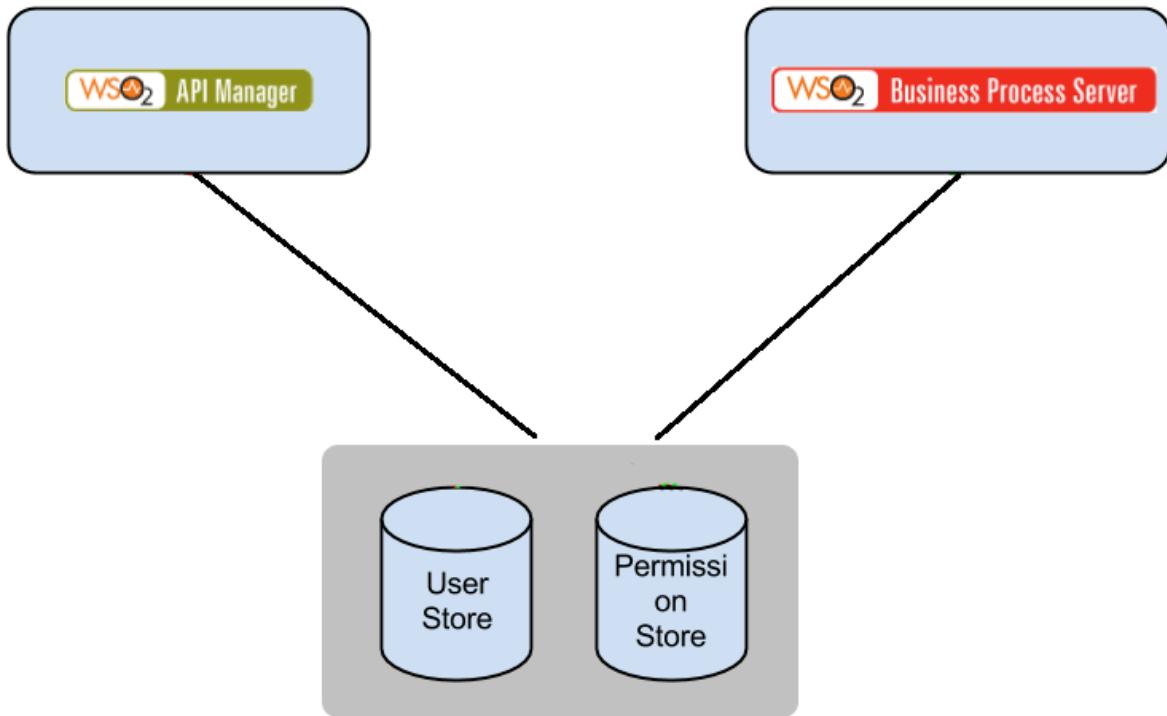


Figure: API Manager and BPS share the same user and permission store

! **If you are using WSO2 BPS 3.2.0,** please copy the `<APIM_HOME>/repository/components/patches/patch0009` folder to the `<BPS_HOME>/repository/components/patches` folder and restart the BPS server for the patch to be applied. This patch has a fix to a bug that causes the workflow configurations to fail in multi-tenant environments.

This patch is built into the BPS version 3.5.0 onwards.

Follow the steps below to deploy a BPEL and a human task for a tenant in the API Manager:

Sharing the user/permission stores with the BPS and API Manager

1. Create a database for the shared user and permission store as follows:

```
mysql> create database workflow_ustore;
Query OK, 1 row affected (0.00 sec)
```

✓ Make sure you copy the database driver (in this case, mysql driver) to the `/repository/components/lib` folder before starting each server.

2. Run the `<APIM_HOME>/dbscripts/mysql.sql` script (the script may vary depending on your database type) on the database to create the required tables.
3. Open the `<APIM_HOME>/repository/conf/datasources/master-datasources.xml` and create a datasource pointing to the newly created database. For example,

```

<datasource>
    <name>USTORE</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/ustore</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>

<url>jdbc:mysql://127.0.0.1:3306/workflow_ustore?autoReconnect=true&relaxAutoCommit=true</url>
            <username>root</username>
            <password>root</password>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>

```

4. Repeat step 2 in the BPS as well.
5. Point the datasource name in <APIM_HOME>/repository/conf/user-mgt.xml to the new datasource. (note that the user store is configured using the <UserStoreManager> element).

 If you already have a user store such as the IDAP in your environment, you can point to it from the user-mgt.xml file, instead of the user store that we created in step1.

In the following example, the same JDBC user store (that is shared by both the API Manager and the BPS) is used as the permission store as well:

```

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

```

6. Repeat step 4 in the BPS as well.

Sharing the data in the registry with the BPS and API Manager

To deploy BPELs in an API Manager tenant space, the tenant space should be accessible by both the BPS and API Manager, and certain tenant-specific data such as key stores needs to be shared with both products. Follow the steps below to create a registry mount to share the data stored in the registry:

1. Create a separate database for the registry:

```
mysql> create database workflow_regdb;
Query OK, 1 row affected (0.00 sec)
```

2. Run the <APIM_HOME>/dbscripts/mysql.sql script (the script may vary depending on your database type) on the database to create the required tables.
3. Create a new datasource in <APIM_HOME>/repository/conf/datasources/master-datasources.xml as done before:

```
<datasource>
    <name>REG_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/regdb</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>

<url>jdbc:mysql://127.0.0.1:3306/workflow_regdb?autoReconnect=true&relaxAutoC
ommmit=true</url>
        <username>root</username>
        <password>root</password>
        <driverClassName>com.mysql.jdbc.Driver</driverClassName>
        <maxActive>50</maxActive>
        <maxWait>60000</maxWait>
        <testOnBorrow>true</testOnBorrow>
        <validationQuery>SELECT 1</validationQuery>
        <validationInterval>30000</validationInterval>
    </configuration>
    </definition>
</datasource>
```

4. Add the following entries to <APIM_HOME>/repository/conf/registry.xml:

```

<dbConfig name="sharedregistry">
    <dataSource>jdbc/regdb</dataSource>
</dbConfig>

<remoteInstance url="https://localhost:9443/registry">
    <id>mount</id>
    <dbConfig>sharedregistry</dbConfig>
    <readOnly>false</readOnly>
    <enableCache>true</enableCache>
    <registryRoot>/</registryRoot>
</remoteInstance>
<!-- This defines the mount configuration to be used with the remote instance
and the target path for the mount --&gt;
&lt;mount path="/_system/config" overwrite="true"&gt;
    &lt;instanceId&gt;mount&lt;/instanceId&gt;
    &lt;targetPath&gt;/_system/nodes&lt;/targetPath&gt;
&lt;/mount&gt;
&lt;mount path="/_system/governance" overwrite="true"&gt;
    &lt;instanceId&gt;mount&lt;/instanceId&gt;
    &lt;targetPath&gt;/_system/governance&lt;/targetPath&gt;
&lt;/mount&gt;
</pre>

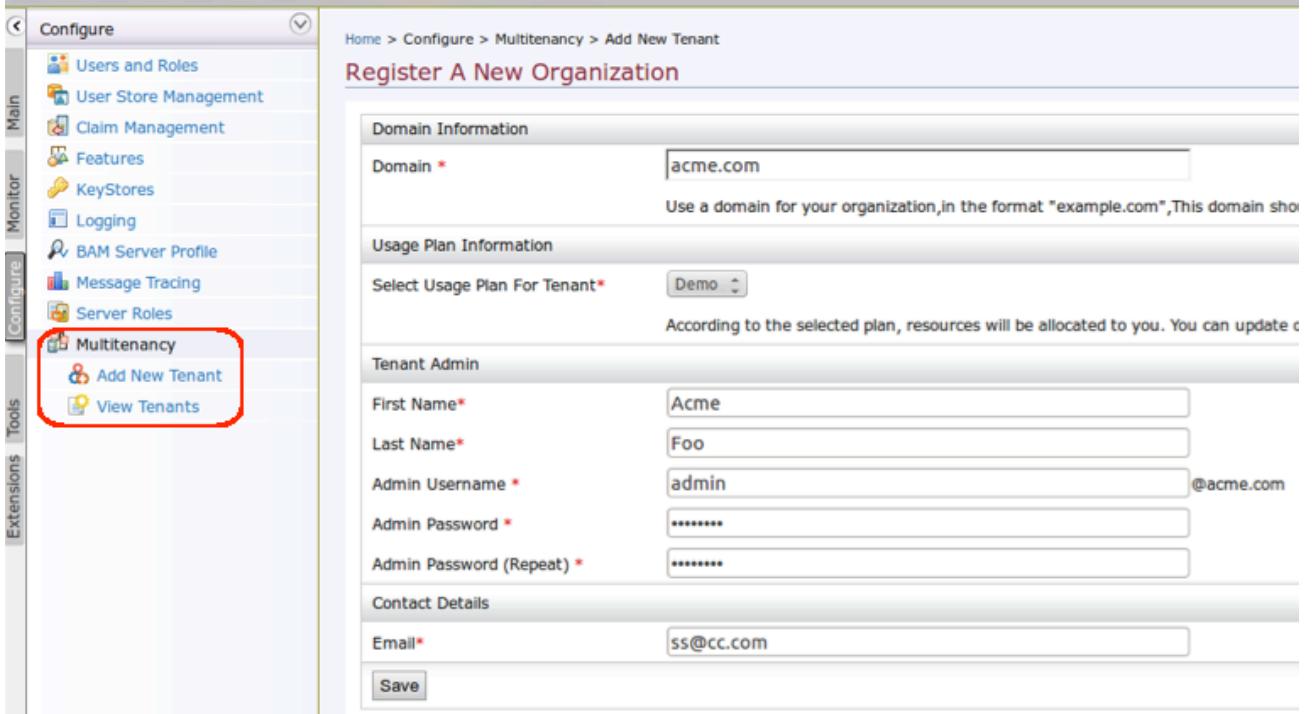
```

5. Repeat the above three steps in the BPS as well.

Creating a BPEL

In this section, you create a BPEL that has service endpoints pointing to services hosted in the tenant's space. This example uses the [Application Creation](#) workflow.

1. Set a port offset of 2 to the BPS using the `<BPS_HOME>/repository/conf/carbon.xml` file. This prevents any port conflicts when you start more than one WSO2 products on the same server.
2. Log in to the API Manager's management console (<https://localhost:9443/carbon>) and create a tenant using the **Configure** -> **Multitenancy** menu.



The screenshot shows the WSO2 API Manager management console interface. The left sidebar has a 'Configure' tab selected, with a 'Multitenancy' section highlighted by a red box. Under 'Multitenancy', there are two options: 'Add New Tenant' and 'View Tenants'. The main content area is titled 'Register A New Organization'. It contains several input fields and dropdowns for tenant configuration. The 'Domain' field is set to 'acme.com'. The 'Select Usage Plan For Tenant' dropdown is set to 'Demo'. The 'Tenant Admin' section includes fields for 'First Name' (Acme), 'Last Name' (Foo), 'Admin Username' (admin), 'Admin Password' (redacted), 'Admin Password (Repeat)' (redacted), and 'Email' (ss@cc.com). A 'Save' button is at the bottom.

3. Create a copy of the BPEL located in <APIM_HOME>/business-processes/application-creation/BPEL.
4. Extract the contents of the new BPEL archive.
5. Copy ApplicationService.epr and ApplicationCallbackService.epr from <APIM_HOME>/business-processes/epr folder to the folder extracted before. Then, rename the two files as ApplicationService-Tenant.epr and ApplicationCallbackService-Tenant.epr respectively.
6. Open ApplicationService-Tenant.epr and change the wsa:Address to http://localhost:9765/services/t/<tenant domain>/ApplicationService.
7. Point the deploy.xml file of the extracted folder to the new .epr files provided in the BPEL archive. For example,

```

<invoke partnerLink="AAPL">
    <service name="applications:ApplicationService" port="ApplicationPort">
        <endpoint xmlns="http://wso2.org/bps/bpel/endpoint/config"
        endpointReference="ApplicationService-Tenant.epr"></endpoint>
    </service>
</invoke>

<invoke partnerLink="CBPL">
    <service
    name="callback.workflow.apimgt.carbon.wso2.org:WorkflowCallbackService"
    port="WorkflowCallbackServiceHttpsSoap11Endpoint">
        <endpoint xmlns="http://wso2.org/bps/bpel/endpoint/config"
        endpointReference="ApplicationCallbackService-Tenant.epr"></endpoint>
    </service>
</invoke>

```

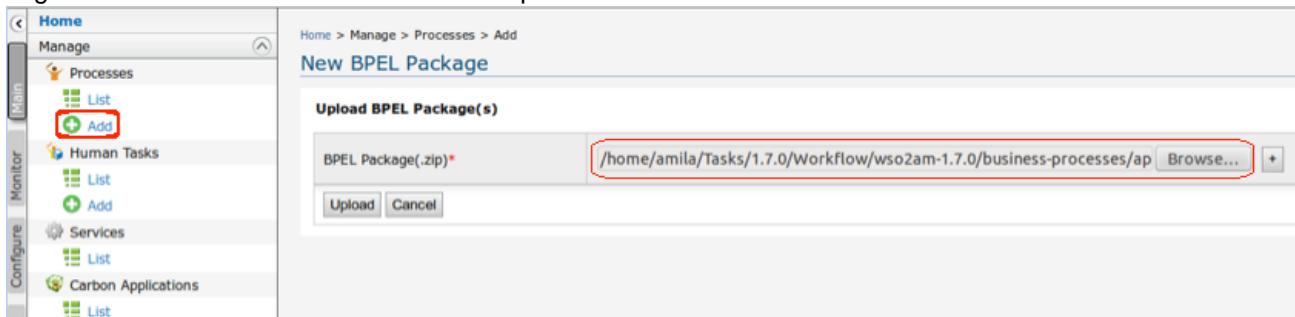
8. Zip the content and create a BPEL archive in the following format:

```

ApplicationApprovalWorkFlowProcess_1.0.0-Tenant.zip
|_ApplicationApprovalWorkFlowProcess.bpel
|_ApplicationApprovalWorkFlowProcessArtifacts.wsdl
|_ApplicationCallbackService-Tenant.epr
|_ApplicationService-Tenant.epr
|_ApplicationsApprovalTaskService.wsdl
|_SecuredService-service.xml
|_WorkflowCallbackService.wsdl
|_deploy.xml

```

9. Log into the BPS as the tenant admin and upload the BPEL.



Creating a human task

Similar to creating a BPEL, create a HumanTask that has service endpoints pointing to services hosted in the tenant's space.

1. Create a copy of the HumanTask archive in <APIM_HOME>/business-processes/application-creation/HumanTask and extract its contents.
2. Edit the SOAP service port-bindings in ApplicationApprovalTaskService.wsdl. For example,

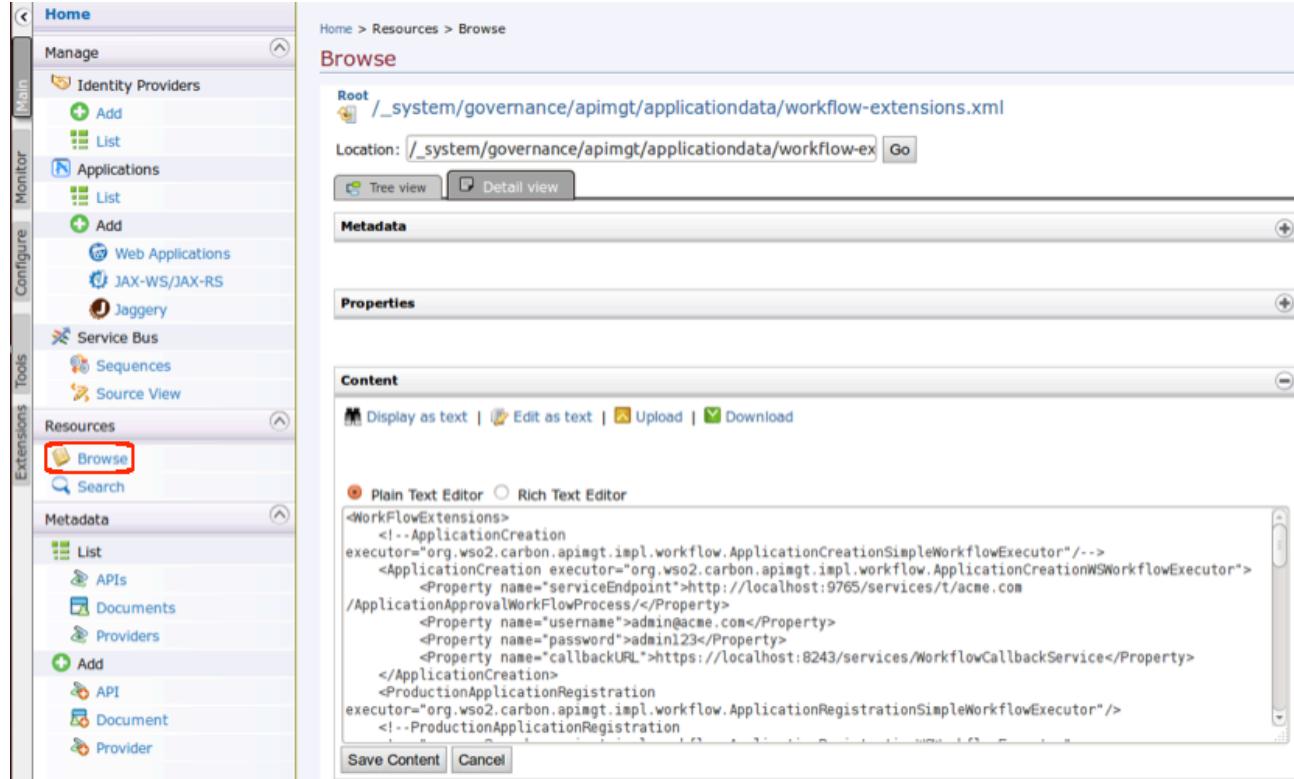
```

<wsdl:service name="ApplicationService">
  <wsdl:port name="ApplicationPort" binding="tns:ApplicationSoapBinding">
    <soap:address location="http://localhost:9765/services/t/<tenant
domain>/ApplicationService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="ApplicationReminderService">
  <wsdl:port name="ApplicationReminderPort"
binding="tns:ApplicationSoapBindingReminder">
    <soap:address location="http://localhost:9765/services/t/<tenant
domain>/ApplicationReminderService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="ApplicationServiceCB">
  <wsdl:port name="ApplicationPortCB" binding="tns:ApplicationSoapBindingCB">
    <soap:address location="http://localhost:9765/services/t/<tenant
domain>/ApplicationServiceCB" />
  </wsdl:port>
</wsdl:service>

```

3. Create the HumanTask archive by zipping all the extracted files.
4. Log into the BPS as the tenant admin and upload the HumanTask.
5. Log into the API Manager's management console as the tenant admin and select **Resources > Browse** menu.
6. Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` in the registry and change the **service endpoint as a tenant-aware service URL** (e.g., `http://localhost:9765/services/t/<tenant_domain>/ApplicationApprovalWorkFlowProcess`). Also set the **credentials**

als as the **tenant admin's credentials** of the ApplicationCreationWSWorkflowExecutor file. For example,



The screenshot shows the WSO2 API Manager Governance interface. The left sidebar has a 'Main' tab selected, with 'Browse' highlighted. The main content area shows the 'Browse' page for the 'Root' node. The 'Properties' tab is visible, and the 'Content' tab is selected, displaying XML code for the ApplicationCreationWSWorkflowExecutor. The XML code includes properties for service endpoint, username, password, and callback URL.

```

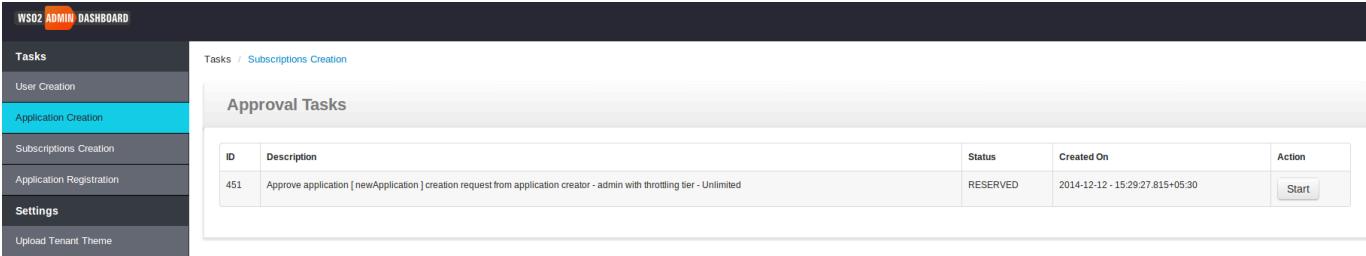
<WorkFlowExtensions>
    <!--ApplicationCreation
    executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationCreationSimpleWorkflowExecutor"-->
    <ApplicationCreation executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationCreationWSWorkflowExecutor">
        <Property name="serviceEndpoint">http://localhost:9765/services/t/acme.com</Property>
        <Property name="username">admin@acme.com</Property>
        <Property name="password">admin123</Property>
        <Property name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
    </ApplicationCreation>
    <ProductionApplicationRegistration
    executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationSimpleWorkflowExecutor"/>
        <!--ProductionApplicationRegistration
    </ProductionApplicationRegistration

```

⚠ Be sure to disable the SimpleWorkflowExecutor and enable the ApplicationCreationWSWorkflowExecutor.

Testing the workflow

You have now completed configuring the Application Creation workflow for a tenant. Whenever a tenant user logs in to the tenant store and create an application, the workflow will be invoked. You log in to the Admin Dashboard Web application (<https://<Server Host>:9443/admin-dashboard>) as the tenant admin and browse **Application Creation** menu to see all approval tasks have been created for newly created applications.



The screenshot shows the WSO2 Admin Dashboard. The left sidebar has a 'Tasks' section with 'Application Creation' selected and highlighted in blue. The main content area shows the 'Tasks' page with 'Subscriptions Creation' selected. The 'Approval Tasks' table lists a single task with the following details:

ID	Description	Status	Created On	Action
451	Approve application [newApplication] creation request from application creator - admin with throttling tier - Unlimited	RESERVED	2014-12-12 - 15:29:27.815+05:30	Start

Configuring Workflows in a Cluster

If you are working in a clustered API Manager setup with the API Store, Publisher, Gateway and Key Manager in separate servers, do the workflow configurations that are discussed in the previous topics in the **API Store node**. In addition, do the following configurations.

In this guide, you access the Admin Dashboard (<https://<Server Host>:9443/admin-dashboard>) Web application using the same node as the API Publisher. This is recommended because workflow management is an administrative task and is meant to reside within a private network as the Publisher. Typically, the Admin

Dashboard from the same user store as the API Manager. Therefore, you can use the Admin Dashboard residing in the Publisher node instead of having it separately. This eliminates the need for a dedicated workflow management node. You need a dedicated node if the Admin Dashboard users reside in a separate user store.

1. If you want to change the user roles that can access the Admin Dashboard, open the `<APIM_HOME>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf/site.json` file that is in the node from where you access the Admin Dashboard (the API Publisher node in this example) and change its `Allowed Roles` parameter. You can add multiple user roles as a comma-separated list.
2. By default, workflow related configuration files have the port of the Business Process Server with an offset of 2. If you set up the BPS with a different port offset, change the workflow server URLs in the `site.json` file accordingly.
3. Point the `<Address>` sub element of the `<endpoint>` element to the API Store node in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/WorkflowCallbackService.xml` file of the API Store node.

```

<endpoint>
  <address
    uri="https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag" format="rest"/>
</endpoint>

```

4. Add the IP address and the port of the API Store to the `<Address>` element of the `.epr` file of the workflow that you configure. You can find the `.epr` file by the name of the workflow in the `<APIM_HOME>/business-processes/epr` folder.
5. Go to the `<APIM_HOME>/business-processes/<workflow name>/BPEL` folder and unzip the file that is there by the name of the workflow. For example, `<APIM_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip` .
6. Go inside the unzipped folder and do the following:

Action	Example
Open the ApprovalTask WSDL file and point the address elements of the server where the BPEL runs.	<p>In the <code><APIM_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0/UserApprovalTask.wsdl</code> file:</p> <pre> <wsdl:service name="UserApprovalService"> <wsdl:port name="UserApprovalPort" binding="tns:UserApprovalBinding"> <soap:address location="http://localhost:9783/services/UserApprovalService"> </wsdl:port> </wsdl:service> <wsdl:service name="UserApprovalServiceCB"> <wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBinding"> <soap:address location="http://localhost:9783/services/UserApprovalServiceCB"> </wsdl:port> </wsdl:service> </wsdl:service> </pre>

<p>Open the ProcessArtifacts WSDL file and point the address elements to the API Store node.</p>	<p>In the <code><APIM_HOME>/business-processes/user-signup/BPEL/UserSignupProcess.wsdl</code> file:</p> <pre data-bbox="546 264 1512 559"> <service name="UserSignupProcess"> <port binding="tns:UserSignupProcessBinding" name="UserSignupProcessPort"> <soap:address location="http://apim.180.erandi.store.com:80/1/UserSignupProcess"/> </port> </service> <service name="UserSignupProcessCallback"> <port binding="tns:UserSignupProcessCallbackBinding" name="UserSignupProcessCallbackPort"> <soap:address location="http://apim.180.erandi.store.com:80/1/UserSignupProcessCallback"/> </port> </service> </pre>
<p>Open the CallbackService WSDL file and point the address elements to the Business Process Server node.</p>	<p>In the <code><APIM_HOME>/business-processes/user-signup/BPEL/UserSignupProcess.wsdl</code> file:</p> <pre data-bbox="546 770 1512 1700"> <wsdl:service name="WorkflowCallbackService"> <wsdl:port name="WorkflowCallbackServiceHttpsSoap11Endpoint" binding="ns:WorkflowCallbackServiceSoap11Binding"> <soap:address location="https://10.100.5.63:8243/services/WorkflowCallbackService"/> </wsdl:port> <wsdl:port name="WorkflowCallbackServiceHttpSoap11Endpoint" binding="ns:WorkflowCallbackServiceSoap11Binding"> <soap:address location="http://10.100.5.63:8280/services/WorkflowCallbackService"/> </wsdl:port> <wsdl:port name="WorkflowCallbackServiceHttpsSoap12Endpoint" binding="ns:WorkflowCallbackServiceSoap12Binding"> <soap12:address location="https://10.100.5.63:8243/services/WorkflowCallbackService"/> </wsdl:port> <wsdl:port name="WorkflowCallbackServiceHttpSoap12Endpoint" binding="ns:WorkflowCallbackServiceSoap12Binding"> <soap12:address location="http://10.100.5.63:8280/services/WorkflowCallbackService"/> </wsdl:port> <wsdl:port name="WorkflowCallbackServiceHttpsEndpoint" binding="ns:WorkflowCallbackServiceHttpsBinding"> <http:address location="https://10.100.5.63:8243/services/WorkflowCallbackService"/> </wsdl:port> <wsdl:port name="WorkflowCallbackServiceHttpEndpoint" binding="ns:WorkflowCallbackServiceHttpBinding"> <http:address location="http://10.100.5.63:8280/services/WorkflowCallbackService"/> </wsdl:port> </wsdl:service> </pre>

7. Go to the `<APIM_HOME>/business-processes/<workflow name>/HumanTask` folder and unzip the file that is there by the name of the workflow. For example, `<APIM_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip` .
8. Go inside the unzipped folder and do the following:

Action	Example
If you changed the default admin user, open the ApprovalTask HT file and apply the changes there.	Change the admin instances in <APIM_HOME>/business-processes/user-signup/HumanTask-1.0.0/UserApprovalTask.ht file.
Open the ApprovalTask WSDL file and point the two address elements to the Business Process Server node.	<p>In the <APIM_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0/UserApprovalTask.wsdl file:</p> <pre> <wsdl:service name="UserApprovalService"> <wsdl:port name="UserApprovalPort" binding="tns:UserApprovalBinding"> <soap:address location="http://localhost:9783/services/UserApprovalService" /> </wsdl:port> </wsdl:service> <wsdl:service name="UserApprovalServiceCB"> <wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalBindingCB"> <soap:address location="http://localhost:9783/services/UserApprovalServiceCB" /> </wsdl:port> </wsdl:service> </pre>

Changing the Default User Role in Workflows

The default user role in the workflow configuration files is the admin role. If you change this to something else, you need to change the following files:

1. Change the credentials in the .epr files of the <BPS_HOME>/repository/conf/epr folder.
2. Change the credentials in work-flow configurations in API Manager Registry (`_system/governance/apimgt/applicationdata/workflow-extensions.xml`)
3. Point the same database which has the permissions that is used by API Manager to BPS
4. Share L D A P s if exists
5. The credential details in apimanager.xml should be changed

If the default role is changed then the .ht file of the relevant human task should be changed accordingly. If the default role is changed then the site.json (allowedRoles) of (<Product_Home>/repository/deployment/server/jaggeryapps/admin-dashboard/site/conf) should be changed.

The current documentation does not provide any information regarding the above details. It would be a help to the user if the are included in the documentation.

Adding new Throttling Tiers

API Manager admins can add new throttling tiers and define extra properties to throttling tiers using the management console as discussed below. For a description of throttling tiers, see [API-level throttling](#).

1. Log in to the API Manager's Management Console and select **Browse** under **Resources** menu.
2. Select the file: `/_system/governance/apimgt/applicationdata/tiers.xml`.

Browse

Root /

Location: / Tree view Detail view

/
 _system
 config
 governance
 apimgt
 applicationdata
 provider
 tiers.xml
 event
 permission

3. In the **Contents** panel, click **Edit as text** link and the throttling policy opens.
4. You can add a new policy configuration by editing the XML code. For example, we have added a new tier called `Platinum` by including the following XML code block soon after the `<throttle:MediatorThrottle Assertion>` element.

Tier DisplayName : You can add this **optional** attribute to each throttle ID of `tiers.xml` file in order to decouple the throttle policy name defined in `tiers.xml` from the tier name showing in APIPublisher/Store UIs. That is, a user can add a different throttle display name to appear in APIPublisher/Store UIs without changing the throttle ID policy name. The configuration below has a `displayName` as `platinio` for the throttle value `platinum`. This value is displayed in APIPublisher/Store apps.

Tier Attributes : In the configuration below, there's a commented out XML section starting from the XML tag `<throttle:Attributes>`. You can use it to define additional attributes related to each throttling tier definition. For example, if the throttling tier `Platinum` has attributes called `PaymentPlan` and `Availability`, first uncomment the `<throttle:Attributes>` section and then define the new attributes as follows:

```

<wsp:Policy>
  <throttle:ID throttle:type="ROLE"
  throttle:displayName="platino">Platinum</throttle:ID>
    <wsp:Policy>
      <throttle:Control>
        <wsp:Policy>
          <throttle:MaximumCount>50</throttle:MaximumCount>
          <throttle:UnitTime>60000</throttle:UnitTime>
          <!--It's possible to define tier level attributes as below for
          each tier level. For eg:Payment Plan for a tier-->
          <wsp:Policy>
            <throttle:Attributes>
              <!--throttle:Attribute1>xxxx</throttle:Attribute1-->
              <!--throttle:Attribute2>xxxx</throttle:Attribute2-->
              <throttle:PaymentPlan>monthly</throttle:PaymentPlan>
              <throttle:Availability>FullTime</throttle:Availability>
            </throttle:Attributes>
          </wsp:Policy>
        </wsp:Policy>
      </throttle:Control>
    </wsp:Policy>
  </wsp:Policy>

```

5. After the edits, click **Save Content**. Your new throttling policy (Platinum) is now successfully saved in the Repository used by WSO2 API Manager. You can view this new throttle tier available for selection when creating a new API through the API Publisher.

Adding a Reverse Proxy Server

A reverse proxy server retrieves information from a server and sends it to a client as though the information originated from the sever rather than the reverse proxy server. You can use a reverse proxy server to block access to selected applications in a server. For example, this is useful when you want to expose the token API in such a way that the clients can authenticate against OAuth2 using the same port that their API's are on.

The API Manager comes with two Web applications as the Publisher and Store. You can route the requests that come to them through a proxy server by editing the `<AM_HOME>/repository/deployment/server/jaggeryapps/store(/publisher)/site/conf/site.json` file. For example, to use a reverse proxy server for the API Store, edit the `<AM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` file with the context and request URL as shown below.

```

"context" : "/public/store",
"request_url": "https://localhost/public/store/",

```

If you set up the reverse proxy server correctly, when you access the URL <https://localhost/public/store>, you will be directed to the API Store.

To do the same for the API Publisher, edit the `<AM_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json` file.

Also note that if you want to change all the default API Manager ports, you do so by editing the `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` file.

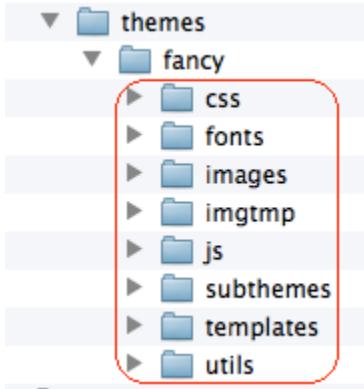
Adding a new API Store Theme

A **theme** consists of UI elements such as logos, images, copyrights messages, landing page text, background colors etc. WSO2 API Store comes with a default theme.

The folder structure of the API Store themes

The default theme of the API Store is called **Fancy**. You find it inside the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/fancy` folder. If you do not have access to the file system, do [download the default theme from here](#).

The easiest way to create a new theme is to copy the files of an existing theme to a folder by the name of your new theme, and do the modifications you want to the files inside it. All themes have the same folder structure as shown below:



You can add a new theme as a main theme or a sub theme.

- **A main theme** is saved inside the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes` folder
- **A sub theme** is saved inside the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/<folder of the main theme>/subthemes` folder.

Because a sub theme is saved inside a main theme, it needs to contain only the files that are different from the main theme. Any file you add inside the sub theme will override the corresponding files in the main theme. The rest of the files will be inherited from the main theme.

Let's see how to create a new theme and set it to the API Store:

- [Writing a sub theme of the main theme](#)
- [Setting the new theme as the default theme](#)
- [Adding the new theme to the Themes menu](#)

Writing a sub theme of the main theme

Because a main theme already has most of the UIs and the syntax and logic of Jaggery code defined, in a typical scenario, you do not have to implement a theme from scratch. Rather, you just add in your edits as a sub theme of the existing main theme as given below:

1. Download the default main theme [from here](#), unzip it and rename the folder according to the name of your new theme (e.g., **ancient**). Let's call this folder the `<THEME_HOME>`.
2. To change the logo of the API Store, replace the `logo.png` file inside the `<THEME_HOME>/images` folder with [this logo](#) (or anything else of your choice.)
3. To change the copyrights note in the footer, open the `<THEME_HOME>/templates/page/base/template.jag` file using a text editor, search for the word "Copyright" and change the text. For example, let's add our company name as "copyright", "© Copyright 2011 – 2014 **My Company**."
4. Open the `<THEME_HOME>/css/styles-layout.css` file using a text editor and add the following CSS code to the end of the file. Note the code comments to get an idea what each line of code does.

```

/* Change the color of the header */
.header{
    background:#002EB8;
}
/* Change the font of the menus, headings, labels etc. to Verdana. You give
several fonts here to ensure maximum compatibility, if in case one font fails in
a given browser/OS. Fonts will be applied in the order you list them. */
body,textarea,pre,.navbar-search .search-query{
    font-family: Verdana, Arial, Helvetica, monospace, sans-serif;
}
label, input, button, select, textarea{
    font-family: Verdana, Arial, Helvetica, monospace, sans-serif;
}
h1,h2,h3,h4,h5{
    font-family: Verdana, Arial, Helvetica, monospace, sans-serif;
}
/* To change the background color of the body */
body{
    background:#D6DEF6;
}
/* To change the color of the buttons. Note that changing only the background
color will not have a visual impact if you leave the gradients as they are */
.btn-primary {
    background-color: #800004;
    background-image: linear-gradient(to bottom, #cc0022, #cc0044);
}
/* To change the colour of the menus, navigation elements when they are clicked
*/
.menu-content .navbar .nav > .active > a, .navbar .nav > .active > a:hover,
.navbar .nav > .active > a:focus {
    background:#F0F3FC;
}

```

5. As you plan to upload this as a sub theme of the default main theme, delete all the files in your <THEME_HOME> folder except the ones that you edited. The rest of the files will be automatically applied from the main theme.

Setting the new theme as the default theme

You can set your new theme as the default theme in two ways:

- Saving directly in the file system
- Uploading through the Admin Dashboard

Saving directly in the file system

If you have access to the file system, do the following:

1. Save the <THEME_HOME> folder inside the <APIM_HOME>/repository/deployment/server/jaggery apps/store/site/themes/fancy/subthemes folder. This will make your new theme a sub theme of fancy.
2. Open the <APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file and add the following code to it. It specifies the base theme as fancy, which is overridden by the sub theme ancient.

```

"theme" : {
    "base" : "fancy",
    "subtheme" : "ancient"
}

```

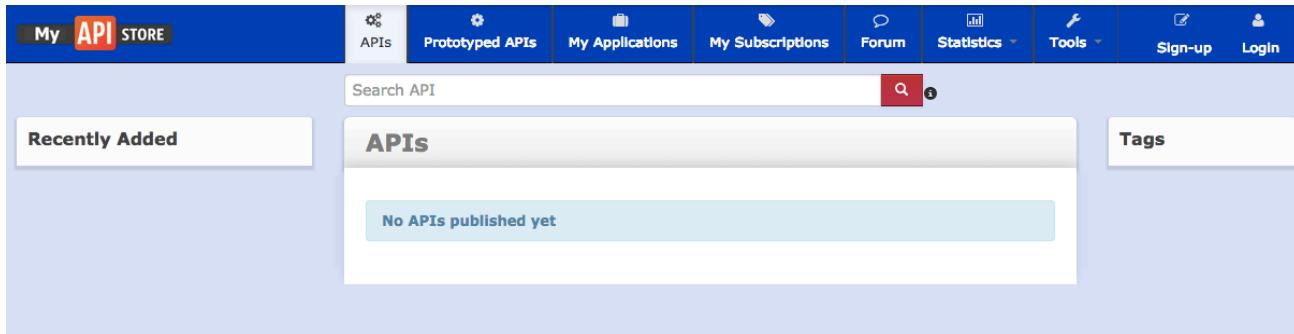
3. Open the API Store and note the new theme applied to it.

Uploading through the Admin Dashboard

If you do not have access to the file system, you can upload the theme through the Admin Dashboard Web application as shown below:

1. Go inside the <THEME_HOME> folder, select all the folders inside it and right click to archive all the selected files and folders together. Then rename the archive files to `ancient.zip`.
2. Log in to WSO2 Admin Dashboard Web application using the URL `https://<Server Host>:9443/admin-dashboard`.
For example, if you are a WSO2 Cloud user and want to upload a new theme, log in to the URL `api.cloud.wso2.com/admin-dashboard` with the user name as `email@domain` with the @ in the email replaced by a dot (e.g., `john@gmail.com@MyCompany`).
3. Click the **Upload Tenant Theme** menu and upload your zip file.

4. Open the API Store and note the new theme applied to it.



Adding the new theme to the Themes menu

Once you are done modifying the new theme, add it to the **Themes** menu in the API Store along with a thumbnail image as follows:

1. Open the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/fancy/templates/user/login/template.jag` file and find the HTML table that defines the theme thumbnails.
2. Add a new row under the `<table>` element with the following code. It adds `thumb-ancient.png` as the thumbnail image of our theme. Be sure save the image in the `...fancy/images` folder.

```

<td>
    <div class="thumbnail" <% if(jagg.getUserTheme().base == "fancy" &&
jagg.getUserTheme().subtheme == "ancient") { %>currentTheme<% } %>">
        <a data-theme="fancy" data-subtheme="ancient" class="badge themeLabel"
onclick="applyTheme(this)">
            " />
            <br /><div class="themeName">Ancient</div>
        </a>
    </div>
</td>

```

Transforming API Message Payload

When a request comes to the API Manager, it sends the response in the same format of the request. For example, the API Manager handles JSON to JSON transformations out of the box. If the backend does not accept messages of the same content type of the request message, it must be transformed to a different format. The API Gateway of the API Manager handles these transformations using message builders and formatters.

When a message comes in to the API Gateway, the receiving transport selects a **message builder** based on the message's content type. It uses that builder to process the message's raw payload data and convert it into JSON. Conversely, when sending a message out from the Gateway, a **message formatter** is used to build the outgoing stream from the message. As with message builders, the message formatter is selected based on the message's content type.

- [JSON message builders and formatters](#)
- [XML representation of JSON payloads](#)
- [Converting a payload between XML and JSON](#)

 Note that if you edit an API's synapse configuration as mentioned in this guide and then go back to the API Publisher and save the API, your changes will be overwritten. As a result, we do not recommend changing the API's synapse configuration directly. The recommended way to extend an API's mediation flow is by engaging In/Out sequences.

Also see the following sections in the WSO2 ESB documentation. WSO2 ESB is used to implement the API Gateway through which API messages are transformed:

- Accessing content from JSON payloads
- Logging JSON payloads
- Constructing and transforming JSON payloads
- Troubleshooting, debugging, and logging

JSON message builders and formatters

There are two types of message builders and formatters for JSON. The default builder and formatter keep the JSON representation intact without converting it to XML. You can access the payload content using the JSON Path or XPath and convert the payload to XML at any point in the mediation flow.

- `org.apache.synapse.commons.json.JsonStreamBuilder`
- `org.apache.synapse.commons.json.JsonStreamFormatter`

If you want to convert the JSON representation to XML before the mediation flow begins, use the following builder and formatter instead. Note that some data loss can occur during the JSON -> XML -> JSON conversion process.

- `org.apache.synapse.commons.json.JsonBuilder`
- `org.apache.synapse.commons.json.JsonFormatter`

The builders and formatters are configured respectively in the `messageBuilders` and `messageFormatters` sections of the Axis2 configuration files located in the `<PRODUCT_HOME>/repository/conf/axis2` directory. Both types of JSON builders use `StAXON` as the underlying JSON processor.

The following builders and formatters are also included for compatibility with older API Manager versions:

- `org.apache.axis2.json.JSONBuilder/JSONMessageFormatter`
- `org.apache.axis2.json.JSONStreamBuilder/JSONStreamFormatter`
- `org.apache.axis2.json.JSONBadgerfishOMBuilder/JSONBadgerfishMessageFormatter`



Always use the same type of builder and formatter combination. Mixing different builders and formatters will cause errors at runtime.

If you want to handle JSON payloads that are sent using a media type other than `application/json`, you must register the JSON builder and formatter for that media type in the following two files at minimum (for best results, register them in all Axis2 configuration files found in the `<PRODUCT_HOME>/repository/conf/axis2` directory):

- `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml`
- `<PRODUCT_HOME>/repository/conf/axis2/axis2_blocking_client.xml`

For example, if the media type is `text/javascript`, register the message builder and formatter as follows:

```
<messageBuilder contentType="text/javascript"
    class="org.apache.synapse.commons.json.JsonStreamBuilder"/>

<messageFormatter contentType="text/javascript"
    class="org.apache.synapse.commons.json.JsonStreamFormatter"/>
```



When you modify the builders/formatters in Axis2 configuration, make sure that you have enabled only one correct message builder/formatter pair for a given media type.

XML representation of JSON payloads

When building the XML tree, JSON builders attach the converted XML infoset to a special XML element that acts as the root element of the final XML tree. If the original JSON payload is of type `object`, the special element is `<json`

Object/>. If it is an array, the special element is <jsonArray/>. Following are examples of JSON and XML representations of various objects and arrays.

Null objects

JSON:

```
{ "object":null}
```

XML:

```
<jsonObject>
  <object></object>
</jsonObject>
```

Empty objects

JSON:

```
{ "object":{ } }
```

XML:

```
<jsonObject>
  <object></object>
</jsonObject>
```

Empty strings

JSON:

```
{ "object":"" }
```

XML:

```
<jsonObject>
  <object></object>
</jsonObject>
```

Empty array

JSON:

```
[ ]
```

XML (JsonStreamBuilder):

```
<jsonArray></jsonArray>
```

XML (JsonBuilder):

```
<jsonArray>
  <?xml-multiple jsonElement?>
</jsonArray>
```

Named arrays

JSON:

```
{ "array" : [ 1 , 2 ] }
```

XML (JsonStreamBuilder):

```
<jsonObject>
  <array>1</array>
  <array>2</array>
</jsonObject>
```

XML (JsonBuilder):

```
<jsonObject>
  <?xml-multiple array?>
  <array>1</array>
  <array>2</array>
</jsonObject>
```

JSON:

```
{ "array" : [ ] }
```

XML (JsonStreamBuilder):

```
<jsonObject></jsonObject>
```

XML (JsonBuilder):

```
<jsonObject>
  <?xml-multiple array?>
</jsonObject>
```

Anonymous arrays

JSON:

```
[1,2]
```

XML (JsonStreamBuilder):

```
<jsonArray>
  <jsonElement>1</jsonElement>
  <jsonElement>2</jsonElement>
</jsonArray>
```

XML (JsonBuilder):

```
<jsonArray>
  <?xml-multiple jsonElement?>
  <jsonElement>1</jsonElement>
  <jsonElement>2</jsonElement>
</jsonArray>
```

JSON:

```
[1, []]
```

XML (JsonStreamBuilder):

```
<jsonArray>
  <jsonElement>1</jsonElement>
  <jsonElement>
    <jsonArray></jsonArray>
  </jsonElement>
</jsonArray>
```

XML (JsonBuilder):

```
<jsonArray>
  <?xml-multiple jsonElement?>
  <jsonElement>1</jsonElement>
  <jsonElement>
    <jsonArray>
      <?xml-multiple jsonElement?>
    </jsonArray>
  </jsonElement>
</jsonArray>
```

XML processing instructions (PIs)

Note that the addition of `xml-multiple` processing instructions to the XML payloads whose JSON representations contain arrays. `JsonBuilder` (via StAXON) adds these instructions to the XML payload that it builds during the JSON to XML conversion so that during the XML to JSON conversion, `JsonFormatter` can reconstruct the arrays

that are present in the original JSON payload. `JsonFormatter` interprets the elements immediately following a processing instruction to construct an array.

Special characters

When building XML elements, the '\$' character and digits are handled in a special manner when they appear as the first character of a JSON key. Following are examples of two such occurrences. Note the addition of the `_JsonReader_PS_` and `_JsonReader_PD_` prefixes in place of the '\$' and digit characters, respectively.

JSON:

```
{ "$key":1234 }
```

XML:

```
<jsonObject>
  <_JsonReader_PS_key>1234</_JsonReader_PS_key>
</jsonObject>
```

JSON:

```
{ "32X32": "image_32x32.png" }
```

XML:

```
<jsonObject>
  <_JsonReader_PD_32X32>image_32x32.png</_JsonReader_PD_32X32>
</jsonObject>
```

Converting a payload between XML and JSON

To convert an XML payload to JSON, set the `messageType` property to `application/json` in the `axis2` scope before sending message to an endpoint. Similarly, to convert a JSON payload to XML, set the `messageType` property to `application/xml` or `text/xml`. For example:

```

<api name="admin--TOJSON" context="/tojson" version="1.0" version-type="url">
    <resource methods="POST GET DELETE OPTIONS PUT" url-mapping="/*">
        <inSequence>
            <property name="POST_TO_URI" value="true" scope="axis2"/>
            <property name="messageType" value="application/json" scope="axis2"/>
            <filter source="$ctx:AM_KEY_TYPE" regex="PRODUCTION">
                <then>
                    <send>
                        <endpoint name="admin--Test_APIproductionEndpoint_0">
                            <http
uri-template="http://localhost:9767/services/StudentService">
                                <timeout>
                                    <duration>30000</duration>
                                    <responseAction>fault</responseAction>
                                </timeout>
                                <suspendOnFailure>
                                    <errorCodes>-1</errorCodes>
                                    <initialDuration>0</initialDuration>
                                    <progressionFactor>1.0</progressionFactor>
                                    <maximumDuration>0</maximumDuration>
                                </suspendOnFailure>
                                <markForSuspension>
                                    <errorCodes>-1</errorCodes>
                                </markForSuspension>
                            </http>
                        </endpoint>
                    </send>
                </then>
                <else>
                    <sequence key="_sandbox_key_error_" />
                </else>
            </filter>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
            <property name="id" value="A"/>
            <property name="policyKey"
value="gov:/apimgt/applicationdata/tiers.xml"/>
        </handler>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
    </handlers>
</api>

```

An example command to invoke above API:

```
curl -v -X POST -H "Content-Type:application/xml" -H "Authorization: Bearer xxx" -d@request1.xml "http://10.100.1.110:8280/tojson/1.0"
```

If the request payload is as follows:

```
<coordinates>
  <location>
    <name>Bermuda Triangle</name>
    <n>25.0000</n>
    <w>71.0000</w>
  </location>
  <location>
    <name>Eiffel Tower</name>
    <n>48.8582</n>
    <e>2.2945</e>
  </location>
</coordinates>
```

The response payload will look like this:

```
{
  "coordinates": {
    "location": [
      {
        "name": "Bermuda Triangle",
        "n": 25.0000,
        "w": 71.0000
      },
      {
        "name": "Eiffel Tower",
        "n": 48.8582,
        "e": 2.2945
      }
    ]
  }
}
```

Note that we have used the Property mediator to mark the outgoing payload to be formatted as JSON. For more information about the Property Mediator, see the [Property Mediator](#) page on WSO2 ESB documentation.

```
<property name="messageType" value="application/json" scope="axis2" />
```

Similarly if the response message needs to be transformed, set the messageType property in the outSequence.

```

<api name="admin--TOJSON" context="/tojson" version="1.0" version-type="url">
    <resource methods="POST GET DELETE OPTIONS PUT" url-mapping="/*">
        <inSequence>
            <property name="POST_TO_URI" value="true" scope="axis2"/>
            <filter source="$ctx:AM_KEY_TYPE" regex="PRODUCTION">
                <then>
                    <send>
                        <endpoint name="admin--Test_APIproductionEndpoint_0">
                            <http
uri-template="http://localhost:9767/services/StudentService">
                                <timeout>
                                    <duration>30000</duration>
                                    <responseAction>fault</responseAction>
                                </timeout>
                                <suspendOnFailure>
                                    <errorCodes>-1</errorCodes>
                                    <initialDuration>0</initialDuration>
                                    <progressionFactor>1.0</progressionFactor>
                                    <maximumDuration>0</maximumDuration>
                                </suspendOnFailure>
                                <markForSuspension>
                                    <errorCodes>-1</errorCodes>
                                </markForSuspension>
                            </http>
                        </endpoint>
                    </send>
                </then>
                <else>
                    <sequence key="_sandbox_key_error_" />
                </else>
            </filter>
        </inSequence>
        <outSequence>
            <property name="messageType" value="application/json" scope="axis2"/>
            <send/>
        </outSequence>
    </resource>
    <handlers>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
            <property name="id" value="A"/>
            <property name="policyKey"
value="gov:/apimgt/applicationdata/tiers.xml"/>
        </handler>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler"/>
        <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler"/>
        <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>
    </handlers>
</api>

```

Working with Security

This section covers the following topics:

- Passing Enduser Attributes to the Backend Using JWT
- Encrypting Passwords
- Maintaining Logins and passwords
- Saving Access Tokens in Separate Tables
- Configuring WSO2 Identity Server as the Key Manager
- Configuring Transport Level Security
- Enabling the Java Security Manager

Passing Enduser Attributes to the Backend Using JWT

JSON Web Token (JWT) is used to represent claims that are transferred between two parties such as the enduser and the backend.

A claim is an attribute of the user that is mapped to the underlying user store. It is encoded as a JavaScript Object Notation (JSON) object that is used as the payload of a JSON Web Signature (JWS) structure, or as the plain text of a JSON Web Encryption (JWE) structure. This enables claims to be digitally signed.

A set of claims is called a dialect (e.g., <http://wso2.org/claims>). The general format of a JWT is {token info}.{claims list}.{signature}. The API implementation uses information such as logging, content filtering and authentication/authorization that is stored in this token. The token is Base64-encoded and sent to the API implementation in a HTTP header variable. The JWT is self-contained and is divided into three parts as the header, the payload and the signature. For more information on JWT, see <http://openid.net/specs/draft-jones-json-web-token-07.html#anchor3> .

To authenticate endusers, the API Manager passes attributes of the API invoker to the backend API implementation using JWT. In most production deployments, service calls go through the API Manager or a proxy service. If you enable JWT generation in the API Manager, each API request will carry a JWT to the back-end service. When the request goes through the API manager, the JWT is appended as a transport header to the outgoing message. The back-end service fetches the JWT and retrieves the required information about the user, application, or token.

An example of a JWT is given below:

```
{
  "typ": "JWT",
  "alg": "NONE"
} {
  "iss": "wso2.org/products/am",
  "exp": 1345183492181,
  "http://wso2.org/claims/subscriber": "admin",
  "http://wso2.org/claims/applicationname": "app2",
  "http://wso2.org/claims/apicontext": "/placeFinder",
  "http://wso2.org/claims/version": "1.0.0",
  "http://wso2.org/claims/tier": "Silver",
  "http://wso2.org/claims/enduser": "sumedha"
}
```

The above token contains,

- Token expiration time ("exp")
- Subscriber to the API, usually the app developer ("http://wso2.org/claims/subscriber")
- Application through which API invocation is done ("http://wso2.org/claims/applicationname")
- Context of the API ("http://wso2.org/claims/apicontext")
- API version ("http://wso2.org/claims/version")
- Tier/price band for the subscription ("http://wso2.org/claims/tier")

- Enduser of the app who's action invoked the API ("<http://wso2.org/claims/enduser>")

Let's see how to enable and pass information in the JWT or completely alter the JWT generation logic in the API Manager:

- [Configuring JWT](#)
- [Customize the JWT generation](#)

Configuring JWT

Before passing enduser attributes, you enable and configure the JWT implementation in the `<APIM_HOME>/repository/conf/api-manager.xml` file. The relevant elements are described below. If you do not configure these elements, they take their default values.

Element	Description
<code><EnableTokenGeneration></code>	Set this value to <code>true</code> to enable JWT. Note that if you publish APIs before J
<code><SecurityContextHeader/></code>	The name of the HTTP header to which the JWT is attached.
<code><ClaimsRetrieverImplClass/></code>	<p>By default, the <code><ClaimsRetrieverImplClass></code> parameter is commented out. To use a custom class to generate the JWT token:</p> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <code><ClaimsRetrieverImplClass>org.wso2.carbon.apimgt.impl.token.Claim</code> </div> <p>By default, the following are encoded to the JWT:</p> <ul style="list-style-type: none"> • subscriber name • application name • API context • API version • authorised resource owner name <p>In addition, you can also write your own class by extending the interface and implementing the following methods of the interface:</p>

Method	Description
<code>void init() throws APIManagementException;</code>	Used to perform initialization tasks. Is equivalent to the <code>init()</code> method of the <code>org.wso2.carbon.apimgt.impl.token.ClaimRetriever</code> interface.
<code>SortedMap<String, String> getClaims(String endUserName) throws APIManagementException;</code>	Returns a sorted map of claims. The key is the attribute name and the value is the corresponding user attribute value. The ordering is defined by the sorted map.
<code>String getDialectURI(String endUserName);</code>	The dialect URI to which the attribute names are mapped. For example, if the <code>getClaims</code> method returns <code>{email, name}</code> , the dialect URIs are <code>http://wso2.org/claims_email</code> , <code>http://wso2.org/claims_name</code> .
	The default implementation (<code>org.wso2.carbon.apimgt.impl.token.ClaimRetriever</code>) returns the user's attributes defined under the dialect URI with the same dialect URI. The order of the claims is specified, no additional claims are added.

<ConsumerDialectURI/>	<p>The dialect URI under which the user's claims are be looked for. Only works above.</p> <p>The JWT token contains all claims define in the <ConsumerDialectURI> element. To get a list of users to be included in the JWT, simply uncomment this element <code>claims</code> to the JWT token.</p>
<SignatureAlgorithm/>	<p>The signing algorithm used to sign the JWT. The general format of the JWT specified as the algorithm, signing is turned off and the JWT looks as {token} period at the end.</p> <p>This element can have only two values- the default value, which is SHA256\</p>

Customize the JWT generation

The JWT that is generated by default (see [example above](#)) has predefined attributes that are passed to the backend. These include basic application-specific details, subscription details, and user information that are defined in the JWT generation class that comes with the API Manager by the name `org.wso2.carbon.apimgt.impl.token.JWTGenerator`. If you want to pass additional attributes to the backend with the JWT or completely change the default JWT generation logic, do the following:

1. Write your own custom JWT implementation class by extending the default `JWTGenerator` class. A typical example of implementing your own claim generator is given below. It implements the `populateCustomClaims()` method to generate some custom claims and adds them to the JWT.

```

import org.wso2.carbon.apimgt.impl.APIConstants;
import org.wso2.carbon.apimgt.impl.dto.APIKeyValidationInfoDTO;
import org.wso2.carbon.apimgt.impl.token.JWTGenerator;
import org.wso2.carbon.apimgt.api.*;

import java.util.Map;

public class CustomTokenGenerator extends JWTGenerator {

    public Map populateStandardClaims(APIKeyValidationInfoDTO
keyValidationInfoDTO, String apiContext, String version)
        throws APIManagementException {
        Map claims = super.populateStandardClaims(keyValidationInfoDTO,
apiContext, version);
        boolean isApplicationToken =
keyValidationInfoDTO.getUserType().equalsIgnoreCase(APIConstants.ACCESS_TOKEN_USE
R_TYPE_APPLICATION) ? true : false;
        String dialect = getDialectURI();
        if (claims.get(dialect + "/enduser") != null) {
            if (isApplicationToken) {
                claims.put(dialect + "/enduser", "null");
                claims.put(dialect + "/enduserTenantId", "null");
            } else {
                String enduser = claims.get(dialect + "/enduser");
                if (enduser.endsWith("@carbon.super")) {
                    enduser = enduser.replace("@carbon.super", "");
                    claims.put(dialect + "/enduser", enduser);
                }
            }
        }
        return claims;
    }

    public Map populateCustomClaims(APIKeyValidationInfoDTO keyValidationInfoDTO,
String apiContext, String version, String accessToken)
        throws APIManagementException {
        Long time = System.currentTimeMillis();
        String text = "This is custom JWT";
        Map map = new HashMap();
        map.put("current_timestamp", time.toString());
        map.put("messge" , text);
        return map;
    }
}

```

2. Build your class and add the JAR file to <APIM_HOME>/repository/components/lib directory.
3. Add your class in the <TokenGeneratorImpl> element of the <APIM_HOME>/repository/conf/api-mana ger.xml file.

```

<APIConsumerAuthentication>
  ...
<TokenGeneratorImpl>org.wso2.carbon.test.CustomTokenGenerator</TokenGeneratorImpl>
  ...
</APIConsumerAuthentication>

```

4. Set the `<EnableTokenGeneration>` element to **true** in the `api-manager.xml` file.
5. Restart the server.

Encrypting Passwords

Encrypting passwords provides better security and less vulnerability to security attacks than saving passwords in plain text. It is recommended in a production setup. WSO2 API Manager provides a secure vault implementation that encrypts passwords, stores them in the registry, maps them to aliases and uses the alias instead of the actual passwords in configuration files. At runtime, the API Manager looks up aliases and decrypts the passwords. The secure vault is unable to encrypt the passwords of registry resources at the moment.

The steps below explain how to encrypt passwords in different contexts:

- [Encrypting passwords in configuration files](#)
- [Encrypting secure endpoint passwords](#)

Encrypting passwords in configuration files

1. Shutdown the server if it is already running and open `<APIM_HOME>/repository/conf/security/cipher-tool.properties` file. It contains all the aliases to different server components.
2. Note that the file has several aliases already defined as the alias name and the value where the value is `<file name>//<xpath to the property value to be secured>`, `<true if the XML element starts with a capital letter>`. Uncomment the entries you want to encrypt.

```

transports.https.keystorePass=mgt-transports.xml//transports/transport[@name='http
ps']/parameter[@name='keystorePass'],false
Carbon.Security.KeyStore.Password=carbon.xml//Server/Security/KeyStore/Password,t
rue
Carbon.Security.KeyStore.KeyPassword=carbon.xml//Server/Security/KeyStore/KeyPass
word,true
Carbon.Security.TrustStore.Password=carbon.xml//Server/Security/TrustStore/Passwo
rd,true
UserManager.AdminUser.Password=user-mgt.xml//UserManager/Realm/Configuration/Admi
nUser/Password,true
Datasources.WSO2_CARBON_DB.Configuration.Password=master-datasources.xml//datasou
rces-configuration/datasources/datasource[name='WSO2_CARBON_DB']/definition[@type
='RDBMS']/configuration/password,false
#Datasource.WSO2AM_DB.configuration.password=master-datasources.xml//datasources-
configuration/datasources/datasource[name='WSO2AM_DB']/definition[@type='RDBMS']/
configuration/password,false
#Datasource.WSO2AM_STATS_DB.configuration.password=master-datasources.xml//dataso
urces-configuration/datasources/datasource[name='WSO2AM_STATS_DB']/definition[@ty
pe='RDBMS']/configuration/password,false
#UserStoreManager.Property.ConnectionPassword=user-mgt.xml//UserManager/Realm/Use
rStoreManager/Property[@name='ConnectionPassword'],true
#UserStoreManager.Property.password=user-mgt.xml//UserManager/Realm/UserStoreMana
ger/Property[@name='password'],true
#AuthManager.Password=api-manager.xml//APIManager/AuthManager/Password,true

...

```

3. Open `<APIM_HOME>/repository/conf/security/cipher-text.properties` file, which maps the default alias to their plain text passwords in square brackets. Uncomment the ones you want.

```
Carbon.Security.KeyStore.Password=[wso2carbon]
```

4. If you are on Linux or a Unix-based operating system, run the cipher tool available [here](#). If you are on Windows, get the cipher tool from the `<APIM_HOME>/bin` folder. Due to a known issue in the 1.8.0 release on Linux, we provide the `.sh` file separately. This script reads the aliases, encrypts their plain-text passwords, and stores them in the secure vault. If you are using the default primary keystore, give `wso2carbon` as its password when prompted.

 **Tip:** By default, the primary keystore, which is `<APIM_HOME>/repository/resources/securit
y/wso2carbon.jks` is used as the secure vault. If you want to use another keystore or a custom
callback class to handle decryption, modify the `<APIM_HOME>/repository/conf/security/se
cret-conf.properties` file as described in [WSO2 Carbon Secure Vault](#) in the WSO2 Carbon
documentation.

```
On Windows: ciphertool.bat -Dconfigure
On Linux: sh ciphertool.sh -Dconfigure
```

5. Note that the configuration files are automatically updated with the relevant password alias after running the cipher tool. For example, as the `Carbon.Security.KeyStore.Password` property is uncommented in this example, after you run the cipher tool, the plain-text password in `<APIM_HOME>/repository/conf/c`

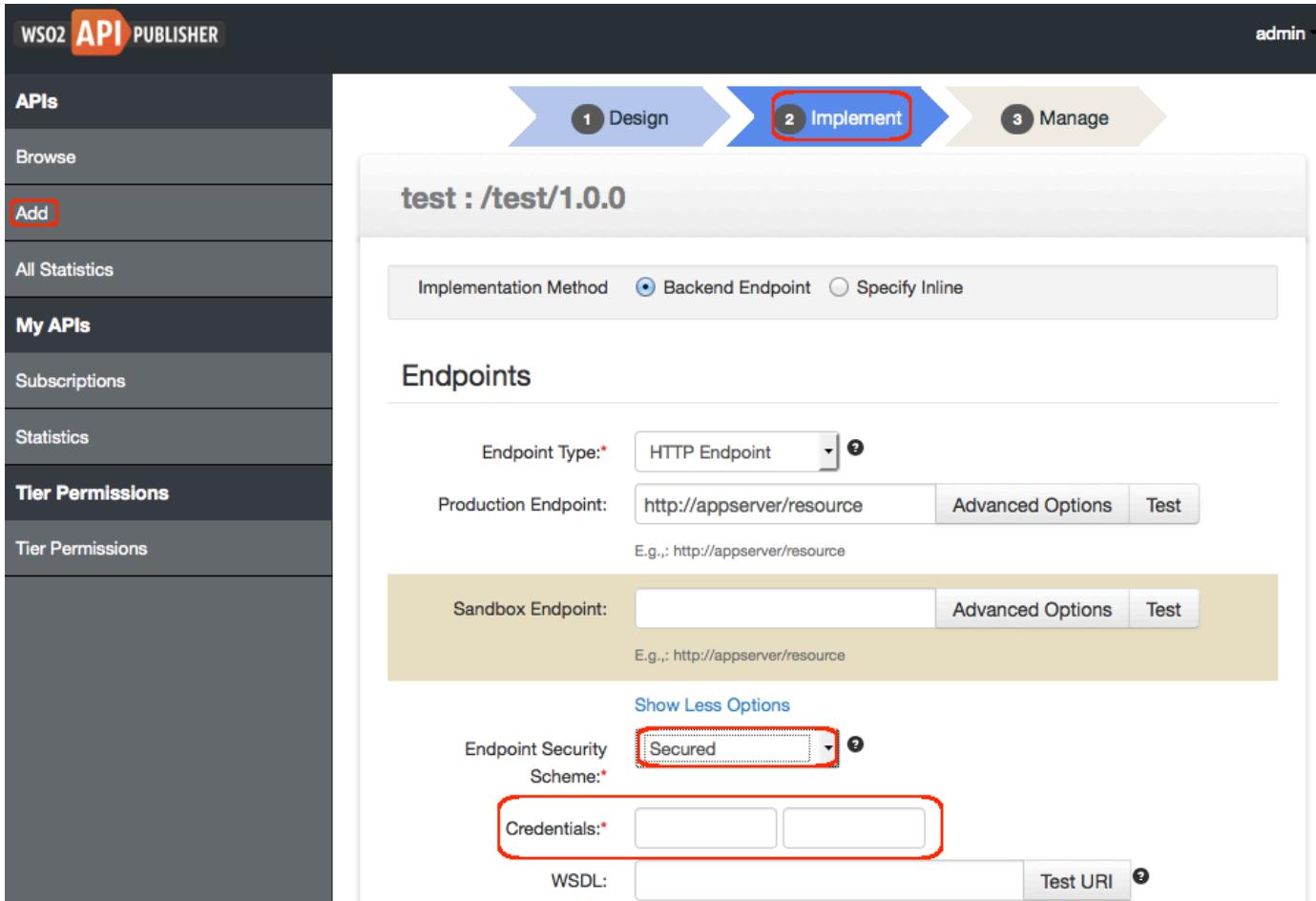
arbon.xml file will be replaced by the alias as follows.

```
<KeyStore>
...
    <!-- Keystore password-->
    <Password
svns:secretAlias="Carbon.Security.KeyStore.Password">password</Password>
...
</KeyStore>
```

 **Tip:** As you encrypted the primary keystore's password in this example, you are prompted to enter the primary keystore password every time you start the server.

Encrypting secure endpoint passwords

When creating an API using the API Publisher, you specify the endpoint of its backend implementation in the **Implement** tab. If you select the endpoint as secured, you are prompted to give credentials in plain-text.



The screenshot shows the WSO2 API Publisher interface. The left sidebar has a dark theme with the following navigation items: APIs (selected), Browse, Add, All Statistics, My APIs, Subscriptions, Statistics, Tier Permissions, and Tier Permissions. The main area has a light theme. At the top, there are three tabs: 1 Design, 2 Implement (which is highlighted with a red box), and 3 Manage. Below these tabs, the title is 'test : /test/1.0.0'. Under the title, there is a 'Implementation Method' section with radio buttons for 'Backend Endpoint' (selected) and 'Specify Inline'. The main content area is titled 'Endpoints'. It shows fields for 'Endpoint Type' (set to 'HTTP Endpoint'), 'Production Endpoint' (set to 'http://appserver/resource'), 'Sandbox Endpoint' (empty), and 'Endpoint Security Scheme' (set to 'Secured'). The 'Secured' dropdown and the 'Credentials' input fields (which are part of a multi-field input group) are both highlighted with a red box. Below the 'Endpoint Security Scheme' field is a 'Show Less Options' link. At the bottom of the form are 'WSDL' and 'Test URI' fields.

The steps below show how to secure the endpoint's password that is given in plain-text in the UI.

1. Shut down the server if it is already running and set the element `<EnableSecureVault>` in `<APIM_HOME>/repository/conf/api-manager.xml` to true. By default, the system stores passwords in configuration files in plain text because this value is set to false.
2. Define synapse property in the `synapse.properties` file as follows: `synapse.xpath.func.extensions=or`

g.wso2.carbon.mediation.security.vault.xpath.SecureVaultLookupXPathFunctionProvider.

3. Run the cipher tool available in <APIM_HOME>/bin. If on windows, the file is `ciphertool.bat`. If you are using the default keystore, give `wso2carbon` as the primary keystore password when prompted.

```
sh ciphertool.sh -Dconfigure
```

 **Tip:** See [Configuring Transport Level Security](#) for information on configuring cipher at the Tomcat level.

Maintaining Logins and passwords

Changing the super admin password

See [How do I change the default admin password and what files should I edit after changing it?](#)

 **Do you have any special characters in passwords?**

If you specify passwords inside XML files, take care when giving special characters in the user names and passwords. According to XML specification (<http://www.w3.org/TR/xml/>), some special characters can disrupt the configuration. For example, the ampersand character (&) must not appear in the literal form in XML files. It can cause a Java Null Pointer exception. You must wrap it with CDATA (http://www.w3schools.com/xml/xml_cdata.asp) as shown below or remove the character:

```
<Password>
<! [ CDATA[ xnvYh?@VHAkc?qZ%Jv855&A4a , %M8B@h ] >
</Password>
```

Recovering a password

See [How can I recover the admin password used to log in to the management console?](#)

Setting up primary and secondary logins

In a standalone deployment of the API Manager instance, users of the API Store can have a secondary login name in addition to the primary login name. This gives the user flexibility to provide either an email or a user name to log in. You can configure the API Store to treat both login names as belonging to a single user. Users can invoke APIs with the same Accesstoken without having to create a new one for the secondary login.

You can configure this capability using the steps below.

1. Configure user login under the `<OAuth>` element in `<APIM_HOME>/repository/conf/identity.xml` file.
 - a. Mention your primary and secondary login names. Set the `primary` attribute of the primary login to `true` and the `primary` attribute of the secondary login to `false`.
 - b. Primary login doesn't have a `ClaimUri`. Leave this field empty.
 - c. Provide the correct `ClaimUri` value for the secondary login

An example is given below:

```

<OAuth>
    .... .
    <LoginConfig>
        <UserIdLogin primary="true">
            <ClaimUri></ClaimUri>
        </UserIdLogin>
        <EmailLogin primary="false">
            <ClaimUri>http://wso2.org/claims/emailaddress</ClaimUri>
        </EmailLogin>
    </LoginConfig>
</OAuth>

```

2. In the API Store of a distributed setup, the `serverURL` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file should point to the key manager instance's service endpoint. This allows users to connect to the key manager's user store to perform any operations related to API Store such as login, access token generation etc. For example,

```

<AuthManager>
    <!--Server URL of the Authentication service -->
    <ServerURL>https://localhost:9444/services/</ServerURL>

    <!-- Admin username for the Authentication manager. -->
    <Username>admin</Username>

    <!-- Admin password for the Authentication manager.-->
    <Password>admin</Password>
</AuthManager>

```

 **Tip:** In a distributed setup, the API Store's user store needs to point to the key manager user store.

 **Tip:** Be sure to keep the secondary login name unique to each user.

 **Tip:** Did you change the super admin user's password? See

Saving Access Tokens in Separate Tables

You can configure the API Manager instances to store access tokens in different tables according to their user store domains. This is referred to as **user token partitioning** and it ensures better security when there are multiple user stores configured in the system. To configure user stores other than the default one, see [Configuring Secondary User Stores](#).

The following topics explain how to enable user token partitioning:

- Enabling assertions
- Storing keys in different tables

Enabling assertions

You use assertions to embed parameters into tokens and generate a strong access token. You can also use these parameters later for other processing. At the moment, the API Manager only supports UserName as an assertion.

By default, assertions are set to `false` in `<APIM_HOME>/repository/conf/identity.xml`. To enable it, set the `<UserName>` element to `true`. You can add a user name to an access token when generating the key, and verify it by encoding the retrieved access token with Base64.

<APIM_HOME>/repository/conf/identity.xml

```
<EnableAssertions>
    <UserName>true</UserName>
</EnableAssertions>
```

Storing keys in different tables

1. If the <UserName> assertion is enabled, set the <EnableAccessTokenPartitioning> element in <APIM_HOME>/repository/conf/identity.xml file to true. It determines whether you want to store the keys in different tables or not.

```
<EnableAccessTokenPartitioning>true</EnableAccessTokenPartitioning>
```

2. Set the user store domain names and mappings to new table names. For example,
 - if userId = foo.com/admin where 'foo.com' is the user store domain name, then a 'mapping:domain' combo can be defined as 'A:foo.com'
 - 'A' is the mapping for the table that stores tokens relevant to users coming from 'foo.com' user store

In this case, the actual table name is `IDN_OAUTH2_ACCESS_TOKEN_A`. We use a mapping simply to prevent any issues caused by lengthy table names when lengthy domain names are used. You must manually create the tables you are going to use to store the access tokens in each user store (i.e., manually create the tables `IDN_OAUTH2_ACCESS_TOKEN_A` and `IDN_OAUTH2_ACCESS_TOKEN_B` according to the following defined domain mapping). This table structure is similar to the `IDN_OAUTH2_ACCESS_TOKEN` table defined in the api-manager dbscript, which is inside the <APIM_HOME>/dbscripts/apimgt directory.

You can provide multiple mappings separated by commas as follows. Note that the domain names need to be specified in upper case.

```
<AccessTokenPartitioningDomains>A:FOO.COM,
B:BAR.COM</AccessTokenPartitioningDomains>
```

3. According to the information given above, change the <OAuth> element in the <APIM_HOME>/repository/conf/identity.xml file as shown in the following example:

<APIM_HOME>/repository/conf/identity.xml

```

<!-- Assertions can be used to embed parameters into access token.-->
<EnableAssertions>
    <UserName>true</UserName>
</EnableAssertions>

<!-- This should be set to true when using multiple user stores and keys should
    saved into different tables according to the user store. By default all the
    application keys are saved in to the same table. UserName Assertion should be
    'true' to use this.-->
<AccessTokenPartitioning>
    <EnableAccessTokenPartitioning>true</EnableAccessTokenPartitioning>
    <!-- user store domain names and mappings to new table names. eg: if you
        provide 'A:foo.com', foo.com should be the user store domain
        name and 'A' represent the relavant mapping of token storing table, i.e.
        tokens relevant to the users comming from foo.com user store
        will be added to a table called IDN_OAUTH2_ACCESS_TOKEN_A. -->
    <AccessTokenPartitioningDomains>A:foo.com,
    B:bar.com</AccessTokenPartitioningDomains>
</AccessTokenPartitioning>

```

Configuring WSO2 Identity Server as the Key Manager

If your production environment already has an instance of WSO2 Identity Server, you can use it as the Key Manager rather than setting up an additional WSO2 API Manager instance to work as the Key Manager. If you set up the Identity Server, you can get the added advantage of being able to use authentication/authorization features specific to the Identity Server.

For setup instructions, see the [Clustering Guide](#).

Configuring Transport Level Security

The transport level security protocol of the Tomcat server is configured in the <PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml file. Note that the `sslProtocol` attribute is set to TLS (Transport Layer Security) by default.

See the following topics for configuration:

- [Disabling SSL version 3](#)
- [Disabling the weak ciphers](#)

Disabling SSL version 3

i It is necessary to disable SSL version 3 in WSO2 products because of a bug ([Poodle Attack](#)) in the SSL version 3 protocol that could expose critical data encrypted between clients and servers. The Poodle Attack makes the system vulnerable by telling the client that the server does not support the more secure TLS protocol. This forces the server to connect via SSL 3.0. You can mitigate the effect of this bug by disabling SSL version 3 protocol in your server.

Follow the steps below to disable SSL 3.0 support.

1. Make a backup of the <PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml file and stop the server.
2. Find the connector configuration that is corresponding to TLS (usually, this connector has the port set to 9443 and the `sslProtocol` as TLS).
 - If you are using JDK 1.6, remove the `sslProtocol="TLS"` attribute from the configuration and

replace it with `sslEnabledProtocols="TLSv1"` as shown below.

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

- If you are using JDK 1.7, remove the `sslProtocol="TLS"` attribute from the above configuration and replace it with `sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"` as shown below.

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

3. Start the server.

To test if SSL version 3 is disabled:

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

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

3. The output of the command before and after disabling SSL version 3 is shown below.

Before SSL version 3 is disabled:

```
Supported versions: SSLv3 TLSv1.0
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
SSLv3
  RSA_EXPORT_WITH_RC4_40_MD5
  RSA_WITH_RC4_128_MD5
  RSA_WITH_RC4_128_SHA
  RSA_EXPORT_WITH_DES40_CBC_SHA
  RSA_WITH_DES_CBC_SHA
  RSA_WITH_3DES_EDE_CBC_SHA
  DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
  DHE_RSA_WITH_DES_CBC_SHA
  DHE_RSA_WITH_3DES_EDE_CBC_SHA
  RSA_WITH_AES_128_CBC_SHA
  DHE_RSA_WITH_AES_128_CBC_SHA
  RSA_WITH_AES_256_CBC_SHA
  DHE_RSA_WITH_AES_256_CBC_SHA
(TLSv1.0: idem)
```

After SSL version 3 is disabled:

```

Supported versions: TLSv1.0
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
  TLSv1.0
    RSA_EXPORT_WITH_RC4_40_MD5
    RSA_WITH_RC4_128_MD5
    RSA_WITH_RC4_128_SHA
    RSA_EXPORT_WITH_DES40_CBC_SHA
    RSA_WITH_DES_CBC_SHA
    RSA_WITH_3DES_EDE_CBC_SHA
    DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
    DHE_RSA_WITH_DES_CBC_SHA
    DHE_RSA_WITH_3DES_EDE_CBC_SHA
    RSA_WITH_AES_128_CBC_SHA
    DHE_RSA_WITH_AES_128_CBC_SHA
    RSA_WITH_AES_256_CBC_SHA
    DHE_RSA_WITH_AES_256_CBC_SHA

```

 The TLSv1 protocol used in the `catalina-server.xml` file can sometimes create the following security alert when the client is run:

```
BEAST status: vulnerable
```

However, it is still recommended to use the TLSv1 protocol as it is possible to overcome this vulnerability from the client side.

Disabling the weak ciphers

A cipher is an algorithm for performing encryption or decryption. When the `sslProtocol` is set to `TLS`, only the `TLS` and default ciphers are enabled. However, the strength of the ciphers will not be considered when they are enabled. Therefore, to disable the weak ciphers, you enter only the ciphers that you want the server to support in a comma-separated list in the `ciphers` attribute. Also, if you do not add this cipher attribute or keep it blank, all SSL ciphers by JSSE will be supported by your server. This will enable the weak ciphers.

1. Make a backup of the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file and stop the server (same as for [disabling SSL version 3](#)).
2. Add the `cipher` attribute to the existing configuration in the `catalina-server.xml` file by adding the list of ciphers that you want your server to support as follows: `ciphers="<cipher-name>,<cipher-name>".`

```

ciphers="SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,TLS_RSA_WITH_AES_128_C
BC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_3D
ES_EDE_CBC_SHA,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA"

```

3. Start the server.

Changing the pass-through transport configs

If you have enabled the pass-through transport, do the following:

1. Stop the server.
2. Open the <PRODUCT_HOME>/repository/conf/axis2/axis2.xml file and add the following under the <transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener"> element:
 - If you are using JDK 1.6, add the parameter given below:

```
<transportReceiver name="passthru-https"
class="org.wso2.carbon.transport.passthru.PassThroughHttpSSLListener">
<parameter name="HttpsProtocols">TLSv1</parameter>
.....
</transportReceiver>
```

- If you are using JDK 1.7, add the parameter given below:

```
<transportReceiver name="passthru-https"
class="org.wso2.carbon.transport.passthru.PassThroughHttpSSLListener">
<parameter name="HttpsProtocols">TLSv1,TLSv1.1,TLSv1.2</parameter>
.....
</transportReceiver>
```

3. Start the server.
4. Test the pass-through transport using the following command with the corresponding port:

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

Enabling the Java Security Manager

The Java Security Manager is used to define various security policies that prevent untrusted code from manipulating your system. Enabling the Java Security Manager for WSO2 products activates the Java permissions that are in the <PRODUCT_HOME>/repository/conf/sec.policy file. You modify this file to change the Java security permissions as required.

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

Before you begin, ensure that you have Java 1.6 installed.

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

```
keytool -genkey -alias signFiles -keyalg RSA -keystore signkeystore.jks -validity
3650 -dname "CN=Sanjeewa,OU=Engineering, O=WSO2, L=Colombo, ST=Western,
C=LK" Enter keystore password:
Re-enter new password:
Enter key password for
(RTURN if same as keystore password)
```

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

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

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

$ keytool -import -alias signFiles -file sign-cert.cer -keystore
repository/resources/security/wso2carbon.jks
Enter keystore password:
Owner: CN=Sanjeewa, OU=Engineering, O=WSO2, L=Colombo, ST=Western, C=LK
Issuer: CN=Sanjeewa, OU=Engineering, O=WSO2, L=Colombo, ST=Western, C=LK
Serial number: 5486f3b0
Valid from: Tue Dec 09 18:35:52 IST 2014 until: Fri Dec 06 18:35:52 IST 2024
Certificate fingerprints:
MD5: 54:13:FD:06:6F:C9:A6:BC:EE:DF:73:A9:88:CC:02:EC
SHA1: AE:37:2A:9E:66:86:12:68:28:88:12:A0:85:50:B1:D1:21:BD:49:52
Signature algorithm name: SHA1withRSA
Version: 3
Trust this certificate? [no]: yes
Certificate was added to keystore
```

4. Prepare the scripts to sign the JARs and grant them the required permission. For example, the `signJar.sh` script given below can be used to sign each JAR file separately or you can use the `signJars.sh` script, which runs a loop to read all JARs and sign them.

signJar.sh script

```
#!/bin/bash
set -e
jarfile=$1
keystore_file="signkeystore.jks"
keystore_keyalias='signFiles'
keystore_storepass='wso2123'
keystore_keypass='wso2123'
signjar="$JAVA_HOME/bin/jarsigner -sigalg MD5withRSA -digestalg SHA1
-keystore $keystore_file -storepass $keystore_storepass -keypass
$keystore_keypass"
verifyjar="$JAVA_HOME/bin/jarsigner -keystore $keystore_file -verify"
echo "Signing $jarfile"
$signjar $jarfile $keystore_keyalias
echo "Verifying $jarfile"
$verifyjar $jarfile
# Check whether the verification is successful.
if [ $? -eq 1 ]
then
    echo "Verification failed for $jarfile"
fi
```

signJars.sh script

```
#!/bin/bash
if [[ ! -d $1 ]]; then
    echo "Please specify a target directory"
    exit 1
fi
for jarfile in `find . -type f -iname \*.jar`
do
    ./signJar.sh $jarfile
done
```

5. Execute the following commands to sign the JARs in your product:

```
./signJars.sh /HOME/user/<product-pack>
```

 Every time you add an external JAR to the WSO2 product, sign them manually using the above instructions for the Java Security Manager to be effective. You add external JARs to the server when extending the product, applying patches etc.

6. Open the startup script in the <PRODUCT_HOME>/bin folder. For Linux, it is `wso2server.sh`.

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

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

8. Create a `sec.policy` file with the required security policies in the <PRODUCT_HOME>/repository/conf folder and start the server. Starting the server makes the Java permissions defined in the `sec.policy` file to take effect.

An example of a `sec.policy` file is given below. It includes mostly WSO2 Carbon-level permissions.

```

grant {
    // Allow socket connections for any host
    permission java.net.SocketPermission "*:1-65535", "connect,resolve";

    // Allow to read all properties. Use -Ddenied.system.properties in wso2server.sh
    to restrict properties
    permission java.util.PropertyPermission "*", "read";

    permission java.lang.RuntimePermission "getClassLoader";

    // CarbonContext APIs require this permission
    permission java.lang.management.ManagementPermission "control";

    // Required by any component reading XMLs. For example:
    org.wso2.carbon.databridge.agent.thrift:4.2.1.
    permission java.lang.RuntimePermission
    "accessClassInPackage.com.sun.xml.internal.bind.v2.runtime.reflect";

    // Required by org.wso2.carbon.ndatasource.core:4.2.0. This is only necessary
    after adding above permission.
    permission java.lang.RuntimePermission
    "accessClassInPackage.com.sun.xml.internal.bind";
};


```

Working with Statistics

The following topics describe how to monitor API invocations and how to collect and summarize statistics in order to monetize API usage.

- [Publishing API Runtime Statistics](#)
- [Integrating with Google Analytics](#)
- [Viewing API Statistics](#)

Publishing API Runtime Statistics

You can set up **WSO2 Business Activity Monitor** (version 2.5.0 is used here) to collect and analyze runtime statistics from the API Manager. To publish data from the API Manager to BAM, the Thrift protocol is used. Information processed in BAM is stored in a database from which the API Publisher retrieves information before displaying in the corresponding UI screens.

By default, `org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataPublisher` is configured to push data events to WSO2 BAM. If you use a product other than WSO2 BAM to collect and analyze runtime statistics, you write a new data publishing agent by extending `APIMgtUsageDataPublisher`. Find the API templates inside `<APIM_HOME>/repository/resources/api_templates`. When writing a new data publishing agent, make sure the data publishing logic you implement has a minimal impact to API invocation.

 **Info:** The datasource and database names used in this guide are just examples. They may vary depending on your configurations.

- [Prerequisites](#)
- [Configure WSO2 API Manager](#)
- [Configure WSO2 BAM](#)
- [Troubleshoot common issues](#)
- [Change the statistics database](#)

Prerequisites

- JDK 1.6.* or 1.7



If you install JDK in Program Files in the Windows environment, avoid the space by using PROGRA~1 when specifying environment variables for JAVA_HOME and PATH. Else, the server throws an exception.

- Cygwin (<http://www.cygwin.com>): Required **only if you are using Windows**. WSO2 BAM analytics framework depends on Apache Hadoop, which requires Cygwin in order to run on Windows. Install at least the basic net (OpenSSH,tcp_wrapper packages) and security related Cygwin packages. After Cygwin installation, update the PATH variable with C:/cygwin/bin and restart BAM.

Configure WSO2 API Manager

1. Do the following changes in <APIM_HOME>/repository/conf/api-manager.xml file:

- Enable API usage tracking by setting the <APIUsageTracking> element to true.
- Because you will apply an offset to the default BAM port later in this guide, you need to apply the same offset to the default Thrift port. To do that, change the port value to 7614 in the <ThriftPort> element of this file. The API Manager will then push the data to BAM through port 7614, using the Thrift protocol.
- Uncomment and set the data source used for getting BAM statistics in <DataSourceName> element.
- Set <BAMServerURL> to tcp://<BAM host IP>:7614/ where <BAM host IP> is the machine IP address. Do not use localhost unless you're in a disconnected mode.



<BAMServerURL> refers to the endpoint to which events will be published from the API Gateway. This endpoint is also known as the event receiver. You can define multiple event receiver groups, each with one or more receivers. A receiver group is defined within curly braces and receiver URLs are delimited by commas.

For example, <BAMServerURL>{tcp://localhost:7612/,tcp://localhost:7613/}, {tcp://localhost:7712/,tcp://localhost:7713/}</BAMServerURL>. This example has two receiver groups defined with two receivers in each group. When a request passes through the API Gateway, an event will be published to one selected receiver in each group.

```

<APIUsageTracking>
    <!-- Enable/Disable the API usage tracker. -->
    <Enabled>true</Enabled>

    <PublisherClass>org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataBridgeDataPublisher</PublisherClass>
        <ThriftPort>7614</ThriftPort>
        <BAMServerURL>tcp://<BAM host IP>:7614/</BAMServerURL>
        <BAMUsername>admin</BAMUsername>
        <BAMPassword>admin</BAMPassword>
        <!-- JNDI name of the data source to be used for getting BAM statistics. This data source should
            be defined in the master-datasources.xml file in conf/datasources
            directory. -->
        <DataSourceName>jdbc/WSO2AM_STATS_DB</DataSourceName>
    </APIUsageTracking>

```



Tip: You can give a comma-separated list of URLs as the <BAMServerURL> to manage failover. If the BAM server in the first URL fails, the request will be routed to the second one.



Tip: Are you working with an API Manager cluster? If so,

- Configure the `<APIUsageTracking>` element as given above in all API Gateway nodes
- Configure the following sub elements under the `<APIUsageTracking>` element in the API Publisher and Store nodes:

```
<DataSourceName>jdbc/WSO2AM_STATS_DB</DataSourceName>
```

- If you use **destination-based usage tracking**, enable the `<APIUsageTracking>` element in the Publisher node and connect to a running BAM instance. This is because the API Manager uses a separate BAM mediator to do destination based usage tracking.

- Specify the datasource definition under the `<datasource>` element in the `<APIM_HOME>/repository/conf/datasources/master-datasources.xml` file. The tables are created automatically when the Hive script runs. You just need to create the schema. The example below connects to a MySQL instance:

⚠ The **WSO2AM_STATS_DB** database is not available in `<BAM_HOME>/repository/database` directory at this point. It is created only after BAM starts up.

```
<datasource>
  <name>WSO2AM_STATS_DB</name>
  <description>The datasource used for getting statistics to API Manager</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_STATS_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>
      <url>jdbc:mysql://localhost:3306/stats_db?autoReconnect=true&amp;lt;/url>
      <username>db_username</username>
      <password>db_password</password>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <maxActive>50</maxActive>
      <maxWait>60000</maxWait>
      <testOnBorrow>true</testOnBorrow>
      <validationQuery>SELECT 1</validationQuery>
      <validationInterval>30000</validationInterval>
    </configuration>
  </definition>
</datasource>
```

- Save the database driver JAR inside both `<AM_HOME>/repository/components/lib` and `<BAM_HOME>/repository/components/lib` folders.

Next, prepare BAM to collect and analyze statistics from the API Manager.

Configure WSO2 BAM

- Download WSO2 BAM 2.5.0 from location: <http://wso2.com/products/business-activity-monitor>.
- Apply an offset of 3 to the default BAM port by editing the `<BAM_HOME>/repository/conf/carbon.xml` file.

```
<Offset>3</Offset>
```

This increments all ports used by the server by 3, which means the BAM server will run on port 9446. Port offset is used to increment the default port by a given value. It avoids possible port conflicts when multiple WSO2 products run in same host.

- Do the following changes in <BAM_HOME>/repository/conf/datasources/bam_datasources.xml file:

- Copy/paste WSO2AM_STATS_DB definition from the API Manager's master-datasources.xml file. You edited it in step 2. WSO2AM_STATS_DB is used to fetch analytical data from the database.
- Uncomment the hostName element in <BAM_HOME>/repository/conf/data-bridge/data-bridge-config.xml file and give the IP address instead of localhost. You can get the IP from the <API M_HOME>/repository/conf/api-manager.xml file's, <BAMServerURL><BAM host IP> element.

```
<thriftDataReceiver>
  <hostName>localhost</hostName>
  <port>7611</port>
  <securePort>7711</securePort>
</thriftDataReceiver>
```

 If you are using the **BAM 2.4.1** or older, replace the port of WSO2BAM_CASSANDRA_DATASOURCE in URL (jdbc:cassandra://localhost:9163/EVENT_KS). Note that localhost is used here; not the machine IP. Cassandra is bound by default on localhost, unless you change the data-bridge/data-bridge-config.xml file.

- Copy the file <APIM_HOME>/statistics/API_Manager_Analytics.tbox to directory <BAM_HOME>/repository/deployment/server/bam-toolbox.

If this folder is not in the BAM installation directory by default, create it. The toolbox describes the information collected, how to analyze the data, as well as the location of the database where the analyzed data is stored.

- Open <BAM_HOME>/repository/conf/etc/hector-config.xml file and change the port to **localhost:9163**. You must add the other nodes too when configuring a clustered setup.

```
<Nodes>localhost:9163</Nodes>
```

 This step is not needed if you are using **WSO2 BAM 2.5.0**.

- Restart BAM server by running <BAM_HOME>/bin/wso2server.[sh/bat].

Troubleshoot common issues

Given below is how to do troubleshoot some common issues users come across:

- Do you get an out of memory issue?
See the [performance tuning guide](#) for recommendations to tune the server for optimal performance.
- Do you get an exception as **unable to connect to server** Cassandra?
Check if you changed the Cassandra port according to the port offset applied to the default BAM port. See [Step 3](#) under configuring BAM section.
- Do you get a **connection refused exception** on the BAM console?
This happens when you execute Hive scripts prior to changing the default port. Add the following line at the beginning of the Hive scripts and rerun: `drop table <hive_cassandra_table_name>;` You can find the Hive scripts deployed with the toolbox file, which is inside <BAM_HOME>/repository/deployment/server/bam-toolbox folder. For information, see [Editing an Analytic Script](#) in WSO2 BAM documentation.

Change the statistics database

To use a different database than the default H2 for statistical publishing, you must change the properties of the datasource element, and additionally delete some metadata tables created by previous executions of the Hive script, if there are any.

To delete the metadata tables,

1. Log in to BAM management console and select **Add** in **Analytics** menu.
2. Go to the Script Editor in the window that opens.
3. Execute the following script.

```
drop TABLE APIRequestData;
drop TABLE APIRequestSummaryData;
drop TABLE APIVersionUsageSummaryData;
drop TABLE APIResourcePathUsageSummaryData;
drop TABLE APIResponseData;
drop TABLE APIResponseSummaryData;
drop TABLE APIFaultData;
drop TABLE APIFaultSummaryData;
drop TABLE APIDestinationData;
drop TABLE APIDestinationDataSummaryData;
```

4. If there are previous executions of the Hive scripts, manually execute them again by going to **Main > Analytics > List** in the management console of BAM. Alternatively, you can wait until the periodical execution time occurs.

After configuring WSO2 BAM to render and produce statistics of APIs hosted and managed in the API Manager, you can view them through various statistical dashboards in the API Publisher, depending on your permission levels. For information, see [Viewing API Statistics](#).

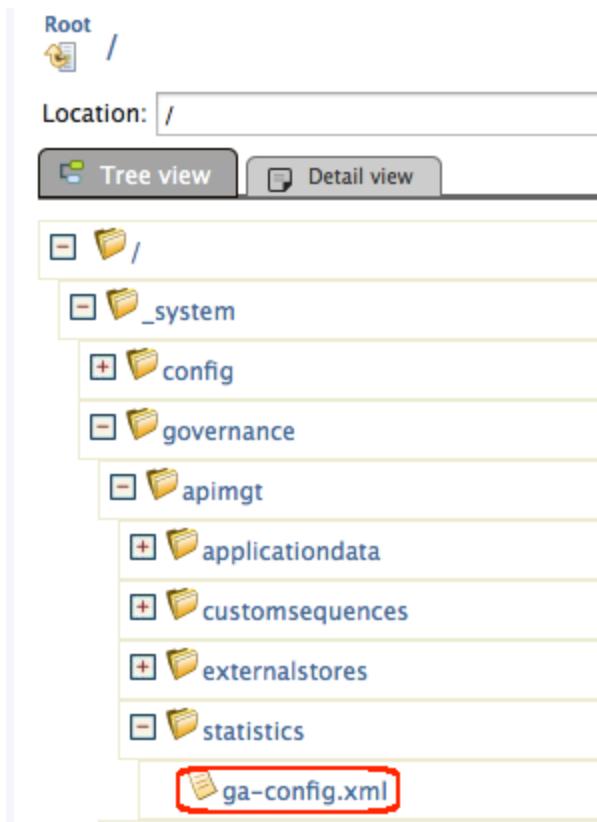
You can configure the API Manager to track runtime statistics of API invocations through Google Analytics (<http://www.google.com/analytics>). Google Analytics is a service that allows you to track visits to a website and generate detailed statistics on them.

This guide explains how to setup API Manager in order to feed runtime statistics to Google analytics for summarization and display.

1. Setup a Google Analytics account if not subscribed already and receive a Tracking ID, which is of the format "UA-XXXXXXX-X". A Tracking ID is issued at the time an account is created with Google Analytics.
2. Log in to the API Manager management console (<https://localhost:9443/carbon>) using admin/admin credentials and go to **Main > Resources > Browse** menu.



3. Navigate to `/_system/governance/apimgt/statistics/ga-config.xml` file.



4. Change the <Enabled> element to true, set your tracking ID in <TrackingID> element and **Save**.

The screenshot shows the 'Content' editor with the 'Edit as text' tab selected. The XML code is as follows:

```

<!--Google Analytics publisher configuration. Create Google Analytics account
and obtain a Tracking ID. Refer http://support.google.com/analytics/bin/answer.py?hl=en&
answer=1009694 -->
<GoogleAnalyticsTracking>
    <!--Enable/Disable Google Analytics Tracking -->
    <Enabled>true</Enabled>

    <!-- Google Analytics Tracking ID -->
    <TrackingID>UA-XXXXXXXX-X</TrackingID>

</GoogleAnalyticsTracking>

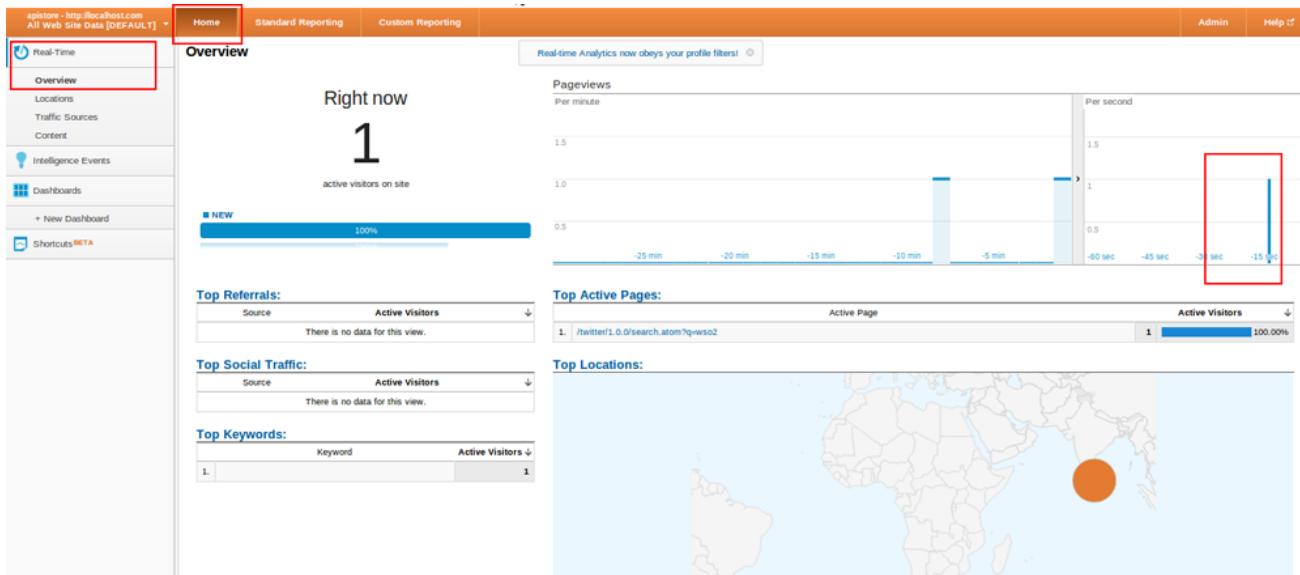
```

Buttons at the bottom are 'Save Content' and 'Cancel'.

5. API Manager is now integrated with Google Analytics. A user who has subscribed to a published API through the API Store should see an icon as **Real-Time** after logging into their Google Analytics account. Click on this icon and select **Overview**.
6. Invoke the above API using the embedded **WSO2 REST Client** (or any third-part rest client such as cURL).

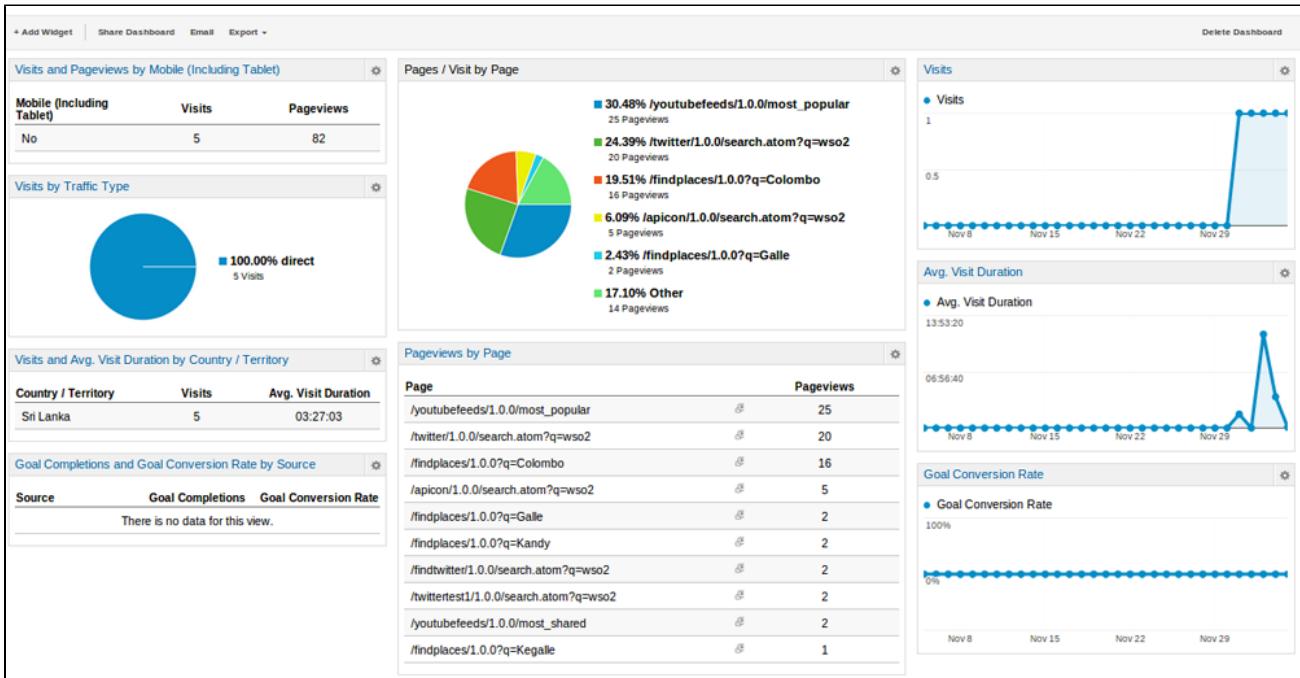
Real-time statistics

7. This is one invocation of the API. Accordingly, Google Analytics graphs and statistics will be displayed at runtime. This example displays the **PageViews** per second graph and 1 user as active.



Report statistics

Google analytics reporting statistics take more than 24 hours from the time of invocation to populate. Shown below is a sample Dashboard with populated statistics.



There are widgets with statistics related to Audience, Traffic, Page Content, Visit Duration etc. You can add any widget of your preference to dashboard.

Viewing API Statistics

API statistics are provided in both API Publisher and API Store Web applications. Apart from the number of subscriptions per API, all other statistical dashboards require that an instance of WSO2 Business Activity Monitor (version 2.3.0 or above) is installed. For instructions to set up BAM, see [Publishing API Runtime Statistics](#). Once BAM is set up, follow the instructions below to view statistics through the API Publisher.

First, trigger some activities via the API Gateway by invoking a few APIs.

 The graphs you see in the API Manager statistical dashboards before setting up the BAM are just samples. They are not based on real runtime statistics of your server.

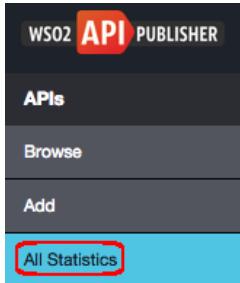
The sections below explain how to access the statistical dashboards:

- [API Publisher statistics](#)
- [API Store statistics](#)

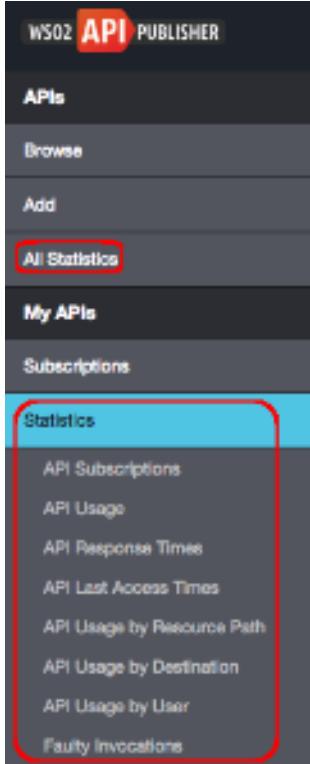
API Publisher statistics

Log in to the API Publisher. If you have API creator and publisher privileges, the statistical menus that you see change as described below:

- If you have permission as publisher, the **All Statistics** menu will be visible in the left panel of the API Publisher.



- If you have permission to create APIs, in addition to the **All Statistics** menu, you also see the **Statistics** menu in the left panel of the API Publisher. The latter shows stats specific to the APIs created by you.



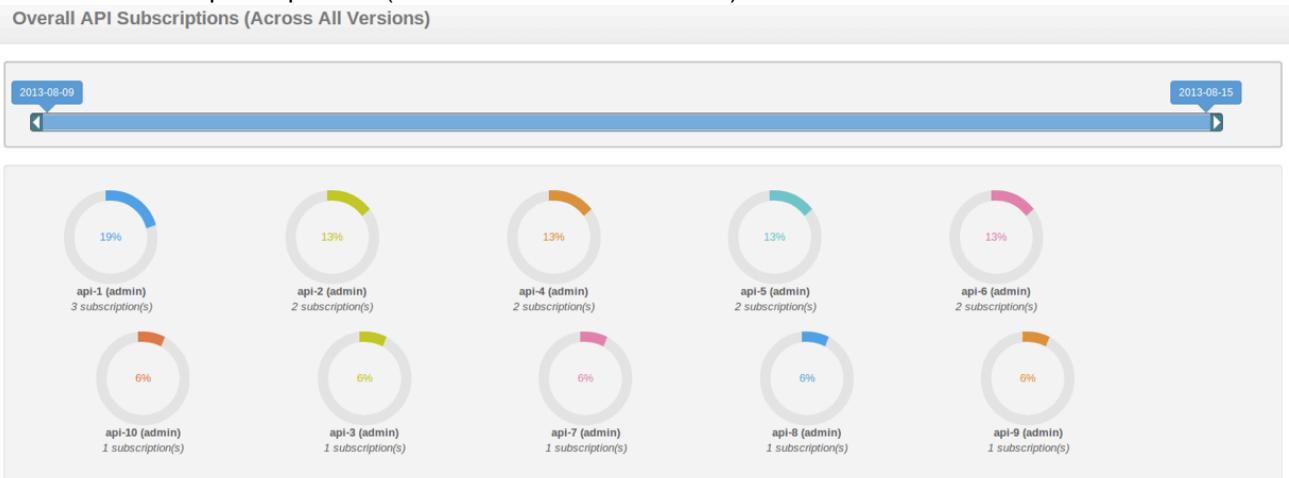
- Anyone who can create and/or publish APIs can view API-level usage and subscription statistics by clicking on a selected API and referring to its **Versions** and **Users** tabs.

The screenshot shows the WSO2 API Publisher interface. On the left, there's a sidebar with 'APIs' and sub-options 'Browse', 'Add', and 'All Statistics'. The main content area shows 'PhoneVerification - 1.0.0' with tabs for 'Overview', 'Lifecycle' (highlighted with a red box), 'Versions' (highlighted with a red box), 'Docs', and 'Users' (highlighted with a red box).

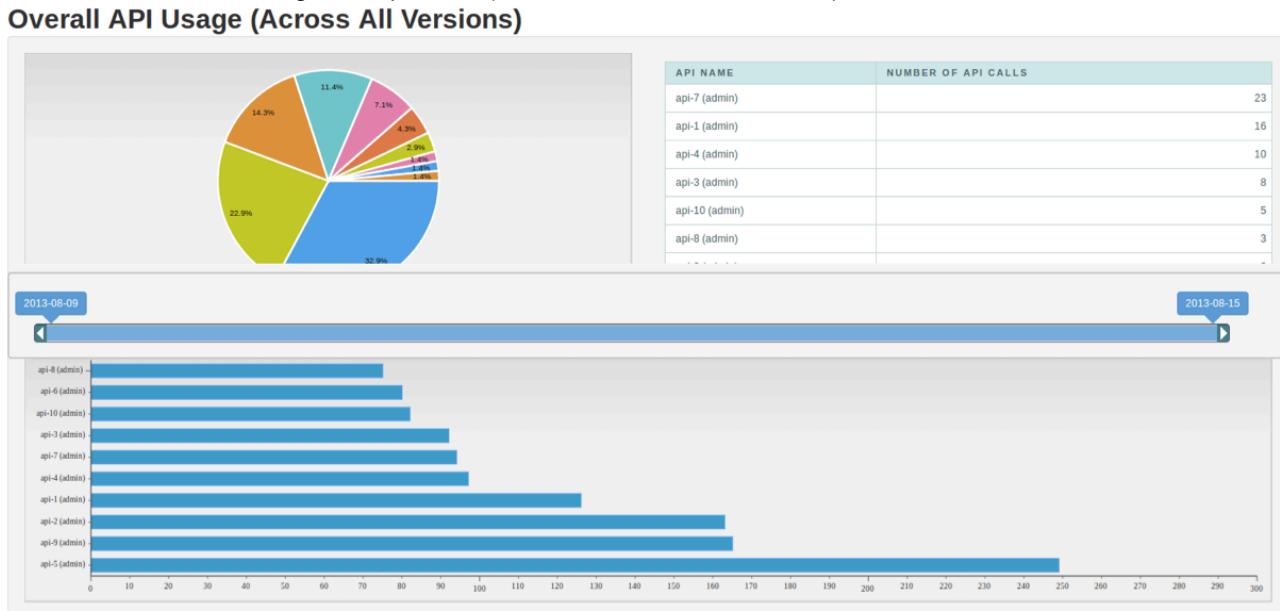
Several examples of usage and performance statistics are given below:

- If you want to see destination-based usage tracking, you must first enable it. See [API Usage by Destination](#).

- Number of subscriptions per API (across all versions of an API)



- Number of API calls being made per API (across all versions of an API)



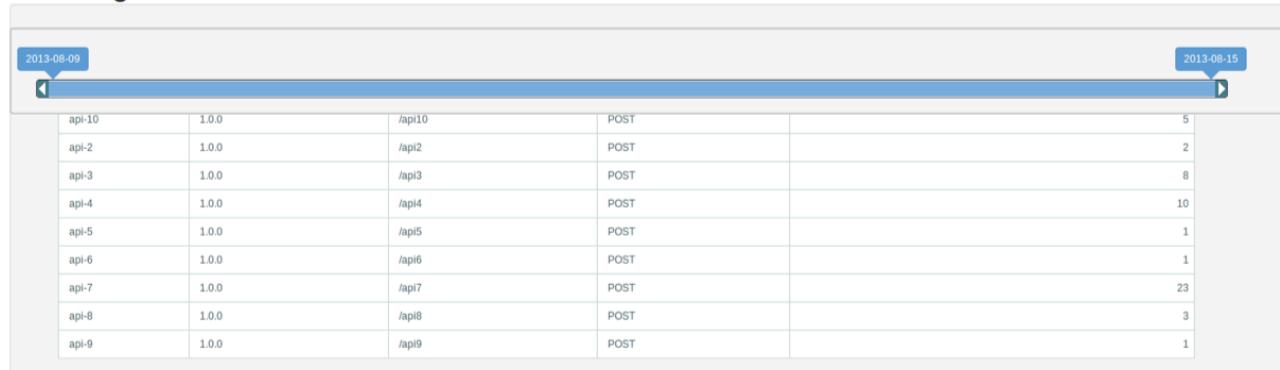
- The subscribers who did the last 10 API invocations and the APIs/versions they invoked

API Last Access Times (Across All Versions / Last 10 invocations)

API	LAST ACCESSED VERSION	SUBSCRIBER	ACCESS TIME
api-5 (admin)	1.0.0	user2	8/15/2013 9:40:00 AM
api-6 (admin)	1.0.0	user2	8/15/2013 9:40:00 AM
api-10 (admin)	1.0.0	user2	8/15/2013 9:36:00 AM
api-7 (admin)	1.0.0	user2	8/15/2013 9:35:00 AM
api-8 (admin)	1.0.0	user2	8/15/2013 9:35:00 AM
api-9 (admin)	1.0.0	user2	8/15/2013 9:35:00 AM
api-1 (admin)	1.0.0	user2	8/15/2013 9:31:00 AM
api-4 (admin)	1.0.0	user2	8/15/2013 9:31:00 AM
api-3 (admin)	1.0.0	user1	8/15/2013 9:26:00 AM
api-2 (admin)	1.0.0	user1	8/15/2013 9:25:00 AM

- Usage of an API and from which resource path (per API version)

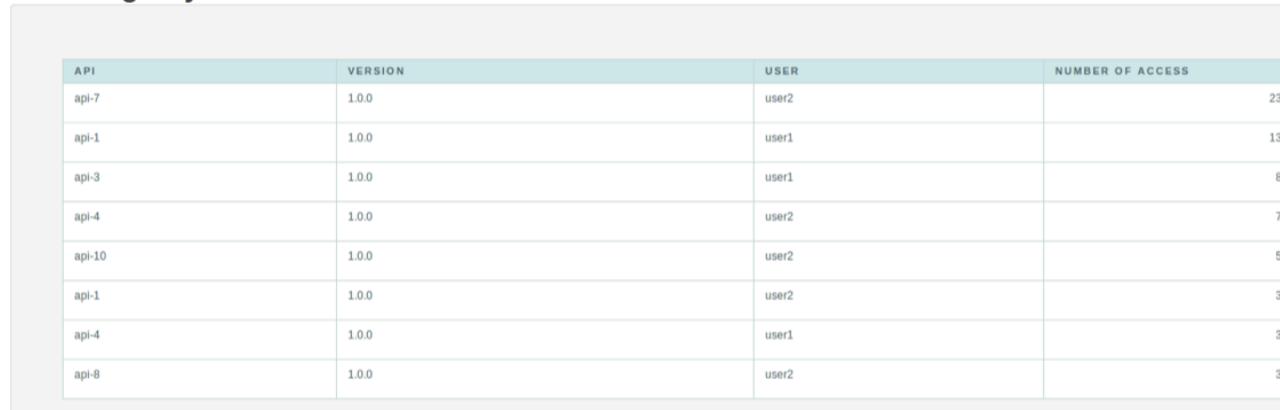
API Usage from Resource Path



API	VERSION	RESOURCE PATH	METHOD	NUMBER OF ACCESS
api-10	1.0.0	/api10	POST	5
api-2	1.0.0	/api2	POST	2
api-3	1.0.0	/api3	POST	8
api-4	1.0.0	/api4	POST	10
api-5	1.0.0	/api5	POST	1
api-6	1.0.0	/api6	POST	1
api-7	1.0.0	/api7	POST	23
api-8	1.0.0	/api8	POST	3
api-9	1.0.0	/api9	POST	1

- Number of times a user has accessed an API

API Usage By User

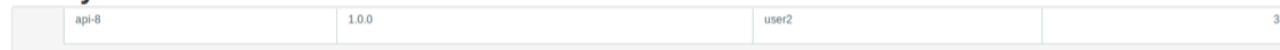


API	VERSION	USER	NUMBER OF ACCESS
api-7	1.0.0	user2	23
api-1	1.0.0	user1	13
api-3	1.0.0	user1	8
api-4	1.0.0	user2	7
api-10	1.0.0	user2	5
api-1	1.0.0	user2	3
api-4	1.0.0	user1	3
api-8	1.0.0	user2	3

- The number of API invocations that failed to reach the endpoint per API per user

In a faulty API invocation, the message is mediated through the `fault` sequence. By default, the API Manager considers an API invocation to be faulty when the backend service is unavailable.

Faulty Invocations



api-8	1.0.0	user2	3
-------	-------	-------	---

- API Usage by Destination

An overview of the requests that leave the API Gateway to destination endpoints. It's particularly useful when the same API can reach different destinations such as load-balanced endpoints. This graph is not enabled by default. You must do it manually as follows:

- Enable usage tracking in the `<APIM_HOME>/repository/conf/api-manager.xml` file:

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

2. When creating the API, enable the graph from the **Implement** tab of the API Publisher UI:

The screenshot shows the 'Implement' tab of the API Publisher for the 'PhoneVerification' API. The 'Implementation Method' is set to 'Backend Endpoint'. The 'Endpoints' section includes fields for 'Production Endpoint' (http://ws.cdyne.com/phoneverify), 'Sandbox Endpoint' (empty), 'Endpoint Security Scheme' (Non Secured), 'WSDL' (empty), and 'WADL' (empty). The 'Destination-Based Usage Tracking' dropdown is highlighted with a red box and is set to 'Enabled'.

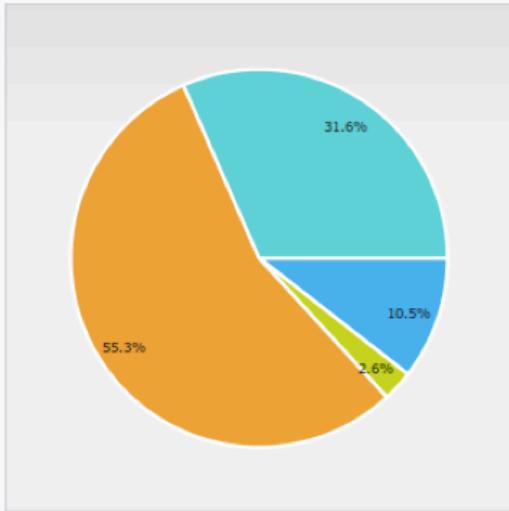
API Store statistics

Log in to the API Store. You can self subscribe to the store. Next, click the Statistics menu.

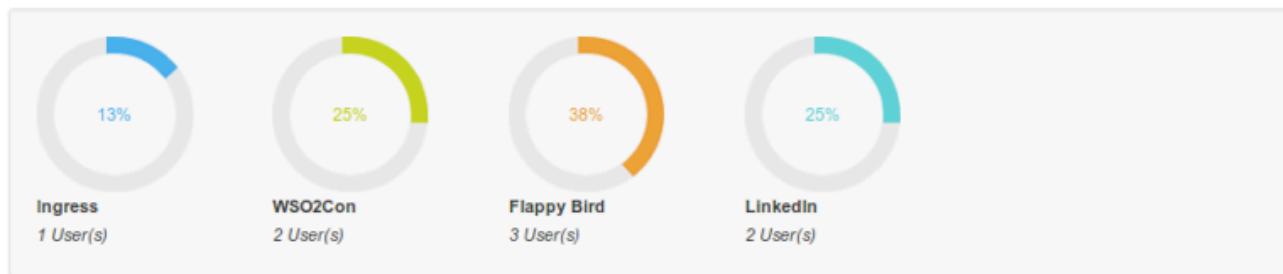
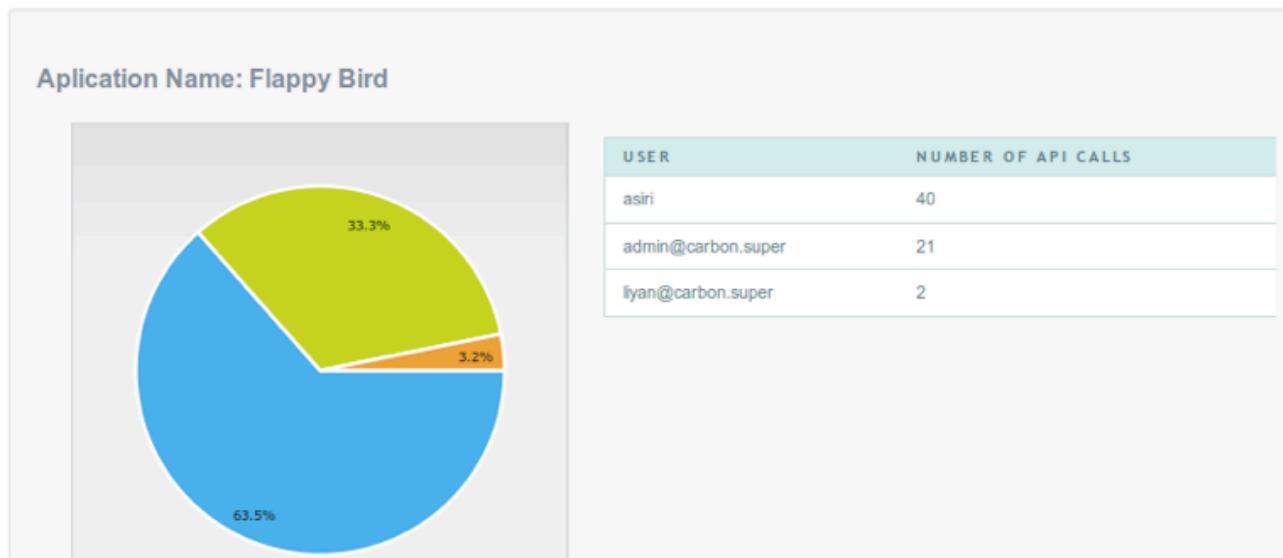
The screenshot shows the navigation bar of the WSO2 API Store. The 'Statistics' menu item is highlighted with a red box.

Several examples of usage and performance statistics are given below:

- API usage per application

Application Name: WSO2Con

- Users who make the most API invocations, per application

Registered Users For Applications**Top Users For Applications**

- API usage from resource path, per application

API Usage from Resource Path

APPLICATION NAME	API NAME	API USAGE FROM RESOURCE PATH PER APPLICATION
Ingress	Foursquare	/top_rated (GET)
	Google	/most_popular (GET) /top_rated (GET)
	facebook	/top_rated (GET)
	tube	/top_rated (GET)
WSO2Con	Foursquare	/most_shared (GET)
	Google	/most_popular (GET)
	facebook	/most_popular (GET)
		/most_viewed (GET)
		/top_rated (GET)
	tube	/most_popular (GET)
Flappy Bird	Foursquare	/most_shared (GET)
	Google	/top_rated (GET)
	facebook	/most_viewed (GET)

- Number of faulty API invocations, per application
In a faulty API invocation, the message is mediated through the `fault` sequence. By default, the API Manager considers an API invocation to be faulty when the backend service is unavailable.

Faulty Invocations per Application

APPLICATION NAME	API NAME	FAULTY API INVOCATION COUNT
Ingress	Foursquare	1
	Google	6
WSO2Con	Foursquare	4
	Google	1
Flappy Bird	Foursquare	8
	Google	6

Samples

The WSO2 API Manager comes with a set of working samples that demonstrate some of its basic features and capabilities. The following topics provide information on executing these samples and evaluating their results.

- [Setting up the Samples](#)
- [Deploying and Testing YouTube API](#)
- [Generating Billing Data](#)
- [Invoking APIs using a Web App Deployed in WSO2 AS](#)
- [Deploying and Testing a Wikipedia API](#)

Setting up the Samples

The API Manager binary distribution comes with a number of samples to demonstrate API Manager's basic functionality. These samples are located in `<APIM_HOME>/samples` folder. Inside this directory, there are sub directories for each sample. Each sub directory contains the relevant configurations, scripts and instructions required to run the a sample. Each sample contains an `APIPopulator` script, which drives the API Manager via a REST API.

The sections below describe the generic setup instructions and prerequisites to run API Manager samples:

- [Prerequisites](#)
- [Setting up samples](#)

Executing these steps only once is enough to try multiple samples in a single API Manager installation.

Prerequisites

- Java Development Kit/JRE version 1.6.* or 1.7.*
- Apache Ant 1.6.x or later
- An HTTP client tool such as cURL (<http://curl.haxx.se>)
- A JavaScript compatible web browser
- An active Internet connection

Setting up samples

1. Download and install the API Manager according to the instructions given in [Getting Started](#).
2. Before installing samples, you must configure libraries. Go to `<APIM_HOME>/bin` directory using a command prompt (on Windows) or text Linux console (on Linux) and type `ant` command. This step populates master data required for the API Manager to start up. For example, on Windows:

```
C:\wso2\wso2am-1.0.0\wso2am-1.0.0\bin>ant
Buildfile: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\bin\build.xml

setup:
  [mkdir] Created dir: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\lib
  [copy] Copying 76 files to C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\lib
  [mkdir] Created dir: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\tmp\setup
  [unzip] Expanding: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\components\plugins\org.wso2.ca
s2am-1.0.0\tmp\setup
  [unzip] Expanding: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\components\plugins\org.wso2.ca
2am-1.0.0\tmp\setup
  [unzip] Expanding: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\components\plugins\org.wso2.ca
-1.0.0\tmp\setup
  [delete] Deleting directory C:\wso2\wso2am-1.0.0\wso2am-1.0.0\tmp\setup
  [unzip] Expanding: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\components\plugins\h2-database
1.0.0\repository\lib
  [unzip] Expanding: C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\components\plugins\org.wso2.ca
repository\lib
  [copy] Copying 1 file to C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\lib
  [move] Moving 39 files to C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\lib
  [delete] Deleting directory C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\lib\META-INF
  [delete] Deleting directory C:\wso2\wso2am-1.0.0\wso2am-1.0.0\repository\lib\org

BUILD SUCCESSFUL
Total time: 15 seconds
```

3. Start the API Manager by executing `<APIM_HOME>/bin/wso2server.bat` (on Windows) or `<APIM_HOME>/bin/wso2server.sh` (on Linux). For more information, see [This step also populates more master data required for the server to start up](#).
4. Next, shut down the API Manager.



It is a must to shut down the server before executing step 5 below.

5. Run the `ant` command inside `<APIM_HOME>/samples/Data` directory. An output similar to following appears:

```
C:\wso2\wso2am-1.0.0-Beta3\wso2am-1.0.0-Beta2\samples\Data>ant
Buildfile: C:\wso2\wso2am-1.0.0-Beta3\wso2am-1.0.0-Beta2\samples\Data\build.xml

init:
populate-user-database:
  [sql] Executing resource: C:\wso2\wso2am-1.0.0-Beta3\wso2am-1.0.0-Beta2\samples\Data\UserPopulator.sql
  [sql] 10 of 10 SQL statements executed successfully

BUILD SUCCESSFUL
Total time: 1 second
```

It executes the `UserPopulator.sql`, which creates two user accounts as `provider1` and `subscriber1`. You can use them to log in to the API Publisher and API Store respectively.

6. Start the API Manager again and log in to the API Publisher (<http://localhost:9763/publisher>) using `username/password` as `provider1/provider1`. Similarly, log in to the API Store (<https://localhost:9443/store>) using `username/password` as `subscriber1/subscriber1`.

Next, proceed to executing the samples as described from the next section onwards.

Deploying and Testing YouTube API

- [Introduction](#)
- [Prerequisites](#)
- [Building the Sample](#)
- [Executing the Sample](#)

Introduction

This sample demonstrates how to subscribe to a published API and consume its functionality using the API Store Web application. The API used here provides YouTube feeds.

Prerequisites

Samples Setup1. Execute the steps in [. When you are done, you will have the API Manager started and the relevant scripts run to create user accounts for API Publisher and API Store.](#)

Building the Sample

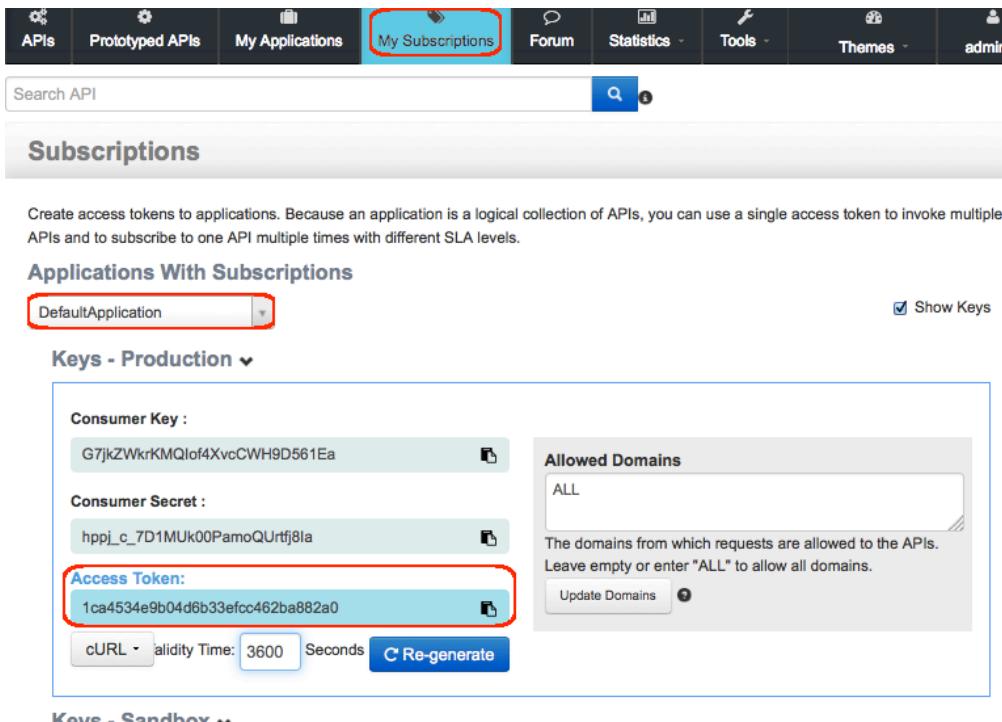
1. First, we need to add an API in the API Publisher and publish it to the API Store. To do that, simply run the APIPopulator.sh (for Linux) or APIPopulator.bat (for Windows) files from folder, <AM_HOME>/samples/YoutubeFeeds.

2. The script will add an API to the API Publisher in Published state. This API can then be consumed by any user signed in to the API Store.

Executing the Sample

Subscribing to the API

1. Log in to the API Store (<https://localhost:9443/store>) with credentials subscriber1/subscriber1.
2. Click the **APIs** tab at the top of the page and select the YoutubeFeeds API.
3. Next, subscribe to this API. Simply select the default application and throttling tier as **Gold**.
4. You will be asked to navigate to **My Subscriptions** tab.
5. Next, you can generate a key to the application. This key allows you to invoke APIs subscribed under a given application. Click on the **Generate** option to obtain an Application key. For example,



The screenshot shows the WSO2 API Manager API Store interface. The top navigation bar includes tabs for APIs, Prototyped APIs, My Applications, and My Subscriptions (which is highlighted with a red box). Below the navigation is a search bar and a toolbar with a magnifying glass icon. The main content area is titled 'Subscriptions'. A sub-section titled 'Applications With Subscriptions' shows a dropdown menu set to 'DefaultApplication' (also highlighted with a red box) and a 'Show Keys' checkbox. Below this is a section for 'Keys - Production' containing fields for 'Consumer Key' (G7jkZWkrKMQlof4XvcCWH9D561Ea), 'Consumer Secret' (hppj_c_7D1MUK00PamoQUrtfj8la), and an 'Access Token' (1ca4534e9b04d6b33efcc462ba882a0). The 'Access Token' field is also highlighted with a red box. A note indicates that the domains from which requests are allowed are 'ALL'. A 'C Re-generate' button is visible. Below this section is another for 'Keys - Sandbox', which notes that keys are not yet generated for this application.

Invoking the API

6. Once you have obtained a key, you can invoke the API using a REST client of your choice. In this example, we use cURL (<http://curl.haxx.se>).
7. Copy and paste following into a new console window and execute it.

```
curl -H "Authorization :Bearer 9nEQnijLZ0Gi0gZ6a3pZICktVUca"
http://localhost:8280/youtube/1.0.0/most_popular
```

where, access token = 9nEQnijLZ0Gi0gZ6a3pZICktVUca. Replace this value with the access token you generated through the API Store in step 5 above.

The access token is passed in the Authorization header as a value of "Bearer". The Authorization header of the message is prefixed by the string "Bearer". This is because, WSO2 API Manager enforces OAuth security on all the published APIs. Any consumer that talks to the API Manager should send their credential (application key) as per the OAuth bearer token profile. If you don't send an application key or send a wrong key, you will receive a 401 Unauthorized response in return.

8. You should be able to see results from YouTube on your console. For example,

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns='http://www.w3.org/2005/Atom' xmlns:app='http://purl.org/atom/app#'
      xmlns:media='http://search.yahoo.com/mrss/'
      xmlns:openSearch='http://a9.com/-/spec/opensearchrss/1.0/'
      xmlns:gd='http://schemas.google.com/g/2005'
      xmlns:yt='http://gdata.youtube.com/schemas/2007'>
  <id>http://gdata.youtube.com/feeds/api/standardfeeds/most_popular</id>
  <updated>2012-07-26T04:51:52.363-07:00</updated>
  <category scheme='http://schemas.google.com/g/2005#kind'
            term='http://gdata.youtube.com/schemas/2007#video' />
  <title type='text'>Most Popular</title>
  <logo>http://www.youtube.com/img/pic_youtubelogo_123x63.gif</logo>
  <link rel='alternate' type='text/html' href='http://www.youtube.com/browse?s=bzb' />...
```

9. Access various other feeds in the YouTube API by changing the last segment of the invoked URL. For example,

```
curl -H "Authorization :Bearer 9nEQnijLZ0Gi0gZ6a3pZICktVUca"
http://localhost:8280/youtube/1.0.0/top_rated
curl -H "Authorization :Bearer 9nEQnijLZ0Gi0gZ6a3pZICktVUca"
http://localhost:8280/youtube/1.0.0/most_shared
curl -H "Authorization :Bearer 9nEQnijLZ0Gi0gZ6a3pZICktVUca"
http://localhost:8280/youtube/1.0.0/most_viewed
```

Replace 9nEQnijLZ0Gi0gZ6a3pZICktVUca with the access token you generated through the API Store in step 5 above.

Generating Billing Data

- [Introduction](#)
- [Prerequisites](#)
- [Building and running the sample](#)

Introduction

This sample demonstrates how to setup WSO2 Business Activity Monitor (BAM) to collect and summarize runtime statistics from the WSO2 API Manager and generate bills for API consumers on usage.

Prerequisites

[Installation Prerequisites](#). Java Development Kit/JRE version 1.6.* or 1.7.*. Also see

- Download and install **WSO2 BAM 2.4.1 or later** using the instructions given in BAM Installation Guide: docs.wso2.org/business-activity-monitor/Getting+Started.

Building and running the sample

Configuring BAM

1. Open <BAM_HOME>/repository/conf/carbon.xml file where <BAM_HOME> is the BAM binary distribution folder that was downloaded as a prerequisite above. Change the carbon.xml file's port offset to 1.

This is done to avoid any port conflicts of running two WSO2 Carbon instances in the same machine.

<Offset>1</Offset>

2. Copy the `API_Manager_Analytics.tbox` in `<APIM_HOME>/samples/Billing` folder to `<BAM_HOME>/repository/deployment/server/bam-toolbox` folder. Create the `bam-toolbox` directory, if it doesn't exist already.



If you have `API_Manager_Analytics.tbox` to the `<BAM_HOME>/statistics` folder before, then you have to uninstall it first and install the new toolbox through the BAM Admin Console. Else, the Hive script used to summarize data on a monthly basis will not get executed.

The API Manager Analytic Toolbox : A toolbox is an installable archive, with a `.tbox` extension. It contains necessary artifacts that models a complete usecase, from collecting data, analyzing through defined Hive scripts to summarizing data through gadgets, Jaggery scripts and other dashboard components.

3. Connect the datasource to the database where the analytical data is stored using the `<BAM_HOME>/repository/conf/datasources/master-datasources.xml` file as follows. In the example, `WSO2AM_STATS_DB` is the datasource used to fetch the analytical data stored in an H2 database. If you want to use a different database, see [Changing the statistics database](#).

```

<datasource>
    <name>WSO2AM_STATS_DB</name>
    <description>The datasource used for getting statistics to API Manager</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_STATS_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <!-- JDBC URL to query the database -->

<url>jdbc:h2:repository/database/APIMGTSTATS_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>

```

4. Because you changed the default BAM port in step 2 above, you must change the Cassandra port given in JDBC connection url in the following datasource configuration found in `bam-datasources.xml` file. (In WSO2 BAM 2.4.0, this is done in `master-datasources.xml`). Since the port offset is 1, the Cassandra port must be 9 1 6 1 .

[Default Ports of WSO2 Products](#) For a list of default ports used by WSO2 products, see .

```

<datasource>
    <name>WSO2BAM_CASSANDRA_DATASOURCE</name>
    <description>The datasource used for Cassandra data</description>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:cassandra://localhost:9161/EVENT_KS</url>
            <username>admin</username>
            <password>admin</password>
        </configuration>
    </definition>
</datasource>

```

⚠ If you run the Hive scripts before changing the default Cassandra port according to the BAM port offset, you keep getting an exception. To overcome this, add the following line at the beginning of the Hive script and rerun.

```
drop table <hive_cassandra_table_name>;
```

5. Add the Cassandra port in the `<Nodes>` element of the `<BAM_HOME>/repository/conf/etc/hector-configuration.xml` file.
6. Start WSO2 BAM server by running `wso2server.bat` (on Windows) and `wso2server.sh` (on Linux)

Configuring API Manager

1. To enable API statistics collection, configure the following properties in `<APIM_HOME>/repository/conf/api-manager.xml` file. Ensure that `<DataSourceName>` name is the same as JNDI config name in `master-datasources.xml` file in BAM.

```

<!-- Enable/Disable the API usage tracker. -->
<Enabled>true</Enabled>

<!-- JNDI name of the data source to be used for getting BAM statistics. This data
source should
be defined in the master-datasources.xml file in conf/datasources directory. -->
<DataSourceName>jdbc/WSO2AM_STATS_DB</DataSourceName>

<!-- Enable/Disable Usage metering and billing for api usage -->
<EnableBillingAndUsage>true</EnableBillingAndUsage>

```

2. Configure the data source definition in `<APIM_HOME>/repository/conf/datasources/master-datasources.xml` file.

Note: Replace `<BAM_HOME>` in the configuration below with the path to the actual BAM distribution location and the JNDI names must match the ones defined earlier in API Manager.

```

<datasource>
    <name>WSO2AM_STATS_DB</name>
    <description>The datasource used for getting statistics to API
Manager</description>
    <jndiConfig>
        <!-- This jndi name should be same as the DataSourceName defined in
api-manager.xml -->
        <name>jdbc/WSO2AM_STATS_DB</name>
    </jndiConfig>
    <definition type="RDBMS" >
        <configuration>
            <!-- JDBC URL to query the database -->

<url>jdbc:h2:<BAM_HOME>/repository/database/APIMGTSTATS_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. Copy <APIM_HOME>/samples/Billing/billing-conf.xml file into <APIM_HOME>/repository/conf folder.

Viewing billing information

Once the above configurations are done, log in to API Store Web application (<https://<YourHostName>:9443/Monetization> in the menu bar at the top of the page). You will see the menu items required for API.

 If you are a new user, there will not be any billing information at the beginning.

Invoking APIs using a Web App Deployed in WSO2 AS

- [Introduction](#)
- [Prerequisites](#)
- [Building the sample](#)
- [Executing the sample](#)

Introduction

This sample demonstrates a pizza ordering scenario with backend services deployed in WSO2 Application Server (AS) to which we create APIs in WSO2 API Manager. Then, we invoke those APIs using a Web application deployed in WSO2 AS.

Prerequisites

- Download and install WSO2 Application Server. For instructions, see [Installation](#).
- As you run the Application Server on the same server as the API Manager, increment the default port of the Application Server to avoid port conflicts. To do this, go to <AS_HOME>/repository/conf/carbon.xml and change <Offset>2</Offset>.

Building the sample

1. Go to <APIM_HOME>/samples/PizzaShack/pizza-shack-api in command shell and run mvn clean install to build the sample.
2. Go to <APIM_HOME>/samples/PizzaShack/pizza-shack-web in command shell and run mvn clean install.
3. See step 7 below if you are rebuilding the sample.

Executing the sample

1. Log in to the API Publisher (<https://localhost:9443/publisher>) and create the following APIs.

Delivery API

```
API Name= pizzaShack
Context = /pizzashack/delivery
Version = 1.0.0
Production Endpoint
URL=http://localhost:9765/pizzashack-api-1.0.0/api/delivery
API Resources =Keep the default values
```

Order API

```
API Name= pizzashack-order
Context = /pizzashack/order
Version = 1.0.0
Production Endpoint
URL=http://localhost:9765/pizzashack-api-1.0.0/api/order
API Resources =Keep the default values
```

Menu API

```
API Name= pizzashack-menu
Context = /pizzashack/menu
Version = 1.0.0
Production Endpoint
URL=http://localhost:9765/pizzashack-api-1.0.0/api/menu
API Resources =Keep the default values
```

2. Navigate to the **Lifecycles** tab of each API and promote them to **PUBLISHED** state. This will push the APIs to the Gateway and they will be available for subscription in the API Store.
3. Log in to the API Store (<https://localhost:9443/store>) and click on each API created earlier. Next, subscribe to each of them using the default application.
4. After subscription, a message appears. Choose **Go to My Subscriptions**.
5. The **Subscriptions** page opens. Create a production key by clicking the **Generate** button associated with it. You also have the option to increase the default token validity period, which is 1 hour.
6. You get the access token, a consumer key and a consumer secret. Replace the consumer key and secret pair in <APIM_HOME>/samples/PizzaShack/pizza-shack-web/src/main/webapp/WEB-INF/web.xml with the newly generated ones. For example,

```
<context-param>
  <param-name>consumerKey</param-name>
  <param-value>szsHscDYLeKUcwA1GhPARQlfus</param-value>
</context-param>
<context-param>
  <param-name>consumerSecret</param-name>
  <param-value>wJEfRDE3JeFnGMuwVNseNzsXM1sa</param-value>
</context-param>
```

You now have three APIs subscribed under an application and an access token to the application. Next, we deploy a Web application in the Application Server and use it to invoke the APIs.

7. Rebuild the sample.

 **If you are rebuilding this sample**, execute the following steps:

- a. Remove the following module from <APIM_HOME>/samples/PizzaShack/pom.xml file: <module>pre-processor</module>.
- b. Delete the PizzaShack.zip file from <APIM_HOME>/samples/PizzaShack.
- c. Go to <APIM_HOME>/samples/PizzaShack in command shell and run mvn clean install.

8. Start WSO2 AS (<https://localhost:9445/console>) and log into its management console. For instructions, see [A S documentation](#) (If the AS documentation link doesn't load, please clear your browser cache and retry).

9. Deploy the following into the Application Server.

- <APIM_HOME>/samples/PizzaShack/pizza-shack-web/target/pizzashack.war
- <APIM_HOME>/samples/PizzaShack/pizza-shack-api/target/pizzashack-api-1.0.0.war

10. After deploying, access the application using <http://localhost:9765/pizzashack>. It opens the application in a Web browser.

11. You can use this application to order pizza. Internally, the APIs get invoked when you use the application.

Deploying and Testing a Wikipedia API

- [Introduction](#)
- [Building the Sample](#)
- [Executing the Sample](#)

Introduction

This sample demonstrates how to subscribe to a published API and consume its functionality using the API Store Web application. We use the Wikipedia API here.

Building the Sample

Samples Setup. Execute the steps in When you are done, you will have the API Manager started and the relevant scripts run to create user accounts for API Publisher and API Store.

 The scripts used for this sample do not work in Windows. Support for Windows will be added in an upcoming release.

Executing the Sample

1. If you haven't done so already, start the API Manager and log in to the API Publisher (<http://localhost:9763/publisher>) using credentials provider1/provider1.
2. There are no APIs created yet. To create one and publish it to the API Store, run the following:
 - On Linux: <APIM_HOME>/samples/WikipediaAPI/APIPopulator.sh
 - On Windows: <APIM_HOME>/samples/WikipediaAPI/APIPopulator.bat
3. Refresh the API Publisher to see the Wikipedia API created.
4. You can now access Wikipedia through this newly-deployed API. Log in to the API Store (<http://localhost:9763/store>) using credentials subscriber1/subscriber1.
5. Select the **Applications** tab at the top of the page, and create a new application. Provide any name you like.
6. Select the **APIs** tab at the top of the page, select the WikipediaAPI API and subscribe to it using the newly-created application.
7. Go to the **My Subscriptions** tab and select your application. Click the **Generate** button associated with the production system to obtain an application access token.

8. You are now ready to invoke the API. Copy and paste following into a new console and execute it. Be sure to replace the string '9nEQnijLZ0Gi0gZ6a3pZICktVUca' with the application access token you obtained earlier.

```
curl -H "Authorization :Bearer 9nEQnijLZ0Gi0gZ6a3pZICktVUca"  
"http://localhost:8280/wikipedia/1.0.0?format=json&action=query&titles=MainPage&p  
rop=revisions&rvprop=content"
```

9. You must see the JSON result from the Wikipedia API on you console. For example,

```
{"query":{ "pages":{ "5982813":{ "pageid":5982813,"ns":0,"title":"MainPage","revisio  
ns": [ { "contentformat": "text/x-wiki", "contentmodel": "wikitext", "/*": "#Redirect  
[[Main Page]]\n\n{{Redr|mod|rcc}}"} ]}}}}
```

See http://www.mediawiki.org/wiki/API:Main_page for more information about the Wikipedia API. You can try out various API actions and features similar to step 9.

Published APIs

The following topics discuss the APIs exposed from the API Publisher and API Store Web applications using which you can create and manage APIs. You can consume APIs directly through their UIs or an external REST client like cURL or the [WSO2 REST client](#). The Token APIs exposed in API Manager are also described here.

- [Publisher APIs](#)
- [Store APIs](#)
- [Token API](#)
- [WSO2 Admin Services](#)

Publisher APIs

Publisher APIs provide the following REST resources.

[[Login](#)] [[Logout](#)] [[Add API](#)] [[Update API](#)] [[Get All APIs](#)] [[Get an API](#)] [[Remove an API](#)] [[Copy an API](#)] [[Check Older Version](#)] [[Change API Status](#)] [[Add/Update an API Document](#)] [[Remove an API Document](#)] [[Get all Throttling Tiers](#)] [[Check if API Exists](#)] [[Validate Roles](#)]

 **Note:** When you access any API other than the login and logout APIs through an external REST client such as cURL, first invoke the login API to ensure that user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations.

The responses is a JSON message.

Login

Description	Log in to API Publisher web application.
URI	http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag
URI Parameters	action=login&username=xxx&password=xxx
HTTP Methods	POST
Example	curl -X POST -c cookies http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag -d 'action=login&username=admin&password=admin'

Logout

Description	Log out from API Publisher web application.
URI	http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag
URI Parameters	?action=logout
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag ?action=logout

Add API

Description	Add a new API.
URI	http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag
URI Parameters	Given below are the parameters that you can pass with an Add-API call. Mandatory ones are marked

Parameter name	Syntax
Action*	action=addAPI
Name*	name=xxx
Context*	context=/xxx
Version*	version=x.x.x
API visibility	visibility=<public private restricted> The default is public. If you select restricted, mention to which roles as follows: You can read more about API visibility from here .
Thumbnail image	thumbUrl<URL> To add a thumbnail image, create a file object of that thumbnail and pass it with the
Description	description=xxx
Tags	tags=x,y,z
Resources*	resourceCount=0&resourceMethod-0=GET&resourceMethodAuthType-0=Application
Endpoints*	This example adds an HTTP production endpoint: <code>endpoint_config={ "productions", " actionSelect":"fault", " actionDuration":60000} } , "endpoi</code> To give advanced endpoint configurations, add the JSON implementation inside "co You add sandbox endpoints in the same way. The only difference is that instead of <code>endpoi</code> If you want to add other types of endpoints like the Address and WSDL, follow the e <ul style="list-style-type: none"> • For address endpoints: <code>endpoint_config={ "production_endpoints":{ "url":"http://serv</code> • For WSDL endpoints: <code>endpoint_config={ "production_endpoints":{ "url":"http://serv</code> • For failover endpoints: <code>endpoint_config={ "production_endpoints":{ "url":"http://serv</code> • For load balanced endpoints: <code>endpoint_config" { "production_endpoints":[{ "url":"http://se</code>
Endpoint security scheme	endpointType=<secured nonsecured> The default is non-secured but if you select 'secured', you must pass the credentials
WSDL and WADL	wsdl=xxx&wadl=xxx

Update API

Description	Update an existing API.
URI	http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag
URI Parameters	Update API: Parameters are same as in Add API except that action=updateAPI and you can only update Url, description, tags, endpointType, endpoint_config (can change the endpoint URL etc,) httpTiersCollection and can also add new resources. See example below.
HTTP Methods	POST

Example	Update API : curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag &provider=admin&version=1.0.0&visibility=public&description=Youtube Feeds&endpointType=nonsecured&http_checked=http&https_checked=https&&wsdl=&tags=youtube www.10bigideas.com.au/www/573/files/pf-thumbnail-youtube_logo.jpg&context=/youtube&tiersCollectionMethod-0=GET&resourceMethodAuthType-0=Application&resourceMethodThrottlingTier-0=Unlimited&endpoint_config={"production_endpoints": {"url": "http://gdata.youtube.com/feeds/api/standardfeeds"}}
---------	--

Get All APIs

Description	Lists all the created APIs.
URI	http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag
URI Parameters	?action=getAllAPIs
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag ?action=getAllAPIs

Get an API

Description	Get details of a specific API.
URI	http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag
URI Parameters	action=getAPI&name=xxx&version=xxx&provider=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag -d "action=getAPI&name=PhoneVerification&version=1.0.0&provider=admin"

Remove an API

Description	Remove an API.
URI	http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag
URI Parameters	action=removeAPI&name=xxx&version=xxx&provider=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag -d "action=removeAPI&name=PhoneVerification&version=1.0.0&provider=admin"

Copy an API

Description	Copy an API to a newer version.
URI	http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag
URI Parameters	action=createNewAPI&provider=xxx&apiName=xxx&version=xxx&newVersion=xxx

HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/publisher/site/blocks?action=createNewAPI&provider=admin&apiName=PhoneVerification&version=1.0.0&newVersion=2.

Check Older Version

Description	Does older version of API exist.
URI	http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag
URI Parameters	?action=isAPIOlderVersionExist&provider=xxx&name=xxx&version=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag?action=isAPIOlderVersion&version=1.0.0

Change API Status

Description	Change the API's status.
URI	http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag
URI Parameters	action=updateStatus&name=xxx&version=1.0.0&provider=apiCreateName&status=PUBLISHED&put
HTTP Methods	POST
Example	curl -X POST -b cookies ' http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag?action=updateStatus&name=PhoneVerification&version=1.0.0&provider=admin&status=PUBLISHED

Add/Update an API Document

Description	Add a new API document.
URI	http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag
URI Parameters	A d d action=addDocumentation&provider=xxx&apiName=xxx&version=xxx&docName=xxx&docType=xxx&docVisibility=xxx Note that docVisibility is applicable only if you have enabled it. See API documentation visibility . U p d a t e action=addDocumentation&mode=Update&provider=xxx&apiName=xxx&version=xxx&docName=xxx
HTTP Methods	POST
Example	Add Document: curl -X POST -b cookies ' http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag?action=addDocumentation&provider=admin&apiName=PhoneVerification&version=1.0.0&docName=testDoc&docType=how to&sourceType=inline&docUrl=&sum

Remove an API Document

Description	Remove an API document.
URI	http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag
URI Parameters	action=removeDocumentation&provider=xxx&apiName=xxx&version=xxx&docName=xxx&docType=>
HTTP Methods	POST
Example	curl -X POST -b cookies ">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag?action=removeDocumentation&provider=admin&apiName=PhoneVerification&version=1.0.0&docName=PhoneVerification&docType=>

Get all Throttling Tiers

Description	Get the throttling tiers that can be applied to APIs
URI	http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?
URI Parameters	action=getTiers
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=getTiers

Check if API Exists

Description	Check if an API by a given name exists in the API Publisher
URI	http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag
URI Parameters	action=isAPINameExist&apiName=<name of the API>
HTTP Methods	GET
Example	curl -b cookies " http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINameExist&apiName=PhoneVerification "

Validate Roles

Description	Check if the user logged in user is any one in a given list of users
URI	http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag
URI Parameters	action=validateRoles&roles=<list of roles>
HTTP Methods	GET
Example	curl -b cookies " http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=validateRoles&roles=admin "

Store APIs

Store APIs provide the following REST resources.

[[Login](#)] [[Logout](#)] [[User Signup](#)] [[Get all Paginated Published APIs](#)] [[Get Published APIs by Application](#)] [[Add an Application](#)] [[Update an Application](#)] [[Get Applications](#)] [[Remove an Application](#)] [[Generate an Application Key](#)] [[Add a Subscription](#)] [[List Subscriptions](#)] [[Remove a Subscription](#)] [[Get all Documentation](#)] [[Add an API Comment](#)]

 **Note:** When you access any API other than the login and logout APIs through an external REST client such as cURL, first invoke the login API to ensure that user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations.

The responses is a JSON message.

Login

Description	Log in to API Store.
URI	http://localhost:9763/store/site/blocks/user/login/ajax/login.jag
URI Parameters	action=login&username=xxx&password=xxx
HTTP Methods	POST
Example	curl -X POST -c cookies http://localhost:9763/store/site/blocks/user/login/ajax/login.jag -d 'action=login&username=admin&password=admin'

Logout

Description	Log out from API Store.
URI	http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout
URI Parameters	?action=logout
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout

User Signup

Description	Add a new API Consumer.
URI	http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag
URI Parameters	action=addUser&username=xxx&password=xxx&allFieldsValues=firstname lastname email
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag -d "action=addUser&username=user1&password=test123&allFieldsValues=firstname lastname email"

Get all Paginated Published APIs

Description	Get a list of all published APIs in paginated form so that browsing is easier.
URI	http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag

URI Parameters	<p><code>action=getAllPaginatedPublishedAPIs</code>, tenant, start, end</p> <p>The <code>start</code> and <code>end</code> parameters determine the range of APIs you want to retrieve. For example, if <code>start=1</code> and <code>end=10</code>, the first 10 APIs that appear in the API Store will be returned. Note that both 0 and 1 represent the first API in the store, so <code>start=0</code> and <code>start=1</code> both specify that you want to start with the first API.</p>
HTTP Methods	GET
Example	<p>To get the first 100 APIs in the API Store:</p> <pre>curl -b cookies "http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllPaginatedPublishedAPIs&tenant=carbon.super&start=1&end=100"</pre>

 Please note that the `getAllPublishedAPIs` API is now deprecated. You can get the same functionality from `getAllPaginatedPublishedAPIs`.

Get Published APIs by Application

Description	Get a list of published APIs filtered by the subscribed Application. Login API needs be called prior to calling this API.
URI	http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag
URI Parameters	<code>action=getSubscriptionByApplication&app=App1</code>
HTTP Methods	GET
Example	<pre>curl -b cookies 'http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getSubscriptionByApplication&app=DefaultApplication'</pre>

Add an Application

Description	Add a new application.
URI	http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag
URI Parameters	<code>action=addApplication&application=xxx&tier=xxx&description=xxx&callbackUrl</code>
HTTP Methods	POST
Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag -d 'action=addApplication&application=NewApp1&tier=Unlimited&description=&callbackUrl='</pre>

Update an Application

Description	Update an existing application.
URI	http://localhost:9763/store/site/blocks/application/application-update/ajax/application-update.jag
URI Parameters	<code>action=updateApplication&applicationOld=xxx&applicationNew=xxx&callbackUrlNew=xxx&description</code>

HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/blocks/application/application-update/ ?action=updateApplication&applicationOld=NewApp1&applicationNew=NewApp2&tier=Unlimited&des

Get Applications

Description	Get list of applications.
URI	http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag
URI Parameters	?action=getApplications
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag ?action=getApplications

Remove an Application

Description	Remove an existing application.
URI	http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag
URI Parameters	action=removeApplication&application=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag -d "action=removeApplication&application=NewApp2"

Generate an Application Key

Description	Generate the key and secret values for a new application.
URI	http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag
URI Parameters	action=generateApplicationKey&application=<app_name>&keytype=<PRODUCTION Sandbox>&callbackUrl=<URL>&authorizedDomains=<The domains from which requests are allowed to the APIs>&validityTime=<time duration in seconds>
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d 'action=generateApplicationKey&application=NewApp1&keytype=PROD&callbackUrl=&authorizedDomains=ALL&validityTime=360000'

Add a Subscription

Description	Add a new API subscription.
URI	http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag

URI Parameters	<ul style="list-style-type: none"> To add a subscription by application ID: action=addSubscription&name=xxx&version=xxx&provider=xxx&applicationId=xxx To add a subscription by application name: action=addAPISubscription&name=xxx&version=xxx&provider=xxx&applicationName=xxx
HTTP Methods	POST
Example	<ul style="list-style-type: none"> curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag 'action=addSubscription&name=TestAPI&version=1.0.0&provider=admin&tier=Gold&applicationId=xxx' curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag 'action=addAPISubscription&name=TestAPI&version=1.0.0&provider=admin&tier=Gold&applicationName=xxx'

List Subscriptions

Description	List all API subscriptions.
URI	http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag
URI Parameters	action=getAllSubscriptions
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag ?action=getAllSubscriptions

Remove a Subscription

Description	Remove an API subscription.
URI	http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag
URI Parameters	action=removeSubscription&name=xxx&version=xxx&provider=xxx&applicationId=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag -d 'action=removeSubscription&name=PhoneVerification&version=1.0.0&provider=admin&applicationId=xxx'

Get all Documentation

Description	Get all documents create for a given API
URI	http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag
URI Parameters	action=getAllDocumentationOfApi&name=<API Name>&version=x.x.x&provider=<Name of the API provider>
HTTP Methods	GET
Example	curl -b cookies "http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag?action=getAllDocumentationOfApi&name=PhoneVerification&version=1.0.0&provider=admin"

Add an API Comment

Description	Add a comment for an API.
URI	http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag
URI Parameters	action=addComment&name=xxx&version=xxx&provider=xxx&comment=xxx
HTTP Methods	POST
Example	curl -X POST -b cookies http://ore/site/blocks/comment/comment-add/ajax/comment-add.jag -d 'action=addComment&name=PhoneVerification&version=1.0.0&provider=admin&comment=test comment'

Token API

Users need access tokens to invoke APIs subscribed under an application. Access tokens are passed in the HTTP header when invoking APIs. The API Manager provides a Token API that you can use to generate and renew user and application access tokens. The response of the Token API is a JSON message. You extract the token from the JSON and pass it with an HTTP Authorization header to access the API.

Let's take a look at how to generate/renew access tokens and authorize them. WSO2 API Manager supports the four most common **authorization grant types** and you can also define additional types such as SAML.

- Generating access tokens with user credentials (password grant type)
- Generating access tokens with authorization code (authorization code grant type)
- Exchanging SAML2 bearer tokens with OAuth2 (SAML extension grant type)
- Renewing access tokens
- Revoking access tokens
- Configuring the token expiration time

Generating access tokens with user credentials (password grant type)

You can obtain an access token by providing the resource owner's username and password as an authorization grant. It requires the base64 encoded string of the `consumer-key:consumer-secret` combination. You need to meet the following prerequisites before using the Token API to generate a token.

Prerequisites

- A valid user account in the API Store. You can self sign up if it is [enabled by an admin](#).
- A valid consumer key and consumer secret pair. Initially, these keys must be generated through the management console by clicking the **Generate** link on **My Subscriptions** page. You can find more details in [Invoke an API using the Integrated REST Client](#).
- A running API Gateway instance (typically an API Manager instance should be running). For instructions on API Gateway, see [Components](#).
- If you have multiple Carbon servers (such as API Manager and WSO2 Application Server) running on the same computer, you must [change the port offset](#) to avoid port conflicts. Setting the port offset causes API Manager to run on a different port from the default.
- If the Key Manager is running on a different server from the API Gateway instance, change the host and port of the endpoints of the default APIs that are in `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` to the correct address of the Key Manager.

Invoking the Token API to generate tokens

1. Combine the consumer key and consumer secret keys in the format `consumer-key:consumer-secret` and encode the combined string using base64. Encoding to base64 can be done using the URL: <http://base64encode.org>.
Here's an example consumer key and secret combination : `wU62Dj1yDBnq87G1BwplfqvmAbAa:ksdSdoefDDP7wpaElfqvmjDue`.
2. Access the Token API by using a REST client such as the [WSO2 REST Client](#) or Curl, with the following

parameters.

- Assuming that both the client and the API Gateway are run on the same server, the token API url is `http://localhost:8243/token`
- **payload** - "grant_type=password&username=<username>&password=<password>&scope=<scope>". Replace the <username> and <password> values as appropriate. <scope> is optional, you can leave it off if necessary
- **headers** - `Authorization: Basic <base64 encoded string>, Content-Type: application/x-www-form-urlencoded`. Replace the <base64 encoded string> as appropriate.

For example, use the following cURL command to access the Token API. It generates two tokens as an access token and a refresh token. You can use the refresh token at the time a token is renewed .

```
curl -k -d "grant_type=password&username=<username>&password=<password>" -H
"Authorization: Basic
SVPzSWk2SERiQjV1OFZLZfpBblVpX2ZaM2Y4YTpHbTBisjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzhh,
Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

CuRL command with Scopes

```
curl -k -d
"grant_type=password&username=<username>&password=<password>&scope=<scope1>
<scope2>" -H "Authorization: Basic
SVPzSWk2SERiQjV1OFZLZfpBblVpX2ZaM2Y4YTpHbTBisjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzhh,
Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

 **Tip:** If you define a **scope** for an API's resource, the API can only be accessed through a token that is issued for the scope of the said resource. For example, if you define a scope named 'update' and issue one token for the scopes 'read' and 'update', the token is allowed to access the resource. However, if you issue the token for the scope named 'read', the request to the API will be blocked.

 **!** The Token API endpoint is specified in `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api/_TokenAPI_.xml` file. When running the server on a different port from the default (i.e., 9443), or if your Key Manager is running on a different machine from your API Gateway, you must update the endpoint inside the `_TokenAPI_.xml` file as described in the [prerequisites](#).

User access tokens have a fixed expiration time, which is set to 60 minutes by default. Before deploying the API manager to users, extend the default expiration time by editing the `<AccessTokenDefaultValidityPeriod>` tag in `<PRODUCT_HOME>/repository/conf/identity.xml`.

When a user access token expires, the user can try regenerating the token as explained in the [Renew user tokens](#) section.

Instead of using the Token API, you can generate access tokens from the API Store UI. See [Invoke an API using the Integrated REST Client](#) for information.

[Generating access tokens with authorization code \(authorization code grant type\)](#)

Instead of requesting authorization directly from the resource owner (resource owner's credentials), in this grant type, the client directs the resource owner to an authorization server. The authorization server works as an intermediary between the client and resource owner to issues an authorization code, authenticate the resource owner and obtain authorization. As this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a Web browser) and receiving incoming requests (via redirection) from the authorization server.

The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint (you can use the `/authorize` endpoint for the authorization code grant type of OAuth 2.0). It includes the client identifier, `response_type`, requested scope, and a redirection URI to which the authorization server sends the user-agent back after granting access. The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner granted or denied the client's access request. Assuming the resource owner grants access, the authorization server then redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes an authorization code.

The client then requests an access token from the authorization server's `/token` endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. It then includes the redirection URI used to obtain the authorization code for verification. The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI matches the URI used to redirect the client from the `/authorize` endpoint in the previous response. If valid, the authorization server responds back with an access token and, optionally, a refresh token.

Invoking the Token API to generate tokens

Assuming that both the client and the API Gateway are run on the same server, the Authorization API URL is <https://localhost:8243/authorize>.

- query component: `response_type=code&client_id=<consumer_key>&scope=PRODUCTION&redirect_uri=<application_callback_url>`
- headers: `Content-Type: application/x-www-form-urlencoded`

For example, the client directs the user-agent to make the following HTTP request using TLS.

```
GET
/authorize?response_type=code&client_id=wU62DjlyDBnq87G1BwplfqvmAbAa&scope=PRODUCTION&
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcbs
HTTP/1.1
Host: server.example.com
Content-Type:
application/x-www-form-urlencoded
```

The authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location:
https://client.example.com/cb?code=Spxl0BeZQQYbYS6WxSbIA
```

Now the client makes the following HTTP request using TLS to the `/token` endpoint.

```

POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic
SVpzSWk2SERiQjV1OFZLZFpBb1VpX2ZaM2Y4YTpHbTBiSjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzh
Content-Type:
application/x-www-form-urlencoded
grant_type=authorization_code&code=Splx10BeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F%2F
client%2Eexample%2Ecom%2Fc

```

The `/token` endpoint responds in the same way like in password grant type.

Note that if you are using a separate server for authentication (e.g., a distributed API Manager setup or an instance of WSO2 Identity Server as the authentication server), be sure to give the full URL of the authentication server in `<API_HOME>/repository/conf/security/application-authenticators.xml` file. The default configuration has a relative path, which works in a standalone API Manager setup:

```

<Authenticators>
    <Authenticator name="BasicAuthenticator" disabled="false" factor="1">
        <Status value="10" loginPage="/authenticationendpoint/login.do" />
    </Authenticator>
</Authenticators>

```

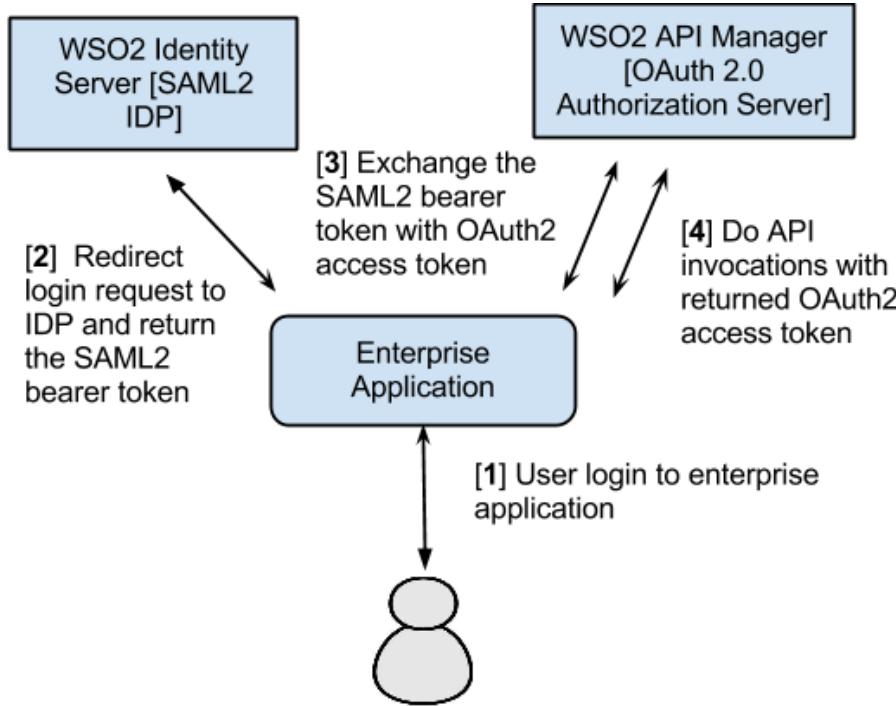
Exchanging SAML2 bearer tokens with OAuth2 (SAML extension grant type)

SAML 2.0 is an XML-based protocol. It uses security tokens containing assertions to pass information about an end-user between a SAML authority and a SAML consumer. A SAML authority is an identity provider (IDP) and a SAML consumer is a service provider (SP).

Enterprise applications use SAML2 to engage a third-party identity provider to grant access to systems that are only authenticated against the enterprise application. These enterprise applications might need to consume OAuth-protected resources through APIs, after validating them against an OAuth2.0 authentication server. However, an enterprise application that already has a working SAML2.0 based SSO infrastructure between itself and the IDP prefers to use the existing trust relationship, even if the OAuth authorization server is entirely different from the IDP. The SAML2 Bearer Assertion Profile for OAuth2.0 leverages this existing trust relationship. It presents the SAML2.0 token to the authorization server and exchanges it to an OAuth2.0 access token.

WSO2 API Manager provides SAML2 Bearer Assertion Profile Support with the OAuth 2.0 feature. **WSO2 Identity Server** (version 4.5.0 onwards) or any other SAML2 Identity provider can act as an identity service provider for the systems enabled with SSO. WSO2 API Manager acts as the OAuth authorization server. This way, an enterprise application can exchange the SAML2.0 bearer token that it retrieves when authenticating against an IDP (e.g., WSO2 Identity Server) with an OAuth2.0 access token from an OAuth authorization server (e.g., WSO2 API Manager). It can then use the OAuth2 token in API invocations.

The diagram below depicts this scenario:



The scenarios of the above diagram are explained below:

Scenario [1]: User initiates login call to an enterprise application .

Scenario [2]:

- As the application is a SAML SP, it redirects the user to the SAML2.0 IDP to log in.
- The user provides credentials at the IDP and is redirected back to the SP with a SAML2.0 token signed by the IDP.
- The SP verifies the token and logs the user to the application.
- The SAML 2.0 token is stored in the user's session by the SP.

Scenario [3]:

- The enterprise application (SP) wants to access an OAuth2 protected API resource through WSO2 API Manager.
- The application makes a request to the API Manager to exchange the SAML2 bearer token for an OAuth2.0 access token.
- The API Manager validates the assertion and returns the access token.

Scenario [4]: User does API invocations through the API Manager by setting it as an Authorization header with the returned OAuth2 access token.

Before you configure the token exchange, do the following:

- Register to a valid user account in the API Store.
- Get a valid consumer key and consumer secret. Initially, these keys must be generated through the management console by clicking the **Generate** link on **My Subscriptions** page. For more information, see [Invoke an API using the Integrated REST Client](#).
- Set up a running API Gateway instance.
- If you have multiple WSO2 servers (such as WSO2 API Manager and WSO2 Application Server) running on the same machine, change the port offset and update the token API's endpoint accordingly. Additionally, if the key manager is on a different server from the API Gateway, update the token API endpoint to use the correct host and port. For more information, see [this prerequisite](#) in the previous section.

Configuring the token exchange

We use **WSO2 Identity Server 5.0.0** as the IDP to get a SAML token and the API Manager as the OAuth server.

1. Log in to the API Manager's management console (<https://localhost:9443/carbon>) using admin/admin credentials and select **Add** under **Identity Providers** menu in the **Main** menu.



2. Provide the following values to configure the IDP:

- Under **Basic Information**
 - **Identity Provider Name:** Enter a unique name for IDP
 - **Identity Provider Public Certificate:** Export the public certificate of WSO2 IS and import it here. Alternatively, you can create a self-signed certificate and then export it as a .cer file using the following commands:

```
keytool -genkey -alias wookie -keyalg RSA -keystore wookieKeystore.jks
-keysize 4096
keytool -v -export -file keystore1.cer -keystore keystore1.jks -alias
keystore1
```

- **Alias:** Give the name of the alias if the Identity Provider identifies this token endpoint by an alias. E.g., <https://localhost:9443/oauth2/token>
- Under **Federated Authenticators -> SAML2 Web SSO Configuration**
 - **Enable SAML2 Web SSO:** true
 - **Identity Provider Entity Id:** The SAML2 issuer name specified when generating the assertion token, which contains the unique identifier of the IDP. **You give this name when configuring the SP.**
 - **Service Provider Entity Id:** Issuer name given when configuring the SP
 - **SSO URL:** Enter the IDP's SAML2 Web SSO URL value. E.g., <https://localhost:9444/samlsso/> if you have offset the default port, which is 9443.

Add Identity Provider

Basic Information

Identity Provider Name:*	IS	Enter a unique name for this identity provider				
Display Name:	Specify the identity provider's display name					
Description:	A meaningful description about the identity provider					
Federation Hub Identity Provider:	<input type="checkbox"/> Check if this points to a federation hub identity provider					
Home Realm Identifier:	Enter the home realm identifier for this identity provider					
Identity Provider Public Certificate:	<input type="button" value="Browse..."/> No file selected. <input type="button" value="Delete Identity Provider Public Certificate"/> Delete Identity Provider Public Certificate <small>Upload identity provider's public certificate in PEM format</small>					
Alias:	https://localhost:9443/oauth2/toke					

If the resident identity provider is known by an alias at the federated identity provider specify it

Claim Configuration

Role Configuration

Federated Authenticators

OpenID Configuration

SAML2 Web SSO Configuration

Enable SAML2 Web SSO Specifies if SAML2 Web SSO is enabled for this identity provider

Default Specifies if SAML2 Web SSO is the default

Identity Provider Entity Id:*

Service Provider Entity Id:*

SSO URL:*

Enable Authentication Request Signing Specifies if the SAML2 authentication request to the identity provider must be signed or not

Enable Assertion Encryption Specify if SAMLAssertion element is encrypted

Enable Assertion Signing Specify if SAMLAssertion element is signed

Enable Logout Specifies if logout/single Logout is enabled for this identity provider

Logout Url:

Enable Logout Request Signing Specifies if SAML2 logout request to the identity provider must be signed or not

Enable Authentication Response Signing Specifies if SAML2 authentication response from the identity provider must be signed or not

SAML2 Web SSO User ID Location: User ID found in 'Name Identifier'. User ID found among claims Specifies the location to find the user identifier in the SAML2 assertion

Additional Query Parameters:

OAuth2/OpenID Connect Configuration

WS-Federation (Passive) Configuration

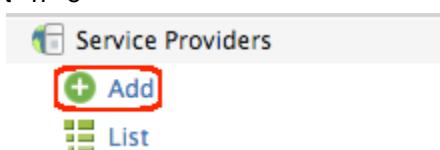
Facebook Configuration

Just-in-Time Provisioning

Outbound Provisioning Connectors

Update Cancel

3. Log in to the management console of the Identity Server and select **Add** under **Service Providers** menu in the main menu.



4. Choose to edit the service provider that you just registered and select **SAML2 Web SSO Configuration**.

Service Providers

Basic Information

Service Provider Name:*	TestSP
<small> ⓘ A unique name for the service provider</small>	
Description:	
<small> ⓘ A meaningful description about the service provider</small>	

Claim Configuration
 Role/Permission Configuration
 Inbound Authentication Configuration
 SAML2 Web SSO Configuration
 OAuth/OpenID Connect Configuration
 WS-Trust Security Token Service Configuration
 WS-Federation (Passive) Configuration
 OpenID Configuration

5. Provide the following values to configure the SP:

- **Issuer:** Give any name
- **Assertion Consumer URL:** The URL to which the IDP sends the SAML response. E.g., https://localhost:9443/store/jagg/jaggery_acs.jag
- **Enable Response Signing:** true
- **Enable Assertion Signing:** true
- **Enable Audience Restriction:** true
- **Audience:** URL of the token API. E.g., <https://localhost:9443/oauth2/token>

Register New Service Provider

Edit Service Provider(TestSP)

Issuer *	TestSP
Assertion Consumer URL *	https://localhost:9443/store/jagg/jagg
NameID format	urn:oasis:names:tc:SAML:1.1:nameid-format
<input type="checkbox"/> Use fully qualified username in the NameID <input checked="" type="checkbox"/> Enable Response Signing <input checked="" type="checkbox"/> Enable Assertion Signing <input type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests <input type="checkbox"/> Enable Assertion Encryption 	
<p>Certificate Alias</p> <div style="border: 1px solid #ccc; padding: 2px; width: 150px;">wso2carbon.cert</div>	
<input type="checkbox"/> Enable Single Logout <p>Custom Logout URL</p> <div style="border: 1px solid #ccc; padding: 2px; width: 150px;"></div>	
<input checked="" type="checkbox"/> Enable Attribute Profile <input type="checkbox"/> Include Attributes in the Response Always	
<input checked="" type="checkbox"/> Enable Audience Restriction <p>Audience</p> <div style="border: 1px solid #ccc; padding: 2px; width: 150px;">https://localhost:9443/oauth2/token</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; margin-left: 10px;">Add Audience</div>	
<input type="checkbox"/> Enable Recipient Validation <p>Recipient</p> <div style="border: 1px solid #ccc; padding: 2px; width: 150px;"></div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; margin-left: 10px;">Add Recipient</div>	
<input type="checkbox"/> Enable IdP Initiated SSO	

Let's see how to get a signed SAML2 token (encoded assertion value) when authenticating against a SAML2 IDP. With the authentication request, you pass attributes such as the SAML2 issuer name, token endpoint and the restricted audience. In this guide, we use a command-line client program to create the SAML2 assertion.

6. Get the SAML token using the client JAR. An example command is given below. TestSP is the name of the issuer.

```
java -jar SAML2AssertionCreator.jar TestSP admin
https://localhost:9443/oauth2/token
https://localhost:9443/oauth2/token/home/dinusha/nothing/WSO2/API-Manager/saml-oa
uth/wso2is-5.0.0/rhbepository/resources/security/wso2carbon.jks wso2carbon
wso2carbon wso2carbon
```

7. Get the OAuth Access token. An example command is given below.

```
curl -k -d
"grant_type=urn:ietf:params:oauth:grant-type:saml2-bearer&assertion=<ASSERTION_PR
OVIDED_BY_CLIENT>&scope=PRODUCTION" -H "Authorization: Basic <Base64 encoded
consumer key:consumer secret>, Content-Type:
application/x-www-form-urlencoded" https://localhost:9443/oauth2/token
```

Invoking Token API to generate tokens

Follow the steps below to invoke the token API to generate access tokens from SAML2 assertions.

1. Combine the consumer key and consumer secret keys as consumer-key:consumer-secret. Encode the combined string using base64 (<http://base64encode.org>). Here's an example consumer key and secret combination: `wU62DjlyDBnq87G1Bwp1fqvmAbAa:ksdSdoefDDP7wpaElfqvmjDue`.
2. Access the token API using a REST client such as the [WSO2 REST Client](#) or Curl. The parameters are explained below:
 - Assuming that both the client and the API Gateway run on the same server, the Token API URL is `http://localhost:8243/token`.
 - Create a SAML2 Assertion. You can use the command line client program from [here](#). Extract the ZIP file, change directory into the extracted folder and execute the following command in the command line. You will get SAML2 Assertion XML String and base64-URL Encoded Assertion XML String. Use base64-URL Encoded Assertion XML String as SAML2_Encoded Assertion_Token.

```
java -jar SAML2AssertionCreator.jar <Identity_Provider_Entity_Id> admin
https://localhost:9443/oauth2/token https://localhost:9443/oauth2/token
<Identity_Provider_JKS_file> <Identity_Provider_JKS_password>
<Identity_Provider_certificate_alias>
```

The arguments are as follows:

- The saml:Issuer (a unique identifier of the identity provider) value
- The saml:Subject -> saml:NameID value
- The value of saml:Subject -> saml:SubjectConfirmation -> saml:SubjectConfirmationData.Recipient
- The fourth argument can take multiple values separated by commas. They are added to the saml:AudienceRestriction element of the token. Each value is added as a saml:Audience element within saml:AudienceRestriction.
- Pointer to the Java Key Store (JKS) file to be used for credentials
- The JKS password
- The alias of the public certificate
- The password of the private key that is used for signing
- payload - "grant_type=urn:ietf:params:oauth:grant-type:saml2-bearer&assertion=<SAML2_Encoded Assertion_Token> &scope=PRODUCTION". Replace the <SAML2_Encoded Assertion_Token> value as appropriate.
- headers - Authorization :Basic <base64 encoded consumer-key:consumer-secret>, Content-Type: application/x-www-form-urlencoded. Replace the <base64 encoded consumer-key:consumer-secret> as appropriate.

For example, the following Curl command is used to access the Token API. It generates an access token and a refresh token. You can use the refresh token at the time a [token is renewed](#).

```
curl -k -d
"grant_type=urn:ietf:params:oauth:grant-type:saml2-bearer&assertion=<SAML2_Encode
d_Assertion_Token>&scope=PRODUCTION" -H "Authorization: Basic
SVpzSWk2SERiQjV1OFZLZfpBb1VpX2ZaM2Y4YTpHbTBiSjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzhh,
Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

 The Token API endpoint is specified in <APIM_HOME>/repository/deployment/server/synapse-configs/default/api/_TokenAPI_.xml file. When running the server on a different port from the default (i.e., 9443), or if your Key Manager is running on a different server from your API Gateway, you must update the endpoint inside the _TokenAPI_.xml file as described [here](#).

Renewing access tokens

After an access token is generated, sometimes you might have to renew the old token due to expiration or security concerns. You can renew an access token using a refresh token, by issuing a REST call to the Token API with the following parameters.

- Assuming that both the client and the API Gateway are run on the same server, the Token API URL is <https://localhost:8243/token>.
- payload - "grant_type=refresh_token&refresh_token=<retoken>&scope=PRODUCTION". Replace the <retoken> value with the refresh token generated in the [previous section](#).
- headers - Authorization :Basic <base64 encoded string>, Content-Type: application/x-www-form-urlencoded. Replace <base64 encoded string> as appropriate.

For example, the following cURL command can be used to access the Token API.

```
curl -k -d "grant_type=refresh_token&refresh_token=<retoken>&scope=PRODUCTION" -H
"Authorization: Basic
SVpzSWk2SERiQjV1OFZLZfpBb1VpX2ZaM2Y4YTpHbTBiSjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVzhh,
Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

The above REST message grants you a renewed access token along with a refresh token, which you can use the next time you renew the access token. A refresh token can be used only once. At the moment, a refresh token never expires, but we will provide a way to configure an expiration time in a future release.

Revoking access tokens

After issuing an access token, a user or an admin can revoke it in case of theft or a security violation. You can do this by calling Revoke API using a utility like cURL. The Revoke API's endpoint URL is <http://localhost:8280/revoke>.

Parameters required to invoke this API are as follows:

- The token to be revoked
- Consumer key and consumer secret key. Must be encoded using Base64 algorithm

For example, `curl -k -d "token=<ACCESS_TOKEN_TO_BE_REVOKED>" -H "Authorization: Basic Base64Encoded(Consumer key:consumer secret)" http://localhost:8280/revoke`.

 When the API Gateway cache is enabled (it is enabled by default), even after revoking a token, it might still be available in the cache to consumers until the cache expires in approximately 15 minutes. You can clear the cache manually by restarting the server.

Configuring the token expiration time

Configuring the token expiration time

User access tokens have a fixed expiration time, which is set to 60 minutes by default. Before deploying the API Manager to users, extend the default expiration time by editing the `<AccessTokenDefaultValidityPeriod>` element in `<PRODUCT_HOME>/repository/conf/identity.xml`.

Also take the **time stamp skew** into account when configuring the expiration time. The time stamp skew is used to manage small time gaps in the system clocks of different servers. For example, let's say you have two Key Managers and you generate a token from the first one and authenticate with the other. If the second server's clock runs 300 seconds ahead, you can configure a 300s time stamp skew in the first server. When the first Key Manager generates a token (e.g., with the default life span, which is 3600 seconds), the time stamp skew is deducted from the token's life span. The new life span is 3300 seconds and the first server calls the second server after 3200 seconds.

You configure the time stamp skew using the `<TimestampSkew>` element in `<PRODUCT_HOME>/repository/conf/identity.xml`.

Ideally, the time stamp skew should not be larger than the token's life span. Also, note that when the API Gateway cache is enabled (it is enabled by default), even after a token expires, it will still be available in the cache for consumers until the cache expires in approximately 15 minutes.

WSO2 Admin Services

WSO2 products are managed internally using SOAP Web services known as **admin services**. WSO2 products come with a management console UI, which communicates with these admin services to facilitate administration capabilities through the UI.

A service in WSO2 products is defined by the following components:

- Service component: provides the actual service
- UI component: provides the Web user interface to the service
- Service stub: provides the interface to invoke the service generated from the service WSDL

There can be instances where you want to call back-end Web services directly. For example, in test automation, to minimize the overhead of having to change automation scripts whenever a UI change happens, developers prefer to call the underlying services in scripts. The topics below explain how to discover and invoke these services from your applications.

Discovering the admin services

By default, the WSDLs of admin services are hidden from consumers. Given below is how to discover them.

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 `-D osgiConsole` option, such as `sh <PRODUCT_HOME>/bin/wso2server.sh -D osgiConsole` 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:

```
osgi> listAdminServices
Admin services deployed on this server:
1. ProvisioningAdminService, ProvisioningAdminService, https://192.168.219.1:8243/services/ProvisioningAdminService
2. SynapseApplicationAdmin, SynapseApplicationAdmin, https://192.168.219.1:8243/services/SynapseApplicationAdmin
3. CarbonAppUploader, CarbonAppUploader, https://192.168.219.1:8243/services/CarbonAppUploader
4. OperationAdmin, OperationAdmin, https://192.168.219.1:8243/services/OperationAdmin
5. SequenceAdminService, SequenceAdminService, https://192.168.219.1:8243/services/SequenceAdminService
6. MediationLibraryAdminService, MediationLibraryAdminService, https://192.168.219.1:8243/services/MediationLibraryAdminService
7. StatisticsAdmin, StatisticsAdmin, https://192.168.219.1:8243/services/StatisticsAdmin
8. LoggedUserInfoAdmin, LoggedUserInfoAdmin, https://192.168.219.1:8243/services/LoggedUserInfoAdmin
9. MediationStatisticsAdmin, MediationStatisticsAdmin, https://192.168.219.1:8243/services/MediationStatisticsAdmin
10. TopicManagerAdminService, TopicManagerAdminService, https://192.168.219.1:8243/services/TopicManagerAdminService
11. MessageProcessorAdminService, MessageProcessorAdminService, https://192.168.219.1:8243/services/MessageProcessorAdminService
12. ApplicationAdmin, ApplicationAdmin, https://192.168.219.1:8243/services/ApplicationAdmin
13. NDataSourceAdmin, NDataSourceAdmin, https://192.168.219.1:8243/services/NDataSourceAdmin
14. ServiceGroupAdmin, ServiceGroupAdmin, https://192.168.219.1:8243/services/ServiceGroupAdmin
15. ClassMediatorAdmin, ClassMediatorAdmin, https://192.168.219.1:8243/services/ClassMediatorAdmin
```

7. 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.

8. Note that the admin service's URL appears as follows in the list you discovered in step 6:

```
AuthenticationAdmin, AuthenticationAdmin, https://<host
IP>:8243/services/AuthenticationAdmin
```

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();
    }
}

```

 To resolve dependency issues, if any, add the following dependency JARs location to the class path: <PRODUCT_HOME>/repository/components/plugins.

 The the AuthenticationAdminStub class requires `org.apache.axis2.context.ConfigurationContext` as a parameter. You can give a null value there.

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.

```

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);
    }

    public void deleteService(String[] serviceGroup) throws RemoteException {
        serviceAdminStub.deleteServiceGroups(serviceGroup);
    }

    public ServiceMetaDataWrapper listServices() throws RemoteException {
        return serviceAdminStub.listServices("ALL", "*", 0);
    }
}

```

The following sample code lists the back-end Web services:

```
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();
    }
}
```

Admin Guide

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

- [Migrating the API Manager](#)
- [Deploying and Clustering the API Manager](#)
- [Tuning Performance](#)

Migrating the API Manager

If you have multiple instances of the WSO2 API Manager and want to move your data and deployment artifacts from one instance to another (such as moving from development to test or production), follow the steps below.

1. Get a data dump from all the tables in the apimgt schema and dump them to the schema in the new environment.
2. Open `<APIM_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases in the new environment.
 - User Store
 - Registry database
 - API Manager Databases
3. Edit the registry configurations in the `<APIM_HOME>/repository/config/registry.xml` and the user database in the `<APIM_HOME>/repository/conf/user-mgt.xml` file.
4. Move all your synapse configurations by copying and replacing `<APIM_HOME>/repository/deployment/server/synapse-config/default` directory to the same directory in the new environment.

 If you changed the default URLs in `AuthorizeAPI.xml` and `TokenAPI.xml` files, do not replace them when copying. They are application-specific APIs.

Migrate tenants

5. If you have **multiple tenants** added to your API Manager instance, follow the steps below to migrate tenant configurations:
 - a. Copy the contents from `<APIM_HOME>/repository/tenants` directory to the same directory in the new environment.
 - b. Execute the following steps for all tenants in your system.

Migrate external stores

6. If you have **external stores** configured under the `<ExternalAPIStructures>` element in `<APIM_HOME>/repository/conf/api-manager.xml` file, follow the steps below:
 - a. Log in to APIM management console and click the **Resources -> Browse** menu.
 - b. Load `/_system/governance/apimgt/externalstores/external-api-stores.xml` resource in the registry browser UI, configure your external stores there and save.

Migrate Google analytics

7. If you have **Google Analytics** configured under `<GoogleAnalyticsTracking>` element in `<APIM_HOME>/repository/conf/api-manager.xml` file, follow the steps below:
 - a. Log in to APIM management console and go to **Resources -> Browse** menu.
 - b. Load `/_system/governance/apimgt/statistics/ga-config.xml` resource in the registry browser UI, configure the Google analytics and save.

Migrate workflows

8. If you have **Workflows** configured under `<WorkFlowExtensions>` element in `<APIM_HOME>/repository/conf/api-manager.xml` file, follow the steps below:
 - a. Log in to APIM management console and go to **Resources -> Browse** menu.
 - b. Load `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource in the registry browser UI, configure your workflows and save.



Upgrading from a Previous Release

See [Upgrading from the Previous Release](#) in the following situations:

- The new environment you are migrating to has a different database version. In this case, you must upgrade the older database.
- You want to upgrade from a previous API Manager release to a new one.

Deploying and Clustering the API Manager

You can install multiple instances of WSO2 products in a cluster to ensure proper load balancing. When one instance becomes unavailable or is experiencing high traffic, another instance handles the requests.

- For information on clustering, see [Clustering WSO2 API Manager](#).
- For information on deployment patterns, see [Deployment Patterns of WSO2 API Manager](#).

Tuning Performance

This section describes some recommended performance tuning configurations to optimize the API Manager. It assumes that you have set up the API Manager on Unix/Linux, which is recommended for a production deployment. We also recommend a [distributed API Manager setup](#) for most production systems. Out of all components of an API Manager distributed setup, the API Gateway is the most critical, because it handles all inbound calls to APIs. Therefore, we recommend you to have at least a 2-node cluster of API Gateways in a distributed setup.

- [OS-level settings](#)
- [JVM-level settings](#)
- [APIM-level settings](#)



Important:

- Performance tuning requires you to modify important system files, which affect all programs running on the server. We recommend you to familiarize yourself with these files using Unix/Linux documentation before editing them.
- The values we discuss here are general recommendations. They might not be the optimal values for the specific hardware configurations in your environment. We recommend you to carry out load tests on your environment to tune the API Manager accordingly.

OS-level settings

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.



It is not recommended to use `net.ipv4.tcp_tw_recycle = 1` when working with network address translation (NAT), such as if you are deploying products in EC2 or any other environment configured with NAT.

```

net.ipv4.tcp_fin_timeout = 30
fs.file-max = 2097152
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.core.rmem_default = 524288
net.core.wmem_default = 524288
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.ipv4.ip_local_port_range = 1024 65535

```

2. To alter the number of allowed open files for system users, configure the following settings in /etc/security/limits.conf file of Linux (be sure to include the leading * character).

```

* soft nofile 4096
* hard nofile 65535

```

Optimal values for these parameters depend on the environment.

3. To alter the maximum number of processes your user is allowed to run at a given time, configure the following settings in /etc/security/limits.conf file of Linux (be sure to include the leading * character). Each carbon server instance you run would require upto 1024 threads (with default thread pool configuration). Therefore, you need to increase the nproc value by 1024 per each carbon server (both hard and soft).

```

* soft nproc 20000
* hard nproc 20000

```

JVM-level settings

If one or more worker nodes in a clustered deployment require access to the management console, increase the entity expansion limit as follows in the <APIM_HOME>/bin/wso2server.bat file (for Windows) or the <APIM_HOME>/bin/wso2server.sh file (for Linux/Solaris). The default entity expansion limit is 64000.

```
-DentityExpansionLimit=10000
```

APIM-level settings

Improvement Area	Performance Recommendations
API Gateway nodes	Increase memory allocated by modifying /bin/wso2server.sh with the following setting: <ul style="list-style-type: none"> -Xms2048m -Xmx2048m -XX:MaxPermSize=1024m

NHTTP transport of API Gateway	<p>Recommended values for <AM_HOME>/repository/conf/nhttp.properties file are given below. Note that the commented out values in this file are the default values that will be applied if you do not change anything.</p> <p>Property descriptions:</p> <table border="1" data-bbox="339 333 1498 728"> <tr> <td data-bbox="339 333 551 382">snd_t_core</td><td data-bbox="551 333 1498 382">Transport sender worker pool's initial thread count</td></tr> <tr> <td data-bbox="339 382 551 430">snd_t_max</td><td data-bbox="551 382 1498 430">Transport sender worker pool's maximum thread count</td></tr> <tr> <td data-bbox="339 430 551 614">snd_io_threads</td><td data-bbox="551 430 1498 614">Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.</td></tr> <tr> <td data-bbox="339 614 551 663">snd_alive_sec</td><td data-bbox="551 614 1498 663">Sender-side keep-alive seconds</td></tr> <tr> <td data-bbox="339 663 551 728">snd_qlen</td><td data-bbox="551 663 1498 728">Sender queue length, which is infinite by default</td></tr> </table> <p>Recommended values:</p> <pre># HTTP Sender thread pool parameters • snd_t_core=200 • snd_t_max=250 • snd_alive_sec=5 • snd_qlen=-1 • snd_io_threads=16 # HTTP Listener thread pool parameters • lst_t_core=200 • lst_t_max=250 • lst_alive_sec=5 • lst_qlen=-1 • lst_io_threads=16 #timeout parameters • http.socket.timeout.receiver: Recommended socket timeout for listener is 120000 • http.socket.timeout.sender: Recommended socket timeout for sender is 120000</pre>	snd_t_core	Transport sender worker pool's initial thread count	snd_t_max	Transport sender worker pool's maximum thread count	snd_io_threads	Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.	snd_alive_sec	Sender-side keep-alive seconds	snd_qlen	Sender queue length, which is infinite by default
snd_t_core	Transport sender worker pool's initial thread count										
snd_t_max	Transport sender worker pool's maximum thread count										
snd_io_threads	Sender-side IO workers, which is recommended to be equal to the number of CPU cores. I/O reactors usually employ a small number of dispatch threads (often as few as one) to dispatch I/O event notifications to a greater number (often as many as several thousands) of I/O sessions or connections. Generally, one dispatch thread is maintained per CPU core.										
snd_alive_sec	Sender-side keep-alive seconds										
snd_qlen	Sender queue length, which is infinite by default										

PassThrough transport of API Gateway	<p>Recommended values for <code><AM_HOME>/repository/conf/passthru-http.properties</code> file are given below. Note that the commented out values in this file are the default values that will be applied if you do not change anything.</p> <p>Property descriptions</p> <table border="1" data-bbox="339 327 1498 728"> <tr> <td data-bbox="339 327 763 411">worker_thread_keepalive_sec</td><td data-bbox="763 327 1498 411">Defines the keep-alive time for extra threads in the worker pool</td></tr> <tr> <td data-bbox="339 411 763 496">worker_pool_queue_length</td><td data-bbox="763 411 1498 496">Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool</td></tr> <tr> <td data-bbox="339 496 763 580">io_threads_per_reactor</td><td data-bbox="763 496 1498 580">Defines the number of IO dispatcher threads used per reactor</td></tr> <tr> <td data-bbox="339 580 763 644">http.max.connection.per.host.port</td><td data-bbox="763 580 1498 644">Defines the maximum number of connections per host port</td></tr> <tr> <td data-bbox="339 644 763 728">worker_pool_queue_length</td><td data-bbox="763 644 1498 728">Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.</td></tr> </table> <p>Recommended values</p> <ul style="list-style-type: none"> • <code>worker_thread_keepalive_sec</code>: Default value is 60s. This should be less than the socket timeout. • <code>worker_pool_queue_length</code>: Set to -1 to use an unbounded queue. If a bound queue is used and the queue gets filled to its capacity, any further attempts to submit jobs will fail, causing some messages to be dropped by Synapse. The thread pool starts queuing jobs when all the existing threads are busy and the pool has reached the maximum number of threads. So, the recommended queue length is -1. • <code>io_threads_per_reactor</code>: Value is based on the number of processor cores in the system (<code>Runtime.getRuntime().availableProcessors()</code>) • <code>http.max.connection.per.host.port</code> : Default value is 32767, which works for most systems but you can tune it based on your operating system (for example, Linux supports 65K connections) • <code>worker_pool_size_core</code>: 400 • <code>worker_pool_size_max</code>: 500 • <code>io_buffer_size</code>: 16384 • <code>http.socket.timeout</code>: 60000 • <code>snd_t_core</code>: 200 • <code>snd_t_max</code>: 250 • <code>snd_io_threads</code>: 16 • <code>lst_t_core</code>: 200 • <code>lst_t_max</code>: 250 • <code>lst_io_threads</code>: 16 <p>Make the number of threads equal to the number of processor cores.</p>	worker_thread_keepalive_sec	Defines the keep-alive time for extra threads in the worker pool	worker_pool_queue_length	Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool	io_threads_per_reactor	Defines the number of IO dispatcher threads used per reactor	http.max.connection.per.host.port	Defines the maximum number of connections per host port	worker_pool_queue_length	Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.
worker_thread_keepalive_sec	Defines the keep-alive time for extra threads in the worker pool										
worker_pool_queue_length	Defines the length of the queue that is used to hold runnable tasks to be executed by the worker pool										
io_threads_per_reactor	Defines the number of IO dispatcher threads used per reactor										
http.max.connection.per.host.port	Defines the maximum number of connections per host port										
worker_pool_queue_length	Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.										

Time-out configurations	<p>The API Gateway routes the requests from your client to an appropriate endpoint. The most common reason for your client getting a timeout is when the Gateway's timeout is larger than the client's timeout values. You can resolve this by either increasing the timeout on the client's side or by decreasing it on the API Gateway's side.</p> <p>Here are few parameters, in addition to the timeout parameters discussed in the previous sections.</p> <table border="1" data-bbox="326 375 1512 1134"> <tr> <td data-bbox="326 375 747 671">synapse.global_timeout_interval</td><td data-bbox="747 375 1512 671"> <p>Defines the maximum time that a callback is waiting in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in Gateway.</p> <p>Global timeout is defined in the <code><APIM_HOME>/repository/conf/synapse.properties</code> file. Recommended value is 120000.</p> </td></tr> <tr> <td data-bbox="326 671 747 1134">Endpoint-level timeout</td><td data-bbox="747 671 1512 1134"> <p>You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.</p> <p>The example below sets the endpoint to 30 seconds and executes the fault handler in case of a timeout.</p> <pre data-bbox="799 887 1452 1077"> <timeout> <duration>10000</duration> <responseAction>fault</responseAction> </timeout> </pre> </td></tr> </table>	synapse.global_timeout_interval	<p>Defines the maximum time that a callback is waiting in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in Gateway.</p> <p>Global timeout is defined in the <code><APIM_HOME>/repository/conf/synapse.properties</code> file. Recommended value is 120000.</p>	Endpoint-level timeout	<p>You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.</p> <p>The example below sets the endpoint to 30 seconds and executes the fault handler in case of a timeout.</p> <pre data-bbox="799 887 1452 1077"> <timeout> <duration>10000</duration> <responseAction>fault</responseAction> </timeout> </pre>
synapse.global_timeout_interval	<p>Defines the maximum time that a callback is waiting in the Gateway for a response from the backend. If no response is received within this time, the Gateway drops the message and clears out the callback. This is a global level parameter that affects all the endpoints configured in Gateway.</p> <p>Global timeout is defined in the <code><APIM_HOME>/repository/conf/synapse.properties</code> file. Recommended value is 120000.</p>				
Endpoint-level timeout	<p>You can define timeouts per endpoint for different backend services, along with the action to be taken in case of a timeout.</p> <p>The example below sets the endpoint to 30 seconds and executes the fault handler in case of a timeout.</p> <pre data-bbox="799 887 1452 1077"> <timeout> <duration>10000</duration> <responseAction>fault</responseAction> </timeout> </pre>				

Key management nodes

Set the following in <APIM_HOME>/repository/conf/axis2/axis2_client.xml file:

```
<parameter name="defaultMaxConnPerHost">1000</parameter>
<parameter name="maxTotalConnections">30000</parameter>
```

Set the MySQL maximum connections:

```
mysql> show variables like "max_connections";
max_connections was 151
set to global max_connections = 250;
```

Set the open files limit to 200000 by editing the /etc/sysctl.conf file:

```
sudo sysctl -p
```

Set the following in <APIM_HOME>/repository/conf/tomcat/catalina-server.xml file.

```
maxThreads="750"
minSpareThreads="150"
disableUploadTimeout="false"
enableLookups="false"
connectionUploadTimeout="120000"
maxKeepAliveRequests="600"
acceptCount="600"
```

Set the following connection pool elements in <APIM_HOME>/repository/conf/datasource/master-datasources.xml file:

```
<maxActive>50</maxActive>
<maxWait>60000</maxWait>
<testOnBorrow>true</testOnBorrow>
<validationQuery>SELECT 1</validationQuery>
<validationInterval>30000</validationInterval>
```

Note that you set the <testOnBorrow> element to true and provide a validation query (e.g., in Oracle, SELECT 1 FROM DUAL), which is run to refresh any stale connections in the connection pool. Set a suitable value for the <validationInterval> element, which defaults to 3000 milliseconds. It determines the time period after which the next iteration of the validation query will be run on a particular connection. It avoids excess validations and ensures better performance.

Reference Guide

The following topics provide reference information for working with WSO2 API Manager:

- Product Profiles
- Default Product Ports
- Changing the Default Ports with Offset
- Error Handling
- WSO2 Patch Application Process

Product Profiles

When a WSO2 product starts, it starts all features and related artifacts bundled with it. Multi-profile support allows you to run the product on a selected profile so that only features specific to that profile along with common features start up with the server. This enables better resource utilization.

Given below are the different profiles available in WSO2 API Manager.

Profile	Command Option with Profile Name	Description
Gateway manager	-Dprofile=gateway-manager	Used when the API Gateway acts as a manager node in a cluster. This profile starts frontend/UI features such as login as well as backend services that allow the product instance to communicate with other nodes in the cluster.
Gateway worker	-Dprofile=gateway-worker	Used when API Gateway acts as a worker node in a cluster. This profile only starts the backend features for data processing and communicating with the manager node.
Key Manager	-Dprofile=api-key-manager	Starts only the features relevant to the Key Manager component of API Manager.
API Publisher	-Dprofile=api-publisher	Starts only the front end/backend features relevant to the API Publisher Web interface.
API Store	-Dprofile=api-store	Starts only the front end/backend features relevant to the API Store Web interface.

 Note that the WSO2 products platform currently doesn't block/allow Web applications depending on profiles. Starting a product on a preferred profile only blocks/allows the relevant OSGI bundles. As a result, even if you start the server on a profile such as the `api-store` for example, you will still be able to access the API Publisher Web application.

Execute the following commands to start a product on any profile:

OS	Command
Windows	<code><PRODUCT_HOME>/bin/wso2server.bat -Dprofile=<preferred-profile> --run</code>
Linux/Solaris	<code>sh <PRODUCT_HOME>/bin/wso2server.sh -Dprofile=<preferred-profile></code>

How multi-profiling works

Starting a product on a preferred profile starts only a subset of features bundled in the product. In order to identify what feature bundles apply to which profile, each product maintains a set of `bundles.info` files in `<PRODUCT_HOME>/repository/components/<profile-name>/configuration/org.eclipse.equinox.simpleconfigurator` directories. The `bundles.info` files contain references to the actual bundles. Note that `<profile-name>` in the directory path refers to the name of the profile. For example, when there's a product profile named `webapp`, references to all the feature bundles required for `webapp` profile to function are in a `bundles.info` file

saved in <PRODUCT_HOME>/repository/components/**webapp**/configuration/org.eclipse.equinox.simpleconfigurator directory.

Note that when you start the server without using a preferred profile, the server refers to <PRODUCT_HOME>/repository/components/**default**/configuration/org.eclipse.equinox.simpleconfigurator/bundles.info file by default. This file contains references to all bundles in <PRODUCT_HOME>/repository/components/plugins directory, which is where all components/bundles of a product are saved.

Default Product Ports

This page describes the default ports that are used for each WSO2 product when the port offset is 0.

 **Note** that it is recommended to disable the HTTP transport in an API Manager production setup. Using the bearer token over HTTP is a violation of the OAuth specification and can lead to security vulnerabilities.

- 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 - RMIServer 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](#) | [Complex Event Processor](#) | [Elastic Load Balancer](#) | [ESB](#) | [Identity Server](#) | [Message Broker](#) | [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 [APIM 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)

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.

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 GCM (Google Cloud Message) and APNS (Apple Push Notification Service) and enroll to WSO2 EMM.

Android :

The ports to open are 5228, 5229 and 5230. GCM typically only uses 5228, but it sometimes uses 5229 and 5230. GCM 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 Wifi 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

Changing the Default Ports with Offset

When you run multiple WSO2 products/clusters or multiple instances of the same product on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts. An offset defines the number by which all ports in the runtime (e.g., HTTP/S ports) will be increased. For example, if the default HTTP port is 9763 and the offset is 1, the effective HTTP port will change to 9764. For each additional WSO2 product instance, you set the port offset to a unique value. The offset of the default ports is considered to be 0.

There are two ways to set an offset to a port:

- Pass the port offset to the server during startup. The following command starts the server with the default port incremented by 3: `./wso2server.sh -DportOffset=3`
- Set the Ports section of `<PRODUCT_HOME>/repository/conf/carbon.xml`. E.g., `<Offset>3</Offset>`

Usually, when you offset the server's port, it automatically changes all ports it uses. However, there are few exceptions in the API Manager where you have to manually adjust some ports.

Changing the Thrift client and server ports

The port offset specified earlier in `carbon.xml` does not affect the ports of the Thrift client and server because Thrift is run as a separate server within WSO2 servers. Therefore, you must change the Thrift ports separately using `<ThriftClientPort>` and `<ThriftServerPort>` elements in the `<APIM_HOME>/repository/conf/api-manager.xml` file. For example, the following configuration sets an offset of 2 to the default Thrift port, which is 10397:

```

<!--
    Configurations related to enable thrift support for key-management related
    communication.

    If you want to switch back to Web Service Client, change the value of
    "KeyValidatorClientType" to "WSClient".
    In a distributed environment;
    -If you are at the Gateway node, you need to point "ThriftClientPort" value to
    the "ThriftServerPort" value given at KeyManager node.
    -If you need to start two API Manager instances in the same machine, you need
    to give different ports to "ThriftServerPort" value in two nodes.
    -ThriftServerHost - Allows to configure a hostname for the thrift server. It
    uses the carbon hostname by default.
    -->

    <KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
    <ThriftClientPort>10399</ThriftClientPort>
    <ThriftClientConnectionTimeOut>10000</ThriftClientConnectionTimeOut>
    <ThriftServerPort>10399</ThriftServerPort>
    <!--ThriftServerHost>localhost</ThriftServerHost-->
    <EnableThriftServer>true</EnableThriftServer>

```

When you run multiple instances of the API Manager in distributed mode, the Gateway and Key Manager (used for validation and authentication) can run on two different JVMs. Communication between API Gateway and Key Manager happens in either of the following ways:

- Through a Web service call
- Through a Thrift call

The default communication mode is using Thrift. Assume that the Gateway port is offset by 2, Key Manager port by 5 and the default Thrift port is 10397. If the Thrift ports are changed by the offsets of Gateway and Key Manager, the Thrift client port (Gateway) will now be 10399 while the Thrift server port (Key Manager) will change to 10402. This causes communication between the Gateway and Key Manager to fail because the Thrift client and server ports are different.

To fix this, you must change the Thrift client and server ports of Gateway and Key Manager to the same value. In this case, the difference between the two offsets is 3, so you can either increase the default Thrift client port by 3 or else reduce the Thrift server port by 3.

Changing the offset of the Workflow Callback Service

The API Manager has a service that listens to workflow callbacks. This service configuration is in <AM_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/WorkflowCallbackService.xml. Change the port value of the <address uri> element. For example,

```
<address
uri="https://localhost:9445/store/site/blocks/workflow/workflow-listener/ajax/workflow
-listener.jag" format="rest"/>
```

For a list of all default ports opened in WSO2 API Manager, see [Default Product Ports](#).

Error Handling

When errors/exception occur in the system, the API Manager throws XML-based error responses by default. To change the format of the error response that is sent to the client, you change the auth failure handler in the <AM_HOME>/repository/deployment/server/synapse-configs/default/sequences/_auth_failure_handler.xml file. Given below is the default configuration:

```
<sequence name="auth_failure_handler"> <property name="error_message_type"
value="application/xml"/> <sequence key="build"/> </sequence>
```

If you change application/xml to something like **application/json**, the error response will be sent in JSON format.

Given below are some error codes and their meanings.

API handlers error codes

Error code	Error Message	Description
900900	Unclassified Authentication Failure	An unspecified error has occurred
900901	Invalid Credentials	Invalid Authentication information provided
900902	Missing Credentials	No authentication information provided
900903	Access Token Expired	Access Token has expired. Renew the access token .
900904	Access Token Inactive	Access token has become inactive. Generate new access token .
900905	Incorrect Access Token Type is provided	The access token type used is not supported when invoking the API. The supported access token types are Application Accesses Token and User Accesses Token. See Access Tokens .
900906	No matching resource found in the API for the given request	A resource with the name in the request can not be found in the API.

900907	The requested API is temporarily blocked	The status of the API has been changed to an inaccessible/unavailable state.
900908	Resource forbidden	The user invoking the API has not been granted access to the required resource.
900909	The subscription to the API is inactive	Happens when the API user is blocked.
900910	The access token does not allow you to access the requested resource	Can not access the required resource with the provided access token. Check the valid resources that can be accessed with this token.
900800	Message throttled out	The maximum number of requests that can be made to the API within a designated time period is reached and the API is throttled for the user.
700700	API blocked	This API has been blocked temporarily. Please try again later or contact the system administrators.

Sequences error codes

Error code	Description
900901	Production/sandbox key offered to the API with no production/sandbox endpoint
403	No matching resource found in the API for the given request

In addition to the above error codes, we have engaged Synapse-level error codes to the default fault sequence and custom fault sequences (e.g.,_token_fault_.xml) of the API Manager. For information, see [Error Handling](#) in WSO2 ESB documentation.

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



Before you begin

- You can download all WSO2 Carbon Kernel patches from [here](#).
- Before you apply a patch, check its `README.txt` file for any configuration changes required.

Applying service packs to the product

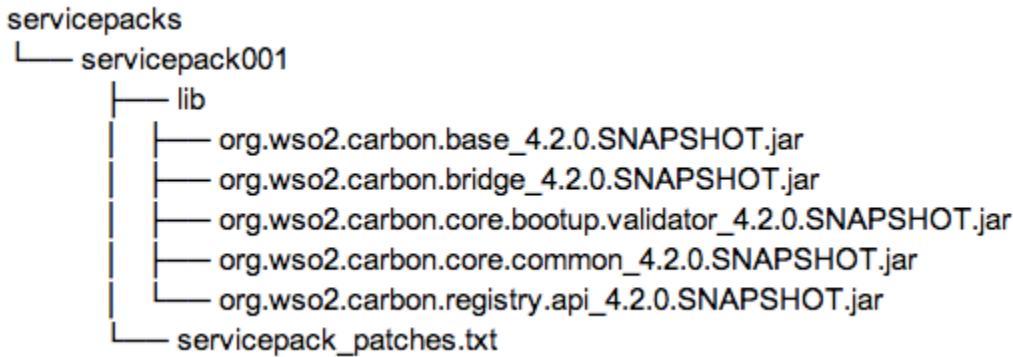
Carbon 4.2.0 Kernel supports service packs. A service pack is a collection of patches in a single pack. It contains two elements:

- 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/` director

y. For example, the image below shows how a new service pack named `servicepack001` is added to this directory.



2. Start your product. The following steps will be executed:

- 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.
- 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.
- 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.

i 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.

i 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/.metadata/prePatchedJARs.txt` meta file 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.

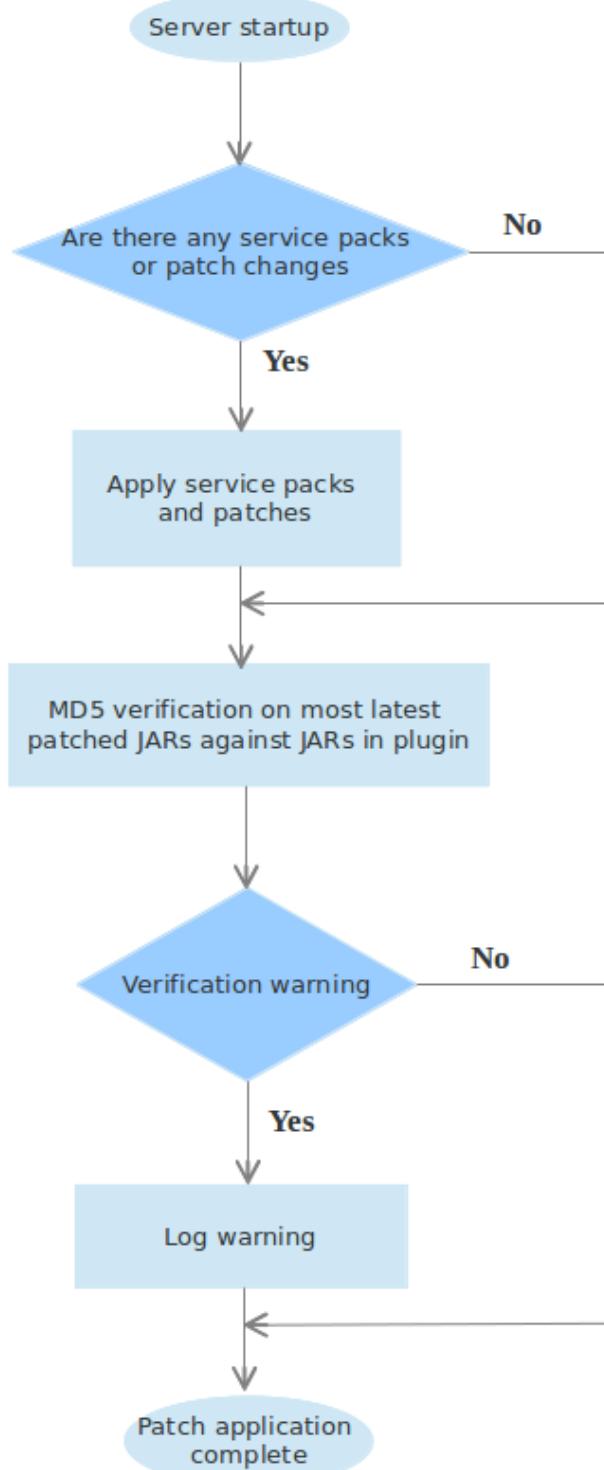
Overview of the patch application process

The diagram below shows how the patch application process is implemented when you start the server.

FAQ

- About WSO2 API Manager
- What is WSO2 API Manager?
- What is the open source license of the API Manager?
- How do I download and get started quickly?
- Is there commercial support available for WSO2 API Manager?
- What are the default ports opened in the API Manager?
- What are the technologies used underneath WSO2 API Manager?
- Can I get involved in APIM development activities?
- Does the API Manager use Thrift and where can I find information about it?
- Installation
- What are the minimum requirements to run WSO2 API Manager?
- What Java versions are supported by the API Manager?
- How do I deploy a third-party library into the API Manager?
- Do you provide automated installation scripts based on Puppet or similar solutions?
- Is it possible to connect the API Manager directly to an LDAP or Active Directory where the corporate identities are stored?
- Can I extend the management console UI to add custom UIs?
- I don't want some of the features that come with WSO2 API Manager. Can I remove them?
- How can I change the memory allocation for the API Manager?
- Clustering and deployment
- Where can I look up details of different deployment patterns and clustering configurations of the API Manager?
- What is the recommended way to manage multiple artifacts in a product cluster?
- Is it recommended to run multiple WSO2 products on a single server?
- Can I install features of other WSO2 products to the API Manager?
- How can I set up a reverse proxy server to pass server requests?
- Functionality

- I cannot see all the APIs that I published on the API Store. Why is this?
- When editing an API's resource's parameters, how can I add multiple options to the parameter Response Content Type?
- I edited the resource parameter Response Content Type of a published API. But the changes are not reflected in the API Store after saving. What should I do?
- I have set up the API Manager with WSO2 BAM to collect and analyze runtime statistics. But, the 'API Usage by Destination' graph shows no data. Why is this?
- Authentication and security



- How can I manage authentication centrally in a clustered environment?
- How can I manage the API permissions/visibility?
- How can I add security policies (UT, XACML etc.) for the services?
- How can I disable self signup capability to the API Store? I want to engage my own approval mechanism.
- Is there a way to lock a user's account after a certain number of failed login attempts to the API Store?
- How do I change the default admin password and what files should I edit after changing it?
- How can I recover the admin password used to log in to the management console?
- Can I give special characters in the passwords that appear in the configuration files?
- Troubleshooting
 - Why do I get the following warning:
org.wso2.carbon.server.admin.module.handler.AuthenticationHandler - Illegal access attempt while trying to authenticate APIKeyValidationService?
 - I hit the DentityExpansionLimit and it gives an error as
{org.wso2.carbon.apimgt.hostobjects.APIStoreHostObject} - Error while getting Recently Added APIs Information. What is the cause of this?
 - I get a Hostname verification failed exception when trying to send requests to a secured endpoint. What should I do?
 - When I add new users or roles, I get an error message as 'Entered user name is not conforming to policy'. What should I do?
 - The access token I generated from the My Subscriptions page does not work but I can invoke the API via the Token API. What can I do?
 - When I call a REST API, I find that a lot of temporary files are created in my server and they are not cleared. This takes up a lot of space. What should I do?
- General questions
 - Can I implement an API facade with the API Manager?
 - How can I write automated test scripts for the API Manager?

About WSO2 API Manager

What is WSO2 API Manager?

WSO2 API Manager is a complete solution for creating, publishing and managing all aspects of an API and its life cycle. See [About API Manager](#).

What is the open source license of the API Manager?

Apache Software License Version 2.0

How do I download and get started quickly?

Go to <http://wso2.com/products/api-manager> to download the binary or source distributions. See [Getting Started](#).

Is there commercial support available for WSO2 API Manager?

It is completely supported from evaluation to production. See [WSO2 Support](#).

What are the default ports opened in the API Manager?

See [Default Ports of WSO2 Products](#).

What are the technologies used underneath WSO2 API Manager?

The API Manager is built on top of [WSO2 Carbon](#), an OSGi based components framework for SOA. See [component S](#).

Can I get involved in APIM development activities?

Not only are you allowed, but also encouraged. You can start by subscribing to dev@wso2.org and architecture@wso2.org mailing lists. Feel free to provide ideas, feedback and help make our code better. For more information on

contacts, mailing lists and forums, see [Getting Support](#).

Does the API Manager use Thrift and where can I find information about it?

That the default communication protocol of Key Manager is Thrift. See <http://thrift.apache.org/static/files/thrift-20070401.pdf> for information on Thrift.

Installation

What are the minimum requirements to run WSO2 API Manager?

Minimum requirement is Oracle Java SE Development Kit (JDK). See [Installation Prerequisites](#).

What Java versions are supported by the API Manager?

See [Installation Prerequisites](#).

How do I deploy a third-party library into the API Manager?

Copy any third-party JARs to <APIM_HOME>/repository/components/lib directory and restart the server.

Do you provide automated installation scripts based on Puppet or similar solutions?

Yes. For information, [contact us](#).

Is it possible to connect the API Manager directly to an LDAP or Active Directory where the corporate identities are stored?

Yes. You can configure the API Manager with multiple user stores. See [Configuring User Stores](#).

Can I extend the management console UI to add custom UIs?

Yes, you can extend the management console (default URL is <https://localhost:9443/carbon>) easily by writing a custom UI component and simply deploying the OSGi bundle.

I don't want some of the features that come with WSO2 API Manager. Can I remove them?

Yes, you can do this using the **Features** menu under the **Configure** menu of the management console (default URL is <https://localhost:9443/carbon>).

How can I change the memory allocation for the API Manager?

The memory allocation settings are in <APIM_HOME>/bin/wso2server.sh file.

Clustering and deployment

Where can I look up details of different deployment patterns and clustering configurations of the API Manager?

See [WSO2 clustering and deployment guide](#).

What is the recommended way to manage multiple artifacts in a product cluster?

For artifact governance and lifecycle management, we recommend you to use a shared [WSO2 Governance Registry](#) instance.

Is it recommended to run multiple WSO2 products on a single server?

This is not recommended in a production environment involving multiple transactions. If you want to start several WSO2 products on a single server, you must change their default ports to avoid port conflicts. See [Changing the Default Ports with Offset](#).

Can I install features of other WSO2 products to the API Manager?

Yes, you can do this using the management console. The API Manager already has features of WSO2 Identity Server, WSO2 Governance Registry, WSO2 ESB etc. embedded in it. However, if you require more features of a certain product, it is recommended to use a separate instance of it rather than install its features to the API Manager.

How can I set up a reverse proxy server to pass server requests?

See [Adding a Reverse Proxy Server](#).

Functionality***I cannot see all the APIs that I published on the API Store. Why is this?***

If you have multiple versions of an API published, only the latest version is shown in the API Store. To display multiple versions, set the `<DisplayMultipleVersions>` element to `true` in `<APIM_HOME>/repository/conf/api-manager.xml` file.

When editing an API's resource's parameters, how can I add multiple options to the parameter Response Content Type?

You cannot do this using the UI. Instead, edit the Swagger definition of the API as `content_type: ["text/xml", "text/plain"]` for example.

I edited the resource parameter Response Content Type of a published API. But the changes are not reflected in the API Store after saving. What should I do?

If you edited the **Response Content Type** using the UI, please open the API's Swagger definition, do your changes and save. Then the changes should be reflected back in the API Store. This will be fixed in a future release.

I have set up the API Manager with WSO2 BAM to collect and analyze runtime statistics. But, the 'API Usage by Destination' graph shows no data. Why is this?

To populate this graph, you must enable destination-based usage tracking manually. See [Viewing API Statistics](#) on how to do that.

Authentication and security***How can I manage authentication centrally in a clustered environment?***

You can enable centralized authentication using a WSO2 Identity Server based security and identity gateway solution, which [enables SSO](#) (Single Sign On) across all the servers.

How can I manage the API permissions/visibility?

To set visibility of the API only to selected user roles in the server, see [API Visibility](#).

How can I add security policies (UT, XACML etc.) for the services?

This should be done in the backend services in the Application Server or WSO2 ESB.

How can I disable self signup capability to the API Store? I want to engage my own approval mechanism.

To disable the self signup capability, open the APIM management console and click the **Resources -> Browse** menu. The registry opens. Navigate to `/_system/governance/apimgt/applicationdata/sign-up-config.xml` and set `<SelfSignUp><Enabled>` element to false. To engage your own signup process, see [Adding a User Signup Workflow](#).

Is there a way to lock a user's account after a certain number of failed login attempts to the API Store?

If your identity provider is WSO2 Identity Server, this facility comes out of the box. If not, install the identity-mgt feature to the API Manager and configure it. For information, see [Account Lock/Unlock](#) page in the Identity Server documentation.

How do I change the default admin password and what files should I edit after changing it?

To change the default admin password, log in to the management console with admin/admin credentials and use the "Change my password" option. After changing the password, change the following elements in <APIM_HOME>/repository/conf/api-manager.xml file:

```

<AuthManager>
  <Username>admin</Username>
  <Password>newpassword</Password>
</AuthManager>

<APIGateway>
  <Username>admin</Username>
  <Password>newpassword</Password>
</APIGateway>

<APIKeyManager>
  <Username>admin</Username>
  <Password>newpassword</Password>
</APIKeyManager>

```

How can I recover the admin password used to log in to the management console?

Use <APIM_HOME>/bin/chpasswd.sh script.

Can I give special characters in the passwords that appear in the configuration files?

If the config file is in XML, take care when giving special characters in the user names and passwords. According to XML specification (<http://www.w3.org/TR/xml/>), some special characters can disrupt the configuration. For example, the ampersand character (&) must not appear in the literal form in XML files. It can cause a Java Null Pointer exception. You must wrap it with CDATA (http://www.w3schools.com/xml/xml_cdata.asp) as shown below or remove the character:

```

<Password>
  <![CDATA[ xnvYh?@VHAkc?qZ%Jv855&A4a,%M8B@h ]>
</Password>

```

Troubleshooting***Why do I get the following warning: org.wso2.carbon.server.admin.module.handler.AuthenticationHandler - Illegal access attempt while trying to authenticate APIKeyValidationService?***

- Did you change the default admin password? If so, you need to change the credentials stored in the <APIKeyManager> element of the <APIM_HOME>/repository/conf/api-manager.xml file of the API Gateway node/s.
- Have you set the priority of the SAML2SSOAuthenticator handler higher than that of the BasicAuthenticator handler in the authenticators.xml file? If so, the SAML2SSOAuthenticator handler tries to manage the basic authentication requests as well. Set a lower priority to the SAML2SSOAuthenticator than the BasicAuthenticator handler as follows:

```

<Authenticator name="SAML2SSOAuthenticator" disabled="false">
    <Priority>0</Priority>
    <Config>
        <Parameter name="LoginPage">/carbon/admin/login.jsp</Parameter>
        <Parameter name="ServiceProviderID">carbonServer</Parameter>
        <Parameter
name="IdentityProviderSSOServiceURL">https://localhost:9444/samlss</Parameter>
        <Parameter
name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</
Parameter>
        <Parameter name="ISAuthnReqSigned">false</Parameter>
        <!--<Parameter
name="AssetionConsumerServiceURL">https://localhost:9443/acs</Parameter>-->
    </Config>
</Authenticator>

```

I hit the `DentityExpansionLimit` and it gives an error as `{org.wso2.carbon.apimgt.hostobjects.APIStoreHostObject}` - Error while getting Recently Added APIs Information. What is the cause of this?

This error occurs in JDK 1.7.0_45 and is fixed in JDK 1.7.0_51 onwards. See [here](#) for details of the bug.

In JDK 1.7.0_45, all XML readers share the same `XMLSecurityManager` and `XMLLimitAnalyzer`. When the total count of all readers hits the entity expansion limit, which is 64000 by default, the `XMLLimitAnalyzer`'s total counter is accumulated and the `XMLInputFactory` cannot create more readers. If you still want to use update 45 of the JDK, try restarting the server with a higher value assigned to the `DentityExpansionLimit`.

I get a Hostname verification failed exception when trying to send requests to a secured endpoint. What should I do?

Set the `<parameter name="HostnameVerifier">` element to `AllowAll` in `<APIM_HOME>/repository/conf/axis2/axis2.xml` file's `HTTPS` transport sender configuration. For example, `<parameter name="HostnameVerifier">AllowAll</parameter>`.

This parameter verifies the hostname of the certificate of a server when the API Manager acts as a client and does outbound service calls.

When I add new users or roles, I get an error message as 'Entered user name is not conforming to policy'. What should I do?

This is because your user name or password length or any other parameter is not conforming to the `RegEx` configurations of the user store. See [Managing Users and Roles](#).

The access token I generated from the My Subscriptions page does not work but I can invoke the API via the Token API. What can I do?

Are you trying to invoke an API that is protected by OAuth scopes? If so, the tokens generated from the [My Subscriptions](#) page do not work. You must use the [Token API](#).

When I call a REST API, I find that a lot of temporary files are created in my server and they are not cleared. This takes up a lot of space. What should I do?

There might be multiple configuration context objects created per each API invocation. Please check whether your client is creating a configuration context object per each API invocation. Also, configure a HouseKeeping task in the `<APIM_HOME>/repository/conf/carbon.xml` file to clear the temporary folders. For example.

```

<HouseKeeping>
    <AutoStart>true</AutoStart>

        <!-- The interval in *minutes*, between house-keeping runs -->
        <Interval>10</Interval>

        <!-- The maximum time in *minutes*, temp files are allowed to live in the
        system. Files/directories which were modified more than
        "MaxTempFileLifetime" minutes ago will be removed by the house-keeping task
        -->
        <MaxTempFileLifetime>30</MaxTempFileLifetime>
</HouseKeeping>

```

General questions

Can I implement an API facade with the API Manager?

You can use the API Manager and WSO2 ESB to implement an [API facade architecture pattern](#). WSO2 recommends this architecture if you are performing heavy mediation in your setup. For implementation details of an API facade, see [implementing an API facade with WSO2 API management platform](#).

As the API Manager does not have the ESB's GUI to perform mediation functions, you need to use the XML-based source view for configuration. Alternatively, you can create the necessary mediation sequences using the GUI of the ESB, and copy them from the ESB to the API Manager.

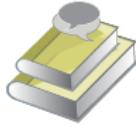
Also see [the following use cases](#) in WSO2 ESB documentation for more information on REST to SOAP conversion.

How can I write automated test scripts for the API Manager?

Use WSO2 Test Automation Framework (TAF) as explained in [Writing a Test Case for API Manager](#).

Getting Support

In addition to this documentation, there are several ways to get help as you work on WSO2 products.

	Explore learning resources: For tutorials, articles, whitepapers, webinars, and other learning resources, look in the Resources menu on the WSO2 website . 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.
	Try our support options: 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 http://wso2.com/support/ .
	Ask questions in the user forums at http://stackoverflow.com . Ensure that you tag your question with appropriate keywords such as <i>WSO2</i> 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 http://wso2.org/mail .
	Report issues , submit enhancement requests, track and comment on issues using our public bug-tracking system, and contribute samples, patches, and tips & tricks (see the WSO2 Contributor License Agreement).

Site Map

Use this site map to quickly find the topic you're 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.