



API Manager

Documentation

WSO2 API Manager

Documentation

Version 2.x.x

Table of Contents

1. WSO2 API Manager Documentation	8
1.1 Introduction	8
1.1.1 Overview	9
1.1.2 About this Release	9
1.2 Quick Start Guide	10
1.3 Key Concepts	47
1.4 Tutorials	70
1.4.1 Getting Started	71
1.4.1.1 Create and Publish an API	71
1.4.1.2 Subscribe to an API	80
1.4.1.3 Invoke an API using the Integrated API Console	87
1.4.1.4 Edit an API Using the Swagger UI	91
1.4.2 API Publishing	100
1.4.2.1 Add API Documentation	101
1.4.2.1.1 Add API Documentation In-line, using a URL or a File	101
1.4.2.1.2 Add Apache Solr-Based Indexing	106
1.4.2.2 Manage the API Lifecycle	109
1.4.2.2.1 Create a New API Version	109
1.4.2.2.2 Deploy and Test as a Prototype	112
1.4.2.2.3 Publish the New Version and Deprecate the Old	118
1.4.2.2.4 Customize API Life Cycle	120
1.4.2.3 Publish to Multiple External API Stores	125
1.4.2.4 Publish through Multiple API Gateways	133
1.4.2.5 Block Subscription to an API	138
1.4.2.6 Enforce Throttling and Resource Access Policies	142
1.4.2.7 Change the Default Mediation Flow of API Requests	147
1.4.2.8 Map the Parameters of your Backend URLs with the API Publisher URLs	161
1.4.2.9 Convert a JSON Message to SOAP and SOAP to JSON	170
1.4.2.10 Invoke an API using a SOAP Client	183
1.4.2.11 Create and Publish an API from Swagger definition	189
1.4.2.12 Create a WebSocket API	198
1.4.2.13 Create a Prototyped API with an Inline Script	207
1.4.2.14 Pass a Custom Authorization Token to the Backend	211
1.4.2.15 Create and Publish a SOAP API	222
1.4.2.16 Disable Message Chunking	229
1.4.2.17 Enable API Indexing on Remote Publisher and Store Nodes	230
1.4.2.18 Remove Specific Request Headers From Response	230
1.4.2.19 Scope Management with OAuth Scopes	231
1.4.2.20 Generate REST API from SOAP Backend	231
1.4.3 Developer Portal	236
1.4.3.1 Include Additional Headers in the API Console	236
1.4.3.2 Log in to the API Store using Social Media	240
1.4.3.3 Test an API using a Testing Tool	251
1.4.3.4 Use the Community Features	262
1.4.3.5 Write a Client Application Using the SDK	266

1.4.3.6 Obtaining User Profile Information with OpenID Connect	272
1.4.3.7 Cleaning Up Partially Created Keys	274
1.5 Deep Dive	275
1.5.1 Installation Guide	276
1.5.1.1 Installation Prerequisites	276
1.5.1.2 Installing the Product	280
1.5.1.2.1 Installing on Linux or OS X	280
1.5.1.2.2 Installing on Solaris	282
1.5.1.2.3 Installing on Windows	283
1.5.1.2.4 Installing as a Linux Service	286
1.5.1.2.5 Installing as a Windows Service	288
1.5.1.3 Running the Product	294
1.5.1.4 Basic Health Checks	301
1.5.2 WSO2 API Manager Tooling	302
1.5.2.1 Installing the API Manager Tooling Plug-In	302
1.5.3 Product Administration	307
1.5.3.1 Upgrading from the Previous Release	316
1.5.3.1.1 Upgrading from the Previous Release when WSO2 IS is the Key Manager	329
1.5.3.2 Deploying WSO2 API Manager	332
1.5.3.2.1 Deployment Patterns	333
1.5.3.2.2 Deploying API Manager using Single Node Instances	344
1.5.3.2.3 Using Puppet Modules to Set up WSO2 API-M	368
1.5.3.2.4 Distributed Deployment of API Manager	539
1.5.3.2.5 Configuring WSO2 Identity Server as a Key Manager	578
1.5.3.2.6 Deploying API Manager with Kubernetes or OpenShift Resources	602
1.5.3.2.7 Configuring Admin App Event Publishing for Traffic Manager HA Setup	605
1.5.3.2.8 Minimum High Availability Deployment for WSO2 APIM Analytics	611
1.5.3.2.9 Configuring rsync for Deployment Synchronization	642
1.5.3.3 Changing the Default API-M Databases	644
1.5.3.4 Administration - API-M Analytics	660
1.5.3.4.1 Re-indexing Existing Data	660
1.5.3.5 Common Runtime and Configuration Artifacts	661
1.5.3.6 JMX Monitoring	662
1.5.4 Configuring the API Manager	667
1.5.4.1 Enabling CORS for APIs	667
1.5.4.2 Enabling Access Control Support for API Publisher	673
1.5.4.3 Adding Custom Properties to APIs	676
1.5.4.4 Customizing the API Store	678
1.5.4.5 Configuring Multiple Tenants	687
1.5.4.5.1 Managing Tenants	688
1.5.4.6 Adding Internationalization and Localization	690
1.5.4.7 Configuring Single Sign-on with SAML2	692
1.5.4.7.1 Configuring API Manager for SSO	692
1.5.4.7.2 Configuring External IDP through Identity Server for SSO	696
1.5.4.7.3 Configuring Identity Server as IDP for SSO	705
1.5.4.8 Changing the Default Transport	714
1.5.4.9 Configuring Caching	716
1.5.4.10 Prevent API Suspension	722

1.5.4.11 Working with Databases	724
1.5.4.11.1 Managing Datasources	724
1.5.4.12 Managing Users and Roles	734
1.5.4.12.1 Adding User Roles	735
1.5.4.12.2 Adding Users	738
1.5.4.13 Configuring User Stores	741
1.5.4.13.1 Realm Configuration	741
1.5.4.13.2 Changing the RDBMS	746
1.5.4.13.3 Configuring Primary User Stores	746
1.5.4.14 Directing the Root Context to the API Store	763
1.5.4.15 Adding Links to Navigate Between the Store and Publisher	763
1.5.4.16 Maintaining Separate Production and Sandbox Gateways	764
1.5.4.17 Configuring Transports	767
1.5.4.18 Transforming API Message Payload	768
1.5.4.19 Sharing Applications and Subscriptions	779
1.5.4.19.1 Sharing Applications Between Multiple Groups	781
1.5.4.20 Configuring API Monetization Category Labels	785
1.5.4.21 Enabling Notifications	787
1.5.4.22 Working with Access Tokens	790
1.5.4.23 Performance Tuning and Testing Results	794
1.5.4.23.1 Tuning Performance	794
1.5.4.23.2 WSO2 API-M Performance and Capacity Planning	808
1.5.4.24 Removing Unused Tokens from the Database	814
1.5.4.25 Migrating the APIs to a Different Environment	821
1.5.4.26 Generating SDKs	826
1.5.4.27 Revoke OAuth2 Application	831
1.5.4.28 Configuring Keystores in WSO2 API Manager	833
1.5.4.29 Logging	835
1.5.4.29.1 Application Logs	836
1.5.4.29.2 Setting Up Logging	838
1.5.4.29.3 System Logs	841
1.5.4.30 Message Tracing	843
1.5.4.31 Whitelisting Host Names for API Store	849
1.5.5 Extending the API Manager	850
1.5.5.1 Managing Workflow Extensions	850
1.5.5.1.1 Adding an Application Creation Workflow	851
1.5.5.1.2 Adding an Application Registration Workflow	855
1.5.5.1.3 Adding an API Subscription Workflow	862
1.5.5.1.4 Adding a User Signup Workflow	866
1.5.5.1.5 Invoking the API Manager from the BPEL Engine	869
1.5.5.1.6 Customizing a Workflow Extension	870
1.5.5.1.7 Configuring Workflows for Tenants	878
1.5.5.1.8 Configuring Workflows in a Cluster	888
1.5.5.1.9 Changing the Default User Role in Workflows	891
1.5.5.1.10 Cleaning Up Workflow Tasks	891
1.5.5.1.11 Adding an API State Change Workflow	893
1.5.5.2 Writing Custom Handlers	899
1.5.5.3 Adding Mediation Extensions	905

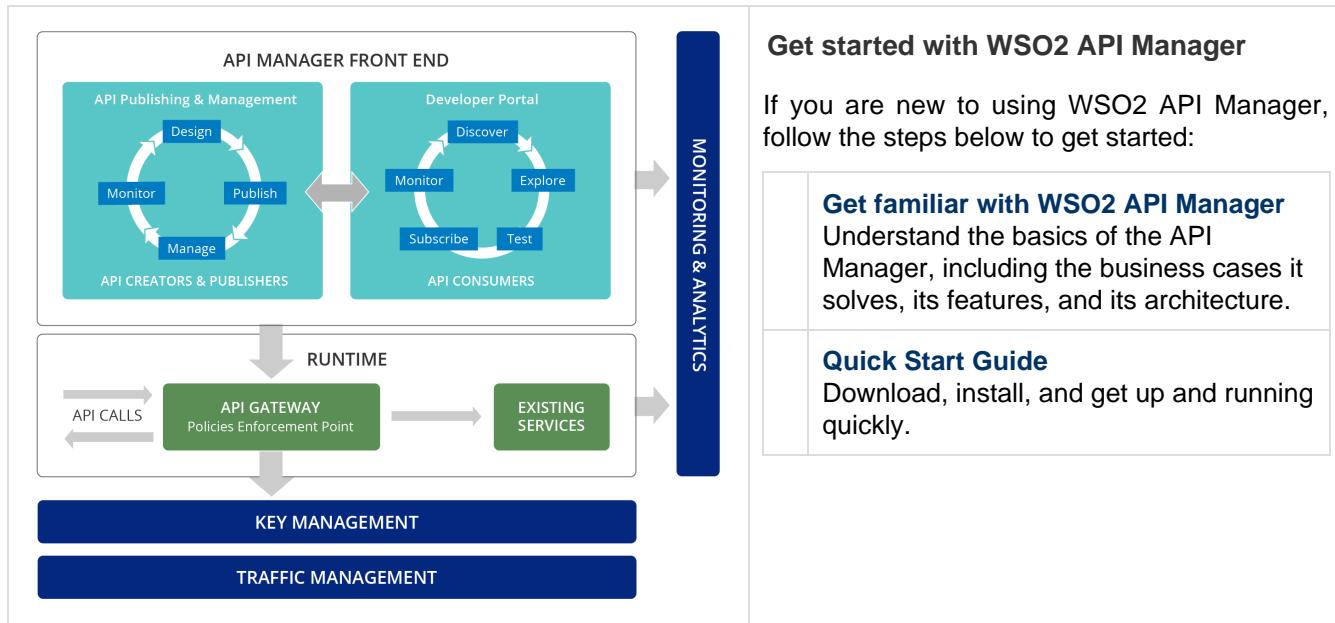
1.5.5.4 Adding a Reverse Proxy Server	910
1.5.5.5 Writing Custom Grant Types	915
1.5.5.6 Extending Key Validation	915
1.5.5.7 Extending the Key Manager Interface	916
1.5.5.8 Adding a New API Store Theme	917
1.5.5.9 Extending Scope Validation	920
1.5.5.10 Extending the API Life Cycle	922
1.5.5.11 Customize the API Store and Gateway URLs for Tenants	929
1.5.5.12 Editing API Templates	934
1.5.5.13 Customizing Login Pages for API Store and API Publisher	934
1.5.5.14 API Gateways with Dedicated Backends	938
1.5.5.15 Securing OAuth Token with HMAC Validation	942
1.5.5.16 Customizing User SignUp in API Store	944
1.5.6 Working with Security	949
1.5.6.1 Passing Enduser Attributes to the Backend Using JWT	952
1.5.6.2 Dynamic SSL Certificate Installation	957
1.5.6.3 Maintaining Logins and Passwords	960
1.5.6.4 Saving Access Tokens in Separate Tables	965
1.5.6.5 Configuring WSO2 Identity Server as the Key Manager	967
1.5.6.6 Configuring a Third-Party Key Manager	967
1.5.6.7 Enabling Role-Based Access Control Using XACML	974
1.5.6.8 Encrypting OAuth Keys	983
1.5.6.9 Provisioning Out-of-Band OAuth Clients	984
1.5.6.10 Basic Auth	989
1.5.6.11 Digest Auth	990
1.5.6.12 Working with Encrypted Passwords	991
1.5.6.12.1 Encrypting Secure Endpoint Passwords	994
1.5.6.13 Regular Expression Threat Protection for API Gateway	996
1.5.6.14 Mutual SSL Support for API Gateway	100
1.5.6.15 Securing APIs	100
1.5.6.15.1 Multi Factor Authentication (MFA) for Publisher and Developer Portals	101
1.5.7 Working with Throttling	101
1.5.7.1 Introducing Throttling Use-Cases	101
1.5.7.2 Setting Maximum Backend Throughput Limits	102
1.5.7.3 Setting Throttling Limits	102
1.5.7.4 Adding New Throttling Policies	102
1.5.7.5 Managing Throttling	103
1.5.7.6 Enforcing Throttling to an API	104
1.5.7.7 Engaging a new Throttling Policy at Runtime	104
1.5.7.8 Engaging Multiple Throttling Policies to a Single API	105
1.5.8 Working with Endpoints	105
1.5.9 Analytics	106
1.5.9.1 Configuring APIM Analytics	106
1.5.9.2 Analyzing APIM Statistics with Batch Analytics	107
1.5.9.2.1 Introducing the WSO2 API Manager Statistics Model	107
1.5.9.2.2 Viewing API Statistics	108
1.5.9.2.3 Using Geolocation Based Statistics	110
1.5.9.3 Managing Alerts with Real-time Analytics	111

1.5.9.3.1 Alert Types	111
1.5.9.3.2 Configuring Alerts	112
1.5.9.3.3 Subscribing for Alerts	112
1.5.9.3.4 Viewing Alerts	112
1.5.9.4 Analyzing Logs with the Log Analyzer	112
1.5.9.4.1 Viewing Live Logs	113
1.5.9.4.2 Analyzing the Log Overview	113
1.5.9.4.3 Analyzing Application Errors	113
1.5.9.4.4 Analyzing Access Token Errors	113
1.5.9.4.5 Analyzing API Deployment Statistics	114
1.5.9.4.6 Analyzing Login Errors	114
1.5.9.4.7 Analyzing the Number of API Failures	114
1.5.9.5 Integrating with Google Analytics	114
1.5.9.6 Monitoring with WSO2 Carbon Metrics	114
1.5.9.6.1 Enabling Metrics and Storage Types	114
1.5.9.6.2 Configuring Metrics Properties	115
1.5.9.6.3 Using JVM Metrics	115
1.5.9.7 Installing WSO2 API-M Analytics Features	115
1.5.9.8 Purging Analytics Data	116
1.5.9.9 Default Ports of WSO2 API-M Analytics	116
1.5.9.10 Troubleshooting the Analytics Profile	116
1.5.9.11 Updating WSO2 API Manager Analytics	116
1.5.10 Reference Guide	116
1.5.10.1 Product Profiles	116
1.5.10.2 Default Product Ports	117
1.5.10.3 Changing the Default Ports with Offset	117
1.5.10.4 Error Handling	117
1.5.10.5 Message Flow in the API Manager Gateway	118
1.5.10.6 Updating WSO2 API Manager	118
1.5.10.7 Accessing API Manager by Multiple Devices Simultaneously	118
1.5.11 Developer Guide	118
1.5.11.1 Working with the Source Code	118
1.5.11.2 Java Documentation	118
1.5.11.3 WSO2 Admin Services	118
1.5.11.4 Product APIs	119
1.5.11.4.1 RESTful APIs	119
1.5.11.4.2 Token API	119
1.5.11.4.3 Deprecated APIs	122
1.5.11.5 Working with Audit Logs	124
1.5.11.6 Enabling Authentication Session Persistence	125
1.5.11.7 Enabling Monetization of APIs	125
1.5.11.8 Using the Registry REST API	126
1.6 FAQ	126
1.7 Sample Scenarios	127
1.7.1 API Development	127
1.7.1.1 Developer Optimized APIs Development - Sample Documentation	127
1.7.1.2 Managing Public, Partner vs Private APIs Sample Documentation	128
1.7.1.3 Ownership, permission and collaborative API development - Sample Documentation	128

1.7.2 API Governance	128
1.7.3 API Lifecycle Management	129
1.7.4 API Rate Limiting	129
1.7.5 API Rate Monetization	130
1.7.6 API Security Sample	130
1.7.7 API Versioning	130

WSO2 API Manager Documentation

Welcome to WSO2 API Manager Documentation! [WSO2 API Manager \(WSO2 API-M\)](#) 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.



Get started with WSO2 API Manager

If you are new to using WSO2 API Manager, follow the steps below to get started:

Get familiar with WSO2 API Manager

Understand the basics of the API Manager, including the business cases it solves, its features, and its architecture.

Quick Start Guide

Download, install, and get up and running quickly.

Deep dive into WSO2 API Manager

Tutorials	Deep Dive	Admin Guide
Product APIs	Analytics	Reference Guide

Introduction

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

- [Overview](#)
- [About this Release](#)

Overview

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, it is also powered by the [WSO2 Data Analytics Server](#) and is immediately ready for massively scalable deployments.

WSO2 API Manager is [highly performant](#), 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 artifacts. The default communication protocol of the Key Manager is Thrift.

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

Updates

For a list of available updates released after WSO2 API Manager 2.1.0, see the following:

- [Update-1](#)
- [Update-2](#)
- [Update-3](#)
- [Update-4](#)
- [Update-5](#)
- [Update-6](#)
- [Update-7](#)
- [Update-8](#)
- [Update-9](#)
- [Update-10](#)

For version specific documentation for 2.x.x, see the following:

- [WSO2 API Manager 2.0.0](#)
- [WSO2 API Manager 2.1.0](#)

Compatible WSO2 product versions

WSO2 APIM 2.1.0 is based on [WSO2 Carbon 4.4.11](#) and is expected to be compatible with any of the WSO2 products that are based on any Carbon 4.4.x version. If you get any compatibility issues, please contact team [WSO2](#). For information on the third-party software required with APIM 2.1.0, see [Installation Prerequisites](#). For more information on the products in each Carbon platform release, see the [Release Matrix](#).

Known issues

For a list of known issues, see [WSO2 API Manager 2.x.x - Known Issues](#).

Quick Start Guide

WSO2 API Manager is a complete solution for designing and publishing APIs, creating and managing a developer community, and for securing and routing API traffic in a scalable manner. It leverages proven components from the WSO2 platform to secure, integrate and manage APIs. In addition, it integrates with the [WSO2 analytics platform](#) and provides out of the box reports and alerts, giving you instant insights into the APIs behavior.

Before you begin,

1. Install [Oracle Java SE Development Kit \(JDK\)](#) version 1.7.* or 1.8.* and set the JAVA_HOME environment variable. Refer [Installing the product documentation](#) to set JAVA_HOME environment variable for different operating systems
2. Download the latest version of WSO2 API Manager from <https://github.com/wso2/product-apim/releases/tag/v2.1.0-update10>.
3. Start WSO2 API Manager by going to the <API-M_HOME>/bin directory using the command-line and then executing wso2server.bat (for Windows) or wso2server.sh (for Linux.)

Let's go through the use cases of the API Manager:

- [Invoking your first API](#)
- Understanding the API Manager concepts
- Deep diving into the API Manager
 - Creating users and roles
 - Creating an API from scratch
 - Adding API documentation
 - Adding interactive documentation
 - Versioning the API
 - Associating Scopes to API Resources
 - Publishing the API
 - Subscribing to the API
 - Invoking the API
 - Monitoring APIs and viewing statistics

Invoking your first API

Follow the steps in this section to quickly deploy a sample API, publish it, subscribe to it, and invoke it.

1. Open the API Publisher (<https://<hostname>:9443/publisher>) and sign in with **admin/admin** credentials.
2. Exit from API creation tutorial by clicking the close icon(X) on top right corner.



3. Click the **Deploy Sample API** button. It deploys a sample API called `PizzaShackAPI` into the API Manager.

This **Deploy Sample API** option is available only when there are no APIs in API Publisher. If you have already created a API, this option will not be available.

The screenshot shows the WSO2 API Publisher interface. At the top, there is a navigation bar with the WSO2 logo, a home icon, and the text "HOME / APIS". Below the navigation bar, there is a blue header bar with a gear icon labeled "APIS", a close button "X", and a "ADD NEW API" button with a plus sign. On the left side, there is a sidebar with two items: "STATISTICS" and "MANAGE SUBSCRIPTIONS". The main content area is titled "All APIs" and contains a message: "No APIs created yet. Click one of the buttons below to get started." Below this message are two buttons: "New API..." and "Deploy Sample API". The "Deploy Sample API" button is highlighted with a red border.

4. Click `PizzaShackAPI` to open it.

The screenshot shows the WSO2 API Publisher interface. The top navigation bar includes the WSO2 logo, a home icon, and the text "HOME / APIS". Below the navigation bar are three main menu items: "APIS" (selected), "STATISTICS", and "MANAGE SUBSCRIPTIONS". A blue header bar at the top right contains a close button ("X"), an "ADD NEW API" button, and a search bar with a magnifying glass icon and an information icon. The main content area is titled "All APIs" and displays a search bar. A card for the "PizzaShackAPI" is highlighted with a red border. The card contains the following information:

-
- PizzaShackAPI**
- 1.0.0
- admin
- 0 Users
- PUBLISHED
-

5. Go to the **Lifecycle** tab and note that the **State** is PUBLISHED. The API is already published to the API Store.

PizzaShackAPI - 1.0.0

Overview **Lifecycle** Versions Docs Users

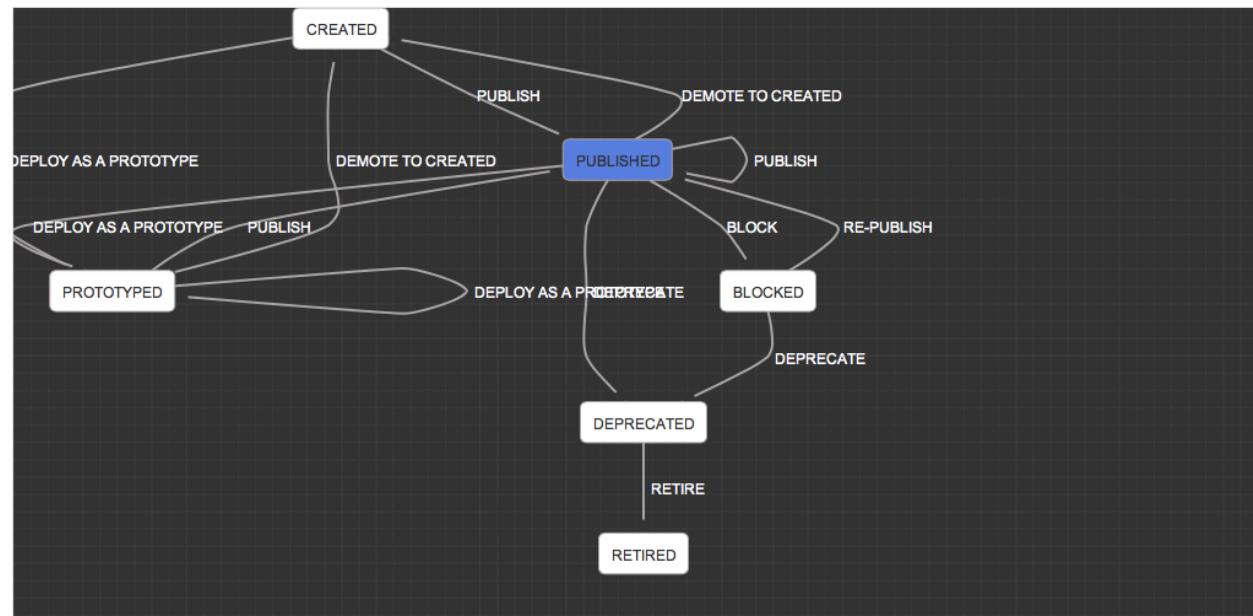
Current State: **PUBLISHED**

Block

Deploy as a Prototype

Demote to Created

Deprecate

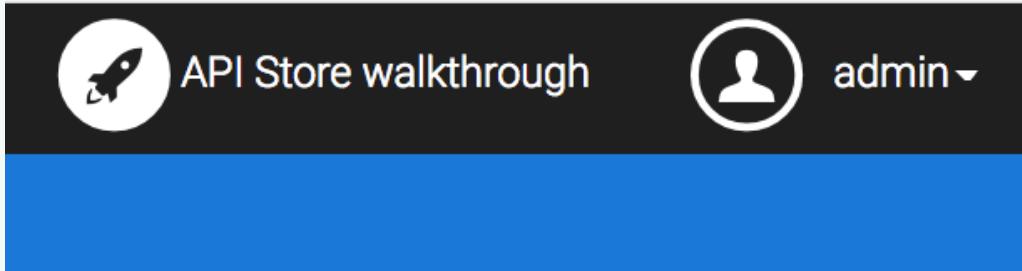


6. Sign in to the API Store (<https://<hostname>:9443/store>) with the **admin/admin** credentials and click on the PizzaShackAPI API.

The screenshot shows the WSO2 API Store interface. The top navigation bar includes the WSO2 logo, a search bar, and a dropdown menu. Below the navigation is a sidebar with links for APIS, APPLICATIONS, FORUM, and STATISTICS, and a TAGS section containing the word "pizza". The main content area is titled "APIs" and displays a card for the "PizzaShackAPI". The card features a thumbnail image of a pizza, the API name "PizzaShackAPI", its version "1.0.0", the developer "Jane Roe", and a five-star rating.

API Store Walkthrough

You can click "API Store walkthrough" to view the interactive tutorial to invoke the API.



7. Select the default application and an available tier, and click **Subscribe**.

PizzaShackAPI - 1.0.0

Version: 1.0.0
By: Jane Roe
Updated: 21/Jul/2016 13:41:55 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications
DefaultApplication

Tiers
Unlimited

Subscribe

- When the subscription is successful, click **View Subscriptions** on the information message that appears. Click the **Production Keys** tab and click **Generate Keys** to generate an **access token** to invoke the API.

DefaultApplication

Production Keys (highlighted with a red box)

No Keys Found
No keys are generated for this type in this application.

Grant Types
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

SAML2 IWA-NTLM Implicit Refresh Token
 Client Credential Code Password

Callback URL

Access token validity period
 Seconds.

Generate keys (highlighted with a red box)

You have now successfully subscribed to an API. Let's invoke the API using the integrated Swagger-based API Console.

- Click the **APIs** menu again and click the **PizzaShackAPI** to open it. When the API opens, click its **API Console** tab.

PizzaShackAPI - 1.0.0

Expand the GET method (which retrieves the menu) and click **Try it out**.

Implementation Notes
Return a list of available menu items

Response Class (Status 200)
OK. List of APIs is returned.

```
{
  "list": [
    {
      "price": "string",
      "description": "string",
      "name": "string",
      "image": "string"
    }
  ]
}
```

Response Content Type application/json

Header	Description	Type	Other

Response Messages

HTTP Status Code	Reason	Response Model	Headers
304	Not Modified. Empty body because the client has already the latest version of the requested resource.		
406	Not Acceptable. The requested media type is not supported	Model Example Value <pre>{ "message": "string", "error": [{ "message": "string", "code": 0 }], "description": "string", "code": 0, "moreInfo": "string" }</pre>	

Try it out!

Note the response for the API invocation. It returns the list of menu items.

Response Body

```
[
  {
    "name": "BBQ Chicken Bacon",
    "icon": "/images/6.png",
    "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions in barbecue sauce",
    "price": "20.99"
  },
  {
    "name": "Chicken Parmesan",
    "icon": "/images/1.png",
    "description": "Grilled chicken, fresh tomatoes, feta and mozzarella cheese",
    "price": "20.99"
  },
  {
    "name": "Chilly Chicken Cordon Bleu",
    "icon": "/images/10.png",
    "description": "Spinash Alfredo sauce topped with grilled chicken, ham, onions and mozzarella",
    "price": "26.99"
  },
  {
    "name": "Double Bacon 6Cheese",
    "icon": "/images/11.png",
    "description": "Double Bacon 6Cheese",
    "price": "26.99"
  }
]
```

You have deployed a sample API, published it to the API Store, subscribed to it, and invoked the API using our integrated API Console.

Understanding the API Manager concepts

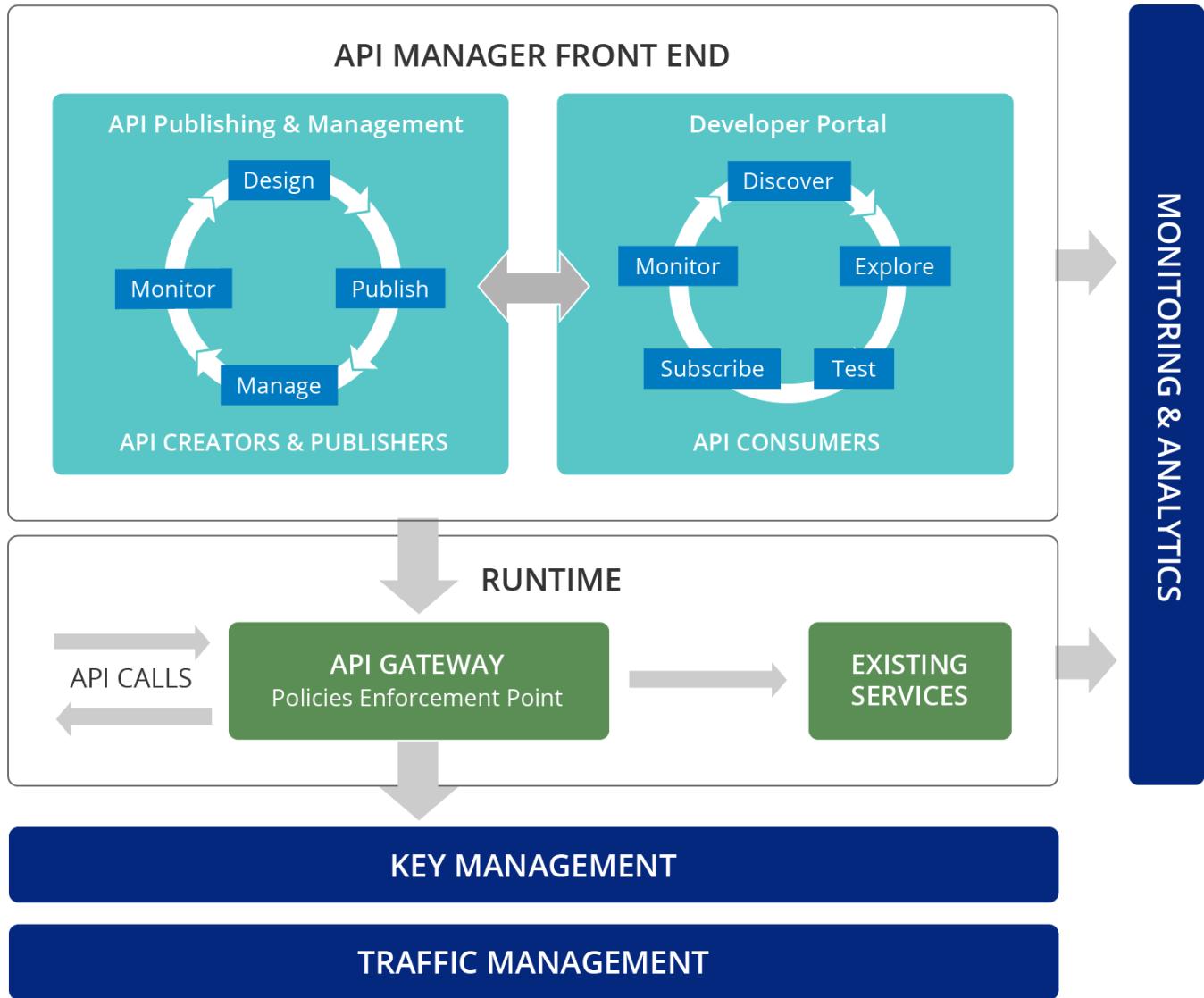
Before we look into the API management activities in detail, let's take a look at the basic API management concepts.

[Components] [Users and roles] [API lifecycle] [Applications] [Throttling tiers] [API keys] [API resources]

Components

The API Manager consists of the following components:

- **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 (Developer Portal):** 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`.
- **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. The API Gateway connects with the Key Manager to check the validity of subscriptions, OAuth tokens, and API invocations. The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway.
- **Traffic Manager:** Helps users to regulate API traffic, make APIs and applications available to consumers at different service levels and secures APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time.
- **WSO2 API Manager Analytics:** Provides a host of statistical graphs, an alerting mechanism on predetermined events and a log analyzer.



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 (e.g., 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.
- **Consumer:** A consumer uses the API Store to discover APIs, see the documentation and forums, and rate/comment on the APIs. Consumers subscribe to APIs to obtain API keys.

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 of the backend services they rely on. This lifecycle is exposed in the API Publisher 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.

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 with an API at subscription time and can be defined at an API-level, resource-level, subscription-level and application-level (per token). They define the throttling limits enforced by the API Gateway, e.g., 10 TPS (transactions per second). The final throttle limit granted to a given user on a given API is ultimately defined by the consolidated output of all throttling tiers together. The API Manager comes with three predefined tiers for each level and a special tier called `Unlimited`, which you can disable by editing the `<ThrottlingConfigurations>` element of the `<API-M_HOME>/repository/conf/api-manager.xml` file.

In API Manager 2.0.0 onwards, **Advanced Throttling** is enabled by default with following configuration in `<API-M_HOME>/repository/conf/api-manager.xml`.

```
<ThrottlingConfigurations>
    <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
    .....
<ThrottlingConfigurations>
```

If you are disabling **Advanced Throttling** in any case by setting the value of `<EnableAdvanceThrottling> false` also, Advanced Throttling is disabled and basic Throttling mechanism is enabled thereafter. In such a scenario, if you want to disable the Unlimited Throttling tier of basic Throttling configurations, you need to disable it under `<TierManagement>` by setting `<EnableUnlimitedTier> false`.

```
<TierManagement>
    <EnableUnlimitedTier>true</EnableUnlimitedTier>
</TierManagement>
```

Predefined Subscription Tiers.

Throttling Tier	Description
Unlimited	Allows unlimited requests

Gold	Allows 5000 requests per minute
Silver	Allows 2000 requests per minute
Bronze	Allows 1000 requests per minute

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 that is deployed on many devices).

Application access token: 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 NtBQkXoKElu0H1a1fQ0DWfo6IX4a") 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 which is 60 minutes by default, you open the <API-M_HOME>/repository/conf/identity/identity.xml file and change the value of the element <AccessTokenDefaultValidityPeriod>. If you set a negative value, the token never expires. **Changes to this value are applied only to the new applications that you create.**

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.

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=password`. 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, DELETE, PATCH, HEAD, and OPTIONS. You can give multiple values at once.
- **uri-template:** A URI template as defined in <http://tools.ietf.org/html/rfc6570>. (e.g., /phoneneverify/<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, Application User, or Application & 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.
- Application & Application User: An application access token together with a user access token is required to access the API resource.

Deep diving into the API Manager

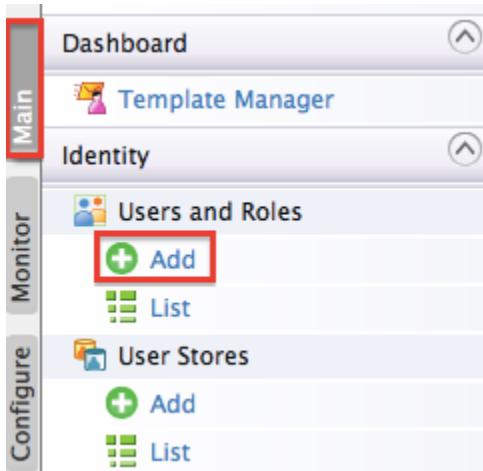
Let's take a look at the typical API management activities in detail:

- Creating users and roles
- Creating an API from scratch
- Adding API documentation
- Adding interactive documentation
- Versioning the API
- Associating Scopes to API Resources
- Publishing the API
- Subscribing to the API
- Invoking the API
- Monitoring APIs and viewing statistics

Creating users and roles

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

1. Sign in to the Management Console (<https://<hostname>:9443/carbon>) of the API Manager using `admin/admin` credentials.
2. Click **Add** in the **Users and Roles** section under the **Main** menu.



3. Click **Add New Role**.



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

Add New Role

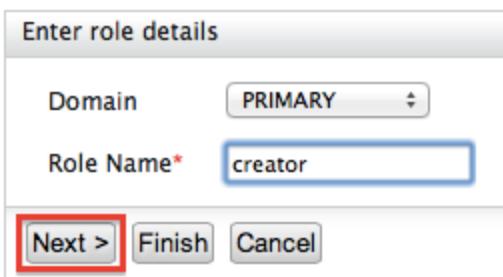
Step 1 : Enter role details

Enter role details

Domain PRIMARY

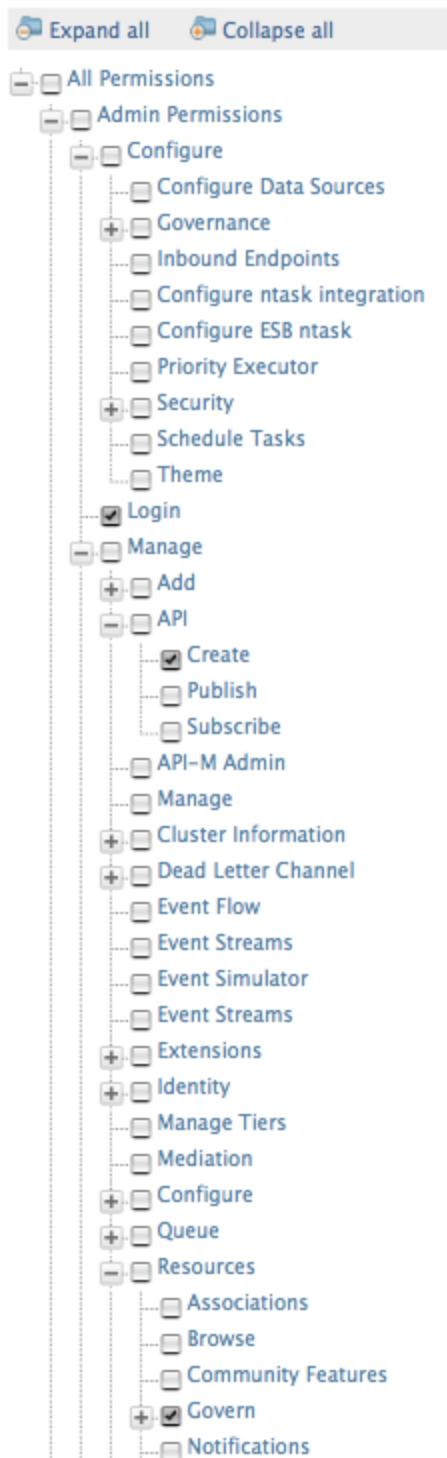
Role Name* creator

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



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

- All Permissions > Admin Permissions > Configure > Governance and all underlying permissions
- All Permissions > Admin Permissions > Login
- All Permissions > Admin Permissions > Manage > API > Create
- All Permissions > Admin Permissions > Manage > Resources > Govern and all underlying permissions



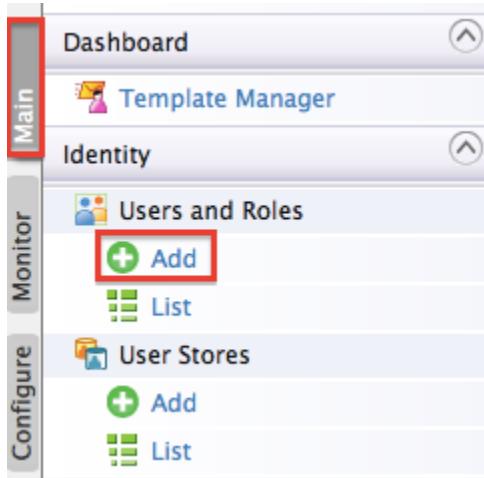
6. Similarly, create the publisher role with the following permissions.
 - All Permissions > Admin Permissions > Login
 - All Permissions > Admin Permissions > Manage > API > Publish
7. Note that the API Manager comes with the subscriber role available by default. It has the following permissions:
 - All Permissions > Admin Permissions > Login
 - All Permissions > Admin Permissions > Manage > API > Subscribe
8. The roles you added (creator and publisher) are now displayed under **Roles**.

Roles	
Search Roles Select Domain <input type="button" value="ALL-USER-STORE-DOMAINS"/> Enter role name pattern (* for all) <input type="text" value="*"/> <input type="button" value="Search Roles"/>	
Name	Actions
admin	<input type="button" value="Assign Users"/> <input type="button" value="View Users"/>
Application/admin_DefaultApplication_PRODUCTION	<input type="button" value="Rename"/> <input type="button" value="Permissions"/> <input type="button" value="Assign Users"/> <input type="button" value="View Users"/> <input type="button" value="Delete"/>
creator	<input type="button" value="Rename"/> <input type="button" value="Permissions"/> <input type="button" value="Assign Users"/> <input type="button" value="View Users"/> <input type="button" value="Delete"/>
Internal/everyone	<input type="button" value="Permissions"/>
Internal/subscriber	<input type="button" value="Rename"/> <input type="button" value="Permissions"/> <input type="button" value="Assign Users"/> <input type="button" value="View Users"/> <input type="button" value="Delete"/>
publisher	<input type="button" value="Rename"/> <input type="button" value="Permissions"/> <input type="button" value="Assign Users"/> <input type="button" value="View Users"/> <input type="button" value="Delete"/>

For more details Add roles and permission assign to roles, see [Adding User Roles](#).

Let's create users for each of the roles.

9. Click **Add** in the **Users and Roles** section under the **Main** menu.



10. Click **Add New User**.

Add Users and Roles

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

Add New 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 click **Finish**.

Add User

Step 2 : Select roles of the user

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

Users of Role

Select all on this page | Unselect all on this page

<input type="checkbox"/> admin
<input type="checkbox"/> creator
<input checked="" type="checkbox"/> publisher
<input checked="" type="checkbox"/> Internal/everyone
<input type="checkbox"/> Internal/subscriber
<input type="checkbox"/> Application/admin_DefaultApplication_PRODUCTION

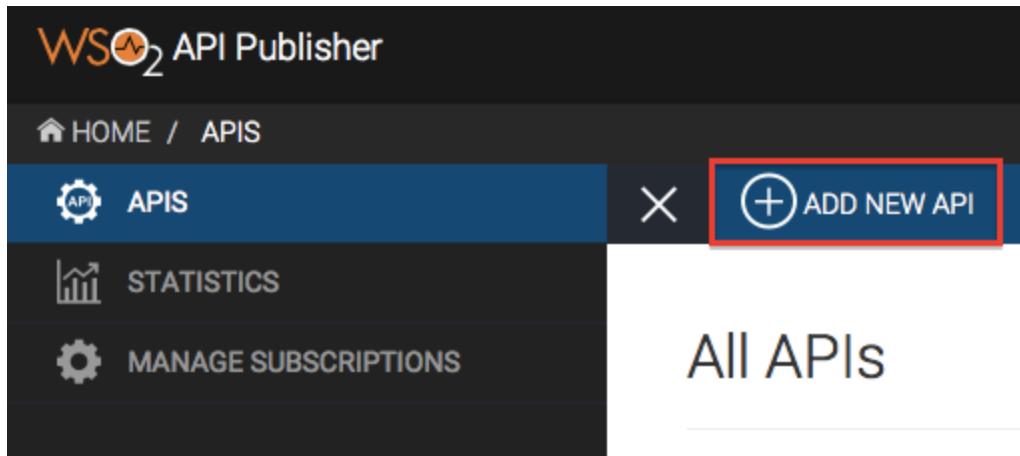
Finish **Cancel**

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

Creating an API from scratch

Let's create an API from scratch.

1. Sign in to the API Publisher (<https://<hostname>:9443/publisher>) as **apicreator** .
2. In the **APIS** menu, click **Add New API**.



3. Select the option to design a new API and click **Start Creating**.

Let's get started!

Add New API

I Have an Existing API
Use an existing API's endpoint or the API Swagger definition to create an API.

I Have a SOAP Endpoint
Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

Design a New REST API
Design and prototype a new REST API.

Start Creating

Design a New Websocket API
Design and prototype a new Websocket API.

4. Give the information in the table below.

Field	Sample value
Name	PhoneVerification
Context	/phoneverify
Version	1.0.0
Access Control	All
Visibility on Store	Public
API Definition	<ul style="list-style-type: none"> URL pattern: CheckPhoneNumber <p>Note that this URL Pattern is the name of one of the resources that we are going to invoke from the backend service.</p> <ul style="list-style-type: none"> Request types: GET, POST

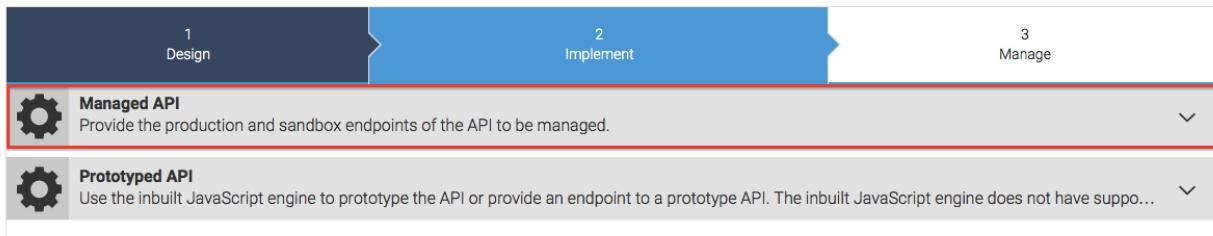
Click **Add** and then click **Next: Implement >** to move on to the next page.

Design API

The screenshot shows the 'Design' tab selected in the top navigation bar. The 'General Details' section contains fields for Name (PhoneVerification), Context (/phoneverify), Version (1.0.0), Access Control (All), Visibility on Store (Public), Description (empty), and Tags (empty). A 'Thumbnail Image' section with a placeholder image and a 'Select image' button is also present. The 'Dimensions (max): 100 x 100 pixels' constraint is noted. The 'API Definition' section shows a URL Pattern (/phoneverify/1.0.0) with a subpath CheckPhoneNumber, and checkboxes for various HTTP methods (GET, POST, PUT, DELETE, PATCH, HEAD) with GET and POST checked. An 'Import' and 'Edit Source' button are available. A 'Save' button and a 'Next: Implement >' link are at the bottom.

5. Select the **Managed API** option.

PhoneVerification: /phoneverify/1.0.0

For instructions on how to Implement Prototyped API, see [Create a Prototyped API with an Inline Script](#).Give the following information and click **Next:Manage >** once you are done.

Field	Sample value
Endpoint type	HTTP/REST Endpoint
Production endpoint	Endpoint is http://ws.cdyne.com/phoneverify/phoneverify.asmx . To verify the URL, click the Test button next to it. In this example, we use a phone validation service exposed by the Cdyne services provider. This service has SOAP and REST interfaces. This sample service has two operations: CheckPhoneNumber and CheckPhoneNumbers.
Sandbox endpoint	Endpoint is http://ws.cdyne.com/phoneverify/phoneverify.asmx . To verify the URL, click the Test button next to it.

Endpoint Type is selected to specify whether the endpoint is based on a URI template or based on a URI template or an address specified in the 'To' header.

The screenshot shows the 'Implement' tab of the WSO2 API Manager interface. It includes sections for 'Managed API' (with a note about production and sandbox endpoints), 'Message Mediation Policies' (with an enable checkbox), 'CORS configuration' (with an enable checkbox), and a 'Save' button followed by a 'Next : Manage >' button.

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type :* (highlighted)

Load Balanced Failover

Production Endpoint :* (highlighted) Test

Sandbox Endpoint :* (highlighted) Test

Show More Options

Related Links

- For details about Message Mediation Policies, see [Message Mediation Policies](#)
- For details about CORS Configurations, see [CORS Configuration](#)

6. Provide the following information in the **Manage** tab. Leave default values for the rest of the parameters in the UI.

Field	Value	Description
Subscription Tiers	<Select all available tiers>	The API can be available for subscription at different levels of service. They allow you to limit the number of successful hits to an API during a given period of time.

Configurations

Make this the Default Version: No default version defined for the current API

Transports: * HTTPS HTTP

Response Caching: Disabled

Throttling Settings

Maximum Backend Throughput : Unlimited Specify

Subscription Tiers: * **Unlimited** : Allows unlimited requests
 Gold : Allows 5000 requests per minute
 Silver : Allows 2000 requests per minute
 Bronze : Allows 1000 requests per minute

Advanced Throttling Policies: Apply to API Apply per Resource
Select Policy per Resource

ⓘ Refer documentation for more information about each throttling setting.

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

Adding API documentation

1. In the **APIS** menu, click the thumbnail of the API to open it.
2. Click on the API's **Docs** tab and click **Add New Document**.

PhoneVerification - 1.0.0

Overview Versions **Docs** Users

Add New Document

ⓘ No documentation associated with the API
 There is no documentation created for this API. You can add new documentation to this API by clicking the "Add New Document" button.

3. 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' named PhoneVerification, specifying in-line content as the source and optionally entering a summary. When you have finished, click **Add Document**.

PhoneVerification - 1.0.0

The screenshot shows the 'Add New Document' dialog for the 'PhoneVerification' API. The 'Docs' tab is selected in the top navigation bar. The dialog form includes fields for 'Name*' (PhoneVerification), 'Summary*' (Check the validity of a phone number), and a 'Type' section with 'How To' selected. The 'Source' section shows 'Inline' selected. At the bottom are 'Add Document' and 'Cancel' buttons, with 'Add Document' highlighted by a red box. A note below the dialog states: 'No documentation associated with the API. There is no documentation created for this API. You can add new documentation to this API by clicking the "Add New Document" button.'

5. Once the document is added, click **Edit Content** to open an embedded editor.

PhoneVerification - 1.0.0

The screenshot shows the 'Edit Content' dialog for the 'PhoneVerification' document. The 'Docs' tab is selected in the top navigation bar. The dialog includes a search bar ('Filter by ...'), a table listing the document, and pagination controls ('Show 10 entries'). The table has columns for Name, Type, Modified On, and Actions. The 'Actions' column for the 'PhoneVerification' row contains three buttons: 'Edit Content' (highlighted with a red box), 'Update', and 'Delete'. Below the table, it says 'Showing 1 to 1 of 1 entries'.

6. Enter your API's documentation and click **Save and Close**.

PhoneVerification

Determine whether a phone line 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 timezone to restrict calling times
- Validate wireless vs landline data for compliance

ul » li

Save **Save and Close** **Cancel**

Adding interactive documentation

WSO2 API Manager has an integrated [Swagger UI](#), which is part of the [Swagger project](#).

Swagger is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe services in the same way that interfaces describe lower-level programming code.

The [Swagger UI](#) is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generates documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation and more discoverability. The Swagger UI has YAML code, and its UI facilitates easier code indentation, provides keyword highlighting, and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger UI and it has provides the facility to download your API definition as YAML or JSON file. Go to the [Swagger 2.0 specification](#) for more information.

1. Open the API Publisher (<https://<hostname>:9443/publisher>) and sign in as **apicreator**.
2. Click the **Edit** icon for the PhoneVerification API. This opens the API in its edit mode.

All APIs

The screenshot shows the 'All APIs' page with two API definitions listed:

- PizzaShackAPI** (Version 1.0.0, Admin, 1 User, Published):
 - Icon: Image of a pizza.
 - Actions: Edit, Delete.
- PhoneVerificationAPI** (Version 1.0.0, apicreator, 0 Users, Created):
 - Icon: Red square with white letter P.
 - Actions: Edit (button highlighted with a red box).

- Click the **Edit Source** button under the **API Definition** section.

The screenshot shows the 'API Definition' editor for the 'PhoneVerificationAPI' (version 1.0.0). The URL pattern is set to '/phoneverify/1.0.0'. The 'Edit Source' button is highlighted with a red box. The interface includes sections for methods (GET, POST) and their corresponding resource paths, along with summary links and implementation status.

Method	Path	Summary	Implementation
GET	/CheckPhoneNumber	+ Summary	(Status: IMPLEMENTED)
POST	/CheckPhoneNumber	+ Summary	(Status: IMPLEMENTED)

Buttons at the bottom include 'Save' and 'Next: Implement >'. Other buttons like 'Import' and 'Edit Source' are also visible.

- The API definition as a YAML code opens in a separate page. Expand its GET method, add the following parameters and click **Apply Changes**.

```
parameters:  
  - in: query  
    name: PhoneNumber  
    description: Give the phone number to be validated  
    type: string  
    required: true  
  - in: query  
    name: LicenseKey  
    description: Give the license key as 0 for testing purpose  
    type: string  
    required: true
```

```

1  swagger: '2.0'
2  paths:
3    /CheckPhoneNumber:
4      get:
5        responses:
6          '200':
7            description: ''
8            x-auth-type: Application & Application User
9            x-throttling-tier: Unlimited
10       parameters:
11         - in: query
12           name: PhoneNumber
13           description: Give the phone number to be validated
14           type: string
15           required: true
16         - in: query
17           name: LicenseKey
18           description: Give the license key as 0 for testing purpose
19           type: string
20           required: true
21   post:
22     responses:
23       '200':
24         description: ''
25     parameters:
26       - name: Payload
27         description: Request Body
28         required: false
29         in: body
30         schema:
31           type: object
32           properties:
33             payload:
34               type: string
35           x-auth-type: Application & Application User
36           x-throttling-tier: Unlimited
37   info:
38     title: PhoneVerification
39     version: 1.0.0

```

- Back in the API Publisher, note that the changes you did appear in the API Console's UI. You can add more parameters and edit the summary/descriptions using the API Publisher UI as well. Once done, click **Save**.

API Definition

URL Pattern Url Pattern E.g.: path/to/resource

GET POST PUT DELETE PATCH HEAD **more**

GET	/CheckPhoneNumber	+ Summary			
Description : + Add Implementation Notes					
Produces : Empty		Consumes : Empty			
Parameters :					
Parameter Name	Description	Parameter Type	Data Type	Required	Delete
PhoneNumber	Give the phone number to be validated	query	string	True	<input type="button" value="⊖"/>
LicenseKey	Give the license key as 0 for testing purpose	query	string	True	<input type="button" value="⊖"/>
<input style="width: 15%; border: 1px solid #ccc; height: 20px; margin-right: 10px;" type="text"/> <input type="button" value="⊕ Add Parameter"/>					
POST	/CheckPhoneNumber	+ Summary			
Save Next: Implement >					

Versioning the API

Let's create a new version of this API.

1. Sign in to the API Publisher as **apicreator** if you are not logged in already.
2. Click the PhoneVerification API to open it and then click **Create New Version**.



PhoneVerification - 1.0.0

Overview	Versions	Docs	Users
			
 0 Users			
CREATED			
Docs			
 View in Store			
Visibility	Public		
Context	/phoneverify/1.0.0		
Production URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx		
Sandbox URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx		
Date Last Updated	July 22, 2016 9:20:02 PM GMT+05:30		
Tier Availability	Bronze,Unlimited,Gold,Silver		
Default API Version	None		
Published Environments	Production and Sandbox		

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

PhoneVerification - 1.0.0

Create New Version

New Version:*	<input type="text" value="2.0.0"/>
Make this the Default Version ?	<input type="checkbox"/>
<div style="background-color: #f0f0f0; padding: 5px;"> i Info! No default version defined for the current API </div>	
Done Cancel	

4. Note that the new version of the API is created in the API Publisher.

Associating Scopes to API Resources

Different API resources can be associated with different user roles. For an example, consider the following resources and the operations:

GET	/orders
POST	/orders
GET	/items
POST	/items

In order to map a scope to an API resource, the following should be done:

- API creator should first create scopes, by clicking on Add Scopes in the Manage tab.

The screenshot shows the WSO2 API Manager's 'Manage' tab. In the 'Resources' section, there is a button labeled '+ Add Scopes' which is highlighted with a red box.

- Fill the scope related information in the dialog that pops up. Note that Scope Key and Roles are the most important attributes. Click on Add Scope button on the right hand side bottom.

The screenshot shows the 'Define Scope' dialog box. It includes fields for 'Scope Key' (item_view), 'Scope Name' (View Items), 'Roles' (manager, agent), and a 'Description' text area (E.g., This scope will group all the administration APIs). At the bottom right, there are 'Add Scope' and 'Close' buttons, with the 'Add Scope' button highlighted with a red box.

The roles added here are validated against the user store to check if they exist. However, this can be overridden so that the roles are not checked in the user store. For thus purpose, set the Java system property **disableRoleValidationAtScopeCreation** to true at the server startup:

- Open <API-M_HOME>/bin/wso2server.(sh|bat) file.
- Add -DdisableRoleValidationAtScopeCreation=true at the end of the file.
- Restart the server.

- Now the scope with key 'item_view' is added with roles manager and agent. To associate this scope with the get operation on the /time resource, click on the +Scope sign on the right hand side of the resource.

The screenshot shows the configuration for the '/item' resource. The 'Summary' tab is active. In the 'Scopes' section, it says 'Unlimited' and has a '+ Scope' button, which is highlighted with a red box.

- From the drop down menu that appears, select the scope name and click on the tick sign to the right. Now the scope will be associated with the GET operation on the resource /item.

The screenshot shows the configuration for the '/item' resource. The 'Summary' tab is active. In the 'Scopes' section, a dropdown menu is open with 'View Items' selected, accompanied by a checkmark and a close button. The 'View Items' option is highlighted with a red box.

See [Scope Management with OAuth Scopes](#) for an in depth example.

Publishing the API

1. Sign in to the API Publisher as the `apipublisher` user that you created earlier in this guide, and click the `PhoneVerification` API's version 2.0.0.

All APIs

API Name	Version	State
PizzaShackAPI	1.0.0	PUBLISHED
PhoneVerification	1.0.0	CREATED
PhoneVerification	2.0.0	CREATED

2. The API opens. Go to its **Lifecycle** tab and click **Publish**.

PhoneVerification - 2.0.0

Overview Lifecycle Versions Docs Users

Current State: **CREATED**

Require re-subscription when publish the API
 Deprecate old versions after publish the API

Publish Deploy as a Prototype

The check boxes mean the following:

- **Require re-subscription when publish the API:** Invalidates current user subscriptions, forcing users to subscribe again.
- **Deprecate old versions after publish the API:** If selected, any prior versions of the API that are published will be set to the DEPRECATED state automatically.

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

The screenshot shows the WSO2 API Store homepage. On the left, there's a sidebar with links for APIS, APPLICATIONS, FORUM, STATISTICS, and TAGS. Under TAGS, the word "pizza" is selected. The main area is titled "APIs" and displays two API cards:

- PizzaShackAPI**: Version 1.0.0, created by Jane Roe. It has a 5-star rating. The thumbnail image shows a slice of pizza with toppings like mushrooms and olives.
- PhoneVerification**: Version 2.0.0, created by apicreator. It has a 5-star rating. The thumbnail image is a large red square with a white letter "P".

Subscribing to the API

1. Go to the API Store (<https://<hostname>:9443/store>) and create an account using the **Sign-up** link.

The screenshot shows the WSO2 API Store interface with the "Sign Up" button highlighted in a red box. The page layout is identical to the previous screenshot, showing the sidebar with the "pizza" tag selected and the two API cards below it.

Users who **sign-up** through the API Store are assigned the **subscriber** role by default. Therefore, you do not need to specify the role through the management console to be able to subscribe to an API.

2. Fill the details in the Sign Up form appears and click **Sign Up**.

Users who registered with the API Store Signup can be view by login to the Management Console (<https://localhost:9443/carbon>) and accessing Users and **Roles > Users > List**.

Details entered in the sign up will be updated in the default profile related to each user in the management console.

3. After signing up, sign in to the API Store and click the PhoneVerification 2.0.0 API that you published earlier.
4. Note that you can now see the subscription options. Select the default application and the Bronze tier. Click **Subscribe**.

PhoneVerification - 2.0.0

Version: 2.0.0
By: apicreator
Updated: 22/Jul/2016 21:27:16 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications
DefaultApplication

Tiers
Bronze

Subscribe

- Once the subscription is successful, click **View Subscriptions** in the information message that appears to review your subscriptions.

Subscription Successful

You have successfully subscribed to the API.

[View Subscriptions](#)

[Stay on this page](#)

- Click the **Production Keys** tab of the application and then click **Generate Keys** to generate an access token that you use later to invoke the API. If you have already generated keys before, click **Re-generate**.

WSO2 API Store

APPLICATIONS APPLICATION LIST EDIT

APIS FORUM STATISTICS

DefaultApplication

Details Production Keys Sandbox Keys Subscriptions

No Keys Found
No keys are generated for this type in this application.

Grant Types
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

SAML2 IWA-NTLM Implicit Refresh Token
 Client Credential Code Password

Callback URL

Access token validity period
 Seconds.

Generate keys

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

- 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

The screenshot shows the WSO2 API Manager interface. At the top, there's a large red 'P' logo. To its right, detailed information about the 'PhoneVerification' API version 2.0.0 is displayed, including its creator ('apicreator'), update date ('22/Jul/2016 21:27:16 PM IST'), status ('PUBLISHED'), and rating ('5 stars'). On the right side, there are dropdown menus for 'Applications' and 'Tiers', and a 'Subscribe' button. Below this, a navigation bar has tabs for 'Overview', 'API Console' (which is highlighted with a red box), 'Documentation', and 'Forum'. Under the 'API Console' tab, there are sections for 'Try' (set to 'DefaultApplication') and 'Using' (set to 'Production'). A 'Set Request Header' section shows an 'Authorization' header with the value 'Bearer 1200d184-1c1f-36b3-ad5f-50015945f008'. At the bottom, there's a 'Swagger (/swagger.json)' link and some visibility options.

- Expand the GET method of the resource `CheckPhoneNumber`. Note the parameters that you added when creating the interactive documentation now appear with their descriptions so that as a subscriber, you know how to invoke this API.

The screenshot shows the 'CheckPhoneNumber' resource under the 'default' endpoint. It lists two methods: 'GET /CheckPhoneNumber' and 'POST /CheckPhoneNumber'. The 'POST' method is highlighted with a green background. Below the methods, a note says '[BASE URL: /phoneverify/2.0.0 , API VERSION: 2.0.0]'. The 'GET /CheckPhoneNumber' method is expanded, showing its parameters. Two parameters are listed: 'PhoneNumber' (required) and 'LicenseKey' (required). Both parameters have their descriptions and data types ('string') displayed. Below the parameters, there's a section for 'Response Messages' with a table showing HTTP status codes (200), reasons, response models, and headers. A 'Try it out!' button is also present.

- Give sample values for the `PhoneNumber` and `LicenseKey` and click **Try it out!** to invoke the API.

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			
Try it out!			

4. Note the response for the API invocation. Since 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/PhoneVe
rify/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 />
```

Response Code

```
200
```

Response Headers

```
{
    "pragma": "no-cache",
    "content-type": "text/xml; charset=utf-8",
    "cache-control": "no-cache",
    "expires": "-1"
}
```

You have invoked an API using the API Console.

Troubleshooting

When using the API Console, the web browser sends an HTTPS request to the Gateway. If the certificate in the Gateway is not CA signed, the browser will not accept it. Therefore, you may get the following error.

```
ERROR - SourceHandler I/O error: Received fatal alert: unknown_ca
javax.net.ssl.SSLException: Received fatal alert: unknown_ca
```

As a workaround, you can access the Gateway URL on a new browser tab and trust the certificate from the

browser.

Monitoring APIs and viewing statistics

Both the API publisher and store provide several statistical dashboards.

A P I P u b l i s h e r s t a t i s t i c s

Statistics

APIs

 Published APIs Over Time

 API Usage

 API Response Times

 API Last Access Times

 Usage by Resource Path

 Usage by Destination

 API Usage Comparison

 API Throttled Requests

 Faulty Invocations

 API Latency Time

 API Usage Across Geo Locations

 API Usage Across User Agent

Applications

 App Throttled Requests

 Applications Created Over Time

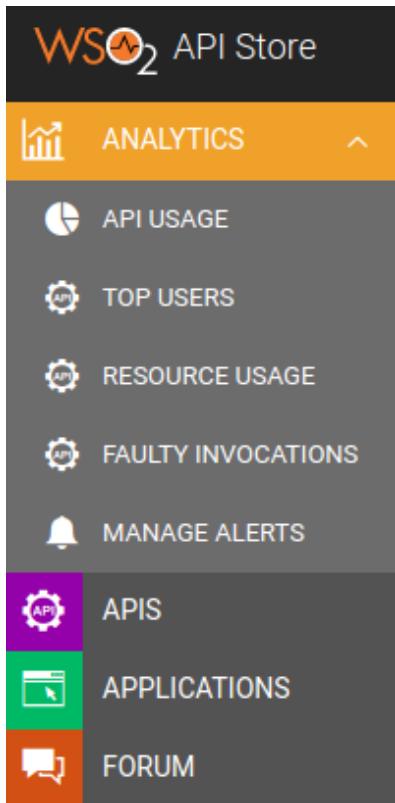
Subscriptions

 API Subscriptions

 Developer Signups Over Time

 Subscriptions Created Over Time

API Store statistics



The steps below explain how to configure WSO2 API Manager Analytics with the API Manager. The statistics in these dashboards are based on data from WSO2 APIM Analytics which is similar to WSO2 Data Analytics Server (DAS).

Let's do the configurations first.

Before you begin,

1. Download the WSO2 APIM Analytics distribution by clicking **ANALYTICS** in the [WSO2 API Management page](#). It is best to download and extract it to the same directory to which you downloaded WSO2 API Manager.
2. If you have the API Manager server running, stop the server.
3. If you are running on Windows, download the `snappy-java_1.1.1.7.jar` from [here](#) and copy the JAR file to the `<ANALYTICS_HOME>\repository\components\lib` directory.

1. To enable Analytics, open the `<API-M_HOME>/repository/conf/api-manager.xml` file and set the `Enabled` property under `<Analytics>` to true as shown below. Save this change.

```
<Enabled>true</Enabled>
```

2. Open the `<API-M_HOME>/repository/conf/log4j.properties` file. Add `DAS_AGENT` to the end of the `log4j.rootLogger` property as shown in the example below.

```
log4j.rootLogger=ERROR, CARBON_CONSOLE, CARBON_LOGFILE, CARBON_MEMORY,  
CARBON_SYS_LOG, ERROR_LOGFILE, DAS_AGENT
```

3. Start the WSO2 APIM Analytics server, and then start the WSO2 API Manager server.

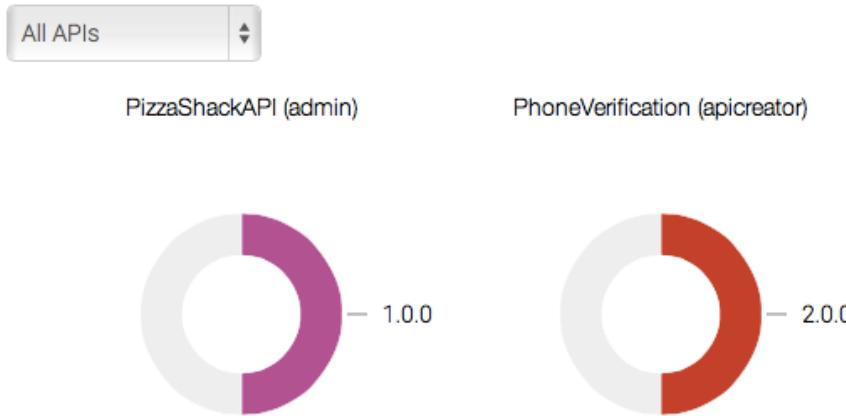
To avoid connection errors during API Manager startup, start WSO2 APIM Analytics before WSO2 API Manager.

- a. On Windows: `wso2server.bat --run`
- b. On Linux/Mac OS: `sh wso2server.sh`

By default, WSO2 API Manager has a port offset of 0 (no port offset) and WSO2 API Manager Analytics has an offset of 1.

4. Invoke several APIs to generate some statistical data and wait a few seconds.
5. Connect to the API Publisher as a creator and click one of the statistical dashboards available in the **Statistics** menu. For example,

Overall API Subscriptions (Across All Versions)



The **Statistics** menu is available for API creators and shows statistics of all APIs. Additionally, API creators can also see the following:

- Statistics of the APIs created by them by selecting the **My APIs** option in the drop down menu above each table or graph.
- The subscriptions of each API by clicking **Manage Subscriptions**.
- The alerts that can be configured for their APIs by clicking **Manage Alert Types**.

This concludes the API Manager quick start. You have set up the API Manager and gone through the basic use cases of the product. For more advanced use cases, see the [Tutorials](#), [Deep Dive](#) and [Admin Guide](#) of the API Manager documentation.

Key Concepts

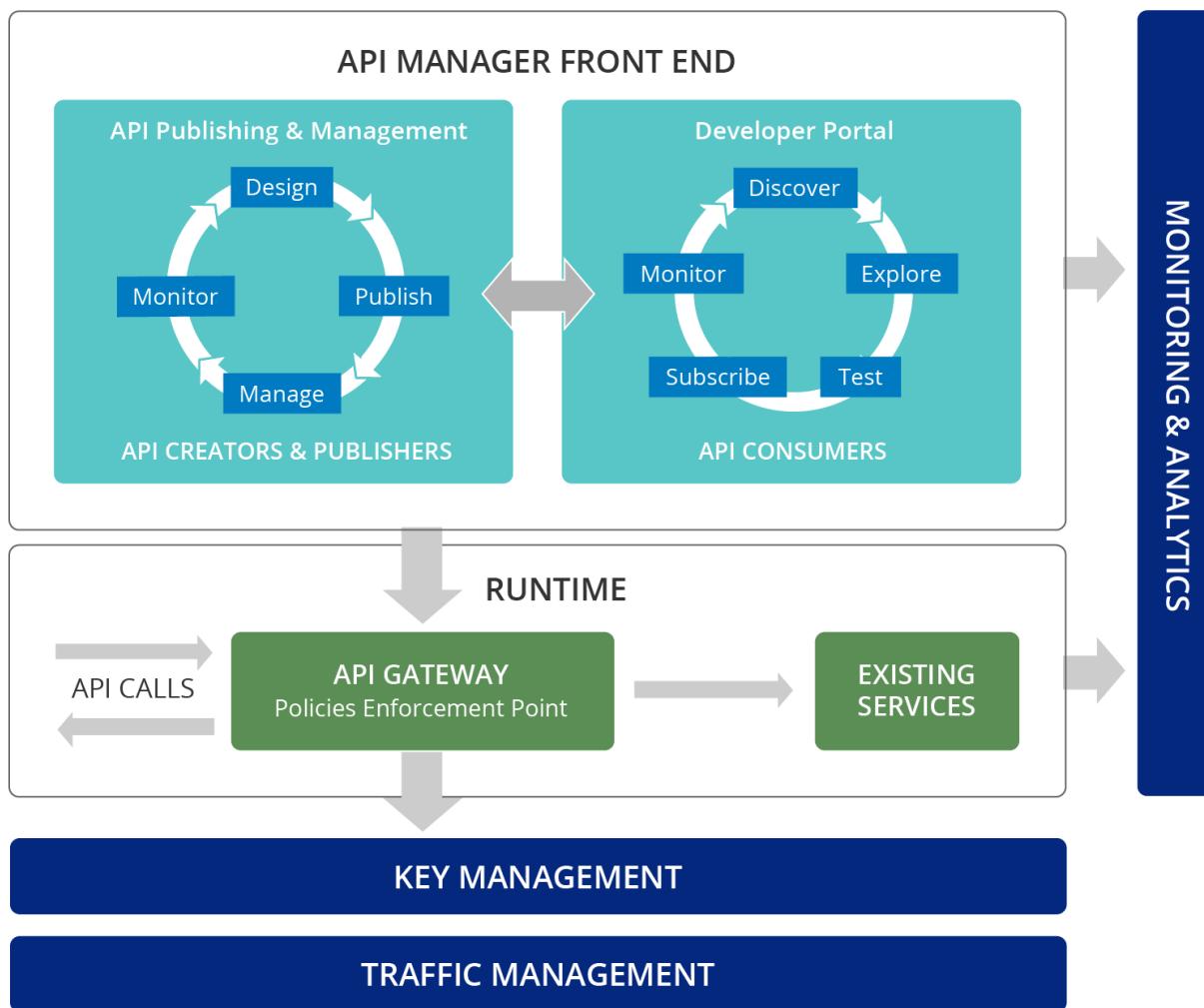
Let's take a look at some concepts and terminology that you need to know in order to follow the use cases.

[API Manager components][Handlers][Users and roles][API lifecycle][Applications][Callback URL][Access tokens][Throttling tiers][Multi tenanted API management][API visibility and subscription Availability][API documentation visibility][API resources][HTTP methods][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 consists of the following high-level components as illustrated in the diagram and listed below:



[API Publisher][API Store (Developer Portal)][API Gateway][Key Manager][Traffic Manager][Analytics]

API Publisher

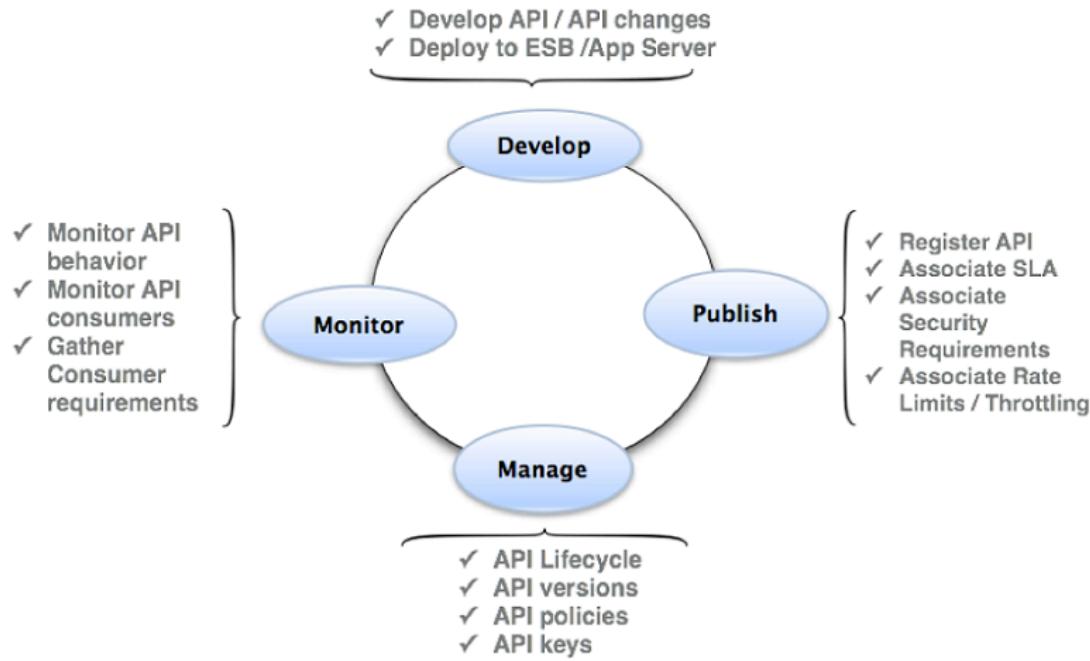
API development is usually done by someone who understands the technical aspects of the API, interfaces, documentation, versions etc., while API management is typically carried out by someone who understands the business aspects of the APIs. In most business environments, API development is a responsibility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

from API publication and management.

WSO2 API Manager provides a simple Web interface called **WSO2 API Publisher** for API development 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 API Publisher URL is `https://<YourHostName>:9443/publisher` and it is accessible on HTTPS only. The default credentials are admin/admin.

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

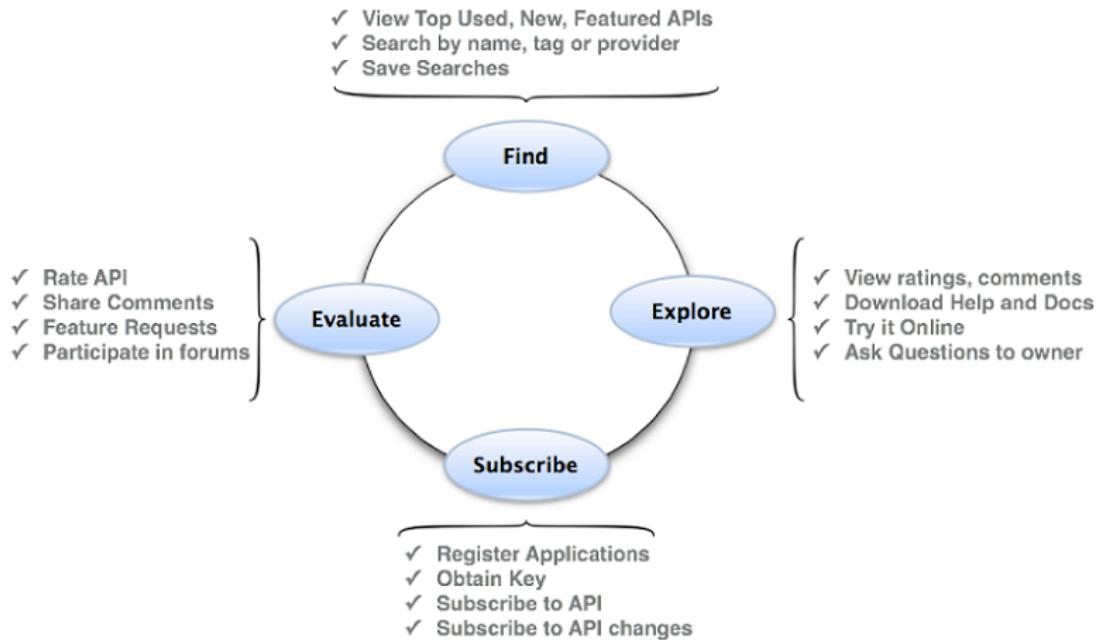


API Store (Developer Portal)

The API Store Web application provides a collaborative interface for API publishers to host and advertise their APIs and for API consumers to **self register**, discover, evaluate, subscribe to and use secured, protected, authenticated APIs.

The API Store URL is `https://<YourHostName>:9443/store` and it is accessible on HTTPS only. The default credentials are admin/admin.

The diagram below shows common API consumer lifecycle activities:



API Gateway

A runtime, backend component (an [API proxy](#)) developed using [WSO2 ESB](#). API Gateway secures, protects, manages, and scales API calls. It intercepts API requests, applies policies such as throttling and security using handlers, and manages API statistics. Upon validation of a policy, the Gateway passes web service calls to the actual backend. If the service call is a token request, the Gateway passes it directly to the [Key Manager](#).

When WSO2 API Manager is running, you can access the Gateway using the following URL: <https://localhost:9443/carbon>. You integrate a monitoring and analytics component to the API Manager by [configuring WSO2 API Manager Analytics](#). This component provides reports, statistics and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and, carry out detailed analysis of the logs.

Although the API Gateway contains ESB features, it is recommended not to use it for ESB-specific tasks. Use it only for Gateway functionality related to API invocations. For example, if you want to call external services like SAP, use a separate [ESB cluster](#) for that purpose.

Key Manager

Manages all clients, security and access token-related operations. The Gateway connects with the Key Manager to check the validity of [OAuth](#) tokens, subscriptions and API invocations. When a subscriber creates an application and generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth client and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database.

The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway. All tokens used for validation are based on the OAuth 2.0.0 protocol. Secure authorization of APIs is provided by the OAuth 2.0 standard for key management. The API Gateway supports API authentication with OAuth 2.0, and enables IT organizations to enforce rate limits and throttling policies.

The Key Manager properly decouples the operations for creating OAuth applications and validating access tokens so that you can even plug in a third party-authorization server for key validations.

You can avoid making the Gateway connect with the Key Manager every time it receives an API invocation call, by enabling API Gateway [caching](#). When caching is not enabled, a verification call happens every time the Gateway receives an API invocation call. For this verification, the Gateway passes an access token, the API, and API version

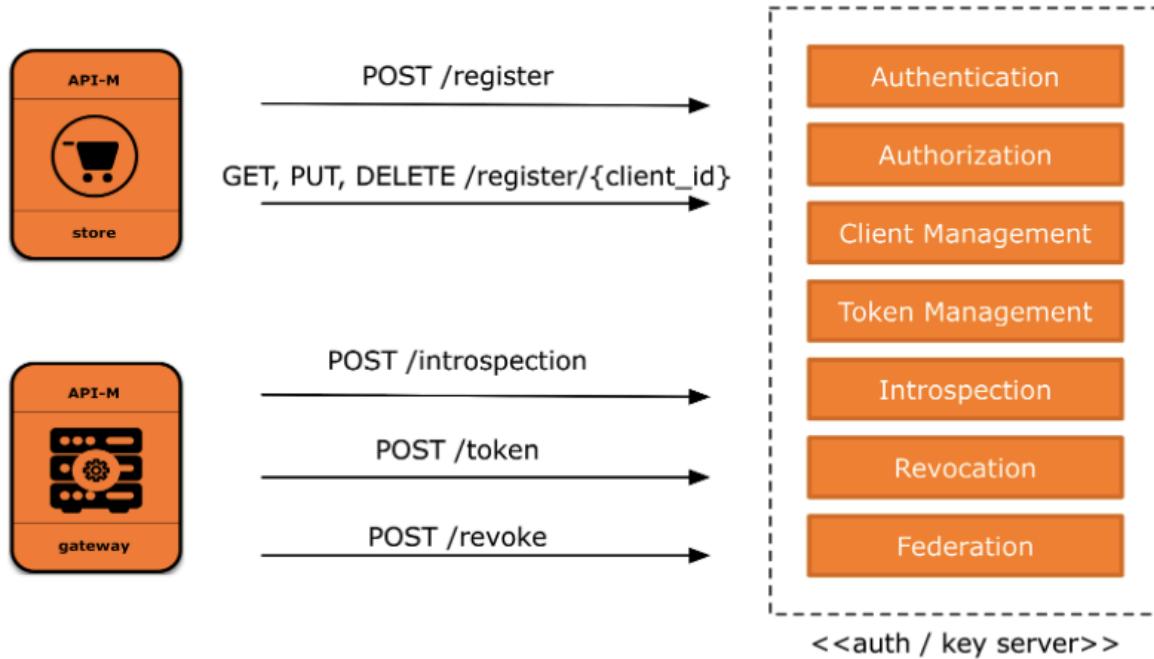
to the Key Manager. Communication between the API Gateway and the Key Manager happens in either of the following ways:

- Through a Web service call
- Through a **Thrift** call (Thrift is the default communication protocol and is much faster than SOAP over HTTP)

If your setup has a cluster of multiple Key Manager nodes that are fronted by a **load balancer** that does not support Thrift, change the key management protocol from **Thrift** to **WSClient** using the `<KeyValidatorClientType>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file. Thrift uses **TCP** load balancing.

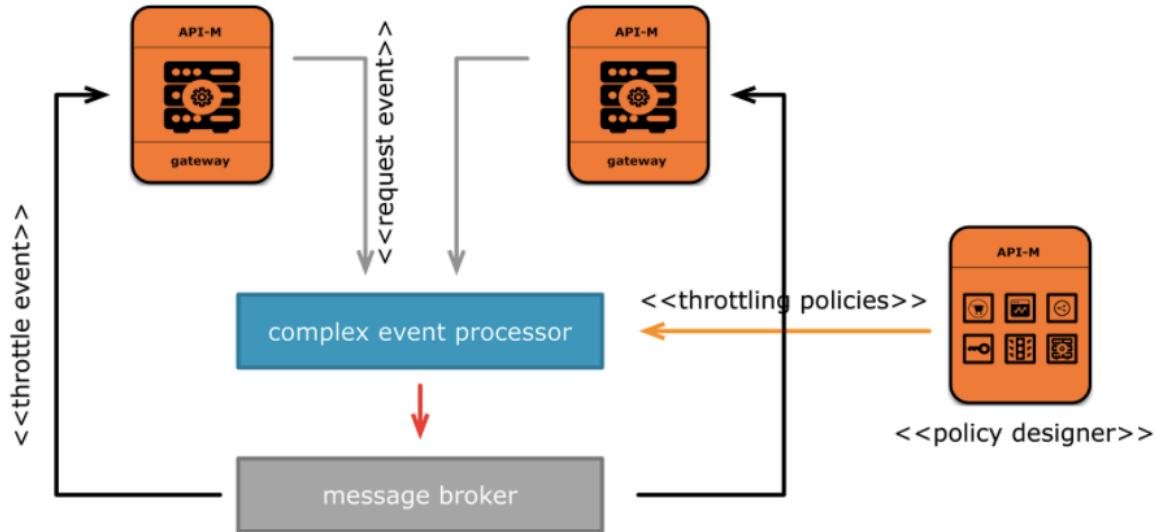
In a typical production environment, you can configure one of the following setups:

- Configure a WSO2 API Manager instance as the Key Manager in a separate server. See [Product Profiles](#).
- Configure an instance of WSO2 Identity Server as the Key Manager. See [Configuring WSO2 Identity Server as the Key Manager](#).
- Configure a third-party authorization server for key validations and an API Manager instance for the rest of the key management operations. See [Configuring a Third-Party Key Manager](#).



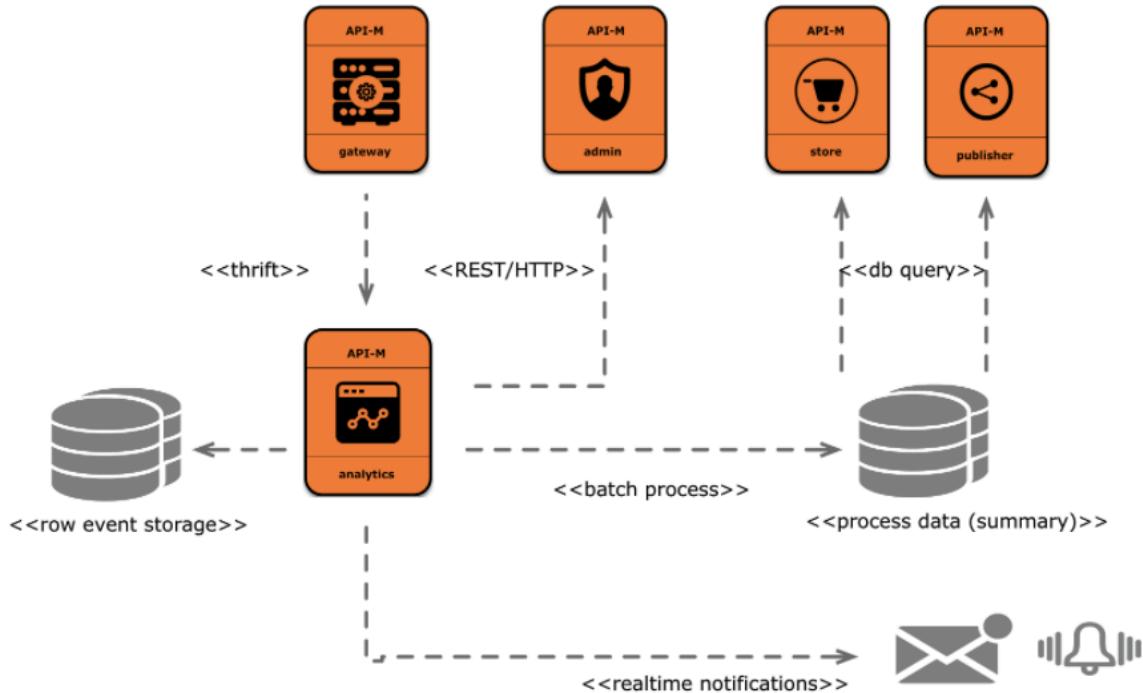
Traffic Manager

The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests. For more information, see [Working with Throttling](#).



Analytics

Additionally, monitoring and analytics are provided by the analytics component, WSO2 API Manager Analytics. This component provides a host of statistical graphs, an alerting mechanism on pre-determined events and a log analyzer. For more information, see [Analytics](#).



Handlers

When an API is created, a file with its synapse configuration is added to the API Gateway. You can find it in the <AP> folder. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration. You find the default handlers in any API's Synapse definition.

For a detailed description of handlers and how to write a custom handler, see [Writing Custom Handlers](#).

Users and roles

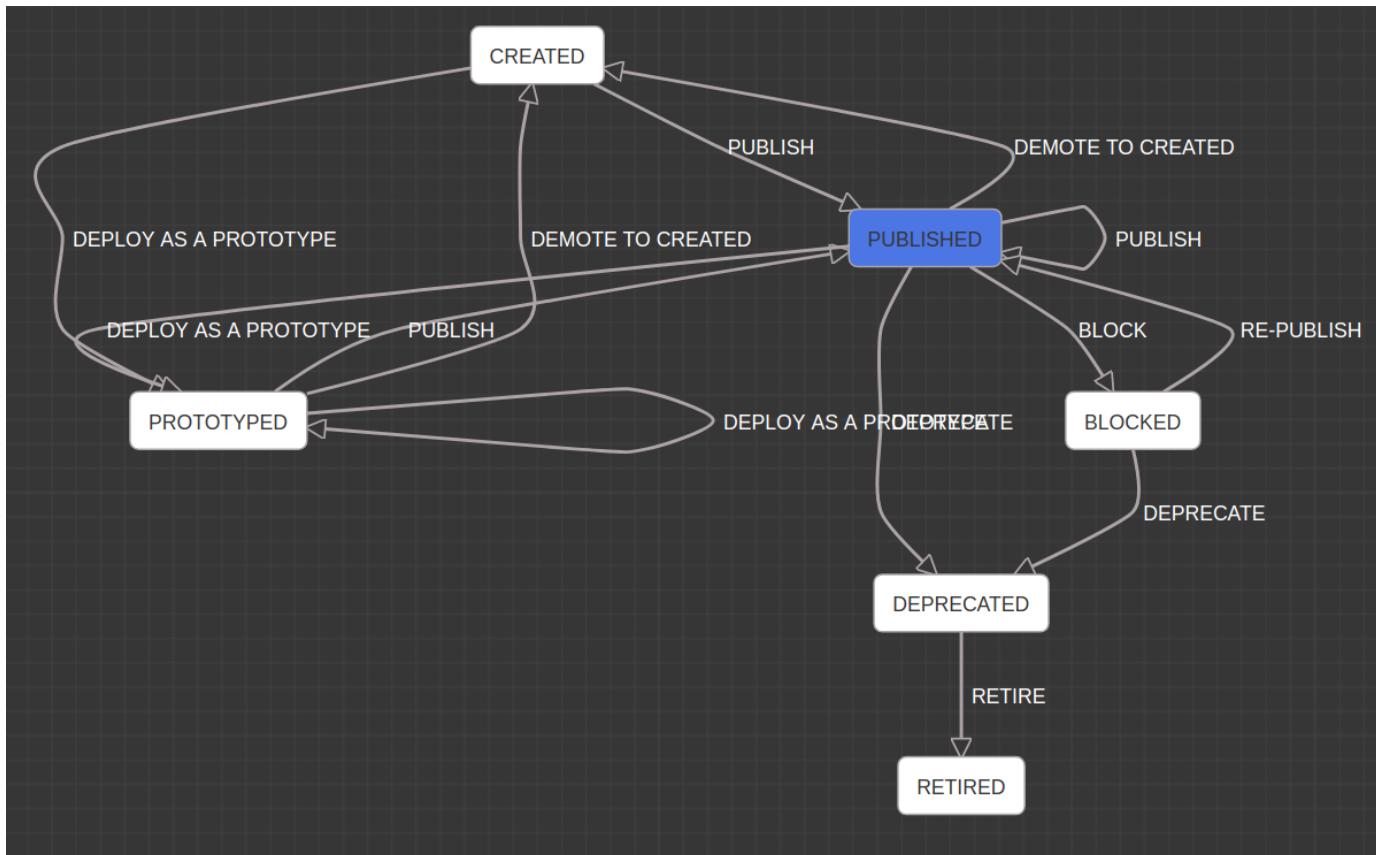
The API Manager offers four distinct community roles that are applicable to most enterprises:

- **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 the 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:

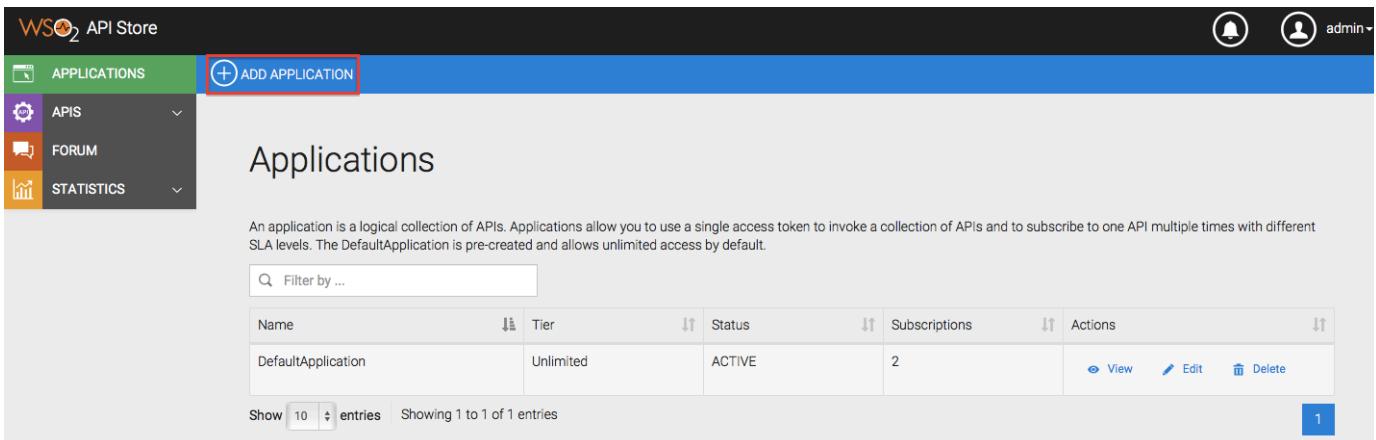
- **CREATED:** API metadata is added to the API Store, but it is not deployed in the API Gateway and therefore, is not visible to subscribers in the API Store.
- **PROTOTYPED:** The API is deployed and published in the API Store as a prototype. A prototyped API is usually a mock implementation made public 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 tiers/Service Level Agreement (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.



Name	Tier	Status	Subscriptions	Actions
DefaultApplication	Unlimited	ACTIVE	2	View Edit Delete

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

Callback URL

A callback URL is a URL that sends a callback to a specific server or program soon after your application request is sent. The callback URL is specified when generating or re-generating production or sandbox keys for an application. You can either provide a single application callback URL or a RegEx pattern as the callback URL. You use a RegEx pattern as the callback URL when you need to specify multiple callback URLs for an application.

For example, consider a situation where you have two (2) service providers that need to use the same application that have the following callback URLs:

- <https://mytestapp.com/callback>
- <https://testapp:8000/callback>

In this instance, your callback URL should have the following RegEx pattern:

```
regexp=(https://mytestapp.com/callback|https://testapp:8000/callback)
```

You can configure any RegEx pattern to match the callback URLs that you need to register with the application. However, it is mandatory to have the prefix `regexp=` before the pattern.

Access tokens

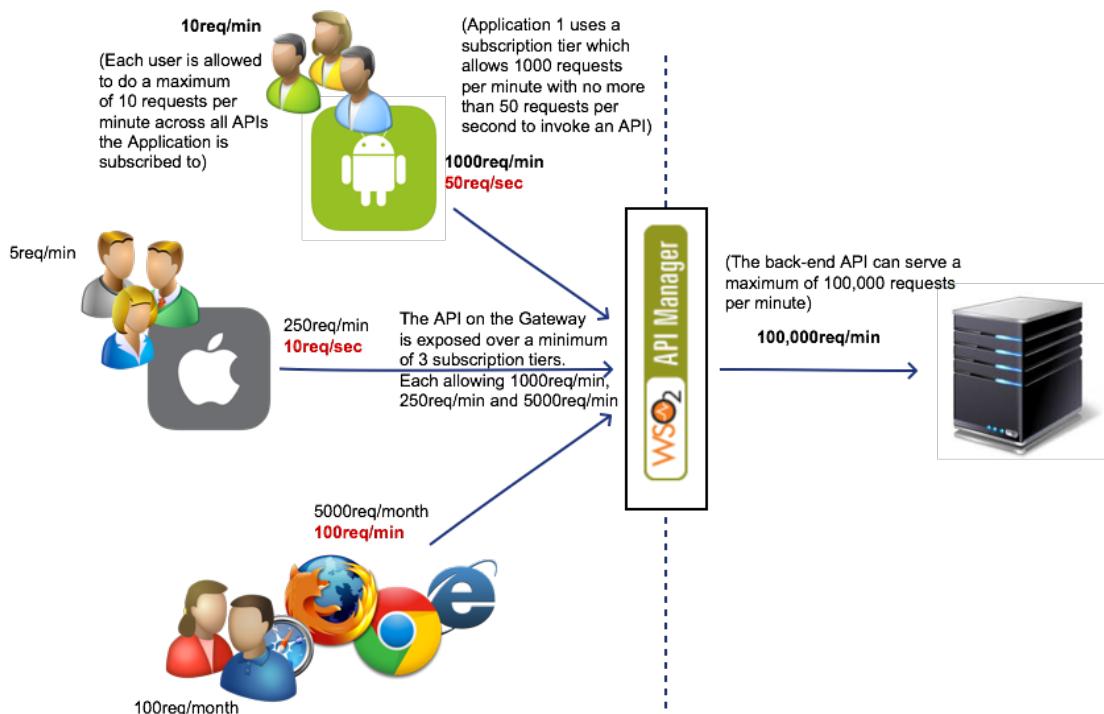
An **access token** is a simple string that is passed as an HTTP header of a request. For example, "Authorization : Bearer NtBQkXoKElu0H1a1fQ0DWf06IX4a." Access tokens authenticate API users and applications, and ensure better security (e.g., prevent certain types of **DoS attacks**). Note that DoS attacks made to the key manager with random access tokens can not be prevented. DoS attacks with the same fake access token can affect the Gateway as well). If a token that is passed with a request is invalid, the request is discarded at the first stage of processing. Access tokens work equally well for SOAP and REST calls.

For more information on different types of access tokens and how to generate them, see [Working with Access Tokens](#).

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 certain types of Denial of Service (DoS) attacks
- 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 purposes



You can define throttling at the API, application, and resource 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. For more information about throttling, see [Setting Throttling Limits](#).

Multi tenanted API management

A tenant in WSO2 API Manager is a separate business level entity, such as an organizational unit or a department. Multi tenancy enables such organizational units/departments to share the same API Manager deployment and the respective resources, but function individually with an isolated view of the deployment. A tenant does not need to be aware of the other tenants in the system as by design the Multitenancy creates an isolated space for each tenant, although they are sharing the same deployment.

See the section [Configuring Multiple Tenants](#) for information on how to create tenants.

In WSO2 API Manager deployment, API Visibility and Subscription Availability are the two main applications of tenancy.

API visibility

API Manager allows users to control API visibility and subscription availability. API visibility can be one of the following options:

- Public
- Restricted by roles
- Visible to my domain

Subscription availability

Subscription availability has three options. Those options are as follows:

- Available to current Tenant Only
- Available to All the Tenants
- Available to Specific Tenants

See the next section on [API Visibility and Subscription](#) for more details on each category.

Additionally, it's possible to configure a secondary userstore per tenant as well. For more information, see [Configuring Secondary Userstores](#).

API visibility and subscription Availability

API visibility

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

- **Public:** The API is visible to all users who are registered and anonymous (who use APIs without login to the store, for example testing and demonstration), and can be advertised in multiple stores (central and non-WSO2 stores).
- **Restricted by Roles:** The API is visible to its tenant domain and only to the user roles that you specify. You should provide the roles separated by commas in the UI or as a cURL parameter when creating or editing the API.
- **Visible to my domain:** The API is visible to all users who are registered to the API's tenant domain. This option is **available only in a multi-tenanted environment**. It's not applicable when there is only one active tenant (super tenant) in the system.

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

- The **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 the visibility set 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.

By default, an API has public visibility. You can set the API visibility in the **Design** tab of the API Publisher at the time the API is created or updated.

Design API

The screenshot shows the 'Design' tab of the API Designer. In the 'General Details' section, the 'Visibility' dropdown is open, displaying three options: 'Public' (selected), 'Visible to my Domain' (highlighted in orange), and 'Restricted by roles'. Other fields shown include 'Name' (PhoneVerification), 'Context' (/phoneverify), 'Version' (1.0.0), and a 'Description' field with a placeholder 'Maximum 20000 characters.'

When using the REST API directly, these visibility options are available as **public**, **private** and **restricted**.

API visibility level specified in the UI	API visibility level specified in the REST API
Public	public i.e. visibility=public
Visible to my domain	private i.e. visibility=private
Restricted by roles	restricted i.e. visibility=restricted&roles=role1,role2,role3

Subscription availability

The subscription availability option has three values as follows. You can set subscription availability to an API through the API Publisher's **Manage** tab.

- **Available to current tenant only:** only users in the current organization/tenant domain can subscribe to the API.
- **Available to all the tenants:** users of all organizations/tenant domains can subscribe to the API.
- **Available to specific tenants:** users of the organizations/tenant domains that you specify, as well as the current tenant domain, can subscribe to the API.

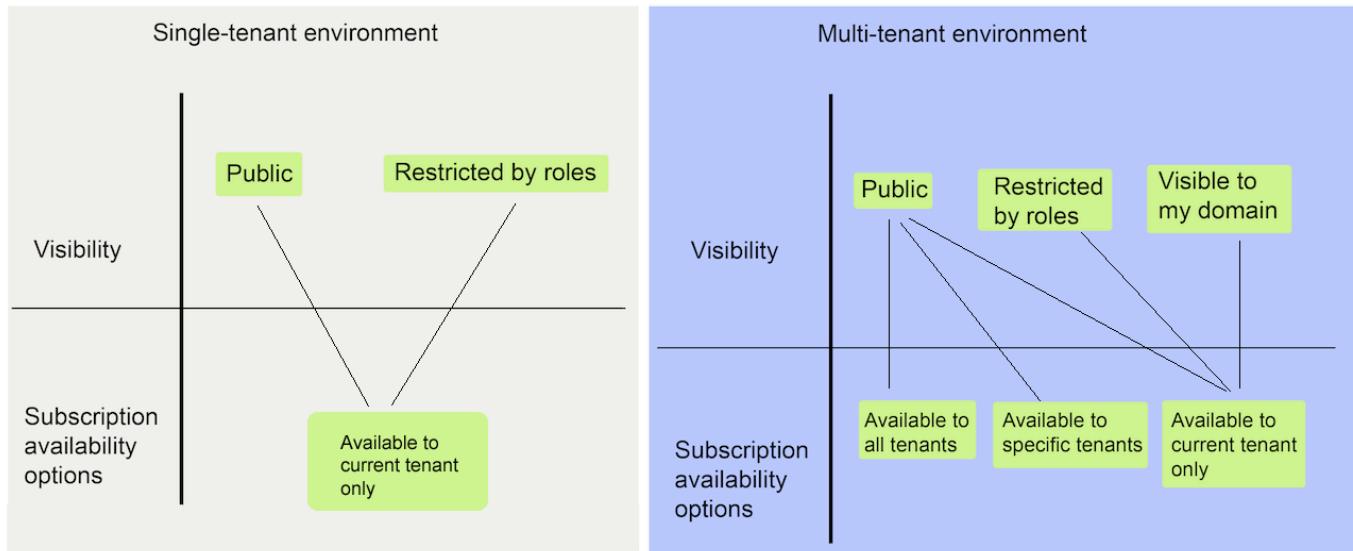
PhoneVerification : /phoneverify/1.0.0

The screenshot shows the 'Design' tab of the API configuration interface. Under 'Configurations', there is a note about the default version and transport settings (HTTPS and HTTP selected). The 'Subscriptions' dropdown is open, showing options: 'Available to current tenant only' (selected), 'Available to all the tenants', 'Available to specific tenants', and 'Visible to my domain'. In the 'Throttling Settings' section, it shows 'Unlimited' maximum backend throughput and subscription tiers: Unlimited (selected), Gold, Silver, and Bronze.

Subscription is only available to the current tenant in the following instances:

- When there is only one tenant in your system.
- Even if there are multiple tenants in your system, when you select **Visible to my domain** or **Restricted by roles** as the API's visibility in the previous step.

The diagram below depicts the relationship between the API's visibility and subscription availability:



Refer the article [Multi Tenant API Management with WSO2 API Manager](#) for examples and real world usage of the above concepts.

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 the `<API-M_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, click the **Docs** tab of an API, and click **Add New Document** to see a new drop-down list added to select visibility from:

PhoneVerification - 1.0.0

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 of which handles a particular type of request. A resource has a set of methods that operate on it. The methods are analogous to a method or a function, and a resource is analogous to an object instance or a class in an object-oriented programming language. There are a few standard methods defined for a resource (corresponding to the standard HTTP GET, POST, PUT and DELETE methods.)

The diagram below shows a resource by the name `CheckPhoneNumber` added with four HTTP methods.

API Definition

URL Pattern

/phoneverify/1.0.0

Url Pattern E.g.: path/to/resource

 GET POST PUT DELETE PATCH HEAD **more**

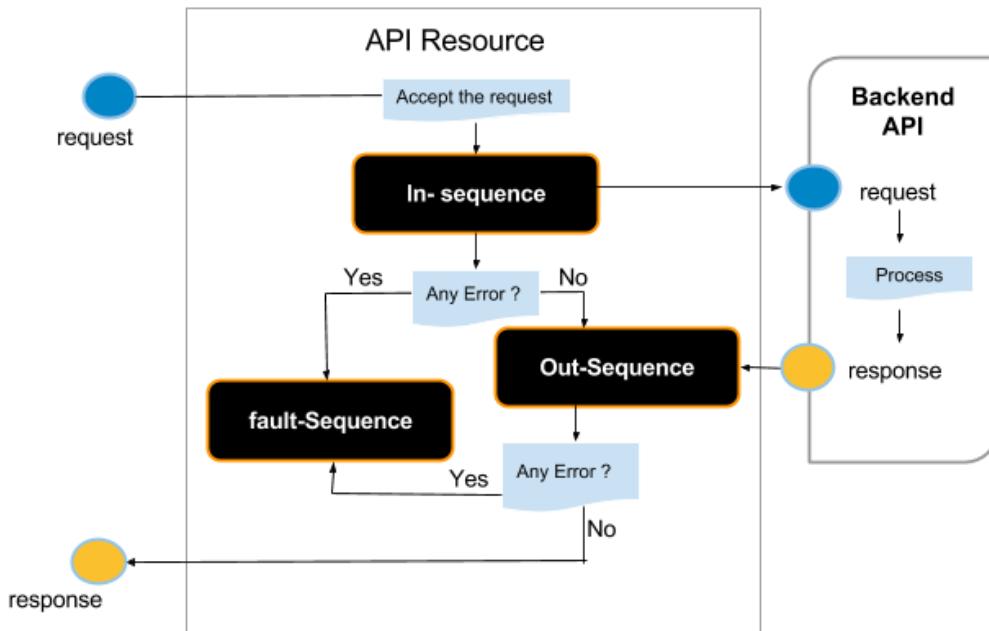
Add

GET	/CheckPhoneNumber	+ Summary	
POST	/CheckPhoneNumber	+ Summary	
PUT	/CheckPhoneNumber	+ Summary	
DELETE	/CheckPhoneNumber	+ Summary	

When you add resources to an API, you define a URL pattern and **HTTP methods**. A resource can also have a list of **OAuth scopes**.

<p>URL Pattern</p>	<p>A URL pattern can be one of the following types:</p> <ul style="list-style-type: none"> • As a url-mapping. E.g., /state/town/* • As a uri-template. E.g., /{state}/{town} <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 data-bbox="861 502 1449 734"><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 texas and the value of town to houston. 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>
---------------------------	---

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 diagram below shows the flow of a request sent to an API resource.



The default in-sequence, out-sequence and fault sequences are generated when the API is published.

HTTP methods

HTTP methods specify the desired action to be performed on an API's resource. You can select multiple methods from **GET**, **POST**, **PUT**, **DELETE**, **PATCH**, **HEAD** and **OPTIONS**. A method has attributes such as an OAuth scope, authentication type, response content type, parameters etc. as the diagram below shows:

The screenshot shows the 'API Definition' section of the WSO2 API Manager interface. It includes fields for 'URL Pattern' (set to '/phoneverify/1.0.0') and 'HTTP Methods' (checkboxes for GET, POST, PUT, DELETE, PATCH, HEAD, more). Below these are sections for 'Description', 'Parameters', and a table of 'Operations'.

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

Operations table:

Method	Path	Summary	Actions
POST	/CheckPhoneNumber	+ Summary	
PUT	/CheckPhoneNumber	+ Summary	
DELETE	/CheckPhoneNumber	+ Summary	

The main attributes of a method are described below:

OAuth scopes	You can define a list of OAuth scopes to an API's resource and assign one of them to each HTTP method.
Authentication type	<p>The authentication type can be 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 (Cross Origin Resource Sharing) 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>
Response content type	You can use this attribute to document in what type the backend sends the response back to the API Manager. Note that this attribute doesn't do any message type conversion , but used simply as a way of letting the user know what type the response will be.
Parameters	Parameters of an HTTP method are analogous to arguments of a function in an object-oriented programming language. A resource's parameters are cached in the resource cache at the API Gateway.

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 the <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-Methods>GET,PUT,POST,DELETE,PATCH,OPTIONS</Access-Control-Allow-Methods>

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

    <Access-Control-Allow-Credentials>false</Access-Control-Allow-Credentials>
</CORSConfiguration>
```

The elements are described below:

Header	Description	Sample values
Access-Control-Allow-Origin	Determines whether a resource can be shared with the resource of a given origin. The API Gateway validates the origin request header value against the list of origins defined under the Access Control Allow Origins configuration(this can be All Allow Origins or a specific value like localhost). If the host is in the allowed origin list, it will be set as the Access-Control-Allow-Origin response header in the response.	All Allow Origins(*), localhost
Access-Control-Allow-Headers	Determines, as part of the response to a preflight request (a request that checks to see if the CORS protocol is understood), which header field names can be used during the actual request. The gateway will set the header values defined under Access Control Allow Headers configurations.	authorization, Access-Control-Allow-Origin, Content-type, SOAPAction
Access-Control-Allow-Methods	This header specifies the method(s) allowed when accessing the resource in response to a preflight request. Required methods can be defined under the Access Control Allow Method configuration.	GET, PUT, POST, DELETE, PATCH, OPTIONS
Access-Control-Allow-Credentials	Determines whether or not the response to the request can be exposed to the page. It can be exposed when the header value is true. The header value can be set to true/false by enabling/disabling the Access Control Allow Credentials configuration.	true, false

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. You can find the API's configuration in the <APIM_HOME>/repository/deployment/server/synapse-configs/default/api folder. Change your code according to the sample given here.

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

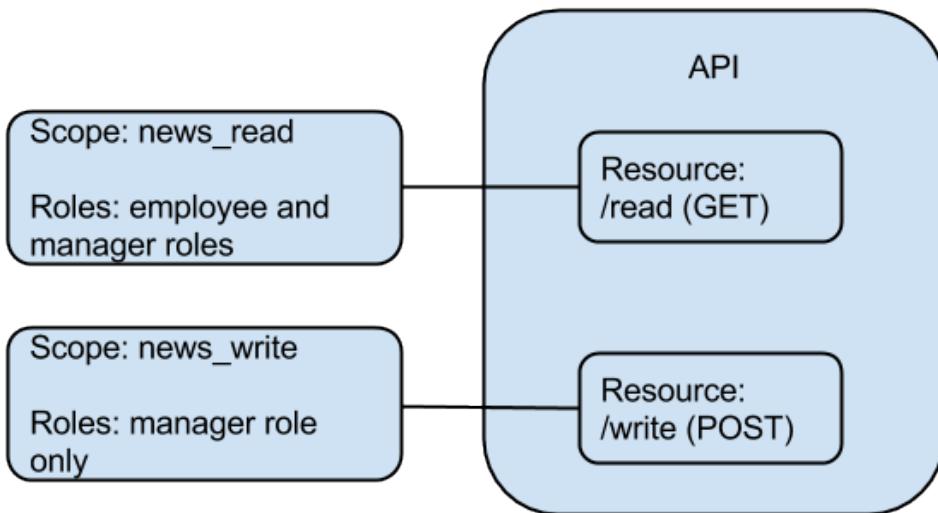
OAuth scopes

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.

OAuth provides a method for clients to access a protected resource on behalf of a resource owner. OAuth 2 bearer token is a security token that any party in possession of it can use the token for authentication. Refer [OAuth 2.0 Specification of Bearer Token Usage](#) for more information.

How scopes work

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



Assume there are two users **Tom** and **John**. Tom is assigned the employee role. John is assigned both employee and manager roles.

Tom requests a token through the Token API as `grant_type=password&username=tom&password=xxxxx&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>
```

Applying a scope

You 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 under **Resources**.

The screenshot shows the 'Manage' tab in the API Publisher. Under the 'Resources' section, there is a table with two rows. The first row has 'GET' and 'POST' columns, each with a 'Scopes:' dropdown and a '+ Add Scope' button. The second row has 'Scopes:' and '+ Add Scope' columns. The '+ Add Scope' button in the first row is highlighted with a red box.

Scopes:	+ Add Scope
GET /CheckPhoneNumber + Summary	Application & Application User Unlimited
POST /CheckPhoneNumber + Summary	Application & Application User Unlimited

On the screen that appears, enter a scope key, scope name and optionally, allowed roles and a description. Click **Add Scope**.

Define Scope

Scope Key * news_read

Scope Name * Read News

Roles employee, manager

Description Eg: This scope will group all the administration APIs

Add Scope **Close**

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	<p>The user role(s) that are allowed to obtain a token against this scope. E.g., manager, employee.</p> <p>Note that the role name is case sensitive in the DBMSs that are case sensitive, such as PostgreSQL.</p> <p>When the role you specify is in a secondary user store, you have to give the role as <userstore name>/<role name>.</p>

To apply the scope, you add the scope to a resource, save and publish the API.

GET	/CheckPhoneNumber + Summary	Application & Application User	Unlimited	+ Scope
POST	/CheckPhoneNumber + Summary	Application & Application User	Unlimited	+ Scope

Tip: When you generate access tokens for applications with APIs protected by scope/s in the API Store, a **scopes** drop down list is displayed in the **Production Keys** tab of the application, where you can select the scope/s after the token is generated.

Generating Access Tokens

The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://[REDACTED]:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://[REDACTED]:8243/token
```

Generate a Test Access Token

Access Token

.....

Above token has a validity period of **3600** seconds. And the token has (**am_application_scope,default**) scopes.

Scopes	
Read News	
Validity period	
3600	Seconds.
Re-generate	

For a complete example, please refer the article : [An Overview of Scope Management with WSO2 API Manager](#)

Scope whitelisting

A scope is not always used for controlling access to a resource. You can also use it to simply mark an access token. There are scopes that cannot be associated to roles (e.g., openid, device_). Such scopes do not have to have roles associated with them. Skipping role validation for scopes is called scope whitelisting.

If you do not want a role validation for a scope in an API's request, add the scope under the `<OAuthConfigurations>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file and restart the server. It will be whitelisted. For example,

```
<ScopeWhitelist>
    <Scope>^device_.*</Scope>
    <Scope>some_random_scope</Scope>
</ScopeWhitelist>
```

Next, invoke the Token API to get a token for the scope that you just whitelisted. For example,

```
curl -k -d
"grant_type=password&username=admin&password=admin&scope=some_random_scope"
-H "Authorization: Basic
WmRFUFBvZmZwYVFnR25ScG5izldtcUtSS3IwYTpSaG5ocEVJYUVCMEN3T1FReWpiZTJwaDBzcl
Vh" -H "Content-Type: application/x-www-form-urlencoded"
https://10.100.0.3:8243/token
```

Note that the issued token has the scope you requested. You get the token without any role validation as the scope is whitelisted.

```
{ "scope": "some_random_scope", "token_type": "bearer", "expires_in": 3600, "refresh_token": "59e6676db0addca46e68991e44f2b8b8", "access_token": "48855d444db8
83171c347fa21ba77e8" }
```

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, see [Working with Endpoints](#).

Sequences

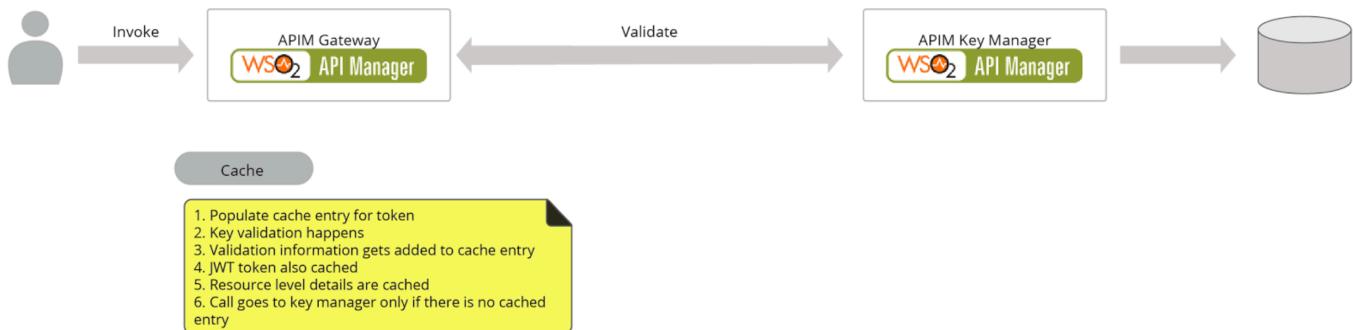
The API Manager has a default mediation flow that is executed in each API invocation. There are 3 default sequences engaged as `in`, `out` and `fault` which perform following.

Sequence	Description
in	In-sequence is the first place that will be mediated through, once a request is dispatched to a resource of an API. At the end of the in-sequence the request can be forwarded to a back-end application for further processing.
out	Any responses coming from the back-end system are mediated through the out-sequence of the resource of the API.

fault	<p>Fault sequence is there to handle any errors that may occur while mediating a message through a resource.</p> <p>When the sequence or the proxy service encounters an error during mediation or while forwarding a message, the message that triggered the error is delegated to the specified fault sequence. Using the available mediators it is possible to log the erroneous message, forward it to a special error-tracking service, and send a SOAP fault back to the client indicating the error. We need to configure the fault sequence with the correct error handling instead of simply dropping messages. For more information, see Error Handling.</p>
-------	--

Caching

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



Tutorials

This section covers the following usecases of the product:

- [Getting Started](#)
- [API Publishing](#)
- [Developer Portal](#)

Getting Started

How do I...

- [Create and Publish an API](#)
- [Subscribe to an API](#)
- [Invoke an API using the Integrated API Console](#)
- [Edit an API Using the Swagger UI](#)

Create and Publish an API

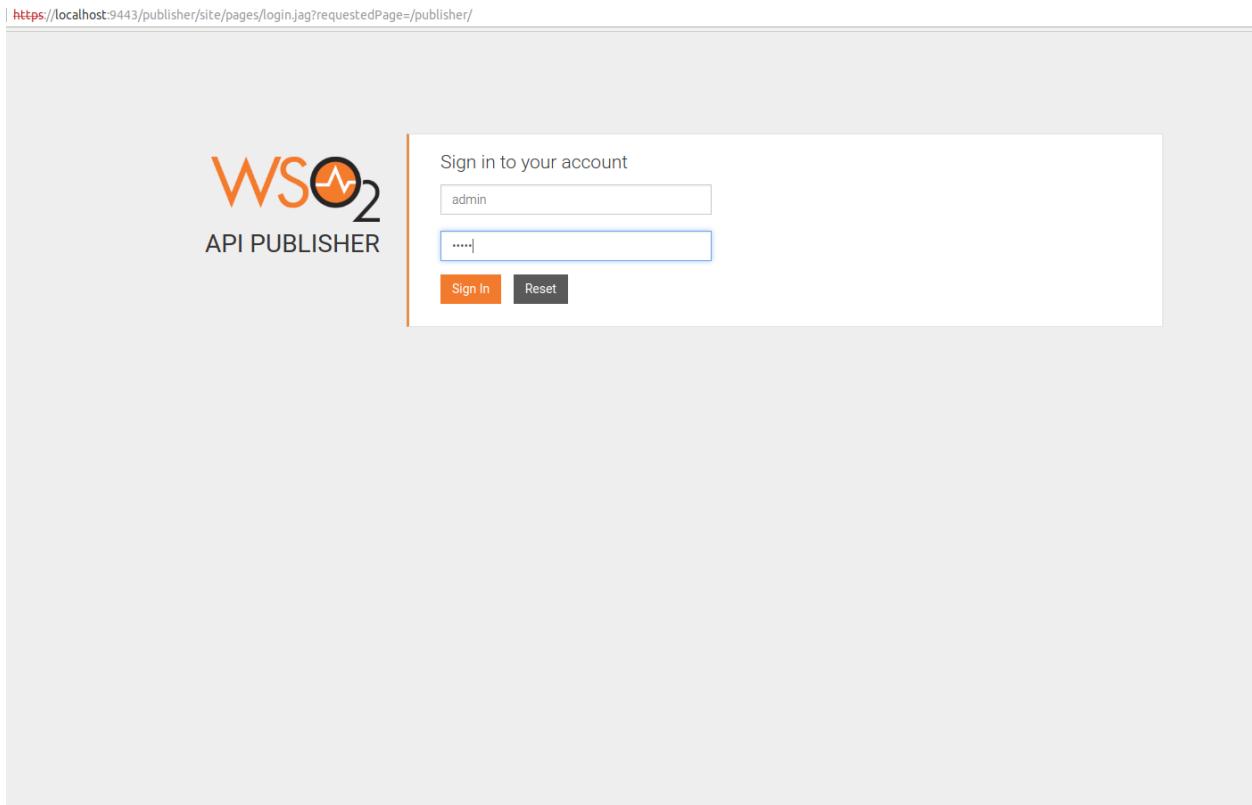
API creation is the process of linking an existing backend API implementation to the [API Publisher](#) so that you can manage and monitor the [API's lifecycle](#), documentation, security, community, and subscriptions. Alternatively, you can provide the API implementation in-line in the [API Publisher](#)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

1. Sign in to the WSO2 API Publisher.

<https://<hostname>:9443/publisher> (e.g., <https://localhost:9443/publisher>). Use **admin** as the username and password.



2. In the **APIS** menu, click **Add New API**.

A screenshot of the WSO2 API Publisher main dashboard. The top navigation bar shows "WSO2 API Publisher", "HOME / APIS", and a "LOG OUT" button. The left sidebar has links for "APIS" (selected), "STATISTICS", and "MANAGE SUBSCRIPTIONS". The main content area is titled "All APIs" and displays a message: "No APIs created yet. Click one of the buttons below to get started." Below this are two buttons: "New API..." and "Deploy Sample API".

3. Select **Design New REST API** and click **Start Creating**.

Let's get started!

Add New API

<input type="radio"/>	I Have an Existing API Use an existing API's endpoint or the API Swagger definition to create an API.	▼
<input type="radio"/>	I Have a SOAP Endpoint Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.	▼
<input checked="" type="radio"/>	Design a New REST API Design and prototype a new REST API.	^
<input style="width: 100%; height: 30px; background-color: #005a9c; color: white; font-weight: bold; border: 1px solid black; padding: 0; margin: 0;" type="button" value="Start Creating"/>		
<input type="radio"/>	Design a New Websocket API Design and prototype a new Websocket API.	▼

4. Give the information in the table below and click **Add** to add the resource.

Field	Sample value		
Name	PhoneVerification		
Context	/phoneverify <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The API context is used by the Gateway to identify the API. Therefore, the API context must be unique. This context is the API's root context when invoking the API through the Gateway.</p> </div>		
	<p>Tip: You can define the API's version as a parameter of its context by adding the {version} into the context. For example, {version}/phone verify. The API Manager assigns the actual version of the API to the {version} parameter internally. For example, https://localhost:8243/1.0.0/phoneverify. Note that the version appears before the context, allowing you to group your APIs based on the versions.</p>		
Version	1.0.0		
Visibility	Public		
Tags	phone, checkNumbers <div style="border: 1px solid #28a745; padding: 10px; margin-top: 10px;"> <p>Tags can be used to filter out APIs matching some search criteria. We recommend adding tags that explain the functionality and purpose of the API. Subscribers can search for APIs based on tags.</p> </div>		
Resources	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">URL pattern</td> <td style="padding: 5px;">CheckPhoneNumber</td> </tr> </table>	URL pattern	CheckPhoneNumber
URL pattern	CheckPhoneNumber		

	Request types	GET, POST
The selection of the HTTP method should match with the actual backend resource. For example, if the actual backend contains the GET method to retrieve details of a phone number, then that resource should match with a GET resource type and with a proper context.		

The screenshot shows the 'Design' tab of the WSO2 API Manager interface. It includes a navigation bar with '1 Design' and '2 Implement' tabs, and a main area for 'General Details'. The 'General Details' section contains fields for Name (PhoneVerification), Context (phoneverify), Version (1.0.0), Visibility (Public), Description (maximum 20000 characters), and Tags (Add tags). Below this is the 'API Definition' section, which includes a URL Pattern (phoneverify/1.0.0/CheckPhoneNumber) and a method selection dropdown showing GET, POST, PUT, DELETE, PATCH, HEAD, and more. A red box highlights the '+ Add' button in the 'API Definition' section.

For more information on URL patterns, see [API Resources](#).

- After you add the resource, click its GET method to expand it. Update the value for **Produces** as application/xml and the value for **Consumes** as application/json.

In the resource definition, we define the MIME types. **Consumes** refers to the MIME type of request accepted by the backend service and **Produces** refers to the MIME type of response produced by the backend service which you define as the endpoint of the API.

- Next, add the following parameters. You use these parameters to invoke the API using our integrated API Console, which is explained 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
------------	---	-------	--------	------

API Definition

URL Pattern Url Pattern E.g.: path/to/resource **Import** **Edit Source**

GET POST PUT DELETE PATCH HEAD **more**

(+ Add)

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

Description : [+ Add Implementation Notes](#)

Produces : application/xml **Consumes : application/json**

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
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](#) (i)

Save **Next: Implement >**

HTTP Post

By design, the HTTP POST method specifies that the web server accepts data enclosed within the body of the request. Therefore, when adding a POST method, API Manager adds the payload parameter to the POST method by default.

Import or Edit API definition

API Definition

URL Pattern Url Pattern E.g.: path/to/resource **Import** **Edit Source**

GET POST PUT DELETE PATCH HEAD **more**

(+ Add)

To import an existing swagger definition from a file or a URL, click **Import**. Click **Edit Source** to manually edit the API swagger definition.

- Once done, click **Next: Implement >**. Alternatively, click **Save** to save all the changes made to the API. You can come back later to edit it further by

selecting the API and clicking **Edit**. For details about the states of the API, see [Manage the API Lifecycle](#).

The following parameter types can be defined according to the resource parameters you add.

Parameter Type	Description
query	Contains the fields added as part of the invocation URL that holds the data to be used to call the backend service.
header	Contains the case-sensitive names followed by a colon (:) and then by its value which carries additional information with the request which defines the operating parameters of the transaction.
formData	Contains a property list of attribute names and values which includes in the body of the message.
body	An arbitrary amount of data of any type which sends with a POST message

You can use the following Data type categories, supported by swagger.

- [primitive](#) (input/output)
- [containers](#) (as arrays/sets) (input/output)
- [complex](#) (as models) (input/output)
- [void](#) (output)
- [File](#) (input)

8. Click the **Managed API** option.

PhoneVerification: /phoneverify/1.0.0

The screenshot shows the WSO2 API Manager interface. At the top, there are three tabs: 'Design' (blue), 'Implement' (orange, currently active), and 'Manage' (grey). Below these tabs, there are two sections: 'Managed API' (selected) and 'Prototyped API'. The 'Managed API' section contains the text: 'Provide the production and sandbox endpoints of the API to be managed.' The 'Prototyped API' section contains the text: 'Use the inbuilt JavaScript engine to prototype the API or provide an endpoint to a prototype API. The inbuilt JavaScript engine does not hav...'. A red box highlights the 'Managed API' section.

9. The **Implement** tab opens. Enter the information in the table below.

Field	Sample value
Endpoint type	HTTP/REST endpoint

Load balanced and fail over endpoints

The load balanced and failover endpoint types are not selected in this example. For details about these endpoint types, see [Working with Endpoints](#) and [ESB Endpoints](#).

Production endpoint	This sample service has two operations as CheckPhoneNumber and CheckPhoneNumbers. Let's use CheckPhoneNumber here. http://ws.cdyne.com/phoneverify/phoneverify.asmx To verify the URL, click the Test button next to it. (This is the actual endpoint where the API implementation can be found)
Sandbox endpoint	This sample service has two operations as CheckPhoneNumber and CheckPhoneNumbers. Let's use CheckPhoneNumber here. http://ws.cdyne.com/phoneverify/phoneverify.asmx To verify the URL, click the Test button next to it.

For more information on Endpoints, please see [Working with Endpoints](#).

The screenshot shows the 'Implement' tab of the WSO2 API Manager interface. At the top, there are three tabs: 'Design' (selected), 'Implement' (highlighted in blue), and 'Manage'. Below the tabs, under the heading 'Managed API', it says 'Provide the production and sandbox endpoints of the API to be managed.' A dropdown menu labeled 'Endpoint Type : * ?' is set to 'HTTP/REST Endpoint'. Under 'Endpoint: * (specify at least one)', there are two entries: 'Production : ?' with the value 'https://ws.cdyne.com/phoneverify.asmx' and a 'Test' button, and 'Sandbox : ?' with the value 'https://ws.cdyne.com/phoneverify.asmx' and a 'Test' button. Both entries are highlighted with a red box. Below these entries are links for 'Manage Certificates' and 'Show More Options'.

Message Mediation Policies

Enable Message Mediation Check to select a message mediation policy to be executed in the message flow

CORS configuration

Enable API based CORS Configuration

Save **Next: Manage**

For additional information, see [Enabling CORS for APIs](#) and [Adding Mediation Extensions](#). For details on adding and managing certificates, see [Dynamic SSL Certificate Installation](#).

You can deploy your API as a **Prototyped API** in the **Implement** tab. A prototyped API is usually a mock implementation made public in order to get feedback about its usability. You can implement it **In line** or by specifying an **endpoint**.

PhoneVerification: /phoneverify/1.0.0

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Prototyped API
Use the inbuilt JavaScript engine to prototype the API or provide an endpoint to a prototype API. The inbuilt JavaScript engine does not have support to prototype SOAP APIs.

Implementation Method: Inline Endpoint

Prototype Endpoint: Valid Test

CORS configuration
Enable API based CORS Configuration

Save Deploy as a Prototype

Users can invoke the API without a subscription after publishing the API to the Store. For more information, see [Deploy and Test as a Prototype](#).

10. Click **Next: Manage >** and enter the information in the table below.

Field	Sample value	Description
Transports	HTTP and HTTPS	<p>The transport protocol on which the API is exposed. Both HTTP and HTTPS transports are selected by default. If you want to limit API availability to only one transport (e.g., HTTPS), un-check the other transport.</p> <div style="border: 1px solid red; padding: 10px; margin-top: 10px;"> <p>You can only try out HTTPS based APIs via the API Console, because the API Store runs on HTTPS.</p> </div>
Subscription Tiers	Select all	The API can be available at different levels of service. They allow you to limit the number of successful hits to an API during a given period.

The screenshot shows the 'Configurations' tab in the WSO2 API Manager. Under 'Throttling Settings', it displays:

- Maximum Backend Throughput:** Unlimited (radio button selected)
- Subscription Tiers:** Unlimited, Gold, Silver, Bronze (checkboxes selected, highlighted with a red box)
- Advanced Throttling Policies:** Apply per Resource (radio button selected)
- Select Policy per Resource** button
- (i) Refer documentation for more information about each throttling setting.

Make Default Version

Make this the Default Version checkbox ensures that the API is available in the Gateway without a version specified in the production and sandbox URLs. This option allows you to create a new version of an API and set it as the default version. Then, you can invoke the same resources in the client applications without changing the API gateway URL. This allows you to create new versions of an API with changes, at the same time, allowing existing clients applications to be invoked without the client having to change the URLs.

Overview API Console Documentation SDKs Forum

Production and Sandbox Endpoints

Production and Sandbox URLs:

http://192.168.42.1:8280/phoneverify/1.0.0
http://192.168.42.1:8280/phoneverify/

https://192.168.42.1:8243/phoneverify/1.0.0
https://192.168.42.1:8243/phoneverify/

For more information on **maximum backend throughput** and **advanced throttling policies**, see [Working with Throttling](#).

- Click **Save & Publish**. This publishes the API that you just created to the API Store so that subscribers can use it.

You can save partially complete or completed APIs without publishing it. Select the API and click on the **Lifecycle** tab to [manage the API Lifecycle](#).

You have created an API.

Related Tutorials

- Create and Publish an API from Swagger definition
- Create a Prototyped API with an Inline Script
- Create a WebSocket API
- Create and Publish a SOAP API

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

The examples here use the PhoneVerification REST API, which is created in the section [Create and Publish an API](#).

1. Sign in to the WSO2 API Store (<https://<hostname>:9443/store>) and click on an API (e.g., PhoneVerification 1.0.0) to open it.

In a multi-tenanted WSO2 API Manager setup, you can access any tenant's store using the URL `http://<hostname>:<port>/store?tenant=<tenant_name>`.

2. Note the subscription options for the REST API.

PhoneVerification - 1.0.0

3. Click the **Applications** menu and click **Add Application** to create a new application.

4. Enter the name as TestApp and select the per token quota as 50PerMin for the application and click **Add**.

Add Application

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name* Characters left: 63

Per Token Quota Allows 50 request per minute
This feature allows you to assign an API request quota per access token. Allocated quota will be shared among all the subscribed APIs of the application.

Description

Add **Cancel**

5. Click **APIs** and click on the PhoneVerification API to view the API's subscription options.
6. Select the application that you just created, a tier, and click **Subscribe**.

PhoneVerification - 1.0.0

Version: 1.0.0
By: admin
Updated: 22/Jul/2016 17:12:01 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications
TestApp

Tiers
Bronze

Subscribe

7. Click the **View Subscriptions** button when prompted.
The **Subscriptions** tab opens.
8. Click the **Production Keys** tab.

If you have a supported callback URL which sends a callback to a specific server or a program soon after your application request is sent, you can specify it under **Callback URL** field under Production Keys.

TestApp

Details Production Keys Sandbox Keys Subscriptions

ⓘ No Keys Found

No keys are generated for this type in this application.

Grant Types

The application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

- | | | | |
|---|--|-----------------------------------|--|
| <input checked="" type="checkbox"/> Refresh Token | <input checked="" type="checkbox"/> SAML2 | <input type="checkbox"/> Implicit | <input checked="" type="checkbox"/> Password |
| <input checked="" type="checkbox"/> Client Credential | <input checked="" type="checkbox"/> IWA-NTLM | <input type="checkbox"/> Code | |

Callback URL

Scopes

No Scopes Found..

Access token validity period

3600 Seconds

Generate keys

- Click **Generate Keys** to create an application access token. You can use this token to invoke all APIs that you subscribe to using the same application.

You can set a token validity period in the **Access token validity period** text box. By default, it is set to one hour. If you set a minus value (e.g., -1), the token never expires.

By default the Client Credentials grant type will be used to generate access token. Make sure the Client Credentials grant type is selected when generating keys from the UI. Refer [Token API](#) for more information on how to generate supported grant types of WSO2 API Manager.

Access Tokens with specific Scopes

Access tokens can be generated for specific scopes. A scope acts as a limiting factor on what API resources can be accessed using a token.

To generate an access token corresponding to a scope, use the drop down menu under **Scopes** and select the required scope parameter.

If you are using the WSO2 Identity Server 5.3.0 as the Key Manager for your API Manager deployment, generating keys will result in creation of a [Service Provider](#) on the Identity Server.

- Install [cURL](#) if it is not there in your environment.

cURL comes by default in some operating systems. You can also use a REST client instead.

- Open the command line and execute the following cURL command:

FormatExampleOutput

```
curl -k -H "Authorization: Bearer <access_token>" -v
'<api_url><payload>'
```

Be sure to replace the placeholders as follows:

- <**access token**>: Give the test token generated in step 8. Click **Applications**, click on the respective application, which in this case is TestApp, click **Production Key**, and click **copy button** to copy the access token.

Make sure you have updated flash plugin in your web browser in order to get the **copy button** working.

Generating Access Tokens

The following cURL command shows how to generate an access token using the Password Grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://[REDACTED]:8243/token
```

In a similar manner, you can generate an access token using the Client Credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://[REDACTED]:8243/token
```

Generate a Test Access Token

Access Token



Above token has a validity period of **3600** seconds. And the token has (**am_application_scope default**) scopes.

Scopes	No Scopes Found..
Validity period	3600 Seconds
Re-generate	

- <**API URL**>: Click on the respective API, which in this case is "PhoneVerification - 1.0.0". When the API's **Overview** tab appears in the API Store copy the production URL and append the payload to it. For example, <https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0>

Overview API Console Documentation Forum SDKs

Production and Sandbox Endpoints

Production and Sandbox URLs:

https://[REDACTED]:8243/phoneverify/1.0.0	
http://[REDACTED]:8280/phoneverify/1.0.0	

```
curl -k -H "Authorization :Bearer
3dfafa3a-b1e3-3550-8a25-88e4b4fe2fb3"
'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
```

Note the result <Valid>true</Valid> that appears in the command line.

```
<?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 />
  <TelecomCounty />
  <TelecomState>TF</TelecomState>
  <TelecomZip />
  <TimeZone />
  <Lat />
  <Long />
  <Wireless>false</Wireless>
</PhoneReturn>
```

Troubleshooting

If you get an error that states "Invalid Credentials", carryout the following steps to overcome the error. This error is a result of the access token getting expired. The default validity period of the access token is 1 hour.

- a. Optionally, you can update the token validity period in the **Access token validity period** text box so that the access token will be valid for a longer period, or you can even set a minus value (e.g., -1) so that the token never expires.
- b. Re-generate the access token. Click **Applications**, click on the respective application (i.e., TestApp), click **Production Key**, and click **Re-generate**. Thereafter, use the new access token when running the cURL commands.

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

FormatExampleOutput

```
curl -k -H "Authorization :Bearer <access token>" --data  
"PhoneNumber=<phone_number>&LicenseKey=<license_key>" <api_url>
```

```
curl -k -H "Authorization :Bearer  
3dfafa3a-b1e3-3550-8a25-88e4b4fe2fb3" --data  
"PhoneNumber=18006785432&LicenseKey=0"  
https://localhost:8243/phoneneverify/1.0.0/CheckPhoneNumber
```

```

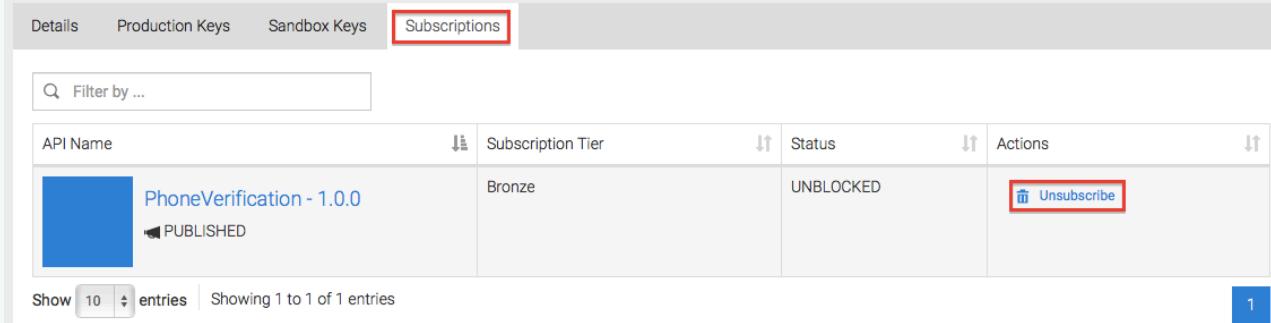
<?xml version="1.0" encoding="utf-8"?>
<PhoneReturn xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://ws.cdyne.com/PhoneVerify/query">
  <Company>Toll Free</Company>
  <Valid>true</Valid>
  <Use>Assigned to a code holder for normal use.</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 />
  <TelecomCounty />
  <TelecomState>TF</TelecomState>
  <TelecomZip />
  <TimeZone />
  <Lat />
  <Long />
  <Wireless>false</Wireless>
</PhoneReturn>

```

You have subscribed to an API and invoked it.

To unsubscribe from an API, click the **Applications** menu and click **View** next to the application used for the subscription. Go to the **Subscriptions** tab, locate the API, and click the **Unsubscribe** link associated with it.

TestApp



API Name	Subscription Tier	Status	Actions
PhoneVerification - 1.0.0 PUBLISHED	Bronze	UNBLOCKED	Unsubscribe

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

WSO2 API Manager (WSO2 APIM) has an integrated Swagger UI, which is part of the Swagger project.

[Swagger](#) is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interacts with the service with a minimal amount of implementation logic. Swagger helps describe a service in the same way that interfaces describe lower-level programming code.

The [Swagger UI](#) is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation and more discoverability. The Swagger UI has JSON code and its UI facilitates easier code indentation, keyword highlighting, and shows syntax errors on the fly. You can add resource parameters, summaries, and descriptions to your APIs using the Swagger UI.

For more information also, see the [Swagger 2.0 specification](#).

Let's see how to use the API Console in the Store to invoke an 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](#) - This is needed only if you have the **API Store and Gateway in different ports** or you want to invoke an API with **inline endpoints**.

You can only try out HTTPS based APIs via the API Console, because the API Store runs on HTTPS.

The examples here use the PhoneVerification REST API, which is created in the section [Create and Publish an API](#).

1. Sign in to the WSO2 API Store and click an API (e.g., PhoneVerification).
<https://<hostname>:9443/store>
2. Subscribe to the API (e.g., PhoneVerification 1.0.0) using the **default application** and an available tier.

PhoneVerification - 1.0.0

The screenshot shows the PhoneVerification API details page. On the left, there is a large red square icon with a white letter 'P'. To the right, there is a summary card with the following details:

Version:	1.0.0
By:	admin
Updated:	22/Jul/2016 17:12:01 PM IST
Status:	PUBLISHED
Rating:	★★★★★ (5)

To the right of the summary card is a sidebar with the following sections:

- Applications:** DefaultApplication
- Tiers:** Bronze

At the bottom of the sidebar is a blue button labeled "Subscribe" with a key icon.

3. Click on the **Applications** menu and open the default application which you used to subscribe to the API. Click the **Production Keys** tab and click **Generate keys** to generate a production key.

DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

Info No Keys Found

No keys are generated for this type in this application.

Grant Types

Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM	<input type="checkbox"/> Implicit	<input checked="" type="checkbox"/> Refresh Token
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code	<input checked="" type="checkbox"/> Password	

Callback URL

Access token validity period

3600 Seconds.

Generate keys

Production and Sandbox Tokens

To generate keys for the Sandbox endpoint, go to the **Sandbox Keys** tab. For more details, see [Maintaining Separate Production and Sandbox Gateways](#).

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

PhoneVerification - 1.0.0

Version: 1.0.0
By: admin
Updated: 21/Nov/2017 12:15:04 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications: Select Application...
Tiers: Bronze

Subscribe

Overview API Console Documentation Forum SDKs

Try DefaultApplication
Using Production Key

Set Request Header Authorization : Bearer 77c30c98-bea0-3c82-b71d-36385a52041c

Swagger (/swagger.json)
Show/Hide | List Operations | Expand Operations

default

GET /CheckPhoneNumber
POST /CheckPhoneNumber

[BASE URL: /phoneverify/1.0.0 , API VERSION: 1.0.0]

If you have subscribed to an application, the retrieved access token value appears automatically as the **Authorization Bearer Token**.

Documentation

The **Documentation** tab contains any relevant documents that are attached to the API.

Overview API Console Documentation SDKs Forum

How To

Name	Summary
Phone Verification Doc	Documentation for PhoneVerification

Download

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>

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

Try it out!

POST /CheckPhoneNumber

[BASE URL: /phoneverify/1.0.0 , API VERSION: 1.0.0]

BASE URL

This appears at the bottom of the console. Using the base URL and the parameters, the system creates the API URL in the form `https://<host_name>:8243/<context>/<version>/<resource><backend_service>`

- <resource> - The resource of the URL, if any.
- <backend_service> - This refers to the backend service requirements included as parameters, if any. For example, in the following API URL, /phoneverify is the context, 1.0.0 is the version, and CheckPhoneNumber is the resource: <https://localhost:8243/phoneverify/1.0.0/CheckPhoneNumber>

Troubleshooting

If you **cannot invoke the API's HTTPS endpoint** (this 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.

If the API Manager has a **certificate signed by a Certificate Authority (CA)**, the HTTPS endpoints should work out of the box.

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

The screenshot shows the WSO2 API Manager's integrated Swagger UI interface. It displays a successful API response for a phone verification query. The response body is an XML document with various fields like Company, Valid, Use, State, RC, OCN, OriginalNumber, ClearNumber, SwitchName, SwitchType, Country, and CLLI. A specific field, <Valid>true</Valid>, is highlighted with a red border. The response code is 200, and the response headers include pragma, content-type, cache-control, and expires.

You have invoked an API using the Swagger API Console.

Edit an API Using the Swagger UI

WSO2 API Manager has an integrated Swagger Editor, which is part of the Swagger project.

Swagger is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe a service in the same way that interfaces describe lower-level programming code.

The [Swagger Editor](#) is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation and more discoverability. The Swagger Editor has JSON code and its UI facilitates easier code indentation, keyword highlighting and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger Editor.

API Manager supports for both [Open API 3.0](#) and [Open API 2.0](#) specifications and you can simply create, import, edit and consume the APIs defined in both specifications.

In this tutorial, let's see how you can add interactive documentation to an API by directly editing the Swagger code via the API Publisher UI.

This tutorial uses the PhoneVerification API created in [Create and Publish an API](#).

1. Sign in to the WSO2 API Publisher and choose to design a new REST API.
<https://<hostname>:9443/publisher>

Let's get started!

Add New API

The screenshot shows the 'Add New API' interface. There are four options listed:

- I Have an Existing API**: Use an existing API's endpoint or the API Swagger definition to create an API.
- I Have a SOAP Endpoint**: Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.
- Design a New REST API**: Design and prototype a new REST API. This option is currently selected, indicated by a blue circle.
- Show More Options**

A red box highlights the **Start Creating** button, which is also highlighted with a blue background.

The screenshot shows the 'Design a New REST API' sub-options interface. It includes:

- Design a New REST API**: Design and prototype a new REST API.
- Show Fewer Options**
- Supported API Specification:** A dropdown menu showing 'OpenAPI 2.0' and 'OpenAPI 3.0', with 'OpenAPI 3.0' checked.
- Start Creating**

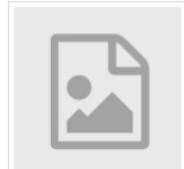
A red box highlights the 'Start Creating' button.

2. Click **Start Creating**.
3. In the **Design** tab, give an API name, a context and a version, and click **Edit Source** under the **API Definition** section.
The Swagger UI opens.

Design API

1 Design 2 Implement 3 Manage

General Details

Name:*	PhoneVerification	Thumbnail Image
Context:*	/phoneverify	
Version:*	1.0.0	Dimensions (max): 100 x 100 pixels
Access Control:	All	Select image
Visibility on Store:	Public	
Description:	Maximum 20000 characters.	
Tags:	Add tags Type a Tag and Enter	

API Definition

URL Pattern	/phoneverify/1.0.0 E.g., path/to/resource	<input type="button" value="Import"/> <input style="border: 2px solid red;" type="button" value="Edit Source"/>
<input type="checkbox"/> GET <input type="checkbox"/> POST <input type="checkbox"/> PUT <input type="checkbox"/> DELETE <input type="checkbox"/> PATCH <input type="checkbox"/> HEAD more		
<input type="button" value="⊕ Add"/>		

4. Add a GET and POST method for the API.
 - a. Add the following code under the `paths` object as shown in the screenshot.

In the code below, note that you have a resource defined with the URL pattern `/CheckPhoneNumber` under the `paths` object. This is followed by the HTTP methods GET and POST. For each HTTP method, the following parameters are defined.

- **responses:** An object to hold responses that can be used across operations. See the Swagger specification for details.
- **x-auth-type:** WSO2-specific object to define the authentication type of the method.
- **x-throttling-tier:** WSO2-specific object to define the throttling tier of the method.

```

/CheckPhoneNumber:
  get:
    responses:
      '200':
        description: ''
        x-auth-type: Application & Application User
        x-throttling-tier: Unlimited
  post:
    responses:
      '200':
        description: ''
        x-auth-type: Application & Application User
        x-throttling-tier: Unlimited

```

Troubleshooting

If you get an error after adding the API definition in the Swagger UI, first check the indentation of the code that you added, which defines the API, because Swagger throw errors if the indentation is not correct.

The screenshot shows the Swagger Editor interface. The top navigation bar includes 'Discard Changes', 'Apply Changes', 'Swagger Editor', 'File', 'Edit', 'Generate Server', and 'Generate Client'. The main area displays the following JSON code:

```

1 swagger: "2.0"
2 paths:
3   /CheckPhoneNumber:
4     get:
5       responses:
6         '200':
7           description: ''
8           x-auth-type: Application & Application User
9           x-throttling-tier: Unlimited
10      post:
11        responses:
12          '200':
13            description: ''
14            x-auth-type: Application & Application User
15            x-throttling-tier: Unlimited
16 info:
17   title: PhoneVerification
18   version: ""
19

```

A red box highlights the 'paths' section of the code. To the right, a preview window titled 'PhoneVerification' shows two operations: 'default' (GET /CheckPhoneNumber) and 'POST /CheckPhoneNumber'.

b. Click **Apply Changes**.

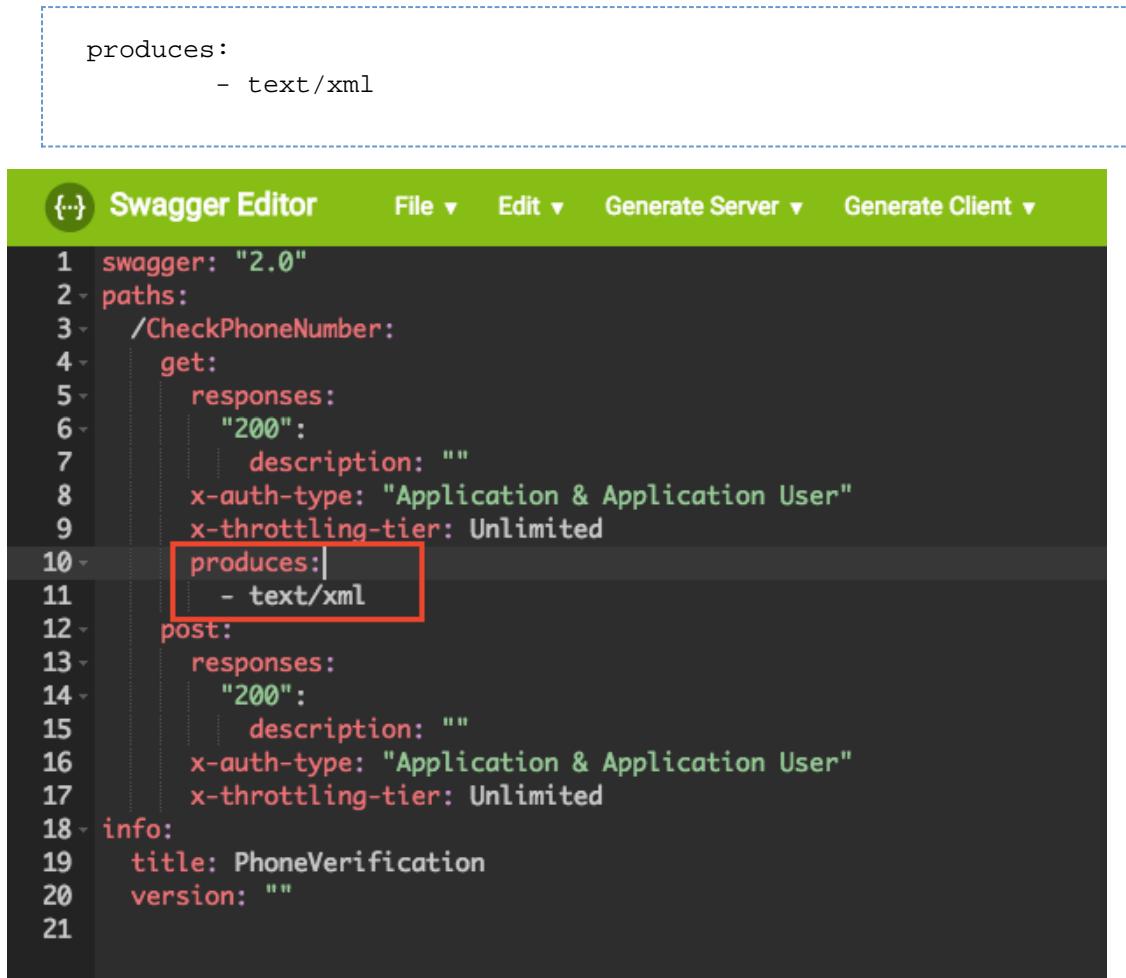
This adds a resource with two HTTP methods into the API which is visible in the WSO2 API Publisher.

The screenshot shows the WSO2 API Publisher interface under the 'API Definition' tab. It includes fields for 'URL Pattern' (/{context}/{version}) and 'Import' and 'Edit Source' buttons. Below are checkboxes for HTTP methods: GET, POST, PUT, DELETE, PATCH, HEAD, and a 'more' button. A large red box highlights the list of methods. At the bottom, a table lists the methods with their corresponding URLs: 'GET /CheckPhoneNumber + Summary' and 'POST /CheckPhoneNumber + Summary'. Each row has a delete icon in the top right corner.

Let's assume that the backend of this API sends the response in XML format. Let's document this under the GET method in the resource that we just added.

5. Change the response content type to XML.

- Click **Edit Source** and add the following code under the GET method.



```

1  swagger: "2.0"
2  paths:
3    /CheckPhoneNumber:
4      get:
5        responses:
6          "200":
7            description: ""
8            x-auth-type: "Application & Application User"
9            x-throttling-tier: Unlimited
10       produces:
11         - text/xml
12       post:
13         responses:
14           "200":
15             description: ""
16             x-auth-type: "Application & Application User"
17             x-throttling-tier: Unlimited
18   info:
19     title: PhoneVerification
20     version: ""
21

```

b. Click

Apply

Changes.

The response content type that you updated is visible when you expand the GET method in the WSO2 API Publisher.

You can use this attribute to document the type of the response message that the backend sends. **It does not do any message type conversion.** You can add multiple values as a **c o m m a - s e p a r a t e d list.**

Example:

```

produces:
- text/xml, application/json

```

API Definition

URL Pattern

GET POST PUT DELETE PATCH HEAD more

GET /CheckPhoneNumber [+ Summary](#)

Description : [+ Add Implementation Notes](#)

Produces : **text/xml** Consumes : **Empty**

Parameters :

Parameter Name

POST /CheckPhoneNumber [+ Summary](#)

6. Define parameters that correspond to the GET method.

- a. Click **Edit Source** and add the following code, which defines two parameters to the method, under the GET method.

```
parameters:
  - name: PhoneNumber
    in: query
    required: true
    type: string
    description: Give the phone number to be validated
  - name: LicenseKey
    in: query
    required: true
    type: string
    description: Give the license key as 0 for testing purpose
```

```
1  swagger: "2.0"
2  paths:
3    /CheckPhoneNumber:
4      get:
5        responses:
6          "200":
7            description: ""
8            x-auth-type: "Application & Application User"
9            x-throttling-tier: Unlimited
10           produces:
11             - text/xml
12             parameters:
13               - name: PhoneNumber
14                 in: query
15                 required: true
16                 type: string
17                 description: Give the phone number to be validated
18               - name: LicenseKey
19                 in: query
20                 required: true
21                 type: string
22                 description: Give the license key as 0 for testing purpose
23
24   post:
25     responses:
26       "200":
27         description: ""
28         x-auth-type: "Application & Application User"
29         x-throttling-tier: Unlimited
30   info:
31     title: PhoneVerification
```

GET /CheckPhoneNumber

Parameters

Name	Description
PhoneNumber * required	Give the phone number to be validated string (query)
LicenseKey * required	Give the license key as 0 for testing purpose string (query)

Responses

Code	Description
200	

Response content type

- b. Click **Apply Changes**.

The two parameters with their descriptions that you added are visible when you expand the GET method in WSO2 API Publisher.

The screenshot shows the WSO2 API Publisher interface. At the top, there's a blue header bar with 'GET' and the endpoint '/CheckPhoneNumber'. Below it is a summary section with 'Description : + Add Implementation Notes' and 'Parameters :'. A table lists two parameters: 'PhoneNumber' (query, string, required) and 'LicenseKey' (query, string, required). At the bottom, there's a search bar for 'Parameter Name' and a button to '+ Add Parameter'.

7. Add a summary and description for the GET method.

- Click **Edit Source** and add the following code, which defines a summary and description, to the GET method.

```
summary: Check the validity of your phone number
description: "Phone Verification validates a telephone number and
returns carrier information, location routing etc."
```

The screenshot shows the Swagger Editor on the left and the PhoneVerification API interface on the right. In the Swagger Editor, the 'parameters' section for the GET method includes a 'summary' and 'description' block. The 'PhoneVerification' interface shows the expanded GET method with its summary and description, along with its parameters: 'PhoneNumber' and 'LicenseKey'.

- Click **Apply Changes**.

The summary and description of the GET method that you added is visible when you expand the GET method in WSO2 API Publisher.

The screenshot shows the expanded GET method in the WSO2 API Publisher. It includes the summary 'Check the validity of your phone number', the description 'Phone Verification validates a telephone number returns carrier information, location routing etc.', the 'Produces : text/xml' and 'Consumes : Empty' fields, and the parameter table from the previous step.

8. Add parameters to the POST method and also change the POST method datatype.

- Click **Edit Source** and add the following code under the POST method, which defines two parameters

named PhoneNumber and LicenseKey to pass in the payload. It also changes the datatypes of the parameters to application/x-www-form-urlencoded as the backend expects that datatype.

```

consumes:
  - application/x-www-form-urlencoded
parameters:
  - name: PhoneNumber
    in: formData
    required: true
    type: string
    description: Give the phone number to be validated
  - name: LicenseKey
    in: formData
    required: true
    type: string
    description: Give the license key as 0 for testing
purpose

```

The screenshot shows the Swagger Editor interface. On the left, the code editor displays the API definition with the expanded POST method for '/CheckPhoneNumber'. The 'consumes' and 'parameters' sections are highlighted with a red box. On the right, the API details panel shows the method as POST /CheckPhoneNumber, parameters (PhoneNumber and LicenseKey), responses (200), and code samples.

b. Click Apply Changes.

The two parameters with their descriptions that you added are visible when you expand the POST method in WSO2 API Publisher.

The screenshot shows the WSO2 API Publisher interface. It displays the expanded POST method for '/CheckPhoneNumber'. The 'Consumes' field is highlighted with a red box. Below it, a table lists the parameters: PhoneNumber and LicenseKey, both with their descriptions and other details like type and required status.

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

9. Change the title of the API.

- Click **Edit Source**, and add the following code in the Swagger UI. This is the title that is visible to the consumers in WSO2 API Store after the API is published.

The screenshot shows the Swagger Editor interface. On the left, the API definition is displayed in JSON format. A red box highlights the 'info' section, which contains the title 'PhoneVerificationAPI'. On the right, the API's name 'PhoneVerificationAPI' is shown in a large red-bordered box. Below it, the 'default' endpoint is listed with two operations: a GET method for '/CheckPhoneNumber' and a POST method for '/CheckPhoneNumber'.

```

info:
  title: PhoneVerificationAPI

```

You can see how this change is reflected in the WSO2 API Store in step 15.

- b. Click **Apply Changes** and complete the API creation process.
10. Complete the rest of the API creation process.
For more information, see step 6 onwards under [Create and Publish an API](#).
11. Click **Go to APIStore** and click on the API that you just published.

The screenshot shows the WSO2 API Store interface. A large red box highlights the title 'PhoneVerificationAPI - 1.0.0'. To the left is a large red square icon with a white letter 'P'. To the right, detailed information about the API is provided:

- Version:** 1.0.0
- By:** admin
- Updated:** 08/Feb/2018 14:44:33 PM IST
- Status:** PUBLISHED
- Rating:** ★★★★☆ (with one star highlighted)

The API opens.

12. Click **API Console**.
Note that the changes that you made earlier are now appearing in the WSO2 API Store for consumers.

default

GET /CheckPhoneNumber Check the validity of your phone number

Phone Verification validates a telephone number returns carrier information, location routing etc.

Parameters

Name **Description**

PhoneNumber * required string (query) Give the phone number to be validated

LicenseKey * required string (query) Give the license key as 0 for testing purpose

Responses

Response content type **text/xml**

Code **Description**

200

In this tutorial, you have seen how the integrated Swagger Editor can be used to design, describe, and document your API, so that the API consumers get a better understanding of the API's functionality.

API Publishing

How do I...

- Add API Documentation
- Manage the API Lifecycle
- Publish to Multiple External API Stores
- Publish through Multiple API Gateways
- Block Subscription to an API
- Enforce Throttling and Resource Access Policies
- Change the Default Mediation Flow of API Requests
- Map the Parameters of your Backend URLs with the API Publisher URLs
- Convert a JSON Message to SOAP and SOAP to JSON
- Invoke an API using a SOAP Client
- Create and Publish an API from Swagger definition
- Create a WebSocket API
- Create a Prototyped API with an Inline Script
- Pass a Custom Authorization Token to the Backend

- Create and Publish a SOAP API
- Disable Message Chunking
- Enable API Indexing on Remote Publisher and Store Nodes
- Remove Specific Request Headers From Response
- Scope Management with OAuth Scopes
- Generate REST API from SOAP Backend

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 to 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 WSO2 API Publisher are as follows:

- **In-line:** Hosts documentation (How-tos, Samples, SDK, forums etc.) in WSO2 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 integrated API Console**

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

1. Sign in to the WSO2 API Publisher.
`https://<hostname>:9443/publisher`
2. Click the **Edit** icon shown below for the API (e.g., `PhoneVerification 1.0.0`) to which you want to add documentation.

The screenshot shows the WSO2 API Publisher interface. At the top, there's a navigation bar with the WSO2 logo, 'HOME / APIs', and a 'ADD NEW API' button. Below the navigation bar, there are two main sections: 'STATISTICS' and 'MANAGE SUBSCRIPTIONS'. The main content area is titled 'All APIs' and contains a search bar. Two API entries are listed side-by-side:

Name	Version	Owner	Users	Status	Action
PhoneVerification...	1.0.0	admin	1 User	PUBLISHED	
PhoneVerification...	1.0.0	admin	0 Users	PUBLISHED	

In the above screenshot, the APIs are different as they have different names (i.e., PhoneVerification and PhoneVerificationAPI).

3. Click **Go to Overview**.
4. **Add in-line documentation.**
 - a. Select the **Docs** tab of the API and click **Add New Document**.

PhoneVerification - 1.0.0

The screenshot shows the 'PhoneVerification - 1.0.0' API overview page. The 'Docs' tab is selected and highlighted with a red box. A button labeled 'Add New Document' is also highlighted with a red box. A message box displays the text: 'No documentation associated with the API' and 'There is no documentation created for this API. You can add new documentation to this API by clicking the "Add New Document" button.'

- b. Enter the following details to create documentation in-line.

Name	PhoneVerification
------	-------------------

Type	How To
Source	In-line
Summary	Check the validity of a phone number

PhoneVerification - 1.0.0

Overview Lifecycle 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
 Inline
 URL
 File

Add Document Cancel

No documentation associated with the API
There is no documentation created for this API. You can add new documentation to this API by clicking the "Add New Document" button.

- c. Click **Add Document**.
- d. Click **Edit Content** to open an embedded editor.

Update button can be used to update/change the document information.

PhoneVerification - 1.0.0

Overview Lifecycle Versions **Docs** Users

Add New Document

Filter by ...

Name	Type	Modified On	Actions
PhoneVerification	How To	1/27/2017, 5:22:36 PM	Edit Content Update Delete

Show 10 entries | Showing 1 to 1 of 1 entries

- e. Edit the document content in-line using the embedded editor and click **Save and Close**.

PhoneVerification

Determine whether the phone line is wireless or landline.

Any verification operation which sells sets of contact numbers generated through its marketing operations will gain a premium for an efficient list of "clean" numbers. Call center outbound telephone campaigns save time and resources bypassing undefined or mechanized equipment numbers.

Telephone numbers entered into an online sign-up form can be checked in real time, by implementing [AJAX](#) (background web page processing) type call to a telephone number verification service while the form is still being filled in.

p » a

Save **Save and Close** **Cancel**

The API's **Doc** tab opens.

5. Add documentation using a URL.

- Click **Add New Document** to add another doc type.
- Enter the following information to create another doc using a URL.

Name	CDYNE Wiki
Summary	CDYNE Phone Notify API
Type	Other (specify)
Source	URL https://cdyne.com/downloads/SPECS_Phone-Notify.pdf

Overview Lifecycle Versions **Docs** Users

Add New Document

Name*	CDYNE Wiki
Summary*	CDYNE Phone Notify 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) * Phone Notify
Source	<input type="radio"/> Inline <input checked="" type="radio"/> URL * https://cdyne.com/downloads/SPECS_Phone-Notify.pdf

Add Document **Cancel**

c. Click

The API's **Doc** tab opens.

6. Add documentation using a file.

- Click **Add New Document** to add yet another document using a file.
- Enter the following information.

A d d

D o c u m e n t .

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

PhoneVerification - 1.0.0

Overview Lifecycle Versions **Docs** Users

Add New Document

Name*
API Manager Samples

Summary*
API Manager Samples

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

Source
 Inline
 URL
 File
 WS02 API Manager Sam
eg : pdf, xml, txt, wsdl, ms-word, ms-powerpoint, ms-excel

Add Document Cancel

- c. Click **Add** **Document**. You have now added three documents to the API: in-line, using a URL, and a file.

PhoneVerification - 1.0.0

Overview Lifecycle Versions **Docs** Users

Add New Document

Filter by ...

Name	Type	Modified On	Actions
API Manager Samples	Samples	1/30/2017, 4:34:40 PM	Open Update Delete
CDYNE Wiki	Other	1/30/2017, 3:40:17 PM	View Update Delete
PhoneVerification	How To	1/27/2017, 5:22:36 PM	Edit Content Update Delete

Show 10 entries | Showing 1 to 3 of 3 entries

7. Log in to the WSO2 API Store and click the PhoneVerification 1.0.0 API.
<https://<hostname>/store>

The screenshot shows the WSO2 API Store interface. The top navigation bar includes 'APIS' (selected), a search bar, and a help icon. The left sidebar has sections for 'PROTOTYPED APIS', 'APPLICATIONS' (selected, highlighted in green), 'FORUM', and 'STATISTICS'. Below these are 'TAGS' and a 'PIZZA' tag. The main content area displays two API cards. The first card, 'PhoneVerification 1.0.0' by 'admin', is highlighted with a red border. The second card, 'PizzaShackAPI 1.0.0' by 'admin', shows a small image of a pizza.

8. Go to the API's **Documentation** tab and see the documents listed by type. Click the links to see the documentation content. As a subscriber, you can read the documentation and learn about the API.

The screenshot shows the 'PhoneVerification - 1.0.0' API documentation page. It features a large logo 'P' and a summary box with details: Version 1.0.0, By admin, Updated 25/Jan/2017 11:22:02 AM IST, and Status PUBLISHED. Below this are tabs for Overview, API Console, Documentation (selected and highlighted in red), Forum, and SDKs. The Documentation tab content includes sections for 'How To' (with a table for Name and Summary) and 'Other' and 'Samples' sections.

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

Add Apache Solr-Based Indexing

WSO2 API Manager has Apache Solr based indexing for API documentation content. It provides both the API Publisher and Store a full-text search facility to search through the API documentation, and find the documents and related APIs. The search syntax is **doc:keyword**. The 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, which are configured using the <Indexers> element in <APIIM_HOME>/repository/conf/registry.xml file.

- 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 indexes, you can write your own indexer implementation and register it as follows:

1. Write a custom indexer.
The following is the 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.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 the <API-M_HOME>/repository/components/lib directory.
3. Update the <Indexers> element in the <API-M_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.
mediaTypeRegEx	A regular expression (regex) pattern to match the media type.
profiles	API-M profiles in which the indexer is available.

4. Restart the server.
5. Add the API documentation using the new media type and thereafter search some term in the documentation using the syntax (doc:keyword).

You can now see how the documentation has got indexed according to the media type.

Manage the API Lifecycle

In order to provide flexibility of integration between APIs with internal and external environments the lifecycle of an API is important. APIs provided by WSO2 API Manager has a lifecycle containing [six stages](#) which allows you to identify in which state that the API is currently on.

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](#)
- [Customize API Life Cycle](#)

Create a New API Version

DisplayMultipleVersions

A new **API version** is created when you want to change a published API's behavior, authentication mechanism, [resources](#), [throttling tiers](#), target audiences etc. It is not recommended to modify a published API that has subscribers plugged to it.

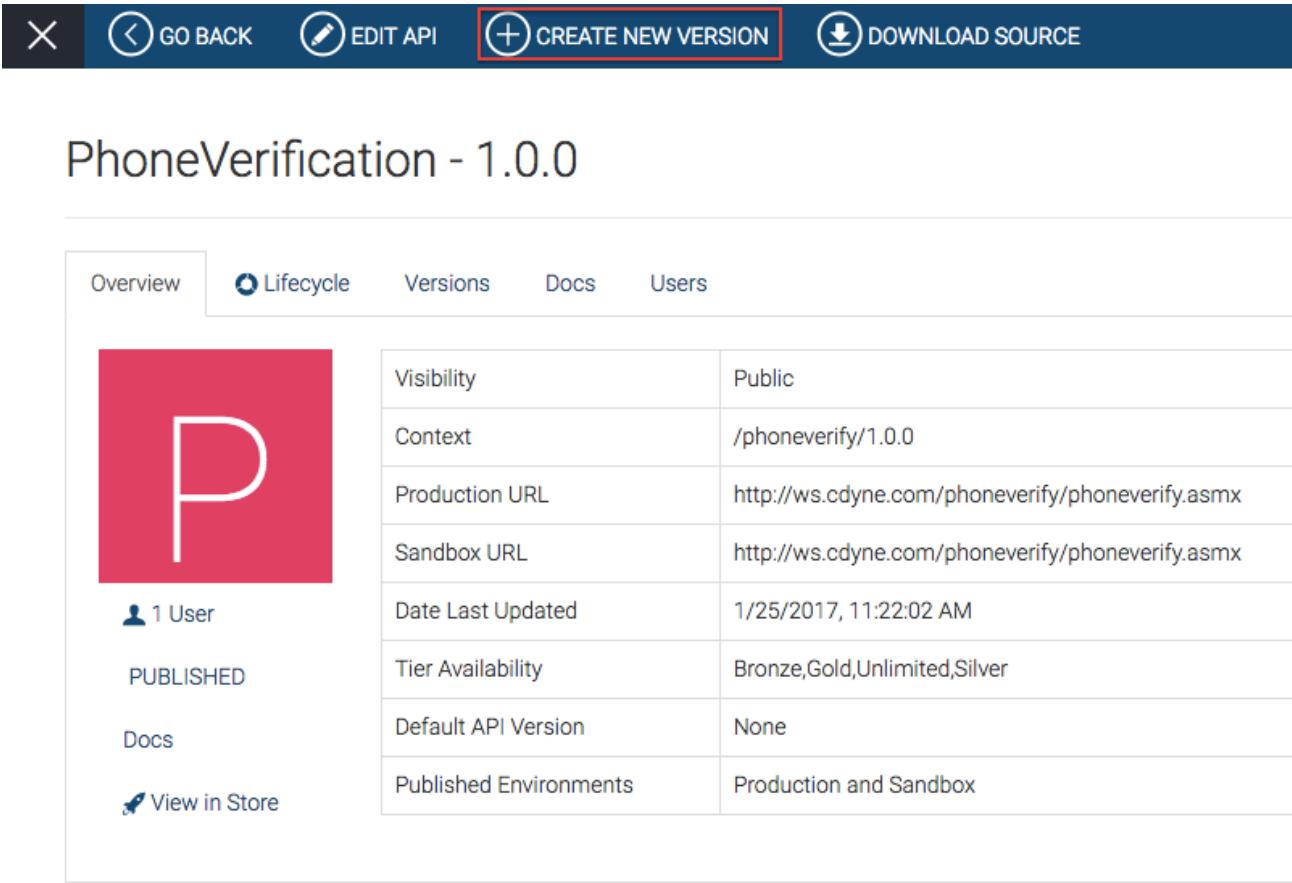
After creating a new version, you typically deploy it as a prototype for early promotion. A prototype can be used for testing, without subscription, along with the published versions of the API. After a period of time during which the new version is used in parallel with the older versions, the prototyped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

The example here uses the PhoneVerification API, which you created in the [Create and Publish an API](#) section.

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

1. Sign in to the WSO2 API Publisher.
<https://<hostname>:9443/publisher>. Refer [step 1](#) of [Create and Publish an API](#) to log into publisher.
2. Select the API that you want to create a version of (e.g., PhoneVerification 1.0.0).
The API opens.

3. Click **Create New Version**.



The screenshot shows the 'PhoneVerification - 1.0.0' API overview page. At the top, there are navigation links: 'GO BACK', 'EDIT API', 'CREATE NEW VERSION' (which is highlighted with a red border), and 'DOWNLOAD SOURCE'. Below the header, the API name 'PhoneVerification - 1.0.0' is displayed. A large red square icon with a white letter 'P' is shown. To its right, there are tabs for 'Overview', 'Lifecycle', 'Versions', 'Docs', and 'Users'. The 'Lifecycle' tab is selected. On the left, there are summary statistics: '1 User', 'PUBLISHED', 'Docs', and a link to 'View in Store'. To the right, a table provides detailed information about the API:

Visibility	Public
Context	/phoneverify/1.0.0
Production URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx
Sandbox URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx
Date Last Updated	1/25/2017, 11:22:02 AM
Tier Availability	Bronze,Gold,Unlimited,Silver
Default API Version	None
Published Environments	Production and Sandbox

4. Give a version number, select the default version option, and click **Done**.
The **APIS** page opens.

PhoneVerification - 1.0.0

Create New Version

New Version: *

Make this the Default Version ?

Info!

No default version defined for the current API

Done **Cancel**

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, two API URLs are listed in its **Overview** tab 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 is used as the default until the new default API is published (or prototyped).

- Click the **Edit** icon of the new API version to edit it.

All APIs

API Name	Version	Owner	Users	Status	Actions
PhoneVerificationAPI	1.0.0	admin	1 User	PUBLISHED	
PhoneVerificationAPI	2.0.0	admin	0 Users	CREATED	
PizzaShackAPI	1.0.0	admin	0 Users	PUBLISHED	

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

API Definition

URL Pattern Url Pattern E.g.: path/to/resource

GET POST PUT DELETE PATCH HEAD more

POST	/CheckPhoneNumber + Summary	
GET	/CheckPhoneNumber + Summary	

Note that there is a known issue in API Manager 2.1.0, where the new versions of APIs created with SOAP Endpoint cannot be modified and saved with the existing WSDL endpoint set when creating the new version. Therefore as a workaround, edit the API and change the existing WSDL endpoint to the correct WSDL endpoint before doing other modifications to the new version of the API.

The screenshot shows the 'Design' tab of an API configuration page. The API name is 'Bmi: /bmi/2.0.0'. The 'WSDL' field contains the URL '/registry/resource/_syst' and is highlighted with a red box. Other fields include 'Visibility: Public', 'Description' (empty), 'Tags' (empty), and a 'Thumbnail Image' section with a placeholder image and a 'Select image' button.

- 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 the `<APIM_HOME>/repository/conf/api-manager.xml` file, and restart the server.

You have created a new version of an API. In the next tutorial, let's learn how to [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 that the users request and publish the API.

The example here uses the API PhoneVerification 2.0.0, which you created in the previous tutorial.

- Sign in to the WSO2 API Publisher and select the API (e.g., PhoneVerification 2.0.0) that you want to prototype.

<https://<hostname>:9443/publisher>

All APIs

API Name	Version	Creator	User Count	Status
PhoneVerification	1.0.0	admin	1 User	PUBLISHED
PhoneVerification	2.0.0	admin	0 Users	CREATED
PizzaShackAPI	1.0.0	admin	0 Users	PUBLISHED

- Click **GO TO OVERVIEW**.

- Click the **Lifecycle** tab of the API and click **Deploy as Prototype**.
After creating a new version, you typically deploy it as a prototype for the purpose of testing and early promotion.

PhoneVerification - 2.0.0

Overview Lifecycle Versions Docs Users

Current State: **CREATED**

Require re-subscription when publish the API

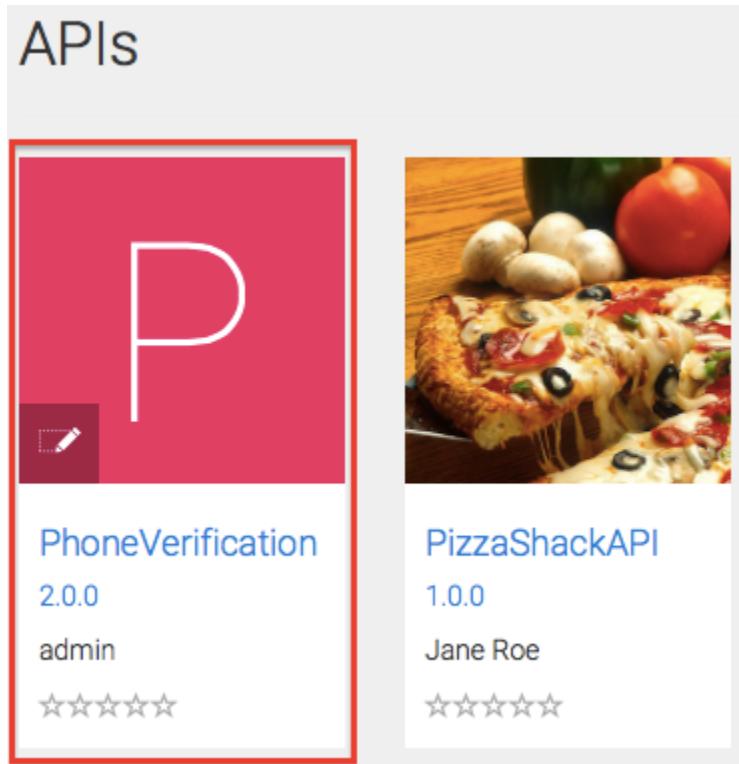
Deprecate old versions after publish the API

Publish Deploy as a Prototype

Tip: Leave the **Require Re-Subscription** check box cleared if you want all users who are subscribed to the older version of the API to be automatically subscribed to the new version. If not, they need to subscribe to the new version again. You can choose to deprecate old versions of this API at this

stage by selecting the **Deprecate Old Versions** check box.

- Sign in to the API Store and click on the newly prototyped API.
`https://<hostname>:9443/store`



The APIs **Overview** page opens. Note the following:

- There are no subscription options.
- There are two sets of URLs (with and without the version). This is because you marked the 2.0.0 version as the default version in [step 4 of the previous tutorial](#).
- Other features such as documentation, social media, and forums are available.

PhoneVerification - 2.0.0



Version: 2.0.0
By: admin
Updated: 31/Jan/2017 17:42:08 PM IST
Status: PROTOTYPED
Rating: ★★★★☆

[Overview](#)[API Console](#)[Documentation](#)[Forum](#)[SDKs](#)

Production and Sandbox Endpoints

Production and Sandbox URLs:

<https://192.168.1.3:8243/phoneverify/2.0.0>
[https://\[REDACTED\]:8243/phoneverify/](https://[REDACTED]:8243/phoneverify/)

<http://192.168.1.3:8280/phoneverify/2.0.0>
[http://\[REDACTED\]:8280/phoneverify/](http://[REDACTED]:8280/phoneverify/)



Share

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

Comments

Characters left: 450

[Add](#)

No comments yet

5. Click the API Console tab.
Note that the POST method is not available as we removed that in the new version.

PhoneVerification - 2.0.0

The screenshot shows the API details page for 'PhoneVerification - 2.0.0'. It includes a large red square icon with a white 'P' in the top-left corner. To its right, there's a summary box with the following information:

- Version:** 2.0.0
- By:** admin
- Updated:** 31/Jan/2017 18:14:09 PM IST
- Status:** PROTOTYPED
- Rating:** ★★★★☆ (with a small 'X' icon)

Below this, a navigation bar has tabs: Overview, API Console (which is highlighted with a red border), Documentation, Forum, and SDKs. Further down, a 'Swagger (/swagger.json)' link is visible, along with 'Show/Hide', 'List Operations', and 'Expand Operations' buttons. A 'default' section is shown, featuring a 'GET /CheckPhoneNumber' method. The base URL '[BASE URL: /phoneverify/2.0.0 , API VERSION: 2.0.0]' is also present.

Let's invoke the prototyped API.

6. In the **API Console** of the prototyped API, expand the GET method, enter the following parameter values, and invoke the API.

PhoneNumber	E.g., 18006785432
LicenseKey	Give 0 for testing purposes.

The screenshot shows the expanded 'GET /CheckPhoneNumber' method. It displays two parameters:

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

Below the parameters, a 'Response Messages' section is shown, listing the HTTP status code '200' and a button labeled 'Try it out!'. The base URL '[BASE URL: /phoneverify/2.0.0 , API VERSION: 2.0.0]' is also present at the bottom.

[BASE URL: /phoneverify/2.0.0 , API VERSION: 2.0.0]

Note the response that appears in the console. You do not have to subscribe to the API or pass an authorization key to invoke a prototyped API.

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/PhoneVe
rify/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 />

```

Response Code

200

Response Headers

```
{
  "pragma": "no-cache",
  "content-type": "text/xml; charset=utf-8",
  "cache-control": "no-cache",
  "expires": "-1"
}
```

7. Similarly, try to invoke the 1.0.0 version of the API without an access token and note that you get an authentication error as "Missing credentials", because version 1.0.0 is a published API.

GET /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
PhoneNumber	18006785432	Give the phone number to be validated	query	string
LicenseKey	0	Give the license key as 0 for testing purpose	query	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/xml' --header 'Authorization: Bearer ' 'https://[REDACTED]:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
```

Request URL

```
https://[REDACTED]:8243/phoneverify/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0
```

Request Headers

```
{
  "Accept": "application/xml"
}
```

Response Body

```

<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
  <ams:code>900902</ams:code>
  <ams:message>Missing Credentials</ams:message>
  <ams:description>Required OAuth credentials not provided. Make sure your API invocation call has a header: "Authorization: Bearer ACCESS_TOKEN"</ams:description>
</ams:fault>

```

In this tutorial, you have prototyped an API and tested it along with its older and published versions. In the [next tutorial](#), you can learn how to 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 sign 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 you how to publish an API to its default API Store:

1. Sign in to the WSO2 API Publisher as a user who has the publisher role assigned.
<https://<hostname>:9443/publisher>
2. Click on the API that you prototyped in the previous tutorial (e.g., PhoneVerification 2.0.0).

All APIs

API Name	Version	Owner	Users	Status	Action
PhoneVerification	1.0.0	admin	1 User	PUBLISHED	
PhoneVerification	2.0.0	admin	1 User	PROTOTYPED	
PizzaShackAPI	1.0.0	admin	0 Users	PUBLISHED	

3. Click **GO TO OVERVIEW**.
4. Go to the API's **Lifecycle** tab and click **Publish**.

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

PhoneVerification - 2.0.0

Current State: **PROTOTYPED**

Require re-subscription when publish the API

Deprecate old versions after publish the API

Publish Demote to Created

Tip: Leave the **Require Re-Subscription** check box cleared if you want all users who are subscribed to the older version of the API to be automatically subscribed to the new version. If not, they need to subscribe to the new version again. You can choose to deprecate old versions of this API at this stage by selecting the **Deprecate Old Versions** check box.

The API is now published to the default API Store.

5. Sign in to the default Store and click on the APIs menu to see the API that you just published listed there.
6. Go back to the WSO2 API Publisher and click the API that you want to deprecate (e.g., PhoneVerification 1.0.0).
7. Go to the API's **Lifecycle** tab and click **Deprecate**.

PhoneVerification - 1.0.0

Current State: **PUBLISHED**

Block Deploy as a Prototype Demote to Created **Deprecate**

The API is now deprecated.

8. Go back to the WSO2 API Store, click the **Applications** menu and open the TestApp Application, which you use to [subscribe to the API](#). Click the **Subscriptions** tab to see the deprecated API. The subscriptions made to the older API versions should be deprecated now.

TestApp

Details	Production Keys	Sandbox Keys	Subscriptions												
<input type="text" value="Filter by ..."/> <table border="1"> <thead> <tr> <th>API Name</th> <th>Subscription Tier</th> <th>Status</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td>PhoneVerification - 1.0.0 DEPRECATED</td> <td>Bronze</td> <td>UNBLOCKED</td> <td>Unsubscribe</td> </tr> <tr> <td>PhoneVerification - 2.0.0 PUBLISHED</td> <td>Bronze</td> <td>UNBLOCKED</td> <td>Unsubscribe</td> </tr> </tbody> </table> <p>Show 10 entries Showing 1 to 2 of 2 entries</p>				API Name	Subscription Tier	Status	Actions	PhoneVerification - 1.0.0 DEPRECATED	Bronze	UNBLOCKED	Unsubscribe	PhoneVerification - 2.0.0 PUBLISHED	Bronze	UNBLOCKED	Unsubscribe
API Name	Subscription Tier	Status	Actions												
PhoneVerification - 1.0.0 DEPRECATED	Bronze	UNBLOCKED	Unsubscribe												
PhoneVerification - 2.0.0 PUBLISHED	Bronze	UNBLOCKED	Unsubscribe												

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.

Customize API Life Cycle

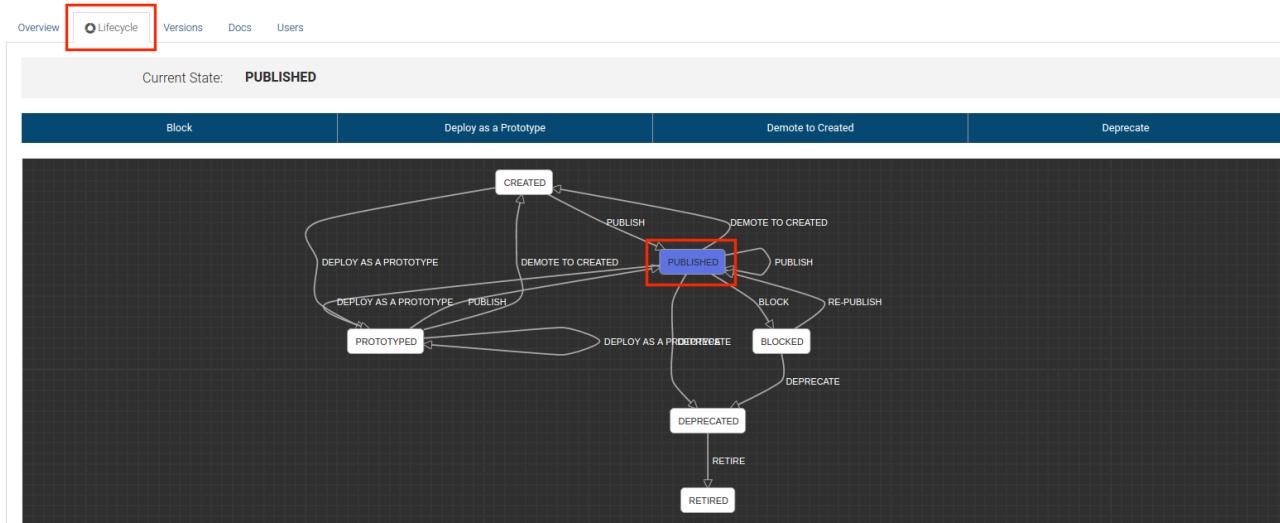
APIs created in WSO2 API Manager have their own life cycle consisting of the following: a set of life cycle states, specific actions for each state transition, and a checklist of items before a state transition occurs. An API has a predefined life cycle consists of [six states](#). This tutorial demonstrates how you can edit the default API lifecycle and customize it according to your requirements.

This capability is not available in API Manager versions prior to 1.10.0.

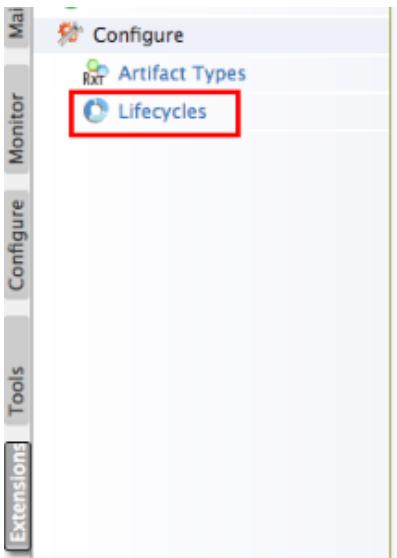
Follow the steps below to add a new state to the default life cycle.

1. Log into the API Publisher and select an API you have previously created. Click Lifecycle to view the current states available by default.

PizzaShackAPI - 1.0.0



2. Open the management console: <https://localhost:9443/carbon>.
3. Navigate to **Extensions > Configure > Lifecycles**.



4. Click the **View/Edit** link corresponding to the default API LifeCycle.

Home > Extensions > Configure > Lifecycles

Lifecycles

Lifecycle Name	Actions
APILifeCycle	View/Edit Delete
ServiceLifeCycle	View/Edit Delete

[Add New Lifecycle](#)
 [Find Resources/Collections With Lifecycles](#)

5. You will be able to see the APILifeCycle configurations.

```

<aspect name="APILifeCycle"
class="org.wso2.carbon.governance.registry.extensions.aspects.Default
LifeCycle">
    <configuration type="literal">
        <lifecycle>
            <scxml xmlns="http://www.w3.org/2005/07/scxml"
version="1.0"
initialstate="Created">
                <state id="Created">
                    <datamodel>
                        <data name="checkItems">
                            <item name="Deprecate old versions after
publish the API" forEvent="">
                                </item>
                            <item name="Require re-subscription when
publish the API" forEvent="">
                                </item>
                        </data>
                        <data name="transitionExecution">
                            <execution forEvent="Deploy as a
Prototype">

```

```

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Publish"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    </data>
</datamodel>
<transition event="Publish" target="Published"/>
<transition event="Deploy as a Prototype"
target="Prototyped"/>
</state>
<state id="Prototyped">
    <datamodel>
        <data name="checkItems">
            <item name="Deprecate old versions after
publish the API" forEvent="">
                </item>
            <item name="Require re-subscription when
publish the API" forEvent="">
                </item>
            </data>
            <data name="transitionExecution">
                <execution forEvent="Publish"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Demote to Created"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    </data>
</datamodel>
<transition event="Publish" target="Published"/>
<transition event="Demote to Created"
target="Created"/>
    <transition event="Deploy as a Prototype"
target="Prototyped"/>
</state>

<state id="Published">
    <datamodel>
        <data name="transitionExecution">
            <execution forEvent="Block"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Deprecate"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Demote to Created"

```

```

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Deploy as a
Prototype">

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    </data>
    </datamodel>
    <transition event="Block" target="Blocked"/>
    <transition event="Deploy as a Prototype"
target="Prototyped"/>
        <transition event="Demote to Created"
target="Created"/>
            <transition event="Deprecate"
target="Deprecated"/>
                <transition event="Publish" target="Published"/>

            </state>
            <state id="Blocked">
                <datamodel>
                    <data name="transitionExecution">
                        <execution forEvent="Re-Publish"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    <execution forEvent="Deprecate"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    </data>
    </datamodel>
    <transition event="Deprecate"
target="Deprecated"/>
        <transition event="Re-Publish"
target="Published"/>
            </state>
            <state id="Deprecated">
                <datamodel>
                    <data name="transitionExecution">
                        <execution forEvent="Retire"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
    </execution>
    </data>
    </datamodel>
    <transition event="Retire" target="Retired"/>
</state>
<state id="Retired">
</state>
</scxml>

```

```

        </lifecycle>
    </configuration>
</aspect>

```

6. Copy the following sample and paste in the file, to add a sample state to the API Lifecycle.

```

<state id="Rejected">
<datamodel>
    <data name="checkItems">
        <item name="Deprecate old versions after rejecting the
API" forEvent="">
            </item>
        <item name="Remove subscriptions after rejection" forEvent="">
            </item>
    </data>
    <data name="transitionExecution">
        <execution forEvent="Re-Submit"
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
            </execution>
        <execution forEvent="Retire"
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
            </execution>
    </data>
</datamodel>
    <transition event="Re-Submit" target="Published"/>
    <transition event="Retire" target="Retired"/>
</state>

```

The sample **REJECTED** state is added between **PUBLISHED** and **RETIRIED**. It uses the Re-submit and Retire state transition events to change to the consequent states. Custom checklist items are also given under "checkItems", which are tasks to be done in a state transition. You can select/deselect these items in the management console.

For all state transitions, the same execution class is used ([org.wso2.carbon.apimgt.impl.executors.APIExecutor](#)). However, you can plug your own execution code when modifying the life cycle configuration. For example, if you want to add notifications for a specific state transition, you can plug your own custom execution class for that particular state in the API life cycle. Any changes are updated in the **Lifecycle** tab accordingly.

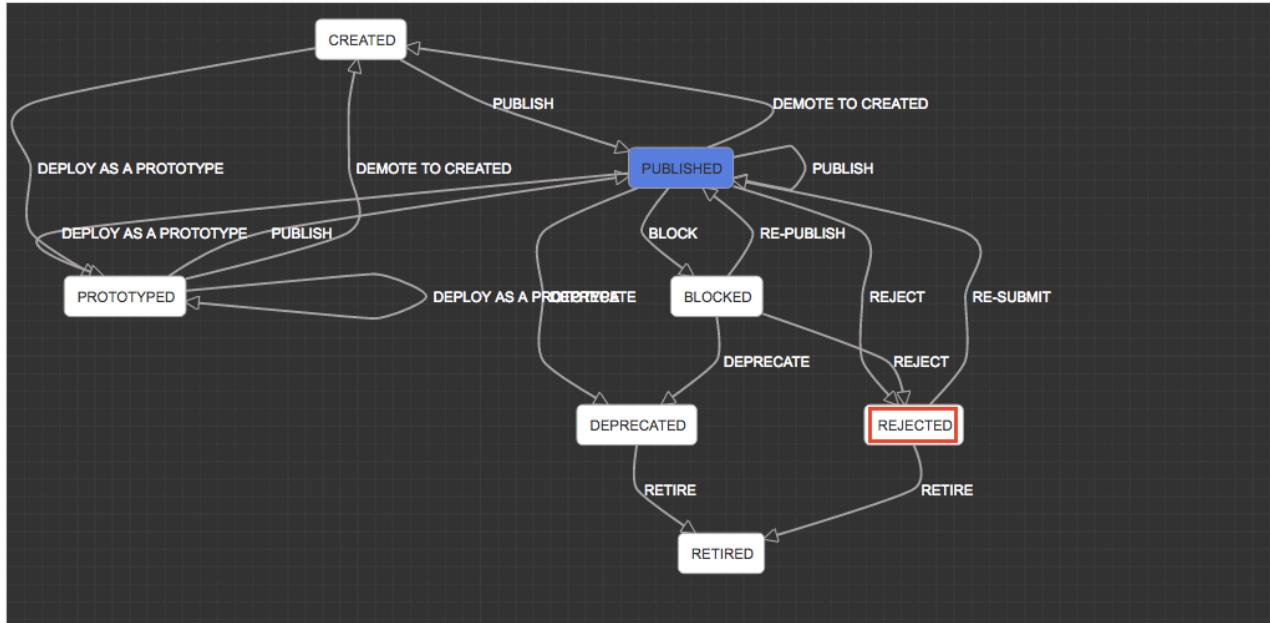
7. Add a new transition event under the PUBLISHED state, to show the state change to REJECTED.

```

...
<transition event="Reject" target="Rejected"/>
...

```

8. Go to <API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/locales/jaggery/locale_default.json . Add "reject": "Rejected" to make the transition event visible in API Publisher. Note that the key value in the JSON pair should be lowercase.
9. Re-open API Publisher by restarting the server and check the Lifecycle to see the changes.



Consider the following points when extending and customizing the API lifecycle XML configuration.

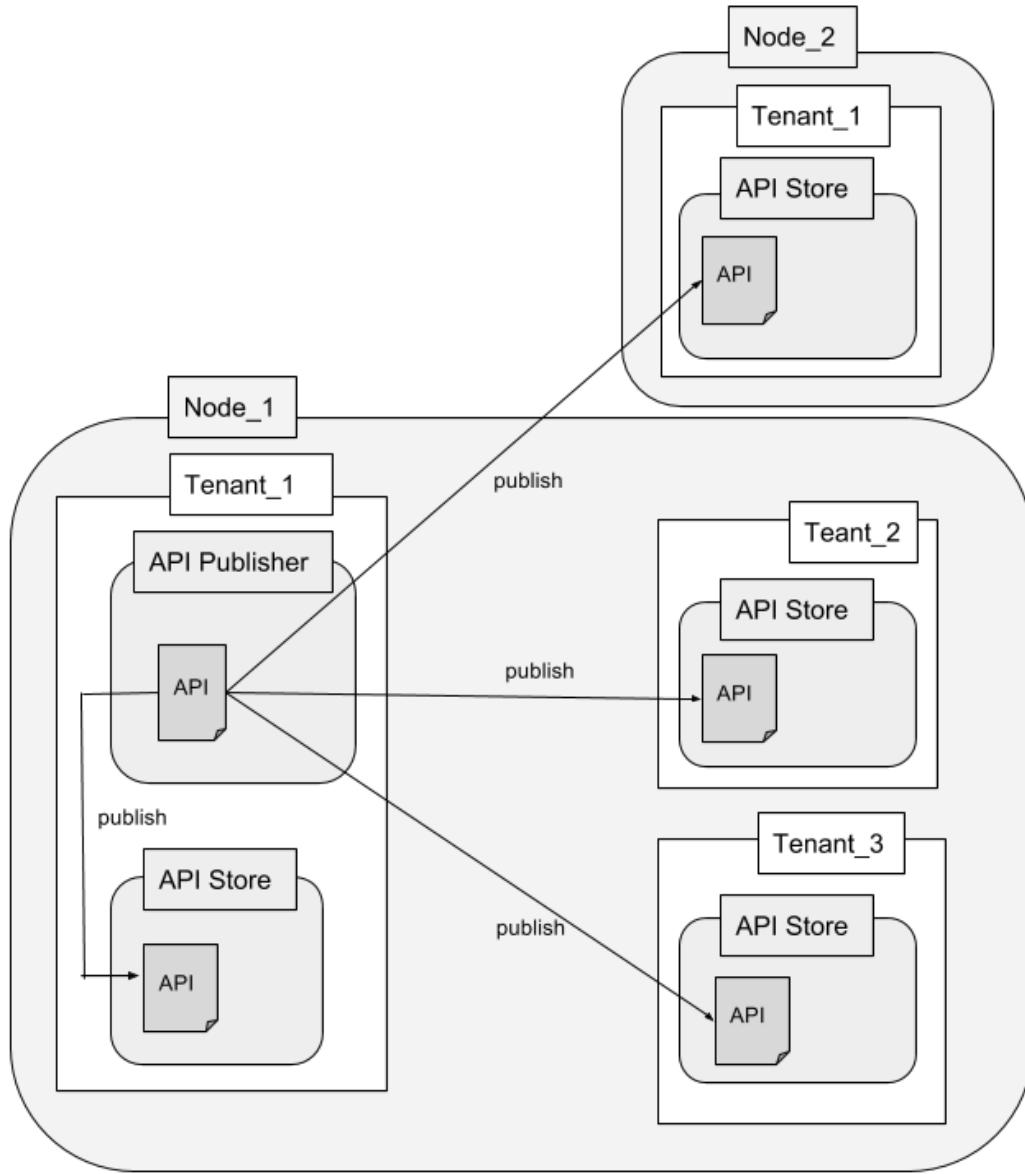
- Do not change the life cycle name since it needs to be engaged with the APIs dynamically.
- Make sure you keep the **PUBLISHED** and **PROTOTYPED** states as those two states will be used by API Publisher in the API creation wizard.

For more details on customizing the API lifecycle, see [Extending the API Life Cycle](#).

Publish to Multiple External API Stores

You can share an API with 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; however, the users must visit your (the original publisher's) store to subscribe to the APIs.

Following diagram explains publishing to multiple API Stores by an API Publisher.



The API Publisher of Tenant_1 located in Node_1 is publishing an API to its API Store. Other than that API Publisher publish the API to following three external stores.

1. API Store of Tenant_2 in same node.
2. API Store of Tenant_3 in same node.
3. API Store of Tenant_1 in Node 2

The capability to publish to external API Stores is not configured by default. Follow the steps below to configure it. In this guide, we use two separate instances of WSO2 API Manager and we publish from one instance to the Store of the other instance.

1. Copy the WSO2 API Manager product pack to two different locations.
If needed, you can download the WSO2 API Manager product pack from [here](#).
2. Go to the `<API-M_HOME>/repository/conf/carbon.xml` file of the **second** instance and change its `port` by `an offset of 1`.
The port offset is set to avoid the port conflicts that occur when you run more than one WSO2 product on the same host.

<Offset>1</Offset>

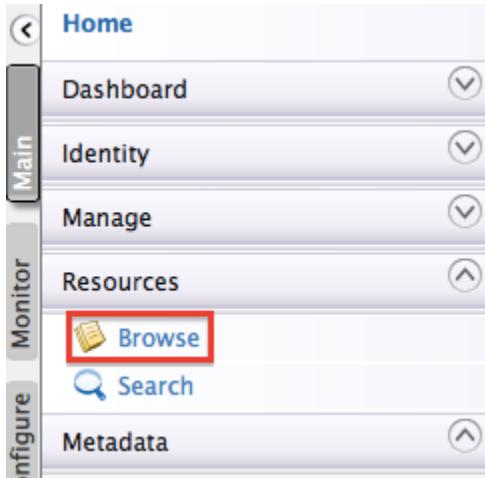
3. Start both API-M servers.

Let's publish from the first instance of WSO2 API Manager to the Store of the second instance, which in this tutorial we consider as the external API Store.

4. Sign in to the WSO2 API-M management console of the **first** instance (<https://<Server Host>:9443/carbon>) as admin.

In a **multi-tenant environment**, you must sign in using the tenant's credentials.

5. Click **Main > Resources > Browse**.



The Registry opens.

6. Go to the `/_system/governance/apimgt/externalstores/external-api-stores.xml` resource.

Browse

Root /

Location: /

Tree view **Detail view**

```

    /_
      - _system
      - config
      - governance
      - apimgt
      - applicationdata
      - customsequences
      - externalstores
        - external-api-stores.xml (highlighted)
      - statistics
      - event
      - forumtopics
      - permission
      - repository
      - trunk
      - local
  
```

7. Click the **Edit as Text** link, uncomment the `<StoreURL>` element under the `<ExternalAPIStores>` element, and add the details of each external API store that you need to publish APIs to. In this example,
 - `http://localhost:9764/store` is the API Store of the second WSO2 API Manager instance.
 - You publish to its super tenant's Store (`admin/admin`).
 - For this tutorial change the `DisplayName` to `Store2`, so that it is clear that we are referring to the second WSO2 API-M instance, which we are using as the external Store.
 - The port is `9764` as you incremented it by `1` in [step 2](#).
 - If the second WSO2 API Manager instance has multiple tenants and you want to publish to a tenant's Store, the tenant's Store URL and credentials must be given here.

```
<ExternalAPIStores>
    <StoreURL>http://localhost:9763/store</StoreURL>
        <ExternalAPIStore id="Store2" type="wso2"
    className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher">
            <DisplayName>Store2</DisplayName>
            <Endpoint>http://localhost:9764/store</Endpoint>
            <Username>admin</Username>
            <Password>admin</Password>
        </ExternalAPIStore>
    </ExternalAPIStores>
```

If you want to configure more than one external store, change the configuration in `<ExternalAPIStore>` and add it to the **external-api-stores.xml**.

For example if we have three API Stores one is super tenant and other two are tenant stores, we can configure these three external stores as below.

```

<ExternalAPIStores>

    <!--Configuration to set the store URL of the current running
APIM deployment.
APIs published to external stores will be redirected to this
URL-->

    <StoreURL>http://<ip_address>:<port>localhost:9763/store</Store
URL>

        <ExternalAPIStore id="SLStore" type="wso2"
className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublis
her">
            <DisplayName>SL-Store</DisplayName>

            <Endpoint>http://<ip_address>:<port>/store</Endpoint>
                <Username>admin</Username>
                <Password>admin</Password>
            </ExternalAPIStore>

            <ExternalAPIStore id="USStore" type="wso2"
className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublis
her">
                <DisplayName>US-Store</DisplayName>

            <Endpoint>http://<ip_address>:<port>/store</Endpoint>
                <Username>{tenantadmin_username}@{tenant_domain}</Username>
                <Password>{tenantadmin_password}</Password>
            </ExternalAPIStore>

            <ExternalAPIStore id="UKStore" type="wso2"
className="org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublis
her">
                <DisplayName>UKStore</DisplayName>

            <Endpoint>http://1<ip_address>:<port>/store</Endpoint>
                <Username>{tenantadmin_username}@{tenant_domain}</Username>
                <Password>{tenantadmin_password}</Password>
            </ExternalAPIStore>

        </ExternalAPIStores>

```

In a **multi-tenant environment**, each tenant can publish to different external Stores by changing the

above file in their tenant space. For more information on how APIs appear and are available for subscription in a multi-tenant environment, see [API visibility and subscription](#). Note that publishing to an external Store only means that the API is advertised there. To subscribe, you must always register and sign in to the original publisher's tenant Store.

Note the following in the configuration above:

Element	Description
<ExternalAPIStore id="" type="" className="">	<p><code>id</code>: The external store identifier, which is a unique value. <code>type</code>: The type of the Store. This can be a WSO2-specific API Store or an external one, which has a different implementation from the default API Store. <code>className</code>: The implementation class inside the WSO2 API Manager distribution.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The default <code>className</code> specified is <code>org.wso2.carbon.apimgt.impl.publishers.WSO2APIPublisher</code>, which is used when WSO2 specific API Store is used. However, if you are using an external API Store, the class should be customized by extending the interface <code>org.wso2.carbon.apimgt.api.model.APIPublisher</code>, and the fully qualified class name of the new implementation should be used for the <code>className</code> parameter.</p> </div>
<StoreURL>	The URL of the API Store of the current API-M deployment. This is the URL to the API in the original publisher's store. APIs that are published to external stores are redirected to this URL.
<DisplayName>	The name of the external API Store that is displayed in the Publisher UI.
<Endpoint>	The URL of the external API Store.
<Username> and <Password>	The credentials of a user who has permissions to create and publish APIs.

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

- Click **Save Content**.
- Sign in to the API Publisher of the first instance as admin/admin and if you do not have any APIs that are in the published state created, [create an API](#).

In a multi-tenant environment, sign in to the API Publisher using your tenant's credentials.

- Click on the newly created or existing API. Here you see a new tab named **External API Stores** added to the API Publisher console.

This tab is only visible when viewing API's that are in the published state.

PhoneVerification - 1.0.0

Overview	 Lifecycle	Versions	Docs	Users	External API Stores
		Visibility	Public		
 0 Users		Context	/phoneverify/1.0.0		
 PUBLISHED		Production URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx		
		Sandbox URL	http://ws.cdyne.com/phoneverify/phoneverify.asmx		
 Docs		Date Last Updated	July 30, 2016 10:36:31 AM GMT+05:30		
 View in Store		Tier Availability	Bronze,Gold,Unlimited,Silver		
		Default API Version	None		
		Published Environments	Production and Sandbox		

- 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 receives a request to delete the API.

11. Select the Store that you want to publish to (in this case, Store2) and click **Save**.

PhoneVerification - 1.0.0

Overview	 Lifecycle	Versions	Docs	Users	External API Stores
<div style="border: 1px solid #ccc; padding: 10px;"> <p>Display in External Stores: <input checked="" type="checkbox"/> Store2</p> <p>Save Cancel</p> </div>					

12. Sign in to the external API Store (in this case, `http://localhost:9764/store`) and click on the API that you just published.

A link appears as **Visit Publisher Store**, which directs you to the original publisher's store (in this case, `http://localhost:9763/store`) through which you can subscribe to the API.

PhoneVerification - 1.0.0

Version: 1.0.0
By: admin
Updated: 30/Jul/2016 10:42:45 AM IST
Status: PUBLISHED

[Visit Publisher Store](#)

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

Publish through Multiple API Gateways

You can configure multiple API Gateway environments in WSO2 API Manager that publish to a single API Store when you require distributing the gateway load comes in. It helps you to distribute the API Gateway load to multiple nodes and also gives you some logical separation (e.g., production vs. sandbox) between the APIs in the API Store. When you publish an API through multiple Gateway environments, the APIs in the API Store will have different server hosts and ports.

The steps below explain how to configure and publish to multiple Gateways. In this guide, we set up three (3) WSO2 API Manager (WSO2 API-M) instances in the same server. In a typical production environment, the Gateways will ideally be in separate servers.

- **Instance 1:** Acts as the node that provides the API Publisher, Store, and the Key Manager functionality.
- **Instance 2:** Acts as a production Gateway node.
- **Instance 3:** Acts as a sandbox Gateway node.

1. Copy the WSO2 API Manager (WSO2 API-M) product pack into three (3) separate folders. Let's add offsets to the default ports of the two Gateway instances. A port offset ensures that there are no port conflicts when more than one WSO2 product runs on the same server.
2. Open the <API-M_HOME>/repository/conf/carbon.xml file in the **second** API Manager instance, and add an offset of 1 to its default port. This increments its default server port, which is 9443, by 1.

```
<Offset>1</Offset>
```

3. Open the <API-M_HOME>/repository/conf/carbon.xml file in the **third** API Manager instance and add an offset of 2 to its default port. This increments its default server port, which is 9443, by 2.

```
<Offset>2</Offset>
```

4. Open the <API-M_HOME>/repository/conf/api-manager.xml files in the **second and the third** API Manager instances and set the <EnableThriftServer> property to false. This is done to disable the thrift server in the two Gateway instances. Thrift server is needed for the Key Manager functionality. It is not needed in the Gateway instances.

```
<EnableThriftServer>false</EnableThriftServer>
```

5. Open the <API-M_HOME>/repository/conf/api-manager.xml files in the **second and the third** Gateway instances and change the following. This is done for the two Gateway instances to be able to communicate with the Key Manager that is in the

first API Manager instance.

```
<AuthManager>
    <ServerURL>https://<IP of the first
instance>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
</AuthManager>
...
<APIKeyValidator>
    <ServerURL>https://<IP of the first
instance>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ....
    <RevokeAPIURL>https://<IP of the first
instance>:8243/revoke</RevokeAPIURL>
</APIKeyValidator>
```

6. Open the `<API-M_HOME>/repository/conf/api-manager.xml` files in **all** the Gateway instances and uncomment the following configuration:

```
<ThriftClientPort>10397</ThriftClientPort>
```

You are done configuring the two API Gateway instances.

7. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file in the **first** API Manager instance, add two API Gateway environments under the `<Environments>` element, and comment out the `<environment>` element that comes by default. This is done to point to the two API Gateway instances from the first instance.

Example

```
<Environments>
    <Environment type="production">
        <Name>Production Gateway</Name>
        <Description>Production Gateway Environment</Description>
        <ServerURL>https://localhost:9444/services/</ServerURL>
        <Username>admin</Username>
        <Password>admin</Password>

        <GatewayEndpoint>http://localhost:8281,https://localhost:8244</GatewayEndpoint>
    </Environment>
    <Environment type="sandbox">
        <Name>Sandbox Gateway</Name>
        <Description>Sandbox Gateway Environment</Description>
        <ServerURL>https://localhost:9445/services/</ServerURL>
        <Username>admin</Username>
        <Password>admin</Password>

        <GatewayEndpoint>http://localhost:8282,https://localhost:8245</GatewayEndpoint>
    </Environment>
</Environments>
```

Tip: The Gateway environment names must be unique.

Tip: The environments you add here will be visible in a drop-down list in the API Console tab of the API Store. It allows subscribers to send API requests to any selected Gateway.

The screenshot shows the WSO2 API Manager API Store interface. At the top, there are tabs for Overview, API Console, Documentation, Forum, and SDKs. The API Console tab is active. Below the tabs, there are three dropdown menus: 'Try' set to 'DefaultApplication', 'Using' set to 'Production', and 'On Environment' set to 'Production and Sandbox'. The 'On Environment' dropdown is highlighted with a red border. At the bottom, there are fields for 'Set Request Header' with 'Authorization : Bearer' and 'Access Token' options. On the right side, there is a 'Swagger (/swagger.json)' section with links for 'Show/Hide', 'List Operations', and 'Expand Operations'. Below the header, there is a section titled 'default' with three buttons: 'GET /', 'POST /', and 'PUT /'. The 'GET /' button is highlighted in blue, while 'POST /' and 'PUT /' are in green and orange respectively.

To stop a given Gateway environment from being displayed in the API Console tab, you can set the `api-console` attribute to `false` in the `<environment>` element in the `api-manager.xml` file. For example, `<Environment type="production" api-console="false">`.

8. Start all the WSO2 API-M instances.
9. Sign in to the API Publisher in the **first** WSO2 API-M instance and choose to edit an API.

All APIs

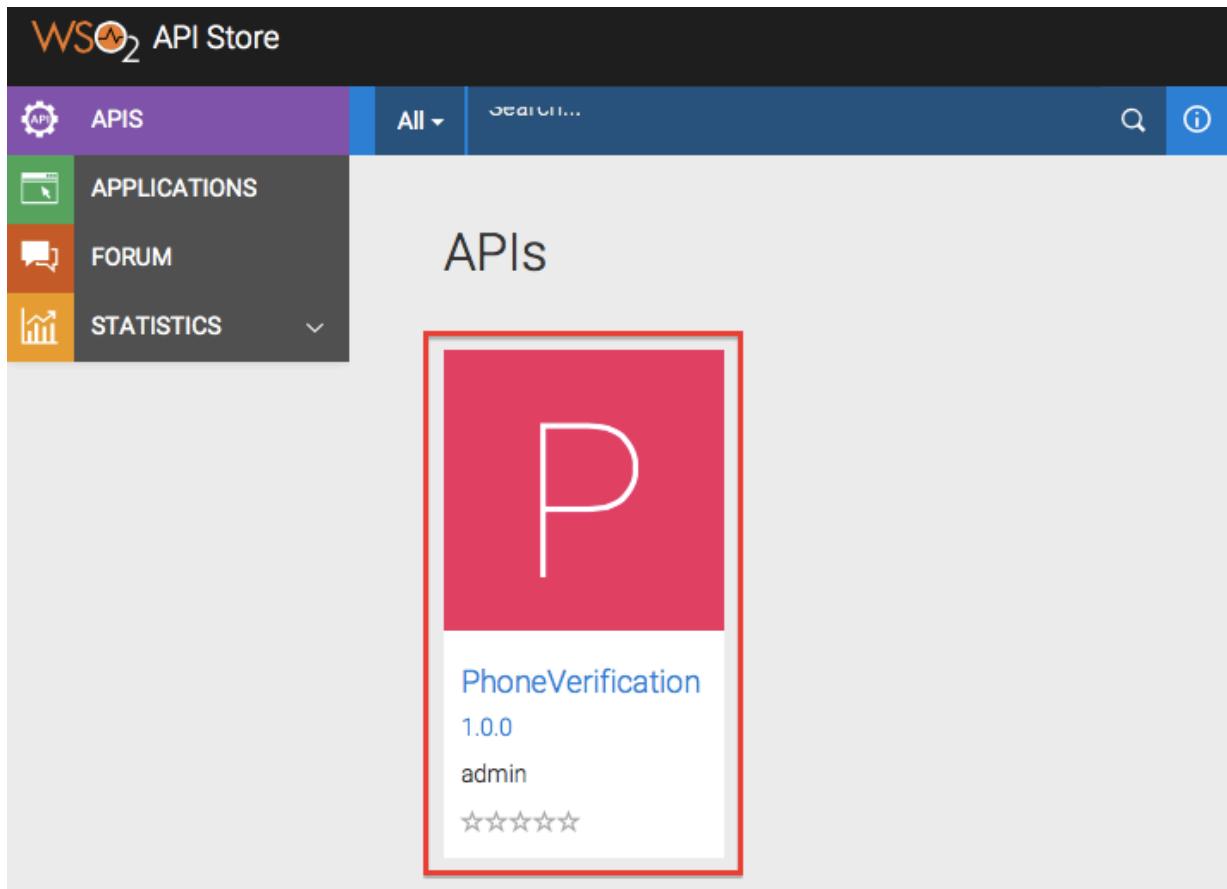
	
PizzaShackAPI	PhoneVerifica...
1.0.0	1.0.0
admin	apicreator
1 User	0 Users
PUBLISHED	CREATED
 	  Edit

10. Navigate to the **Manage** tab, expand the **Gateway Environments** section. Note the two Gateway environments listed there.

Gateway Environments			
Environment Name	Type	Description	
<input checked="" type="checkbox"/> Production Gateway	production	Production Gateway Environment	
<input checked="" type="checkbox"/> Sandbox Gateway	sandbox	Sandbox Gateway Environment	

In a typical production setup, you will publish the API to the sandbox Gateway first, and thereafter publish it to the production Gateway. In this case, let's publish to both.

11. Select both Gateways and **Save and Publish** the API.
12. Sign in to the API Store (of the **first** instance) and click the API to open it.



13. In the API's **Overview** tab, note that it has two sets of URLs for the two Gateway instances:

PhoneVerification - 1.0.0

Version: 1.0.0
By: admin
Updated: 09/Mar/2017 16:06:05 PM IST
Status: PUBLISHED
Rating: ★★★★★

Applications
DefaultApplication

Tiers
Unlimited

Subscribe

Overview API Console Documentation Forum SDKs

Sandbox Endpoints

Sandbox Gateway URLs:

- https://localhost:8245/phoneverify/1.0.0
- http://localhost:8282/phoneverify/1.0.0

Production Endpoints

Production Gateway URLs:

- https://localhost:8244/phoneverify/1.0.0
- http://localhost:8281/phoneverify/1.0.0

You have published an API to the API Stores through multiple Gateway environments.

If you have published your API through more than one Gateway,

When you have generated keys for the Applications, the sample cURL command shows how to generate an access token using the Password Grant type provides the Gateway URL of the first published Gateway Environments listed in API Publisher as shown in the step 10.

Generating Access Tokens

The following cURL command shows how to generate an access token using the Password Grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic bTJmDBzcWZUTENVQ2InWGRic3FuTmlTOVhzYTpETlJBV0hZUExnZmZkbmVlc0tHTXkxQLB3ckFh" \
https://localhost:8244/token
```

In a similar manner, you can generate an access token using the Client Credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic bTJmDBzcWZUTENVQ2InWGRic3FuTmlTOVhzYTpETlJBV0hZUExnZmZkbmVlc0tHTXkxQLB3ckFh" \
https://localhost:8244/token
```

Change this gateway URL according to the Gateway that you want to publish the API if you are using this Curl command to generate acces tokens.

If you wish to use the API-M pack that you used as the first instance to tryout other tutorials, please ensure to delete the API Gateway configurations that you added in [step 6](#), and uncomment the default <Environment> configurations in the <API-M>/repository/conf/api-manager.xml file.

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 subscription 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 that s/he has 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.

Using block subscription to an API, we can control only the subscriptions created for a specific API by a user. If you want to block all API requests from a specific application/user/specific IP addresse or to a specific API, you can use [request blacklisting](#).

Blocking can be done at 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. This is 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 it.

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 descriptions on the concepts that you need to know when you block subscriptions to an API:

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

1. Sign in to the WSO2 API Publisher.

- Create two APIs by the names `TestAPI1` and `TestAPI2` and publish them to the WSO2 API Store. For more information, see [Create and Publish an API](#).

All APIs

The screenshot shows the 'All APIs' page with a search bar at the top. Below the search bar are two rows of API cards. The first row contains a card for 'PhoneVerificationAPI' (version 1.0.0, published by admin, 1 user, PUBLISHED) and a card for 'PizzaShackAPI' (version 1.0.0, published by admin, 0 users, PUBLISHED). The second row contains two cards for 'TestAPI1' and 'TestAPI2', both version 1.0.0, published by admin, 0 users, and PUBLISHED. The 'TestAPI1' and 'TestAPI2' cards are highlighted with a red border.

API Name	Version	Published By	Users	Status
PhoneVerificationAPI	1.0.0	admin	1 User	PUBLISHED
PizzaShackAPI	1.0.0	admin	0 Users	PUBLISHED
TestAPI1	1.0.0	admin	0 Users	PUBLISHED
TestAPI2	1.0.0	admin	0 Users	PUBLISHED

- Sign in to the WSO2 API Store. Click on the **APIs** menu.
Note that the two APIs are visible in the APIs page.
- Subscribe to both APIs using the same application.
You can use the default application or create your own.

The screenshot shows the 'View API' page for 'TestAPI2 - 1.0.0'. On the left is a large purple square icon with a white 'T'. To its right are details: Version 1.0.0, By admin, Updated 30/Jul/2016 12:41:56 PM IST, Status PUBLISHED, and Rating 5 stars. To the right is a sidebar with a red border containing 'Applications' (set to DefaultApplication), 'Tiers' (set to Bronze), and a 'Subscribe' button.

- Click the **View Subscriptions** button when prompted.
The **Subscriptions** tab opens.
- Click the **Production Keys** tab and click **Generate Keys** to create an application access token.
If you have already generated an access token before, click **Re-generate** to renew the token.

DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

Show Keys

Consumer Key
.....

Consumer Secret
.....

Grant Types
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can select one or more grant types.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code

Callback URL
.....

Update

Generating Access Tokens
The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

Generate a Test Access Token

Access Token
.....

Above token has a validity period of **3600** seconds. And the token has (**am_application_scope default**) scopes.

Scopes
No Scopes Found..

Validity period
3600 Seconds.

Re-generate

7. Invoke both APIs using the access token you received in the previous step. We use cURL here. The command is,

Command

```
curl -k -H "Authorization: Bearer <access_token>" '<API_URL>'
```

Be sure to replace the placeholders as follows:

- <access_token> : Give the token generated in [step 6](#).
- <API_URL> : Go to the API's **Overview** tab in the API Store and copy the production URL and append the payload to it.

Here's an example:

```
curl -k -H "Authorization :Bearer dda01682642ebf1285430d4d276201e5"
'https://localhost:8243/phoneverify3/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
```

```
curl -k -H "Authorization :Bearer dda01682642ebf1285430d4d276201e5" 'https://localhost:8243/phoneverify3/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
<?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 />
<TelecomCounty />
<TelecomState>TF</TelecomState>
<TelecomZip />
<TimeZone />
<Lat />
<Long />
<Wireless>false</Wireless>
```

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

8. Sign back in to the API Publisher and click **Manage Subscriptions**.

It shows all APIs/applications that each user is subscribed to.

User	Application	API	Still Subscribed?	Action
admin	DefaultApplication	PhoneVerification-1.0.0	Yes	Production & Sandbox Block
admin	DefaultApplication	TestAPI1-1.0.0	Yes	Production & Sandbox Block
admin	DefaultApplication	TestAPI2-1.0.0	Yes	Production & Sandbox Block

9. Block subscription for TestAPI1 using the DefaultApplication by selecting the **Production& Sandbox** option and clicking the **Block** link.

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

Manage Subscriptions

Manage Subscriptions					
User	Application	API	Still Subscribed?		
admin	DefaultApplication	PhoneVerification-1.0.0	Yes	Production & Sandbox	<input checked="" type="checkbox"/> Block
admin	DefaultApplication	TestAPI1-1.0.0	Yes	Production & Sandbox	<input checked="" type="checkbox"/> Unblock
admin	DefaultApplication	TestAPI2-1.0.0	Yes	Production & Sandbox	<input checked="" type="checkbox"/> Block

Show 10 entries | Showing 1 to 3 of 3 entries

10. Sign back in to the API Store.
11. 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.

Note that you can invoke TestAPI2 again, but 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 block is removed.

```
curl -k -H "Authorization :Bearer 8f1cd672d11a953af863c622e71d1721" 'https://[REDACTED]:8243/test1/1.0.0/CheckPhoneNumber?PhoneNumber=18006785432&LicenseKey=0'
<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: /test1/1.0.0, version: 1.0.0</ams:description>
```

12. Go to the **Applications** page in the API Store, select the application that you used to subscribe to the API. Note that your subscription is blocked.

DefaultApplication

Details	Production Keys	Sandbox Keys	Subscriptions
<input type="text"/> Filter by ...			
API Name	Subscription Tier	Status	Actions
PhoneVerification - 1.0.0 PUBLISHED	Bronze	UNBLOCKED	<input type="button"/> Unsubscribe
TestAPI1 - 1.0.0 PUBLISHED	Unlimited	BLOCKED	<input type="button"/> Unsubscribe
TestAPI2 - 1.0.0 PUBLISHED	Bronze	UNBLOCKED	<input type="button"/> Unsubscribe

Show 10 entries | Showing 1 to 3 of 3 entries

13. Go back to the API Publisher's **Manage Subscriptions** page and **unblock** the subscription.
14. Invoke TestAPI1 again.

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 API Manager (WSO2 API-M), 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](#) to create and publish the PhoneVerification API and then the [Subscribe to an API](#) to subscribe to the 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. Sign in to the API Store and select the PhoneVerification API.

The screenshot shows the API Store interface for the PhoneVerification API version 1.0.0. On the left is a large red square icon with a white letter 'P'. To its right is a white card containing the following details:

- Version:** 1.0.0
- By:** admin
- Updated:** 22/Jul/2016 17:12:01 PM IST
- Status:** PUBLISHED
- Rating:** ★★★★☆

To the right of the card is a sidebar with two dropdown menus and a button:

- Applications:** DefaultApplication
- Tiers:** Bronze
- Subscribe**

A red rectangular box highlights the Applications, Tiers, and Subscribe area.

2. Go to the Default Application, click the **Production Keys** tab and generate an access token. If you already have an access token for the application, you have to regenerate it after 1 hour.

DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

Show Keys

Consumer Key
.....

Consumer Secret
.....

Grant Types
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can select one or more grant types.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code

Callback URL
.....

Update

Generating Access Tokens
The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

Generate a Test Access Token

Access Token
.....

Above token has a validity period of **3600** seconds. And the token has (**am_application_scope default**) scopes.

Scopes
No Scopes Found..

Validity period
3600 Seconds.

Re-generate

Let's invoke this API.

- Click on the API, then go to its **API Console** tab and expand the GET method.

PhoneVerification - 1.0.0

The screenshot shows the WSO2 API Manager interface for the 'PhoneVerification' API version 1.0.0. The top navigation bar includes tabs for 'Overview', 'API Console' (which is selected and highlighted with a red border), 'Documentation', and 'Forum'. Below the navigation, there's a large red square icon with a white letter 'P'. To the right of the icon, details about the API are listed: Version 1.0.0, By admin, Updated 22/Jul/2016 17:12:01 PM IST, Status PUBLISHED, and Rating 5 stars. On the right side, there are sections for 'Applications' (with a dropdown menu 'Select Application...'), 'Tiers' (set to Unlimited), and a 'Subscribe' button. Below these sections, there are dropdown menus for 'Try' (set to 'TestApp') and 'Using' (set to 'Production'). A 'Set Request Header' section shows an Authorization header with the value 'Authorization : Bearer 4dd4bf2e-eae0-31a0-8850-6338213a8100'. At the bottom right, there's a link to 'Swagger (/swagger.json)' and options to 'Show/Hide', 'List Operations', and 'Expand Operations'.

4. Give values to the parameters and click **Try it out** to invoke the API.

This screenshot shows the 'Try it out' interface for the '/CheckPhoneNumber' endpoint. It features a table for 'Parameters' with two rows: 'PhoneNumber' (value: 18006785432) and 'LicenseKey' (value: 0). Both input fields are highlighted with a red border. Below the table, there's a section for 'Response Messages' showing a single row for status code 200. At the bottom, a red-bordered button labeled 'Try it out!' is visible. A green bar at the bottom indicates the method is POST and the URL is /CheckPhoneNumber. A note at the bottom states: '[BASE URL: /phoneverify/1.0.0 , API VERSION: 1.0.0]'.

Note the response that appears in the API Console. As we used a valid phone number in this example, the response returns as 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>
        <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 />

```

Note that you subscribed to the API on the Bronze throttling tier. The Bronze tier allows you to make a 1000 calls to the API per minute. If you exceed your quota, you get a throttling error as shown below.

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 the API using an unavailable resource name.

5. Go to the API's **Overview** page in the API Store and get the API's URL.

PhoneVerification - 1.0.0



Version: 1.0.0
By: admin
Updated: 22/Jul/2016 16:59:07 PM IST
Status: PUBLISHED

[Overview](#) [API Console](#) [Documentation](#) [Forum](#)

Production and Sandbox Endpoints

Production and Sandbox URLs:

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

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

6. Install cURL or any other REST client.
7. Go to the command-line invoke the API using the following cURL command.

```
curl -k -H "Authorization :Bearer <access token in step 3>" '<API's URL in step 9>/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

Note that the PhoneVerification API's resource name is **CheckPhoneNumber**, but we use an undefined resource name as **CheckPhoneNum**. Here's an example:

```
curl -k -H "Authorization :Bearer 63cc9779d6557f4346a9a28b5cf8b53" 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

8. Note that the call gets blocked by the API Gateway with a 'no matching resource' message. It doesn't reach your backend services as you are trying to access a REST resource that is not defined for the API.

```
curl -k -H "Authorization :Bearer 63cc9779d6557f4346a9a28b5cf8b53" 'https://[REDACTED]:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
<ams:fault xmlns:ams="http://wso2.org/apimanager/security"><ams:code>900006</ams:code>
<ams:message>No matching resource found in the API for the given request</ams:message>
<ams:description>Access failure for API: /phoneverify/1.0.0, version: 1.0.0. Check the
```

You have seen how the API Gateway enforces throttling and resource access policies for APIs.

Change the Default Mediation Flow of API Requests

This tutorial uses the WSO2 API Manager Tooling Plug-in.

The API Gateway has a [default mediation flow](#) for the API invocation requests that it receives. You can extend this default mediation flow to do additional custom mediation for the messages in the API Gateway. An extension is provided as a [synapse mediation sequence](#). You design all sequences using a tool such as the [WSO2 API Manager](#)

Tooling Plug-in and then store the sequence in the Gateway's registry.

Let's see how to create a custom sequence using the WSO2 API Manager Tooling Plug-in and then deploy and use it in your APIs.

1. Sign in to the API Publisher.
2. Click **Add** to create an API with the following information and then click **Next: Implement >**.

Field		Sample value
Name		YahooWeather
Context		/weather
Version		1.0
Resources	URL pattern	current/{country}/{zipcode}
	Request types	GET method to return the current weather conditions of a zip code that belongs to a particular country

The screenshot shows the API creation interface with two main tabs: "General Details" and "API Definition".

General Details:

- Name:** YahooWeather
- Context:** /weather
- Version:** 1.0
- Visibility:** Public
- Description:** (Empty text area)
- Thumbnail Image:** Placeholder for an image, with a "Select Image" button and a note about dimensions (max: 100 x 100 pixels).
- Tags:** Add tags (Empty text area)

API Definition:

- URL Pattern:** /weather/1.0
- Method Selection:** GET, POST, PUT, DELETE, PATCH, HEAD, more (with a "+ Add" button).
- Summary:** /current/{country}/{zipcode} + Summary
- Buttons:** Save, Next: Implement > (highlighted in red), and other standard save and preview buttons.

3. The **Implement** tab opens. Select **Managed API**, provide the information given in the table below and click **Manage**.

Field	Sample value
Endpoint type	HTTP/REST endpoint

Production endpoint	You can find the Yahoo weather API's endpoint from https://developer.yahoo.com/weather/ . Copy the part before the '?' sign to get this URL: https://query.yahooapis.com/v1/public/yql To verify the URL, click the Test button next to it.
Sandbox endpoint	https://query.yahooapis.com/v1/public/yql To verify the URL, click the Test button next to it.

The screenshot shows the 'Design' tab of the WSO2 API Manager interface. At the top, there are three tabs: 'Design' (selected), 'Implement', and 'Manage'. Below the tabs, under the heading 'Managed API', there is a section for providing production and sandbox endpoints. The 'Endpoint Type' is set to 'HTTP/REST Endpoint'. Under 'Production Endpoint', the URL 'https://query.yahooapis.com/v1/public/yql' is entered and marked as valid. Under 'Sandbox Endpoint', the same URL is entered and also marked as valid. There are options for 'Load Balanced' and 'Failover'. Below these fields, there is a 'Show More Options' link. Further down, there is a section for 'Message Mediation Policies' with a checkbox for enabling mediation. Another section for 'CORS configuration' has a checkbox for enabling API-based CORS. At the bottom of the page, there are two buttons: 'Save' and 'Next : Manage >'. The 'Next : Manage >' button is highlighted with a red border.

- Click **Next : Manage >** to go to the Manage tab, provide the following information and click **Save & Publish** once you are done.

Field	Sample value
Tier Availability	Gold
Keep the default values for the other attributes	

YahooWeather : /weather/1.0

Configurations

Make this the Default Version: No default version defined for the current API

Transports: * HTTPS HTTP

Response Caching:

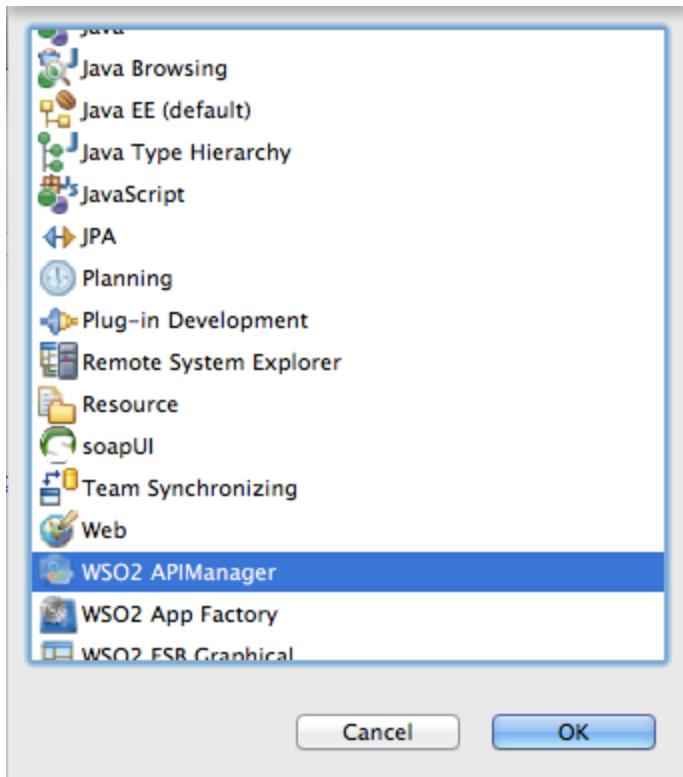
Throttling Settings

Maximum Backend Throughput: Unlimited Specify

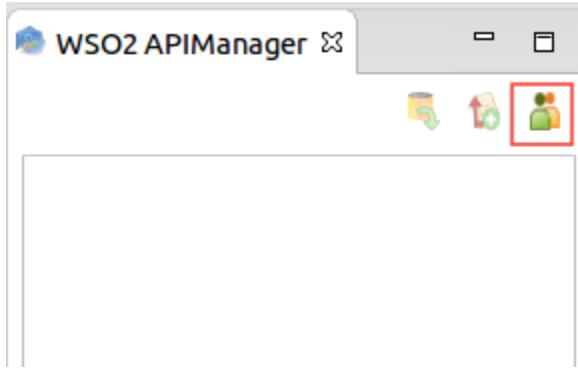
Subscription Tiers: * **Unlimited**: Allows unlimited requests
 Gold: Allows 5000 requests per minute
 Silver: Allows 2000 requests per minute
 Bronze: Allows 1000 requests per minute

Advanced Throttling Policies: Apply to API Apply per Resource

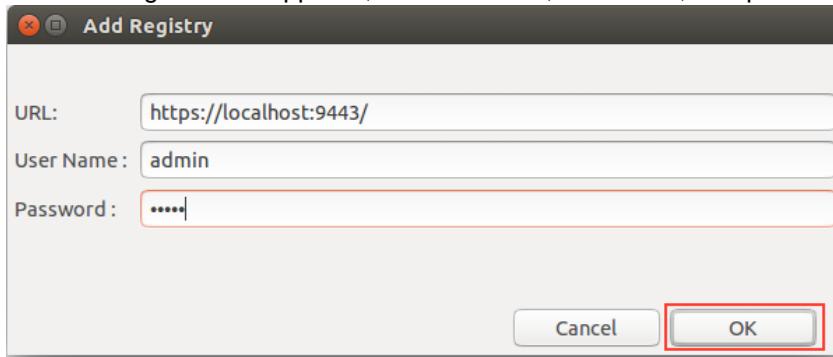
5. Download and install the [WSO2 API Manager Tooling Plug-in](#) by following one of the three possible methods described in [Installing the API Manager Tooling Plug-In](#) if you have not done so already. Start Eclipse by double clicking on the Eclipse application, which is inside the downloaded folder.
6. Navigate to the **Window** menu, click **Perspective**, **Open Perspective**, and **Other** to open the Eclipse perspective selection window.
7. On the dialog box that appears, click **WSO2 APIManager** and click **OK**.



8. On the APIM perspective, click the **Login** icon as shown below.

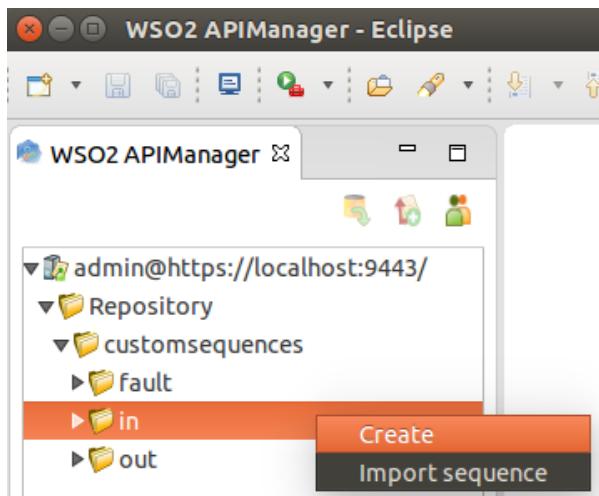


9. On the dialog box that appears, enter the URL, username, and password of the Publisher server.



10. On the tree view that appears, expand the folder structure of the existing API.

11. Right-click on the **in sequence** folder and click **Create** to create a new **in sequence**.



This is because you want the custom sequence to be invoked in the `In` direction or the request path. If you want it to be involved in the `Out` or `Fault` paths, select the respective folder under `customsequences`.

Tip: If you prefer not to use the registry to upload the sequence or want to engage a sequence to all APIs in WSO2 API-M at once, you can do so by saving the mediation sequence XML file in the file system. See [Adding Mediation Extensions](#) for details.

12. Name the sequence `YahooWeatherSequence`.
13. Your sequence now appears on the Developer Studio console. From under the **Mediators** section, drag and drop a **Property mediator** to your sequence and give the following values to the property mediator.

The **Property Mediator** has no direct impact on the message, but rather on the message context flowing through Synapse. You can retrieve the properties set on a message later through the [Synapse XPath Variables](#) or the `get-property()` extension function. In this sequence, we are using two property mediators and set a Synapse XPath variable and a `get-property()` function to the two mediators respectively to retrieve the properties set to the message context during the execution.

Property Name	New Property
New Property Name	YQL
Value Type	Expression
Value Expression	<p>For the XPath expression, we take a query part in the Yahoo API's endpoint (https://developer.yahoo.net/weather)</p> <pre>concat(' ?q=select%20*%20from%20weather.forecast%20where%20woeid%20eq%2012248447&format=json')</pre> <p>Note that the full URL of the Yahoo endpoint is https://query.yahooapis.com/v1/publictables.org%2Falltableswithkeys and we are extracting the query part (<code>q=</code>) from the endpoint highlighted and provide the <code>z</code> value as the parameter.</p>

Property Scope	<p>Synapse</p> <p>Since this is a mediation level Property keep the Property Scope as Synapse. This is the</p>
-----------------------	--

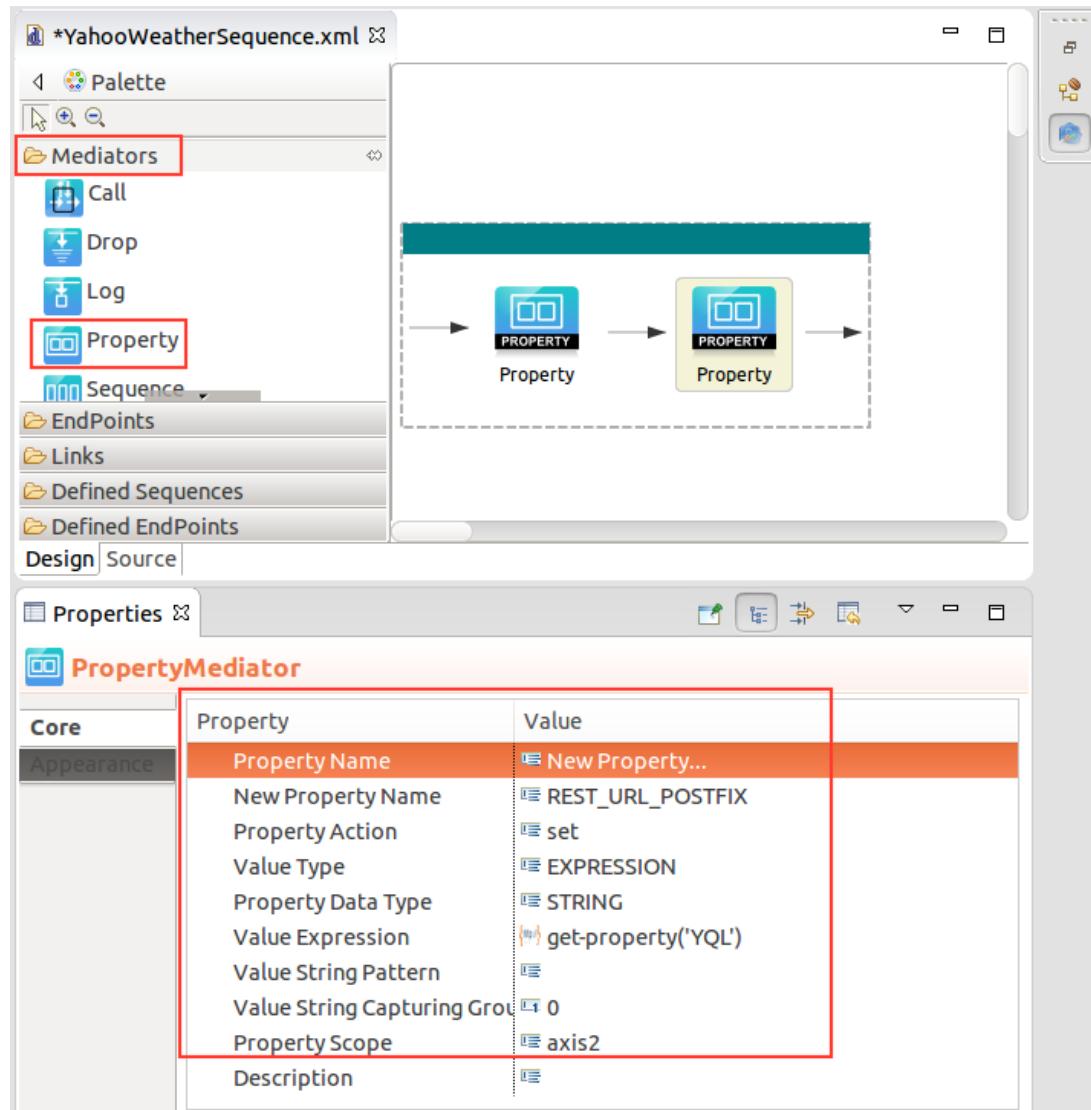
Core																							
Appearance	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #f2f2f2;">Property</th> <th style="background-color: #f2f2f2;">Value</th> </tr> </thead> <tbody> <tr> <td>Property Name</td> <td>New Property...</td> </tr> <tr> <td>New Property Name</td> <td>YQL</td> </tr> <tr> <td>Property Action</td> <td>set</td> </tr> <tr> <td>Value Type</td> <td>EXPRESSION</td> </tr> <tr> <td>Property Data Type</td> <td>STRING</td> </tr> <tr> <td style="background-color: #ffcc00;">Value Expression</td> <td>concat('?q=select%20*%20from%20weather.f')</td> </tr> <tr> <td>Value String Pattern</td> <td></td> </tr> <tr> <td>Value String Capturing Group</td> <td>0</td> </tr> <tr> <td>Property Scope</td> <td>Synapse</td> </tr> <tr> <td>Description</td> <td></td> </tr> </tbody> </table>	Property	Value	Property Name	New Property...	New Property Name	YQL	Property Action	set	Value Type	EXPRESSION	Property Data Type	STRING	Value Expression	concat('?q=select%20*%20from%20weather.f')	Value String Pattern		Value String Capturing Group	0	Property Scope	Synapse	Description	
Property	Value																						
Property Name	New Property...																						
New Property Name	YQL																						
Property Action	set																						
Value Type	EXPRESSION																						
Property Data Type	STRING																						
Value Expression	concat('?q=select%20*%20from%20weather.f')																						
Value String Pattern																							
Value String Capturing Group	0																						
Property Scope	Synapse																						
Description																							

14. Similarly, add another property mediator with the following values. This is an HTTP transport property that appends its value to the address endpoint URL. Once you are done, save the sequence.

Property Name	New Property
New Property Name	REST_URL_POSTFIX
Value Type	Expression
Value Expression	get-property('YQL')

Property Scope Axis2

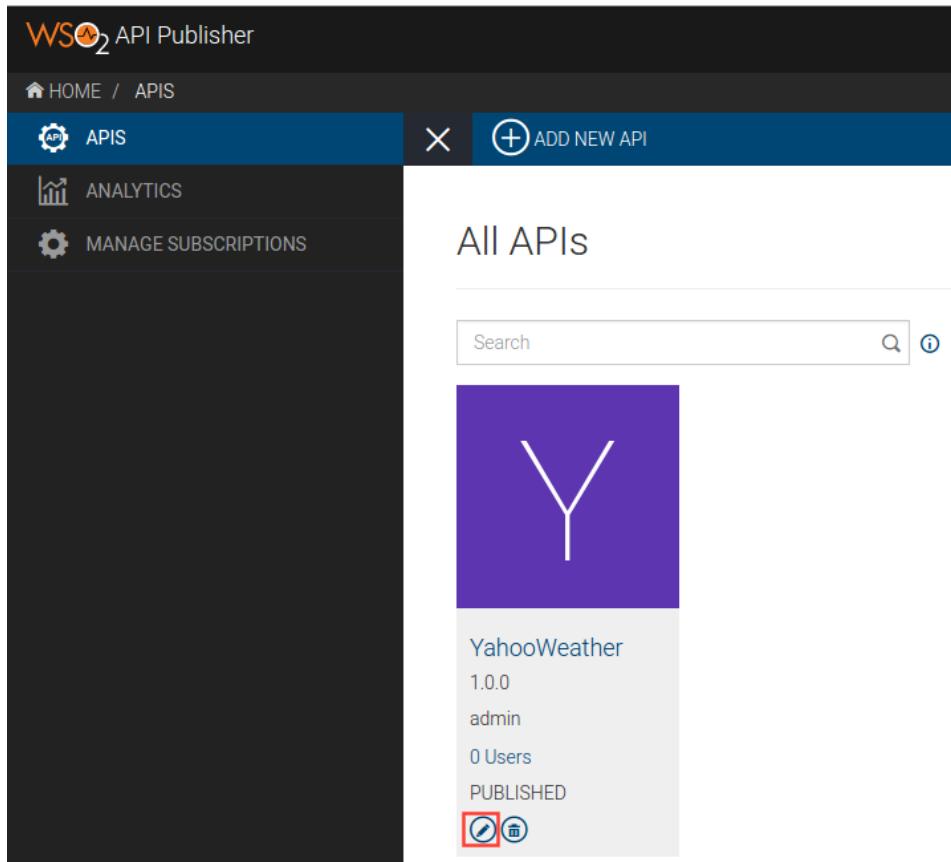
Since this is a transport level Property you need to set the Property Scope as **Axis2**.



15. Navigate to the **File** menu, and click **Save** to save the sequence.
16. Right-click on the sequence and click **Commit File** to push the changes to the Publisher server.

Alternatively, you can create a CAR file including the sequences and can deploy it in API Manager. For more information on deploying sequences in a CAR file refer [Deploying Sequences](#).

17. Sign in to the API Publisher again, select the API that you created earlier, and click the **Edit** link right next to its name to go to the edit wizard.



18. Navigate to the API's **Implement** tab, select the **Enable Message Mediation** check box and select the sequence that you created for the In flow. Next, click **Manage** and **Save & Publish** the API again.

Tip: It might take a few minutes for the sequence to be uploaded into the API Publisher. If it isn't there, please check again later.

YahooWeather: /weather/1.0

The screenshot shows the 'Implement' tab of the WSO2 API Manager interface for the 'YahooWeather' API. The 'Managed API' section includes fields for 'Endpoint Type' (set to 'HTTP/REST Endpoint'), 'Production Endpoint' (set to 'https://query.yahooapis.com/v1/public/yql'), and 'Sandbox Endpoint' (set to 'https://query.yahooapis.com/v1/public/yql'). Below these are 'Load Balanced' and 'Failover' checkboxes. The 'Message Mediation Policies' section has a red box around the 'In Flow' configuration, which is set to 'YahooWeatherSequence'. There are also sections for 'Out Flow' (None) and 'Fault Flow' (None). The 'CORS configuration' section has a checkbox for 'Enable API based CORS Configuration' which is unchecked. At the bottom are 'Save' and 'Next : Manage >' buttons, with 'Next : Manage >' being highlighted with a red border.

When selecting a mediator, make sure that it is a non-blocking mediator as blocking mediators are not supported in API Gateway custom mediations. For more details, see [Adding Mediation Extensions](#).

- Sign in to the API Store, subscribe to the API that you just published, and generate the access tokens in order to invoke the API.

The screenshot shows the API Store page for the 'YahooWeather - 1.0.0' API. It displays the API's logo (a purple square with a white 'Y'), version (1.0.0), author (admin), update date (04/May/2017 17:49:21 PM IST), status (PUBLISHED), and rating (5 stars). To the right is a sidebar with a red border containing 'Applications' (DefaultApplication) and 'Tiers' (Gold) dropdowns, and a 'Subscribe' button.

- Click the **API Console** tab of the API. It opens the integrated API Console using which you can invoke the API.

21. Give the following values for the parameters and invoke the API. You can also give any other value of your choice.

country	usa
zipcode	95004

GET /current/{country}/{zipcode}

Parameters

Parameter	Value	Description	Parameter Type	Data Type
country	usa		path	string
zipcode	95004		path	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

[Try it out!](#)

Note the response that you get as a JSON object from Yahoo.

```
{
  "query": {
    "count": 1,
    "created": "2017-05-04T12:49:03Z",
    "lang": "en-US",
    "results": {
      "channel": {
        "units": {
          "distance": "mi",
          "pressure": "in",
          "speed": "mph",
          "temperature": "F"
        },
        "title": "Yahoo! Weather - Aromas, CA, US",
        "unit": "us"
      }
    }
  }
}
```

```

    "link": "http://us.rd.yahoo.com/dailynews/rss/weather/Country__Country/*https://weather.yahoo.com/country/state/city-12797499/",
        "description": "Yahoo! Weather for Aromas, CA, US",
        "language": "en-us",
        "lastBuildDate": "Thu, 04 May 2017 05:49 AM PDT",
        "ttl": "60",
        "location": {
            "city": "Aromas",
            "country": "United States",
            "region": "CA"
        },
        "wind": {
            "chill": "50",
            "direction": "245",
            "speed": "4"
        },
        "atmosphere": {
            "humidity": "98",
            "pressure": "999.0",
            "rising": "0",
            "visibility": "7.5"
        },
        "astronomy": {
            "sunrise": "6:9 am",
            "sunset": "7:58 pm"
        },
        "image": {
            "title": "Yahoo! Weather",
            "width": "142",
            "height": "18",
            "link": "http://weather.yahoo.com",
            "url": "http://l.yimg.com/a/i/brand/purplelogo//uh/us/news-wea.gif"
        },
        "item": {
            "title": "Conditions for Aromas, CA, US at 05:00 AM PDT",
            "lat": "36.878021",
            "long": "-121.618973",
            "link": "http://us.rd.yahoo.com/dailynews/rss/weather/Country__Country/*https://weather.yahoo.com/country/state/city-12797499/",
            "pubDate": "Thu, 04 May 2017 05:00 AM PDT",
            "condition": {
                "code": "33",
                "date": "Thu, 04 May 2017 05:00 AM PDT",
                "temp": "51",
                "text": "Mostly Clear"
            },
            "forecast": [
                {
                    "code": "30",
                    "date": "04 May 2017",

```

```
        "day": "Thu",
        "high": "74",
        "low": "55",
        "text": "Partly Cloudy"
    },
{
    "code": "28",
    "date": "05 May 2017",
    "day": "Fri",
    "high": "71",
    "low": "53",
    "text": "Mostly Cloudy"
},
{
    "code": "30",
    "date": "06 May 2017",
    "day": "Sat",
    "high": "65",
    "low": "47",
    "text": "Partly Cloudy"
},
{
    "code": "12",
    "date": "07 May 2017",
    "day": "Sun",
    "high": "62",
    "low": "48",
    "text": "Rain"
},
{
    "code": "30",
    "date": "08 May 2017",
    "day": "Mon",
    "high": "69",
    "low": "46",
    "text": "Partly Cloudy"
},
{
    "code": "30",
    "date": "09 May 2017",
    "day": "Tue",
    "high": "69",
    "low": "48",
    "text": "Partly Cloudy"
},
{
    "code": "28",
    "date": "10 May 2017",
    "day": "Wed",
    "high": "70",
    "low": "52",
    "text": "Mostly Cloudy"
},
```

```
{
    "code": "30",
    "date": "11 May 2017",
    "day": "Thu",
    "high": "72",
    "low": "52",
    "text": "Partly Cloudy"
},
{
    "code": "30",
    "date": "12 May 2017",
    "day": "Fri",
    "high": "72",
    "low": "48",
    "text": "Partly Cloudy"
},
{
    "code": "34",
    "date": "13 May 2017",
    "day": "Sat",
    "high": "71",
    "low": "46",
    "text": "Mostly Sunny"
},
],
"description": "<![CDATA[<img
src=\"http://l.yimg.com/a/i/us/we/52/33.gif\"/>\n<BR />\n<b>Current
Conditions:</b>\n<BR />Mostly Clear\n<BR />\n<BR
/><b>Forecast:</b>\n<BR /> Thu - Partly Cloudy. High: 74Low: 55\n<BR
/> Fri - Mostly Cloudy. High: 71Low: 53\n<BR /> Sat - Partly Cloudy.
High: 65Low: 47\n<BR /> Sun - Rain. High: 62Low: 48\n<BR /> Mon -
Partly Cloudy. High: 69Low: 46\n<BR />\n<BR />\n<a
href=\"http://us.rd.yahoo.com/dailynews/rss/weather/Country_Country/
*https://weather.yahoo.com/country/state/city-12797499/\">Full
Forecast at Yahoo! Weather</a>\n<BR />\n<BR />\n(provided by <a
href=\"http://www.weather.com\">The Weather Channel</a>)\n<BR
/>\n]]>",
"guid": {
    "isPermaLink": "false"
}
}
}
```

```
    }
}
```

In this tutorial, you created a sequence to change the default mediation flow of API requests, deployed it in the API Gateway and invoked an API using the custom mediation flow.

Please note that following mediators are not usable within custom sequences since they are not supported by API Gateway custom medications.

- Call mediator in non-blocking mode
- Send mediator

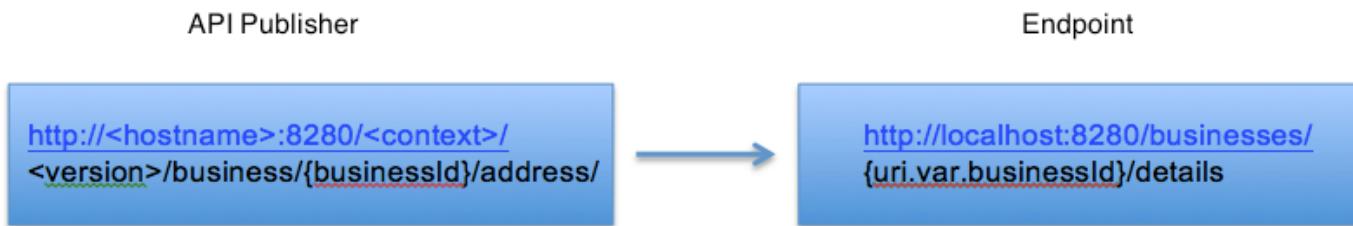
Map the Parameters of your Backend URLs with the API Publisher URLs

This tutorial uses the WSO2 API Manager Tooling Plug-in.

This tutorial explains how to map your backend URLs to the pattern that you want in the API Publisher. Note the following:

1. The URL pattern of the APIs in the Publisher is `http://<hostname>:8280/<context>/<version>/<API resource>`.
2. You can define variables as part of the URI template of your API's resources. For example, in the URI template `/business/{businessId}/address/`, `businessId` is a variable.
3. The variables in the resources are read during mediation runtime using property values with the "uri.var." prefix. For example, this HTTP endpoint gets the businessId that you specify in the resource `http://localhost:8280/businesses/{uri.var.businessId}/details`.
4. The URI template of the API's resource is automatically appended to the end of the HTTP endpoint at runtime. You can use the following mediator setting to remove the URL postfix from the backend endpoint: `<property name="REST_URL_POSTFIX" scope="axis2" action="remove" />`.

We do the following mapping in this tutorial:



Before you begin, note that a mock backend implementation is set up in this tutorial for the purpose of demonstrating the API invocation. If you have a local API Manager setup, save [this file](#) in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` folder to set up the mock backend.

1. Log in to the API Publisher, design a new API with the following information, click **Add** and then click **Next: Implement >**.

Field		Sample value
Name		TestAPI
Context		/test

Version		1.0.0
Visibility		Public
Resources	URL pattern	/business/{businessId}/address/
	Request types	GET

1 Design 2 Implement 3 Manage

General Details

Name: *

Context: *

Version: *

Visibility:

Description:

Tags:
Type a Tag and Enter

Thumbnail Image

Select image

Dimensions (max): 100 x 100 pixels

API Definition

URL Pattern Import Edit Source

GET POST PUT DELETE PATCH HEAD [more](#)

Add

Save Next: Implement >

2. The **Implement** tab opens. Give the information in the table below.

Field	Sample value
Endpoint type	HTTP/REST endpoint
Production endpoint	http://localhost:8280/businesses/{uri.var.businessId}/details
Sandbox endpoint	http://localhost:8280/businesses/{uri.var.businessId}/details

TestAPI: /test/1.0.0

1 Design 2 Implement 3 Manage

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type :* ?

Load Balanced Failover

Production Endpoint :* ?

Sandbox Endpoint :* ?

Show More Options

Message Mediation Policies

Enable Message Mediation Check to select a message mediation policy to be executed in the message flow

CORS configuration

Enable API based CORS Configuration

Actions

3. Click **Next: Manage >** to go to the Manage tab, select the Gold tier and publish the API.

1 Design 2 Implement 3 Manage

Configurations

Make this the Default Version: No default version defined for the current API

Transports: * HTTPS HTTP

Response Caching: Disabled

Throttling Settings

Maximum Backend Throughput: Unlimited Specify

Subscription Tiers: * **Unlimited**: Allows unlimited requests
 Gold: Allows 5000 requests per minute
 Silver: Allows 2000 requests per minute
 Bronze: Allows 1000 requests per minute

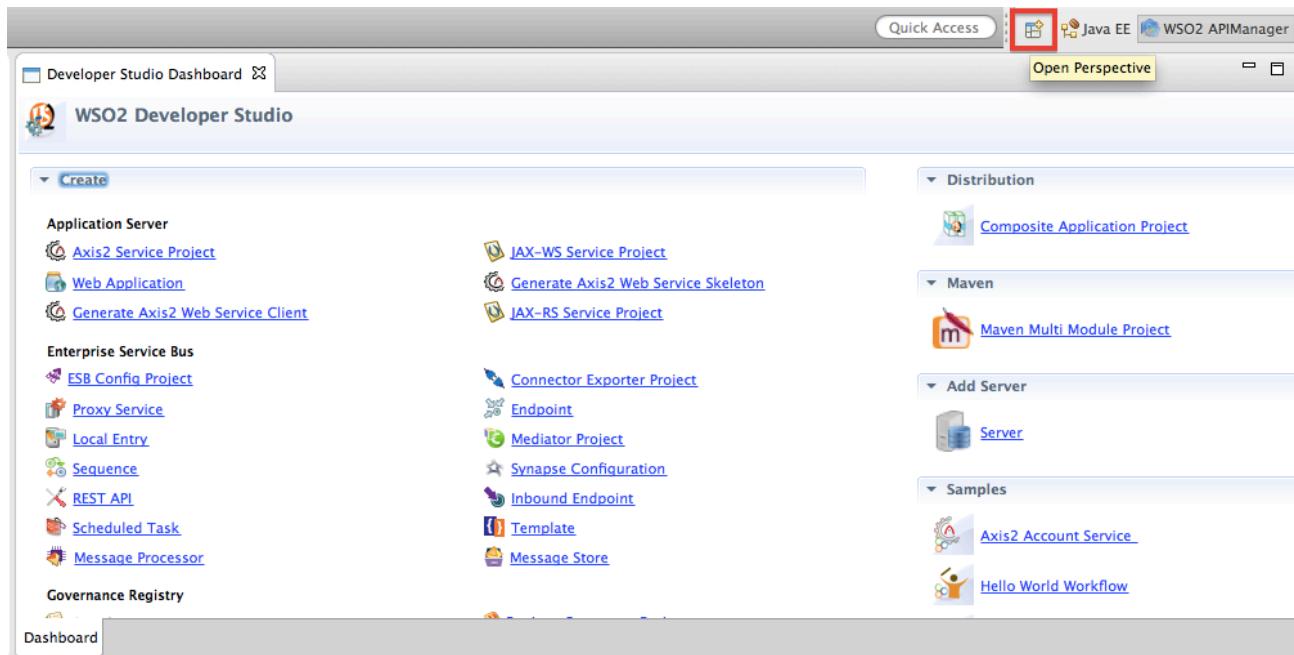
Advanced Throttling Policies: Apply to API Apply per Resource
[Select Policy per Resource](#)

As the API's resource is appended to its endpoint by Synapse at runtime, let's write a custom sequence to remove this appended resource.

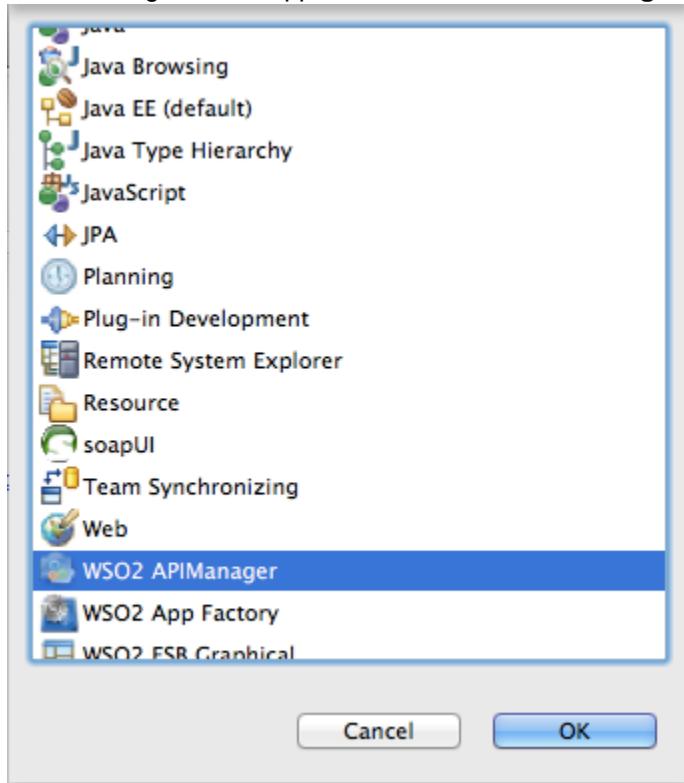
4. Copy the the following to a text editor and save the file in XML format (e.g., TestSequence.xml).

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="TestSequence">
    <property name="REST_URL_POSTFIX" scope="axis2" action="remove"/>
</sequence>
```

5. Download and install the [WSO2 API Manager Tooling Plug-in](#) if you have not done so already. Open Eclipse by double clicking the Eclipse.app file inside the downloaded folder.
6. Click **Window > Open Perspective > Other** to open the Eclipse perspective selection window. Alternatively, click the **Open Perspective** icon shown below at the top right corner.



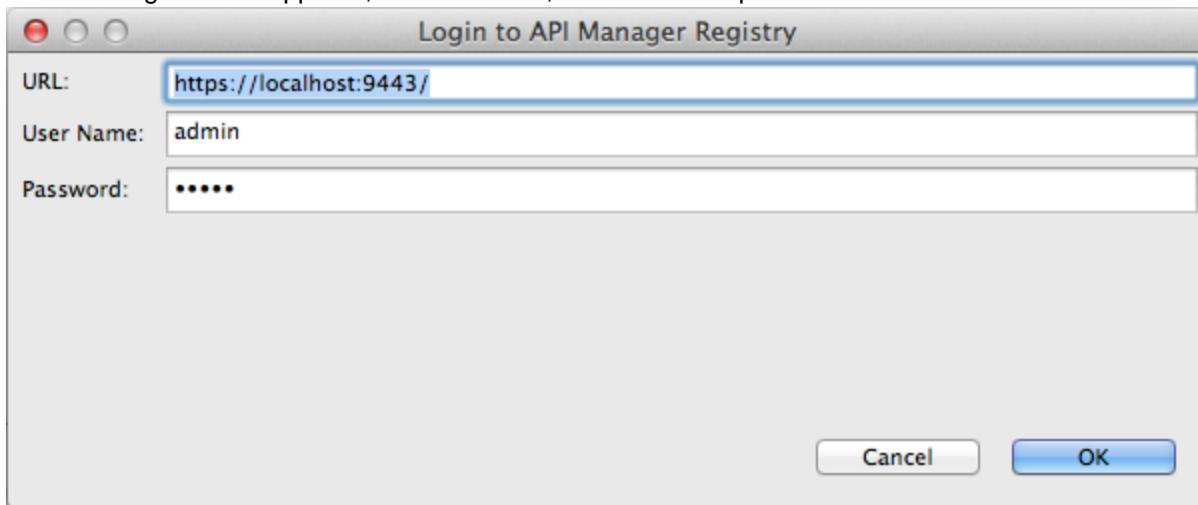
7. On the dialog box that appears, click **WSO2 APIManager** and click **OK**.



8. On the APIM perspective, click the **Login** icon as shown below.

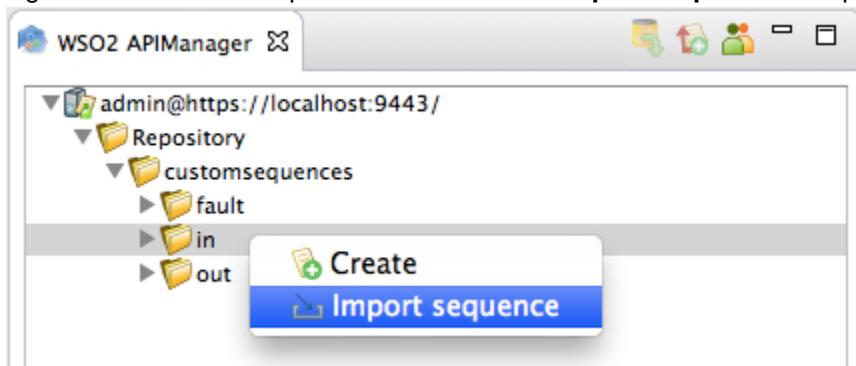


9. On the dialog box that appears, enter the URL, username and password of the Publisher server.



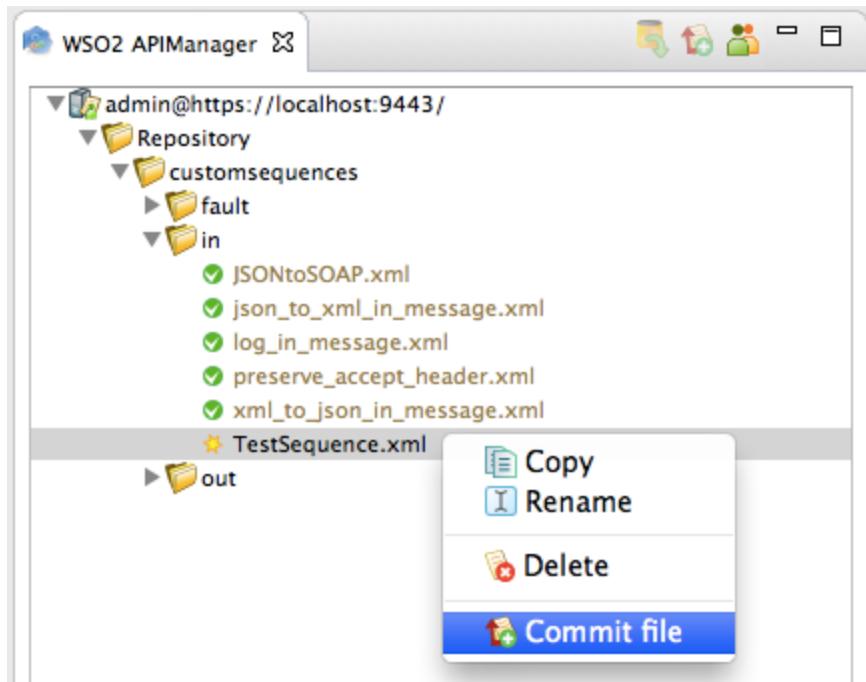
10. On the tree view that appears, expand the folder structure of the existing API.

11. Right-click on the **in** sequence folder and click **Import Sequence** to import the sequence you create above.



12. Browse to the `TestSequence.xml` file you created in step 4.

13. Your sequence now appears on the APIM perspective. Right-click on the imported sequence and click **Commit File** to push the changes to the Publisher server.



14. Log back into the API Publisher, click **Edit** and go to the **Implement** tab. Select the **Enable Message Mediation** check box and engage the **In** sequence that you created earlier.

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type: * Load Balanced Failover

Production Endpoint: *

Sandbox Endpoint: *

Show More Options

Message Mediation Policies

Enable Message Mediation	<input checked="" type="checkbox"/> Check to select a message mediation policy to be executed in the message flow
In Flow	<input type="text" value="TestSequence"/> <input type="button" value="Upload In Flow"/>
Out Flow	<input type="text" value="None"/> <input type="button" value="Upload Out Flow"/>
Fault Flow	<input type="text" value="None"/> <input type="button" value="Upload Fault Flow"/>

CORS configuration

Enable API based CORS Configuration

Save **Next : Manage >**

TestSequence.xml removes the URL postfix from the backend endpoint, since the URI template of the API's resource is automatically appended to the end of the URL at runtime. Therefore the **request** URL is modified by adding this sequence to the **In flow**.

15. Save and Publish the API.

You have created an API. Let's subscribe to the API and invoke it.

16. Log in to the API Store and subscribe to the API.

TestAPI - 1.0.0

Version: 1.0.0
By: admin
Updated: 30/Jul/2016 11:59:27 AM IST
Status: PUBLISHED
Rating: ★★★★★

Applications: DefaultApplication
Tiers: Gold

Subscribe

Overview API Console Documentation Forum

17. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.
18. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. If you have already generated a token before, click **Re-generate** to renew the access token.

DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

Show Keys

Consumer Key
.....

Consumer Secret
.....

Grant Types
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can select one or more grant types.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code

Callback URL
.....

Update

Generating Access Tokens
The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

Generate a Test Access Token

Access Token
.....

Above token has a validity period of **3600** seconds. And the token has (**am_application_scope default**) scopes.

Scopes
No Scopes Found..

Validity period
3600 Seconds.

Re-generate

19. Click the **API Console** tab of your API.

TestAPI - 1.0.0

20. Note that the `businessId` is added in the UI as a parameter. Give a `businessId` and click **Try it out!** to invoke the API.

21. Note the response that you get. According to the mock backend used in this tutorial, you get the response Received Request.

In this tutorial, you mapped the URL pattern of the APIs in the Publisher with the endpoint URL pattern of a sample backend.

Convert a JSON Message to SOAP and SOAP to JSON

This tutorial uses the WSO2 API Manager Tooling Plug-in and the **PhoneVerification** API created in [Create and Publish an API](#).

The API Gateway has a default mediation flow for the API invocation requests that it receives. You can extend this default mediation flow to do additional custom mediation for the messages in the API Gateway. An extension is provided as a synapse mediation sequence. You can design sequences using a tool such as the [WSO2 API](#)

Manager Tooling Plug-in and then store the sequence in the Gateway's registry.

Let's see how to convert message types using custom sequences. In this tutorial, we convert a JSON payload to SOAP before sending it to a SOAP backend. Then we receive the response in SOAP and convert it back to JSON.

1. Log in to the API Publisher and click the **PhoneVerification** API.
2. Click the **Edit** icon to go to its edit mode.

All APIs

The screenshot shows the 'All APIs' page with two API cards displayed:

- PizzaShackAPI**: Version 1.0.0, created by admin, 1 User, PUBLISHED. It has a 'Edit' button highlighted with a red box.
- PhoneVerificationAPI**: Version 1.0.0, created by apicreator, 0 Users, CREATED. It has an 'Edit' button highlighted with a yellow box.

3. Create the following resource and add it to the API.

Tip: The resource you create here invokes the [SOAP 1.2 Web service of the backend](#). Therefore, the recommended method is HTTP POST. As you do not include the payload in a query string, avoid giving any specific name in the URL pattern, which will be amended to the actual backend URL.

Field		Sample value
Resources	URL pattern	/*
	Request types	POST

API Definition

URL Pattern: /phoneverify/1.0.0 /*

Import | Edit Source

GET POST PUT DELETE PATCH HEAD more

Add

4. After the resource is added, expand it and edit the parameter as follows. This parameter is used to pass the payload to the backend.

Parameter name	Description	Parameter Type	Data Type	Required
Payload	Pass the phone number and license key	body	string	True

POST /* +Summary

Description : + Add Implementation Notes

Produces : application/json Consumes : application/json

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
Payload	Pass the phone number and license key	body	string	True	

Parameter Name: + Add Parameter

Next, let's write a sequence to convert the JSON payload to a SOAP request. We do this because the backend accepts SOAP requests.

5. Navigate to the **Implement** page and change the endpoint of the API to <http://ws.cdyne.com/phoneverify/phoneverify.asmx?WSDL>. Once the edits are done, click **Save**.

1 Design 2 Implement 3 Manage

Managed API
Provide the production and sandbox endpoints of the API to be managed.

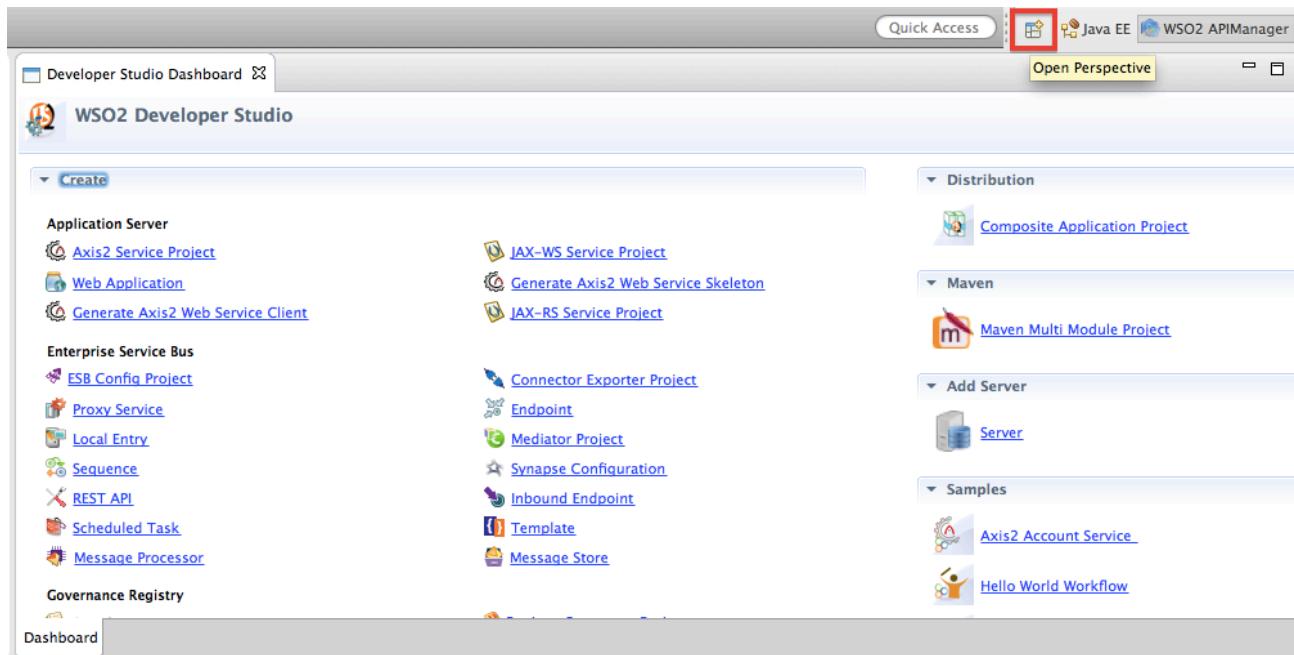
Endpoint Type : * ? Load Balanced Failover

Production Endpoint : * ?

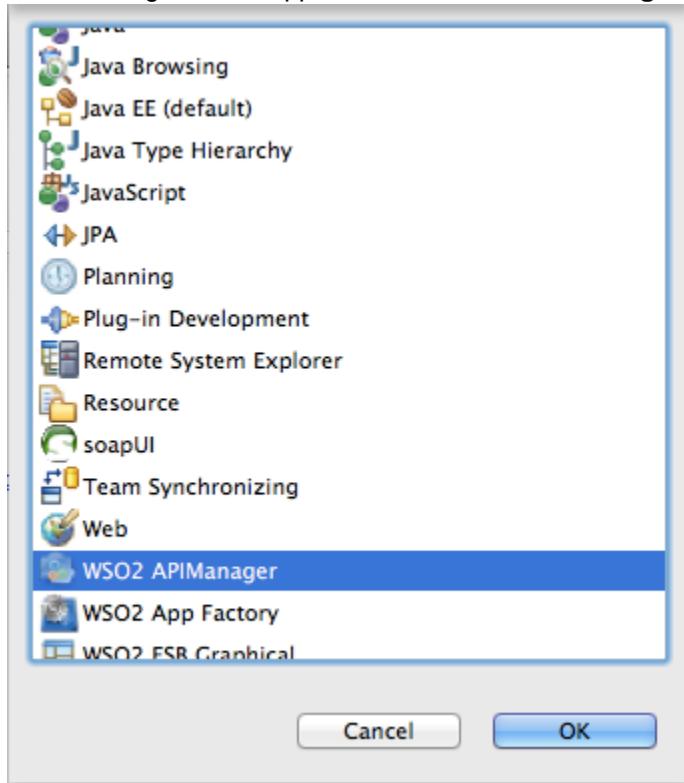
Sandbox Endpoint : * ?

Show More Options

6. Download and install the [WSO2 API Manager Tooling Plug-in](#) if you have not done so already. Open Eclipse by double clicking the `Eclipse.app` file inside the downloaded folder.
7. Click **Window > Open Perspective > Other** to open the Eclipse perspective selection window. Alternatively, click the **Open Perspective** icon shown below at the top right corner.



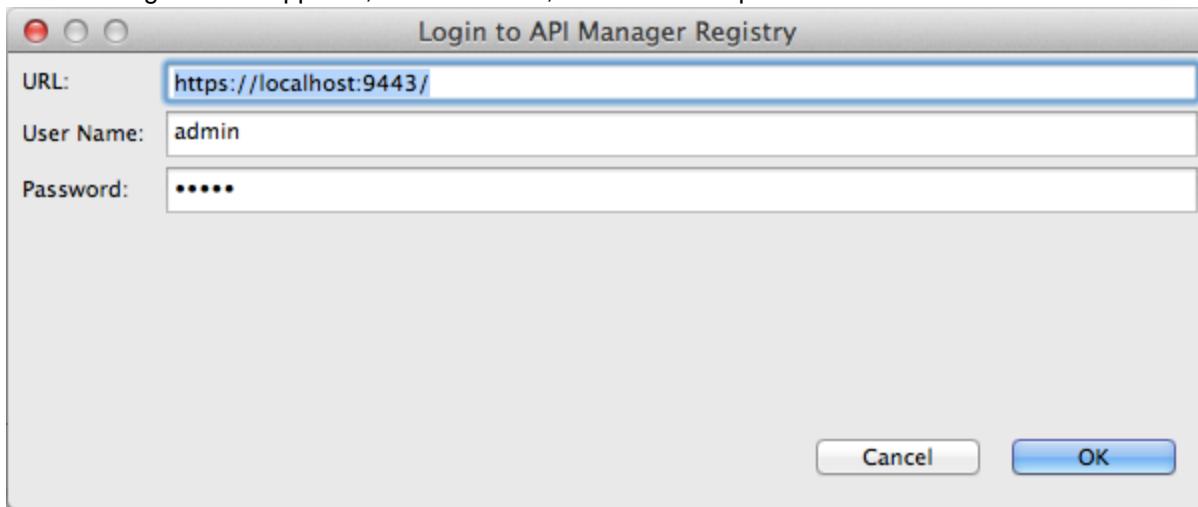
8. On the dialog box that appears, click **WSO2 APIManager** and click **OK**.



9. On the APIM perspective, click the **Login** icon as shown below.

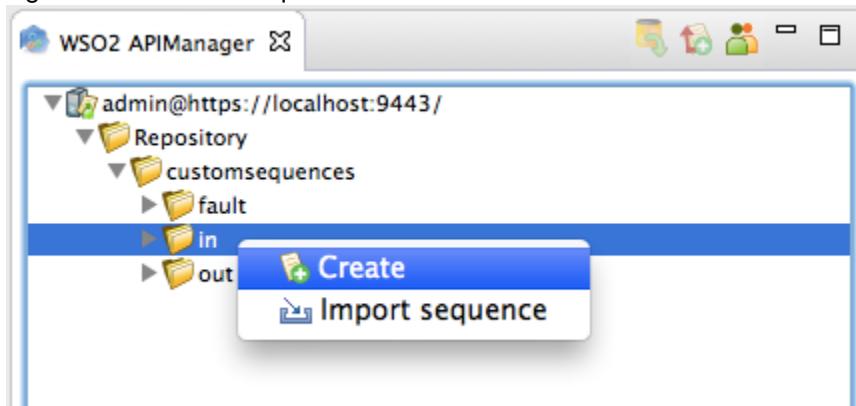


10. On the dialog box that appears, enter the URL, username and password of the Publisher server.

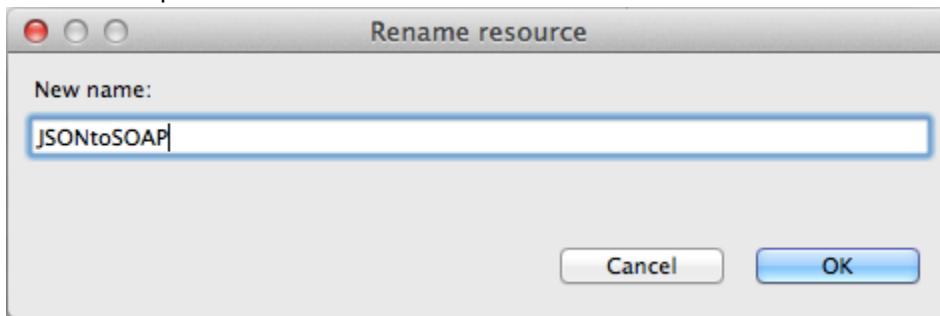


11. On the tree view that appears, expand the folder structure of the existing API.

12. Right-click on the **In** sequence folder and click **Create** to create a new **In** sequence.



13. Name the sequence **JSONtoSOAP**.

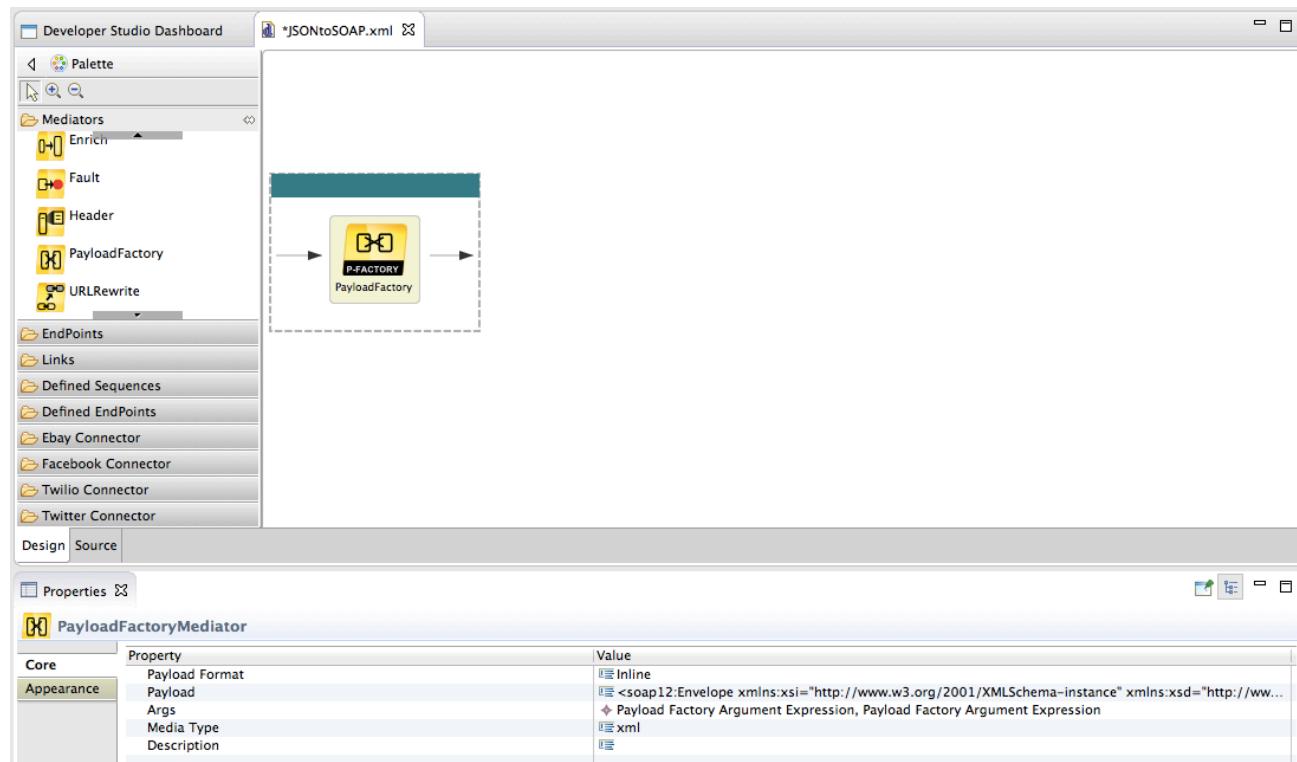


14. Your sequence now appears on the APIM perspective. From under the **Mediators** section, drag and drop a **PayloadFactory** mediator to your sequence and give the following values to the mediator.

Tip: The **PayloadFactory** mediator transforms the content of your message. The `<args>` elements define arguments that retrieve values at runtime by evaluating the provided expression against the SOAP body. You can configure the format of the request/response and map it to the arguments.

For example, in the following configuration, the values for the format parameters **PhoneNumber** and **LicenseKey** will be assigned with values that are taken from the `<args>` elements (arguments,) in that particular order.

For details on how you got this configuration, see [PayloadFactory Mediator](#) in the WSO2 ESB documentation.



Payload

```

<soap12:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <CheckPhoneNumber
      xmlns="http://ws.cdyne.com/PhoneVerify/query">
      <PhoneNumber>$1</PhoneNumber>
      <LicenseKey>$2</LicenseKey>
    </CheckPhoneNumber>
  </soap12:Body>
</soap12:Envelope>

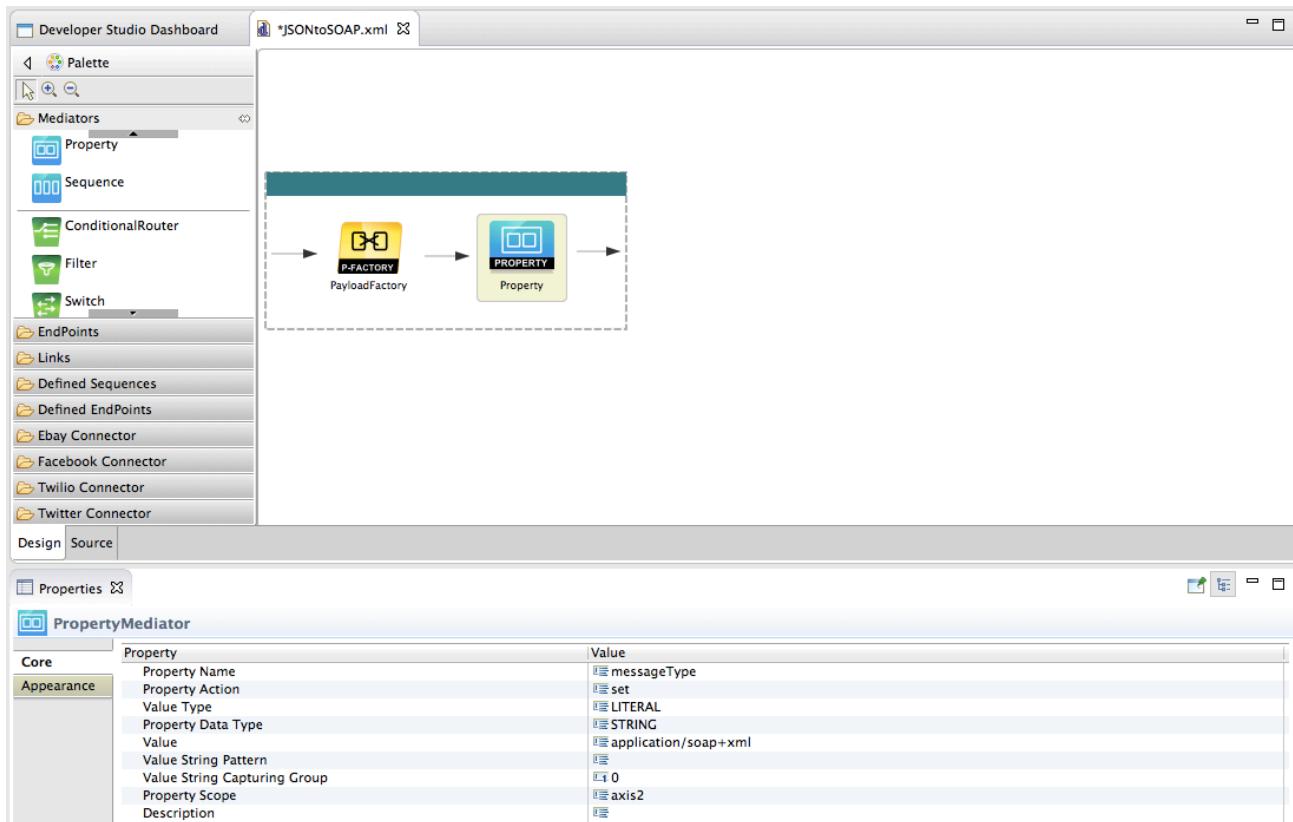
```

Args

Give the arguments as a json expression as follows:

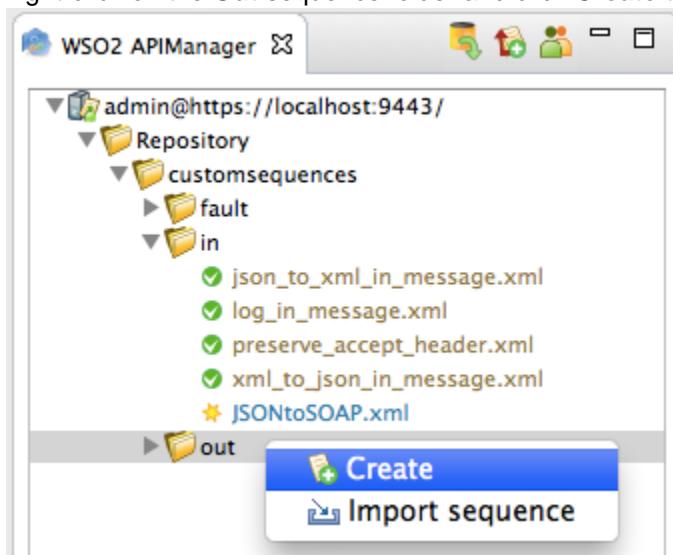
Type	Value	Evaluat
expression	//request/PhoneNumber	xml
expression	//request/LicenseKey	xml

15. Similarly, add a **Property** mediator to the same sequence and give the following values to the property mediator. This mediator changes the payload type of the outgoing message to soap+xml. More information about the Property mediator can be found [here](#).

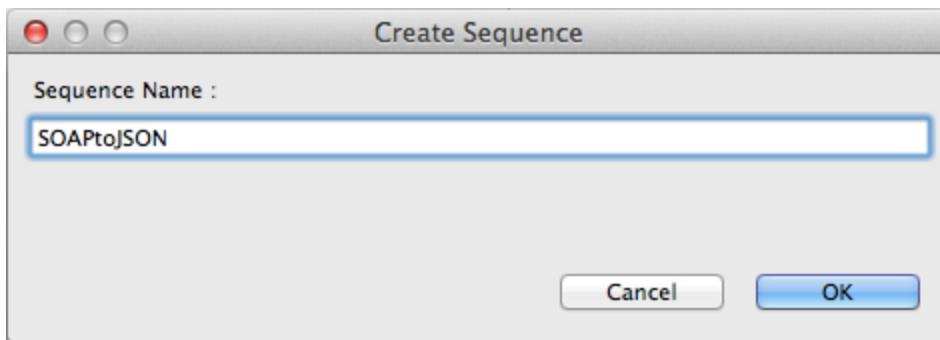


Property Name	messageType
Value Type	Literal
Value	application/soap+xml
Property Scope	axis2

16. Save the sequence, which is in XML format (e.g., `JSONtoSOAP.xml`). This will be the **In** sequence for your API. Next, create an **Out** sequence.
17. Right-click on the **Out** sequence folder and click **Create** to create a new **Out** sequence.



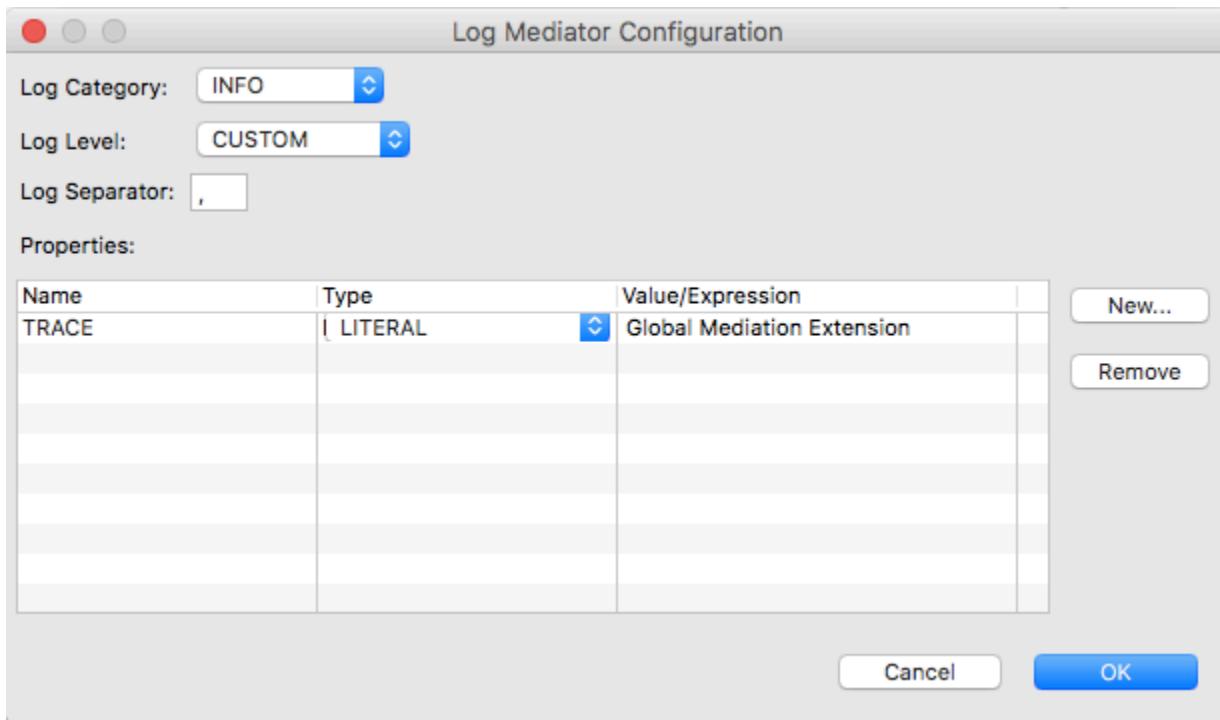
18. Name the sequence `SOAPtoJSON`.



19. Add a **Log** mediator to the sequence and give the following values. Note that the property value provided is a string literal.

The **Log mediator** is used to log mediated messages. Having a custom log level allows to log only the properties added to the Log mediator configuration. More information can be found [here](#).

Log Category	INFO
Log Level	CUSTOM
Log Separator	,
Properties	Name: TRACE Type: LITERAL Value/Expression: Global Mediation Extension



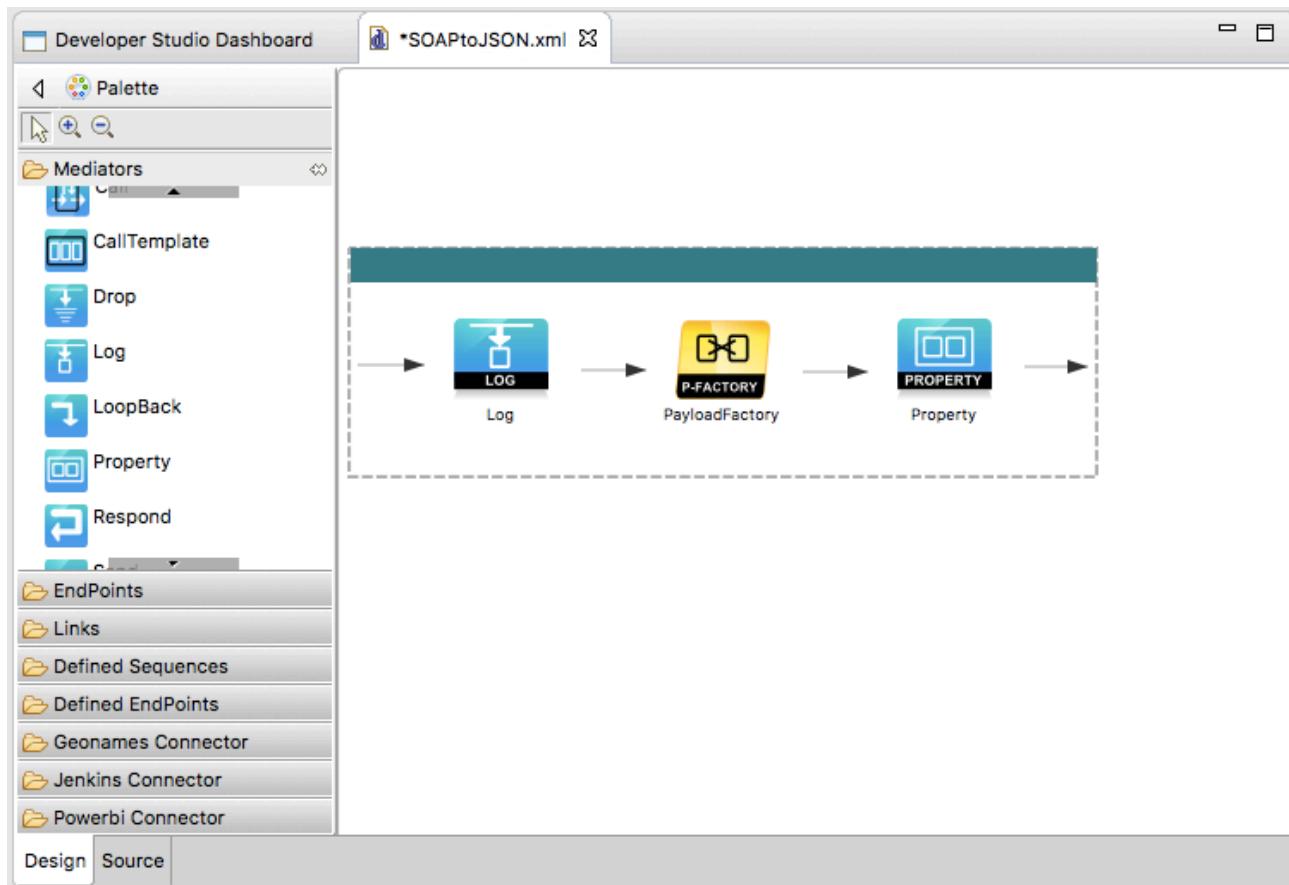
20. Similarly, add a **PayloadFactory** mediator with the following values. This mediator in the out sequence is used to transform the SOAP message content returned from the backend into JSON.

Payload Format	Inline
----------------	--------

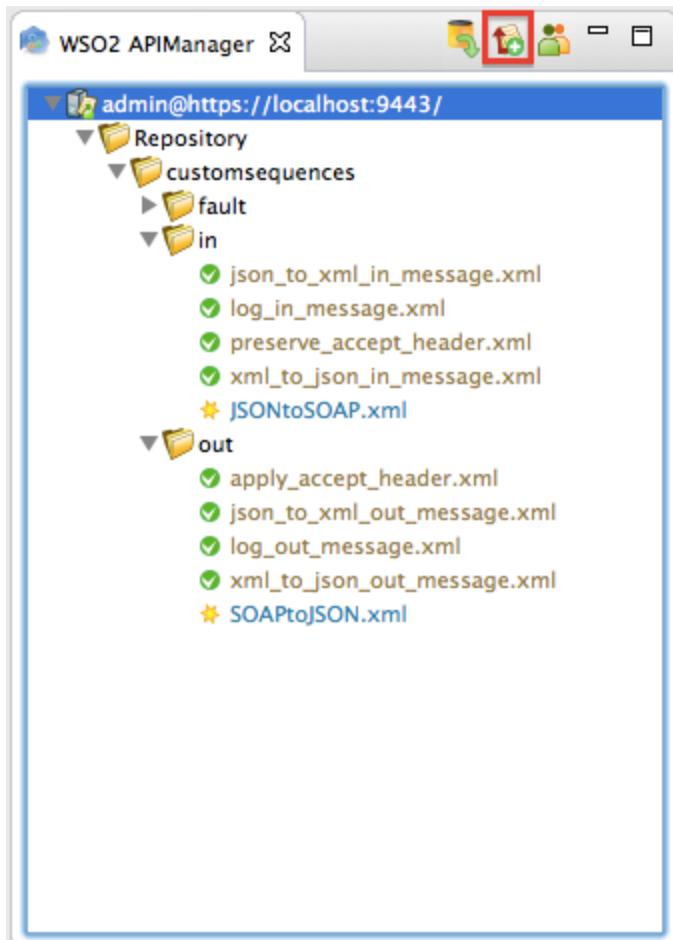
Payload	<CheckPhoneNumber xmlns="http://ws.cdyne.com/PhoneVerify/query"><PhoneNumber>\$1</PhoneNumber><LicenseKey>\$2</LicenseKey></CheckPhoneNumber>
Args	<p>Type: Expression Value: //request/PhoneNumber Evaluator: xml</p> <p>Type: Expression Value: //request/LicenseKey Evaluator: xml</p>
Media Type	xml

21. Finally, add a **Property** mediator with the following values. This mediator changes the payload type of the incoming message to json.

Property Name	messageType
Property Action	set
Value Type	Literal
Property Data Type	String
Value	application/json
Value String Capturing Group	0
Property Scope	axis2



22. Save the sequence, which is in XML format (e.g., `SOAPtoJSON.xml`). This will be the **Out** sequence for your API.
23. Click the **Push all changes to the server** icon shown below to commit your changes to the Publisher server.



24. Log back in to the API Publisher, click the **Edit** link associated with the API and navigate to the **Implement** tab.
b. Select the **Enable Message Mediation** check box and engage the **In** and **Out** sequences that you created earlier.

Message Mediation Policies

<input checked="" type="checkbox"/> Enable Message Mediation	<input checked="" type="checkbox"/> Check to select a message mediation policy to be executed in the message flow
In Flow	<input type="text" value="JSONtoSOAP"/> <input type="button" value="Upload In Flow"/>
Out Flow	<input type="text" value="SOAPtoJSON"/> <input type="button" value="Upload Out Flow"/>
Fault Flow	<input type="text" value="None"/> <input type="button" value="Upload Fault Flow"/>

JSONtoSOAP in sequence will serve the purpose of transforming the JSON payload to SOAP before sending it to the SOAP backend. **SOAPtoJSON out sequence** will transform the SOAP message returned from the backend to JSON.

25. **Save** the API.
You have created an API, a resource to access the SOAP backend and engaged sequences to the request and response paths to convert the message format from JSON to SOAP and back to JSON. Let's subscribe to the API and invoke it.
26. Log in to the API Store and subscribe to the API and create an access token if you have not done so already.

PhoneVerification - 1.0.0

Version: 1.0.0
By: admin
Updated: 22/Jul/2016 17:12:01 PM IST
Status: PUBLISHED
Rating: ★★★★★

Applications
DefaultApplication

Tiers
Bronze

Subscribe

27. Go to the API Console tab and expand the POST method.
28. Give the payload in the body parameter in JSON format and click **Try it out**. Here's a sample JSON payload: {"request":{"PhoneNumber":"18006785432","LicenseKey":"0"}}

Parameter	Value	Description	Parameter Type	Data Type
Payload	{"request": {"PhoneNumber": "18006785432", "LicenseKey": "0"}}	Pass the phone number and license key	body	Model Model Schema {}

Parameter content type:
application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

Try it out!

29. Note that you get a JSON response to the JSON request whereas the backend accepts SOAP messages. The request and response are converted by the sequences that you engaged.

Response Body

```
{
    "CheckPhoneNumberResponse": {
        "CheckPhoneNumberResult": {
            "Company": "Toll Free",
            "Valid": true,
            "Use": "Assigned to a code holder for normal use.",
            "State": "TF",
            "RC": null,
            "OCN": null,
            "OriginalNumber": 18006785432,
            "CleanNumber": 8006785432,
            "SwitchName": null,
            "SwitchType": null,
            "Country": "United States",
            "CLLI": null,
            "PrefixType": "Landline",
            "LATA": null,
            "sms": "Landline",
            "Email": null,
            "AssignDate": null,
            "TelecomCity": null,
        }
    }
}
```

In this tutorial, you converted a message from JSON to SOAP and back to JSON using **In** and **Out** sequences.

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](#)

The examples here use the PhoneVerification API, which is created in section [Create and Publish an API](#).

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 (e.g., DefaultApplication), the **Bronze tier** and subscribe to the API.

PhoneVerification - 1.0.0

Version: 1.0.0
By: admin
Updated: 22/Jul/2016 17:12:01 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications
DefaultApplication

Tiers
Bronze

Subscribe

- Click the **Applications** menu, open the default application using which you subscribed to the API, and generate a production key.

DefaultApplication

Details Production Keys Sandbox Keys Subscriptions

No Keys Found
No keys are generated for this type in this application.

Grant Types

Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM	<input type="checkbox"/> Implicit	<input checked="" type="checkbox"/> Refresh Token
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code	<input checked="" type="checkbox"/> Password	

Callback URL

Access token validity period

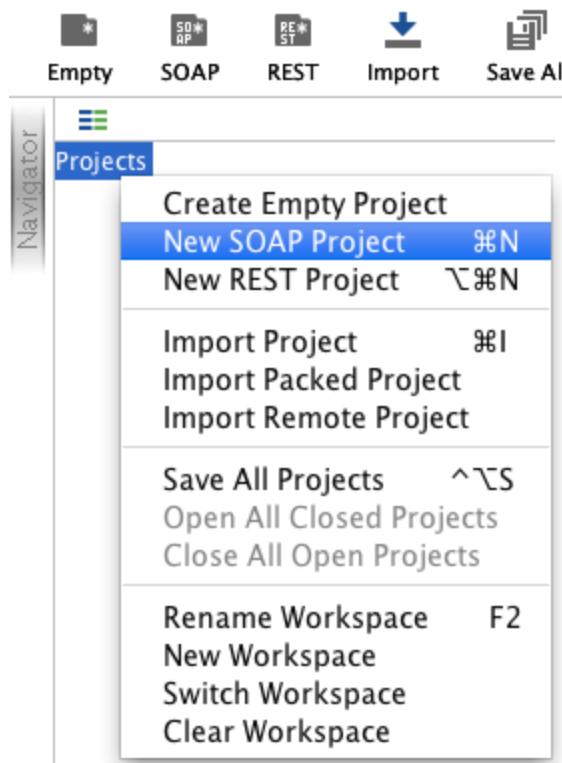
3600	Seconds.
------	----------

Generate keys

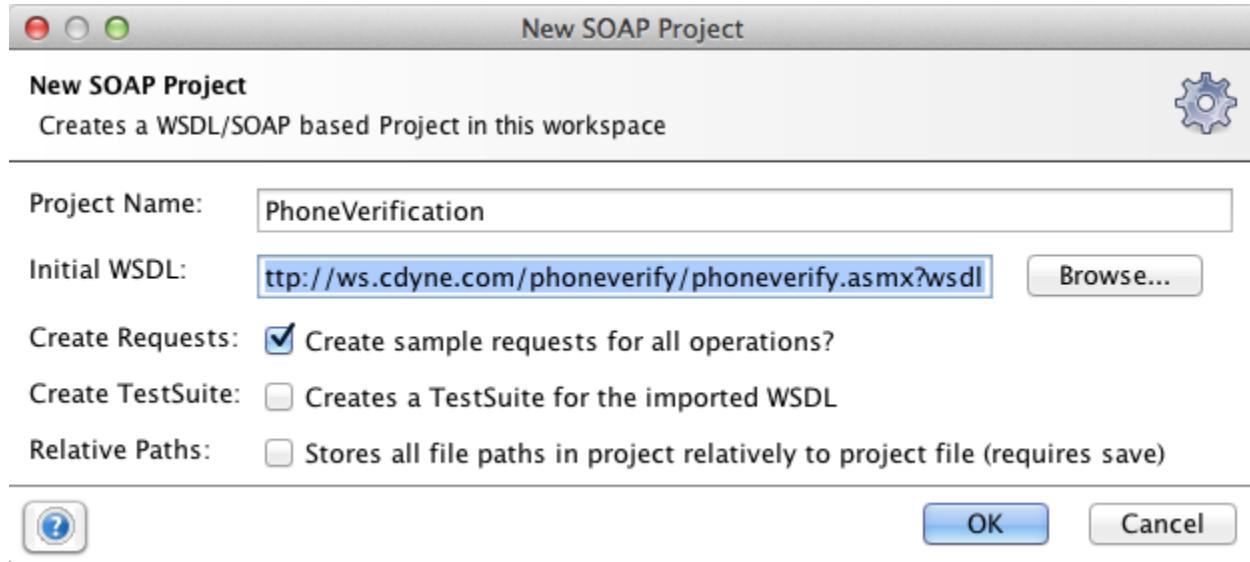
- Copy the access token to the clipboard as you need it later to invoke the API.

Make sure the flash plugin of your web browser is updated in order to get the **copy button** in the Store UI working.

- Download the SOAP UI installation that suits your operating system from <https://www.soapui.org/downloads/soapui.html> and open its console.
- 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, click the **Header** tab and add an authorization header to your request by clicking the add icon.

The screenshot shows the WSO2 API Manager interface with the following details:

- Toolbar:** Empty, SOAP, REST, Import, Save All, Forum, Trial, Preferences, Proxy.
- Navigator:** Projects section showing PhoneVerification, PhoneVerifySoap, CheckPhoneNumber, Request 1 (highlighted with a red box), PhoneVerifySoap12, CheckPhoneNumber, and CheckPhoneNumbers.
- Request Editor:**
 - Request URL: http://ws.cdyne.com/phoneverify/phoneverify.asmx
 - Method: SOAP
 - Message Type: Request 1
 - XML Tab: Shows the SOAP message structure:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
<soapenv:Header/>
<soapenv:Body>
<quer:CheckPhoneNumber>
<!--Optional:-->
<quer:PhoneNumber>?</quer:PhoneNumber>
<!--Optional:-->
<quer:LicenseKey>?</quer:LicenseKey>
</quer:CheckPhoneNumber>
</soapenv:Body>
</soapenv:Envelope>
```
 - Raw Tab: Shows the raw XML message.
- Modal Dialog:** Add HTTP Header. Title: Add HTTP Header. Content: Specify name of header to add: Authorization. Buttons: Cancel, OK.
- Request Properties:**

Property	Value
Name	Request 1
Description	

 - Headers Tab (highlighted with a red box): Contains a table with columns Header and Value. One row is present: Header Auth, Value (empty).
 - Other tabs: Auth, Attachments (0), WS-A, WS-RM, JMS Headers, JMS Properties (0).

9. Give the value of the Authorization header as 'Bearer <the access token you copied in step 4>'.

The screenshot shows the WSO2 API Manager interface. At the top, there are icons for Empty, SOAP, REST, Import, Save All, Forum, Trial, Preferences, and Proxy. The main area is divided into several sections:

- Navigator:** Shows the project structure under "PhoneVerification". A "Request 1" node under "PhoneVerifySoap/CheckPhoneNumber" is selected.
- Request Editor:** Titled "Request 1", it displays the SOAP message structure. The XML code is:


```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
<soapenv:Header/>
<soapenv:Body>
  <quer:CheckPhoneNumber>
    <!--Optional:-->
    <quer:PhoneNumber>?</quer:PhoneNumber>
    <!--Optional:-->
    <quer:LicenseKey>?</quer:LicenseKey>
  </quer:CheckPhoneNumber>
</soapenv:Body>
</soapenv:Envelope>

```
- Headers:** A table showing the "Authorization" header with the value "Bearer 9f7315c763f7fd20d4703c7a69e57c5d".
- Request Properties:** A table with one row: "Name" (Value: Request 1).
- Buttons:** Auth, Headers (1), Attachments (0), WS-A, WS-RM, JMS Headers, JMS Property (0).

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.

The screenshot shows the WSO2 API Manager interface. On the left, the 'Projects' sidebar lists several projects, including 'PhoneVerification' which contains 'PhoneVerifySoap' and 'CheckPhoneNumber' (selected). Below this is a 'Request Properties' panel with a table:

Property	Value
Name	Request 1
Description	

The main workspace displays a 'Request 1' configuration. The 'Raw' tab shows the XML request body:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <quer:CheckPhoneNumber>
      <!--Optional:-->
      <quer:PhoneNumber>123456</quer:PhoneNumber>
      <!--Optional:-->
      <quer:LicenseKey>0</quer:LicenseKey>
    </quer:CheckPhoneNumber>
  </soapenv:Body>
</soapenv:Envelope>

```

The 'Headers' tab shows an 'Authorization' header with the value 'Bearer dff1c4917b5f4a802d9ad700442453d8'. The 'Raw' tab also lists other tabs: Auth, Headers (1), Attachments (0), WS-A, WS-RM, JMS Headers, and JMS Property (0).

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.

The screenshot shows the WSO2 API Manager interface after the request has been processed. The 'Raw' tab shows the XML response body:

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Header/>
  <soap:Body>
    <CheckPhoneNumberResponse xmlns="http://ws.cdyne.com/PhoneVerify/query">
      <CheckPhoneNumberResult>
        <Valid>false</Valid>
        <OriginalNumber>123456</OriginalNumber>
        <CleanNumber>23456</CleanNumber>
        <Wireless>false</Wireless>
      </CheckPhoneNumberResult>
    </CheckPhoneNumberResponse>
  </soap:Body>
</soap:Envelope>

```

The 'Raw' tab also lists other tabs: XML, Raw, XML, Raw. The 'Headers' tab shows an 'Authorization' header with the value 'Bearer 9f7315c763f7fd2...'. The 'Raw' tab also lists other tabs: Auth, Headers (1), Attachments (0), WS-A, WS-RM, JMS Headers, and JMS Property (0).

You have invoked an API using a SOAP client.

You can treat the Admin Services APIs as if they were back-end server APIs, and get all the benefits of API management for the admin services.

Do the following WSO2 API Manager to expose SOAP APIs with OAUTH2.0.

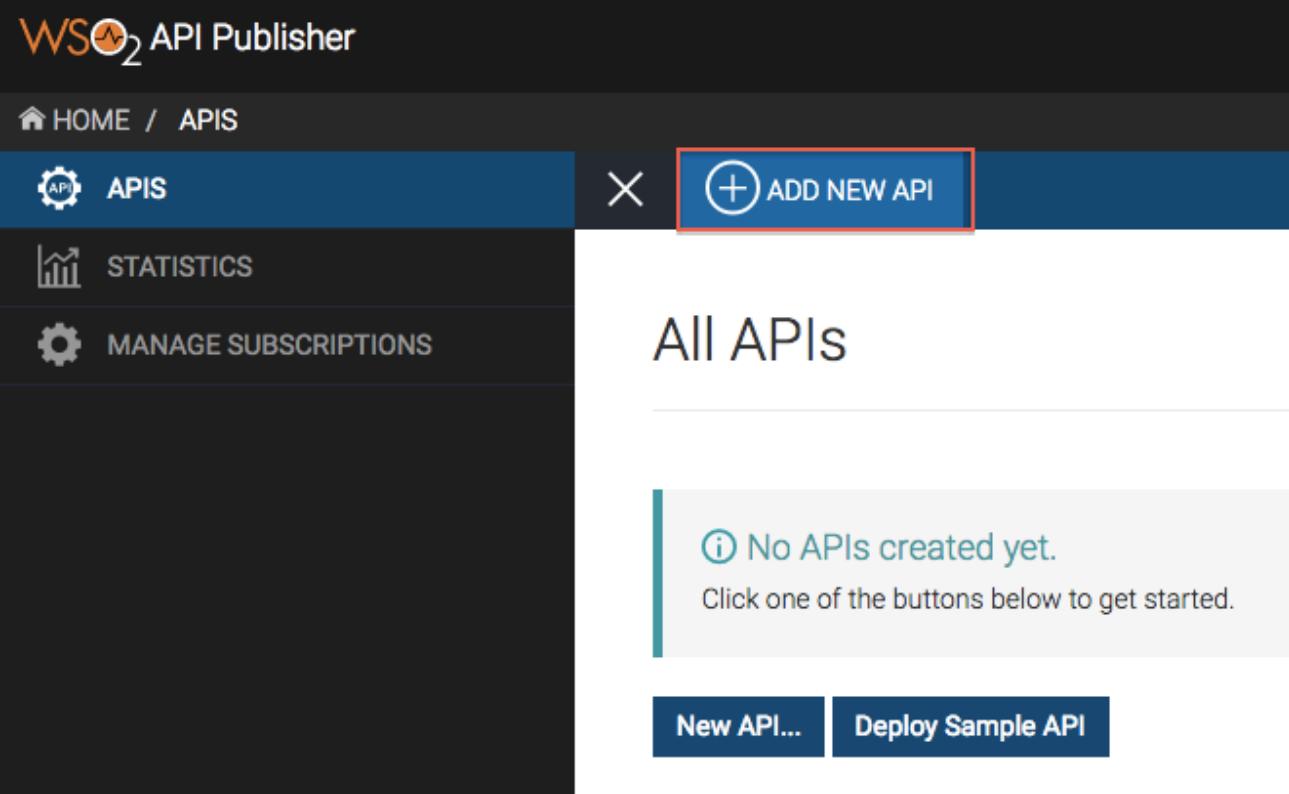
1. Create a SOAP API for an admin service in the publisher.
2. Add backend admin credentials for basic auth in the 'implement' page.
3. Then, publish the app with a scope protecting access - use the the admin role for your scope.

Admins can then subscribe to the app and invoke it using OAUTH2.0 security.

Create and Publish an API from Swagger definition

Swagger definition is a format to describe REST APIs. In this tutorial you create and publish an API in WSO2 API Manager using Swagger definition, when you have an existing API.

1. Sign in to the WSO2 API Publisher.
`https://<hostname>:9443/publisher` (ex: `https://localhost:9443/publisher`). Use **admin** as username and password.
2. In the **APIS** menu, click **Add New API**.



The screenshot shows the WSO2 API Publisher dashboard. At the top, there's a header with the WSO2 logo and navigation links for 'HOME / APIS'. Below the header, a blue navigation bar contains the 'APIS' tab, which is currently selected, and a prominent 'ADD NEW API' button with a plus sign icon, which is also highlighted with a red box. To the left of the main content area, there are two more menu items: 'STATISTICS' and 'MANAGE SUBSCRIPTIONS', each with its own icon. The main content area is titled 'All APIs' and displays a message: 'No APIs created yet.' with an information icon. It also includes a call-to-action: 'Click one of the buttons below to get started.' Below this message are two blue buttons: 'New API...' and 'Deploy Sample API'.

3. Select **I Have an Existing API**. Select **Swagger URL** and type the "http://petstore.swagger.io/v2/swagger.json" URL in the text box. Click **Start Creating**

Add New API

I Have an Existing API
Use an existing API's endpoint or the API Swagger definition to create an API.

Swagger File Swagger URL
`http://petstore.swagger.io/v2/swagger.json`

Start Creating

I Have a SOAP Endpoint
Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

Design a New REST API
Design and prototype a new REST API.

Design a New Websocket API
Design and prototype a new Websocket API.

4. Give the information in the table below.

Field	Sample value
Name	Petstore
Context	/petstore
Version	1.0.0
Visibility	Public
Tags	pets

Design API

1 Design 2 Implement 3 Manage

General Details

Name: * 	<input type="text" value="Petstore"/>
Context: * 	<input type="text" value="/petstore"/>
Version: * 	<input type="text" value="1.0.0"/>
Visibility: 	<input type="button" value="Public"/>
Description:	<input type="text"/>
Maximum 20000 characters.	
Tags: 	<input type="text" value="pets"/> <input type="button" value="Add tags"/>
Type a Tag and Enter	

Thumbnail Image


Dimensions (max): 100 x 100 pix

5. Notice that all the **API resources** are created automatically when the Swagger URL is specified.

POST	/pet	Add a new pet to the store	
PUT	/pet	Update an existing pet	
GET	/pet/findByStatus	Finds Pets by status	
GET	/pet/findByTags	Finds Pets by tags	
GET	/pet/{petId}	Find pet by ID	
POST	/pet/{petId}	Updates a pet in the store with form data	
DELETE	/pet/{petId}	Deletes a pet	
POST	/pet/{petId}/uploadImage	uploads an image	
GET	/store/inventory	Returns pet inventories by status	
POST	/store/order	Place an order for a pet	
GET	/store/order/{orderId}	Find purchase order by ID	
DELETE	/store/order/{orderId}	Delete purchase order by ID	
POST	/user	Create user	
POST	/user/createWithArray	Creates list of users with given input array	
POST	/user/createWithList	Creates list of users with given input array	
GET	/user/login	Logs user into the system	
GET	/user/logout	Logs out current logged in user session	
GET	/user/{username}	Get user by user name	
PUT	/user/{username}	Updated user	
DELETE	/user/{username}	Delete user	

[Save](#) [Next: Implement >](#)

6. Click **Edit Source** to edit the Swagger file and remove security headers. This is required to invoke the API in the Store using the Swagger UI.

API Definition

URL Pattern: /petstore/1.0.0 E.g., path/to/resource

GET POST PUT DELETE PATCH HEAD [more](#)

[+ Add](#)

POST	/pet	Add a new pet to the store	
PUT	/pet	Update an existing pet	
GET	/pet/findByStatus	Finds Pets by status	
GET	/pet/findByTags	Finds Pets by tags	

7. Remove the security tag from the `/pet` POST resource given below. This is required to enable API invocation using API (store) console.

Swagger - Post resource

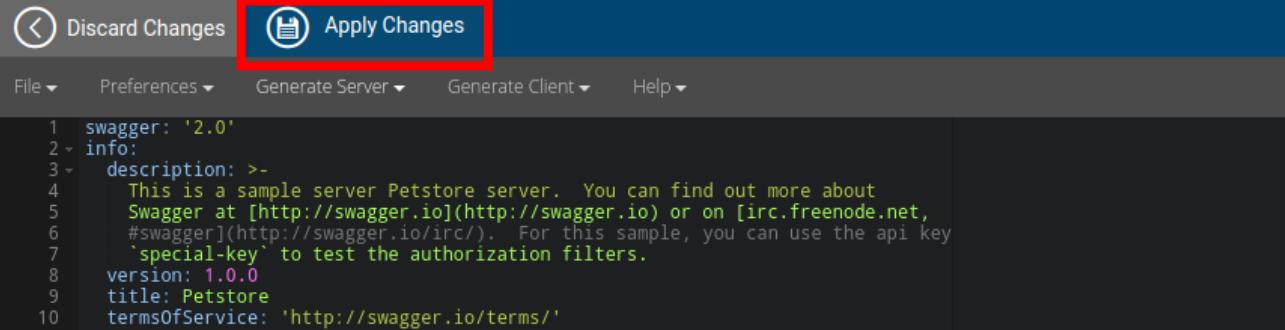
```
//remove the following code snippet
security:
    - petstore_auth:
        - 'write:pets'
        - 'read:pets'
```

8. Remove the security `pet/{petId}` GET resource given below

Swagger - Get resource

```
//remove the following code snippet
security:
    - api_key: []
```

9. After removing the security tags, click **Apply Changes** to save the changes.



```

1  swagger: '2.0'
2  info:
3  description: >-
4      This is a sample server Petstore server. You can find out more about
5      Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode.net,
6      #swagger](http://swagger.io/irc/). For this sample, you can use the api key
7      'special-key' to test the authorization filters.
8  version: 1.0.0
9  title: Petstore
10 termsOfService: 'http://swagger.io/terms/'
```

Click **Next: Implement**.

GET	/user/logout	Logs out current logged in user session
GET	/user/{username}	Get user by user name
PUT	/user/{username}	Updated user
DELETE	/user/{username}	Delete user

Save

Next: Implement >

10. Click the **Managed API** option. Enter the information in the table under the **Implement** tab. Click **Next: Manage**.

Field	Sample value
Endpoint type	HTTP/REST endpoint
Production endpoint	http://petstore.swagger.io/v2/
Sandbox endpoint	Providing production endpoint only is sufficient.

PetStore: /petstore/1.0.0

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type: * **HTTP/REST Endpoint**

Load Balanced Failover

Endpoint: * (specify at least one)

Production :	http://petstore.swagger.io/v2/	<input type="button" value="Test"/>
Sandbox :	E.g.: http://appserver/resource	<input type="button" value="Test"/>

Show More Options

Message Mediation Policies

Enable Message Mediation Check to select a message mediation policy to be executed in the message flow

CORS configuration

Enable API based CORS Configuration **Save** **Next : Manage >**11. Select options for **Transports** and **Subscription Tiers**.

PetStore : /petstore/1.0.0

1 Design 2 Implement

Configurations

Make this the Default version:
No default version defined for the current API

Transports: * **HTTPS** **HTTP**

Response Caching:

Throttling Settings

Maximum Backend Throughput: ? Unlimited Specify

Subscription Tiers: * **Unlimited**: Allows unlimited requests
 Gold: Allows 5000 requests per minute
 Silver: Allows 2000 requests per minute
 Bronze: Allows 1000 requests per minute

The options are described in the table given below.

Field	Sample value	Description
Transports	HTTP and HTTPS	The transport protocol on which the API is exposed. Both HTTP and HTTPS transports are selected by default. If you want to limit API availability to only one transport (e.g., HTTPS), un-check the other transport.
Subscription Tiers	Gold, Silver	The API can be available at different levels of service. They allow you to limit the number of successful hits to an API during a given period of time.

12. Click **Save & Publish**.

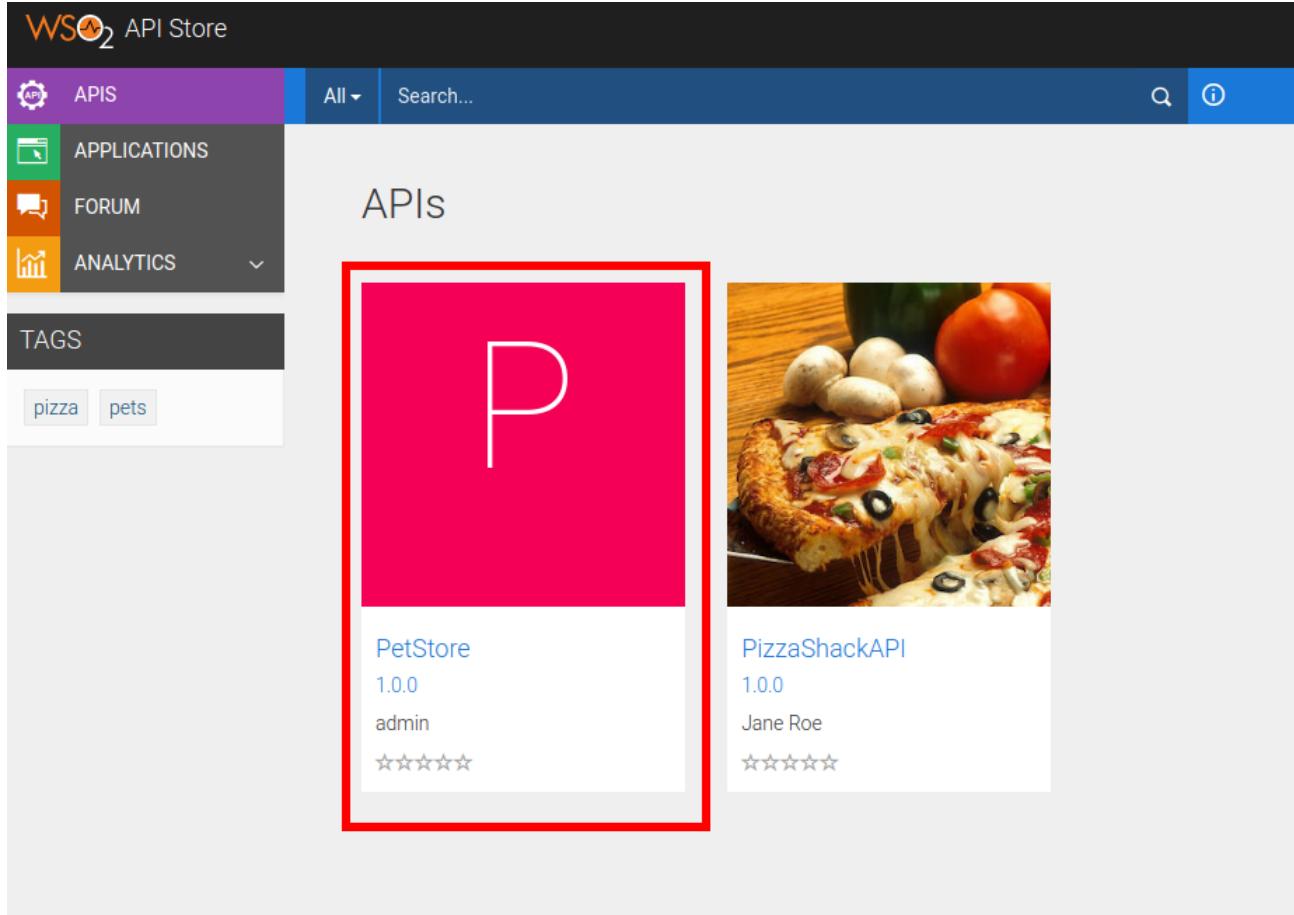
Now you have successfully published an api using swagger defintion. Lets now invoke the API from **API Store**.

Invoking the API

1. Sign in to the WSO2 API Store.

`https://<hostname>:9443/store` (ex: `https://localhost:9443/store`). Use **admin** as username and password

2. Select the **PetStore** API from the Store.



The screenshot shows the WSO2 API Store interface. The top navigation bar includes the WSO2 logo, a search bar, and a user icon. The main content area is titled "APIs". On the left, there's a sidebar with "APIS" selected, followed by "APPLICATIONS", "FORUM", and "ANALYTICS". Below that is a "TAGS" section with "pizza" and "pets" buttons. The main list displays two APIs: "PetStore" and "PizzaShackAPI". The "PetStore" card is highlighted with a red border. It shows the API name, version 1.0.0, owner admin, and a 5-star rating. The "PizzaShackAPI" card shows its name, version 1.0.0, owner Jane Roe, and a 5-star rating. To the right of the cards, there's a thumbnail image of a pizza.

3. Subscribe to the API using the "**DefaultApplication**" and using "**Gold**" tier. For more information, see [Subscribe to an API](#)

PetStore - 1.0.0

Version: 1.0.0
By: admin
Updated: 24/Nov/2017 13:34:40 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications: DefaultApplication
Tiers: Gold
Subscribe

Overview API Console Documentation SDKs Forum

Production and Sandbox Endpoints
Production and Sandbox URLs:

<http://192.168.42.1:8280/petstore/1.0.0>

<https://192.168.42.1:8243/petstore/1.0.0>

Share

- Click on **View Subscriptions** in the pop up.

GO BACK

PetStore - 1.0.0

Subscription Successful

You have successfully subscribed to the API.

View Subscriptions Stay on this page

DefaultApplication

Gold

Subscribe

Overview API Console Documentation SDKs Forum

Production and Sandbox Endpoints

You can view the APIs that the Default Application has subscribed.

DefaultApplication

Subscriptions

API Name	Subscription Tier	Status	Actions
PizzaShackAPI - 1.0.0 PUBLISHED	Unlimited	UNBLOCKED	Unsubscribe
PetStore - 1.0.0 PUBLISHED	Gold	UNBLOCKED	Unsubscribe

Show 10 entries Showing 1 to 2 of 2 entries

- Click on **Petstore - 1.0.0** from the API list.

Token generation

Generate the token for Default application, if you have not generated a token already/

DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

Filter by ...

API Name	Subscription Tier
 PizzaShackAPI - 1.0.0 PUBLISHED	Unlimited
 PetStore - 1.0.0 PUBLISHED	Gold

Show 10 entries Showing 1 to 2 of 2 entries

6. Go to the **API Console** for the PetStore API

PetStore - 1.0.0



Version: 1.0.0
By: admin
Updated: 24/Nov/2017 13:34:40 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Overview **API Console** Documentation SDKs Forum

Production and Sandbox Endpoints
Production and Sandbox URLs:

http://192.168.42.1:8280/petstore/1.0.0	
https://192.168.42.1:8243/petstore/1.0.0	

Show

Applications
Select Application...

Tiers
Gold

7. Click on the **POST pet** resource and give the following example as the request body and click **Try it out**.
Click to see the response.

Set Request Header Authorization : Bearer 8dc4d668-d737-302e-b73b-848bbf3e93e4

Swagger (/swagger.json)
[Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

pet : Everything about your Pets

POST /pet Add a new pet to the store

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	<pre>{ "id": 0, "name": "lion_shepard" } { "status": "available" }</pre>	Pet object that needs to be added to the store	body	Model Example Value

Parameter content type: application/json ▾

Response Messages

HTTP Status Code	Reason	Response Model	Headers
405	Invalid input		

[Try it out!](#)

Request Body Response

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "Dogs"
  },
  "name": "Rover",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "lion_shepard"
    }
  ],
  "status": "available"
}
```

```
{
  "id": 9123612807670061000,
  "category": {
    "id": 0,
    "name": "Dogs"
  },
  "name": "Rover",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "lion_shepard"
    }
  ],
  "status": "available"
}
```

You have now successfully invoked the Petstore API.

Related Tutorials

- [Create and Publish an API](#)
- [Create a WebSocket API](#)
- [Create a Prototyped API with an Inline Script](#)

Create a WebSocket API

WebSocket is a protocol similar to HTTP that is part of the HTML5 specification. It enables simultaneous two-way communication (full-duplex communication) between the client and the server over a single connection. The WebSocket protocol is designed to achieve the following:

- Reduce unnecessary network traffic and latency
- Allow streaming through proxies and firewalls while simultaneously supporting upstream and downstream communication
- Be backward compatible with the pre-WebSocket world by starting up as an HTTP connection before switching to WebSocket frames

A WebSocket API allows an API creator to expose a WebSocket backend as an API to offer services via a WebSocket protocol while providing [OAuth security](#), [throttling](#), analytics, etc.

In this tutorial, you create and publish an API with a WebSocket backend and then invoke it using a Netty-based WebSocket client. You can use any WebSocket client to invoke the API.

1. Sign in to the API Publisher.
<https://<hostname>:9443/publisher> (e.g.: <https://localhost:9443/publisher>). Use admin as username and password.
2. In the **APIS** menu, click **Add New API**.

The screenshot shows the WSO2 API Publisher interface. At the top, there's a navigation bar with 'HOME / APIS'. Below it, a blue header bar contains three tabs: 'APIS' (selected), 'STATISTICS', and 'MANAGE SUBSCRIPTIONS'. To the right of the tabs is a red-bordered button labeled '+ ADD NEW API'. The main content area is titled 'All APIs'.

3. Select the option to design a new WebSocket API and click **Start Creating**.

Let's get started!

Add New API

The configuration screen lists four options:

- I have an Existing API: Use an existing API's endpoint or the API definition to create an API.
- I have a SOAP Endpoint: Use an existing SOAP endpoint to create a managed API. Import WSDL of the SOAP service.
- Design New REST API: Design and prototype a new REST API.
- Design New Websocket API: Design and prototype a new Websocket API.

A red box highlights the 'Start Creating' button at the bottom of the screen.

4. The **Design** tab of the API opens. Give the information in the table below and click **Next: Implement >** to proceed to the implementation phase.

Field	Sample value
Name	EchoWebSocket
Context	/echowebsocket
Version	1.0

Design API

1 Design 2 Implement 3 Manage

General Details

Name: *	EchoWebSocket	Thumbnail Image
Context: *	/echowebsocket	
Version: *	1.0	Dimensions (max): 100 x 100 pixels
Visibility: *	Public	Select image
Description:	<p>Maximum 20000 characters.</p> <p>Type a Tag and Enter</p>	
Tags: *	Add tags	

Save **Next: Implement >**

5. Click the **Managed API** option.
6. Provide the production endpoint and sandbox endpoint, which is `ws://echo.websocket.org:80` in this example, and click **Next: Manage >**.

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.

The **Test** button for the production and sandbox endpoints does not work for WebSocket APIs and is a known issue.

EchoWebSocket: /echowebsocket/1.0

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Production Endpoint :* Test

Sandbox Endpoint :* Test

Save Next : Manage >

7. In the **Manage** tab, select the **Gold** tier, scroll down and click **Save and Publish**.

EchoWebSocket : /echowebsocket/1.0

Configurations
Make this the Default Version: No default version defined for the current API

Throttling Settings
Subscription Tiers: * **Unlimited** : Allows unlimited requests
 Gold : Allows 5000 requests per minute (highlighted with a red box)
 Silver : Allows 2000 requests per minute
 Bronze : Allows 1000 requests per minute

Advanced Throttling Policies:

- You have now published the WebSocket API to the API Store. Let's subscribe to it.
8. When prompted, choose to open the newly published API in the API Store.
 9. The EchoWebSocket API opens. Select an application (e.g., DefaultApplication), the **Gold tier** and subscribe to the API.

EchoWebSocket - 1.0

Version: 1.0
By: admin
Updated: 15/Dec/2016 21:31:30 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications: DefaultApplication
Tiers: Gold

Subscribe

10. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.
11. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. If you have already generated keys before, click **Re-generate**.

DefaultApplication

Production Keys

Show Keys

Consumer Key

Consumer Secret

Grant Types

SAML2 (checked) IWA-NTLM (checked)
Client Credential (checked) Code (unchecked)
Implicit (unchecked) Password (checked)
Refresh Token (checked)

Callback URL

Update

You can also add a **Callback URL**, if you have not added it already when creating the API. You have now subscribed to an API in the API Store and can invoke it using a WebSocket client. In this tutorial, you invoke it using a [Netty-based WebSocket client](#).

12. In your client application, set the WebSocket API URL as shown in the API Store.

EchoWebSocket - 1.0



Version: 1.0

By: admin

Updated: 15/Dec/2016 21:31:30 PM IST

Status: PUBLISHED

Rating: ★★★★★ ✘

Overview

Documentation

Forum

SDKs

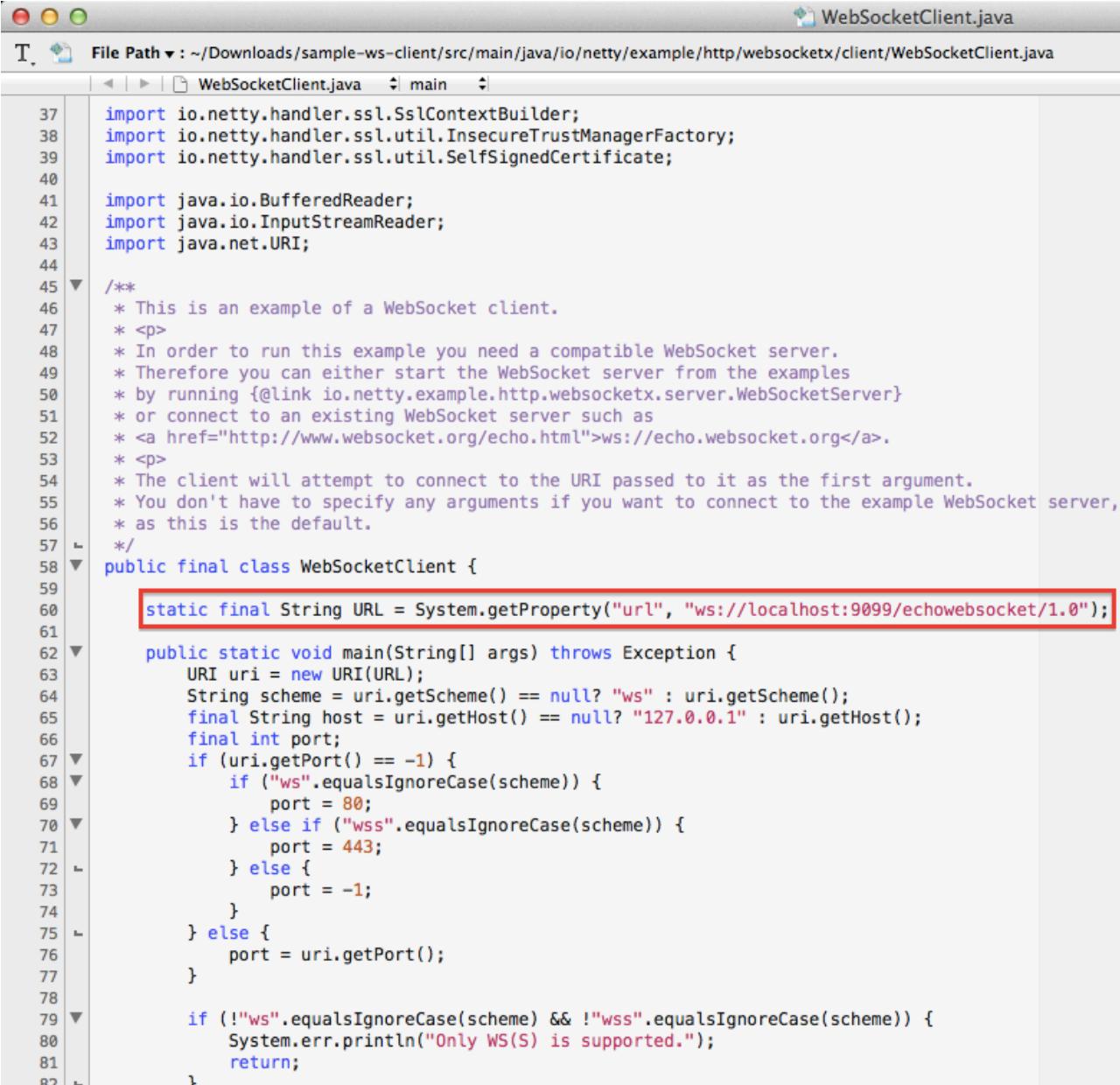
Production and Sandbox Endpoints

Production and Sandbox URLs:

ws://localhost:9099/echowebsocket/1.0

SDK generation is not supported for WebSocket APIs as they don't have swagger definitions.

13. In this example, make sure that the URL in the `sample-ws-client/src/main/java/io/netty/examp
le/http/websocketx/client/WebSocketClient.java` file matches the one in the API Store.

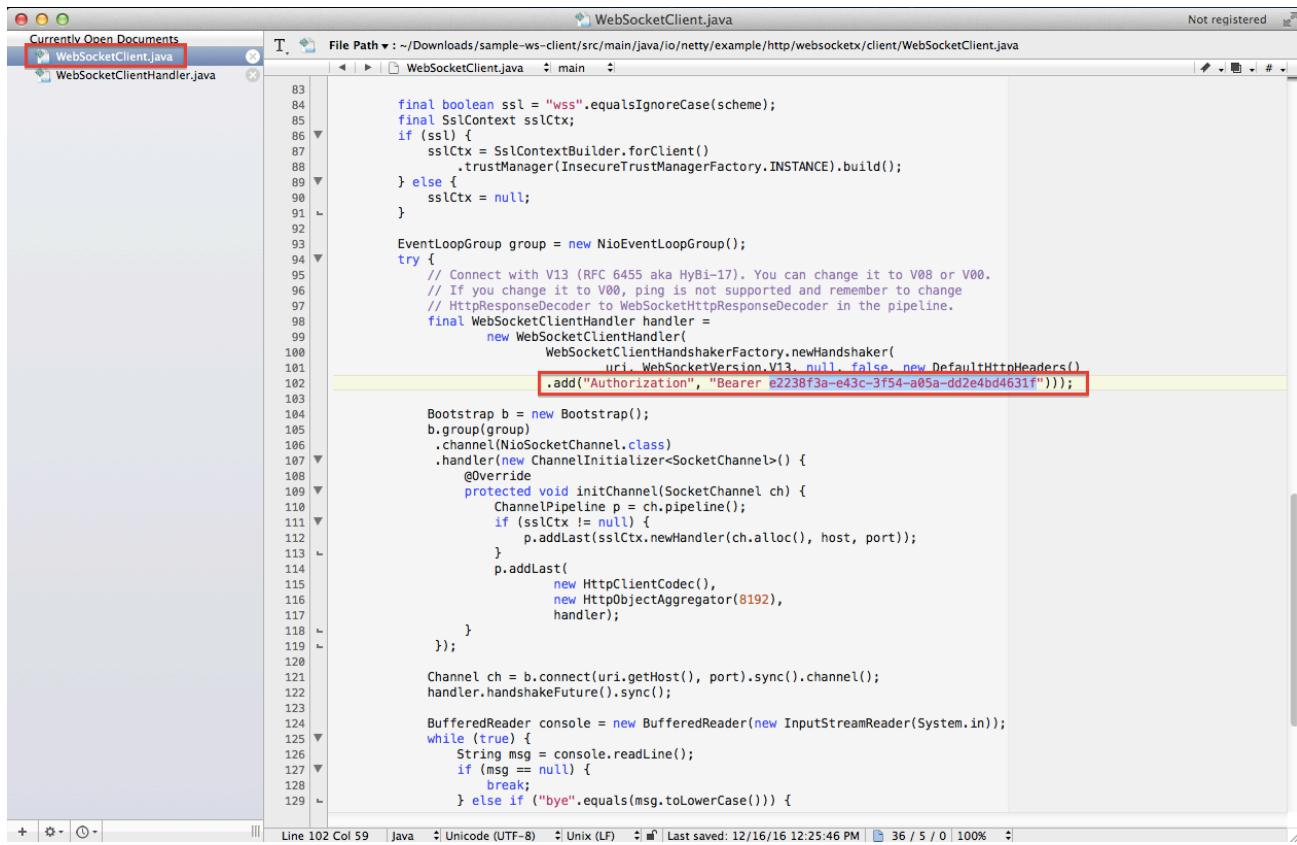


```

37 import io.netty.handler.ssl.SslContextBuilder;
38 import io.netty.handler.ssl.util.InsecureTrustManagerFactory;
39 import io.netty.handler.ssl.util.SelfSignedCertificate;
40
41 import java.io.BufferedReader;
42 import java.io.InputStreamReader;
43 import java.net.URI;
44
45 /**
46 * This is an example of a WebSocket client.
47 * <p>
48 * In order to run this example you need a compatible WebSocket server.
49 * Therefore you can either start the WebSocket server from the examples
50 * by running {@link io.netty.example.http.websocketx.server.WebSocketServer}
51 * or connect to an existing WebSocket server such as
52 * <a href="http://www.websocket.org/echo.html">ws://echo.websocket.org</a>.
53 * <p>
54 * The client will attempt to connect to the URI passed to it as the first argument.
55 * You don't have to specify any arguments if you want to connect to the example WebSocket server,
56 * as this is the default.
57 */
58 public final class WebSocketClient {
59
60     static final String URL = System.getProperty("url", "ws://localhost:9099/echowebsocket/1.0");
61
62     public static void main(String[] args) throws Exception {
63         URI uri = new URI(URL);
64         String scheme = uri.getScheme() == null? "ws" : uri.getScheme();
65         final String host = uri.getHost() == null? "127.0.0.1" : uri.getHost();
66         final int port;
67         if (uri.getPort() == -1) {
68             if ("ws".equalsIgnoreCase(scheme)) {
69                 port = 80;
70             } else if ("wss".equalsIgnoreCase(scheme)) {
71                 port = 443;
72             } else {
73                 port = -1;
74             }
75         } else {
76             port = uri.getPort();
77         }
78
79         if (!"ws".equalsIgnoreCase(scheme) && !"wss".equalsIgnoreCase(scheme)) {
80             System.err.println("Only WS(S) is supported.");
81             return;
82         }
83     }
84 }

```

14. In the same file, copy and paste the Authorization Bearer access token you generated in step 11 as shown below.

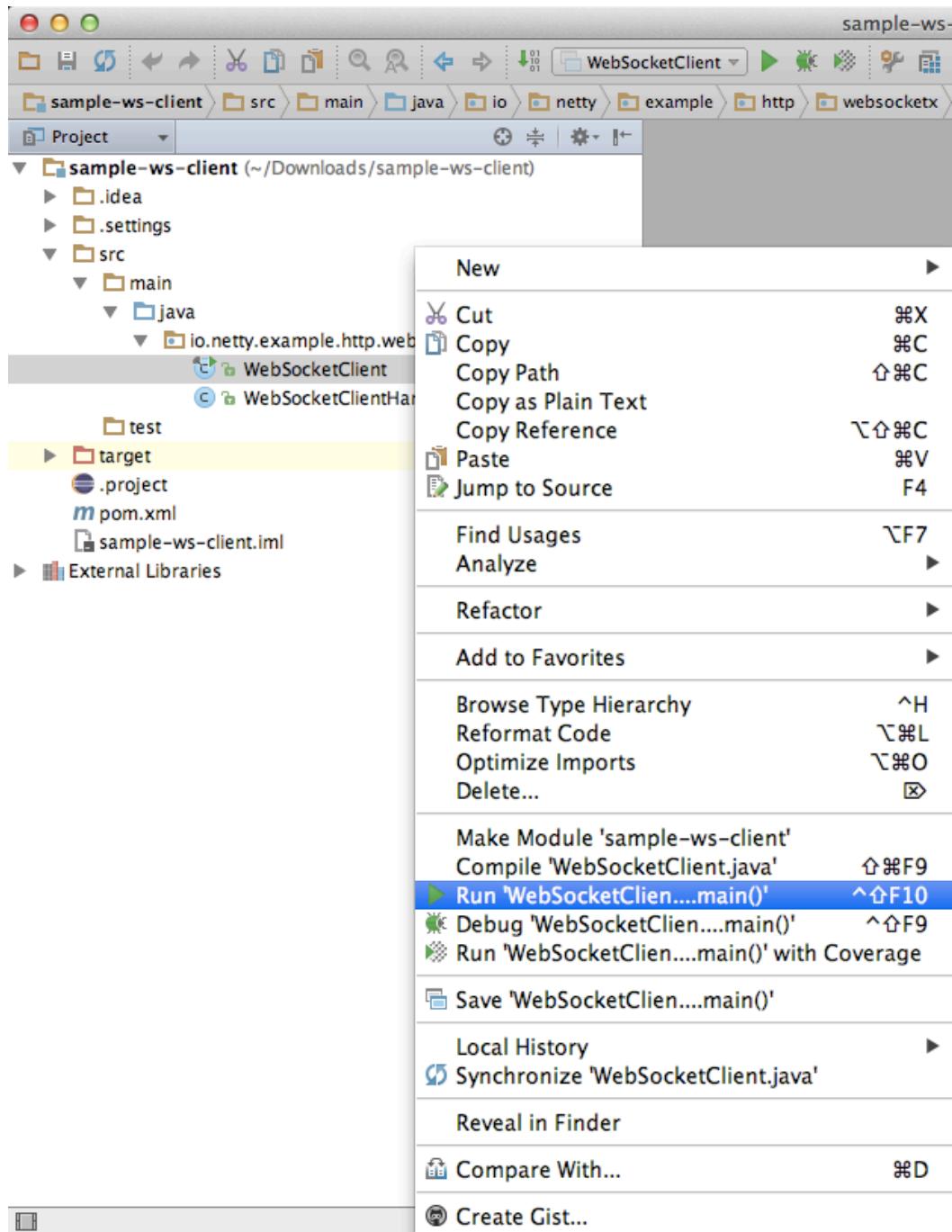


```

    final boolean ssl = "wss".equalsIgnoreCase(scheme);
    final SslContext sslCtx;
    if (ssl) {
        sslCtx = SslContextBuilder.forClient()
            .trustManager(InsecureTrustManagerFactory.INSTANCE).build();
    } else {
        sslCtx = null;
    }
    EventLoopGroup group = new NioEventLoopGroup();
    try {
        // Connect with V13 (RFC 6455 aka HyBi-17). You can change it to V08 or V00.
        // If you change it to V00, ping is not supported and remember to change
        // HttpResponseDecoder to WebSocketHttpDecoder in the pipeline.
        final WebSocketClientHandler handler =
            new WebSocketClientHandler(
                WebSocketClientHandshakerFactory.newHandshaker(
                    uri, WebSocketVersion.V13, null, false, new DefaultHttpHeaders()
                    .add("Authorization", "Bearer e2238f3a-e43c-3f54-a05a-dd2e4bd4631f")));
        Bootstrap b = new Bootstrap();
        b.group(group)
            .channel(NioSocketChannel.class)
            .handler(new ChannelInitializer<SocketChannel>() {
                @Override
                protected void initChannel(SocketChannel ch) {
                    ChannelPipeline p = ch.pipeline();
                    if (sslCtx != null) {
                        p.addLast(sslCtx.newHandler(ch.alloc(), host, port));
                    }
                    p.addLast(
                        new HttpClientCodec(),
                        new HttpObjectAggregator(8192),
                        handler);
                }
            });
        Channel ch = b.connect(uri.getHost(), port).sync().channel();
        handler.handshakeFuture().sync();
        BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
        while (true) {
            String msg = console.readLine();
            if (msg == null) {
                break;
            } else if ("bye".equals(msg.toLowerCase())) {

```

15. Save your changes.
16. Open the sample-ws-client directory you downloaded in step 11 using an IDE. This tutorial uses IntelliJ Idea 15 CE as the IDE.
17. Run the WebSocket client as shown below.



18. Type a message in the WebSocket client and you will see that it echoes the message as intended by the WebSocket API.

```

Run: WebSocketClient WebSocketClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
WebSocket Client connected!
test message
WebSocket Client received message: test message
test message 2
WebSocket Client received message: test message 2

```

Related Tutorials

- Create and Publish an API
- Create and Publish an API from Swagger definition
- Create a Prototyped API with an Inline Script

Create a Prototyped API with an Inline Script

In this tutorial, you create a prototyped API with an inline script, deploy it as a prototype, and invoke it using the API Console integrated in the API Store. You create APIs with inline scripts typically for testing purposes. 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 that the users request and publish the API.

1. Sign in to the API Publisher.
<https://<hostname>:9443/publisher> (e.g., <https://localhost:9443/publisher>). Use admin as username and password.
2. Select the option to design a new REST API and click **Start Creating**.

Let's get started!

Add New API

I have an Existing API
Use an existing API's endpoint or the API definition to create an API.

I have a SOAP Endpoint
Use an existing SOAP endpoint to create a managed API. Import WSDL of the SOAP service.

Design New REST API
Design and prototype a new REST API.

Start Creating

Design New Websocket API
Design and prototype a new Websocket API.

3. Give the information in the table below. To add resources, click the **Add** button. Since the URL Pattern used here is a variable, it is denoted within curly braces.

Field		Sample value

Name		Location_API
Context		/location
Version		1.0.0
Resources	URL pattern	{town}
	Request types	GET

1 Design 2 Implement 3 Manage

General Details

Name: * Location_API

Context: * /location

Version: * 1.0.0

Visibility: * Public

Description:

Thumbnail Image



Select image

Dimensions (max): 100 x 100 pixels

Tags: * Add tags

Type a Tag and Enter

API Definition

URL Pattern /location/1.0.0/{town}

Import Edit Source

GET POST PUT DELETE PATCH HEAD more

+ Add

4. After the resource is added, expand its GET method and note that a parameter by the name town is added under the resource. You use it to pass the payload to the backend. Once done, click **Next: Implement >**.

To specify multiple parameters in the API resource, separate the parameters with a forward slash.

{param1} / {param2}

GET /{town} [+ Summary](#)

Description : [+ Add Implementation Notes](#)

Produces : Empty Consumes : Empty

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
town	+ Empty	path	string	True	

Parameter Name [\(+\) Add Parameter](#)

[Save](#) [Next Implement >](#)

- In the **Prototyped API** section under the **Implement** tab, select the implementation method as **Inline**.
Location_API: /location/1.0.0

1 Design 2 Implement 3 Manage

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Prototyped API
Use the inbuilt JavaScript engine to prototype the API or provide an endpoint to a prototype API. The inbuilt JavaScript engine does not hav... [^](#)

Implementation Method Inline Endpoint

Resources

GET /{town} [+ Summary](#)

CORS configuration

Enable API based CORS Configuration

[Save](#) [Deploy as a Prototype](#)

The inline JavaScript engine does not provide support for SOAP APIs. If you opt for the endpoint implementation method instead of inline, you need to provide an endpoint to a prototype API. For example, <http://ws.cdyne.com/phoneverify/phoneverify.asmx>

- Expand the GET method and give the following as the script. It reads the payload that the user sends with the API request and returns it as a JSON value. The value **mc** is the message context.

```
mc.setProperty('CONTENT_TYPE', 'application/json');
var town = mc.getProperty('uri.var.town');
mc.setPayloadJSON('{"Town" : "'+town+'"}');
```

GET

/town

[+ Summary](#)

Description : [+ Add Implementation Notes](#)

Response Content Type :

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required
town	+ Empty	path	string	True

Script :

```

1 mc.setProperty('CONTENT_TYPE', 'application/json');
2 var town = mc.getProperty('uri.var.town');
3 mc.setPayloadJSON('{ "Town" : "'+town+'"}');

```

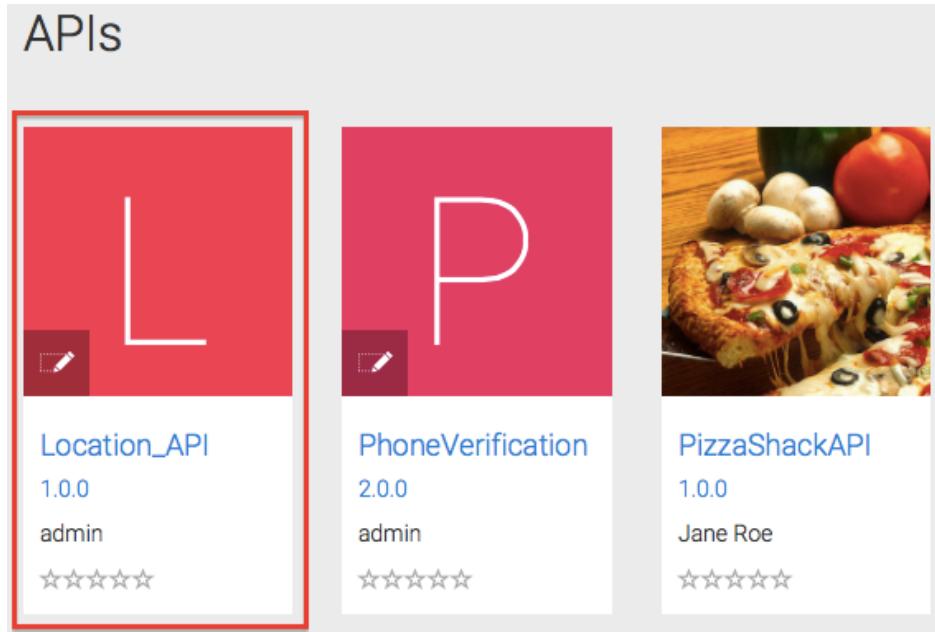
CORS configuration

Enable API based CORS Configuration

[Save](#) [Deploy as a Prototype](#)

7. Click **Deploy as a Prototype**.
8. Go to the API Store and note that the newly deployed API is listed there.

Tip: You can invoke prototyped APIs without signing in to the API Store or subscribing to the API. The purpose of a prototype is advertising and giving an early implementation for users to test.



9. Click the API to open it and go to its **API Console** tab.

Location_API - 1.0.0



Version: 1.0.0
By: admin
Updated: 30/Jul/2016 22:26:40 PM IST
Status: PROTOTYPED

Overview API Console Documentation Forum

10. Expand the GET method, give any value for the town (say London) and invoke the API.

GET /{town}

Parameters

Parameter	Value	Description
town	London	

Response Messages

HTTP Status Code	Reason	Response Model
200		

Try it out!

11. Note the payload you gave as a JSON output in the response.

Response Body

```
{
  "Town": "London"
}
```

You have created an API with inline script, deployed it as a prototype and invoked it through the integrated API Console.

An API can also be prototyped by moving the API to the prototyped state in the API lifecycle. For more information, see the [Deploy and Test as a Prototype](#) tutorial.

Related Tutorials

- [Create and Publish an API](#)
- [Create a WebSocket API](#)
- [Create and Publish an API from Swagger definition](#)

Pass a Custom Authorization Token to the Backend

This tutorial uses the WSO2 API Manager Tooling Plug-in.

When you send an API request to the backend, you pass a token in the Authorization header of the request. The API Gateway uses this token to authorize access, and then drops it from the outgoing message. If you wish to use a different (or a custom generated) authorization token than the application generated access token, you can use it as a token exchange mechanism in mediation logic of the API. In this tutorial, we explain how to pass a custom authorization token that is different to the authorization token generated for the application.

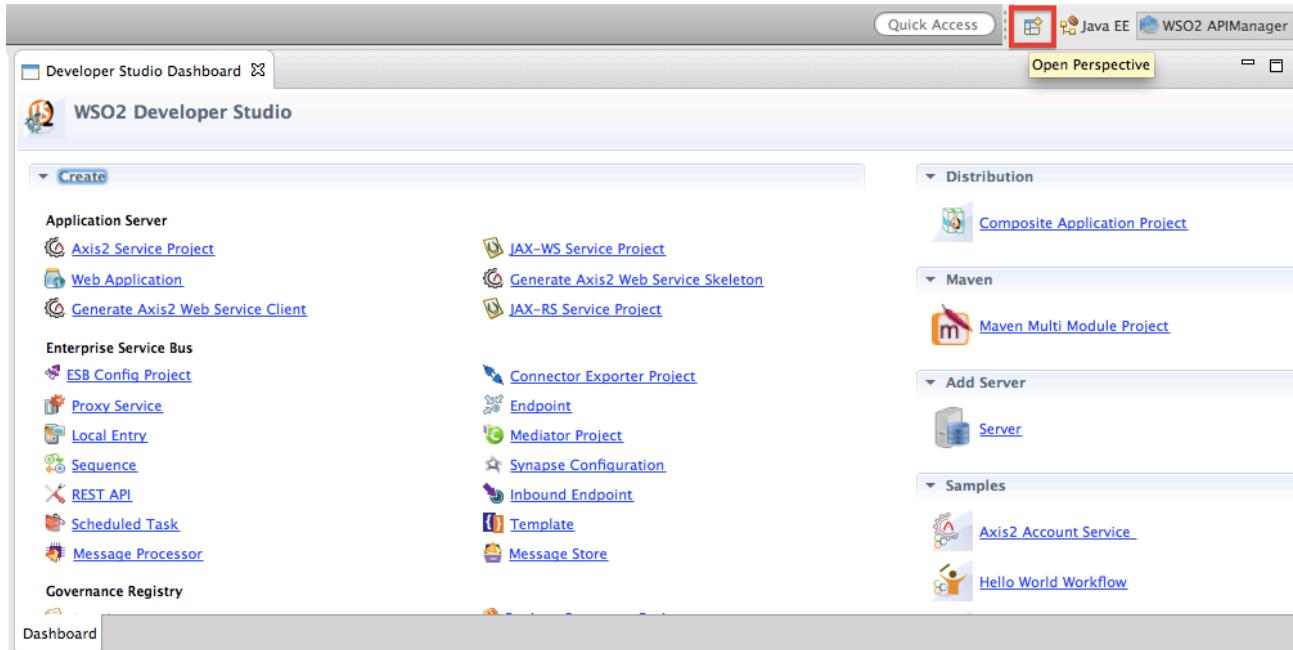
In this tutorial, you have a sample JAX-RS backend and it always expects 1234 as the authorization token. In your API request, you pass the token that is generated in the Authorization header, and 1234 in a Custom header. The mediation extension you write extracts the value of the Custom header, and sets it as the Authorization header before sending it to the backend.

Here's a summary:

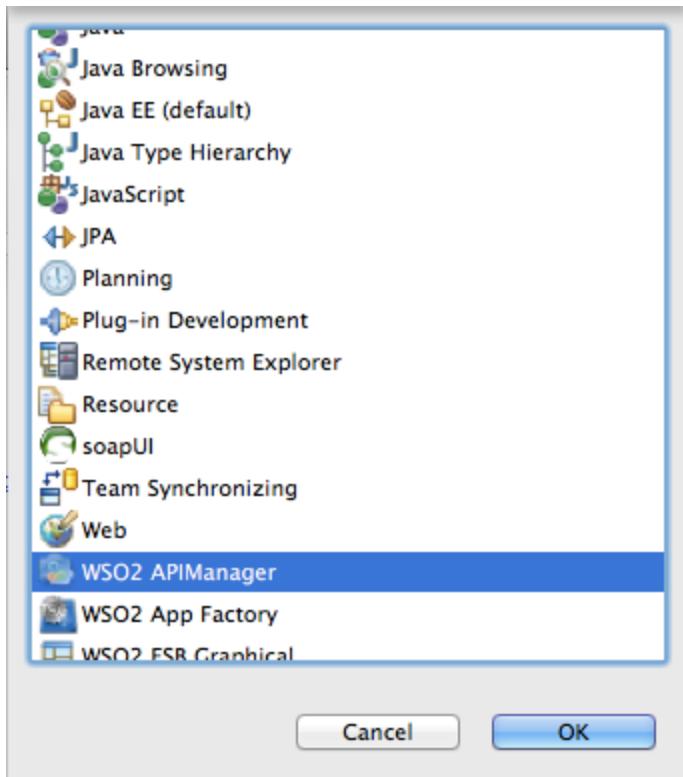
```
Client (headers: Authorization, custom) -> Gateway (drop: Authorization, convert: custom->Authorization) -> Backend
```

Let's get started.

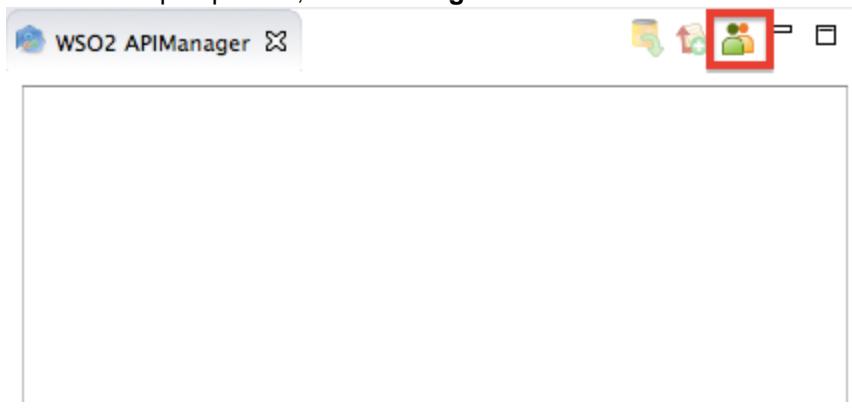
1. Download and install the [WSO2 API Manager Tooling Plug-in](#) if you have not done so already. Open Eclipse by double clicking the Eclipse.app file inside the downloaded folder.
2. Click **Window > Open Perspective > Other** to open the Eclipse perspective selection window. Alternatively, click the **Open Perspective** icon shown below at the top right corner.



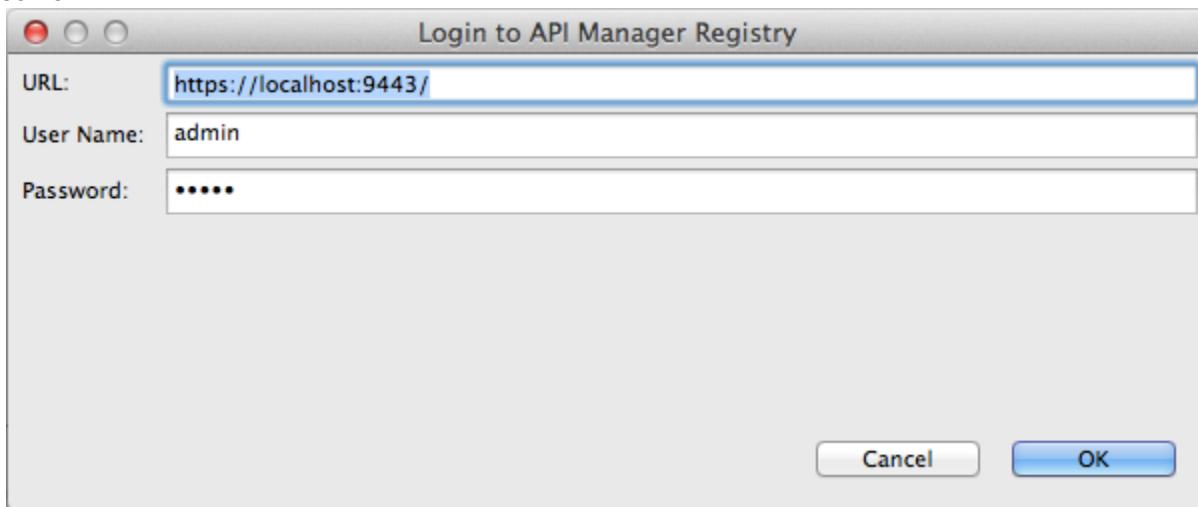
3. On the dialog box that appears, click **WSO2 APIManager** and click **OK**.



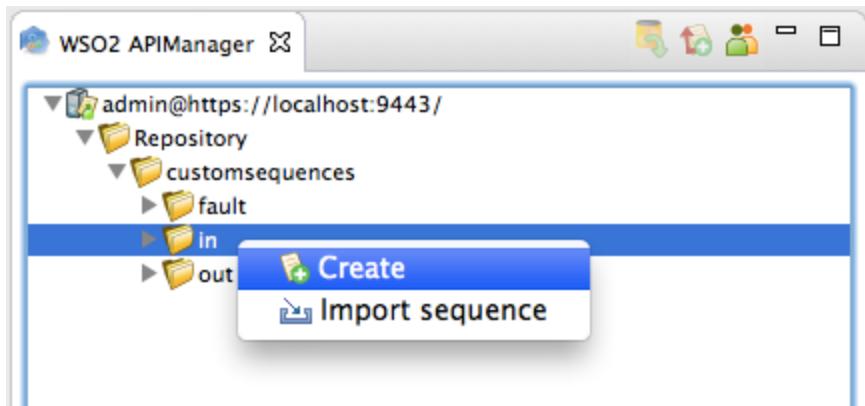
- On the APIM perspective, click the **Login** icon as shown below.



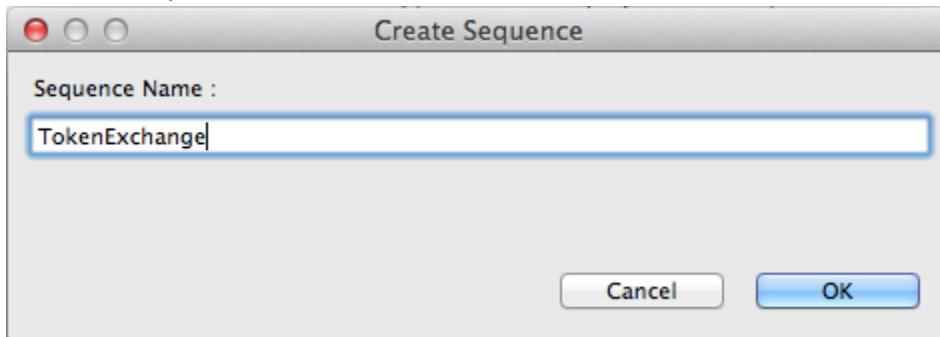
- On the dialog box that appears, enter the URL, username and password (by default `admin`) of the Publisher server.



- On the tree view that appears, expand the folder structure of the existing API.
- Right-click on the `in sequence` folder and click **Create** to create a new `in sequence`.



8. Name the sequence TokenExchange.

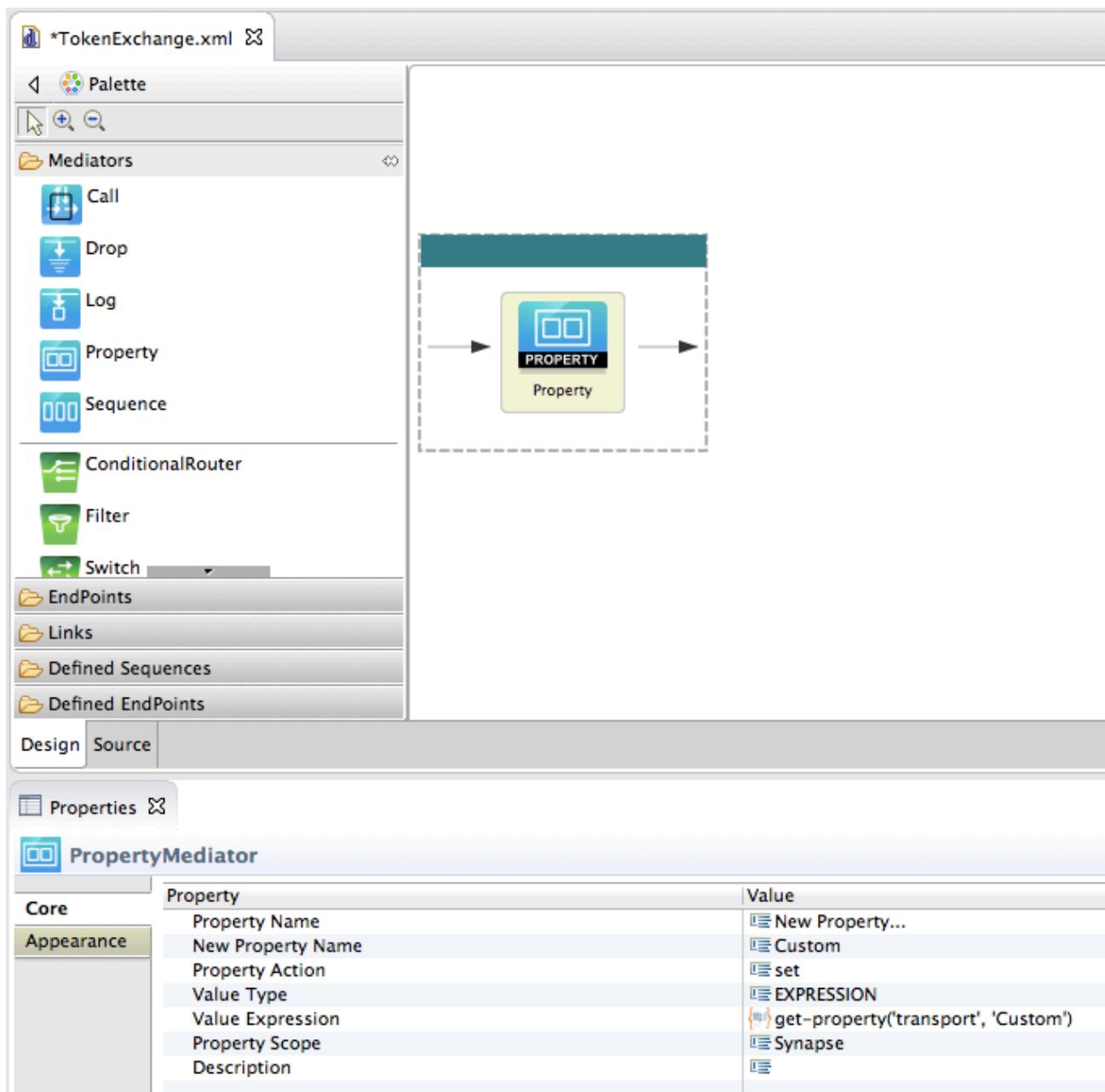


9. Your sequence now appears on the APIM perspective. From under the **Mediators** section, drag and drop a **Property** mediator to your sequence and give the following values to the mediator.

Tip: The **Property Mediator** has no direct impact on a message, but rather on the message context flowing through [Synapse](#). For more information, see [Property Mediator](#) in the WSO2 EI documentation.

The following property mediator is used to assign the Custom transport level property to another property called **Custom**.

Property Name	New Property
New Property Name	Custom
Value Type	EXPRESSION
Value Expression	get-property('transport', 'Custom')



- Similarly, add another **Property** mediator to your sequence and give the following values to the mediator. This property mediator is used to construct a transport level property called **Authorization** and assign itself the value of the Custom property created above.

Property Name	New Property
New Property Name	Authorization
Value Type	EXPRESSION
Value Expression	get-property('Custom')
Property Scope	transport

The screenshot shows the WSO2 API Manager Studio interface for editing a mediation sequence named "TokenExchange.xml".

Palette:

- Mediators:**
 - Call
 - Drop
 - Log
 - Property
 - Sequence
- ConditionalRouter**
- Filter**
- Switch**
- EndPoints**
- Links**
- Defined Sequences**
- Defined EndPoints**

Design tab selected.

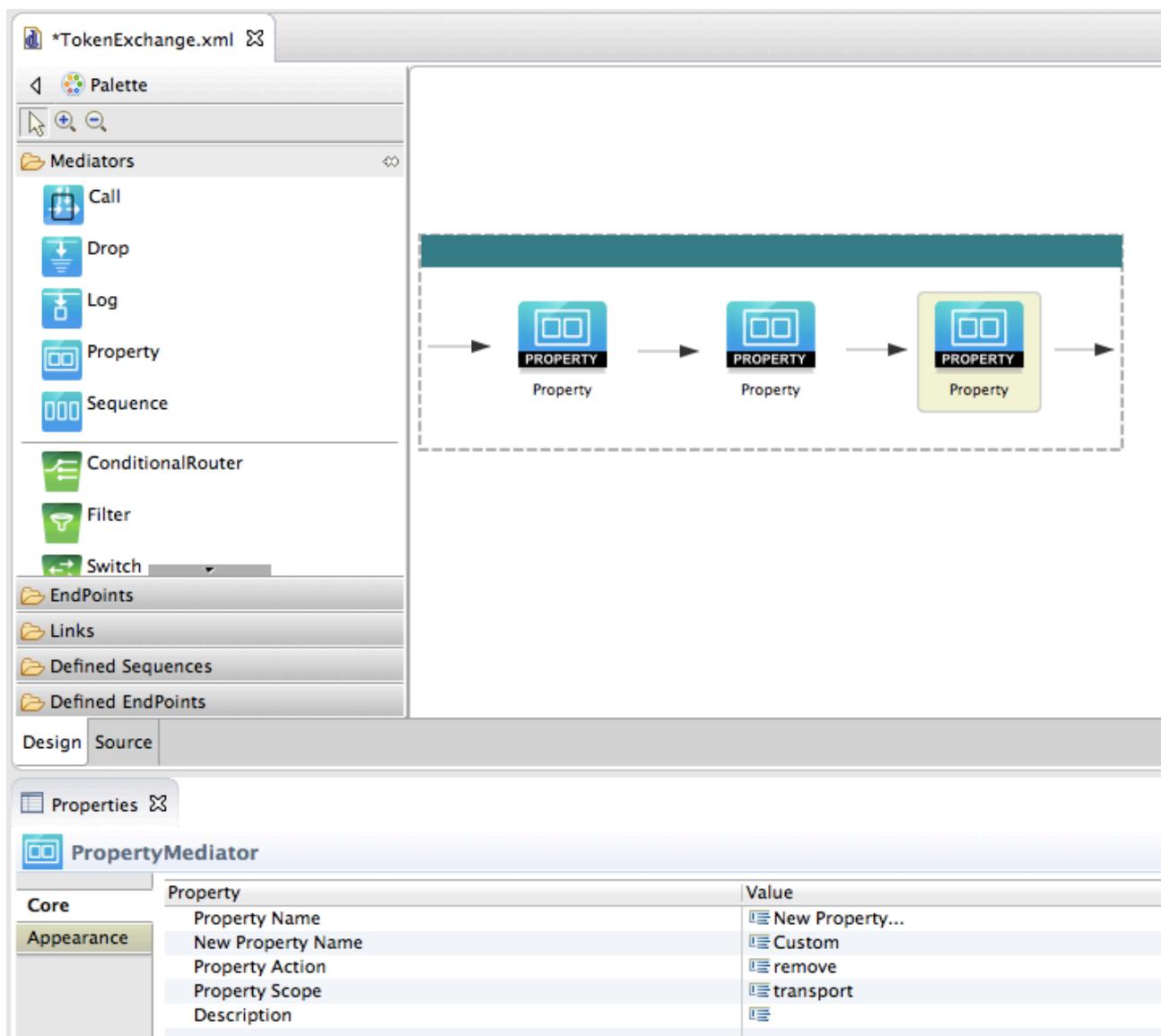
Properties View:

PropertyMediator

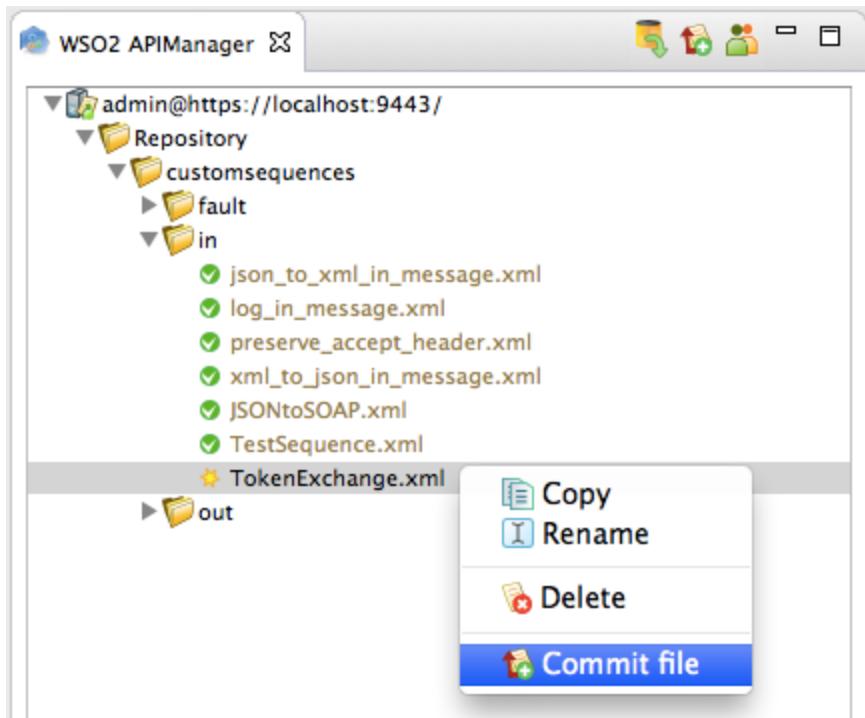
Property	Value
Core	New Property...
Appearance	Authorization
Property Name	set
New Property Name	EXPRESSION
Property Action	get-property('Custom')
Value Type	transport
Value Expression	
Property Scope	
Description	

11. Add a third **Property** mediator to your sequence and give the following values to the mediator. This property mediator is used to remove the **Custom** property from the transport level.

Property Name	New Property
New Property Name	Custom
Property Action	remove
Property Scope	transport



12. Save the sequence.
13. Right-click on the sequence and click **Commit File** to push the changes to the Publisher server.



Let's create a new API and engage the sequence you created to it.

14. Log in to the API Publisher, click the **Add** link and give the information in the table below.

Field	Sample value
Name	TestAPI1
Context	/test1
Version	1.0.0
Visibility	Public

General Details

Name:*	<input type="text" value="TestAPI"/>	Thumbnail Image
Context:*	<input type="text" value="/test"/>	 Select image
Version:*	<input type="text" value="1.0.0"/>	Dimensions (max): 100 x 100 pixels
Visibility:	<input type="button" value="Public"/>	
Description:	<small>Maximum 20000 characters.</small>	
Tags:	<input type="text" value="Add tags"/> <small>Type a Tag and Enter</small>	

15. Leave the **Resources** section blank, and click **Next: Implement >**. Add a wildcard resource (*) when prompted. Click **Next: Implement >** again to move to the **Implement** tab.

The screenshot shows the 'API Definition' section of the WSO2 API Manager. At the top, there is a URL Pattern input field containing '/test/1.0.0' and a 'Url Pattern E.g.: path/to/resource' placeholder. Below it are checkboxes for various HTTP methods: GET, POST, PUT, DELETE, PATCH, HEAD, and a 'more' option. There is also an 'Import' button and an 'Edit Source' button. A large 'Add' button with a plus sign is centered below the method checkboxes. Below this is a table listing five operations:

Method	Path	Description	Action
GET	/*	+ Summary	
POST	/*	+ Summary	
PUT	/*	+ Summary	
DELETE	/*	+ Summary	
PATCH	/*	+ Summary	

At the bottom left are 'Save' and 'Next: Implement >' buttons.

16. The **Implement** tab opens. Give the information in the table below.

Field	Sample value
Endpoint type	HTTP endpoint
Production endpoint	https://appserver.cloud.wso2.com/t/clouddemo/webapps/authheadersample-1.0.0/services
Sandbox endpoint	https://appserver.cloud.wso2.com/t/clouddemo/webapps/authheadersample-1.0.0/services

TestAPI: /test/1.0.0

The screenshot shows the 'Managed API' section of the WSO2 API Manager. It has three tabs at the top: 'Design' (selected), 'Implement' (highlighted in blue), and 'Manage'. The 'Implement' tab contains a form for managing endpoints:

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type :* ?

Production Endpoint :* ?

Sandbox Endpoint :* ?

Show More Options

17. Select the **Enable Message Mediation** check box, engage the **In** sequence that you created earlier and click **Manage**.

Message Mediation Policies

Enable Message Mediation	<input checked="" type="checkbox"/> Check to select a message mediation policy to be executed in the message flow	
In Flow	TokenExchange	Upload In Flow
Out Flow	None	Upload Out Flow
Fault Flow	None	Upload Fault Flow

In Flow, Out Flow and Fault Flow represent the custom In, Out and Fault sequences attached to the API by the user other than the default sequence definition of the API.

18. In the **Manage** tab, select the **Gold** tier and click **Save and Publish** to publish the API to the API Store.
TestAPI : /test/1.0.0

1 Design 2 Implement 3 Manage

Configurations

Make this the Default Version:
No default version defined for the current API

Transports: * HTTPS HTTP

Response Caching: Disabled

Throttling Settings

Maximum Backend Throughput: ? Unlimited Specify

Subscription Tiers: * ? **Unlimited**: Allows unlimited requests
 Gold: Allows 5000 requests per minute
 Silver: Allows 2000 requests per minute
 Bronze: Allows 1000 requests per minute

Let's subscribe to the API and invoke it.

19. Log in to the API Store and subscribe to the API using an available application and the Gold tier. If there are no applications available by default, create one.

TestAPI - 1.0.0

Version: 1.0.0
By: admin
Updated: 30/Jul/2016 13:27:26 PM IST
Status: PUBLISHED
Rating: ★★★★☆

Applications
DefaultApplication

Tiers
Gold

Subscribe

20. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.
21. Click the **Production Keys** tab and click **Generate Keys** to create an application access token.

DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

No Keys Found
No keys are generated for this type in this application.

Grant Types

SAML2 IWA-NTLM Implicit Refresh Token
 Client Credential Code Password

Callback URL

Access token validity period
 Seconds.

Generate keys

22. Install any REST client in your machine. We use **cURL** here.
23. Go to the command line, and invoke the API using the following cURL command. In this command, you pass the token that the backend expects, i.e., 1234, in the **Custom** header with the authorization token that the system generates in the **Authorization** header.

```
curl -H "Authorization: Bearer <access token>" -H "Custom: Bearer 1234" <API URL>
```

Note the following:

- **<access token>** is the token that you got in step 20.
- **<API URL>** appears on the API's **Overview** page in the API Store. Copy the HTTP endpoint. If you select the HTTPS endpoint, be sure to run the cURL command with the -k option.

Here's an example:

```
curl -k -H "Authorization: Bearer 2e25097b2b3fbbfb44f5642fa8a495a1" -H "Custom: Bearer 1234" https://localhost:8243/test/1.0.0
```

24. Note the response that you get in the command line. According to the sample backend used in this tutorial, you get the response as "Request Received."

```
$ curl -k -H "Authorization: Bearer 2e25097b2b3fbbfb44f5642fa8a495a1" -H "Custom: Bearer 1234" https://localhost:8243/test1/1.0.0
<Response><code>200</code><message>Request Received</message><description>Request Received successfully</description></Response>
```

In this tutorial, you passed a custom token that the backend expects along with the system-generated Authorization token, and invoked an API successfully by swapping the system's token with your custom token.

Create and Publish a SOAP API

WSO2 API Manager supports the management of both REST and SOAP APIs.

In this tutorial, we create and publish an **API with a SOAP endpoint** and then invoke it using the integrated API Console and a third-party tool (SOAP UI).

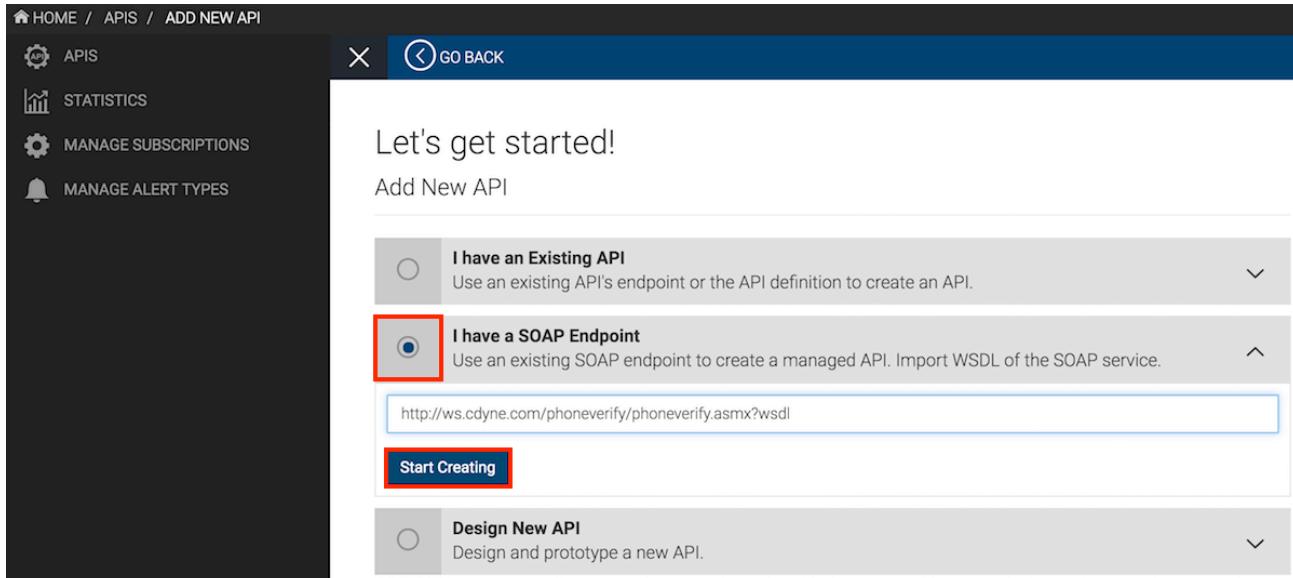
See the following topics for a description of the concepts that you need to know when invoking an API:

- Applications
- Throttling
- Access tokens

Let's get started.

1. Log in to the API Publisher and click **ADD NEW API**.
2. Select the option to design an API with an existing SOAP endpoint, give the endpoint URL and click **Start Creating**.

This example uses the WSDL <http://ws.cdyne.com/phoneverify/phoneverify.asmx?wsdl> from CDYNE as the endpoint here, but you can use any SOAP backend of your choice.



3. The **Design** tab of the API opens. Give the information in the table below and click **Next: Implement >** to proceed to the implementation phase.

Field	Sample value
Name	SoapTest
Context	/soapttest
Version	1.0

The screenshot shows the WSO2 API Manager interface for creating a new API. The 'Design' tab is selected (highlighted with a red box). The 'General Details' section contains fields for Name (SoapTest), Context (soapttest), Version (1.0), Visibility (Public), Description (empty), and Tags (Add tags). The 'API Definition' section shows a WSDL endpoint of <http://ws.cdyne.com/>. The 'Manage' section is visible at the top right, showing a thumbnail image placeholder and a 'Select image' button. The 'Next: Implement >' button is highlighted with a red box.

General Details

Name: * SoapTest

Context: * soapttest

Version: * 1.0

Visibility: Public

Description:

Tags: Add tags

Type a Tag and Enter

API Definition

WSDL: http://ws.cdyne.com/ Test URI

Save Next: Implement >

4. Click the **Managed API** option.
5. Select **HTTP/SOAP Endpoint**, provide the production endpoint, which is <http://ws.cdyne.com/phonereverify/phonestoreverify.asmx> in this example, and click **Manage**.

Managed API
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type: *

Load Balanced Failover

Production Endpoint: *

Sandbox Endpoint: *

Show More Options

Message Mediation Policies

Enable Message Mediation Check to select a message mediation policy to be executed in the message flow

CORS configuration

Enable API based CORS Configuration

Save **Next : Manage >**

6. In the **Manage** tab, select the **Gold** tier, scroll down and click **Save and Publish**.

If you wish to add scopes to the resources that were created, click **Add Scopes** and specify the scopes you want to add. If you specify a scope, you need to use the same scope when generating access tokens for the subscribed application and when invoking the API. For more information on working with the scopes, see [OAuthscopes](#).

Resources					
Method	Path	Scopes:	Application & Application User	Quota	Scope
POST	/*	<input type="button" value="+ Add Scopes"/>	Application & Application User	Unlimited	<input type="button" value="+ Scope"/>

Save **Save & Publish** **Cancel**

Configurations

Make this the Default version: No default version defined for the current API

Transports: * HTTPS HTTP

Response Caching: Disabled

Subscriptions: Available to current tenant only

Throttling Settings

Maximum Backend Throughput: Unlimited Specify

Subscription Tiers: * Unlimited: Allows unlimited requests
 Gold: Allows 5000 requests per minute (highlighted)
 Silver: Allows 2000 requests per minute
 Bronze: Allows 1000 requests per minute

Advanced Throttling Policies: Apply to API Apply per Resource (highlighted)

Select Policy per Resource

Refer documentation for more information about each throttling setting.

You have now published the SOAP API to the API Store. Let's subscribe to it.

Note that when creating this API, the default option, **Apply per Resource**, was selected under **Advanced Throttling Policies**. For more information on setting advanced throttling policies, see [Enforce Throttling and Resource Access Policies](#).

7. Log in to the API Store and open the newly created API from the store.
8. The SoapTest API opens. Select an application (e.g., DefaultApplication), the **Gold** tier and subscribe to the API.

SoapTest - 1.0

Version: 1.0
By: nirdesha.wso2.com@companyn3
Updated: 27/Oct/2016 05:22:43 AM GMT-08:00
Status: PUBLISHED
Rating: ★★★★★

Applications: DefaultApplication

Tiers: Gold

Subscribe

9. Click the **APPLICATIONS** menu and click the **Production Keys** tab. If you have an access token already generated, scroll down and click **Re-generate**. By default, access tokens expire an hour after creation, unless you change the expiration time.

APPLICATIONS

APIS

STATISTICS

APPLICATION LIST

EDIT

DefaultApplication

Details Production Keys **Sandbox Keys** Subscriptions

Show Keys

Consumer Key

.....

Consumer Secret

.....

Grant Types

Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM	<input type="checkbox"/> Implicit	<input checked="" type="checkbox"/> Refresh Token
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code	<input checked="" type="checkbox"/> Password	

Callback URL

.....

Update

Generating Access Tokens

You have now subscribed to an API in the API Store. Let's invoke the API.

- Back in the API Store, click the API to open it and go to its **API Console** tab.

APIS

APPLICATIONS

STATISTICS

GO BACK

SoapTest - 1.0

S

Version: 1.0
By: nirdesha.wso2.com@companyn3
Updated: 27/Oct/2016 05:22:43 AM GMT-08:00
Status: **PUBLISHED**
Rating: ★★★★★

Applications
Select Application...

Tiers
Gold

Subscribe

Overview API Console Documentation

Try DefaultApplication

Using Production

Set Request Header Authorization : Bearer e2e7d654-84a7-3e9c-9dd8-2b2a31624e2c

- Expand the POST method, enter the following, and invoke the API.

SOAP Request

```
<soapenv:Envelope  
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:quer="http://ws.cdyne.com/PhoneVerify/query">  
    <soapenv:Header/>  
    <soapenv:Body>  
  
        <quer:CheckPhoneNumber>  
            <!--Optional:-->  
  
            <quer:PhoneNumber>180067854  
32</quer:PhoneNumber>  
            <!--Optional:-->  
  
            <quer:LicenseKey>0</quer:L  
icenseKey>  
  
        </quer:CheckPhoneNumber>  
        </soapenv:Body>  
    </soapenv:Envelope>
```

SOAP Action

<http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber>

POST /*

Parameters

Parameter	Value	Description
SOAP Request	<pre><!--Optional:--> <quer:LicenseKey>0</quer:LicenseKey> </quer:CheckPhoneNumber> </soapenv:Body> </soapenv:Envelope></pre>	SOAP request.

Parameter content type:

text/xml

SOAPAction	http://ws.cdyne.com/PhoneVerify/query/CheckP	SOAPAction header for soap 1.1
------------	---	--------------------------------

Response Messages

HTTP Status Code	Reason	Response Model
200	OK	

[Try it out!](#)

12. Note the API response that appears on the console.

Response Body

```
<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><CheckPhoneNumberResponse xmlns="http://ws.cdyne.com/PhoneVerify/query"><CheckPhoneNumberResult><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 /><TelecomCounty /><TelecomState>TF</TelecomState><TelecomZip /><TimeZone /><Lat /><Long /><Wireless>false</Wireless></CheckPhoneNumberResult></CheckPhoneNumberResponse></soap:Body></soap:Envelope>
```

Response Code

200

Response Headers

```
{
  "pragma": "no-cache",
  "content-type": "text/xml; charset=utf-8",
  "cache-control": "no-cache",
  "expires": "-1"
}
```

In this tutorial, you have created an API with a SOAP backend and invoked it using both the integrated Swagger API Console in the API Manager, as well as an external tool.

You can invoke this API using a third party tool, such as SOAP UI, as well. For more information on how to invoke an API using a SOAP client, see [Invoke an API using a SOAP Client](#).

Disable Message Chunking

When processing large messages, message chunking facilitates sending the message as multiple independent chunks. Message chunking is set using the Transfer-Encoding: chunked header. However, some legacy backends might not support chunked messages. To disable sending chunked messages to the backend for a specific API, follow the steps below:

1. Create an XML file with the following content:

disable chunking

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence xmlns="http://ws.apache.org/ns/synapse"
  name="disableChunkingSeq">
  <property name="DISABLE_CHUNKING" value="true" scope="axis2"/>
</sequence>
```

2. Use the same sequence and apply it as a mediation extension to the inflow of this particular API. For more details, see [Creating per-API extensions](#). Once the API is published, chunking is disabled for the message

that is sent to the backend.

To stop chunked messages from being sent to the client, you can apply the same mediation extension to the out sequence as well.

Enable API Indexing on Remote Publisher and Store Nodes

WSO2 API Manager uses Solr indexing to display API details on API Publisher and Store. In the tenant mode, the indexing process starts when a user of a tenant logs in to the Store. However, when there are large numbers of APIs, the indexing process can take a significant time and any new APIs may not be visible in Store nodes until the indexing process finishes. Therefore, in order to avoid the latter mentioned delay, you can start the API indexing for tenants on a remote Publisher/Store nodes as follows:

Starting API indexing on remote Publisher/Store nodes depends on Hazelcast clustering, therefore; clustering needs to be enabled in the Publisher/Store nodes and they should be in the same clustering domain.

1. Enable Hazelcast clustering for Publisher and Store nodes. Make sure that all Publisher and Store nodes are in the same cluster domain and the nodes are properly joined to the cluster.
2. Open <API-M_HOME>/bin/wso2server.sh file.
3. Add the Java System Property `enableTenantLoadNotification=true` at the bottom of the file:

```
.....
-DworkerNode=false \
-DenableTenantLoadNotification=true \
org.wso2.carbon.bootstrap.Bootstrap $*
status=$?
done
```

4. Repeat this on all Publisher and Store nodes, and restart them.

Now, when a tenant user logs in to a particular Publisher/Store the other Publisher/Store nodes are notified. Therefore, the indexing jobs will start immediately.

Remove Specific Request Headers From Response

1. Shutdown the server if it is already running.
2. Open the <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/main.xml file.
3. Add the name of the header to be removed as a property, just before the beginning of send mediator, as shown below

```
<property name="" scope="transport" action="remove" />
```

```
<property name="Accept" scope="transport" action="remove" />
<property name="X-JWT-Assertion" scope="transport" action="remove" />
<property name="Cookie" scope="transport" action="remove" />
<send/>
```

4. Open the <APIM_HOME>/repository/deployment/server/synapse-configs/default/sequence

s/fault.xml file.

- Add the name of the header to be removed as a property property, just before the beginning of "CORS request handler" sequence, as shown below.

```
<property name="<name of the header to be removed>" scope="transport"
action="remove" />
```

```
<property name="Accept" scope="transport" action="remove" />
<property name="X-JWT-Assertion" scope="transport" action="remove" />
<property name="Cookie" scope="transport" action="remove" />
<sequence key="_cors_request_handler_" />
```

- Start the server.

Note : The above method removes only the specified headers from the response. If you need to remove all the headers, follow the instructions below.

- Open the <APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences/main.xml file.
- Add the TRANSPORT_HEADERS property, after the beginning of <out> sequence opening tag, as shown in the example below.

Example

```
<out>
<property name="TRANSPORT_HEADERS" action="remove"
scope="axis2" />
```

- Open the <APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences/fault.xml file.
- Add the TRANSPORT_HEADERS property before the <send> mediator, as shown in the example below.

Example

```
<property name="TRANSPORT_HEADERS" action="remove"
scope="axis2" />
<send/>
```

Scope Management with 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. For a detailed description and a sample real world scenario, please see the article [An Overview of Scope Management with WSO2 API Manager](#).

Generate REST API from SOAP Backend

This feature allows users to expose their legacy SOAP backends as REST APIs through WSO2 API Manager.

WSO2 API Manager supports WSDL 1.1 based SOAP backends.

The following instructions explain how to generate REST APIs in WSO2 API Manager for an existing SOAP backend.

Before you begin...

Make sure that you have a valid WSDL URL from the SOAP backend. It should belong to the WSDL 1.1 version.

1. Log into the API Publisher and click **ADD NEW API**
2. Select I have a SOAP endpoint. You will see the following two options to create APIs for SOAP backend.
 - **Pass Through** – Create a pass through proxy for SOAP requests coming to the API Gateway
 - **Generate REST APIs** – This option is used to generate REST API definitions from the given WSDL URL.
3. Provide the WSDL URL for the SOAP backend and click **Start Creating**. The default option is **Pass Through**

Let's get started!

Add New API

I Have an Existing API
Use an existing API's endpoint or the API Swagger definition to create an API.

I Have a SOAP Endpoint
Use an existing SOAP endpoint to create a managed API. Import the WSDL of the SOAP service.

http://appserver/services/echo?wsdl

Pass Through Generate Rest Apis

Start Creating

Design a New REST API
Design and prototype a new REST API.

Design a New Websocket API
Design and prototype a new Websocket API.

4. Select the **Generate REST APIs** option and go to the **Design** tab. Click on **Edit Source** to edit the swagger specification of the API.

The screenshot shows the WSO2 API Manager's Swagger Editor interface. On the left, the Swagger Editor displays the API definition code:

```

1 swagger: "2.0"
2 paths:
3   /citiesByCountry:
4     get:
5       tags:
6         - citiesByCountry
7       parameters:
8         - description: ""
9           in: query
10          name: CountryName
11          type: string
12       responses:
13         "200":
14           description: ""
15           x-auth-type: "Application & Application User"
16           x-throttling-tier: Unlimited
17   /weather:
18     get:
19       tags:
20         - weather
21       parameters:
22         - description: ""
23           in: query
24           name: CityName
25           type: string
26         - description: ""
27           in: query
28           name: CountryName
29           type: string
30       responses:
31         "200":
32           description: ""

```

On the right, the generated API resources are listed under the 'globalweather' API version 1.0.0:

- citiesByCountry** (GET /citiesByCountry)
- weather** (GET /weather)

Click **Apply Changes** to save your API.

- The generated API definitions will be added to the API as shown below.

The screenshot shows the WSO2 API Manager's API Definition page. It lists two API endpoints:

- /weather** (GET): Description: + Add Implementation Notes. Produces: Empty. Consumes: Empty. Parameters: CityName, CountryName. Both parameters are of type query, string type, and required.
- /citiesByCountry** (GET): Description: + Add Implementation Notes. Produces: Empty. Consumes: Empty. Parameters: CountryName. Parameter is of type query, string type, and required.

The definition properties are mapped with a swagger vendor-specific field `x-xpath`, which is used to map the SOAP binding operation parameters with the REST parameters. This should not be edited by the user. If a parameter does not have this field it will be not mapped with backend operation

- Go to the **Implement** tab and view the **SOAP Mapping** section. Click on a resource to view the In and Out sequences of the API.

SOAP to REST Mapping

Resources

GET	/citiesByCountry /weather	+ Summary + Summary
GET		

Description : [+ Add Implementation Notes](#)

Response Content Type :

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required
CityName		query	string	
CountryName		query	string	

Sequences :

In Sequence Out Sequence

```

1 <header description="SOAPAction" name="SOAPAction" scope="transport" value="http://www.webserviceX.NET/GetWeather"/>
2 <property name="REST_URL_POSTFIX" scope="axis2" action="remove"/>
3 <property expression="$url:CityName" name="req.var.CityName"/>
4 <property expression="$url:CountryName" name="req.var.CountryName"/>
5
6 <payloadFactory description="transform" media-type="xml">
7   <format>
8     <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:web="http://www.webserviceX.NET">
9       <soapenv:Header/>
10      <soapenv:Body>
11        <web:GetWeather xmlns:web="http://www.webserviceX.NET">
12          <web:CityName>$1</web:CityName>
13          <web:CountryName>$2</web:CountryName>
14        </web:GetWeather>
15
16      </soapenv:Body>

```

7. The following sample shows the generated API In-sequence for a GET method with query parameters.

```

GET
https://<host_name>:8243/weather/1.0.0/weather?CityName=xxxxx&Country
Name=xxxxx

```

```

<property name="HTTP_METHOD" value="POST" scope="axis2"
type="STRING" />
    <header name="SOAPAction"
        scope="transport"
        value="http://www.webserviceX.NET/GetWeather"
        description="SOAPAction"/>
    <property name="REST_URL_POSTFIX" scope="axis2"
action="remove" />
        <property name="req.var.CityName"
expression="$url:CityName" />
        <property name="req.var.CountryName"
expression="$url:CountryName" />
            <payloadFactory media-type="xml" description="transform">
                <format>
                    <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:web="http://www.webserviceX.NET">
                    <soapenv:Header/>
                    <soapenv:Body>
                        <web:GetWeather>
                            <web:CityName>$1</web:CityName>
                            <web:CountryName>$2</web:CountryName>
                        </web:GetWeather>
                    </soapenv:Body>
                </soapenv:Envelope>
            </format>
            <args>
                <arg evaluator="xml"
expression="get-property('req.var.CityName' ) " />
                <arg evaluator="xml"
expression="get-property('req.var.CountryName' ) " />
            </args>
        </payloadFactory>
        <property name="messageType"
            value="application/soap+xml"
            scope="axis2"
            type="STRING"
            description="messageProperty"/>

```

The incoming JSON message parameters will be stored using properties. The SOAP payload needed for the backend is generated using a payload factory mediator.

8. Enter the SOAP endpoint URL. The endpoint should be HTTP/SOAP Endpoint as shown below.

Endpoint Type :* ? **HTTP/SOAP Endpoint**

Load Balanced Failover

Endpoint: * (specify at least one)

Production : ?	http://www.webservicex.com/globalweather.asmx	Test
Sandbox : ?	E.g.: http://appserver/resource	Test

[Manage Certificates](#)

[Show More Options](#)

9. Go to the **Manage** tab and the relevant configuration. Click **Publish** to publish the API to the API Store. For instructions, see [Create and Publish an API](#).
10. Navigate to the API Store and invoke the API.

Developer Portal

How do I...

- [Include Additional Headers in the API Console](#)
- [Log in to the API Store using Social Media](#)
- [Test an API using a Testing Tool](#)
- [Use the Community Features](#)
- [Write a Client Application Using the SDK](#)
- [Obtaining User Profile Information with OpenID Connect](#)
- [Cleaning Up Partially Created Keys](#)

Include Additional Headers in the API Console

The Swagger API Console is a JavaScript client that runs in the API Store and makes JavaScript calls from the Store to the API Gateway. You must specify any additional headers that you want to add to the API Console under the CORS ([Cross Origin Resource Sharing](#)) configuration.

Open the CORS configuration in the `<APIM_HOME>/repository/conf/api-manager.xml` file, enable CORS if it is not enabled already and specify the additional headers (SOAPAction, in this case) under the `<Access-Control-Allow-Headers>` element:

CORS configurations in api-manager.xml

```

<CORSConfiguration>
    <Enabled>true</Enabled>
    <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>

    <Access-Control-Allow-Methods>GET,PUT,POST,DELETE,PATCH,OPTIONS</Access-Control-Allow-Methods>

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

```

This configuration is only valid for APIs created through the API manager Publisher application. All the other OAuth token related APIs (/authorize, /revoke, /token, /userinfo) are not affected from this. To enable CORS configuration to these APIs as well, see [Enabling CORS for OAuth Token related APIs](#).

Next, let's see how to add the header as a parameter to the API Console.

1. Log in to the API Publisher and click the API that you want to invoke (e.g., PhoneVerification).
2. Click the **Edit** link next to the API's name, navigate down to the **API Definition** section and click on the **POST** method to expand it.

3. Update the **Produces** and **Consumes** fields to `text/xml` and create the following header using the **Add Parameter** button.

Parameter name	Values
SOAPAction	Description: Set to <code>http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</code> Parameter Type: Header Data Type: String Required: False

Description : + Add Implementation Notes

Produces : text/xml **Consumes :** text/xml

Parameters :

Parameter Name	Description	Parameter Type	Data Type	Required	Delete
Payload	Request Body	body	+ Empty	False	(edit)
SOAPAction	Set to <code>http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber</code>	header	string	False	(edit)

Parameter Name (+) Add Parameter

Save Next: Implement >

4. Once you are done, click **Save**.
5. Log in to the API Store, subscribe to the API and generate an access token for the application you subscribed with.
If it's an API that you are already subscribed to, you might have to re-generate the access token from the **Applications** page.

DefaultApplication

Details Production Keys Sandbox Keys Subscriptions

i No Keys Found

No keys are generated for this type in this application.

Grant Types

Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM	<input type="checkbox"/> Implicit	<input checked="" type="checkbox"/> Refresh Token
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code	<input checked="" type="checkbox"/> Password	

Callback URL

Access token validity period

3600	Seconds.
------	----------

Generate keys

6. Click on the API again to open it and then click its **API Console** tab.

PhoneVerification - 1.0.0



Version: 1.0.0
By: admin
Updated: 30/Jul/2016 11:31:45 AM IST
Status: PUBLISHED
Rating: ★★★★★ 0

Applications:

Tiers:

Subscribe

Overview API Console Documentation Forum

Try:

Using: Key

Set Request Header:

[Swagger \(/swagger.json \)](#)

Show/Hide | List Operations | Expand Operations

default

<input type="button" value="GET"/>	/CheckPhoneNumber
<input type="button" value="POST"/>	/CheckPhoneNumber

7. Expand the POST method, fill the parameter values and invoke the API. For example,

Parameter	Value
Body	<p>This is the example SOAP request that we copied from the SOAP UI of the previous tutorial:</p> <pre><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> <soap:Body> <CheckPhoneNumber xmlns="http://ws.cdyne.com/PhoneVerify/query"> <PhoneNumber>650 745 4499 </PhoneNumber> <!-- Optional LicenseKey parameter--> <LicenseKey>0</LicenseKey> </CheckPhoneNumber> </soap:Body> </soap:Envelope></pre>
Parameter Content Type	text/xml
SOAPAction	http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber

POST /CheckPhoneNumber

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Payload	<pre><!--Optional:--> <quer:PhoneNumber>123456</quer:PhoneNumber> <!--Optional:--> <quer:LicenseKey>0</quer:LicenseKey> </quer:CheckPhoneNumber> </soapenv:Body> </soapenv:Envelope></pre>	Request Body	body	Model Example Value
				<pre><?xml version="1.0"?> <inline_model> <payload>string</payload> </inline_model></pre>

Parameter content type: `text/xml`

SOAPAction `http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber` Set to `http://ws.cdyne.com/PhoneVerify/query/CheckPhoneNumber` header string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

[Try it out!](#)

8. Note the result that appears on the console.

Response Body

```
<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><CheckPhoneNumberResponse xmlns="http://ws.cdyne.com/PhoneVerify/query"><CheckPhoneNumberResult><Company>TELEPORT COMMUNICATIONS GROUP</Company><Valid>true</Valid><Use>Assigned to a code holder for normal use</Use><State>CA</State><NC>SNFC CNTRC</NC><OCN>7145</OCN><OriginalNumber>650 745 4499 </OriginalNumber><CleanNumber>6507454499</CleanNumber><SwitchName>SAN FRANCISCO</SwitchName><SwitchType></SwitchType></Country><Country>US</Country><CLID>SNFCCAFJUJS0</CLID><PrefixType>CLEC - (Competitive Local Exchange Carrier)</PrefixType><LATA>722</LATA><SMS>CLBC - (Competitive Local Exchange Carrier)</SMS><Email></Email><AssignDate>19/11/2014</AssignDate><TelecomCity>San Francisco</TelecomCity><TelecomCounty>San Francisco</TelecomCounty><TelecomState>CA</TelecomState><TelecomZip>94104</TelecomZip><Timezone>PST</Timezone><Lat></Lat><Long></Long><Wireless>false</Wireless><LNRN>4152290000</LNRN></CheckPhoneNumberResult></CheckPhoneNumberResponse></soap:Body></soap:Envelope>
```

Response Code

200

Response Headers

```
{
  "pragma": "no-cache",
  "content-type": "text/xml; charset=utf-8",
  "cache-control": "no-cache",
  "expires": "1"
}
```

You have added SOAP parameters to the API Console and invoked a SOAP service using the API Console.

Enabling CORS for Oauth Token related APIs

Enabling CORS configuration through `api-manager.xml` is only valid for APIs created through the API manager Publisher application. Hence enabling CORS for Oauth token related APIs (`/authorize`, `/revoke`, `/token`, `/userinfo`) can be carried out as follows.

Based on the API that you need to enable CORS, add the following handler configuration to the relevant API synapse file present in `<APIM_HOME>/repository/deployment/server/synapse-configs/default/api` / folder. It should be added within the `<handlers>` parent element.

```
<handler
class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler"
>
<property name="apiImplementationType" value="ENDPOINT" />
</handler>
```

The following are the mappings of the synapse files corresponding to the Oauth token related APIs.

Endpoint	Synapse configuration
<code>/authorize</code>	<code>_AuthorizeAPI_.xml</code>
<code>/revoke</code>	<code>_RevokeAPI_.xml</code>
<code>/token</code>	<code>_TokenAPI_.xml</code>
<code>/userinfo</code>	<code>_UserInfoAPI_.xml</code>

Log in to the API Store using Social Media

You can integrate WSO2 Identity Server with WSO2 API Manager and use your social media credentials to log in to the API Store and API Publisher. This tutorial shows you how to integrate Facebook authentication and log in to the API Store. Before following these steps, configure WSO2 Identity Server to provide Single Sign On for WSO2 API Manager by following [Configuring External IDP through Identity Server for SSO](#).

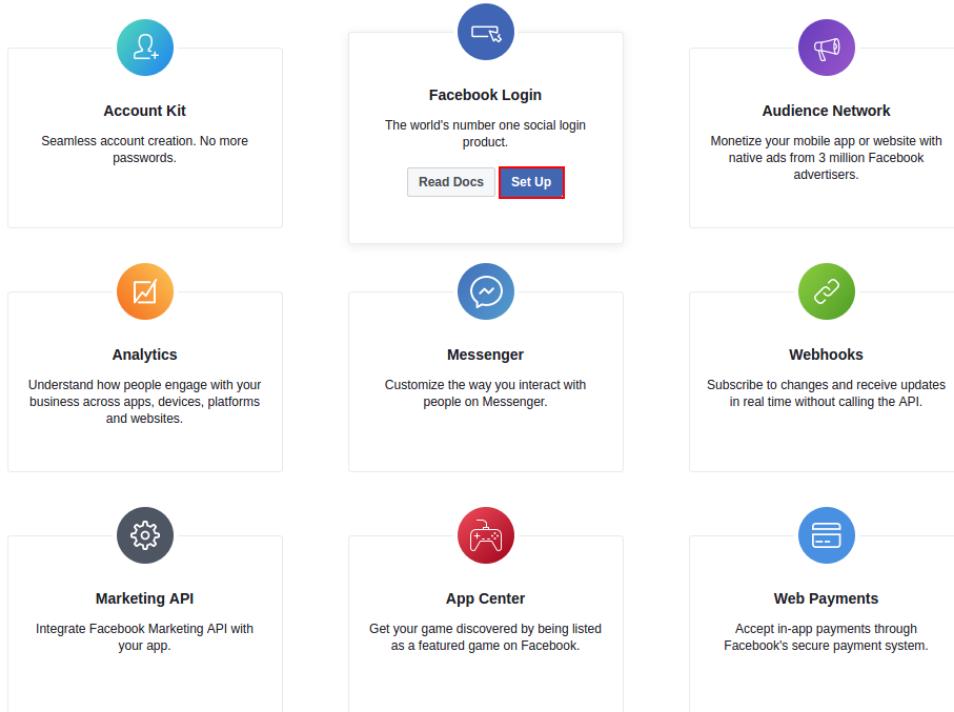
- Create a Facebook application
- Configure Facebook login with Identity Server
- Configuring requested claims for user authentication in Facebook Identity Provider
- Configure service providers to the Publisher and Store with the Facebook Identity Provider
- Test Facebook authentication
- Configure associate social login in IS dashboard

Note that the Facebook application development UI might be slightly different from the demonstrated UIs in this tutorial due to the frequent updates in the Facebook Developer Portal.

Create a Facebook application

1. Go to <https://developers.facebook.com/> and log in using your Facebook credentials.
2. Select **Apps** in the navigation and create a new app by clicking **Add a New App**.
3. Enter the name of your app and your email address. Click **Create App ID**.
4. Click **Set Up** of Facebook Login Product.

Select a product



5. Select **Website** here when working with this sample. You can select any other platform you wish to use

Add a New App

Select a platform to get started



iOS



Android



Facebook Canvas



Website

If you're developing on another platform or want to skip this step for now, use the [basic setup](#).

Change the port offset to 1 by modifying <Offset> element value in <Identity_server_home>/repository/conf/carbon.xml as following.

```
<Offset>1</Offset>
```

6. Add the serverURL of WSO2 Identity Server (which is configured with offset=1) <https://localhost:9444/> and

click Save and Continue.

If you have changed the hostname of identity server use that instead of localhost.

For example, if the host name is identity.com, then the server url is <https://identity.com:9444/>

iOS Android **Web** Other

1. Tell Us about Your Website

Tell us what the URL of your site is.

Site URL

`https://localhost:9444/`

Save

Continue

2. Set Up the Facebook SDK for Javascript

3. Check Login Status

4. Add the Facebook Login Button

5. Next Steps

7. Go to **Set Up the Facebook SDK for Javascript**. Click **Next**

8. Click Dashboard and go to the Developer Dashboard. You can find your App ID and the App Secret as shown in the image below.

Dashboard

OAuthTest

This app is in development mode and can only be used by app admins, developers and testers [?]

API Version [?] **v2.9**

App ID **1915528662057031**

App Secret **cd139c582be9aa32e57dc5411ec02d78**

Reset

Facebook Analytics

Set up Analytics

Analytics helps you grow your business and learn about the actions people take in your app. It only takes 5 minutes to set up.

Try Demo View Quickstart Guide

9. Go to **Settings** from the navigation bar. Select a Category. Add the correct website URL as shown below and click **Save Changes**.

App ID	App Secret
1915528662057031	***** Show
Display Name	Namespace
OAuthTest	
App Domains	Contact Email
localhost X	[REDACTED]
Privacy Policy URL	Terms of Service URL
Privacy policy for Login dialog and App Details	Terms of Service for Login dialog and App Details
App Icon (1024 x 1024)	Category Apps for Pages ▾
	
Website	Quick Start X
Site URL	https://localhost:9444/
+ Add Platform	

10. Click on the new **Facebook Login** product you have added and configure it as follows.

Client OAuth Settings

Client OAuth Login
Enables the standard OAuth client token flow. Secure your application and prevent abuse by locking down which token redirect URIs are allowed with the options below. Disable globally if not used. [?]

Web OAuth Login
Enables web based OAuth client login for building custom login flows. [?]

No **Force Web OAuth Reauthentication**
When on, prompts people to enter their Facebook password in order to log in on the web. [?]

No **Embedded Browser OAuth Login**
Enables browser control redirect uri for OAuth client login. [?]

Valid OAuth redirect URIs

No **Login from Devices**
Enables the OAuth client login flow for devices like a smart TV [?]

Deauthorize

Deauthorize Callback URL

Client	OAuth	Login	-	Yes
Web	OAuth	Login	-	Yes
Valid OAuth redirect URLs - https://localhost:9444/commonauth				

After the user authorizes the application, the authorization server redirects the user back to the application with access token or the authorization code in the URL. Since the redirected URL contains sensitive information, it is required to assure that the service does not redirect to arbitrary locations. The best way to ensure that the user is directed to the appropriate location is to define an **OAuth redirect URL** as shown above.

Now you have configure Facebook as your Identity Provider

Configure Facebook login with Identity Server

Let's see how to configure WSO2 Identity Server to work with Facebook for user authentication, so that when you try to login to the API Publisher or Store, WSO2 Identity Server will redirect to Facebook to do the authentication. As a prerequisite, you have to configure WSO2 Identity Server by [adding a new identity provider](#).

1. Download the WSO2 Identity Server [here](#) .
2. [Configure Single Sign On with WSO2 API Manager 2.1.0](#).
3. Log in to the [Management Console](#) of WSO2 Identity Server as an administrator.
4. Go to the **Identity** section under the Main tab. Click **Add** under **Identity Providers** and enter following details.

Identity provider Name	Alias
facebook	https://localhost:9444/oauth2/token

Add New Identity Provider

Basic Information

Identity Provider Name: [*]	<input type="text" value="facebook"/> <small> ⓘ Enter a unique name for this identity provider</small>
Display Name:	<input type="text"/>
Description:	<input type="text"/> <small> ⓘ A meaningful description about the identity provider</small>
Federation Hub Identity Provider:	<input checked="" type="checkbox"/> <small> ⓘ Check if this points to a federation hub identity provider</small>
Home Realm Identifier:	<input type="text"/> <small> ⓘ Enter the home realm identifier for this identity provider</small>
Identity Provider Public Certificate:	<input type="file"/> No file chosen <small> ⓘ Upload identity provider's public certificate in PEM format</small>
Alias:	<input type="text" value="https://localhost:9444/oauth2/token"/> <small> ⓘ If the resident identity provider is known by an alias at the federated identity provider specify it</small>

To authenticate the user with Facebook (External System) we have to configure the federated authenticator. For more details, see [Federated Authentication](#).

5. Go to **Facebook Configuration** under **Federated Authenticators**.
6. Enter the **Client ID** and **Client Secret** values obtained from the Facebook app created in the previous section
7. Select **Enable Facebook Authenticator** and select **Default** to make it the default authentication method.
8. Enter the User information fields you want to retrieve separated by commas under **User Information fields**.
9. Click **Register**.

The **Scope** defines the permission to access particular information from a Facebook profile. See the [Permissions Reference](#) for a list of the different permission groups in Facebook APIs.

Federated Authenticators

- SAML2 Web SSO Configuration
- OAuth2/OpenID Connect Configuration
- WS-Federation (Passive) Configuration
- Facebook Configuration**

Enable Facebook Authentication	<input type="checkbox"/> <small>Specifies if Facebook authentication is enabled for this identity provider</small>
Default	<input type="checkbox"/> <small>Specifies if Facebook authentication is the default</small>
Client Id: [*]	<input type="text" value="902543619862471"/>
	<small>Enter Facebook client identifier value</small>
Client Secret: [*]	<input type="text" value="*****"/> <small>Show</small>
	<small>Enter Facebook client secret value</small>
Scope:	<input type="text" value="email"/>
	<small>Enter a comma separated list of permissions to request from the user</small>
User Information Fields:	<input type="text" value="first_name,gender"/>
	<small>Enter comma-separated user information fields you want to retrieve</small>
Callback Url:	<input type="text" value="https://localhost:9443/commonauth"/>
	<small>Enter value corresponding to callback url</small>

Configuring requested claims for user authentication in Facebook Identity Provider

We need to acquire the identity information by configuring claims for use Authentication in facebook. Let's see how you can configure Identity Server with Facebook by mapping the claims. For more information on claim Mapping refer [Claim Management](#).

1. Go to the **Identity** section under the **Main** tab. Select **List** under **Identity Providers** .
2. Click **Edit** to edit the facebook identity provider you created.
3. Go to **Basic Claim Configuration** under **Claim Configuration**
4. Select the **Define Custom Claim Dialect** option under **Select Claim mapping Dialect** . Click **Add Claim Mapping** to add custom claim mappings as follows.

Add Identity Provider

Basic Information

Identity Provider Name: [*]	<input type="text" value="facebook"/>	<small>⑦ Enter a unique name for this identity provider</small>
Display Name:	<input type="text"/>	<small>⑦ Specify the identity provider's display name</small>
Description:	<input type="text"/>	<small>⑦ A meaningful description about the identity provider</small>
Federation Hub Identity Provider:	<input type="checkbox"/>	<small>⑦ Check if this points to a federation hub identity provider</small>
Home Realm Identifier:	<input type="text"/>	<small>⑦ Enter the home realm identifier for this identity provider</small>
Identity Provider Public Certificate:	<input type="button" value="Choose File"/> No file chosen	<small>⑦ Upload identity provider's public certificate in PEM format</small>
Alias:	<input type="text" value="https://localhost:9443/oauth2/token/"/>	<small>⑦ If the resident identity provider is known by an alias at the federated identity provider specify it</small>

Claim Configuration

Basic Claim Configuration

Select Claim mapping Dialect:	<input type="radio"/> Use Local Claim Dialect	<input checked="" type="radio"/> Define Custom Claim Dialect
Identity Provider Claim URIs:	Add Claim Mapping	
	<small>⑦ Add Claims URLs supported by Identity Provider</small>	
User ID Claim URI:	<input type="text" value="first_name"/>	<input type="text" value="http://wso2.org/claims/givenname"/>
	<small>⑦ Specifies the claim URI that identifies the user identifier at the identity provider</small>	

If you prefer to use the User ID as your first name of Facebook account, configure `first_name` claim as above. You need to select the same claim as **UserID Claim URI**.

- The following are some common attribute names. You can map these names to any suitable **Local Claim URI**. (Local Claim is a set of standard claim values which are local to the WSO2 Identity Server)

- `id`
- `email`
- `name`
- `first_name`
- `last_name`
- `link`
- `gender`
- `locale`
- `age_range`

For more information, see [Permissions Reference - Facebook Login](#).

Configure service providers to the Publisher and Store with the Facebook Identity Provider

To federate logging in to the Publisher and Store with Facebook, you need to configure the the service provider with the Facebook Identity Provider.

You have to allow the usage of email addresses as usernames, to use email addresses. To allow using email addresses as usernames, edit the `<IS_HOME>/repository/conf/carbon.xml` file. For details, see [Email Authentication](#). Addition to this, change the Realm configuration in `<API-M_HOME>/repository/conf/user-mgt.xml` for the `AdminUser` username to use the email attribute of the admin user like below.

```
<AdminUser>
  <UserName>admin@wso2.com</UserName>
  <Password>admin</Password>
</AdminUser>
```

1. Go to the Management console of WSO2 Identity Server (<https://localhost:9444/carbon>) and click on **Service Providers**.
2. Click **Edit** to edit the API_PUBLISHER.

Service Provider ID	Description	Actions
API_PUBLISHER		Edit Delete
API_STORE		Edit Delete

3. Go to the **Local and Outbound Authentication Configuration** section. Select the Identity Provider you created from the dropdown list under **Federated Authentication**.
4. Make sure that **Federated Authentication** is selected. Click **Update** to save the changes.

5. Repeat steps 1 to 4 and configure the API_STORE service provider.

Test Facebook authentication

1. Access the API Publisher via <https://localhost:<port-number>/publisher>. Observe the request redirect to the WSO2 IS SAML2.0 based SSO login page and then Facebook login page.
2. Enter the username and password of your Facebook account.

Facebook Login

Email or Phone:

Password:

Keep me logged in

Log In or [Sign up for Facebook](#)

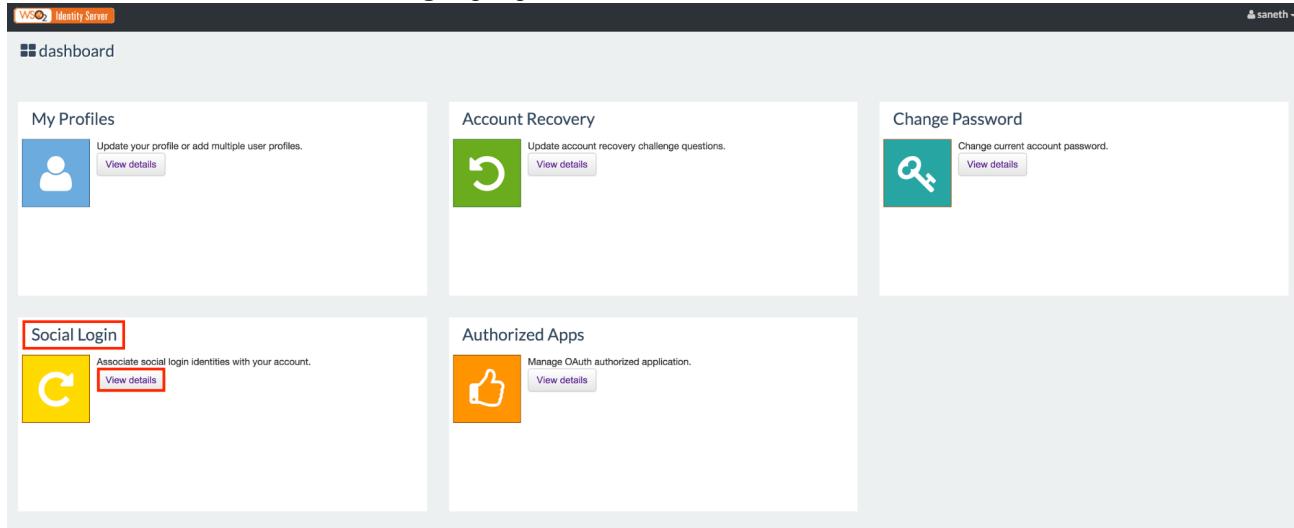
[Forgot your password?](#)

- After the login is authenticated successfully, you will be logged into API Publisher. Your username will be the first name of your Facebook account. This is because you have already configured the first name as the **User ID Claim URI**. If you configure your **User ID Claim URI** with `last_name`, your username will be the last name of your Facebook account.

Configure associate social login in IS dashboard

Identity Server has a dashboard which offers multiple options for users to maintain user accounts. Associating a social login for their account is one of the options provided in this dashboard. This dashboard can be accessed in the following url : `https://<IS_HOST>:<IS_PORT>/dashboard`. By association the social login you have the option to use local claims, instead of showing the logged name as facebook username you can use logged users as the username in user local user store

- Login to the dashboard with API Store user account.
- Click **View Details** in the **Social Login** gadget.



- Click **Associate Social Login** to give your facebook account details.

Social Login

Social Login	Identity Provider	Action
https://localhost:9443/openid/saneth	Primary OpenID	

4. Enter your IDP ID (facebook) and your username (as configured in **Subject Claim URI**) and click **Register**.

IDP ID*

User Name*

facebook

Saneth

Register Cancel

5. Select **Local & Outbound Configuration** and check **Assert identity using mapped local subject identifier**.

Local & Outbound Authentication Configuration

Authentication Type:*

Default

Local Authentication

Federated Authentication

Advanced Configuration

Assert identity using mapped local subject identifier

Always send back the authenticated list of identity providers

basic

facebook

After logging in to API Publisher, you will see the configured local claim appearing as your username.

You have now successfully logged in to the API Publisher using your facebook credentials.

Test an API using a Testing Tool

When an enterprise exposes its APIs for internal or external consumption, application developers (both internal and external) write applications using these exposed APIs. Before the API is embedded within an application, it needs to be tested in order to make sure that the API can be successfully adopted. SmartBear's Ready API! is an API testing tool widely used for this purpose. The WSO2 API Manager plugin, developed in partnership with SmartBear, allows seamless integration between the two products allowing application developers to work with the Ready! API platform to test APIs exposed via WSO2 API Manager.

This tutorial explains how to integrate Ready! API with WSO2 API Manager and then test APIs that are exposed in the API Manager. It also explains how Ready! API can be used to generate OAuth 2.0 tokens with different grant

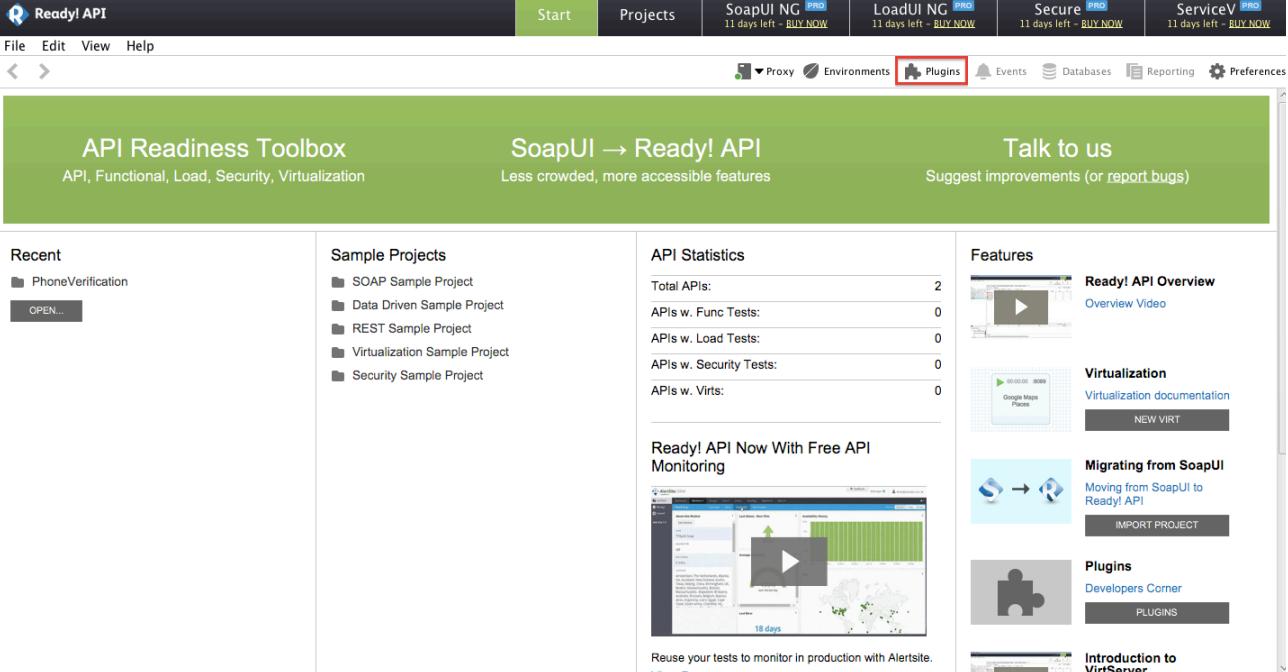
types and the benefits this integration can bring to an application developer.

This tutorial uses the PhoneVerification API, which is created in section [Create and Publish an API](#).

Installing the WSO2 API Manager plugin in Ready! API

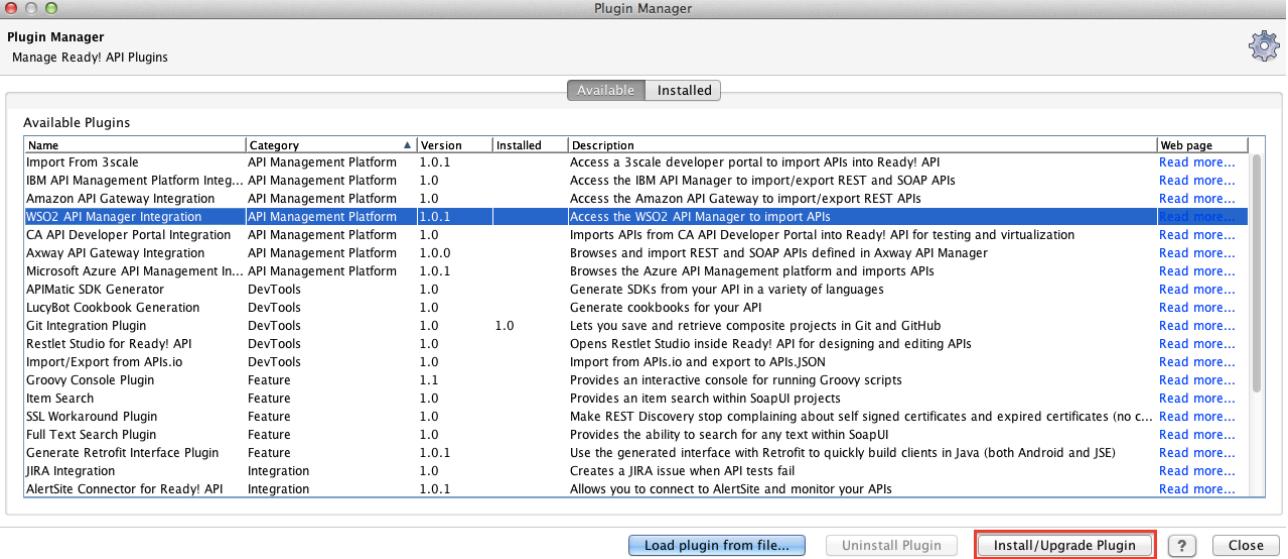
Ready! API supports WSO2 API Manager versions 1.8.0 onwards. Ready! API 1.7.0 has been used in this guide.

1. Download and run Ready! API.
2. Click the **Plugins** button shown below to open the the Plugin Manager.



The screenshot shows the Ready! API interface with a green header bar. The top navigation bar includes tabs for Start, Projects, SoapUI NG PRO (with a 'BUY NOW' button), LoadUI NG PRO (with a 'BUY NOW' button), Secure PRO (with a 'BUY NOW' button), and ServiceV PRO (with a 'BUY NOW' button). Below the header, there are several buttons: File, Edit, View, Help, Proxy, Environments, Plugins (which is highlighted with a red box), Events, Databases, Reporting, and Preferences. The main content area has three sections: 'API Readiness Toolbox' (with sub-sections API, Functional, Load, Security, Virtualization), 'SoapUI → Ready! API' (described as 'Less crowded, more accessible features'), and 'Talk to us' (with a link to 'Suggest improvements (or report bugs)'). On the left, there's a 'Recent' sidebar with a single entry: 'PhoneVerification' with an 'OPEN...' button. The central part contains sections for 'Sample Projects' (SOAP Sample Project, Data Driven Sample Project, REST Sample Project, Virtualization Sample Project, Security Sample Project) and 'API Statistics' (Total APIs: 2, APIs w. Func Tests: 0, APIs w. Load Tests: 0, APIs w. Security Tests: 0, APIs w. Virts: 0). A banner at the bottom of this section promotes 'Ready! API Now With Free API Monitoring' with a video thumbnail. To the right, there's a 'Features' sidebar with links to 'Ready! API Overview', 'Virtualization documentation', 'Migrating from SoapUI', 'Plugins', and 'Introduction to VirServer'.

3. Browse the plugin repository, locate the WSO2 API Manager plugin for Ready! API and click **Install/Upgrade Plugin**.



The screenshot shows the Ready! API Plugin Manager interface. The title bar says 'Plugin Manager' and 'Manage Ready! API Plugins'. Below the title bar, there are tabs for 'Available' and 'Installed', with 'Available' currently selected. The main table lists available plugins under the heading 'Available Plugins'. The columns are Name, Category, Version, Installed, Description, and Web page. One plugin, 'WSO2 API Manager Integration', is highlighted with a red box. At the bottom of the table are buttons for 'Load plugin from file...', 'Uninstall Plugin', 'Install/Upgrade Plugin' (which is highlighted with a red box), and 'Close'.

Name	Category	Version	Installed	Description	Web page
Import From 3scale	API Management Platform	1.0.1		Access a 3scale developer portal to import APIs into Ready! API	Read more...
IBM API Management Platform Integ...	API Management Platform	1.0		Access the IBM API Manager to import/export REST and SOAP APIs	Read more...
Amazon API Gateway Integration	API Management Platform	1.0		Access the Amazon API Gateway to import/export REST APIs	Read more...
WSO2 API Manager Integration	API Management Platform	1.0.1	1.0	Access the WSO2 API Manager to import APIs	Read more...
CA API Developer Portal Integration	API Management Platform	1.0		Imports APIs from CA API Developer Portal into Ready! API for testing and virtualization	Read more...
Axway API Gateway Integration	API Management Platform	1.0.0		Browses and import REST and SOAP APIs defined in Axway API Manager	Read more...
Microsoft Azure API Management Inte...	API Management Platform	1.0.1		Browses the Azure API Management platform and imports APIs	Read more...
APIMatic SDK Generator	DevTools	1.0		Generate SDKs from your API in a variety of languages	Read more...
LucyBot Cookbook Generation	DevTools	1.0		Generate cookbooks for your API	Read more...
Git Integration Plugin	DevTools	1.0		Lets you save and retrieve composite projects in Git and GitHub	Read more...
Restlet Studio for Ready! API	DevTools	1.0		Opens Restlet Studio inside Ready! API for designing and editing APIs	Read more...
Import/Export from APIs.io	DevTools	1.0		Import from APIs.io and export to APIs.json	Read more...
Groovy Console Plugin	Feature	1.1		Provides an interactive console for running Groovy scripts	Read more...
Item Search	Feature	1.0		Provides an item search within SoapUI projects	Read more...
SSL Workaround Plugin	Feature	1.0		Make REST Discovery stop complaining about self signed certificates and expired certificates (no .crt files)	Read more...
Full Text Search Plugin	Feature	1.0		Provides the ability to search for any text within SoapUI	Read more...
Generate Retrofit Interface Plugin	Feature	1.0.1		Use the generated interface with Retrofit to quickly build clients in Java (both Android and JSE)	Read more...
JIRA Integration	Integration	1.0		Creates a JIRA issue when API tests fail	Read more...
AlertSite Connector for Ready! API	Integration	1.0.1		Allows you to connect to AlertSite and monitor your APIs	Read more...

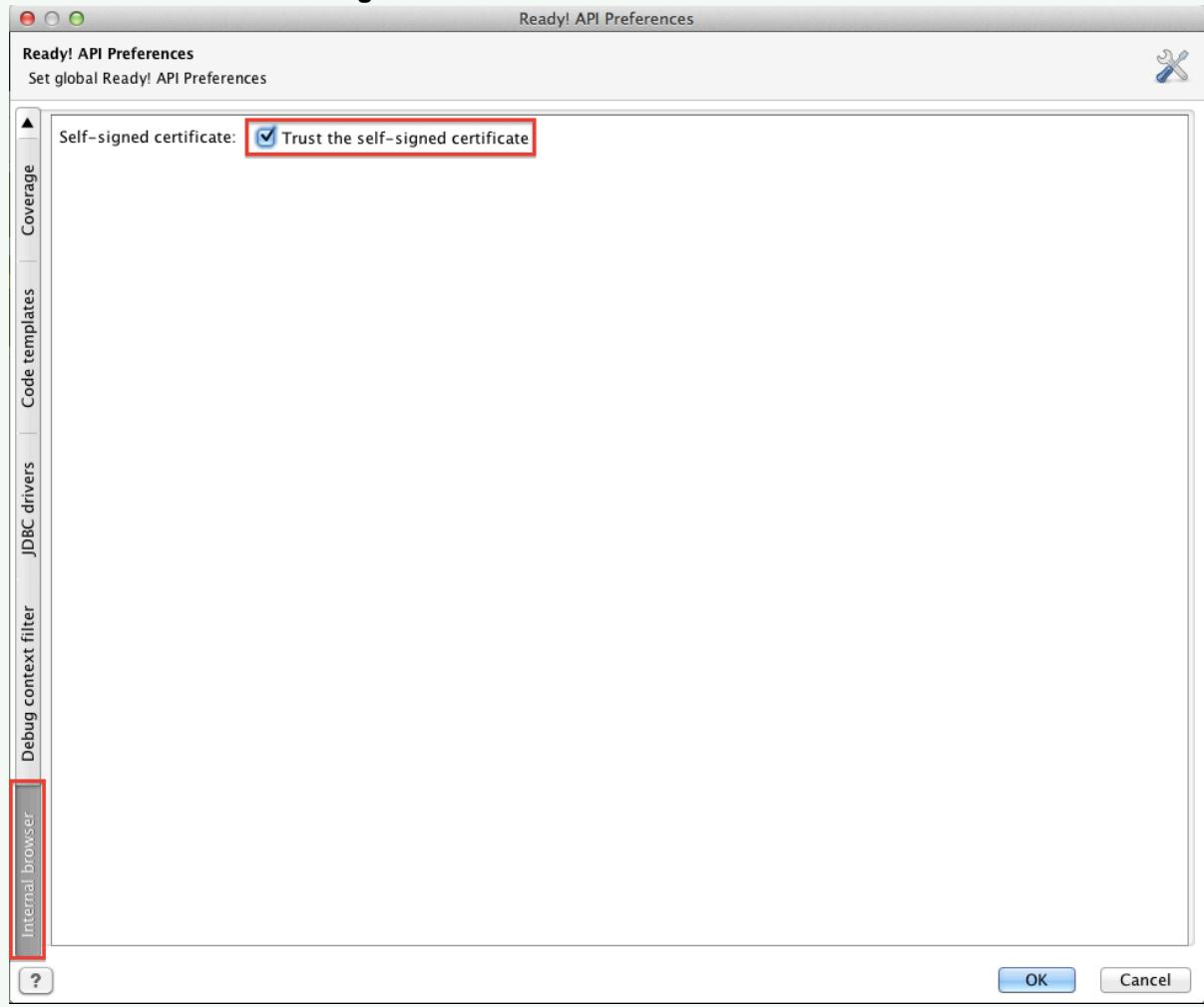
4. Click **Yes** on the confirmation message that appears.

Let's test an API exposed via WSO2 API Manager.

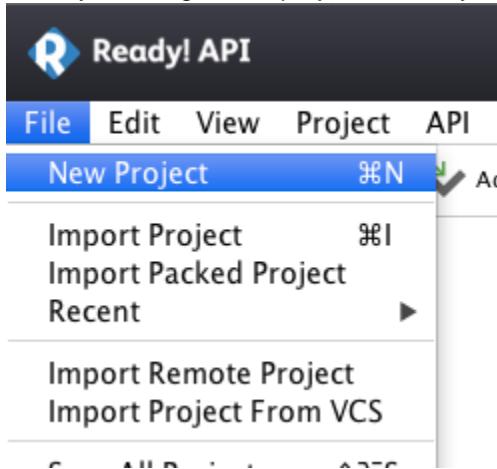
Testing APIs using Ready! API

1. Run the WSO2 API Manager server.
2. Run Ready! API if it's not already open.

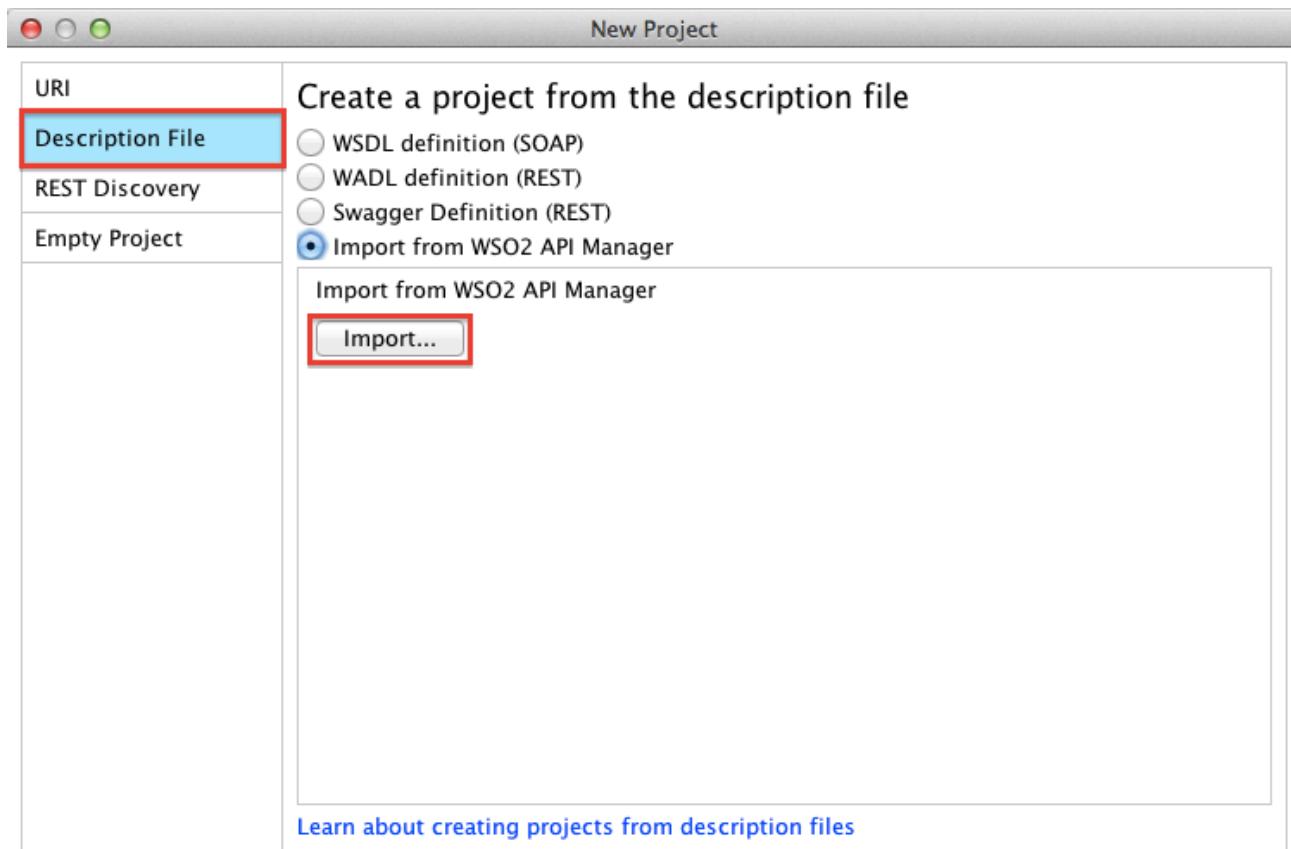
If you are using a self-signed certificate with WSO2 API Manager, you need to explicitly configure Ready! API to trust this certificate. Click **File > Preferences** and then select the **Internal Browser** tab. Select the **Trust the self-signed certificate** check box and click **OK**.



3. Start by creating a new project in Ready! API. Click **File > New Project**.



4. In the **Description File** tab, select the **Import from WSO2 API Manager** option and click **Import**.



[Learn about creating projects](#)

[OK](#)

[Cancel](#)

5. On the dialog box that appears, enter the information of the API Store from which API information needs to be extracted and click **OK**.

It is possible to import APIs from tenant stores as well.

Project Name	API Manager Project
API Store URL	https://localhost:9443/store/
API Store User Name	admin
API Store Password	admin
API Manager Version	2.0.0

Create Project From API Specification on WSO2 API Store

Creates a new Project from API specification on WSO2 API Store in this workspace

Project Name:	API Manager Project
API Store URL:	https://localhost:9443/store/
API Store User Name:	admin
API Store password:	*****
Tenant Domain:	
API Manager Version:	2.0.0

OK **Cancel**

6. Select one or more APIs from the list of APIs available in the API Store and click **OK**.

New Project

URI Create a project from the description file

Select API to Import

Select API to Import
Please select from the list which API specification(s) you want to import to the project.

Api List:	Name	Version	Provider	Description
PhoneVerification	1.0.0	admin		
PhoneVerification	2.0.0	admin		
PizzaShackAPI	1.0.0	admin	This is a simple API for...	
StockQuote	1.0.0	admin	StockQuote SOAP service	

Generate Test Suite: Yes
 No

Generate Load test: Yes
 No

OK **Cancel**

Your project is created and the required APIs are imported to Ready! API.

7. Select an API (e.g. PhoneVerification) and expand the API to see all the HTTP verbs associated with it.
Select the require HTTP verb (e.g. GET) to test the API.

The screenshot shows the WSO2 API Manager interface. On the left, there's a tree view of projects and APIs. A specific GET request under the PhoneVerification API is selected and highlighted with a red box. On the right, the 'Request' tab is active, displaying a table of parameters:

Name	Value	Style	Level
PhoneNumber		QUERY	METHOD
LicenseKey		QUERY	METHOD

Any query parameters that were added when creating the API in the API Manager appear here. You can also add your own resource path or query parameters to test the API.

- To add your own parameters, click the **Parameters** field and then click the **Plus** icon.

The screenshot shows the 'Parameters' dialog box. It has a table with one row:

Name	Value
PhoneNumber	

At the bottom left, there is a blue 'Add' button, which is highlighted with a red box. At the bottom right, there is a 'Close' button.

- Enter the required information for the API (e.g. 18006785432 as the phone number).
- Once all the required information is added for the API, you need to add the API OAuth 2.0 token to invoke the

API. Click the **Auth** tab at the bottom of the screen.

GET – default_request

Method: GET Endpoint: https://...:8243 Resource: /phoneverify/1.0.0/CheckPhone

Request

Name	Value	Style	Level
PhoneNumber	18006785432	QUERY	METHOD
LicenseKey		QUERY	METHOD

Required: Sets if parameter is required

Authorization: wso2-api-manager-default

Access Token:
Enter an existing access token or click "Get Access Token" below.
▼ Get Access Token

[Learn about authorization](#)

Auth (wso2-api-manager-default) Headers (0) Attachments (0) Representations (1) JMS H ▲ ▶

You can either get a test access token from the API Store or use the inbuilt OAuth 2.0 access token generation option. In this example, the inbuilt token generation option is used.

The inbuilt OAuth 2.0 access token generation option allows an access token to be generated with different grant types, which can be used to test the key generation process of WSO2 API Manager without requiring an application to perform the OAuth 2.0 key generation. This option can also be used in cases where you want to test access to different HTTP verbs and resource paths using different types of scopes and API keys.

11. In the **Auth** tab, click **Get Access Token**.

GET – default_request

Method: GET Endpoint: https://[REDACTED]:8243 Resource: /phonverify/1.0.0/CheckPh

Request

Name	Value	Style	Level
PhoneNumber	18006785432	QUERY	METHOD
LicenseKey		QUERY	METHOD

Required: Sets if parameter is required

Authorization: wso2-api-manager-default

Access Token:
Enter an existing access token or click "Get Access Token" below.
▼ Get Access Token

[Learn about authorization](#)

Auth (wso2-api-manager-default) Headers (0) Attachments (0) Representations (1) JMS H ▶ ▷

- On the dialog box that appears, enter the following information.

OAuth 2 Flow - You can choose from different grant types to generate your token. In this example, we have used the **Client Credentials Grant** type.

Client Identification and Client Secret - Get these values from the API Store. Browse to the application that the API being tested is subscribed to (e.g. DefaultApplication) and copy the values from the **Production Keys** tab.

DefaultApplication

Application

Details Production Keys Sandbox Keys Subscriptions SDKs

Consumer Key

FdvaoluAfCmwsXXNZLsUcBi3XKQa

Consumer Secret

utbMPpY0ffFrstEfmlQm4Yke8YfUa

Access Token URI - Provide the URL of the access token endpoint of the API Manager. By default, this URL is <https://localhost:9443/oauth2/token>. If you are using a componentized API manager

deployment, the URI should point to the Key Manager component of the deployment.

Scope - Define the scope under which a token should be generated. If you have not defined any scope restrictions when creating the API you can leave this blank.

13. Once done, click **Get Access Token**.

Get access token from the authorization server

OAuth 2 Flow: Client Credentials Grant

Client Identification: FdvaoluAfCmwsXXNZLsUcBi3XKQa

Client Secret: utbMPpY0ffFrstEfmlQm4Yke8YfUa

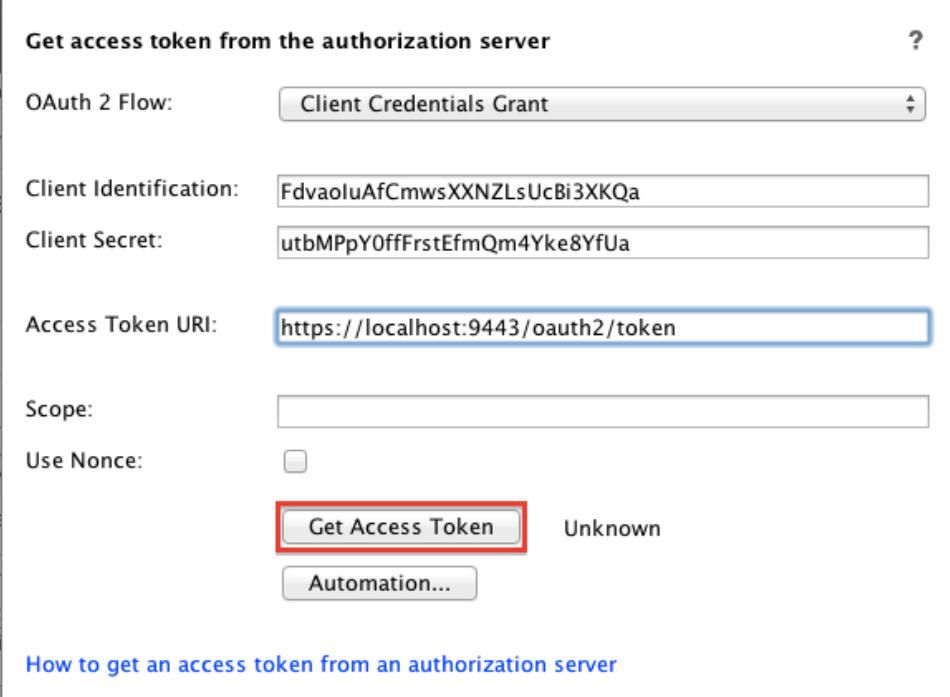
Access Token URI: <https://localhost:9443/oauth2/token>

Scope:

UseNonce:

Get Access Token Unknown
Automation...

[How to get an access token from an authorization server](#)



14. The access token is retrieved from the server.

Get access token from the authorization server

OAuth 2 Flow: Client Credentials Grant

Client Identification: FdvaoluAfCmwsXXNzLsUcBi3XKQa

Client Secret: utbMPpY0ffFrstEfMqM4Yke8YfUa

Access Token URI: https://localhost:9443/oauth2/token

Scope:

UseNonce:

Get Access Token **✓ Retrieved from server**

Automation...

How to get an access token from an authorization server

Access Token: **69896c7d970580d5f2fe3992de09a3bd** ✓ Retrieved from server

Enter an existing access token or click "Get Access Token" below.

▲ Get Access Token

Learn about authorization

Auth (wso2-api-manager-default) Headers (0) Attachments (0) Representations (1) JMS

15. Now you can invoke the API by sending the request. The response is displayed as shown below. If you encounter an error, make sure that the values given for the endpoint, resource and parameters are correct.

GET – default_request

Method: GET Endpoint: https://...:8243 Resource: /phoneverify/1.0.0/CheckPhoneNumber Parameters: ?PhoneNumber=18006785432

Request

Name	Value	Style	Level
PhoneNumber	18006785432	QUERY	METHOD
LicenseKey		QUERY	METHOD

Required: Sets if parameter is required

Authorization: wso2-api-manager-default

Access Token: **69896c7d970580d5f2fe3992de09a3bd** ✓ Retrieved from server

Enter an existing access token or click "Get Access Token" below.

▼ Get Access Token

Response

XML Node	Value
PhoneReturn	Toll Free
Company	true
Valid	Assigned to a code holder...
Use	TF
State	
RC	
OCN	
OriginalNumber	18006785432
CleanNumber	8006785432
SwitchName	
SwitchType	
Country	United States
CLLI	
PrefixType	Landline

Overview

Headers (13) Attachments (0) SSL Info (1 certs) Representations (7)

You have successfully tested an API.

To use the access token taken from the API Store,

- Login to the WSO2 API Manager Store and browse to the application that the API being tested is subscribed to (e.g. DefaultApplication).

- b. In the **Production** Keys tab, generate (or regenerate) a test access token and copy it.

Generate a Test Access Token

Access Token

ea13122e1b1b92310fa44c5b89997aa5



Above token has a validity period of **3600** seconds. And the token has (**am_application_scope,default**) scopes.

Scopes	Select..	Validity period	3600
Seconds.	C Re-generate		

- c. Go back to the Ready! API, paste the access token for the API and send the request.

GET - default_request

Method	Endpoint	Resource
GET	https://[REDACTED]:8243	/phonereverify/1.0.0/CheckPhc

Submit request to specified endpoint URL (Alt-Ctrl-Enter)

Request

Raw	+ X [REDACTED] ?												
<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> <th>Style</th> <th>Level</th> </tr> </thead> <tbody> <tr> <td>PhoneNumber</td> <td>18006785432</td> <td>QUERY</td> <td>METHOD</td> </tr> <tr> <td>LicenseKey</td> <td></td> <td>QUERY</td> <td>METHOD</td> </tr> </tbody> </table>		Name	Value	Style	Level	PhoneNumber	18006785432	QUERY	METHOD	LicenseKey		QUERY	METHOD
Name	Value	Style	Level										
PhoneNumber	18006785432	QUERY	METHOD										
LicenseKey		QUERY	METHOD										

Required: Sets if parameter is required

Authorization: wso2-api-manager-default

Access Token: **ea13122e1b1b92310fa44c5b89997aa5**

Enter an existing access token or click "Get Access Token" below.

▼ Get Access Token

Learn about authorization

Auth (wso2-api-manager-default) Headers (0) Attachments (0) Representations (1) JMS [REDACTED]

- d. The response is displayed as shown below. If you encounter an error, make sure that the values given for the endpoint, resource and parameters are correct.

The screenshot shows the WSO2 API Manager interface. At the top, it displays a 'GET - default_request' with the method set to 'GET', endpoint as 'https://[redacted]:8243', resource as '/phoneverify/1.0.0/CheckPhoneNumber', and parameters including '?PhoneNumber=18006785432'. The 'Request' tab shows a table with columns 'Name', 'Value', 'Style', and 'Level'. It contains two rows: 'PhoneNumber' with value '18006785432' and style 'QUERY', and 'LicenseKey' with value '' and style 'METHOD'. Below this is a section for 'Authorization' with a dropdown set to 'wso2-api-manager-default' and an input field containing '69896c7d970580d5f2fe3992de09a3bd'. A note says 'Enter an existing access token or click "Get Access Token" below.' followed by a 'Get Access Token' button. The 'Response' tab shows XML, JSON, and HTML representations. The XML representation is highlighted with a red box and contains nodes like PhoneReturn, Company, Valid, Use, State, RC, OCN, OriginalNumber, CleanNumber, SwitchName, SwitchType, Country, CLLI, and PrefixType. The JSON representation is also visible. At the bottom, there are tabs for 'Overview', 'Raw', 'Outline', and 'Headers (13)', 'Attachments (0)', 'SSL Info (1 certs)', and 'Representations (7)'.

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

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 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:/phoneverify 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 each API 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

Version: 1.0.0
By: admin
Updated: 30/Jul/2016 11:05:37 AM IST
Status: PUBLISHED
Rating: ★★★★★

Overview API Console Documentation Forum

Production and Sandbox Endpoints
Production and Sandbox URLs:

- [https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)
- [http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

Share

Social Sites Embed Email

f t g+ digg

Comments

Characters left: 450

Add

No comments yet

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

Comments

Characters left: 450

Add

comment1

Posted By admin on July 30, 2016 11:13:58 AM GMT+05:30

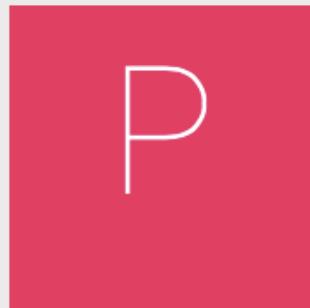
comment2

Posted By admin on July 30, 2016 11:14:05 AM GMT+05:30

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 advertise APIs.

PhoneVerification - 1.0.0



Version: 1.0.0
By: admin
Updated: 30/Jul/2016 11:05:37 AM IST
Status: PUBLISHED
Rating: 5

Overview API Console Documentation Forum

Production and Sandbox Endpoints

Production and Sandbox URLs:

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

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

Share

Social Sites Embed Email



Comments

Characters left: 450

Add

No comments yet

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

The screenshot shows the 'Overview' tab selected for the 'PhoneVerification - 1.0.0' API. The page displays the following details:

- Version:** 1.0.0
- By:** admin
- Updated:** 30/Jul/2016 11:05:37 AM IST
- Status:** PUBLISHED
- Rating:** ★★★★☆ (4 stars)

On the right side, there are dropdown menus for **Applications** (set to DefaultApplication) and **Tiers** (set to Unlimited), and a **Subscribe** button.

Below the details, there are links for **Overview**, **API Console**, **Documentation**, and **Forum**. The **Overview** link is highlighted with a red border.

Production and Sandbox Endpoints

Production and Sandbox URLs:

- [https://\[REDACTED\]:8243/phoneverify/1.0.0](https://[REDACTED]:8243/phoneverify/1.0.0)
- [http://\[REDACTED\]:8280/phoneverify/1.0.0](http://[REDACTED]:8280/phoneverify/1.0.0)

Share

Social Sites | **Embed** | Email

`<iframe width="450" height="120" src="https://10.100.5.34:9443/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 to go to the forum, where you can initiate conversations and share your opinions with other users.

PhoneVerification - 1.0.0

The screenshot shows the **Forum** tab selected for the 'PhoneVerification - 1.0.0' API. The page displays the following details:

- Version:** 1.0.0
- By:** admin
- Updated:** 30/Jul/2016 11:05:37 AM IST
- Status:** PUBLISHED
- Rating:** ★★★★☆ (4 stars)

On the right side, there are dropdown menus for **Applications** (set to DefaultApplication) and **Tiers** (set to Unlimited), and a **Subscribe** button.

Below the details, there are links for **Overview**, **API Console**, **Documentation**, and **Forum**. The **Forum** link is highlighted with a red border.

Search Forum | **Q Search** | **Create New Topic**

No topics found.
There are no forum topics available.

Write a Client Application Using the SDK

A software development kit (SDK) is a set of software development tools that allows to create applications for a specific platform. If an API consumer wants to create an application, they can generate a client side SDK for a supported language/framework and use it to write a software application to consume the subscribed APIs. This tutorial shows you how to write a client application using an SDK.

In this example, we use the sample API in WSO2 API Manager as a demonstration. To deploy the sample API, log in to the API Publisher and click the **Deploy Sample API** button. Note that the button only appears if no APIs have been created so far, in the given tenant space.

If a different API is used, the SDK functions to invoke the API are based on the specifications of that API.

- Follow the steps in [Invoke your first API](#), to deploy the sample API, subscribe and generate keys.

Access Token

Once the keys are generated, copy the access token. You can use this token to invoke APIs that you subscribe to using the same application.

- Go to the API Store. Select your API and download the SDK for Java. For more details, see [Generating client SDKs in the API Store](#).

PizzaShackAPI - 1.0.0



Version: 1.0.0

By: Jane Roe

Updated: 06/Oct/2016 20:23:

Status: PUBLISHED

Rating: ★★★★★ ×

Overview API Console Documentation Forum **SDKs**

Download client-side SDKs for this API.

[java](#)

[android](#)

- In this example, you would have downloaded the `PizzaShackAPI_1.0.0_java.zip` file. This file name includes the API name, version, and language of the SDK. Unzip the `PizzaShackAPI_1.0.0_java.zip` file.

▼ [Expand to see the folder structure of the unzipped file...](#)

```
PizzaShackAPI_1.0.0_java
build.gradle
build.sbt
docs
  DefaultApi.md
  ErrorListItem.md
  Error.md
  MenuItem.md
  Order.md
```

```
git_push.sh
gradle
    wrapper
        gradle-wrapper.jar
        gradle-wrapper.properties
gradle.properties
gradlew
gradlew.bat
LICENSE
pom.xml
README.md
settings.gradle
src
    main
        AndroidManifest.xml
        java
            org
                wso2
                    client
                        api
                            ApiCallback.java
                            ApiClient.java
                            ApiException.java
                            ApiResponse.java
                            auth
                                ApiKeyAuth.java
                                Authentication.java
                                HttpBasicAuth.java
                                OAuthFlow.java
                                OAuth.java
                                Configuration.java
                                JSON.java
                                Pair.java
                                PizzaShackAPI
                                    DefaultApi.java
                                    ProgressRequestBody.java
                                    ProgressResponseBody.java
                                    StringUtil.java
                            model
                                PizzaShackAPI
                                    Error.java
                                    ErrorListItem.java
                                    MenuItem.java
                                    Order.java
test
    java
        org
            wso2
                client
```

```

api
  PizzaShackAPI
    DefaultApiTest.java

```

4. Build the SDK using maven.

When it's done, you can include this SDK as a dependency in your software project. Details of this maven dependency are included in the README.md file.

Maven dependency

```

<dependency>
  <groupId>org.wso2</groupId>
  <artifactId>org.wso2.client.PizzaShackAPI</artifactId>
  <version>1.0.0</version>
  <scope>compile</scope>
</dependency>

```

Build using maven

You can build the SDK using the `mvn clean install` command inside the root directory. For more information see [Maven Start Guide](#).

- After creating a maven project, import the following with respect to the SDK. These classes will be accessible from the code once the SDK is built using maven and will be included as maven dependencies in the project.

```

import org.wso2.client.api.ApiClient;
import org.wso2.client.api.PizzaShackAPI.DefaultApi;
import org.wso2.client.model.PizzaShackAPI.Menu;

```

- Create an instance of the `DefaultApi` object in the java code. This instance is needed to get the API client which handles the operations related to consuming the API, using the resources of the API.

```

DefaultApi defaultApi = new DefaultApi();

```

- The API client of the `DefaultApi` object instance is used to set HTTP request headers with the required data. Note that these HTTP request headers might differ from one API to another, depending on the implementation of the API. A sample is show below.

```

ApiClient apiClient = defaultApi.getApiClient();
apiClient.addDefaultHeader("Accept", "application/json");

```

- Include the access token as a header in the API client object, to invoke the API.

```
String accessToken = "bc392b16-6ce2-3208-9023-8938fbc376ea";
apiClient.addDefaultHeader("Authorization", "Bearer " + accessToken);
```

You need an access token to invoke the API. It is important to have a valid subscription before using the SDK, to obtain an access token. Note that the obtained access token has an [expiration time](#).

9. Set the base path to the API client.

```
apiClient.setBasePath("http://localhost:8280/pizzashack/1.0.0");
```

The base path for the client application is the production (or sandbox) URL of the API, found in the **Overview** tab of the API in the API Store.

The screenshot shows the 'API Console' tab selected in the navigation bar. Below it, the 'Production and Sandbox Endpoints' section displays two URLs: 'http://192.168.42.1:8280/pizzashack/1.0.0' and 'https://192.168.42.1:8243/pizzashack/1.0.0'. There are also sections for 'Description' (a simple API for Pizza Shack online pizza delivery store) and 'Business Information'.

10. Once the `ApiClient` object has all the required data, set the `ApiClient` for the instance of the `DefaultApi` object.

```
defaultApi.setApiClient(apiClient);
```

11. Finally, we can call the available function in the SDK to get the response from the API.

```
List<MenuItem> menuItems = (List<MenuItem>) defaultApi.menuGet();
```

`MenuItem` is a model class generated with SDK

Complete java code can be found below

[Java Codepom file](#)

```
import org.wso2.client.api.ApiClient;
import org.wso2.client.api.ApiException;
import org.wso2.client.api.PizzaShackAPI.DefaultApi;
import org.wso2.client.model.PizzaShackAPI.MenuItem;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class APIClient {

    public static void main(String[] args) throws ApiException {
        DefaultApi defaultApi = new DefaultApi();
        String accessToken = "bc392b16-6ce2-3208-9023-8938fbc376ea";
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Accept", "application/json");
        headers.put("Authorization", "Bearer " + accessToken);
        ApiClient apiClient = defaultApi.getApiClient();
        apiClient.addDefaultHeader("Accept", "application/json");

        apiClient.addDefaultHeader("Authorization", "Bearer " +
        accessToken);
        apiClient.setLenientOnJson(true);
        apiClient.setBasePath("http://localhost:8280/pizzashack/1.0.0");
        defaultApi.setApiClient(apiClient);
        List<MenuItem> menuItems = (List<MenuItem>) defaultApi.menuGet();

        System.out.println(menuItems);
    }
}
```

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.pizzashack.client</groupId>
  <artifactId>pizzashack-api</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.wso2</groupId>
      <artifactId>org.wso2.client.PizzaShackAPI</artifactId>
      <version>1.0.0</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>

</project>

```

Obtaining User Profile Information with OpenID Connect

OpenID Connect is an **authentication protocol** that is a simple identity layer on top of the OAuth 2.0 protocol. It allows clients to verify the identity of the end-user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner.

You can use WSO2 API Manager to obtain basic profile information about the user who generates the access token. To obtain this information, the `openid` scope needs to be passed, when generating the access token. API manager will send a JWT which contains information about the user who is generating the token, as part of the response for this request. You can configure the information returned with the JWT token.

Follow the instructions below to obtain user profile information with OpenID connect with WSO2 API Manager.

1. Obtain a token using password grant type and `openid` scope. For more information on token generation with password grant type, see [Password Grant Type](#). The format of the cURL command and a sample is given below :

[Format Sample](#)

```

curl -k -d
"grant_type=password&username=<USERNAME>&password=<PASSWORD>&scope=op
enid" -H "Authorization: Basic <BASE64 ENCODED
CONSUMER_KEY:CONSUMER_SECRET>, Content-Type:
application/x-www-form-urlencoded"
https://<GATEWAY_HOSTNAME>:<PORT>/token

```

```
curl -k -d
"grant_type=password&username=testuser&password=testuserpassword&scope=openid" -H "Authorization: Basic
M1J6RFNrRFI5ZmQ5czRqY296R2xfVjh0QU5JYTpXeElqSkFJd0dqRWVYOHDHZGFFcGM1W
194RjRh, Content-Type: application/x-www-form-urlencoded"
https://apim.wso2.com:8243/token
```

You will receive a response in the fowmat shown below. Note that the `id_token` parameter contains the JWT related to user information.

```
{
  "access_token": "83705add-d77e-3cc8-9b6a-53d210ed3fed",
  "refresh_token": "4b283fb8-942f-316d-ba90-44b4c76ae419",
  "scope": "openid",
  "id_token": "eyJ4NXQiOijObUptT0dVeE16WmxZak0yWkRSaE5UWmxZVEExWXpkaFpUUmlPV0UwTldJMk0ySm1PVGMxWkEiLCJraWQiOjkMGVjNTE0YTMyYjZmODhjMGFiZDEyYTI4NDA2OTliZGQzZGViYTkIiwiYWxnIjoiUlMyNTYifQ.eyJhdF9oYXNoIjoiY1hoV012SXdSYlBnVDBBTG1hekpiUSIsImFjciI6InVybjptYWNlOmluY29tbW9uOmlhcDpzaWx2ZxiILCJzdWIiOijzdWJzY3JpYmVyQGNhcmJvbi5zdXBlcIIsImF1ZCI6WyJLb05EbGVTckYzbmFYV3doYXzhbzRiQm9NWWNhI10sImF6cCI6IkTvTkRsZVNrjNuYVhXd2hdmFvNGJCb01ZY2EiLCJvcmdhbml6YXRpb24iOijXU08yIiwickXNzIjoiHR0cHM6XC9cLZE3Mi4xNi4yLjExMT05NDQzXC9vYXV0aDJcL3Rva2VuIiwiZXhwIjoxNTEzOTUwNDEzLCJpYXQiOjE1MTE5NDY4MTMsImVtYWlsIjoiic3ViMUBnbWFpbC5jb20ifQ.gdj0jn4PX5R4j5Y0ZNyEwi2G-NPq3_iw89NqkRxeszdcMLvDP-ncRWMaYyUYC-bQqADekTdQUC6ACSVUlJBKau3Oy8uu-AO8paJIm-hWEX_PBqoMRtFztxggmKFaL6G0rdRBIu8Lzl5lbX2cTKss_zYwNmcpDsKDwdQDmL089Wg",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

- The following two options are available to view the actual user information.

- Decoding the `id_token`
- Invoking the `userinfo` endpoint

Decoding the `id_token`

By decoding the `id_token`, a payload similar to the following can be obtained, with user information such as email, organization, etc.

```
{
  "at_hash": "cXhWIVIWbPgT0ALmazJHQ",
  "acr": "urn:mace:incommon:iap:silver",
  "sub": "user1@carbon.super",
  "aud": [
    "KoNDleSrF3naXWwhavao4bBoMYca"
  ],
  "azp": "KoNDleSrF3naXWwhavao4bBoMYca",
  "organization": "WSO2",
  "iss": "https://172.16.2.111:9443/oauth2/token",
  "exp": 1511950413,
  "iat": 1511946813,
  "email": "user1@gmail.com"
}
```

For an online tool to decode the JWT, go to <https://jwt.io/>

Invoking the userinfo endpoint

You can obtain user information as a payload by invoking the userinfo endpoint with the access token obtained in step 1. The format of the cURL command and a sample is given below

FormatSample

```
curl -k -v -H "Authorization: Bearer <ACCESS_TOKEN>"  
https://<GATEWAY_HOSTNAME>:<PORT>/userinfo
```

```
curl -k -v -H "Authorization: Bearer 83705add-d77e-3cc8-9b6a-53d210ed3fed"  
https://apim.wso2.com:8243/userinfo
```

The response will be a JSON payload as shown below:

```
{
  "sub": "user1@carbon.super",
  "organization": "WSO2",
  "email": "user1@gmail.com"
}
```

By default, only the username (sub) information will be available in the response. You can customize the user information returned by configuring the claims of the relevant Service Provider generated for the Application created in Store. For more information, see [Service Provider Claim Configuration](#).

Cleaning Up Partially Created Keys

An application created in WSO2 API Manager has a corresponding OAuth application in the Key Manager side. There can be occasions where applications are created/deleted partially, where the OAuth application is successfully created/deleted but there is stale data left on the API Manager side. This can happen due to network failures between the API Manager and the Key Manager nodes, partial deletion of applications, etc. In such situations, when a user navigates to the **Production Keys** tab, a **Clean up** button is visible, which allows you to, for instance, delete the remaining application data from the API Manager side.

The screenshot shows a user interface for managing OAuth keys. At the top, there is a navigation bar with four tabs: 'Details' (selected), 'Production Keys' (highlighted in orange), 'Sandbox Keys', and 'Subscriptions'. Below the tabs, a message box displays the text: 'Error! You have partially created keys. Please click the Clean Up button and try again.' To the right of this message is a blue rectangular button labeled 'Clean up'. The rest of the page is mostly blank white space.

Deep Dive

This section provides information about the features, functionality, solution development, testing and debugging options of WSO2 API Manager.

- [Installation Guide](#)
- [WSO2 API Manager Tooling](#)
- [Product Administration](#)
- [Configuring the API Manager](#)
- [Extending the API Manager](#)
- [Working with Security](#)
- [Working with Throttling](#)
- [Working with Endpoints](#)
- [Analytics](#)
- [Reference Guide](#)
- [Developer Guide](#)

Installation Guide

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

- [Installation Prerequisites](#)
- [Installing the Product](#)
- [Running the Product](#)
- [Basic Health Checks](#)

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

Physical	<ul style="list-style-type: none"> • 3 GHz Dual-core Xeon/Opteron (or latest) • 4 GB RAM (2 GB for JVM and 2 GB for the operating system) • 10 GB free disk space • ~ Recommended minimum - 2 Cores. For high concurrencies and better performances - 4 Cores. <p>Disk space is based on the expected storage requirements that are calculated by considering the file uploads and the backup policies. For example, if three WSO2 product instances are running in a single machine, it requires a 4 GHz CPU, 8 GB RAM (2 GB for the operating system and 6 GB (2 GB for each WSO2 product instance)) and 30 GB of free space.</p>
----------	---

Virtual Machine (VM)	<ul style="list-style-type: none"> • 2 compute units minimum (each unit having 1.0-1.2 GHz Opteron/Xeon processor) • 4 GB RAM • 10 GB free disk space • One CPU unit for the operating system and one for JVM. <p>Three WSO2 product instances running would require VM of 4 compute units, 8 GB RAM, and 30 GB free space. ~ 512 MB heap size. This is generally sufficient to process typical SOAP messages but the requirements vary with larger message sizes and the number of messages processed concurrently.</p>
EC2	<ul style="list-style-type: none"> • 1 c3.large instance to run one WSO2 product instance. <p>Three WSO2 product instances can be run in 1 EC2 Extra-Large instance. Based on the I/O performance of the c3.large instance, it is recommended to run multiple instances in a larger instance (c3.xlarge or c3.2xlarge).</p>
Cassandra data nodes	<ul style="list-style-type: none"> • 4 core processors • 8 GB RAM <p>For more information, see the Cassandra documentation on hardware recommendations for enterprise implementations.</p>

Environment compatibility

<p>Operating Databases</p> <p>systems /</p>	<ul style="list-style-type: none"> All WSO2 Carbon-based products are Java applications that can be run on any platform that is Oracle JDK 1.7/1.8 compliant. However, we do not recommend OpenJDK as we do not support it or test our products with it. <p>WSO2 API Manager is also compatible with IBM JDK 1.7.*/1.8.*. For more information on JDKs that WSO2 products are tested with, see Tested Operating Systems and J D K s. If you want to start WSO2 API Manager with IBM JDK, open the <code><API-M_HOME>/repository/conf/security/Owasp.CsrfGuard.Carbon.properties</code> file and replace <code>org.owasp.csrfguard.PRNG.Provider=SUN</code> with <code>org.owasp.csrfguard.PRNG.Provider=IBMJCE</code>.</p> <ul style="list-style-type: none"> 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. On a production deployment, it is recommended that WSO2 products are installed on latest releases of RedHat Enterprise Linux or Ubuntu Server LTS. For environments that WSO2 products are tested with, see Compatibility of WSO2 Products. To find out if this version of the product has issues running on your OS due to the JDK version, see Known Incompatibilities. 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
-------------	---------	---------

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.*
Web Browser	<ul style="list-style-type: none"> To access the Management Console. The Web browser must be JavaScript enabled to take full advantage of the Management console. <p style="border: 1px solid #ccc; padding: 5px;">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:

- See our [compatibility matrix](#) to find out if this version of the product is fully tested on Linux or OS X.
- See the [known incompatibilities](#) section to find out if this version of the product has issues running on your OS due to the JDK version.

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

Installing the API Manager

1. Download the latest version of the API Manager from <https://github.com/wso2/product-apim/releases/tag/v2.1.0-update10>.
2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as <API_M_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 1.7.0_80 in your system, add the following two lines at the bottom of the file, replacing /usr/java/jdk1.7.0_80 with the actual directory where the JDK is installed.

On Linux:

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

On OS X:

```
export
JAVA_HOME=/System/Library/Java/JavaVirtualMachines/1.7.0.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:
5. The system returns the JDK installation path.

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`
```

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](#).

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:

- See our [compatibility matrix](#) to find out if this version of the product is fully tested on Solaris.
- See the [known incompatibilities](#) section to find out if this version of the product has issues running on your OS due to the JDK version.

Follow the instructions below to install API Manager on Solaris.

Installing the required applications

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

Installing the API Manager

1. Download the latest version of the API Manager from <https://github.com/wso2/product-apim/releases/tag/v2.1.0-update10>.
2. Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as `<API_M_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 1.7.0_80` in your system, add the following two lines at the bottom of the file, replacing `/usr/java/jdk1.7.0_80` with the actual directory where the JDK is installed.

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

The file should now look like this:

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

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:

- See our [compatibility matrix](#) to find out if this version of the product is fully tested on Windows.
- See the [known incompatibilities](#) section to find out if this version of the product has issues running on your OS due to the JDK version.

Follow the instructions below to install API Manager on Windows.

Installing the required applications

1. Be sure your system meets the [Installation Prerequisites](#). 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` windows.exe is stored in this path.

Installing the API Manager

1. Download the latest version of the API Manager from <https://github.com/wso2/product-apim/releases/tag/v2.1>

- .0-update10.
- Extract the archive file to a dedicated directory for the API Manager, which will hereafter be referred to as <API-M_HOME>.

Installing and setting up snappy-java

- Download the snappy-java_1.1.1.7.jar from [here](#).
- Copy the jar to <API-M_HOME>\repository\components\lib.
- If the API Manager server is currently running, restart it to apply the changes.

Setting up JAVA_HOME

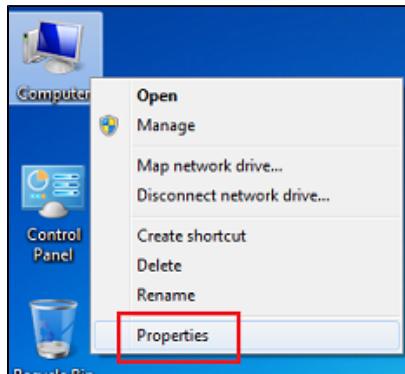
You must set your JAVA_HOME environment variable to point to the directory where the Java Development Kit (JDK) is installed on the computer. Typically, the JDK is installed in a directory under C:/Program Files/Java, such as C:/Program Files/Java/jdk1.7.0_80. If you have multiple versions installed, 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.

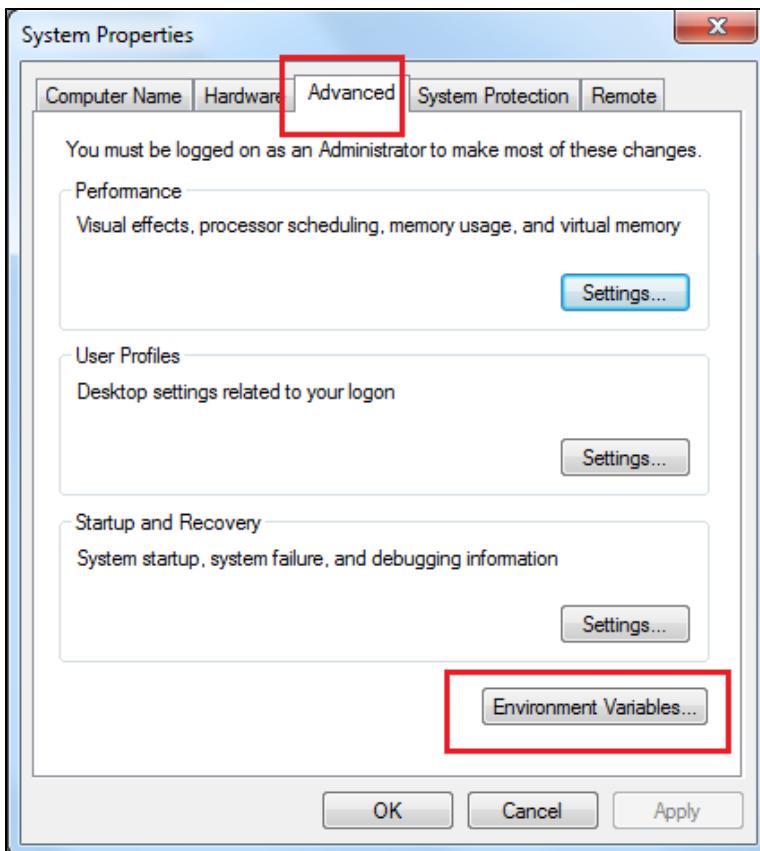
You set up JAVA_HOME using the System Properties, as described below. Alternatively, if you just want to set JAVA_HOME temporarily for the current command prompt window, [set it at the command prompt](#).

Setting up JAVA_HOME using the system properties

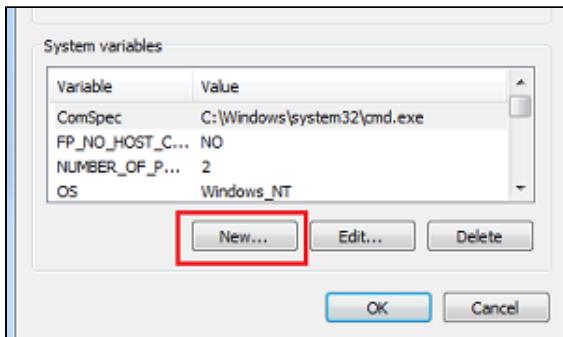
- Right-click the **My Computer** icon on the desktop and click **Properties**.



- In the System Properties window, click the **Advanced** tab, and then click **Environment Variables**.



3. Click **New** 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/jdk1.7.0_80

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/jdk1.7.0_80

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](#)

Before you begin:

- See our [compatibility matrix](#) to find out if this version of the product is fully tested on your OS.
- See the [known incompatibilities](#) section to find out if this version of the product has issues running on your OS due to the JDK version.

Prerequisites

Install JDK 1.7 or later or 1.8.* 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. <API-M_HOME> can vary depending on the WSO2 product's directory.

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

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

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

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

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

If you want to keep the scripts in a location other than /etc/init.d/ folder, you can add a symbolic

link to the script in `/etc/init.d/` and keep the actual script in a separate location. Say your script name is `prodserver` and it is in `/opt/WSO2/` folder, then the commands for adding a link to `/etc/init.d/` is as follows:

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

3. Install the startup script to respective runlevels using the `update-rc.d` command. 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

Before you begin:

- See our [compatibility matrix](#) to find out if this version of the product is fully tested on your OS.
- See the [known incompatibilities](#) section to find out if this version of the product has issues running on your OS due to the JDK version.

Prerequisites

- Install JDK and set up the `JAVA_HOME` environment variable.
- Download and install a service wrapper library to use for running WSO2 API Manager 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](#).

Manual Configurations

Add the following class path to the `wrapper.conf` file manually to avoid errors in the WSO2 API Manager Management Console:

```
wrapper.java.classpath.3 =
${carbon_home}\repository\components\plugins\commons-lang_2.6.0.
wso2v1.jar
```

Minimal wrapper.conf configuration

```
*****
# working directory
*****
wrapper.working.dir=${carbon_home}\
# Java Main class.
# YAJSW: default is "org.rzo.yajsw.app.WrapperJVMMain"
# DO NOT SET THIS PROPERTY UNLESS YOU HAVE YOUR OWN IMPLEMENTATION
# wrapper.java.mainclass=
*****
# tmp folder
# yajsw creates temporary files named in_.. out_.. err_.. jna..
# per default these are placed in jna.tmpdir.
# jna.tmpdir is set in setenv batch file to <yajsw>/tmp
*****
wrapper.tmp.path = ${jna_tmpdir}
*****
# Application main class or native executable
# One of the following properties MUST be defined
*****
# Java Application main class
wrapper.java.app.mainclass=org.wso2.carbon.bootstrap.Bootstrap
# Log Level for console output. (See docs for log levels)
wrapper.console.loglevel=INFO
# Log file to use for wrapper output logging.
wrapper.logfile=${wrapper_home}\log\wrapper.log
# Format of output for the log file. (See docs for formats)
#wrapper.logfile.format=LPTM
# Log Level for log file output. (See docs for log levels)
#wrapper.logfile.loglevel=INFO
# Maximum size that the log file will be allowed to grow to before
# the log is rolled. Size is specified in bytes. The default value
# of 0, disables log rolling by size. May abbreviate with the 'k' (kB) or
# 'm' (mB) suffix. For example: 10m = 10 megabytes.
# If wrapper.logfile does not contain the string ROLLNUM it will be
```

```

automatically added as suffix of the file name
wrapper.logfile.maxsize=10m
# Maximum number of rolled log files which will be allowed before old
# files are deleted. The default value of 0 implies no limit.
wrapper.logfile.maxfiles=10
# Title to use when running as a console
wrapper.console.title="WSO2 Carbon"
*****
# Wrapper Windows Service and Posix Daemon Properties
*****
# Name of the service
wrapper.ntservice.name="WSO2CARBON"
# Display name of the service
wrapper.ntservice.displayname="WSO2 Carbon"
# Description of the service
wrapper.ntservice.description="Carbon Kernel"
*****
# Wrapper System Tray Properties
*****
# enable system tray
wrapper.tray = true
# TCP/IP port. If none is defined multicast discovery is used to find the
port
# Set the port in case multicast is not possible.
wrapper.tray.port = 15002
*****
# Exit Code Properties
# Restart on non zero exit code
*****
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
*****
# Trigger actions on console output
*****
# On Exception show message in system tray
wrapper.filter.trigger.0=Exception
wrapper.filter.script.0=scripts\trayMessage.gv
wrapper.filter.script.0.args=Exception
*****
# genConfig: further Properties generated by genConfig
*****
placeHolderSoGenPropsComeHere=
wrapper.java.command = ${java_home}\bin\java
wrapper.java.classpath.1 = ${java_home}\lib\tools.jar
wrapper.java.classpath.2 = ${carbon_home}\bin\*.jar
wrapper.app.parameter.1 = org.wso2.carbon.bootstrap.Bootstrap
wrapper.app.parameter.2 = RUN
wrapper.java.additional.1 =
-Xbootclasspath\:/a:${carbon_home}\lib\xboot\*.jar
wrapper.java.additional.2 = -Xms256m
wrapper.java.additional.3 = -Xmx1024m
wrapper.java.additional.4 = -XX:MaxPermSize=256m
wrapper.java.additional.5 = -XX:+HeapDumpOnOutOfMemoryError

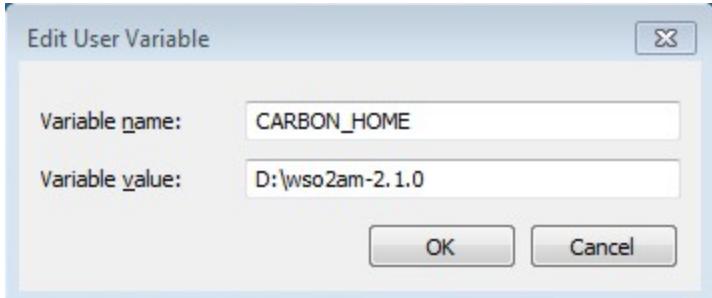
```

```
wrapper.java.additional.6 =
-XX:HeapDumpPath=${carbon_home}\repository\logs\heap-dump.hprof
wrapper.java.additional.7 = -Dcom.sun.management.jmxremote
wrapper.java.additional.8 =
-Djava.endorsed.dirs=${carbon_home}\lib\endorsed;${java_home}\jre\lib\
\endorsed
wrapper.java.additional.9 = -Dcarbon.registry.root=\
wrapper.java.additional.10 = -Dcarbon.home=${carbon_home}
wrapper.java.additional.11 = -Dwso2.server.standalone=true
wrapper.java.additional.12 = -Djava.command=${java_home}\bin\java
wrapper.java.additional.13 = -Djava.io.tmpdir=${carbon_home}\tmp
wrapper.java.additional.14 = -Dcatalina.base=${carbon_home}\lib\tomcat
wrapper.java.additional.15 =
-Djava.util.logging.config.file=${carbon_home}\repository\conf\tomcat\
tomcat-log.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
wrapper.java.additional.26 =
-Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false
```

```
wrapper.java.additional.27 = -Dfile.encoding=UTF8
wrapper.java.additional.28 = -DworkerNode=false
wrapper.java.additional.29 = -Dorg.wso2.ignoreHostnameVerification=true
```

Setting up CARBON_HOME

Extract WSO2 API Manager that you want to run as a Windows service, and then set the Windows environment variable CARBON_HOME to the extracted product directory location which is wso2am-2.1.0 here.



Running the product in console mode

You will now verify that YAJSW is configured correctly for running the WSO2 API Manager 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\ vfs_cache" as temporary files store.
```

Working with the WSO2CARBON service

To install the Carbon-based product WSO2 API Manager 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.VfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\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.VfsLog info
INFO: Using "C:\DOCUMENT\ADMINI\LOCALS\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\ADMINI\LOCALS\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\ADMINI\LOCALS\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>
```

Running the Product

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

The Management Console uses the default HTTP-NIO transport, which is configured in the <PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml file. (<PRODUCT_HOME> is the directory where you installed the WSO2 product you want to run.) You must properly configure the HTTP-NIO transport in this file to access the Management Console. For more information on the HTTP-NIO transport, see the related topics section at the bottom of this page.

The following sections describe how to run the product.

- Starting the server

- Accessing the Management Console
- Accessing the API Publisher
- Accessing the API Store
- Stopping the server
- Related topics

Starting the server

Follow the instructions below to start your WSO2 product based on the Operating System you use.

On Windows/Linux/Mac OS

To start the server, you run <PRODUCT_HOME>\bin\wso2server.bat (on Windows) or <PRODUCT_HOME>/bin/wso2server.sh (on Linux/Mac OS) from the command prompt as described below. Alternatively, you can install and run the server as a Windows or Linux service (see the related topics section at the end of this page).

1. Open a command prompt by following the instructions below.
 - On Windows: Click **Start -> Run**, type cmd at the prompt, and then press **Enter**.
 - On Linux/Mac OS: Establish an SSH connection to the server, log on to the text Linux console, or open a terminal window.
2. Navigate to the <PRODUCT_HOME>/bin/ directory using the Command Prompt.
3. Execute one of the following commands:
 - To start the server in a typical environment:
 - On Windows: wso2server.bat --run
 - On Linux/Mac OS: sh wso2server.sh
 - To start the server in the background mode of Linux: sh wso2server.sh start
To stop the server running in this mode, you will enter: sh wso2server.sh stop
 - To provide access to the production environment without allowing any user group (including admin) to log in to the Management Console:
 - On Windows: wso2server.bat --run -DworkerNode
 - On Linux/Mac OS: sh wso2server.sh -DworkerNode
 - To check for additional options you can use with the startup commands, type -help after the command, such as:
sh wso2server.sh -help (see the related topics section at the end of this page).
4. The operation log appears in the command window. When the product server has successfully started, the log displays the message "WSO2 Carbon started in 'n' seconds".

On Solaris

To start the server, you run <PRODUCT_HOME>/bin/wso2server.sh from the Command Prompt as described below.

Following instructions are tested on an Oracle Solaris 10 8/11 x86 environment.

1. Click **Launch -> Run Applications**, type dtterm at the Prompt, and then press **Enter**, to open a Command Prompt.
2. Navigate to the <PRODUCT_HOME>/bin/ directory using the Command Prompt.
3. Execute the following command: bash wso2server.sh
4. The operation log appears in the command window. When the product server has successfully started, the log displays the message "WSO2 Carbon started in 'n' seconds".

If you are starting the product in service/nohup mode in Solaris, do the following:

1. Update the <PRODUCT_HOME>/bin/wso2server.sh file as follows:
 - a. Search for the following occurrences: **nohup sh "\$CARBON_HOME"/bin/wso2server.sh**

\$args > /dev/null 2>&1 &

- b. Replace those occurrences with the following: **nohup bash "\$CARBON_HOME"/bin/wso2server.sh \$args > /dev/null 2>&1 &**

The only change is replacing sh with bash. This is required only for Solaris.

2. Update your **PATH** variable to have **/usr/xpg4/bin/sh** as the first element. This is because **/usr/xpg4/bin/sh** contains an **sh** shell that is newer than the default **sh** shell. You can set this variable as a system property in the **wso2server.sh** script or you can run the following command on a terminal:

```
export PATH=/usr/xpg4/bin/sh:$PATH
```

3. Start the product by following the above instructions.

Accessing the Management Console

Once the server has started, you can run the Management Console by typing its URL in a Web browser. The following sections provide more information about running the Management Console:

- [Working with the URL](#)
- [Signing in](#)
- [Getting help](#)
- [Configuring the session time-out](#)
- [Restricting access to the Management Console and Web applications](#)

Working with the URL

The URL appears next to “Mgt Console URL” in the start script log that is displayed in the command window. For example:

```
[2017-12-12 18:40:34,831] INFO - StartupFinalizerServiceComponent Server : WSO2 API Manager-2.1.0
[2017-12-12 18:40:34,832] INFO - StartupFinalizerServiceComponent WSO2 Carbon started in 51 sec
[2017-12-12 18:40:34,998] INFO - JMSListener Started to listen on destination : throttleData of type topic for listener Siddhi-JMS-Consumer#throttleData
[2017-12-12 18:40:35,113] INFO - 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 is installed, you can type **localhost** instead of the IP address as follows: <https://localhost:9443/carbon>

You can change the Management Console URL by modifying the value of the **<MgtHostName>** property in the **<PRODUCT_HOME>/repository/conf/carbon.xml** file. When the host is internal or not resolved by a DNS, map the hostname alias to its IP address in the **/etc/hosts** file of your system, and then enter that alias as the value of the **<MgtHostName>** property in **carbon.xml**. For example:

```
In /etc/hosts:
127.0.0.1      localhost
```

```
In carbon.xml:
<MgtHostName>localhost</MgtHostName>
```

Signing in

At the sign-in screen, you can sign in to the Management Console using **admin** as both the username and

password.

When the Management Console sign-in page appears, the Web browser typically displays an "insecure connection" message, which requires your confirmation before you can continue.

The Management Console is based on the 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. Therefore, 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 actually belongs to the given organization.

Getting help

The tabs and menu items in the navigation pane on the left may vary depending on the features you have installed. To view information about a particular page, click the **Help** link at the top right corner of that page, or click the **Docs** link to open the documentation for full information on managing the product.

Configuring the session time-out

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

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

In products like WSO2 API Manager where web applications such as API Publisher/API Store exist, you can configure a session time out for those web apps by changing the repository/conf/tomcat/web.xml file as follows:

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

Restricting access to the Management Console and Web applications

You can restrict access to the Management Console of your product by binding the Management Console with selected IP addresses. You can either restrict access to the Management Console only, or you can restrict access to all Web applications in your server as explained below.

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

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

The `RemoteAddrValve` Tomcat valve defined in this file only applies to the Management Console, and thereby all outside requests to the Management Console are blocked.

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

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

The `RemoteAddrValve` Tomcat valve defined in this file applies to each Web application hosted on the WSO2 product server. Therefore, all outside requests to any Web application are blocked.

- You can also restrict access to particular servlets in a Web application by adding a Remote Address Filter to the `<PRODUCT_HOME>/repository/conf/tomcat/web.xml` file and by mapping that filter to the servlet URL. In the Remote Address Filter that you add, you can specify the IP addresses that should be allowed to access the servlet. The following example from a `web.xml` file illustrates how access to the Management Console page (`/carbon/admin/login.jsp`) is granted only to one IP address.

```
<filter>
    <filter-name>Remote Address Filter</filter-name>

    <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-cl
ass>
        <init-param>
            <param-name>allow</param-name>
            <param-value>127.0.01</param-value>
        </init-param>
    </filter>

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

Any configurations (including valves defined in the `<PRODUCT_HOME>/repository/conf/tomcat/cata
lina-server.xml` file) apply to all Web applications and are globally available across the server,
regardless of the host or cluster. For more information about using remote host filters, see the [Apache
Tomcat documentation](#).

Accessing the API Publisher

Once the server has started, you can run the API Publisher by typing its URL in a Web browser. The following sections provide more information about running the API Publisher:

- Working with the URL
- Signing in

Working with the URL

The URL appears next to “API Publisher Default Context” in the start script log that is displayed in the command window. For example:

```
[2017-12-12 18:40:35,114] INFO - CarbonUIServiceComponent API Publisher Default Context : https://localhost:9443/publisher
```

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

You can use this URL to access the API Publisher on this computer from any other computer connected to the Internet or LAN. When accessing the API Publisher from the same server where it is installed, you can type `localhost` instead of the IP address as follows: `https://localhost:9443/publisher`

Signing in

At the sign-in screen, you can sign in to the API Publisher using **admin** as both the username and password.

When the API Publisher sign-in page appears, the Web browser typically displays an “insecure connection” message, which requires your confirmation before you can continue.

The API Publisher is based on the 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. Therefore, when you try to access the API Publisher, 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 API Publisher 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 actually belongs to the given organization

Accessing the API Store

Once the server has started, you can run the API Store by typing its URL in a Web browser. The following sections provide more information about running the API Store:

- [Working with the URL](#)
- [Signing in](#)

Working with the URL

The URL appears next to “API Store Default Context” in the start script log that is displayed in the command window. For example:

```
[2017-12-12 18:40:35,114] INFO - CarbonUIServiceComponent API Store Default Context : https://localhost:9443/store
```

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

You can use this URL to access the API Store on this computer from any other computer connected to the Internet or LAN. When accessing the API Store from the same server where it is installed, you can type `localhost` instead of the IP address as follows: `https://localhost:9443/store`

Signing in

At the API Store home page, you can click sign in link at top right corner to sign-in to the API Publisher using **admin** as both the username and password.

When the API Store home page appears, the Web browser typically displays an "insecure connection" message, which requires your confirmation before you can continue.

The API Store is based on the 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. Therefore, when you try to access the API Store, 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 API Store 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 actually belongs to the given organization

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. If you started the server in background mode in Linux, enter the following command instead:

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

Restricting Access to the Management Console and Web Applications:

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

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

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

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

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

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

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

- You can also restrict access to particular servlets in a web application by adding a Remote Address Filter to the `web.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/tomcat/` directory)

, and by mapping that filter to the servlet url. In the Remote Address Filter that you add, you can specify the IP addresses that should be allowed to access the servlet.

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

```
<filter>
    <filter-name>Remote Address Filter</filter-name>

    <filter-class>org.apache.catalina.filters.RemoteAddrFilter</filter-class>
        <init-param>
            <param-name>allow</param-name>
            <param-value>127.0.0.1</param-value>
        </init-param>
    </filter>

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

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

Related topics

- [Installing as a Windows Service](#)
- [Installing as a Linux Service](#)

Basic Health Checks

Basic health checks can be performed on an API Manager node by connecting to relevant ports. See the following table for the ports that can be used for health checks in a fully distributed deployment.

API Manager Profile	Ports that can be used for health checks
Gateway Manager	9763 (HTTP), 9443 (HTTPS)
Gateway Worker	8280 (HTTP), 8243 (HTTPS)
API Store	9673 (HTTP), 9443 (HTTPS)
API Publisher	9673 (HTTP), 9443 (HTTPS)
Traffic Manager	5672 (TCP), 7611 (TCP), 7711 (TCP)
Key Manager	9673 (HTTP), 9443 (HTTPS)

For more information on each profile, see [API Manager Profiles](#).

There can be scenarios where even though the ports are responding, the Services are not properly started. It is advisable to use Service level health checks to ensure that the services are started. For example, API Manager by default is shipped with the simple axis2 service named Version. This service returns the version of the API Manager instance that is running currently.

A sample cURL command and the response from the Version service are given below.

cURL command:

[Format Sample](#)

```
curl -v http://<HOSTNAME>:<PORT>/services/Version
```

```
curl -v http://localhost:9763/services/Version
```

response:

```
<ns:getVersionResponse
xmlns:ns="http://version.services.core.carbon.wso2.org"><return>WSO2 API
Manager-2.1.0</return></ns:getVersionResponse>
```

WSO2 API Manager Tooling

This section describes how you can use the tooling support provided via [WSO2 API Manager tooling](#) to create custom sequences and use it with the API Manager.

See the following topics for detailed information:

- [Installing the API Manager Tooling Plug-In](#) - install WSO2 API Manager tooling
- [Adding Mediation Extensions](#) - extend the default mediation flow by adding custom mediation sequences
- [Pass a Custom Authorization Token to the Backend](#) - pass a custom authorization token that is different to the authorization token generated for the application
- [Change the Default Mediation Flow of API Requests](#) - create a custom sequence and then deploy and use it in your APIs
- [Convert a JSON Message to SOAP and SOAP to JSON](#) - convert message types using custom sequences
- [Map the Parameters of your Backend URLs with the API Publisher URLs](#) - map your backend URLs to the pattern that you want in the API Publisher

Installing the API Manager Tooling Plug-In

The API Manager tooling plug-in gives the capabilities of a complete Eclipse-based development environment for the API Manager. You can develop services, features, and artifacts, and manage their links and dependencies through a simplified graphical editor.

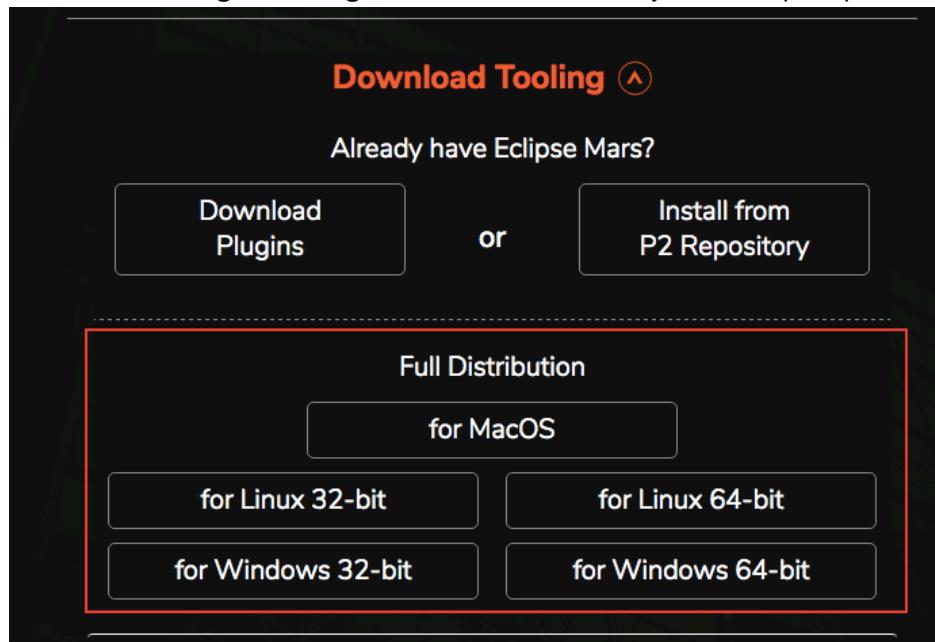
You can install multiple [WSO2 product tooling plug-ins](#) on top of the same Eclipse instance.

There are 3 possible methods you can use to install the tooling plug-in.

- [Install the plug-in with pre-packaged Eclipse](#) - This method uses a complete plug-in installation **with pre-packaged Eclipse**, so that you do not have to install Eclipse separately.
- [Install the plug-in on Eclipse Mars using the P2 URL](#) - This method requires you to **install Eclipse Mars separately** in your system if you do not have it already.
- [Install the plug-in on Eclipse Mars using the P2 .zip file](#) - This method requires you to **install Eclipse Mars separately** in your system if you do not have it already.

Install the plug-in with pre-packaged Eclipse

On the [API Manager product page](#), click **Download**, click **Download Tooling** and then click on the respective link under **Full Distribution** to download the distribution according to your operating system under the **Eclipse JavaEE Mars + API Manager Tooling 2.1.0** section. Note that you will be prompted to enter the email address here.

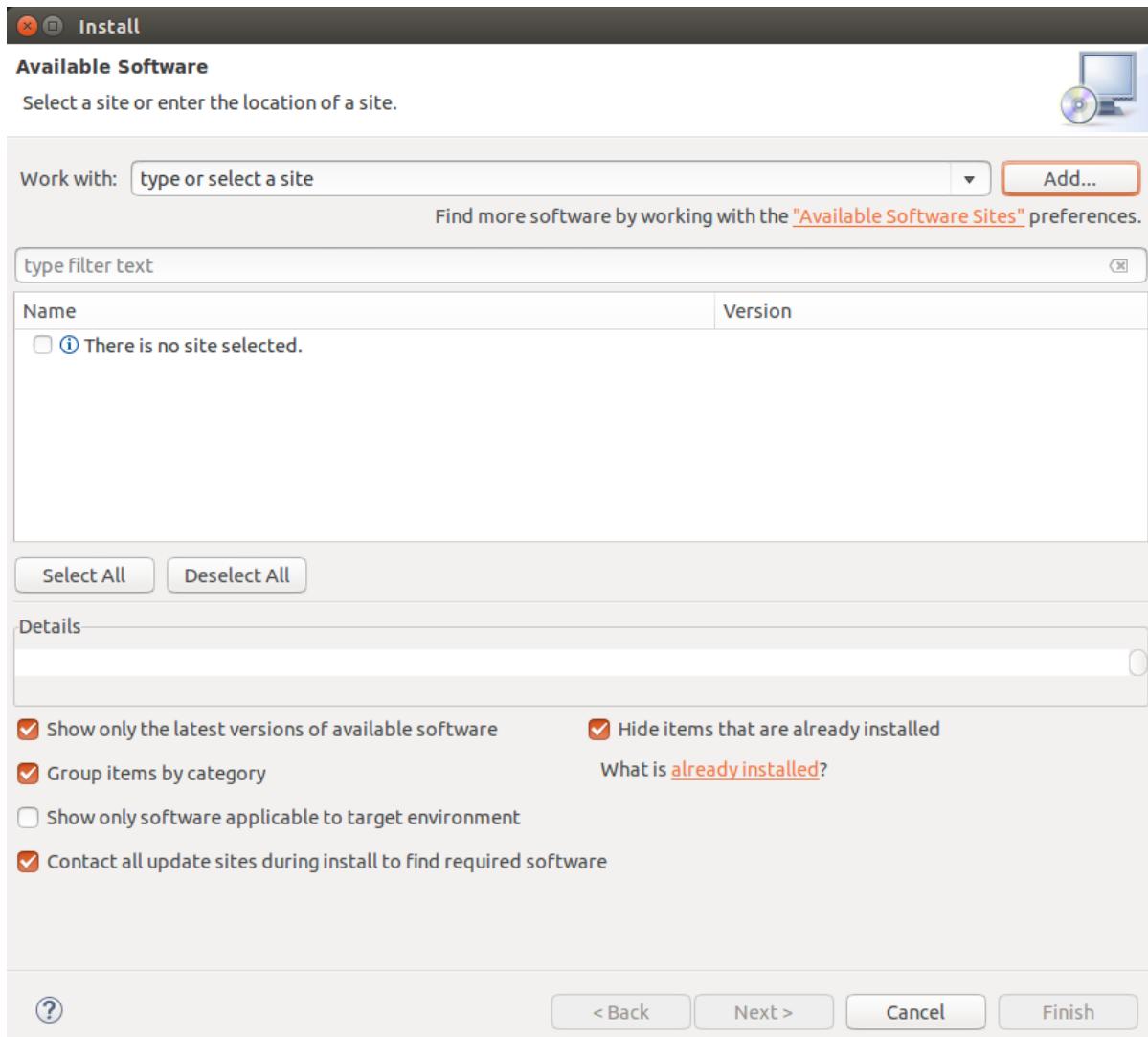


Install the plug-in on Eclipse Mars using the P2 URL

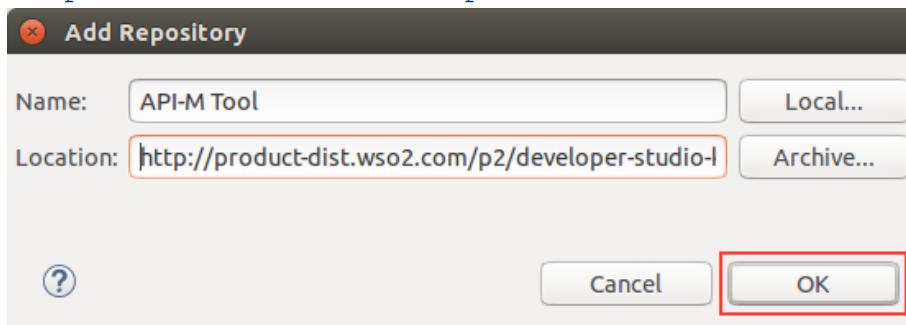
1. Make sure you have Eclipse IDE for Java EE Developers (Mars 2) installed.

Note that its recommended to use Eclipse Mars, since the plug-in has been tested on this version.

2. Open Eclipse and on the **Help** menu click **Install New Software**.
3. Click **Add** in the dialog box that appears.



- Add the repository by entering **API-M Tool** as the name and <http://product-dist.wso2.com/p2/developer-studio-kernel/4.1.0/apim-tools/releases/2.1.0/> as the location and click **OK**.



- Select the required software and click **Next**. Select **WSO2 API Manager Tools** and **WSO2 ESB Tools** check box. You need to install the WSO2 ESB Tools as WSO2 API-M tooling totally depends on WSO2 Enterprise Service Bus (WSO2 ESB).

If you are using a fresh instance of Eclipse with no other **WSO2 product tooling plug-ins** installed, you need to select the Developer Studio checkboxes as well.

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the ["Available Software Sites"](#) preferences.

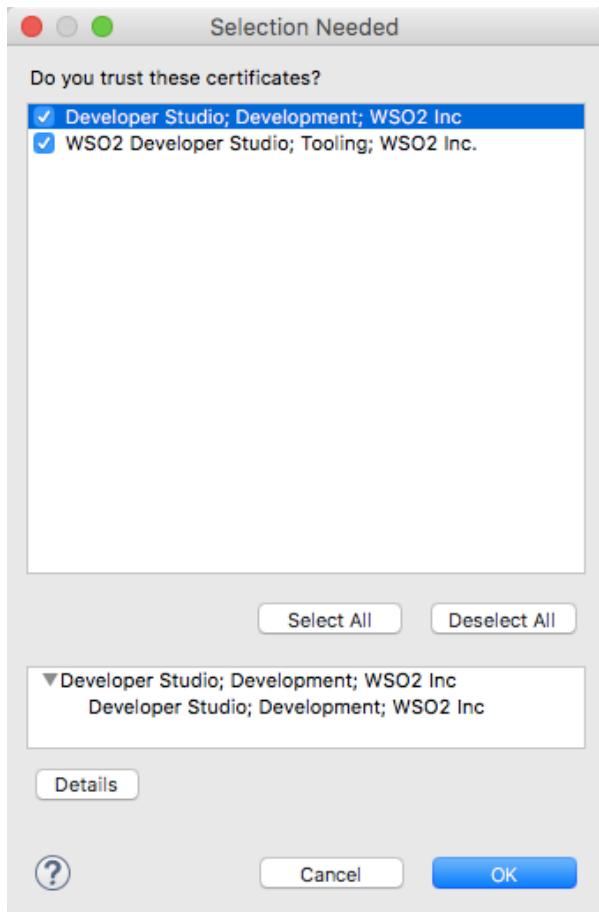
type filter text

Name	Version
<input type="checkbox"/> ⓘ There is no site selected.	

Details

Show only the latest versions of available software Hide items that are already installed
 Group items by category What is [already installed](#)?
 Show only software applicable to target environment
 Contact all update sites during install to find required software

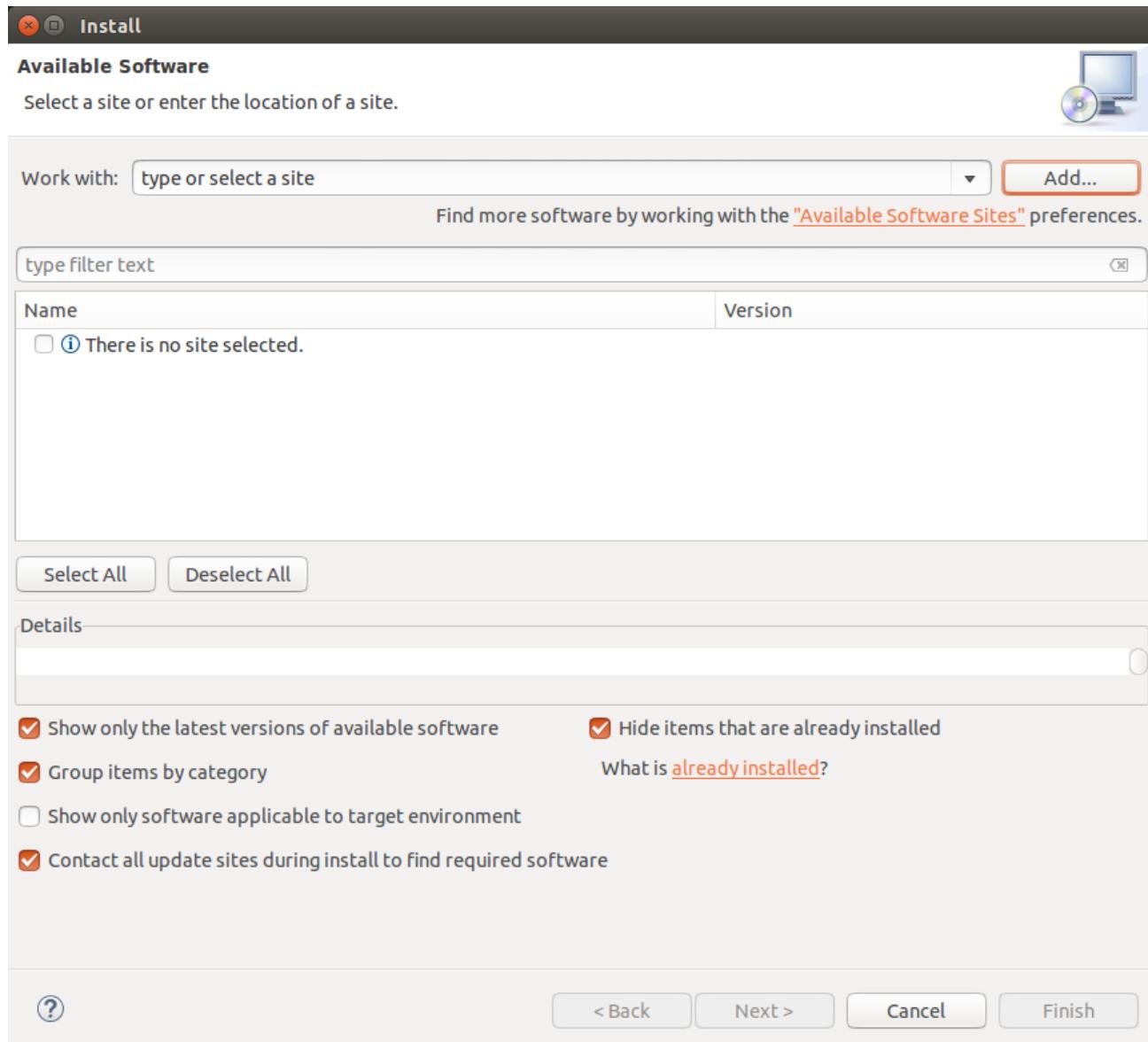
6. Review the items to be installed and click **Next**.
7. Read and accept the license agreements and click **Finish**.
8. If a security warning appears saying that the authenticity or validity of the software cannot be established, click **OK**.
9. Select the WSO2 Developer Studio related certificates and click **OK**.



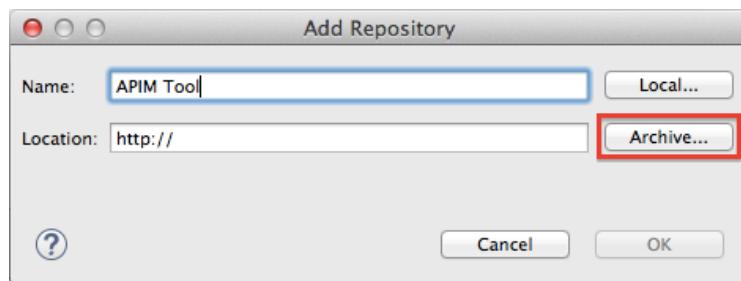
10. Click **OK** to restart Eclipse to complete the installation.

Install the plug-in on Eclipse Mars using the P2 .zip file

1. Make sure you have Eclipse IDE for Java EE Developers (Mars 2) installed.
2. On the [API Manager product page](#), click **Download**, click **Download Tooling**, and then click **Download Plugins** to download the P2 .zip file.
3. Open Eclipse and navigate to the **Help** menu and click **Install New Software**.
4. On the dialog box that appears, click **Add**.



- Give APIM Tool as the name and click **Archive**.



- Give APIM Tool as the name and click **Archive**.
- Navigate to the downloaded .zip file and click **OK**.
- Select all the check boxes and click **Next**.
- Read and accept the license agreements and click **Finish**.
- If a security warning appears saying that the authenticity or validity of the software cannot be established, click **OK**.
- Restart Eclipse to complete the installation.

Product Administration

WSO2 API Manager is shipped with default configurations that allow you to download, install and get started with your product instantly. However, when you go into production, it is recommended to change some of the default

settings to ensure that you have a robust system that is suitable for your operational needs. Also, you may have specific use cases that require specific configurations to the server. If you are a product administrator, the follow content will provide an overview of the administration tasks that you need to perform when working with WSO2 API Manager (WSO2 API-M).

What is the WSO2 Administration Guide?

The WSO2 Administration Guide is a WSO2 guide that is common to all WSO2 products, and it explains all you need to know about how to set up and configure a WSO2 product to meet your enterprise requirements. In the subsequent sections we point you to the WSO2 Administration Guide where necessary.

[Upgrading from a previous release][Changing the default database][Configuring users, roles, and permissions][Configuring security][Configuring multitenancy][Configuring the registry][Performance tuning][Changing the default ports][Managing product features][Configuring custom proxy paths][Customizing error pages][Customizing the management console][Applying patches][Monitoring the server]

Upgrading from a previous release

If you are upgrading from WSO2 API Manager 2.0.0 to WSO2 API Manager 2.1.0, see the [upgrading instructions for WSO2 API Manager](#).

Changing the default database

By default, WSO2 products are shipped with an embedded H2 database, which is used for storing user management and registry data. We recommend that you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. when you set up your production environment. You can change the default database configuration by simply setting up a new physical database and updating the configurations in the product server to connect to that database.

For instructions on setting up and configuring databases, see the following sections in the WSO2 Administration Guide.

Setting up the Physical Database

You can use the scripts provided with WSO2 API-M to install and configure several other types of relational databases. For more information, see the following sections.

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

Changing the Carbon Database

WSO2 API-M is shipped with an H2 database, which serves as the default Carbon database. You can change this default database to one of the standard databases listed below.

- [Changing to Embedded Derby](#)
- [Changing to Embedded H2](#)
- [Changing to IBM DB2](#)
- [Changing to IBM Informix](#)
- [Changing to MariaDB](#)
- [Changing to MSSQL](#)
- [Changing to MySQL](#)
- [Changing to Oracle](#)
- [Changing to Oracle RAC](#)
- [Changing to PostgreSQL](#)
- [Changing to Remote H2](#)

When changing the default database, which is H2, to any other RDBMS, be sure to change all the other required datasources (`WSO2AM_DB`, `WSO2AM_STATS_DB`, `WSO2_MB_STORE_DB`, `WSO2_METRICS_DB`, etc.) to the same RDBMS. For more information, see [Changing the Default API-M Databases](#).

Configuring users, roles, and permissions

The user management feature in your product allows you to create new users and define the permissions granted to

each user. You can also configure the user stores that are used for storing data related to user management.

- For more information on working with user management, see the following sections in the WSO2 Administration Guide.

Configuring the System Administrator	The admin user is the super tenant who is able to manage all other users, roles and permissions in the system by using the management console of the product. Therefore, the user that has admin permissions has to be stored in the primary user store when you start the system for the first time. The documentation on setting up primary user stores explains how to configure the administrator while configuring the user store. The information under this topic explains the main configurations that are relevant to setting up the system administrator.
Configuring the Authorization Manager	<p>According to the default configuration in WSO2 products, the users, roles and permissions are stored in the same repository (i.e., the default, embedded H2 database). However, you can change this configuration in such a way that the users and roles are stored in one repository (user store) and the permissions are stored in a separate repository. A user store can be a typical RDBMS, a LDAP, or an external Active Directory.</p> <p>The repository that stores permissions should always be a RDBMS. The Authorization Manager configuration in the <code>user-mgt.xml</code> file connects the system to this RDBMS. The information under this topic instructs you through setting up and configuring the Authorization Manager.</p>
Configuring User Stores	<p>The user management feature in WSO2 API-M allows you to maintain multiple user stores for storing users and their roles. See the following topics for instructions:</p> <ul style="list-style-type: none"> Configuring the Primary User Store <ul style="list-style-type: none"> Configuring a JDBC User Store Configuring a Read-Only LDAP User Store Configuring a Read-Write Active Directory User Store Configuring a Read-Write LDAP User Store Configuring Secondary User Stores Working with Properties of User Stores Writing a Custom User Store Manager

Managing Users, Roles and Permissions

The user management functionality is provided by default in WSO2 API-M, and it supports the role-based authentication model where privileges of a user are based on the role attached. For more information on managing users, roles and permissions see the following sections in the WSO2 Administration Guide.

- [Changing a Password](#)
- [Configuring Roles](#)
- [Configuring Users](#)
- [Role-based Permissions](#)
- [Managing User Attributes](#)

- For WSO2 API Manager specific user management related information, see the following sections in the WSO2 API Manager Guide.

Configuring security

After you install WSO2 API Manager, it is recommended to change the default security settings according to the requirements of your production environment. As API Manager is built on top of the WSO2 Carbon Kernel (version 4.4.11), the main security configurations applicable to API Manager are inherited from the Carbon kernel.

For instructions on configuring security in your server, see the following topics in the WSO2 Administration Guide.

Important!

If you are configuring your **production environment**, be sure to check the [Security Guidelines for Production Deployment](#) before applying any security configurations.

Securing Passwords in Configuration Files

All WSO2 Carbon products contain some configuration files with sensitive information such as passwords. Let's take a look at how such plain text passwords in configuration files can be secured using the Secure Vault implementation that is built into Carbon products.

The following topics will be covered under this section:

- [Encrypting Passwords with Cipher Tool](#)
 - Encrypting passwords using the automated process
 - Encrypting passwords manually
 - Changing encrypted passwords
- [Resolving Encrypted Passwords](#)
 - Enter password in command-line
 - Start server as a background job
- [Carbon Secure Vault Implementation](#)
 - Customizing the Secure Vault configuration
 - Creating a Secret Callback Handler
 - Creating a custom Secret Repository

Configuring Transport-Level Security	<p>The transport level security protocol of the Tomcat server is configured in the <PRODUCT_HOME>/core/conf/tomcat/catalina-server.xml file. Note that the sslProtocol attribute is set to "TLS" by default.</p> <p>The following topics will guide you through the configuration options:</p> <ul style="list-style-type: none"> • Enabling TLS and disabling SSL support <ul style="list-style-type: none"> • Disabling SSL support for products with JDK 1.8 • Enabling TLSv1.1/TLSv1.2 for products with JDK 1.7 • Disabling weak ciphers <ul style="list-style-type: none"> • Disabling weak ciphers for the Tomcat transport • Disables weak ciphers for the PassThrough transport • Changing the server name in HTTP response headers
Enabling Java Security Manager	<p>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>/core/repository/conf/sec.policy file. You modify this file to change the Java security permissions as required.</p>
Using Asymmetric Encryption	<p>WSO2 products use asymmetric encryption by default for the purposes of authentication and data encryption. In asymmetric encryption, keystores (with key pairs and certificates) are created and stored for the product. It is possible to have multiple keystores so that the keys used for different use cases are kept unique. The following topics explain more details on keystores.</p> <ul style="list-style-type: none"> • Understanding keystores • Setting up keystores for WSO2 products • Default keystore settings in WSO2 products • Managing keystores
Using Symmetric Encryption	<p>WSO2 Carbon-based products use asymmetric encryption by default as explained in the previous section. From Carbon 4.4.3 onwards, you have the option of switching to symmetric encryption in your WSO2 product. Using symmetric encryption means that a single key will be shared for encryption and decryption of information.</p>

Mitigating Cross Site Request Forgery Attacks	<p>Cross Site Request Forgery (CSRF) attacks trick you to send a malicious request, by forcing you to execute unwanted actions on an already authenticated web browser. For more information, see the following sections that describe the impact of the Cross Site Request Forgery (CSRF) attack and how to mitigate it.</p> <ul style="list-style-type: none"> • How can CSRF attacks be harmful? • Mitigating CSRF attacks <ul style="list-style-type: none"> • Securing web applications • Securing Jaggery applications
Mitigating Cross Site Scripting Attacks	<p>Cross Site Scripting (XSS) attacks use web applications to inject malicious scripts or a malicious payload, generally in the form of a client side script, into trusted legitimate web applications. For more information, see the following sections that describe the impact of XSS attack and the approaches you can use to mitigate it.</p> <ul style="list-style-type: none"> • How can XSS attacks be harmful? • Mitigating XSS attacks
Enabling HostName Verification	<p>Hostname verification is enabled in WSO2 products by default, which means that when a hostname is being accessed by a particular client, it will be verified against the hostname specified in the product's SSL certificate.</p>
Configuring TLS Termination	<p>When you have Carbon servers fronted by a load balancer, you have the option of terminating SSL for HTTPS requests. This means that the load balancer will be decrypting incoming HTTPS messages and forwarding them to the Carbon servers as HTTP. This is useful when you want to reduce the load on your Carbon servers due to encryption. To achieve this, the load balancer should be configured with TLS termination and the Tomcat RemotelpValve should be enabled for Carbon servers.</p>

Configuring multitenancy

You can create multiple tenants in your product server, which will allow you to maintain tenant isolation in a single server/cluster. For instructions on configuring multiple tenants for your server, see [Working with Multiple Tenants](#) in the WSO2 Administration Guide and to understand how multitenancy works with the API Store, see [Managing Tenants](#).

Configuring the registry

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

For instructions on setting up and configuring the registry for your server, see [Working with the Registry](#) in the WSO2 Administration Guide.

Performance tuning

You can optimize the performance of your WSO2 server by using configurations and settings that are suitable to your production environment. At a basic level, you need to have the appropriate OS settings, JVM settings etc. Since WSO2 products are all based on a common platform called Carbon, most of the OS, JVM settings recommended for production are common to all WSO2 products. Additionally, there will be other performance enhancing configuration recommendations that will depend on very specific features used by your product.

For instructions on the Carbon platform-level performance tuning recommendations, see [Performance Tuning](#) in the WSO2 Administration Guide.

For instructions on performance tuning recommendations that are specific to WSO2 API Manager, see [tuning performance](#) in the WSO2 API-M Guide.

Changing the default ports

When you run multiple WSO2 products, multiple instances of the same product, or multiple WSO2 product clusters on the same server or virtual machines (VMs), you must change their default ports with an offset value to avoid port conflicts.

What is port offset?

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be offset. For example, if the HTTP port is defined as 9763 and the portOffset is 1, the effective HTTP port will be 9764. Therefore, for each additional WSO2 product, instance, or cluster you add to a server, set the port offset to a unique value (the default is 0).

For the list of ports in all WSO2 products, see [Default Ports of WSO2 Products](#) in the WSO2 Administration Guide.

For instructions on configuring ports, see [Changing the Default Ports](#) in the WSO2 Administration Guide.

Managing product features

Each WSO2 product is a collection of reusable software units called features where a single feature is a list of components and/or other feature. By default, WSO2 API Manager is shipped with the features that are required for your main use cases.

For instructions on working with features, see the following topics in the WSO2 Administration Guide.

Feature Management	Each enterprise middleware product is a collection of reusable software units called features. Similarly, WSO2 Carbon consists of a collection of features where a single feature is a list of components and/or other feature.
Managing the Feature Repository	<p>The WSO2 Feature Repository consists of features that are bundled into WSO2 products that are based on a particular Carbon release. The feature repository for WSO2 products based on Carbon 4.4.x versions is http://product-dist.wso2.com/p2/carbon/releases/wilkes/. For more information, see the following sections in the WSO2 Administration Guide.</p> <ul style="list-style-type: none"> • Accessing the available repository list • Adding a repository • Editing a repository • Enabling/disabling a repository • Deleting a repository
Installing Features	<p>For more information on installing features in WSO2 products, see the following sections in the WSO2 Administration Guide.</p> <ul style="list-style-type: none"> • Installing Features using pom Files • Installing Features via the UI
Uninstalling Features	If required, you can browse through the list of installed features and uninstall features that you do not require. However, if there are other features that depend on the feature that you are uninstalled, those dependent sub features need to be uninstalled first, before attempting to uninstall the main feature.
Recovering from Unsuccessful Feature Installation	<p>After installing features, if you encounter server issues or startup failures, you can revert the current configuration by restoring a previous one using either the management console or the command line. The latter is recommended if you cannot start the server.</p> <ul style="list-style-type: none"> • Reverting using the management console • Reverting using the command line

Configuring custom proxy paths

This feature is particularly useful when multiple WSO2 products (fronted by a proxy server) are hosted under the same domain name. By adding a custom proxy path you can host all products under a single domain and assign proxy paths for each product separately .

For instructions on configuring custom proxy paths, see [Adding a Custom Proxy Path](#) in the WSO2 Administration Guide.

Customizing error pages

You can make sure that sensitive information about the server is not revealed in error messages, by customizing the error pages in your product.

For instructions, see [Customizing Error Pages](#) in the WSO2 Administration Guide.

Customizing the management console

Some of the WSO2 products, such as WSO2 API Manager consist of a web user interface named the management console. This allows administrators to configure, monitor, tune, and maintain the product using a simple interface. You can customize the look and feel of the management console for your product.

For instructions, see [Customizing the Management Console](#) in the WSO2 Administration Guide.

Applying patches

For information on updating WSO2 API-M with the latest available patches (issued by WSO2) using the WSO2 Update Manager (WUM) tool, see [Updating WSO2 Products](#) in the WSO2 Administration Guide.

Monitoring the server

Monitoring is an important part of maintaining a product server. Listed below are the monitoring capabilities that are available for WSO2 API Manager.

- **Monitoring server logs:** A properly configured logging system is vital for identifying errors, security threats and usage patterns in your product server. For instructions on monitoring the server logs, see [Monitoring Logs](#) in the WSO2 Administration Guide.
- Monitoring using WSO2 metrics: WSO2 API Manager is shipped with JVM Metrics, which allows you to monitor statistics of your server using Java Metrics. For instructions on setting up and using Carbon metrics for monitoring, see [Using WSO2 Metrics](#) in the WSO2 Administration Guide.
- JMX-based monitoring: For information on monitoring your server using JMX, see [JMX Monitoring](#) in the WSO2 Administration Guide.

Upgrading from the Previous Release

The following information describes how to upgrade your API Manager server **from APIM 1.8.0/1.9.0/1.9.1/1.10.0/2.0.0 to 2.1.0**.

To upgrade **from a version older than 1.8.0**, follow the instructions in the document that was released immediately after your current release and upgrade incrementally.

[Upgrading from 2.0.0 to 2.1.0](#)[Upgrading from 1.10.0 to 2.1.0](#)[Upgrading from 1.8.0/1.9.0/1.9.1 to 2.1.0](#)

Follow the instructions below to upgrade your WSO2 API Manager server **from WSO2 API-M 2.0.0 to 2.1.0**.

If you are using WSO2 IS as a Key Manager, follow the instructions mentioned in [Upgrading from the Previous Release when WSO2 IS is the Key Manager](#).

- Step 1 - Migrate the configurations
 - Step 1.1 - Migrate the API Manager configurations
 - Step 1.2 - Migrate the configurations for the API-M Analytics profile
- Step 2 - Upgrade API Manager to 2.1.0

Step 1 - Migrate the configurations

Step 1.1 - Migrate the API Manager configurations

Do not copy entire configuration files from the current version of API Manager to the new one as some configuration files (for example, `api-manager.xml`) may have changed. Instead, redo the configuration changes in the new configuration files.

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

1. Back up all databases in your API Manager instances along with the synapse configs of all the tenants and super tenant.

You find the synapse configs of the super tenant in the `<CURRENT_API-M_HOME>/repository/deployment` directory. Synapse configs of tenants are in the `<CURRENT_APIM_HOME>/repository/tenants` directory.

If you use a **clustered/distributed API Manager setup**, back up the available configurations in the API Gateway node.

2. Download API Manager 2.1.0 from <http://wso2.com/products/api-manager/>.
3. Open the `<API-M_2.1.0_HOME>/repository/conf/datasources/master-datasources.xml` file and provide the datasource configurations for the following databases. You can copy the configuration values from the same file in the current API Manager instance already being used.
 - User Store
 - Registry database/s
 - API Manager Databases
 - Statistics Database (If Statistics are configured.)
4. Edit the registry configurations in the `<API-M_2.1.0_MANAGER_HOME>/repository/conf/registry.xml` file and the user database in the `<API-M_2.1.0_MANAGER_HOME>/repository/conf/user-mgt.xml` file similar to the configurations of the current API Manager.

In a **clustered/distributed API Manager setup**, carryout steps 5 and 6 on the Gateway node.

5. Move all your synapse configurations except the files mentioned below by copying and replacing the `<CURRENT_APIM_HOME>/repository/deployment/server/synapse-config` directory to the `<API-M_2.1.0_MANAGER_HOME>/repository/deployment/server/synapse-config`

NOTE: Do not replace the files listed below from the `<CURRENT_API-M_HOME>/repository/deployment` folder to APIM 2.1.0. These are application-specific APIs and sequences. If you made any custom changes to the files below, please merge them to the corresponding files in 2.1.0.

`/api/_AuthorizeAPI_.xml`

```
/api/_RevokeAPI_.xml
/api/_TokenAPI_.xml
/api/_UserInfoAPI_.xml
/sequences/_auth_failure_handler_.xml
/sequences/_build_.xml
/sequences/_cors_request_handler_.xml
/sequences/fault.xml
/sequences/main.xml
/sequences/_production_key_error_.xml
/sequences/_resource_mismatch_handler_.xml
/sequences/_sandbox_key_error_.xml
/sequences/_throttle_out_handler_.xml
/sequences/_token_fault_.xml
/proxy-services/WorkflowCallbackService.xml
```

6. Move all your tenant synapse configurations by updating the configurations made in the <CURRENT_API-M_H> directory to the <API-M_2.1.0_MANAGER_HOME>/repository/tenants directory.

NOTE: Get the files listed below from the <API-M_2.1.0_MANAGER_HOME>/repository/deployment directory and replace the corresponding files in the <API-M_2.1.0_MANAGER_HOME>/repository/tenants directory.

```
_auth_failure_handler_.xml
_build_.xml
_cors_request_handler_.xml
fault.xml
main.xml
_production_key_error_.xml
_resource_mismatch_handler_.xml
_sandbox_key_error_.xml
_throttle_out_handler_.xml
_token_fault_.xml
```

In a **clustered/distributed API Manager setup**, carryout this step on the Gateway node.

Step 1.2 - Migrate the configurations for the API-M Analytics profile

Follow the steps below to migrate the configurations required to run the APIM Analytics profile for WSO2 APIM 2.1.0 when you are upgrading from APIM Analytics 2.0.0.

These steps are required only if the database type configured with the APIM Analytics profile is either H2 or MySQL.

1. Download WSO2 API Manager Analytics 2.1.0 from [here](#).
2. If you want to connect your WSO2 API Manager Analytics 2.1.0 installation to the same databases that were used with your API Manager Analytics 2.0.0 installation, edit the configurations in the <API-M_ANALYTICS_2>

- file similar to the configurations in the same file of your WSO2 API Manager Analytics 2.0.0 installation.
- Replace the Statistics DB (STAT_DB) configurations in the <API-M_ANALYTICS_2.1.0_HOME>/repository/conf/stats-datasources.xml file with the configurations in the same file of your WSO2 API Manager Analytics 2.0.0 Installation.

Step 2 - Upgrade API Manager to 2.1.0

- Stop all running WSO2 API Manager server instances.
- Make sure you backed up all the databases and synapse configs as instructed in step 1 of the previous section.
- Run the respective migration script based on your environment to carryout the migration process.

Linux/Mac OS

Run the `apim200_to_apim210_gateway_artifact_migrator.sh` script as shown below to migrate from API Manager 2.0.0 to 2.1.0.

```
./apim200_to_apim210_gateway_artifact_migrator.sh
<API-definitions-path>
```

<API-definition-path> - This is the location where the WSO2 API-M 2.1.0 API definitions reside, which are copied from API-M 2.0.0 deployment.

The API definition paths <API-definition-path> are as follows:

- Super Tenant - <API-M_2.1.0_HOME>/repository/deployment/server/synapse-configs/default
- Tenant - <API-M_2.1.0_HOME>/repository/tenants/<tenant-id>/synapse-configs/default

Where <API-M_2.1.0_HOME> can be for example /Users/user12/Documents/wso2am-2.1.0.

Run the PowerShell script (`apim200_to_apim210_gateway_artifact_migrator.ps1`) as follows:

- Open a Windows command prompt and type the following command.

```
powershell
```

A message about PowerShell appears, and the shell changes to PS.

- Run the powershell script by passing the location of the gateway artifacts that you need to migrate. You can do this in the home directory of the existing API-M 2.0.0 installation

```
.\apim200_to_apim210_gateway_artifact_migrator.ps1
<API-definitions-path>
```

<API-definition-path> - This is the location where the WSO2 API-M 2.1.0 API definitions reside, which are copied from API-M 2.0.0 deployment.

- Super Tenant - <API-M_2.1.0_HOME>/repository/deployment/server/synapse-configs/default

- Tenant - <API-M_2.1.0_HOME>/repository/tenants/<tenant-id>/synapse-confi gs/default

Where <API-M_2.1.0_HOME> can be for example /Users/user12/Documents/wso2am-2.1.0.

It may take a considerable amount of time, which is proportionate to the amount of artifacts, to complete the migration process.

Troubleshooting

Why do I get the following error - apim200_to_apim210_gateway_artifact_migrator.ps1 cannot be loaded because the execution of scripts is disabled on this system?

When running the `apim200_to_apim210_gateway_artifact_migrator.ps1` script, if the execution process is aborted with the above error, it means that the execution of unknown scripts are disabled in the system. To overcome this issue run the following command in the terminal/command-line as the Administrator to allow the execution of such scripts.

```
Set-ExecutionPolicy RemoteSigned
```

4. Do the following to re-index the artifacts in the registry:
 - a. Rename the `<lastAccessTimeLocation>` element in the `<API-M_2.1.0_HOME>/repository/comp` file. If you use a **clustered/distributed API Manager setup**, change the file in the API Publisher node. For example, change the `/_system/local/repository/components/org.wso2.carbon` registry path to `/_system/local/repository/components/org.wso2.carbon.registry/in`
 - b. Shut down API Manager 2.1.0 if you have already started it, backup and delete the `<API-M_2.1.0_H`OME>/ `solr` directory if it exists.
5. Upgrade the Identity component in WSO2 API Manager from version 5.2.0 to 5.3.0.
 - a. Download the 5.2.0 to 5.3.0 migration DB scripts from [here](#).
 - b. Unzip and find the correct DB script in `<MIGRATION_SCRIPT_HOME>/wso2is-5.3.0-migration/` `dbscripts/identity/migration-5.2.0_to_5.3.0` directory.
 - c. Execute the corresponding DB script against the `WSO2AM_DB` database manually.
 - d. Copy the `/wso2is-5.3.0-migration/dropins/org.wso2.carbon.is.migrate.client-5.` `3.0.jar` file into the `<APIM_2.1.0_HOME>/repository/components/dropins` directory.
 - e. Start API Manager 2.0.0 with the command line option `-Dmigrate -Dcomponent=identity` to carry out the complete Identity and User Store DB migration.

This concludes the upgrade process.

Tip 1: The migration client that you use in this guide automatically migrates your tenants, workflows, external user stores etc. to the upgrade environment. There is no need to migrate them manually.

Tip 2: If you are using SVN based deployment synchronizer, start with a clean SVN repository and point the new deployment's nodes to the new SVN repository. Also, any existing .svn directories in the new deployment's <API-M_2.1.0_HOME>/repository/deployment/server/synapse-configs/default and <API-M_2.1.0_HOME>/repository/tenants/<tenant-id>/synapse-configs/default locations should be removed before starting the servers. See [Configuring Deployment Synchronization](#) for more details.

Follow the instructions below to upgrade your WSO2 API Manager server from APIM 1.10.0 to 2.1.0.

If you are using WSO2 IS as a Key Manager, follow the instructions mentioned in [Upgrading from the Previous Release when WSO2 IS is the Key Manager](#).

- Step 1 - Migrate the configurations
 - Step 1.1 - Migrate the API Manager configurations
 - Step 1.2 - Migrate the statistics related data from WSO2 DAS to API Manager Analytics
- Step 2 - Upgrade API Manager to 2.1.0

Step 1 - Migrate the configurations

Step 1.1 - Migrate the API Manager configurations

Do not copy entire configuration files from the current version of API Manager to the new one as some configuration files (for example, api-manager.xml) may have changed. Instead, redo the configuration changes in the new configuration files.

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

1. Back up all databases in your API Manager instances along with the synapse configs of all the tenants and super tenant.
- You find the synapse configs of the super tenant in the <CURRENT_API-M_HOME>/repository/deployment directory. Synapse configs of tenants are in the <CURRENT_APIM_HOME>/repository/tenants directory.

If you use a **clustered/distributed API Manager setup**, back up the available configurations in the API Gateway node.

2. Download API Manager 2.1.0 from <http://wso2.com/products/api-manager/>.
3. Open the <API-M_2.1.0_HOME>/repository/conf/datasources/master-datasources.xml file and provide the datasource configurations for the following databases. You can copy the configuration values from the same file in the current API Manager instance already being used.
 - User Store
 - Registry database/s
 - API Manager Databases
 - Statistics Database (If Statistics are configured.)
4. Edit the registry configurations in the <API-M_2.1.0_MANAGER_HOME>/repository/conf/registry.xml file and the user database in the <API-M_2.1.0_MANAGER_HOME>/repository/conf/user-mgt.xml file similar to the configurations of the current API Manager.

In a **clustered/distributed API Manager setup**, carryout steps 5 and 6 on the Gateway node.

- Move all your synapse configurations except the files mentioned below by copying and replacing the <CURRENT_API_M_HOME>/repository/deployment/server/synapse-config directory to the <API-M_2.1.0_MANAGER_HOME>/repository/deployment/server/synapse-config

NOTE: Do not replace the files listed below from the <CURRENT_API-M_HOME>/repository/deployment/server/synapse-config directory to APIM 2.1.0. These are application-specific APIs and sequences. If you made any custom changes to the files below, please merge them to the corresponding files in 2.1.0.

```
/api/_AuthorizeAPI_.xml
/api/_RevokeAPI_.xml
/api/_TokenAPI_.xml
/api/_UserInfoAPI_.xml
/sequences/_auth_failure_handler_.xml
/sequences/_build_.xml
/sequences/_cors_request_handler_.xml
/sequences/fault.xml
/sequences/main.xml
/sequences/_production_key_error_.xml
/sequences/_resource_mismatch_handler_.xml
/sequences/_sandbox_key_error_.xml
/sequences/_throttle_out_handler_.xml
/sequences/_token_fault_.xml
/proxy-services/WorkflowCallbackService.xml
```

- Move all your tenant synapse configurations by updating the configurations made in the <CURRENT_API-M_HOME>/repository/tenants directory to the <API-M_2.1.0_MANAGER_HOME>/repository/tenants directory.

NOTE: Get the files listed below from the <API_M_2.1.0_MANAGER_HOME>/repository/deployment/server/synapse-config directory and replace the corresponding files in the <API_M_2.1.0_MANAGER_HOME>/repository/tenants directory.

```
_auth_failure_handler_.xml
_build_.xml
_cors_request_handler_.xml
fault.xml
main.xml
_production_key_error_.xml
_resource_mismatch_handler_.xml
_sandbox_key_error_.xml
_throttle_out_handler_.xml
_token_fault_.xml
```

In a **clustered/distributed API Manager setup**, carryout this step on the Gateway node.

Step 1.2 - Migrate the statistics related data from WSO2 DAS to API Manager Analytics

From WSO2 API Manager 2.0.0 onwards, statistics can be configured only for RDBMS since the API Manager 1.10.0 REST based analytic configuration no longer exist. This section walks you through how to migrate statistics

data from previous versions of API Manager to 2.1.0.

If you have configured analytics using RDBMS,

1. Take a backup of the API Manager statistics DB.
2. Add the API Manager statistics DB as a datasource in WSO2 API Manager Analytics.

If you configured analytics using the REST client in APIM 1.10.0 with DAS 3.0.x,

1. Configure analytics using API Manager 1.10.0 and DAS 3.0.x with the RDBMS client according to the instructions in [Publishing API Runtime Statistics Using RDBMS](#).
2. Wait for data to appear on the RDBMS and API Manager statistics dashboard.
3. Replace the Statistics DB (STAT_DB) configurations, which is configured in WSO2 DAS for API Manager 1.10.0, in the `<API-M_ANALYTICS_2.1.0_HOME>/repository/conf/stats-datasources.xml` file.

Finally, execute relevant db script found in `migration-scripts/110-200-migration/stat/` on the STAT_DB

Once done, configure analytics in API Manager 2.1.0 according to the instructions in [Configuring APIM Analytics](#).

Step 2 - Upgrade API Manager to 2.1.0

1. Stop all running WSO2 API Manager server instances.
2. Make sure you backed up all the databases and synapse configs as instructed in step 1 of the previous section.
3. Before you run the migration client, open the `<API-M_2.1.0_HOME>/repository/conf/datasources/master-datasources.xml` file, and set the `<username>`, and `<password>` elements of the AM_DB JNDI to that of a user who has permissions to alter tables in the database.

Tip: After you are done running the migration client, you can switch these credentials back to a user with lesser privileges.

For example,

```
<datasource>
    ...
    <definition type="RDBMS">
        <configuration>
            ...
            <username>xxxxxx</username>
            <password>xxxxxx</password>
            ...
        </configuration>
    </definition>
</datasource>
```

4. Upgrade the Identity component in WSO2 API Manager from version 5.1.0 to 5.2.0.

a. Download the migration DB scripts for version 5.1.0 to 5.2.0 from [here](#).

b. Unzip and copy it to the `<API-M_HOME>/dbscripts/identity` directory.

Copying the DB scripts are optional as it is done for reference purposes to indicate the version to which you have upgraded your APIM IS component.

- c. Execute the DB scripts, which corresponds to the RDBMS that you are working with, manually.

Apply the respective DB script that is inside the following directories against the respective DB as follows:

DB script directory
<wso2is-5.2.0-migration>/dbscripts/identity/migration-5.1.0_to_5.2.0
<wso2is-5.2.0-migration>/dbscripts/migration-5.1.0_to_5.2.0

For example, if you are working with a MySQL DB run the `mysql.sql` script.

It is important to complete the WSO2 API Manager Identity component related migration from version 5.1.0 to 5.2.0 first, **before** carrying out the WSO2 API-M migration from WSO2 1.10.0 to WSO2 API-M 2.0.0 to avoid the following error.
`com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Table 'WSO2A`

5. Download and extract the [WSO2 API Manager Migration Client](#) and do the following to upgrade your version of WSO2 API Manager to 2.0.0:
- Copy the `org.wso2.carbon.apimgt.migrate.client-2.0.X.jar` file to the `<API-M_2.1.0_HOME>/migrate` directory. If you use a **clustered/distributed API Manager setup**, copy the JAR file to a Publisher and a Gateway nodes.
 - Copy the `migration-script` folder into `<API-M_2.1.0_HOME>`. If you use a **clustered/distributed API Manager setup**, copy the `migration-script` folder to a Publisher node.

- i. Start API Manager 2.1.0 with the following command-line options to migrate the database, registry and the file system separately in the given order.

Description
To migrate the database only. This migrates the AM_DB database. Please ensure that the <API-M_HOME>/repository/conf/dbscripts/migration script file has an entry for AM_DB.
To migrate the registry only. This migrates the registry-related resources such as .rxt and swagger definitions.
To migrate the file system only. This migrates the synapse config files such as APIs that reside in the file system. Therefore, you must run this command on the Gateway node/s of a distributed API Manager setup.
To migrate to the new throttling engine and generate throttle policies. This migrates synapse config files, the APIM database with new throttle policies and generates throttle policies to the <API-M_HOME>/executionplans directory. Therefore, you must run this command against a node that has synapse config files and the AM_DB. After the migration, copy the <APIM_HOME>/repository/deployment/server/executionplans directory of the APIM node to the repository/deployment/server/executionplans directory of the Traffic Manager node.
<p>Please note that you need to use either this command or the one below in order to migrate to the new throttling engine of APIM 2.0.0.</p>
<p>To migrate to the new throttling engine and deploy throttle policies to the Traffic Manager node. This migrates synapse config files, the API-M database with new throttle policies and deploys policies to the Traffic Manager node. Therefore, you must run this command against a node that has synapse config files, the AM_DB and is pointed to the Traffic Manager node.</p>
<p>Please use this command if you want to deploy the throttle policies directly to the Traffic Manager node while performing the migration.</p>
<p>If you are using a clustered/distributed API Manager setup, run with the following options <code>-DmigrateDB=true -DmigrateReg=true -Dcomponent=apim -DmigrateFS=true</code> in the API Publisher node and the <code>-DmigrateFS=true -Dcomponent=apim -DmigrateDB=true</code> options in the API Gateway node.</p>
<p>To migrate to the new throttling engine of API-M 2.1.0, you have to use a node that has AM_DB pointed as well as API gateway synapse configurations in place in the file system. If you only use the "migrateThrottling" option, you can use the API Gateway node. Make sure that AM_DB is pointed to the gateway at the time of migration. You can start the gateway server with the <code>-Dcomponent=apim -DmigrateFromVersion=1.10.0 -DmigrateThrottling=true</code> options.</p>

command. After completing this execution, make sure you copy the generated throttle policies in the <API-M_HOME>/executionplans directory to the <API-M_TRAFFIC_MANAGER_HOME>/throttle directory.

If you use both the options -migrateThrottling,deployPolicies, make sure that the Traffic Manager node is configured with the Gateway node and also make sure you avoid passing the -Dprofile option while performing the migration. When using these options, you do not have to manually copy the throttling policies to the Traffic Manager. You can start the gateway server with -Dcomponent=apim -DmigrateFrom command.

Troubleshooting

If you have [enabled token encryption](#) in your database, as part of database migration, those consumer keys need to be decrypted. If you see the error below, it means that the migration client has failed to decrypt some of the keys in the old database.

```
TID: [-1234] [] [2016-04-11 08:27:16,249] ERROR
{org.wso2.carbon.apimgt.migration.client.MigrateFrom1
9to110} - Cannot decrypt consumer key :
DRHbt68uSU4+7xtCEIzuf42CMaqXbNjYl3OYVJ0VL/H6EsFo8GBRa
ZGUhLH1TWIHzYrvoeOpb1YbauvRRIN/b6VqEd9m8HuYOIuLkkDd
AM_APPLICATION_KEY_MAPPING table
{org.wso2.carbon.apimgt.migration.client.MigrateFrom1
9to110}
```

If this error occurs, by default, the migration client will terminate without updating databases in order to maintain database integrity. However, you can change this behavior by adding the following argument and running the API-M migration client again from the beginning. **Please note that the database is then updated with the keys where decryption was successful and failed keys are permanently deleted.**

-DremoveDecryptionFailedKeysFromDB=true

Expected Errors

The following are some errors that you will come across. **Do not get alarmed** as these errors are to be expected since you are yet to complete the migration process.

- You may notice the following exception being thrown in the console when API Manager 2.1.0 is started at this point,

```
org.wso2.carbon.idp.mgt.IdentityProviderManagementException: Error occurred while retrieving Identity Provider information for tenant : carbon.super and Identity Provider name : LOCAL
```

This is due to the fact that the `IDP_METADATA` table does not exist in the API Manager database. The instructions given from **step 9** onwards addresses the creation of the `IDP_METADATA` table after which this exception will no longer be thrown.

- You may notice gateway artifact deployment related errors. The following is one such exception that is thrown in the console when WSO2 API Manager 2.1.0 is started at this point.

```
Error loading class :  
org.wso2.carbon.apimgt.usage.publisher.APIMgtFaultHandler - Class not found  
{org.apache.synapse.config.xml.ClassMediatorFactory}
```

These errors get eliminated after executing the Gateway artifact migration script in **step 6**.

6. Run the respective migration script based on your environment to carryout the migration process.

`Linux/Mac OS`

Run the `apim200_to_apim210_gateway_artifact_migrator.sh` script as shown below to migrate from API Manager 2.0.0 to 2.1.0.

You need to run this script for the super tenant path and for each of the tenant paths.

```
./apim200_to_apim210_gateway_artifact_migrator.sh  
<API-definitions-path>
```

`<API-definition-path>` - This is the location where the WSO2 API-M 2.1.0 API definitions reside.

The API definition paths `<API-definition-path>` are as follows:

- Super Tenant - `<API-M_2.1.0_HOME>/repository/deployment/server/synapse-configs/default`
- Tenant - `<API-M_2.1.0_HOME>/repository/tenants/<tenant-id>/synapse-configs/default`

Where `<API-M_2.1.0_HOME>` can be for example `/Users/user12/Documents/wso2am-2.1.0`

Run the PowerShell script (`apim200_to_apim210_gateway_artifact_migrator.ps1`) as follows:

- Open a Windows command prompt and type the following command.

```
powershell
```

A message about PowerShell appears, and the shell changes to PS.

- Run the powershell script by passing the location of the gateway artifacts that you need to migrate. You can do this in the home directory of the existing API-M 2.0.0 installation.

You need to run this script for the super tenant path and for each of the tenant paths.

```
.\apim200_to_apim210_gateway_artifact_migrator.ps1  
<API-definitions-path>
```

<API-definition-path> - This is the location where the WSO2 API-M 2.1.0 API definitions reside.

The API definition paths <API-definition-path> are as follows:

- Super Tenant - <API-M_2.1.0_HOME>/repository/deployment/server/synapse-configs/default
- Tenant - <API-M_2.1.0_HOME>/repository/tenants/<tenant-id>/synapse-configs/default

Where <API-M_2.1.0_HOME> can be for example /Users/user12/Documents/wso2am-2.1.0

It may take a considerable amount of time, which is proportionate to the amount of artifacts, to complete the migration process.

Troubleshooting

Why do I get the following error - apim200_to_apim210_gateway_artifact_migrator.ps1 cannot be loaded because the execution of scripts is disabled on this system?

When running the apim200_to_apim210_gateway_artifact_migrator.ps1 script, if the execution process is aborted with the above error, it means that the execution of unknown scripts are disabled in the system. To overcome this issue run the following command in the terminal/command-line as the Administrator to allow the execution of such scripts.

```
Set-ExecutionPolicy RemoteSigned
```

- Do the following to re-index the artifacts in the registry:

- Rename the <lastAccessTimeLocation> element in the <API-M_2.1.0_HOME>/repository /c file. If you use a **clustered/distributed API Manager setup**, change the file in the API Publisher

- node. For example, change the `/_system/local/repository/components/org.wso2.carbon.registry` path to `/_system/local/repository/components/org.wso2.carbon.registry/in`
- Shut down API Manager 2.1.0, backup and delete the `<API-M_2.1.0_HOME>/solr` directory.
- Upgrade the Identity component in WSO2 API Manager from version 5.2.0 to 5.3.0.
 - Download the 5.2.0 to 5.3.0 migration DB scripts from [here](#).
 - Unzip and find the correct DB script in `<MIGRATION_SCRIPT_HOME>/wso2is-5.3.0-migration/dbscripts/identity/migration-5.2.0_to_5.3.0` directory.
 - Execute the corresponding DB script against the `WSO2AM_DB` database manually.
 - Copy the `/wso2is-5.3.0-migration/dropins/org.wso2.carbon.is.migrate.client-5.3.0.jar` file into the `<APIM_2.1.0_HOME>/repository/components/dropins` directory.
 - Start API Manager 2.0.0 with the command line option `-Dmigrate -Dcomponent=identity` to carry out the complete Identity and User Store DB migration.

This concludes the upgrade process.

Tip 1: The migration client that you use in this guide automatically migrates your tenants, workflows, external user stores etc. to the upgrade environment. There is no need to migrate them manually.

Tip 2: If you are using SVN based deployment synchronizer, start with a clean SVN repository and point the new deployment's nodes to the new SVN repository. Also, any existing .svn directories in the new deployment's `<API-M_2.1.0_HOME>/repository/deployment/server/synapse-configs/default` and `<API-M_2.1.0_HOME>/repository/tenants/<tenant-id>/synapse-configs/default` locations should be removed before starting the servers. See [Configuring Deployment Synchronization](#) for more details.

Follow the instructions below to upgrade your WSO2 API Manager server **from APIM 1.8.0/1.9.0/1.9.1 to 2.1.0**.

If you are using WSO2 IS as a Key Manager, follow the instructions mentioned in [Upgrading from the Previous Release when WSO2 IS is the Key Manager](#).

Step 1 - Upgrade WSO2 API Manager to 2.0.0

It is not possible to directly upgrade from WSO2 API Manager 1.8.0/1.9.0/1.9.1 to 2.1.0.

Upgrade your current WSO2 API-M version (1.8.0/1.9.0/1.9.1) to WSO2 API-M 2.0.0.

Step 2 - Upgrade WSO2 API Manager to 2.1.0

After you have successfully migrated your current WSO2 API-M version to 2.1.0, upgrade from API-M 2.0.0 to API-M 2.1.0. For more information, click below:

[Upgrading from API-M 2.0.0 to API-M 2.1.0](#)

This concludes the upgrade process.

Upgrading from the Previous Release when WSO2 IS is the Key Manager

The following information describes how to upgrade your **WSO2 API Manager (WSO2 API-M)** environment **from**

APIM 1.8.0/1.9.0/1.9.1/1.10.0/2.0.0 to 2.1.0 when WSO2 Identity Server (WSO2 IS) is the Key Manager.

Upgrading from 2.0.0 to 2.1.0
Upgrading from 1.10.0 to 2.1.0
Upgrading from 1.8.0/1.9.0/1.9.1 to 2.1.0

If you wish to upgrade your APIM environment from **API-M 2.0.0 to 2.1.0**, which is using the **internal WSO2 Identity Server (WSO2 IS) capabilities**, follow the instructions in [Upgrading from the Previous Release](#).

Follow the instructions below to upgrade WSO2 API-M **from WSO2 API-M 2.0.0 to 2.1.0** when using **WSO2 IS** as the **Key Manager**:

1. Download the [pre-packaged WSO2 Identity Server 5.3.0 - Key Manager](#).

2. Migrate the WSO2 Identity Server from version 5.2.0 to 5.3.0.

For more information, see [Upgrading from a Previous Release](#) in the WSO2 Identity Server 5.3.0 documentation.

3. Migrate the configuration of the Key Manager.

Compare the old `<IS_HOME>/repository/conf/api-manager.xml` file with the latest `api-manager.xml` file in WSO2 IS 5.3.0, and redo the configuration changes in the new configuration files.

This completes the WSO2 IS migration process. Now, you can proceed with the WSO2 APIM migration.

4. Migrate WSO2 API-M from 2.0.0 to 2.1.0.

Follow the instructions mentioned in the Upgrading from 2.0.0 to 2.1.0 tab, which is in [Upgrading from the Previous Release](#), but **skip step 2 - (5)**, which explains how to migrate the APIM Identity component.

If you wish to upgrade your APIM environment from 1.10.0 to 2.1.0, which is using the **internal WSO2 Identity Server (WSO2 IS) capabilities** that exists in WSO2 API Manager (**WSO2 API-M**), go to [Upgrading from the Previous Release](#).

Follow the instructions below to upgrade WSO2 API-M **from WSO2 API-M 1.10.0 to 2.1.0** when using **WSO2 IS** as the **Key Manager**:

1. Upgrade the **WSO2 Identity Server** from version **5.1.0 to 5.2.0**.

- a. Download the [pre-packaged WSO2 Identity Server 5.2.0 - Key Manager](#).

- b. Migrate the WSO2 Identity Server from version 5.1.0 to 5.2.0.

For more information, see [Upgrading from a Previous Release](#) in the WSO2 Identity Server 5.2.0 documentation.

Expected Errors

You will come across the following error. Do not get alarmed as this error is to be expected since you are

yet to complete the migration process.

```
ERROR {org.wso2.carbon.apimgt.impl.dao.ApiMgtDAO} - Failed
to check is exist: 50PerMin--1234
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException:
Table 'apimgt.AM_POLICY_APPLICATION' doesn't exist
```

You should complete upgrading **WSO2 Identity Server** from version **5.1.0 to 5.2.0** (step 1) to start upgrading the **WSO2 Identity Server** from version **5.2.0 to 5.3.0** (step 2).

2. Upgrade the **WSO2 Identity Server** from version **5.2.0 to 5.3.0**.

- Download the pre-packaged **WSO2 Identity Server 5.3.0 - Key Manager**.
- Migrate the **WSO2 Identity Server** from version 5.2.0 to 5.3.0.

For more information, see [Upgrading from a Previous Release in the WSO2 Identity Server 5.3.0 documentation](#).

- Migrate the configuration of the Key Manager.

Compare the old <IS_HOME>/repository/conf/api-manager.xml file with the latest api-manager.xml file in WSO2 IS 5.3.0, and redo the configuration changes in the new configuration files.

This completes the IS migration process. Now, you can proceed with the WSO2 API Manager migration.

3. Migrate **WSO2 API-M** from **1.10.0** to **2.1.0**.

Follow the instructions mentioned under **Upgrading from 1.10.0 to 2.1.0**, which is in [Upgrading from the Previous Release](#), but skip step 2 - (4) and (8), which explains how to migrate the APIM Identity component.

If you wish to upgrade your APIM environment from 1.8.0/1.9.0/1.9.1 to 2.1.0, which is using the **internal WSO2 Identity Server (WSO2 IS) capabilities** that exists in WSO2 API Manager (**WSO2 API-M**), go to [Upgrading from the Previous Release](#).

Follow the instructions below to upgrade WSO2 API-M from **WSO2 API-M 1.8.0/1.9.0/1.9.1 to 2.1.0** when using **WSO2 IS** as the **Key Manager**:

1. Upgrade the **WSO2 Identity Server** from version **5.0.0 to 5.1.0**.

- Download the pre-packaged **WSO2 Identity Server 5.1.0 - Key Manager**.
 - Migrate the **WSO2 Identity Server** from version 5.0.0 to 5.1.0.
- For more information, see [Upgrading from a Previous Release in the WSO2 Identity Server 5.1.0 documentation](#).

2. Upgrade the **WSO2 Identity Server** from version **5.1.0 to 5.2.0**.

- Download the pre-packaged **WSO2 Identity Server 5.2.0 - Key Manager**.
 - Migrate the **WSO2 Identity Server** from version 5.1.0 to 5.2.0.
- For more information, see [Upgrading from a Previous Release in the WSO2 Identity Server 5.2.0 documentation](#).

Expected Errors

You will come across the following error. Do not get alarmed as this error is to be expected since you are yet to complete the migration process.

```
ERROR {org.wso2.carbon.apimgt.impl.dao.ApiMgtDAO} - Failed
to check if exist: 50PerMin--1234
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException:
Table 'apimgt.AM_POLICY_APPLICATION' doesn't exist
```

3. Upgrade the WSO2 Identity Server from version 5.2.0 to 5.3.0.

- Download the pre-packaged WSO2 Identity Server 5.3.0 - Key Manager.
- Migrate the WSO2 Identity Server from version 5.2.0 to 5.3.0.

For more information, see [Upgrading from a Previous Release](#) in the WSO2 Identity Server 5.3.0 documentation.

- Migrate the configuration of the Key Manager.

Compare the old <IS_HOME>/repository/conf/api-manager.xml file with the latest api-manger.xml file in WSO2 IS 5.3.0, and redo the configuration changes in the new configuration files.

This completes the IS migration process. Now, you can proceed with the WSO2 API Manager migration.

4. Migrate WSO2 API-M from 1.8.0/1.9.0/1.9.1 to 2.1.0.

- Migrate WSO2 API-M from 1.8.0/1.9.0/1.9.1 to 2.0.0.

Follow the instructions mentioned in [Upgrading from the Previous Release](#), but skip steps (7) and (10).

- Migrate WSO2 API-M from 2.0.0 to 2.1.0.

Follow the instructions mentioned under **Upgrading from 2.0.0 to 2.1.0**, which is in [Upgrading from the Previous Release](#), but skip step 2 - (5), which explains how to migrate the APIM Identity component.

Deploying WSO2 API Manager

In a typical production environment, you set up the different WSO2 API Manager (WSO2 API-M) components (API Publisher, Store, Gateway, Key Manager, and Traffic Manager) in separate servers so that you can scale them independently. You also install multiple instances of a component in a cluster to ensure proper load balancing. When one node becomes unavailable or is experiencing high traffic, another node handles the requests.

The following topics explain the deployment aspect of WSO2 API Manager in detail.

- Deployment Patterns
- Deploying API Manager using Single Node Instances

- Using Puppet Modules to Set up WSO2 API-M
- Distributed Deployment of API Manager
- Configuring WSO2 Identity Server as a Key Manager
- Deploying API Manager with Kubernetes or OpenShift Resources
- Configuring Admin App Event Publishing for Traffic Manager HA Setup
- Minimum High Availability Deployment for WSO2 APIM Analytics
- Configuring rsync for Deployment Synchronization

Deployment Patterns

WSO2 API Manager includes five main components as the Publisher, Store, Gateway, Traffic Manager and Key Manager. In a stand-alone APIM setup, these components are deployed in a single server. However, in a typical production setup, they need to be deployed in separate servers for better performance. Installing and configuring each or selected component/s in different servers is called a distributed setup.

Note: It is recommended to separate the worker and manager nodes in scenarios where you have multiple Gateway nodes. See [Separating the Worker and Manager Nodes](#) for information on why it is advantageous to separate the worker and manager nodes.

This topic includes the following sections.

- Main components of a distributed setup
- WSO2 API Manager deployment patterns

Main components of a distributed setup

The following diagram illustrates the main components of an API Manager distributed deployment.

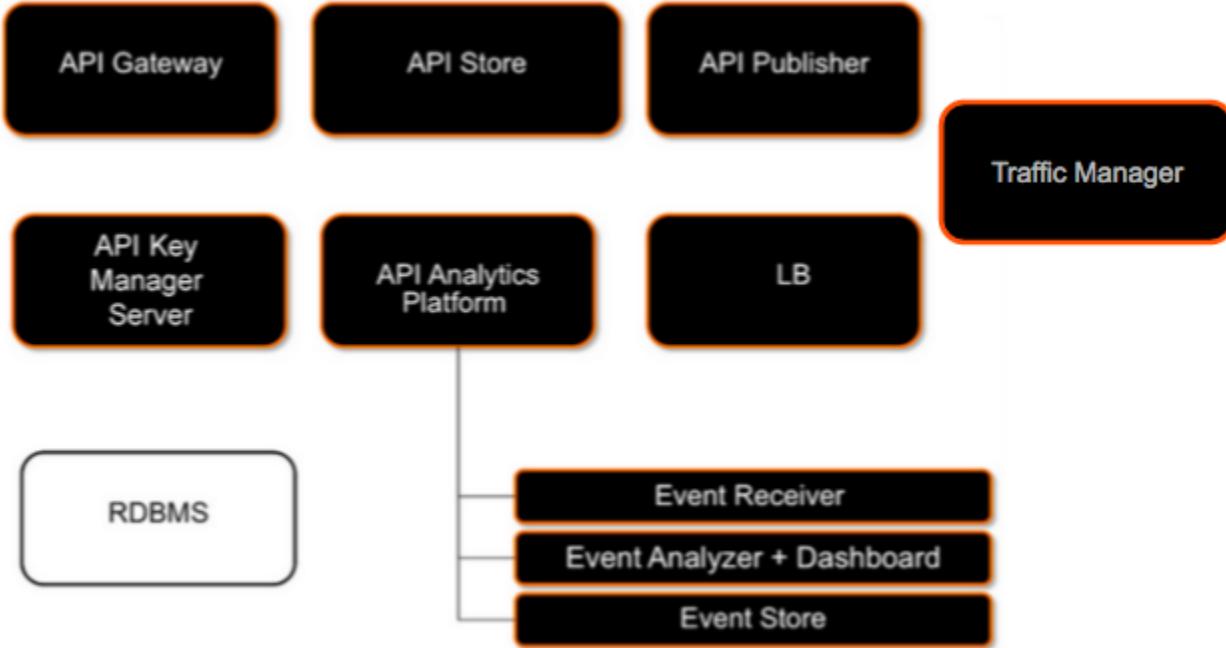


Figure: Main components of an APIM distributed setup

Let's take a look at each component in the above diagram. For more information on these components, see [Architecture](#).

Component	Description
-----------	-------------

API Gateway	This component is responsible for securing, protecting, managing, and scaling API calls. The API gateway 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. We use a set of handlers for security validation and throttling purposes in the API Gateway. Upon validation, it will pass Web service calls to the actual back end. If it is a token request call, it will then directly pass the call to the Key Manager Server to handle it.
API Store	This component provides a space for consumers to self-register, discover API functionality, subscribe to APIs, evaluate them, and interact with API publishers. Users can view existing APIs and create their own application by bundling multiple APIs together into one application.
API Publisher	This component enables API providers to easily publish their APIs, share documentation, provision API keys, and gather feedback on API features, quality, and usage. You can create new APIs by pointing to the actual back end service and also define rate-limiting policies available for the API.
API Key Manager Server	<p>This component is responsible for all security and key-related operations. When an API call is sent to the Gateway, it calls the Key Manager server and verifies the validity of the token provided with the API call. If the Gateway gets a call requesting a fresh access token, it forwards the username, password, consumer key, and consumer secret key obtained when originally subscribing to the API to the Key Manager. All tokens used for validation are based on OAuth 2.0.0 protocol. All secure authorization of APIs is provided using the OAuth 2.0 standard for Key Management. The API Gateway supports API authentication with OAuth 2.0, and it enables IT organizations to enforce rate limits and throttling policies for APIs by consumer.</p> <p>Login/Token API in the Gateway node should point to the token endpoint of Key Manager node. The token endpoint of the Key Manager node is at https://localhost:9443/oauth2endpoints/token. In a distributed setup, it should be <code>https://<IP of key manager node>:<port-with-offset-of-keymanager-node>/oauth2endpoints/token</code>.</p> <div style="border: 1px solid #ccc; padding: 10px;"> <p>In a clustered environment, you use session affinity to ensure that requests from the same client always get routed to the same server.</p> <p>Session affinity is not mandatory when configuring a Key Manager cluster with a load balancer. However, authentication via session ID fails when session affinity is disabled in the load balancer.</p> <p>The Key Manager first tries to authenticate the request via the session ID. If it fails, the Key Manager tries to authenticate via basic authentication.</p> </div>
API Traffic Manager	The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests.
LB (load balancers)	The distributed deployment setup depicted above requires two load balancers. We set up the first load balancer, which is an instance of NGINX Plus, internally to manage the cluster. The second load balancer is set up externally to handle the requests sent to the clustered server nodes, and to provide failover and autoscaling. As the second load balancer, you can use an instance of NGINX Plus or any other third-party product.

RDBMS (shared databases)	<p>The distributed deployment setup depicted above shares the following databases among the APIM components set up in separate server nodes.</p> <ul style="list-style-type: none"> User Manager Database : Stores information related to users and user roles. This information is shared among the Key Manager Server, Store, and Publisher. Users can access the Publisher for API creation and the Store for consuming the APIs. API Manager Database : Stores information related to the APIs along with the API subscription details. The Key Manager Server uses this database to store user access tokens required for verification of API calls. Registry Database : Shares information between the Publisher and Store. When an API is published through the Publisher, it is made available in the Store via the sharing registry database.
--------------------------------	--

Message flows

The three main use cases of API Manager are API publishing, subscribing and invoking. Described below is how the message flow happens in these use cases.

- Publishing APIs**

A user assigned to the **publisher role** can publish APIs. This is done via the Publisher server. When an API is published in the API Publisher, it will be available in the API Store. Furthermore, the API Gateway must be updated with this API so that users can invoke it. As we are using a clustered Gateway, all Gateway server nodes in the cluster are updated with the published API details, enabling any of these Gateway nodes to serve API calls that are received. When an API is published, it is also pushed to the registry database, so that it can be made available on the store via the shared database.

- Subscribing to APIs**

A user with the **subscriber role** logs into the API Store and subscribes to an API. The user must then generate an access token to be able to invoke the API. When the subscriber requests to generate the token, a request is sent to the Key Manager Server cluster. The token is then generated, and the access token details are displayed to the subscriber via the Store.

- I n v o k i n g A P I s**

Subscribed users can invoke an API to which they have subscribed. When the API is invoked, the request is sent to the API Gateway server cluster. The Gateway server then forwards the request to the Key Manager server cluster for verification. Once the request is verified, the Gateway connects to the back-end implementation and obtains the response, which is sent back to the subscriber via the Gateway server.

The diagram below summarizes these message flows where the Publisher is referred to as the publishing portal and the Store is referred to as the developer portal.

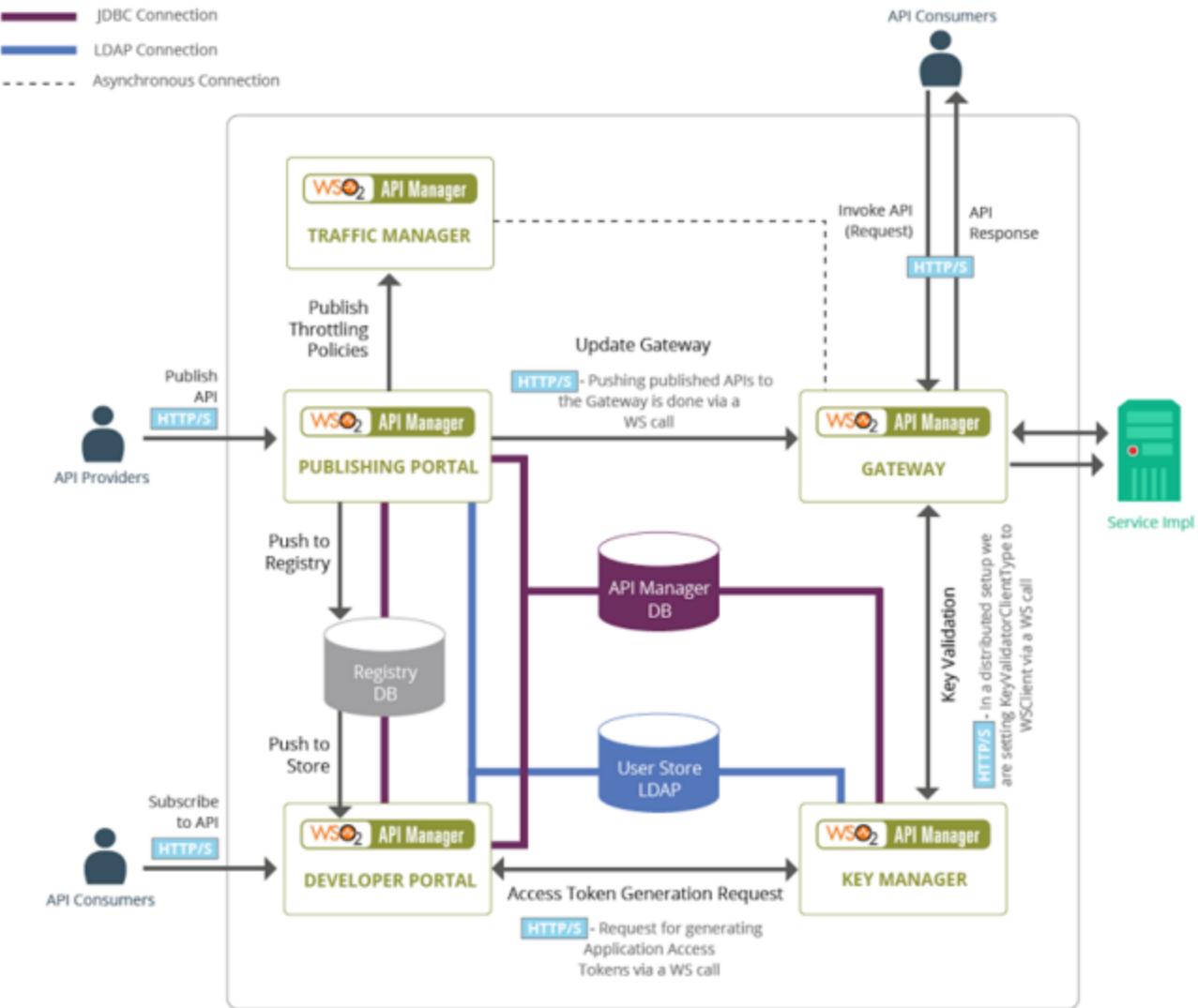


Figure: WSO2 API Manager message flows

WSO2 API Manager deployment patterns

Note the following:

- The following 5 patterns illustrate the **latest deployment patterns** for WSO2 API Manager. We **have NOT yet released the Puppet Modules for these 5 new patterns**. These deployment patterns have been designed by reviewing and refining the **7 older API-M deployment patterns** in order to provide a more simplified set of patterns that meet the most required production use cases.
- If you want to try out API-M deployment with Puppet, you need to use one of the older API-M deployment patterns. For more information, see [Using Puppet Modules to Set up WSO2 API-M](#).

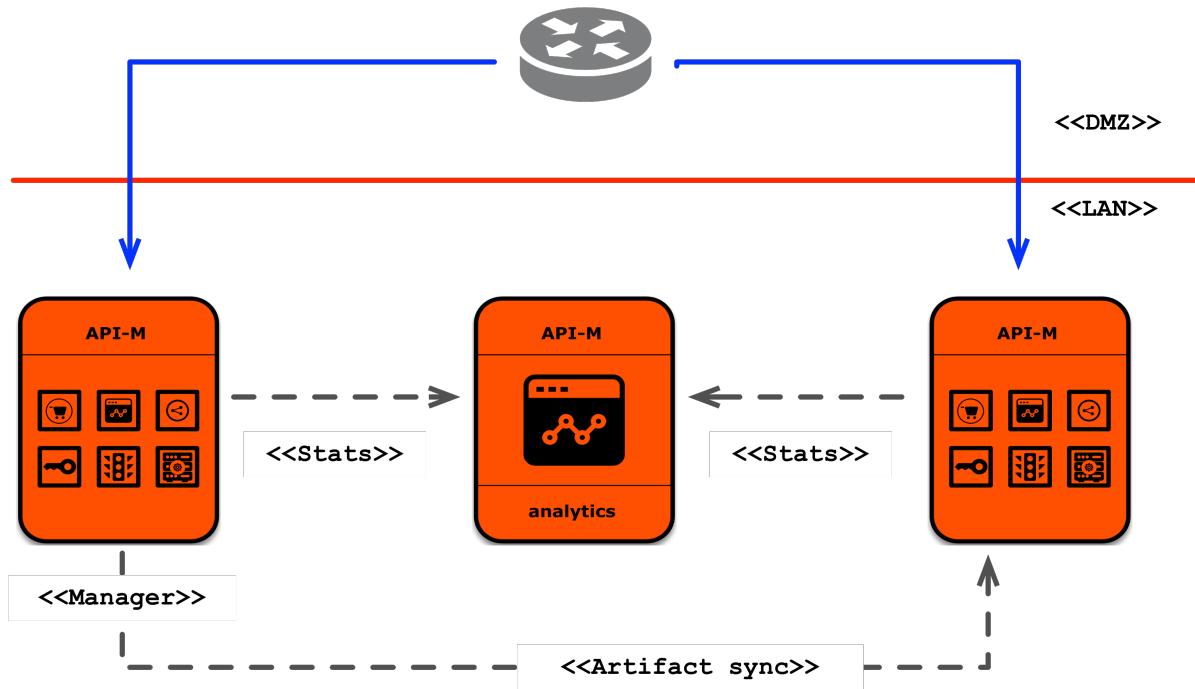
- Latest deployment patterns
- Older deployment patterns

Latest deployment patterns

[[Pattern 1](#)] [[Pattern 2](#)] [[Pattern 3](#)] [[Pattern 4](#)] [[Pattern 5](#)]

Pattern 1

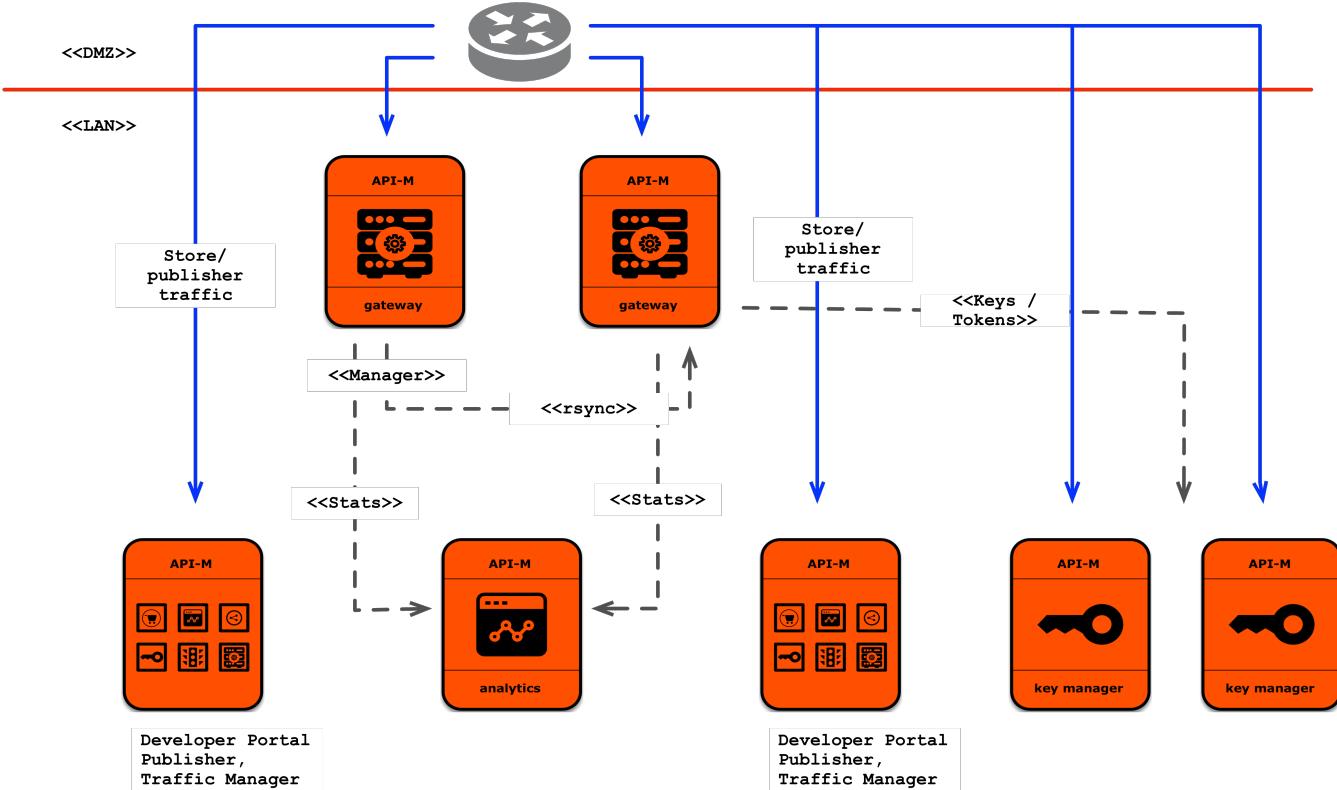
Single node (all-in-one) deployment.



You can use this pattern when you are working with a low throughput.

Pattern 2

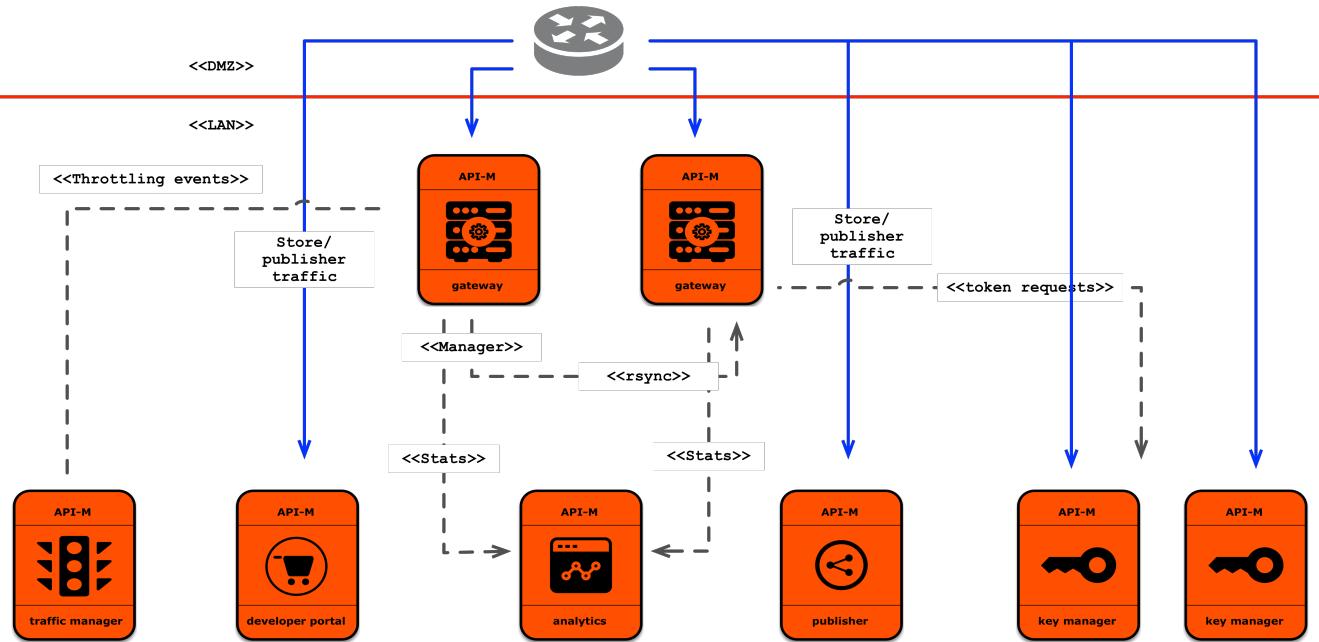
Deployment with a separate Gateway and separate Key Manager.



You can use this pattern when you require a high throughput scenario that requires a shorter token lifespan.

Pattern 3

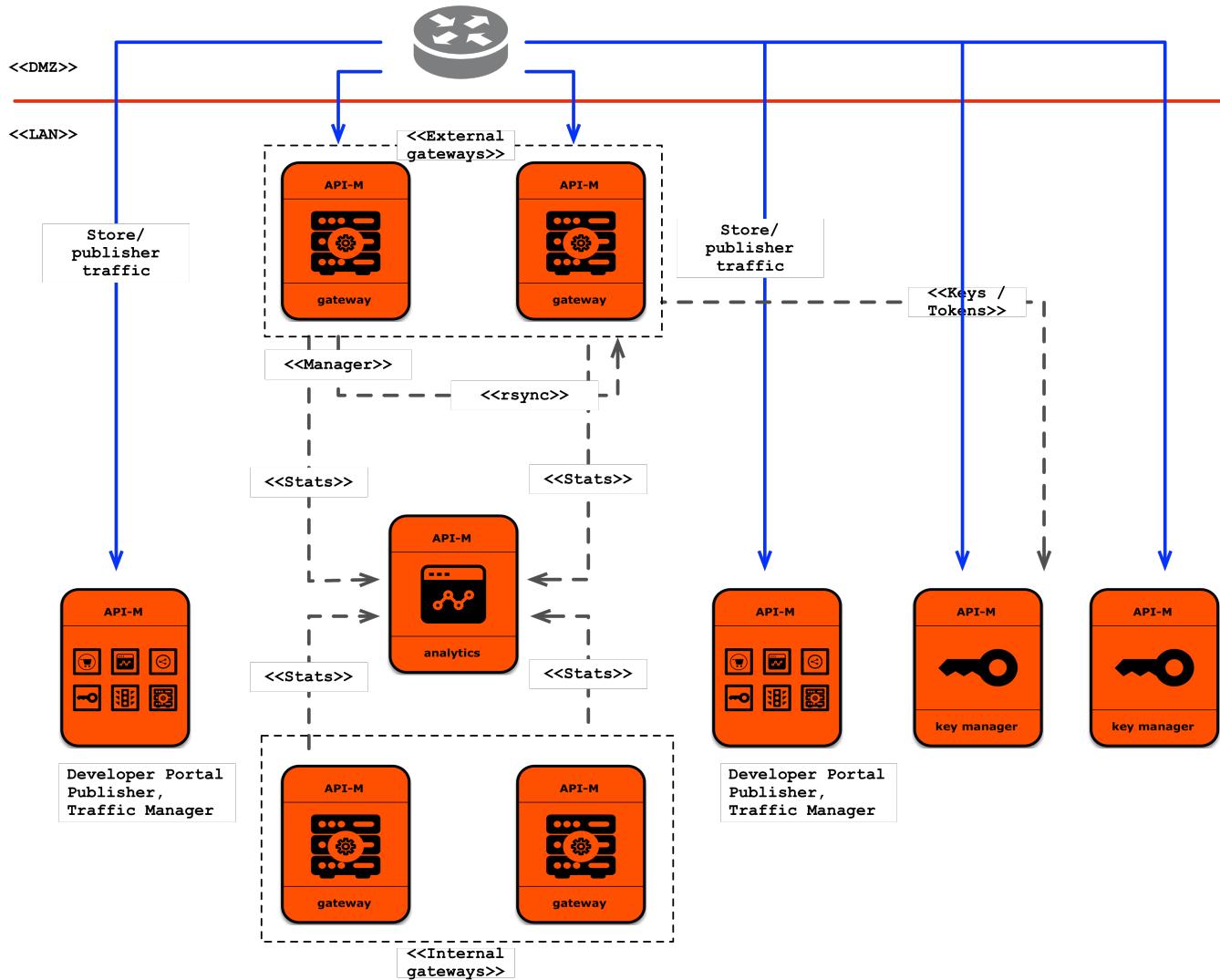
Fully distributed setup.



You can use this pattern to maintain scalability at each layer and higher flexibility at each component.

Pattern 4

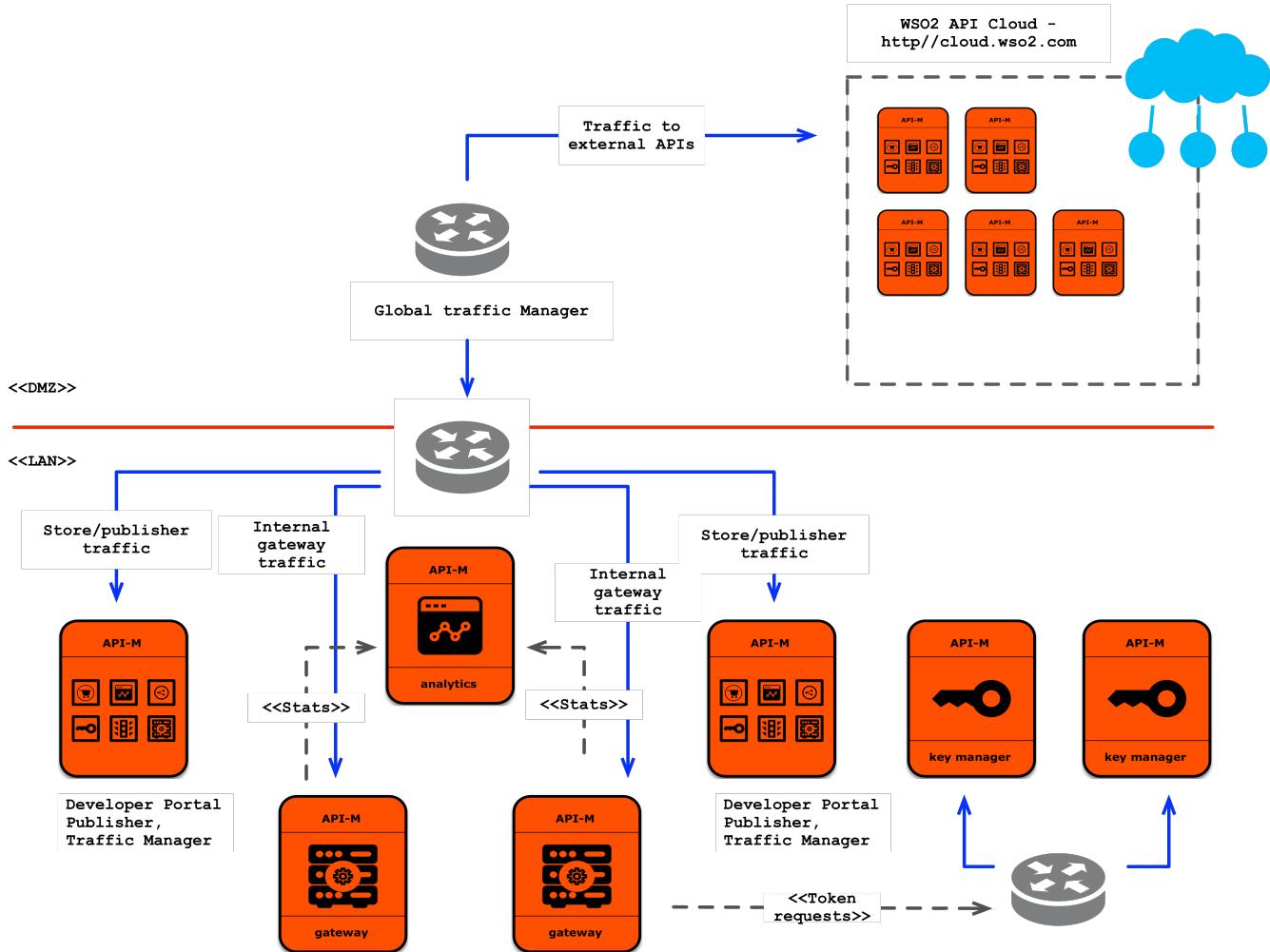
Internal and external (on-premise) API Management.



You can use this pattern when you require a separate internal and external API Management with separated Gateway instances.

Pattern 5

Internal and external (public and private cloud) API Management.



You can use this pattern when you wish to maintain a cloud deployment as an external API Gateway layer.

Older deployment patterns

[[Pattern 0](#)] [[Pattern 1](#)] [[Pattern 2](#)] [[Pattern 3](#)] [[Pattern 4](#)] [[Pattern 5](#)] [[Pattern 6](#)] [[Pattern 7](#)]

Pattern 0

Single node deployment.



Figure: All-in-one instance

This pattern consists of a stand-alone API-M setup with a single node deployment. This pattern uses the embedded H2 databases.

Pattern 1

Single node deployment.



Figure: All-in-one instance

This pattern consists of a stand-alone WSO2 API-M setup with a single node deployment. This pattern uses external RDBMS (e.g., MySQL databases). The only difference between pattern-0 and pattern-1 is that pattern-0 uses embedded H2 databases and pattern-1 is configured to use external RDBMS.

Pattern 2

Single node deployment, which has all WSO2 API-M components in one instance, with Analytics.

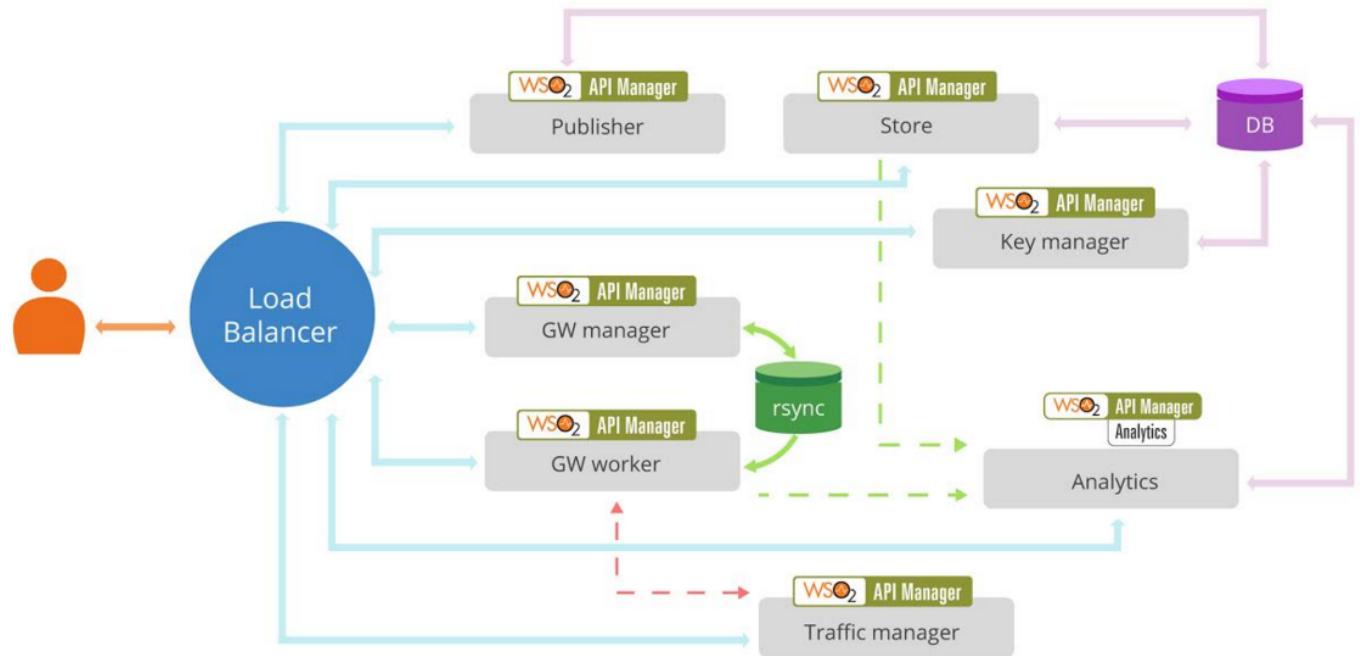


Figure: All-in-one instance with analytics

This pattern consists of a stand-alone WSO2 API-M setup with a single node deployment and with a single wso2am-analytics server instance. This pattern uses external MySQL databases. You can use the H2 database when using API-M with Analytics, however, in a production scenario it is advised to use another DBMS other than H2.

Pattern 3

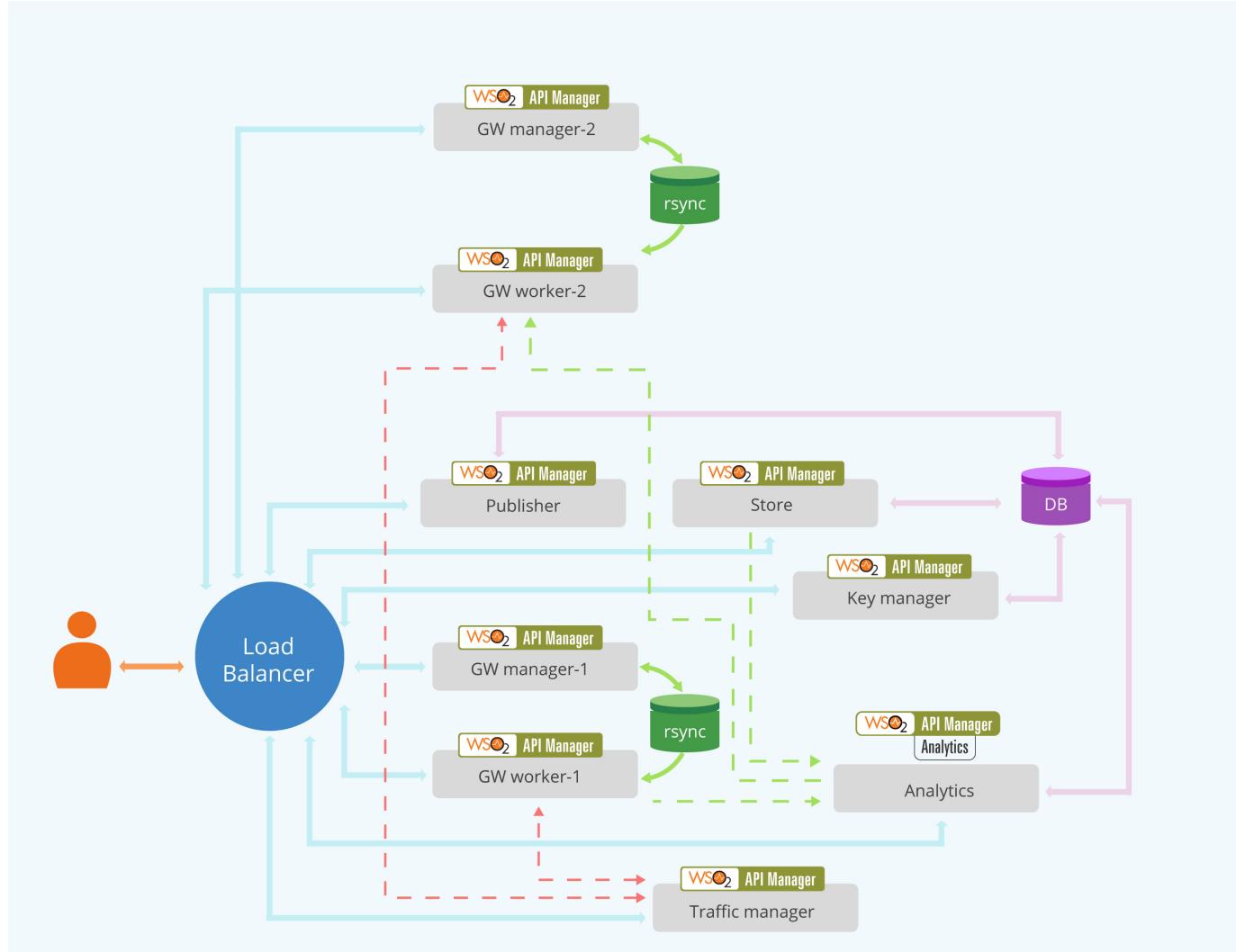
Gateway worker/manager separation.



This pattern consists of a fully distributed WSO2 API-M setup (including a Gateway cluster of one manager and one worker) with a single wso2am-analytics server instance. This pattern uses external MySQL databases.

Pattern 4

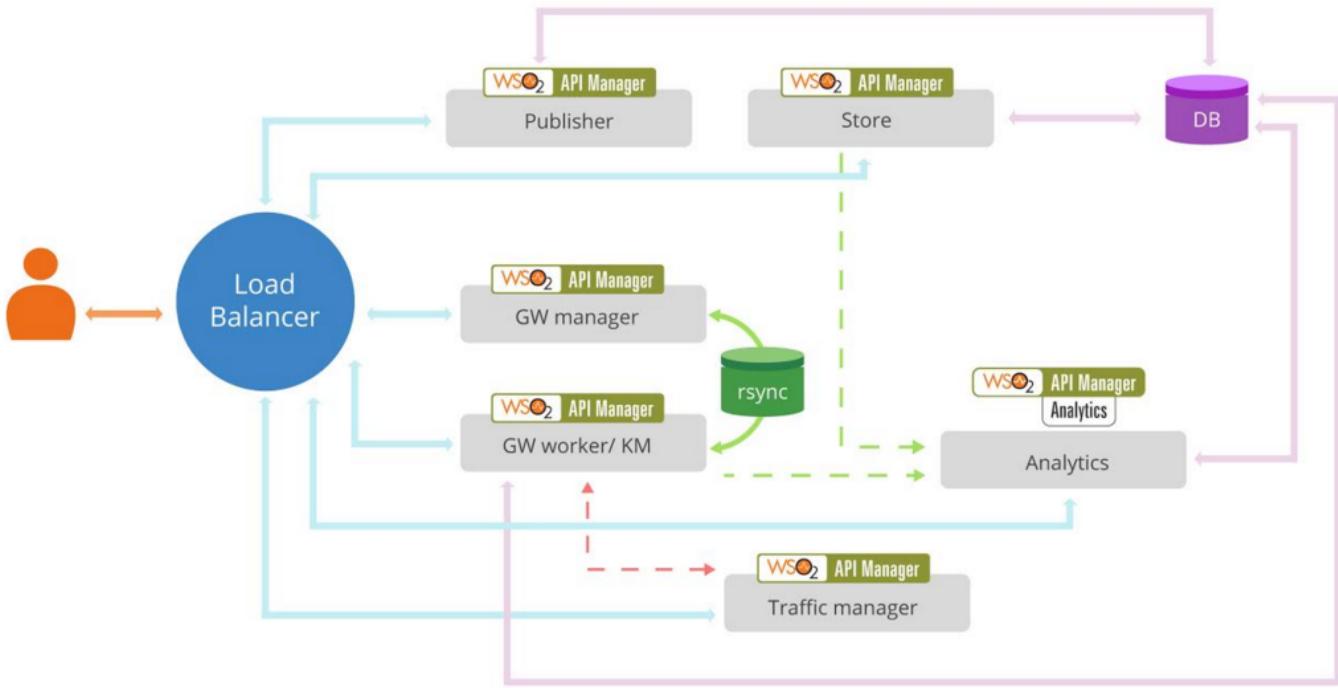
Fully distributed WSO2 API-M setup with a single wso2am-analytics server.



This pattern consists of a fully distributed API-M setup including two Gateway clusters, where each has one manager and one worker, with a single wso2am-analytics server instance. You can have the gateway environments in any preferred environment (e.g., local-area network (LAN) and demilitarized zone (DMZ)).

Pattern 5

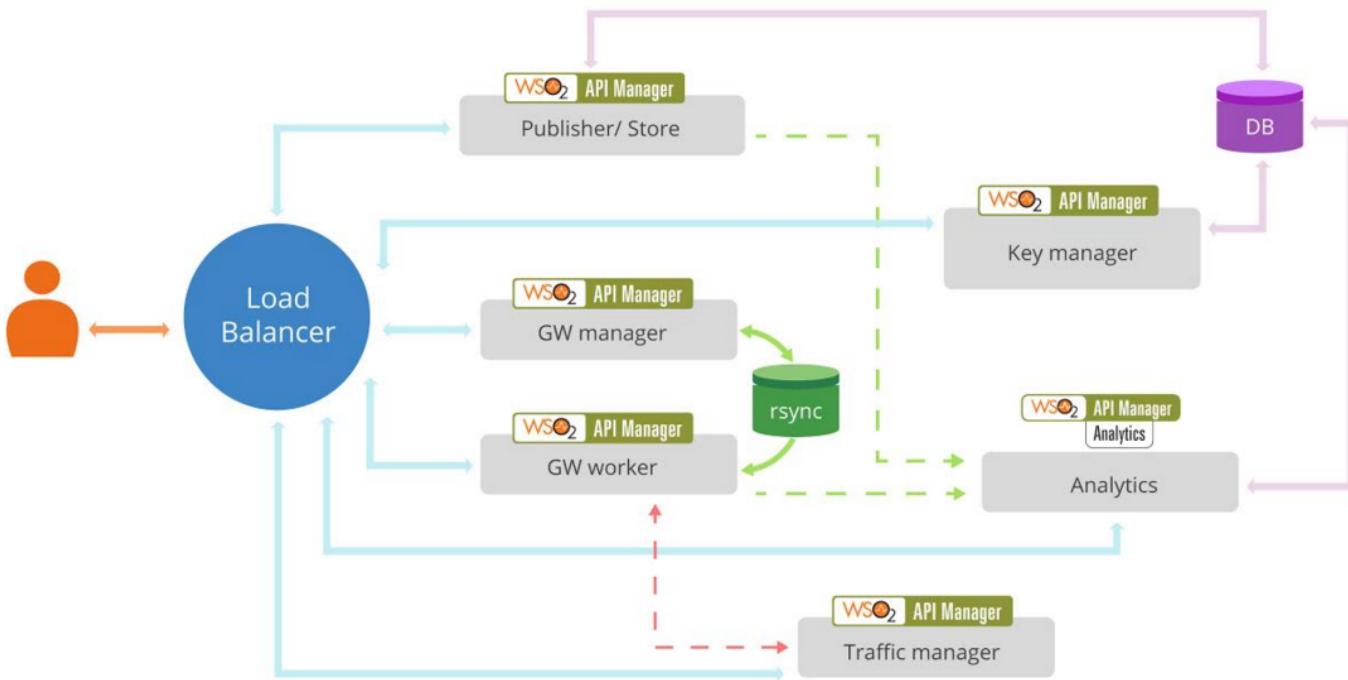
Gateway worker/manager separation. Gateway worker and Key Manager in the same node.



This pattern consists of a distributed WSO2 API-M setup including a Gateway cluster of one manager and one worker and the Gateway worker is merged with the Key Manager. It also consists of a single wso2am-analytics server instance and it uses external MySQL databases.

Pattern 6

Gateway worker/manager separation. Store in the same node as the Publisher.



This pattern consists of a distributed WSO2 API-M setup (including a Gateway cluster of one manager and one worker) of which the Publisher is merged with the Store. It also consists of a single wso2am-analytics server instance and it uses external MySQL databases.

Pattern 7

WSO2 Identity Server acts as a Key Manager node for the WSO2 API Manager.



This pattern consists of a stand-alone WSO2 APIM setup with a single node deployment. The pattern uses external MySQL databases. The only difference of this pattern from pattern-1 is that this uses WSO2 Identity Sever as the Key Manager.

Clustering Gateways and Key Managers with key caching

For key validation, the Gateway can usually handle 3,000 transactions per second, whereas the Key Manager can only handle 500 transactions per second. To improve performance, the key cache is enabled on the Gateway by default, which allows the system to handle 3,000 transactions per second. However, if you need better security, you can enable the cache on the Key Manager instead. Note the following about clustering with key caching:

- When the cache is enabled at the Gateway, you can have two Gateways per Key Manager.
- When the cache is enabled at the Key Manager and disabled at the Gateway, you can have only one Gateway per Key Manager.
- If both caches are disabled (not recommended), even with only one Gateway per Key Manager, the system may not be able to handle the load, as the Key Manager will only be able to handle 500 transactions per second.

For more information, see [Key cache in the WSO2 API Manager documentation](#).

Traffic Manager scalable deployment patterns

See the article on Scalable Traffic Manager deployment patterns part 1 and part 2.

Deploying API Manager using Single Node Instances

In a typical production deployment, API Manager is deployed as components (Publisher, Store, Gateway, Key Manager and Traffic Manager). While this provides very high performance and a high level of scalability, it may be too complex if you want to run API Manager as a small to medium scale API Management solution. A WSO2 API-M single node deployment, which has all the API-M components in one instance, would be simple to set up and requires less resources when compared with a distributed deployment. It is ideal for any organization that wants to start small and iteratively build up a robust API Management Platform.

WSO2 provides two options for organizations that are interested in setting up a small to medium scale API Management solution.

- Setting up on WSO2 API Cloud, which is a subscription based API Management solution. You can access this service by creating an account in [WSO2 API Cloud](#).
- If you are interested in setting up a single node API Manager instance, which has all the API-M components in one instance, on-premise, you can [download the latest version of API Manager](#) and follow the instructions given below to set up the instance.

Prerequisites

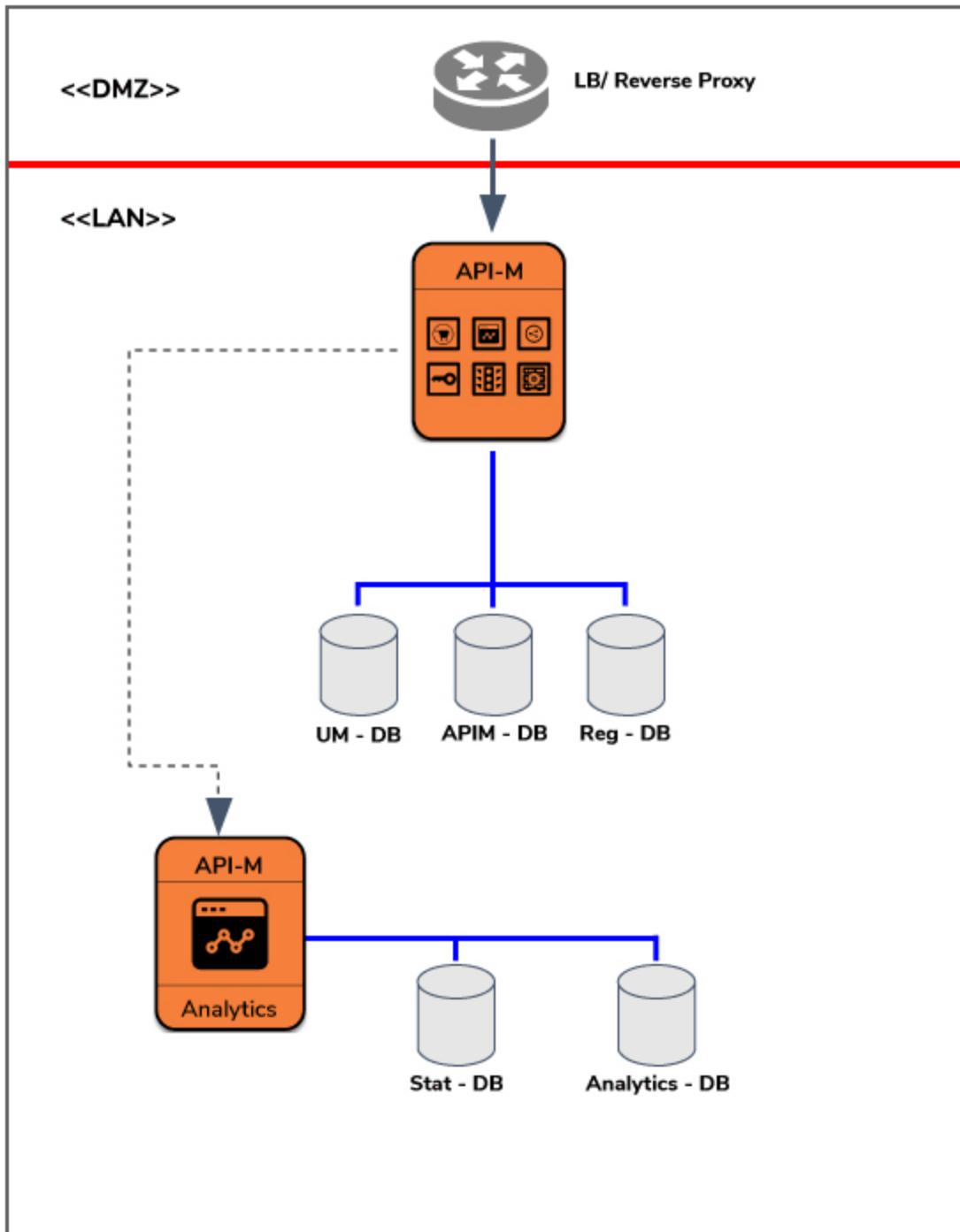
Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. Since this is an all-in-one deployment, it is recommended to use a higher hardware specification. You can further fine tune your operating system for production by tuning performance . For more information on installing the product on different operating systems, see Installing the Product .
Software	Oracle JDK 1.8

You can deploy a single node API Manager instance in the following methods:

- Single node deployment
- Active/active deployment

Single node deployment

In this setup, API traffic is served by one all-in-one instance of WSO2 API Manager.



Pros	Cons
------	------

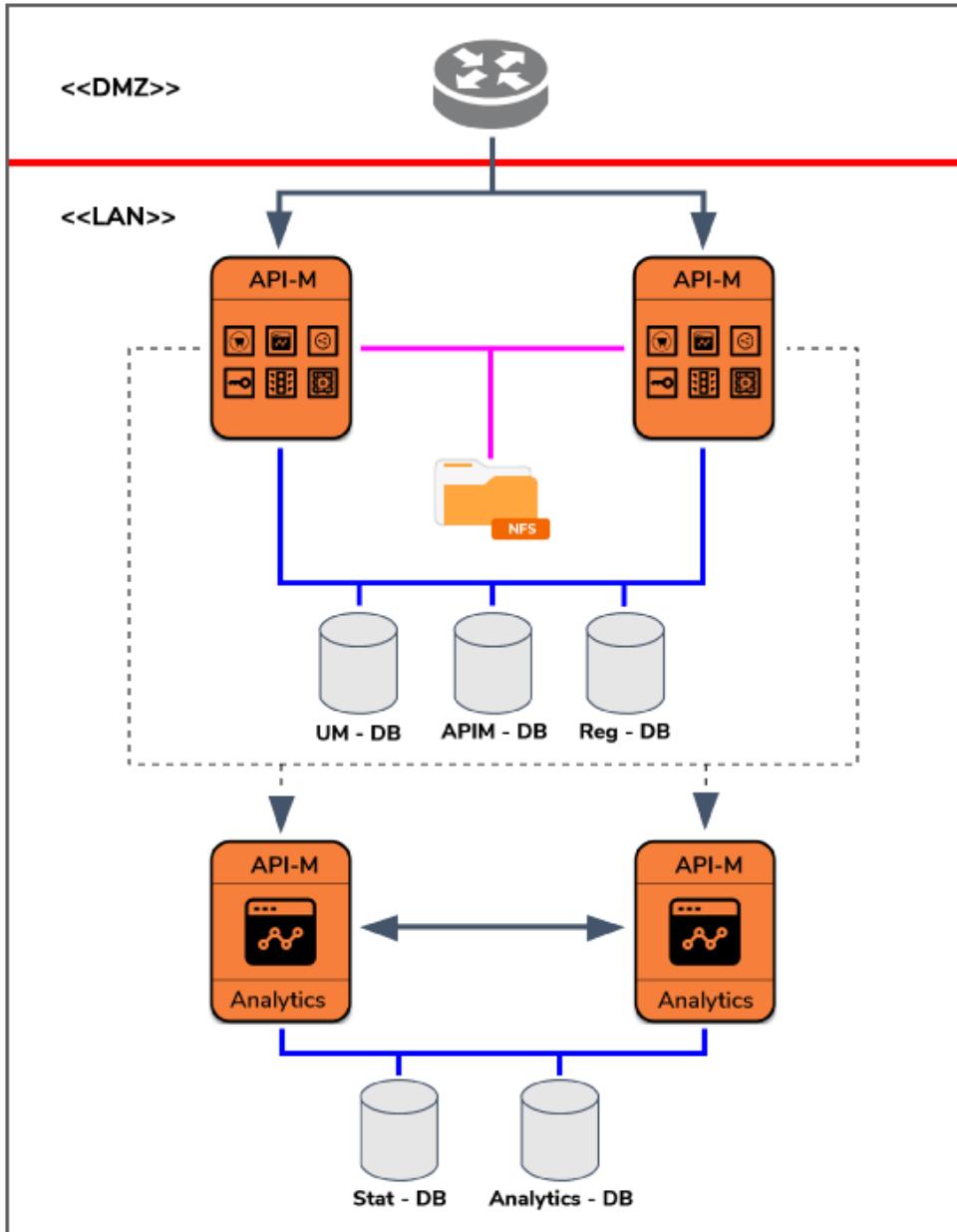
- | | |
|--|---|
| <ul style="list-style-type: none">• Production support is required only for a single API Manager node (you receive 24*7 WSO2 production support).• Deployment is up and running within hours.• Can handle up to 43 million API calls a day (up to 500 API calls a second)• Minimum hardware/cloud infrastructure requirements (only one node).• Suitable for anyone new to API Management. | <ul style="list-style-type: none">• Deployment does not provide High Availability.• Not network friendly. Deploying on a demilitarized zone (DMZ) would require a Reverse Proxy. |
|--|---|

For more information on:

- Manually configuring the production servers from scratch, see [Configuring a Single Node](#).
- Configuring production servers using Puppet scripts, see [Using Puppet Modules to Set up WSO2 API-M](#).

Active/active deployment

In this setup, API traffic is served by two single node (all-in-one) instances of WSO2 API Manager.



For more information on:

- Manually configuring the production servers from scratch, see [Configuring an Active-Active Deployment](#).
- Configuring production servers using Puppet scripts, see [Using Puppet Modules to Set up WSO2 API-M](#).

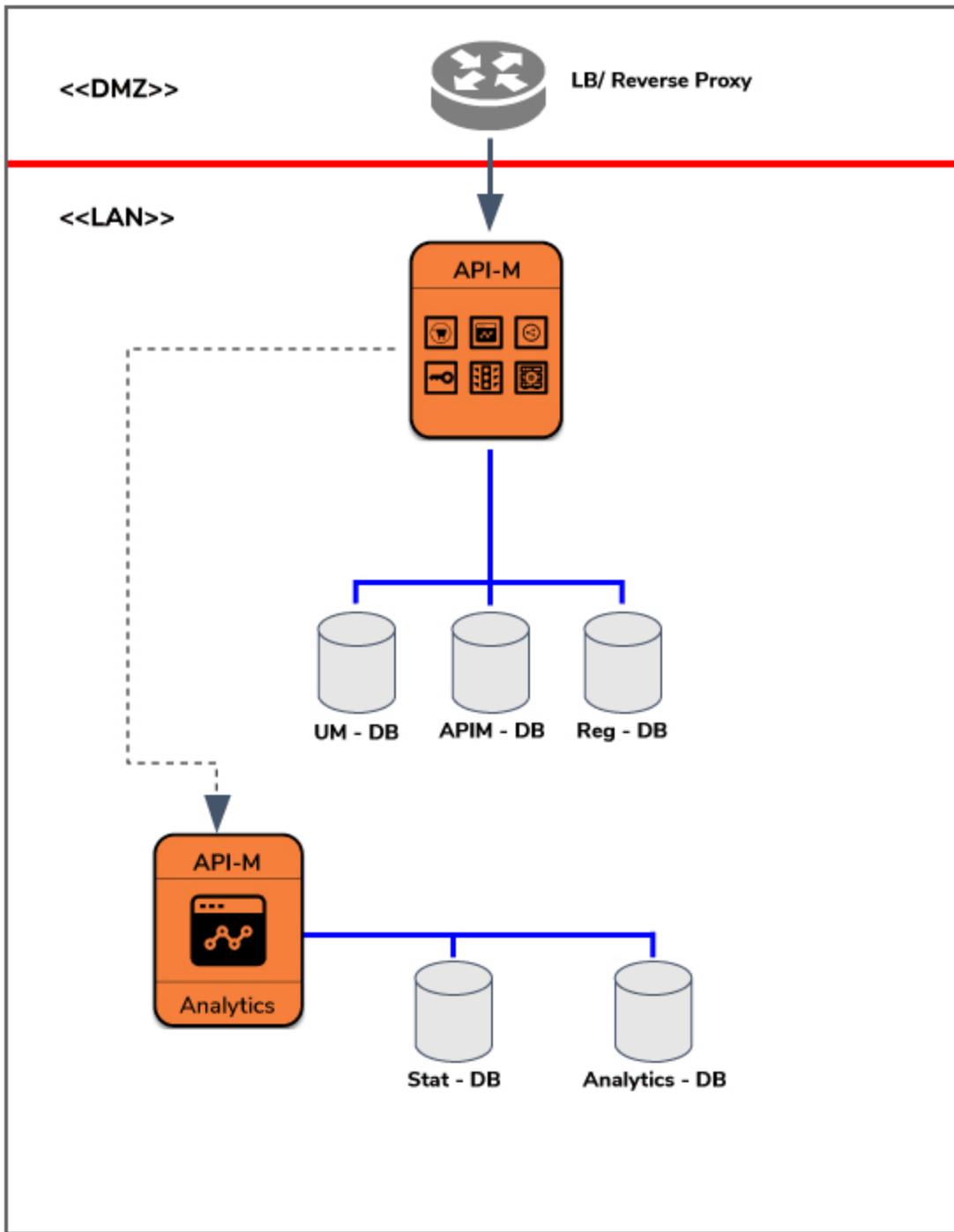
Pros	Cons
<ul style="list-style-type: none"> • The system is highly available. • Production support is required for 2 API Manager nodes (you receive 24*7 WSO2 production support). • Can handle up to 86 million API calls a day (up to 1000 API calls a second) • Deployment is up and running within hours. 	<ul style="list-style-type: none"> • Not network friendly. Deploying on a DMZ would require a Reverse Proxy.

Configuring a Single Node

This page walks you through how to manually configure and deploy WSO2 API Manager in a standalone single instance, without using a distributed or HA deployment patterns. For an overview of Single Node and Active-Active deployments, Please see [Deploying API Manager using Single Node Instances](#).

If you want to deploy WSO2 API-M using a hybrid single node deployment where WSO2 Identity Server is used as the Key Manager while the rest of the WSO2 API-M components are all in one node, see one of the following sections.

- If you are using the WSO2 Identity Server 5.3.0 as the Key Manager distribution, see [Configuring WSO2 Identity Server as a Key Manager](#).
- If you are using an existing WSO2 Identity Server 5.3.0 setup or you want to use the standard WSO2 Identity Server 5.3.0 distribution, see [Configuring an External IdP as a Key Manager](#).



Step 1 - Create a SSL certificate

Create a SSL certificate on the WSO2 API Manager node. For more information, see [Creating SSL Certificates](#) in the Administration Guide.

Note that this step is optional according to the setup that you configure. All WSO2 products are by default shipped with a keystore file and truststore file (stored in the `<PRODUCT_HOME>/repository/resources/security/` directory). The default keystore that is shipped with a WSO2 product (`wso2carbon.jks`) is by default configured for all of the below purposes.

- Authenticating the communication over Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocols.
- Encrypting sensitive data such as plain-text passwords found in both product-level and product feature-level configurations/configuration files using secure vault.
- Encrypting and signing SOAP messages using WS-Security.

However, in a production environment, it is advised to set up several different keystores with separate trust chains for the above use cases. For more information refer [Recommendations for setting up keystores in WSO2 products](#).

Step 2 - Configure the load balancer

For information on configuring the load balancer see [Configuring the Proxy Server and the Load Balancer](#).

Step 3 - Configure the databases

For information on configuring the databases, see [Installing and Configuring the Databases](#).

Step 4 - Configure hostnames that are used to expose APIs

This step is only required if you are using a hostname to expose APIs.

▼ [Click here for more information](#).

Add this hostname in the <API-M_HOME>/repository/conf/api-manager.xml file. Update the <Gateway Endpoint> element with your chosen hostname as shown below. In this case we are using localhost as the hostname:

```
<!-- Endpoint URLs for the APIs hosted in this API gateway.-->
<GatewayEndpoint> http://localhost,https://localhost </GatewayEndpoint>
```

Step 5 - Configure your deployment with production hardening

Ensure that you have taken into account the respective security hardening factors (e.g., changing and encrypting the default passwords, configuring JVM security etc.) before deploying WSO2 API-M. For more information, see the [Production Deployment Guidelines](#) in the Administration Guide.

Step 6 - Configure API-M Analytics

If you wish to work with the API-M Analytics aspect of WSO2 API-M such as, to view reports, statistics, and graphs on the APIs deployed in WSO2 API-M, configure WSO2 API-M Analytics.

▼ [Click here for information on configuring API-M Analytics](#).

Use the **standard setup** instead of the quick setup if you are working on configuring a production scenario.

This section explains how to configure analytics for WSO2 API Manager (WSO2 API-M). The API Manager integrates with the [WSO2 Analytics platform](#) to provide reports, statistics and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and carry out detailed analysis of the logs. WSO2 API Manager has an enhanced distribution of Analytics to cater to API Manager specific scenarios which is used here to configure APIM Analytics.

By default, WSO2 API Manager has a port offset of 0 (no port offset) and WSO2 API Manager Analytics has an offset of 1. Therefore, this guide assumes that you do not have any other carbon servers running on the same machine with port offsets of 0 or 1.

Click on the **Quick Setup** tab to set up analytics for quick demos and try-out scenarios, or click on the **Standard Setup** tab to set up analytics for a production environment.

WSO2 recommends using the API-M Analytics (powered by [WSO2 Data Analytics Server](#)) distribution to set up the minimum high availability deployment with API Manager. For configuration details, see [Minimum High Availability Deployment for WSO2 APIM Analytics](#).

Quick Setup Standard Setup

This mode of setup is only recommended for non-critical demos and quick try-out scenarios. Once you run the servers you will notice that there are database files being created in a folder called **tmpStatDB** in the directory where you installed the two servers. These are H2 databases that hold the summarized data of your API Analytics. In a more standard deployment, this data is recorded in database servers such as MySQL, Oracle, etc. Therefore, this mode of operation is not recommended for production grade or other critical deployments.

1. Download the WSO2 API Manager and the WSO2 API-M Analytics distributions (zip files), and extract both files to the same directory (preferably an empty directory).
 - To download the WSO2 API Manager distribution, click **DOWNLOAD** and then click **DOWNLOAD Server** in the [WSO2 API Manager page](#).
 - To download the WSO2 API-M Analytics distribution, click **DOWNLOAD** and then click **DOWNLOAD ANALYTICS** in the [WSO2 API Manager page](#).
2. Take the following steps to install WSO2 APIM Analytics. Because this procedure is identical to installing WSO2 Data Analytics Server (DAS), these steps take you to the DAS documentation for details.
 - a. Ensure that you have met the [Installation Prerequisites](#).
 - b. Go to the installation instructions relevant to your operating system:
 - [Installing on Linux](#)
 - [Installing on Windows](#)
 - [Installing as a Windows Service](#)
 - [Installing as a Linux Service](#)
3. To enable Analytics, open the `<API-M_HOME>/repository/conf/api-manager.xml` file and set the `Enabled` property under `Analytics` to `true` as shown below. Save this change.

```
<Enabled>true</Enabled>
```

If you are working on a distributed (clustered) setup of API Manager, do the configurations instructed to be done in API Manager in Publisher, Store and Gateway nodes.

4. Share the `WSO2AM_STATS_DB` datasource between WSO2 API-M and WSO2 API-M Analytics as follows.
 - a. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and make sure that a configuration for the `WSO2AM_STATS_DB` datasource is included with your

datasource configurations. The default configuration which is already available in `master-datasources.xml` file is as follows. It is configured for the in-built h2 database by default. You can change the datasource according to the database you use referring to change statistics datasource.

```
<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:h2:.../tmpStatDB/WSO2AM_STATS_DB;DB_CLOSE_ON_EXIT=FALSE;LOCK_TIMEOUT=60000;AUTO_SERVER=TRUE</url>
            <username>wso2carbon</username>
            <password>wso2carbon</password>
            <defaultAutoCommit>false</defaultAutoCommit>
            <driverClassName>org.h2.Driver</driverClassName>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        </configuration>
    </definition>
</datasource>
```

If you are changing the datasource to a different database like MySQL, note that you do not need to run the database scripts against the created databases as the tables for the datasources are created at runtime.

- b. Open the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/stats-datasources.xml` file and make sure that the same configuration for WSO2AM_STATS_DB in the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file (mentioned in the previous sub step) is added in it.
5. Open the `<API-M_HOME>/repository/conf/log4j.properties` file. Add the `DAS_AGENT` to the end of the root logger as shown below.

This configuration is required only if you need to analyze WSO2 API-M logs using the Log Analyzer. When you are disabling analytics, make sure to remove this configuration to avoid errors.

If you are working on a distributed (clustered) setup of API Manager, follow these steps to configure log analyzer in Gateway node.

```
log4j.rootLogger=<other loggers>, DAS_AGENT
```

6. Start the WSO2 API-M Analytics server, and then start the API Manager server. To start a WSO2 product server, navigate to the <PRODUCT_HOME>/bin directory in your console and run one of the following scripts as relevant.
 - a. On Windows: wso2server.bat --run
 - b. On Linux/Mac OS: sh wso2server.sh

If Analytics is properly configured in WSO2 API Manager you will see the following log in the Analytics server during API Manager server startup.

```
INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
```

You can now start using the API Manager for its usual operations and the required Analytics functionality is enabled.

If you are configuring API-M Analytics with MSSQL and you get an error when you start the API-M Analytics server stating that a table cannot have more than one clustered index, follow the steps below.

1. Open the <API-M_ANALYTICS_HOME>/repository/components/features/org.wso2.carbon.analytics.spark.server_VERSION/spark-jdbc-config.xml file.
2. Update the value for the <indexCreateQuery> element of the MySQL database as shown below.

```
<database name="Microsoft SQL Server">
<indexCreateQuery>CREATE INDEX {{TABLE_NAME}}_INDEX ON
{{TABLE_NAME}} ({{INDEX_COLUMNS}})</indexCreateQuery>
</database>
```

Downloading WSO2 API-M Analytics

Follow the instructions below to download the binary distribution of WSO2 API-M Analytics.

The binary distribution contains the binary files for both MS Windows, and Linux-based operating systems. You can also download, and build the source code.

1. Go to the [WSO2 API Manager](#) page.
2. Click **DOWNLOAD** and then click **DOWNLOAD ANALYTICS** to download the WSO2 API-M Analytics product pack.

Installing WSO2 API-M Analytics

Take the following steps to install WSO2 APIM Analytics. Because this procedure is identical to installing WSO2 Data Analytics Server (DAS), these steps take you to the DAS documentation for details.

1. Ensure that you have met the [Installation Prerequisites](#).
2. Go to the installation instructions relevant to your operating system:
 - [Installing on Linux](#)
 - [Installing on Windows](#)
 - [Installing as a Windows Service](#)
 - [Installing as a Linux Service](#)

Configuring WSO2 API Manager to publish statistics

Follow the instructions below to do the required configurations for WSO2 API-M to publish statistics in the WSO2 API-M Analytics server.

To download the WSO2 API Manager distribution, click **DOWNLOAD** and then click **DOWNLOAD Server** in the [WSO2 API Manager page](#).

If you are working on a distributed (clustered) setup of API Manager, do the configurations instructed to be done in API Manager in Publisher, Store and Gateway nodes.

1. Open the `<API-M_HOME>/repository/conf/api-manager.xml` file.
2. Under the `<Analytics>` sub element, set the `Enabled` parameter to `true`.
3. Configure the following parameters if required.

Parameter	Value
<code><DASServerURL></code>	<code><protocol>://<hostname>:<port>/</code>

<DASUsername>	A valid administrator username
<DASPassword>	The password of the username specified.
<DASRestApiURL>	<code>https://<host>:<port></code>
<DASRestApiUsername>	A valid administrator username
<DASRestApiPassword>	The password of the username specified.
<SkipEventReceiverConnection>	false
<PublisherClass>	<code>org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageDataBridge</code>
<PublishResponseMessageSize>	false

<Streams>	<pre> <Request> <Name>org.wso2.apimgt.statistics.request</Name> <Version>1.1.0</Version> </Request> <Response> <Name>org.wso2.apimgt.statistics.response</Name> <Version>1.1.0</Version> </Response> <Fault> <Name>org.wso2.apimgt.statistics.failure</Name> <Version>1.0.0</Version> </Fault> <Throttle> <Name>org.wso2.apimgt.statistics.throttling</Name> <Version>1.0.0</Version> </Throttle> <Workflow> <Name>org.wso2.apimgt.statistics.workflow</Name> <Version>1.0.0</Version> </Workflow> <ExecutionTime> <Name>org.wso2.apimgt.statistics.executiontime</Name> <Version>1.0.0</Version> </ExecutionTime> <AlertTypes> <Name>org.wso2.analytics.apim.alert</Name> <Version>1.0.0</Version> </AlertTypes> </pre>
<StatsProviderImpl>	org.wso2.carbon.apimgt.usage.client.impl.APIUsageStatisticsR...

4. Save the changes.
5. Change the hostname to the actual hostname configured for the analytics pack in the <API-M_HOME>/repository/deployment/server/jaggeryapps/portal/configs/designer.json file as follows.

This step avoids the host verification errors that can occur when multiple WSO2 product instances publish statistics.

```

} ,
"host": {
    "hostname": "localhost",
    "port": "",
    "protocol": ""
}

```

Configuring the Log Analyzer

This configuration is required only if you want to analyze WSO2 API-M logs using the Log Analyzer. If you are working on a distributed (clustered) setup of API Manager, follow these steps below in Gateway node.

Follow the steps below to configure the Log Analyzer.

1. Add `DAS_AGENT` to the end of the root logger in the `<API-M_HOME>/repository/conf/log4j.properties` file as shown below.

```
log4j.rootLogger=<other loggers>, DAS_AGENT
```

Then check and make sure that the following configuration is available in the same file. Modify the values for `userName`, `password` and/or `url` if required.

The values given below are the default values of the configuration.

```

# DAS_AGENT is set to be a Custom Log Appender.
log4j.appenders.DAS_AGENT=org.wso2.carbon.analytics.shared.data.agents.log4j.appenders.LogEventAppender
# DAS_AGENT uses PatternLayout.
log4j.appenders.DAS_AGENT.layout=org.wso2.carbon.analytics.shared.data.agents.log4j.util.TenantAwarePatternLayout
log4j.appenders.DAS_AGENT.columnList=%D,%S,%A,%d,%c,%p,%m,%H,%I,%Stacktrace
log4j.appenders.DAS_AGENT.userName=admin
log4j.appenders.DAS_AGENT.password=admin
log4j.appenders.DAS_AGENT.url=tcp://localhost:7612
log4j.appenders.DAS_AGENT.maxTolerableConsecutiveFailure=5
log4j.appenders.DAS_AGENT.streamDef=loganalyzer:1.0.0

```

2. To view log analytics on the API-M Admin portal, you have to set the DAS REST API configurations (`DASRestApiURL`, `DASRestApiUsername`, `DASRestApiPassword`). You can give the same previous value as `<DASUsername>` to `<DASRestApiUsername>` and `<DASPassword>` to `<DASRestApiPassword>`. The `<DASRestApiURL>` value should be in the following format,

https://<DAS Hostname>:<DAS Https port>. Go to the <API-M_HOME>/repository/conf/api-manager.xml file and change the value given below.

```
<DASRestApiURL>https://analytics.wso2.com:9444</DASRestApiURL>
```

3. If the API-M server and the DAS server run on two different hosts with their default certificates, set the HostnameVerifier to AllowAll in the Pass Through HTTP SSL Sender section of the <API-M_HOME>/repository/conf/axis2/axis2.xml file as follows. If this setting is not changed, the javax.net.ssl.SSLException: Host name verification failed for host exception may occur for the API-M instance.

```
<transportSender name="https">
    <parameter name="non-blocking" locked="false">true</parameter>
    <parameter name="keystore" locked="false">
        <KeyStore>
            <Location>repository/resources/security/wso2carbon.jks</Location>
            <Type>JKS</Type>
            <Password>wso2carbon</Password>
            <KeyPassword>wso2carbon</KeyPassword>
        </KeyStore>
    </parameter>
    <parameter name="truststore" locked="false">
        <TrustStore>
            <Location>repository/resources/security/client-truststore.jks</Location>
            <Type>JKS</Type>
            <Password>wso2carbon</Password>
        </TrustStore>
    </parameter>
    <parameter name="HostnameVerifier">AllowAll</parameter>
</transportSender>
```

4. If the WSO2 API Manager was started before these configurations were done, restart it in order to apply the changes.

Securing log4j properties file with a secure vault

Using secure vaults allows you to avoid exposing passwords by having them in plain text in the log4j properties file. Follow the procedure below if you want to secure the log4j properties file with a secure vault.

1. If you have not already generated the secret-conf.properties file with default values, navigate to the <API-M_HOME>/bin directory and execute the following command to generate it. This generates the secret-conf.properties file in the <API-M_HOME>/repository/conf/security directory. If

you have already generated this file, proceed to Step 2.

```
sh ciphertool.sh -Dconfigure
```

Enter wso2carbon as the keystore password when the following appears in the console.

[Please Enter Primary KeyStore Password of Carbon Server :]

2. Execute the following command from the <API-M_HOME>/bin directory. This generates the encrypted value for the clear text password.

```
sh ciphertool.sh
```

- a. Enter wso2carbon when the following appears in the console.

[Please Enter Primary KeyStore Password of Carbon Server :]

- b. Enter admin as the input value when the following appears in the log. (This value is entered based on the secret-conf.properties file you generated in step 1.)

[Enter Plain text value :]

[Please Enter value Again :]

The following output is displayed in the console.

Encryption is done Successfully

The encrypted value is:

```
MpfXhKP+iJSImA/KNa+DoOXCPQAYF3JLh1FNAdG6F3naWK+N1/WEWOJkFx4kK34i1VtkywNN9SiC  
MRQGTFw+nqzK5/INgcFFdoxv491M/FJw8CyXKQ0JWdxw5QJPtrjJzvGp6Rj6xt4ysb6HdG5uNG+a  
1E0lqdmzGUYQ6oejIlk=
```

3. Open the <API-M_HOME>/repository/conf/security/ cipher-text.properties file and add the following entry.

```
log4j.appender.DAS_AGENT.password=MpfXhKP+iJSImA/KNa+DoOXCPQAYF3JLh  
1FNAdG6F3naWK+N1/WEWOJkFx4kK34i1VtkywNN9SiC  
MRQGTFw+nqzK5/INgcFFdoxv491M/FJw8CyXKQ0JWdxw5QJPtrjJzvGp6Rj6xt4ysb6  
HdG5uNG+a  
1E0lqdmzGUYQ6oejIlk=
```

4. Open the <API-M_HOME>/repository/conf/log4j.properties file and configure the password as log4j.appender.DAS_AGENT.password=secretAlias:log4j.appender.DAS_AGENT.password (as shown below).

```

log4j.appender.DAS_AGENTt=org.wso2.carbon.analytics.shared.data.agents.log4j.appenders.LogEventAppender
log4j.appender.DAS_AGENT.layout=org.wso2.carbon.analytics.shared.data.agents.log4j.util.TenantAwarePatternLayout
log4j.appender.DAS_AGENT.columnList=%D,%S,%A,%d,%c,%p,%m,%H,%I,%Stacktrace
log4j.appender.DAS_AGENT.userName=admin
log4j.appender.DAS_AGENT.password=secretAlias:log4j.appenders.DAS_AGENT.password
log4j.appender.DAS_AGENT.url=tcp://localhost:7612
log4j.appender.DAS_AGENT.maxTolerableConsecutiveFailure=5
log4j.appenders.DAS_AGENT.streamDef=loganalyzer:1.0.0

```

5. Start the WSO2 API Manager server by running one of the following commands from the <API-M_ANALYTICS_HOME>/bin directory.

- On Windows: wso2server.bat --run
- On Linux/Mac OS: sh wso2server.sh

Enter wso2carbon when the following appears in the console.
[Enter KeyStore and Private Key Password :]

If you want to start the server as a background process, carry out the following steps before starting the server.

1. Create a file named password-tmp.txt in the <API-M_HOME> directory. Add wso2carbon (the primary keystore password) to this file and save.

By default, the password provider assumes that both the private key and the keystore passwords are the same. If you want them to be different, the private key password should be entered in the second line of the file.

2. The keystore password is picked from the password-tmp.txt file. This file is automatically deleted from the file system when you start the server. Make sure to add this temporary file back whenever you start the server as a background process.

If you name the password file password-persist.txt instead of password-tmp.txt, then the file is not deleted once the server is started. Therefore, it is not required to provide the password in subsequent startups.

Configuring databases

Configuring databases allow you to persist data relating to APIs, process them and analyze. Follow the procedure below to configure databases.

The following is a list of database versions that are compatible with WSO2 API-M Analytics.

- Postgres 9.5 and later
- MySQL 5.6
- MySQL 5.7
- Oracle 12c
- MS SQL Server 2012
- DB2

1. Open the <API-M_ANALYTICS_HOME>/repository/conf/datasources/analytics-datasources.xml file. Note that two datasources named as WSO2_ANALYTICS_EVENT_STORE_DB and WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB are configured by default to point to the H2 databases.
2. Create two database schemas in your database server (MySQL, Oracle, etc) for the two datasources, and change the configurations of those datasources to point to the relevant schemas. A sample configuration is given below.

The database user you provide here requires permissions to create tables.

Note that you do not need to run the database scripts against the created databases as the tables for the datasources are created at runtime.

```
<datasource>
    <name>WSO2_ANALYTICS_EVENT_STORE_DB</name>
    <description>The datasource used for analytics record store</description>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:mysql://localhost:3306/stats_200?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>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

- If you are using **Oracle**, its recommended to increase the DB block size as described in <http://www.oratable.com/ora-01450-maximum-key-length-exceeded/>, to avoid the error

'ORA-01450: maximum key length (6398) exceeded'.

- If you are using **DB2**, run [this script](#) before you start the WSO2 API-M Analytics server.
- If you are using **MySQL 5.7**, open `<API-M_ANALYTICS_HOME>/repository/conf/analytics/spark/spark-jdbc-config.xml` and configure the `stringType` property under the `typeMapping` element which is under `<database name="mysql">` element as follows.

```
<stringType>VARCHAR(100)</stringType>
```

If you are using **MSSQL**, add the `SendStringParametersAsUnicode` property to the database connection URL in the data source configuration in the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/analyticss-datasources.xml` file as shown below to avoid deadlock issues that are caused when the same table row is updated in two or more sessions at the same time.

```
<url>SQLSERVER_JDBC_URL;SendStringParametersAsUnicode=false</url>
```

3. Share the `WSO2AM_STATS_DB` datasource between WSO2 API-M and WSO2 API-M Analytics as follows.
 - a. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and make sure that a configuration for the `WSO2AM_STATS_DB` datasource is included. The default configuration is as follows.

```
<datasource>
  <name>WSO2AM_STATS_DB</name>
  <description>The datasource used for setting statistics to API Manager</description>
  <jndiConfig>
    <name>jdbc/WSO2AM_STATS_DB</name>
  </jndiConfig>
  <definition type="RDBMS">
    <configuration>

      <url>jdbc:mysql://localhost:3306/WSO2AM_STATS_DB?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>
      <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
  </definition>
</datasource>
```

you need to enable analytics in publisher, store and gateway nodes. However, you need to add this datasource configuration in gateway nodes. Following table provides more information on Analytics usage of API Manager components in a distributed environment.

Component	Enable statistics	Events Published	Read statsDB
Gateway_Manager	YES only if accept request	YES only if accept request	NO
Gateway_worker	YES	YES	NO
Key Manager	NO	NO	NO
Publisher	YES	NO	YES
Store	YES	YES	YES
Traffic Manager	NO	NO	NO

You do not need to enable analytics in Key Manager and Traffic Manager nodes as those components do not read or publish statistics. Though gateway nodes publish events, they are not reading statistics database. Therefore you are not required to add the WSO2AM_STATS_DB datasource configuration in gateway nodes. Publisher node read statistics but not publishing events. Therefore you can disable event publisher initialization at startup in publisher by setting `<SkipEventReceiverConnection>` value to true in `<PUBLISHER_HOME>/repository/conf/api-manager.xml`. API Store nodereads statistics and also publish events. Therefore we need to keep the statsource configuration for statsDB in Store node as well.

- b. Open the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/stats-datasources.xml` file and make sure that the same configuration in the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file (mentioned in the previous sub step) is added in it.
 - 4. Create a schema in your database server similar to the `WSO2AM_STATS_DB` datasource. Make sure that this datasource points to the relevant schema.
- The database user you provide here requires permissions to create tables.
- 5. Download and copy the relevant database driver JAR file to the `<API-M_ANALYTICS_HOME>/repository/components/lib` directory.
 - 6. Start the WSO2 API-M Analytics server.

Troubleshooting

If you are configuring API-M Analytics with MSSQL and you get an error when you start the API-M Analytics server stating that a table cannot have more than one clustered index, follow the steps below.

1. Open the <API-M_ANALYTICS_HOME>/repository/components/features/org.wso2.carbon.analytics.spark.server_VERSION/spark-jdbc-config.xml file.
2. Update the value for the <indexCreateQuery> element of the MySQL database as shown below.

```
<database name="Microsoft SQL Server">
<indexCreateQuery>CREATE INDEX {{TABLE_NAME}}_INDEX ON
{{TABLE_NAME}} ({{INDEX_COLUMNS}})</indexCreateQuery>
</database>
```

3. Restart the server for the above changes to take effect.

Purging Analytics Data

You can remove historical data in API Manager Analytics by data purging. By purging data, you can achieve high performance in data analysis without removing analyzed summary data. When purging data, only the stream data generated by API Manager is purged. Refer [Purging Analytics Data](#) for more information.

Step 7 - Start the WSO2 API-M server

Start the server using the following standard start-up script. For more information, see [Starting the server](#).

[Linux/Mac OS](#) [Windows](#)

```
cd <API-M_HOME>/bin/
sh wso2server.sh
```

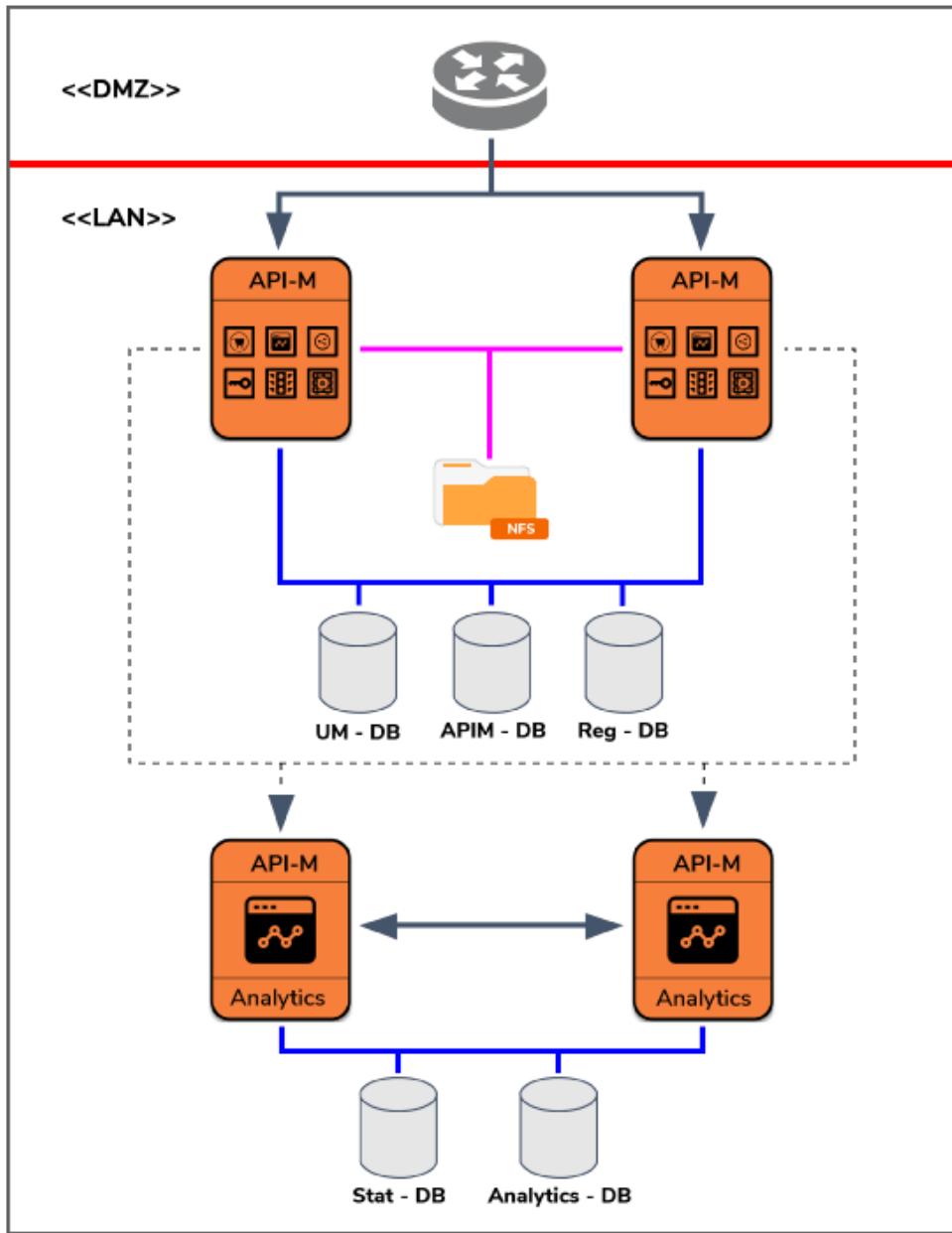
```
cd <API-M_HOME>\bin\
wso2server.bat --run
```

Configuring an Active-Active Deployment

This page walks you through how to manually configure WSO2 API Manager (WSO2 API-M) with two active nodes that each have all the components of the API-M together in one instance (all-in-one instance).

If you want to deploy WSO2 API-M using a hybrid active-active deployment pattern where WSO2 Identity Server is used as the Key Manager in high availability mode while the rest of the WSO2 API-M components are all in one node, see one of the following sections.

- If you are using the WSO2 Identity Server 5.3.0 as the Key Manager distribution, see [Configuring WSO2 Identity Server as a Key Manager](#).
- If you are using an existing WSO2 Identity Server 5.3.0 setup or you want to use the standard WSO2 Identity Server 5.3.0 distribution, see [Configuring an External IdP as a Key Manager](#).



Step 1 - Create a SSL certificate

Create a SSL certificate on the first WSO2 API-M all-in-one active node. For more information, see [Creating SSL Certificates](#) in the Administration Guide.

Step 2 - Configure the load balancer

For information on configuring the load balancer, see [Configuring the Proxy Server and the Load Balancer](#).

Step 3 - Configure the databases

For information on configuring the databases, see [Installing and Configuring the Databases](#).

Step 4 - Configure the Publisher with the Gateway

Configure the API Publisher in both nodes to be able to publish to the API-M Gateway of one of the nodes. Do this by pointing the <ServerURL> to the same Gateway node.

You need to configure this in the <API-M_HOME>/repository/conf/api-manager.xml file.

```
<APIGateway>
<ServerURL>https://localhost:${mgt.transport.https.port}${carbon.context}
services/</ServerURL>
</APIGateway>
```

Step 5 - Configure the content synchronization mechanism

Configure a shared file system as the content synchronization mechanism. You can use a common shared file system such as Network File System (NFS) or any other shared file system that is available. You need to mount the <API-M_HOME>/repository/deployment/server directory of the two nodes to the shared file system, in order to share all APIs and throttling policies between all the nodes.

Shared file system is the first preference that WSO2 recommends to synchronize the artifacts among the nodes, because APIs and throttling decisions can be published to any of the nodes; thereby, avoiding the vulnerability of a single point of failure that is present when using remote synchronization (rsync). However, if you are unable to maintain a shared file system, you can synchronize content using rsync. For information on setting up a rsync based deployment synchronization, see [Configuring rsync for Deployment Synchronization](#).

Step 6 - Configure Throttling

▼ [Click here for information on configuring Throttling.](#)

1. Configure the data publisher in the <DataPublisher> section which comes under the <ThrottlingConfigurations> section in the <API-M_HOME>/repository/conf/api-manager.xml file. You need to make these configuration changes so that the Gateway can publish data to the Traffic Manager in its own node and the Traffic Manager in the other node, so that the same event is sent to both servers at the same time. The WSO2 Complex Event Processor (WSO2 CEP) component that lies within the Traffic Manager acts as the data receiver and process the data to come up with the Throttling decisions.

[FormatExample](#)

```
<DataPublisher>
<Enabled>true</Enabled>
<Type>Binary</Type>
<ReceiverUrlGroup>{tcp://<node1-hostname>:<node1-port>} , {tcp://<node2-hostname>:<node2-port>} </ReceiverUrlGroup>
<AuthUrlGroup>{ssl://<node1-hostname>:<node1-port>} , {ssl://<node2-hostname>:<node2-port>} </AuthUrlGroup>
.....
</DataPublisher>
```

```

<DataPublisher>
    <Enabled>true</Enabled>
    <Type>Binary</Type>
    <ReceiverUrlGroup>{tcp://127.0.0.1:9612},{tcp://127.0.0.1:9613}</ReceiverUrlGroup>
    <AuthUrlGroup>{ssl://127.0.0.1:9712},{ssl://127.0.0.1:9713}</AuthUrlGroup>
    .....
</DataPublisher>

```

- Save your changes.

Step 7 - Configure the second WSO2 API-M node

Make a copy of the active instance configured above and use this copy as the second active instance.

When making a copy of the node, you need to also make a copy of the SSL certificate that you created for node 1 in step 1.

Step 8 - Configure your deployment with production hardening

Ensure that you have taken into account the respective security hardening factors (e.g., changing and encrypting the default passwords, configuring JVM security etc.) before deploying WSO2 API-M. For more information, see the [Production Deployment Guidelines](#) in the Administration Guide.

Step 9 - Configure Analytics

Optionally, if you need to work with the analytics aspect of WSO2 API-M such as, to view reports, statistics, and graphs on the APIs deployed in WSO2 API-M, [configure API-M Analytics](#).

Step 10 - Start the WSO2 API-M servers

Start the WSO2 API-M servers using the standard start-up script. For more information, see [Starting the server](#).

Linux/Mac OS Windows

```

cd <API-M_HOME>/bin/
sh wso2server.sh

```

```

cd <API-M_HOME>\bin\
wso2server.bat --run

```

Using Puppet Modules to Set up WSO2 API-M

If you wish, you can use the WSO2 API Manager (WSO2 API-M) Puppet Module to install and configure WSO2 API Manager using any of the 7 deployment patterns (plus the single node deployment with embedded H2 databases, namely pattern 0). Configuration data is managed using Hiera. Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner.

Follow one of the following guides to set up an API-M deployment using Puppet.

Please note that the API-M deployment patterns that we have listed under the [Deployment Patterns](#) section are newer than the deployment patterns that we have discussed in the following section.

- Using Puppet Modules to Set up WSO2 API-M with Pattern 0
- Using Puppet Modules to Set up WSO2 API-M with Pattern 1
- Using Puppet Modules to Set up WSO2 API-M with Pattern 2
- Using Puppet Modules to Set up WSO2 API-M with Pattern 3
- Using Puppet Modules to Set up WSO2 API-M with Pattern 4
- Using Puppet Modules to Set up WSO2 API-M with Pattern 5
- Using Puppet Modules to Set up WSO2 API-M with Pattern 6
- Using Puppet Modules to Set up WSO2 API-M with Pattern 7

Using Puppet Modules to Set up WSO2 API-M with Pattern 0

The following diagram illustrates pattern 0, which has a stand-alone WSO2 API Manager (WSO2 API-M) setup with a single node deployment. This pattern uses an embedded H2 database.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the WSO2 Puppet Modules to [deploy WSO2 API-M as an single node instance](#).

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Create two instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
- Step 3 - Set up Puppet Agent
 - 3.1 - Install Puppet Agent

- 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agent
- Step 5 - Configure the keyStore and client trustStore
- Step 6 - Run the WSO2 API-M Puppet Agent

Step 1 - Create two instances

Create two vanilla instances of which one will be configured as the Puppet Master and the other will be configured as the Puppet Agent in the subsequent steps. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install
ntp ; service ntp restart
```

- b. Navigate to the /tmp directory

```
cd /tmp
```

- c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb &&
dpkg -i puppetlabs-release-trusty.deb && apt-get update &&
apt-get install puppetmaster
```

- Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

- Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

- Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

- For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

- Log in to the server that you are going to set up as the Puppet Master, as a super user.
- Create the following directory.

```
/etc/puppet/environments/production/
```

- Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and [master] sections respectively.
You need to update this command in order to define where to look for hiera.yaml file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- `dns_alt_names=puppetmaster,puppet`
This defines the hostnames that the Puppet Master uses to sign the certificates.
- `autosign=true`
This determines that the certificate will be signed automatically.

- Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new dns_alt_names.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common/releases
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-base/releases
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-base/releases
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-am/releases
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-am/releases

7. Formulate the Puppet Modules.
When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- Extract the wso2-wso2base-1.0.0.tar.gz file, rename the folder name as wso2base, and copy that folder to the /etc/puppet/environments/production/modules/ directory.
- Extract the wso2-wso2am_runtime-2.1.0.tar.gz file, rename the folder name as wso2am_runtime, and copy that folder to the /etc/puppet/environments/production/modules/ directory.
- Optionally, install the Java module from Puppet Forge.

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

- d. Install the stdlib module from Puppet Forge.
You need to install the stdlib module, because WSO2 Puppet Modules use some of the functions in the stdlib module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.
Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.
- Unzip the wso2-puppet-common-1.0.0.zip file and copy the hiera.yaml file to the /etc/puppet directory.
 - Edit the YAML data directory location as follows in the hiera.yaml file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- c. Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains the common Hiera data.
 - d. Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.
 - e. Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/product` ion directory.
9. Formulate the `site.pp` manifest.
- Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a `.pp` extension. The `site.pp` manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the `site.pp` manifest:
- a. Copy the `wso2-puppet-common-1.0.0/manifests/site.pp` file to the `/etc/puppet/environments/production/manifests` directory.
 - b. Construct the files/packs as follows:
 - i. Copy the WSO2 API-M pack file (`wso2am-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_runtime/files` directory.
 - ii. Create the following directory - `/etc/puppet/environments/production/modules/wso2base/files`
 - iii. Copy the JDK installation file (e.g., `jdk-8u112-linux-x64.tar.gz`) to the `/etc/puppet/environments/production/modules/wso2base/files` directory.
 - iv. Add the JDK version (home) and filename information in the `/etc/puppet/environments/production/modules/wso2base/manifests/java.pp` file.
 - v. Add the host entry in the `/etc/puppet/hieradata/production/wso2/common.yaml` file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up Puppet Agent

Carry out the following instructions on the other instance that you created to configure and set up the Puppet Agent.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the `/etc/hosts` file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent `/etc/puppet/puppet.conf` file by appending the following to the `[main]` section.
You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the `/etc/hosts` file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agent

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Modify the `deployment.conf` file as follows.

Pattern-0 Configurations Format

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M with pattern 0.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-0
```

```

product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>

```

Step 5 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The WSO2 API-M GitHub repository includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepacked` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```

keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>

```

```

keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000

```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```

keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer

```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

FormatExample

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 6 - Run the WSO2 API-M Puppet Agent

1. Log in to Puppet Agent as a super user.
2. Copy the modified deployment.conf file to the /opt directory.
3. Copy the setup.sh file to the /opt directory and execute the following command.

As the root user, you need to execute this command to add the execute permission to the setup.sh file.

commandsetup.sh

```
chmod 755 setup.sh
```

```
#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

4. Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

Note that WSO2 API-M 2.1.0 gets installed in the following directory.

/mnt/wso2am-2.1.0

For API-M 2.1.0 the complete deployment time (deployment and server startup) is approximately 1 minute.

Using Puppet Modules to Set up WSO2 API-M with Pattern 1

The following diagram illustrates pattern 1 which consists of a stand-alone WSO2 API-M setup with a single node deployment. This pattern uses external MySQL databases. The only difference between this pattern 0 and pattern 1 is that, pattern 0 uses embedded H2 databases and pattern 1 is configured to use external MySQL databases.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 1 to deploy WSO2 API-M as a single node instance in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Create two instances
- Step 2 - Set up Puppet Master
 - 2.1 - Install Puppet Master
- Step 3 - Set up Puppet Agent
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent

- Step 4 - Set Facter variables for the WSO2 API-M Puppet Agent
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore
- Step 7 - Run the WSO2 API-M Puppet Agent

Step 1 - Create two instances

Create two vanilla instances of which one will be configured as the Puppet Master and the other will be configured as the Puppet Agent in the subsequent steps. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Step 2 - Set up Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install
ntp ; service ntp restart
```

- b. Navigate to the /tmp directory

```
cd /tmp
```

- c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb &&
dpkg -i puppetlabs-release-trusty.deb && apt-get update &&
apt-get install puppetmaster
```

- Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

- Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

- Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

- For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

- Log in to the server that you are going to set up as the Puppet Master, as a super user.
- Create the following directory.

```
/etc/puppet/environments/production/
```

- Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and [master] sections respectively. You need to update this command in order to define where to look for hiera.yaml file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- dns_alt_names=puppetmaster,puppet

This defines the hostnames that the Puppet Master uses to sign the certificates.

- autosign=true

This determines that the certificate will be signed automatically.

- Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new dns_alt_names.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common/releases
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-base/releases
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-base/releases
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-am/releases
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-am/releases

7. Formulate the Puppet Modules.
When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- Extract the wso2-wso2base-1.0.0.tar.gz file, rename the folder name as wso2base, and copy that folder to the /etc/puppet/environments/production/modules/ directory.
- Extract the wso2-wso2am_runtime-2.1.0.tar.gz file, rename the folder name as wso2am_runtime, and copy that folder to the /etc/puppet/environments/production/modules/ directory.
- Optionally, install the Java module from Puppet Forge.

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

- d. Install the stdlib module from Puppet Forge.
You need to install the stdlib module, because WSO2 Puppet Modules use some of the functions in the stdlib module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.
Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.
- Unzip the wso2-puppet-common-1.0.0.zip file and copy the hiera.yaml file to the /etc/puppet directory.
 - Edit the YAML data directory location as follows in the hiera.yaml file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- c. Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains the common Hiera data.
 - d. Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.
 - e. Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/product` ion directory.
9. Formulate the `site.pp` manifest.
- Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a `.pp` extension. The `site.pp` manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the `site.pp` manifest:
- a. Copy the `wso2-puppet-common-1.0.0/manifests/site.pp` file to the `/etc/puppet/environments/production/manifests` directory.
 - b. Construct the files/packs as follows:
 - i. Copy the WSO2 API-M pack file (`wso2am-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_runtime/files` directory.
 - ii. Create the following directory - `/etc/puppet/environments/production/modules/wso2base/files`
 - iii. Copy the JDK installation file (e.g., `jdk-8u112-linux-x64.tar.gz`) to the `/etc/puppet/environments/production/modules/wso2base/files` directory.
 - iv. Add the JDK version (home) and filename information in the `/etc/puppet/environments/production/modules/wso2base/manifests/java.pp` file.
 - v. Add the host entry in the `/etc/puppet/hieradata/production/wso2/common.yaml` file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up Puppet Agent

Carry out the following instructions on the second instance to set up the Puppet Agent.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the `/etc/hosts` file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent `/etc/puppet/puppet.conf` file by appending the following to the `[main]` section.
You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the `/etc/hosts` file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the WSO2 API-M Puppet Agent

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Modify the `deployment.conf` file as follows:

Pattern-1 Configurations Format

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M with pattern 1.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

```

product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>

```

Step 5 - Update the clustering related configurations

▼ [Click here for more information on updating the relevant clustering related configurations.](#)

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 1**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml file.

```

wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org

```

If you are using pattern 1, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml file.

```

wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org

```

If you are using pattern 2, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml file.

```
wso2::hosts_mapping:
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
    database
        driver_class_name:
            "%{hiera('wso2::datasources::mysql::driver_class_name')}"
        url:
            jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
        username:
            "%{hiera('wso2::datasources::mysql::username')}"
        password:
            "%{hiera('wso2::datasources::mysql::password')}"
        jndi_config: jdbc/WSO2AM_DB
        max_active:
            "%{hiera('wso2::datasources::common::max_active')}"
        max_wait:
            "%{hiera('wso2::datasources::common::max_wait')}"
        test_on_borrow:
            "%{hiera('wso2::datasources::common::test_on_borrow')}"
        default_auto_commit:
            "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
            "%{hiera('wso2::datasources::mysql::validation_query')}"
        validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes.
Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The WSO2 API-M GitHub repository includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepacked` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the WSO2 API-M Puppet Agent

Run the following command to set up and install WSO2 API-M 2.1.0.

1. Log in to Puppet Agent as a super user.
 2. Copy the modified deployment.conf file to the /opt directory.
 3. Copy the setup.sh file to the /opt directory and execute the following command.
- As the root user, you need to execute this command to add the execute permission to the setup.sh file.

[commandsetup.sh](#)

```
chmod 755 setup.sh
```

```
#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

- Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

Note that WSO2 API-M 2.1.0 gets installed in the following directory.

/mnt/wso2am-2.1.0

For API-M 2.1.0 the complete deployment time (deployment and server startup) is approximately 1 minute.

Using Puppet Modules to Set up WSO2 API-M with Pattern 2

The following diagram illustrates pattern 2 which consist of a stand-alone APIM setup with a single node deployment, with a single wso2am-analytics server instance. The databases used in this pattern are external MySQL databases.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware

Ensure that the minimum hardware requirements mentioned in the [hardware requirements](#) section are met. You can further fine tune your operating system for production by [tuning performance](#).

Software

- Oracle JDK 1.8
- Puppet 3.3.8 and above and below 4.0.0
- Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 2 to deploy WSO2 API-M in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Create three instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
 - 2.2 - Configure the Puppet Master
- Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agents
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore
- Step 7 - Run the Puppet Agents

Step 1 - Create three instances

Create three vanilla instances that do not have any configurations. You will configure these three instances in the subsequent steps as the Puppet Master, Puppet Agent for WSO2 API Manager Analytics, and Puppet Agent for WSO2 API Manager. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install  
ntp ; service ntp restart
```

- b. Navigate to the /tmp directory

```
cd /tmp
```

- c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb &&  
dpkg -i puppetlabs-release-trusty.deb && apt-get update &&  
apt-get install puppetmaster
```

3. Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

4. Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

5. Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

6. For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Create the following directory.

```
/etc/puppet/environments/production/
```

3. Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and [master] sections respectively.

You need to update this command in order to define where to look for hiera.yaml file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- dns_alt_names=puppetmaster,puppet

This defines the hostnames that the Puppet Master uses to sign the certificates.

- autosign=true

This determines that the certificate will be signed automatically.

4. Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new dns_alt_names.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-wso2base
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-wso2base
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am
wso2-wso2am_analytics-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am
wso2am-analytics-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am

7. Formulate the Puppet Modules.
- When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the

product.

- Extract the `wso2-wso2base-1.0.0.tar.gz` file, rename the folder name as `wso2base`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Extract the `wso2-wso2am_runtime-2.1.0.tar.gz` file, rename the folder name as `wso2am_runtime`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Extract `wso2-wso2am_analytics-2.1.0.tar.gz`, rename the folder name as `wso2am_analytics`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Optionally, install the [Java module from Puppet Forge](#).

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

- Install the [stdlib module from Puppet Forge](#). You need to install the `stdlib` module, because WSO2 Puppet Modules use some of the functions in the `stdlib` module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.

Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.

- Unzip the `wso2-puppet-common-1.0.0.zip` file and copy the `hiera.yaml` file to the `/etc/puppet` directory.
- Edit the YAML data directory location as follows in the `hiera.yaml` file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains the common Hiera data.
- Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.
- Unzip `wso2am-analytics-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.
This folder contains WSO2 API Manager Analytics specific Hiera data. You need to merge the common Hiera data and API Manager Analytics specific Hiera data.
- Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/product` ion directory.

If you already having the common hieradata and API Manager specific Hiera data, You have to merge the analytics hieradata with common hieradata and API Manager specific hieradata.

9. Formulate the site.pp manifest.

Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a `.pp`

p extension. The site.pp manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the site.pp manifest:

- Copy the wso2-puppet-common-1.0.0/manifests/site.pp file to the /etc/puppet/environments/production/manifests directory.
- Construct the files/packs as follows:
 - Copy the WSO2 API-M pack file (wso2am-2.1.0.zip) to the /etc/puppet/environments/production/modules/wso2am_runtime/files directory.
 - Copy the WSO2 API-M Analytics pack file (wso2am-analytics-2.1.0.zip) to the /etc/puppet/environments/production/modules/wso2am_analytics/files directory.
 - Create the following directory - /etc/puppet/environments/production/modules/wso2base/files
 - Copy the JDK installation file (e.g., jdk-8u112-linux-x64.tar.gz) to the /etc/puppet/environments/production/modules/wso2base/files directory.
 - Add the JDK archive name, Java home and installation directory information in the /etc/puppet/environments/production/modules/wso2base/manifests/java.pp file. Note that these values are treated as the defaults if no other value is specified via the hiera files.

```
$deploymentdir      = '/mnt/jdk-8u112',
$source            = 'jdk-8u112-linux-x64.tar.gz',
$java_home         = '/opt/java',
```

- Add the host entry in the /etc/puppet/hieradata/production/wso2/common.yaml file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents

You need to **carry out the following steps separately on each of the two instances** to configure and set up one instance as the WSO2 API-M Analytics server and the other instance as the WSO2 API-M server.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

- Log in to the server that you are going to set up as the Puppet Agent as a super user.
- Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

- Edit the /etc/hosts file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent

needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent `/etc/puppet/puppet.conf` file by appending the following to the `[main]` section.

You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the `/etc/hosts` file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agents

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Set the Facter variables by modifying the `deployment.conf` file in each of the Puppet Agent instances.

Pattern-2 API-M Analytics Pattern-2 API-M Format

The following are the configurations that you need when running the WSO2 API-M Analytics Puppet Agent node.

```
product_name=wso2am_analytics
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M with pattern 2.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-2
```

```
product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>
```

Step 5 - Update the clustering related configurations

[Click here for more information on updating the relevant clustering related configurations.](#)

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 2**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 1, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 2, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml file.

```
wso2::hosts_mapping:
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
    apim_keymanager:  
        ip: 192.168.57.186  
        name: km.dev.wso2.org  
    apim_store:  
        ip: 192.168.57.21  
        name: store.dev.wso2.org  
    apim_publisher:  
        ip: 192.168.57.219  
        name: pub.dev.wso2.org  
    apim_gateway_manager:  
        ip: 192.168.57.216  
        name: mgt-gw.dev.wso2.org  
    apim_gateway_worker:  
        ip: 192.168.57.247  
        name: gw.dev.wso2.org  
    apim_traffic_manager:  
        ip: 192.168.57.35  
        name: tm.dev.wso2.org  
    svn:  
        ip: 192.168.100.1  
        name: svn.gw.am.dev.wso2.org  
    apim_analytics_server:  
        ip: 192.168.57.29  
        name: analytics.dev.wso2.org  
    apim_gateway_manager_dmz:  
        ip: 192.168.57.5  
        name: dmz-mgt-gw.dev.wso2.org  
    apim_gateway_worker_dmz:  
        ip: 192.168.57.218  
        name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
    database
        driver_class_name:
            "%{hiera('wso2::datasources::mysql::driver_class_name')}"
        url:
            jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
        username:
            "%{hiera('wso2::datasources::mysql::username')}"
        password:
            "%{hiera('wso2::datasources::mysql::password')}"
        jndi_config: jdbc/WSO2AM_DB
        max_active:
            "%{hiera('wso2::datasources::common::max_active')}"
        max_wait:
            "%{hiera('wso2::datasources::common::max_wait')}"
        test_on_borrow:
            "%{hiera('wso2::datasources::common::test_on_borrow')}"
        default_auto_commit:
            "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
            "%{hiera('wso2::datasources::mysql::validation_query')}"
        validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes.
Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The [WSO2 API-M GitHub repository](#) includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepended` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the Puppet Agents

Carry out the instructions to run the Puppet Agent **separately on both the Puppet Agent instances** in the following order.

1. WSO2 API-M Analytics Puppet Agent
2. WSO2 API-M Puppet Agent

1. Log in to Puppet Agent as a super user.
2. Copy the modified deployment.conf file to the /opt directory.
3. Copy the setup.sh file to the /opt directory and execute the following command.
As the root user, you need to execute this command to add the execute permission to the setup.sh file.

commandsetup.sh

```
chmod 755 setup.sh

#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

- Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

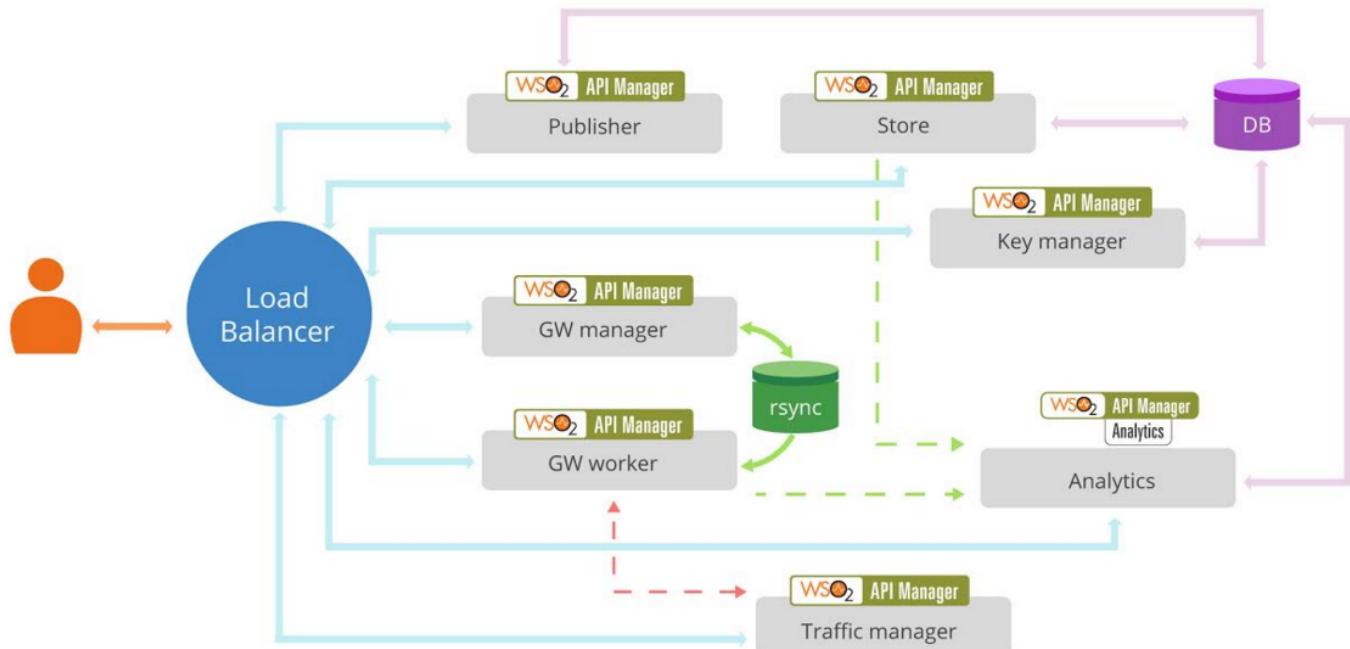
When running WSO2 API-M or WSO2 API-M Analytics, <PRODUCT_HOME> corresponds to the following directory.

Node	Installed Directory
WSO2 API-M Analytics	/mnt/wso2am-analytics-2.1.0
WSO2 API-M	/mnt/wso2am-2.1.0

Using Puppet Modules to Set up WSO2 API-M with Pattern 3

The following diagram illustrates pattern 3 that consists of a fully distributed WSO2 API Manager (WSO2 API-M) setup (including a Gateway cluster of one manager and one worker) with a single `wso2am-analytics` server instance. This pattern uses external MySQL databases.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 3 to deploy WSO2 API-M in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Create eight instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
 - 2.2 - Configure the Puppet Master
- Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agents
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore
- Step 7 - Run the Puppet Agents

Step 1 - Create eight instances

Create eight vanilla instances, which do not have any configurations. In the subsequent steps you will configure one instance as the Puppet Master and the rest of the instances as Puppet Agents for the following nodes. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the `puppet.conf` file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Node	Hieradata file	Hostname
Store	<code>api-store.yaml</code>	<code>store.dev.wso2.org</code>
Publisher	<code>api-publisher.yaml</code>	<code>pub.dev.wso2.org</code>
Gateway Manager	<code>gateway-manager.yaml</code>	<code>mgt-gw.dev.wso2.org</code>
Gateway Worker	<code>gateway-worker.yaml</code>	<code>gw.dev.wso2.org</code>
Key Manager	<code>api-key-manager.yaml</code>	<code>km.dev.wso2.org</code>
Traffic Manager	<code>traffic-manager.yaml</code>	<code>tm.dev.wso2.org</code>
Analytics	<code>default.yaml</code>	<code>analytics.dev.wso2.org</code>

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install ntp ; service ntp restart
```

b. Navigate to the /tmp directory

```
cd /tmp
```

c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppetmaster
```

3. Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

4. Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

5. Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

6. For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Create the following directory.

```
/etc/puppet/environments/production/
```

3. Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and [master] sections respectively.
You need to update this command in order to define where to look for hiera.yaml file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- `dns_alt_names=puppetmaster,puppet`
This defines the hostnames that the Puppet Master uses to sign the certificates.
- `autosign=true`
This determines that the certificate will be signed automatically.

4. Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new `dns_alt_names`.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common/releases/download/v1.0.0/wso2-puppet-common-1.0.0.zip
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-wso2base/releases/download/v1.0.0/wso2-wso2base-1.0.0.tar.gz
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-wso2base/releases/download/v1.0.0/wso2base-puppet-module-hieradata-1.0.0.zip
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am/releases/download/v2.1.0/wso2-wso2am_runtime-2.1.0.tar.gz
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am/releases/download/v2.1.0/wso2am-runtime-puppet-module-hieradata-2.1.0.zip
wso2-wso2am_analytics-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am_analytics/releases/download/v2.1.0/wso2-wso2am_analytics-2.1.0.tar.gz
wso2am-analytics-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am_analytics/releases/download/v2.1.0/wso2am-analytics-puppet-module-hieradata-2.1.0.zip

7. Formulate the Puppet Modules.
When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- a. Extract the `wso2-wso2base-1.0.0.tar.gz` file, rename the folder name as `wso2base`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- b. Extract the `wso2-wso2am_runtime-2.1.0.tar.gz` file, rename the folder name as `wso2am_runtime`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- c. Extract `wso2-wso2am_analytics-2.1.0.tar.gz`, rename the folder name as `wso2am_analytics`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- d. Optionally, install the Java module from Puppet Forge.

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

- e. Install the `stdlib` module from Puppet Forge. You need to install the `stdlib` module, because WSO2 Puppet Modules use some of the functions in the `stdlib` module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.

Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.

- Unzip the `wso2-puppet-common-1.0.0.zip` file and copy the `hiera.yaml` file to the `/etc/puppet` directory.
- Edit the YAML data directory location as follows in the `hiera.yaml` file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the `hieradata` folder to the `/etc/puppet/` directory.

This folder contains the common Hiera data.

- Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.

This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.

- Unzip `wso2am-analytics-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.

This folder contains WSO2 API Manager Analytics specific Hiera data. You need to merge the common Hiera data and API Manager Analytics specific Hiera data.

- Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/production` directory.

If you already having the common hieradata and API Manager specific Hiera data, You have to merge the analytics hieradata with common hieradata and API Manager specific hieradata.

9. Formulate the `site.pp` manifest.

Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a `.pp` extension. The `site.pp` manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the `site.pp` manifest:

- Copy the `wso2-puppet-common-1.0.0/manifests/site.pp` file to the `/etc/puppet/environments/production/manifests` directory.
- Construct the files/packs as follows:
 - Copy the WSO2 API-M pack file (`wso2am-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_runtime/files` directory.
 - Copy the WSO2 API-M Analytics pack file (`wso2am-analytics-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_analytics/files` directory.

- iii. Create the following directory - /etc/puppet/environments/production/modules/wso2base/files
- iv. Copy the JDK installation file (e.g., jdk-8u112-linux-x64.tar.gz) to the /etc/puppet/environments/production/modules/wso2base/files directory.
- v. Add the JDK archive name, Java home and installation directory information in the /etc/puppet/environments/production/modules/wso2base/manifests/java.pp file. Note that these values are treated as the defaults if no other value is specified via the hiera files.

```
$deploymentdir      = '/mnt/jdk-8u112',
$source            = 'jdk-8u112-linux-x64.tar.gz',
$java_home         = '/opt/java',
```

- vi. Add the host entry in the /etc/puppet/hieradata/production/wso2/common.yaml file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents

You need to **carry out the following steps separately on each of the seven instances** to configure and set up an instance for the Store, Publisher, Key Manager, Traffic Manager, Analytics, Gateway Manager, and the Gateway Worker.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the /etc/hosts file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.

2. Modify the Puppet Agent `/etc/puppet/puppet.conf` file by appending the following to the `[main]` section:

You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the `/etc/hosts` file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agents

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Set the Facter variables by modifying the `deployment.conf` file in each of the Puppet Agent instances.

API-M Analytics Store Publisher Key Manager Gateway Manager Format

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Analytics with pattern 1.

```
product_name=wso2am_analytics
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Store with pattern 3.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-store
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-3
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Publisher with pattern 3.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-publisher
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-3
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Key Manager with pattern 3.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-key-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-3
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Manager with pattern 3.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-3
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Worker with pattern 3.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-worker
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-3
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M

Traffic Manager with pattern 3.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=traffic-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-3
```

```
product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>
```

Step 5 - Update the clustering related configurations

▼ [Click here for more information on updating the relevant clustering related configurations.](#)

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 3**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 1, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 2, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml file.

```
wso2::hosts_mapping:
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
database
    driver_class_name:
    "%{hiera('wso2::datasources::mysql::driver_class_name')}"
    url:
    jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
    username:
    "%{hiera('wso2::datasources::mysql::username')}"
    password:
    "%{hiera('wso2::datasources::mysql::password')}"
    jndi_config: jdbc/WSO2AM_DB
    max_active:
    "%{hiera('wso2::datasources::common::max_active')}"
    max_wait:
    "%{hiera('wso2::datasources::common::max_wait')}"
    test_on_borrow:
    "%{hiera('wso2::datasources::common::test_on_borrow')}"
    default_auto_commit:
    "%{hiera('wso2::datasources::common::default_auto_commit')}"
    validation_query:
    "%{hiera('wso2::datasources::mysql::validation_query')}"
    validation_interval:
    "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes.
Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The [WSO2 API-M GitHub repository](#) includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepended` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the Puppet Agents

Carry out the instructions to run the Puppet Agent **separately on each of the seven Puppet Agent instances** in the following order.

1. Initially, start the following nodes in given order:
 - a. WSO2 API-M Analytics
 - b. Traffic Manager
 - c. Gateway Manager
 - d. Gateway Worker
2. Thereafter, start the Key Manager, Publisher, and Store in any order.

1. Log in to Puppet Agent as a super user.

2. Copy the modified deployment.conf file to the /opt directory.
3. Copy the setup.sh file to the /opt directory and execute the following command.
As the root user, you need to execute this command to add the execute permission to the setup.sh file.

commandsetup.sh

```
chmod 755 setup.sh
```

```
#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

4. Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

When running WSO2 API-M or WSO2 API-M Analytics, <PRODUCT_HOME> corresponds to the following directory.

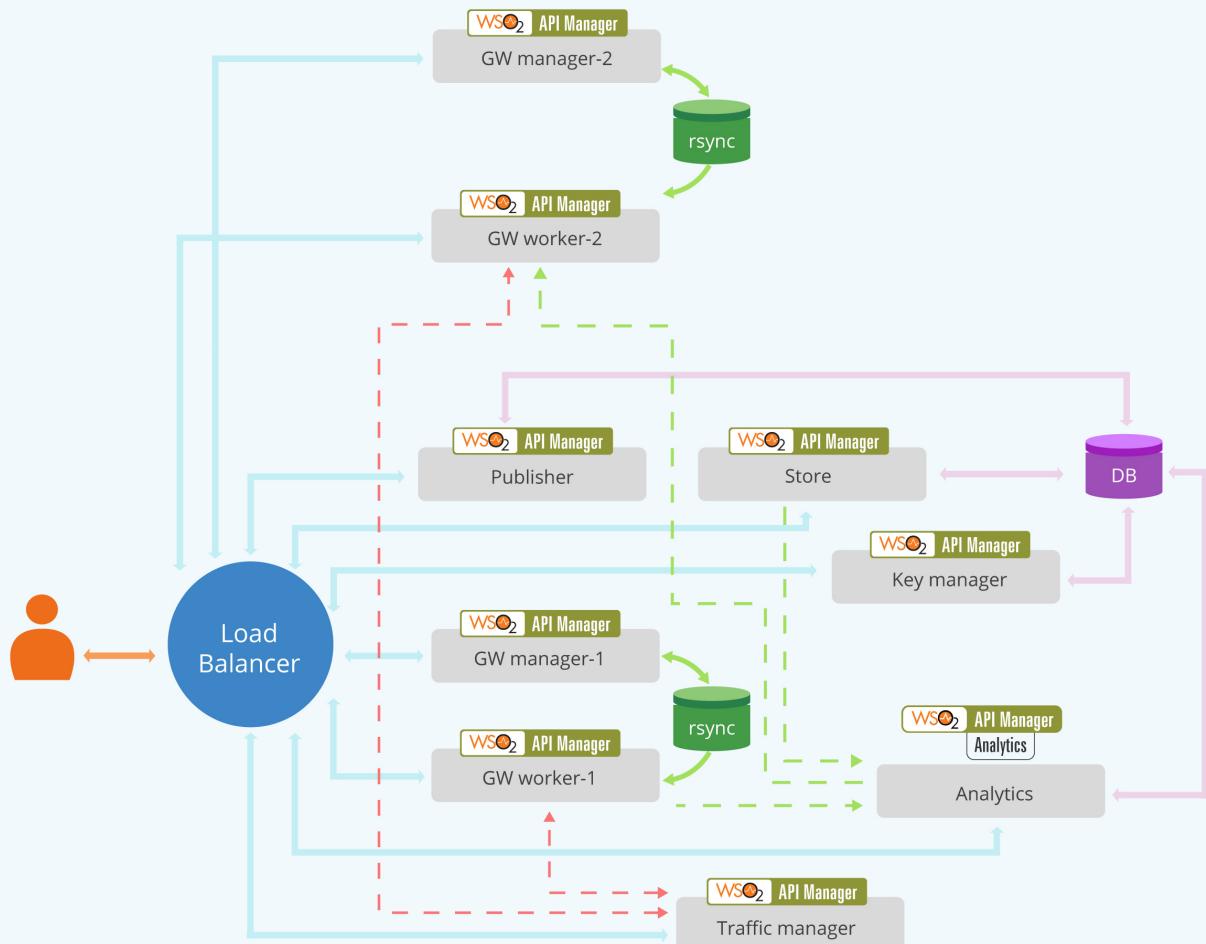
Node	Installed Directory
WSO2 API-M Analytics	/mnt/wso2am-analytics-2.1.0
WSO2 API-M	/mnt/wso2am-2.1.0
<p>This is applicable to the following nodes. Store, Publisher, Key Manager, Traffic Manager, Analytics, Gateway Manager, and Gateway Worker.</p>	

Using Puppet Modules to Set up WSO2 API-M with Pattern 4

The following diagram illustrates pattern 4 which consists of a fully distributed API-M carry including two Gateway clusters, where each has one manager and one worker, with a single wso2am-analytics server instance. You can have the gateway environments in any preferred environment (e.g., local-area network (LAN) and demilitarized zone (DMZ)).

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are

newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 4 to deploy WSO2 API-M in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to

Ubuntu 14.04 LTS.

- Step 1 - Create ten instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
 - 2.2 - Configure the Puppet Master
- Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agents
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore
- Step 7 - Run the Puppet Agents

Step 1 - Create ten instances

Create ten vanilla instances, which do not have any configurations. In the subsequent steps you will configure one instance as the Puppet Master and the rest of the instances as Puppet Agents for the following nodes. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Node	Hieradata file	Hostname
Store	api-store.yaml	store.dev.wso2.org
Publisher	api-publisher.yaml	pub.dev.wso2.org
Gateway Manager DMZ	gateway-manager-dmz.yaml	dmz-mgt-gw.dev.wso2.org
Gateway Worker DMZ	gateway-worker-dmz.yaml	dmz-gw.dev.wso2.org
Gateway Manager LAN	gateway-manager-lan.yaml	mgt-gw.dev.wso2.org

Gateway Worker LAN	gateway-worker-lan.yaml	gw.dev.wso2.org
Key Manager	api-key-manager.yaml	km.dev.wso2.org
Traffic Manager	traffic-manager.yaml	tm.dev.wso2.org
Analytics	default.yaml	analytics.dev.wso2.org

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install ntp ; service ntp restart
```

- b. Navigate to the /tmp directory

```
cd /tmp
```

- c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppetmaster
```

3. Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

4. Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

5. Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

6. For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Create the following directory.

```
/etc/puppet/environments/production/
```

3. Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and [master] sections respectively.

You need to update this command in order to define where to look for hiera.yaml file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- dns_alt_names=puppetmaster,puppet
This defines the hostnames that the Puppet Master uses to sign the certificates.
- autosign=true
This determines that the certificate will be signed automatically.

4. Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new dns_alt_names.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-wso2base
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-wso2base
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am
wso2-wso2am_analytics-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am
wso2am-analytics-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am

7. Formulate the Puppet Modules.

When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- Extract the `wso2-wso2base-1.0.0.tar.gz` file, rename the folder name as `wso2base`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Extract the `wso2-wso2am_runtime-2.1.0.tar.gz` file, rename the folder name as `wso2am_runtime`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Extract `wso2-wso2am_analytics-2.1.0.tar.gz`, rename the folder name as `wso2am_analytics`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Optionally, install the Java module from Puppet Forge.

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

e. Install the stdlib module from Puppet Forge.

You need to install the `stdlib` module, because WSO2 Puppet Modules use some of the functions in the `stdlib` module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.

Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.

- Unzip the `wso2-puppet-common-1.0.0.zip` file and copy the `hiera.yaml` file to the `/etc/puppet` directory.
- Edit the YAML data directory location as follows in the `hiera.yaml` file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the `hieradata` folder to the `/etc/puppet/` directory.

This folder contains the common Hiera data.

- Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.

This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.

- Unzip `wso2am-analytics-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory.

This folder contains WSO2 API Manager Analytics specific Hiera data. You need to merge the common Hiera data and API Manager Analytics specific Hiera data.

- Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/product` directory.

If you already having the common hieradata and API Manager specific Hiera data, You have to merge the analytics hieradata with common hieradata and API Manager specific hieradata.

9. Formulate the site.pp manifest.

Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a .pp extension. The site.pp manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the site.pp manifest:

- Copy the wso2-puppet-common-1.0.0/manifests/site.pp file to the /etc/puppet/environments/production/manifests directory.
- Construct the files/packs as follows:
 - Copy the WSO2 API-M pack file (wso2am-2.1.0.zip) to the /etc/puppet/environments/production/modules/wso2am_runtime/files directory.
 - Copy the WSO2 API-M Analytics pack file (wso2am-analytics-2.1.0.zip) to the /etc/puppet/environments/production/modules/wso2am_analytics/files directory.
 - Create the following directory - /etc/puppet/environments/production/modules/wso2base/files
 - Copy the JDK installation file (e.g., jdk-8u112-linux-x64.tar.gz) to the /etc/puppet/environments/production/modules/wso2base/files directory.
 - Add the JDK archive name, Java home and installation directory information in the /etc/puppet/environments/production/modules/wso2base/manifests/java.pp file. Note that these values are treated as the defaults if no other value is specified via the hiera files.

```
$deploymentdir      = '/mnt/jdk-8u112',
$source             = 'jdk-8u112-linux-x64.tar.gz',
$java_home          = '/opt/java',
```

- Add the host entry in the /etc/puppet/hieradata/production/wso2/common.yaml file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents

You need to **carry out the following steps separately on each of the nine instances** to configure and set up an instance for the Store, Publisher, Key Manager, Traffic Manager, Analytics, Gateway Manager and the Gateway Worker.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

- Log in to the server that you are going to set up as the Puppet Agent as a super user.
- Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the `/etc/hosts` file that corresponds to your Puppet Agent as follows:
The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent `/etc/puppet/puppet.conf` file by appending the following to the `[main]` section

You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the `/etc/hosts` file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agents

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Set the Facter variables by modifying the `deployment.conf` file in each of the Puppet Agent instances.

API-M Analytics Store Publisher Key Manager	Gateway Manager in DMZ	Gateway Worker in DMZ
Gateway Manager in LAN	Gateway Worker in LAN	Traffic Manager Format

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Analytics with pattern 1.

```
product_name=wso2am_analytics
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Store with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-store
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Publisher with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-publisher
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Key Manager with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-key-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Manager in DMZ with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-manager-dmz
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M

Gateway Worker in DMZ with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-worker-dmz
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Manager in LAN with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-manager-lan
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Worker in LAN with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-worker-lan
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Traffic Manager with pattern 4.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=traffic-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-4
```

```

product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>

```

Step 5 - Update the clustering related configurations

Click here for more information on updating the relevant clustering related configurations.

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 4**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml file.

```

wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org

```

If you are using pattern 1, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml file.

```

wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org

```

If you are using pattern 2, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml file.

```
wso2::hosts_mapping:
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
    database
        driver_class_name:
            "%{hiera('wso2::datasources::mysql::driver_class_name')}"
        url:
            jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
        username:
            "%{hiera('wso2::datasources::mysql::username')}"
        password:
            "%{hiera('wso2::datasources::mysql::password')}"
        jndi_config: jdbc/WSO2AM_DB
        max_active:
            "%{hiera('wso2::datasources::common::max_active')}"
        max_wait:
            "%{hiera('wso2::datasources::common::max_wait')}"
        test_on_borrow:
            "%{hiera('wso2::datasources::common::test_on_borrow')}"
        default_auto_commit:
            "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
            "%{hiera('wso2::datasources::mysql::validation_query')}"
        validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

- Configure deployment synchronization in each of the Gateway related nodes. Use one of the following ways to carryout this configuration.

- Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The [WSO2 API-M GitHub repository](#) includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepended` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the Puppet Agents

Carry out the instructions to run the Puppet Agent **separately on each of the nine Puppet Agent instances** in the following order.

1. Initially, start the following nodes in given order:
 - a. WSO2 API-M Analytics
 - b. Traffic Manager
 - c. Gateway Manager in DMZ and LAN
 - d. Gateway Worker in DMZ and LAN

The respective Gateway DMZ and LAN nodes can start in parallel.

2. Thereafter, start the Key Manager, Publisher and Store in any order.

1. Log in to Puppet Agent as a super user.
 2. Copy the modified deployment.conf file to the /opt directory.
 3. Copy the setup.sh file to the /opt directory and execute the following command.
- As the root user, you need to execute this command to add the execute permission to the setup.sh file.

```
commandsetup.sh
```

```
chmod 755 setup.sh
```

```
#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

4. Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

When running WSO2 API-M or WSO2 API-M Analytics, <PRODUCT_HOME> corresponds to the following directory.

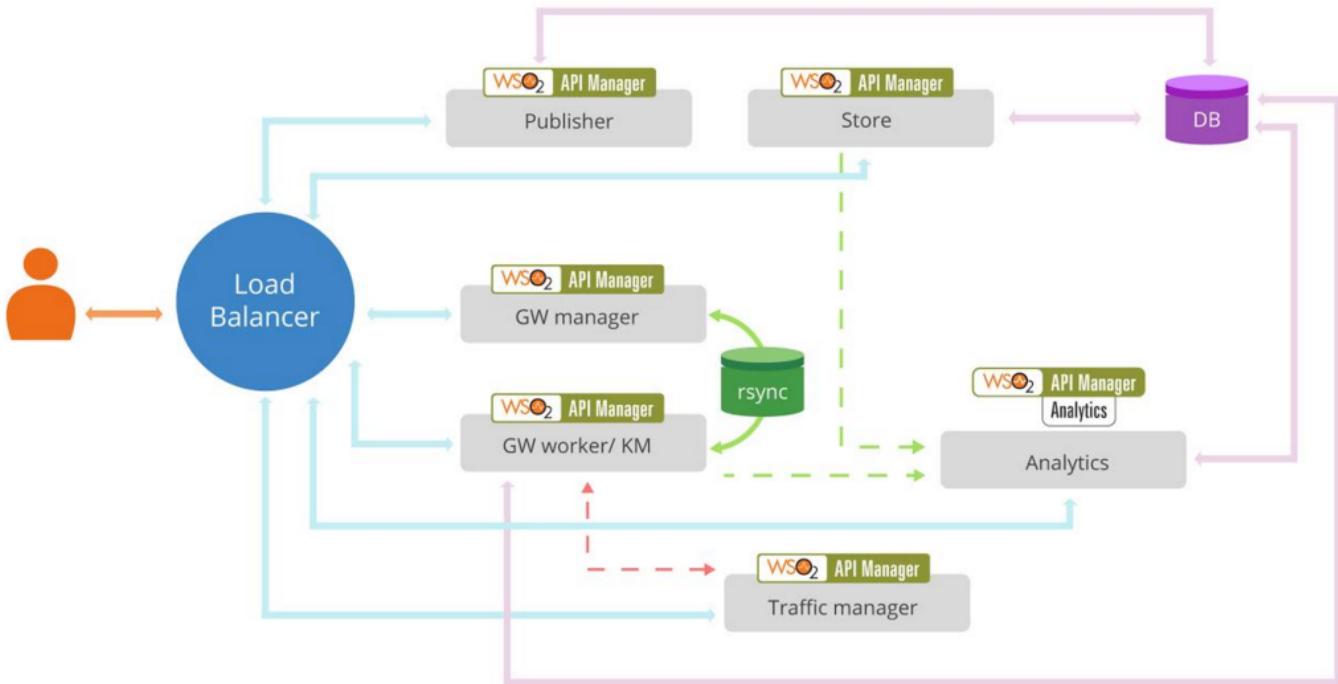
Node	Installed Directory
WSO2 API-M Analytics	/mnt/wso2am-analytics-2.1.0
WSO2 API-M	/mnt/wso2am-2.1.0
<p>This is applicable to the following nodes. Store, Publisher, Key Manager, Traffic Manager, Analytics, Gateway Manager (DMZ & LAN), and Gateway Worker (DMZ & LAN).</p>	

Using Puppet Modules to Set up WSO2 API-M with Pattern 5

The following diagram illustrates pattern 5 that consists of a distributed WSO2 API Manager (WSO2 API-M) setup including a Gateway cluster of one manager and one worker and the Gateway worker is merged with the Key

Manager. It also consists of a single `wso2am-analytics` server instance and it uses external MySQL databases.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 5 to deploy WSO2 API-M in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Create seven instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
 - 2.2 - Configure the Puppet Master

- Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agents
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore
- Step 7 - Run the Puppet Agents

Step 1 - Create seven instances

Create seven vanilla instances, which do not have any configurations. In the subsequent steps you will configure one instance as the Puppet Master and the rest of the instances as Puppet Agents for the following nodes. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Node	Hieradata file	Hostname
Store	api-store.yaml	store.dev.wso2.org
Publisher	api-publisher.yaml	pub.dev.wso2.org
Gateway Manager	gateway-manager.yaml	mgt-gw.dev.wso2.org
Gateway Worker &	gw-plus-km.yaml	am.dev.wso2.org
Key Manager		
Traffic Manager	traffic-manager.yaml	tm.dev.wso2.org
Analytics	default.yaml	analytics.dev.wso2.org

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpd date pool.ntp.org ; apt-get update && sudo apt-get -y install  
ntp ; service ntp restart
```

- b. Navigate to the /tmp directory

```
cd /tmp
```

- c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb &&  
dpkg -i puppetlabs-release-trusty.deb && apt-get update &&  
apt-get install puppetmaster
```

3. Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

4. Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

5. Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

6. For production deployments [configure Puppet Master with Passenger and Apache](#)

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Create the following directory.

```
/etc/puppet/environments/production/
```

3. Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and

[master] sections respectively.
You need to update this command in order to define where to look for `hiera.yaml` file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- `dns_alt_names=puppetmaster,puppet`
This defines the hostnames that the Puppet Master uses to sign the certificates.
- `autosign=true`
This determines that the certificate will be signed automatically.

4. Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new `dns_alt_names`.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common/releases/download/v1.0.0/wso2-puppet-common-1.0.0.zip
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-wso2base/releases/download/v1.0.0/wso2-wso2base-1.0.0.tar.gz
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-wso2base/releases/download/v1.0.0/wso2base-puppet-module-hieradata-1.0.0.zip
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am-runtime/releases/download/v2.1.0/wso2-wso2am_runtime-2.1.0.tar.gz
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am-runtime/releases/download/v2.1.0/wso2am-runtime-puppet-module-hieradata-2.1.0.zip
wso2-wso2am_analytics-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am-analytics/releases/download/v2.1.0/wso2-wso2am_analytics-2.1.0.tar.gz
wso2am-analytics-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am-analytics/releases/download/v2.1.0/wso2am-analytics-puppet-module-hieradata-2.1.0.zip

7. Formulate the Puppet Modules.
When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- a. Extract the `wso2-wso2base-1.0.0.tar.gz` file, rename the folder name as `wso2base`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- b. Extract the `wso2-wso2am_runtime-2.1.0.tar.gz` file, rename the folder name as `wso2am_runtime`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- c. Extract `wso2-wso2am_analytics-2.1.0.tar.gz`, rename the folder name as `wso2am_analyti`

- cs, and copy that folder to the /etc/puppet/environments/production/modules/ directory.
- Optional, install the Java module from Puppet Forge.

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

- Install the stdlib module from Puppet Forge.
You need to install the stdlib module, because WSO2 Puppet Modules use some of the functions in the stdlib module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

- Formulate the Hiera data.
Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.
 - Unzip the wso2-puppet-common-1.0.0.zip file and copy the hiera.yaml file to the /etc/puppet directory.
 - Edit the YAML data directory location as follows in the hiera.yaml file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- Unzip the wso2base-puppet-module-hieradata-1.0.0.zip file and copy the hieradata folder to the /etc/puppet/ directory.
This folder contains the common Hiera data.
- Unzip wso2am-runtime-puppet-module-hieradata-2.1.0.zip and copy the hieradata folder to the /etc/puppet/ directory.
This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.
- Unzip wso2am-analytics-puppet-module-hieradata-2.1.0.zip and copy the hieradata folder to the /etc/puppet/ directory.
This folder contains WSO2 API Manager Analytics specific Hiera data. You need to merge the common Hiera data and API Manager Analytics specific Hiera data.
- Rename the /etc/puppet/hieradata/dev directory to the /etc/puppet/hieradata/production directory.

If you already having the common hieradata and API Manager specific Hiera data, You have to merge the analytics hieradata with common hieradata and API Manager specific hieradata.

- Formulate the site.pp manifest.
Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a .pp extension. The site.pp manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the site.pp manifest:
 - Copy the wso2-puppet-common-1.0.0/manifests/site.pp file to the /etc/puppet/environments/production/manifests directory.
 - Construct the files/packs as follows:
 - Copy the WSO2 API-M pack file (wso2am-2.1.0.zip) to the /etc/puppet/environments

- /production/modules/wso2am_runtime/files directory.
- ii. Copy the WSO2 API-M Analytics pack file (wso2am-analytics-2.1.0.zip) to the /etc/puppet/environments/production/modules/wso2am_analytics/files directory.
 - iii. Create the following directory - /etc/puppet/environments/production/modules/wso2base/files
 - iv. Copy the JDK installation file (e.g., jdk-8u112-linux-x64.tar.gz) to the /etc/puppet/environments/production/modules/wso2base/files directory.
 - v. Add the JDK archive name, Java home and installation directory information in the /etc/puppet/environments/production/modules/wso2base/manifests/java.pp file. Note that these values are treated as the defaults if no other value is specified via the hiera files.

```
$deploymentdir      = '/mnt/jdk-8u112',
$source            = 'jdk-8u112-linux-x64.tar.gz',
$java_home         = '/opt/java',
```

- vi. Add the host entry in the /etc/puppet/hieradata/production/wso2/common.yaml file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents

You need to **carry out the following steps separately on each of the six instances** to configure and set up an instance for the Store, Publisher, Traffic Manager, Analytics, Gateway Manager and the Gateway Worker/Key Manager.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the /etc/hosts file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent /etc/puppet/puppet.conf file by appending the following to the [main] section.

You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the /etc/hosts file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agents

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Set the Facter variables by modifying the deployment.conf file in each of the Puppet Agent instances.

API-M Analytics	Store	Publisher	Gateway Manager	Gateway Worker/Key Manager	Traffic Manager	Format
-----------------	-------	-----------	-----------------	----------------------------	-----------------	--------

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Analytics with pattern 1.

```
product_name=wso2am_analytics
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Store with pattern 5.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-store
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-5
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Publisher with pattern 5.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=api-publisher
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-5
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Manager with pattern 5.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-5
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Worker/Key Manager with pattern 5.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gw-plus-km
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-5
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Traffic Manager with pattern 5.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=traffic-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-5
```

```

product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>

```

Step 5 - Update the clustering related configurations

[Click here for more information on updating the relevant clustering related configurations.](#)

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 5**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml file.

```

wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org

```

If you are using pattern 1, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml file.

```

wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org

```

If you are using pattern 2, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml file.

```
wso2::hosts_mapping:
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
    database
        driver_class_name:
            "%{hiera('wso2::datasources::mysql::driver_class_name')}"
        url:
            jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
        username:
            "%{hiera('wso2::datasources::mysql::username')}"
        password:
            "%{hiera('wso2::datasources::mysql::password')}"
        jndi_config: jdbc/WSO2AM_DB
        max_active:
            "%{hiera('wso2::datasources::common::max_active')}"
        max_wait:
            "%{hiera('wso2::datasources::common::max_wait')}"
        test_on_borrow:
            "%{hiera('wso2::datasources::common::test_on_borrow')}"
        default_auto_commit:
            "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
            "%{hiera('wso2::datasources::mysql::validation_query')}"
        validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes.
Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The [WSO2 API-M GitHub repository](#) includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepended` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the Puppet Agents

Carry out the instructions to run the Puppet Agent **separately on each of the six Puppet Agent instances** in the following order.

1. Initially, start the following nodes in given order:
 - a. WSO2 API-M Analytics
 - b. Traffic Manager
 - c. Gateway Manager
 - d. Gateway Worker/Key Manager
2. Thereafter, start the Publisher and Store in any order.

1. Log in to Puppet Agent as a super user.

2. Copy the modified deployment.conf file to the /opt directory.
3. Copy the setup.sh file to the /opt directory and execute the following command.
As the root user, you need to execute this command to add the execute permission to the setup.sh file.

commandsetup.sh

```
chmod 755 setup.sh
```

```
#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

4. Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

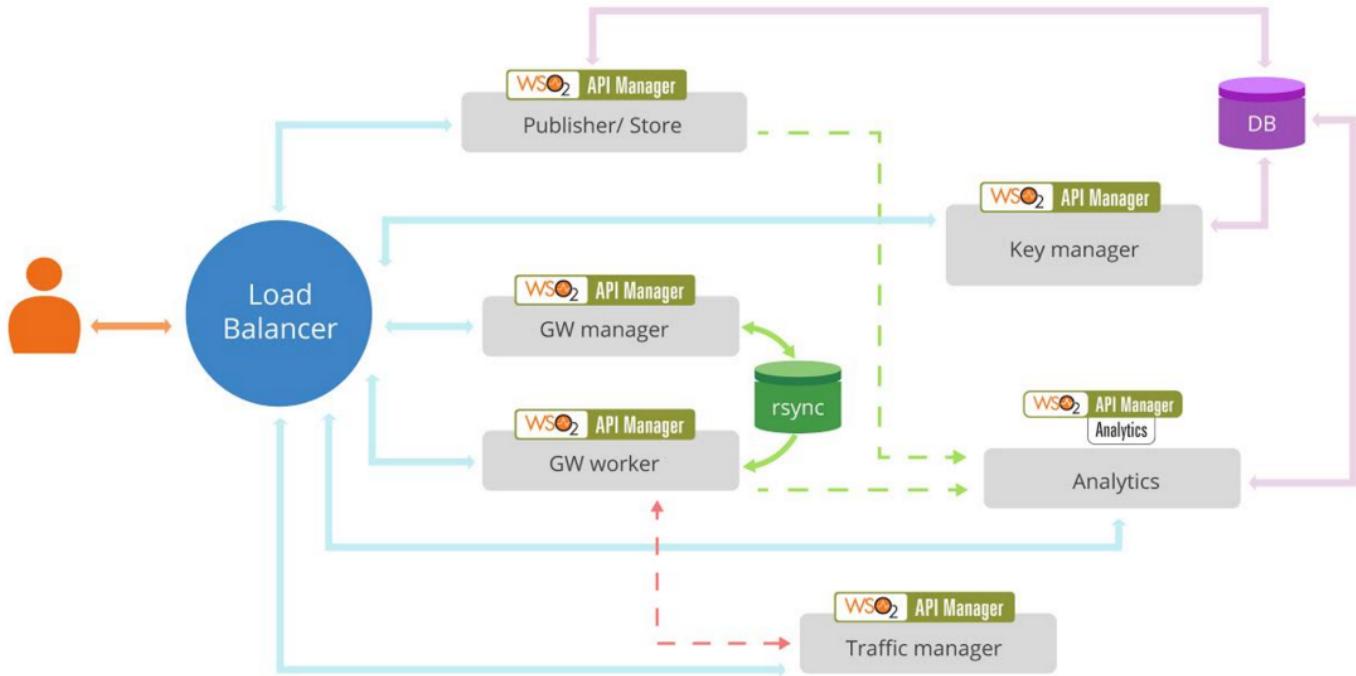
When running WSO2 API-M or WSO2 API-M Analytics, <PRODUCT_HOME> corresponds to the following directory.

Node	Installed Directory
WSO2 API-M Analytics	/mnt/wso2am-analytics-2.1.0
WSO2 API-M	/mnt/wso2am-2.1.0
This is applicable to the following nodes. Store, Publisher, Traffic Manager, Analytics, Gateway Manager, and Gateway Worker & Key Manager.	

Using Puppet Modules to Set up WSO2 API-M with Pattern 6

The following diagram illustrates pattern 6 that consists of a distributed WSO2 API Manager (WSO2 API-M) setup (including a Gateway cluster of one manager and one worker) of which the Publisher is merged with the Store. It also consists of a single wso2am-analytics server instance and it uses external MySQL databases.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 6 to deploy WSO2 API-M in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Set up seven instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
 - 2.2 - Configure the Puppet Master
- Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agents
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore

- Step 7 - Run the Puppet Agents

Step 1 - Set up seven instances

Create seven vanilla instances, which do not have any configurations. In the subsequent steps you will configure one instance as the Puppet Master and the rest of the instances as Puppet Agents for the following nodes. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Node	Hieradata file	Hostname
Publisher & Store	publisher-plus-store.yaml	am.dev.wso2.org
Gateway Manager	gateway-manager.yaml	mgt-gw.dev.wso2.org
Gateway Worker	gateway-worker.yaml	gw.dev.wso2.org
Key Manager	api-key-manager.yaml	km.dev.wso2.org
Traffic Manager	traffic-manager.yaml	tm.dev.wso2.org
Analytics	default.yaml	analytics.dev.wso2.org

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install ntp ; service ntp restart
```

b. Navigate to the /tmp directory

```
cd /tmp
```

c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppetmaster
```

3. Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

4. Set the hostname property by removing the current entry and adding the following in the etc/hostname file.

```
puppetmaster
```

5. Edit your Puppet Master hosts file /etc/hosts as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

6. For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Create the following directory.

```
/etc/puppet/environments/production/
```

3. Modify the Puppet Master /etc/puppet/puppet.conf file by appending the following to the [main] and [master] sections respectively.
You need to update this command in order to define where to look for hiera.yaml file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- `dns_alt_names=puppetmaster,puppet`
This defines the hostnames that the Puppet Master uses to sign the certificates.
- `autosign=true`
This determines that the certificate will be signed automatically.

4. Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

5. Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new `dns_alt_names`.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common/releases/download/v1.0.0/wso2-puppet-common-1.0.0.zip
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-wso2base/releases/download/v1.0.0/wso2-wso2base-1.0.0.tar.gz
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-wso2base/releases/download/v1.0.0/wso2base-puppet-module-hieradata-1.0.0.zip
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am/releases/download/v2.1.0/wso2-wso2am_runtime-2.1.0.tar.gz
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am/releases/download/v2.1.0/wso2am-runtime-puppet-module-hieradata-2.1.0.zip
wso2-wso2am_analytics-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am_analytics/releases/download/v2.1.0/wso2-wso2am_analytics-2.1.0.tar.gz
wso2am-analytics-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am_analytics/releases/download/v2.1.0/wso2am-analytics-puppet-module-hieradata-2.1.0.zip

7. Formulate the Puppet Modules.
When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- a. Extract the `wso2-wso2base-1.0.0.tar.gz` file, rename the folder name as `wso2base`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- b. Extract the `wso2-wso2am_runtime-2.1.0.tar.gz` file, rename the folder name as `wso2am_runtime`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- c. Extract `wso2-wso2am_analytics-2.1.0.tar.gz`, rename the folder name as `wso2am_analytics`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- d. Optionally, install the Java module from Puppet Forge.

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

- e. Install the `stdlib` module from Puppet Forge. You need to install the `stdlib` module, because WSO2 Puppet Modules use some of the functions in the `stdlib` module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.

Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.

- Unzip the `wso2-puppet-common-1.0.0.zip` file and copy the `hiera.yaml` file to the `/etc/puppet` directory.
- Edit the YAML data directory location as follows in the `hiera.yaml` file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the `hieradata` folder to the `/etc/puppet/` directory. This folder contains the common Hiera data.
- Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory. This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.
- Unzip `wso2am-analytics-puppet-module-hieradata-2.1.0.zip` and copy the `hieradata` folder to the `/etc/puppet/` directory. This folder contains WSO2 API Manager Analytics specific Hiera data. You need to merge the common Hiera data and API Manager Analytics specific Hiera data.
- Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/production` directory.

If you already having the common hieradata and API Manager specific Hiera data, You have to merge the analytics hieradata with common hieradata and API Manager specific hieradata.

9. Formulate the `site.pp` manifest.

Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a `.pp` extension. The `site.pp` manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the `site.pp` manifest:

- Copy the `wso2-puppet-common-1.0.0/manifests/site.pp` file to the `/etc/puppet/environments/production/manifests` directory.
- Construct the files/packs as follows:
 - Copy the WSO2 API-M pack file (`wso2am-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_runtime/files` directory.
 - Copy the WSO2 API-M Analytics pack file (`wso2am-analytics-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_analytics/files` directory.

- iii. Create the following directory - /etc/puppet/environments/production/modules/wso2base/files
- iv. Copy the JDK installation file (e.g., jdk-8u112-linux-x64.tar.gz) to the /etc/puppet/environments/production/modules/wso2base/files directory.
- v. Add the JDK archive name, Java home and installation directory information in the /etc/puppet/environments/production/modules/wso2base/manifests/java.pp file. Note that these values are treated as the defaults if no other value is specified via the hiera files.

```
$deploymentdir      = '/mnt/jdk-8u112',
$source            = 'jdk-8u112-linux-x64.tar.gz',
$java_home         = '/opt/java',
```

- vi. Add the host entry in the /etc/puppet/hieradata/production/wso2/common.yaml file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

When coping hieradata, note that if you already have common hieradata and WSO2 API Manager specific Hiera data, then you have to merge the Analytics hieradata with common hieradata and API Manager specific hieradata.

Step 3 - Set up the WSO2 API-M Analytics and API-M Puppet Agents

You need to **carry out the following steps separately on each of the six instances** to configure and set up an instance for the Publisher/Store, Traffic Manager, Analytics, Key Manager, Gateway Manager, and the Gateway Worker.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the /etc/hosts file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent /etc/puppet/puppet.conf file by appending the following to the [main] section.

You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the /etc/hosts file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agents

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Set the Facter variables by modifying the deployment.conf file in each of the Puppet Agent instances.

API-M Analytics	Publisher/Store	Key Manager	Gateway Manager	Gateway Worker	Traffic Manager	Format
-----------------	-----------------	-------------	-----------------	----------------	-----------------	--------

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Analytics with pattern 1.

```
product_name=wso2am_analytics
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Publisher/Store with pattern 6.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=publisher-plus-store
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-6
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Key Manager with pattern 6.

```
product_name=wso2is_prepacked
product_version=5.3.0
product_profile=default
environment=production
vm_type=openstack
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Manager with pattern 6.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-6
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Gateway Worker with pattern 6.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=gateway-worker
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-6
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Traffic Manager with pattern 6.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=traffic-manager
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-6
```

```
product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>
```

Step 5 - Update the clustering related configurations

[Click here for more information on updating the relevant clustering related configurations.](#)

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 6**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 1, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 2, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml file.

```
wso2::hosts_mapping:
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
    database
        driver_class_name:
            "%{hiera('wso2::datasources::mysql::driver_class_name')}"
        url:
            jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
        username:
            "%{hiera('wso2::datasources::mysql::username')}"
        password:
            "%{hiera('wso2::datasources::mysql::password')}"
        jndi_config: jdbc/WSO2AM_DB
        max_active:
            "%{hiera('wso2::datasources::common::max_active')}"
        max_wait:
            "%{hiera('wso2::datasources::common::max_wait')}"
        test_on_borrow:
            "%{hiera('wso2::datasources::common::test_on_borrow')}"
        default_auto_commit:
            "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
            "%{hiera('wso2::datasources::mysql::validation_query')}"
        validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes.
Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring the keyStore and client trustStore.](#)

The [WSO2 API-M GitHub repository](#) includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepended` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the Puppet Agents

Carry out the instructions to run the Puppet Agent **separately on each of the six Puppet Agent instances** in the following order.

1. Initially, start the following nodes in given order:
 - a. WSO2 API-M Analytics
 - b. Traffic Manager
 - c. Gateway Manager
 - d. Gateway Worker
2. Thereafter, start the Publisher/Store and Key Manager in any order.

1. Log in to Puppet Agent as a super user.

2. Copy the modified deployment.conf file to the /opt directory.
3. Copy the setup.sh file to the /opt directory and execute the following command.
As the root user, you need to execute this command to add the execute permission to the setup.sh file.

commandsetup.sh

```
chmod 755 setup.sh
```

```
#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

4. Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

When running WSO2 API-M or WSO2 API-M Analytics, <PRODUCT_HOME> corresponds to the following directory.

Node	Installed Directory
WSO2 API-M Analytics	/mnt/wso2am-analytics-2.1.0
WSO2 API-M	/mnt/wso2am-2.1.0
<p>This is applicable to the following nodes. Publisher/Store, Key Manager, Traffic Manager, Analytics, Gateway Manager, and Gateway Worker.</p>	

Using Puppet Modules to Set up WSO2 API-M with Pattern 7

The following diagram illustrates pattern 7 that consists of a stand-alone WSO2 API Manager (WSO2 API-M) setup with a single node deployment. The pattern uses external MySQL databases. The only difference of this pattern from pattern 1 is that this uses WSO2 Identity Sever (WSO2 IS) as the Key Manager.

Please note that the API-M deployment patterns that are listed under the [Deployment Patterns](#) section are newer than the deployment pattern that we have discussed in the following section.



Prerequisites

Hardware	Ensure that the minimum hardware requirements mentioned in the hardware requirements section are met. You can further fine tune your operating system for production by tuning performance .
Software	<ul style="list-style-type: none"> • Oracle JDK 1.8 • Puppet 3.3.8 and above and below 4.0.0 • Debian 6 (or higher) or Ubuntu 12.04 (or higher)

Follow the instructions below to use the Puppet Modules in pattern 7 to deploy WSO2 API-M in a production environment.

This guide assumes that you are familiar with Puppet, which is a configuration automation platform. Puppet uses a client-server model where the managed servers, which are referred to as Puppet Agents, communicate and retrieve configuration profiles from the Puppet Master. If you are not familiar with Puppet, you can follow the [self-paced training](#), which should take about 2-3 days.

The Puppet Modules related instructions have been tested with Puppet 3.3.8 and on the following operating systems: Ubuntu 14.04, RedHat Enterprise Linux 6.7. The following installation instructions are specific to Ubuntu 14.04 LTS.

- Step 1 - Create three instances
- Step 2 - Set up the Puppet Master
 - 2.1 - Install Puppet Master
 - 2.2 - Configure the Puppet Master
- Step 3 - Set up the WSO2 API-M Puppet Agents
 - 3.1 - Install Puppet Agent
 - 3.2 - Configure Puppet Agent
- Step 4 - Set Facter variables for the Puppet Agents
- Step 5 - Update the clustering related configurations
- Step 6 - Configure the keyStore and client trustStore
- Step 7 - Run the Puppet Agents

Step 1 - Create three instances

Create eight vanilla instances, which do not have any configurations. In the subsequent steps you will configure one instance as the Puppet Master and the rest of the instances as Puppet Agents for the following nodes. The server that acts as the Puppet Master controls the configuration information in the WSO2 API-M the deployment, and the managed agent node, which is referred to as the Puppet Agent, requests their configuration catalog from the Puppet Master.

- Key Manager/WSO2 IS

- WSO2 API-M

The latter mentioned instances can be either cloud instances, physical computer instances, or local Virtual Machine (VM) instances. You should be able to SSH in to all the instances.

The instructions on this page is for using the default puppet environment, production. However, if you are using a different puppet environment, you should either specify at the Puppet master side or the Puppet Agent side:

Puppet Master Puppet Agent

Specify environment in the main section of the puppet.conf file in Puppet Master.

```
[main]
environment=<environment-name>
```

Prepend --environment flag to puppet agent run command.

```
puppet agent -vt --environment=<environment-name>
```

Step 2 - Set up the Puppet Master

Carry out the following instructions on any one of the instances, which you created in step 1, to configure and set up the Puppet Master.

2.1 - Install Puppet Master

1. Log in to the server that you are going to set up as the Puppet Master, as a super user.
2. Execute the following commands to install Puppet Master.
 - a. Sync time between the Master and the Agent.

```
ntpdate pool.ntp.org ; apt-get update && sudo apt-get -y install
ntp ; service ntp restart
```

- b. Navigate to the /tmp directory

```
cd /tmp
```

- c. Download and install Puppet distribution.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb &&
dpkg -i puppetlabs-release-trusty.deb && apt-get update &&
apt-get install puppetmaster
```

3. Check the Puppet version to ensure that the Puppet version is 3.3.8 and above and below 4.0.0.

```
puppet -V
```

- Set the hostname property by removing the current entry and adding the following in the `/etc/hostname` file.

```
puppetmaster
```

- Edit your Puppet Master hosts file `/etc/hosts` as follows and set your Puppet Master hostname. The following configuration is added in order to define the IP address that corresponds to the Puppet Master.

```
127.0.0.1 localhost 127.0.0.1 puppetmaster
```

- For production deployments configure Puppet Master with Passenger and Apache

This step is recommended in a production environment. However, it is optional in a testing environment.

2.2 - Configure the Puppet Master

- Log in to the server that you are going to set up as the Puppet Master, as a super user.
- Create the following directory.

```
/etc/puppet/environments/production/
```

- Modify the Puppet Master `/etc/puppet/puppet.conf` file by appending the following to the `[main]` and `[master]` sections respectively.
You need to update this command in order to define where to look for `hiera.yaml` file.

```
[main]
dns_alt_names=puppetmaster,puppet
environmentpath = $confdir/environments
hiera_config = /etc/puppet/hiera.yaml

[master]
autosign=true
```

- `dns_alt_names=puppetmaster,puppet`
This defines the hostnames that the Puppet Master uses to sign the certificates.
- `autosign=true`
This determines that the certificate will be signed automatically.

- Before restarting the Puppet Master, clean all certificates, including the Puppet Master's certificate that has its old Domain Name Service (DNS) alt names.

```
puppet cert clean --all
```

- Restart the Puppet Master for the new configuration changes to take effect and to regenerate the certificate with the new `dns_alt_names`.

```
service puppetmaster restart
```

6. Download the following packages from the respective GitHub pages.

Package	Download location
wso2-puppet-common-1.0.0.zip	https://github.com/wso2/puppet-common
wso2-wso2base-1.0.0.tar.gz	https://github.com/wso2/puppet-wso2base
wso2base-puppet-module-hieradata-1.0.0.zip	https://github.com/wso2/puppet-wso2base
wso2-wso2am_runtime-2.1.0.tar.gz	https://github.com/wso2/puppet-wso2am
wso2am-runtime-puppet-module-hieradata-2.1.0.zip	https://github.com/wso2/puppet-wso2am
wso2-wso2is_prepacked-5.3.0.tar.gz	https://github.com/wso2/puppet-wso2is
wso2is-prepacked-puppet-module-hieradata-5.3.0.zip	https://github.com/wso2/puppet-wso2is

7. Formulate the Puppet Modules.

When using Puppet to configure WSO2 API-M, Puppet Modules are used to install, configure, and start the product.

- Extract the `wso2-wso2base-1.0.0.tar.gz` file, rename the folder name as `wso2base`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Extract the `wso2-wso2am_runtime-2.1.0.tar.gz` file, rename the folder name as `wso2am_runtime`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Extract `wso2-wso2is_prepacked-5.3.0.tar.gz`, rename the folder name as `wso2is_prepacked`, and copy that folder to the `/etc/puppet/environments/production/modules/` directory.
- Optionally, install the [Java module from Puppet Forge](#).

By default there is a Java module built into WSO2 Base Puppet Module. You need to carry out this step only when you need to configure an external Java module.

```
puppet module install 7terminals-java
```

e. Install the stdlib module from Puppet Forge.

You need to install the `stdlib` module, because WSO2 Puppet Modules use some of the functions in the `stdlib` module for data validation purposes.

```
puppet module install puppetlabs-stdlib
```

8. Formulate the Hiera data.

Hiera provides a mechanism to separate the configuration data from Puppet scripts and manage them in a set of YAML files in a hierarchical manner. Therefore, the API-M related configuration data is managed using Hiera. Follow the instructions below to formulate the Hiera data.

- Unzip the `wso2-puppet-common-1.0.0.zip` file and copy the `hiera.yaml` file to the `/etc/puppet` directory.
- Edit the YAML data directory location as follows in the `hiera.yaml` file.

```
:yaml: :datadir: "/etc/puppet/hieradata/%{::environment}"
```

- c. Unzip the `wso2base-puppet-module-hieradata-1.0.0.zip` file and copy the hieradata folder to the `/etc/puppet/` directory.
This folder contains the common Hiera data.
- d. Unzip `wso2am-runtime-puppet-module-hieradata-2.1.0.zip` and copy the hieradata folder to the `/etc/puppet/` directory.
This folder contains WSO2 API Manager specific Hiera data. You need to merge the common Hiera data and API Manager specific Hiera data.
- e. Unzip `wso2am-analytics-puppet-module-hieradata-2.1.0.zip` and copy the hieradata folder to the `/etc/puppet/` directory.
This folder contains WSO2 API Manager Analytics specific Hiera data. You need to merge the common Hiera data and API Manager Analytics specific Hiera data.
- f. Unzip `wso2is-prepackaged-puppet-module-hieradata-5.3.0.zip` and copy the hieradata folder to the `/etc/puppet/` directory.
This folder contains WSO2 Prepackaged IS as Key Manager specific Hiera data. You need to merge the common Hiera data and IS prepackaged specific Hiera data.
- g. Rename the `/etc/puppet/hieradata/dev` directory to the `/etc/puppet/hieradata/production` directory.

If you already having the common hieradata and API Manager specific Hiera data, You have to merge the analytics hieradata with common hieradata and API Manager specific hieradata.

9. Formulate the `site.pp` manifest.
- Puppet programs are referred to as manifests. Manifests are composed using Puppet code and possess a `.pp` extension. The `site.pp` manifest is the default main manifest that Puppet executes first. Follow the instructions below to formulate the `site.pp` manifest:

- a. Copy the `wso2-puppet-common-1.0.0/manifests/site.pp` file to the `/etc/puppet/environments/production/manifests` directory.
- b. Construct the files/packs as follows:
 - i. Copy the WSO2 API-M pack file (`wso2am-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_runtime/files` directory.
 - ii. Copy the WSO2 API-M Analytics pack file (`wso2am-analytics-2.1.0.zip`) to the `/etc/puppet/environments/production/modules/wso2am_analytics/files` directory.
 - iii. Create the following directory - `/etc/puppet/environments/production/modules/wso2base/files`
 - iv. Copy the JDK installation file (e.g., `jdk-8u112-linux-x64.tar.gz`) to the `/etc/puppet/environments/production/modules/wso2base/files` directory.
 - v. Add the JDK version (home) and filename information in the `/etc/puppet/environments/production/modules/wso2base/manifests/java.pp` file.
 - vi. Add the host entry in the `/etc/puppet/hieradata/production/wso2/common.yaml` file as follows:

```
puppetmaster: ip_address: [your_puppet_master_ip_here]
hostname: puppet
```

This is an optional step. You can add any host entry in this manner.

Step 3 - Set up the WSO2 API-M Puppet Agents

You need to **carry out the following steps separately on each of the two instances** to configure and set up an instance for API-M and IS/Key Manager.

The steps involved in setting up a Puppet Agent are identical irrespective of the WSO2 API-M pattern that you are deploying.

3.1 - Install Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Execute the following commands to install Puppet.

```
wget https://apt.puppetlabs.com/puppetlabs-release-trusty.deb && dpkg -i puppetlabs-release-trusty.deb && apt-get update && apt-get install puppet
```

3. Edit the /etc/hosts file that corresponds to your Puppet Agent as follows: The following configuration is added in order to define the Puppet Master's IP address as the Puppet Agent needs to know the Puppet Master's IP address to be able to communicate with it.

```
127.0.0.1 localhost
192.168.19.XXX puppet # Puppet Master's IP address
```

3.2 - Configure Puppet Agent

1. Log in to the server that you are going to set up as the Puppet Agent as a super user.
2. Modify the Puppet Agent /etc/puppet/puppet.conf file by appending the following to the [main] section

You need to do this to add the hostname of the Puppet Master. This hostname is mapped to the IP address in the /etc/hosts file. The Puppet Agent uses this information to discover the Puppet Master's hostname in order to connect to the Puppet Master.

```
[main]
server = puppet
```

Step 4 - Set Facter variables for the Puppet Agents

Facter refers to Puppet's cross-platform system profiling library. It discovers and reports node specific information, which is available in your Puppet manifests in the form of variables. The Puppet Agent uses Facter to send its node information to the Puppet Master. Follow the instructions below to set the Facter variables.

1. Set the Facter variables by modifying the deployment.conf file in each of the Puppet Agent instances.

Pattern-1 IS/Key Manager Pattern-7 API-M Format

The following are the configurations that you need when running the Puppet Agent node for WSO2 IS/Key Manager with pattern 1.

```
product_name=wso2is_prepacked
product_version=5.3.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-1
```

The following are the configurations that you need when running the Puppet Agent node for WSO2 API-M Publisher with pattern 7.

```
product_name=wso2am_runtime
product_version=2.1.0
product_profile=default
vm_type=openstack
environment=production
platform=default
use_hieradata=true
pattern=pattern-7
```

```
product_name=<product_name>_runtime
product_version=<product_version>
product_profile=<hiera_file_name_without_extension>
environment=production
vm_type=openstack
use_hieradata=true
platform=default
pattern=<pattern>
```

Step 5 - Update the clustering related configurations

▼ [Click here for more information on updating the relevant clustering related configurations.](#)

The following section contains clustering information for all the patterns. Therefore, make sure to **only update the configurations based on pattern 7**.

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

1. Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

[Pattern 0](#)[Pattern 1](#)[Pattern 2](#)[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)[Pattern 7](#)

If you are using pattern 0, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/ default.yaml` file.

```
wso2::hosts_mapping:  
  api_manager:  
    ip: 127.0.0.1  
    name: am.dev.wso2.org
```

If you are using pattern 1, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml` file.

```
wso2::hosts_mapping:  
  api_manager:  
    ip: 127.0.0.1  
    name: am.dev.wso2.org
```

If you are using pattern 2, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml` file.

```
wso2::hosts_mapping:  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  api_manager:  
    ip: 127.0.0.1  
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/ common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher-Store cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway

Worker node in the ENV2.

```
wka:
members:
-
  hostname: 192.168.57.5
  port: 4000
-
  hostname: 192.168.57.218
  port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
members:
-
  hostname: 192.168.57.219
  port: 4000
-
  hostname: 192.168.57.21
  port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP

addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
members:
-
  hostname: 192.168.57.216
  port: 4000
-
  hostname: 192.168.57.247
  port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
    name: WSO2_AM_DB
    description: The datasource used for API Manager
    database
        driver_class_name:
            "%{hiera('wso2::datasources::mysql::driver_class_name')}"
        url:
            jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
        username:
            "%{hiera('wso2::datasources::mysql::username')}"
        password:
            "%{hiera('wso2::datasources::mysql::password')}"
        jndi_config: jdbc/WSO2AM_DB
        max_active:
            "%{hiera('wso2::datasources::common::max_active')}"
        max_wait:
            "%{hiera('wso2::datasources::common::max_wait')}"
        test_on_borrow:
            "%{hiera('wso2::datasources::common::test_on_borrow')}"
        default_auto_commit:
            "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
            "%{hiera('wso2::datasources::mysql::validation_query')}"
        validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used fpr managing APIs
configdb	database used for config registry
govregdb	database userd for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocesseddatastoredb** which are used for Analytics recode store.

- i. You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store/mysql-mb.sql

You do not need to run the database scripts agains following databases as the database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddatastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (mysql-connector-java-5.1.39-bin.jar) to the locations /etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib and /etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib in Puppet Master.
- ii. Uncomment the file_list entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not mysql-connector-java-5

.1.39-bin.jar, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:  
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes.
Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

B a s e d

Make the respective SVN based configurations in the following files based on the pattern that you are using.

Pattern	Files to Update
Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worke
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worke
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.ya
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manag /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worke

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
 - i. Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - ii. Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - iii. Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1.jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215.jar"
```

Step 6 - Configure the keyStore and client trustStore

[Click here for more information on configuring keyStore and client trustStores.](#)

The [WSO2 API-M GitHub repository](#) includes a custom keyStore and client trustStore in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The same files are copied into the `wso2am_analytics` module and `wso2is_prepended` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[FormatExample](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048
-keystore wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[FormatExample](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[FormatExample](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Step 7 - Run the Puppet Agents

Carry out the instructions to run the Puppet Agent **separately on both the Puppet Agent instances** in the following order.

1. Key Manager
2. WSO2 API-M

1. Log in to Puppet Agent as a super user.
2. Copy the modified deployment.conf file to the /opt directory.
3. Copy the setup.sh file to the /opt directory and execute the following command.
As the root user, you need to execute this command to add the execute permission to the setup.sh file.

```
commandsetup.sh
```

```
chmod 755 setup.sh

#!/bin/bash
echo "#####
echo " Starting cleanup "
echo "#####
ps aux | grep -i wso2 | awk '{print $2}' | xargs kill -9
echo "#####
echo " Setting up environment "
echo "#####
mkdir -p /etc/facter/facts.d
cp deployment.conf /etc/facter/facts.d/deployment_pattern.txt
echo "#####
echo " Installing "
echo "#####
puppet agent --enable
puppet agent -vt
puppet agent --disable
```

- Run the following command to set up and install the Puppet Agent.

```
./setup.sh
```

When running WSO2 API-M or WSO2 IS as the Key Manager, <PRODUCT_HOME> corresponds to the following directory.

Node	Installed Directory
WSO2 IS as the Key Manager	/mnt/wso2is-km-5.3.0
WSO2 API-M	/mnt/wso2am-2.1.0

Configuring Clustering for a Selected Pattern

Hiera data sets that match the distributed profiles of WSO2 API Manager (api-store, api-publisher, api-key-manager, gateway-manager, gateway-worker, traffic-manager) are shipped with clustering related configuration that is already enabled. Therefore, you need to only do a few changes to setup a distributed deployment in your preferred deployment pattern, before running the Puppet Agent.

- Add or update the host name mapping list. Puppet adds the required host entries explicitly in the /etc/hosts file in the Puppet Agent. For this purpose you have to update the hosts mappings appropriately in the respective Hiera data file (YAML file) based on the pattern that you are using.

Pattern 0 Pattern 1 Pattern 2 Pattern 3 Pattern 4 Pattern 5 Pattern 6 Pattern 7

If you are using pattern 0, update the following section in the /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-0/default.yaml file.

```
wso2::hosts_mapping:  
  api_manager:  
    ip: 127.0.0.1  
    name: am.dev.wso2.org
```

If you are using pattern 1, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-1/ default.yaml` file.

```
wso2::hosts_mapping:  
  api_manager:  
    ip: 127.0.0.1  
    name: am.dev.wso2.org
```

If you are using pattern 2, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-2/ default.yaml` file.

```
wso2::hosts_mapping:  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  api_manager:  
    ip: 127.0.0.1  
    name: am.dev.wso2.org
```

If you are using pattern 3, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org
```

If you are using pattern 4, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/ common.yaml` file.

```
wso2::hosts_mapping:  
  apim_keymanager:  
    ip: 192.168.57.186  
    name: km.dev.wso2.org  
  apim_store:  
    ip: 192.168.57.21  
    name: store.dev.wso2.org  
  apim_publisher:  
    ip: 192.168.57.219  
    name: pub.dev.wso2.org  
  apim_gateway_manager:  
    ip: 192.168.57.216  
    name: mgt-gw.dev.wso2.org  
  apim_gateway_worker:  
    ip: 192.168.57.247  
    name: gw.dev.wso2.org  
  apim_traffic_manager:  
    ip: 192.168.57.35  
    name: tm.dev.wso2.org  
  svn:  
    ip: 192.168.100.1  
    name: svn.gw.am.dev.wso2.org  
  apim_analytics_server:  
    ip: 192.168.57.29  
    name: analytics.dev.wso2.org  
  apim_gateway_manager_dmz:  
    ip: 192.168.57.5  
    name: dmz-mgt-gw.dev.wso2.org  
  apim_gateway_worker_dmz:  
    ip: 192.168.57.218  
    name: dmz-gw.dev.wso2.org
```

If you are using pattern 5, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtimes/pattern-5/common.yaml` file.

```
wso2::hosts_mapping:
  apim_store:
    ip: 192.168.57.21
    name: store.dev.wso2.org
  apim_publisher:
    ip: 192.168.57.219
    name: pub.dev.wso2.org
  apim_gateway_manager:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gw_plus_km:
    ip: 192.168.57.247
    name: am.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 6, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_r` `untime/pattern-6/` `common.yaml` file.

```
wso2::hosts_mapping:
  apim_keymanager:
    ip: 192.168.57.186
    name: km.dev.wso2.org
  apim_publisher_plus_store:
    ip: 192.168.57.21
    name: am.dev.wso2.org
  apim_gateway:
    ip: 192.168.57.216
    name: mgt-gw.dev.wso2.org
  apim_gateway_worker:
    ip: 192.168.57.247
    name: gw.dev.wso2.org
  apim_traffic_manager:
    ip: 192.168.57.35
    name: tm.dev.wso2.org
  svn:
    ip: 192.168.100.1
    name: svn.gw.am.dev.wso2.org
  apim_analytics_server:
    ip: 192.168.57.29
    name: analytics.dev.wso2.org
```

If you are using pattern 7, update the following section in the `/etc/puppet/hieradata/wso2/wso2am_runtimes/pattern-7/default.yaml` file.

```
wso2::hosts_mapping:
  api_manager:
    ip: 127.0.0.1
    name: am.dev.wso2.org
  identity_server:
    ip: 192.168.57.35
    name: is.dev.wso2.org
```

2. Add the Well Known Address (WKA) list for Gateway clusters and Publisher-Store cluster. The required configurations for clustering are already added, but you need to update the WKA IP addresses in the respective Hiera data files based on your deployment.

This is **not applicable** to patterns 0, 1, 2, and 7 as those patterns do not have Gateway or Publisher-Store clusters.

[Pattern 3](#)[Pattern 4](#)[Pattern 5](#)[Pattern 6](#)

There are 2 clusters in the pattern-3 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of Gateway Manager node and the Gateway Worker node.

```
wka:
  members:
    -
      hostname: 192.168.57.216
      port: 4000
    -
      hostname: 192.168.57.247
      port: 4000
```

There are 3 clusters in the pattern-4 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway Cluster

There are 2 Gateway clusters in this pattern. One is in the ENV1 and the other one is in the ENV2. Each of those clusters consist of a Gateway Manager node and a Gateway Worker node.

- Configure the Gateway cluster in the ENV1
Update the WKA list in both the `gateway-manager-env1.yaml` and `gateway-worker-env1.yaml` files with the IP addresses of Gateway Manager node and Gateway Worker node in the ENV1.

```
wka:
  members:
    -
      hostname: 192.168.57.216
      port: 4000
    -
      hostname: 192.168.57.247
      port: 4000
```

- Configure the Gateway cluster in the ENV2
Update the WKA list in both the `gateway-manager-env2.yaml` and `gateway-worker-env2.yaml` files with the IP addresses of the Gateway Manager node and the Gateway Worker

node in the ENV2.

```
wka:
  members:
    -
      hostname: 192.168.57.5
      port: 4000
    -
      hostname: 192.168.57.218
      port: 4000
```

There are 2 clusters in the pattern-5 deployment. Update the Well Known Address (WKA) list for the clusters as follows:

a. Publisher - Store

cluster

This is a cluster of the Publisher node and the Store node. Update the WKA list in both the `api-publisher.yaml` and the `store.yaml` files with the IP addresses of Publisher and Store nodes.

```
wka:
  members:
    -
      hostname: 192.168.57.219
      port: 4000
    -
      hostname: 192.168.57.21
      port: 4000
```

b. Gateway cluster

This is a cluster for the Gateway Manager node and the node that has the Gateway Worker together with the Key Manager. Update the WKA list in both the `gateway-manager.yaml` and `gw-plus-km.yaml` files with the IP addresses of Gateway Manager node and the node that has the Gateway Worker together with the Key Manager.

```
wka:
  members:
    -
      hostname: 192.168.57.216
      port: 4000
    -
      hostname: 192.168.57.247
      port: 4000
```

There is one cluster in the pattern-6 deployment. Update the Well Known Address (WKA) list for the cluster as follows:

The Gateway cluster is a cluster for the Gateway Manager node and the Gateway Worker node. Update the WKA list in both the `gateway-manager.yaml` and `gateway-worker.yaml` files with the IP addresses of

Gateway Manager node and the Gateway Worker node.

```
wka:
  members:
    -
      hostname: 192.168.57.216
      port: 4000
    -
      hostname: 192.168.57.247
      port: 4000
```

3. Modify all the MySQL based datasources in the respective Hiera data file (YAML file) that corresponds to the pattern you are using in order to point to the external MySQL servers.

This is **not applicable** for pattern 0 as pattern-0 uses a H2 database.

Update the following in the `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/ default.yaml` file for patterns 1, 2 and 7 and `/etc/puppet/hieradata/wso2/wso2am_runtime/[pattern-number]/ common.yaml` file for patterns 3 to 6.

- a. You need to only replace the IP address, with the IP address of database server that you are using.

If you want to use any other database except MySQL, update the data sources appropriately.

Example:

```
wso2_am_db:
  name: WSO2_AM_DB
  description: The datasource used for API Manager database
  driver_class_name:
    "%{hiera('wso2::datasources::mysql::driver_class_name')}"
  url:
    jdbc:mysql://192.168.57.210:3306/apimgtdb?autoReconnect=true
  username: "%{hiera('wso2::datasources::mysql::username')}"
  password: "%{hiera('wso2::datasources::mysql::password')}"
  jndi_config: jdbc/WSO2AM_DB
  max_active:
    "%{hiera('wso2::datasources::common::max_active')}"
    max_wait: "%{hiera('wso2::datasources::common::max_wait')}"
    test_on_borrow:
      "%{hiera('wso2::datasources::common::test_on_borrow')}"
      default_auto_commit:
        "%{hiera('wso2::datasources::common::default_auto_commit')}"
        validation_query:
          "%{hiera('wso2::datasources::mysql::validation_query')}"
          validation_interval:
            "%{hiera('wso2::datasources::common::validation_interval')}"
```

- b. Create following database scemas in MySQL server. The databases need to be created in each of the pattern is listed below.

Names of the databases that need to be created	description
apimgtdb	database used for managing APIs
configdb	database used for config registry
govregdb	database used for gov registry
userdb	database used for user management and user store
mbstoredb	database used for message broker
statdb	database used for getting statistics for API Manager

Addition to that if you are using **pattern-6**, create **analyticseventstoredb** and **analyticprocessdatastoredb** which are used for Analytics recode store.

- You need to run mysql database scripts for following tables as mentioned below.

database	script
apimgtdb	<API-M_HOME>/dbscripts/apimgt/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
configdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
govregdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
userdb	<API-M_HOME>/dbscripts/mysql.sql If you are using MySQL 5.7 or later version, select mysql5.7.sql located in same directory without using mysql.sql script.
mbstoredb	<API-M_HOME>/dbscripts(mb-store)/mysql-mb.sql

You do not need to run the database scripts agains following databases as the

database tables of them will be created at runtime.

- **statdb**
- **analyticseventstoredb**
- **analyticprocesseddastoredb**

c. If you are using a MySQL database make the following changes:

- i. Download and copy the mysql driver jar (`mysql-connector-java-5.1.39-bin.jar`) to the locations `/etc/puppet/environments/production/modules/wso2am_runtime/files/configs/repository/components/lib` and `/etc/puppet/environments/production/modules/wso2am_analytics/files/configs/repository/components/lib` in Puppet Master.
- ii. Uncomment the `file_list` entry for JDBC connector JAR in the relevant hiera data files.

```
wso2::file_list:
-
"repository/components/lib/%{hiera('wso2::datasources::mysql::connector_jar')}"
```

- iii. Update the JAR file name appropriately if your file name is not `mysql-connector-java-5.1.39-bin.jar`, which is set as the default value.

```
wso2::datasources::mysql::connector_jar:
mysql-connector-java-5.1.39-bin.jar
```

4. Configure deployment synchronization in each of the Gateway related nodes. Use one of the following ways to carryout this configuration.

- **Rsync**

WSO2 recommends rsync instead of SVN, for deployment synchronization as it is an efficient, easy to use and lightweight solution compared to SVN.

Why can't I use SVN based deployment synchronization (Dep Sync)?

WSO2 has identified some inconsistencies when using Hazelcast clustering. As a result, from API-M 2.1.0 onwards WSO2 API-M has been designed so that it is possible to deploy API-M in a clustered setup without using Hazelcast clustering, so that users can use Hazelcast clustering only when absolutely necessary. However, if you use deployment synchronization as a content synchronization mechanism, you are compelled to use Hazelcast clustering. Therefore, WSO2 does not recommend using SVN based deployment synchronization.

If you prefer to use rsync, see [Configuring rsync for Deployment Synchronization](#).

- **S V N**

Make the respective SVN based configurations in the following files based on the pattern that you are using.

B a s e d

Pattern	Files to Update
---------	-----------------

Pattern 3	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-manager /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-3/gateway-worker.yaml
Pattern 4	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manager /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worker.yaml /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-manager /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-4/gateway-worker.yaml
Pattern 5	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gateway-manager /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-5/gw-plus-km.yaml
Pattern 6	/etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-manager /etc/puppet/hieradata/wso2/wso2am_runtime/pattern-6/gateway-worker.yaml

- a. Uncomment the SVN based deployment synchronization configurations in the following files based on the pattern and update the configurations where required. Patterns 3 to 6 are configured for SVN based deployment synchronization; however, the configurations are commented out by default.

Example

```
wso2::dep_sync:
  enabled: true
  auto_checkout: true
  auto_commit: true
  repository_type: svn
  svn:
    url: http://svnrepo.example.com/repos/
    user: username
    password: password
    append_tenant_id: true
```

- b. Copy the required JARs for SVN, into the respective locations.
- Copy the `svnkit-all-1.8.7.wso2v1.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/dropins` directory.
 - Copy the `trilead-ssh2-1.0.0-build215.jar` into the `<PUPPET_HOME>/modules/wso2am/files/configs/repository/components/lib` directory.
 - Uncomment the `file_list` entries for the latter mentioned two JAR files in the respective Hiera data files related to gateway nodes.

```
wso2::file_list:
  -
  "repository/components/dropins/svnkit-all-1.8.7.wso2v1
  .jar"
  -
  "repository/components/lib/trilead-ssh2-1.0.0-build215
  .jar"
```

Configuring the keyStore and client trustStore for a Selected Pattern

The [WSO2 API-M GitHub repository](#) includes a custom `keyStore` and `client trustStore` in the `puppet-apim/wso2am/files/configs/repository/resources/security` directory for the initial setup (i.e., testing) purpose. The

same files are copied into the `wso2am_analytics` module and `wso2is_prepacked` module as well. This `wso2carbon.jks` keyStore is created for `CN=*.dev.wso2.org`, and its self-signed certificate is imported into the `client-truststore.jks`. When running the Puppet Agent, these two files replace the existing default `wso2carbon.jks` and `client-truststore.jks` files.

In production environments, it is recommended to replace these with your own keyStores and trustStores with certification authority (CA) signed certificates. In addition, if you also change the hostnames given by default in these patterns, you have to create your own hostnames. For more information, see [Creating New Keystores](#).

Follow the steps below to create a new keystore and client-truststore with self-signed certificates.

1. Generate a Java keyStore and key pair with a self-signed certificate.

[Format](#)[Example](#)

```
keytool -genkey -alias <alias-name> -keyalg RSA -keysize 2048
-keystore <keystore-name> -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
<password> -keypass <password> -validity <validity-period>
```

```
keytool -genkey -alias wso2carbon -keyalg RSA -keysize 2048 -keystore
wso2carbon.jks -dname
"CN=*.dev.wso2.org,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass
wso2carbon -keypass wso2carbon -validity 2000
```

2. Export the certificate from the latter mentioned keyStore.

[Format](#)[Example](#)

```
keytool -export -keystore <keystore-name> -alias
<alias-of-the-certificate> -file <output-file-name>.cer
```

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2carbon.cer
```

3. Import the latter mentioned certificate into a trustStore.

[Format](#)[Example](#)

```
keytool -import -alias <alias-of-the-certificate> -file
<input-file-name>.cer -keystore <client-truststore-name> -storepass
<trust-store-password>
```

```
keytool -import -alias wso2carbon -file wso2carbon.cer -keystore
client-truststore.jks -storepass wso2carbon
```

Distributed Deployment of API Manager

WSO2 API Manager (WSO2 API-M) is a complete API management solution, used for creating and publishing APIs, creating and managing a developer community, and scalably routing API traffic. The WSO2 API-M solution includes a Publisher, Store, Gateway, Key Manager, and Traffic Manager component.

Typically, when you get started with API Manager in a development environment, you deploy WSO2 API Manager as a single, all-in-one instance with all its components on a single server. For details, see [Deploying API Manager using Single Node Instances](#).

However, in a production deployment, these components are deployed in a distributed manner. Therefore, you can create a distributed deployment of WSO2 API-M's five main components: Publisher, Store, Gateway, Key Manager, and Traffic Manager. This page describes how to set up and deploy WSO2 API-M as a distributed deployment.

- Understanding the WSO2 API-M architecture
- Understanding the distributed deployment
- Deploying WSO2 API-M in a distributed setup

Note that your configurations may vary depending on the [API Manager clustering deployment pattern](#) you choose. If you are using multi-tenancy, all nodes should use the same user store, as all servers are servicing the same set of tenants, and it has to share the same Governance Registry space across all nodes.

Understanding the WSO2 API-M architecture

WSO2 API Manager uses the following main components:

Publisher	Enables API providers to easily publish their APIs, share documentation, provision API keys, and gather feedback on API features, quality, and usage.
Store	Enables consumers to self-register, discover API functionality, subscribe to APIs, evaluate them, and interact with API publishers.
Key Manager	Responsible for all security and key-related operations.
Gateway	Responsible for securing, protecting, managing, and scaling API calls.
Traffic Manager	Used to make a decision on throttling.

For more information on the above, see the [main components of a distributed system](#).

Additionally, API Manager uses the following databases, which are shared among the server nodes.

- **User Manager database** - Stores information related to users and user roles. This information is shared among the Key Manager Server, Store, and Publisher. Users can access the Publisher for API creation and the Store for consuming the APIs.
- **API Manager database** - Stores information related to the APIs along with the API subscription details. The Key Manager Server uses this database to store user access tokens that are used for verification of API calls.
- **Registry database** - Shares information between the Publisher and Store. When an API is published through the Publisher, it is made available in the Store via the sharing registry database. Although you would normally share information between the Publisher and Store components only, if you are planning to create this setup for a multi-tenanted environment (create and work with tenants), it is required to share the information in this database between the Gateway and Key Manager components as well.
- **Statistics database** - Stores information related to API statistics. After you [configure API-M analytics](#), it writes summarized data to this database. The Publisher and Store can then query this database to display

- the statistics data.
- **Message Broker database** - Traffic Manager uses this database as the message store for broker when [advanced throttling](#) is used.

The API Manager components use the databases as follows:

	(API Manager Database) apimgtdb	(User Manager Database) userdb	(Registry Database) regdb	(Statistics Database) statdb	(Message Broker Database) mbstoredb
Publisher	Used	Used	Used	Used	Not used
Store	Used	Used	Used	Used	Not used
Key Manager	Used	Used	Used (in multi-tenancy mode)	Not used	Not used
Gateway	Not used	Used (in multi-tenancy mode)	Used (In multi-tenancy mode/In multiple Gateway mode when Google Analytics is used)	Not used	Not used
Traffic Manager	Not used	Not used	Not used	Not used	Used

Notes

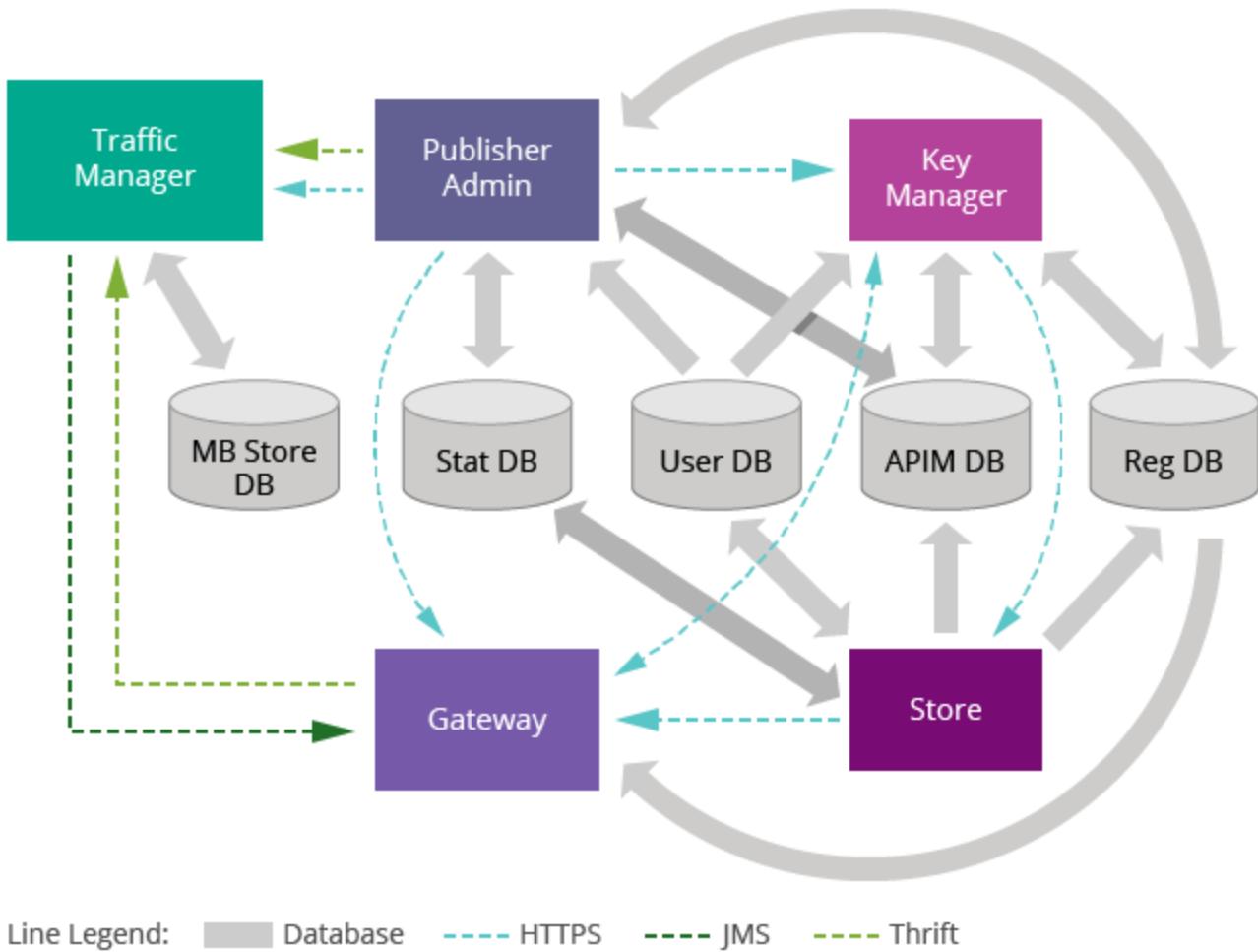
- Although the Gateway does not use the API Manager database, this connection is required; therefore, do not remove the default configuration in the <API-M_HOME>/repository/conf/data/sources/master-datasources.xml file.
- If you have more than one Traffic Manager node, each Traffic Manager node must have its own WSO2 _MB_STORE_DB database.

When we consider a distributed deployment of WSO2 API Manager, we have the option of separating the five components and clustering each component as needed. Let's look more closely at how the API Manager components are deployed separately.

Understanding the distributed deployment

In the following diagram, the five components are set up in a distributed deployment, and the three databases are connected to the relevant components respectively. The entire setup is also fronted by a load balancer. Notice the port offset and the HTTPS port of each WSO2 server component.

In a clustered setup, if the **Key Manager** is **NOT fronted by a load balancer**, you have to set the KeyValidatorClientType element to ThriftClient in the <API-M_HOME>/repository/conf/api-manager.xml file, to enable Thrift as the communication protocol. You need to configure this in all the Gateway and Key Manager components.



Deploying WSO2 API-M in a distributed setup

Follow the instructions below to deploy WSO2 API-M in the above mentioned distributed environment:

- Step 1 - Install and configure WSO2 API Manager
- Step 2 - Install and configure the databases
- Step 3 - Configure the connections among the components
- Step 4 - Start the cluster

Step 1 - Install and configure WSO2 API Manager

The following steps describe how to download, install, and configure API Manager. We will create five instances of API Manager for this.

1. Download the [WSO2 API Manager](#) in each of the five servers in the cluster for distributed deployment.
2. Unzip the API Manager zipped archive, and rename each of those directories respectively as Key Manager, Gateway, Publisher, Store, and Traffic Manager. These five directories are located in a server of their own and will be used for each component of the API Manager. Each of these unzipped directories are referred to as `<API-M_HOME>` or `<PRODUCT_HOME>` in this document.
3. Only do this step if you wish to run WSO2 servers (TrafficManager, KeyManager, Gateway, Publisher, and Store) on the same machine. You need to set the `Offset` attribute to avoid any port usage conflict, as follows:
 - a. In each of the created directories, open the `<PRODUCT_HOME>/repository/conf/carbon.xml` file
 - b. Edit the `<offset>` attribute in each file as shown in the Port Offset column in the following table. The port value will automatically be increased as shown in the Port Value column, allowing all five WSO2

server instances to run on the same machine.

WSO2 Server instance	Port Offset	Port Value
Key Manager	0	9443
Gateway	1	9444
Publisher	2	9445
Store	3	9446
Traffic Manager	4	9447

This step is optional and only required if all server instances are running in the same machine. This is not a recommended approach for production environments. Note that you need to change all ports used in your configurations based on the offset value if you are to do this. For more information, see [Changing the Default Ports with Offset](#). This document does NOT use port offsets within the configurations.

4. In each of the five servers, replace the default certificates (where CN=localhost) with new certificates generated with proper CN values, in order to avoid seeing an error stating that the hostname in the certificate didn't match.

Note that, you should use the same primary key store for all API Manager instances here in order to decrypt registry resources. Refer [Configuring Primary Keystores](#) documentation for more information. When creating the keystore, always use a longer validity period so that it will avoid the need of migration on the registry data when shifting to a new keystore.

5. Change the Hostname and the Management Hostnames in each of the five nodes by following the steps below:

- a. Open the <API-M_HOME>/repository/conf/carbon.xml file
- b. Find the commented out parameters **HostName** and **MgtHostName**, uncomment and modify the values. MgtHostName is the hostname of the management console, while HostName is used for the backend services. For an example:

```
<HostName>apim.wso2.com</HostName>
...
<MgtHostName>mgt.apim.wso2.com</MgtHostName>
```

6. Make sure to keep the following required web apps and remove the unnecessary web apps from each node from the <API-M_HOME>/repository/deployment/server/webapps directory.

This step is mandatory for the Traffic Manager node, but is optional for all other nodes.

The following are the **web apps required for each node**:

Profile	Required web apps

Gateway Manager	am#sample#pizzashack#v1 (only needed if the Pizzashack sample API is used) api#am#admin#v0.11 (only needed if you want to perform administrative tasks through Gateway Manager) authenticationendpoint
Gateway Worker	am#sample#pizzashack#v1 (only needed if the Pizzashack sample API is used) api#am#admin#v0.11 (only needed if you want to perform administrative tasks through Gateway Manager)
Traffic Manager	shindig (shindig web app is used for the WSO2 CEP Dashboard)
Key Manager	authenticationendpoint client-registration#v0.11 oauth2 throttle#data#v1
API Publisher	am#sample#pizzashack#v1 (only needed if the Pizzashack sample API is used) api#am#publisher#v0.11 authenticationendpoint
API Store (Developer Portal)	api#am#store#v0.11 authenticationendpoint

Step 2 - Install and configure the databases

The following steps describe how to download and install MySQL Server, create the databases, configure the data sources, and configure the API Manager components to connect to them. Additionally, you may need to [configure your API Analytics platform with API Manager](#) in the distributed setup.

1. Download and install MySQL Server.
2. Download the MySQL JDBC driver.
3. Unzip the downloaded MySQL driver zipped archive, and copy the MySQL JDBC driver JAR (`mysql-connector-java-x.x.xx-bin.jar`) into the `<PRODUCT_HOME>/repository/components/lib` directory in all the nodes in the cluster.
4. Enter the following command in a terminal/command window, where `username` is the username you want to use to access the databases:

```
mysql -u username -p
```

5. When prompted, specify the password that will be used to access the databases with the username you specified.
6. Create the databases using the following commands, where `<API-M_HOME>` is the path to any of the API Manager instances you installed, and `username` and `password` are the same as those you specified in the previous steps.

About using MySQL in different operating systems

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 following is how your database creation command should look.

```
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 following is how your database creation command should look.

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

If you are using MySQL to configure your datasources, we recommend to change your Database collation to a **case sensitive** one. Refer [MySQL Official Manual](#) for more information. Because when the DataBase or Table is having a case-insensitive collation in MySQL 5.6 or 5.7, (default collation(**latin1_swedish_ci**) is case insensitive) if a user creates an API with mixed case letters, delete the API and, then create another API with same name but in lower case letters, The later created one will lose the permission information because when deleting an API, It keeps the Registry Collection left behind.

This issue could be avoided if you use a **case sensitive** collation for database and tables. In that case, when creating the second API(which is having the same name but in all lowercase letters), it will create a new record with the lowercase name in the UM_PERMISSION table.

Additional notes

- Ensure that MySQL is configured so that all nodes can connect to it.
- If you are using MySQL version 5.7 or later, use the `mysql5.7.sql` script instead of the `mysql.sql` script.
- Table creation of the statistics database is handled by the Analytics scripts when you [configure APIM Analytics](#), so you will create the statistics database in this step but will not specify a source script.

```

mysql> create database apimgtdb;
mysql> use apimgtdb;
mysql> source <API-M_HOME>/dbscripts/apimgt/mysql.sql;
mysql> grant all on apimgtdb.* TO username@localhost identified by
'password';

mysql> create database userdb;
mysql> use userdb;
mysql> source <API-M_HOME>/dbscripts/mysql.sql;
mysql> grant all on userdb.* TO username@localhost identified by
'password';

mysql> create database regdb;
mysql> use regdb;
mysql> source <API-M_HOME>/dbscripts/mysql.sql;
mysql> grant all on regdb.* TO username@localhost identified by
'password';

mysql> create database statdb;
mysql> grant all on statdb.* TO username@localhost identified by
'password';

mysql> create database mbstoredb;
mysql> use mbstoredb;
mysql> source <API-M_HOME>/dbscripts(mb-store/mysql-mb.sql;
mysql> grant all on mbstoredb.* TO username@localhost identified by
'password';

```

7. Configure the data sources for the five databases as follows:

In this guide we are using MySQL to configure the datasources. You can change the datasource according to the database that you use by following [Changing the Default API-M Databases](#).

- Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file in all five WSO2 API Manager components.

Note: When configuring clustering, ignore the `WSO2_CARBON_DB` data source configuration.
- Enable the components to access the WSO2 API Manager database by modifying the `wso2am_db` data source in the `master-datasources.xml` files in the Publisher, Store, and Key Manager nodes by changing the URL as indicated below, replacing `apimgtdb.mysql-wso2.com` with the hostname you specified in step 4 (`carbondb.mysql-wso2.com`).

```
<datasource>
    <name>WSO2AM_DB</name>
    <description>The datasource used for the API Manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:mysql://apimgtdb.mysql-wso2.com:3306/apimgtdb?autoReconnect=true</url>
            <username>user</username>
            <password>password</password>
            <defaultAutoCommit>false</defaultAutoCommit>
            <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>
```

- c. Enable the Key Manager, Publisher, and Store components to access the user management database by configuring the **wso2um_DB** data source in their `master-datasources.xml` files as follows:

```

<datasource>
    <name>WSO2UM_DB</name>
    <description>The datasource used by user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2UM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:mysql://userdb.mysql-wso2.com:3306/userdb?autoReconnect=true</url>
            <username>user</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>

```

- d. Enable the Publisher and Store components to access the registry and statistics databases by configuring the **WSO2REG_DB** and **WSO2AM_STATS_DB** data sources in their `master-datasources.xml` files as follows:

```

<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used by the registry</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:mysql://regdb.mysql-wso2.com:3306/regdb?autoReconnect=true</url>
            <username>user</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>

<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://statdb.mysql-wso2.com:3306/statdb?autoReconnect=true</url>
            <username>user</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>

```

- e. Enable the Traffic Manager component to access the Message Broker database by configuring the `ws02_MB_STORE_DB` data source in its `master-datasources.xml` file as follows:

```

<datasource>
    <name>WSO2_MB_STORE_DB</name>
    <description>The datasource used for message broker
database</description>
    <jndiConfig>
        <name>WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:mysql://mbstoredb.mysql-wso2.com:3306/mbstoredb?autoRe
connect=true</url>
            <username>user</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>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

- f. If you created the databases on other servers instead of your local machine, map the IP addresses to the data source URLs by opening the /etc/hosts file, and adding an entry for each remote database. The following is an example of this.
 - 127.0.0.1 apimgtdb.mysql-wso2.com
 - 127.0.0.2 userdb.mysql-wso2.com
 - 127.0.0.3 regdb.mysql-wso2.com
 - 127.0.0.4 statdb.mysql-wso2.com
 - 127.0.0.5 mbstoredb.mysql-wso2.com
8. To give the Key Manager, Publisher, and Store components access to the user management database with shared permissions, open the <API-M_HOME>/repository/conf/user-mgt.xml file in each of these three components and add or modify the dataSource property of the <configuration> element as follows:

```

<configuration>
...
<Property name="dataSource">jdbc/WSO2UM_DB</Property>
</configuration>

<UserStoreManager
class="org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager">
<Property
name="TenantManager">org.wso2.carbon.user.core.tenant.JDBCTenantManag
er</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="ReadGroups">true</Property>
<Property name="WriteGroups">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>
<Property name="MaxRoleNameListLength">100</Property>
<Property name="MaxUserNameListLength">100</Property>
<Property name="SharedGroupEnabled">false</Property>
<Property name="SCIMEnabled">false</Property>
</UserStoreManager>

```

9. To give the Publisher and Store components access to the registry database, open the <API-M_HOME>/repository/conf/registry.xml file in each of these two components and configure them as follows.

Although it is mentioned that you need to do this on the Publisher and Store components only, if you are planning to create this setup for a multi-tenanted environment (create and work with tenants), it is required to perform the steps below on the Gateway and Key-Manager components as well.

Note: Do not replace the following configuration when adding in the mounting configurations mentioned below. The registry mounting configurations mentioned in the following steps must be added beneath the following entry, which is already in the configuration file.

```
<dbConfig name="wso2registry">
  <dataSource>jdbc/WSO2CarbonDB</dataSource>
</dbConfig>
```

This configuration points to the local H2 database. This configuration is necessary and must always exist in this file.

- In the Publisher component's `registry.xml` file, add or modify the `dataSource` attribute of the `<dbConfig name="govregistry">` element as follows:

```
<dbConfig name="govregistry">
  <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>
<remoteInstance url="https://localhost">
  <id>gov</id>

  <cacheId>user@jdbc:mysql://regdb.mysql-wso2.com:3306/regdb</cacheId>
  <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>
```

`CacheId` is a unique identification of remote instance. When configuring the remote instance, it is recommended to modify `<cacheId>` with the corresponding values of your setup as in this format.

`<username>@<JDBC_URL_to_registry_database>`

Note that you do not need to specify the `remoteInstance URL` in above configuration since it is not used because we are not using WS mounting in latest API Manager versions including this version.

We have pointed the **governance registry** and **config registry** to the same database here. If you want to, you can use two databases for the two registries shared with Publisher and Store nodes.

- Add the same configuration mentioned in section **a** of this step to the `Store registry.xml` file of the Store node as well.
10. Skip caching by adding the following configuration under the `<indexingConfiguration>` element in the `<API-M_HOME>/repository/conf/registry.xml` file in all Publisher and Store nodes.

This helps to avoid facing caching related issues in the Store and Publisher nodes by directly getting the API information from the database.

```
<skipCache>true</skipCache>
```

Step 3 - Configure the connections among the components

You will now configure the inter-component relationships of the distributed setup by modifying their <API-M_HOME>/repository/conf/api-manager.xml files.

In a clustered environment, you use Session Affinity to ensure that requests from the same client always get routed to the same server.

It is **mandatory** to set up Session Affinity in the load balancers that front the **Publisher** and **Store** clusters, and it is **optional** in the load balancer (if any) that fronts a **Key Manager** cluster or Gateway Cluster.

However, authentication via session ID fails when Session Affinity is disabled in the load balancer.

First time authentication happens via Basic Auth and the Gateway gets a cookie. This cookie is used in every consequent request along with the Basic Auth credentials. The admin service validates the cookie and if the validation fails it re-authenticate using basic auth and issues a new cookie.

This section includes the following sub-topics.

- Step 3.1 - Configure the Key Manager
- Step 3.2 - Configure the API Publisher
- Step 3.3 - Configure the API Store
- Step 3.4 - Configure the Traffic Manager
- Step 3.5 - Configure the Gateway

Step 3.1 - Configure the Key Manager

This section involves setting up the Key Manager node and enabling it to work with the other components in the distributed setup.

1. Open the <APIM_HOME>/repository/conf/api-manager.xml file.
2. Change the <ServerURL> element that appears under the <APIGateway> section, so that it points to the API Manager server. If you are using a distributed mode, this needs to point to the Gateway Manager node. This is done so that when a user is deleted or when the role of a user is updated in the Key Manager, it will update the Gateway cache by clearing the cache entries of a particular user. The port value you enter here should be the management transport port. For more information, see [Default Product Ports](#).

- <ServerURL>[https://\\${GATEWAY_SERVER_HOST}:{port}/services/](https://${GATEWAY_SERVER_HOST}:{port}/services/)</ServerURL>
3. Configure the API key validator in the Key Manager. The Thrift protocol is normally enabled by default. However, if you have disabled the Thrift protocol, enable it as follows in the <API-M_HOME>/repository/conf/api-manager.xml file.

```
<APIKeyValidator>

    <KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
    <EnableThriftServer>true</EnableThriftServer>
    <ThriftServerHost>localhost</ThriftServerHost>
    <ThriftServerPort>10397</ThriftServerPort>
    ...
</APIKeyValidator>
```

If you wish to encrypt the Auth Keys (access tokens, client secrets and authorization codes follow [Encrypting OAuth Keys](#).

The default Thrift server port is 10397. You need to only uncomment the following code and set the Thrift server port if you need to use a port that differs from the default value.

```
<ThriftServerPort>{port}</ThriftServerPort>
```

Step 3.2 - Configure the API Publisher

This section involves setting up the API Publisher node and enabling it to work with the other components in the distributed setup.

1. Open the <API-M_HOME>/repository/conf/api-manager.xml file in the API Publisher node.
2. Configure the API Gateway node using the following configurations. This configuration is used to publish synapse definition in the Gateway node. If gateway cluster is used add the details of the gateway manager.

```
<AuthManager>
    <ServerURL>https://<IP of the Key Manager>:9443/services/</ServerURL>
    <Username>${admin.username}</Username>
    <Password>${admin.password}</Password>
</AuthManager>

<APIGateway>
    <Environments>
        <Environment type="hybrid" api-console="true">
            <Name>Production and Sandbox</Name>
            <Description>This is a hybrid gateway that handles both production and sandbox token traffic.</Description>
            <ServerURL>https://<API-Gateway-Host or IP>:9443/services/</ServerURL>
                <Username>${admin.username}</Username>
                <Password>${admin.password}</Password>
        <GatewayEndpoint>http://<API-Gateway-Host>:8280,https://<API-Gateway-Host>:8243</GatewayEndpoint>
        </Environment>
    </Environments>
</APIGateway>
```

Change admin password

To change the admin password go to [Changing the super admin password](#). See the note given under step 2 for instructions to follow if your password has special characters.

- Configure the Traffic Manager node-related configuration in the API Publisher. This configuration enabled Throttling Policies, Custom Templates, and Block conditions to be published into the Gateway node.

```

<ThrottlingConfigurations>
    <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
    <DataPublisher>
        <Enabled>false</Enabled>
        .....
    </DataPublisher>
    <PolicyDeployer>

    <ServiceURL>https://<Traffic-Manager-Host>:9443/services/</ServiceURL>
        <Username>${admin.username}</Username>
        <Password>${admin.password}</Password>
    </PolicyDeployer>
    <BlockCondition>
        <Enabled>false</Enabled>
        .....
    </BlockCondition>
    <JMSSConnectionDetails>
        <Enabled>false</Enabled>
        .....
    </JMSSConnectionDetails>
    <JMSEventPublisherParameters>

        <java.naming.factory.initial>org.wso2.andes.jndi.PropertiesFileInitialContextFactory</java.naming.factory.initial>

        <java.naming.provider.url>repository/conf/jndi.properties</java.naming.provider.url>

        <transport.jms.DestinationType>topic</transport.jms.DestinationType>
        <transport.jms.Destination>throttleData</transport.jms.Destination>

        <transport.jms.ConcurrentPublishers>allow</transport.jms.ConcurrentPublishers>

        <transport.jms.ConnectionFactoryJNDIName>TopicConnectionFactory</transport.jms.ConnectionFactoryJNDIName>
    </JMSEventPublisherParameters>
    .....
</ThrottlingConfigurations>
```

- Set the <DisplayURL> to true and provide the URL of the Store.

When using a **distributed/clustered API Manager setup**, this configuration must be done in the API Publisher node/s.

Example

```
<APIStore>
    <DisplayURL>true</DisplayURL>
    <URL>https://<hostname>:9443/store</URL>
</APIStore>
```

- <hostname> - The hostname of the API Store node.

5. Open the <API-M_HOME>/repository/conf/jndi.properties file and make the following changes.

```
connectionfactory.TopicConnectionFactory =
amqp://admin:admin@clientid/carbon?brokerlist='tcp://<Traffic-Manager
-host>:5672'
topic.throttleData = throttleData
```

If you change the default admin: admin username and password in the user-mgt.xml, that username and password should be changed at the broker connection URL as well.

6. Disable the Thrift Server to optimize performance.
You need to configure this in the Publisher <API-M_HOME>/repository/conf/api-manager.xml file.

```
<APIKeyValidator>
...
<EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

Step 3.3 - Configure the API Store

This section involves setting up the API Store node and enabling it to work with the other components in the distributed setup.

1. Open the <API-M_HOME>/repository/conf/api-manager.xml file in the Store component.
2. Make the following changes to the api-manager.xml file. In a cluster of API gateways add the details of gateway workers in <GatewayEndpoint> under <APIGateway> tag and add the gateway Manager's URL as the <ServerURL>.

If your Gateway Cluster is fronted by a Load Balancer, use the load balancer endpoints under <GatewayEndpoint> configuration and add the Load balancer URL as <ServerURL> in the below configuration.

```

<RevokeAPIURL>https://<IP of the Gateway>:8243/revoke</RevokeAPIURL>

<APIKeyValidator>
    <ServerURL>https://<IP of the Key Manager>:9443/services/</ServerURL>
    <Username>${admin.username}</Username>
    <Password>${admin.password}</Password>
    ...
</APIKeyValidator>

<AuthManager>
    <ServerURL>https://<IP of the Key Manager>:9443/services/</ServerURL>
    <Username>${admin.username}</Username>
    <Password>${admin.password}</Password>
</AuthManager>

<APIGateway>
    <Environments>
        <Environment type="hybrid">
            ...
            <ServerURL>https://<IP or hostname of the
Gateway>:9443/services/</ServerURL>
            <Username>${admin.username}</Username>
            <Password>${admin.password}</Password>
            <GatewayEndpoint>http://<IP of the Gateway>:8280,https://<IP of the
Gateway>:8243</GatewayEndpoint>
        </Environment>
    </Environments>
    ...
</APIGateway>

```

If you wish to encrypt the Auth Keys (access tokens, client secrets and authorization codes follow [Encrypting OAuth Keys](#).

If there are multiple Gateway workers and they are fronted by a load balancer, you can use the relevant load balancer endpoints for HTTP and HTTPS, for **GatewayEndpoint** shown in the previous configuration. Following is an example of such a configuration:

```

<APIGateway>
...
<Environments>
<Environment type="hybrid">
...
<GatewayEndpoint>http://<load_balancer_endpoint>:8280,https://<load_balancer_endpoint>:8243</GatewayEndpoint>
</Environment>
</Environments>
...
</APIGateway>

```

3. Make the following throttling related changes which correspond to the Traffic Manager.

```

<ThrottlingConfigurations>
    <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
    <DataPublisher>
        <Enabled>false</Enabled>
        .....
        </DataPublisher>
        .....
        <BlockCondition>
            <Enabled>false</Enabled>
        .....
        </BlockCondition>
        <JMSConnectionDetails>
            <Enabled>false</Enabled>
        .....
        </JMSConnectionDetails>
        .....
    </ThrottlingConfigurations>

```

4. Disable the Thrift Server to optimize performance.
You need to configure this in the Store <API-M_HOME>/repository/conf/api-manager.xml file.

```

<APIKeyValidator>
...
<EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>

```

It is not recommended to share the solar directory between the servers (Store and Publisher). We need to have separate solar directories for each server so that they will perform solr indexing separately.

If you get an error like below in both or one of the nodes, check whether you have shared the

solr directory.

```
org.apache.solr.common.SolrException:
```

```
SolrCore 'registry-indexing' is not available due to init failure:  
Index locked for write for core registry-indexing
```

If you wish to configure a cluster of store nodes, you need to configure the deployment synchronization between those store nodes. Refer [Configuring rsync for Deployment Synchronization](#) for more information on configuring deployment synchronization between nodes.

Step 3.4 - Configure the Traffic Manager

This section involves setting up the Traffic Manager node and enabling it to work with the other components in the distributed setup.

1. Replace the <API-M_HOME>/repository/conf/registry.xml file with the <API-M_HOME>/repository/conf/registry_TM.xml file.
2. Replace the <API-M_HOME>/repository/conf/axis2/axis2.xml file with the <API-M_HOME>/repository/conf/axis2/axis2_TM.xml file.
3. Remove all the existing Jaggery apps. Do this by removing all the contents from the <API-M_HOME>/repository/deployment/server/jaggeryapps directory.
4. As mentioned in [step 1 \(5\)](#), make sure to remove all the existing webapps in the following directory **with the exception** of the shindig web application.
<API-M_HOME>/repository/deployment/server/webapps
5. Disable the Thrift Server to optimize performance. You need to configure this in the Traffic Manager <API-M_HOME>/repository/conf/api-manager.xml file

```
<APIKeyValidator>  
...  
  <EnableThriftServer>false</EnableThriftServer>  
</APIKeyValidator>
```

6. Start the Traffic Manager node using the following command.

```
sh wso2server.sh -Dprofile=traffic-manager
```

Always start traffic Manager with profile **-Dprofile=traffic-manager** to avoid FATAL errors occur like below.

```
FATAL - ServiceBusInitializer Couldn't initialize the ESB...  
org.apache.synapse.SynapseException: The synapse.xml location  
./..  
  ./repository/deployment/server/synapse-configs  
/default doesn't exist
```

If you have a firewall between the Traffic Manager and the Gateway, you need to configure the heartbeat value to keep the JMS connection alive. To configure this, open the <APIM_HOME>/repository/conf/advanced/qpid-config.xml file and set the heartbeat to a non-zero value as shown below.

```
<heartbeat>
  <delay>60</delay>
  <timeoutFactor>2.0</timeoutFactor>
</heartbeat>
```

Step 3.5 - Configure the Gateway

- **Follow the instructions below only if you are deploying WSO2 API-M with one Gateway node.**
- If you are using two Gateway nodes for high availability (HA), skip the following Gateway configurations and follow the instructions given in the [Distributed Deployment of the Gateway](#) document.

This section involves setting up the Gateway node and enabling it to work with the other components in the distributed setup.

1. Open the <API-M_HOME>/repository/conf/api-manager.xml file in the Gateway node.
2. Modify the api-manager.xml file as follows. This configures the connection to the Key Manager component.

```
<APIKeyValidator>
  <ServerURL>https://<IP of the Key
Manager>:9443/services/</ServerURL>
  <Username>${admin.username}</Username>
  <Password>${admin.password}</Password>
  ...
</APIKeyValidator>
```

3. Configure key management related communication in the Gateway node.

Key Manager cluster fronted by a load balancer
Key Manager cluster without a load balancer

- a. In a clustered setup if the Key Manager is fronted by a load balancer, you have to use WSClient as KeyValidatorClientType in the <API-M_HOME>/repository/conf/api-manager.xml file. This should be configured in all Gateway and Key Manager components.

```
<KeyValidatorClientType>WSClient</KeyValidatorClientType>
```

- b. Disable the Thrift Server to optimize performance.

You need to configure this in the Gateway <API-M_HOME>/repository/conf/api-manager.xml file.

```
<APIKeyValidator>
...
<EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

- In a clustered setup if the Key Manager is NOT fronted by a load balancer, you have to use `ThriftClient` as `KeyValidatorClientType` in the <API-M_HOME>/repository/conf/api-manager.xml file. This should be configured in all Gateway and Key Manager components.

```
<KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
```

- Disable the Thrift Server to optimize performance. You need to configure this in the Gateway <API-M_HOME>/repository/conf/api-manager.xml file.

```
<EnableThriftServer>false</EnableThriftServer>
```

- Set the Thrift Client port in the Gateway to the same port as the Thrift Server port, which you defined in the Key Manager. This is done to enable the Gateway to communicate with the Key Manager.

Do this step **only if** one of the following scenarios are applicable to your setup.

- You started or you plan on starting the Key Manager or Gateway with a port offset.
- Your Thrift Server port in the Key Manager differs from the default value, which is 10397

```
<ThriftClientPort>{port}</ThriftClientPort>
```

- Specify the hostname or IP of the Key Manager. The default is `localhost`. In a distributed deployment you must set this parameter in both Key Manager nodes and Gateway nodes only if the Key Manager is running on a separate machine. Gateway uses this parameter to connect to the key validation Thrift service.

```
<ThriftServerHost>localhost</ThriftServerHost>
```

- If you need to enable JSON Web Token (JWT), you have to enable it in all Gateway and Key Manager components. For more information on configuring JWT, see [Generating JSON Web Token](#).

- Configure throttling for Traffic Manager on gateway node.

Note: 9611 and 9711 are the Traffic Manager receiver ports for the Binary type.

```
<ThrottlingConfigurations>
    <EnableAdvanceThrottling>true</EnableAdvanceThrottling>
    <DataPublisher>
        <Enabled>true</Enabled>
        <Type>Binary</Type>

    <ReceiverUrlGroup>tcp://<Traffic-Manager-host>:9611</ReceiverUrlGroup>

    <AuthUrlGroup>ssl://<Traffic-Manager-host>:9711</AuthUrlGroup>
    .....
        </DataPublisher>
        <PolicyDeployer>

    <ServiceURL>https://<Traffic-Manager-host>:9443/services/</ServiceURL>
    .....
        </PolicyDeployer>
    .....
        <JMSConnectionDetails>
            <Enabled>true</Enabled>
            <ServiceURL>tcp://<Traffic-Manager-host>:5672</ServiceURL>
        .....
            </JMSConnectionDetails>
    </ThrottlingConfigurations>
```

Publishing to Multiple Traffic Managers

If you have more than one Traffic Manager instance and you are publishing to both as per the deployment pattern selected, the **ReceiverUrlGroup** and **AuthUrlGroup** should be changed to contain all Traffic Manager receiver URLs. As an example, if you are using two Traffic Manager instances and data should be published to both of them, the ReceiverUrlGroup and AuthUrlGroup should be configured as follows:

```

<ThrottlingConfigurations>
.....
<DataPublisher>
<Enabled>true</Enabled>
<Type>Binary</Type>

<ReceiverUrlGroup>{tcp://<Traffic-Manager-host-1>:9611},{tcp://<Traffic-Manager-host-2>:9611}</ReceiverUrlGroup>

<AuthUrlGroup>{ssl://<Traffic-Manager-host-1>:9711},{ssl://<Traffic-Manager-host-2>:9711}</AuthUrlGroup>
.....
</DataPublisher>
</ThrottlingConfigurations>

```

Here, Traffic-Manager-host-1 and Traffic-Manager-host-2 are the IPs/Hostnames of two Traffic Manager nodes. For more information on publishing patterns to multiple Traffic Managers, see [Setting up multi-reciever data agent](#).

- Comment out the following section to prevent warning messages during startup of the Gateway node in the cluster setup.

```

<JMSEventPublisherParameters>
<java.naming.factory.initial>org.wso2.andes.jndi.PropertiesFileInitialContextFactory</java.naming.factory.initial>
<java.naming.provider.url>repository/conf/jndi.properties</java.naming.provider.url>
<transport.jms.DestinationType>topic</transport.jms.DestinationType>
<transport.jms.Destination>throttleData</transport.jms.Destination>
<transport.jms.ConcurrentPublishers>allow</transport.jms.ConcurrentPublishers>
<transport.jms.ConnectionFactoryJNDIName>TopicConnectionFactory</transport.jms.ConnectionFactoryJNDIName>
</JMSEventPublisherParameters>

```

SSL Keys and Certificates

WSO2 API Manager is by default shipped with a keystore (wso2carbon.jks) and a trust store (client-truststore.jks). See [Configuring Keystore and Truststore](#) section on how to create a new key store and a trust store with a private key and a self-signed certificate. Refer [Administration guide](#) for recommendations for setting up keystores in WSO2 products.

It is not recommended to share the solar directory between the nodes. We need to have separate solar directories for the servers. The servers will perform separate indexing for themselves.

Below is a description about the most important configuration parameters for throttling, in

<API-M_HOME>/repository/conf/api-manager.xml.

EnableAdvanceThrottling : Indicates if advanced throttling is enabled or not
 DataPublisher : Specify the details of throttle data publisher
 ReceiverUrlGroup : URL group of throttle data publisher event receiver
 AuthUrlGroup : URL group of throttle data publisher authentication
 BlockCondition : Indicates if there is a blocking condition
 JMSConnectionDetails : Contains the parameters related to the JMS connection
 JMSConnectionParameters : Parameters related to the JMS connection such as destination type
 JMSEventPublisherParameters : Parameters related to JMS event publishing such as destination, JNDI name of the connection factory

Step 4 - Start the cluster

You can start up the nodes in any order that you prefer, but it is recommended to setup the Traffic Manager first. You can start the nodes [using profiles](#) or [using the standard startup script](#).

When invoking WSO2 API Manager from any profile other than the Gateway profile, you need to do the following.

1. Open the <API-M_HOME>/repository/conf/axis2/axis2.xml file and comment out the following.

```
<transportSender name="ws"
class="org.wso2.carbon.websocket.transport.WebsocketTransportSe
nder">
<parameter name="ws.outflow.dispatch.sequence"
locked="false">outflowDispatchSeq</parameter>
<parameter name="ws.outflow.dispatch.fault.sequence"
locked="false">outflowFaultSeq</parameter>
</transportSender>
```

2. Delete the `WebSocketInboundEndpoint.xml` file from the <API-M_HOME>/repository/deploy
yment/server/synapse-configs/default/inbound-endpoints directory.

Distributed Deployment of the Gateway

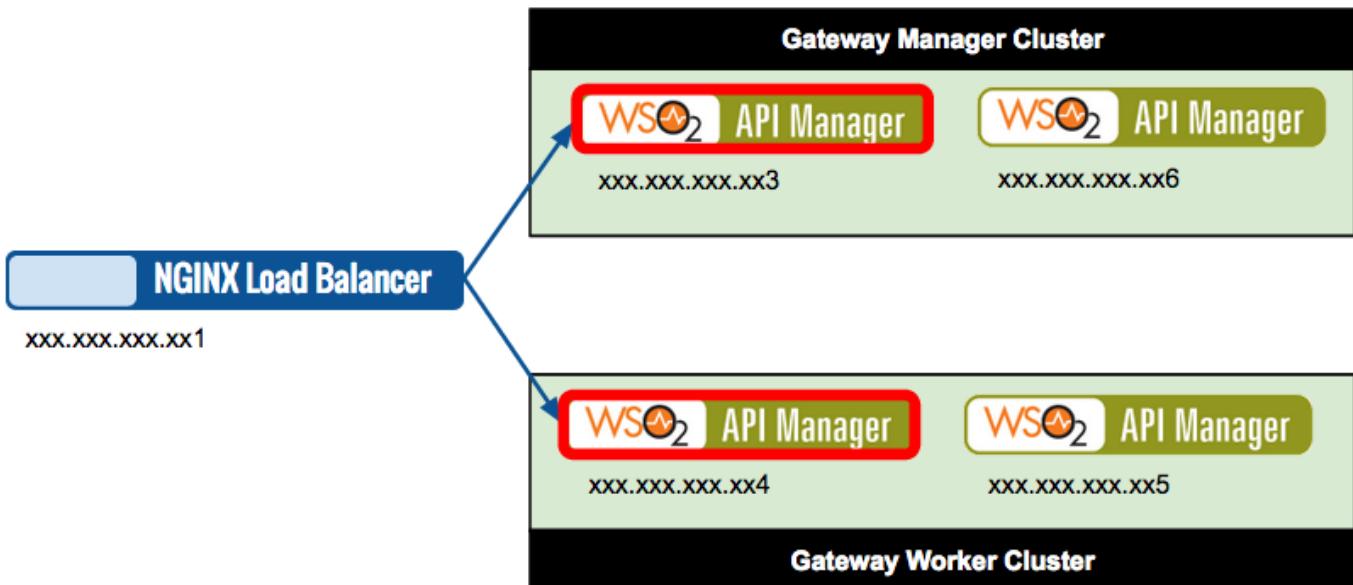
This topic provides instructions on how to configure the multiple Gateways in WSO2 API Manager (WSO2 API-M) in a distributed deployment to facilitate high availability (HA). The instructions in this topic are based on the [Distributed Deployment of API Manager](#) and these configurations will only work if the configurations in that topic are done correctly. For instance, all datasource configurations are already done for the Gateway in that topic and hence do not need to be repeated here.

The following sections provide specific instructions on configuring the Gateway cluster.

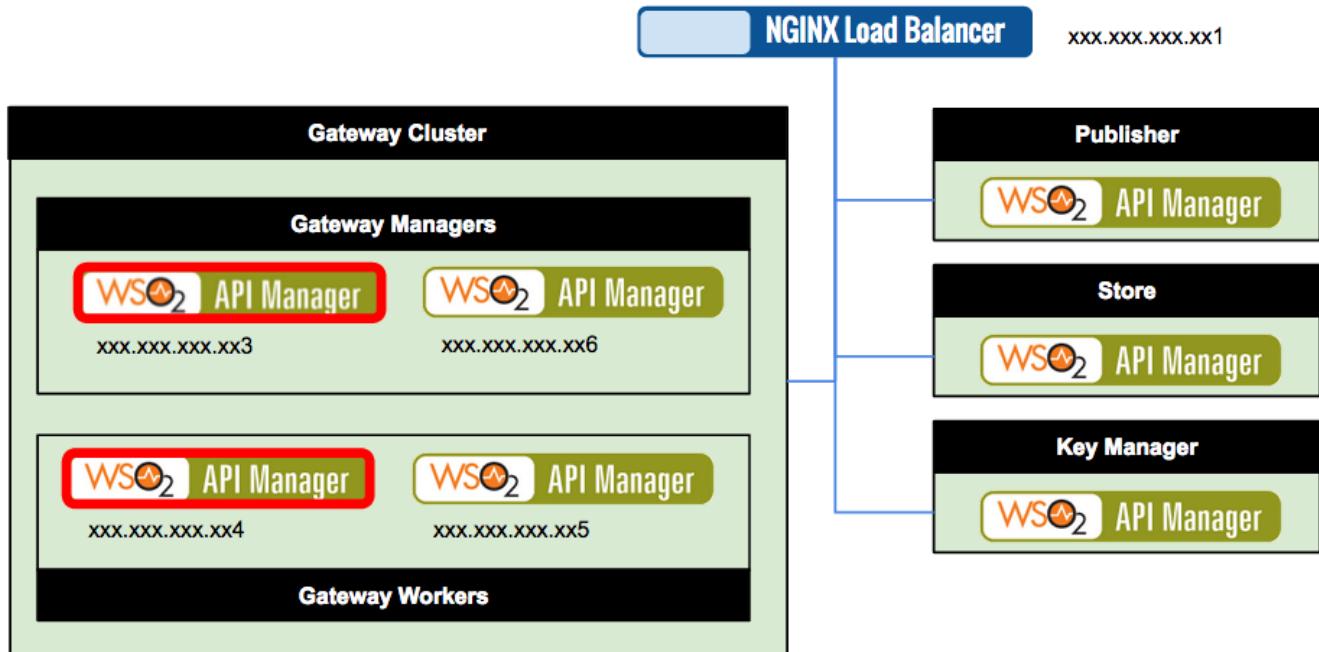
- Step 1 - Configure the load balancer
- Step 2 - Create SSL certificates
- Step 3 - Configure the Gateway manager
- Step 4 - Configure the Gateway worker
- Step 5 - Optionally configure Hazelcast
- Step 6 - Start the Gateway Nodes

Gateway deployment pattern

The configurations in this topic will be done based on the following pattern. This pattern is used as a basic Gateway cluster where the worker nodes and manager nodes are separated.



[Click here to view a sample of the full API Manager cluster.](#)



Step 1 - Configure the load balancer

NGINX is used for this scenario, but you can use any load balancer that you prefer here. The following are the configurations that need to be done for the load balancer.

Use the following steps to configure [Nginx](#) as the load balancer for WSO2 products.

1. Install Nginx using the following command.
`$sudo apt-get install nginx`
2. Configure Nginx Plus to direct the HTTP requests to the two worker nodes via the HTTP 80 port using the `http` block. To do this, create a VHost file (`am.http.conf`) in the `/etc/nginx/conf.d/` directory and add the following configurations into it.

```

upstream wso2.am.com {
    sticky cookie JSESSIONID;
    server xxx.xxx.xxx.xx4:9763;
    server xxx.xxx.xxx.xx5:9763;
}

server {
    listen 80;
    server_name am.wso2.com;
    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_read_timeout 5m;
        proxy_send_timeout 5m;
        proxy_pass http://wso2.am.com;
    }
}

```

3. Configure Nginx Plus to direct the HTTPS requests to the two worker nodes via the HTTPS 443 port using ht¹. To do this, create a VHost file (am.https.conf) in the /etc/nginx/conf.d/ directory and add the following configurations into it.

```

upstream ssl.wso2.am.com {
    sticky cookie JSESSIONID;
    server xxx.xxx.xxx.xx4:9443;
    server xxx.xxx.xxx.xx5:9443;
}

server {
    listen 443;
    server_name am.wso2.com;
    ssl on;
    ssl_certificate /etc/nginx/ssl/wrk.crt;
    ssl_certificate_key /etc/nginx/ssl/wrk.key;
    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_read_timeout 5m;
        proxy_send_timeout 5m;
        proxy_pass https://ssl.wso2.am.com;
    }
}

```

4. Configure Nginx Plus to access the Management Console as `https://mgt.am.wso2.com/carbon` via HTTPS 443 port. This is to direct requests to the manager node. To do this, create a VHost file (`mgt.am.conf`) in the `/etc/nginx/conf.d/` directory and add the following configurations into it.

```
server {
    listen 443;
    server_name mgt.am.wso2.com;
    ssl on;
    ssl_certificate /etc/nginx/ssl/mgt.crt;
    ssl_certificate_key /etc/nginx/ssl/mgt.key;

    location / {
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For
        $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_read_timeout 5m;
        proxy_send_timeout 5m;
        proxy_pass https://xxx.xxx.xxx.xx3:9443/;
    }
    error_log /var/log/nginx/mgt-error.log ;
    access_log /var/log/nginx/mgt-access.log;
}
```

5. Restart the Nginx Plus server.
`$sudo service nginx restart`

Tip: You do not need to restart the server if you are simply making a modification to the VHost file. The following command should be sufficient in such cases.
`sudo service nginx reload`

Step 2 - Create SSL certificates

Create SSL certificates for both the manager and worker nodes using the instructions that follow.

- In the sample configurations given under **Step 1 - Configure the Load Balancer**, the names of ssl key and the certificate for the manager node is `mgt.key` and `mgt.crt`, and for worker node its `wrk.key` and `wrk.crt` respectively. Use these names for `<key_name>` and `<certificate_name>` when you are generating keys and certificates.
- While creating keys, enter the host name (`am.wso2.com` or `mgt.am.wso2.com`) as the Common Name.

1. Create the Server Key.

```
$sudo openssl genrsa -des3 -out <key_name>.key 1024
```

2. Certificate Signing Request.

```
$sudo openssl req -new -key <key_name>.key -out server.csr
```

3. Remove the password.

```
$sudo cp <key_name>.key <key_name>.key.org
$sudo openssl rsa -in <key_name>.key.org -out <key_name>.key
```

4. Sign your SSL Certificate.

```
$sudo openssl x509 -req -days 365 -in server.csr -signkey
<key_name>.key -out <certificate_name>.crt
```

5. Copy the key and certificate files generated in above step 4 to /etc/nginx/ssl/ location.

You have now configured the load balancer to handle requests sent to am.wso2.com and to distribute the load among the worker nodes in the worker sub-domain of the wso2.am.domain cluster.

You are now ready to set up the cluster configurations. The next step is to configure the Gateway manager.

Step 3 - Configure the Gateway manager

Management nodes specialize in management of the setup. Only management nodes are authorized to add new artifacts into the system or make configuration changes. Management nodes are usually behind an internal firewall and are exposed to clients running within the organization only. This section involves setting up the Gateway node and enabling it to work with the other components in the distributed setup.

- Step 3.1 - Configure the common configurations
- Step 3.2 - Configure the carbon.xml file
- Step 3.3 - Configure the catalina-server.xml file
- Step 3.4 - Map the hostnames to IPs

Step 3.1 - Configure the common configurations

Carryout the following configurations on the Gateway manager node.

Note that these configurations are common to the Gateway Manager and Gateway Worker nodes.

1. Open the <API-M_HOME>/repository/conf/api-manager.xml file in the Gateway node.
2. Modify the api-manager.xml file as follows. This configures the connection to the Key Manager component.

```
<APIKeyValidator>
    <ServerURL>https://<IP of the Key
Manager>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
</APIKeyValidator>
```

Change admin password

To change the admin password go to [Changing the super admin password](#). See the note given under step 2 for instructions to follow if your password has special characters.

3. Configure key management related communication.

Cluster fronted by a load balancer Cluster without a load balancer

- In a clustered setup if the Key Manager is fronted by a load balancer, you have to use `WSClient` as `KeyValidatorClientType` in `<API-M_HOME>/repository/conf/api-manager.xml`. This should be configured in all Gateway and Key Manager components.

```
<KeyValidatorClientType>WSClient</KeyValidatorClientType>
```

- Disable the Thrift Client to optimize performance.

You need to configure this in the Gateway `<API-M_HOME>/repository/conf/api-manager.xml` file.

```
<APIKeyValidator>...
  <EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

- In a clustered setup if the Key Manager is NOT fronted by a load balancer, you have to use `ThriftClient` as `KeyValidatorClientType` in `<API-M_HOME>/repository/conf/api-manager.xml`. This should be configured in all Gateway and Key Manager components.

```
<KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
```

- Disable the Thrift Client to optimize performance.

You need to configure this in the Gateway `<API-M_HOME>/repository/conf/api-manager.xml` file.

```
<APIKeyValidator>
  ...
  <EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

- Specify the `ThriftClientPort` and `ThriftServerPort` values. 10397 is the default.

```
<ThriftClientPort>10397</ThriftClientPort>
<ThriftServerPort>10397</ThriftServerPort>
```

- Specify the hostname or IP of the Key Manager. The default is `localhost`. In a distributed deployment we must set this parameter in both Key Manager nodes and Gateway nodes only if the

Key Manager is running on a separate machine. Gateway uses this parameter to connect to the key validation thrift service.

```
<ThriftServerHost>localhost</ThriftServerHost>
```

4. If you need to enable JSON Web Token (JWT) you have to enable it in all Gateway and Key Manager components.

For more information on enabling JWT, see [Generating JSON Web Token](#).

Step 3.2 - Configure the carbon.xml file

The following configurations are done in the <GATEWAY_MANAGER_HOME>/repository/conf/carbon.xml file.

1. Open <GATEWAY_MANAGER_HOME>/repository/conf/carbon.xml.
2. Locate the <HostName> tag and add the cluster host name: <HostName>am.wso2.com</HostName>
3. Locate the <MgtHostName> tag and uncomment it. Make sure that the management host name is defined as follows: <MgtHostName> mgt.am.wso2.com </MgtHostName>

Step 3.3 - Configure the catalina-server.xml file

Specify the following configurations in the catalina-server.xml file located in the <GATEWAY_MANAGER_HOME>, directory.

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9763"
           proxyPort="80"
           -----
           />
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9443"
           proxyPort="443"
           -----
           />
```

The TCP port number is the value that this Connector will use to create a server socket and await incoming connections. Your operating system will allow only one server application to listen to a particular port number on a particular IP address.

Step 3.4 - Map the hostnames to IPs

Open the server's /etc/hosts file and add the following.

```
<GATEWAY-WORKER-IP> am.wso2.com
```

In this example, it would look like this:

```
xxx.xxx.xxx.xx4 am.wso2.com
```

Once you replicate these configurations for all the manager nodes, your Gateway manager is configured. Next configure the Gateway worker.

Step 4 - Configure the Gateway worker

Worker nodes specialize in serving requests to deployment artifacts and reading them. They can be exposed to

external clients.

- Step 4.1 - Configure the common configurations
- Step 4.2 - Configure the carbon.xml file
- Step 4.3 - Configure the catalina-server.xml file
- Step 4.4 - Map the hostnames to IPs

Step 4.1 - Configure the common configurations

Carryout the following configurations on the Gateway worker node.

Note that these configurations are common to the Gateway Manager and Gateway Worker nodes.

1. Open the <API-M_HOME>/repository/conf/api-manager.xml file in the Gateway node.
2. Modify the api-manager.xml file as follows. This configures the connection to the Key Manager component.

```
<APIKeyValidator>
    <ServerURL>https://<IP of the Key
Manager>:9443/services/</ServerURL>
    <Username>admin</Username>
    <Password>admin</Password>
    ...
</APIKeyValidator>
```

Change admin password

To change the admin password go to [Changing the super admin password](#). See the note given under step 2 for instructions to follow if your password has special characters.

3. Configure key management related communication.

Cluster fronted by a load balancer Cluster without a load balancer

- a. In a clustered setup if the Key Manager is fronted by a load balancer, you have to use `WSClient` as `KeyValidatorClientType` in <API-M_HOME>/repository/conf/api-manager.xml. This should be configured in all Gateway and Key Manager components.

```
<KeyValidatorClientType>WSClient</KeyValidatorClientType>
```

- b. Disable the Thrift Client to optimize performance.

You need to configure this in the Gateway <API-M_HOME>/repository/conf/api-manager.xml file.

```
<APIKeyValidator>...
    <EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

- a. In a clustered setup if the Key Manager is NOT fronted by a load balancer, you have to use `ThriftClient` as `KeyValidatorClientType` in <API-M_HOME>/repository/conf/api-manager.xml. This

should be configured in all Gateway and Key Manager components.

```
<KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
```

- b. Disable the Thrift Client to optimize performance.

You need to configure this in the Gateway <API-M_HOME>/repository/conf/api-manager.xml file.

```
<APIKeyValidator>
...
<EnableThriftServer>false</EnableThriftServer>
</APIKeyValidator>
```

- c. Specify the ThriftClientPort and ThriftServerPort values. 10397 is the default.

```
<ThriftClientPort>10397</ThriftClientPort>
<ThriftServerPort>10397</ThriftServerPort>
```

- d. Specify the hostname or IP of the Key Manager. The default is localhost. In a distributed deployment we must set this parameter in both Key Manager nodes and Gateway nodes only if the Key Manager is running on a separate machine. Gateway uses this parameter to connect to the key validation thrift service.

```
<ThriftServerHost>localhost</ThriftServerHost>
```

4. If you need to enable JSON Web Token (JWT) you have to enable it in all Gateway and Key Manager components.

For more information on enabling JWT, see [Generating JSON Web Token](#).

Step 4.2 - Configure the carbon.xml file

1. Open <GATEWAY_WORKER_HOME>/repository/conf/carbon.xml on each worker node.
2. Specify the host name as follows.
<HostName>am.wso2.com</HostName>

You can [configure the Deployment Synchronizer](#), which is also done in the **carbon.xml** file.

Step 4.3 - Configure the catalina-server.xml file

Make the following configuration changes in the **catalina-server.xml** file which is found in the <GATEWAY_WORKER_HOM> directory.

```

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9763"
           proxyPort="80"
-----
/>
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="9443"
           proxyPort="443"
-----
/>
```

In the next section, we will map the host names we specified to real IPs.

Step 4.4 - Map the hostnames to IPs

Open the server's /etc/hosts file and add the following.

```
<GATEWAY-MANAGER-IP> mgt.am.wso2.com
```

In this example, it would look like this:

```
xxx.xxx.xxx.xx3 mgt.am.wso2.com
```

Step 5 - Optionally configure Hazelcast

You can seamlessly deploy WSO2 API Manager using local caching in a clustered setup without Hazelcast clustering. However, there are edge case scenarios where you need to enable Hazelcast clustering. To identify whether you need Hazelcast clustering and to configure Hazelcast clustering if needed, see [Working with Hazelcast Clustering](#).

Step 6 - Start the Gateway Nodes

Step 6.1 - Start the Gateway manager

1. Comment the following configurations in the <API-M_HOME>/repository/conf/api-manager.xml file.

```
<JMSEventPublisherParameters>

<java.naming.factory.initial>org.wso2.andes.jndi.PropertiesFileInitia
lContextFactory</java.naming.factory.initial>

<java.naming.provider.url>repository/conf/jndi.properties</java.namin
g.provider.url>

<transport.jms.DestinationType>topic</transport.jms.DestinationType>
    <transport.jms.Destination>throttleData</transport.jms.Destination>

<transport.jms.ConcurrentPublishers>allow</transport.jms.ConcurrentPu
blishers>

<transport.jms.ConnectionFactoryJNDIName>TopicConnectionFactory</tran
sport.jms.ConnectionFactoryJNDIName>
</JMSEventPublisherParameters>
```

2. Start the Gateway manager by typing the following command in the terminal.

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

Step 6.2 - Start the Gateway worker

It is recommendation is to delete the <API-M_HOME>/repository/deployment/server directory and create an empty server directory in the worker node. This is done to avoid any SVN conflicts that may arise. Note that when you do this, you may have to restart the worker node after you start it in order to avoid an error.

1. Update the <API-M_HOME>/repository/conf/api-manager.xml file by commenting out the following configurations.

```
<JMSEventPublisherParameters>

<java.naming.factory.initial>org.wso2.andes.jndi.PropertiesFileInitia
lContextFactory</java.naming.factory.initial>

<java.naming.provider.url>repository/conf/jndi.properties</java.namin
g.provider.url>

<transport.jms.DestinationType>topic</transport.jms.DestinationType>
    <transport.jms.Destination>throttleData</transport.jms.Destination>

<transport.jms.ConcurrentPublishers>allow</transport.jms.ConcurrentPu
blishers>

<transport.jms.ConnectionFactoryJNDIName>TopicConnectionFactory</tran
sport.jms.ConnectionFactoryJNDIName>
</JMSEventPublisherParameters>
```

- Start the Gateway worker by typing the following command in the terminal.

```
sh <GATEWAY_WORKER_HOME>/bin/wso2server.sh -Dprofile=gateway-worker
```

The additional `-Dprofile=gateway-worker` argument indicates that this is a worker node specific to the Gateway. This parameter basically makes a server read-only. A node with this parameter will not be able to do any changes such as writing or making modifications to the deployment repository, etc. Starting this node as a Gateway worker ensures that Store and Publisher related functionality is disabled. This parameter also ensures that the node starts as the [worker profile](#), where the UI bundles will not be activated and only the back end bundles will be activated once the server starts up. See [API Manager product profiles](#) for more information on the various product profiles available in API Manager.

Working with Hazelcast Clustering

Hazelcast is an in-memory data grid, which is used in clustering and distributed shared memory. When using Hazelcast as a clustering implementation, data is evenly distributed among the nodes in a cluster. Hazelcast clustering is disabled in WSO2 API Manager (WSO2 APIM) by default, because you can successfully deploy WSO2 API-M without Hazelcast. The following subsections explain as to when you need Hazelcast clustering with WSO2 API Manager (WSO2 APIM) and how you can configure it.

- [When to use Hazelcast](#)
- [Deploying WSO2 API-M Manager with Hazelcast clustering](#)
- [Enabling Hazelcast clustering](#)

When to use Hazelcast

You can seamlessly deploy WSO2 API Manager using local caching in a clustered setup without Hazelcast clustering. However, the following are the edge case scenarios where you need to enable Hazelcast clustering.

- **Immediate revocation of tokens among the gateways**

If you deploy WSO2 API-M without Hazelcast clustering, you can set the time-to-live (TTL) value in secs in order to define the period after which the token will automatically be revoked in the other gateways. However, if you want the token in all the gateways to be immediately revoked, then you need to enable Hazelcast clustering.

- **Backend service throttling**

The endpoint throttling limits and the spike arrest throttling limits will not be shared across the cluster when Hazelcast clustering is disabled.

Deploying WSO2 API-M Manager with Hazelcast clustering

Follow the instructions below to deploy WSO2 API-M with Hazelcast clustering:

1. Carryout the instructions related to deploying WSO2 API-M.
2. Carryout the instructions to [enable Hazelcast clustering](#).
3. Start the WSO2 API-M servers.

Enabling Hazelcast clustering

Follow the instructions below to enable Hazelcast clustering when deploying WSO2 API-M:

- Step 1 - Enable Hazelcast clustering in the Gateway Manager
- Step 2 - Enable Hazelcast clustering in the Gateway Worker

Step 1 - Enable Hazelcast clustering in the Gateway Manager

1. Open the <GATEWAY_MANAGER_HOME>/repository/conf/axis2/axis2.xml file.
2. Locate the clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default).
 - a. Enable clustering for this node:

```
<clustering
class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
```

- b. Set the membership scheme to wka to enable the well-known address registration method (this node will send cluster initiation messages to WKA members that we will define later):

```
<parameter name="membershipScheme">wka</parameter>
```

- c. Specify the name of the cluster this node will join:

```
<parameter name="domain">wso2.am.domain</parameter>
```

- d. Specify the host used to communicate cluster messages. This is the IP of the Gateway manager node.

```
<parameter name="localMemberHost">xxx.xxx.xxx.xx3</parameter>
```

- e. Specify the port used to communicate cluster messages:

```
<parameter name="localMemberPort">4500</parameter>
```

This port number will not be affected by the port offset in carbon.xml. If this port number is already assigned to another server, the clustering framework will automatically increment this port number. However, if two servers are running on the same machine, you must ensure that a unique port is set for each server.

- f. Do the following port mapping configurations for the Gateway manager node. There are two types of transports in API Manager and when a request comes into the API Manager, it always goes to the

default transport which is the PTT/NIO transport. So when you access the management console of the Gateway Manager node, you send a servlet request. If you do not specify the port mapping parameter in the manager node, it would hit the PTT/NIO transport and the request would fail.

```
<parameter name="properties">
    <property name="backendServerURL"
    value="https://${hostName}:${httpsPort}/services/" />
    <property name="mgtConsoleURL"
    value="https://${hostName}:${httpsPort}"/>
    <property name="subDomain" value="mgt" />
    <property name="port.mapping.80" value="9763" />
    <property name="port.mapping.443" value="9443" />
</parameter>
```

- g. The receiver's HTTP/HTTPS port values are without the `portOffset` addition; they get auto-incremented by `portOffset`. In the case of an ESB cluster, the '`WSDLEPRPrefix`' parameter should point to the worker node's host name (`am.wso2.com`) and load balancer's http (80)/https (443) transport ports.
- h. Change the members listed in the `<members>` element. This defines the WKA members.

```
<members>
    <member>
        <hostName>xxx.xxx.xxx.xx3</hostName>
        <port>4500</port>
    </member>
    <member>
        <hostName>xxx.xxx.xxx.xx4</hostName>
        <port>4200</port>
    </member>
</members>
```

Here we configure the manager node and worker node as the well-known members.

it is recommended to add at least two well-known members here. This is done to ensure that there is high availability for the cluster.

You can also use IP address ranges for the `hostName`. For example, `192.168.1.2-10`. This should ensure that the cluster eventually recovers after failures. One shortcoming of doing this is that you can define a range only for the last portion of the IP address. You should also keep in mind that the smaller the range, the faster the time it takes to discover members, since each node has to scan a lesser number of potential members.

Step 2 - Enable Hazelcast clustering in the Gateway Worker

1. Open the `<GATEWAY_WORKER_HOME>/repository/conf/axis2/axis2.xml` file.
2. Locate the clustering section and verify or configure the properties as follows (some of these properties are already set correctly by default):
 - a. Enable clustering for this node:

```
<clustering
class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClus-
teringAgent" enable="true">
```

- b. Set the membership scheme to `wka` to enable the Well Known Address registration method (this node will send cluster initiation messages to WKA members that we will define later):

```
<parameter name="membershipScheme">wka</parameter>
```

- c. Specify the name of the cluster this node will join:

```
<parameter name="domain">wso2.am.domain</parameter>
```

- d. Specify the host used to communicate cluster messages. This is the IP address of the Gateway worker.

```
<parameter name="localMemberHost">xxx.xxx.xxx.xx4</parameter>
```

- e. Specify the port used to communicate cluster messages (if this node is on the same server as the manager node, or another worker node, be sure to set this to a unique value, such as 4000 and 4001 for worker nodes 1 and 2).

```
<parameter name="localMemberPort">4200</parameter>
```

This port number will not be affected by the port offset in `carbon.xml`. If this port number is already assigned to another server, the clustering framework will automatically increment this port number. However, if two servers are running on the same machine, you must ensure that a unique port is set for each server.

- f. Define the sub-domain as worker by adding the following property under the `<parameter name="properties" >` element:

```
<property name="subDomain" value="worker" />
```

- g. Define the manager and worker nodes as well-known members of the cluster by providing their host name and `localMemberPort` values. The manager node is defined here because it is required for the [Deployment Synchronizer](#) to function in an efficient manner. The deployment synchronizer uses this configuration to identify the manager and synchronize deployment artifacts across the nodes of a cluster.

```

<members>
  <member>
    <hostName>xxx.xxx.xxx.xx3</hostName>
    <port>4500</port>
  </member>
  <member>
    <hostName>xxx.xxx.xxx.xx4</hostName>
    <port>4200</port>
  </member>
</members>

```

it is recommended to add at least two well-known members here. This is done to ensure that there is high availability for the cluster.

You can also use IP address ranges for the `hostName`. For example, `192.168.1.2-10`. This should ensure that the cluster eventually recovers after failures. One shortcoming of doing this is that you can define a range only for the last portion of the IP address. You should also keep in mind that the smaller the range, the faster the time it takes to discover members, since each node has to scan a lesser number of potential members.

Configuring WSO2 Identity Server as a Key Manager

WSO2 recommends that you use the pre-packaged Identity Server as a Key Manager, as explained below, because most of the configurations already exist in the pre-packaged version. However, if you wish to use your existing WSO2 Identity Server (WSO2 IS) set up as the Key Manager, see [Configuring an External IdP as a Key Manager](#).

This configuration uses a **pre-packaged WSO2 Identity Server (WSO2 IS) 5.3.0** with **WSO2 API Manager (WSO2 API-M) 2.1.0**. This pre-packaged version is different to WSO2 Identity Server 5.3.0 because the configurations required to connect the Identity Server as the Key Manager of the API Manager are already packaged in it. The Key Manager feature of WSO2 API Manager is already installed and configured in it. Follow the instructions below to configure IS as the Key Manager.

- Step 1 - Set up the databases
- Step 2 - Download WSO2 API-M and WSO2 IS
- Step 3 - Optionally, configure port offset for WSO2 API-M or WSO2 IS
- Step 4 - Configure the Identity Server as Key Manager
- Step 5 - Configure the API Manager
- Step 6 - Start the servers

Step 1 - Set up the databases

In a typical production environment, it is possible that databases are already set up for the deployment. However, you need to have databases created for the registry and API management. This example uses MySQL to create the databases, but you can use any of the supported databases. See [Setting up the Physical Database](#) for more information on setting up different databases.

1. Create the following databases in the MySQL database server.
 - apimgt - API-M specific information such as API details, Applications and Subscriptions are stored in this database
 - userstore - If you are using a JDBC user store, details of the users are stored in this database. Additionally the permissions and internal roles will be stored in this DB.
 - registry - used as a repository for storing and sharing resources, service metadata, etc.

2. Create a user ‘apiuser’ with password ‘apimanager’. Grant all permissions for this user in the above three databases. For example:

```
grant all on apimgt.* TO apiuser@localhost identified by "apimanager";
grant all on userstore.* TO apiuser@localhost identified by
"apimanager";
grant all on registry.* TO apiuser@localhost identified by
"apimanager";
```

About the userstore

It can be preferable to use an LDAP to store users. This is different to the userstore database you create here. This userstore database is for storing permissions and internal roles, while the LDAP stores users and their role mapping.

About MySQL drivers

Download the [MySQL JDBC driver](#). Make sure you add the MySQL JDBC driver to the WSO2 IS and WSO2 API-M servers by copying the JAR file into the <PRODUCT_HOME>/repository/components/lib directory.

Step 2 - Download WSO2 API-M and WSO2 IS

- Download the [pre-packaged WSO2 Identity Server](#) and unzip it. <IS_HOME> will refer to the root folder of the unzipped WSO2 IS pack.
- Download WSO2 API Manager from [here](#) and unzip it. <APIM_HOME> will refer to the root folder of the unzipped WSO2 API-M pack.

This is also available as a WUM update. For more information, see [Updating WSO2 Products](#).

Step 3 - Optionally, configure port offset for WSO2 API-M or WSO2 IS

This is only required if you are running both WSO2 API Manager and WSO2 Identity Server on the same Virtual Machine (VM).

carbon.xml

```
<Offset>1</Offset>
```

Step 4 - Configure the Identity Server as Key Manager

1. Open the <IS_HOME>/repository/conf/datasources/master-datasources.xml file and add the following datasources.

The WSO2_CARBON_DB datasource points to the local registry, which store important configurations (e.g., registry mounting paths). Therefore, WSO2 recommends that you do not change this

datasource. Simply add the following datasources in the `master-datasources.xml` file. In addition, note that the `WSO2AM_DB` is already added in the `master-datasources.xml` file, so you do not need to add it again. However, you must edit this datasource to point to your new database as this still points to the default H2 database.

In this example `WSO2AM_DB`, `WSO2REG_DB` and `WSO2UM_DB` are included which have changed the datasource pointing MySQL database. Optionally If you want to change other databases such as `WSO2_MB_STORE_DB` and `WSO2AM_STATS_DB` as well to MySQL follow [Changing the Default API-M Databases](#).

For more information on the configuration parameters of a datasource refer [Configuring master-datasources.xml](#)

master-datasources.xml

```
<datasource>
    <name>WSO2AM_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/apimgt?autoReconnect=true&relaxAutoCommit=true&</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>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

<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&</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>

<datasource>
    <name>WSO2UM_DB</name>
    <description>The datasource used for user management</description>
    <jndiConfig>
        <name>jdbc/WSO2UM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

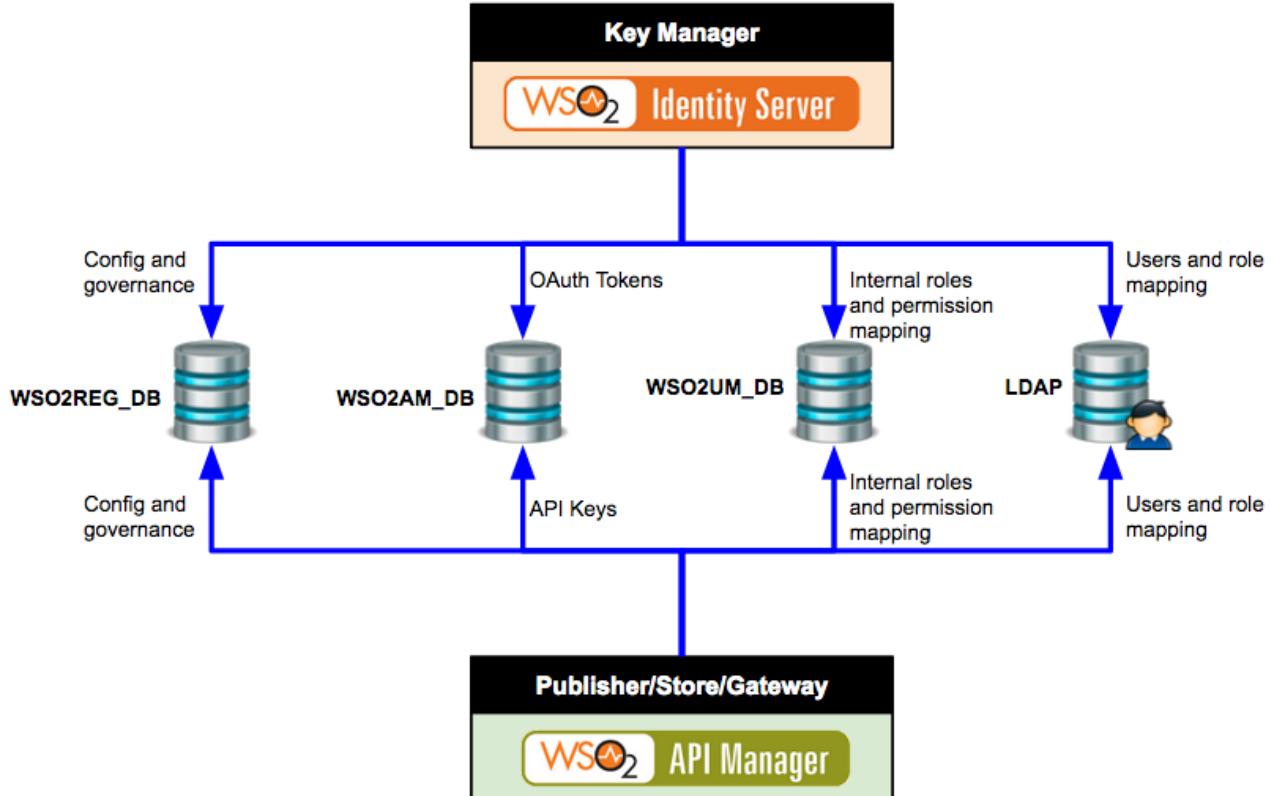
<url>jdbc:mysql://localhost:3306/userstore?autoReconnect=true&relaxAutoCommit=true&;
</url>
        <username>apiuser</username>
        <password>apimanager</password>
        <driverClassName>com.mysql.jdbc.Driver</driverClassName>
        <maxActive>50</maxActive>
        <maxWait>60000</maxWait>
        <testOnBorrow>true</testOnBorrow>
        <validationQuery>SELECT 1</validationQuery>
        <validationInterval>30000</validationInterval>
```

```

        </configuration>
    </definition>
</datasource>

```

The following diagram illustrates how databases are shared between IS and APIM as per the above configuration.



- **WSO2REG_DB** - This is used to keep the registry information. The registry database is shared between WSO2 IS as the Key Manager and WSO2 API-M to share artifacts such as, metadata configurations, policies, and API details.
- **WSO2UM_DB** - This is used to store the permissions (i.e., permission store) and the internal roles of the users.
- **WSO2AM_DB** - This will be used to keep the identity data and API-related data. This includes OAuth tokens and keys. When serving key-validation requests, the key manager validates whether there are subscriptions made by the particular key. For this WSO2AM_DB should be accessed.
- **LDAP** - This stores the users and their role mapping. You do not need to configure the datasource configuration in the master-datasources.xml file for this.

2. Make the following change to the <IS_HOME>/repository/conf/registry.xml file. Create the registry mounts by inserting the following sections into the registry.xml file.

When doing this change, do not replace the existing <dbConfig> for "wso2registry". Simply add the following configuration to the existing configurations. This configuration is related to the local H2 DB, which is used to store the mount information, indexing configuration and anything local to the node, and hence should not be removed even if separate governance and config registries are used.

Cacheld is a unique identification of remote instance. When configuring the remote instance, it is recommended to modify <cacheld> with the corresponding values of your set up as in this format.

```
<username>@<JDBC_URL_to_registry_database>
```

Note that you do not need to change or modify the remoteInstance URL that comes default in above configuration since it is not used because we are not using WS mounting in the latest API Manager versions including this version.

Please refer [Configuring registry.xml](#) for more information on the properties and values of remote mount configuration.

registry.xml

```

<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <cacheId>apiuser@jdbc:mysql://localhost:3306/registry</cacheId>
    <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>

```

3. Change the datasource in the user-mgt.xml file found in the <IS_HOME>/repository/conf/ directory to point to the WSO2UM_DB. This is done to point to the correct database for user management purposes. By default, this configuration is pointing to the embedded H2 database.

user-mgt.xml configurations

```

<Realm>
    <Configuration>
        ...
        <Property name="dataSource">jdbc/WSO2UM_DB</Property>
    </Configuration>
    ...
</Realm>

```

4. Make sure you add the user store configuration correctly in the `<IS_HOME>/repository/conf/user-mgt.xml` file so that both the Identity Server and API Manager point to the same user store. For more information on configuring user stores, see [here](#).

You must change the `<UserStoreManager>` element here since the internal LDAP user store is used by default. The `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">` code block needs to be removed or modified and the right code block must be used. You could alternatively use the embedded LDAP in the Identity Server as your user store. For more information, see [Configuring the Primary User Store](#)

5. Make sure Datasource name in `JDBCPersistenceManager` is `jdbc/WSO2AM_DB` in the `<IS_HOME>/repository/conf/identity/identity.xml` file.

```
<JDBCPersistenceManager>
    <DataSource>
        <Name>jdbc/WSO2AM_DB</Name>
    </DataSource>
    ...
<JDBCPersistenceManager>
```

Configure the Key Manager to enable communication between the Key Manager and the Gateway.

This step is **only applicable if you have enabled Hazelcast clustering** in the API-M Gateway node.

- a. When users and roles are removed via the Key Manager, if the corresponding user tokens are cached on the Gateway, these tokens will only get invalidated when the cache is timed out. However, if Hazelcast clustering is enabled, token invalidation takes place immediately. Therefore, you need to enable communication between the Key Manager and Gateway to enable immediate token invalidation.

For this purpose open the `<IS_HOME>/repository/conf/api-manager.xml` file and change the `<ServerURL>` element that appears under the `<APIGateway>` section, so that it points to the API Manager server.

```
<ServerURL>https://{$gateway-server-host}:{port}/services/</ServerURL>
```

- If you are working with a **single Gateway** in distributed setup, you need to replace `{GATEWAY_SERVER_HOST}` with the host of the **Gateway** node.
 - If you are working with **Gateways** in a **High Availability (HA)** setup that uses **rsync**, you need to replace `{GATEWAY_SERVER_HOST}` with the host of the **Gateway Manager** node.
 - The port value you enter here should be the management transport port. For more information, see [Default Product Ports](#).
- b. When tokens are revoked, the corresponding token cache entries should be cleared in the gateway. For this purpose, open the `<IS_HOME>/repository/conf/api-manager.xml` file and change the `<RevokeAPIURL>` element that appears under the `<OAuthConfigurations>` section, so that it points to the API Manager server, or the gateway worker server if its a distributed setup. Note the port used here is the NIO port, which is by default 8243 for HTTPS.

```
<RevokeAPIURL>https://${gateway-worker-server-host}:{nio-port}/revoke</RevokeAPIURL>
```

If you are using a load balancer front the APIM/Gateway nodes, you can use the load balancer endpoints for the configurations mentioned under #1 and #2 above.

6. Enable communication between Key Manager and Traffic Manager by modifying below configuration in <IS_HOME>/repository/conf/api-manager.xml. Point the server URL and user credentials of Traffic Manager in PolicyDeployer.

```
<PolicyDeployer>
  <ServiceURL>https://<IP_address_of_traffic_manager>:${mgt.transport.port}${carbon.context}services/</ServiceURL>
  <Username>${admin.username}</Username>
  <Password>${admin.password}</Password>
</PolicyDeployer>
```

7. JSON Web Token (JWT) Configuration must be done in the <IS_HOME>/repository/conf/api-manager.xml file in the Identity Server.

JWTs represent a set of claims as a JSON object that is base64url encoded and digitally signed and/or encrypted. For information on JWT Token generation, see [Passing Enduser Attributes to the Backend Using JWT](#).

- Enable the `ClaimsRetrieverImplClass`, `ConsumerDialectURI` and `SignatureAlgorithm` elements by uncommenting it.
- Set `<SignatureAlgorithm>` property value to `NONE` or `SHA256withRSA`.

If you wish to encrypt the Auth Keys (access tokens, client secrets and authorization codes follow [Encrypting OAuth Keys](#) by modifying api-manager.xml in both Identity Server and API Manager products.

Step 5 - Configure the API Manager

1. Point to the WSO2 Identity Server so that it acts as the Key Manager of the API Manager.

When we are using WSO2 Identity Server as Key Manager, WSO2 API Manager should aware the URL of Identity Server to handover the Key validation and Authorization related tasks.

Make the following changes in the <API-M_HOME>/repository/conf/api-manager.xml file under the `<ServerURL>` element that appears under the `<APIKeyValidator>` section and `<AuthManager>` section

- a. Change the `ServerURL` of the `AuthManager` to point to IS.

```
<ServerURL>https://${IS_SERVER_HOST}:{port}/services/</ServerURL>
```

- b. Change the `ServerURL` of the `APIKeyValidator` to point to IS.

```
<ServerURL>https://${IS_SERVER_HOST}:{port}/services/</ServerURL>
```

- c. Change the `<KeyValidatorClientType>` from `ThriftClient` to `WSClient`.

Since API gateway and key manager are two separate components (distributed) they talk to each other via API calls. Out of the box this API call happens via Thrift in WSO2 API Manager. But when it comes to production environment with high availability, it's recommended to switch to WSClient for key validation.

- d. Change <EnableThriftServer> to false to optimize performance.

The following is an example of how this configuration would look like:

```
api-manager.xml
<APIKeyValidator>

<ServerURL>https://localhost:9443{+portoffset}/services/</ServerURL>
    <Username>${admin}</Username>
    <Password>${admin}</Password>
    ...
</APIKeyValidator>
```

For example, in the above configuration if the port offset is 1 the server URL is as follows:

<https://localhost:9444/services/>

2. Open the <API-M_HOME>/repository/conf/datasources/master-datasources.xml file and add the following datasources.

In this example WSO2AM_DB, WSO2REG_DB and WSO2UM_DB are included which have changed the datasource pointing MySQL database. Optionally If you want to change other databases such as WSO2_MB_STORE_DB and WSO2AM_STATS_DB as well to MySQL follow [Changing the Default API-M Databases](#).

```
master-datasources.xml
<datasource>
    <name>WSO2AM_DB</name>
    <description>The datasource used for API Manager database</description>
    <jndiConfig>
        <name>jdbc/WSO2AM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/apimgt?autoReconnect=true&relaxAutoCommit=true</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>
        <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
</definition>
```

```

    </datasource>

    <datasource>
        <name>WSO2REG_DB</name>
        <description>The datasource used for registry and user
manager</description>
        <jndiConfig>
            <name>jdbc/WSO2REG_DB</name>
        </jndiConfig>
        <definition type="RDBMS">
            <configuration>
                <url>jdbc:mysql://localhost:3306/registry?autoReconnect=true&rela
xAutoCommit=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>

    <datasource>
        <name>WSO2UM_DB</name>
        <description>The datasource used for registry and user
manager</description>
        <jndiConfig>
            <name>jdbc/WSO2UM_DB</name>
        </jndiConfig>
        <definition type="RDBMS">
            <configuration>

                <url>jdbc:mysql://localhost:3306/userstore?autoReconnect=true&rel
axAutoCommit=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>

```

```

        </configuration>
    </definition>
</datasource>

```

For more information on the configuration parameters of a datasource refer [Configuring master-datasources.xml](#)

3. Open the `user-mgt.xml` file found in the `<API-M_HOME>/repository/conf` directory and change the permission datasource by adding the configuration below.

user-mgt.xml configurations
<pre> <Realm> <Configuration> ... <Property name="dataSource">jdbc/WSO2UM_DB</Property> </Configuration> ... </Realm> </pre>

4. Configure the `<UserStoreManager>` section of the `<API-M_HOME>/repository/conf/user-mgt.xml` file of the API Manager. Make sure you add the user store configuration correctly. This is the same configuration that you did in the Identity Server and the easiest way to configure this is to copy over the configurations you did in the `<IS_HOME>/repository/conf/user-mgt.xml` file. For more information on how to configure this, see [here](#).

Note: If you are using the embedded LDAP that comes with the IS, then you need to point to the particular LDAP user store from API Manager. You can copy this configuration from the `<IS_HOME>/repository/conf/user-mgt.xml` file to the `<API-M_HOME>/repository/conf/user-mgt.xml` file.

When copying configurations, note that you must update the port numbers. For instance, when configuring the `ConnectionURL` property, you must update the port number as it will point to the port number of the Identity Server when starting up if you copy it directly.

<pre> <Property name="ConnectionURL">ldap://<ip_address_of_IS>:10389</Property> </pre>
--

Note that if you have incremented the port offset of IS, then this port value 10389 should be incremented by the given port offset.

5. Make sure Datasource name in JDBCersistenceManager is `jdbc/WSO2AM_DB` in the `<API-M_HOME>/repository/conf/identity/identity.xml` file.

```
<JDBCPersistenceManager>
    <DataSource>
        <Name>jdbc/WSO2AM_DB</Name>
    </DataSource>
    ...
<JDBCPersistenceManager>
```

6. Create the registry mounts. Open the `<API-M_HOME>/repository/conf/registry.xml` file and insert the following configurations.

Cacheld is a unique identification of remote instance. When configuring the remote instance, it is recommended to modify `<cacheld>` with the corresponding values of your set up as in this format.

```
<username>@<JDBC_URL_to_registry_database>
```

Note that you do not need to change or modify the `remoteInstance URL` that comes default in above configuration since it is not used because we are not using WS mounting in latest API Manager versions including this version.

Please refer [Configuring registry.xml](#) for more information on the properties and values of remote mount configuration.

registry.xml

```
<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <cacheld>root@jdbc:mysql://localhost:3306/registrydb</cacheld>
    <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>
```

7. Run the database scripts provided in `<API-M_HOME>/dbscripts` directory and `<API-M_HOME>/dbscripts/api_mgt` according to the database you use in order to create the tables required. For mysql, the corresponding scripts to be executed are,

- For apimgt database : run `<API-M_HOME>/dbscripts/apimgt/mysql.sql`

- For userstore database : run <API-M_HOME>/dbscripts/mysql.sql
- For registry database : run <API-M_HOME>/dbscripts/mysql.sql

Alternatively, you can start the server with **-Dsetup** in order to create the tables of the databases. Note that if you are using MySQL 5.7 or later version, rename the **mysql5.7.sql** script file to **mysql.sql** to perform the table creation with **-Dsetup**.

Step 6 - Start the servers

Start both the WSO2 API Manager and WSO2 Identity Server in the following order.

1. Start the WSO2 Identity Server for the changes to take effect.
sh <IS_HOME>/bin/wso2server.sh

Tip: You may notice the following error messages when starting up the server. This occurs since some API Manager directories are not available in the Identity Server. These are not critical errors, so they can be ignored. Alternatively, you can create the listed directories in the Identity Server pack.

```
[ 2015-09-26          22:59:20,821]           ERROR
{org.wso2.carbon.apimgt.impl.utils.APIUtil} - Custom sequence template
location unavailable for custom sequence type in :
repository/resources/customsequences/in
[ 2015-09-26          22:59:20,821]           ERROR
{org.wso2.carbon.apimgt.impl.utils.APIUtil} - Custom sequence template
location unavailable for custom sequence type out :
repository/resources/customsequences/out
[ 2015-09-26          22:59:20,821]           ERROR
{org.wso2.carbon.apimgt.impl.utils.APIUtil} - Custom sequence template
location unavailable for custom sequence type fault :
repository/resources/customsequences/fault
```

2. Start the WSO2 API Manager.
sh <API-M_HOME>/bin/wso2server.sh

If you have configured hostnames for API Manager and Identity Server, in the server startup you will see following warning in API Manager backend logs.

```
WARN
{org.wso2.carbon.apimgt.gateway.throttling.util.BlockingConditionRetriever} - Failed retrieving Blocking Conditions from
remote endpoint: sun.security.validator.ValidatorException: PKIX
path building failed:
sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target.
Retrying after 15 seconds...
{org.wso2.carbon.apimgt.gateway.throttling.util.BlockingConditionRetriever}
```

The reason for this is the default certificates that come with WSO2 Servers are created for localhost. When API Manager boots up, it makes an HTTP call to a webapp in Key Manager (throttle data at KM_URL/throttle/data/v1/keyTemplates). For that, API Manager decides the URL of the Key Manager from the URL configured in the api-manager.xml which is localhost.

To overcome this issue, you need to create self-signed certificates for APIM and Identity Server

hostnames. Then export the public certs of API Manager to trust-store.jks of Identity Server and vice versa. This should resolve the SSL handshake failure.

Configuring an External IdP as a Key Manager

WSO2 API Manager allows you to configure an external Identity provider to responsible on the Key Manager related tasks such as security and access token related actions. We are using WSO2 Identity Server as the external Identity Provider in this documentation. By integrating WSO2 Identity Server as Key Manager, we can decouple the operations for creating OAuth applications and token validation while increasing the flexibility in terms of organizational security policies to integrate with WSO2 API Manager. Addition to the Key Manager feature, the users are allowed to consume the identity server specific authentication and authorization features as well as ESB features when we use WSO2 Identity Server as Key Manager. Other than that, it improves the reuse of Identity Server to be used as the Key Manager where it is already used in your distributed setup.

WSO2 recommends that you use the pre-packaged WSO2 Identity Server 5.3.0 with WSO2 API Manager 2.1.0, as most of the configurations already exist in the pre-packaged version. For details, see [Configuring WSO2 Identity Server as a Key Manager](#).

We use **WSO2 Identity Server (WSO2 IS)** as the sample external IdP and **MySQL** as the sample database here. If you wish to use a different external identity provider or database, you can use it by configuring with WSO2 API Manager.

Note that WSO2 IS 5.3.0 is the compatible WSO2 Identity Provider version to configure as the Key Manager for WSO2 API Manager 2.1.0.

Follow the instructions below to configure WSO2 IS the Key Manager:

- Step 1 - Download WSO2 API-M and WSO2 IS
- Step 2 - Optionally, configure port offset for WSO2 API-M or WSO2 IS
- Step 3 - Configure the Identity Server
- Step 4 - Configure the API Manager

Step 1 - Download WSO2 API-M and WSO2 IS

- Download WSO2 Identity Server 5.3.0 from the [Identity Server product page](#) and [install it](#).
- Download WSO2 API Manager from [here](#) and [install it](#).

Step 2 - Optionally, configure port offset for WSO2 API-M or WSO2 IS

This is only required if you are running both WSO2 API Manager and WSO2 Identity Server on the same Virtual Machine (VM).

What is port offset?

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime such as the HTTP/S ports will be offset. For example, if the HTTP port is defined as 9763 and the portOffset is 1, the effective HTTP port will be 9764. Therefore, for each additional WSO2 product, instance, or cluster you add to a server, set the port offset to a unique value (the default is 0).

Open the <PRODUCT_HOME>/repository/conf/carbon.xml file and change the offset to 1. This increments

the product's default port by one. <PRODUCT_HOME> refers to the product to which you are configuring a port offset and it can be either <IS_HOME> or <API-M_HOME>.

carbon.xml

```
<Offset>1</Offset>
```

Step 3 - Configure the Identity Server

1. Sign in to WSO2 Identity Server and access the [Management Console](#).
2. After starting the Identity Server, install the Key Manager feature.

Warning : Installing this feature results in changes to some configuration files in Identity Server. As a result, you will lose the current configurations in the <IS_HOME>/repository/conf/identity.xml and <IS_HOME>/repository/conf/datasources/master-datasources.xml files. Make sure you take a backup of the identity.xml file before making any changes.

To install the feature:

- a. Navigate to the **Features** section in the **Configure** menu of the management console.
- b. Add the following feature repository in the **Feature Management** section in the Identity Server. See [Managing the Feature Repository](#) for information on how to do this.
P2 Repo: <http://product-dist.wso2.com/p2/carbon/releases/wilkes/>
- c. After adding the repository, navigate to the **Available Features** tab and find the feature in that repository by clicking the **Find Features** button. The list of available features appear.

- i. Expand the **API Key Manager** feature from the **Features** category, expand **API Key Manager 6.1.66** and select **Api management Key Manager**.

- ii. Click **Install** and go through the wizard to complete the installation. For more information on how to do this, see [Installing Features](#).
3. Make the following changes in the <IS_HOME>/repository/conf/api-manager.xml file.
 - a. Change the **GatewayType** property to the following. You need to do this because the default value here is Synapse. As the Synapse runtime is used for

various ESB related functionality that is not available in the Identity Server, you must change this to No
 n
 e
<GatewayType>None</GatewayType>

- b. Configure the Key Manager to enable communication between the Key Manager and the Gateway.

This step is **only applicable if you have enabled Hazelcast clustering** in the API-M Gateway node.

- i. When users and roles are removed via the Key Manager, if the corresponding user tokens are cached on the Gateway, these tokens will only get invalidated when the cache is timed out. However, if Hazelcast clustering is enabled, token invalidation takes place immediately. Therefore, you need to enable communication between the Key Manager and Gateway to enable immediate token invalidation.

For this purpose open the <IS_HOME>/repository/conf/api-manager.xml file and change the <ServerURL> element that appears under the <APIGateway> section, so that it points to the API Manager server.

```
<ServerURL>https:// ${gateway-server-host} : {port} /services /<
/ServerURL>
```

- If you are working with a **single Gateway** in distributed setup, you need to replace {GATEWAY_SERVER_HOST} with the host of the **Gateway** node.
- If you are working with **Gateways** in a **High Availability (HA)** setup that uses **rsync**, you need to replace {GATEWAY_SERVER_HOST} with the host of the **Gateway Manager** node.
- The port value you enter here should be the management transport port. For more information, see [Default Product Ports](#).

- ii. When tokens are revoked, the corresponding token cache entries should be cleared in the gateway. For this purpose, open the <IS_HOME>/repository/conf/api-manager.xml file and change the <RevokeAPIURL> element that appears under the <OAuthConfiguration>s> section, so that it points to the API Manager server, or the gateway worker server if its a distributed setup. Note the port used here is the NIO port, which is by default 8243 for HTTPS.

```
<RevokeAPIURL>https:// ${gateway-worker-server-host} : {nio-po
rt} /revoke</RevokeAPIURL>
```

If you are using a load balancer front the APIM/Gateway nodes, you can use the load balancer endpoints for the configurations mentioned under #1 and #2 above.

- c. Change EnableThriftServer to false.
 The Identity Server does not come with a Thrift server and this causes issues at runtime if not disabled.
<EnableThriftServer>false</EnableThriftServer>

4. Open the <IS_HOME>/repository/conf/datasources/master-datasources.xml file and add the following datasources.

Ensure that you do not change the 'WSO2_CARBON_DB' datasource and simply add the following datasources in the master-datasources.xml file.

Note that the `WSO2AM_DB` is added in the `master-datasources.xml` file by default. However, you must edit this datasource to point to your new database as this still points to the default H2 database. The following code includes a sample of the `WSO2AM_DB` datasource as a sample configuration when pointing to the new database.

In this example, `WSO2AM_DB`, `WSO2REG_DB` and `WSO2UM_DB` are included, and the datasources have been changed to point to the MySQL database. Optionally, you can change other databases such as the `WSO2_MB_STORE_DB` and `WSO2AM_STATS_DB` as well to point to the MySQL database by following [Changing the Default API-M Databases](#).

For more information on the configuration parameters of a datasource, see [Configuring master-datasources.xml](#) in the WSO2 Administration Guide.

```
<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>
            <url>jdbc:mysql://localhost:3306/apimgt?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>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

<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>

<datasource>
<name>WSO2UM_DB</name>
<description>The datasource used for user management</description>
<jndiConfig>
    <name>jdbc/WSO2UM_DB</name>
</jndiConfig>
<definition type="RDBMS">
    <configuration>

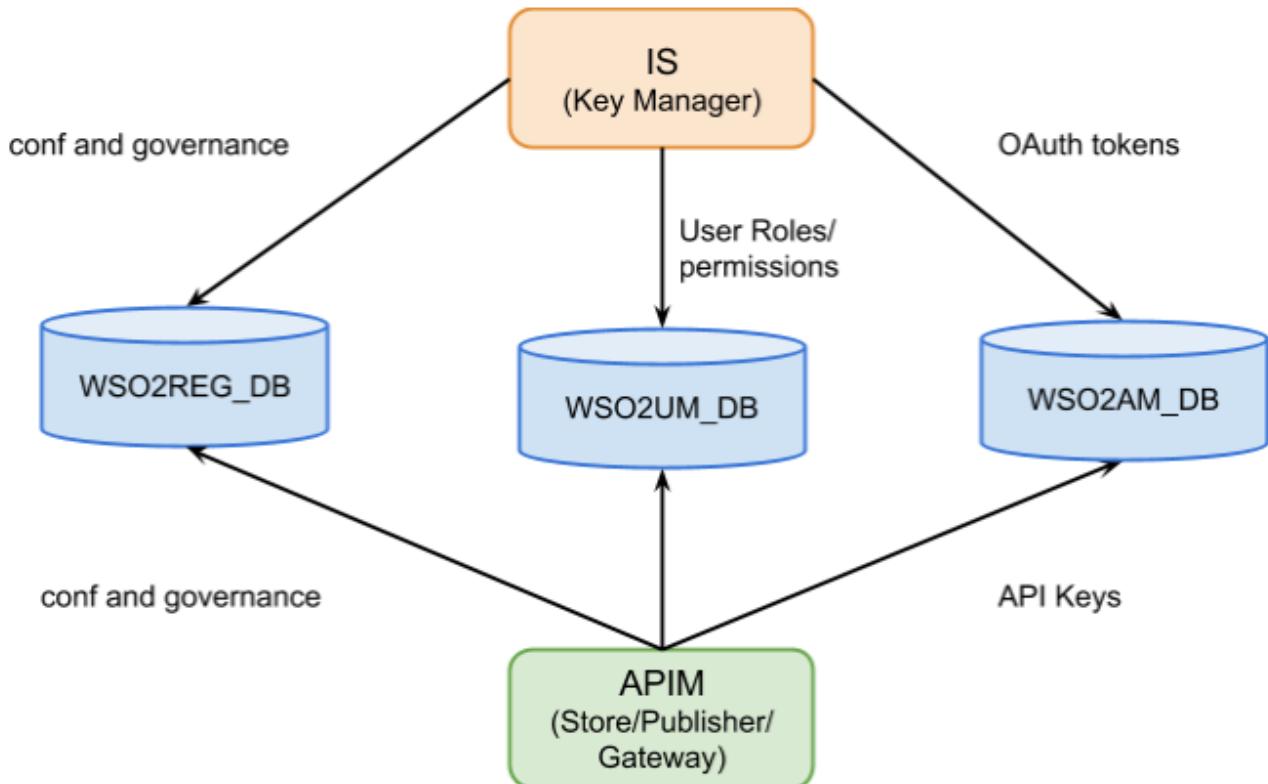
        <url>jdbc:mysql://localhost:3306/userstore?autoReconnect=true&relaxAutoCommit=true&
        </url>
        <username>apiuser</username>
        <password>apimanager</password>
        <driverClassName>com.mysql.jdbc.Driver</driverClassName>
        <maxActive>50</maxActive>
        <maxWait>60000</maxWait>
        <testOnBorrow>true</testOnBorrow>
        <validationQuery>SELECT 1</validationQuery>
        <validationInterval>30000</validationInterval>
```

```

        </configuration>
    </definition>
</datasource>

```

The following diagram illustrates how databases are shared between IS and API-M as per the above configuration.



- **WSO2REG_DB** - This DB maintains the registry information. The registry database is shared between WSO2 IS as the Key Manager and WSO2 API-M to share artifacts such as, metadata configurations, policies, and API details.
- **WSO2UM_DB** - This DB maintains to Store the permissions (i.e. permission store) and the internal roles of the users.
- **WSO2AM_DB** - This DB maintains the identity data and API-related data. This includes OAuth tokens and keys. When serving key-validation requests, the Key Manager access the WSO2AM_DB to validate whether there are subscriptions made by the particular key.

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.

Cacheld is a unique identification of remote instance. When configuring the remote instance, it is recommended to modify <cacheld> with the corresponding values of your setup as in this format.

```
<username>@<JDBC_URL_to_registry_database>
```

Note that you do not need to specify the remoteInstance URL in above configuration since it is not used because we are not using WS mounting in latest API Manager versions including this version.

For more information on the properties and values of remote mount configuration, see [Configuring registry.xml](#).

```
<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <cacheId>apiuser@jdbc:mysql://localhost:3306/registry</cacheId>
    <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>
```

1. Change the datasource in the `<IS_HOME>/repository/conf/user-mgt.xml` file to `WSO2UM_DB`.

user-mgt.xml configurations
<pre><Realm> <Configuration> ... <Property name="dataSource">jdbc/WSO2UM_DB</Property> </Configuration> ... </Realm></pre>

2. Add the user store configuration correctly in the `<IS_HOME>/repository/conf/user-mgt.xml` file so that both WSO2 Identity Server and WSO2 API Manager point to the same user store. For more information on configuring user stores, see [Configuring the Realm](#) in the WSO2 IS documentation.

You must change the `<UserStoreManager>` element here since the internal LDAP user store is used by default. You need to remove or modify the `<UserStoreManager class="org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager">` code block so that you use the right code block.

3. Modify the datasource name under `<JDBCPersistenceManager>` to `jdbc/WSO2AM_DB` in the `<IS_HOME>/repository/conf/identity/identity.xml` file.

```
<JDBCPersistenceManager>
    <DataSource>
        <Name>jdbc/WSO2AM_DB</Name>
    </DataSource>
    ...
<JDBCPersistenceManager>
```

4. Create tables and indexes for the following databases in the MySQL database server.

- userstore
- registry
- apimgt

To create tables and indexes for the userstore and registry database, use the <IS_HOME>/dbscripts/mysql.sql script.

Note that MySQL is used as an example here and you can use a different database if required.

When creating the apimgt DB, run the following script; <API-M_HOME>/dbscripts/apimgt/mysql.sql. The script in the <API-M_HOME>/dbscripts/apimgt/ directory has all the tables required to manage OAuth access tokens and also includes other identity-related features.

5. Create a user named apiuser with the password apimanager. Grant all permissions for this user in the above three databases. For example:

```
grant all on apimgt.* TO apiuser@localhost identified by "apimanager";
grant all on userstore.* TO apiuser@localhost identified by
"apimanager";
grant all on registry.* TO apiuser@localhost identified by
"apimanager";
```

6. Enable communication between Key Manager and Traffic Manager by modifying below configuration in <IS_HOME>/repository/conf/api-manager.xml. Point the server URL and user credentials of Traffic Manager in PolicyDeployer.

```
<PolicyDeployer>
    <ServiceURL>https://<IP_address_of_traffic_manager>:${mgt.transport.
https.port}${carbon.context}services/</ServiceURL>
    <Username>${admin.username}</Username>
    <Password>${admin.password}</Password>
</PolicyDeployer>
```

7. Configure the JSON Web Token (JWT) . For more information on JWT Token generation, see [Passing Enduser Attributes to the Backend Using JWT](#).

- Make the following changes in the <IS_HOME>/repository/conf/api-manager.xml file in the Identity Server.
 - Enable the ClaimsRetrieverImplClass, ConsumerDialectURI and SignatureAlgorithm by uncommenting the respective elements.
 - Set SignatureAlgorithm to NONE or SHA256withRSA.
- Add the following under the <OAuth> element in the <IS_HOME>/repository/conf/identity/identity.xml file. This is done to validate the scripts before issuing the token.

```
<OAuthScopeValidator
class="org.wso2.carbon.identity.oauth2.validators.JDBCScopeValidator"/>
```

8. Start the server for the changes to take effect.

Step 4 - Configure the API Manager

1. Open the <API-M_HOME>/repository/conf/datasources/master-datasources.xml file and add the following datasources.

For more information on the configuration parameters of a datasource, see [Configuring master-datasources.xml](#) in the WSO2 Administration Guide.

In this example WSO2AM_DB, WSO2REG_DB and WSO2UM_DB are included which have changed the datasource pointing MySQL database. Optionally If you want to change other databases such as WSO2_MB_STORE_DB and WSO2AM_STATS_DB as well to MySQL follow [Changing the Default API-M Databases](#).

```
<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>
            <url>jdbc:mysql://localhost:3306/apimgt?autoReconnect=true&amp;relaxAutoCommit=true&amp;</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>
        <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used for registry and user manager</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
```

```
</jndiConfig>
<definition type="RDBMS">
    <configuration>
<url>jdbc:mysql://localhost:3306/registry?autoReconnect=true&rela
xAutoCommit=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>

<datasource>
    <name>WSO2UM_DB</name>
    <description>The datasource used for registry and user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2UM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

<url>jdbc:mysql://localhost:3306/userstore?autoReconnect=true&rel
axAutoCommit=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>

```

Open the **user-mgt.xml** file found in the `<API-M_HOME>/repository/conf` directory and change the permission datasource.

- Add the datasource configuration as shown below.

user-mgt.xml configurations

```

<Realm>
    <Configuration>
        ...
        <Property name="dataSource">jdbc/WSO2UM_DB</Property>
    </Configuration>
    ...
</Realm>

```

- Configure the `<UserStoreManager>` section of the `<API-M_HOME>/repository/conf/user-mgt.xml` file of the API Manager.

Make sure you add the user store configuration correctly. This is the same configuration that you did in the Identity Server. For more information on how to do this, see [here](#).

- Make sure the datasource name that is defined under `<JDBCPersistenceManager>` is `jdbc/WSO2AM_DB` in the `<API-M_HOME>/repository/conf/identity/identity.xml` file.

```

<JDBCPersistenceManager>
    <DataSource>
        <Name>jdbc/WSO2AM_DB</Name>
    </DataSource>
    ...
<JDBCPersistenceManager>

```

- Create the registry mounts. Open the `<API-M_HOME>/repository/conf/registry.xml` file and insert the following sections.

`Cacheld` is a unique identification of remote instance. When configuring the remote instance, it is recommended to modify `<cacheld>` with the corresponding values of your setup as in this format.

```
<username>@<JDBC_URL_to_registry_database>
```

Note that you do not need to specify the `remoteInstance URL` in above configuration since it is not used because we are not using WS mounting in latest API Manager versions including this version.

- For more information on the properties and values of remote mount configuration, see [Configuring registry.xml](#).

```

<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>

<remoteInstance url="https://localhost">
    <id>gov</id>
    <dbConfig>govregistry</dbConfig>
    <cacheId>apiuser@jdbc:mysql://localhost:3306/registry</cacheId>
    <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>

```

5. Open the `api-manager.xml` file, which is in the `<API-M_HOME>/repository/conf` directory, and change the following.

- Change the `ServerURL` of the `AuthManager` to point to WSO2 IS.

```
<ServerURL>https://${IS_SERVER_HOST}:{port}/services/ </ServerURL>
```

- Change the `ServerURL` of the `APIKeyValidator` to point to WSO2 IS.

```
<ServerURL> https://${IS_SERVER_HOST}:{port}/services/ </ServerURL>
```

- Change the `KeyValidatorClientType` from `ThriftClient` to `WSClient`.

- Change `EnableThriftServer` to `false`.

Make sure you add the relevant JDBC driver to both servers (i.e., put the `.jar` file into the `<PRODUCT_HOME>/repository/components/lib` directory).

Deploying API Manager with Kubernetes or OpenShift Resources

Follow the instructions below to use Kubernetes (K8s) or Openshift resources for container-based deployments of WSO2 API Manager (API-M).

In the context of this document, `<KUBERNETES_HOME>` refers to a local copy of the [wso2/kubernetes-apim](#) Git repository that **includes common resources that can be used for Kubernetes and OpenShift**.

- Checkout the WSO2 kubernetes-apim repository using `git clone`:

```

git clone https://github.com/wso2/kubernetes-apim.git
git checkout tags/v2.1.0-2

```

2. Pull required Docker images from the **WSO2 Docker Registry** using `docker pull`:

```
docker login docker.wso2.com

docker pull docker.wso2.com/wso2am-analytics-kubernetes:2.1.0
docker pull docker.wso2.com/wso2am-kubernetes:2.1.0
docker pull docker.wso2.com/apim-rdbms-kubernetes:2.1.0
```

You can also build the Docker images by following the guide in the <KUBERNETES_HOME>/base/[RE ADME.md](#). In addition, note that the same images can be used for OpenShift.

3. Copy the Images into Kubernetes/Openshift nodes or to a Registry. Copy the required Docker images over to the Kubernetes Nodes. For example:

- a. Use `docker save` to create a TAR file of the required image.
 - b. `scp` the TAR file to each node.
 - c. Use `docker load` to load the image from the copied TAR file on the nodes.
- Alternatively, if a private Docker registry is used, transfer the images there.

4. Deploy Kubernetes/Openshift Resources.

Before you begin the deployment, make sure that you have the following prerequisites.

- Set up Network File System (NFS) to deploy any pattern. NFS is used as the persistent volume for API Manager servers. As a result, setting up NFS is required to deploy any pattern. Therefore, you need to complete the following:
 - a. Update the NFS server IP in <KUBERNETES_HOME>/pattern-X/artifacts/volumes/persistent-volumes.yaml
 - b. Create the required directories in the NFS server for each pattern as mentioned in the <KUBERNETES_HOME>/pattern-X/artifacts/volumes/persistent-volumes.yaml
 For example, for pattern-1, create the directories as /exports/pattern-1/apim
- It is recommended to use a MySQL or any database cluster in a production environment. Only one MySQL container is used with host path mount in these deployments.

5. Deploy Kubernetes/Openshift Resources:

The following instructions have been **tested on OpenShift v3.6.0 and Kubernetes v1.6.1 and NFS is tested in Kubernetes v1.6.1**.

- [Deploy a pattern on Kubernetes](#)
- [Deploy a pattern on OpenShift](#)

Deploy a pattern on Kubernetes

- a. Create a namespace named wso2.

```
kubectl create namespace wso2
```

- b. Create a service account named wso2svcacct in the wso2 namespace.

```
kubectl create serviceaccount wso2svcacct -n wso2
```

- c. Deploy any pattern by running the `deploy-kubernetes.sh` script that is inside the pattern folder (`<KUBERNETES_HOME>/pattern-X/` directory).

```
./deploy-kubernetes.sh
```

- d. Access the management console using the following command to list ingresses in the deployment.

```
kubectl get ingress
```

Add relevant hosts and IP addresses to the `/etc/hosts` file. The following are sample access URLs. However, note that this will varies based on the pattern that you are using.

- <https://wso2apim>
- <https://wso2apim-analytics>
- <https://wso2apim-gw>

If required, **undeploy a pattern on Kubernetes**

You can undeploy any pattern by running `undeploy-kubernetes.sh` script that is inside the pattern folder (`<KUBERNETES_HOME>/pattern-X/` directory).

```
./undeploy-kubernetes.sh
```

Deploy a pattern on Openshift

- a. Create a user named `admin` and assign the user to the `cluster-admin` role. This user with the `cluster-admin` role is used to deploy the OpenShift artifacts.

```
oc login -u system:admin
oc create user admin --full-name=admin
oc adm policy add-cluster-role-to-user cluster-admin admin
```

- b. Create a new project named `wso2`.

```
oc new-project wso2 --description="WSO2 API Manager 2.1.0"
--display-name="wso2"
```

Create a service account named `wso2svcacct` in the `wso2` project and assign it the `anyuid` security context constraint.

```
oc create serviceaccount wso2svcacct
oc adm policy add-scc-to-user anyuid -z wso2svcacct -n wso2
```

- c. Deploy any pattern by running the `deploy-openshift.sh` script inside the pattern folder (`<KUBERNETES_HOME>/pattern-X/` directory).

```
./deploy-openshift.sh
```

- d. Access the Management Console using the following command to list the routes in the deployment.

```
oc get routes
```

Add relevant hosts and IP addresses to the `/etc/hosts` file.

The following are sample access URLs. Note that this varies based on the pattern that you are using.

- <https://wso2apim>
- <https://wso2apim-analytics>
- <https://wso2apim-gw>

If required, **undeploy a pattern on OpenShift**

You can undeploy any pattern by running the `undeploy-openshift.sh` script that is inside the pattern folder (`<KUBERNETES_HOME>/pattern-X/directory`).

```
./undeploy-openshift.sh
```

6. Customize the deployment (If required).

Configurations are bound with the `wso2` namespace. Therefore, if you are changing the hostnames or the namespace, do the following:

- a. Change `wso2.svc` to `<namespace>.svc` in all the configuration files.
- b. Update the `KUBERNETES_NAMESPACE` parameter with the correct namespace in all the `<API-M_HOME>/repository/conf/axis2/axis2.xml` files.
- c. Update Docker base images.
Use a CA signed certificate and update the `client-truststore.jks` and `wso2carbon.jks` files which are in the following location.
<https://github.com/wso2/kubernetes-apim/tree/2.1.0-nfs/base/apim/files>.

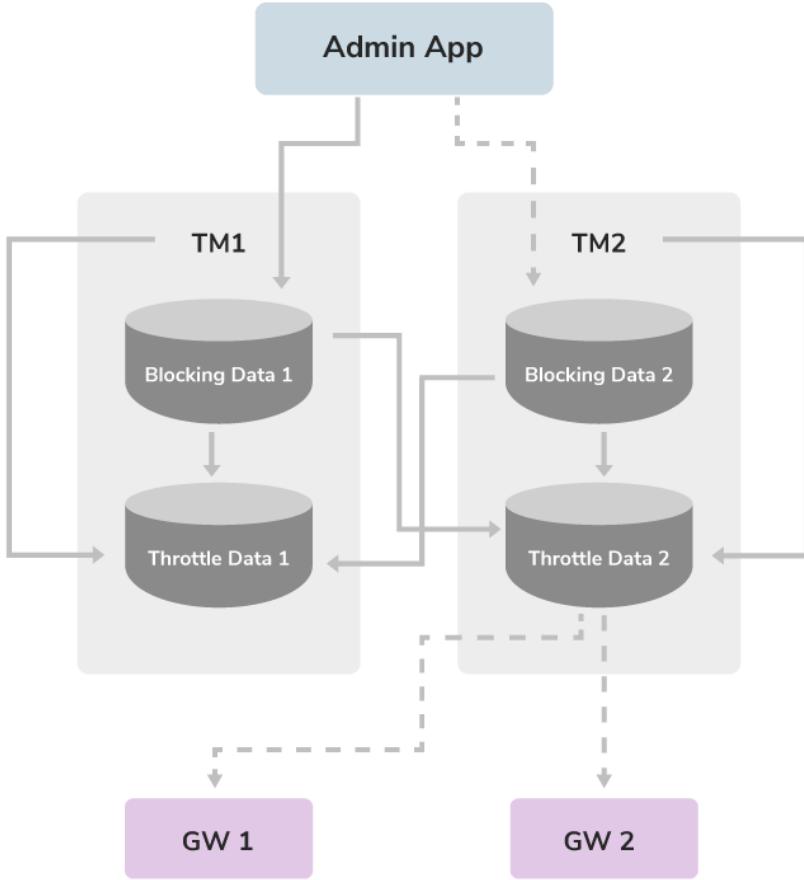
Configuring Admin App Event Publishing for Traffic Manager HA Setup

The Admin App, which is accessed via the Admin Portal (<https://<APIM-host>:<APIM-port>/admin>) is generally setup in the Publisher node when WSO2 API Manager (WSO2 API-M) is deployed in a distributed setup.

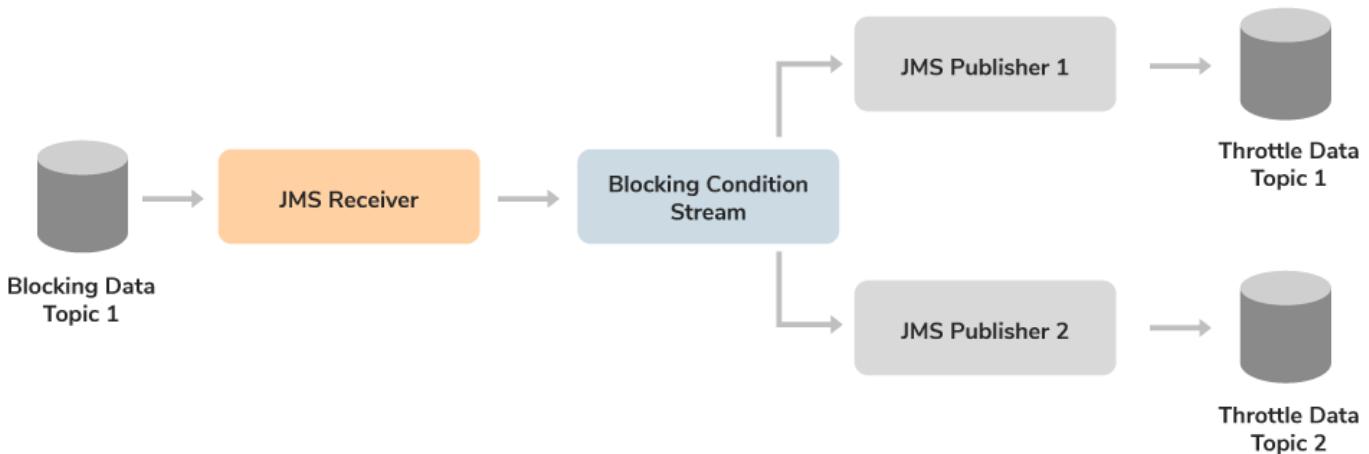
When the Traffic Manager (TM) is set up for high availability (HA), you need to configure the Admin App to publish events to the `blockingData` topic in a failover manner to enable event publishing in the Admin App. When the `blockingData` topic receives the message from the Admin App, it publishes the messages to the `throttleData` topics in both the Traffic Managers, and the Gateways receive the messages from `throttleData` topics.

If there is a failure in Traffic Manager 1 (TM1), the Admin App starts publishing to the `blockingData` event in the other Traffic Manager (`blockingData2`). When TM1 is restored, `blockingData2` publishes the latest events to `throttleData1`. As a result, the Gateways that are connected to TM1 always receive the latest changes with

regard to the blacklist conditions.



Furthermore, as illustrated in the diagram below a JMS receiver listens to the `blockingData` topic. When the JMS receiver receives an event, it sends it through an event stream and publishes it through the attached JMS publishers to the `throttleData` topics.



Follow the instructions below to configure Admin App event publishing when the Traffic Managers are setup for high availability.

Step 1 - Configure the Admin App

1. Change the topic name to `blockingData` in the Event publisher configurations, which is in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

api-manager.xml

```
<JMSEventPublisherParameters>

<java.naming.factory.initial>org.wso2.andes.jndi.PropertiesFileInitia
lContextFactory</java.naming.factory.initial>

<java.naming.provider.url>repository/conf/jndi.properties</java.namin
g.provider.url>

<transport.jms.DestinationType>topic</transport.jms.DestinationType>
<transport.jms.Destination>blockingData</transport.jms.Destination>

<transport.jms.ConcurrentPublishers>allow</transport.jms.ConcurrentPu
blishers>

<transport.jms.ConnectionFactoryJNDIName>TopicConnectionFactory</tran
sport.jms.ConnectionFactoryJNDIName>
</JMSEventPublisherParameters>
```

- Configure the blockingData topic in the <API-M_HOME>/repository/conf/jndi.properties file.

jndi.properties

```
connectionfactory.TopicConnectionFactory =
amqp://admin:admin@clientID/carbon?failover='roundrobin'%26cyclecount
='2'%26brokerlist='tcp://127.0.0.1:5674?retries='5'%26connectdelay='5
0';tcp://127.0.0.1:5675?retries='5'%26connectdelay='50'
topic.blockingData = blockingData
```

Step 2 - Configure the Traffic Manager nodes

You need to carry out the following configurations in both the Traffic Manager nodes in order to send the JMS messages from the blockingData topic to the throttleData topics.

- Add a JMS receiver.
Create a file named blockingEventReceiver.xml with the following content, and move it to the <API-M_HOME>/repository/deployment/server/eventreceivers directory, to add a new event receiver.

blockingEventReceiver.xml

```

<eventReceiver xmlns="http://wso2.org/carbon/eventreceiver"
name="blockingEventReceiver" statistics="enable" trace="enable">
    <from eventAdapterType="jms">
        <property name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Destination">blockingData</property>
        <property
name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFile
InitialContextFactory</property>
        <property
name="java.naming.provider.url">repository/conf/jndi2.properties</pro
perty>
        <property
name="transport.jms.SubscriptionDurable">false</property>
        <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory
</property>
    </from>
    <mapping customMapping="disable" type="map" />
    <to streamName="org.wso2.blocking.condition.stream" version="1.0.0"
/>
</eventReceiver>

```

2. Configure the blocking event receiver.
Create a file named `jndi2.properties` in the `<API-M_HOME>/repository/conf` directory with the following content.

jndi2.properties

```

connectionfactory.TopicConnectionFactory =
amqp://admin:admin@clientid/carbon?brokerlist='tcp://localhost:5674'
connectionfactory.QueueConnectionFactory =
amqp://admin:admin@clientID/test?brokerlist='tcp://localhost:5674'
topic.blockingData = blockingData

```

Changing the admin password

To change the admin **password** go to [Changing the super admin password](#). See the note given under step 2 for instructions to follow if your password has special characters.

You need to add this configuration so that the JMS receiver namely `blockingEventReceiver` can listen to the local `blockingData` topic.

3. Add an event stream.
Create a file named `org.wso2.blocking.condition.stream_1.0.0.json` in the `<API-M_HOME>/re`

pository/deployment/server/eventstreams directory.

org.wso2.blocking.condition.stream_1.0.0.json

```
{
    "name": "org.wso2.blocking.condition.stream",
    "version": "1.0.0",
    "nickName": "",
    "description": "",
    "payloadData": [
        {
            "name": "blockingCondition",
            "type": "STRING"
        },
        {
            "name": "conditionValue",
            "type": "STRING"
        },
        {
            "name": "state",
            "type": "STRING"
        },
        {
            "name": "tenantDomain",
            "type": "STRING"
        },
        {
            "name": "keyTemplateValue",
            "type": "STRING"
        },
        {
            "name": "keyTemplateState",
            "type": "STRING"
        }
    ]
}
```

4. Add JMS publishers to publish blocking data.

- Add a JMS publisher to publish events from **blockingData1** to **throttleData1**. Create a file named **blockingconditionpublisher1.xml** in the <API-M_HOME>/repository/deployment/server/eventpublishers directory, to add an event publisher.

blockingconditionpublisher1.xml

```

<eventPublisher xmlns="http://wso2.org/carbon/eventpublisher"
name="blockingconditionpublisher1" processing="enable"
statistics="disable" trace="enable">
    <from streamName="org.wso2.blocking.condition.stream"
version="1.0.0" />
    <mapping customMapping="disable" type="map" />
    <to eventAdapterType="jms">
        <property
name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Destination">throttleData</property>
        <property
name="transport.jms.ConcurrentPublishers">allow</property>
        <property
name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
        <property
name="java.naming.provider.url">repository/conf/jndi.properties</property>
        <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    </to>
</eventPublisher>

```

- b. Add a JMS publisher to publish events from **blockingData1** to **throttleData2**. Create a file named `blockingconditionpublisher2.xml` in the `<API-M_HOME>/repository/deployment/server/eventpublishers` directory, to add another event publisher.

blockingconditionpublisher2.xml

```

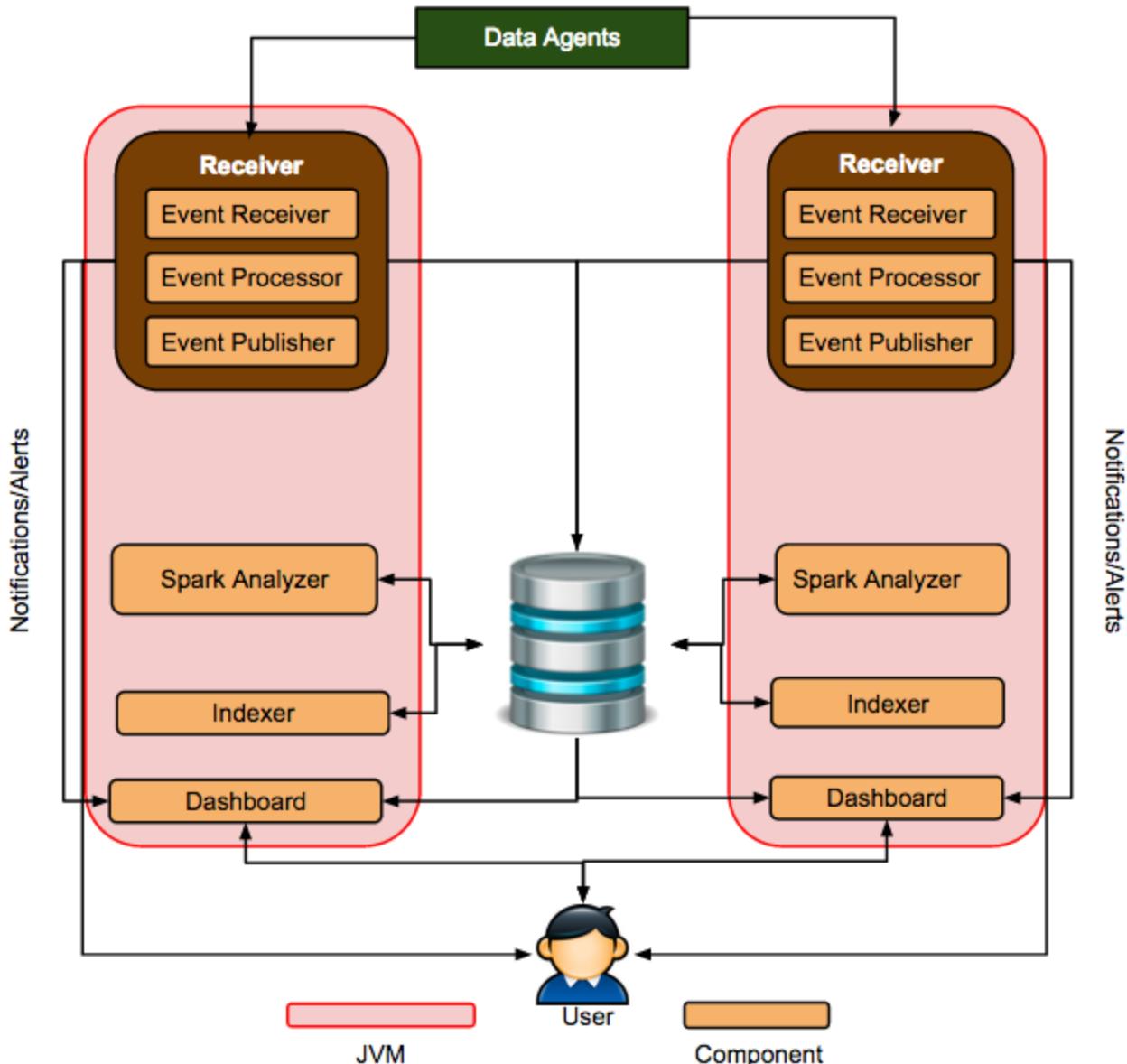
<eventPublisher xmlns="http://wso2.org/carbon/eventpublisher"
name="blockingconditionpublisher2" processing="enable"
statistics="disable" trace="enable">
    <from streamName="org.wso2.blocking.condition.stream"
version="1.0.0" />
    <mapping customMapping="disable" type="map" />
    <to eventAdapterType="jms">
        <property
name="transport.jms.DestinationType">topic</property>
        <property
name="transport.jms.Destination">throttleData</property>
        <property
name="transport.jms.ConcurrentPublishers">allow</property>
        <property
name="java.naming.factory.initial">org.wso2.andes.jndi.PropertiesFileInitialContextFactory</property>
        <property
name="java.naming.provider.url">repository/conf/jndi3.properties
</property>
        <property
name="transport.jms.ConnectionFactoryJNDIName">TopicConnectionFactory</property>
    </to>
</eventPublisher>

```

- Make sure that you add the correct JNDI configurations to point to the correct message topic.
- You need to carry out all the Traffic Manger related configurations mentioned in step 2 in the other Traffic Manager node (TM2) as well.

Minimum High Availability Deployment for WSO2 APIM Analytics

This section explains how to configure WSO2 API Manager Analytics in a distributed setup. You can configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and to carry out detailed analysis of logs relating to the APIs. WSO2 APIM Analytics is powered by WSO2 DAS. The following diagram indicates the minimum deployment pattern used for high availability.



APIM Analytics supports a deployment scenario that has focus on high availability (HA) along with HA processing. To enable HA processing, you should have two APIM Analytics servers in a cluster.

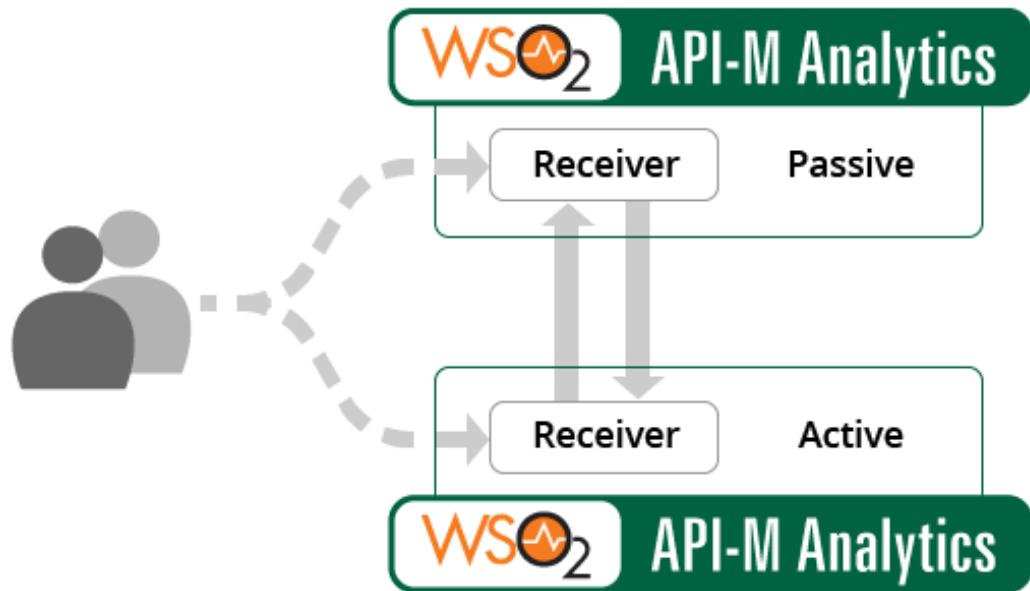
For this deployment, both nodes should be configured to receive all events. To achieve this, clients can either send all the requests to both the nodes or each request to any one of the two nodes (i.e., using load balancing or failover mechanisms). If clients send all the requests to both nodes, the user has to specify that events are duplicated in the cluster (i.e., the same event comes to all the members of the cluster). Alternatively, if a client sends a request to one node, internally it sends that particular request to the other node as well. This way, even if the clients send requests to only one node, both API-M Analytics nodes receive all the requests.

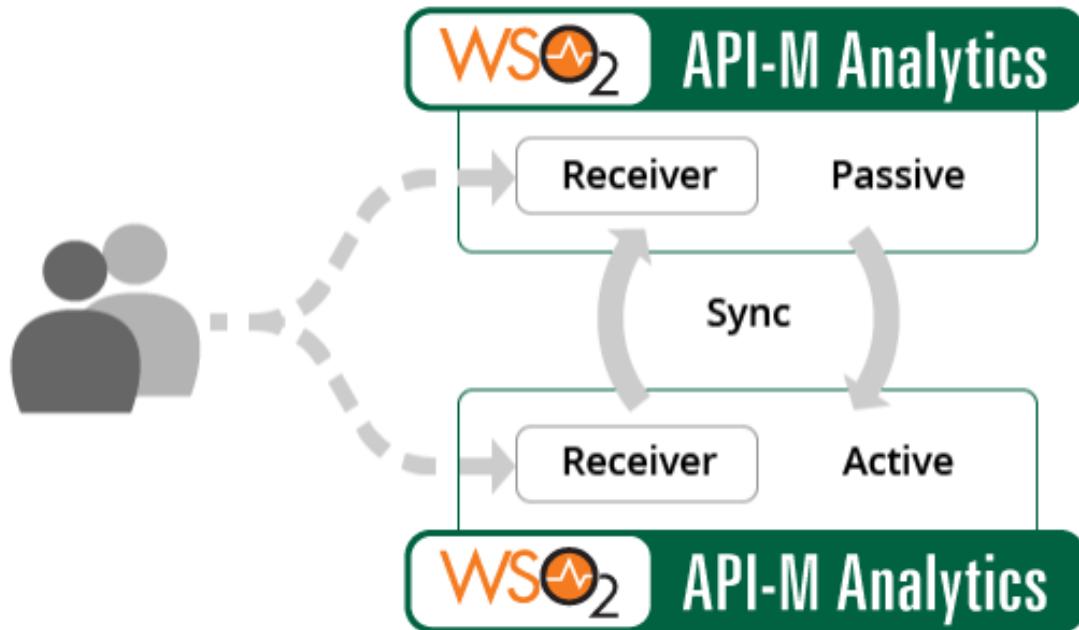
In this scenario, one API-M Analytics node works in active mode and the other works in passive mode. However, both nodes process all the data.

If the active node fails, the other node becomes active and receives all the requests.



When the failed node is up again, it fetches all the internal states of the current active node via synching.





The newly arrived node then becomes the passive node and starts processing all the incoming messages to keep its state synched with the active node so that it can become active if the current active node fails.

Warning: Some of the requests may be lost during the time the passive node switches to the active mode.

Prerequisites

Before you configure a minimum high availability API-M Analytics cluster, the following needs to be carried out.

1. Download the WSO2 API-M Analytics distribution. Click **DOWNLOAD ANALYTICS** in the [WSO2 API Manager page](#).
2. Take the following steps to install WSO2 APIM Analytics. Because this procedure is identical to installing WSO2 Data Analytics Server (DAS), these steps take you to the DAS documentation for details.
 - a. Ensure that you have met the [Installation Prerequisites](#).
 - b. Go to the installation instructions relevant to your operating system:
 - [Installing on Linux](#)
 - [Installing on Windows](#)
 - [Installing as a Windows Service](#)
 - [Installing as a Linux Service](#)
3. Follow the steps below to set up MySQL.
 - a. Download and install [MySQL Server](#).
 - b. Download the [MySQL JDBC driver](#).
 - c. Unzip the downloaded MySQL driver zipped archive, and copy the MySQL JDBC driver JAR (`mysql-connector-java-x.x.xx-bin.jar`) into the `<APIM Analytics_HOME>/repository/components/lib` directory of all the nodes in the cluster.
 - d. Enter the following command in a terminal/command window, where `username` is the username you want to use to access the databases.
`mysql -u username -p`
 - e. When prompted, specify the password that will be used to access the databases with the `username` you specified.
 - f. Create two databases named `userdb` and `regdb`.

About using MySQL in different operating systems

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 following is how your database creation command should look.

```
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 following is how your database creation command should look.

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

- g. Execute the following script for the two databases you created in the previous step.

```
mysql> source <APIM Analytics_HOME>/dbscripts/mysql.sql;
```

From WSO2 Carbon Kernel 4.4.6 onwards there are two MySQL DB scripts available in the product distribution. Click [here](#) to identify as to which version of the MySQL script to use.

- ▼ [Click here to view the commands for performing steps f and g](#)

```
mysql> create database userdb;
mysql> use userdb;
mysql> source <APIM Analytics_HOME>/dbscripts/mysql.sql;
mysql> grant all on userdb.* TO username@localhost identified
by "password";
```

```
mysql> create database regdb;
mysql> use regdb;
mysql> source <APIM Analytics_HOME>/dbscripts/mysql.sql;
mysql> grant all on regdb.* TO username@localhost identified
by "password";
```

- h. Create the following databases in MySQL.

- WSO2_ANALYTICS_EVENT_STORE_DB
- WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB

It is recommended to create the databases with the same names given above because they are the default JNDI names that are included in the <APIM Analytics_HOME>/repository/conf/analytics/analytics-conf.xml file as shown in the extract below. If you change the name, the analytics-conf.xml file should be updated with the changed name.

```

<analytics-record-store name="EVENT_STORE">

    <implementation>org.wso2.carbon.analytics.datasource.rdbms
        .RDBMSAnalyticsRecordStore</implementation>
        <properties>
            <property
                name="datasource">WSO2_ANALYTICS_EVENT_STORE_DB</property>
            <property
                name="category">read_write_optimized</property>
        </properties>
    </analytics-record-store>
    <analytics-record-store name="EVENT_STORE_WO">

        <implementation>org.wso2.carbon.analytics.datasource.rdbms
            .RDBMSAnalyticsRecordStore</implementation>
            <properties>
                <property
                    name="datasource">WSO2_ANALYTICS_EVENT_STORE_DB</property>
                    <property name="category">write_optimized</property>
            </properties>
        </analytics-record-store>
        <analytics-record-store name="PROCESSED_DATA_STORE">

            <implementation>org.wso2.carbon.analytics.datasource.rdbms
                .RDBMSAnalyticsRecordStore</implementation>
                <properties>
                    <property
                        name="datasource">WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB</
                        property>
                    <property
                        name="category">read_write_optimized</property>
                </properties>
            </analytics-record-store>

```

Required configurations

When configuring the minimum high availability cluster following setups should be done for both nodes.

1. Do the following database-related configurations.
 - a. Follow the steps below to configure the <APIM Analytics_HOME>/repository/conf/datasources/master-datasources.xml file as required
 - i. Enable the all the nodes to access the users database by configuring a datasource to be used by user manager as shown below.

```

<datasource>
    <name>WSO2UM_DB</name>
    <description>The datasource used by user
manager</description>
    <jndiConfig>
        <name>jdbc/WSO2UM_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://[MySQL DB url]:[port]/userdb</url>
            <username>[user]</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>

```

- ii. Enable the nodes to access the registry database by configuring the WSO2REG_DB data source as follows.

```

<datasource>
    <name>WSO2REG_DB</name>
    <description>The datasource used by the
registry</description>
    <jndiConfig>
        <name>jdbc/WSO2REG_DB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://[MySQL DB url]:[port]/regdb</url>
            <username>[user]</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>

```

For detailed information about registry sharing strategies, see the library article [Sharing Registry Space across Multiple Product Instances](#).

- b. Point to `WSO2_ANALYTICS_EVENT_STORE_DB` and `WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB` in the `< A P I M Analytics_HOME>/repository/conf/datasources/analytics-datasources.xml` file as shown below.

```

<datasources-configuration>
    <providers>

        <provider>org.wso2.carbon.ndatasource.rdbms.RDBMSDataSourceReade
r</provider>
    </providers>

    <datasources>
        <datasource>
            <name>WSO2_ANALYTICS_EVENT_STORE_DB</name>
            <description>The datasource used for analytics record
store</description>
            <definition type="RDBMS">
                <configuration>
                    <url>jdbc:mysql://[MySQL DB
url]:[port]/WSO2_ANALYTICS_EVENT_STORE_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>
                    <defaultAutoCommit>false</defaultAutoCommit>
                </configuration>
            </definition>
        </datasource>
        <datasource>
            <name>WSO2_ANALYTICS_PROCESSED_DATA_STORE_DB</name>
            <description>The datasource used for analytics record
store</description>
            <definition type="RDBMS">
                <configuration>
                    <url>jdbc:mysql://[MySQL DB
url]:[port]/WSO2_ANALYTICS_PROCESSED_DATA_STORE_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>
                </configuration>
            </definition>
        </datasource>
    </datasources>
</datasources-configuration>

```

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

```

        </datasource>
    </datasources>
</datasources-configuration>

```

For more information, see [Datasources in DAS documentation](#).

- c. To share the user store among the nodes, open the <APIM Analytics_HOME>/repository/conf/user-mgt.xml file and modify the dataSource property of the <configuration> element as follows.

```

<configuration>
...
<Property name="dataSource">jdbc/WSO2UM_DB</Property>
</configuration>

```

The datasource name specified in this configuration should be the same as the datasource used by user manager that you configured in sub step **a, i**.

- d. In the <APIM Analytics_HOME>/repository/conf/registry.xml file, add or modify the data source attribute of the <dbConfig name="govregistry"> element as follows.

```

<dbConfig name="govregistry">
    <dataSource>jdbc/WSO2REG_DB</dataSource>
</dbConfig>
<remoteInstance url="https://localhost:9443/registry">
    <id>gov</id>
    <cacheId>user@jdbc:mysql://localhost:3306/regdb</cacheId>
    <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>

```

Do not replace the following configuration when adding in the mounting configurations. The registry mounting configurations mentioned in the above steps should be added in addition to the following.

```
<dbConfig name="wso2registry">
    <dataSource>jdbc/WSO2CarbonDB</dataSource>
</dbConfig>
```

2. Update the <APIM Analytics_HOME>/repository/conf/axis2/axis2.xml file as follows to enable Hazelcast clustering for both nodes.

a. Set

clustering

```
class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" to true as shown below to enable Hazelcast clustering.
```

```
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent" enable="true">
```

- b. Enable **wka** mode on both nodes as shown below. For more information on **wka** mode, see [About Membership Schemes](#) in the Administration Guide.

```
<parameter name="membershipScheme">wka</parameter>
```

- c. Add both the nodes as well known members in the cluster under the **members** tag in each node as shown in the example below.

```
<members>
    <member>
        <hostName>[node1 IP]</hostName>
        <port>[node1 port]</port>
    </member>
    <member>
        <hostName>[node2 IP]</hostName>
        <port>[node2 port]</port>
    </member>
</members>
```

- d. For each node, enter the respective server IP address as the value for the **localMemberHost** property as shown below.

```
<parameter name="localMemberHost">[Server_IP_Address]</parameter>
```

- ▼ Click here to view the complete clustering section of the axis2.xml file. with the changes mentioned above.

```
<clustering
    class="org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent"
        enable="true">
```

```

<!--
    This parameter indicates whether the cluster has to be
    automatically initialized
        when the AxisConfiguration is built. If set to "true"
    the initialization will not be
        done at that stage, and some other party will have to
    explicitly initialize the cluster.
-->
<parameter name="AvoidInitiation">true</parameter>

<!--
    The membership scheme used in this setup. The only
    values supported at the moment are
        "multicast" and "wka"

    1. multicast - membership is automatically discovered
    using multicasting
        2. wka - Well-Known Address based multicasting.
    Membership is discovered with the help
        of one or more nodes running at a Well-Known
    Address. New members joining a
        cluster will first connect to a well-known
    node, register with the well-known node
        and get the membership list from it. When new
    members join, one of the well-known
        nodes will notify the others in the group. When
    a member leaves the cluster or
        is deemed to have left the cluster, it will be
    detected by the Group Membership
        Service (GMS) using a TCP ping mechanism.
-->

<parameter name="membershipScheme">wka</parameter>

<!--<parameter name="licenseKey">xxx</parameter>-->
<!--<parameter
name="mgtCenterURL">http://localhost:8081/mancenter/</parameter>-->

<!--
    The clustering domain/group. Nodes in the same group will
    belong to the same multicast
        domain. There will not be interference between nodes in
    different groups.
-->
<parameter name="domain">wso2.carbon.domain</parameter>

<!-- The multicast address to be used -->
<!--<parameter name="mcastAddress">228.0.0.4</parameter>-->

<!-- The multicast port to be used -->
<parameter name="mcastPort">45564</parameter>

<parameter name="mcastTTL">100</parameter>

```

```

<parameter name="mcastTimeout">60</parameter>

<!--
    The IP address of the network interface to which the
multicasting has to be bound to.
    Multicasting would be done using this interface.
-->
<!--
    <parameter
name="mcastBindAddress">10.100.5.109</parameter>
-->
<!-- The host name or IP address of this member -->

<parameter name="localMemberHost">[node IP]</parameter>

<!--
    The bind address of this member. The difference between
localMemberHost & localMemberBindAddress
        is that localMemberHost is the one that is advertised
by this member, while localMemberBindAddress
        is the address to which this member is bound to.
-->
<!--
    <parameter name="localMemberBindAddress">[node
IP]</parameter>
-->

<!--
    The TCP port used by this member. This is the port through
which other nodes will
        contact this member
-->
<parameter name="localMemberPort">[node port]</parameter>

<!--
    The bind port of this member. The difference between
localMemberPort & localMemberBindPort
        is that localMemberPort is the one that is advertised
by this member, while localMemberBindPort
        is the port to which this member is bound to.
-->
<!--
    <parameter name="localMemberBindPort">4001</parameter>
-->

<!--
Properties specific to this member
-->
<parameter name="properties">
    <property name="backendServerURL"
value="https://${hostName}:${httpsPort}/services//"/>
    <property name="mgtConsoleURL"

```

```

        value="https://${hostName}:${httpsPort}/">
            <property name="subDomain" value="worker"/>
        </parameter>

        <!--
            Uncomment the following section to load custom Hazelcast
            data serializers.
        -->
        <!--
            <parameter name="hazelcastSerializers">
                <serializer
typeClass="java.util.TreeSet">org.wso2.carbon.hazelcast.serializer.
TreeSetSerializer
                </serializer>
                <serializer
typeClass="java.util.Map">org.wso2.carbon.hazelcast.serializer.MapS
erializer</serializer>
            </parameter>
        -->

        <!--
            The list of static or well-known members. These entries
            will only be valid if the
            "membershipScheme" above is set to "wka"
        -->
        <members>
            <member>
                <hostName>[node1 IP]</hostName>
                <port>[node1 port]</port>
            </member>
            <member>
                <hostName>[node2 IP]</hostName>
                <port>[node2 port]</port>
            </member>
        </members>

        <!--
            Enable the groupManagement entry if you need to run this
            node as a cluster manager.
            Multiple application domains with different
            GroupManagementAgent implementations
            can be defined in this section.
        -->
        <groupManagement enable="false">
            <applicationDomain name="wso2.as.domain"
                               description="AS group"
agent="org.wso2.carbon.core.clustering.hazelcast.HazelcastGroupMa
gementAgent"
                               subDomain="worker"

```

```

        port="2222" />
    </groupManagement>
</clustering>

```

3. Configure the <APIM Analytics_HOME>/repository/conf/event-processor.xml file as follows to cluster API-M Analytics in the Receiver.

- a. Enable the HA mode by setting the following property.

```
<mode name="HA" enable="true">
```

- b. Disable the Distributed mode by setting the following property.

```
<mode name="Distributed" enable="false">
```

- c. For each node, enter the respective server IP address under the HA mode Config section as shown in the example below.

When you enable the HA mode for WSO2 API-M Analytics, the following are enabled by default:

- **State persistence:** If there is no real time use case that requires any state information after starting the cluster, you should disable event persistence by setting the persistence attribute to false in the <APIM Analytics_HOME>/repository/conf/event-processor.xml file as shown below.

```

<persistence enable="false">

    <persistenceIntervalInMinutes>15</persistenceInterval
    InMinutes>

    <persisterSchedulerPoolSize>10</persisterSchedulerPoo
    lSize>
        <persister
            class="org.wso2.carbon.event.processor.core.internal.
            persistence.FileSystemPersistenceStore">
            <property
                key="persistenceLocation">cep_persistence</property>
        </persister>
    </persistence>

```

When state persistence is enabled for WSO2 API-M Analytics, the internal state of API-M Analytics is persisted in files. These files are not automatically deleted. Therefore, if you want to save space in your API-M Analytics pack, you need to delete them manually.

These files are created in the <APIM Analytics_HOME>/cep_persistence/<tenant-id> directory. This

directory has a separate sub-directory for each execution plan. Each execution plan can have multiple files. The format of each file name is <TIMESTAMP>_<EXECUTION_PLAN_NAME> (e.g, 1493101044948_MyExecutionPlan). If you want to clear files for a specific execution plan, you need to leave the two files with the latest timestamps and delete the rest.

- **Event synchronization:** However, if you set the event.duplicated.in.cluster=true property for an event receiver configured in a node, API-M Analytics does not perform event synchronization for that receiver.

```
<!-- HA Mode Config -->
<mode name="HA" enable="true">
    ...
    <eventSync>
        <hostName>[Server_IP_Address]</hostName>
```

▼ [Click here to view the complete event-processor.xml file with the changes mentioned above.](#)

```
<eventProcessorConfiguration>
    <mode name="SingleNode" enable="false">
        <persistence enable="false">

            <persistenceIntervalInMinutes>15</persistenceIntervalInMinutes>

            <persisterSchedulerPoolSize>10</persisterSchedulerPoolSize>
                <persister
                    class="org.wso2.carbon.event.processor.core.internal.persistence.FileSystemPersistenceStore">
                    <property
                        key="persistenceLocation">cep_persistence</property>
                    </persister>
                </persistence>
            </mode>

            <!-- HA Mode Config -->
            <mode name="HA" enable="true">
                <nodeType>
                    <worker enable="true"/>
                    <presenter enable="false"/>
                </nodeType>

                <checkMemberUpdateInterval>10000</checkMemberUpdateInterval>
                <eventSync>
                    <hostName>172.18.1.217</hostName>
                    <port>11224</port>
                    <reconnectionInterval>20000</reconnectionInterval>
                    <serverThreads>20000</serverThreads>
                    <!--Size of TCP event publishing client's send buffer
                    in bytes-->
```

```

<publisherTcpSendBufferSize>5242880</publisherTcpSendBufferSize>
    <!--Character encoding of TCP event publishing
client-->
    <publisherCharSet>UTF-8</publisherCharSet>
    <publisherBufferSize>1024</publisherBufferSize>

<publisherConnectionStatusCheckInterval>30000</publisherConnections
tatusCheckInterval>
    <!--Number of events that could be queued at receiver
before they are synced between CEP/DAS nodes-->
    <receiverQueueSize>1000000</receiverQueueSize>
    <!--Max total size of events that could be queued at
receiver before they are synced between CEP/DAS nodes-->
    <receiverQueueMaxSizeMb>10</receiverQueueMaxSizeMb>
    <!--Number of events that could be queued at publisher
to sync output between CEP/DAS nodes-->
    <publisherQueueSize>1000000</publisherQueueSize>
    <!--Max total size of events that could be queued at
publisher to sync output between CEP/DAS nodes-->
    <publisherQueueMaxSizeMb>10</publisherQueueMaxSizeMb>
</eventSync>
<management>
    <hostName>172.18.1.217</hostName>
    <port>10005</port>
    <tryStateChangeInterval>15000</tryStateChangeInterval>
    <stateSyncRetryInterval>10000</stateSyncRetryInterval>
</management>
<presentation>
    <hostName>0.0.0.0</hostName>
    <port>11000</port>
    <!--Size of TCP event publishing client's send buffer
in bytes-->
<publisherTcpSendBufferSize>5242880</publisherTcpSendBufferSize>
    <!--Character encoding of TCP event publishing
client-->
    <publisherCharSet>UTF-8</publisherCharSet>
    <publisherBufferSize>1024</publisherBufferSize>

<publisherConnectionStatusCheckInterval>30000</publisherConnections
tatusCheckInterval>
    </presentation>
</mode>

<!-- Distributed Mode Config -->
<mode name="Distributed" enable="false">
    <nodeType>
        <worker enable="true"/>
        <manager enable="true">
            <hostName>0.0.0.0</hostName>
            <port>8904</port>
        </manager>
        <presenter enable="false">

```

```

        <hostName>0.0.0.0</hostName>
        <port>11000</port>
    </presenter>
</nodeType>
<management>
    <managers>
        <manager>
            <hostName>localhost</hostName>
            <port>8904</port>
        </manager>
        <manager>
            <hostName>localhost</hostName>
            <port>8905</port>
        </manager>
    </managers>
    <!--Connection re-try interval to connect to Storm
Manager service in case of a connection failure-->
    <reconnectionInterval>20000</reconnectionInterval>
    <!--Heart beat interval (in ms) for event listeners in
"Storm Receivers" and "CEP Publishers" to acknowledge their
availability for receiving events"-->
    <heartbeatInterval>5000</heartbeatInterval>
    <!--Storm topology re-submit interval in case of a
topology submission failure-->

<topologyResubmitInterval>10000</topologyResubmitInterval>
    </management>
    <transport>
        <!--Port range to be used for events listener servers
in "Storm Receiver Spouts" and "CEP Publishers"-->
        <portRange>
            <min>15000</min>
            <max>15100</max>
        </portRange>
        <!--Connection re-try interval (in ms) for connection
failures between "CEP Receiver" to "Storm Receiver" connections
and "Storm Publisher" to "CEP Publisher" connections-->
        <reconnectionInterval>20000</reconnectionInterval>
        <!--Size of the output queue of each "CEP Receiver"
which stores events to be published into "Storm Receivers" .
This must be a power of two-->

<cepReceiverOutputQueueSize>8192</cepReceiverOutputQueueSize>
    <!--Size of the output queue of each "Storm Publisher"
which stores events to be published into "CEP Publisher" .
This must be a power of two-->

<stormPublisherOutputQueueSize>8192</stormPublisherOutputQueueSize>
    <!--Size of TCP event publishing client's send buffer
in bytes-->

<tcpEventPublisherSendBufferSize>5242880</tcpEventPublisherSendBuff
erSize>
```

```
<!--Character encoding of TCP event publishing  
client-->  
  
<tcpEventPublisherCharSet>UTF-8</tcpEventPublisherCharSet>  
    <!--Size of the event queue in each storm spout which  
stores events to be processed by storm bolts -->  
        <stormSpoutBufferSize>10000</stormSpoutBufferSize>  
  
<connectionStatusCheckInterval>20000</connectionStatusCheckInterval  
>  
    </transport>  
    <presentation>  
  
        <presentationOutputQueueSize>1024</presentationOutputQueueSize>  
            <!--Size of TCP event publishing client's send buffer  
in bytes-->  
  
        <tcpEventPublisherSendBufferSize>5242880</tcpEventPublisherSendBuff  
erSize>  
            <!--Character encoding of TCP event publishing  
client-->  
  
        <tcpEventPublisherCharSet>UTF-8</tcpEventPublisherCharSet>  
  
<connectionStatusCheckInterval>20000</connectionStatusCheckInterval  
>  
    </presentation>  
    <statusMonitor>  
        <lockTimeout>60000</lockTimeout>  
        <updateRate>60000</updateRate>  
    </statusMonitor>  
    <stormJar>org.wso2.cep.storm.dependencies.jar</stormJar>  
    <distributedUIUrl></distributedUIUrl>
```

```
<memberUpdateCheckInterval>20000</memberUpdateCheckInterval>
</mode>
</eventProcessorConfiguration>
```

The following node types are configured for the HA deployment mode in the <APIM Analytics_HOME>/repository/conf/event-processor.xml file.

- **eventSync**: Both the active and the passive nodes in this setup are event synchronizing nodes as explained in the introduction. Therefore, each node should have the host and the port on which it is operating specified under the <eventSync> element.

Note that the eventSync port is not automatically updated to the port in which each node operates via port offset.

- **management**: In this setup, both the nodes carry out the same tasks, and therefore, both nodes are considered manager nodes. Therefore, each node should have the host and the port on which it is operating specified under the <management> element.

Note that the management port is not automatically updated to the port in which each node operates via port offset.

- **presentation**: You can optionally specify only one of the two nodes in this setup as the presenter node. The dashboards in which processed information is displayed are configured only in the presenter node. Each node should have the host and the port on which the assigned presenter node is operating specified under the <presentation> element. The host and the port as well as the other configurations under the <presentation> element are effective only when the `presenter enable="false"` property is set under the <!-- HA Mode Config --> section.

4. Update

t h e

< A P I M

Analytics_HOME>/repository/conf/analytics/spark/spark-defaults.conf file as follows to use the Spark cluster embedded within API-M Analytics.

- Keep the `carbon.spark.master` configuration as `local`. This instructs Spark to create a Spark cluster using the Hazelcast cluster.
- Enter `2` as the value for the `carbon.spark.master.count` configuration. This specifies that there should be two masters in the Spark cluster. One master serves as an active master and the other serves as a stand-by master.

The following example shows the <APIM Analytics_HOME>/repository/conf/analytics/spark/spark-defaults.conf file with changes mentioned above.

```
carbon.spark.master local
carbon.spark.master.count 2
```

For more information, see [Spark Configurations in DAS documentation](#).

Important: If the path to <APIM Analytics_HOME> is different in the two nodes, please do the

following.

UNIX environment	Windows environment
------------------	---------------------

Create a symbolic link to <APIM Analytics_HOME> in both nodes, where paths of those symbolic links are identical. This ensures that if we use the symbolic link to access API-M Analytics, we can use a common path. To do this, set the following property in the <APIM Analytics_HOME>/repository/conf/analytics/spark/spark-defaults.conf file.

```
carbon.das.symbolic.link /home/ubuntu/das/das_symlink/
```

In the Windows environment there is a strict requirement to have both API-M Analytics distributions in a common path.

5. In order to share the C-Apps deployed among the nodes, configure the SVN-based deployment synchronizer. For detailed instructions, see [Configuring SVN-Based Deployment Synchronizer](#).

API-M Analytics Minimum High availability Deployment set up does not use a manager and a worker. For the purpose of configuring the deployment synchronizer, you can add the configurations relevant to the manager for the node of your choice, and add the configurations relating to the worker for the other node.

If you do not configure the deployment synchronizer, you are required to deploy any C-App you use in the API-M Analytics Minimum High Availability Deployment set up to both the nodes.

6. If the physical API-M Analytics server has multiple network interfaces with different IPs, and if you want Spark to use a specific Interface IP, open either the <APIM Analytics_HOME>/bin/load-spark-env-vars.sh file (for Linux) or <APIM Analytics_HOME>/bin/load-spark-env-vars.bat file (for Windows), and add the following parameter to configure the Spark IP address.

```
export SPARK_LOCAL_IP=<IP_Address>
```

7. Note that if you are deploying in an environment where file systems do not get persisted automatically, it's required to persist and share some directories under <APIM_ANALYTICS_HOME> directory. Please see APIM Analytics section in the [Common Runtime and Configuration Artifacts](#) page.

Starting the cluster

Once you complete the configurations mentioned above, start the two API-M Analytics nodes. If the cluster is successfully configured, the following CLI logs are generated.

- The following is displayed in the CLIs of both nodes, and it indicates that the registry mounting is successfully done.

```
[2016-01-28 14:20:53,596] INFO
{org.wso2.carbon.registry.core.jdbc.EmbeddedRegistryService} -
Configured Registry in 107ms
[2016-01-28 14:20:53,631] INFO
{org.wso2.carbon.registry.core.jdbc.EmbeddedRegistryService} -
Connected to mount at govregistry in 7ms
[2016-01-28 14:20:53,818] INFO
{org.wso2.carbon.registry.core.jdbc.EmbeddedRegistryService} -
Connected to mount at govregistry in 0ms
```

- A CLI log similar to the following is displayed for the first node you start to indicate that it has successfully started.

```
[2016-01-28 14:32:40,283] INFO
{org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent} -
Using wka based membership management scheme
[2016-01-28 14:32:40,284] INFO
{org.wso2.carbon.core.clustering.hazelcast.util.MemberUtils} - Added
member: Host:10.100.0.46, Remote Host:null, Port: 4000, HTTP:-1,
HTTPS:-1, Domain: null, Sub-domain:null, Active:true
[2016-01-28 14:32:40,284] INFO
{org.wso2.carbon.core.clustering.hazelcast.util.MemberUtils} - Added
member: Host:10.100.0.46, Remote Host:null, Port: 4001, HTTP:-1,
HTTPS:-1, Domain: null, Sub-domain:null, Active:true
[2016-01-28 14:32:41,665] INFO
{org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent} -
Hazelcast initialized in 1379ms
[2016-01-28 14:32:41,728] INFO
{org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent} -
Local member: [9c7619a9-8460-465d-8fd0-7eab1c464386] -
Host:10.100.0.46, Remote Host:null, Port: 4000, HTTP:9763, HTTPS:9443,
Domain: wso2.carbon.domain, Sub-domain:worker, Active:true
[2016-01-28 14:32:41,759] INFO
{org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent} -
Elected this member [9c7619a9-8460-465d-8fd0-7eab1c464386] as the
Coordinator node
[2016-01-28 14:32:41,847] INFO
{org.wso2.event.processor.manager.core.internal.HAManager} -
CEP HA Snapshot Server started on 0.0.0.0:10005
[2016-01-28 14:32:41,850] INFO
{org.wso2.event.processor.manager.core.internal.HAManager} -
Became CEP HA Active Member
[2016-01-28 14:32:41,885] INFO
{org.wso2.carbon.core.clustering.hazelcast.HazelcastClusteringAgent} -
Cluster initialization completed
```

- Once you start the second node, a CLI log similar to the following will be displayed for the first node to indicate that another node has joined the cluster.

```
[2016-01-28 14:34:13,252] INFO  
{org.wso2.carbon.core.clustering.hazelcast.wka.WKABasedMembershipSche  
me} - Member joined [504bceff-4a08-46fe-83e6-b9561d3fff81]:  
/10.100.0.46:4001  
[2016-01-28 14:34:15,963] INFO  
{org.wso2.carbon.event.processor.manager.commons.transport.client.TCP  
EventPublisher} - Connecting to 10.100.0.46:11224  
[2016-01-28 14:34:15,972] INFO  
{org.wso2.carbon.event.processor.manager.core.internal.EventHandler} -  
CEP sync publisher initiated to Member '10.100.0.46:11224'
```

- A CLI log similar to the following is displayed for the second node once it joins the cluster.

```
[2016-01-28 14:34:27,086] INFO  
{org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor}  
- Spark Master map size after starting masters : 2
```

Following are some exceptions you may view in the start up log when you start the cluster.

- When you start the passive node of the HA cluster, the following errors are displayed.
▼ [Click here to view the errors](#)

```

ERROR
{org.wso2.carbon.event.processor.manager.core.internal.HAManager} - CEP HA State syncing failed, No execution plans exist
for tenant -1234
org.wso2.carbon.event.processor.manager.core.exception.EventM
anagementException: No execution plans exist for tenant
-1234
    at
org.wso2.carbon.event.processor.core.internal.CarbonEventProc
essorManagementService.restoreState(CarbonEventProcessorManag
ementService.java:83)
    at
org.wso2.carbon.event.processor.manager.core.internal.HAManager
.syncState(HAManager.java:336)
    at
org.wso2.carbon.event.processor.manager.core.internal.HAManager
.access$100(HAManager.java:49)
    at
org.wso2.carbon.event.processor.manager.core.internal.HAManager$2.run(HAManager.java:276)
    at
java.util.concurrent.Executors$RunnableAdapter.call(Executors
.java:511)
    at
java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFut
ureTask.access$201(ScheduledThreadPoolExecutor.java:180)
    at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFut
ureTask.run(ScheduledThreadPoolExecutor.java:293)
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPooLE
xecutor.java:1142)
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPool
Executor.java:617)
    at java.lang.Thread.run(Thread.java:745)

```

This is because the artifacts are yet to be deployed in the passive node even though it has received the sync message from the active node. This error is no longer displayed once the start up for the passive node is complete.

- When the Apache Spark Cluster is not properly instantiated, the following errors are displayed.
 - ▼ [Click here to view the errors](#)

```

[2016-09-13 13:59:34,000]  INFO
{org.wso2.carbon.event.processor.manager.core.internal.Carbon
EventManagementService} - Starting polling event receivers
[2016-09-13 14:00:05,018]  ERROR
{org.wso2.carbon.analytics.spark.core.CarbonAnalyticsProcesso
rService} - Error while executing query :          CREATE

```

```

TEMPORARY TABLE isSessionAnalyticsPerMinute USING
CarbonAnalytics OPTIONS (tableName
"org_wso2_is_analytics_stream_SessionStatPerMinute", schema
"meta_tenantId INT -i, bucketId LONG, bucketStart LONG -i,
bucketEnd LONG -i, year INT, month INT, day INT, hour INT,
minute INT, activeSessionCount LONG, newSessionCount LONG,
terminatedSessionCount LONG, _timestamp LONG -i", primaryKeys
"meta_tenantId, bucketId, bucketStart, bucketEnd",
incrementalParams "isSessionAnalyticsPerHour, HOUR",
mergeSchema "false")
org.wso2.carbon.analytics.spark.core.exception.AnalyticsExecutionException: Exception in executing query CREATE TEMPORARY
TABLE isSessionAnalyticsPerMinute USING CarbonAnalytics
OPTIONS (tableName
"org_wso2_is_analytics_stream_SessionStatPerMinute", schema
"meta_tenantId INT -i, bucketId LONG, bucketStart LONG -i,
bucketEnd LONG -i, year INT, month INT, day INT, hour INT,
minute INT, activeSessionCount LONG, newSessionCount LONG,
terminatedSessionCount LONG, _timestamp LONG -i", primaryKeys
"meta_tenantId, bucketId, bucketStart, bucketEnd",
incrementalParams "isSessionAnalyticsPerHour, HOUR",
mergeSchema "false")
at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.executeQueryLocal(SparkAnalyticsExecutor.java:764)
at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.executeQuery(SparkAnalyticsExecutor.java:721)
at
org.wso2.carbon.analytics.spark.core.CarbonAnalyticsProcessorService.executeQuery(CarbonAnalyticsProcessorService.java:201
)
at
org.wso2.carbon.analytics.spark.core.CarbonAnalyticsProcessorService.executeScript(CarbonAnalyticsProcessorService.java:15
1)
at
org.wso2.carbon.analytics.spark.core.AnalyticsTask.execute(AnalyticsTask.java:60)
at
org.wso2.carbon.ntask.core.impl.TaskQuartzJobAdapter.execute(TaskQuartzJobAdapter.java:67)
at org.quartz.core.JobRunShell.run(JobRunShell.java:213)
at
java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)

```

```

        at java.lang.Thread.run(Thread.java:745)
Caused by:
org.wso2.carbon.analytics.spark.core.exception.AnalyticsExecutionException: Spark SQL Context is not available. Check if
the cluster has instantiated properly.
        at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.executeQueryLocal(SparkAnalyticsExecutor.java:755)
        ... 11 more
[2016-09-13 14:00:05,018] ERROR
{org.wso2.carbon.analytics.spark.core.CarbonAnalyticsProcesso
rService} - Error while executing query :          CREATE
TEMPORARY TABLE activeSessionTable USING CarbonAnalytics
OPTIONS (tableName
"ORG_WSO2_IS_ANALYTICS_STREAM_ACTIVESESSIONS", schema
"meta_tenantId INT -i -f, sessionId STRING -i -f,
startTimestamp LONG -i, renewTimestamp LONG -i,
terminationTimestamp LONG -i, year INT, month INT, day INT,
hour INT, minute INT, action INT -i -f, username STRING -i
-f, userstoreDomain STRING -i -f, remoteIp STRING -i -f,
region STRING -i -f, tenantDomain STRING -i -f,
serviceProvider STRING -i -f, identityProviders STRING -i -f,
rememberMeFlag BOOLEAN, userAgent STRING -i -f,
usernameWithTenantDomainAndUserstoreDomain STRING -i -f,
_timestamp LONG -i", primaryKeys "meta_tenantId, sessionId",
mergeSchema "false")
org.wso2.carbon.analytics.spark.core.exception.AnalyticsExecutionException: Exception in executing query CREATE TEMPORARY
TABLE activeSessionTable USING CarbonAnalytics OPTIONS
(tableName "ORG_WSO2_IS_ANALYTICS_STREAM_ACTIVESESSIONS",
schema "meta_tenantId INT -i -f, sessionId STRING -i -f,
startTimestamp LONG -i, renewTimestamp LONG -i,
terminationTimestamp LONG -i, year INT, month INT, day INT,
hour INT, minute INT, action INT -i -f, username STRING -i
-f, userstoreDomain STRING -i -f, remoteIp STRING -i -f,
region STRING -i -f, tenantDomain STRING -i -f,
serviceProvider STRING -i -f, identityProviders STRING -i -f,
rememberMeFlag BOOLEAN, userAgent STRING -i -f,
usernameWithTenantDomainAndUserstoreDomain STRING -i -f,
_timestamp LONG -i", primaryKeys "meta_tenantId, sessionId",
mergeSchema "false")
        at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.executeQueryLocal(SparkAnalyticsExecutor.java:764)
        at
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsExecutor.executeQuery(SparkAnalyticsExecutor.java:721)
        at
org.wso2.carbon.analytics.spark.core.CarbonAnalyticsProcessorService.executeQuery(CarbonAnalyticsProcessorService.java:201
)
        at
org.wso2.carbon.analytics.spark.core.CarbonAnalyticsProcessor

```

```
Service.executeScript(CarbonAnalyticsProcessorService.java:15
1)
at
org.wso2.carbon.analytics.spark.core.AnalyticsTask.execute(An
alyticsTask.java:60)
at
org.wso2.carbon.ntask.core.impl.TaskQuartzJobAdapter.execute(
TaskQuartzJobAdapter.java:67)
at org.quartz.core.JobRunShell.run(JobRunShell.java:213)
at
java.util.concurrent.Executors$RunnableAdapter.call(Executors
.java:511)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPooLE
xecutor.java:1142)
at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPool
Executor.java:617)
at java.lang.Thread.run(Thread.java:745)
Caused by:
org.wso2.carbon.analytics.spark.core.exception.AnalyticsExecu
tionException: Spark SQL Context is not available. Check if
the cluster has instantiated properly.
at
```

```
org.wso2.carbon.analytics.spark.core.internal.SparkAnalyticsE
xecutor.executeQueryLocal(SparkAnalyticsExecutor.java:755)
... 11 more
```

All the nodes in the Spark cluster should be started in order to stop this exception from occurring.

Testing the HA deployment

The HA deployment you configured can be tested as follows.

1. Access the Spark UIs of the active master and the stand-by master using < node ip>:8081 in each node.
 - Information relating to the active master is displayed as shown in the example below.

The screenshot shows the Spark Master UI at spark://10.100.5.109:7077. The main page displays the following information:

- URL:** spark://10.100.5.109:7077
- REST URL:** spark://10.100.5.109:6066 (cluster mode)
- Workers:** 2
- Cores:** 2 Total, 2 Used
- Memory:** 2.0 GB Total, 1024.0 MB Used
- Applications:** 1 Running, 0 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE (highlighted with a red box)

Workers

Worker Id	Address	State	Cores	Memory
worker-20151013104058-10.100.5.109-11000	10.100.5.109:11000	ALIVE	1 (1 Used)	1024.0 MB (512.0 MB Used)
worker-20151013104058-10.100.5.109-11001	10.100.5.109:11001	ALIVE	1 (1 Used)	1024.0 MB (512.0 MB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20151013104103-0000 (kill)	CarbonAnalytics	2	512.0 MB	2015/10/13 10:41:03	niranda	RUNNING	1.1 h

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

- Information relating to the stand-by master is displayed as shown in the example below.

Spark Master at spark://10.100.5.109:7078

URL: spark://10.100.5.109:7078
REST URL: spark://10.100.5.109:6067 (cluster mode)

Workers: 0
Cores: 0 Total, 0 Used
Memory: 0.0 B Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: STANDBY

Workers

Worker Id	Address	State	Cores	Memory
-----------	---------	-------	-------	--------

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

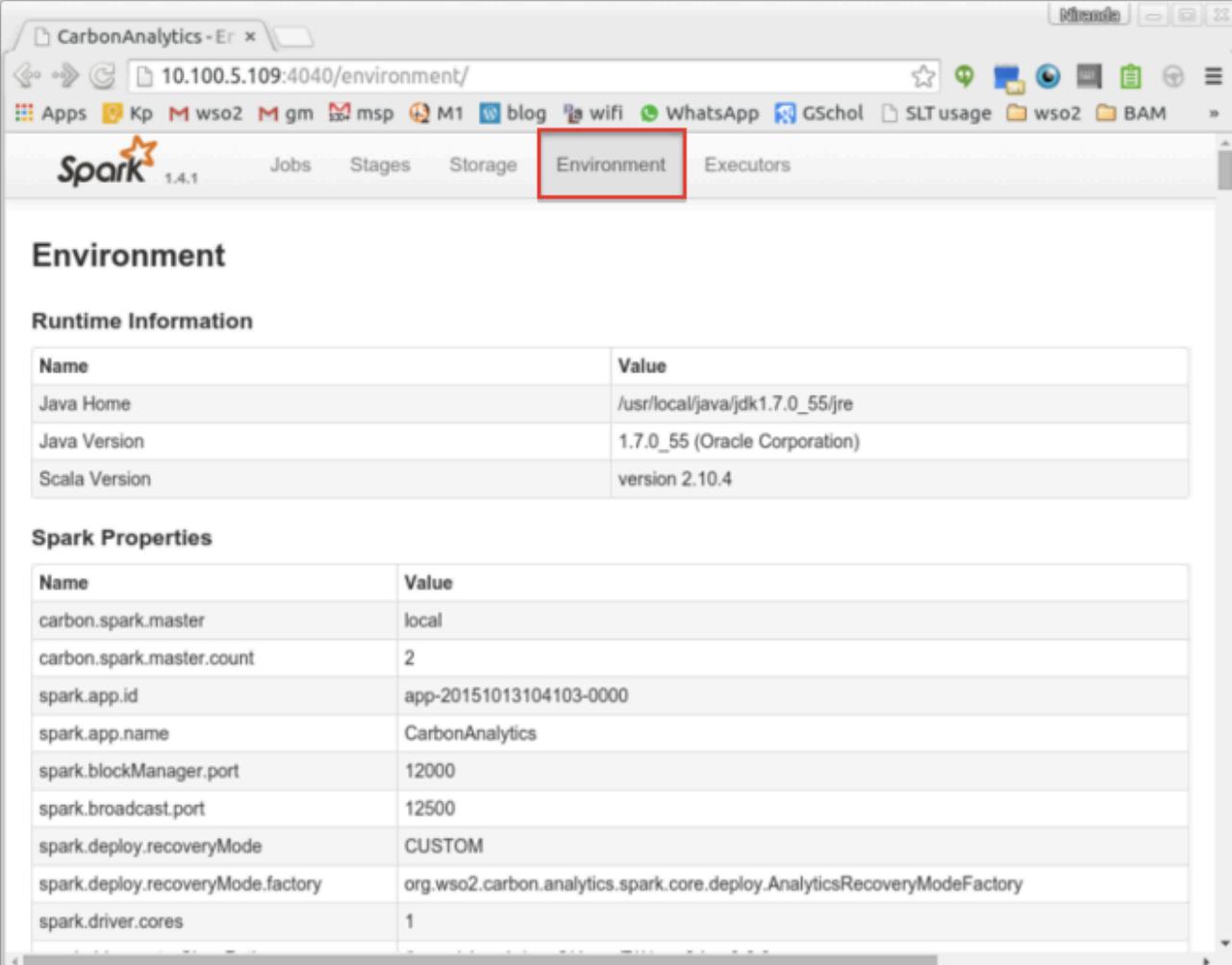
2. Click the links under **Running Applications** in the Spark UI of the active master to check the Spark application UIs of those applications. A working application is displayed as shown in the following example.

The screenshot shows the Spark UI interface for a CarbonAnalytics application. At the top, there's a navigation bar with links like Apps, Kp, wso2, gm, msp, M1, blog, wifi, WhatsApp, GSchol, SLT usage, wso2, and BAM. Below the bar, the Spark logo is visible next to the version 1.4.1. The main content area is titled "Spark Jobs (?)". It displays the following statistics:
Total Uptime: 1.1 h
Scheduling Mode: FAIR
Completed Jobs: 3

Below this, there's a link to "Event Timeline". The section titled "Completed Jobs (3)" contains a table with the following data:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	collect at SparkAnalyticsExecutor.java:749	2015/10/13 10:46:47	0.3 s	1/1	16/16
1	collect at SparkAnalyticsExecutor.java:749	2015/10/13 10:46:17	2 s	1/1	16/16
0	collect at SparkAnalyticsExecutor.java:749	2015/10/13 10:44:18	1 s	1/1	1/1

- Click the **Environment** tab of a Spark application UI to check whether all the configuration parameters are correctly set. You can also check whether the class path variables in this tab can be accessed manually.



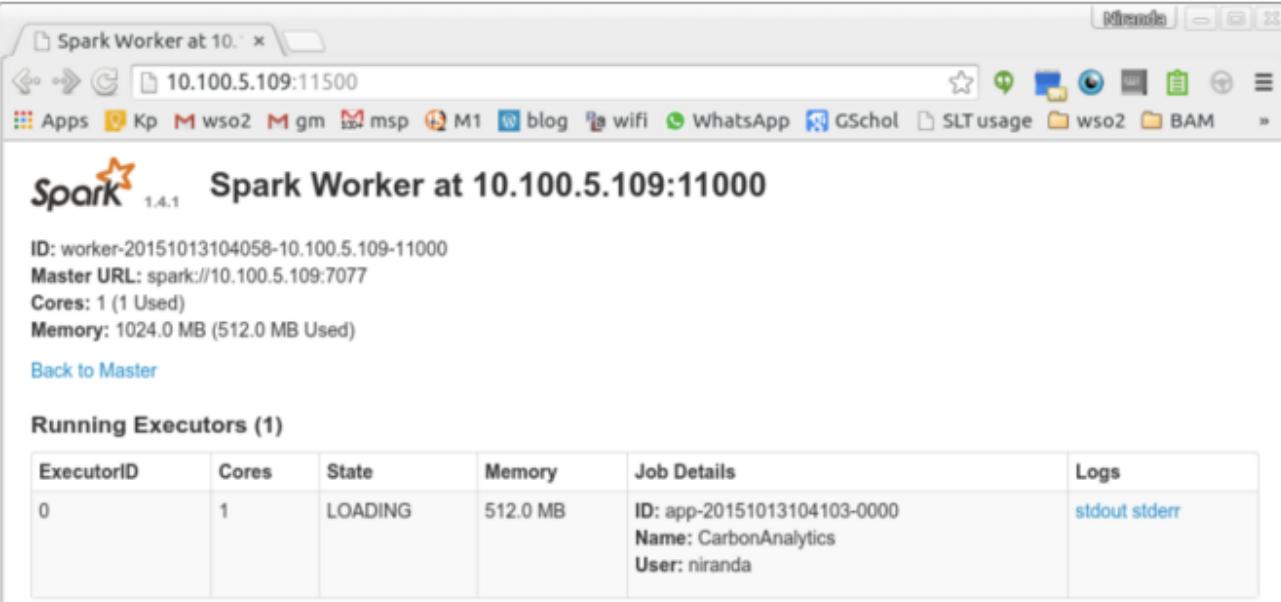
The screenshot shows the Spark UI interface. At the top, there is a navigation bar with tabs: Jobs, Stages, Storage, Environment (which is highlighted with a red box), and Executors. Below the navigation bar, the title "Environment" is displayed. Under "Environment", there is a section titled "Runtime Information" containing a table with four rows:

Name	Value
Java Home	/usr/local/java/jdk1.7.0_55/re
Java Version	1.7.0_55 (Oracle Corporation)
Scala Version	version 2.10.4

Below the runtime information, there is a section titled "Spark Properties" containing a table with nine rows:

Name	Value
carbon.spark.master	local
carbon.spark.master.count	2
spark.app.id	app-20151013104103-0000
spark.app.name	CarbonAnalytics
spark.blockManager.port	12000
spark.broadcast.port	12500
spark.deploy.recoveryMode	CUSTOM
spark.deploy.recoveryMode.factory	org.wso2.carbon.analytics.spark.core.deploy.AnalyticsRecoveryModeFactory
spark.driver.cores	1

- Check the Spark UIs of workers to check whether they have running executors. If a worker UI does not have running executors or if it is continuously creating executors, it indicates an issue in the Spark cluster configuration. The following example shows a worker UI with a running executor.



The screenshot shows the Spark Worker UI at the URL `10.100.5.109:11000`. The title of the browser tab is "Spark Worker at 10.100.5.109:11000". The page displays the following information:

- ID: worker-20151013104058-10.100.5.109-11000
- Master URL: spark://10.100.5.109:7077
- Cores: 1 (1 Used)
- Memory: 1024.0 MB (512.0 MB Used)

A link "Back to Master" is present. Below this, there is a section titled "Running Executors (1)" with a table:

ExecutorID	Cores	State	Memory	Job Details	Logs
0	1	LOADING	512.0 MB	ID: app-20151013104103-0000 Name: CarbonAnalytics User: niranda	stdout stderr

- Check the symbolic parameter, and check if you could manually access it via a `cd <directory>` command

in the CLI.

6. Log into the API-M Analytics Management Console and navigate to **Main => Manage => Batch Analytics => Console** to open the **Interactive Analytics Console**. Run a query in this console.

Configuring rsync for Deployment Synchronization

Deployment synchronization can be done using `rsync`, which is a file copying tool. These changes must be done in the manager node and in the same directory.

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

Tip: Different nodes are separated by new lines.

workers-list.txt

```
ubuntu@192.168.1.1:~/setup/192.168.1.1/apim/apim_worker/repository/deployment/server
ubuntu@192.168.1.2:~/setup/192.168.1.2/apim/apim_worker/repository/deployment/server
```

If you have configured tenants in worker nodes, you need to add the **repository/tenants** directory of the worker node to the list to synchronize tenant space

For example, if the node **ubuntu@192.168.1.1** needs to be synced with both the super tenant and the tenant space, the following two entries should be added to the `workers-list.txt`.

workers-list.txt

```
ubuntu@192.168.1.1:~/setup/192.168.1.1/apim/apim_worker/repository/deployment/server
ubuntu@192.168.1.1:~/setup/192.168.1.1/apim/apim_worker/repository/tenants
```

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

Note: You must create your own SSH key and define it as the `pem_file`. Alternatively, you can use an existing SSH key. Refer [SSH documentation](#) on generating and using the SSH Keys.

Specify the `manager_server_dir` depending on the location in your local machine. Change the `logs.txt` file path and the lock location based on where they are located in your machine.

rsync-for-carbon-depsync.sh

```
#!/bin/sh

manager_server_dir=~/wso2am-2.1.0/repository/deployment/server/
pem_file=~/ssh/carbon-440-test.pem

#delete the lock on exit
trap 'rm -rf /var/lock/depsync-lock' EXIT

mkdir /tmp/carbon-rsync-logs/

#keep a lock to stop parallel runs
if mkdir /var/lock/depsync-lock; then
    echo "Locking succeeded" >&2
else
    echo "Lock failed - exit" >&2
    exit 1
fi

#get the workers-list.txt
pushd `dirname $0` > /dev/null
SCRIPTPATH=`pwd`
popd > /dev/null
echo $SCRIPTPATH

for x in `cat ${SCRIPTPATH}/workers-list.txt`
do
echo ======>>
/tmp/carbon-rsync-logs/logs.txt;
echo Syncing $x;
rsync --delete -arve "ssh -i $pem_file -o StrictHostKeyChecking=no"
$manager_server_dir $x >> /tmp/carbon-rsync-logs/logs.txt
echo ======>>
/tmp/carbon-rsync-logs/logs.txt;
done
```

3. Create a Cron job that executes the above file every minute for deployment synchronization. Do this by running the following command in your command line.

Note: You can run the Cron job on one given node (master) at a given time. If you switch it to another node, you must stop the Cron job on the existing node and start a new Cron job on the new node after updating it with the latest files so far .

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

Changing the Default API-M Databases

When you use WSO2 API Manager (WSO2 API-M), you need the following databases in addition to the [Carbon database](#). By default, WSO2 API-M is shipped with embedded H2 databases for the following in addition to the Carbon database. These databases are stored in the <API-M_HOME>/repository/database directory.

- **WSO2AM_DB:** For API-M-specific data.
- **WSO2MB_DB:** For message brokering data.
- **WSO2METRICS_DB:** For storing data for Metrics monitoring.

For instructions on changing the default Carbon database, see [Changing the Carbon Database in the WSO2 Product Administration Guide](#).

Database Capacity

When planning the capacity of the underlying databases, note that the database holding the Access Tokens (WSO2AM_DB) and Statistics Data (WSO2AM_STATS_DB) will grow with the usage and the traffic on the gateway. To remove historical data see [Removing Unused Tokens from the Database](#) and [Purging Analytics Data](#)

Given below are the steps you need to follow in order to change the default databases listed above.

- Step 1 - Set up the database
- Step 2 - Create the datasource connection
 - Create the datasource connection for the API-M database
 - Create the datasource connection for the MB database (MB Store in WSO2 API-M)
 - Create the datasource connection for the Metrics database
 - Create the datasource connection for the Analytics database
 - Optionally, add DB configurations
- Step 3 - Create database tables
 - Create database tables in the API-M database
 - Create database tables in the MB database
 - Create database tables in the Metrics database
 - Create database tables when the server starts

Step 1 - Set up the database

You can set up the following database types for the API-M-specific databases:

- Setting up a MySQL database
- Setting up an MS SQL database
- Setting up an Oracle database
- Setting up an IBM DB2 database
- Setting up a PostgreSQL database

Note that we recommend to use Fail Over configuration over Load Balanced configuration with the MySQL clusters.

Step 2 - Create the datasource connection

A datasource is used to establish the connection to a database. By default, datasource connections for the API-M database, API-M statistics database, and the Message Brokering database are configured in the `master-datasources.xml` file. The datasource connection for the Metrics database is configured in the `metrics-datasources.xml` file. These datasource configurations point to the default H2 databases, which are shipped with the product. After setting up new databases to replace the default H2 databases, you can either change the default configurations in the above-mentioned files or configure new datasources.

Create the datasource connection for the API-M database

Follow the steps below.

1. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and locate the `<datasource>` configuration element.
2. Update the **URL** pointing to your database, the **username** and **password** required to access the database, and the **driver** details as shown below. Optionally, you can update the **other elements** for your database connection.

MySQLMS SQLOracleIBM DB2PostgreSQL

```

<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>
            <url>jdbc:mysql://localhost:3306/WSO2AM_DB</url>
            <username></username>
            <password></password>

            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

```
<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>

            <url>jdbc:jtds:sqlserver://localhost:1433/WSO2AM_DB</url>
                <username></username>
                <password></password>

            <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver
Classname>
                <maxActive>200</maxActive>
                <maxWait>60000</maxWait>
                <minIdle>5</minIdle>
                <testOnBorrow>true</testOnBorrow>
                <validationQuery>SELECT 1</validationQuery>
                <validationInterval>30000</validationInterval>
                <defaultAutoCommit>false</defaultAutoCommit>
            </configuration>
        </definition>
    </datasource>
```

```

<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>
            <url>jdbc:oracle:thin:@localhost:1521/orcl</url>
        <username></username>
            <password></password>
        <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
            <maxActive>100</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

```

<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>
            <url>jdbc:db2://SERVER_NAME:PORT/WSO2AM_DB</url>
            <username></username>
            <password></password>

        <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>360000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
    </definition>
</datasource>

```

```

<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>
            <url>jdbc:postgresql://localhost:5432/WSO2AM_DB</url>
            <username></username>
            <password></password>

            <driverClassName>org.postgresql.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

Create the datasource connection for the MB database (MB Store in WSO2 API-M)

Follow the steps below.

1. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and locate the `<datasource>` configuration element.
2. Update the **URL** pointing to your database, the **username** and **password** required to access the database, and the **driver** details as shown below. Further, be sure to set the `<defaultAutoCommit>` element to false for the MB database. Optionally, you can update the **other elements** for your database connection.

If you are using PostgreSQL, make sure to remove the `<validationQuery>` property from the datasource configuration.

MySQLMS SQLOracleIBM DB2PostgreSQL

```
<datasource>
    <name>WSO2_MB_STORE_DB</name>
    <description>The datasource used for message broker
database</description>
    <jndiConfig>
        <name>jdbc/WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:mysql://localhost:3306/WSO2MB_DB</url>
            <username></username>
            <password></password>

            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

```
<datasource>
    <name>WSO2_MB_STORE_DB</name>
    <description>The datasource used for message broker
database</description>
    <jndiConfig>
        <name>jdbc/WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:jtds:sqlserver://localhost:1433/WSO2MB_DB</url>
                <username></username>
                <password></password>

            <driverClassName>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver
ClassName>
                <maxActive>200</maxActive>
                <maxWait>60000</maxWait>
                <minIdle>5</minIdle>
                <testOnBorrow>true</testOnBorrow>
                <validationQuery>SELECT 1</validationQuery>
                <validationInterval>30000</validationInterval>
                <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

```

<datasource>
    <name>WSO2_MB_STORE_DB</name>
    <description>The datasource used for message broker
database</description>
    <jndiConfig>
        <name>jdbc/WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:oracle:thin:@localhost:1521/orcl</url>
        <username></username>
            <password></password>
        <driverClassName>oracle.jdbc.driver.OracleDriver</driverClassName>
            <maxActive>100</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

```

<datasource>
    <name>WSO2_MB_STORE_DB</name>
    <description>The datasource used for message broker
database</description>
    <jndiConfig>
        <name>jdbc/WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:db2://SERVER_NAME:PORT/WSO2MB_DB</url>
            <username></username>
            <password></password>

        <driverClassName>com.ibm.db2.jcc.DB2Driver</driverClassName>
            <maxActive>80</maxActive>
            <maxWait>360000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
        <defaultAutoCommit>false</defaultAutoCommit>
    </configuration>
    </definition>
</datasource>

```

```

<datasource>
    <name>WSO2_MB_STORE_DB</name>
    <description>The datasource used for message broker
database</description>
    <jndiConfig>
        <name>jdbc/WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:postgresql://localhost:5432/WSO2MB_DB</url>
            <username></username>
            <password></password>

            <driverClassName>org.postgresql.Driver</driverClassName>
                <maxActive>80</maxActive>
                <maxWait>60000</maxWait>
                <minIdle>5</minIdle>
                <testOnBorrow>true</testOnBorrow>
                <validationInterval>30000</validationInterval>
                <defaultAutoCommit>false</defaultAutoCommit>
            </configuration>
        </definition>
    </datasource>

```

3. Optionally, you can update the configuration elements [given below](#) for your database connection.

Create the datasource connection for the Metrics database

Follow the steps below.

1. Open the `<API-M_HOME>/repository/conf/datasources/metrics-datasources.xml` file and locate the `<datasource>` configuration element.
2. Update the **URL** pointing to your database, the **username** and **password** required to access the database, and the **driver** details as shown below. Optionally, you can update the [other elements](#) for your database connection.

MySQLMS SQLOraclePostgreSQL

```

<datasource>
    <name>WSO2_METRICS_DB</name>
    <description>The MySQL datasource used for WSO2 Carbon Metrics</description>
    <jndiConfig>
        <name>jdbc/WSO2MetricsDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <driverClassName>com.mysql.jdbc.Driver</driverClassName>
            <url>jdbc:mysql://localhost/wso2_metrics</url>
            <username>root</username>
            <password>root</password>
            <maxActive>50</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>true</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

```

<datasource>
    <name>WSO2_METRICS_DB</name>
    <description>The MSSQL datasource used for WSO2 Carbon Metrics</description>
    <jndiConfig>
        <name>jdbc/WSO2MetricsDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>

<driverClassName>net.sourceforge.jtds.jdbc.Driver</driverClassName>

<url>jdbc:jtds:sqlserver://localhost:1433/wso2_metrics</url>
            <username>sa</username>
            <password>sa</password>
            <maxActive>200</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>true</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

```
<datasource>
    <name>WSO2_METRICS_DB</name>
    <description>The Oracle datasource used for WSO2 Carbon
Metrics</description>
    <jndiConfig>
        <name>jdbc/WSO2MetricsDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <driverClassName>oracle.jdbc.OracleDriver</driverClassName>
            <url>jdbc:oracle:thin:@localhost:1521/wso2_metrics</url>
            <username>scott</username>
            <password>tiger</password>
            <maxActive>100</maxActive>
            <maxWait>60000</maxWait>
            <minIdle>5</minIdle>
            <testOnBorrow>true</testOnBorrow>
            <validationQuery>SELECT 1 FROM DUAL</validationQuery>
            <validationInterval>30000</validationInterval>
            <defaultAutoCommit>true</defaultAutoCommit>
            <databaseProps>
                <property
name="SetFloatAndDoubleUseBinary">true</property>
            </databaseProps>
        </configuration>
    </definition>
</datasource>
```

```

<datasource>
    <name>WSO2_METRICS_DB</name>
    <description>The MSSQL datasource used for WSO2 Carbon Metrics</description>
    <jndiConfig>
        <name>jdbc/WSO2MBStoreDB</name>
    </jndiConfig>
    <definition type="RDBMS">
        <configuration>
            <url>jdbc:postgresql://localhost:5432/wso2_metrics</url>
            <username></username>
            <password></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>
            <defaultAutoCommit>true</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>

```

Create the datasource connection for the Analytics database

This section is only applicable if you have downloaded the [WSO2 API Analytics distribution](#) to use WSO2 API Analytics with WSO2 API-M.

The API Manager integrates with the WSO2 Analytics platform to provide reports, statistics, and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs, and detect unusual activity, manage locations via geo location statistics, and carry out detailed analysis of the logs.

Follow the steps below to create the datasource connection for the Analytics database:

When working with Analytics, ensure that the `WSO2AM_DB` database is of the same RDBMS type as the Analytics database. For example, if the Analytics related DBs are created in MySQL, the API-M databases (`WSO2AM_DB`) should also be created in MySQL.

The following is a list of database versions that are compatible with WSO2 API-M Analytics.

- Postgres 9.5 and later
- MySQL 5.6
- MySQL 5.7
- Oracle 12c
- MS SQL Server 2012
- DB2

1. Open the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/analytics-datasources.xml` file. Note that two datasources named as `WSO2_ANALYTICS_EVENT_STORE_DB` and `WSO2_ANALYTIC_S_PROCESSED_DATA_STORE_DB` are configured by default to point to the H2 databases.

2. Create two database schemas in your database server (MySQL, Oracle, etc) for the two datasources, and change the configurations of those datasources to point to the relevant schemas. A sample configuration is given below.

The database user you provide here requires permissions to create tables.

Note that you do not need to run the database scripts against the created databases as the tables for the datasources are created at runtime.

```
<datasource>
    <name>WSO2_ANALYTICS_EVENT_STORE_DB</name>
    <description>The datasource used for analytics record store</description>
    <definition type="RDBMS">
        <configuration>

            <url>jdbc:mysql://localhost:3306/stats_200?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>
            <defaultAutoCommit>false</defaultAutoCommit>
        </configuration>
    </definition>
</datasource>
```

- If you are using **Oracle**, its recommended to increase the DB block size as described in <http://www.oratable.com/ora-01450-maximum-key-length-exceeded/>, to avoid the error 'ORA-01450: maximum key length (6398) exceeded'.
 - If you are using **DB2**, run [this script](#) before you start the WSO2 API-M Analytics server.
 - If you are using **MySQL 5.7**, open `<API-M_ANALYTICS_HOME>/repository/conf/analytics/spark/spark-jdbc-config.xml` and configure the `stringType` property under the `typeMapping` element as follows.
- ```
<stringType>VARCHAR(100)</stringType>
```

If you are using **MSSQL**, add the `SendStringParametersAsUnicode` property to the database connection URL in the data source configuration in the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/analytics-datasources.xml` file as shown below to avoid deadlock issues that are caused when the same table row is updated in two or more sessions at the same time.

```
<url>SQLSERVER_JDBC_URL;SendStringParametersAsUnicode=false</url>
```

3. Share the WSO2AM\_STATS\_DB datasource between WSO2 API-M and WSO2 API-M Analytics as follows.
  - a. Open the <API-M\_HOME>/repository/conf/datasources/master-datasources.xml file and make sure that a configuration for the WSO2AM\_STATS\_DB datasource is included. The default configuration is as follows.

```

<datasource>
 <name>WSO2AM_STATS_DB</name>
 <description>The datasource used for setting statistics to API
Manager</description>
 <jndiConfig>
 <name>jdbc/WSO2AM_STATS_DB</name>
 </jndiConfig>
 <definition type="RDBMS">
 <configuration>

 <url>jdbc:mysql://localhost:3306/WSO2AM_STATS_DB?autoReconnect=t
rue&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>
 <defaultAutoCommit>false</defaultAutoCommit>
 </configuration>
 </definition>
</datasource>

```

you need to enable analytics in publisher, store and gateway nodes. However, you need to add this datasource configuration in gateway nodes. Following table provides more information on Analytics usage of API Manager components in a distributed environment.

| Component       | Enable statistics          | Events Published           | Read statsDB |
|-----------------|----------------------------|----------------------------|--------------|
| Gateway_Manager | YES only if accept request | YES only if accept request | NO           |
| Gateway_worker  | YES                        | YES                        | NO           |
| Key Manager     | NO                         | NO                         | NO           |
| Publisher       | YES                        | NO                         | YES          |
| Store           | YES                        | YES                        | YES          |
| Traffic Manager | NO                         | NO                         | NO           |

You do not need to enable analytics in Key Manager and Traffic Manager nodes as those components do not read or publish statistics. Though gateway nodes publish events, they are not reading statistics database. Therefore you are not required to add the WSO2AM\_STATS\_DB datasource configuration in gateway nodes. Publisher node read

statistics but not publishing events. Therefore you can disable event publisher initialization at startup in publisher by setting **<SkipEventReceiverConnection>** value to true in **<PUBLISHER\_HOME>/repository/conf/api-manager.xml**. API Store node reads statistics and also publish events. Therefore we need to keep the statsource configuration for statsDB in Store node as well.

- b. Open the **<API-M\_ANALYTICS\_HOME>/repository/conf/datasources/stats-datasources.xml** file and make sure that the same configuration in the **<API-M\_HOME>/repository/conf/datasources/master-datasources.xml** file (mentioned in the previous sub step) is added in it.
4. Create a schema in your database server similar to the **WSO2AM\_STATS\_DB** datasource. Make sure that this datasource points to the relevant schema.

The database user you provide here requires permissions to create tables.

5. Download and copy the relevant database driver JAR file to the **<API-M\_ANALYTICS\_HOME>/repository/components/lib** directory.
6. Start the WSO2 API-M Analytics server.

## Troubleshooting

If you are configuring API-M Analytics with MSSQL and you get an error when you start the API-M Analytics server stating that a table cannot have more than one clustered index, follow the steps below.

1. Open the **<API-M\_ANALYTICS\_HOME>/repository/components/features/org.wso2.carbon.analytics.spark.server\_VERSION/spark-jdbc-config.xml** file.
2. Update the value for the **<indexCreateQuery>** element of the MySQL database as shown below.

```
<database name="Microsoft SQL Server">
 <indexCreateQuery>CREATE INDEX {{TABLE_NAME}}_INDEX ON
 {{TABLE_NAME}} ({{INDEX_COLUMNS}})</indexCreateQuery>
</database>
```

3. Restart the server for the above changes to take effect.

### Optionally, add DB configurations

Element	Description
<b>maxActive</b>	The maximum number of active connections that can be allocated at the same time from this pool. Enter any negative value to denote an unlimited number of active connections.
<b>maxWait</b>	The maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. You can enter zero or a negative value to wait indefinitely.
<b>minIdle</b>	The minimum number of active connections that can remain idle in the pool without extra ones being created. You can enter zero to create none.

<b>testOnBorrow</b>	The indication of whether objects are validated before being borrowed from the pool. If the object fails to validate, it is dropped from the pool, and another attempt is made to borrow another.
<b>validationQuery</b>	The SQL query that is used to validate connections from this pool before returning them to the caller.
<b>validationInterval</b>	The indication to avoid excess validation, and only run validation at the most, at this frequency (time in milliseconds). If a connection is due for validation but has been validated previously within this interval, it is not validated again.
<b>defaultAutoCommit</b>	<p>This property is only applicable to the <a href="#">MB Store database</a> of WSO2 APIM, where this property should be explicitly set to <code>false</code>. In all other database connections explained above, auto committing is enabled or disabled at the code level as required for that database, i.e., the default auto commit configuration specified for the RDBMS driver will be effective instead of this property element. Note that auto committing is typically enabled for an RDBMS by default.</p> <p>When auto committing is enabled, each SQL statement will be committed to the database as an individual transaction, as opposed to committing multiple statements as a single transaction.</p>

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

### Step 3 - Create database tables

To create the database tables, connect to the databases that you created earlier and run the scripts provided in the product pack.

#### Create database tables in the API-M database

The DB scripts corresponding to the database type are provided in the `<API-M_HOME>/dbscripts/apimgt` directory.

To create the necessary database tables:

1. Connect to the database and run the relevant script.  
For example, run the following command to create the API-M tables in a **MySQL** database.

```
mysql -u root -p -DWSO2AM_DB <
'<API-M_HOME>/dbscripts/apimgt/mysql.sql';
```

`<API-M_HOME>/dbscripts/mb-store/apimgt/mysql.sql` is the script that should be used for MySQL 5.6 and prior versions. If your database is MySQL 5.7 or later version, use `<API-M_HOME>/dbscripts/apimgt/mb-store/mysql5.7.sql` script file.

2. Restart the WSO2 API-M server.

#### Create database tables in the MB database

The DB scripts corresponding to the database type are provided in the `<API-M_HOME>/dbscripts/mb-store` directory.

To create the necessary database tables:

1. Connect to the database and run the relevant script.  
For example, run the following command to create the MB tables in a **MySQL** database.

```
mysql -u root -p -DWSO2MB_DB <
'<API-M_HOME>/dbscripts/mb-store/mysql.sql';
```

<API-M\_HOME>/dbscripts/mb-store/mysql.sql is the script that should be used for MySQL 5.6 and prior versions. If your database is MySQL 5.7 or later version, use <API-M\_HOME>/dbscripts/mb-store/mysql5.7.sql script file.

2. Restart the WSO2 API-M server.

### Create database tables in the Metrics database

The DB scripts corresponding to the database type are provided in the <API-M\_HOME>/dbscripts/metrics directory.

To create the necessary database tables:

1. Connect to the database and run the relevant script.  
For example, run the following command to create the MB tables in a **MySQL** database.

```
mysql -u root -p -DWSO2_METRICS_DB <
'<API-M_HOME>/dbscripts/metrics/mysql.sql';
```

<API-M\_HOME>/dbscripts/metrics/mysql.sql is the script that should be used for MySQL 5.6 and prior versions. If your database is MySQL 5.7 or later version, use <API-M\_HOME>/dbscripts/metrics/mysql5.7.sql script file.

2. Restart the WSO2 API-M server.

### Create database tables when the server starts

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

- For Windows: <API-M\_HOME>/bin/wso2server.bat `-Dsetup`
- For Linux: <API-M\_HOME>/bin/wso2server.sh `-Dsetup`

### Administration - API-M Analytics

The following sections cover the administration tasks carried out for WSO2 API-M Analytics.

- Re-indexing Existing Data

#### Re-indexing Existing Data

This section covers reindexing data that are already stored in databases configured for WSO2 API-M Analytics.

Data indexed and stored in databases may need to be reindexed due to the following reasons:

- If the index data is corrupted.
- If you change the database to another database, the data in the new database needs to be re-indexed.

To reindex existing data, follow the steps below:

1. Shut down API-M Analytics.
2. Remove all the index data stored in the <API-M\_ANALYTICS\_HOME>/repository/data directory.
3. In the <API-M\_ANALYTICS\_HOME>/repository/conf/analytics/local-shard-allocation-config.conf file, change the mode for all the shards from NORMAL to INIT.
4. Restart API-M Analytics.

## Common Runtime and Configuration Artifacts

The following are the artifacts used commonly in a WSO2 API Manager and API Manager Analytics deployment.

**Persistent Runtime Artifacts** - directories in API Manager which includes deployable files which are valid from a specified date and time at runtime.

**Persistent Configuration Artifacts** - directories in API Manager where the configuration files are included which are used for configurations.

Persistent runtime artifacts could be updated at the runtime, and are expected to be available across instance restarts, VM re-creation or container re-spawning. Persistent file storage systems should be used to ensure this.

Ex.: In a Kubernetes based container environment, its possible to use Persistent Volumes to persist these artifacts.

- API Manager
  - Persistent Runtime Artifacts
  - Persistent Configuration Artifacts
- APIM Analytics
  - Persistent Runtime Artifacts
  - Persistent Configuration Artifacts

## API Manager

### Persistent Runtime Artifacts

- <API-M\_HOME>/repository/deployment/server - Contains webapps that are related to customizing WSO2 API Manager during a deployment. Required for deploying a super tenant
- <API-M\_HOME>/repository/tenants - This is only used when the deployment involves multi-tenancy. For more information, see [Configuring Multiple Tenants](#)
- <API-M\_HOME>/solr - Contains files for Apache solr indexing. For additional information, see [Add Apache Solr-Based Indexing](#)
- <API-M\_HOME>/repository/database - H2 database (For solr indexing)

## Shared Artifacts

The following artifacts can be shared among API Manager nodes

- <API-M\_HOME>/repository/deployment/server
- <API-M\_HOME>/repository/tenants

### Persistent Configuration Artifacts

- <API-M\_HOME>/repository/resources - This folder/artifact contains such as keystores, templates, scripts, synapse configurations and custom sequences etc.
- <API-M\_HOME>/repository/conf - This folder contains the configuration files related to servers, datasources, registry, user management, etc.
- <API-M\_HOME>/bin - Contains files for JVM changes, profile changes, etc.

## APIM Analytics

### Persistent Runtime Artifacts

- <API-M\_ANALYTICS\_HOME>/repository/deployment/server - Contains webapps, execution plans, event receivers, etc that are related to customizing WSO2 APIM Analytics during a deployment. Required for deploying a super tenant.
- <API-M\_ANALYTICS\_HOME>/repository/data - Contains the indexing files
- <API-M\_ANALYTICS\_HOME>/repository/conf/analytics/ - My node id and shard allocation related data are stored in this directory.

## Shared Artifacts

The following artifacts can be shared among API Manager nodes

- <API-M\_ANALYTICS\_HOME>/repository/deployment/server

### Persistent Configuration Artifacts

- <API-M\_ANALYTICS\_HOME>/repository/resources - Resources such as key stores, templates, etc.
- <API-M\_ANALYTICS\_HOME>/repository/conf - This folder contains the configuration files related to servers, datasources, registry, user management, etc.
- <API-M\_ANALYTICS\_HOME>/bin - Contains files for JVM changes, profile changes, etc.

## JMX Monitoring

Java Management Extensions (JMX) is a technology that lets you implement management interfaces for Java applications. **JConsole** is a JMX-compliant monitoring tool, which comes with the Java Development Kit (JDK) 1.5 or later versions. Therefore, when you use a WSO2 product, JMX is enabled by default, which allows you to monitor the product using JConsole.

Go to the [WSO2 Administration Guide](#) for detailed instructions on how to configure JMX for a WSO2 product and how to use **JConsole** for monitoring a product.

### MBeans for WSO2 API Manager

When JMX is enabled, WSO2 ESB API Manager exposes a number of management resources as JMX MBeans that can be used for managing and monitoring the running server. When you start JConsole, you can monitor these MBeans from the **MBeans** tab. While some of these MBeans (**ServerAdmin** and **DataSource**) are common to all WSO2 products, some MBeans are specific to WSO2 API Manager.

The common MBeans are explained in detail in the [WSO2 Administration Guide](#). Listed below are the MBeans that are specific to WSO2 API Manager.

This section summarizes the attributes and operations available for the following ESB specific MBeans:

- [Connection MBeans](#)
- [Latency MBeans](#)
- [Threading MBeans](#)
- [Transport MBeans](#)

## Connection MBeans

These MBeans provide connection statistics for the HTTP and HTTPS transports.

You can view the following Connection MBeans:

- [org.apache.synapse/PassThroughConnections/http-listener](#)
- [org.apache.synapse/PassThroughConnections/http-sender](#)
- [org.apache.synapse/PassThroughConnections/https-listener](#)
- [org.apache.synapse/PassThroughConnections/https-sender](#)

## Attributes

Attribute Name	Description
ActiveConnections	Number of currently active connections.

LastXxxConnections	Number of connections created during last Xxx time period.
RequestSizesMap	A map of number of requests against their sizes.
ResponseSizesMap	A map of number of responses against their sizes.
LastResetTime	Last time connection statistic recordings was reset.

## Operations

Operation Name	Description
reset()	Clear recorded connection statistics and restart recording.

## Latency MBeans

This view provides statistics of the latencies from all backend services connected through the HTTP and HTTPS transports. These statistics are provided as an aggregate value.

You can view the following Latency MBeans:

- org.apache.synapse/PassthroughLatencyView/nio-http-http
- org.apache.synapse/PassthroughLatencyView/nio-https-https

## Attributes

Attribute Name	Description
Avg_Latency	Average latency since latency recording was last reset.
txxx_AvgLatency	Average latency for last xxx time period. For example, LastHourAvgLatency returns the average latency for the last hour.
LastResetTime	Last time latency statistic recording was reset.
Avg_Client_To_Esb_RequestReadTime	Average Time taken to read request by API Manager which is sent by the client
xxx_Avg_Client_To_Esb_RequestReadTime	Average Time taken to read request by gateway which is sent by the client for last xxx time period. For example 15m_Avg_Client_To_Esb_RequestReadTime means average Time taken to read request by API Manager which is sent by the client for last 15 minutes.
Avg_Esb_To_Backend_RequestWriteTime	Average Time taken to write the request from gateway to the backend.
xxx_Avg_Esb_To_Backend_RequestWriteTime	Average Time taken to write the request from gateway to the backend in last xxx time period. For example 15m_Avg_Esb_To_Backend_RequestWriteTime is average Time taken to write the request from gateway to the backend in last 15 minutes.
Avg_Backend_To_Esb_ResponseReadTime	Average Time taken to read the response from gateway to backend.
xxx_Avg_Backend_To_Esb_ResponseReadTime	Average Time taken to read the response from gateway to backend in last xxx time period.

Avg_Esb_To_Client_ResponseWriteTime	Average time taken to write the response from gateway to the client application.
xxx_Avg_Esb_To_Client_ResponseWriteTime	Average time taken to write the response from gateway to the client application in last xxx time period.
Avg_ClientWorker_Queue_Time	Average time where the ClientWorker get queued.
xxx_Avg_ClientWorker_Queue_Time	Average time where the ClientWorker get queued in last xxx time period.
Avg_ServeWorker_Queue_Time	Average time where the ServerWorker get queued.
xxx_Avg_ClientWorker_Queue_Time	Average time where the ServerWorker get queued in last xxx time period.
Avg_Latency_Backend	Average backend latency.
xxx_Avg_Latency_Backend	Average backend latency in last xxx time period.
Avg_Request_Mediation_Latency	Average latency of mediating the requests.
Avg_Response_Mediation_Latency	Average latency of mediating the responses.

## Operations

Operation Name	Description
reset()	Clear recorded latency statistics and restart recording.

## Threading MBeans

These MBeans are only available in the NHTTP transport and not in the default Pass Through transport.

You can view the following Threading MBeans:

- org.apache.synapse/Threading/PassThroughHttpServerWorker

## Attributes

Attribute Name	Description
TotalWorkerCount	Total worker threads related to this server/client.
AvgUnblockedWorkerPercentage	Time-averaged unblocked worker thread percentage.
AvgBlockedWorkerPercentage	Time-averaged blocked worker thread percentage.
LastXxxBlockedWorkerPercentage	Blocked worker thread percentage averaged for last Xxx time period.
DeadLockedWorkers	Number of deadlocked worker threads since last statistics reset.
LastResetTime	Last time thread statistic recordings was reset.

## Operations

Operation Name	Description
reset()	Clear recorded thread statistic and restart recording.

## Transport MBeans

For each transport listener and sender enabled in the ESB, there will be an MBean under the `org.apache.axis2/Transport` domain. For example, when the JMS transport is enabled, the following MBean will be exposed:

- `org.apache.axis2/Transport/jms-sender-n`

You can also view the following Transport MBeans:

- `org.apache.synapse/Transport/passthru-http-receiver`
- `org.apache.synapse/Transport/passthru-http-sender`
- `org.apache.synapse/Transport/passthru-https-receiver`
- `org.apache.synapse/Transport/passthru-https-sender`

## Attributes

Attribute Name	Description
Attribute Name	Description
ActiveThreadCount	Threads active in this transport listener/sender.
AvgSizeReceived	Average size of received messages.
AvgSizeSent	Average size of sent messages.
BytesReceived	Number of bytes received through this transport.
BytesSent	Number of bytes sent through this transport.
FaultsReceiving	Number of faults encountered while receiving.
FaultsSending	Number of faults encountered while sending.
LastResetTime	Last time transport listener/sender statistic recording was reset.
MaxSizeReceived	Maximum message size of received messages.
MaxSizeSent	Maximum message size of sent messages.
MetricsWindow	Time difference between current time and last reset time in milliseconds.
MinSizeReceived	Minimum message size of received messages.
MinSizeSent	Minimum message size of sent messages.
MessagesReceived	Total number of messages received through this transport.
MessagesSent	Total number of messages sent through this transport.
QueueSize	Number of messages currently queued. Messages get queued if all the worker threads in this transport thread pool are busy.
ResponseCodeTable	Number of messages sent against their response codes.
TimeoutsReceiving	Message receiving timeout.
TimeoutsSending	Message sending timeout.

## Operations

Operation Name	Description
start()	Start this transport listener/sender.
stop()	Stop this transport listener/sender.
resume()	Resume this transport listener/sender which is currently paused.
resetStatistics()	Clear recorded transport listener/sender statistics and restart recording.
pause()	Pause this transport listener/sender which has been started.
maintenenceShutdown(long gracePeriod)	Stop processing new messages, and wait the specified maximum time for in-flight requests to complete before a controlled shutdown for maintenance.

## Configuring the API Manager

This section explains how to configure the API Manager:

- Enabling CORS for APIs
- Enabling Access Control Support for API Publisher
- Adding Custom Properties to APIs
- Customizing the API Store
- Configuring Multiple Tenants
- Adding Internationalization and Localization
- Configuring Single Sign-on with SAML2
- Changing the Default Transport
- Configuring Caching
- Prevent API Suspension
- 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
- Transforming API Message Payload
- Sharing Applications and Subscriptions
- Configuring API Monetization Category Labels
- Enabling Notifications
- Working with Access Tokens
- Performance Tuning and Testing Results
- Removing Unused Tokens from the Database
- Migrating the APIs to a Different Environment
- Generating SDKs
- Revoke OAuth2 Application
- Configuring Keystores in WSO2 API Manager
- Logging
- Message Tracing
- Whitelisting Host Names for API Store

### Enabling CORS for APIs

Cross-Origin Resource Sharing (CORS) is a mechanism that allows accessing restricted resources (i.e., fonts, images, scripts, videos and iframes) from domains outside the domain from which the requesting resource originated. By default, web browsers apply the same-origin policy to avoid interactions between different origins. CORS defines a way in which a browser and a server can interact to determine whether or not it is safe to allow the cross-origin requests.

In API Manager, you can enable Cross-Origin Resource Sharing per API or as a global configuration that is applied across all APIs.

- Enabling CORS Globally
- Enabling CORS Per API

#### ***Enabling CORS Globally***

You can enable CORS globally for API Manager by configuring api-manager.xml located in <API-M\_HOME>/repository/conf directory.

Follow the steps below to enable CORS response headers globally. Once this configuration is enabled, it will be applied across all the APIs served by the API Gateway.

1. Open the <API-M\_HOME>/repository/conf/api-manager.xml file.
2. Locate the following configuration and set the <Enabled> attribute to true with the required CORS headers in the response. Once this configuration is applied in the API Gateway, it will affect all the API calls served by the Gateway.

```
<!-- Configuration to enable/disable sending CORS headers in the
Gateway response and define the Access-Control-Allow-Origin header
value.-->
<CORSConfiguration>
 <!-- Configuration to enable/disable sending CORS headers from the
Gateway-->
 <Enabled>true</Enabled>
 <!-- The value of the Access-Control-Allow-Origin header. Default
values are
 API Store addresses, which is needed for swagger to
function. -->
 <Access-Control-Allow-Origin>*</Access-Control-Allow-Origin>
 <!-- Configure Access-Control-Allow-Methods -->

 <Access-Control-Allow-Methods>GET,PUT,POST,DELETE,PATCH,OPTIONS</Access-Control-Allow-Methods>
 <!-- Configure Access-Control-Allow-Headers -->

 <Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction</Access-Control-Allow-Headers>
 <!-- Configure Access-Control-Allow-Credentials -->
 <!-- Specifying this header to true means that the server allows
cookies (or other user credentials) to be included on cross-origin
requests.
 It is false by default and if you set it to true then
make sure that the Access-Control-Allow-Origin header does not contain
the wildcard (*) -->

 <Access-Control-Allow-Credentials>false</Access-Control-Allow-Credentials>
</CORSConfiguration>
```

CORS configuration is enabled by default. Access control can be done by changing the parameters mentioned above in the api-manager.xml file.

**Enabling CORS Per API**

It is required to enable CORS globally before you enable CORS Per API. Therefore if you haven't done it yet, follow the steps in [Enabling CORS Globally](#) before starting the below steps.

1. Sign in to API Publisher and choose to design a new API.

Let's get started!

Add New API

The screenshot shows a step in the 'Add New API' process. There are three options: 'I have an Existing API', 'I have a SOAP Endpoint', and 'Design New API'. The 'Design New API' option is selected and highlighted with a blue circle. Below it is a 'Start Creating' button, which is also highlighted with a red box.

2. Click **Start Creating**.
3. Give the information in the table below and click **Add** to add the resource.

Field	Sample value
Name	WeatherAPI
Context	/weather
Version	v1.0.0
Resources	URL Pattern: current/{country}/{zipcode}
	Request types: GET method to return the current weather conditions of a zip code that belongs to a particular country

## Design API

The screenshot shows the 'Design' tab of the WSO2 API Manager interface. The top navigation bar has 'Design' (highlighted in blue) and 'Implement' tabs. The main area is titled 'General Details' and contains the following fields:

- Name:** WeatherAPI
- Context:** /weather
- Version:** v1.0.0
- Visibility:** Public
- Description:** (Text area with a green 'G' icon)
- Tags:** weather (with an 'x' button) and a text input field for adding more tags.

Below this is the 'API Definition' section, which includes:

- URL Pattern:** /weather/v1.0.0 (with a placeholder 'Url Pattern E.g.: path/to/resource')
- A list of HTTP methods: GET, POST, PUT, DELETE, PATCH, HEAD, with a 'more' link.
- A large 'Add' button with a plus sign.
- A summary row showing 'GET /current/{country}/{zipcode} + Summary'.
- Action buttons: 'Save' and 'Next: Implement >' (the latter is highlighted with a red box).

4. Once done, click **Next: Implement >**
5. In the **Implementation** tab, provide the following endpoint details.

Field	Sample value
Endpoint type	HTTP/REST endpoint
Production endpoint	You can find the Yahoo weather API's endpoint from <a href="https://developer.yahoo.com/weather/">https://developer.yahoo.com/weather/</a> . Copy the part before the '?' sign to get this URL: <a href="https://query.yahooapis.com/v1/public/yql">https://query.yahooapis.com/v1/public/yql</a>

6. Select the **Enable API based CORS Configuration** check box to enable CORS for the API.

## WeatherAPI: /weather/v1.0.0

The screenshot shows the 'Managed API' section of the WSO2 API Manager. At the top, there are three tabs: 'Design' (selected), 'Implement', and 'Manage'. Below the tabs, under 'Managed API', it says 'Provide the production and sandbox endpoints of the API to be managed.' There are fields for 'Endpoint Type' (set to 'HTTP/REST Endpoint'), 'Production Endpoint' (set to 'https://developer.yahoo.com/weather/'), and 'Sandbox Endpoint' (set to 'E.g.: http://appserver/resource'). A 'Test' button is next to each endpoint field. A 'Show More Options' link is also present. Below this, the 'Message Mediation Policies' section has an 'Enable Message Mediation' checkbox and a note about selecting a message mediation policy. A red box highlights the 'CORS configuration' section, which contains an 'Enable API based CORS Configuration' checkbox. A note below it says 'Check to select a message mediation policy to be executed in the message flow'.

7. Once you enable CORS, you will be able to see the CORS response header configuration section. Listed below are the CORS specific response headers supported by the API Gateway and how to configure them.

Header	Description	Sample values
Access-Control-Allow-Origin	Determines whether a resource can be shared with the resource of a given origin. The API Gateway validates the origin request header value against the list of origins defined under the Access Control Allow Origins configuration (this can be All Allow Origins or a specific value like localhost). If the host is in the allowed origin list, it will be set as the Access-Control-Allow-Origin response header in the response.	All Allow Origins(*), localhost
Access-Control-Allow-Headers	Determines, as part of the response to a preflight request (a request that checks to see if the CORS protocol is understood), which header field names can be used during the actual request. The gateway will set the header values defined under Access Control Allow Headers configurations.	authorization, Access-Control-Allow-Origin, Content-type, SOAPAction
Access-Control-Allow-Methods	This header specifies the method(s) allowed when accessing the resource in response to a preflight request. Required methods can be defined under the Access Control Allow Method configuration.	GET, PUT, POST, DELETE, PATCH, OPTIONS

Access-Control-Allow-Credentials	Determines whether or not the response to the request can be exposed to the page. It can be exposed when the header value is true. The header value can be set to true/false by enabling/disabling the Access Control Allow Credentials configuration.	true, false
----------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------

8. Once the CORS configurations are done, click **Next: Manage >**.

CORS configuration

Enable API based CORS Configuration

Access Control Allow Origins  All Allow Origins

Access Control Allow Headers  authorization  Access-Control-Allow-Origin  Content-Type  SOAPAction

Access Control Allow Methods  GET  PUT  POST  DELETE  PATCH  OPTIONS

Access Control Allow Credentials

**Save** **Next : Manage >**

9. Select the **Unlimited** subscription tier and click **Save and Publish** to create and publish the API to the API Store.

WeatherAPI : /weather/v1.0.0

The screenshot shows the 'Manage' tab selected in the top navigation bar. The main content area is divided into several sections:

- Configurations:**
  - "Make this the Default Version:" checkbox (unchecked)
  - "Transports:" dropdown showing "HTTPS" and "HTTP" selected.
  - "Response Caching:" dropdown set to "Disabled".
- Throttling Settings:**
  - "Maximum Backend Throughput:" radio buttons for "Unlimited" (selected) and "Specify".
  - "Subscription Tiers:" dropdown showing "Unlimited" selected, followed by descriptions for Gold, Silver, and Bronze tiers.
  - "Advanced Throttling Policies:" radio buttons for "Apply to API" (unchecked) and "Apply per Resource" (selected), with a "Select Policy per Resource" button below.
  - A note at the bottom of this section: "(i) Refer documentation for more information about each throttling setting."
- Gateway Environments:** A collapsed section indicated by a downward arrow.
- Business Information:** A collapsed section indicated by a downward arrow.
- Resources:**
  - "Scopes:" button with "+ Add Scopes".
  - A table row for a GET method with path "/current/{country}/{zipcode}":
 

GET	/current/{country}/{zipcode}	+ Summary	Application & Application User	Unlimited	+ Scope
-----	------------------------------	-----------	--------------------------------	-----------	---------
  - Action buttons: "Save", "Save & Publish" (highlighted with a red box), and "Cancel".

You have successfully enabled CORS for a specific API.

## Enabling Access Control Support for API Publisher

Visibility settings prevent certain user roles from viewing and modifying APIs created by another user role. This feature allows you to restrict the ability to view and modify APIs for a set of users.

- Enabling Access Control
- Using the API Publisher UI
- Using the REST API

### Enabling Access Control

To enable this feature, open the <API-M\_HOME>/repository/conf/api-manager.xml file. Add the code given below under <APIPublisher>.

Skip this step if you are using **WSO2 API Manager 2.1.0 - Update 2 or later**

```
<APIPublisher>
...
<EnableAccessControl>true</EnableAccessControl>
</APIPublisher>
```

## Instructions to existing users

Skip steps 1-3, if you are using **WSO2 API Manager 2.1.0 - Update 2 or later**

1. Open the <APIM\_Home>/repository/conf/registry.xml file
2. Add the following code as the first sub-element under <indexers>

```
<indexers>
...
<indexer
class="org.wso2.carbon.apimgt.impl.indexing.indexer.CustomAPIIndexer" mediaTypeRegEx="application/vnd.wso2-api\+xml" profiles ="default,api-store,api-publisher"/>
...
</indexers>
```

3. Replace the handler class org.wso2.carbon.registry.indexing.IndexingHandler with org.wso2.carbon.apimgt.impl.handlers.CustomAPIIndexHandler as shown below.

```
<handler
class="org.wso2.carbon.apimgt.impl.handlers.CustomAPIIndexHandler">
<filter class =
"org.wso2.carbon.registry.core.jdbc.handlers.filters.MediaTypeMatcher">
<property
name="mediaType">application/vnd.wso2-api+xml</property>
</filter>
</handler>
```

4. To re-index the registry, update the lastAccessTimeLocation parameter as given below.

```
<lastAccessTimeLocation>/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstimestxyz
</lastAccessTimeLocation>
```

Note that the registry indexing takes some time depending on the number of APIs you have in your store, so the existing APIs may not appear if you are accessing the publisher/store immediately after you start the server.

Restart the server after doing these changes.

#### **Using the API Publisher UI**

1. Log in to API Publisher as an API Creator. For more information on User Roles, see [Managing Users and Roles](#).
2. Create an API. Select **Restricted by roles** for Access Control in the **Design** tab.

**General Details**

Name:*	testAPI
Context:*	testAPI
Version:*	v1.0.0
Access Control: ?	Restricted by roles

3. Add the roles that have permission to view or modify this API.

**General Details**

Name:*	testAPI
Context:*	testAPI
Version:*	v1.0.0
Access Control: ?	Restricted by roles
Visible to Roles:*	creator
Visibility on Store: ?	Public

Ensure that the roles you add are valid. If the current creator is not an APIM admin, there should be at least one role of the current creator.

Users with APIM admin permission are treated differently. Even if an API is restricted to certain set of creators of publishers, it will be visible to all the API creators and publishers with APIM admin role.

#### **Using the REST API**

You can use the existing REST API to add a new API. To create an API with publisher access control restriction, add the two elements shown below in your request body,

```
"accessControl" : "RESTRICTED",
"accessControlRoles" : ["admin"]
```

Note that the roles should be valid. If the API creator is not an API-M admin he/she should at least have one of his/her roles in the accessControlRoles field.

This feature is available and enabled by default in **WSO2 API Manager - Update 2 or later**

The publisher role cache is enabled by default in API Manager. This is to avoid sending repeated requests to the Key Manager node in a distributed deployment, to authenticate user roles.

This WUM update allows you to disable the feature by disabling `<EnablePublisherRoleCache>true</EnablePublisherRoleCache>` under `<CacheConfigurations>`. We recommend enabling the elements shown in the example below.

```
<CacheConfigurations>
 <EnablePublisherRoleCache>true</EnablePublisherRoleCache>
 ...

```

Note that if disabled it results in lowering performance due to repeatedly accessing the Key Manager.

## Adding Custom Properties to APIs

Usually, APIs have a pre-defined set of properties such as the name, version, context, etc. However, there may be instances where you want to add specific custom properties to your API. You can do this in either of the following ways:

- Add custom properties via the API Publisher
- Add custom properties via the REST API

When adding custom properties, note the following:

- Property name should be unique.
- Property name should not contain spaces.
- Property name cannot be case sensitive.
- Property name cannot be any of the following as they are reserved keywords: provider, version, context, status, description, subcontext, doc, lcState, name, tags.

Once custom properties have been added, you can [search for APIs using custom property values](#).

### *Add custom properties via the API Publisher*

1. Log in to the API Publisher as an API creator using the following URL: `https://<localhost>:9443/publisher`.
2. [Create a new API](#) or choose to edit an existing API.
3. In the **Manage** tab, expand the **Meta Information** area, enter a custom property name and value (e.g. property name: environment, property value: preprod) and click the plus (+) sign to add it.

The screenshot shows the 'Edit API' screen in WSO2 API Manager. At the top, there are options for 'Advanced Throttling Policies' (radio buttons for 'Apply to API' and 'Apply per Resource'), a 'Select Policy per Resource' button, and a note about referring to documentation for throttling settings. Below this, there are sections for 'Gateway Environments', 'Business Information', and 'Meta Information'. In the 'Meta Information' section, there is a table for 'Additional Properties' with columns 'Property Name' and 'Property Value'. A row for 'environment' with a value of 'preprod' is selected and highlighted with a red box. At the bottom, there are tabs for 'POST /order', '+ Summary', 'Application & Application User', 'Unlimited', and '+ Scope'.

Property Name	Property Value
environment	preprod

#### 4. Save the API.

##### Add custom properties via the REST API

You can use the existing REST API to add a new API with custom properties. Add the following element to the request body including the relevant properties,

```
"additionalProperties : { "environment" , "preprod" , "secured" , "true" }
```

##### Search using custom properties

You can use the following format to search for an API using the custom properties:

<property\_name>:<property\_value>

For example, if you want to search for the **environment** property with a specific value (e.g. preprod), you can search as shown below:

# All APIs

environment:pre  



**PizzaShackAPI**  
1.0.0  
admin  
0 Users  
PUBLISHED  

 GO BACK  EDIT API  CREATE NEW VERSION  DOWNLOAD SOURCE

 0 Users	Access Control	All
 PUBLISHED	Visibility on Store	Public
 Docs	Context	/pizzashack/1.0.0
 View in Store	Production URL	https://localhost:9443/am/sample/pizzashack/v1/api/
	Sandbox URL	https://localhost:9443/am/sample/pizzashack/v1/api/
	Date Last Updated	12/16/2017, 8:51:24 PM
	Tier Availability	Unlimited
	Default API Version	None
	Tags	pizza
	Business Owner	Jane Roe [marketing@pizzashack.com]
	Technical Owner	John Doe [architecture@pizzashack.com]
	Published Environments	Production and Sandbox

Meta Information 

Property Name	Property Value
environment	preprod

## Customizing the API Store

You can customize the API Store in the following ways:

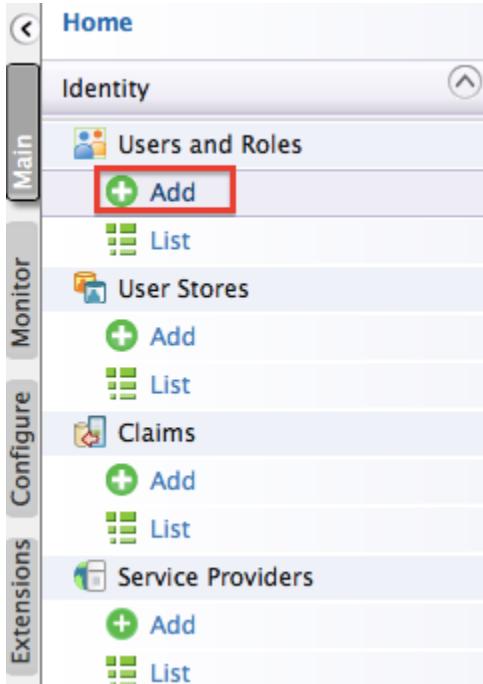
- Enabling or disabling self signup
- Changing the theme

- Changing language settings
- Setting single login for all apps
- Categorizing and grouping APIs
- Customizing the API group
- Customizing error Pages

#### **Enabling or disabling 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. Sign in to the management console (<https://<HostName>:9443/carbon>) as admin (or tenant admin).
2. In the **Main** menu, click **Add** under **Users and Roles**.



3. Click **Add New Role**.



4. Add a role by the name subscriber (or any other name you prefer).

## Add New Role

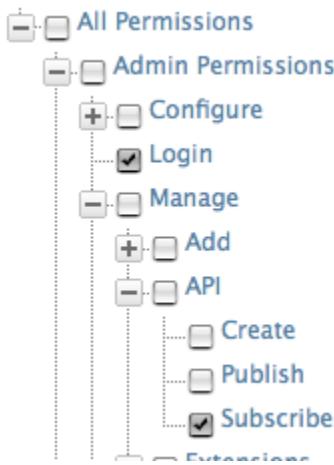
### Step 1 : Enter role details

Enter role details

Domain	PRIMARY
Role Name*	subscriber

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

- Click **Next** and add the following permissions:



- Go to the **Resources > Browse** menu.
- Navigate to the `/_system/governance/apimgt/applicationdata/` directory.
- Click on `sign-up-config.xml` to load the resource in the registry browser UI and select the "Edit as text" option to edit the configurations.

Home > Resources > Browse

## Browse

Root / \_system/governance/apimgt/applicationdata

Location: / \_system/governance/apimgt/applicationdata/ Go

Tree view Detail view

**Metadata**

**Properties**

**Entries**

Name	Created On
app-tiers.xml	2h 47m ago
res-tiers.xml	2h 47m ago
<b>sign-up-config.xml</b>	2h 47m ago
tiers.xml	2h 47m ago
workflow-extensions.xml	2h 47m ago

9. 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 must have the subscriber role created at this point.
- Set <AdminUserName> and <AdminPassword> to the credentials of the super admin, or if you are in a **multitenant setup** and you are not the super admin, to the tenant admin's credentials. **Note** that the super admin's credentials are admin/admin by default. If you changed the default super admin's credentials, using admin/admin will cause errors.

```

<selfsignup>

<enablesignup>true</enablesignup>

<!-- user storage to store users -->
<signupdomain>PRIMARY</signupdomain>

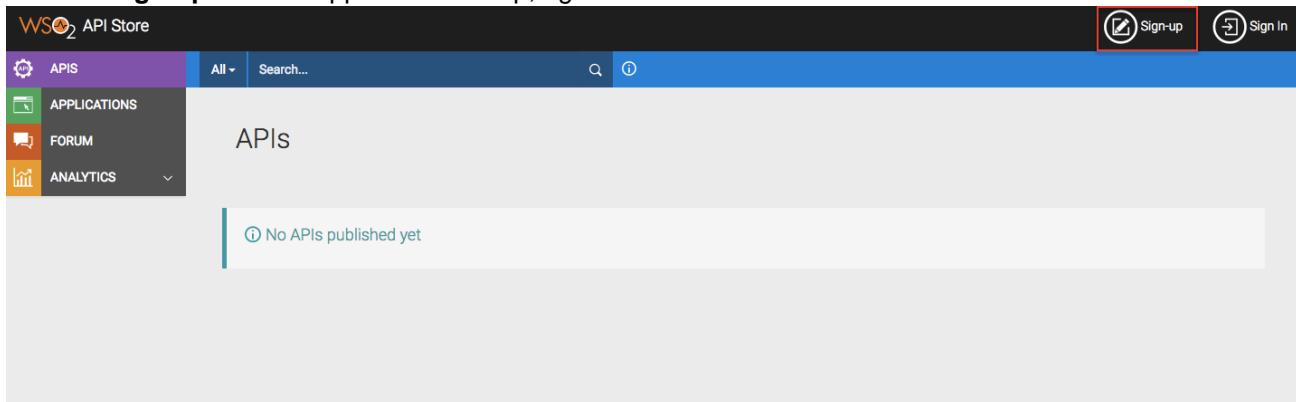
<!-- Tenant admin information. (for clustered setup credentials for
AuthManager) -->
<adminusername>xxxx</adminusername>
<adminpassword>xxxx</adminpassword>

<!-- List of roles for the tenant user -->
<signuproles>
 <signuprole>
 <rolename>subscriber</rolename>
 <isexternalrole>true</isexternalrole>
 </signuprole>
</signuproles>

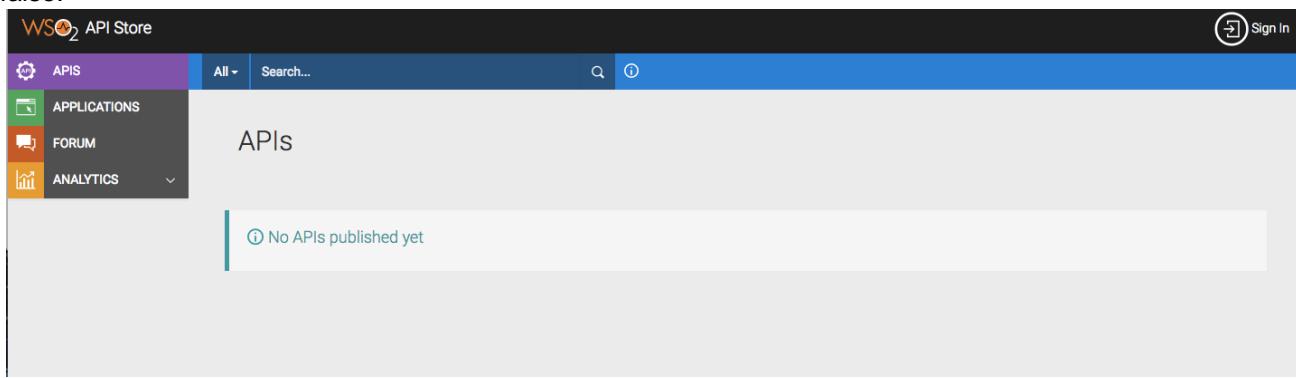
</selfsignup>

```

10. Restart the server and open the API Store (<https://<HostName>:9443/store>)  
Note the **Sign-up** link that appears in the top, right-hand corner of the window.



11. To disable the self signup capability, navigate to the `/_system/governance/apimgt/applicationdata/sign-up-config.xml` file 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**

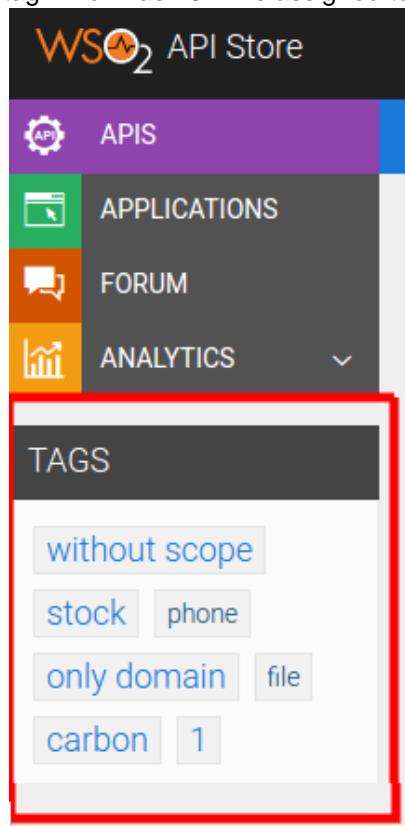
To change the language of the API Store, see [Adding Internationalization and Localization](#).

### **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 and grouping APIs**

API providers add tags to APIs when designing them using the API Publisher. Tags allow API providers to categorize APIs that have similar attributes. When 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 font size of the tag in the Store varies based on the number of APIs that are assigned to it. Therefore, for example the font size of a tag which has 10 APIs assigned to it will be bigger than the font size of a tag that has only 2 APIs assigned to it.



If you want to see the APIs grouped according to different topics in the API Store, add an API group:

Although the way in which you add a Tag and API group appears to be similar there are differences. Therefore, you need to note the following:

- The **group name should always have the suffix -group** and it **can have spaces** in it (e.g., APIs groups-group).
- The **tag name should not have a suffix or prefix**, but it **can have spaces**.

1. Go to <API-M\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf directory, open the site.json file and set the tagWiseMode attribute as true.
2. Add an API group to the APIs that you wish to group.
  - a. Go to the API Publisher (<https://<HostName>:9443/publisher>).
  - b. Click on the edit link of the respective API as shown below.

The screenshot shows the WSO2 API Publisher interface. The top navigation bar has 'HOME / APIS' and 'ADD NEW API' buttons. Below the navigation is a search bar with a magnifying glass icon and an information icon. The main content area is titled 'All APIs' and displays three API entries in cards:

API Name	Version	Owner	Users	Status	Action Buttons
Integration	1.0.0	admin	0 Users	PUBLISHED	
PizzaShackAPI	1.0.0	admin	0 Users	PUBLISHED	
Workflow	1.0.0	admin	0 Users	PUBLISHED	

- c. Add a group name to the APIs that you wish to group.

For example add the "APIs groups-group" tag to the Workflow and Integration APIs.

- d. Save the API for the tag to appear in the Store.
- e. Repeat steps 2 (a) to (d) to add another APIs to the newly created group.

Sign in to the API Store and note the API groups.

The screenshot shows the WSO2 API Store interface. On the left, there's a sidebar with tabs for APIS, APPLICATIONS, FORUM, and ANALYTICS. Below that is a section for TAGS with categories like pizza, Workflow, Social Media, Integration, and Finance. The main area is titled "APIs groups" and displays three cards: "Finance APIs", "Integrator", and "New APIs", each with a blue cloud icon containing gears.

If you wish, you can click on a group to see the APIs that belong to a specific group.

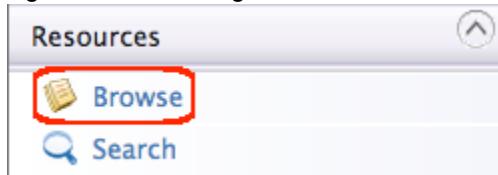
The screenshot shows the "New APIs" group page. The top navigation bar says "APIS GROUPS / NEW APIs". Below it, there are two API entries: "Integration" and "Workflow". Each entry has a thumbnail image, version (1.0.0), owner (admin), and a five-star rating.

API	Version	Owner	Rating
Integration	1.0.0	admin	★★★★★
Workflow	1.0.0	admin	★★★★★

#### ***Customizing the API group***

If you want to change the descriptions and the thumbnail images that come by default, do the following:

1. Sign in to the Management Console and click the **Resources > Browse** menu to open the registry.



2. Create a collection named `tags` under the registry location `/_system/governance/apimgt/application-groups`.

ondata.

The screenshot shows the governance interface at the path `/_system/governance/apimgt/applicationdata`. In the 'Entries' section, the 'Add Collection' button is highlighted with a red box. A modal window titled 'Add Collection' is open, showing fields for 'Name' (set to 'tags'), 'Media Type' (set to 'not specified'), and a 'Description' field. The 'Add' button in the modal is also highlighted with a red box.

3. Give read permission to the `system/wso2.anonymous.role` role.

The screenshot shows the governance interface with the 'Permissions' section open. Under 'New Role Permissions', a row is selected for the role `system/wso2.anonymous.role`, action `Read`, and permission `Allow`. The 'Add Permission' button is highlighted with a red box.

4. Add each tag as collections under the tags collection (e.g., Workflow APIs-group, Integration APIs-group, Quote APIs-group.)
5. Navigate to each tag collection and upload the following:
  - description.txt** with the description of the tag
  - thumbnail.png** for the thumbnail image
6. Back in the API Store, note the changes you did in the registry.

#### Customizing error Pages

In API Manager store/publisher and admin webapps, **jaggery.conf** is the Jaggery configuration file specifies the application specific configurations. In that file we can find following code block which have configured the error pages.

```
"errorPages" :
{
 "401": "/site/pages/error-pages/401.html",
 "403": "/site/pages/error-pages/403.html",
 "404": "/site/pages/error-pages/404.html",
 "500": "/site/pages/error-pages/500.html"
}
```

If such a specified error occurs due to an operation or page redirection inthe web application, it redirects to the specified html page. As an example, if you request for <https://localhost:9443/store/site/conf.site.json>, it gives a 403 response, it serves the html page site/pages/error-pages/403.html specified above.



```
<html>
 <head>
 </head>

 <body>
 <h2>Error 403 : Forbidden</h2>

 <p>
 <h4>You don't have permission to access anything with that kind of
request. </h4>
 </body>
 </html>
```

These error pages are located in `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/pages/error-pages` directory. You can customize these html pages according to your preference (adding css, javascript or jquery functionalities). And also you can create your own html pages to be viewed for errors occured by adding it to the jaggery.conf.

### Configuring Multiple Tenants

The goal of multi-tenancy is to maximize resource sharing by allowing multiple users (tenants) to log in and use a single server/cluster at the same time, in a tenant-isolated manner. That is, each user is given the experience of using his/her own server, rather than a shared environment. Multi-tenancy 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.

For common instructions on configuring multiple tenants for your WSO2 server, see [Working with Multiple Tenants](#) in the WSO2 Administration Guide and to understand how multi-tenancy works with the API Store, see [Managing Tenants](#).

## 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
<b>Domain</b>	The domain name for the organization, which should be unique (e.g., abc.com). This is used as a unique identifier for your domain. You can use it to log into the admin console to be redirected to your specific tenant. The domain is also used in URLs to distinguish one tenant from another.
<b>Select Usage Plan for Tenant</b>	The usage plan defines limitations (such as number of users, bandwidth, etc.) for the tenant. For on-premises deployment, there is only one default plan, i.e., Demo.
<b>First Name/Last Name</b>	The name of the tenant admin.
<b>Admin Username</b>	The login username of the tenant admin. The username always ends with the domain name (e.g., admin@abc.com)
<b>Admin Password</b>	The password used to log in using the admin username specified.
<b>Admin Password (Repeat)</b>	Repeat the password to confirm.
<b>Email</b>	The email address of the admin.

Note that all the above parameters are required parameters.

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. Enter the domain name in the **Enter the Tenant Domain** parameter and click **Find** to find the newly added tenant in the list.

Domain	Email	Created Date	Active	Edit
tenantone.com	tenantone@gmail.com	2017/11/13 14:39:52	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
tenanttwo.com	tenant_two@gmail.com	2017/11/13 14:40:37	<input checked="" type="checkbox"/>	<a href="#">Edit</a>

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,

tenanttwo.com	<a href="#">Visit Store</a>
carbon.super	<a href="#">Visit Store</a>
tenantone.com	<a href="#">Visit Store</a>

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.

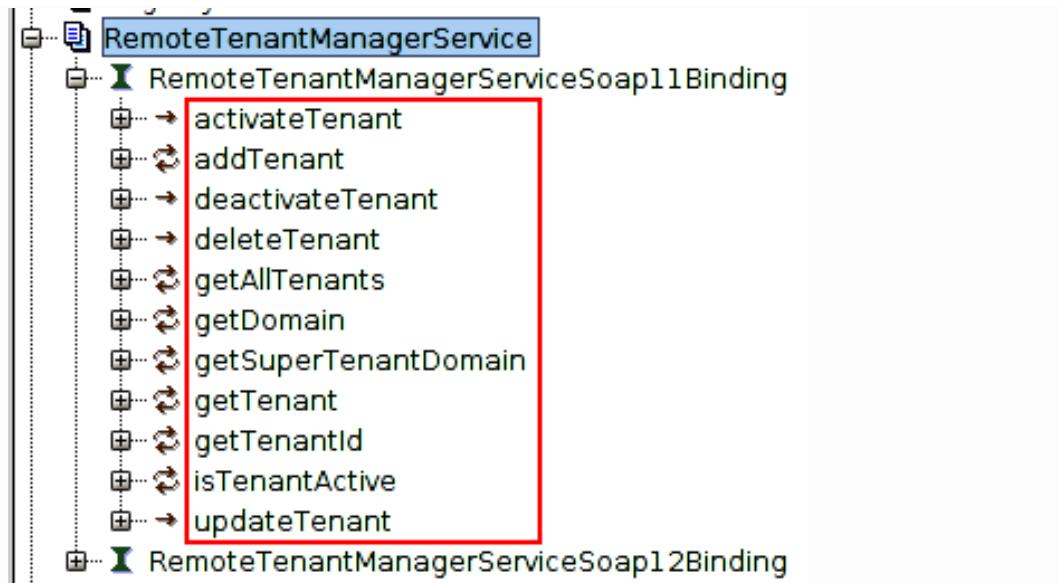
2. Click the **Visit Store** link associated with a given store to open it.
3. Anonymous users can browse all stores and all public APIs that are published to them. However, in order to subscribe to an API, the user must log in.

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](#).

Other tenant management operations such as activating, deactivating, updating and deleting, which are not available in the Management Console UI, can be done through the `RemoteTenantManager` Admin Service. You can invoke these operations using a SOAP client like SOAP UI. Follow the steps below to do the configurations using SOAP UI.

- i. Open the `<API-M_HOME>/repository/conf/carbon.xml` file and set `HideAdminServiceWSDLs` parameter to false.
- ii. Start SOAP UI client, and import the WSDL <https://localhost:9443/services/RemoteTenantManagerService?wsdl>. This assumes that you are running the SOAP UI client from the same machine as the API Manager instance.
- iii. Note that there are several operations shown in the SOAP UI after importing the wsdl file:



- iv. Click on each operation to open the request view. For an example, for activateTenant operation, you can see the following request view:

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://org.wso2.carbon.tenant.mgt/">
 <soapenv:Header/>
 <soapenv:Body>
 <ser:activateTenant>
 <!--Optional:-->
 <ser:tenantId>1</ser:tenantId>
 </ser:activateTenant>
 </soapenv:Body>
</soapenv:Envelope>

```

- v. You can invoke the RemoteTenantManager Admin service and do the tenant activation operation with the corresponding tenant ID. You can perform the other operations via SOAP UI as well. Note that you need to set the admin user credentials from the SOAP UI to invoke tenant admin operations.

If you perform operations such as tenant deletion, even though the tenant details are removed, any data stored in registry, file system, other databases, etc. will not be removed. Such data will need to be removed manually.

## 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

- 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.
- Set the browser's encoding type to UTF-8.

## Introduction to resource files

1. Go to <APIM\_HOME>/repository/deployment/server/jaggeryapps/publisher directory where <APIM\_HOME> is the API Manager distribution's home.
2. 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/documentation.jag?api=i18n>.

### Localizing strings in Jaggery files

3. 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**.

The file name which includes the locale code will differ according to the language selected in your browser. Create a new file for the language you select, if the selected language is not available. The file Name should be **locale\_{localeCode}.json** where the **localeCode** refers to the sub tag of the particular language. Please refer the [IANA Language Subtag Registry page](#) for a list of sub tags.

By matching the accept Language header, the particular file will be selected from the resource file directory of each web application for the localization by the server. If there is no locale file found with the matching sub tag, locale\_default.json file will be served for the localization.

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

1. 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.
2. Once done, open the API Publisher web application in your browser (<https://<YourHostName>:9443/p>

ublisher).

3. 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, to gain access to multiple other related applications without having to repeatedly authenticate themselves. It also allows the web applications to 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 the user role.

A claim is a piece of information about a particular subject and it is an attribute of the user that is mapped to the underlying user store. A claim can be anything that the subject is owned by or associated with, such as name, group, preferences, etc. A claim provides a single and general notion to define the identity information related to the subject. A set of claims is called a dialect (e.g., <http://wso2.org/claims>)

This section covers the following topics.

- Configuring API Manager for SSO
- Configuring External IDP through Identity Server for SSO
- Configuring Identity Server as IDP for SSO

For more information on SAML related terminologies discussed in the sections above, go to [Assertions and Protocols for the OASIS SAML 2.0 documentation](#).

## Configuring API Manager for SSO

You can configure the API Manager for SAML SSO by following the instructions below.

- Configuring the Carbon Console for SSO
- Configuring Publisher/Store for SSO
- Configuring the API Store for SSO in passive mode

### Configuring the Carbon Console for SSO

Open the `<API-M_HOME>/repository/conf/security/authenticators.xml` file and give the configurations as shown below.

- Set disabled attributes in the `<Authenticator>` element to `false`.
- `ServiceProviderID`: The issuer name of the service provider.
- `IdentityProviderSSOServiceURL`: The URL of the IDP. In this example, it is the URL of the Identity Server.

A **Service Provider (SP)** is an entity that provides web services. A service provider relies on a trusted Identity Provider (IdP) for authentication and authorization. In this case, the Identity Server acts as the IdP and does the task of authenticating and authorizing the user of the service provider.

For instructions on how you can configure WSO2 API Manager with IdPs, see the [Related Links](#) section at the bottom of this page.

```

<Authenticator name="SAML2SSOAuthenticator" disabled="false">
 <Priority>10</Priority>
 <Config>
 <Parameter name="LoginPage">/carbon/admin/login.jsp</Parameter>
 <Parameter name="ServiceProviderID">carbonserver</Parameter>
 <Parameter
name="IdentityProviderSSOServiceURL">https://localhost:9444/samlsso</Parameter>
 <Parameter
name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
 </Config>

```

Make sure the `<priority>` element of the `SAML2SSOAuthenticator` is less than that of the `BasicAuthenticator` handler. See [here](#) for more information.

## NamelDPolicyFormat

Service provider and Identity Provider usually communicate with each other about a subject. That subject should be identified through NAME-ID. It should be in some format so that it is easy for the other party to identify it based on the format. Possible values for `NamelDPolicyFormat` are as below.

1. urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified [default]
2. urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
3. urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
4. urn:oasis:names:tc:SAML:2.0:nameid-format:transient

We are using `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified` which is the default `NamelDPolicyFormat` and it is totally depend on the entities implementing on your own wish. For more information on NamelDidentifiers, refer section 8.3 of [SAML Core](#).

If there are many WSO2 products in your environment, you can configure SSO for the management consoles to gain one-time acces to all of them without repeated authentication. You can do this by changing the `SAML2SSOAuthenticator` configuration in the `<PRODUCT_HOME>/repository/conf/security/authenticators.xml` file as shown above.

### Configuring Publisher/Store for SSO

To configure SSO for the API Publisher, open the `<API-M_HOME>/repository/deployment/server/jagger/yapps/publisher/site/conf/site.json` file and give the configurations as shown below.

Note that following parameters should be matched with the Service Provider configurations of Identity Server that you are configuring.

```

ssoConfiguration" : {
 "enabled" : "true",
 "issuer" : "API_PUBLISHER",
 "identityProviderURL" : "https://localhost:9444/samlsso",
 "keyStorePassword" : "",
 "identityAlias" : "wso2carbon",
 "verifyAssertionValidityPeriod": "true",
 "timestampSkewInSeconds": "300",
 "audienceRestrictionsEnabled": "true",
 "responseSigningEnabled": "true",
 "assertionSigningEnabled": "true",
 "keyStoreName" : "",
 "signRequests" : "true",
 "assertionEncryptionEnabled" : "false",
 "idpInit" : "false",
 "idpInitSSOURL" :
 "https://localhost:9444/samlsso?spEntityID=API_PUBLISHER",
}

```

Parameter	Description
enabled	Set this value to <b>true</b> to enable SSO in the application.
issuer	API_PUBLISHER. This value can change depending on the <b>Issuer</b> value defined in WSO2 IS SSO configuration above.
identityProviderURL	<a href="https://localhost:9444/samlss">https://localhost:9444/samlss</a> . 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 <API-M_HOME>/repository/resources /security/wso2carbon.jks. <b>Be sure to give the full path of the keystore here.</b>
keyStorePassword	Password for the above keystore. The default keyStorePassword is wso2carbon.
identityAlias	wso2carbon

By default, in the ssoConfiguration in site.json described above signRequests parameter decide whether to sign the AuthRequest or not. We have enabled it by setting the value **true** in default configuration so that, all the AuthRequests passing will be signed.

The identityAlias parameter is set to wso2carbon in the above example. You can configure an external server by [importing the certificate](#) of the IdP to APIM, and changing the identityAlias parameter value according to the certificate. To configure an IDP initiated SSO, you have to include the following additional parameters in the ssoConfiguration section.

```

...
 "idpInit" : "true",
 "idpInitSSOURl" :
 "https://localhost:9444/samlss?spEntityID=API_PUBLISHER",
 "externalLogoutPage" : "https://localhost:9444/samlss?slo=true"
...

```

To configure SSO for the API Store, open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file and change the `ssoConfiguration` section similarly. This allows users to gain access to the Store applications directly, if they are authenticated against the Publisher.

`idpInitSSOURl` consists of <SAML2.IdPURL> and <SAML2.SPEntityId>.

Properties	Description
SAML2.IdPURL= https://localhost:9443/samlss	The URL of the SAML 2.0 Identity Provider
SAML2.SPEntityId=API_PUBLISHER	A unique identifier for this SAML 2.0 Service Provider application

The `SAML2.SPEntityId` should be the value of issuer you specify under `ssoConfiguration` in the <API-M\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file, which uniquely identifies your SAML identity provider. Therefore, it differs between the Store and Publisher according to the issuer that you configure.

### Configuring the API Store for SSO in passive mode

If the passive mode is disabled and Single Sign-On (SSO) is enabled, it redirects the user to the SSO login page. Therefore, as the WSO2 API Store allows anonymous access, passive mode is enabled by default, so that irrespective of whether SSO is enabled or not it directs the user to the API Store URL, and enables the SSO work flow only when the **Sign In** button is clicked.

To disable the passive mode, set the property named `passive` to `false` in the <API-M\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file.

```

"ssoConfiguration" : {
 ...
 "passive" : "false",
 ...
},

```

By enabling passive mode in SSO Configuration, WSO2 API Manager enables **Passive Authentication** on Single Sign On.

From the two fundamental authentication models which are active and passive, **active authentication** is based on WS-Trust protocol on which a relying party is responsible of issuing the security token associated with the user credentials. But in **passive authentication** which is based on SAML 2.0 and WS-Federation protocols, the relying party does not control the login logic and relies on the IdP to issue the credentials.

#### Related Links

- For instructions on how to configure WSO2 Identity Server as the IdP, see [Configuring Identity Server as IdP for SSO](#).
- For instructions on how to configure an external IdP, see [Configuring External IdP through Identity Server for SSO](#).
- For instructions on how to configure SAML2 Web Single-Sign-On, see [Configuring SAML2 Web Single-Sign-On](#)

## Configuring External IDP through Identity Server for SSO

The topics below explain the configurations you need to make to configure an external IDP through WSO2 Identity Server.

- [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 you need to point both WSO2 IS and WSO2 API Manager to a single user store.

You do this to make sure that a user who tries to log in to the API Manager console, the API Store or the API 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 `<API-M_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 the IS and API Manager.

Refer [Configuring User Stores](#) for more information on Configuring different types of user stores.

1. Create a MySQL database (e.g., `410_um_db`) and run the `<API-M_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 `<API-M_HOME>/dbscripts` directory.

`<API-M_HOME>/dbscripts/mysql.sql` is the script that should be used to run on the `410_um_db` database for MySQL 5.6 and prior versions. If your database is MySQL 5.7 or later version, use `<API-M_HOME>/dbscripts/mysql5.7.sql` script file.

2. Open the `<API-M_HOME>/repository/conf/datasources/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>

```

Refer [Configuring master-datasources.xml](#) for descriptive information about each property of the datasource configuration.

3. Add the same datasource configuration above to the <IS\_HOME>/repository/conf/datasources/master-datasources.xml file.
4. Copy the database driver JAR file to the <IS\_HOME>/repository/components/lib and <API-M\_HOME>/repository/components/lib directories.
5. Open the <API-M\_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 API Publisher 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 API Manager.

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 <API-M\_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;</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. Refer [Configuring registry.xml](#) for more information on the parameters used in configuring registry.xml

```

<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 `<API-M_HOME>/repository/conf/registry.xml` file as well.

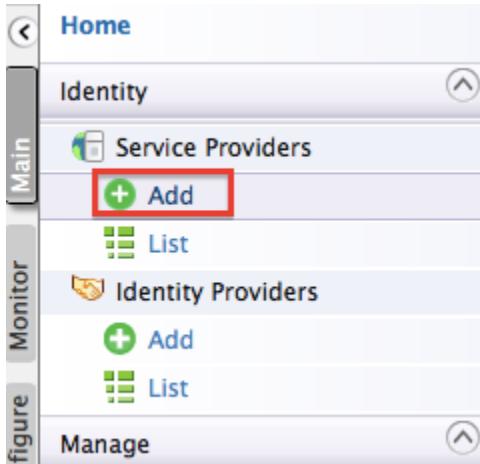
Next, let us look at the SSO configurations.

#### Configuring WSO2 Identity Server as a SAML 2.0 SSO Identity Provider

1. Start the IS server and log in to its Management Console (<https://localhost:9444/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/identity` directory.

2. Click **Add** under the **Service Providers** menu.



3. Give a service provider name and click **Register**.

## Add New Service Provider

### Basic Information

Service Provider Name:<sup>\*</sup>

API\_Manager

⑦ A unique name for the service provider

Description:

⑦ A meaningful description about the service provider

**Register**

Cancel

In a multi tenanted environment, for all tenants to be able to log in to the API Manager web applications, do the following:

- Click the **SaaS Application** option that appears after registering the service provider.

The screenshot shows the 'Basic Information' form for a service provider. The 'Service Provider Name:' field contains 'API\_Manager'. Below it is a descriptive text: 'A unique name for the service provider'. The 'Description:' field is empty. At the bottom of the form, there is a checkbox labeled 'SaaS Application' which is checked. A tooltip below the checkbox states: 'Applications are by default restricted for usage by users of the service provider's tenant. If this application is SaaS enabled it is opened up for all the users of all the tenants.' The entire form is enclosed in a light gray border.

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. Otherwise, only users in TA will be able to log in.

- Add the following inside the `<SSOService>` element in the `<IS_HOME>/repository/conf/identity/identity.xml` file and restart the server.

```
<SSOService>
<UseAuthenticatedUserDomainCrypto>true</UseAuthenticatedUserDomainCrypto>
...
</SSOService>
```

If not, you get an exception stating that the SAML response signature verification fails.

- Since 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. Inside the **Inbound Authentication Configuration** section, expand **SAML2 Web SSO Configuration** and click **Configure**.

## Service Providers

**Basic Information**

Service Provider Name: ⑦ A unique name for the service provider

Description:  
 ⑦ A meaningful description about the service provider

SaaS Application  ⑦ Applications are by default restricted for usage by users of the service provider's tenant. If this application is SaaS enabled it is opened up for all the users of all the tenants.

Claim Configuration  
 Role/Permission Configuration  
 **Inbound Authentication Configuration**

SAML2 Web SSO Configuration

**Configure**

To enable tenant specific SSO with IS 5.3.0 for API\_PUBLISHER and API\_STORE, enable **Use tenant domain in local subject identifier** under the Local & Outbound Authentication Configuration section.

**Local & Outbound Authentication Configuration**

Authentication Type: Default  
 Local Authentication

Federated Authentication  
 Advanced Configuration

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

Use user store domain in local subject identifier

Request Path Authentication Configuration

- Provide the configurations to register the API Publisher as the SSO service provider. These sample values may change depending on 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:
    - Enable Response Signing**
    - Enable Single Logout**
    - Enable Attribute Profile**
      - Include Attributes in the Responses Always**
    - Click Register once done.**

For example:

## Register New Service Provider

Select Mode

- Manual Configuration
- Metadata File Configuration
- URL Configuration

Manual Configuration

Issuer *	API_PUBLISHER
Assertion Consumer URLs *	<input type="text"/> <input type="button" value="Add"/> <div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;"> <a href="https://localhost:9443/publisher/jagg/jaggery_acs.jag">https://localhost:9443/publisher/jagg/jaggery_acs.jag</a> <span style="float: right;"></span> </div>
Default Assertion Consumer URL *	<input type="text" value="https://localhost:9443/publisher/jagg/jaggery_acs.jag"/>
NameID format	urn:oasis:names:tc:SAML:1.1:nameid-form
Certificate Alias	wso2carbon
Response Signing Algorithm *	<input type="text" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
Response Digest Algorithm *	<input type="text" value="http://www.w3.org/2000/09/xmldsig#sha1"/>
<input checked="" type="checkbox"/> Enable Response Signing <input checked="" type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests <input type="checkbox"/> Enable Assertion Encryption <input checked="" type="checkbox"/> Enable Single Logout	
SLO Response URL	<input type="text"/> <small> ⓘ Single logout response accepting endpoint</small>
SLO Request URL	<input type="text"/> <small> ⓘ Single logout request accepting endpoint</small>
<input checked="" type="checkbox"/> Enable Attribute Profile <input checked="" type="checkbox"/> <i>Include Attributes in the Response Always</i>	
<input type="checkbox"/> Enable Audience Restriction Audience <input type="text"/> <input type="button" value="Add"/>	
<input type="checkbox"/> Enable Recipient Validation Recipient <input type="text"/> <input type="button" value="Add"/>	
<input type="checkbox"/> Enable IdP Initiated SSO <input type="checkbox"/> Enable IdP Initiated SLO Return to URL <input type="text"/> <input type="button" value="Add"/>	
<input type="checkbox"/> Enable Assertion Query Request 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](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:
  - **Enable Response Signing**
  - **EnableSingle Logout**
  - **Enable Attribute Profile**

- **Include Attributes in the Responses Always**

- Click **Register** once done.

7. Make sure that the `responseSigningEnabled` element is set to `true` in both the following files:

This is used to sign the SAML2 Responses returned after the authentication process is complete.

- `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json`
- `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json`

8. Add a new Identity Provider in WSO2 Identity Server. For more details on configuring external IDPs in WSO2 IS, see [Configuring an Identity Provider](#).

- Identity Provider Name: ExternalIS
- Do the following changes under Federated Authenticators > SAML2 Web SSO Configurations
  - **Enable SAML2 Web SSO**
  - Check **Default**
  - Set **Service Provider Entity ID**
  - Set SSO URL for the external IDP (e.g. `https://localhost:9453/samlsso`)
  - **Enable Logout**

9. Enable JIT Provisioning for the external IDP. For more information, see [Configuring Just-In-Time Provisioning for an Identity Provider](#).

10. Map the external IDP roles to the roles configured in API Manager. For more information on mapping roles, see [Configuring Roles for an Identity Provider](#).

The screenshot shows the 'Role Configuration' interface. Under 'Identity Provider Roles:', there are two buttons: '+ Add Role Mapping' and 'Add identity provider role mapping'. Below this is a table with columns 'Identity Provider Role' and 'Local Role'. Two rows are listed: 'DevUsers' mapped to 'INTERNAL/subscriber' and 'PubUsers' mapped to 'INTERNAL/publisher'. Each row has a 'Delete' button in the 'Actions' column. At the bottom, there's a note: 'Enter the identity provider's comma separated provisioning roles name'.

Identity Provider Role	Local Role	Actions
DevUsers	INTERNAL/subscriber	
PubUsers	INTERNAL/publisher	

11. Open the management console, and click **Edit** under **Service Providers**.

12. Under **Local & Outbound Authentication Configuration** select **Federated Authentication**. Select the newly created external IDP.

The screenshot shows the 'Local & Outbound Authentication Configuration' page. Under 'Authentication Type:', there are four options: 'Default', 'Local Authentication' (with dropdown 'totp'), 'Federated Authentication' (selected), and 'Advanced Configuration'. The 'Federated Authentication' option and its dropdown are highlighted with a red box.

13. Add `http://wso2.org/claims/role` as the Claim URI under **Claim Configuration**. Select the **Mandatory Claim** check box. Add `http://wso2.org/claims/username` as the Subject Claim URI.

The screenshot shows the 'Claim Configuration' interface. Under 'Requested Claims', there are two entries:

- Local Claim:** http://wso2.org/claims/role
- Subject Claim URI:** http://wso2.org/claims/username

Both entries have a red box drawn around them. To the right of each entry is a checkbox labeled 'Mandatory Claim', which is checked for both.

Additionally, you might need to configure claims to map them to the available claims in WSO2 Identity Server. For more details, see [Configuring Claims for an Identity Provider](#).

## Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

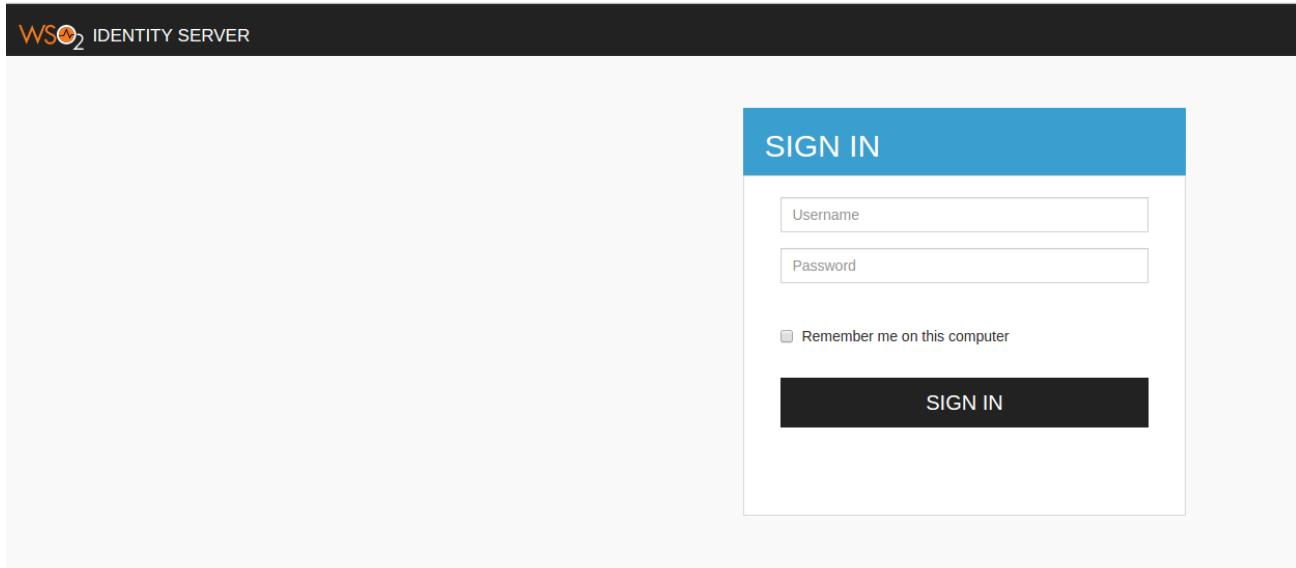
1. Open the <API-M\_Home>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json and modify the following configurations found under **ssoConfiguration**.
  - **enabled:** Set this value to **true** to enable SSO in the application
  - **issuer:** API\_PUBLISHER. This value can change depending on the **Issuer** value defined in the WSO2 IS SSO configuration above.
  - **identityProviderURL:** <https://localhost:9444/samlsso>. 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 API Manager and then point to the API Manager keystore. The default keystore of the API Manager is <API-M\_HOME>/repository/resources/security/wso2carbon.jks. **Make sure you 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 <API-M\_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>
```

You can skip this step if you are using Identity Server 5.3.0 as the IDP.

4. Access the API Publisher (e.g., <https://localhost:9443/publisher>). Observe the request redirecting to the WSO2 IS SAML2.0 based SSO login page. For example,



5. Enter the user credentials. If the user authentication is successful against WSO2 IS, it redirects to the API Publisher with the user that is already authenticated.
6. Access the API Store, click its **Login** link (top, right-hand corner) and verify that the same user is already authenticated in the API Store.

Even with SSO enabled, if the user doesn't have sufficient privileges to access the API Publisher/Store or any other application, s/he will not be authorized to access them.

To learn more about Single Sign-On with WSO2 Identity Server, see [SAML 2.0 Web SSO](#) in the WSO2 Identity Server documentation.

## Configuring Identity Server as IDP for SSO

The **Single Sign-On with SAML 2.0** feature in the API Manager is implemented according to the SAML 2.0 browser-based SSO support that is facilitated by WSO2 Identity Server (WSO2 IS). This feature is available in any WSO2 IS version from 4.1.0 onwards. We use **WSO2 IS 5.3.0** in this guide. WSO2 Identity Server acts as an identity service provider of systems enabled with single sign-on, while the Web applications act as SSO service providers. Using this feature, you can configure SSO across the API Publisher and Store. After configuring, you can access the API Store or API Publisher in a single authentication attempt.

The topics below explain the configurations.

In this documentation, MySQL is used as the database to configure WSO2 API Manager with WSO2 Identity Server. For instructions on replacing the default H2 database with MySQL, see [Setting up MySQL](#).

- 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

Initially, configure your user store(s), if you have not done so already, by following the instructions in [Configuring User Stores](#). Thereafter, point both WSO2 IS and WSO2 API Manager to your user stores(s) using the instructions given below. 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 <API-M\_HOME>/repository/conf/user-mgt.xml file) where the user's role and permissions are stored.

For example, let's share a JDBC user store (MySQL) with both the WSO2 Identity Server and WSO2 API Manager as follows:

1. Download WSO2 API Manager 2.1.0 from [here](#) and unzip it. <API-M\_HOME> refers to the root folder where WSO2 API-M was unzipped.
2. Create a MySQL database (e.g., 410\_um\_db) and run the <API-M\_HOME>/dbscripts/mysql.sql script on it to create the required tables.

There are two MySQL DB scripts available in the product distribution from WSO2 Carbon Kernel 4.4.6 onwards. Click [here](#) to identify as to which version of the MySQL script to use. If you are using a different database type, find the relevant script from the <API-M\_HOME>/dbscripts directory.

3. Open the <API-M\_HOME>/repository/conf/datasources/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, you can share as single user store as follows. If you are sharing multiple datasources, you need to define a datasource for each of the user stores that you are working with, so that they can be shared.

#### 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>
```

Change the database url to the url of the MySQL database you have created above. Modify the username and password parameters in above configuration with your mysql database credentials.

Refer [Configuring master-datasources.xml](#) for descriptive information about each property of the

datasource configuration.

- Download WSO2 Identity Server (WSO2 IS) 5.3.0 from [here](#) and unzip it. <IS\_HOME> refers to the root folder where WSO2 IS was unzipped.

To use WSO2 IS as the Key Manager , download the **WSO2 Identity Server 5.3.0 as a Key Manager** pack, with pre-packaged Key Manager features, from [here](#).

- Add the same datasource configuration above to <IS\_HOME>/repository/conf/datasources/master-datasources.xml file.
- Copy the database driver JAR file to the <IS\_HOME>/repository/components/lib and <API-M\_HOME>/repository/components/lib directories.
- Open the <API-M\_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>
```

- Add the same configuration above to the <IS\_HOME>/repository/conf/user-mgt.xml file.
- 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.

In WSO2 API Manager, the JDBC User Store is enabled by default. By changing the default user store of WSO2 Identity server to JDBC User Store, we are pointing both WSO2 API Manager and WSO2 Identity Server to the same user store so that, their user stores are shared.

## 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.

- 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.

There are two MySQL DB scripts available in the product distribution from WSO2 Carbon Kernel 4.4.6 onwards. Click [here](#) to identify as to which version of the MySQL script to use.

- Add the following datasource configuration to both the <IS\_HOME>/repository/conf/datasources/master-datasources.xml and <API-M\_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&</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>

```

Modify the username and password parameters of above configuration with your mysql database credentials.

Refer [Configuring master-datasources.xml](#) for descriptive information about each property of the datasource configuration.

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>

```

Refer [Configuring registry.xml](#) for more details on configuration details and usage of registry.xml

4. Repeat the above step in the <API-M\_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 WSO2 Identity Server.

```
./wso2server.sh -DportOffset=1
```

You also can change Port offset value by changing <Offset> 1 </Offset> under <Ports> in <IS\_HOME>/repository/conf/carbon.xml file.

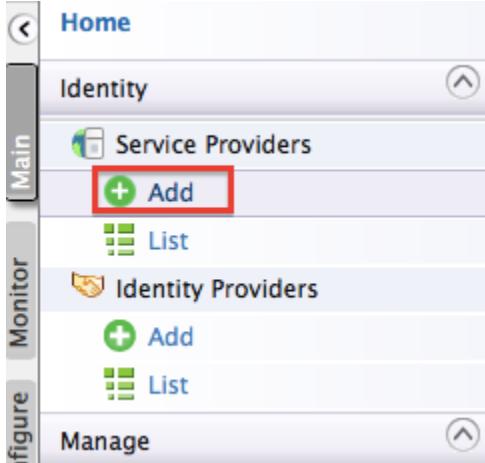
## What is port offset?

The port offset feature allows you to run multiple WSO2 products, multiple instances of a WSO2 product, or multiple WSO2 product clusters on the same server or virtual machine (VM). The port offset defines the number by which all ports defined in the runtime, such as the HTTP/S ports, will be offset. For example, if the HTTPS port is defined as 9443 and the portOffset is 1, the effective HTTPS port will be 9444.

2. Sign in to the WSO2 IS Management Console UI (<https://localhost:9444/carbon>).

If you use signin pages that are hosted externally to sign 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/identity` directory.

3. Select **Add** under the **Service Providers** menu.



4. Give a service provider name and click **Register**.

Add New Service Provider

**Basic Information**

Service Provider Name:*	<input type="text" value="API_PUBLISHER"/> <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.

Service Providers

**Basic Information**

Service Provider Name:*	<input type="text" value="API_PUBLISHER"/> <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="checkbox"/> <small>SaaS Application</small> <small>⑦ Applications are by default restricted for usage by users of the service provider's tenant. If this application is SaaS enabled it is opened up for all the users of all the tenants.</small>	

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/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. Inside the **Inbound Authentication Configuration** section, expand **SAML2 Web SSO Configuration** and click **Configure**.

To enable tenant specific SSO with IS 5.3.0 for API\_PUBLISHER and API\_STORE, enable **Use tenant domain in local subject identifier** under the Local & Outbound Authentication Configuration section.

6. 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](https://localhost:9443/publisher/jagg/jaggery_acs.jag). Change the IP and port accordingly. This is the URL for the Assertion Consumer Services (ACS) page in your running publisher app.
  - Select the following options:
    - **Enable Response Signing**
    - **Enable Single Logout**
  - Click **Register** once done.

For example:

### Register New Service Provider

New Service Provider

Issuer *	API_PUBLISHER
Assertion Consumer URLs *	<a href="https://localhost:9443/publisher/jagg/jaggery">https://localhost:9443/publisher/jagg/jaggery</a> <input type="button" value="Add"/>
Default Assertion Consumer URL *	---Select---
NameID format	urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress
Certificate Alias	wso2carbon.cert
Response Signing Algorithm *	<a href="http://www.w3.org/2000/09/xmldsig#rsa-sha1">http://www.w3.org/2000/09/xmldsig#rsa-sha1</a>
Response Digest Algorithm *	<a href="http://www.w3.org/2000/09/xmldsig#sha1">http://www.w3.org/2000/09/xmldsig#sha1</a>
<input checked="" type="checkbox"/> Enable Response Signing	
<input type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests	
<input type="checkbox"/> Enable Assertion Encryption	
<input checked="" type="checkbox"/> Enable Single Logout	
SLO Response URL	<input type="text"/> <small>② Single logout response accepting endpoint</small>
SLO Request URL	<input type="text"/> <small>② Single logout request accepting endpoint</small>
<input type="checkbox"/> Enable Attribute Profile	

7. 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](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:
    - **Enable Response Signing**
    - **Enable Single Logout**
  - Click **Register** once done.
8. Make sure that the `responseSigningEnabled` element is set to `true` in both the following files:
- `<API-M_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json`
  - `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json`

### Configuring WSO2 API Manager apps as SAML 2.0 SSO service providers

1. Open <API-M\_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/samlsso>. 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 <API-M\_HOME>/repository/resources/security/wso2carbon.jks. **Be sure to give the full path of the keystore here.**
  - **keyStorePassword:** Password for the above keystore. The default keyStorePassword is wso2carbon.
  - **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 <API-M\_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>
```

You can skip this step if you are using Identity Server 5.3.0 as the IDP.

4. Access the API Publisher: [https://localhost:<port\\_number>/publisher](https://localhost:<port_number>/publisher) (e.g., <https://localhost:9443/publisher>). Observe the request redirect to the WSO2 IS SAML2.0 based SSO login page. For example,

 WSO2 IDENTITY SERVER

## SIGN IN

Username

Password

Remember me on this computer

SIGN IN

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.

To learn more about Single Sign-On with WSO2 Identity Server, see [SAML 2.0 Web SSO](#) in the WSO2 Identity Server documentation.

## Changing the Default Transport

On the back end, APIs are Apache Synapse configurations that WSO2 API Manager accesses through a transport. The default API Manager transport is the PassThrough transport, but you can configure a different default transport in your axis2.xml file. For example, to use the HTTP-NIO transport as the default, go to the <APIM\_HOME>/repository/conf/axis2 folder, open the axis2.xml file, and then in the "Transport Ins" and "Transport Outs" sections, comment out the PassThrough configurations and uncomment the configurations for the HTTP-NIO transport.

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.

The following topics provide more information on these transports:

- [HTTP PassThrough transport](#)
- [HTTP-NIO transport](#)
- [Transport receiver parameters](#)
- [Transport sender parameters](#)
- [Connection throttling](#)

### *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 forwarding messages back and forth. The two classes that implement the receiver and sender APIs are `org.apache.synapse.transport.passthru.PassThroughHttpListener` and `org.apache.synapse.transport.passthru.PassThroughHttpSender`, respectively. The PassThrough Transport does not require the binary relay builder and expanding formatter.

### *HTTP-NIO transport*

The HTTP-NIO transport is a module of the Apache Synapse project. Apache Synapse ships the HTTP-NIO transport as the default, non-blocking 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.HttpCoreNIOSender`, respectively. These classes are available in the JAR file named `synapse-nhttp-transport.jar`. 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. The PassThrough transport is the preferred default transport for WSO2 API Manager, but HTTP-NIO is supported for backward compatibility.

### *Transport receiver parameters*

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	true or false	true
bind-address	The address of the interface to which the transport listener should bind.	No	A host name or an IP address	127.0.0.1
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>/	
httpGetProcessor	An extension point used to execute a special interceptor for HTTP GET requests.	Yes	An extension point	org.wso2.carbon.mediation.PassthroughNHttpGetProc
priorityConfigFile	The location of the file containing the configuration for priority based dispatching.	No	A file location	

### Transport sender parameters

Parameter Name	Description	Required	Possible Values
non-blocking	Setting this parameter to true is vital for reliable messaging and a number of other scenarios to work properly.	Yes	true or false
warnOnHTTP500	Logs warnings for HTTP 500 responses only for the specified content-types. For example, <parameter name="warnOnHTTP500" locked="false">x-application/hessian none</parameter> would log warnings for HTTP 500 responses of content-type 'x-application/hessian' or messages missing a content-type.	No	A list of content types separated by " "
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 and an 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 and IP addresses separated by ' '

### 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.

### 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 (i.e., access token, API name, 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. WSO2 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.

The default cache size of any type of cache in a WSO2 product is 10,000 elements/records. Cache eviction occurs from the 10001th element. All caches in WSO2 products can be configured using the `<PRODUCT_HOME>/repository/conf/carbon.xml` file. The `defaultCacheTimeout` is 15 minutes which comes by default.

```
<Cache>
 <!-- Default cache timeout in minutes -->
 <DefaultCacheTimeout>15</DefaultCacheTimeout>
</Cache>
```

These configurations apply globally to all caches. You can override these values for specific caches using the UI or different configuration files as discussed under each section below.

This section covers the following:

- [API Gateway cache](#)
- [Resource cache](#)
- [Key Manager cache](#)
- [Response cache](#)
- [API Store cache](#)

In a distributed environment, the caching configurations you do in one node replicates equally in all nodes.

Apart from response caching, all the other caches are enabled by product. When the WSO2 API Manager components are clustered, they work as distributed caches. This means that a change done by one node is visible to another node in the cluster.

#### ***API Gateway cache***

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 the cache, it calls the Key Manager server. This process is carried out using Web service calls. After the Key Manager server returns the validation information, it gets stored in the Gateway. As 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 because the `<EnableGatewayTokenCache>` element is set to true in the `<API-M_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayTokenCache>true</EnableGatewayTokenCache>
```

If you need to enable Gateway caching across the entire cluster, see [Working with Hazelcast Clustering](#).

#### ***Clearing the API Gateway cache***

If you wish to remove old tokens that might still remain active in the Gateway cache, you need to configure the `<RevokeAPIURL>` element in the `<API-M_HOME>/repository/conf/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 **API Store 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.APIManagerCacheExtensionHandler"/>
 </handlers>
</api>
```

### Resource cache

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						
		Scopes:	Add Scopes			
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	
PATCH	/*	+ Summary	Application & Application User	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 update an API, the resource cache gets invalidated and the changes are reflected within a few minutes.

By default, the resource cache is enabled as the `<EnableGatewayResourceCache>` element is set to true in the `<APIM_HOME>/repository/conf/api-manager.xml` file:

```
<EnableGatewayResourceCache>true</EnableGatewayResourceCache>
```

### **Key Manager cache**

The following caches are available:

- [Key cache](#)
- [OAuth cache](#)

### **Key cache**

In a typical API Manager deployment, the Gateway is deployed in a DMZ while the Key Manager is in MZ. By default, caching is enabled at the Gateway. To avoid caching token-related information in a leniently secured zone, you can store the cache on the Key Manager side. If you do, 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 is checked for the validity of the token.

Storing the cache in the Key Manager causes lower performance than when storing it in the Gateway, but it is more secure. If you enable the key cache in a clustered environment, you should have only one Gateway per Key Manager, whereas you can have two Gateways per Key Manager when the Gateway cache is enabled instead. Note that you should always have one of the caches enabled, but we do not recommend using both caches combined. For more information, see [Clustering Gateways and Key Managers with key caching](#) in the WSO2 Clustering Guide.

You configure the key cache by editing the following elements in the `<APIM_HOME>/repository/conf/api-manager.xml` file:

Purpose	Configuration Elements
Disable the API Gateway cache.	Under <code>&lt;CacheConfigurations&gt;</code> , set <code>&lt;EnableGatewayTokenCache&gt;false&lt;/EnableGatewayTokenCache&gt;</code>
Enable the Key Manager cache.	Under <code>&lt;CacheConfigurations&gt;</code> , set <code>&lt;EnableKeyManagerTokenCache&gt;true&lt;/EnableKeyManagerTokenCache&gt;</code>
Change the key cache duration, which expires after 900 seconds by default.	<code>&lt;Token Cache Expiry&gt;900&lt;/Token Cache Expiry&gt;</code>

### **OAuth cache**

The OAuth token is saved in this cache, which is enabled by default. Whenever a new OAuth token is generated, it is saved in this cache to prevent constant database calls. Unless an OAuth expires or is revoked, the same token is sent back for the same user. Therefore, you do not need to change this cached token most of the time.

### **Response cache**

The API Manager uses [WSO2 ESB's cache mediator](#) to cache response messages for each API. Caching improves performance, because the backend server does not have to process the same data for a request multiple times. You need to set an appropriate timeout period to offset the risk of stale data in the cache.

You need to enable response caching when creating a new API or editing an existing API using the API Publisher. Go to the API Publisher and click **Add New API** (to create a new API) or click the **Edit** icon associated with an existing API. Then, navigate to the **Manage** tab where you find the response caching section. You can set Response caching to **Enabled** and give a timeout value. This enables the default response caching settings.

The screenshot shows the WSO2 API Publisher interface with the 'Manage' tab selected. In the 'Configurations' section, there is a 'Response Caching' dropdown set to 'Enabled' and a 'Cache Timeout (seconds)' input field containing '300'. Both of these fields are highlighted with a red box.

To change the default response caching settings, edit the following cache mediator properties in the `<API-M_HOME>/repository/resources/api_templates/velocity_template.xml` file:

Property	Description
collector	<ul style="list-style-type: none"> <li>• true: specifies that the mediator instance is a response collection instance.</li> <li>• false: specifies that the mediator instance is a cache serving instance.</li> </ul>
maxMessageSize	Specifies the maximum size of a message to be cached in bytes. An optional attribute, with the default value set to unlimited.
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 <a href="#">here</a>.</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>&lt;cache&gt;</code> element in the <code>velocity_template.xml</code> file.</p>

When running a distributed deployment, you need to enable the stream builders on the API Gateway and maintain the standard builders on the API Store.

Follow the instructions below to enable the stream builders in the API gateway:

1. Open the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file.
2. Disable the default standard builders by commenting the following.

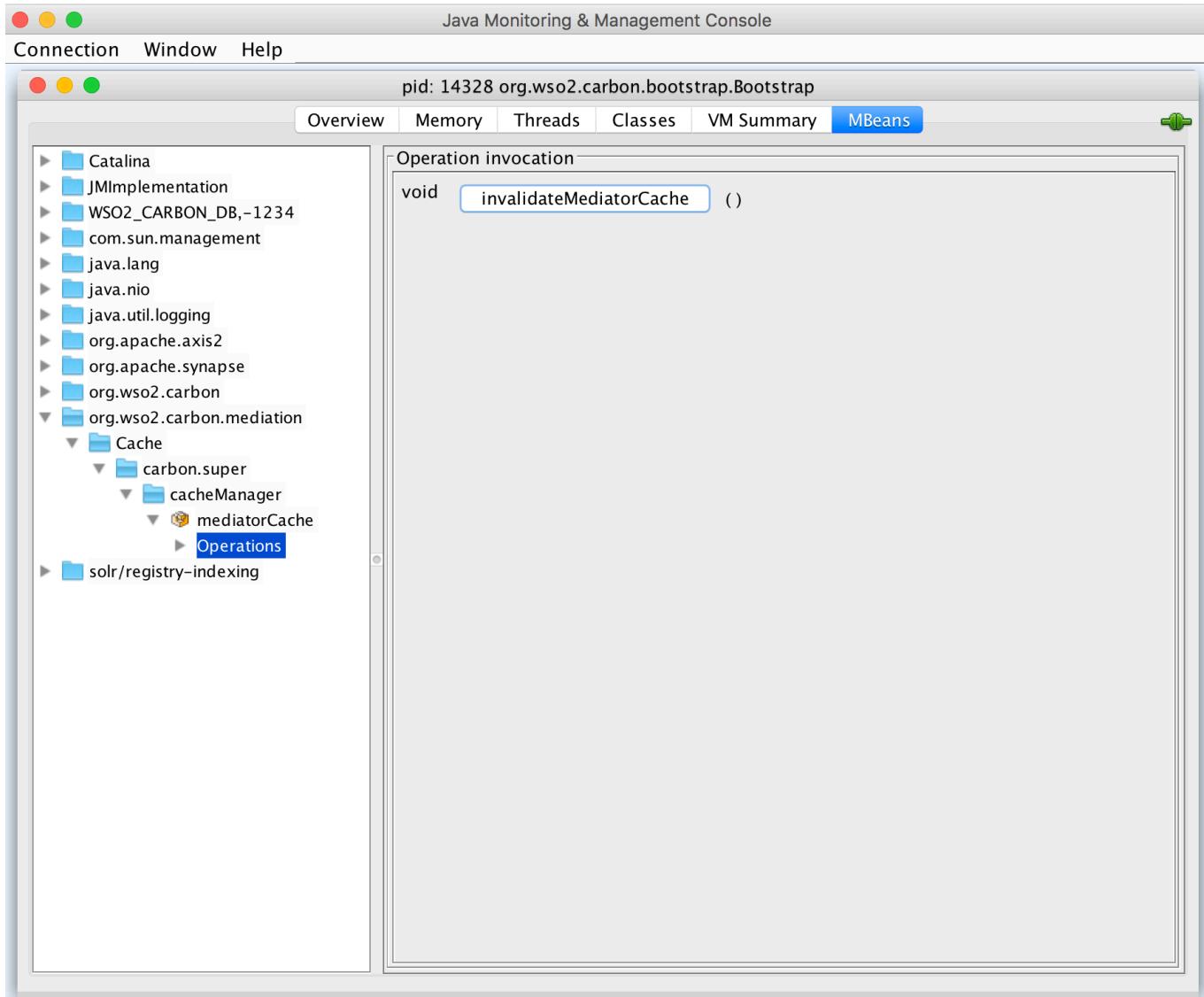
```
!--messageBuilder contentType="application/json"
class="org.apache.synapse.commons.json.JsonBuilder"-->
```

3. Enable the stream builders by uncommenting the following.

```
<messageBuilder contentType="application/json"
class="org.apache.synapse.commons.json.JsonStreamBuilder"/>
```

#### Invalidating Cached Responses Remotely

You can invalidate all cached response remotely by using any JMX monitoring tool such as Jconsole using the exposed MBeans. You can use the `Invalidatemediatocache()` operation of the `org.wso2.carbon.mediation.MBean` for this as shown below.



#### API Store cache

The API Store has several caches to reduce the page-load times and increase its responsiveness when multiple users access it simultaneously.

- **Tag cache:** This cache saves the API's tags after they have been retrieved from registry. If your APIs and associated tags change frequently, it is recommended to configure a smaller cache refresh time (in milliseconds). This cache disabled by default. To enable it, uncomment the `<TagCacheDuration>` element

in the <APIM\_HOME>/repository/conf/api-manager.xml file.

- **Recently-added-API cache:** This cache saves the five most recently added APIs. It is disabled by default. If you have multiple API modifications during a short time period, it is recommended to not enable this cache. To enable it, set the <EnableRecentlyAddedAPICache> to true in the <APIM\_HOME>/repository/conf/api-manager.xml file.

## Prevent API Suspension

WSO2 API Manager suspends your API if the endpoint of your API cannot be reached. The default suspension time is 30 seconds. Any request to your API will not be able to reach your endpoint for 30 seconds and will return an error message as shown below.

```
<am:fault xmlns:am="http://wso2.org/apimanager">
<am:code>303001</am:code>
<am:type>Status report</am:type>
<am:message>Runtime Error</am:message>
<am:description>Currently , Address endpoint : [Name : somename-AT-sometenant--test_me_APIproductionEndpoint_0] [State : SUSPENDED]</am:description>
</am:fault>
```

To prevent or turn off API suspension do the following:

1. Log into API Publisher (<https://<HostName>:9443/publisher>). Select your API and click **Edit API**.

The screenshot shows the WSO2 API Publisher interface. At the top, there is a navigation bar with buttons for 'GO BACK', 'EDIT API' (which is highlighted with a red box), 'CREATE NEW VERSION', and 'DOWNLOAD SOURCE'. Below the navigation bar, the API name 'Petstore - 1.0.0' is displayed. Underneath the API name, there is a navigation menu with tabs: 'Overview' (selected), 'Lifecycle', 'Versions', 'Docs', and 'Users'. On the left side, there is a large red square icon with a white letter 'P' and a small blue user icon below it with the text '0 Users'. In the center, there is a 'Description' section containing a truncated text: 'A sample API that uses a petstore as an example to demonstrate features in th...'. Below the description, there is a table with two rows: 'Visibility' (Public) and 'Context' (/petstore/1.0.0).

2. Go to **Implement** and click the cogwheel icon next to the endpoint you want to re-configure.

**Petstore: /petstore/1.0.0**

1 Design      2 Implement      3 Manage

**Managed API**  
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type :\*   Load Balanced  Failover

Production Endpoint :\*

Sandbox Endpoint :\*

Show More Options

3. In the Advanced Settings dialog box that appears, change **Initial Duration** and **Max Duration**. To turn off suspension, set both values to zero.

**Advanced Endpoint Configuration**

#### Endpoint Suspend State

Error Codes  A list of error codes. If these error codes are received from the endpoint, the endpoint will be suspended.

Factor

#### Endpoint Timeout State

Error Codes

Retries Before Suspension  Retry Delay(ms)

#### Connection Timeout

Action

Duration (ms)

4. Click Save and re-publish the API.

For more details about creating and publishing an API, see [Create and Publish an API](#).

If you are using WSO2 ESB, do the following to avoid backend endpoint suspension.

1. Go to <APIM\_HOME>/repository/deployment/server/synapse-configs/default/api
2. Open the configuration file of the API, that has to be prevented from being suspended. (e.g. admin--testApi\_v1.0.0.xml)
3. Add the following configurations

```
<endpoint name="admin--testApi_APIproductionEndpoint_0">
 <address
 uri="http://localhost:9000/services/SimpleStockQuoteService">
 <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>
 </address>
</endpoint>
```

For more details on configuring different timeouts, see [Timeout configurations for an API call](#) in the Performance guide.

## Working with Databases

The default database that WSO2 API Manager uses to store registry, user manager and product-specific data is the **wso2carbon\_db.h2.db** H2 database located in the <PRODUCT\_HOME>/repository/database folder.

This embedded H2 database is suitable for development and testing. 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 database:

- [Setting up the Physical Database](#)
- [Managing Datasources](#)

## Managing Datasources

A datasource provides information that a server can use to connect to a database. Datasource management is provided by the following feature in the WSO2 feature repository:

Name : WSO2 Carbon - datasource	management	feature
Identifier: org.wso2.carbon.datasource.feature.group		

If datasource management capability is not included in your product by default, add it by installing the above feature, using the instructions given under the Feature Management section of this documentation.

Click **Data Sources** on the **Configure** tab of the product's management console to view, edit, and delete the

datasources in your product instance.

You can view, edit, and delete the datasources in your product instance by clicking **Data Sources** on the **Configure** tab of the product management console. However, you cannot edit or delete the default <WSO2\_CARBON\_DB> datasource.

## 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\*

Description

Data Source Provider\* default

Driver\*

URL\*

User Name

Password

Expose as a JNDI Data Source

Data Source Configuration Parameters

This is the default RDBMS datasource configuration provided by WSO2. You can also write your own RDBMS configuration by selecting the custom datasource option. Enter values for the following fields when using the default RDBMS datasource configuration:

- **Data Source Type:** RDBMS
- **Name:** Name of the datasource (must be a unique value)
- **Data Source Provider:** Specify the datasource provider.
- **Driver:** The class name of the JDBC driver to use. Make sure to copy the JDBC driver relevant to the database engine to the <PRODUCT\_HOME>/repository/components/lib/ directory. For example, if you are using MySQL, specify com.mysql.jdbc.Driver as the driver and copy mysql-connector-java-5.5.25-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*	rdbmsdatasource
Description	RDBMS Data Source
Data Source Provider*	default
Driver*	com.mysql.jdbc.Driver
URL*	jdbc:mysql://localhost:3306/test
User Name	root
Password	*****
<input type="checkbox"/> Expose as a JNDI Data Source	
<input type="checkbox"/> Data Source Configuration Parameters	
<input type="button" value="Test Connection"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/>	

External datasource provider

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

New Data Source

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

### Configuring a JNDI Datasource

Java Naming and Directory Interface (JNDI) is a Java Application Programming Interface (API) that provides naming and directory functionality for Java software clients, to discover and look up data and objects via a name. It helps decoupling object creation from the object look-up. When you have registered a datasource with JNDI, others can discover it through a JNDI look-up and use it.

When adding a datasource, to expose a RDBMS datasource as a JNDI datasource, click **Expose as a JNDI Data Source** to display the JNDI fields as follows:

## New Data Source

The screenshot shows the 'New Data Source' configuration interface. The 'Data Source Type' is set to 'RDBMS'. The 'Name' field is empty. The 'Description' field is also empty. The 'Data Source Provider' is set to 'default'. The 'Driver' field is empty. The 'URL' field is empty. The 'User Name' is 'admin' and the 'Password' is masked as '\*\*\*\*\*'. A red box highlights the 'Expose as a JNDI Data Source' section, which contains fields for 'Name' (empty), 'Use Data Source Factory' (unchecked), and 'JNDI Properties' (with an 'Add Property' button). Below this is the 'Data Source Configuration Parameters' section, which is collapsed. At the bottom are 'Test Connection', 'Save', and 'Cancel' buttons.

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	

The screenshot shows a configuration dialog box. At the top left is the text "Alternate User Name Allowed". To its right is a dropdown menu with the value "false" selected. Below the dropdown are three buttons: "Test Connection", "Save", and "Cancel".

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"> <li>• TRANSACTION_UNKNOWN</li> <li>• TRANSACTION_NONE</li> <li>• TRANSACTION_READ_COMMITTED</li> <li>• TRANSACTION_READ_UNCOMMITTED</li> <li>• TRANSACTION_REPEATABLE_READ</li> <li>• TRANSACTION_SERIALIZABLE</li> </ul>
Initial Size (int)	The initial number of connections created, when the pool is started. Default value is zero.
Max. Active (int)	Maximum number of active connections that can be allocated from this pool at the same time. The default value is 100.
Max. Idle (int)	Maximum number of connections that should be kept in the pool at all times. Default value is 8. Idle connections are checked periodically (if enabled), and connections that have been idle for longer than <code>minEvictableIdleTimeMillis</code> will be released. (also see <a href="#">testWhileIdle</a> )
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 <a href="#">testWhileidle</a> .
Max. Wait (int)	Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception. Default value is 30000 (30 seconds).
Validation Query (String)	The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are SELECT 1 (mysql), select 1 from dual (oracle), SELECT 1 (MS Sql Server).
Test On Return (boolean)	<p>Used to indicate if objects will be validated before returned to the pool. The default value is false.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.</p> </div>
Test On Borrow (boolean)	<p>Used to indicate if objects will be validated before borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. Default value is false.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>For a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. In order to have a more efficient validation, see <code>validationInterval</code> .</p> </div>

Test While Idle (boolean)	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 <a href="#">timeBetweenEvictionRunsMillis</a> .
	For a true value to have any effect, the <a href="#">validationQuery</a> parameter must be set to a non-null string.
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 <a href="#">removeAbandonedTimeout</a> . If set to true, a connection is considered abandoned and eligible for removal, if it has been in use longer than the <a href="#">removeAbandonedTimeout</a> . Setting this to true can recover database connections from applications that fail to close a connection. For more information, see <a href="#">logAbandoned</a> . 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 <a href="#">setAutoCommit</a> 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 <a href="#">setReadOnly</a> 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)	<p>Connection properties that will be sent to our JDBC driver when establishing new connections. Format of the string must be [propertyName=property; ]*. The default value is null.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>The <code>user</code> and <code>password</code> properties will be passed explicitly, so that they do not need to be included here.</p> </div>
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.

<b>Alternate User Name Allowed (boolean)</b>	<p>By default, the <code>jdbc-pool</code> will ignore the <code>DataSource.getConnection(username, password)</code> call, and simply return a previously pooled connection under the globally configured properties <code>username</code> and <code>password</code>, for performance reasons.</p> <p>The pool can however be configured to allow use of different credentials each time a connection is requested. To enable the functionality described in the <code>DataSource.getConnection(username, password)</code> call, simply set the property <code>alternateUsernameAllowed</code>, to true. If you request a connection with the credentials user1/password1, and the connection was previously connected using different user2/password2, then the connection will be closed, and reopened with the requested credentials. This way, the pool size is still managed on a global level, and not on a per-schema level. The default value is false.</p>
----------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Configuring a Custom Datasource

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

New Data Source

Data Source Type\* **Custom**

Custom Data Source Type\*

Name\*

Description

Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tabs SYSTEM "http://www.w3.org/1999/xhtml">
<tab>
<label>General</label>
<content>
<table>
<tr><td>Name:</td><td>MyDS</td></tr>
<tr><td>Description:</td><td>My custom datasource</td></tr>
</table>
</content>
</tab>
<tab>
<label>Configuration</label>
<content>
<table>
<tr><td>1</td><td>2</td></tr>
</table>
</content>
</tab>

```

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.TabularDataBaseDS` is used. For more information, see a sample implementation of a tabular custom datasource at [InMemoryDataSource](#).

A tabular datasource is typically associated with a SQL data services query. WSO2 products use an internal SQL

parser to execute SQL against the custom datasource. For more information, see a sample data service descriptor at [InMemoryDSSample](#). Carbon datasources also support tabular data with the [org.wso2.carbon.dataservices.core.custom.datasource.CustomTabularDataSourceReader](#) datasource reader implementation. If you have Data Services Server installed, for more information see the <PRODUCT\_HOME>\repository\conf\datasources\custom-datasources.xml file, which is a sample Carbon datasource configuration.

#### Custom query datasources

Custom query-based datasources are used for accessing non-tabular data through a query expression. To implement query-based datasources, the [org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryBasedDS](#) interface is used. You can create any non-tabular datasource using the query-based approach. Even if the target datasource does not have a query expression format, you can create and use your own. For example, you can support any NoSQL type datasource using this type of a datasource.

For more information, see a sample implementation of a custom query-based datasource at [EchoDataSource](#), and a sample data service descriptor with custom query datasources in [InMemoryDSSample](#). Carbon datasources also support query-based data with the [org.wso2.carbon.dataservices.core.custom.datasource.CustomQueryDataSourceReader](#) datasource reader implementation. If you have Data Services Server installed, for more information, see the <PRODUCT\_HOME>\repository\conf\datasources\custom-datasources.xml file, which is a sample Carbon datasource configuration.

In the `init` methods of all custom datasources, user-supplied properties will be parsed to initialize the datasource accordingly. Also, a property named `<__DATASOURCE_ID__>`, which contains a UUID to uniquely identify the current datasource, will be passed. This can be used by custom datasource authors to identify the datasources accordingly, such as datasource instances communicating within a server cluster for data synchronization.

Shown below is an example configuration of a custom datasource of type <DS\_CUSTOM\_TABULAR>:

```

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

```

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

## Managing Users and Roles

Before you begin, note the following:

- Only system administrators can add, modify and remove users and roles. To set up administrators, see [Realm 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. For details about writing a simple custom user store manager for WSO2 products, see [Writing a Custom User Store Manager](#).

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](#) and is responsible for creating users in the system, assigning them roles, managing databases, security, etc. The Admin role is also used to access the WSO2 Admin Portal ([https://<APIM\\_Host>:<APIM\\_Port>/admin](https://<APIM_Host>:<APIM_Port>/admin)), where you can define workflow tasks, throttling policies, analytics configurations, etc. The Admin role is available by default with the credentials admin/admin. By default, this role contains all the permissions (including super admin permissions) in the permission tree.
- **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. Governance permission gives to a creator to govern, manage and configure the API artifacts.
- **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.

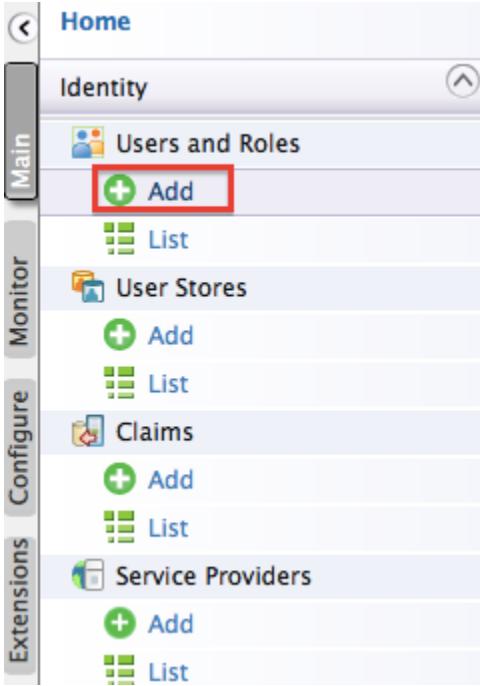
Follow the instructions below to create the `creator`, `publisher` and `subscriber` roles in the API Manager for example.

Creator, Publisher and Subscriber roles are available by default in API Manager.

## Create user roles

1. Log in to the management console ([https://<APIM\\_Host>:<APIM\\_Port>/admin](https://<APIM_Host>:<APIM_Port>/admin)) as admin (default credentials are admin/admin).

2. In the **Main** menu, click **Add** under **Users and Roles**.



3. Click **Add New Role**.

The screenshot shows the 'Add Users and Roles' page. It has three main buttons: 'Add New User', 'Add New Role' (which is highlighted with a red rectangular box), and 'Bulk Import Users'.

4. Enter the name of the user role (e.g., `creator`) and click **Next**.

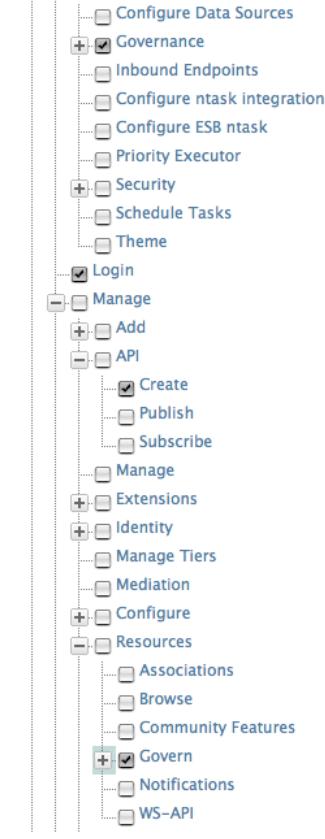
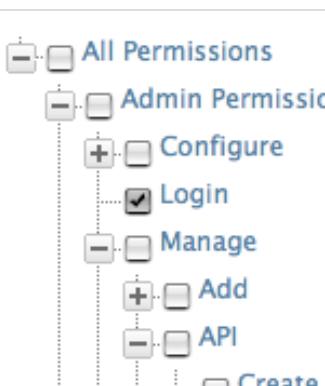
#### Step 1 : Enter role details

The screenshot shows the 'Enter role details' step. It includes fields for 'Domain' (set to 'PRIMARY') and 'Role Name\*' (containing the value 'creator'). At the bottom are buttons for 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a red rectangular box.

**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](#).

5. 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 which are available by default:

Roles	Permissions	UI
-------	-------------	----

creator	<ul style="list-style-type: none"> <li>Configure &gt; Governance and all underlying permissions.</li> <li>Login</li> <li>Manage &gt; API &gt; Create</li> <li>Manage &gt; Resources &gt; Govern and all underlying permissions</li> </ul>	<p><b>Step 2 : Select permissions to add to Role</b></p> <p><a href="#">Expand all</a> <a href="#">Collapse all</a></p>  <pre> graph TD     AP[All Permissions] --&gt; Admin[Admin Permissions]     Admin --&gt; Configure[Configure]     Admin --&gt; GP[Governance]     Admin --&gt; ID[Identity]     Admin --&gt; M[Manage]     Admin --&gt; C[Configure]     Admin --&gt; R[Resources]     Admin --&gt; WS[WS-API]     Configure --&gt; DS[Configure Data Sources]     Configure --&gt; GE[Governance]     Configure --&gt; IE[Inbound Endpoints]     Configure --&gt; CNT[Configure ntask integration]     Configure --&gt; CES[Configure ESB ntask]     Configure --&gt; PE[Priority Executor]     GP --&gt; GE     GP --&gt; IE     GP --&gt; CNT     GP --&gt; CES     GP --&gt; PE     ID --&gt; ST[Schedule Tasks]     ID --&gt; T[Theme]     M --&gt; A[Add]     M --&gt; API[API]     M --&gt; M[Manage]     M --&gt; E[Extensions]     M --&gt; I[Identity]     M --&gt; MT[Manage Tiers]     M --&gt; MEDI[Mediation]     C --&gt; R     R --&gt; AS[Associations]     R --&gt; B[Browse]     R --&gt; CF[Community Features]     R --&gt; G[Govern]     R --&gt; N[Notifications]     R --&gt; WS     G -- checked   </pre>
publisher	<ul style="list-style-type: none"> <li>Login</li> <li>Manage &gt; API &gt; Publish</li> </ul>	 <pre> graph TD     AP[All Permissions] --&gt; Admin[Admin Permissions]     Admin --&gt; Configure[Configure]     Admin --&gt; GP[Governance]     Admin --&gt; ID[Identity]     Admin --&gt; M[Manage]     Admin --&gt; C[Configure]     Admin --&gt; R[Resources]     Admin --&gt; WS[WS-API]     Configure --&gt; DS[Configure Data Sources]     Configure --&gt; GE[Governance]     Configure --&gt; IE[Inbound Endpoints]     Configure --&gt; CNT[Configure ntask integration]     Configure --&gt; CES[Configure ESB ntask]     Configure --&gt; PE[Priority Executor]     GP --&gt; GE     GP --&gt; IE     GP --&gt; CNT     GP --&gt; CES     GP --&gt; PE     ID --&gt; ST[Schedule Tasks]     ID --&gt; T[Theme]     M --&gt; A[Add]     M --&gt; API[API]     M --&gt; M[Manage]     M --&gt; E[Extensions]     M --&gt; I[Identity]     M --&gt; MT[Manage Tiers]     M --&gt; MEDI[Mediation]     C --&gt; R     R --&gt; AS[Associations]     R --&gt; B[Browse]     R --&gt; CF[Community Features]     R --&gt; G[Govern]     R --&gt; N[Notifications]     R --&gt; WS     G -- checked     API --&gt; C     C --&gt; Create[Create]     C --&gt; Publish[Publish]     C --&gt; Subscribe[Subscribe]     M --&gt; Manage[Manage]   </pre>

subscriber	<ul style="list-style-type: none"> <li>• Login</li> <li>• Manage &gt; API &gt; Subscribe</li> </ul>	<pre> graph TD     AP[All Permissions] --&gt; Admin[Admin Permissions]     Admin --&gt; Configure[Configure]     Admin --&gt; Login[Login]     Admin --&gt; Manage[Manage]     Manage --&gt; Add[Add]     Add --&gt; API[API]     API --&gt; Create[Create]     API --&gt; Publish[Publish]     API --&gt; Subscribe[Subscribe]     </pre>
------------	-----------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6. Click **Finish** once you are done adding permissions.

When a user creates an application and subscribes to an API, a role is created automatically as shown below.

"Application/<username>\_<applicationName>\_PRODUCTION"

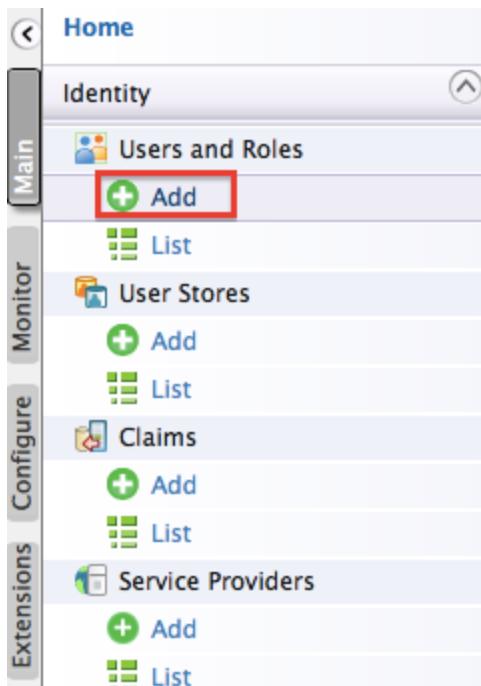
These roles are not assigned any permissions when created. The application is visible only to users of that particular role. For other users to be able to view the application, a user with admin privileges has to assign the role to the users.

## 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 via the Management console. Also, if you want to authenticate users via **e-mail**, **social media**, **multiple user store attributes**, see [Maintaining Logins and Passwords](#).

1. Log in to the Management Console and click **Add** under **Users and Roles** in the **Main** menu.



2. Click Add New User.



3. The Add User page opens. Provide the username and password and click Next.

### Add New User

**Step 1 : Enter user name**

Enter user name	
Domain	PRIMARY
User Name*	apicreator
Password*	*****
Password Repeat*	*****
<input type="button" value="Next &gt;"/> <input type="button" value="Finish"/> <input type="button" value="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](#).

4. 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  
 Application/admin\_DefaultApplication\_PRODUCTION

By default, all WSO2 products have the following roles configured:

- **Admin** - Provides full access to all features and controls. By default, the admin user is assigned to both the **Admin** and the **Everyone** roles.
- **Internal/Everyone** - Every new user is assigned to this role by default. It does not include any permissions.
- **Internal/System** - This role is not visible in the Management Console.

In addition to the above, the following roles exist by default.

- a. Internal/creator
- b. Internal/publisher
- c. Internal/subscriber

Note that there may be more roles configured by default depending on the type of features installed in your product.

5. Click **Finish** to complete.  
 The new user appears in the **Users** list. You can change the user's password, assign it different roles or delete it.

You cannot change the user name of an existing user.

The screenshot shows the 'Users' management interface. At the top, there's a search bar with fields for 'Select Domain' (set to 'ALL-USER-STORE-DOMAINS'), 'Enter user name pattern (\* for all)' (containing a wildcard), and 'Select Claim Uri'. Below the search bar is a table with three columns: 'Name', 'Actions', and another unnamed column. The table contains three rows of user data:

Name	Actions	
admin	Change Password  Assign Roles  View Roles  Delete  User Profile	
apicreator	Change Password  Assign Roles  View Roles  Delete  User Profile	
apipublisher	Change Password  Assign Roles  View Roles  Delete  User Profile	

## Accessing the Admin Dashboard

The Admin Dashboard is intended to be used by API Manager admins. The admin user has special permissions specified in the `/permission/admin/manage/apim_admin` directory. If a new user needs to access the admin dashboard, follow the steps below:

1. Create a user.
2. Create a new role. For more information, see [Adding User Roles](#).
3. Assign the following permissions to the new role you just created: `/permission/admin/manage/apim_admin` and `/permission/admin/configure/login`.
4. Assign the role created in step 2, to the user created in step 1.

Now this user is able to login and perform administrative tasks using the Admin Dashboard.

## 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.

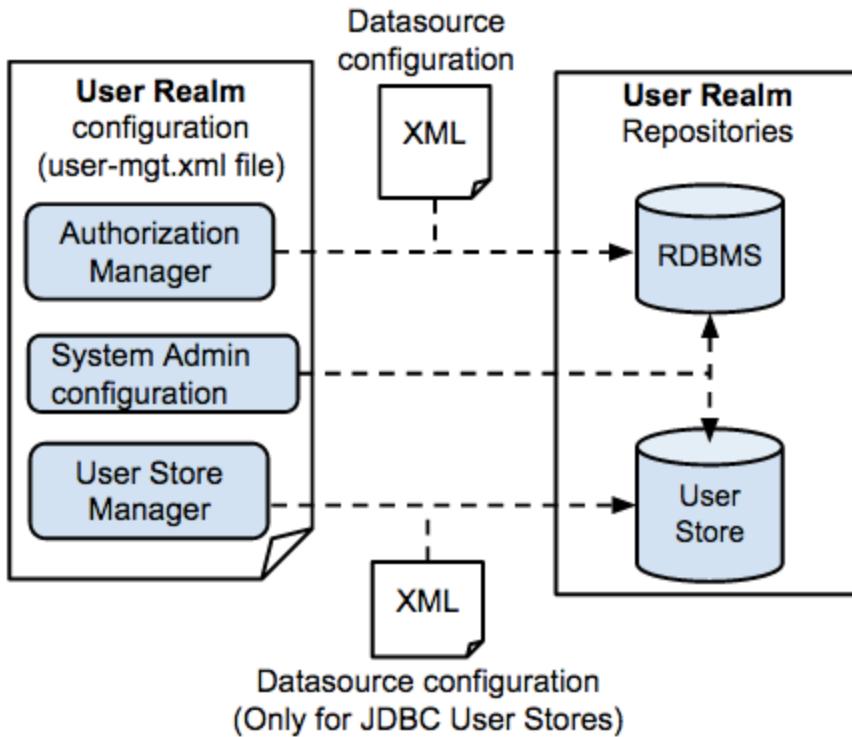
The sections below provide configuration details:

- [Realm Configuration](#)
- [Changing the RDBMS](#)
- [Configuring Primary User Stores](#)

## Realm Configuration

The complete functionality and contents of the User Management module is called a **user realm**. The realm includes the user management classes, configurations and repositories that store information. Therefore, configuring the User Management functionality in a WSO2 product involves setting up the relevant repositories and updating the relevant configuration files.

The following diagram illustrates the required configurations and repositories:



See the following topics for more details:

- Configuring the system administrator
- Configuring the authorization manager

### Configuring the system administrator

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

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

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

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

### **Updating the administrator**

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

```

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

 </Configuration>
 ...
</Realm>

```

Note the following regarding the configuration above.

Element	Description
<AddAdmin>	When <code>true</code> , this element creates the admin user based on the <code>AdminUser</code> element. It also indicates whether to create the specified admin user if it doesn't already exist. When connecting to an external read-only LDAP or Active Directory user store, this property needs to be <code>false</code> if an admin user and admin role exist within the user store. If the admin user and admin role do not exist in the user store, this value should be <code>true</code> , so that the role is added to the user management database. However, if the admin user is not there in the user store, we must add that user to the user store manually. If the <code>AddAdmin</code> value is set to <code>true</code> in this case, it will generate an exception.
<AdminRole>wso2admin</AdminRole>	This is the role that has all administrative privileges of the WSO2 product, so all users having this role are admins of the product. You can provide any meaningful name for this role. This role is created in the internal H2 database when the product starts. This role has permission to carry out any actions related to the Management Console. If the user store is read-only, this role is added to the system as a special internal role where users are from an external user store.
<AdminUser>	Configures the default administrator for the WSO2 product. If the user store is read-only, the admin user must exist in the user store or the system will not start. If the external user store is read-only, you must select a user already existing in the external user store and add it as the admin user that is defined in the <code>&lt;AdminUser&gt;</code> element. If the external user store is in read/write mode, and you set <code>&lt;AddAdmin&gt;</code> to <code>true</code> , the user you specify will be automatically created.
<UserName>	This is the username of the default administrator or super tenant of the user store. If the user store is read-only, the admin user MUST exist in the user store for the process to work.

<Password>	If the user store is read-only, this element and its value are ignored after the server starts for the first time. Therefore we can reset this password back to the original value/variable after server starts for the first time. This password is used only if the user store is read-write and the AddAdmin value is set to true.  Note that the password in the <code>user-mgt.xml</code> file is written to the primary user store when the server starts for the first time. Thereafter, the password will be validated from the primary user store and not from the <code>user-mgt.xml</code> file. Therefore, if you need to change the admin password stored in the user store, you cannot simply change the value in the <code>user-mgt.xml</code> file. To change the admin password, you must use the <b>Change Password</b> option from the management console.
<EveryOneRoleName>	The name of the "everyone" role. All users in the system belong to this role.

## Configuring the authorization manager

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

The repository that stores Permissions should always be an RDBMS. The Authorization Manager configuration in the `user-mgt.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/` directory) connects the system to this RDBMS.

Follow the steps given below to set up and configure the Authorization Manager.

### Step 1: Setting up the repository

By default, the embedded H2 database is used for storing permissions. You can change this as follows:

1. Change the default H2 database or set up another RDBMS for storing permissions.
2. When you set up an RDBMS for your system, it is necessary to create a corresponding datasource, which allows the system to connect to the database.
  - If you are replacing the default H2 database with a new RDBMS, update the `master-datasource.xml` file (stored in the `<PRODUCT_HOME>/repository/conf/datasources/` directory) with the relevant information.
  - Alternatively, create a new XML file with the datasource information of your new RDBMS and store it in the same `<PRODUCT_HOME>/repository/conf/datasources/` directory.

Refer the [related topics](#) for detailed information on setting up databases and configuring datasources.

### Step 2: Updating the user realm configurations

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

1. Set up the database connection by update the datasource information using the `<Property>` element under `<Configuration>`. The jndi name of the datasource should be used to refer to the datasource. In the following example, the jndi name of the default datasource defined in the `<PRODUCT_HOME>/repository/conf/datasources/master-datasources.xml` file is linked from the `user-mgt.xml` file.

```
<Realm>
 <Configuration>

 <Property name="dataSource">jdbc/WSO2CarbonDB</Property>
 </Configuration>
 ...
</Realm>
```

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

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

2. The default Authorization Manager section in the `user-mgt.xml` file is shown below. This can be updated accordingly.

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

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

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

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

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

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

## Related topics

1. See Maintaining Logins and Passwords for information on how to change the super admin credentials.

# 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
    - **For Windows:** wso2server.bat or wso2server.bat

## 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 ReadOnlyLDAPUserS external LDAP user stores.
org.wso2.carbon.user.core.ldap.ReadWriteLDAPUserStoreManager	Use ReadWriteLDAPUserS both read and write operati uncommented in the code in
org.wso2.carbon.user.core.ldap.ActiveDirectoryUserStoreManager	Use ActiveDirectoryUser SELECT UM_USER_NAME FUM_USER_ATTRIBUTE.UM_1UM_USER_ATTRIBUTE.UM_1UM_USER_ATTRIBUTE.UM_1UM_USER_ATTRIBUTE.UM_1UM_USER_ATTRIBUTE.UM_1UM_USER_ATTRIBUTE.UM_1UM_USER_ATTRIBUTE.UM_1UM_USER.UM_TENANT_ID=rectory Domain Service (A Service (AD LDS). This can to use AD as read-only you rReadOnlyLDAPUserStore
org.wso2.carbon.user.core.jdbc.JDBCUserStoreManager	Use JDBCUserStoreMana 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.CommonHybridLDA
 PTenantManager</Property>
 <Property name="ReadOnly">true</Property>
 <Property name="Disabled">false</Property>
 <Property name="MaxUserNameListLength">100</Property>
 </UserStoreManager>
 </Realm>
</UserManager>
```

```

<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=?))</Prope
rty>
 <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="ReadGroups">false</Property>
<Property name="GroupSearchBase">ou=system</Property>
<Property
 name="GroupSearchFilter">(objectClass=groupOfNames)</Property>
<Property name="GroupNameAttribute">cn</Property>
<Property name="MembershipAttribute">member</Property>

```

- 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 use store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the <Property name="Referral">follow</Property> property ensures that all the domains in the directory will be searched to locate the requested object.
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 Store](#)[Active 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.CommonHybridLDA
PTenantManager</Property>
 <Property
 name="ConnectionURL">ldap://localhost:${Ports.EmbeddedLDAP.LDAPServer
Port}</Property>
 <Property name="ConnectionName">uid=admin,ou=system</Property>
 <Property name="ConnectionPassword">admin</Property>
 <Property name="PasswordHashMethod">SHA</Property>
 <Property name="UserNameListFilter">(objectClass=person)</Property>
 <Property name="UserEntryObjectClass">wso2Person</Property>
 <Property name="UserSearchBase">ou=Users,dc=wso2,dc=org</Property>
 <Property
 name="UserNameSearchFilter">(&(objectClass=person)(uid=?))</Prope
rty>
 <Property name="UserNameAttribute">uid</Property>
 <Property name="PasswordJavaScriptRegEx">[\\"S]{5,30}</Property>
 <Property name="UsernameJavaScriptRegEx">[\\"S]{3,30}</Property>
 <Property
 name="UsernameJavaRegEx">^[^~!@#$;%^*+={}\\|\\\\\\<>,\\\'"]{3,30}$<
/Property>
 <Property name="RolenameJavaScriptRegEx">[\\"S]{3,30}</Property>
 <Property
 name="RolenameJavaRegEx">^[^~!@#$;%^*+={}\\|\\\\\\<>,\\\'"]{3,30}$<
/Property>
 <Property name="ReadGroups">true</Property>
 <Property name="WriteGroups">true</Property>
 <Property name="EmptyRolesAllowed">true</Property>
 <Property
 name="GroupSearchBase">ou=Groups,dc=wso2,dc=org</Property>
 <Property
 name="GroupNameListFilter">(objectClass=groupOfNames)</Property>
 <Property name="GroupEntryObjectClass">groupOfNames</Property>
 <Property
 name="GroupNameSearchFilter">(&(objectClass=groupOfNames)(cn=?))<
/Property>
 <Property name="GroupNameAttribute">cn</Property>
 <Property name="SharedGroupNameAttribute">cn</Property>
 <Property
 name="SharedGroupSearchBase">ou=SharedGroups,dc=wso2,dc=org</Property
>
 <Property name="SharedGroupEntryObjectClass">groups</Property>
 <Property
 name="SharedTenantNameListFilter">(object=organizationalUnit)</Proper
ty>
 <Property name="SharedTenantNameAttribute">ou</Property>
```

```
<Property
name="SharedTenantObjectClass">organizationalUnit</Property>
<Property name="MembershipAttribute">member</Property>
<Property name="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.CommonHybridLDA
PTenantManager</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">(&(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)</Proper
ty>
 <Property name="SharedTenantNameAttribute">ou</Property>
 <Property
name="SharedTenantObjectClass">organizationalUnit</Property>
 <Property name="MembershipAttribute">member</Property>
 <Property
name="GroupNameListFilter">(objectcategory=group)</Property>
 <Property
name="GroupNameSearchFilter">(& (objectClass=group) (cn=?))</Proper
ty>
 <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.JDBCAuthorizatio
nManager">
<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="ReadGroups">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 use store configurations in the `user-mgt.xml` file, we are only connecting to one of the domains. Therefore, when a request for an object is received to the user store, the `<Property name="Referral">follow</Property>` property ensures that all the domains in the directory will be searched to locate the requested object.
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.

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-mgt.xml`:

Ensure that you comment out the configurations for other user stores which you are not using when uncommenting `JDBCUserStoreManager`.

```
<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="ReadGroups">true</Property>
 <Property name="WriteGroups">true</Property>
 <Property name="UsernameJavaRegEx">^\S{3,30}\$</Property>
 <Property name="UsernameJavaScriptRegEx">^\S{3,30}\$</Property>
 <Property name="UsernameJavaRegExViolationErrorMsg">Username pattern policy violated</Property>
 <Property name="PasswordJavaRegEx">^\S{5,30}\$</Property>
 <Property name="PasswordJavaScriptRegEx">^\S{5,30}\$</Property>
 <Property name="PasswordJavaRegExViolationErrorMsg">Password length should be within 5 to 30 characters</Property>
 <Property name="RolenameJavaRegEx">^\S{3,30}\$</Property>
 <Property name="RolenameJavaScriptRegEx">^\S{3,30}\$</Property>
 <Property name="CaseInsensitiveUsername">true</Property>
 <Property name="SCIMEnabled">false</Property>
 <Property name="IsBulkImportSupported">true</Property>
 <Property name="PasswordDigest">SHA-256</Property>
 <Property name="StoreSaltedPassword">true</Property>
 <Property name="MultiAttributeSeparator">,</Property>
 <Property name="MaxUserNameListLength">100</Property>
 <Property name="MaxRoleNameListLength">100</Property>
 <Property name="UserRolesCacheEnabled">true</Property>
 <Property name="UserNameUniqueAcrossTenants">false</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="userNmae">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="SCIMEEnabled">false</Property>
<Property name="SelectUserSQL">SELECT * FROM UM_USER WHERE
UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
<Property name="GetRoleListSQL">SELECT UM_ROLE_NAME,
UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ?
AND UM_TENANT_ID=? AND UM_SHARED_ROLE = '0' ORDER BY
UM_ROLE_NAME</Property>
<Property name="GetSharedRoleListSQL">SELECT UM_ROLE_NAME,
UM_TENANT_ID, UM_SHARED_ROLE FROM UM_ROLE WHERE UM_ROLE_NAME LIKE ?
AND UM_SHARED_ROLE = '1' ORDER BY UM_ROLE_NAME</Property>
<Property name="UserFilterSQL">SELECT UM_USER_NAME FROM UM_USER
WHERE UM_USER_NAME LIKE ? AND UM_TENANT_ID=? ORDER BY
UM_USER_NAME</Property>
<Property name="UserRoleSQL">SELECT UM_ROLE_NAME FROM
UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_USER.UM_USER_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND
UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID AND UM_USER_ROLE.UM_TENANT_ID=? AND
UM_ROLE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
<Property name="UserSharedRoleSQL">SELECT UM_ROLE_NAME,
UM_ROLE.UM_TENANT_ID, UM_SHARED_ROLE FROM UM_SHARED_USER_ROLE INNER
JOIN UM_USER ON UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER
JOIN UM_ROLE ON UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE
UM_USER.UM_USER_NAME = ? AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID =
UM_USER.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID =
UM_ROLE.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID =
?</Property>
<Property name="IsRoleExistingSQL">SELECT UM_ID FROM UM_ROLE
WHERE UM_ROLE_NAME=? AND UM_TENANT_ID=?</Property>
<Property name=" GetUserListOfRoleSQL">SELECT UM_USER_NAME FROM
UM_USER_ROLE, UM_ROLE, UM_USER WHERE UM_ROLE.UM_ROLE_NAME=? AND
UM_USER.UM_ID=UM_USER_ROLE.UM_USER_ID AND
UM_ROLE.UM_ID=UM_USER_ROLE.UM_ROLE_ID AND UM_USER_ROLE.UM_TENANT_ID=? AND
UM_ROLE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
<Property name=" GetUserListOfSharedRoleSQL">SELECT UM_USER_NAME
FROM UM_SHARED_USER_ROLE INNER JOIN UM_USER ON
UM_SHARED_USER_ROLE.UM_USER_ID = UM_USER.UM_ID INNER JOIN UM_ROLE ON
UM_SHARED_USER_ROLE.UM_ROLE_ID = UM_ROLE.UM_ID WHERE
UM_ROLE.UM_ROLE_NAME= ? AND UM_SHARED_USER_ROLE.UM_USER_TENANT_ID =
UM_USER.UM_TENANT_ID AND UM_SHARED_USER_ROLE.UM_ROLE_TENANT_ID =

```

```

UM_ROLE.UM_TENANT_ID</Property>
 <Property name="IsUserExistingSQL">SELECT UM_ID FROM UM_USER
WHERE UM_USER_NAME=? AND UM_TENANT_ID=?</Property>
 <Property name=" GetUserPropertiesForProfileSQL ">SELECT
UM_ATTR_NAME, UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE
UM_USER.UM_ID = UM_USER_ATTRIBUTE.UM_USER_ID AND
UM_USER.UM_USER_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
 <Property name=" GetUserPropertyForProfileSQL ">SELECT
UM_ATTR_VALUE FROM UM_USER_ATTRIBUTE, UM_USER WHERE UM_USER.UM_ID =
UM_USER_ATTRIBUTE.UM_USER_ID AND UM_USER.UM_USER_NAME=? AND
UM_ATTR_NAME=? AND UM_PROFILE_ID=? AND
UM_USER_ATTRIBUTE.UM_TENANT_ID=? AND UM_USER.UM_TENANT_ID=?</Property>
 <Property name=" GetUserListForPropertySQL ">SELECT UM_USER_NAME
FROM UM_USER, UM_USER_ATTRIBUTE WHERE UM_USER_ATTRIBUTE.UM_USER_ID =
UM_USER.UM_ID AND UM_USER_ATTRIBUTE.UM_ATTR_NAME =? AND
UM_USER_ATTRIBUTE.UM_ATTR_VALUE =? 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="AddDomainsSQL">INSERT INTO UM_DOMAIN
(UM_DOMAIN_NAME, UM_TENANT_ID) VALUES (?, ?)</Property>

```

```

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

```
<Property name="DomainName">wso2.org</Property>
<Property name="Description"/>
</UserStoreManager>
```

The sample for the external JDBC user store consists of properties pertaining to various SQL statements. This is because the schema may be different for an external user store, and these adjustments need to be made to streamline the configurations with WSO2 products.

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. For more information on parameters need to be configured refer [Configuring a JDBC User Store](#).
5. In the realm configuration section, add the property MultiTenantRealmConfigBuilder and set the value to org.wso2.carbon.user.core.config.multitenancy.SimpleRealmConfigBuilder in order to construct tenant specific realm configurations.

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.

[Writing a Custom User Store Manager](#) - For details about writing a simple custom user store manager for WSO2 products.

### 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 the WSO2 API Publisher to the WSO2 API Store:

1. Set the <DisplayURL> to true, and provide the URL of the Store in the <API-M\_HOME>/repository/co

nf/api-manager.xml file.

If you are using a **distributed/clustered API Manager setup**, this configuration must be done in the API Publisher node/s.

### Example

```
<APIStore>
 <DisplayURL>true</DisplayURL>
 <URL>https://<hostname>:9443/store</URL>
</APIStore>
```

- <hostname> - The hostname of the API Publisher node.

2. Restart the WSO2 API - M server.  
Note that a URL that points to the API Store appears on the top, right-hand corner of the WSO2 API Publisher.

Example:

The screenshot shows the WSO2 API Publisher interface. The top navigation bar includes links for 'API Walkthrough', 'Go to API Store' (which is highlighted with a red border), and a user account icon. The main menu on the left has options for 'HOME', 'APIS' (which is selected and highlighted in blue), 'ANALYTICS', and 'MANAGE SUBSCRIPTIONS'. The central content area is titled 'All APIs' and contains a search bar. A single API entry is listed: 'PizzaShackAPI 1.0.0 admin 0 Users PUBLISHED'. To the right of the API entry is a small thumbnail image of a pizza.

- For information on clustering, see [Clustering WSO2 API Manager](#).
- For information on deployment patterns, see [Deployment Patterns of WSO2 API Manager](#).

## Maintaining Separate Production and Sandbox Gateways

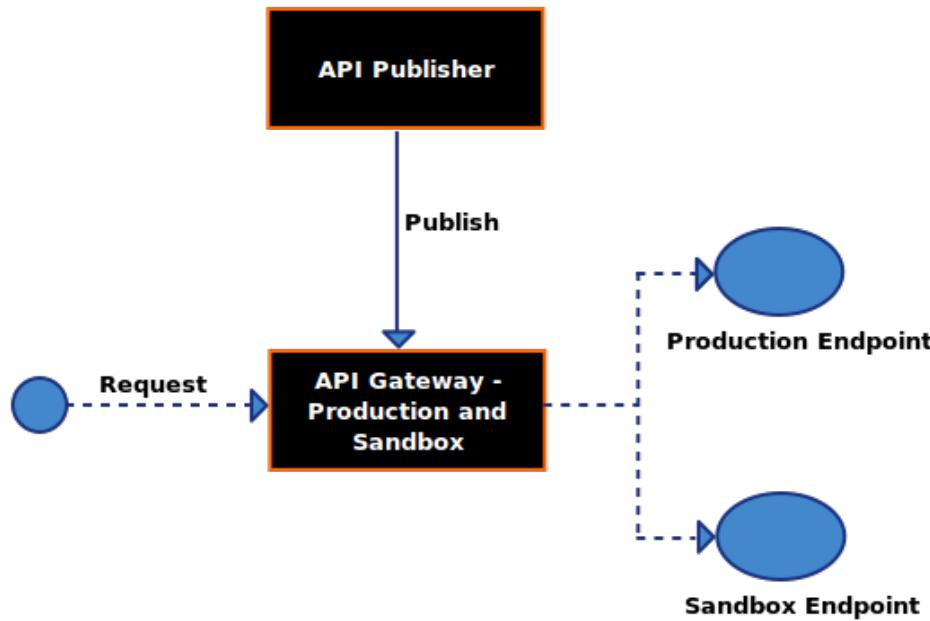
With WSO2 API Manager, you can maintain a production and a sandbox endpoint for a given API. The production endpoint is the actual location of the API, whereas the sandbox endpoint points to its testing/pre-production environment.

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

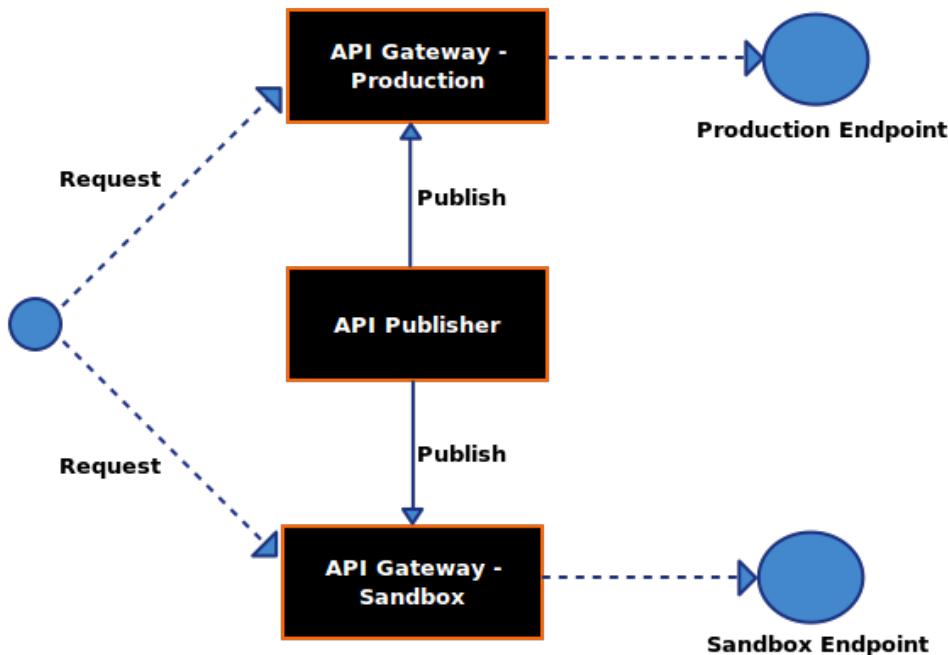
This is the default scenario. Because this Gateway instance handles both production and sandbox token traffic, it is called a hybrid API Gateway. When an API request comes to the API Gateway, it checks whether the requesting token is of type PRODUCTION or SANDBOX and forwards the request to the appropriate 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 `<API-M_HOME>/repository/conf/api-manager.xml` file in API Publisher nodes, as shown in the following example:

```

<Environments>
 <Environment type="production">
 <Name>Production</Name>
 <ServerURL>https://localhost:9445/services/</ServerURL>
 <Username>admin</Username>
 <Password>admin</Password>

 <GatewayEndpoint>http://localhost:8282,https://localhost:8245</GatewayEndpoint>
 </Environment>
 <Environment type="sandbox">
 <Name>Sandbox</Name>
 <ServerURL>https://localhost:9448/services/</ServerURL>
 <Username>admin</Username>
 <Password>admin</Password>

 <GatewayEndpoint>http://localhost:8285,https://localhost:8248</GatewayEndpoint>
 </Environment>
</Environments>

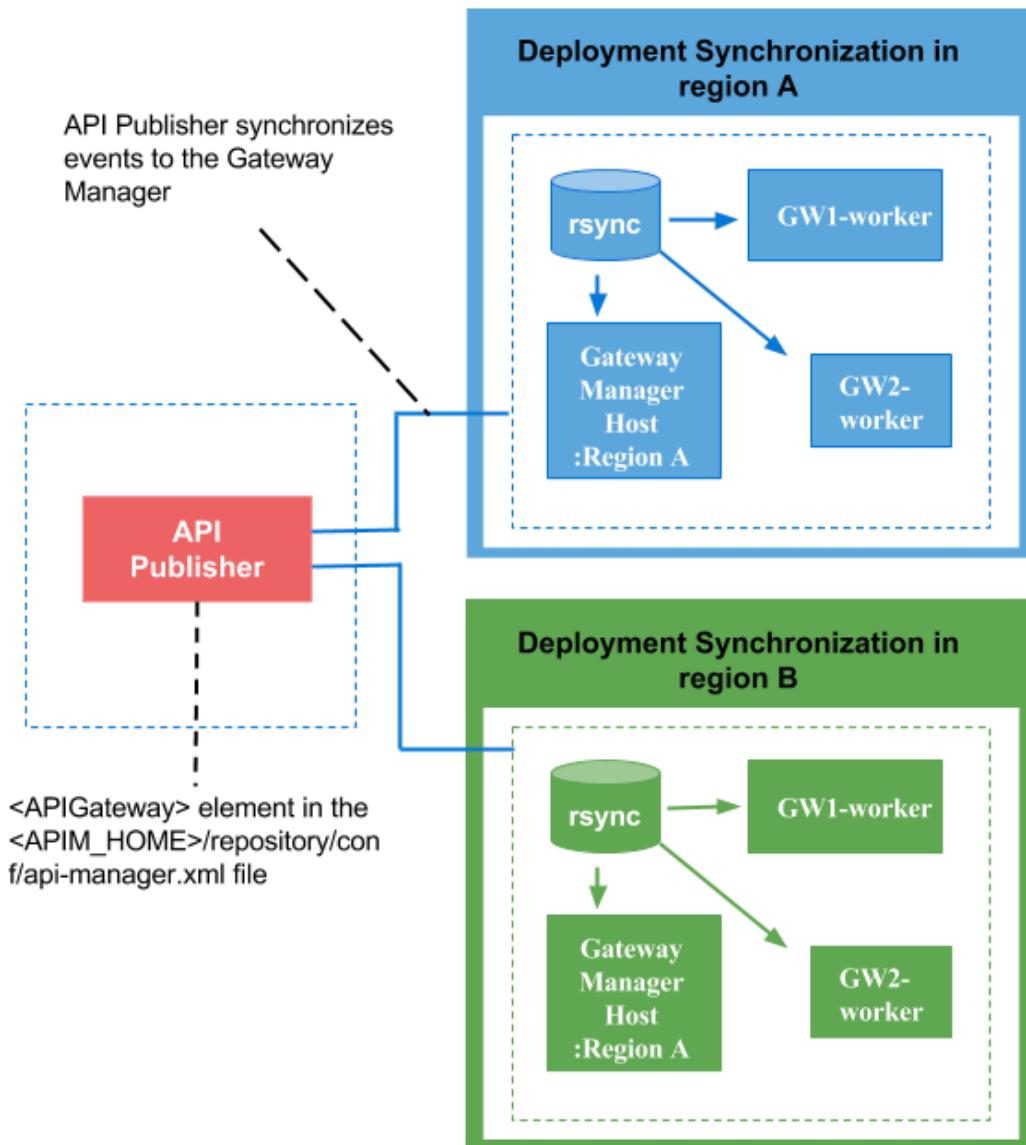
```

The `<ServerURL>` parameter should have the value of the environment instance. For information about the `<GatewayEndpoint>`, see [Working with Endpoints](#).

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 `<API-M_HOME>/repository/conf/api-manager.xml` file is recommended. The diagram below depicts a sample setup:



**Figure:** API Gateways in different geographical regions

Note that in addition to the configuration mentioned above, all the other required configuration for Publisher and other Components should be done. If you are using a multi-tenanted setup, would need to share the registry database mount with the Gateway Sandbox and Production nodes.

## 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
- HTTP Transport Properties

## 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. Therefore, we do not recommend changing the API's synapse configuration directly. The recommended way to extend an API's mediation flow is by [engineering 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"/>
```

To support having spaces inside JSON attributes, change the default JSON builder and formatter to the following pair in either `<APIM_HOME>/repository/conf/axis2/axis2.xml` (super tenant scenario) or `<APIM_HOME>/repository/conf/axis2/tenant-axis2.xml` (tenant scenario) file:

```
<messageBuilder contentType="application/json"
 class="org.apache.axis2.json.JSONStreamBuilder">
 <messageFormatter contentType="application/json"
 class="org.apache.axis2.json.JSONStreamFormatter"/>
```

To support use cases for JSON payloads with arrays, change the default JSON builder and formatter to the following pair in either `<APIM_HOME>/repository/conf/axis2/axis2.xml` (super tenant scenario) or `<APIM_HOME>/repository/conf/axis2/tenant-axis2.xml` (tenant scenario) file:

```
<messageBuilder contentType="application/json"
 class="org.apache.axis2.json.JSONStreamBuilder">
 <messageFormatter contentType="application/json"
 class="org.apache.axis2.json.JSONStreamFormatter"/>
```

Else, in JSON to XML conversion, there might be issues as below:

**JSON**

```
"phoneNumbers": [
 {
 "phoneNumber": "4027161289",
 "phoneNumberType": "home"
 }
]
```

**Converted XML**

```
<?xml-multiple phoneNumbers?>
<phoneNumbers>
 <phoneNumber>4027161289</phoneNumber>
 <phoneNumberType>home</phoneNumberType>
</phoneNumbers>
```

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:

```
<j(jsonObject>
 <object></object>
</j(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>
 </suspendOnFailure>
 <initialDuration>0</initialDuration>
 </http>
 </endpoint>
 </send>
 </then>
 </filter>
 </inSequence>
 </resource>
</api>
```

```
<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.APIAuthenticationH
andler"/>
 <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandl
er">
 <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.APIMgtGoogleAnalyticsTrackin
gHandler"/>
 <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHand
```

```

 ler"/>
 </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.APIAuthenticationH
andler"/>
 <handler
class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandl
er"/>
 </handlers>

```

```
er">
 <property name="id" value="A" />
 <property name="policyKey"
value="gov:/apimgt/applicationodata/tiers.xml"/>
 </handler>
 <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIUsageHandler"/>
 <handler
class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackin
gHandler"/>
 <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHand
```

```

 ler"/>
 </handlers>
</api>

```

# XML to JSON Transformation Parameters

See [JSON Transformation Parameters](#) for additional parameters for converting XML to JSON.

## Sharing Applications and Subscriptions

The API Manager provides the facility to users of a specific logical group such as an organization to view each others' applications and subscriptions.

By default, the API Manager considers the organization name that you give at the time you sign up to the API Store as the group ID. It extracts the claim `http://wso2.org/claims/organization` of a user and uses the value specified in it as the group ID. This way, all users who specify the same organization name belong to the same group and therefore, can view each others' subscriptions and applications. The API Manager also provides flexibility to change this default authentication implementation.

The steps below explain how to share applications and subscriptions.

1. Uncomment the `<GroupingExtractor>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file.

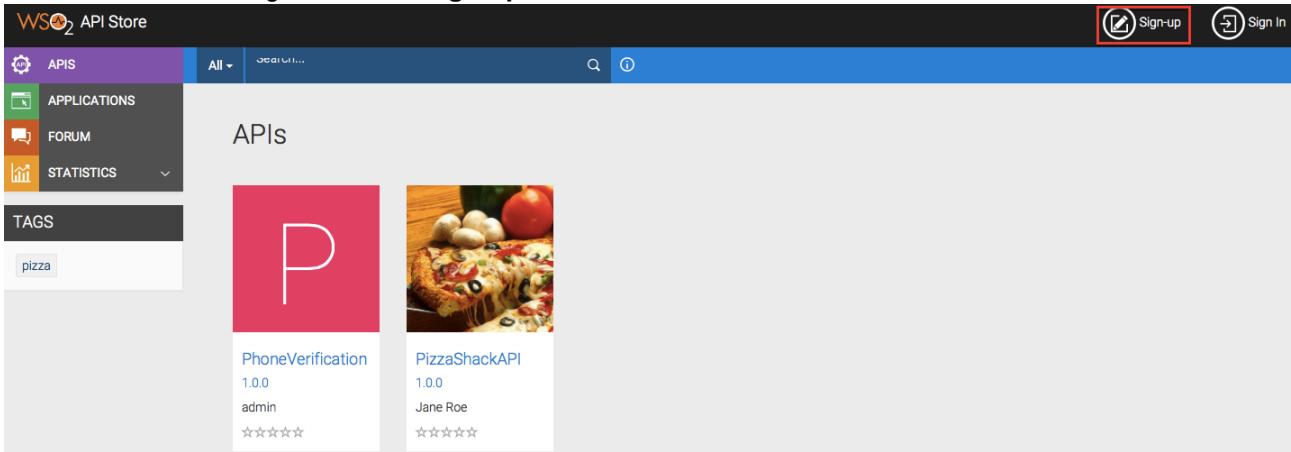
```

<GroupingExtractor>org.wso2.carbon.apimgt.impl.DefaultGroupIdExtractorImpl</GroupingExtractor>

```

This default extractor doesn't work with SAML SSO. You need to write a custom implementation using the [WSO2ISGroupIdExtractor.java](#) class as an example.

2. Start WSO2 API Manager and click **Sign-up**.



3. Sign up to the API store as two different users (User1 and User2) with the same organization name.  
Example:



Create your Account

Username \*

Characters left: 25

Password \*

Re-type Password \*

First Name \*

Last Name \*

Email \*

[Hide Additional Details](#)

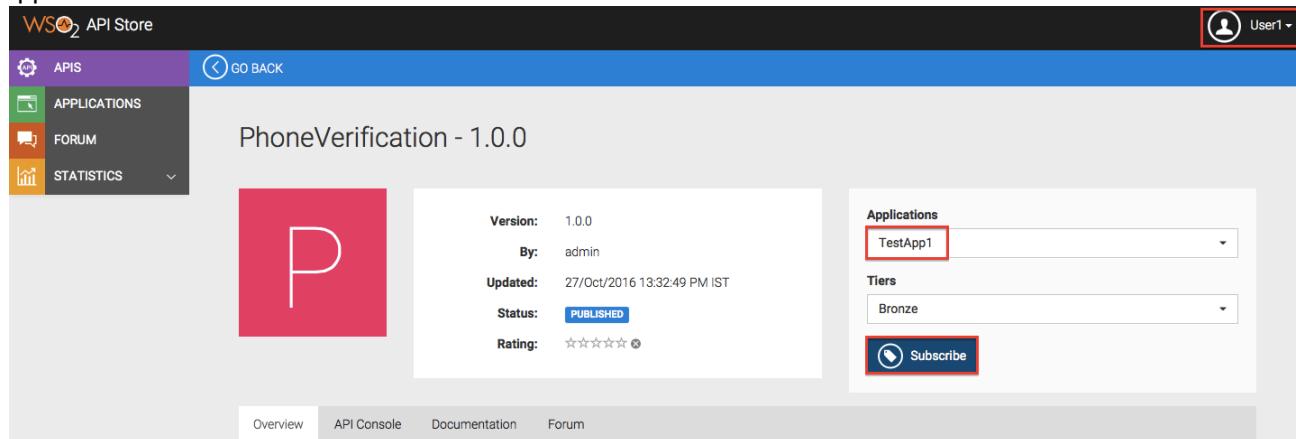
Organization

Address

Country

Land Phone

- Sign in as User1, create a new application (e.g., TestApp1) and subscribe to an API using the new application.



PhoneVerification - 1.0.0

Version: 1.0.0  
By: admin  
Updated: 27/Oct/2016 13:32:49 PM IST  
Status: PUBLISHED  
Rating: ★★★★☆

Applications:

Tiers:

[Subscribe](#)

- Sign out of the API Store and sign back in as User2.

Go to the **Applications** page and note that the previous user's subscription is listed under **Subscriptions** for the application that the previous user created (e.g., TestApp1).

Name	Tier	Workflow Status	Subscriptions	Actions
TestApp1(Shared)	Unlimited	ACTIVE	1	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
DefaultApplication	Unlimited	ACTIVE	0	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>

## Sharing Applications Between Multiple Groups

WSO2 API Manager provides the facility for users to share their applications and subscriptions with a specific logical group/groups such as an organization. Users can view and modify applications and subscriptions belonging to other users in the same group.

- Enabling Multi-Group Sharing
- Using the group sharing feature
- Extending the group ID extractor

### Enabling Multi-Group Sharing

You can enable application sharing between multiple groups by following the steps below.

1. Shutdown the server if its running
2. Uncomment the `<GroupingExtractor>` element in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

```
<GroupingExtractor>org.wso2.carbon.apimgt.impl.DefaultGroupIDExtractorImpl</GroupingExtractor>
```

This default extractor does not work with SAML SSO. To enable SAML SSO, you need to write a custom implementation using the [SAMLGroupIDExtractorImpl.java](#) class

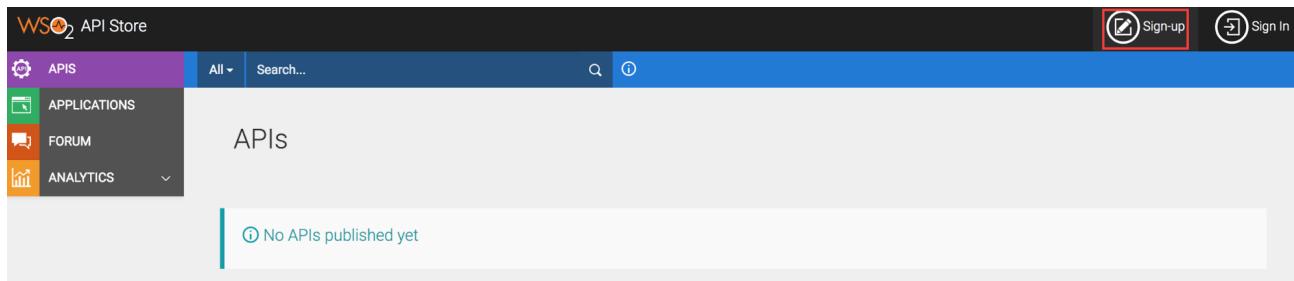
3. Restart the server.

### Using the group sharing feature

Group IDs are extracted using a `GroupingExtractor` class which is an implementation of `NewPostLoginExecutor` interface. The default implementation is done through the `DefaultGroupIDExtractorImpl` class. The organization claim is extracted using the group ID. If a particular user is in more than one organization, provide the organizations as a string separated by commas.

The steps below show how to use the group sharing feature

1. Start WSO2 API Manager and click **Sign-up**.



2. Sign up to the API store as two different users(e.g., usera, userb) belonging to the the same organizations.  
Click **Show Additional Details** to set the organization.



Create your Account

Username \*

Characters left: 25

Password \*

Re-type Password \*

First Name \*

Last Name \*

Email \*

[Hide Additional Details](#)

Organization

Country

Land Phone

Mobile Phone

IM

URL

3. Sign in as **usera** and add application App\_A.
4. Enter the Group ID as **org1** and press enter. Click **Add**. App\_A will be shared with all the users in **org1** group

## Add Application

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name\*  Characters left: 65

Groups  Add groups  
Type a Group and Enter

Per Token Quota  Allows unlimited requests  
This feature allows you to assign an API request quota per access token. Allocated quota will be shared among all the subscribed APIs of the application.

Description

5. Sign out of the API Store. Sign in as **userb**.
6. Go to the **Applications** tab. You will see App\_A which was added by **usera**.

### Applications

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name	Tier	Workflow Status	Subscriptions	Actions
usera/App_A	Unlimited	ACTIVE	0	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
userb/DefaultApplication	Unlimited	ACTIVE	0	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>

Show 10 entries | Showing 1 to 1 of 1 entries

Note that the name of the application creator is appended to the application name to differentiate the applications.

7. Subscribe to the default API using App\_A.

8. Log in to the API Store as **usera**. The subscriptions for App\_A by **userb** will be displayed.

## Extending the group ID extractor

The default implementation picks the organization claim as the group ID. The organization names are returned in a string array. To use a different claim or a different type of group ID , should create your own group ID extractor class by extending `NewPostLoginExecutor` interface and overriding the method below.

```
String[] getGroupingIdentifierList(String response);
```

This particular method will be called when a user is logging into the store and it will return all the groupIds for the logged in user. After logging in users will be able to see the applications created by themselves, and the applications shared with groupIds returned by `getGroupingIdentifierList` Method.

## Configuring API Monetization Category Labels

When defining throttling tiers using the Admin Portal, you have the option to specify a given billing plan for tiers. A tier is defined as either a free or paid tier. Depending on the tiers available for a given API, the following API monetization categories are displayed as labels in the store.

- **Free** - If all subscription tiers are defined as Free, the API uses the **Free billing plan** and the API is labeled as Free in the Store.
- **Paid** - If all subscription tiers are defined as Paid, the API uses the **Commercial billing plan** and the API is labeled as Paid in the Store.
- **Freemium** - If the API has a combination of Free and Paid subscription tiers, the API uses the **Freemium billing plan** and the API is labeled as Freemium in the Store.

Follow the configuration steps below to enable API monetization category labels:

1. Sign in to the API Manager's Management Console (<https://<hostname>:9443/carbon>).
2. Navigate to the **Main** menu, and click **Browse**, which is under the **Resources** tab.
3. Enter the following path in the **Location:** text-box and click **Go**.

/\_system/config/apimgt/applicationdata/tenant-conf.json

4. In the **Contents** panel, click the **Edit as text** link and the `tenant-conf.json` file opens.
5. To enable monetization categories for APIs, set the `EnableMonetization` property to true. By default, it is set to false.
6. Define the subscription tiers as required.

For example if you are working with the unlimited tier,

- To define the unlimited tier as **paid**, set the `IsUnlimitedTierPaid` property to true.
- To define the unlimited tier as **free**, set the `IsUnlimitedTierPaid` property to false.

As Freemium APIs has a combination of paid and free subscription tiers, the configuration involved in defining the subscription tiers will be the same as above. However, Freemium APIs need to have a minimum of one subscription defined as paid and free.

7. After the edits, click **Save Content**.

Note that the above configuration can be done independently on a per tenant basis.

When the above `EnableMonetization` property is set to true for the respective tenant, the API monetization category labels are displayed in the tenant API store.

The screenshot shows the WSO2 API Store interface. On the left, there's a sidebar with links for APIS, APPLICATIONS, FORUM, STATISTICS, and TAGS. A search bar at the top right has the placeholder 'Search...'. Below the sidebar, a search bar shows the query 'pizza'. The main area is titled 'APIs' and displays three API cards:

- Facebook**: Version 1.0.0, created by admin, rated 5 stars. It has a blue background with the Facebook logo and a green 'FREE' badge.
- PizzaShackAPI**: Version 1.0.0, created by Jane Roe, rated 5 stars. It has a background image of a pizza and a blue 'PAID' badge.
- WSO2Service**: Version 1.0.0, created by admin, rated 5 stars. It has an orange background with a black circle containing a white ECG line and a yellow 'FREE' badge.

### Related links

For more information on Monetization of APIs, see [Enabling Monetization of APIs](#).

## Enabling Notifications

You can enable notifications at the creation of new API versions in order to notify existing subscribers (via email) that a new version of the API is available. If you want to send a different type of notification, you can extend the `org.wso2.carbon.apimgt.impl.notification.Notifier` abstract class on your own accordingly.

Follow the instructions below to enable notifications for new API versions:

1. Set the email server configurations in the `<API-M_HOME>/repository/conf/output-event-adapters.xml` file under the `<adapterConfig type="email">` section.

```

<adapterConfig type="email">
 <!-- Comment mail.smtp.user and mail.smtp.password properties
 to support connecting SMTP servers which use trust
 based authentication rather username/password authentication
 -->
 <property key="mail.smtp.from">abcd@gmail.com</property>
 <property key="mail.smtp.user">abcd</property>
 <property key="mail.smtp.password">xxxx</property>
 <property key="mail.smtp.host">smtp.gmail.com</property>
 <property key="mail.smtp.port">587</property>
 <property key="mail.smtp.starttls.enable">true</property>
 <property key="mail.smtp.auth">true</property>
 <!-- Thread Pool Related Properties -->
 <property key="minThread">8</property>
 <property key="maxThread">100</property>
 <property key="keepAliveTimeInMillis">20000</property>
 <property key="jobQueueSize">10000</property>
</adapterConfig>

```

Property	Description
mail.smtp.from	The email address you use to send emails
mail.smtp.user	The email address used to authenticate the mail server. This can be same as mail.smtp.from
mail.smtp.password	Password used to authenticate the mail server.

2. Log in to the Management Console and click **Main > Resource > Browse**.
3. Browse to the `/_system/config/apimgt/applicationdata/tenant-conf.json` file and click **Edit as Text**.
4. Set the `NotificationsEnabled` property to true as shown below:

```

 "NotificationsEnabled": "true",
 "Notifications": [
 {
 "Type": "new_api_version",
 "Notifiers": [
 {
 "Class": "org.wso2.carbon.apimgt.impl.notification.NewAPIVersionEmailNotifier",
 "ClaimsRetrieverImplClass": "org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever",
 "Title": "Version $2 of $1 Released",
 "Template": " <html> <body> <h3 style=\"color:Black;\">We're happy to announce the arrival of the next major version $2 of $1 API which is now available in Our API Store.</h3>Click here to Visit WSO2 API Store</body></html>"
 }
]
 }
]
}

```

A notification type can have multiple notifier classes that help send multiple notifications. In this case, notification sends via EMail but it could be SMS notification. Each notifier has a class attribute containing the full class path. The following properties should be set for the default NewAPIVersionEmailNotifier class :

Property	Description
Class	The full class path of the notifier class.
ClaimsRetrieverImplClass	Subscriber email addresses are extracted from user claims. The default pack uses the org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever class to read the claim values from the user store.
<h2>Claims</h2> <p>A claim is a piece of information about a particular subject. It can be anything that the subject is owned by or associated with, such as name, group, preferences, etc. For information on how to add new claim mappings, see <a href="#">Configuring Claims</a>.</p>	
Title	The subject of the email.
Template	The template of the email body. This can be string values or a valid registry path to a template file.

The following strings are replaced with API specific values in the Title and Template properties.

\$1 - API name

## \$2 - New API version

If you create the subscriber in the Management Console, you need to add the subscriber's email in the subscriber user profile. You can find the user profile when you list the users in the management console.

If you are using a Google mail account, note that Google has restricted third-party apps and less secure apps from sending emails by default. Therefore, you need to configure your account to disable this restriction. For more information about Setting Gmail, see [Creating Users using the Ask Password Option](#).

## Working with Access Tokens

When an Application Developer registers an Application on the API Store, the Application is given a consumer-key and a consumer-secret, which represents the credentials of the Application that is being registered. The consumer-key becomes the unique identifier of the Application, similar to a user's username, and is used to authenticate users. When an Access Token is issued for the Application, it is issued against the latter mentioned consumer-key.

API consumers generate access tokens and pass them in the incoming API requests. The API key (i.e., the generated access token) is a simple string that you need to pass as an HTTP header. For example, "Authorization: Bearer NtBQkXoKELu0H1a1fQ0DWf06IX4a." This works equally well for SOAP and REST calls.

Authorizing requests, which come to published APIs, using access tokens helps you **prevent certain types of denial-of-service (DoS) attacks**. If the token that is passed with a request is invalid, WSO2 API Manager (WSO2 API-M) discards that request in the first stage of processing itself.

WSO2 API Manager provides two types of access tokens for authentication:

- **Application Access Tokens:** Tokens to identify and authenticate an entire application. An application is a logical collection of many APIs. With a single application access token, you can invoke all of these APIs.
- **User Access Tokens:** Tokens to identify the final user of an application. For example, the final user of a mobile application deployed on different devices.

In WSO2 API-M the access token must be unique for the following combinations - CONSUMER\_KEY, AUTHZ\_USER, USER\_TYPE, TOKEN\_STATE, TOKEN\_STATE\_ID and TOKEN\_SCOPE. The latter mentioned constraint is defined in the [IDN\\_OAUTH2\\_ACCESS\\_TOKEN](#) table. Therefore, it is not possible to have more than one Access Token for any of the above combinations.

Let's take a look at how to generate and renew each type of access token.

- Generating application access tokens
- Generating user access tokens
- Generating access tokens per device
- Renewing application access tokens
- Renewing user access tokens
- Changing the default token expiration time

### ***Generating application access tokens***

Application access tokens are tokens that 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 service level agreement (SLA) levels/tiers. Application access tokens leverage OAuth2 to provide simple key management.

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

1. Sign in to WSO2 API Store.
2. Click the **Applications** menu and open the application for which you want to generate an access token.
3. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. You can use this token to invoke all APIs that you subscribe to using the same application.

**DefaultApplication**

Details    **Production Keys**    Sandbox Keys    Subscriptions

**i No Keys Found**  
No keys are generated for this type in this application.

**Grant Types**  
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM	<input type="checkbox"/> Implicit	<input checked="" type="checkbox"/> Refresh Token
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code	<input checked="" type="checkbox"/> Password	

**Callback URL**

**Access token validity period**  
 Seconds.

**Generate keys**

In the **Access token validity period** field, you can set an expiration period to determine the validity period of the token after generation. Set this to a negative value to ensure that the token never expires. Also see [Changing the default token expiration time](#).

**Tip:** When you generate access tokens to APIs protected by **scope/s**, you can select the scope/s and then generate the token for it.

#### Generating user access tokens

User access tokens are tokens that authenticate the final user of an API, and are valid for all APIs subscribed to a user via a particular application. User access tokens allow you to invoke an API even from a third-party application such as 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](#).

By default, access tokens and consumer secrets are not saved in an encrypted format in the database. An admin can **enable encryption** following the instructions in [Encrypting OAuth Keys](#).

**Tip:** If you want to maintain authorization headers in messages, which are going out from the API Gateway, an admin can go to the `<API_Gateway_node>/repository/conf/api-manager.xml` file, uncomment the `<RemoveOAuthHeadersFromOutMessage>` element, set its value to `false`, and then restart the server to apply the changes.

```
<RemoveOAuthHeadersFromOutMessage>false</RemoveOAuthHeadersFromOutMessage>
```

Note that when a user is deleted, the access token is automatically invalidated.

#### **Generating access tokens per device**

WSO2 API Manager returns the same token repeatedly if a valid token exists for the requesting Application, on behalf of the user. However, the latter mentioned scenario becomes an issue if the same user is using the same Application in two devices (e.g., If you have two instances of the same Application running on your iPhone and iPad, and your iPhone already has a token on behalf of you, your iPad will get the same token if you requested for it within the same validity period. Therefore, if one of your devices revoke this token (e.g., revoke on logout), the token that you obtained for your other device becomes invalid as the devices use the identical tokens).

To overcome this problem, WSO2 API Manager provides a mechanism, with the use of [OAuth2.0 Scopes](#), for obtaining a unique Access Token for each device that uses the same Application. Thereby, allowing users to request tokens for different scopes. You need to prefix the `scope` names with the string "device\_". WSO2 API Manager uses special treatment for the scopes that are prefixed with the latter mentioned string by ignoring the usual validations it does when issuing tokens that are associated to scopes. The following is a sample cURL command that you can use to request a token with a "device\_" scope.

```
curl -k -d
"grant_type=password&username=<username>&password=<password>&scope=device_ipad" -H "Authorization :Basic base64encode(consumer-key:consumer-secret),
Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/token
```

Each token request that is made with a different scope, results in a different access token being issued. For example if you received a token named `abc` as a result of the scope `device_ipad`, you will not receive `abc` when you request for the token with the scope `device_iphone`. Note that you can use `device_` scopes in conjunction with other scopes as usual.

#### **Renewing application access tokens**

When an application access token expires, consumers can refresh the token by signing into the API Store, opening the application, and clicking **Re-generate** that appears in the **Production Keys** tab. You can also specify a token expiration time for the application access token. Set this to a negative value to ensure that the token never expires.

## DefaultApplication

Production Keys

**Show Keys**

**Consumer Key**  
.....

**Consumer Secret**  
.....

**Grant Types**  
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can select one or more grant types.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code

**Callback URL**  
.....

**Update**

**Generating Access Tokens**  
The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

**Generate a Test Access Token**

**Access Token**  
.....

Above token has a validity period of **3600** seconds. And the token has (**am\_application\_scope default**) scopes.

**Scopes**  
No Scopes Found..

**Validity period**  
3600 Seconds.

**Re-generate**

### Renewing user access tokens

To renew a user token, issue a REST call to the WSO2 Login API through a REST client. For more information, see [Token API](#).

### Changing the default token expiration time

Access tokens have an expiration time, which is set to 60 minutes by default.

- To change the default expiration time of application access tokens,
  - Change the value of the <AccessTokenDefaultValidityPeriod> element in the <API-M\_HOME>/repository/conf/identity/identity.xml file. Set this to a negative value to ensure that the token never expires. **Changes to this value are applied only to the new applications that you create.**

#### Example

```
<AccessTokenDefaultValidityPeriod>-3600</AccessTokenDefaultValidityPeriod>
```

- Alternatively, you can set a default expiration time through the UI when generating/regenerating the application access token.  
This is explained in [previous sections](#).
- Similarly, to change the default expiration time of user access tokens, edit the value of the <UserAccessTokenDefaultValidityPeriod> element in the <API-M\_HOME>/repository/conf/identity/identity.xml file.

#### Example

```
<UserAccessTokenDefaultValidityPeriod>3800</UserAccessTokenDefaultValidityPeriod>
```

Also see [Configuring Caching](#) for several caching options available to optimize key validation.

## Performance Tuning and Testing Results

The topics in this section provide performance tuning recommendations for the WSO2 API Manager (WSO2 API-M) and the results of performance tests that were carried out on WSO2 API-M.

- [Tuning Performance](#)
- [WSO2 API-M Performance and Capacity Planning](#)

### 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
- WSO2 Carbon platform-level settings
- APIM-level settings
- Throttle data and Analytics-related 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

When it comes to performance, the OS that the server runs plays an important role.

If you are running MacOS Sierra and experiencing long startup times for WSO2 products, try mapping your Mac hostname to 127.0.0.1 and ::1 in the /etc/hosts file as described. For example, if your Macbook hostname is "john-mbpro.local", then add the mapping to the canonical 127.0.0.1 address in the /etc/hosts file, as shown in the example below.

```
127.0.0.1 localhost john-mbpro.local
```

Following are the configurations you can apply to optimize OS-level performance:

1. To optimize network and OS performance, configure the following settings in the /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 the /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 the /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**

When an XML element has a large number of sub elements and the system tries to process all the sub elements, the system can become unstable due to a memory overhead. This is a security risk.

To avoid this issue, you can define a maximum level of entity substitutions that the XML parser allows in the system. You do this using the `entityExpansionLimit` as follows in the `<API-M_HOME>/bin/wso2server.bat` file (for Windows) or the `<API-M_HOME>/bin/wso2server.sh` file (for Linux/Solaris). The default entity expansion limit is 64000.

```
-DentityExpansionLimit=10000
```

In a clustered environment, the entity expansion limit has no dependency on the number of worker nodes.

### **WSO2 Carbon platform-level settings**

In multitenant mode, the WSO2 Carbon runtime limits the thread execution time. That is, if a thread is stuck or taking a long time to process, Carbon detects such threads, interrupts and stops them. Note that Carbon prints the current stack trace before interrupting the thread. This mechanism is implemented as an Apache Tomcat valve. Therefore, it should be configured in the `<PRODUCT_HOME>/repository/conf/tomcat/catalina-server.xml` file as shown below.

```
<Valve
 className="org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionVal
 ve" threshold="600" />
```

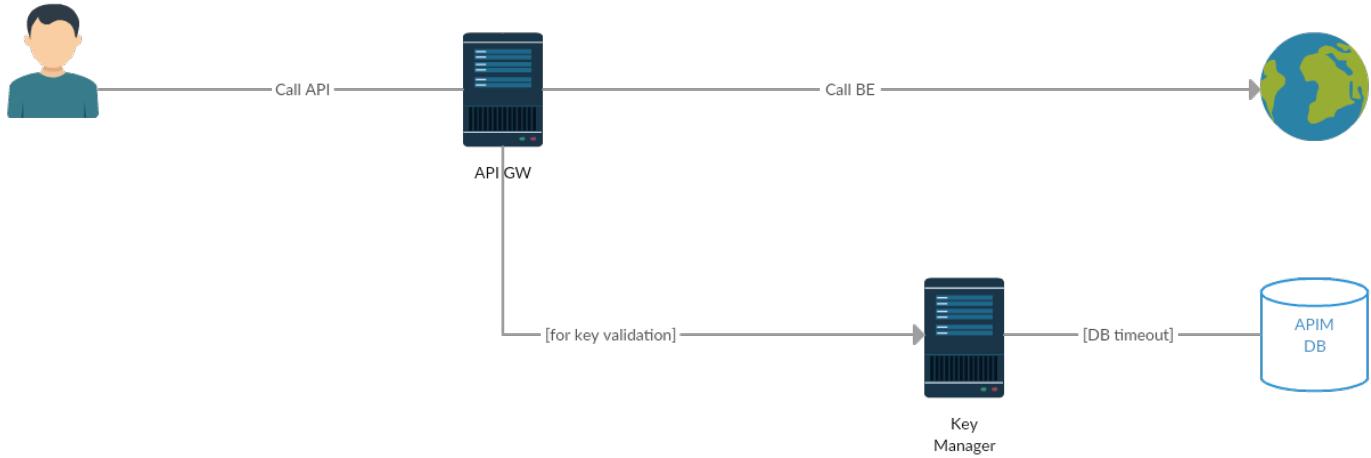
- The `className` is the Java class used for the implementation. Set it to `org.wso2.carbon.tomcat.ext.valves.CarbonStuckThreadDetectionValve`.
- The `threshold` gives the minimum duration in seconds after which a thread is considered stuck. The default value is 600 seconds.

### **APIM-level settings**

- Timeout configurations for an API call
- General APIM-level recommendations
- Registry indexing configurations
- Tuning data-agent parameters

Timeout configurations for an API call

The following diagram shows the communication/network paths that occur when an API is called. The timeout configurations for each network call are explained below.



- Key validation**

Key validation occurs via a Servlet HTTP call and the connection timeout can be configured by changing the following configuration details in the <API-M\_HOME>/repository/conf/axis2/axis2\_client.xml file. All timeout values are in milliseconds.

```

<transportSender name="https"
class="org.apache.axis2.transport.http.CommonsHTTPTransportSender">
<parameter name="SO_TIMEOUT">60000</parameter>
<parameter name="CONNECTION_TIMEOUT">60000</parameter>
</transportSender>

```

If the Key Manager caching is enabled, the calls between the API Gateway and Key Manager are cached. As a result, the Key Manager is not invoked for each API call.

- Client call API Gateway + API Gateway call Backend**

For backend communication, the API Manager uses PassThrough transport. This is configured in the <API-M>.xml file. For more information, see [Configuring passthru-http.properties](#) in the ESB documentation.

Note that the default value for http.socket.timeout differs between WSO2 products. In WSO2 API-M, the default value for http.socket.timeout is 60000ms.

#### General APIM-level recommendations

Some general APIM-level recommendations are listed below:

Improvement Area	Performance Recommendations

API Gateway nodes	<p>Increase memory allocated by modifying the <code>/bin/wso2server.sh</code> file with the following setting:</p> <ul style="list-style-type: none"><li>• <code>-Xms2048m -Xmx2048m -XX:MaxPermSize=1024m</code></li></ul> <p>Set the following in the <code>&lt;API-M_HOME&gt;/repository/conf/axis2/axis2_client.xml</code> file:</p> <div style="border: 1px solid #ccc; padding: 10px;"><ul style="list-style-type: none"><li>• The following Axis2 client configurations are only applicable when Web Services key validation (WS key validation) is enabled.</li><li>• The default values mentioned in the API-M 2.1.0 pack are the values identified at the time of releasing API-M 2.1.0. However, if you want high concurrency, use the values mentioned below:</li></ul></div> <div style="border: 1px dashed #ccc; padding: 10px; margin-top: 10px;"><pre>&lt;parameter name="defaultMaxConnPerHost"&gt;1000&lt;/parameter&gt; &lt;parameter name="maxTotalConnections"&gt;30000&lt;/parameter&gt;</pre></div> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p>The above configurations are only applicable when WS key validation is enabled.</p></div>
-------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

NHTTP transport of API Gateway	<p>Recommended values for the &lt;API-M_HOME&gt;/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><b>Property descriptions:</b></p> <table border="1" data-bbox="339 312 1481 741"> <tbody> <tr> <td data-bbox="339 312 551 365">snd_t_core</td><td data-bbox="551 312 1481 365">Transport sender worker pool's initial thread count</td></tr> <tr> <td data-bbox="339 365 551 424">snd_t_max</td><td data-bbox="551 365 1481 424">Transport sender worker pool's maximum thread count</td></tr> <tr> <td data-bbox="339 424 551 614">snd_io_threads</td><td data-bbox="551 424 1481 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="339 614 551 673">snd_alive_sec</td><td data-bbox="551 614 1481 673">Sender-side keep-alive seconds</td></tr> <tr> <td data-bbox="339 673 551 741">snd_qlen</td><td data-bbox="551 673 1481 741">Sender queue length, which is infinite by default</td></tr> </tbody> </table> <p><b>Recommended values:</b></p> <ul style="list-style-type: none"> <li>• HTTP Sender thread pool parameters <ul style="list-style-type: none"> <li>• snd_t_core=200</li> <li>• snd_t_max=250</li> <li>• snd_alive_sec=5</li> <li>• snd_qlen=-1</li> <li>• snd_io_threads=16</li> </ul> </li> <li>• HTTP Listener thread pool parameters <ul style="list-style-type: none"> <li>• lst_t_core=200</li> <li>• lst_t_max=250</li> <li>• lst_alive_sec=5</li> <li>• lst_qlen=-1</li> <li>• lst_io_threads=16</li> </ul> </li> <li>• timeout parameters <ul style="list-style-type: none"> <li>• http.socket.timeout.receiver: Recommended socket timeout for listener is 120000 ms.</li> <li>• http.socket.timeout.sender: Recommended socket timeout for sender is 120000 ms.</li> </ul> </li> </ul>	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 the &lt;API-M_HOME&gt;/repository/conf/passthru-http.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><b>Property descriptions</b></p> <table border="1" data-bbox="326 316 1485 844"> <tbody> <tr> <td data-bbox="326 316 913 401">worker_thread_keepalive_sec</td><td data-bbox="913 316 1485 401">Defines the keep-alive time for extra threads in the worker pool</td></tr> <tr> <td data-bbox="326 401 913 528">worker_pool_queue_length</td><td data-bbox="913 401 1485 528">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="326 528 913 612">io_threads_per_reactor</td><td data-bbox="913 528 1485 612">Defines the number of IO dispatcher threads used per reactor</td></tr> <tr> <td data-bbox="326 612 913 696">http.max.connection.per.host.port</td><td data-bbox="913 612 1485 696">Defines the maximum number of connections per host port</td></tr> <tr> <td data-bbox="326 696 913 844">worker_pool_queue_length</td><td data-bbox="913 696 1485 844">Determines the length of the queue used by the PassThrough transport thread pool to store pending jobs.</td></tr> </tbody> </table> <p><b>Recommended values</b></p> <ul style="list-style-type: none"> <li>• worker_thread_keepalive_sec: Default value is 60s. This should be less than the socket timeout.</li> <li>• worker_pool_queue_length: 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.</li> <li>• io_threads_per_reactor: Value is based on the number of processor cores in the system. (Runtime.getRuntime().availableProcessors())</li> <li>• http.max.connection.per.host.port : 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).</li> <li>• worker_pool_size_core: 400</li> <li>• worker_pool_size_max: 500</li> <li>• io_buffer_size: 16384</li> <li>• http.socket.timeout: 60000</li> <li>• snd_t_core: 200</li> <li>• snd_t_max: 250</li> <li>• snd_io_threads: 16</li> <li>• lsr_t_core: 200</li> <li>• lsr_t_max: 250</li> <li>• lsr_io_threads: 16</li> </ul> <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.										
Timeout 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 a few parameters, in <b>addition to the timeout parameters discussed in the previous sections</b>.</p>										

synapse.global_timeout_interval	<p>Defines the maximum time that a callback waits 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 the Gateway.</p> <p>The global timeout is defined in the &lt;API-M_HOME&gt;/repository/conf/synapse.properties file. The recommended value is 120000 ms.</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 50 seconds (50000 ms) and executes the fault handler in case of a timeout.</p> <pre data-bbox="1008 804 1383 1085">&lt;timeout&gt; &lt;duration&gt;50000&lt;/duration&gt; &lt;responseAction&gt;fault&lt;/responseAction&gt; &lt;/timeout&gt;</pre> <p>Alternatively, you can set this through the Publisher UI as well, by following the steps below:</p>

1. Log in to the API Publisher (<https://<HostName>:9443/publisher>). Select your API and click **Edit API**.
2. Click the **Implement** tab and click the cogwheel icon next to the endpoint you want to re-configure.
3. In the **Advanced Settings** dialog box that appears, increase the duration by modifying the default property set as 3000 ms.

Note that when the endpoint is suspended, the default action is defined here as invoking the fault sequence.

Advanced Endpoint Configuration

Endpoint Suspend State

Error Codes: Nothing selected

A list of error codes. If these error codes are received from the endpoint, the endpoint will be suspended.

Initial Duration (ms):  Max Duration (ms):   
Factor:

Endpoint Timeout State

Error Codes: Nothing selected

Retries Before Suspension:  Retry Delay(ms):

Connection Timeout

Action: Execute fault sequence  
 Duration (ms)

**Save** **Close**

4. Click **Save** and re-publish the API.

The `http.socket.timeout` parameter needs to be adjusted based on the endpoint-level timeout so that its value is equal or higher than the highest endpoint-level timeout.

If your API is marked as the default version, it has a different template (without the version number) that comes with a pre-defined timeout for the endpoint. This timeout does not change with the changes you do to the API by editing the Advanced Endpoint Configuration. Therefore, if this predefined timeout (60 seconds) is greater than the actual API timeout, it triggers the timeout before the actual configured API timeout.

To overcome this, update the `default_api_template.xml` residing in the `<API-M_HOME>/repository/resources/api_templates` directory by removing the endpoint timeout configuration from the default API. Then, the APIs marked as the default version also trigger the timeout when the actual API timeout is met.

Follow the steps below to update the `default_api_template.xml` to remove the endpoint configuration for the default APIs.

If you are using a distributed (clustered) setup, follow these steps in the Publisher node as it is the API Publisher that creates the API definition and pushes it to the Gateway.

1. Open the `<API-M_HOME>/repository/resources/api_templates/default_api_template.xml` file and remove the following configuration:

```

<timeout>
 <duration>600
 00</duration>
 <responseActi
 on>fault</resp
 onseAction>
</timeout>
<suspendOnFail
ure>
 <progressionF
 actor>1.0</pro
 gressionFactor
 >
</suspendOnFai
lure>
<markForSuspen
sion>
 <retriesBefor
 eSuspension>0<
 /retriesBefore
 Suspension>
 <retryDelay>0
 </retryDelay>
</markForSuspe
nsion>
```

2. Add the following configuration to the same place in the `default_api_template.xml` file.

By adding this configuration, you ensure that the APIs marked as the default version never timeout or are suspended using the endpoint configuration defined in the synapse file of the API.

```
<suspendOnFailure>
 <errorCodes>-1</errorCodes>
 <initialDuration>0</initialDuration>
 <progressionFactor>1.0</progressionFactor>
 <maximumDuration>0</maximumDuration>
</suspendOnFailure>
<markForSuspension>
 <errorCodes>-1</errorCodes>
</markForSuspension>
```

3. Go to the API Publisher and republish the default API by clicking **Save and Publish**.

## Key Manager nodes

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 the <API-M\_HOME>/repository/conf/tomcat/catalina-server.xml file.

If you use WSO2 Identity Server (WSO2 IS) as the Key Manager, then the root location of the above path and the subsequent path needs to change from <API-M\_HOME> to <IS\_HOME>.

```
maxThreads="750"
minSpareThreads="150"
disableUploadTimeout="false"
enableLookups="false"
connectionUploadTimeout="120000"
maxKeepAliveRequests="600"
acceptCount="600"
```

Set the following connection pool elements in the <API-M\_HOME>/repository/conf/datasources/master-datasources.xml file. Time values are defined in milliseconds.

```
<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 30000 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.

## Registry indexing configurations

The registry indexing process is only required to be run on the API Publisher and API Store nodes. To disable the indexing process from running on the other nodes (Gateways and Key Managers), you need to set the <wso2registry><indexingConfiguration><startIndexing> element to false in the <API-M\_HOME>/repository/config file of the relevant nodes.

## Throttle data and Analytics-related settings

This section describes the parameters you need to configure to tune the performance of API-M Analytics and Throttling when it is affected by high load, network traffic etc. You need to tune these parameters based on the deployment environment.

Tuning data-agent parameters

The following parameters should be configured in the <APIM\_ANALYTICS\_HOME>/repository/conf/data-bridge/data-agent-config.xml file. Note that there are two sub-sections in this file, named **Thrift** and **Binary**.

```
<DataAgentsConfiguration>
 <Agent>
 <Name>Thrift</Name>
 ...
 </Agent>

 <Agent>
 <Name>Binary</Name>
 ...
 </Agent>
</DataAgentsConfiguration>
```

The Thrift section is related to Analytics and the Binary section is related to Throttling. Same set of parameters mentioned below can be found in both sections. The parameter descriptions and recommendations are intended towards the for performance tuning of Analytics, but the same recommendations are relevant for Throttling data related tuning in the Binary section. Note that the section for Thrift is relevant only if Analytics is enabled.

Parameter	Description	Default Value	Tuning Recommendation
QueueSize	The number of messages that can be stored in WSO2 API-M at a given time before they are published to the Analytics Server.	32768	<p>This value should be increased when the Analytics server is under stress or there is high network traffic. This prevents the Analytics server from failing due to message error.</p> <p>When the Analytics server is not very busy and queue size can be reduced to avoid an overconsumption of memory.</p> <p>The number specified for this parameter should be a power of 2.</p>

BatchSize	The WSO2 API-M statistical data sent to the Analytics Server to be published in the Analytics Dashboard are grouped into batches. This parameter specifies the number of requests to be included in a batch.	200	This value should be tuned in proportion to the volt Analytics Server. This value should be reduced if your Analytics Server. This value should be increased if statistics and if the QueueSize cannot be further increased in memory.
CorePoolSize	The number of threads allocated to publish WSO2 API-M statistical data to the Analytics Server via Thrift at the time WSO2 API-M is started. This value increases when the throughput of statistics generated increases. However, the number of threads will not exceed the number specified for the MaxPoolSize parameter.	1	The number of available CPU cores should be taken into account. Increasing the core pool size may improve the throughput of the Analytics Dashboard, but latency will also be increased.

MaxPoolSize	The maximum number of threads that should be allocated at any given time to publish WSO2 API-M statistical data to the Analytics Server.	1	The number of available CPU cores should be taken into account when setting this parameter. Increasing the maximum core pool size may improve performance in the Analytics Dashboard, since more threads can handle events. However, latency will also increase since context switching will take place more frequently.
MaxTransportPoolSize	The maximum number of transport threads that should be allocated at any given time to publish WSO2 API-M statistical data to the Analytics Server.	250	This value must be increased when there is an increase in the number of publishers. The value of the <code>tcpMaxWorkerThreads</code> parameter in <code>repository/conf/data-bridge/data-bridge-config.xml</code> must be specified for this parameter and the number of data publishers. The value for this parameter is 250 and the number of receiver threads specified for the <code>tcpMaxWorkerThreads</code> parameter must be 1750 (i.e. that there are enough receiver threads to handle the publishers).
SecureMaxTransportPoolSize	The maximum number of secure transport threads that should be allocated at any given time to publish WSO2 API-M statistical data to the Analytics Server.	250	This value must be increased when there is an increase in the number of publishers. The value of the <code>sslMaxWorkerThreads</code> parameter in <code>repository/conf/data-bridge/data-bridge-config.xml</code> must be specified for this parameter and the number of data publishers. The value for this parameter is 250 and the number of receiver threads specified for the <code>sslMaxWorkerThreads</code> parameter must be 1750 (i.e. that there are enough receiver threads to handle the publishers).

## WSO2 API-M Performance and Capacity Planning

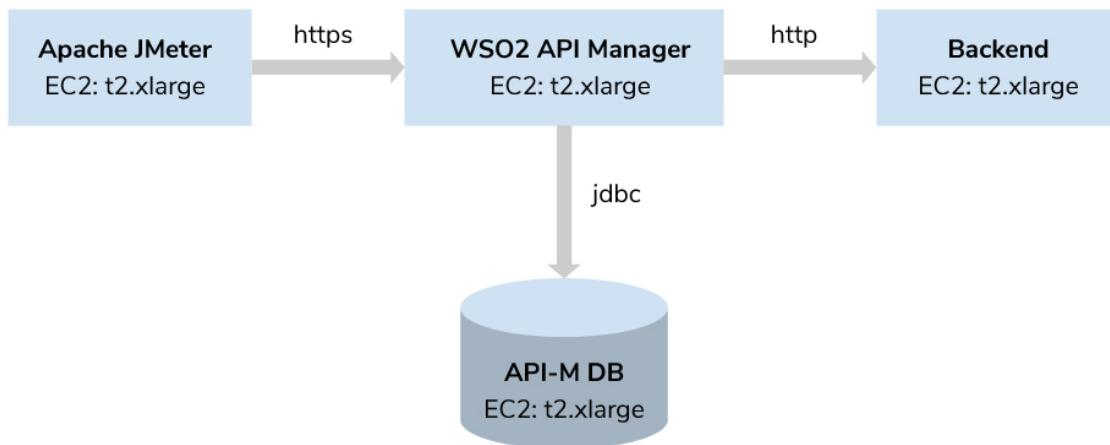
Download the latest Performance Test Results for WSO2 API Manager 2.1.0 from [here](#). If you need any additional information, please [contact us](#).

The following sections analyze the results of WSO2 API Manager performance tests done in the Amazon EC2 environment.

- Deployment
- Backend Service
- Test Scenario
- Measuring Performance
- Performance Testing Tool
- Performance Tuning of WSO2 API Manager

- Performance Test Results
  - Scenario 1: Echo API in WSO2 API Manager
  - Scenario 2: Echo API in WSO2 API Manager and 1 second Think-Time
  - Comparison - Echo API with no think-time vs. 1sec think-time
- Conclusion

## Deployment



The following are the details with regard to the WSO2 API-M 2.1.0 deployment that is depicted above.

- Four (4) EC2 t2.xlarge instances are used for deployment as shown above. Each instance has 4 CPUs and 16GB of memory.
  - The operating system is Ubuntu 16.04.2 LTS.
  - Apache JMeter version is 3.2. is used in this deployment.
- Apache JMeter is the preferred load testing tool at WSO2. As there are a high number of concurrent users, we have increased the number of sockets supported in the server.
- The following commands were used in the instance used by JMeter.

```

sudo sysctl -w net.ipv4.ip_local_port_range="1025 65535"
sudo sysctl -w net.ipv4.tcp_tw_reuse=1

```

- The first command increases the port range used for the client.
- The second command allows to reuse the sockets in the TIME\_WAIT state. These are safe commands that can be used in the client side.  
For more information, go to <https://vincent.bernat.im/en/blog/2014-tcp-time-wait-state-linux>
- The ulimit was increased in all servers.  
For more information, see [Tuning Performance](#).

## Backend Service

The backend service used for testing was developed using Netty. Since there can be up to 3000 users, we used 3000 threads for Netty in order to avoid any bottlenecks in the backend. The Netty server also has a parameter that can be used to simulate the delays in the backend service by simply specifying the sleep time in seconds.

## Test Scenario

The test scenario focuses on performing an API proxy invocation using API Manager that in turn will echo the API. Tests were done using 100, 200, 300, 1000, 2000, and 3000 concurrent users.

## Measuring Performance

Two key performance metrics are used to measure the performance, namely Latency and Throughput. Throughput measures the number of messages that a server processes during a specific time interval (e.g., per second). The throughput is calculated using the following equation.

$$\text{Throughput} = \frac{\text{Number of requests}}{\text{Time to complete the requests}}$$

Latency measures the end-to-end processing time for an operation. Every operation has its own latency. Therefore, we are interested in how latency behaves. In order to see this behavior, we must have the complete distribution of latencies.

### Performance Testing Tool

In Apache JMeter we specified the number of concurrent users, ran the test, and got the results specified under the [Performance Test Results](#) section. The following section provides details of the for the terminology used in the [Performance Test Results](#) section.

- **Error Count** - How many request errors were recorded.
- **Error %** - Percent of requests with errors
- **Average** - The average response time of a set of results
- **Min** - The shortest time taken for a request
- **Max** - The longest time taken for a request
- **90th Percentile** - 90% of the requests took no more than this time. The remaining samples took at least as long as this.
- **95th Percentile** - 95% of the requests took no more than this time. The remaining samples took at least as long as this.
- **99th Percentile** - 99% of the requests took no more than this time. The remaining samples took at least as long as this.
- **Throughput** - The throughput is measured in requests per second.
- **Received KB/sec** - The throughput is measured in received Kilobytes per second.
- **Sent KB/sec** - The throughput is measured in sent Kilobytes per second.

Think time is the delay between two requests. For example, a think time of 1 second means that when there are 2000 concurrent users, each user will send the next request 1 second after he/she gets a response for the previous request. In a realistic scenario there will always be more think-time in between requests and with that, the performance numbers obtained will be much better.

Measuring latency using a single request (e.g., sending a cURL request while the load test is running) may not be useful. This is why we keep the latencies of each request and get statistics from the complete distribution of latencies.

In addition, to the above details, we also obtained some additional details for every test done for a given number of concurrent users.

- **Load Average** - The load average of the system was taken from the sar (System Activity Report) output. We took the maximum load average reported during the test period.
- **GC Throughput** - Time percentage the application was not busy with garbage collection (GC).
- **Total heap usage** - Max memory usage in total reserved heap.
- **Allocated Max** - Max memory allocated for the heap by the Java Virtual Machine (JVM).
- **Heap Usage** - Total heap usage as a percentage of maximum allocated heap.
- **Max heap after full GC** - Maximum size of live objects used by the application.
- **Max heap after full GC %** - The maximum size of live objects as a percentage of maximum allocated heap.

Last 6 details were obtained from the GC logs produced by the WSO2 API Manager.

The following are the GC flags that were used.

```
-XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps
-Xloggc: "$CARBON_HOME/repository/logs/gc.log"
```

The process memory was also obtained for each test. However, we did not include those values as Java is working on an already reserved heap area.

## Performance Tuning of WSO2 API Manager

The heap of WSO2 API Manager was increased to 4GB and that is the only change done for WSO2 API Manager, which had an impact on the performance.

When doing performance tests, the heap memory was initially set at 2GB as this is the recommended heap memory for WSO2 API Manager. However, when the tests were done with large numbers of concurrent users using 2GB heap, the GC throughput observed was less than 90%. We recommend that the GC throughput to be maintained above 90% and therefore we increased the heap memory to 4GB and ran a similar set of tests.

We also increased the socket timeouts in WSO2 API Manager. However, it was not needed as the performance results of WSO2 API Manager never reported latencies greater than 60 seconds, which is the default socket timeout.

The number of worker threads used were 400 (default value) and the number of threads may increase up to 500 (default value).

## Performance Test Results

### Scenario 1: Echo API in WSO2 API Manager

There is no delay in the backend service. This test was done mainly to see the maximum throughput of the WSO2 API Manager in the EC2 environment.

Concurrent Users	Error %	Response Times (ms)						Throughput	Load Average	GC Logs		
		Average	Min	Max	90th Percentile	95th Percentile	99th Percentile			GC Throughput	Heap Usage	Max heap after full GC %
100	0.00%	13.88 ms	1.00 ms	3,930.00 ms	18.00 ms	25.00 ms	121.00 ms	7169.71	8.12	90.16%	95.99%	39.31%
200	0.00%	21.78 ms	1.00 ms	2,404.00 ms	37.00 ms	55.00 ms	125.00 ms	9119.14	10.53	91.73%	95.97%	36.41%
300	0.00%	34.50 ms	1.00 ms	3,584.00 ms	46.00 ms	63.00 ms	120.00 ms	8650.19	11.98	89.08%	96.31%	39.60%
1000	0.00%	116.43 ms	1.00 ms	15,270.00 ms	197.00 ms	245.00 ms	414.00 ms	8514.46	13.82	87.11%	94.67%	47.76%
2000	0.00%	290.91 ms	1.00 ms	15,873.00 ms	968.00 ms	1,415.00 ms	3,099.00 ms	6765.31	13.39	82.42%	92.43%	48.66%
3000	0.00%	545.33 ms	1.00 ms	128,354.00 ms	1,434.00 ms	2,245.95 ms	3,178.96 ms	5403.43	10.58	80.62%	92.46%	43.93%

### Scenario 2: Echo API in WSO2 API Manager and 1 second Think-Time

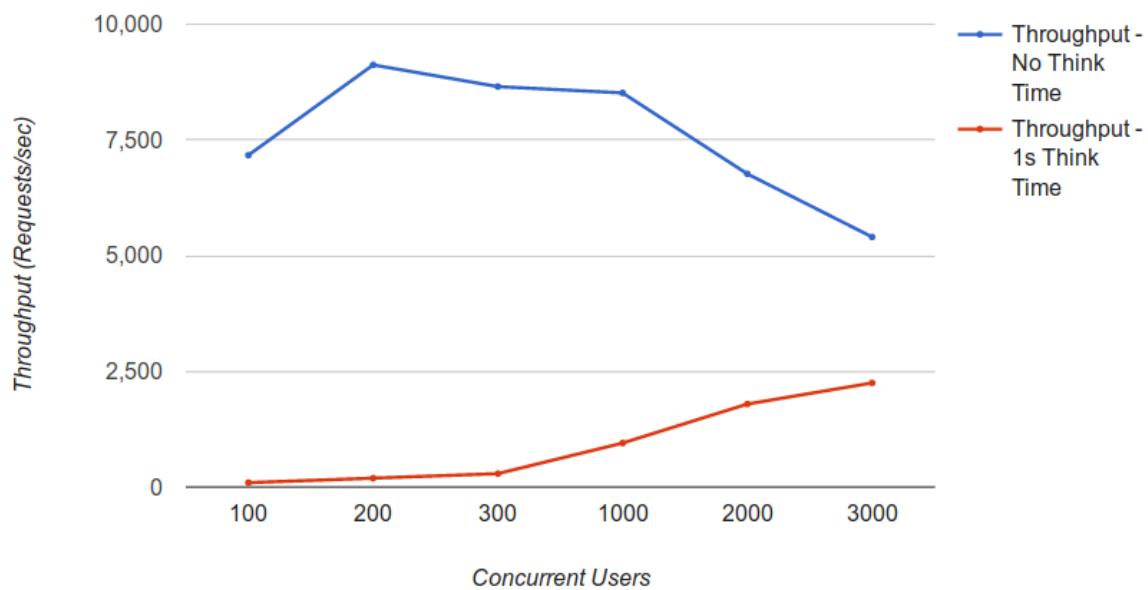
The test mentioned in Scenario 1 was repeated with a 1 second think-time. This means that there is a one second gap in between the requests sent by a user (in JMeter). In a realistic scenario, there will always be some think-time in between requests. This test was done to understand the performance of WSO2 API Manager when there is a think time.

There is no delay in the backend service.

Concurrent Users	Executions	Response Times (ms)							Throughput	Load Average	GC Logs		
		Error %	Average	Min	Max	90th Percentile	95th Percentile	99th Percentile			GC Throughput	Heap Usage	Max heap after full GC %
100	0.00%	3.48 ms	1.00 ms	1,001.00 ms	5.00 ms	11.00 ms	31.00 ms	97.42	0.52	99.72%	36.75%	5.10%	
200	0.00%	3.18 ms	1.00 ms	1,000.00 ms	4.00 ms	5.00 ms	20.00 ms	194.81	0.35	99.63%	41.15%	6.08%	
300	0.00%	3.41 ms	1.00 ms	1,002.00 ms	4.00 ms	8.00 ms	31.00 ms	292.07	1.10	99.56%	40.68%	5.25%	
1000	0.00%	23.33 ms	1.00 ms	4,393.00 ms	38.00 ms	90.00 ms	1,014.00 ms	952.68	1.73	97.58%	58.27%	5.11%	
2000	0.00%	84.09 ms	1.00 ms	5,562.00 ms	166.00 ms	278.95 ms	1,000.00 ms	1794.9	3.00	95.62%	95.25%	23.53%	
3000	0.00%	279.34 ms	1.00 ms	15,377.00 ms	1,961.90 ms	3,167.00 ms	4,136.00 ms	2250.83	5.22	91.50%	91.89%	29.47%	

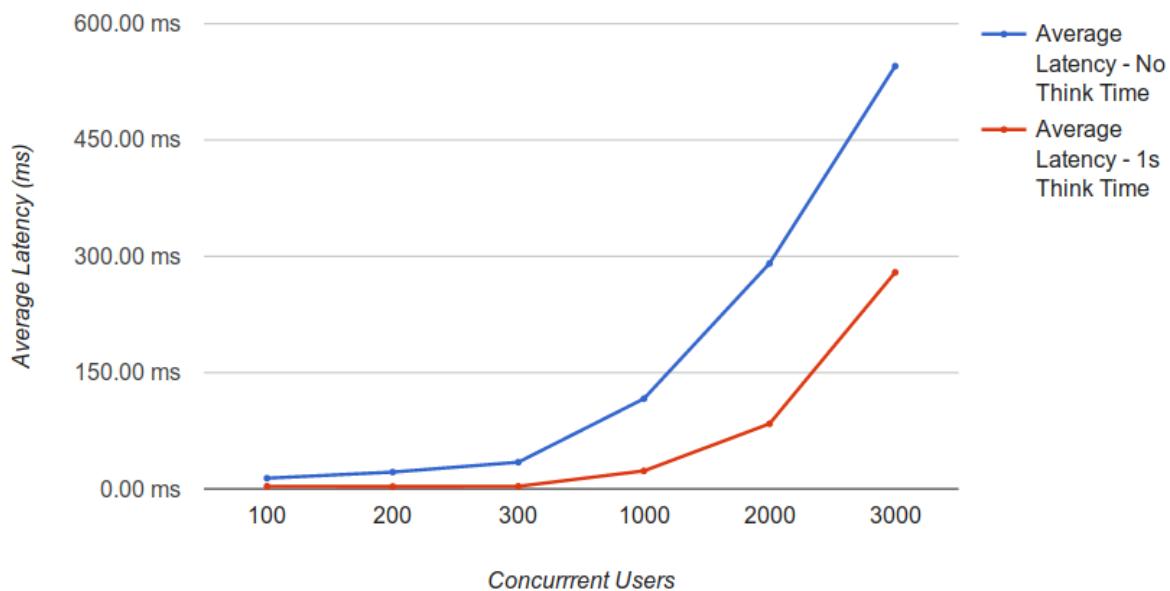
### Comparison - Echo API with no think-time vs. 1sec think-time

Throughput vs Concurrent Users: Echo API (No Backend Delay)



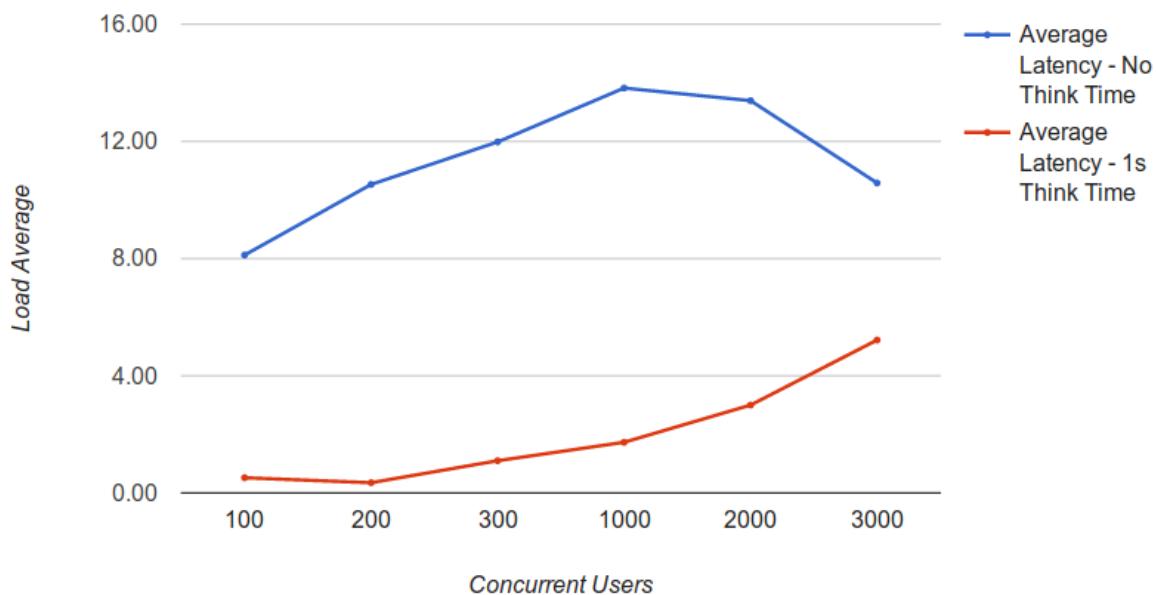
Throughput with 1 second think-time is much less than the throughput without the think time. Adding a think-time reduces the arrival rate of the requests.

### Average Latency vs Concurrent Users: Echo API (No Backend Delay)



Latencies of requests are better as the arrival rate of requests is low.

### Load Average vs Concurrent Users: Echo API (No Backend Delay)



Load average is also much better due to the low arrival rate of requests.

### Conclusion

We analyzed the performance of WSO2 API Manager using 100, 200, 300, 1k, 2k, and 3k threads using 1, 5, and 30 second backend delay.

Except for increasing the heap size of API Manager, there were no other specific optimization techniques (i.e., performance tuning) used to optimize the performance of WSO2 API Manager. We only increased the heap size to 4GB from 2GB, which is the default.

When there is no “think-time” added after a request, it means that for example when there are 2000 concurrent users, each user will send the next request as soon as she gets a response for the previous request. In a realistic scenario, there will always be some think-time in between requests. Therefore, with that the latency numbers obtained are much better. The load average is also much better with think-time.

### Removing Unused Tokens from the Database

As you use WSO2 API Manager, the number of revoked, inactive and expired tokens accumulates in the IDN\_OAUTH2\_ACCESS\_TOKEN table. These tokens are kept in the database for logging and audit purposes, but they can have a negative impact on the server's performance over time. Therefore, it is recommended to clean them periodically as given in the instructions below:

**Tip :** It is safe to run these steps in read-only mode or during a time when traffic on the server is low, but that is not mandatory.

1. Take a backup of the running database.
2. Set up the database dump in a test environment and test it for any issues.

For more information on setting up a database dump, go to the [MySQL](#), [SQL Server](#), and [Oracle](#) official documentation.

**Tip:** We recommend you to test the database dump before the cleanup task as the cleanup can take some time.

3. Run the following script (select one according to your database) on the database dump. It takes a backup of the necessary tables, turns off SQL updates and cleans the database of unused tokens.

MySQLSQL ServerOracle DB

The following script has been tested on MySQL 5.7.

```
-- USE apimdb_cleanup;

DROP PROCEDURE IF EXISTS cleanup_tokens;

DELIMITER $$

CREATE PROCEDURE cleanup_tokens ()
BEGIN

-- Backup IDN_OAUTH2_ACCESS_TOKEN table
DROP TABLE IF EXISTS IDN_OAUTH2_ACCESS_TOKEN_BAK;
CREATE TABLE IDN_OAUTH2_ACCESS_TOKEN_BAK AS SELECT * FROM
IDN_OAUTH2_ACCESS_TOKEN;

-- 'Turn off SQL_SAFE_UPDATES'
SET @OLD_SQL_SAFE_UPDATES = @@SQL_SAFE_UPDATES;
SET SQL_SAFE_UPDATES = 0;

-- 'Keep the most recent INACTIVE key for each CONSUMER_KEY,
-- AUTHZ_USER, TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE';
```

```

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM
(SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE') x GROUP
BY CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH) y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE' AND
ACCESS_TOKEN NOT IN (SELECT * FROM (SELECT ACCESS_TOKEN FROM
IDN_OAUTH2_ACCESS_TOKEN T1 WHERE TIME_CREATED = (SELECT
MAX(TIME_CREATED) AS LATEST_TOKEN_TIME FROM IDN_OAUTH2_ACCESS_TOKEN T2
WHERE TOKEN_STATE = 'INACTIVE' AND T1.CONSUMER_KEY_ID =
T2.CONSUMER_KEY_ID AND T1.AUTHZ_USER = T2.AUTHZ_USER GROUP BY
CONSUMER_KEY_ID , AUTHZ_USER , TOKEN_STATE)) AS T0);

SELECT 'AFTER:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE';

-- 'Keep the most recent REVOKED key for each CONSUMER_KEY,
AUTHZ_USER, TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_REVOKED_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM
(SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED') x GROUP BY
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH) y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED' AND
ACCESS_TOKEN NOT IN (SELECT * FROM (SELECT ACCESS_TOKEN FROM
IDN_OAUTH2_ACCESS_TOKEN T1 WHERE TIME_CREATED = (SELECT
MAX(TIME_CREATED) AS LATEST_TOKEN_TIME FROM IDN_OAUTH2_ACCESS_TOKEN T2
WHERE TOKEN_STATE = 'REVOKED' AND T1.CONSUMER_KEY_ID =
T2.CONSUMER_KEY_ID AND T1.AUTHZ_USER = T2.AUTHZ_USER GROUP BY
CONSUMER_KEY_ID , AUTHZ_USER , TOKEN_STATE)) AS T0);

SELECT 'AFTER:TOTAL_REVOKED_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED';

-- 'Keep the most recent EXPIRED key for each CONSUMER_KEY,
AUTHZ_USER, TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_EXPIRED_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED';

SELECT 'TO BE RETAINED', COUNT(*) FROM (SELECT ACCESS_TOKEN FROM
(SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED') x GROUP BY
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH) y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED' AND

```

```
ACCESS_TOKEN NOT IN (SELECT * FROM (SELECT ACCESS_TOKEN FROM
IDN_OAUTH2_ACCESS_TOKEN T1 WHERE TIME_CREATED = (SELECT
MAX(TIME_CREATED) AS LATEST_TOKEN_TIME FROM IDN_OAUTH2_ACCESS_TOKEN T2
WHERE TOKEN_STATE = 'EXPIRED' AND T1.CONSUMER_KEY_ID =
T2.CONSUMER_KEY_ID AND T1.AUTHZ_USER = T2.AUTHZ_USER GROUP BY
CONSUMER_KEY_ID , AUTHZ_USER , TOKEN_STATE)) AS T0);

SELECT 'AFTER:TOTAL_EXPIRED_TOKENS' , COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED';

-- 'Restore the original SQL_SAFE_UPDATES value'
SET SQL_SAFE_UPDATES = @OLD_SQL_SAFE_UPDATES;
```

```
END$$
DELIMITER ;
```

- Uncomment the following in the above script and replace apimdb with name of your API Manager database.

```
-- USE apimdb_cleanup;
```

```
-- Replace WSO2APIMDB with your database name
USE WSO2APIMDB;
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P' AND name =
'cleanup_tokens')
DROP PROCEDURE cleanup_tokens
GO

CREATE PROCEDURE cleanup_tokens
AS
BEGIN

-- Backup IDN_OAUTH2_ACCESS_TOKEN table
IF (EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME =
'IDN_OAUTH2_ACCESS_TOKEN_BAK'))
BEGIN
 DROP TABLE dbo.IDN_OAUTH2_ACCESS_TOKEN_BAK;
END

SELECT * INTO IDN_OAUTH2_ACCESS_TOKEN_BAK FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN;

-- 'Keep the most recent INACTIVE key for each CONSUMER_KEY,
-- AUTHZ_USER, TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE';
SELECT 'TO BE RETAINED', COUNT(ACCESS_TOKEN) FROM(SELECT
max(ACCESS_TOKEN) ACCESS_TOKEN FROM (SELECT ACCESS_TOKEN,
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE') x
GROUP BY CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH)y;
DELETE FROM dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE =
'INACTIVE' AND ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM(SELECT
max(ACCESS_TOKEN) ACCESS_TOKEN FROM (SELECT ACCESS_TOKEN,
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE') x
GROUP BY CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH)y);
```

```
-- 'Keep the most recent REVOKED key for each CONSUMER_KEY,
AUTHZ_USER, TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_REVOKED_TOKENS', COUNT(*) FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT
max(ACCESS_TOKEN)ACCESS_TOKEN FROM (SELECT ACCESS_TOKEN,
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED') x GROUP BY
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH)y;
DELETE FROM dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED'
AND ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM(SELECT
max(ACCESS_TOKEN) ACCESS_TOKEN FROM (SELECT ACCESS_TOKEN,
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED') x GROUP BY
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH)y);

SELECT 'AFTER:TOTAL_REVOKED_TOKENS', COUNT(*) FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED';

-- 'Keep the most recent EXPIRED key for each CONSUMER_KEY,
AUTHZ_USER, TOKEN_SCOPE combination'
SELECT 'BEFORE:TOTAL_EXPIRED_TOKENS', COUNT(*) FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT max(ACCESS_TOKEN)
ACCESS_TOKEN FROM (SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER,
TOKEN_SCOPE_HASH FROM dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE =
'EXPIRED') x GROUP BY CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH)y;

DELETE FROM dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED'
AND ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM(SELECT
max(ACCESS_TOKEN) ACCESS_TOKEN FROM (SELECT ACCESS_TOKEN,
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH FROM
dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED') x GROUP BY
CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH)y);

SELECT 'AFTER:TOTAL_EXPIRED_TOKENS', COUNT(*) FROM
```

```

dbo.IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED';

END

```

Please note that the stored procedure is not included in the script given below.

The following script has been tested on Oracle 11g.

```

DROP TABLE IDN_OAUTH2_ACCESS_TOKEN_BAK;

CREATE TABLE IDN_OAUTH2_ACCESS_TOKEN_BAK AS SELECT * FROM
IDN_OAUTH2_ACCESS_TOKEN;

SELECT 'BEFORE:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM
(SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE'));

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE' AND
ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM IDN_OAUTH2_ACCESS_TOKEN
T1 WHERE TIME_CREATED = (SELECT MAX(TIME_CREATED) AS LATEST_TOKEN_TIME
FROM IDN_OAUTH2_ACCESS_TOKEN T2 WHERE TOKEN_STATE = 'INACTIVE' AND
T1.CONSUMER_KEY_ID = T2.CONSUMER_KEY_ID AND T1.AUTHZ_USER =
T2.AUTHZ_USER GROUP BY CONSUMER_KEY_ID , AUTHZ_USER , TOKEN_STATE));

SELECT 'AFTER:TOTAL_INACTIVE_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'INACTIVE';

SELECT 'BEFORE:TOTAL_REVOKED_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED';

SELECT 'TO BE RETAINED', COUNT(*) FROM(SELECT ACCESS_TOKEN FROM
(SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED') x GROUP BY
ACCESS_TOKEN,CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH) y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED' AND
ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM IDN_OAUTH2_ACCESS_TOKEN
T1 WHERE TIME_CREATED = (SELECT MAX(TIME_CREATED) AS LATEST_TOKEN_TIME
FROM IDN_OAUTH2_ACCESS_TOKEN T2 WHERE TOKEN_STATE = 'REVOKED' AND
T1.CONSUMER_KEY_ID = T2.CONSUMER_KEY_ID AND T1.AUTHZ_USER =
T2.AUTHZ_USER GROUP BY CONSUMER_KEY_ID , AUTHZ_USER , TOKEN_STATE));

SELECT 'AFTER:TOTAL_REVOKED_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'REVOKED';

```

```
SELECT 'BEFORE:TOTAL_EXPIRED_TOKENS' , COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED';

SELECT 'TO BE RETAINED', COUNT(*) FROM (SELECT ACCESS_TOKEN FROM
(SELECT ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH
FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED') X GROUP BY
ACCESS_TOKEN, CONSUMER_KEY_ID, AUTHZ_USER, TOKEN_SCOPE_HASH) Y;

DELETE FROM IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED' AND
ACCESS_TOKEN NOT IN (SELECT ACCESS_TOKEN FROM IDN_OAUTH2_ACCESS_TOKEN
T1 WHERE TIME_CREATED = (SELECT MAX(TIME_CREATED) AS LATEST_TOKEN_TIME
FROM IDN_OAUTH2_ACCESS_TOKEN T2 WHERE TOKEN_STATE = 'EXPIRED' AND
T1.CONSUMER_KEY_ID = T2.CONSUMER_KEY_ID AND T1.AUTHZ_USER =
T2.AUTHZ_USER GROUP BY CONSUMER_KEY_ID , AUTHZ_USER , TOKEN_STATE));
```

```
SELECT 'AFTER:TOTAL_EXPIRED_TOKENS', COUNT(*) FROM
IDN_OAUTH2_ACCESS_TOKEN WHERE TOKEN_STATE = 'EXPIRED';
```

4. Once the cleanup is over, start the API Manager pointing to the cleaned-up database dump and test thoroughly for any issues.

You can also schedule a cleanup task that will automatically run after a given period of time. Here's an example:

MySQLSQL Server

```
USE 'WSO2AM_DB';
DROP EVENT IF EXISTS 'cleanup_tokens_event';
CREATE EVENT 'cleanup_tokens_event'
 ON SCHEDULE
 EVERY 1 WEEK STARTS '2015-01-01 00:00:00'
 DO
 CALL 'WSO2AM_DB'.'cleanup_tokens'();

-- 'Turn on the event_scheduler'
SET GLOBAL event_scheduler = ON;
```

```
USE WSO2AM_DB ;
GO
-- Creates a schedule named CleanupTask.
-- Jobs that use this schedule execute every day when the time on the
server is 01:00.
EXEC sp_add_schedule
 @schedule_name = N'CleanupTask' ,
 @freq_type = 4,
 @freq_interval = 1,
 @active_start_time = 010000 ;
GO
-- attaches the schedule to the job BackupDatabase
EXEC sp_attach_schedule
 @job_name = N'BackupDatabase',
 @schedule_name = N'CleanupTask' ;
GO
```

Replace WSO2AM\_DB with the name of your API Manager database in the above script.

### Migrating the APIs to a Different Environment

If you maintain multiple environments of the same API Manager version and you want to move all created APIs from one environment to another (such as moving from a development environment to a QA environment or from a QA environment to a production environment), you can migrate the APIs to the new environment by following the steps below:

- Deploying the API import/export tool
- Exporting an API
- Importing an API

- API import/export in a tenanted environment
- Understanding the API import/export tool

#### *Deploying the API import/export tool*

1. Download WSO2 API Manager 2.1.0 from <http://wso2.com/products/api-manager/>.
2. Download the latest WSO2 API import/export tool from [here](#).

Note that the import/export tool attached is specific to this version of WSO2 API Manager.

3. Copy the downloaded `api-import-export-2.1.0-v3.war` file to the `<APIM_HOME>/repository/deployment/server/webapps` folder.
4. Start the API Manager. If the server is already started, the file is automatically deployed as hot deployment is enabled.

#### *Exporting an API*

After successfully deploying the import/export tool, you can export an existing API as a `.zip` archive. Issue the following cURL command using the command line:

```
curl -H "Authorization:Basic
<base64-encoded-credentials-separated-by-a-colon>" -X GET
"https://<APIM_HOST:Port>/api-import-export-<product-version>-<tool-versio
n>/export-api?name=<API-name>&version=<API-version>&provider=<API-provider
>" -k > <exportedApiName>.zip
```

To obtain **<base64-encoded-credentials-separated-by-a-colon>** use a base64 encoder (e.g., <https://www.base64encode.org/>) to encode your username and password which has the admin role using the following format: `<username>:<password>` Thereafter, enter the encoded value for this parameter.

Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -X GET
"https://localhost:9443/api-import-export-2.1.0-v3/export-api?name=PizzaSh
ackAPI&version=1.0.0&provider=admin" -k > myExportedAPI.zip
```

#### *Importing an API*

You can use the archive created in the previous section to import APIs to an API Manager instance.

1. Make sure that API Manager is started and the import/export tool is deployed.

## For Secure Endpoint Enabled APIs:

If you have enabled secure endpoints when creating the API, please follow the steps below before importing the API.

1. Unzip the `.zip` archive created in the previous section.

2. Go to **PizzaShackAPI-1.0.0/Meta-information** and open the api.json file.
3. Modify the "endpointUTPassword" with your endpoint password and save the api.json file.
4. Compress the **PizzaShackAPI-1.0.0** folder to a folder named **myExportedAPI**

2. Run the following cURL command:

```
curl -H "Authorization:Basic
<base64-encoded-username-and-password-separated-by-a-colon>
-F file=@"/full/path/to/the/zip/file" -k -X POST
"https://<APIManagerHost:Port>/api-import-export-<product-version>-<tool-version>/import-api"
```

Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -F
file=@"/Desktop/MyAPIFolder/myExportedAPI.zip" -k -X POST
"https://localhost:9443/api-import-export-2.1.0-v3/import-api"
```

You must add a parameter named **preserveProvider** to the cURL command and set its value to **false** if the API is imported to a different domain than its exported one. This parameter sets the provider of the imported API to the user who is issuing the cURL command. Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -F
file=@"/Desktop/MyAPIFolder/myExportedAPI.zip" -k -X POST
"https://localhost:9443/api-import-export-2.1.0-v3/import-api?p
reserveProvider=false"
```

The **preserveProvider** parameter is used to decide whether to keep the actual Provider as the provider of the API or change the provider to the user who is importing the API to a different environment.

As an example, If **preserveProvider** is set to **true**, when importing an API created by user-1 in environment-1 will be preserved as the provider when and after importing that API to environment-2 by user-2. If **preserveProvider** is set to **false**, when importing that API created by user-1 to the environment-2, the provider will be changed (not preserved) to user-2 who is importing the API.

## Troubleshooting

After importing, if the APIs are not visible in the API Publisher UI, do the following to re-index the artifacts in the registry.

1. Rename the `<lastAccessTimeLocation>` element in the `<API-M_2.1.0_HOME>/repository/conf/registry.xml` file. If you use a **clustered/distributed API Manager setup**, change the file in the API Publisher node. For example, change the `/_system/local/repository/compone`

nts/org.wso2.carbon.registry/indexing/lastaccesstime registry path to /\_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime\_1.

2. Shut down the API Manager 2.1.0, backup and delete the <API-M\_2.1.0\_HOME>/solr directory.

For more information, see [Upgrading the API Manager to 2.1.0](#).

#### **API import/export in a tenanted environment**

- To export an API from a tenant, follow the steps in [Export an API](#). Use the tenant-specific encoded credentials in the cURL command. Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -X GET
"https://<host>:<port>/api-import-export-2.1.0-v3/export-api?name=sampl
e&version=1.0.0&provider=user@domain.com" -k > exportedApiName.zip
```

- To import the API in another tenant, follow the steps in [Importing an API](#). Use the encoded credentials for this tenant in the cURL command. Here's an example:

```
curl -H "Authorization:Basic AbCdEfG" -F
file=@"/home/user/Desktop/exportedApiName.zip" -k -X POST
"https://<host>:<port>/api-import-export-2.1.0-v3/import-api?preservePr
ovider=false"
```

Note that the `preserveProvider` parameter value should be set to `false`.

#### **Understanding the API import/export tool**

The API import/export tool uses a RESTful API, protected by basic authentication. Only users with **admin role** are allowed to access it in the initial phase. To allow access to the import/export feature only for a particular tenant, log in to WSO2 API Manager's Management Console and add the downloaded archive file as a web application to the server.

To access API import/export, you need to assign the **admin role** to the user.

'admin' role is the default role which is specified in the Realm configuration in <APIM-HOME>/repository/conf/user-mgt.xml. It will be changed if you have changed this <AdminRole> value in below configuration.

```
<Realm>
 <Configuration>
 <AddAdmin>true</AddAdmin>
 <AdminRole>admin</AdminRole>

 </Configuration>
</Realm>
```

## The export functionality

API export functionality retrieves the information required for the requested API from the registry and databases and generates a ZIP file, which the exporter can download. This exported ZIP file has the following structure:

```
<APIName>-<version>
|_ Meta Information
 |_ api.json
 |_ swagger.json
|_ Documents
 |_ docs.json
 |_ documents with type 'file'
|_ Image
 |_ icon.<extension>
|_ WSDL
 |_ <ApiName>-<version>.wsdl
|_ Sequences
 |_ In Sequence
 |_<Sequence Name>.xml
 |_ Out Sequence
 |_<Sequence Name>.xml
 |_ Fault Sequence
 |_<Sequence Name>.xml
```

The structure of the ZIP file is explained below:

Sub directory/File	Description
Meta Information	<ul style="list-style-type: none"> <li>api.json: contains all the basic information required for an API to be imported to another environment</li> <li>swagger.json: contains the API swagger definition</li> </ul>
Documents	<ul style="list-style-type: none"> <li>docs.json: contains the summary of all the documents available for the API</li> <li>Add the uploaded files for API documentation also</li> </ul>
Image	Thumbnail image of the API
WSDL	WSDL file of the API
Sequences	The sequences available for the API

Given below is the RESTful API for the export functionality. It is secured using Basic Authentication.

Parameter	Description
URI	<a href="https://&lt;host-name&gt;:9443/api-import-export-&lt;product-version&gt;-&lt;tool-version&gt;">https://&lt;host-name&gt;:9443/api-import-export-&lt;product-version&gt;-&lt;tool-version&gt;</a>
Query parameters	name=<api_name>&version=<api_version>&provider=<provider_name>

HTTP method	GET
Examples	<pre>curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -X GET "https://localhost:9443/api-import-export-2.1.0-v3/export-api?name=test&amp;ver</pre> <p>It gives a data stream as the output. To download it as a zipped archive, use the following command:</p> <pre>curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -X GET "https://localhost:9443/api-import-export-2.1.0-v3/export-api?name=test&amp;ver -k &gt; exportedApi.zip</pre> <p>To verify the output status of the API call:</p> <pre>curl -v -H "Authorization:Basic YWRtaW46YWRtaW4=" -X GET "https://localhost:9443/api-import-export-2.1.0-v3/export-api?name=test&amp;ver -k &gt; exportedApi.zip</pre>

## The import functionality

The import functionality uploads the exported ZIP file of the API to the target environment. It creates a new API with all the registry and database resources exported from the source environment. Note the following:

- The lifecycle status of an imported API will always be CREATED even when the original API in the source environment has a different state. This is to enable the importer to modify the API before publishing it.
- Tiers and sequences are provider-specific. If an exported tier is not already available in the imported environment, that tier is not added to the new environment. However, if an exported sequence is not available in the imported environment, it is added.
- The importer can decide whether to keep the original provider's name or replace it. Set the `preserveProvider` parameter to true to keep it. If you set it to false, the original provider is replaced by the user who is sending the cURL command.
- Cross-tenant imports are not allowed by preserving the original provider. For example, if an API is exported from tenant A and imported to tenant B, the value of the `preserveProvider` parameter must always be false.

Given below is the RESTful API for the import functionality.

Parameter	Description
URI	<code>https://&lt;host&gt;:9443/api-import-export-&lt;product-version&gt;-&lt;tool-version&gt;/import-api</code>
Query parameters	<code>preserveProvider=&lt;true false&gt;</code>
HTTP method	POST
Example	<p>Imports the API with the original provider preserved: <code>curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -F file=@"full/path/to/the/zip/file" -k -X POST https://localhost:9443/api-import-export-2.1.0-v3/import-api"</code></p> <p>Imports the API with the provider set to the current user: <code>curl -H "Authorization:Basic YWRtaW46YWRtaW4=" -F file=@"full/path/to/the/zip/file" -k -X POST https://localhost:9443/api-import-export-2.1.0-v3/import-api?preserveProvider=false</code></p>

## Generating SDKs

Software Development Kits (SDKs) contain the necessary toolkits to create a client application to invoke a particular

API. If an API consumer wants to create an application, they can generate a server stub or client side SDK for a supported language/framework and use it to write a software application to consume the subscribed APIs.

- Generating client SDKs in the API Store
- Generating server stubs and client SDKs in the API Publisher

#### ***Generating client SDKs in the API Store***

**Tip:** A valid API subscription must exist in order to use the SDK. SDK generation is allowed per API.

By default, SDK generation is enabled for Java and Android. If those two languages are sufficient, move to step 3. To enable SDK generation for other languages,

1. Open the <APIM\_Home>/repository/conf/api-manager.xml file and edit the <SupportedLanguages> property of the <SwaggerCodegen> element to include any of the supported languages/frameworks. An example is shown below,

```
<SwaggerCodegen>
 <ClientGeneration>
 <GroupId>org.wso2</GroupId>
 <ArtifactId>org.wso2.client.</ArtifactId>
 <ModelPackage>org.wso2.client.model.</ModelPackage>
 <ApiPackage>org.wso2.client.api.</ApiPackage>
 <!-- Configure supported languages/Frameworks as comma
separated values,
 Supported Languages/Frameworks : android, java, scala,
csharp, cpp, dart, flash, go, groovy, javascript, jmeter,
nodejs, perl, php, python, ruby, swift, clojure, aspNet5,
asyncScala, spring, csharpDotNet2, haskell-->

 <SupportedLanguages>java,android,ruby,php</SupportedLanguages>
 </ClientGeneration>
</SwaggerCodegen>
```

In this example, the supported languages are defined as Java, Android, Ruby and PHP.

2. Once the above configuration is saved, start the API Manager server.
3. Log in to the API Store, open the API and click the **SDKs** tab. Download options are available for each programming language specified earlier in the api-manager.xml file.

The screenshot shows the WSO2 API Store interface. On the left, there's a sidebar with icons for APIs, Applications, Forum, Analytics, and Manage Alerts. The main content area is titled "PizzaShackAPI - 1.0.0". It includes a thumbnail image of a pizza, API metadata (Version: 1.0.0, By: Jane Roe, Updated: 19/Jan/2018 16:44:44 PM IST, Status: PUBLISHED, Rating: 5 stars), and a "Subscribe" button. Below the main content, there are tabs for Overview, API Console, Documentation, SDKs, and Forum. A note says "Download client-side SDKs for this API." followed by download links for Java, Android, Ruby, and PHP.

4. Download the SDK for the preferred language and use it to write software applications to consume the API.

**Tip:** Make sure that you are logged in to the API Store before attempting to download the SDK. As mentioned above, a valid subscription should exist for the API. The SDK cannot be used without a valid access token for the subscription.

#### **Generating server stubs and client SDKs in the API Publisher**

The API Publisher has an embedded **swagger editor** with the ability to generate server code and client SDKs then and there.

Client SDK and Server Stub generation in API Publisher is only supported for Rest APIs.

1. Open an existing API and choose to edit it.



#### PizzaShackAPI

1.0.0

admin

0 Users

PUBLISHED



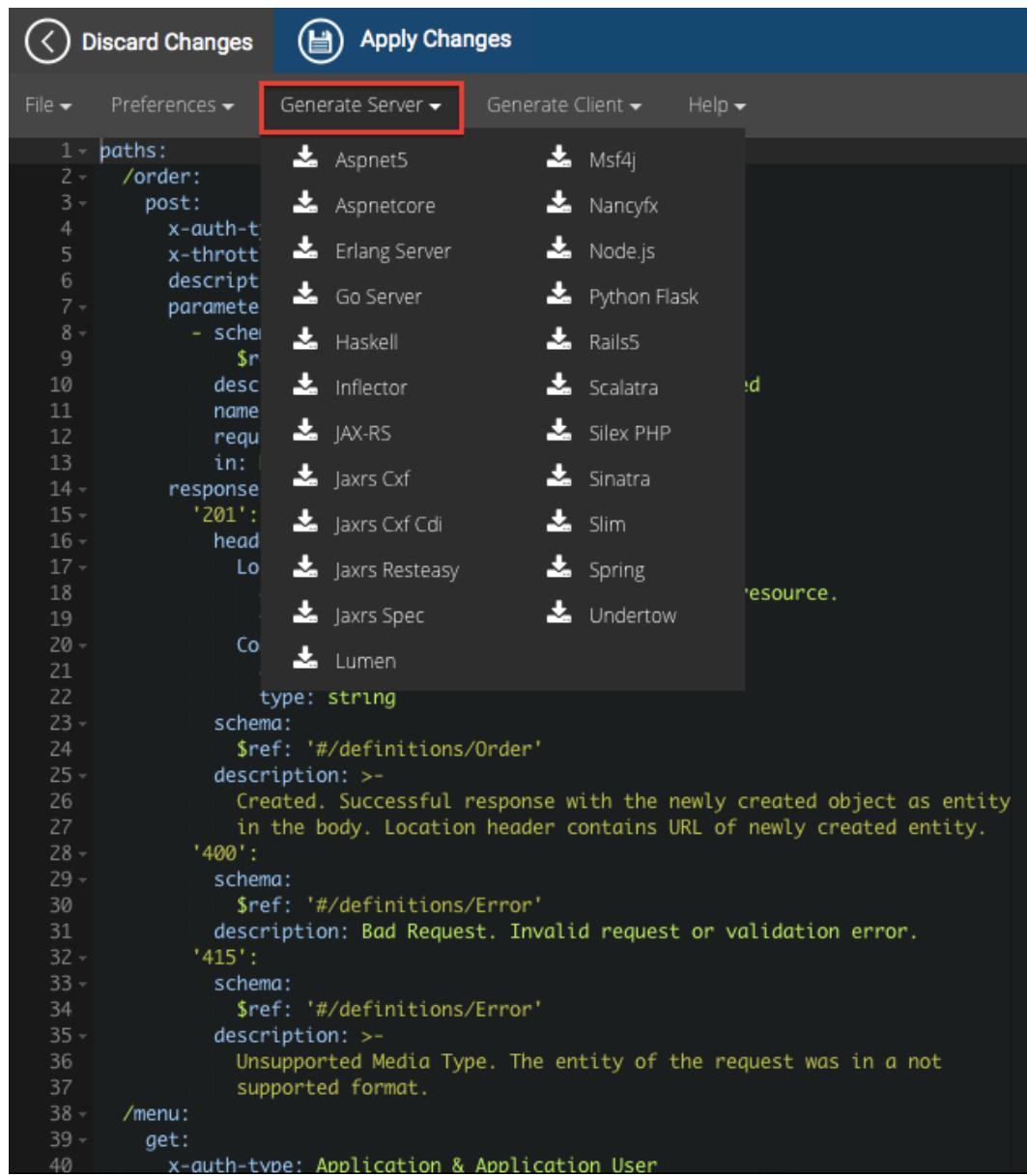
Edit

2. Click **Edit Source** to open the embedded swagger editor.

**API Definition**

URL Pattern	/pizzashack/1.0.0	Url Pattern E.g.: path/to/resource	<input type="button" value="Import"/>	<input type="button" value="Edit Source"/>
<input type="checkbox"/> GET <input type="checkbox"/> POST <input type="checkbox"/> PUT <input type="checkbox"/> DELETE <input type="checkbox"/> PATCH <input type="checkbox"/> HEAD <a href="#">more</a>				
<input type="button" value="⊕ Add"/>				
POST	/order	+ Summary	<input type="button" value=""/>	
GET	/menu	+ Summary	<input type="button" value=""/>	
PUT	/order/{orderId}	+ Summary	<input type="button" value=""/>	
GET	/order/{orderId}	+ Summary	<input type="button" value=""/>	
DELETE	/order/{orderId}	+ Summary	<input type="button" value=""/>	

3. To generate and download a server stub, click **Generate Server** and select a server stub from the list. API deveopers can use the **Generate Server** option to generate the REST API structure based on the swagger definition. The actual backend implementation can be developed on top of the code generated using swagger code generator. You can select from a list of frameworks to generate the actual backend implementation stub of the REST API.



The screenshot shows the WSO2 API Manager interface. At the top, there are two buttons: 'Discard Changes' and 'Apply Changes'. Below them is a navigation bar with 'File', 'Preferences', 'Generate Server' (which is highlighted with a red box), 'Generate Client', and 'Help'. The main area displays a JSON configuration file with code snippets for generating server-side code. The code includes definitions for paths, parameters, and response status codes (201, 400, 415). A list of client frameworks is shown on the right, each with a download icon:

- Aspnet5
- Msf4j
- Aspnetcore
- Nancyfx
- Erlang Server
- Node.js
- Go Server
- Python Flask
- Haskell
- Rails5
- Inflector
- Scalatra
- JAX-RS
- Silex PHP
- Jaxrs Cxf
- Sinatra
- Jaxrs Cxf Cdi
- Slim
- Jaxrs Resteasy
- Spring
- Jaxrs Spec
- Undertow
- Lumen

```

1 paths:
2 /order:
3 post:
4 x-auth-type: Application & Application User
5 x-throttling-level: 1
6 description: Create a new Order
7 parameters:
8 - schema:
9 type: object
10 properties:
11 name: {
12 type: string
13 description: The name of the Order
14 }
15 quantity: {
16 type: integer
17 description: The quantity of the Order
18 }
19 unitPrice: {
20 type: number
21 description: The unit price of the Order
22 }
23 type: string
24 schema:
25 $ref: '#/definitions/Order'
26 description: >-
27 Created. Successful response with the newly created object as entity
28 in the body. Location header contains URL of newly created entity.
29 '400':
30 schema:
31 $ref: '#/definitions/Error'
32 description: Bad Request. Invalid request or validation error.
33 '415':
34 schema:
35 $ref: '#/definitions/Error'
36 description: >-
37 Unsupported Media Type. The entity of the request was in a not
38 supported format.
39 /menu:
40 get:
41 x-auth-type: Application & Application User

```

- To generate and download a client SDK, click **Generate Client** and select a client from the list.

The screenshot shows the WSO2 API Manager interface with the 'Generate Client' dropdown menu open. The menu lists various programming languages and frameworks for generating client SDKs, including Akka Scala, Android, Async Scala, Clojure, C++, Java, Javascript, C#, C#.NET 2.0, Cwki, Dart, Dynamic HTML, Flash, Go, Groovy, HTML, Html2, Java, Javascript Closure Angular, Jaxrs Cxf Client, Jmeter, Objective-C, Perl, PHP, Python, Ruby, Scala, Swagger JSON, Swagger YAML, Swift, Swift3, Tizen, Typescript Angular, Typescript Angular2, Typescript Fetch, and Typescript Node. The 'order' endpoint is selected in the main panel, showing a POST method for creating a new Order.

The generated server stubs and client SDKs are generated using [Swagger Codegen](#).

It is recommended to add the `securityDefinitions` in the swagger definition to be able to pass access tokens when invoking an API. Edit the source of the API from the API Publisher and add the code given below.

```
securityDefinitions:
 default:
 type: oauth2
 authorizationUrl: 'https://<GW-HOST>:<GW-PORT>/authorize'
 flow: implicit
 scopes: {}
 security:
 - default: []
```

#### What's Next

[Write a Client Application Using the SDK](#)

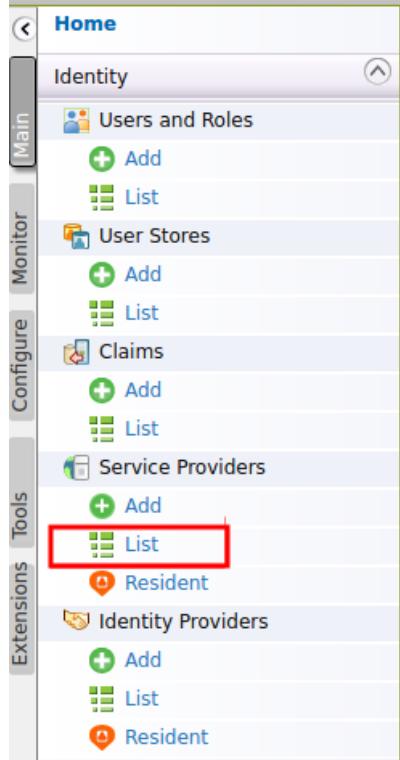
#### Revoke OAuth2 Application

1. An OAuth client is created when an application access token is generated. When a subscriber creates an application and generates an access token to the application using the API Store, the Store makes a call to

the API Gateway, which in turn connects with the Key Manager to create an OAuth client and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database.

You can revoke the access tokens issued for the application by following the instructions below

- Log in to the management console (<https://<HostName>:9443/carbon>)
- In the **Main** menu, click **Service Providers List**.



- Select the **Service Provider** and click **Edit**.

Service Providers		
Service Provider ID	Description	Actions
admin_DefaultApplication_PRODUCTION	Service Provider for application admin_DefaultApplication_PRODUCTION	

Only a user with the relevant permission to manage Service Providers will be able to view/edit the existing Service Providers. A user with only subscribe permission is not able to view or edit the Service Providers. The Admin user by default has permission only to view and edit Service Providers of Applications created by the same user. However, it's possible for the Admin user to assign the relevant permission for a particular Service Provider, to itself by navigating to **Users and Roles List Roles**, and then view/edit the Service Provider.

- Expand the **Inbound Authentication Configurations** section and select **OAuth/OpenID Configuration**

The screenshot shows the 'Service Providers' configuration page. Under 'Basic Information', the 'Service Provider Name' is set to 'admin\_DefaultApplication\_PRODUCTION'. The 'Description' field contains 'Service Provider for application admin\_DefaultApplication\_PRODUCTION'. In the 'SaaS Application' section, there is a note about SaaS applications being restricted by default. Below this, there are several expandable sections: 'Claim Configuration', 'Role/Permission Configuration', 'Inbound Authentication Configuration', 'SAML2 Web SSO Configuration', and 'OAuth/OpenID Connect Configuration'. The 'OAuth/OpenID Connect Configuration' section is currently selected and highlighted with a red box. It displays a table with one row, showing an 'OAuth Client Key' (p\_coFfj7U1Dl05kIr2kZAHVORia) and an 'OAuth Client Secret' (redacted). The 'Actions' column includes 'Show', 'Edit', 'Revoke', 'Regenerate Secret', and 'Delete' buttons. The 'Revoke' button is also highlighted with a red box.

- e. You can revoke/deactivate the OAuth application by clicking **Revoke**. This will revoke all the tokens given for the application.

This screenshot shows a table titled 'OAuth/OpenID Connect Configuration' with one row. The columns are 'OAuth Client Key' (p\_coFfj7U1Dl05kIr2kZAHVORia), 'OAuth Client Secret' (redacted), and 'Actions'. The 'Actions' column contains buttons for 'Edit', 'Revoke', 'Regenerate Secret', and 'Delete'. The 'Revoke' button is highlighted with a red box.

After an OAuth application is revoked, the consumer secret and all the generated access tokens and authorization codes will be invalid. You will not be able to regenerate access tokens in the API store or using the Token API.

To re-activate the application, the consumer secret must be regenerated as shown in the next step

- f. To regenerate the secret of the OAuth Application, click **Regenerate Secret**.

This screenshot shows a table titled 'OAuth/OpenID Connect Configuration' with one row. The columns are 'OAuth Client Key' (p\_coFfj7U1Dl05kIr2kZAHVORia), 'OAuth Client Secret' (redacted), and 'Actions'. The 'Actions' column contains buttons for 'Edit', 'Revoke', 'Regenerate Secret', and 'Delete'. The 'Regenerate Secret' button is highlighted with a red box.

You have to generate new access tokens and authorization codes for the OAuth application through the API Store or using the Token API after regenerating the consumer secret.

## Configuring Keystores in WSO2 API Manager

WSO2 products use asymmetric encryption by default for the purposes of authentication and data encryption. In asymmetric encryption, keystores (with key pairs and certificates) are created and stored for the product. It is possible to have multiple keystores so that the keys used for different use cases are kept unique. For more information about creating and configuring keystores, see [Using Asymmetric Encryption](#).

After you have [created a new keystore and updated the client-truststore.jks file](#), you must update a few configuration files in order to make the keystore work.

For instructions on the default carbon keystore configurations, see [Configuring Keystores in WSO2 Products](#) in the WSO2 Product Administration Guide.

The following common configurations should be completed before you begin the configurations given in this page.

- Configuring the primary keystore
- Configuring a keystore for SSL connections
- Configuring a keystore for java permissions
- Configuring keystores for WS-Security

Make sure you do the configurations below to configure a keystore in WSO2 API Manager.

- Configuring keystores for AMQP and MQTT transports
- Configuring keystores for Jaggery Apps SSO configuration
- Configuring keystores for security
- Configuring keystores for endpoints
- Configuring keystores for advanced transport handling

#### ***Configuring keystores for AMQP and MQTT transports***

To configure AMQP and MQTT transports, open the <API-M\_HOME>/repository/conf/broker.xml file. The values for the location and password parameters under keyStore and trustStore must be updated. The code below shows the default values.

```
<sslConnection enabled="true" port="8672">
 <keyStore>
 <location>repository/resources/security/wso2carbon.jks</location>
 <password>wso2carbon</password>
 </keyStore>
 <trustStore>

 <location>repository/resources/security/client-truststore.jks</location>
 <password>wso2carbon</password>
 </trustStore>
</sslConnection>
```

#### ***Configuring keystores for Jaggery Apps SSO configuration***

Open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json file. Update the values for keyStoreName and keyStorePassword as shown below.

```
"ssoConfiguration" : {
 "enabled" : "true",
 "issuer" : "API_PUBLISHER",
 "identityProviderURL" : "https://localhost:9444/samlsso",
 "keyStorePassword" : "wso2carbon",
 "identityAlias" : "wso2carbon",
 "responseSigningEnabled": "true",
 "assertionSigningEnabled": "true",
 "keyStoreName" : "wso2carbon.jks",
},
```

#### ***Configuring keystores for security***

Open the <API-M\_HOME>/repository/conf/identity/identity.xml file and update the values for Locati

on and Password under the KeyStore section. The default configurations are shown below.

```
<EntitlementSettings>
 <ThriftBasedEntitlementConfig>
 <EnableThriftService>false</EnableThriftService>
 <ReceivePort>${Ports.ThriftEntitlementReceivePort}</ReceivePort>
 <ClientTimeout>10000</ClientTimeout>
 <KeyStore>

 <Location>${carbon.home}/repository/resources/security/wso2carbon.jks</Location>
 <Password>wso2carbon</Password>
 </KeyStore>
 <ThriftHostName>${carbon.host}</ThriftHostName>
 </ThriftBasedEntitlementConfig>
</EntitlementSettings>
```

#### **Configuring keystores for endpoints**

Open the `<API-M_HOME>/repository/conf/identity/EndpointConfig.properties` file and update `client.keyStore` and `client.trustStore` with the location of the keystore and truststore respectively. The default configurations are shown below.

```
tenantListEnabled=false
hostname.verification.enabled=true
mutual.ssl.username=admin
client.keyStore=./repository/resources/security/wso2carbon.jks
Carbon.Security.KeyStore.Password=wso2carbon
client.trustStore=./repository/resources/security/client-truststore.jks
Carbon.Security.TrustStore.Password=wso2carbon
#identity.server.serviceURL=https://localhost:9443/services/
username.header=UserName
key.manager.type=SunX509
trust.manager.type=SunX509
tls.protocol=TLSv1.2
```

#### **Configuring keystores for advanced transport handling**

To have more advanced transport handling functions using keystores, you must update the `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` file and the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file.

#### **Logging**

Logging is one of the most important monitoring tools of a production server. A properly configured logging system is vital for identifying errors, security threats, and usage patterns. WSO2 API Manager uses a log4j based logging mechanism through the Apache Commons Logging facade library. The `log4j.properties` file, which governs how logging is performed by the server, is in the `API-M_HOME/repository/conf` directory. In general, you should not modify the `log4j.properties` file directly. Instead, you set up and modify logging using the ESB management console. The settings in the management console override the settings in the `log4j.properties` file.

A logger is used to log messages for a specific system or application component. Loggers are normally named using a hierarchical, dot-separated namespace and have a child-parent relationship. For example, the logger named `root.sv` is a parent of the logger named `root.sv.sf` and a child of `root`.

The following topics provide more information about logging:

- [Application Logs](#)
- [Setting Up Logging](#)
- [System Logs](#)

## Application Logs

Application logs are events that are recorded when they are invoked by an application or a program running in a system. Similarly, the application logs of a running Carbon instance display the log events of its deployed web applications and web services. The **Application Logs** page has been introduced as a fine-grained view of [system logs](#). While system logs display log events of the entire system holistically, the application logs page allows the user to select a particular application and view only those logs.

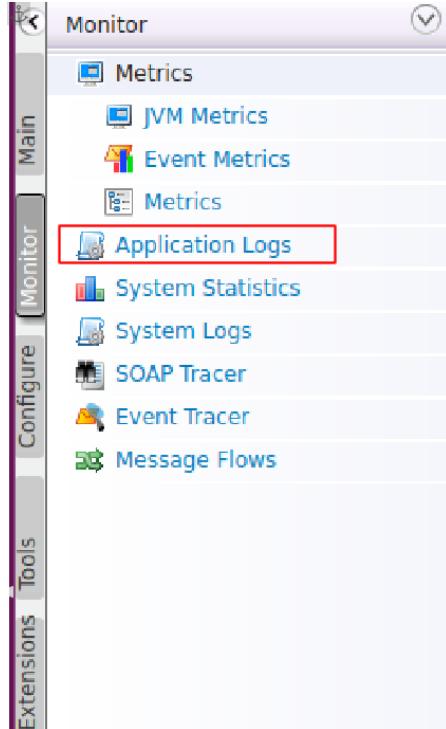
The application logs show logs from the proxy service, not from any logs inside its sequences. To view logs from the log mediator, see [System Logs](#).

The log files can be retrieved in two ways:

- If syslog-ng is configured, log files are taken from the remote location where the log files are hosted using the syslog-ng server.
- If syslog-ng is not configured, log files are taken from the local file system (super-tenant or Stand-alone apps).

For more information on logs and how to change log properties according to your preferences, see [Logging](#). Follow the instructions below to access statistics on application logs.

1. Log in to the product's Management Console and click **Monitor > Application Logs**.



2. The **Application Logs** page appears. This page displays logs of a selected application in a bottom-up manner. For example,

## Application Logs

View ALL echo Search Logs Search

Type	Date	Log Message	
	2012-08-20 12:39:19,757	Exception occurred while trying to invoke service method echoint	More
	2012-08-20 12:39:19,757	Exception occurred while trying to invoke service method echoint	More
	2012-08-20 11:27:16,069	Deploying Axis2 service: echo {super-tenant}	More

## Note

The log messages displayed on this page are obtained from a memory appender. Therefore, the severity (log level) of the displayed log messages is equal to or higher than the threshold of the memory appender. For more information on appenders, loggers, their log levels and logging, see [Apache Log4j 2](#) in the Apache documentation.

3. Use the drop-down list shown below to select a deployed web service or web application to view its log files.

## Application Logs

View ALL Search Logs Search

4. In the **View** list, select the category of logs you want to view. The available categories are:

- TRACE - Trace messages
- DEBUG - Debug messages
- INFO - Information messages
- WARN - Warning messages
- ERROR - Error messages
- FATAL - Fatal error messages
- ALL - Displays all categories of logs

For example,

## Application Logs

View echo Search Logs Search

Type	Date	Log Message	
	2012-08-20 12:39:19,757	Exception occurred while trying to invoke service method echoint	More
	2012-08-20 12:39:19,757	Exception occurred while trying to invoke service method echoint	More

5. You can also search for a specific log using the search function. Enter a keyword (or part of a keyword) and click **Search**. When a search criteria is given, the **View** field is displayed as **Custom**.

For example,

Type	Date	Log Message
	2012-08-17 12:20:42,102	Exception occurred while trying to invoke service method echoint <a href="#">More</a>
	2012-08-17 12:20:42,102	Exception occurred while trying to invoke service method echoint <a href="#">More</a>

## Note

The location of the log files on disk is specified in the `log4j.configuration` file.

## Setting Up Logging

There are two ways to configure log4j logging in WSO2 API Manager. You can manually edit the `log4j.properties` file, or configure logging through the management console. Configuration made through the management console can be persisted in the WSO2 registry so that it is available even after the server restarts. There is also an option to restore the original Log4j configuration from the `log4j.properties` file using the management console. However, if you modify the `log4j.properties` file and restart the server, the earlier log4j configuration persisted in the registry is overwritten.

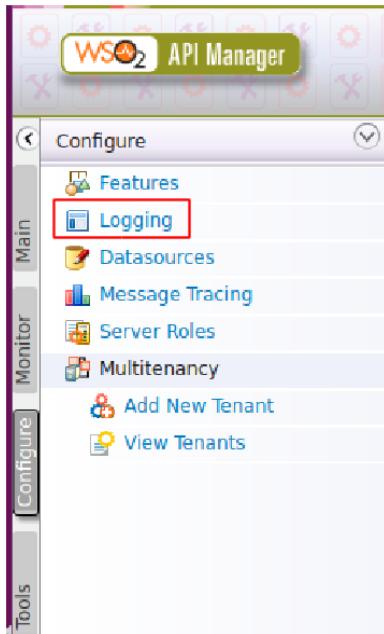
- Configure logging
  - Global Log4J configuration
  - Configure Log4J appenders
  - Configure Log4J loggers

For information on viewing the contents of the logs, see [Application Logs and System Logs](#).

### Configure logging

Follow the instructions below to set up logging.

1. Sign in to the [API-M Management Console](#).
2. Click **Configure > Logging**.



3. If you want your modifications to be persisted and available even after a server restart, select the **Persist All Configurations Changes** check box.

The screenshot shows the 'Logging Configuration' page. At the top, there is a link to 'Home > Configure > Logging' and a 'Help' link. Below the header, there is a checkbox labeled 'Persist All Configuration Changes' which is checked. The page is divided into three main sections: 'Global Log4J Configuration', 'Configure Log4J Appenders', and 'Configure Log4J Loggers'. The 'Global Log4J Configuration' section contains fields for 'Log Level' (set to 'ERROR') and 'Log Pattern' (set to '[%d{ISO8601}] %5p - %x %m (%c)%n'). The 'Configure Log4J Appenders' section shows a single appender named 'CARBON\_CONSOLE' with 'Name' set to 'CARBON\_CONSOLE', 'Log Pattern' set to '[%d{ISO8601}] %5p - %c(%1) %m%n', and 'Threshold' set to 'DEBUG'. The 'Configure Log4J Loggers' section lists several loggers: root, API\_LOGGER\_WSO2AMAuthorizeAPI\_, API\_LOGGER\_WSO2AMRevokeAPI\_, API\_LOGGER\_WSO2AMTokenAPI\_, API\_LOGGER\_WSO2AMUserInfoAPI\_, and API\_LOGGER.admin-MyAPI. Each logger is associated with a 'Parent Logger' of 'root' and has 'Level' set to 'ERROR' and 'Additivity' set to 'True'.

Logger	Parent Logger	Level	Additivity
root	-	ERROR	True
API_LOGGER_WSO2AMAuthorizeAPI_	root	ERROR	True
API_LOGGER_WSO2AMRevokeAPI_	root	ERROR	True
API_LOGGER_WSO2AMTokenAPI_	root	ERROR	True
API_LOGGER_WSO2AMUserInfoAPI_	root	ERROR	True
API_LOGGER.admin-MyAPI	root	ERROR	True

4. Use the options in the following sections to configure the layout and the amount of information about system activity that you want to record.

- Global Log4J configuration
- Configure Log4J appenders
- Configure Log4J loggers

#### ***Global Log4J configuration***

**Global Log4J Configuration**

Log Level	ERROR ▾
Log Pattern	[%d] %5p - %x %m [%c]%n

**Update** **Restore Defaults**

This section allows you to assign a single log level and log pattern to all loggers.

- **Log Level** - Reflects a minimum level that this logger cares about. You can view the hierarchy of levels [below](#)
- **Log Pattern** - Defines the output format of the log file.

### Configure Log4J appenders

**Configure Log4J Appenders**

Name	CARBON_CONSOLE ▾
Log Pattern	[%d{ISO8601}] %5p - %c{1} %m%n
Threshold	DEBUG ▾

**Update**

Log4j allows logging requests to print to multiple destinations. These output destinations are called appenders. You can attach several appenders to one logger.

- **Name** - The name of an appender. By default, WSO2 ESB comes with the following log appenders configured:
  - **CARBON\_CONSOLE** - Logs to the console when the server is running.
  - **CARBON\_LOGFILE** - Writes the logs to the <ESB\_HOME>/repository/logs/wso2-esb.log file.
  - **SERVICE\_APPENDER** - Writes mediation time audit messages to the <ESB\_HOME>/repository/logs/wso2-esb-service.log file.
  - **TRACE\_APPENDER** - Writes mediation time tracing/debug messages to the <ESB\_HOME>/repository/logs/wso2-esb-trace.log file for tracing enabled services.
  - **TRACE\_MEMORYAPPENDER**
  - **CARBON\_MEMORY**
  - **CARBON\_SYS\_LOG** - Allows separation of the software that generates messages from the system that stores them and the software that reports and analyzes them.
- **Log pattern** - Defines the output format of the log file.
- **Threshold** - Filters log entries based on their level. For example, if the threshold is set to WARN, log entries are allowed to pass into the appender if its level is WARN, ERROR or FATAL, while other entries are discarded.

#### Hierarchy of levels

- **TRACE** - Designates informational events that are more fine-grained than DEBUG.
- **DEBUG** - Designates fine-grained informational events that are most useful to debug an application.
- **INFO** - Designates informational messages that highlight the progress of the application at coarse-grained level.
- **WARN** - Designates potentially harmful situations.
- **ERROR** - Designates error events that might still allow the application to continue running.
- **FATAL** - Designates very severe error events that will presumably lead the application to abort.

### Configure Log4J loggers

Configure Log4J Loggers				
Filter Loggers by		Starts With	Contains	
Logger	Parent Logger	Level	Additivity	
root	-	ERROR ▾	True ▾	
API_LOGGER_WSO2AMAuthorizeAPI_	root	ERROR ▾	True ▾	
API_LOGGER_WSO2AMRevokeAPI_	root	ERROR ▾	True ▾	
API_LOGGER_WSO2AMTokenAPI_	root	ERROR ▾	True ▾	
API_LOGGER_WSO2AMUserInfoAPI_	root	ERROR ▾	True ▾	
API_LOGGER_admin--MyAPI	root	ERROR ▾	True ▾	

This section allows you to browse through all loggers, define a log level, and switch on/off additivity to any of them. You can filter loggers using the first few characters (use the **Starts With** button) or using a combination of characters (use the **Contains** button).

- **Logger** - The name of a logger.
- **Parent Logger** - The name of a parent logger.
- **Level** - Allows to select the level (threshold) from the drop-down menu. After you specify the level for a certain logger, a log request for that logger is enabled only if its level is equal or higher to that of the logger's. If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. See [hierarchy of levels](#) above.
- **Additivity** - Allows to inherit all the appenders of the parent Logger if set to True.

#### Example

Use the following procedure to enable logs to view HTTP headers and messages:

1. In the **Filter Loggers by** field, enter **wire** and then click **Contains**.  
You see `org.apache.synapse.transport.http.wire` displayed under **Logger**.

Configure Log4J Loggers				
Filter Loggers by		Starts With	Contains	
Logger	Parent Logger	Level	Additivity	
httpclient.wire.content	root	ERROR ▾	True ▾	
httpclient.wire.header	root	DEBUG ▾	True ▾	
org.apache.http.wire	root	DEBUG ▾	True ▾	
org.apache.synapse.transport.http.wire	org.apache.synapse.transport	INFO ▾	True ▾	
org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor	root	ERROR ▾	True ▾	

2. Change the level of this logger to **DEBUG**.
3. Search for the `org.apache.synapse.transport.http.headers` logger and change the level to **DEBU G**.

Configure Log4J Loggers				
Filter Loggers by		Starts With	Contains	
Logger	Parent Logger	Level	Additivity	
org.apache.synapse.transport.http.headers	org.apache.synapse.transport	INFO ▾	True ▾	

Alternatively, you can uncomment the entry for the two loggers as follows:

1. Go to the `<ESB_Home>/repository/conf` directory and open the `log4j.properties` file with a text editor.
2. Edit the entries for the two loggers as follows by removing the commented (#).  
`log4j.logger.org.apache.synapse.transport.http.headers=DEBUG`  
`log4j.logger.org.apache.synapse.transport.http.wire=DEBUG`
3. Save the changes.

## System Logs

The **System Logs** page displays information about the log files of the current product. The log files can be retrieved in two ways:

- If syslog-ng is configured, log files are taken from the remote location where the log files are hosted using the syslog-ng server.
- If syslog-ng is not configured, log files are taken from the local file system (super-tenant or stand-alone apps).

This page contains the following sections:

- [Viewing system logs](#)
- [Displaying log mediator logs](#)

For more information on logs, see [Logging](#).

### **Viewing system logs**

To view system logs, click **System Logs** on the **Monitor** tab in the Management Console. The log messages displayed on this page are obtained from a memory appender. Therefore, the severity (log level) of the displayed log messages is equal to or higher than the threshold of the memory appender.

- **File Name** - The name of the file containing logs pertaining to a certain period.
- **Date** - The date at which the log file was generated.
- **File Size** - The size of the file in bytes.
- **Action** - Allows to view and download files.

**System Logs**

Logs are taken from the local file system.

File Name	Date	File Size	Action
wso2carbon.log	Current Log	1.0 MB	<a href="#"></a> <a href="#"></a>
wso2carbon.log.2011-07-22	2011-07-22	44.4 KB	<a href="#"></a> <a href="#"></a>
wso2carbon.log.2011-07-25	2011-07-25	232.9 KB	<a href="#"></a> <a href="#"></a>
wso2carbon.log.2011-07-26	2011-07-26	295.9 KB	<a href="#"></a> <a href="#"></a>

Now, you can download an individual log file, or view it in the Management Console. When viewing a log file, you can choose a category such as **ERROR** in the **View** list to filter the messages by that category. You can also enter a search term to find a specific log message, and you can use the **Log Head** field to specify how many log lines to display.

**System Log View**

View: ALL [▼](#) Search Logs  [Search](#) Log Head   [Search](#)

1 2 3 4 5 next >

Type	Log Message
	TID: [] [WSO2 ESB] [2011-07-27 16:38:35,546] ERROR {org.apache.synapse.startup.tasks.MessageInjector} - message not set {org.apache.synapse.startup.tasks.MessageInjector}
	TID: [] [WSO2 ESB] [2011-07-27 16:38:30,546] ERROR {org.apache.synapse.startup.tasks.MessageInjector} - message not set {org.apache.synapse.startup.tasks.MessageInjector}
	TID: [] [WSO2 ESB] [2011-07-27 16:38:25,546] ERROR {org.apache.synapse.startup.tasks.MessageInjector} - message not set {org.apache.synapse.startup.tasks.MessageInjector}
	TID: [] [WSO2 ESB] [2011-07-27 16:38:20,546] ERROR {org.apache.synapse.startup.tasks.MessageInjector} - message not set {org.apache.synapse.startup.tasks.MessageInjector}
	TID: [] [WSO2 ESB] [2011-07-27 16:38:15,546] ERROR {org.apache.synapse.startup.tasks.MessageInjector} - message not set {org.apache.synapse.startup.tasks.MessageInjector}

The location of the log files on disk is specified in the `log4j.configuration` file.

### **Displaying log mediator logs**

When you use the **Log mediator** inside sequences and proxy services in WSO2 API Manager, the logs are stored in the `<PRODUCT_HOME>/repository/logs/wso2carbon.log` file by default. To view these logs from the **System Logs** page, open the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file and uncomment the

following element:

```
<handler class="org.wso2.carbon.utils.logging.handler.TenantDomainSetter"
name="TenantDomainSetter">
```

When you restart the server, the logs will be available on the **System Logs** page.

## Message Tracing

Message Tracing is an identification of each message flow in each transaction which goes through the gateway. WSO2 API Manager provides the facility to do message tracing by installing the Message Tracer feature developed for WSO2 Products. It is used to logging, auditing and debugging message content and its direction.

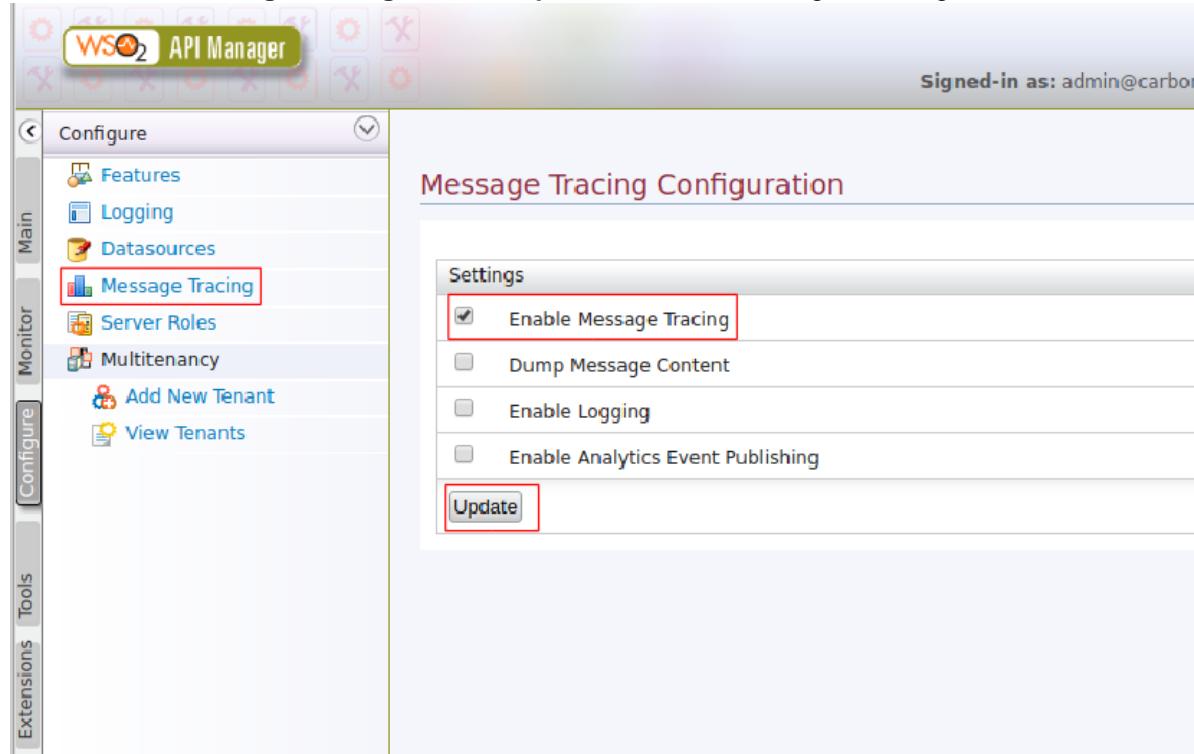
Message Tracer is a part of carbon-analytics. From API Manager 2.1.0 onwards, Message Tracer feature is build into API Manager.

- Configuring Message Tracer
- Tracing Events Publishing to API Manager Analytics
- Results

### Configuring Message Tracer

Follow the below steps to configure Message Tracer in API Manager.

1. Log into Management Console (<https://localhost:9443/carbon>).
2. Navigate to **Configure > Message Tracing** in **Configure Tab** to open the **Message Tracing Configuration**.
3. Select **Enable Message Tracing** and click **Update** to enable Message Tracking.



4. Select following options as well if you wish to get the message tracing in detail.

Configuration	Description
Dump Message Content	Enable Message tracing for the Content of the Message as well.
Enable Logging	Enable Logging into available logging handler to log the tracing message.
Enable Analytics Event Publishing	Publish tracing events to API Manager Analytics

### Tracing Events Publishing to API Manager Analytics

You can trace the messages of event publishing to API Manager Analytics with Message Tracing feature following below steps.

Before you begin,

Make sure you have configured API Manager Analytics. Refer [Configuring APIM Analytics](#) to configure Analytics.

1. Select Enable Analytics Event Publishing in Message Tracer Configuration and click update.

The screenshot shows the WSO2 API Manager configuration interface. On the left, there's a sidebar with tabs: Main, Monitor, and Configure. Under 'Configure', the 'Message Tracing' option is selected and highlighted with a red border. The main panel is titled 'Message Tracing Configuration'. It contains a 'Settings' section with four checkboxes: 'Enable Message Tracing' (checked), 'Dump Message Content' (checked), 'Enable Logging' (checked), and 'Enable Analytics Event Publishing' (checked and highlighted with a red border). Below the settings is a 'Update' button.

Once you have saved the changes, APIM will generate stream definition in <APIIM\_HOME>/repository/deployment/server/eventstreams/**DAS\_MESSAGE\_TRACE\_1.0.0.json**.

2. Copy the DAS\_MESSAGE\_TRACE\_1.0.0.json file to <APIM\_ANALYTICS\_HOME>/repository/deployment/server/eventstreams directory to deploy the same definition in API Manager Analytics.
3. Login to API Manager Analytics (<https://localhost:9444/carbon>) and navigate to **Main > Event > Streams**.

You will be able to see the deployed stream under event streams. Now we need to persist the event stream to save data into a table.

4. Click **Edit** in the stream definition DAS\_MESSAGE\_TRACE:1.0.0 and open the edit view.
5. Click **Next[Persist Event]** at the bottom of the edit view.
6. In the next page select **Persist Event Stream** and select all attribute checkboxes to persist and click on **Save Event Stream** button.

You should configure an Event receiver to point thid Event stream.

7. Go to **Main -> Event -> Receivers** in API Manager Analytics and click on **Add Event Receiver**.

The screenshot shows the 'Available Event Receivers' page in the WSO2 API Manager Analytics interface. The left sidebar includes sections for Home, Dashboard, Template Manager, Analytics Dashboard, Manage, Batch Analytics, Streaming Analytics, Execution Plans, Event, Interactive Analytics, Data Explorer, Activity Explorer, Carbon Applications, Registry, and Tools. The main content area displays a table of 13 active event receivers, each with columns for Name, Message Format, Input Event Adapter Type, Input Stream ID, and Actions. A red box highlights the 'Add Event Receiver' button at the top of the list.

**8. Add following details and click on **Add Event Receiver**.**

property	Value
Event Receiver Name	message_trace_receiver
Input Event Adopter Type	wso2event
Is events duplicated in cluster	false
Event Stream	DAS_MESSAGE_TRACE:1.0.0
Message format	wso2event

The screenshot shows the 'Create a New Event Receiver' form. It includes fields for 'Event Receiver Name' (set to 'message\_trace\_receiver'), 'Input Event Adopter Type' (set to 'wso2event'), 'Is events duplicated in cluster' (set to 'false'), 'Event Stream' (set to 'DAS\_MESSAGE\_TRACE:1.0.0'), and 'Message format' (set to 'wso2event'). The form also contains sections for 'From', 'Usage Tips', 'Adapter Properties', 'To', 'Mapping Configuration', and a summary section at the bottom with the 'Add Event Receiver' button highlighted by a red box.

Next, a publisher need to be configured in order to publish events to API Manager Analytics. To do that,  
**9. Go to Main > Event > Publishers in API Manager Analytics and click Add Event Publisher.**

The screenshot shows the WSO2 API Manager Management Console interface. The left sidebar has sections for Home, Dashboard, Template Manager, Analytics Dashboard, Manage (Batch Analytics, Scripts, Console, Streaming Analytics, Execution Plans, Event, Flow, Streams, Receivers, Publishers), Interactive Analytics (Data Explorer, Activity Explorer), Carbon Applications (List, Add), and Shutdown/Restart. The right panel is titled 'Available Event Publishers' and shows a table with one active event publisher: 'org.wso2.analytics.apim.email.AlertStreamPublisher'. The 'Actions' column for this row contains a green icon with a checkmark and the text 'Event Publisher deployed via CApp'. A red box highlights the 'Add Event Publisher' button at the top of the list.

10. In Create a New Event Publisher page, add the following details and click **Add Event Publisher**.

Property	Value
Event Publisher Name	message_tracer_publisher
Event Source	DAS_MESSAGE_TRACE:1.0.0
Stream Attributes	Keep default values
Output Event Adaptor Type	wso2event
Receiver URL	tcp://localhost:7612
User Name	admin
Password	admin
Protocol	thrift
Publishing mode	non-blocking
Message Format	wso2event

Create a New Event Publisher

Enter Event Publisher Details	
Event Publisher Name*	message_trace_publisher Enter a unique name to identify Event Publisher
<b>From</b>	
Event Source*	DAS_MESSAGE_TRACE-1.0.0 The stream of events that need to be published
Stream Attributes	meta_request_url string, meta_host string, meta_server string, correlation_activity_id string, service_name string, operation_name string, message_direction string, soap_body string, soap_header string, timestampo_lona.
<b>To</b>	
Output Event Adapter Type*	wso2event Select the type of Adapter to publish events
Following url formats are used to publish events For load-balancing: use "," to separate values for multiple endpoints Eg: {tcp://<hostname>:<port>},{tcp://<hostname>:<port>}, ...}	
Usage Tips	For failover: use "!" to separate multiple endpoints Eg: {tcp://<hostname>:<port>}!{tcp://<hostname>:<port>}/ ...
For more than one cluster: use "{}" to separate multiple clusters Eg: {tcp://<hostname>:<port>}/{tcp://<hostname>:<port>}/{tcp://<hostname>:<port>}	
<b>Static Adapter Properties</b>	
Receiver URL	tcp://localhost:7612 Enter the Receiver Url
Authenticator URL	 Enter the Authenticator Url
User Name*	admin Enter the UserName
Password*	***** Enter the Password
Protocol*	http The communication protocol that will be used to published events
Publishing Mode	non-blocking Select how events should be published
Publishing Timeout	0 Timeout for the non-blocking Publishing Mode. Default is '0' (fail immediately)
<b>Mapping Configuration</b>	
Message Format*	wso2event Select the output message format
<input checked="" type="button"/> Advanced	
<input type="button"/> Add Event Publisher <input type="button"/> Test Event Publisher	

## Results

After enabling Message tracing, Dump Message Contnet and Logging, you can see the log Message similar to this from API Console for events like API invocation etc.

```
[2017-01-11 11:43:08,039] INFO - HandlerUtils Message Info: Transaction id=632470820207822377292 Message direction=IN Server name=172.17.0.1:9763 Timestamp=1515651188039 Service name=__SynapseService Operation Name=mediate
[2017-01-11 11:43:08,040] INFO - HandlerUtils Message Info: Transaction id=632470820207822377292 Message direction=OUT Server name=172.17.0.1:9763 Timestamp=1515651188040 Service name=__SynapseService Operation Name=mediate
[2017-01-11 11:43:08,047] INFO - HandlerUtils Message Info: Transaction id=632471565977965101628 Message direction=IN Server name=172.17.0.1:9763 Timestamp=1515651188047 Service name=__SynapseService Operation Name=mediate
[2017-01-11 11:43:08,060] INFO - HandlerUtils Message Info: Transaction id=632471565977965101628 Message direction=OUT Server name=172.17.0.1:9763 Timestamp=1515651188058 Service name=__SynapseService Operation Name=mediate
```

If you have enabled Analytics event Publishing,

1. Login to WSO2 API Manager Analytics (<https://localhost:9444/carbon>)
2. Navigate to **Main > Data Explorer**.
3. Select the table **DAS\_MESSAGE\_TRACE** and click **Search**.

You can see the event data traced in WSO2 Analytics Data Explorer as below.

The screenshot shows the WSO2 API Manager Data Explorer interface. The top navigation bar includes 'Home', 'Manage', 'Interactive Analytics', and 'Data Explorer'. The main area has a 'Search' field and a table configuration section with 'Table Name\*' set to 'DAS\_MESSAGE\_TRACE', 'Maximum Result Count' set to '1000', and search options by date range or query. Below this is a 'Results' section with a note: 'Note: Total record count for the table is not available.' A table displays data from the 'DAS\_MESSAGE\_TRACE' table, with columns: meta\_request\_url, meta\_host, meta\_server, correlation\_activity\_id, service\_name, operation\_name, message\_direction, soap\_body, soap\_header, and timestamp. The data shows several rows of trace logs, each with a timestamp around 14656201. At the bottom, there are navigation links for page numbers and a row count selector.

DAS_MESSAGE_TRACE									
meta_request_url	meta_host	meta_server	correlation_activity_id	service_name	operation_name	message_direction	soap_body	soap_header	timestamp
/sample/1.0.0	172.16.2.136:9763	WSO2 API Manager	161629642707392048231	__SynapseService	mediate	OUT	<soapenv:Body xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"/>	<soapenv:Header xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"/>	14656201
/sample/1.0.0	172.16.2.136:9763	WSO2 API Manager	161629642707392048231	__SynapseService	mediate	IN	<soapenv:Body xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"/>	<soapenv:Header xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"/>	14656201
/sample/1.0.0	172.16.2.136:9763	WSO2 API Manager	1616297111733129600948	__SynapseService	mediate	IN	<soapenv:Body xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"/>	<soapenv:Header xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"/>	14656201
	172.16.2.136:9763	WSO2 API Manager	1616297111733129600948	__SynapseService	mediate	OUT	<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><jsonObject><s>Got it!</s></...>	<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope"/>	14656201

## Whitelisting Host Names for API Store

API Manager provides the capability to whitelist multiple host names if you use different host names to access API Store in your environment.

In this case, **localhost** is by default considered as a whitelisted host name.

Similarly you can whitelist multiple host names for store as follows.

- You need to add the host names to the **whiteListedHostNames** attribute in `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` as comma separated values.

See the following example configuration.

```
"whiteListedHostNames": ["www.wso2.org", "www.example.com"]
```

## Note :

When you try to access API Store with a host which is not whitelisted, or is not specified in `<APIM_HOME>/repository/conf.carbon.xml`, you will notice the following warning being logged in the server logs.

Possible HOST Header Attack is identified. Hence, rewriting to default host in configuration.

## Extending the API Manager

This page summarizes the extension points supported by WSO2 API Manager.

[ [Mediation extensions](#) ] [ [Security extensions](#) ] [ [Branding extensions](#) ] [ [Workflow extensions](#) ] [ [API life cycle extensions](#) ]

### Mediation extensions

- [Writing Custom Handlers](#)
- [Adding Mediation Extensions](#)
- [Customized Error Handling](#)
- [Editing API Templates](#)

### Security extensions

- [Writing Custom Grant Types](#)
- [Customizing the JWT Generation](#)
- [Configuring a Third-Party Key Manager](#)
- [Extending Key Validation](#)
- [Extending the Key Manager Interface](#)
- [Extending Scope Validation](#)
- [Engaging Multiple Throttling Policies to a Single API](#)
- [Sharing Applications and Subscriptions](#)
- [Securing OAuth Token with HMAC Validation](#)

### Branding extensions

- [Adding a new or customizing an existing API Store Theme](#)
- [Customize the API Store and Gateway URLs for Tenants](#)
- [Customizing Login Pages for API Store and API Publisher](#)
- [Customizing User SignUp in API Store](#)

### Workflow extensions

- [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](#)

### API life cycle extensions

- [Extending the API Life Cycle](#)

## Managing Workflow Extensions

The workflow feature enables you to add more control and constraints to the tasks executed within it. For example, you can add a constraint through a workflow where an approval from a manager is required for a user to sign-up to the API Store. You can engage a workflow if you require a third party interventions/approvals for actions which have a pre-defined path such as signing up, registering an application, subscribing to an API, etc.

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
- Cleaning Up Workflow Tasks
- Adding an API State Change Workflow

### Adding an Application Creation Workflow

This section explains how to attach a custom workflow to the application creation operation in WSO2 API Manager (WSO2 API-M). First, see [Workflow Extensions](#) for information on different types of workflow executors.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Change the credentials of the workflow configurations in the registry resource `_system/governance/apimgt/applicationdata/workflow-extensions.xml`.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Unzip the `<API-M>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip` file, update the role as follows in the `ApplicationsApprovalTask.ht` file, and ZIP the `ApplicationsApprovalTask-1.0.0` folder.

#### Format

```
<htd:argument name="role">
 [new-role-name]
</htd:argument>
```

### 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. For more information, 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 WSO2 API-M and WSO2**

**BPS on different machines** (therefore, want to set the hostname to a different value than localhost), you need to search and replace the value 9765 in all the files (.epr) inside the <APIM\_HOME>/business-processes directory with the new port (i.e., the value of 9763 + <port-offset>).

- Open the <BPS\_HOME>/repository/conf/humantask.xml file and <BPS\_HOME>/repository/conf/b4p-coordination-config.xml file and set the TaskCoordinationEnabled property to true.

```
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```

- Copy the following from the <API-M\_HOME>/business-processes/epr directory to the <BPS\_HOME>/repository/conf/epr directory.  
If the <BPS\_HOME>/repository/conf/epr directory does not exist, create it.

Make sure to give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.

- ApplicationService.epr
- ApplicationCallbackService.epr

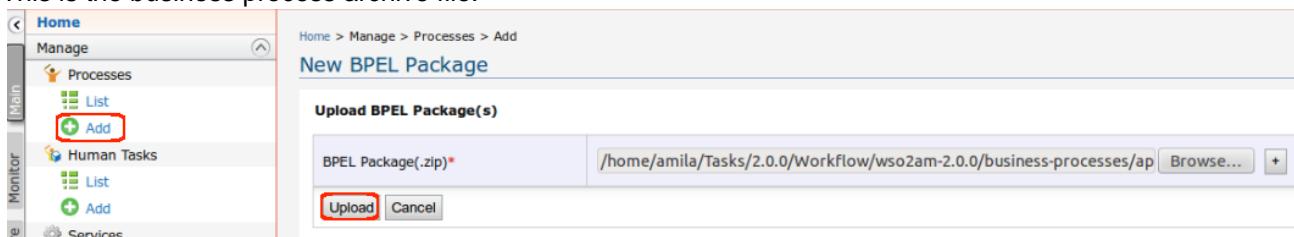
- Start the BPS server and sign in to the management console (<https://<Server Host>:9443+<port-offset>/carbon>).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential
cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST,
uri:/carbon/admin/login_action.jsp, error:required token is
missing from the request)
```

To avoid this issue open <BPS\_HOME>/repository/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the BPS.

- Select **Processes > Add** 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.



- Select **Add** under the **Human Tasks** menu and upload the <APIM\_HOME>/business-processes/application-creation/HumanTask/ApplicationsApprovalTask-1.0.0.zip file to BPS.  
This is the human task archived file.

### Configuring WSO2 API Manager

Open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json file and configure "workFlowServerURL" under "workflows" to point to the BPS server (e.g., "workFlowSer

```
verURL": "https://localhost:9445/services/"
```

### **Engaging the WS Workflow Executor in the API Manager**

First, enable the application creation workflow.

1. Sign in to WSO2 API-M management console (<https://<Server-Host>:9443/carbon>) and select **Browse** under **Resources**.



2. Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor, and enable WS Workflow Executor. In addition, 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.ApplicationCreationWSWorkflowExecutor">
 <Property
name="serviceEndpoint">http://localhost:9765/services/ApplicationApprovalWorkflowProcess/</Property>
 <Property name="username">admin</Property>
 <Property name="password">admin</Property>
 <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
 </ApplicationCreation>
...
</WorkFlowExtensions>
```

All the workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

The application creation WS Workflow Executor is now engaged.

3. Go to the API Store, click **Applications** 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. Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.
4. Sign in to the Admin Portal (<https://localhost:9443/admin>), list all the tasks for application creation and approve the task. It resumes the BPEL process and completes the application creation.
5. Go back to the **Applications** page in WSO2 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:

```

<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 engine sends the workflow-completion request to this URL. This property is configured in the <callBackURL> element in the api-manager.xml file.

## 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 Application registration (Key generation) workflow**

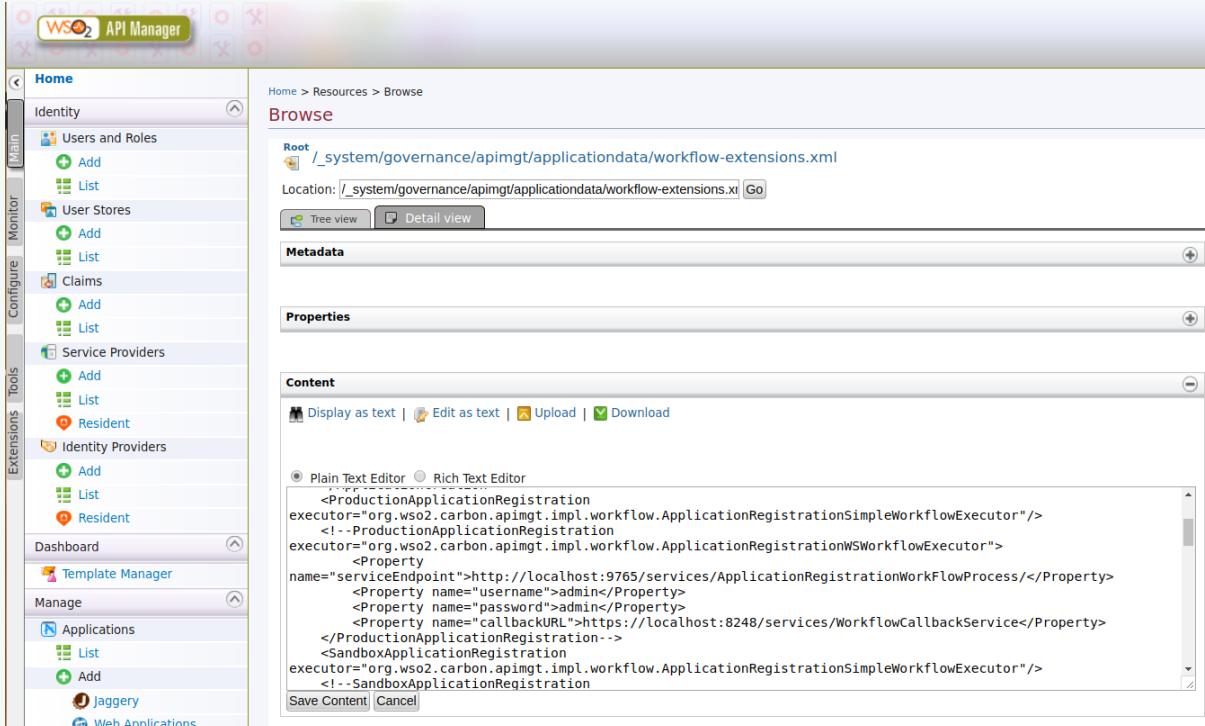
**Application creation** and **Application registration** are different workflows. After an application is created, you can subscribe to available APIs, but you get the consumer key/secret and access tokens only after registering the application. There are two types of registrations that can be done 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.

To make API subscribers go through an approval process before creating any type of access token.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

1. Change the credentials of the workflow configurations in the registry resource \_system/governance/apimgt/applicationdata/workflow-extensions.xml.
  - a. Login to the Management console of WSO2 API Manager in <https://localhost:9443/carbon>.
  - b. Click on browse under Resources in left Navigation under Main tab.
  - c. Go to /\_system/governance/apimgt/applicationdata/workflow-extensions.xml location in registry browser and open the workflow-extensions.xml clicking **Edit as text**.



- d. Uncomment the following two sections and change the credentials of API Manager's default user credentials you have given.

This configuration is provided assuming that WSO2 BPS is running with offset 2. If you are running WSO2 BPS in a different offset change the port of **serviceEndpoint** properties in following configuration according to the changed port offset.

```
<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.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>
```

Make sure to comment out the existing ProductionApplicationRegistration and SandboxApplicationRegistration executors as shown below.

```
<!--ProductionApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationSimpleWorkflowExecutor"-->
<!--SandboxApplicationRegistration
executor="org.wso2.carbon.apimgt.impl.workflow.ApplicationRegistrationSimpleWorkflowExecutor"-->
```

2. Point the database that has the API Manager user permissions to BPS.  
In this step you need to share the user store database in WSO2 API Manager with WSO2 BPS.

a. Copy the following datasource configuration in `<API-M_HOME>/repository/conf/datasources/master-datasources.xml`:

```

<datasource> <name>WSO2UM_DB</name>
 <description>The datasource used by user manager</description>
 <jndiConfig>
 <name>jdbc/WSO2UM_DB</name>
 </jndiConfig>
 <definition type="RDBMS">
 <configuration>

 <url>jdbc:mysql://userdb.mysql-wso2.com:3306/userdb?autoReconnect=true</url>
 <username>user</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>

```

We are using MySQL to configure the datasources in this documentation. You can configure this according to the database you are using. Refer [Setting up the Physical Database](#) for more information.

- b. Change the datasource to point the WSO2UM\_DB by changing the realm configuration in <API-M\_HOME>/repository/conf/user-mgt.xml as shown below.

```

<UserManager>
 <Realm>
 <Configuration>

 <Property
 name="dataSource">jdbc/WSO2UM_DB</Property>
 </Configuration>

 <Realm>
<UserManager>

```

- c. Do the configuration described in a and b in <BPS\_HOME>/repository/conf/datasources/master-datasources.xml and <BPS\_HOME>/repository/conf/user-mgt.xml respectively.
3. Share any LDAPs, if exist.
  4. Unzip the <API-M>/business-processes/application-registration/HumanTask/ApplicationRegistrationTask-1.0.0.zip file, update the role as follows in the ApplicationRegistrationTask.ht file, and ZIP the ApplicationRegistrationTask-1.0.0 folder.

**Format**

```
<htd:argument name="role">
 [new-role-name]
</htd:argument>
```

5. Restart the API Manager server.

**Configuring the Business Process Server**

1. Download [WSO2 Business Process Server](#).
2. Set an offset of 2 to the default BPS port in the <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. For more information, 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) inside <APIM\_HOME>/business-processes folder with the new port (9763 + port offset.)

3. Open the <BPS\_HOME>/repository/conf/humantask.xml file and the <BPS\_HOME>/repository/conf/b4p-coordination-config.xml file and set the TaskCoordinationEnabled property to true.

```
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```

4. Copy the following from the <API-M\_HOME>/business-processes/epr folder to the <BPS\_HOME>/repository/conf/epr folder. If the <BPS\_HOME>/repository/conf/epr folder does not exist, create it.

Make sure you give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.

- RegistrationService.epr
  - RegistrationCallbackService.epr
5. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential
cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST,
uri:/carbon/admin/login_action.jsp, error:required token is
missing from the request)
```

To avoid this issue open <BPS\_HOME>/repository/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the BPS.

6. Log into Management console of WSO2 BPS, select **Add > BPEL** under the **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.



7. Select **Add** under the **Human Tasks** menu and upload the <APIM\_HOME>/business-processes/application-registration/HumanTask/ApplicationRegistrationTask-1.0.0.zip file to BPS. This is the human task archived file.



## Configuring the API Manager

Open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json file and configure "workFlowServerURL" under "workflows" to point to the BPS server (e.g. "workFlowServerURL": "https://localhost:9445/services/")

```
{

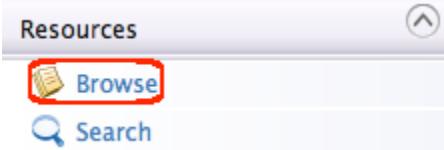
 "context": "/admin",
 "request_url": "READ_FROM_REQUEST",
 "tasksPerPage": 10,
 "allowedPermission": "/permission/admin/manage/apim_admin",
 "workflows": {
 "workFlowServerURL": "https://localhost:9445/services/",
 }

}
```

## Engaging the WS Workflow Executor in the API Manager

First, enable the application registration workflow.

- Start WSO2 API Manager and login to the APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



- Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable WS Workflow Executor as described in the tip provided at the start of this documentation if you haven't done already.

```
<WorkFlowExtensions>
...
<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.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 because you changed its default port (9763) with an offset of 2.

- Log into the API Store (<https://localhost:9443/store>) as a Store user and open the application with which you subscribed to the API.

If you do not have an already created API and an Application subscribed to it, follow [Create and](#)

Publish an API and upto step 8 of Subscribe to an API to create an API and subscribe to it.

- In the **Production Keys** tab of the Application, click the **Generate Keys** button. It invokes the `ApplicationRegistrationWorkFlowProcess.bpel` that is bundled with the `ApplicationRegistrationWorkflowProcess_1.0.0.zip` file and creates a `HumanTask` instance that holds the execution of the BPEL process until some action is performed on it.

- Note that a message appears saying that the request is successfully submitted if the BPEL was invoked correctly.
- Log in to the Admin Portal (`https://<Server Host>:9443/admin`) with admin credentials and list all the tasks for application registrations. Click **Start** to start the Human Task and then change its state. Once you select **Approve** and click **Complete** the task, it resumes the BPEL process and completes the registration.

### Approval Tasks

ID	Description	Status	Created On	Action
651	Approve request to create PRODUCTION Keys for [ App1 ] from application creator - chamalee with throttling tier - Unlimited	IN_PROGRESS	2017-12-12 - 15:24:29.940+05:30	<button>Approve</button> <span style="border: 2px solid red; padding: 2px;">Complete</span>

- Go back to the API Store and view your application.

It shows the application access token, consumer key and consumer secret.

After the registration request is approved, keys are generated by invoking the `APIKeyMgtSubscriber` service hosted in Key Manger 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. When a consumer subscribes to an API, he/she has to subscribe to an application in order to get access to the API. However, when API subscription workflow is enabled, when the consumer subscribes to an application, it initially is in the `On Hold` state, and he/she can not use the API, using its production or sandbox keys, until their subscription is approved.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Unzip the `<API-M>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip` file, update the role as follows in the `SubscriptionsApprovalTask.ht` file, and ZIP the `SubscriptionsApprovalTask-1.0.0` folder.

### Format

```

<htd:argument name="role">
 [new-role-name]
</htd:argument>

```

## 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. For more information, 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) inside the <API-M\_HOME>/business-processes folder with the new port (9763 + port offset.)

3. Open the <BPS\_HOME>/repository/conf/humantask.xml file and <BPS\_HOME>/repository/conf/b4p-coordination-config.xml file and set the TaskCoordinationEnabled property to true.

```
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```

4. Copy the following from <API-M\_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.

Make sure to give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.

- SubscriptionService.epr
- SubscriptionCallbackService.epr

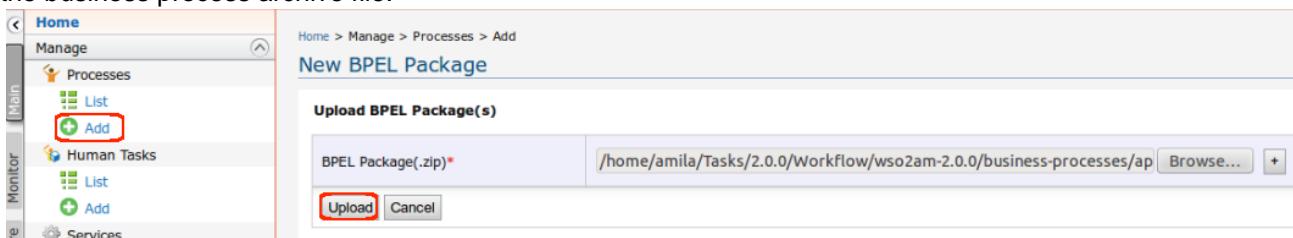
5. Start the BPS server and sign in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential
cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST,
uri:/carbon/admin/login_action.jsp, error:required token is
missing from the request)
```

To avoid this issue open <BPS\_HOME>/repository/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the BPS.

6. Select **Add** under the **Processes** menu and upload the <API-M\_HOME>/business-processes/subscription-creation/BPEL/SubscriptionApprovalWorkFlowProcess\_1.0.0.zip file to BPS. This is the business process archive file.



- Select **Add** under the **Human Tasks** menu and upload the <API-M\_HOME>/business-processes/subscription-creation/HumanTask/SubscriptionsApprovalTask-1.0.0.zip file to BPS. This is the human task archived file.

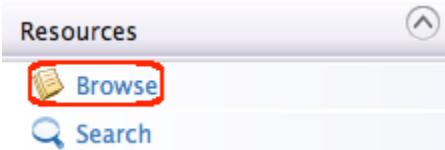
### Configuring the API Manager

Open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json file and configure "workFlowServerURL" under "workflows" to point to the BPS server (e.g. "workFlowServerURL": "https://localhost:9445/services/")

### Engaging the WS Workflow Executor in the API Manager

First, enable the API subscription workflow.

- Sign in to API Manager Management Console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



- Go to the /\_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.SubscriptionCreationWS
WorkflowExecutor">
 <Property
name="serviceEndpoint">http://localhost:9765/services/SubscriptionApp
rovalWorkFlowProcess/</Property>
 <Property name="username">admin</Property>
 <Property name="password">admin</Property>
 <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackSe
rvice</Property>
</SubscriptionCreation>
...
</WorkFlowExtensions>
```

**Note** that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

The application creation WS Workflow Executor is now engaged.

- 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.
- Note the message that appears if the BPEL is invoked correctly, saying that the request is successfully submitted.

5. Sign in to the Admin Portal (<https://<Server Host>:9443/admin>), 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="h
 ttp://workflow.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 <code>callBackURL</code> property in the <code>workflow-extensions.xml</code> file.

## Adding a User Signup Workflow

This section explains how to attach a custom workflow to the user signup operation in the API Manager. First, see [Workflow Extensions](#) for information on different types of workflow executors.

**Before you begin**, if you have changed the API Manager's default user and role, make sure you do the following changes:

- Change the credentials of the workflow configurations in the registry resource `_system/governance/apimgt/applicationdata/workflow-extensions.xml`.
- Point the database that has the API Manager user permissions to BPS.
- Share any LDAPs, if exist.
- Unzip the `<API-M>/business-processes/user-signup/UserApprovalTask-1.0.0.zip` file, update the role as follows in the `UserApprovalTask.ht` file, and ZIP the `UserApprovalTask.ht` folder.

### Format

```
<htd:argument name="role">
 [new-role-name]
</htd:argument>
```

## Configuring the Business Process Server

1. Download [WSO2 Business Process Server](#).
2. Set an offset of 2 to the default BPS port in the `<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. For more information, 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) inside `<apim_home>/business-processes` folder with the new port (9763 + port offset).

**Note:** Make sure that the port offset is updated in the following files as well. Note that the zipped files should be unzipped for you to be able to see the files

1. `<api-m_home>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip/UserApprovalTask.wsdl`
2. `<api-m_home>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip/UserApprovalTask.wsdl`
3. `<api-m_home>/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0.zip/WorkflowCallbackService.wsdl`

3. Open the <BPS\_HOME>/repository/conf/humantask.xml file and <BPS\_HOME>/repository/conf/b4p-coordination-config.xml file and set the TaskCoordinationEnabled property to true. For further information on this configuration see [Configuring Human Task Coordination](#).

```
<TaskCoordinationEnabled>true</TaskCoordinationEnabled>
```

4. Copy the following from the <API-M\_HOME>/business-processes/epr folder to the <BPS\_HOME>/repository/conf/epr folder. If the <BPS\_HOME>/repository/conf/epr folder isn't there, please create it.

Make sure to give the correct credentials in the <BPS\_HOME>/repository/conf/epr files.

- UserSignupService.epr
- UserSignupProcess.epr

5. Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential
cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST,
uri:/carbon/admin/login_action.jsp, error:required token is
missing from the request)
```

To avoid this issue open <BPS\_HOME>/repository/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the BPS.

6. Select **Add** under the **Processes** menu and upload the <API-M\_HOME>/business-processes/user-signup/BPEL/UserSignupApprovalProcess\_1.0.0.zip file to BPS. This is the business process archive file.



7. Select **Add** under the **Human Tasks** menu and upload the <API-M\_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0.zip file to BPS. This is the human task archived file.

## Configuring the API Manager

1. Open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json file and configure "workFlowServerURL" under "workflows" to point to the BPS server (e.g. "workFlowServerURL": "https://localhost:9445/services/")

## 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.UserSignUpWSWorkflowExecutor">
 <Property
name="serviceEndpoint">http://localhost:9765/services/UserSignupProcess</Property>
 <Property name="username">admin</Property>
 <Property name="password">admin</Property>
 <Property
name="callbackURL">https://localhost:8243/services/WorkflowCallbackService</Property>
 </UserSignUp>
...
</WorkFlowExtensions>
```

**Note** that all workflow process services of the BPS run on port 9765 because you changed its default port (9763) with an offset of 2.

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 Portal (<https://<Server Host>:9443/admin>) 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:wo
rkflowExternalRef>

 <wor:callbackURL>https://localhost:8243/services/WorkflowCallbackServ
ice</wor:callBackURL>
 </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 <code>callBackURL</code> property in the <code>api-manager.xml</code> file.

### 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	<a href="https://localhost:8243/services/WorkflowCallbackService">https://localhost:8243/services/WorkflowCallbackService</a> WSDL Location : <a href="http://localhost:8280/services/WorkflowCallbackService?wsdl">http://localhost:8280/services/WorkflowCallbackService?wsdl</a>
REST	<a href="https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag">https://localhost:9443/store/site/blocks/workflow/workflow-listener/ajax/workflow-listener.jag</a>

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/wo
rkflow-listener.jag -d
'workflowReference=b530be39-9174-43b3-acb3-2603a223b094&status=APPROVED&de
scription=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:workflowR
eference>
 <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.work
flow.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` o
bjects. 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 WorkflowResponse 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());
 }
 return new GeneralWorkflowResponse();
}
@Override
public WorkflowResponse complete(WorkflowDTO workflowDTO) throws
WorkflowException {
 workflowDTO.setUpdatedTime(System.currentTimeMillis());
 super.complete(workflowDTO);
 ApiMgtDAO apiMgtDAO = ApiMgtDAO.getInstance();
 try {
 apiMgtDAO.updateSubscriptionStatus(
 Integer.parseInt(workflowDTO.getWorkflowReference()),
APIConstants.SubscriptionStatus.UNBLOCKED);
 } catch(APIManagementException e) {
 throw new WorkflowException("Could not complete subscription
creation workflow", e);
 }
 return new GeneralWorkflowResponse();
}
}
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.

In a distributed setup, the custom workflows should be deployed in the Store node.

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.
  - <API-M\_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.impl\_6.1.66.jar
  - <API-M\_HOME>/repository/components/plugins/org.wso2.carbon.apimgt.api\_6.1.6.jar
  - javax.mail.jar: see <https://java.net/projects/javamail/pages/Home> to download the JAR
2. After exporting the JAR, copy it to <API-M\_HOME>/repository/components/lib directory.
3. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.



4. Go to the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` resource, disable the Simple Workflow Executor and enable the 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.SubscriptionCreationSi
mpleWorkflowExecutor"/-->
 <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 the `<API-M_HOME>/repository/components/p
lugins` directory. Removing it results in a `ClassNotFoundException` thrown at server startup, but it does not affect the server's functionality.

## Configuring HTTP Redirection for Workflows

This section walks you through how to redirect to a third party entity using the redirect URL as part of a custom workflow extension. For example, consider an API Manager user publishes an API and wants to make that API a chargeable API. If there are no payment details of the subscriber, that subscriber is forwarded to a third party entity that handles the payment detail collection etc.

- Writing the custom workflow executor
- Deploying the custom workflow executor
- Using the workflow
- Invoking the API Manager

Each workflow executor in the WSO2 API Manager is inherited from the `org.wso2.carbon.apimgt.impl.work
flow.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.

### **Writing the custom workflow executor**

1. Extend the `WorkflowExecutor` class found in the `org.wso2.carbon.apimgt.impl.workflow` packag
e.
2. Upon extension of the `WorkflowExecutor` class, override the `complete()` and `execute()` methods.
3. The `execute()` method is the first method called by API Manager. Call the `super.execute` method inside the `execute()` method to add the workflow entry to the database.
4. Create a response of type `WorkflowResponse`. For HTTP responses, WSO2 API Manager has an inbuilt

object named `HttpWorkflowResponse` found at `org.wso2.carbon.apimgt.api.WorkflowResponse`. When creating the HTTP workflow response object, specify the additional parameters and the redirect URL. The usage of these parameters are listed below.

Parameter	Mandatory	Description
Redirect URL	Yes	The URL that API Manager will redirect the user to upon workflow execution.
Redirect Confirmation Message	No	A redirection notification can be specified for a notification to appear at the point of redirecting. By default, this is set to a null value so the text must be specified in order for the notification to appear.
Additional Parameters	Yes	If you need to redirect back to the API Manager, call the workflow call back service to complete the workflow. To invoke this service, set the callback URL and the workflow reference ID in the additional parameters. These parameters are sent to the third party entity by query parameters.

5. Implement the `complete()` method, which the third party entity invokes to complete the workflow. Update the workflow status with the workflow status received by the third party entity.
6. A sample implementation of a custom workflow executor is shown below:

```

package org.wso2.sample.workflow;
/*
 * Copyright (c) 2015, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 *
 * WSO2 Inc. licenses this file to you under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except
 * in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

import org.wso2.carbon.apimgt.api.APIManagementException;
import org.wso2.carbon.apimgt.api.WorkflowResponse;
import org.wso2.carbon.apimgt.impl.APIConstants;
import org.wso2.carbon.apimgt.impl.dao.ApiMgtDAO;
import org.wso2.carbon.apimgt.impl.dto.WorkflowDTO;
import org.wso2.carbon.apimgt.impl.workflow.*;

import java.util.List;

/**
 * The lib class
 */

```

```

public class SubscriptionCreationSampleWorkflowExecutor extends
WorkflowExecutor {

 @Override
 public String getWorkflowType() {
 return WorkflowConstants.WF_TYPE_AM_SUBSCRIPTION_CREATION;
 }

 @Override
 public List<WorkflowDTO> getWorkflowDetails(String workflowStatus)
throws WorkflowException {
 return null;
 }
 /**
 * This method executes subscription creation simple workflow and
 return workflow response back to the caller
 *
 * @param workflowDTO The WorkflowDTO which contains workflow
 contextual information related to the workflow
 *
 * @return workflow response back to the caller
 * @throws WorkflowException Thrown when the workflow execution was
 not fully performed
 */
 @Override
 public WorkflowResponse execute(WorkflowDTO workflowDTO) throws
WorkflowException {
 super.execute(workflowDTO);

 HttpWorkflowResponse httpworkflowResponse = new
HttpWorkflowResponse();
 httpworkflowResponse.setRedirectUrl("http://google.lk");
 httpworkflowResponse.setAdditionalParameters("CallbackUrl",
"http://localhost:9763/store/site/blocks/workflow/workflow-listener/a
jax/workflow-listener.jag");
 httpworkflowResponse.setAdditionalParameters("workflowRefId" ,
workflowDTO.getExternalWorkflowReference());
 httpworkflowResponse.setRedirectConfirmationMsg("you will be
redirected to http://google.lk");
 return httpworkflowResponse;
 }

 /**
 * This method completes subscription creation simple workflow and
 return workflow response back to the caller
 *
 * @param workflowDTO The WorkflowDTO which contains workflow
 contextual information related to the workflow
 * @return workflow response back to the caller
 * @throws WorkflowException

```

```

 */
@Override
public WorkflowResponse complete(WorkflowDTO workflowDTO) throws
WorkflowException {
 workflowDTO.setUpdatedTime(System.currentTimeMillis());
 super.complete(workflowDTO);
 log.info("Subscription Creation [Complete] Workflow Invoked.
Workflow ID : " + workflowDTO.getExternalWorkflowReference() +
"Workflow State : " + workflowDTO.getStatus());
 ApiMgtDAO apiMgtDAO = new ApiMgtDAO();
 if(WorkflowStatus.APPROVED.equals(workflowDTO.getStatus())){
 try {

apiMgtDAO.updateSubscriptionStatus(Integer.parseInt(workflowDTO.getWo
rkflowReference()),
 APIConstants.SubscriptionStatus.UNBLOCKED);
 } catch (APIManagementException e) {
 log.error("Could not complete subscription creation
workflow", e);
 throw new WorkflowException("Could not complete
subscription creation workflow", e);
 }
 }else
if(WorkflowStatus.REJECTED.equals(workflowDTO.getStatus())){
 try {

apiMgtDAO.updateSubscriptionStatus(Integer.parseInt(workflowDTO.getWo
rkflowReference()),
 APIConstants.SubscriptionStatus.REJECTED);
 } catch (APIManagementException e) {
 log.error("Could not complete subscription creation
workflow", e);
 throw new WorkflowException("Could not complete
subscription creation workflow", e);
 }
}
return new GeneralWorkflowResponse();
}

```

```
 }
}
```

### **Deploying the custom workflow executor**

1. Once you have written the custom workflow executor, compile it to a .jar file.
2. Place the .jar file in the <APIM\_HOME>/repository/components/lib directory and restart the server.

### **Using the workflow**

1. Log in to API Manager Management Console (<https://<Server Host>:9443/carbon>) and select **Browse** under **Resources**.
2. Navigate to the /\_system/governance/apimgt/applicationdata/workflow-extensions.xml resource and disable the simple workflow executor.
3. Add and enable the implemented executor.
4. Specify the service endpoint where the workflow engine is hosted and the credentials required to access the service, via basic authentication (i.e., username/password based authentication).
5. An example configuration is shown below:

```
<WorkFlowExtensions>
...
<SubscriptionCreation
executor="org.wso2.sample.workflow.SubscriptionCreationSampleWorkflow
Executor">
</SubscriptionCreation>
...
</WorkFlowExtensions>
```

### **Invoking the API Manager**

To invoke the API Manager from a third party entity, see [Invoking the API Manager from the BPEL Engine](#).

### **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 `workflow-extensions.xml` as described in the previous sections.

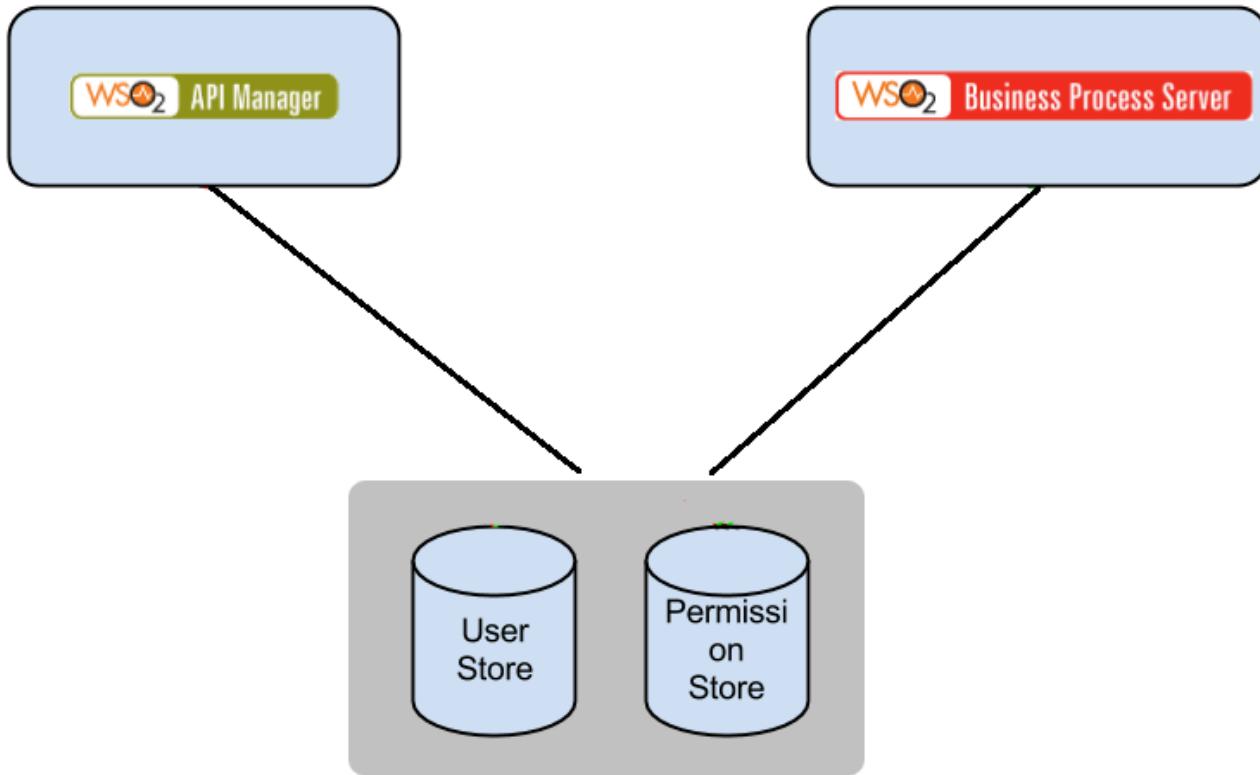
However, in a multi-tenant API Manager setup, not all tenants have access to the file system and not all tenants want to use the same workflow that the super admin has configured in the `api-manager.xml` file. For example, different departments in an enterprise can act as different tenants using the 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/s/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.

From WSO2 Carbon Kernel 4.4.6 onwards there are two MySQL DB scripts available in the product distribution. Click [here](#) to identify as to which version of the MySQL script to use.

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 3 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 5 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&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. 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 -->
<mount path="/_system/config" overwrite="true">
 <instanceId>mount</instanceId>
 <targetPath>/_system/nodes</targetPath>
</mount>
<mount path="/_system/governance" overwrite="true">
 <instanceId>mount</instanceId>
 <targetPath>/_system/governance</targetPath>
</mount>
</pre>

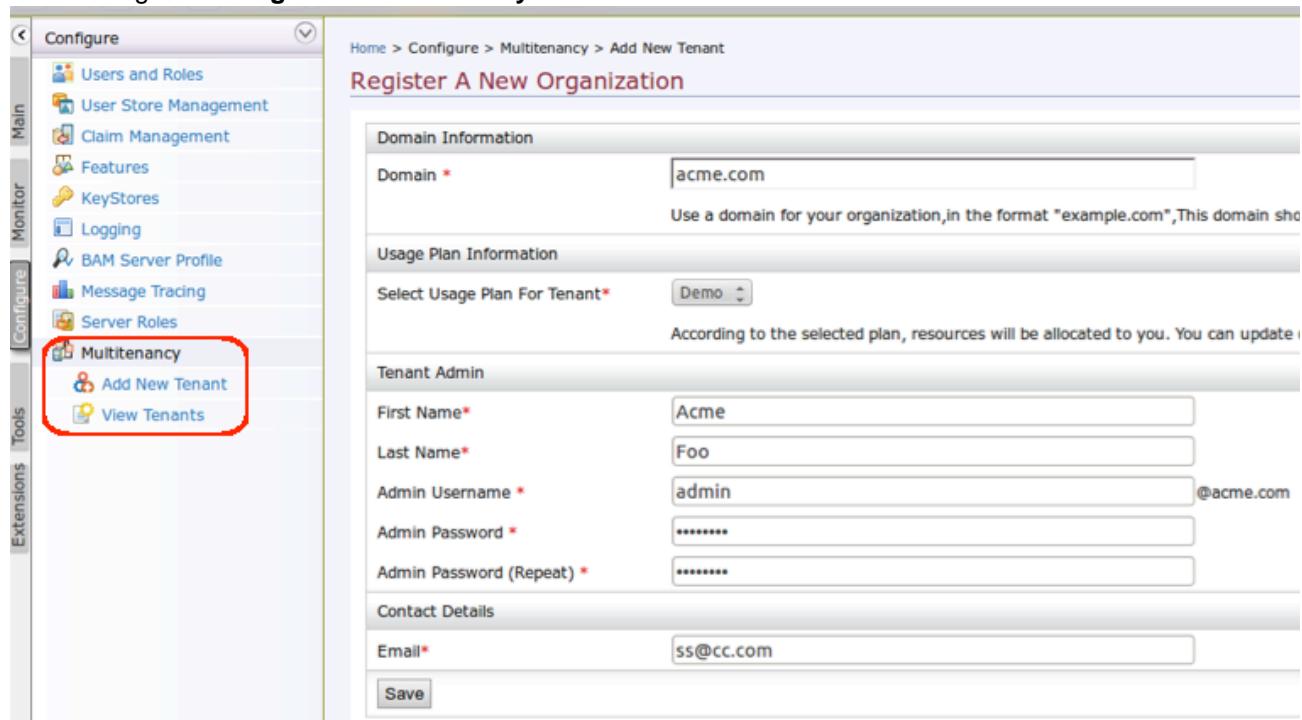
```

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 'Configure' section of the WSO2 API Manager management console. On the left, there is a sidebar with tabs: Main, Monitor, Configure, Tools, and Extensions. Under the 'Configure' tab, there is a list of options: Users and Roles, User Store Management, Claim Management, Features, KeyStores, Logging, BAM Server Profile, Message Tracing, Server Roles, Multitenancy, Add New Tenant, and View Tenants. The 'Multitenancy' option is highlighted with a red box. The main content area is titled 'Register A New Organization'. It contains sections for 'Domain Information' (Domain: acme.com), 'Usage Plan Information' (Select Usage Plan For Tenant: Demo), 'Tenant Admin' (First Name: Acme, Last Name: Foo, Admin Username: admin, Admin Password: [redacted], Admin Password (Repeat): [redacted]), and 'Contact Details' (Email: ss@cc.com). At the bottom is a 'Save' button.

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> and add the tenant admin credentials.

In a distributed setup, the ApplicationService-Tenant.epr's wsa:Address should point to the proxy/load balancer of Business Process Server(BPS cluster) ([http://<BPS\\_LB\\_hostname\\_here>/services/t/<tenant domain>/ApplicationService](http://<BPS_LB_hostname_here>/services/t/<tenant domain>/ApplicationService)). Also, the ApplicationCallbackService-Tenant.epr's wsa:Address should point to APIM cluster's Workflow Callback service endpoint. This is normally deployed at the gateway nodes. The wsa:Address should point to the gateway nodes. ([https://<API\\_gateway\\_LB\\_hostname\\_here>/services/WorkflowCallbackService](https://<API_gateway_LB_hostname_here>/services/WorkflowCallbackService)) and the user credentials which grant access to that service should be used.

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>

```

- 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

```

- Log into the BPS as the tenant admin and upload the BPEL.

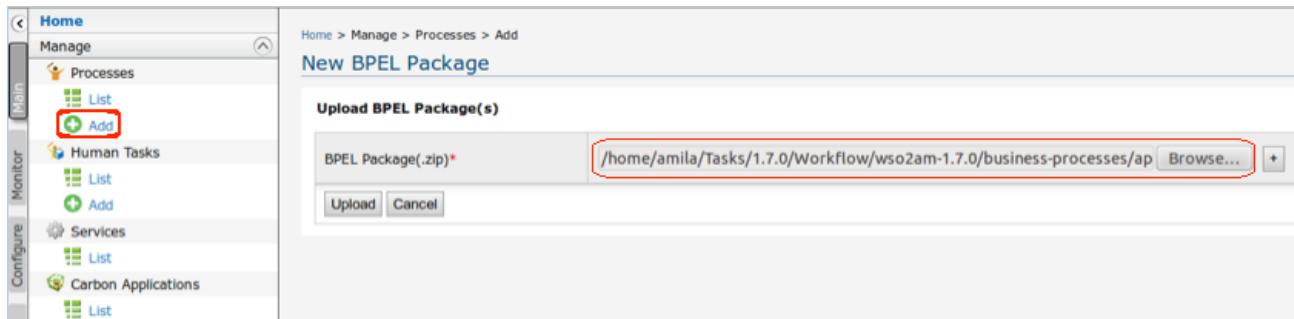
If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```

WARN {org.owasp.csrfguard.log.JavaLogger} - potential
cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST,
uri:/carbon/admin/login_action.jsp, error:required token is
missing from the request)

```

To avoid this issue open <BPS\_HOME>/repository/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the BPS.



### **Creating a Tenant for Authentication**

Step 1: Create a registry resource in the tenant's configuration registry

1. Start the BPS server If it is not started already.
2. Navigate to **Registry>Browse** in the **Main** menu of the management console and click on `/_system/config`.

Home > Registry > Browse

### Browse

Root /

Location: / Go

Tree view Detail view

- /
- \_system
  - config
  - repository
  - TaskCoordination
  - users
- governance
- local

3. Click on **Entries>Add Resource** and fill the form using the values listed below for guidance. See [Adding a Resource](#) for more information.

Method	Name	Media Type
Create Text Content	TaskCoordination	text/plain

4. Click **Add** to finish adding the resource.

Add Resource

Method

Create Text Content

Name *	<input type="text" value="TaskCoordination"/>
Media type	<input type="text" value="text/plain"/>
Description	<input type="text" value="Task Coordination"/>
Content	<input checked="" type="radio"/> Plain Text Editor <input type="radio"/> Rich Text Editor <div style="border: 1px solid #ccc; min-height: 100px;"></div>

Step 2: Create username and password registry properties and define credentials

1. Click on the registry resource you created (Task Coordination) found under the **Entries** section.

Entries				
		Actions		
Name		Created On	Author	
 repository	<input type="checkbox"/> Info <input type="checkbox"/> Actions	56m ago	wso2.system..	
 TaskCoordination	<input type="checkbox"/> Info <input type="checkbox"/> Actions	22m ago	admin	
 users	<input type="checkbox"/> Info <input type="checkbox"/> Actions	56m ago	wso2.system..	

2. Add two new registry properties for the resource called "Username" and "Password", and define the tenant coordination user credentials. To do this, click **Properties>Add New Property** and enter the following values. See [Managing Properties](#) for more information.

Username Property	Password Property
<b>Name:</b> username	<b>Name:</b> password
<b>Value:</b> (username value)	<b>Value:</b> (password value)

**Properties**

[+ Add New Property](#)

Add New Property	
Name*	<input type="text" value="Username"/>
Value	<input type="text" value="wso2_demo@wso2.com"/>

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

3. Click **Add** to finish adding the property.

Properties		
<a href="#">Add New Property</a> <span style="float: right;">( - )</span>		
3 Properties		
Name	Value	Action
Password	wso2demo	<a href="#">Edit</a> <a href="#">Delete</a>
Username	wso2_demo@wso2.com	<a href="#">Edit</a> <a href="#">Delete</a>
resource.source	AdminConsole	<a href="#">Edit</a> <a href="#">Delete</a>

### 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>
```

In a distributed setup, the above addresses should be changed to point to the BPS proxy/loadbalancer. A sample is shown below.

```
<soap:address location="http://<BPS_LB_hostname_here>/services/t/<tenant_d
omain>/ApplicationServiceCB" />
```

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:976

5/services/t/<tenant\_domain>/ApplicationApprovalWorkFlowProcess). Also set the **credentials** as the **tenant admin's credentials** of the ApplicationCreationWSWorkflowExecutor file. For example,

The screenshot shows the WSO2 API Manager Governance interface. On the left, the sidebar has tabs for Home, Main, Monitor, Configure, Tools, and Extensions. The 'Configure' tab is selected. Under 'Resources', the 'Browse' option is highlighted with a red box. The main content area shows a 'Browse' page for the file '/\_system/governance/apimgt/applicationdata/workflow-extensions.xml'. The 'Content' section displays XML code for the ApplicationCreationWSWorkflowExecutor. The XML includes properties for service endpoint, username, password, and callback URL. At the bottom of the content area are 'Save Content' and 'Cancel' buttons.

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 Portal (<https://<Server Host>:9443/admin>) 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 Portal. The left sidebar has a 'TASKS' dropdown with options: USER CREATION, APPLICATION CREATION (which is selected), SUBSCRIPTIONS CREATION, APPLICATION REGISTRATION, API STATE CHANGE, SETTINGS, THROTTLING POLICIES, LOG ANALYZER, and ANALYTICS. The main content area is titled 'Approval Tasks'. It features a search bar labeled 'Filter by ...'. A table lists an approval task with the following details:

ID	Description	Status	Created On
651	Approve application [app1] creation request from application creator - admin with throttling tier - Unlimited	RESERVED	2017-12-21 - 15:54:53.393+05:30

At the bottom, there are buttons for 'Show 10 entries' and 'Showing 1 to 1 of 1 entries'.

## 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 Portal (<https://<Server Host>:9443/admin>) 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 Portal from the same user store as the API Manager. Therefore, you can use the Admin Portal 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 Portal users reside in a separate user store.

1. If you want to change the user roles that can access the Admin Portal, open the `<APIM_HOME>/repository/deployment/server/jaggeryapps/admin/site/conf/site.json` file that is in the node from where you access the Admin Portal (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>&lt;APIM_HOME&gt;/business-processes/user-signup/BPEL/UserSignupApprovalProcess_1.0.0/UserApprovalTask.wsdl</code> file, update the <code>&lt;wsdl:port&gt;</code> and <code>&lt;wsdl:service&gt;</code> elements.</p> <pre style="border: 1px dashed #ccc; padding: 10px;"> &lt;wsdl:service name="UserApprovalService"&gt;     &lt;wsdl:port name="UserApprovalPort" binding="tns:UserApprovalPort"&gt;         &lt;soap:address location="http://localhost:9765/services/UserApprovalService/UserApprovalPort"&gt;         &lt;/wsdl:port&gt;     &lt;/wsdl:service&gt;     &lt;wsdl:service name="UserApprovalServiceCB"&gt;         &lt;wsdl:port name="UserApprovalPortCB" binding="tns:UserApprovalPortCB"&gt;             &lt;soap:address location="http://localhost:9765/services/UserApprovalService/UserApprovalPortCB"&gt;             &lt;/wsdl:port&gt;         &lt;/wsdl:service&gt;     &lt;/wsdl:service&gt; </pre> <p><b>Note</b> that all workflow process services of the BPS run on port 9765 because you have specified the port offset as 2 in the <code>site.json</code> file.</p>

Open the CallbackService WSDL file and point the address elements to the API Store node in NIO port.

In the <APIM\_HOME>/business-processes/user-signup/BPEL/UserSignupP

```

<wsdl:service name="WorkflowCallbackService">
 <wsdl:port name="WorkflowCallbackServiceHttpsSoap11Binding" binding="ns:WorkflowCallbackServiceSoap11Binding">
 <soap:address location="https://localhost:8243/services/WorkflowCallback"/>
 </wsdl:port>
 <wsdl:port name="WorkflowCallbackServiceHttpSoap11Binding" binding="ns:WorkflowCallbackServiceSoap11Binding">
 <soap:address location="http://localhost:8280/services/WorkflowCallback"/>
 </wsdl:port>
 <wsdl:port name="WorkflowCallbackServiceHttpsSoap12Binding" binding="ns:WorkflowCallbackServiceSoap12Binding">
 <soap12:address location="https://localhost:8243/services/WorkflowCallback"/>
 </wsdl:port>
 <wsdl:port name="WorkflowCallbackServiceHttpSoap12Binding" binding="ns:WorkflowCallbackServiceSoap12Binding">
 <soap12:address location="http://localhost:8280/services/WorkflowCallbacks"/>
 </wsdl:port>
 <wsdl:port name="WorkflowCallbackServiceHttpsEndpoint" binding="ns:WorkflowCallbackServiceHttpsEndpoint">
 <http:address location="https://localhost:8243/services/WorkflowCallback"/>
 </wsdl:port>
 <wsdl:port name="WorkflowCallbackServiceHttpEndpoint" binding="ns:WorkflowCallbackServiceHttpEndpoint">
 <http:address location="http://localhost:8280/services/WorkflowCallbacks"/>
 </wsdl:port>
</wsdl:service>
```

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 role, open the ApprovalTask HT file and apply the changes there.

Change the admin instances in <APIM\_HOME>/business-processes/user-signup,/UserApprovalTask-1.0.0/**UserApprovalTask.ht** file. Here's an example, assuming new admin role is apimadmin.

```
<htd:peopleAssignments>
 <htd:potentialOwners>
 <htd:from logicalPeopleGroup="admin">
 <htd:argument name="role">apimadmin</htd:argument>
 </htd:from>
 </htd:potentialOwners>
</htd:peopleAssignments>
```

Open the ApprovalTask WS DL file and point the two address elements to the Business Process Server node.

In the <APIM\_HOME>/business-processes/user-signup/HumanTask/UserApprovalTask-1.0.0/**UserApprovalTask.wsdl** file:

```
<wsdl:service name="UserApprovalService">
 <wsdl:port name="UserApprovalPort"
 binding="tns:UserApprovalBinding">
 <soap:address
 location="http://localhost:9765/services/UserApprovalService"
 </wsdl:port>
 </wsdl:service>
<wsdl:service name="UserApprovalServiceCB">
 <wsdl:port name="UserApprovalPortCB"
 binding="tns:UserApprovalBindingCB">
 <soap:address
 location="http://localhost:9765/services/UserApprovalServiceCB"
 />
 </wsdl:port>
 </wsdl:service>
```

**Note** that all workflow process services of the BPS run on port 9765 because you can't change the default port (9763) with an offset of 2.

## 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/apiManager/applicationdata/workflow-extensions.xml)
3. Point the same database that has the permissions used by the API Manager to the BPS.
4. Share the LDAP, if it exists.
5. If you change the default user role, change the .ht file of the relevant human task.

## Cleaning Up Workflow Tasks

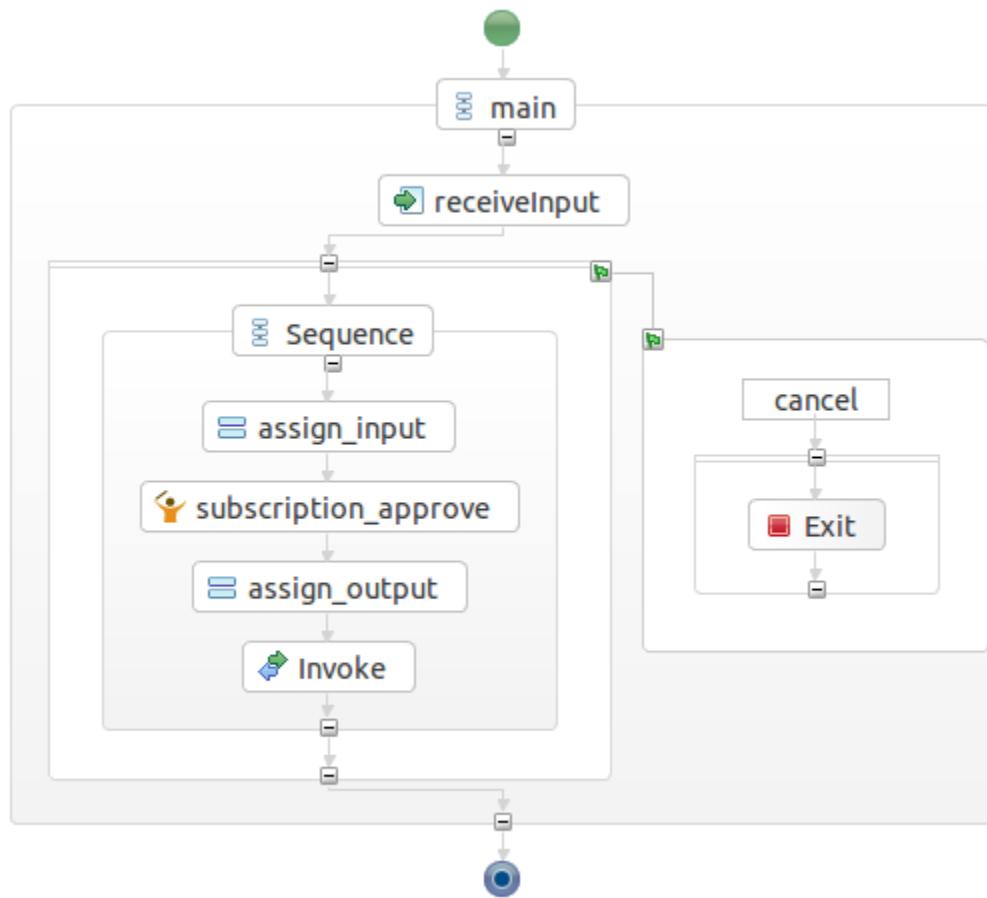
When workflow extensions are enabled using default workflow executors, they create respective approval tasks in WSO2 API Manager.

SO2 Business Process Server (BPS). Each of these tasks are visible to administrators in the Admin Portal. The administrator has the option to accept or reject each of the requests made by other users. At the same time, users have the option to delete the application, subscription or key they created before the administrator accepts or rejects their requests. This leaves unnecessary approval requests in the Admin Portal, which can confuse the administrator.

API Manager provides a task clean up feature to prevent deleted items from showing up in the Admin Portal. The `WorkflowExecutor` class is introduced with the `cleanUpPendingTask(String workflowExtRef)` method, which is triggered by application or subscription deletion. This method implements the logic to notify WSO2 BPS that a task with the `workflowExtRef` ID has been deleted.

The BPEL process in WSO2 BPS should contain a cancel event to support process cancellation. Each BPEL process should support correlation and event cancellation in order to successfully cleanup unnecessary tasks. For more information on BPEL correlation, see [Process Correlation](#) and the [BPEL Correlation Guide](#).

The final BPEL should have a design similar to the following diagram,



Follow the steps below to test this out.

This example assumes that workflows are enabled for application creation, application registration and subscription creation.

1. Log in to the API Store and create two new applications.
2. Log in to the Admin Portal (<https://<Server Host>:9443/admin>) and approve the creation of one application.
3. In the API Store, subscribe an API to the approved application.
4. Generate production and/or sandbox key(s) for the approved application.
5. Check the pending approval tasks in the Admin Portal. You see tasks pending for application creation, application registration and subscription creation.
6. Delete the items you created from the API Store and notice that the respective administrator approval tasks

- are removed.
- If the application with pending subscription and key generation approvals is deleted from the API Store, all the pending subscription and key generation approval tasks are deleted for that application.

## Adding an API State Change Workflow

This section explains how to add a custom workflow to control the API state changes in the API Manager. First see [Workflow Extensions](#) for more information on different types of workflow executors. For more details on API states see [API Lifecycle](#).

### Configuring the Business Process Server

- Download WSO2 Business Process Server.
- 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) inside <APIM\_HOME>/business-processes folder with the new port (9763 + port offset).

- Start the BPS server and log in to its management console (<https://<Server Host>:9443+<port offset>/carbon>).

If you are using Mac OS with High Sierra, you may encounter following warning when login into the Management console due to a compression issue exists in High Sierra SDK.

```
WARN {org.owasp.csrfguard.log.JavaLogger} - potential
cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:xxx.xxx.xx.xx, method:POST,
uri:/carbon/admin/login_action.jsp, error:required token is
missing from the request)
```

To avoid this issue open <BPS\_HOME>/repository/conf/tomcat/catalina-server.xml and change the compression="on" to compression="off" in Connector configuration and restart the BPS.

- Select **Processes > Add > BPMN** and upload the <APIM\_HOME>/business-processes/api-state-change/APIStateChangeApprovalProcess.bar file to BPS.

### Configuring the API Manager

1. Open <APIM\_HOME>/repository/conf/api-manager.xml and set in <Enabled> to true in the <Work flowConfigurations> section.
2. Change the <ServerUrl> if you have configured the BPS to run on a different port offset.

### Engaging the WS Workflow Executor in the API Manager

First, enable the API state change workflow.

1. Log in to the APIM management console (<https://<Server Host>:9443/carbon>) and select Browse under Resources.

2. Go to the /\_system/governance/apimgt/applicationdata/workflow-extensions.xml resource, disable the Simple Workflow Executor and enable WS Workflow Executor.

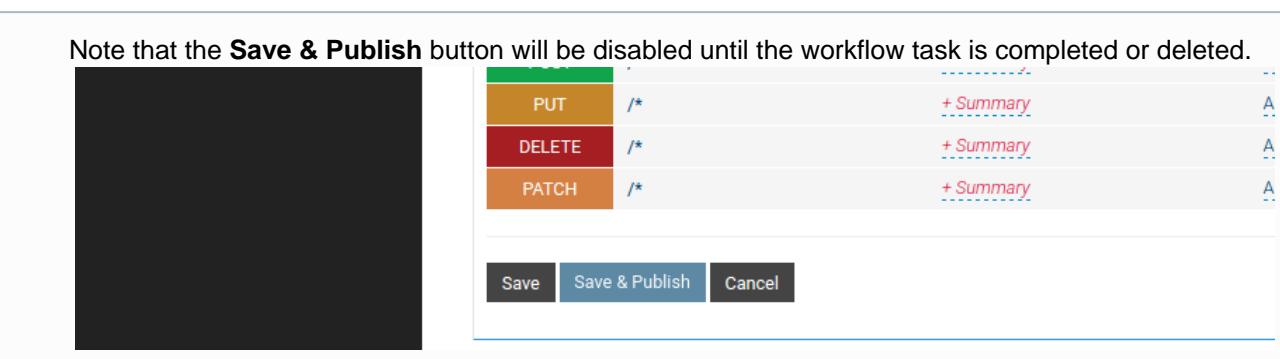
```
<WorkFlowExtensions>
 ...
 <!-- Publisher related workflows -->
 <!--<APIStateChange
 executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeSimpleWorkflowExecutor" />-->
 <APIStateChange
 executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeWSWorkflowExecutor">
 <Property
 name="processDefinitionKey">APIStateChangeApprovalProcess</Property>
 <Property
 name="stateList">Created:Publish, Published:Block</Property>
 </APIStateChange>
 ...
 </WorkFlowExtensions>
```

You have now engaged the API WS Workflow. The default configuration is set for **Created to Publish** and **P**

- published to Block** state changes. See [Advanced Configurations](#) for information on configuring more state changes.
- Log in to the API Publisher (<https://<Server Host>:9443/publisher>) and publish an API. See [Create and Publish an API](#). A message related to the publish workflow will be displayed because the workflow is enabled for **Created to Publish** state change.



The screenshot shows a modal window titled "API Publisher -". It contains a message icon and the text "Lifecycle state change request has been submitted." with an "OK" button.

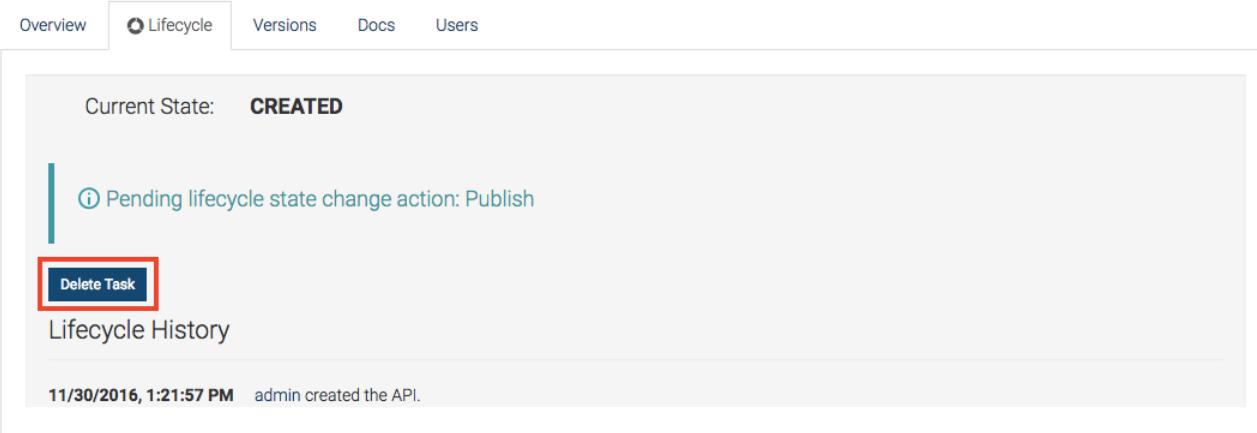
The screenshot shows a modal window with a note: "Note that the **Save & Publish** button will be disabled until the workflow task is completed or deleted." Below this is a table of API methods:

Method	Path	Action
PUT	/*	+ Summary
DELETE	/*	+ Summary
PATCH	/*	+ Summary

At the bottom are "Save", "Save & Publish", and "Cancel" buttons.

- You can revoke the state change by clicking **Delete Task** in the **Lifecycle** tab.

## WorldBank - 1.0.0



The screenshot shows the "Lifecycle" tab for the "WorldBank - 1.0.0" API. The "Current State" is "CREATED". A message says "Pending lifecycle state change action: Publish". A "Delete Task" button is highlighted with a red box. The "Lifecycle History" section shows a log entry: "11/30/2016, 1:21:57 PM admin created the API."

- Log in to the Admin Portal (<https://<Server Host>:9443/admin>) and click API State Change to see the list of tasks awaiting approval.

Click **Assign to Me** to approve the task. Select Approve and click **Complete** to resume and complete the API state change.

## Approval Tasks

### Configuring the BPS for tenants

1. Log in to the BPS with the credentials of the tenant. Select **Processes > Add > BPMN** and upload the `<APIM_HOME>/business-processes/api-state-change/APIStateChangeApprovalProcess.bar` file to BPS.
2. Copy the `<BPS_HOME>/repository/deployment/server/webapps/bpmn.war` web app to `<BPS_HOME>/wso2bps-3.6.0/repository/tenants/<tenant_id>/webapps`.
3. To engage the WS Workflow Executor, log in to the admin console using the credentials of the tenant and repeat step 2 from [Engaging the WS Workflow Executor in the API Manager](#).

### Advanced Configurations

Given below are the configurations that can be changed by editing `<APIM_HOME>/repository/conf/api-manager.xml`

```

<WorkflowConfigurations>
 <Enabled>true</Enabled>
 <ServerUrl>https://localhost:9445/bpmn</ServerUrl>
 <ServerUser>${admin.username}</ServerUser>
 <ServerPassword>${admin.password}</ServerPassword>
 <WorkflowCallbackAPI>https://localhost:${mgt.transport.https.port}/api/am/publisher/v0.10/workflows/update-workflow-status</WorkflowCallbackAPI>
 <TokenEndPoint>https://localhost:${https.nio.port}/token</TokenEndPoint>

 <DCREndPoint>https://localhost:${mgt.transport.https.port}/client-registration/v0.10/register</DCREndPoint>
 <DCREndPointUser>${admin.username}</DCREndPointUser>
 <DCREndPointPassword>${admin.password}</DCREndPointPassword>
 </WorkflowConfigurations>

```

The elements of the above configuration are explained below.

Element name	Description
Enabled	Enables the Admin Portal to approve state change tasks.
ServerUrl	The URL of the BPMN server.
ServerUser	User accessing the BPMN REST API.
ServerPassword	Password of the user accessing the BPMN REST API.
WorkflowCallbackAPI	The REST API invoked by the BPMN to complete the workflow.
TokenEndPoint	The API call to generate the access token is passed to the BPMN process. Once the access token is received, it is used to call the workflow callback API.
DCREndPoint	Endpoint to generate OAuth application. This application is used by the BPMN process to generate the token.
DCREndPointUser	Endpoint user.
DCREndPointPassword	Endpoint password.

## Setting a DCREndPointUser

Create a user with exclusive **apim:apiworkflow** scope permissions when setting a `DCREndPointUser`. Please avoid using super admin credentials. If super admin credentials are used, the created OAuth application will have all the permissions related to scopes in the other REST APIs. Follow the steps below to create a user with **apim:apiworkflow** scope permissions:

1. Log in to APIM management console (<https://<Server Host>:9443/carbon>) and create a role named `workflowCallbackRole`. Set create and publisher or subscriber permissions to this role.
2. Go to **Resources** and click **Browse**. Go to `/_system/config/apimgt/applicationdata/tenant-conf.json` and update the role related to 'apim:api\_workflow' scope with the newly created

role.

```

 ...
 {
 "Name": "apim:api_workflow",
 "Roles": "workflowCallbackRole"
 }
 ...

```

3. Assign this role to a user.
4. Update <DCREndPointUser> and <DCREndPointPassword> with this user's credentials.

For more details on how to create users and roles see [managing users and roles](#).

The configurations that can be changed by editing the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` are given below.

### Simple WorkFlow

```

<APIStateChange
executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeSimpleWorkflowExecutor" />

```

### WS WorkFlow

```

<APIStateChange
executor="org.wso2.carbon.apimgt.impl.workflow.APIStateChangeWSWorkflowExecutor">
 <Property
name="processDefinitionKey">APIStateChangeApprovalProcess</Property>
 <Property name="stateList">Created:Publish,Published:Block</Property>

</APIStateChange>

```

The elements of the above configuration are explained below.

Element Name	Mandatory/Optional	Description
processDefinitionKey	Mandatory	BPMN process definition id. BPMN process provided with AM as default has 'APIStateChangeApprovalProcess' as the id
stateList	Mandatory	This is a comma separated list of the current state and intended action. For example, Created:Publish,Published:Block

serviceEndpoint	Optional	The URL of the BPMN process engine. This overrides the global <ServerUrl> value from the api-manager.xml file. This can be used to connect a separate workflow engine for a tenant.
username	Optional	Username for the external BPMN process engine. This overrides <ServerUser> defined in the api-manager.xml file for the tenant.
password	Optional	password for the external BPMN process engine. This overrides <ServerPassword> defined in the api-manager.xml file for the tenant.

## 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*

When 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. You find the default handlers in any API's Synapse definition as shown below.

```

<handlers>
 <handler
 class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler"
 >
 <property name="apiImplementationType" value="ENDPOINT" />
 </handler>
 <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="policyKeyResource"
 value="gov:/apimgt/applicationdata/res-tiers.xml" />
 <property name="policyKey"
 value="gov:/apimgt/applicationdata/tiers.xml" />
 <property name="policyKeyApplication"
 value="gov:/apimgt/applicationdata/app-tiers.xml" />
 </handler>
 <handler
 class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler" />
 <handler
 class="org.wso2.carbon.apimgt.usage.publisher.APIMgtGoogleAnalyticsTrackingHandler" />
 <property name="configKey"
 value="gov:/apimgt/statistics/ga-config.xml" />
 </handler>
 <handler
 class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler" />
 </handlers>

```

Let's see what each handler does:

- **CORSRequestHandler:** Sets the CORS headers to the request and executes the CORS sequence mediation logic. This handler is thereby responsible for returning the CORS headers from the gateway or routing the requests to the backend and letting the backend send the CORS headers.
- **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 WSO2 Data Analytics Server (WSO2 DAS) for collection and analysis of statistics. This handler only comes into effect if API usage tracking is enabled. See the [Analytics](#) section 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](#).

### **Using APILogMessageHandler**

Message logging is handled by **APIManagerExtensionHandler** for API Manager 1.9.1 and above. Addition to the above mentioned Handlers, **APILogMessageHandler** has introduced from API Manager version 1.9.1 onwards. **APILogMessageHandler** is a sample handler that comes with WSO2 API Manager that can be used for logging.

#### **Why are logs removed from APIManagerExtensionHandler?**

The primary purpose of ExtensionHandler is handling extensions to mediation and not for logging messages. When the logs are also included in ExtensionHandler, there's a limitation to improve the ExtensionHandler for developing features because it breaks the logs.

For example, When the ExtensionHandler moves to the top of the handlers set, most of the logs print null values since the handler runs before the **APIAuthenticationHandler**. Therefore, the logs are removed from the extension handler and **APILogMessageHandler** introduced as a sample.

To achieve logging requirements, this handler is not the only approach and with custom sequences also it is possible to log messages using the Log Mediator.

In order to enable logging by invoking **APILogMessageHandler**, follow the steps below.

#### **To enable Message Logging per API :**

1. Open the synapse Configuration of the API located in <APIM\_HOME>/repository/deployment/server/synapse-configs/default/api directory and add below handler before </Handlers>.

```
<handler
 class="org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageH
 andler"/>
```

2. Copy the following code into the <APIM\_HOME>/repository/conf/log4j.properties file to enable printing DEBUG logs.

```
log4j.logger.org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMe
ssageHandler = DEBUG
```

3. Restart API Manager.

#### **To enable Message Logging into APIS created from publisher automatically :**

1. Open the <APIM\_HOME>/repository/resources/api\_templates/velocity\_template.xml file and copy the following handler before </Handlers>.

```
<handler
class="org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageH
andler"/>
```

## 2. Restart API Manager.

To perform analytics with the logs, see [Analyzing the Log Overview](#).

### ***Writing a custom handler***

The outcome of using a Class Mediator vs. a Synapse Handler are very similar. However, when using a custom handler you need to maintain a customized velocity template file that needs to be manually merged when you upgrade your product to a newer version. Therefore, it is recommended to use custom Handlers when you wish to specify the exact order of execution of JARs as this can not be done with [Mediators](#).

Custom Handler is a way of extending API Manager which the product offer to change the API flow through the API Gateway. What is happening in custom handler can be decided by the code you are writing to extend the product. It can be adding extra security, logging database invocation or something else. This custom handler must extend the org.apache.synapse.rest.AbstractHandler class and implement handleRequest() and handleResponse() methods.

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. A sample implementation is [here](#). Similarly, you can extend the API Manager to support any custom authentication mechanism by writing your own authentication handler class.

Given below is an example implementation. Please find the complete project archive [org.wso2.carbon.test.authenticator.zip](#). You can download, unzip and build the project using maven and Java 7/8.

```

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("userNName")) {
 return true;
 }
 return false;
 }

 private String getAuthorizationHeader(Map headers) {
 return (String) headers.get("Authorization");
 }

 private Map getTransportHeaders(MessageContext messageContext) {
 return (Map) ((Axis2MessageContext)
messageContext).getAxis2MessageContext().
getProperty(org.apache.axis2.context.MessageContext.TRANSPORT_HEADERS);
 }
}

```

### ***Engaging the custom handler***

1. Build the custom authenticaor code downloaded previously, and copy the resulting jar to <API-M\_HOME>/repository/components/dropins directory.

2. Engage the custom handler using the API template as explained below: You can engage a custom handler to all APIs at once or only to selected APIs. To engage a custom handler to APIs, you need to add the custom handler with its logic in the <APIM\_HOME>/repository/resources/api\_templates/velocity\_template.xml file.

It is not recommended to update the API source code via the source view UI or file system when engaging a custom handler to selected APIs, because the customizations get overridden by the publisher updates.

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.CustomAPI
AuthenticationHandler" />
 #foreach($handler in $handlers)
 #if(!$handler.className ==
"org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHa
ndler"))
 <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>
```

You can select to which API(s) you need to engage the handler. Given below is an example of adding only the CustomAPIAuthenticationHandler to the sample PizzaShackAPI.

```

<handlers xmlns="http://ws.apache.org/ns/synapse">
#if($apiName == 'admin--PizzaShackAPI')
 <handler
 class="org.wso2.carbon.sample.auth.CustomAPIAuthenticationHandler"/>
#end
#foreach($handler in $handlers)
 #if($apiName != 'admin--PizzaShackAPI' || !($handler.className ==
"org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHa
ndler"))
 <handler xmlns="http://ws.apache.org/ns/synapse"
 class="$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>

```

3. Restart the API Manager server.

## Adding Mediation Extensions

This tutorial uses the WSO2 API Manager Tooling Plug-in .

The API Gateway has a default mediation flow, which you can extend by adding custom mediation sequences. In API Manager there are 3 **default sequences** engaged as **in**, **out** and **fault**. You create a custom mediation sequence either manually or using a tool such as the **WSO2 API Manager Tooling Plug-in**, and then engage it per API or globally to all APIs of a specific tenant. With custom mediation sequences you can modify the default mediation flow for different usabilities according to your requirement. Log the mediation flow, execute operations on Message context properties, to customize, format the requests and responses are some of them.

- Default mediation flow
- Creating per-API extensions
- Creating global extensions

- The following **mediators are not usable within custom sequences** because they are not supported by the API Gateway.
  - Call mediator in non-blocking mode
  - Send mediator
- When using the Loopback mediator, it is mandatory to set the following property before defining the Loopback mediator in the custom mediator sequence in the following manner.

```

<property name="api.ut.backendRequestTime"
expression="get-property('SYSTEM_TIME')"/>

```

### **Default mediation flow**

You cannot dynamically construct the back-end endpoint of an API using the address endpoints in the WSO2 API Manager. To achieve the requirement of a dynamic endpoint, you can use the default endpoint instead. The default endpoint sends the message to the address specified in the **To** header. The **To** header can be constructed dynamically. For example,

```
<sequence xmlns="http://ws.apache.org/ns/synapse"
name="default-endpoint-seq">
 <property name="service_ep"
expression="fn:concat('http://jsonplaceholder.typicode.com/' , 'posts/')" />
 <header name="To" expression="get-property('service_ep')"/>
</sequence>
```

In this example, you have constructed the `service_ep` property dynamically and assigned the value of this property to the **To** header. The default endpoint sends the message to the address specified in the **To** header, in this case, `http://jsonplaceholder.typicode.com/posts/`. For more details about working with dynamic endpoints, see [Dynamic Endpoints](#).

## **Adding a non-blocking send operation**

In this example, the Send mediator in a proxy service using the [VFS transport](#) is transferring a file to a VFS endpoint. VFS is a non-blocking transport by default, which means a new thread is spawned for each outgoing message. The [Property mediator](#) added before the Send mediator removes the [ClientAPINonBlocking](#) property from the message to perform the mediation in a single thread. This is required when the file being transferred is large and you want to avoid out-of-memory failures.

```
<inSequence>
 <property name="ClientApiNonBlocking"
 value="true"
 scope="axis2"
 action="remove" />
 <send>
 <endpoint name="FileEpr">
 <address uri="vfs:file:///home/shammi/file-out"/>
 </endpoint>
 </send>
</inSequence>
```

### **Creating per-API extensions**

- Create and upload using the WSO2 API Manager Tooling Plug-in
- Create and upload manually in the API Publisher
- Create manually and save in the file system

#### **Create and upload using the WSO2 API Manager Tooling Plug-in**

The **recommended way** to engage a mediation extension sequence per API is to create a custom sequence using the [WSO2 API Manager Tooling Plug-in](#), upload it via its APIM Perspective and then engage it using the API

Publisher. The following tutorial demonstrates how to do this: [Change the Default Mediation Flow of API Requests](#).

### Create and upload manually in the API Publisher

You can also create a mediation sequence manually and upload it from the API Publisher itself. For instance, you can copy the above default mediation flow content into an XML file. In the **Implement** tab of the API, select the **Enable Message Mediation** check box and click the **Upload In Flow** or **Upload Out Flow** field (for the example above, it needs to be uploaded to the **In** flow). Once the file is uploaded, save and publish the API. When you invoke the API, the request is sent to the endpoint referred to in the **To** header.

YahooWeather: /weather/1.0

The screenshot shows the 'Implement' tab of the API Publisher. Under 'Message Mediation Policies', the 'In Flow' section is highlighted with a red box. It contains a dropdown menu set to 'YahooWeatherSequence' and a 'Upload In Flow' button. Below it are sections for 'Out Flow' (None) and 'Fault Flow' (None), each with its own 'Upload' button. Other tabs like 'Design' and 'Manage' are visible at the top.

### Create manually and save in the file system

Alternatively, you can name the mediation XML file in the pattern `<API_NAME>:v<VERSION>--<DIRECTION>` and save it directly in the following location:

- In the **single-tenant mode**, save the XML file in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory.
- In the **multi-tenant mode**, save the XML file in the tenant's synapse sequence folder. For example, if tenant id is 1, then save it in `<API_Gateway>/repository/tenants/1/synapse-configs/default/sequences` folder.

In the naming pattern, the `<DIRECTION>` can be `In` or `Out`. When it is `In`, the extension is triggered on the in-flow (request path) and when it is `Out`, the extension is triggered on the out-flow (response path). 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.

An example synapse configuration of a per-API extension sequence created for the API `admin--TwitterSearch`

version 1.0.0 is given below.

```
<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>
```

You can copy this content into an XML file (e.g., `twittersearch_ext.xml`) and save it in the `<API_Gateway>/repository/deployment/server/synapse-configs/default/sequences` directory.

The above sequence prints a log message on the console whenever the TwitterSearch API is invoked.

#### ***Creating global extensions***

You can also engage mediation extension sequences to all APIs of a specific tenant at once. To do that, simply create the XML with the naming pattern `WSO2AM--Ext--<DIRECTION>` and save it in the `<APIM_HOME>/repository/deployment/server/synapse-configs/default/sequences` directory.

An example synapse configuration of a global extension sequence is given below:

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="WSO2AM--Ext--In">
<property name="Authentication" expression="get-property('transport',
'Authentication')"/>
<property name="Authorization" expression="get-property('Authentication')"
scope="transport" type="STRING"/>
<property name="Authentication" scope="transport" action="remove" />
</sequence>
```

This custom sequence assigns the value of your basic authentication to Authorization header.

You can copy this content into an XML file (e.g., `global_ext.xml`) and save it in the `<API_Gateway>/repository/deployment/server/synapse-configs/default/sequences` directory.

When you invoke your REST API via a REST Client, configure that client to have a custom header (`Authentication`) for your basic authentication credential and configure the **Authorization** header to contain the bearer token for the API. When you send the Authentication and Authorization headers, the Gateway drops the Authorization header, converts the Authentication to Authorization headers and sends to the backend.

**Class Mediator** is one specific example of mediation extension. When creating a class mediator, we are allowed to write a Java class which extends the `org.apache.synapse.mediators.AbstractMediator` class.

This class implements the `mediate()` function which access the message context and provide the facility to customize the mediation flow of the API. Through that we can read properties of the message context into variables and perform operations.

```
package samples.mediators;

import org.apache.synapse.MessageContext;
import org.apache.synapse.mediators.AbstractMediator;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.soap.SOAPFactory;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import javax.xml.namespace.QName;

public class SimpleClassMediator extends AbstractMediator {

 private String variable1 = xxx;

 private static final Log log =
LogFactory.getLog(SimpleClassMediator.class);

 public SimpleClassMediator(){}

 public boolean mediate(MessageContext mc) {
 // Do somthing useful..
 // Implementation of Reading the property values of
Message context and modifying request / logging properties
 return true;
 }

 public String getType() {
 return null;
 }

 public void setTraceState(int traceState) {
 traceState = 0;
 }

 public int getTraceState() {
 return 0;
 }

 public void setVariable1(String newValue) {
 variable1=newValue;
 }

 public String getVariable1() {
 return variable1;
 }

}
```

Then we can export this class as a jar file and add as a library to <API-M\_HOME>/repository/components/lib directory.

By referring this class with the fully qualified class name in a class mediator in the API as below, we can execute it in the insequence or outsequence of the API globally or per API as described above.

```
<class name="samples.mediators.SimpleClassMediator">
<property name="propertyName" value="PropertyValue"/>
...
</class>
```

If any properties are specified in the java class of the class mediator, the corresponding setter methods are invoked once on the class during initialization.

You can use the Class mediator for user-specific, custom developments only when there is no built-in mediator that already provides the required functionality, because maintaining custom classes incurs a high overhead. Therefore, avoid using them unless the scenario is frequently re-used and very user-specific.

Your class mediator might not be picked up and updated, if you use an existing package when creating it.

## 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 reverse proxy sever rather than the actual 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 it against OAuth2 using the same port that their APIs are on.

Follow the instructions below to configure WSO2 API Manager (WSO2 API-M) with Nginx as a reverse proxy (with a proxy context path):

The following instructions focuses on exposing WSO2 API-M user interfaces, namely the API Store, API Publisher and the API-M Management Console, over NGINX.

1. Install and configure NGINX.
  - a. Install NGINX, if not already done.

```
sudo apt-get install nginx
```

- b. Edit the NGINX server configurations in the /etc/nginx/sites-enabled/default/nginx.conf file.

**Tip:** The location of the NGINX configuration file varies based on the OS that you are using and the installation location of NGINX.

```
sudo vi /etc/nginx/sites-enabled/default/nginx.conf
```

### Example

```
server {

 listen 443;
 ssl on;
 ssl_certificate /etc/nginx/ssl/nginx.crt;
 ssl_certificate_key /etc/nginx/ssl/nginx.key;
 location /apimanager/carbon {
 index index.html;
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
 $proxy_add_x_forwarded_for;
 proxy_pass https://localhost:9443/carbon/;
 proxy_redirect https://localhost:9443/carbon/
https://localhost/apimanager/carbon/;
 proxy_cookie_path / /apimanager/carbon/;
 }

 location ~
^/apimanager/store/(.*)registry/resource/_system/governance/apim
gt/applicationdata/icons/(.*)$ {
 index index.html;
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
 $proxy_add_x_forwarded_for;
 proxy_pass
https://127.0.0.1:9443/$1registry/resource/_system/governance/ap
imgt/applicationdata/icons/$2;
 }

 location ~
^/apimanager/publisher/(.*)registry/resource/_system/governance/
apimgt/applicationdata/icons/(.*)$ {
 index index.html;
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
 $proxy_add_x_forwarded_for;
 proxy_pass
https://127.0.0.1:9443/$1registry/resource/_system/governance/ap
imgt/applicationdata/icons/$2;
 }

 location /apimanager/publisher {
```

```
 index index.html;
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
 proxy_pass https://localhost:9443/publisher;
 proxy_redirect https://localhost:9443/publisher
https://localhost/apimanager/publisher;
 proxy_cookie_path /publisher /apimanager/publisher;

}

location /apimanager/store {
 index index.html;
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
 proxy_pass https://localhost:9443/store;
 proxy_redirect https://localhost:9443/store
https://localhost/apimanager/store;
```

```

 proxy_cookie_path /store /apimanager/store;
 }
}

```

## 2. Secure NGINX.

- Create a SSL certificate and copy it to the `ssl` folder.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/ssl/nginx.key -out /etc/nginx/ssl/nginx.crt
```

- Copy the SSL certificate (`.crt` file) to the `<APIM_HOME>/repository/resources/security` directory.

```
cp /etc/nginx/ssl/nginx.crt ./nginx.crt
```

- Add the SSL certificate to your client trust store. You do this to enable external API publishing and web service calls.

```
keytool -import -file nginx.crt -keystore client-truststore.jks -storepass wso2carbon -alias wso2carbon2
```

## 3. Start NGINX.

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

If you need to stop NGINX, run the following command:

```
sudo /etc/init.d/nginx stop
```

## 4. Configure WSO2 API Manager.

- Edit the `<APIM_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json` file with the context and request URL as shown below. This is done to configure the reverse proxy server for WSO2 API Store, so that you can route the requests that come to the store through a proxy server.

```

"reverseProxy" : {
 "enabled" : true,
 "host" : "localhost", // If the reverse proxy does not
have a domain name use the IP
 "context" : "/apimanager/store",
 "regContext": "" // Use this only if a different path is
used for the registry
}

```

- b. Edit the <APIM\_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/site.json file with the context and request URL as shown below. This is done to configure the reverse proxy server for WSO2 API Publisher, so that you can route the requests that come to the publisher through a proxy server.

```

"reverseProxy" : {
 "enabled" : true,
 "host" : "localhost", // If the reverse proxy does not
have a domain name use the IP
 "context" : "/apimanager/publisher",
 "regContext": "" // Use this only if a different path is
used for the registry
}

```

- c. Update the <APIM\_HOME>/repository/conf/carbon.xml file by uncommenting and updating the values of the following properties.

The HostName and the MgtHostName that you specify here will be the host name of the webapp (publisher and store) URL of API Manager that you configure. In this example we use **localhost** as the hostname of API Manager. If you are using a different hostName, make sure the value that you give for these two properties should match the value that you gave for the host property in the previous two steps.

```

<HostName>localhost</HostName>
<MgtHostName>localhost</MgtHostName>

```

- d. Change the value of KeyValidatorClientType to WSClient in the <APIM\_HOME>/repository/conf/api-manager.xml file.

You need to make this change when you change the value of the host, because requests that are made to the Key Manager will also start getting routed through the reverse proxy; therefore, this needs to be over HTTP instead of TCP, which is Thrifts underlying protocol.

```
<KeyValidatorClientType>WSClient</KeyValidatorClientType>
```

5. Start WSO2 API Manager.

**Linux/Mac OS**

**Windows**

```
cd <APIM_HOME>/bin
./wso2server.sh
```

```
cd <APIM_HOME>\bin
./wso2server.bat
```

If you set up the reverse proxy server correctly, when you access the following URLs the following redirections will take place:

Link Accessed	Redirected To
<a href="https://localhost/apimanager/store">https://localhost/apimanager/store</a>	WSO2 API Store
<a href="https://localhost/apimanager/publisher">https://localhost/apimanager/publisher</a>	WSO2 API Publisher

If you want to change all the default WSO2 API Manager ports, you can do so by editing the `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` file.

## Writing Custom Grant Types

OAuth 2.0 authorization servers provide support for four main grant types according to the OAuth 2.0 specification. They also allow you to add custom grant types and extend the existing ones.

See [Writing a Custom OAuth 2.0 Grant Type](#) in the WSO2 identity Server documentation to implement custom grant types for the API Manager. Note that API Manager has already customized the Grant Type handlers for `authorization_code`, `password`, `client_credentials` and `saml2-bearer` grant types. If you require any additional functionality for these grant types, its advisable to extend the following grant handler implementations.

Grant Type	Existing Handler Class (which can be extended if required)
<code>authorization_code</code>	<code>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedAuthorizationCodeHandler</code>
<code>password</code>	<code>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedPasswordGrantHandler</code>
<code>client_credentials</code>	<code>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedClientCredentialsHandler</code>
<code>urn:ietf:params:oauth:grant-type:saml2-bearer</code>	<code>org.wso2.carbon.apimgt.keymgt.handlers.ExtendedSAML2BearerHandler</code>

**Are you using WSO2 Identity Server 5.3.0 as the Key Manager?** If so, be sure to install [WSO2 Identity Server 5.3.0](#).

## Extending Key Validation

In WSO2 API Manager (WSO2 API-M) versions prior to 1.9.0, the components were tightly coupled with the [Key Manager](#) and token validation was done by directly accessing the databases. However, from WSO2 API-M 1.9.0 onwards, you can plug different [OAuth2 providers](#) to the key validation. When you call an API providing an access token, the execution flows through the handlers specified in the API. Among them, the API authentication handler extracts the token from the header and calls `APIKeyValidationService` to get the token validated. Upon validating the token, the API Gateway receives `APIKeyValidationInfoDTO` as the response, using which the rest of the operations are performed.

Before decoupling was done, the entire key validation process was executed inside a single method named `validateKey()`, which performed all the operations by running a single query. After decoupling, that single query was broken down into smaller parts by introducing `KeyValidationHandler`, which runs inside the `validateKey()` operation, providing a way to extend each step.

The `KeyValidationHandler` has four main operations that are executed in the following order:

- **validateToken** - Validates the token. The existing implementation should work for most cases.
- **validateSubscription** - Skips/changes the domain validation.
- **validateScopes** - Relaxes/reduces scope restrictions.
- **GenerateConsumerToken** - Creates different types of tokens.

The default implementation of the `KeyValidationService` is written in a way where you are able to complete the entire key validation flow only by extending the `getTokenMetaData()` method in the `KeyManagerInterface`.

However, there are situations where you need to customize the default key validation flow according to different requirements. In such situations, WSO2 API-M provides the facility to extend the `KeyValidationHandler` and its methods.

A few examples are listed below.

Requirement	Extension
You need to skip trivial steps, because its validation does not add value.	<p>When creating a key via the API Store, the subscriber can specify which domains are allowed to make calls using a token granted against a particular consumer key. If this validation does not add any value, these trivial steps can be ignored and skipped by extending the <code>KeyValidationHandler</code>.</p> <div data-bbox="556 952 1486 1072" style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>For another example scenario, see <a href="#">Skipping Role Validation for Scopes</a>.</p> </div>
You need to avoid going into detail when validating scopes	<p>Consider a situation where a scope is assigned to a resource and you only need to verify if the token that is used to access the API has at least one or more scopes defined for that API without going into much detail. This requirement can be achieved by extending the <code>validateScope()</code> method.</p>
You need to send a different type of token instead of JSON Web Token (JWT) to pass details of API invocation to the backend	<p>A JSON Web Token (JWT) is used to pass details of an API invocation to the backend. If a different type of token is required, you can extend the <code>generateConsumerToken()</code> method to achieve this purpose.</p>

## Extending the Key Manager Interface

In a typical WSO2 API Manager (WSO2 API-M) deployment, different components talk to the `KeyManager` interface to achieve different tasks. For instance -

- After creating an application in API store, subscribers would click on the generate button to register an application. At this point, the API store talks to the `KeyManager` to create an OAuth client and get the Consumer Key/Secret and the Application Access token.
- When the Gateway receives a request, it talks to `KeyManager` and get the token validated. The `KeyManager` checks if the token is active, and whether the token is usable to invoke the resource being accessed. If the token is valid, the `KeyManager` sends additional details about the token (i.e., the Throttling Tier for the subscription and Consumer key) to the Gateway in the response. In turn the Gateway uses these details to determine if the request should be passed to the backend or not.

Therefore, the `KeyManager` interface acts as the bridge between the OAuth Provider and WSO2 API Manager (WSO2 API-M).

Implement the `KeyManager` interface, which is a Java extension point in WSO2 API-M, when you are writing your own implementation to plug an external OAuth2 authorization server, which will act as the Key Manager. For this purpose uncomment and update the API Key Manager details under the `<APIKeyManager>` element and specify the custom class implementation under the `<KeyManagerClientImpl>` element.

```
<KeyManagerClientImpl>org.wso2.carbon.mit.OpenIDClientImpl</KeyManagerClientImpl>
```

The following are the methods that the `KeyManager` interface uses to carry out operations.

- `createApplication` - Creates a new OAuth application in the Authorization Server.
- `updateApplication` - Updates an OAuth application.
- `retrieveApplication` - Retrieves an OAuth application.
- `getNewApplicationAccessToken` - The Store calls this method to get a new application Access Token. This method is called when getting the token for the first time and when the Store needs to refresh the existing token.
- `getTokenMetaData` - Gets details about an access token.
- `getKeyManagerConfiguration` - Gets Key Manager implementation from `api-manager.xml` file.
- `buildAccessTokenRequestFromJSON` - This method will parse the JSON input and add those additional values to the Access Token Request. If it is needed to pass parameters in addition to those specified in the `AccessTokenRequest`, those parameters can be provided in the JSON input.
- `mapOAuthApplication` - You need to use this method when creating an OAuth application in semi-manual mode when you have a consumer key and secret already generated from a Key Manager and you need to map the key and secret with the existing API-M application.
- `buildAccessTokenRequestFromOAuthApp` - This method creates an Access Token Request using the OAuth Application information. If the token request is null, this method creates a new object, else it modifies the provided Access Token request.
- `loadConfiguration`
- `registerNewResource` - This method talks to the `APIResource` registration endpoint of the authorization server and creates a new resource.
- `getResourceByApiId` - This method retrieves the registered resource by the given API ID.
- `updateRegisteredResource` - This method contains information about all the API resource by its `resourceId`.
- `deleteRegisteredResourceByAPIId` - Deletes the registered resource based on the API ID.
- `deleteMappedApplication` - Deletes mapping records of OAuth applications.
- `getActiveTokensByConsumerKey` - Provides all the Active tokens issued against the provided Consumer Key.
- `getAccessTokenByConsumerKey` - Provides details of the Access Token that is displayed on the Store.

## Adding a New API Store Theme

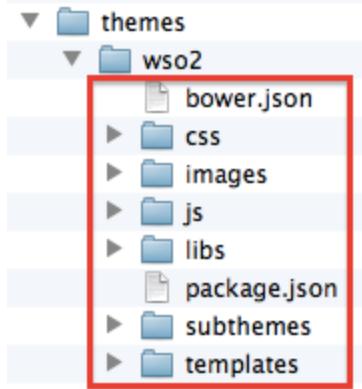
A **theme** consists of UI elements such as logos, images, copyrights messages, landing page text, background colors etc. WSO2 API Store comes with a default theme. You can extend the existing theme by writing a new one or customising the existing one.

### Folder structure of the API Store themes

The default theme of the API Store is named `wso2`. You find it inside the `<API-M_HOME>/repository/deployment`.

folder. If you do not have access to the file system, 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 that is named after 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 `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes` directory
- A **sub theme** is saved inside the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/<main-theme-directory>/subtheme` directory.

As 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 that you add inside the sub-theme overrides the corresponding files in the main theme. The rest of the files are inherited from the main theme.

#### **Tip : How to customize a theme**

Themes are located in the `<API-M_HOME>/repository/deployment/server/jaggeryapps/store/site/themes` folder. There are separate folders for each theme, typically by the name of the theme (e.g., `wso2`), inside the `themes` folder. In addition, there are CSS folders, which contain the CSS files of those themes, inside the individual theme folders. If you need to customize an existing theme, you need to change the corresponding CSS files.

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

#### ***Writing a sub theme of the main theme***

As a main theme already has most of the UIs, 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 refer to this folder as `<THEME_HOME>`.
2. Make any changes you want to the theme. For example, make the following changes in the CSS styles in the `<THEME_HOME>/css/custom.css` file using a text editor and save.
  - Add the following code to change the color of the header to red.

```
header.header-default{
 background:red !important;
}
```

- Update the color given for the search box to #0be2e2.

```
.search-wrap>.form-control, .search-wrap .btn.wrap-input-right {
 background-color: #0be2e2;
 border: 0px;
 color: #FFF;
 height: 40px;
 margin-top:-3px;
}
```

The custom.css file related to the sub-theme should always have the entire set of CSS styles. Therefore, do not delete the code related to the CSS styles that you have not changed in the <API-M\_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/<main-theme-directory>/subtheme/<sub-theme>/css/custom.css file.

3. As you plan to upload the theme 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 are automatically applied from the main theme.

#### **Setting the new theme as the default theme**

The following are the two methods in which you can set your new theme as the default theme:

- Saving directly in the file system
- Uploading through the Admin Portal

#### **Saving directly in the file system**

If you have access to the file system, do the following:

1. Save the <THEME\_HOME> folder that contains the sub-theme of the main theme inside the <APIM\_HOME>/repository/deployment/server/jaggeryapps/store/site/themes/wso2/subthemes folder. This makes your new theme a sub-theme of wso2.
2. Open the <API-M\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file, and add the following code to it. It specifies the base theme as wso2, which is overridden by the sub-theme ancient.

```
"theme" : {
 "base" : "wso2",
 "subtheme" : "ancient"
}
```

3. Open the API Store.  
Note the new theme that is applied to it.

## Uploading through the Admin Portal (Tenants Only)

If you do not have access to the file system, you can upload the theme through the Admin Portal as shown below:

1. Navigate inside the <THEME\_HOME> folder that contains the sub-theme of the main theme, select all the folders inside it, and right-click to compress all the selected files and folders. Then rename the ZIP file based on the name of your sub-theme. For this example use the `ancient.zip` file.
2. Sign in to the WSO2 Admin Portal (<https://<server-host>:9443/admin>) with your tenant username (format `<username>@<domain>.com` `kim@testorg.com`) and password.
3. Expand the **Settings** menu, click **Upload Tenant Theme** and upload your ZIP file.

4. Access the API Store (<https://<server-host>:9443/store>) using your tenant username and password.  
Note the new theme that is applied.

## Extending Scope Validation

OAuth scopes, which were introduced from WSO2 API Manager 1.7.0 onwards, allow you to have fine grained access control to API resources based on the user roles. It allows you to define scopes per API and associate defined scopes with API resources. OAuth 2.0 bearer tokens are obtained for a set of requested scopes and the token obtained is not allowed to access any API resources beyond the associated scopes. For more information, see [OAuth Scopes](#).

API manager uses scopes as a way of defining permissions for a resource. If a resource is assigned a scope, then the token accessing the resource should be generated with that scope. By associating a scope with a role, you can control which users are permitted to have tokens under certain scopes. In this instance, associating a role to a scope seems legitimate.

Validating the role of a requester does not make much sense in some scenarios. For instance, when the scope is used as a means of generating an access token and not for securing a resource (e.g. openid scope). In such

situations, scope validation can be extended to skip role validation for certain scopes.

#### **Skipping role validation for scopes**

When scopes which cannot be associated to roles are requested, the token should be issued without validating the scope. In WSO2 API Manager, you do this by [whitelisting the scope](#) through configuration. Patterns of the whitelisted scopes are specified via a configuration under the `<OAuthConfigurations>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file. Scopes that match the pattern are not validated by role and are available to anyone requesting it.

The following steps show a demonstration:

1. Start the API Manager server and log into the API Store.
2. Create an application. On the **Production Keys** tab of your application, click **Generate Keys**.
3. Obtain the **Base64** encoded representation of the Consumer Key and the Consumer Secret separated by a colon according to the following format.

Base64 ( consumer-key : consumer-secret )

You can also use the curl request listed under the **Generate Access Tokens** section for the steps 3 and 4 based on the grant type.

The screenshot shows the 'Production Keys' tab of an application's configuration page. The tab is highlighted with a red border. Below the tabs, there is a 'Show Keys' button. Under the 'Consumer Key' section, there is a redacted text field. Under the 'Consumer Secret' section, there is another redacted text field. Below these sections is a 'Grant Types' section containing several checkboxes for different authentication methods: SAML2, IWA-NTLM, Client Credential, Code, Implicit, Password, and Refresh Token. Most of these checkboxes are checked. At the bottom of the page is an 'Update' button.

4. Use the Base64 encoded value obtained above in the Authorization header when invoking the following command. This is used to get the token by calling the token API.

Make sure you include a random scope in the request which will be any value suitable for the name of the scope.

```
curl -k -d
"grant_type=password&username=admin&password=admin&scope=some_random_scope"
-H "Authorization: Basic WmRFUFBvZmZwYVFnR25ScG5izldtcUtSS3IwYTpSaG5ocEVJYUVCMEN3T1FReWpiZTJwaDBzclVh"
-H "Content-Type: application/x-www-form-urlencoded"
https://10.100.0.3:8243/token
```

Along with the token, you receive a response from the server similar to the one below.

```
{ "scope": "default", "token_type": "bearer", "expires_in": 3600, "refresh_token": "23fac44e9b7e1ae95a33b85f4f26decd", "access_token": "9474fa104ccb196303f41c8a5ee6f48" }
```

You may not see the scope you requested for in this response as it has not been whitelisted yet.

5. Shut down the server.
6. To whitelist the scope, add the following under the `<OAuthConfigurations>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file and restart the server.

```
<ScopeWhitelist>
<Scope>^device_.*</Scope>
<Scope>some_random_scope</Scope>
</ScopeWhitelist>
```

7. Call the token API using the same request used in step 4. You will receive a response similar to the one below.

```
{ "scope": "some_random_scope", "token_type": "bearer", "expires_in": 3600, "refresh_token": "59e6676db0addca46e68991e44f2b8b8", "access_token": "48855d444db883171c347fa21ba77e8" }
```

You see a successful response along with the whitelisted scope for which you requested.

## Extending the API Life Cycle

APIs created in WSO2 API Manager have their own life cycle consisting of the following: a set of life cycle states, specific actions for each state transition, and a checklist of items before a state transition occurs. In previous API Manager versions, an API had a predefined life cycle consists of [six states](#) which could not be customized or extended. From API Manager 1.10.0 onwards, you can extend the API life cycle with the WSO2 registry based life cycle in API Manager.

- [Default API Lifecycle in WSO2 API Manager](#)
- [Extension Points of API Lifecycle](#)

### Default API Lifecycle in WSO2 API Manager

The WSO2 registry based life cycle provides a configurable way to define the life cycle of an artifact, which can be extended easily, as the default API life cycle is defined as an XML configuration.

Note that this extending capability of the API life cycle is not available in API Manager versions prior to 1.10.0.

To see the default API life cycle configuration, follow the steps below.

1. Start the API Manager server and log into the management console: <https://localhost:9443/carbon>.
2. Navigate to **Extensions > Configure > Lifecycles**.

Lifecycle Name	Actions
APILifeCycle	<a href="#">View/Edit</a> <a href="#">Find Usage</a>
ServiceLifeCycle	<a href="#">View/Edit</a> <a href="#">Delete</a>

[Add New Lifecycle](#) [Find Resources/Collections With Lifecycles](#)

3. Click the **View/Edit** link corresponding to the API LifeCycle. The default API life cycle configuration opens.

```

<aspect name="APILifeCycle"
class="org.wso2.carbon.governance.registry.extensions.aspects.Default
LifeCycle">
 <configuration type="literal">
 <lifecycle>
 <scxml xmlns="http://www.w3.org/2005/07/scxml"
version="1.0"
initialstate="Created">
 <state id="Created">
 <datamodel>
 <data name="checkItems">
 <item name="Deprecate Old Versions"
forEvent="">
 </item>
 <item name="Require Re-Subscription"
forEvent="">
 </item>
 </data>
 <data name="transitionExecution">
 <execution forEvent="Deploy as a
Prototype"
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 <execution forEvent="Publish"
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 </data>
 </datamodel>
 <transition event="Publish" target="Published"/>
 <transition event="Deploy as a Prototype"
target="Prototyped"/>
 </state>
 </transitionExecution>
 </data>
 </datamodel>
 </state>
 </lifecycle>
 </configuration>
 </aspect>

```

```

<state id="Prototyped">
 <datamodel>
 <data name="transitionExecution">
 <execution forEvent="Publish"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 <execution forEvent="Demote to Created"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 </data>
 </datamodel>
 <transition event="Publish" target="Published"/>
 <transition event="Demote to Created"
target="Created"/>
 </state>

 <state id="Published">
 <datamodel>
 <data name="transitionExecution">
 <execution forEvent="Block"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 <execution forEvent="Deprecate"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 <execution forEvent="Demote to Created"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 <execution forEvent="Demote to Prototyped"

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 </data>
 </datamodel>
 <transition event="Block" target="Blocked"/>
 <transition event="Demote to Prototyped"
target="Prototyped"/>
 <transition event="Demote to Created"
target="Created"/>
 <transition event="Deprecate"
target="Deprecated"/>
 <transition event="Publish" target="Published"/>
 </state>
 <state id="Blocked">
 <datamodel>
 <data name="transitionExecution">
 <execution forEvent="Re-Publish"

```

```
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 <execution forEvent="Deprecate">

class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 </data>
 </datamodel>
 <transition event="Deprecate"
target="Deprecated"/>
 <transition event="Re-Publish"
target="Published"/>
 </state>
 <state id="Deprecated">
 <datamodel>
 <data name="transitionExecution">
 <execution forEvent="Retire">

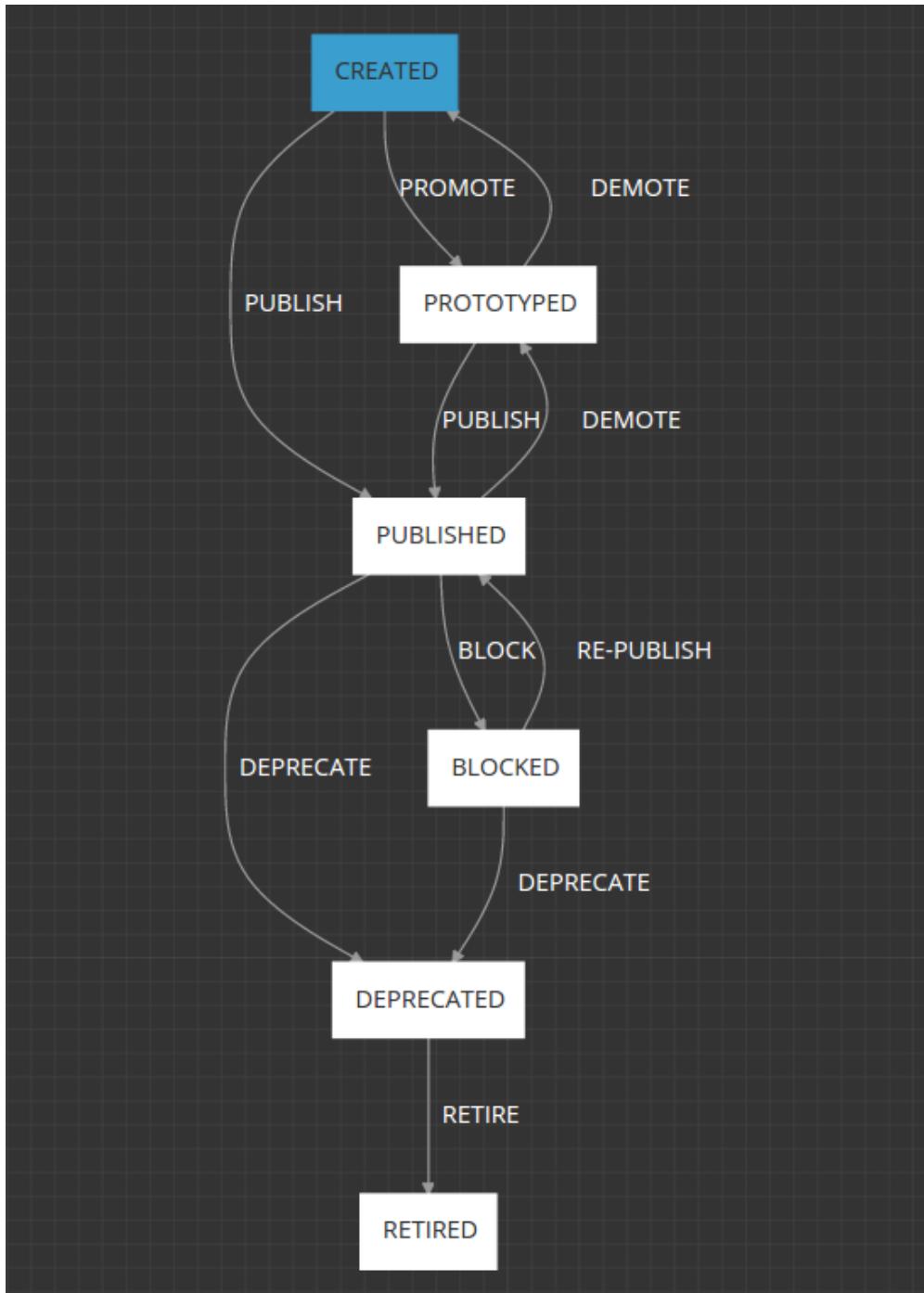
class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 </data>
 </datamodel>
 <transition event="Retire" target="Retired"/>
</state>
<state id="Retired">
</state>
</scxml>
```

```
</lifecycle>
</configuration>
</aspect>
```

The above configuration includes the following important information:

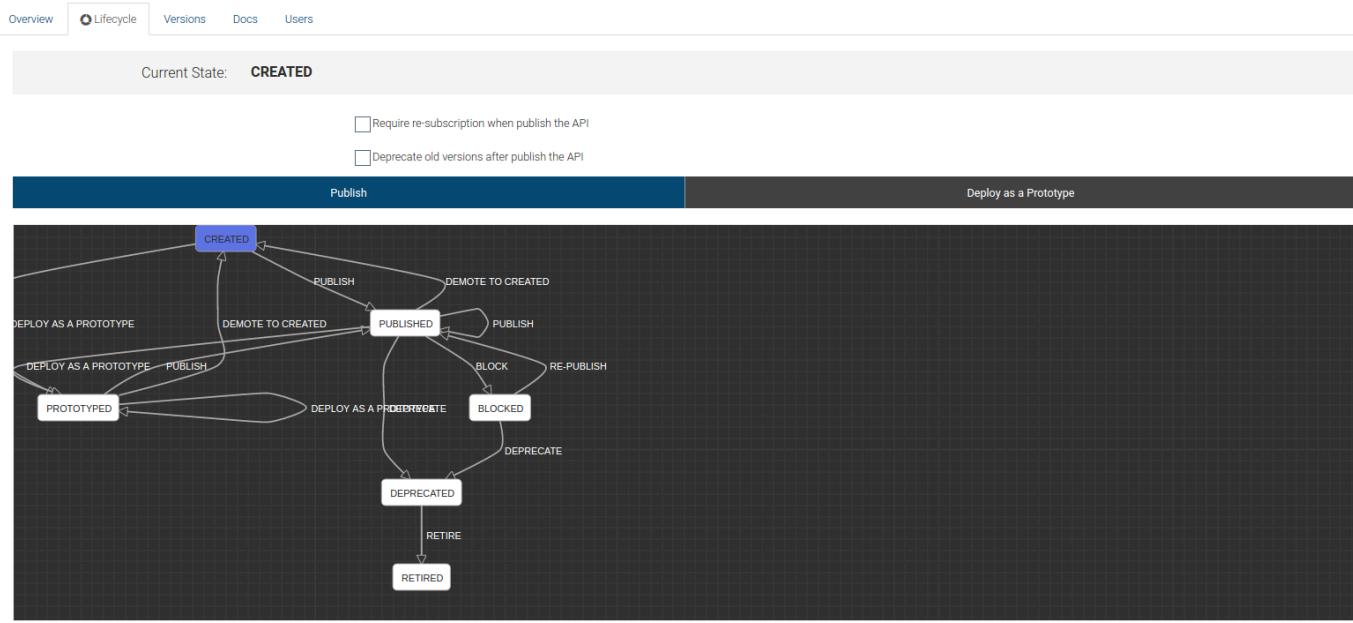
1. Lifecycle name: APILifeCycle
2. Set of six default states: **CREATED, PROTOTYPED, PUBLISHED, BLOCKED, DEPRECATED, RETIRED**
3. Set of checklist items to be satisfied: For example, when the API is in the **CREATED** state and has multiple versions, there are two checks that occur; deprecate old versions and re-subscriptions required.
4. State transition events: Defines from which state to which target state an API can be moved.
5. Actions for each state transition: A triggered action that executes during each state transition. For example, when an API state changes from **CREATED** to **PUBLISHED**, an execution occurs as a relative synapse API where an XML element is created and the related API data is saved in the database. This execution is defined for each state transition in the above registry life cycle configuration.

The state transition events that occur in the default API life cycle is shown in the following diagram:

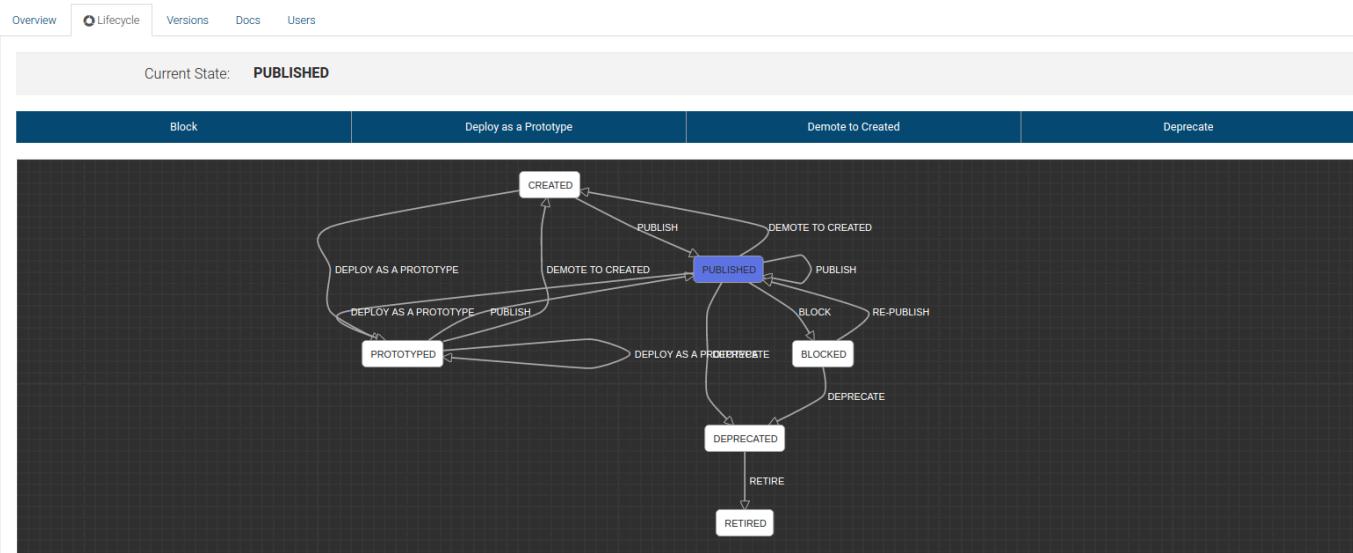


The **Lifecycle** tab shows the current state of an API, the target events defined in the API life cycle for that state, and the set of checklist items.

## PizzaShackAPI - 2.0.0



## PizzaShackAPI - 1.0.0



This UI is static in the default stage and is dynamically generated based on the defined API life cycle in the above XML configuration.

If you customize the default API life cycle configuration including states, transition events or check list items, those changes are updated in the life cycle of the Publisher UI accordingly.

### Extension Points of API Lifecycle

With the integration of the registry life cycle to the API life cycle of WSO2 API Manager, it is possible to extend the existing API life cycle and customize it according to your preference. Following are some extention points where the default API life cycle can be extended by modifying above mentioned XML configuration of the API life cycle.

Consider the following points when extending and customising the API life cycle XML configuration.

- Do not change the life cycle name since it needs to be engaged with the APIs dynamically.
- Make sure you keep the **PUBLISHED** and **PROTOTYPED** states as those two states will be used by API Publisher in the API creation wizard.

Following are some extension points that can be used:

- Define your own life cycle states in the API life cycle
- Change the state transition events as per the environmental preferences
- Add custom checklist items for specific state transitions
- Change the execution code for each state transition

For all state transitions, the same execution class is used ([org.wso2.carbon.apimgt.impl.executors.APIExecutor](#)). However, you can plug your own execution code when modifying the life cycle configuration. For example, if you want to add notifications for a specific state transition, you can plug your own custom execution class for that particular state in the API life cycle. Any changes are updated in the **Lifecycle** tab accordingly.

When a new transition event is introduced to the life cycle, an entry must be made to the `locale_default.json` file in order to view that life cycle transition event in the Publisher **Lifecycle** tab. This is introduced to support multi-language facility. For example, let's say a transition event called **Notify Users** is introduced in the **DEPRECATED** state as follows,

```
<state id="Deprecated">
 <datamodel>
 <data name="transitionExecution">
 <execution forEvent="Retire"
 class="org.wso2.carbon.apimgt.impl.executors.APIExecutor">
 </execution>
 </data>
 </datamodel>
 <transition event="Retire" target="Retired"/>
 <transition event="Notify Users" target="Retired"/>
 </state>
```

You need to add "notify users" : "Notify Users" as an entry in the `<APIM_HOME>/repository/deployment/server/jaggeryapps/publisher/site/conf/locales/jaggery/locale_default.json` file. Note that the key value in this entry should be in lower case (e.g. notify users).

For other languages , add the entry to the relevant file. For further information, see [Adding Internationalization and Localization](#).

## Customize the API Store and Gateway URLs for Tenants

The default URL of WSO2 API Manager Store is `https://<HostName>:9443/store`. You can change the URL of the Gateways and API Store tenants in WSO2 API Manager as follows:

- Install Nginx and create SSL certificates
- Setup custom domain mapping in the registry
- Configure the store webapp

### Install Nginx and create SSL certificates

Follow the steps below to install Nginx and create SSL certificates

1. Run the following command and install Nginx, if not already available.

```
sudo apt-get install nginx
```

2. Create an SSL certificate.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/nginx/ssl/nginx.key -out /etc/nginx/ssl/nginx.crt
```

3. Go to <APIM\_HOME>/repository/resources/security and use the following command to add the certificate to the client trust store.

```
keytool -import -file /etc/nginx/ssl/nginx.crt -keystore
client-truststore.jks -storepass wso2carbon -alias wso2carbon2
```

4. Open /etc/nginx/sites-enabled/default in your terminal and add the following configurations with your custom domain name.

```
server {
 listen 443;
 ssl on;
 ssl_certificate /etc/nginx/ssl/nginx.crt;
 ssl_certificate_key /etc/nginx/ssl/nginx.key;
 location / {
 proxy_set_header X-WSO2-Tenant "ten5.com";
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
 proxy_set_header Host $http_host;
 proxy_pass https://localhost:9443/store/;
 proxy_redirect https://localhost:9443/store/ /;
 proxy_redirect <custom URL>;
 proxy_cookie_path /store/ /;
 }
}
```

## Install nginx in Mac OS

If you are using Mac OS, you need to install nginx using [brew](#) package manager. The commands are as follows.

- Command to install nginx

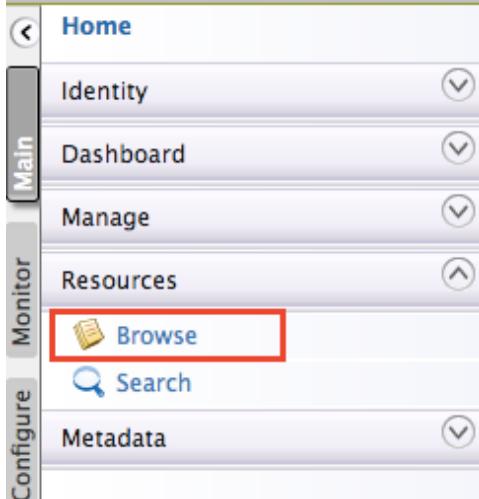
```
brew install nginx
```

- Command to run nginx

```
sudo nginx
```

## Setup custom domain mapping in the registry

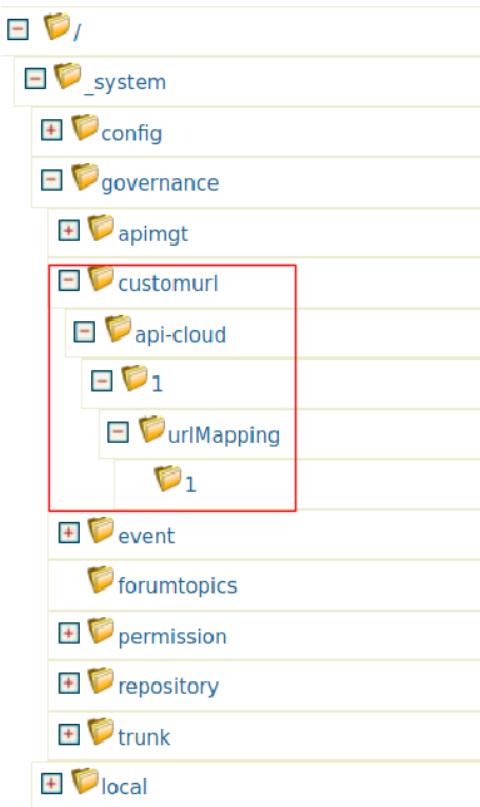
1. Log in to the management console as (<https://<HostName>:9443/carbon>) as admin (or tenant admin).
2. In the **Main** menu, click **Browse** under **Resources**.



3. Navigate to /\_system/governance registry path and create the following directory structure in the registry.

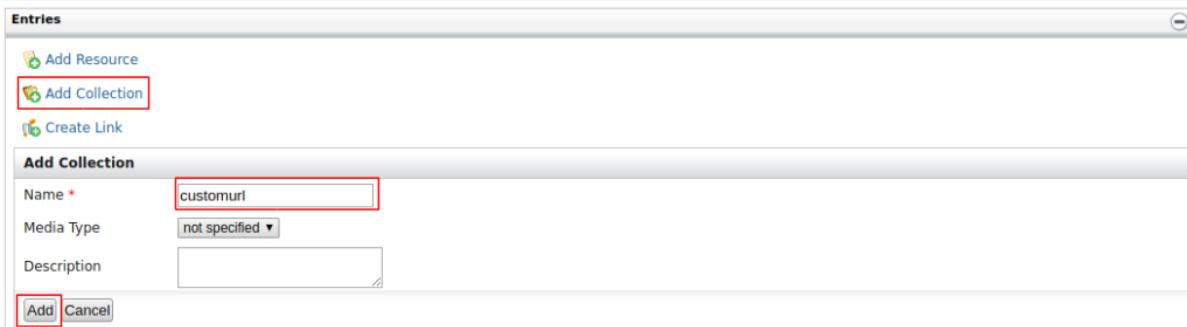
In **API Cloud** this directory structure is created automatically when you are specifying the custom URL through UI.

customurl/api-cloud/<tenant-id>/urlMapping/<tenant-id>.

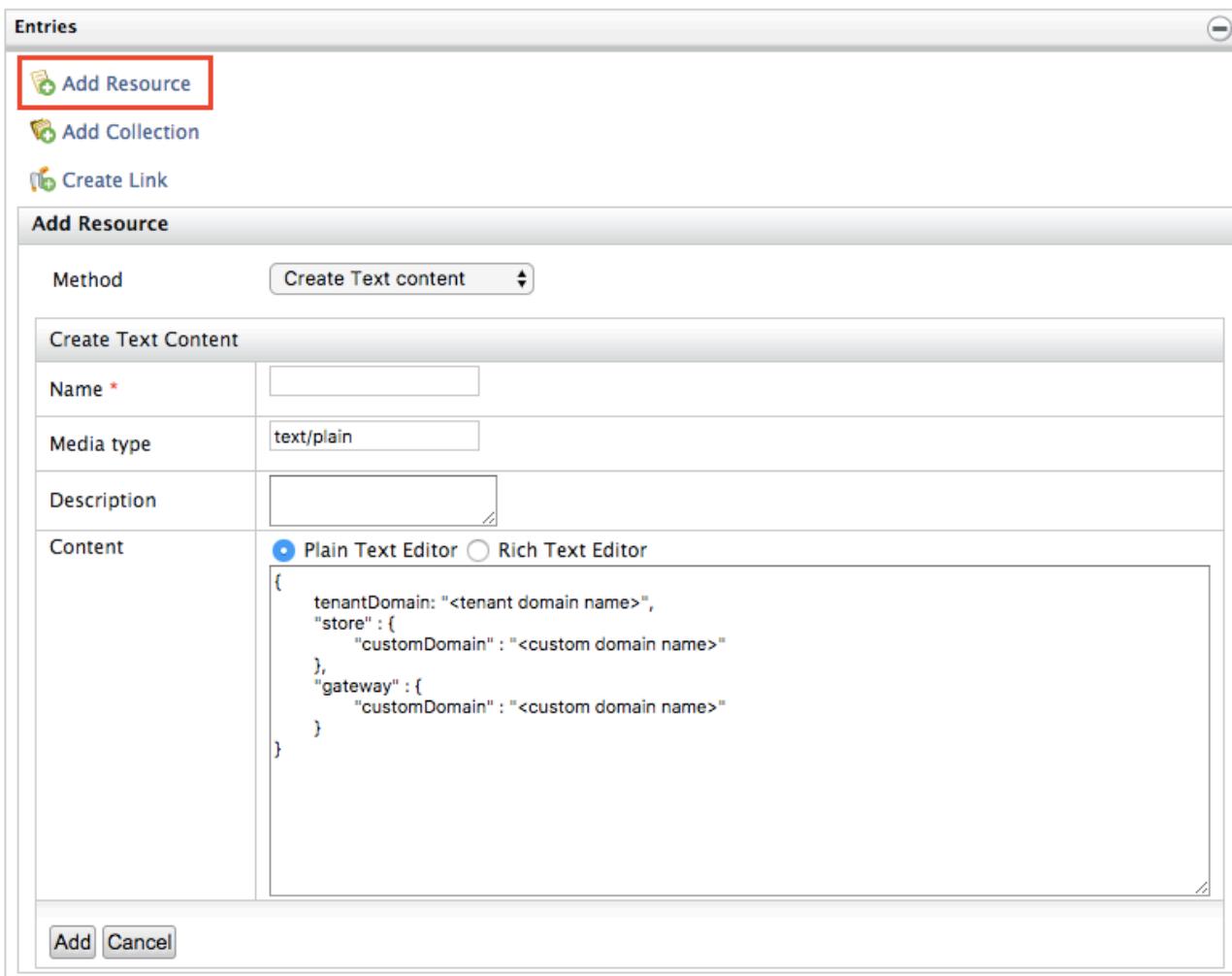


To create a directory in the registry path,

1. navigate to the location in the registry browser, click and open the location.
2. Click **Add Collection** and specify the name of the directory and click **Add**.



4. Navigate to `/_system/governance/customurl/api-cloud/<tenant-id>/urlMapping/<tenant-id>`. Click **Add Resources** under **Entries** and select **Create Text Content**.



- Add the following resource configurations to the registry and click **Add**.

```
{
 "tenantDomain": "<tenant domain name>",
 "store" : {
 "customDomain" : "<custom domain for store>"
 },
 "gateway" : {
 "customDomain" : "<custom domain for gateway>"
 }
}
```

#### Configure the store webapp

- Go to <APIM\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf directory, open the site.json file and add the tenant header parameter as shown below.

```

"reverseProxy" : {
 "enabled" : "auto",
 "host" : "sample.proxydomain.com",
 "context" : "",
 "tenantHeader" : "X-WSO2-Tenant"
},

```

You can choose any name for the header and set the virtual host to create the specific domain.

For details on how to create and manage multiple tenants, see [Managing Tenants](#). You can also see [Multi-tenant Architecture](#) for more information about tenants.

## Editing API Templates

Each API in API manager is represented by an XML file. The elements of this XML file and their attributes are defined in `<APIM_HOME>/repository/resources/api_templates/velocity_template.xml`, which is the default API template that comes with the API Manager. By editing the default template definitions, you can 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.

## Customizing Login Pages for API Store and API Publisher

Custom pages for logging into the server are available for SAML2 SSO, OAuth and OpenID. This section guides you through this customization.

The login pages and other pages like error and notification screens of SAML SSO, OAuth, OpenID and Passive STS are located in the authenticationendpoint webapp file found at `<APIM_HOME>/repository/deployment/server/webapps`.

You can easily customize these pages within this web application by changing the respective JSPs, JavaScript and CSS. If you want to point to a different web application, you can do so by redirecting or forwarding from **authenticationendpoint** to your webapp. In the case of SAML SSO, the 'issuer' id of the service provider is also sent to this webapp. Therefore, different login pages can be given to different service providers by looking at the 'issuer' request parameter.

The following is a sample of how to customize the login page for SAML2 SSO.

### *Customizing the login page for SAML SSO service providers*

Usually WSO2 API Manager displays a default login page for all the SAML SSO service providers that send authentication requests to it. The following steps indicate how to change the default login page into a customized one.

- Registering the two service providers in the Identity Server
- Configuring the login page

### Registering the two service providers in the Identity Server

1. Download [WSO2 Identity Server](#) and extract it.
2. Run the server by executing `wso2is-5.0.0/bin/wso2server.sh` if on a Unix-based systems, or `/bin/wso2server.bat` if on Windows.
3. On the [management console](#), click Add under Service Providers in the Main menu.
4. Enter "publisher" as the Service Provider Name in the form that appears and click Register.

## Add New Service Provider

**Basic Information**

Service Provider Name: <sup>*</sup>	<input type="text" value="publisher"/>	<small>(?) A unique name for the service provider</small>
Description:	<input type="text" value="publisher service provider"/>	
<input type="button" value="Register"/> <input type="button" value="Cancel"/>		

5. In the page that appears next, expand the **Inbound Authentication Configuration** section and the **SAML2 Web SSO Configuration** section. Click **Configure**. The Register New Service Provider page appears.

**Register New Service Provider**

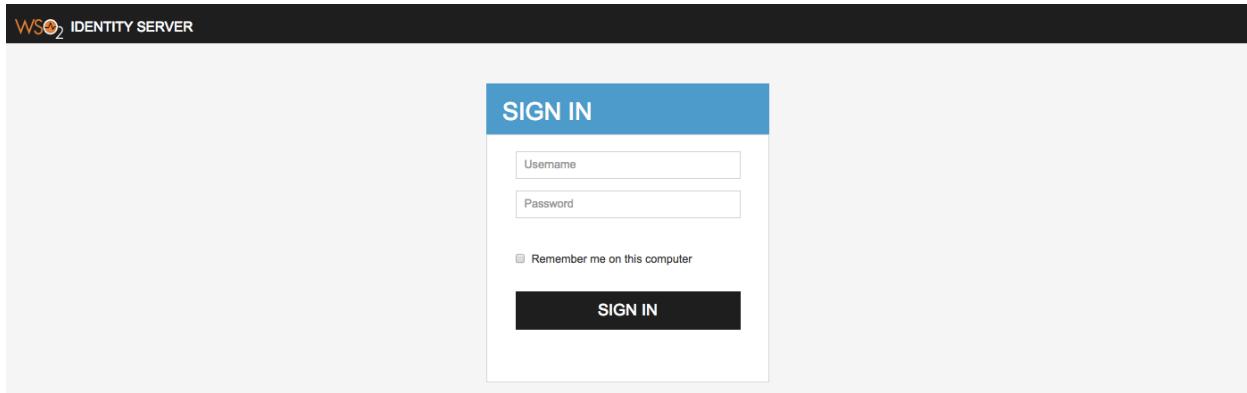
**Edit Service Provider**

Issuer *	<input type="text" value="api_publisher"/>
Assertion Consumer URLs *	<input type="text"/> <a href="https://10.100.5.83:9443/publisher/jagg/jaggery_acs.jag">https://10.100.5.83:9443/publisher/jagg/jaggery_acs.jag</a> <input type="button" value="Add"/> <input type="button" value="Delete"/>
Default Assertion Consumer URL *	<input type="text" value="https://10.100.5.83:9443/publisher/jagg/jaggery_acs.jag"/>
NameID format	<input type="text" value="urn:oasis:names:tc:SAML:1.1:nameid-form"/>
Certificate Alias	<input type="text" value="wso2carbon.cert"/>
Response Signing Algorithm *	<input type="text" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
Response Digest Algorithm *	<input type="text" value="http://www.w3.org/2000/09/xmldsig#sha1"/>
<input checked="" type="checkbox"/> Enable Response Signing <input type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests <input type="checkbox"/> Enable Assertion Encryption <input checked="" type="checkbox"/> Enable Single Logout	
SLO Response URL	<input type="text"/>
SLO Request URL	<input type="text"/>
<input checked="" type="checkbox"/> Enable Attribute Profile <input type="checkbox"/> Include Attributes in the Response Always	
<input type="checkbox"/> Enable Audience Restriction <b>Audience</b> <input type="text"/> <input type="button" value="Add"/>	

Configure the following details for publisher.

- Issuer: api\_publisher
- Assertion Consumer URL: [https://10.100.5.83:9443/publisher/jagg/jaggery\\_acs.jag](https://10.100.5.83:9443/publisher/jagg/jaggery_acs.jag)
- Select Enable Response Signing
- Select Enable Single Logout

6. Repeat steps 1 to 5 and configure the following details for store.
  - Issuer: api\_store
  - Assertion Consumer URL: [https://10.100.5.83:9443/store/jagg/jaggery\\_acs.jag](https://10.100.5.83:9443/store/jagg/jaggery_acs.jag)
  - Select Enable Response Signing
  - Select Enable Single Logout
7. When attempting to login with SAML from WSO2 Identity Server in API publisher and API store, you can see the following default page located at <IS\_HOME>/repository/deployment/server/webapps/authenticationendpoint/login.jsp



For instructions on configuring WSO2 Identity Server as an identity provider, see [Configuring Identity Server as IDP for SSO](#).

## Configuring the login page

### *Understanding the authenticationendpoint web application*

The login page that is displayed during SAML2 SSO, OAuth, OpenID and Passive-STS flows is located inside the webapp named authenticationendpoint. The reason for storing this in a web app is:

- to easily customize the page according to user requirements
- if needed, place that whole web application in an external application server

The Identity Server knows the location of this web application as it is specified in the <IS\_HOME>/repository/conf/identity/application-authentication.xml configuration file. This is referenced as shown below.

```
<AuthenticationEndpointURL>/authenticationendpoint/login.do</Authenticatio
nEndpointURL>
<AuthenticationEndpointRetryURL>/authenticationendpoint/retry.do</Authenti
cationEndpointRetryURL>
```

By default it points to a location inside the Identity Server itself, thus the relative path is given. If it is necessary to point to an external application, the full path should be given instead.

If this web app is moved outside the Identity Server, we must ensure that no one can access the login credentials that are passed between this application and the Identity Server. This means that the external location should ideally be either inside a secured intranet or the transport should be HTTPS. Other similar precautions may be necessary to secure the communication.

The following is the structure of this web app.

Name	Date Modified	Size	Kind
assets	Feb 16, 2016, 3:28 PM	--	Folder
basicauth.jsp	Dec 21, 2015, 7:54 AM	3 KB	JSP source file
create-account.jsp	Dec 21, 2015, 7:54 AM	14 KB	JSP source file
css	Feb 16, 2016, 3:28 PM	--	Folder
domain.jsp	Dec 21, 2015, 7:54 AM	5 KB	JSP source file
fido-auth.jsp	Dec 21, 2015, 7:54 AM	5 KB	JSP source file
fonts	Feb 16, 2016, 3:28 PM	--	Folder
generic-exception-response.jsp	Dec 21, 2015, 7:54 AM	2 KB	JSP source file
images	Feb 16, 2016, 3:28 PM	--	Folder
js	Feb 16, 2016, 3:28 PM	--	Folder
libs	Feb 16, 2016, 3:28 PM	--	Folder
login.jsp	Dec 21, 2015, 7:54 AM	15 KB	JSP source file
logout.jsp	Dec 21, 2015, 7:54 AM	4 KB	JSP source file
META-INF	Feb 16, 2016, 3:28 PM	--	Folder
oauth2_authz.jsp	Dec 21, 2015, 7:54 AM	6 KB	JSP source file
oauth2_consent.jsp	Dec 21, 2015, 7:54 AM	6 KB	JSP source file
oauth2_error.jsp	Dec 21, 2015, 7:54 AM	5 KB	JSP source file
openid_profile.jsp	Dec 21, 2015, 7:54 AM	8 KB	JSP source file
openid.jsp	Dec 21, 2015, 7:54 AM	2 KB	JSP source file
registration.jsp	Dec 21, 2015, 7:54 AM	4 KB	JSP source file
retry.jsp	Dec 21, 2015, 7:54 AM	4 KB	JSP source file
samlso_notification.jsp	Dec 21, 2015, 7:54 AM	2 KB	JSP source file
samlso_redirect.jsp	Dec 21, 2015, 7:54 AM	2 KB	JSP source file
tenant_refresh_endpoint.jsp	Dec 21, 2015, 7:54 AM	1 KB	JSP source file
tenantauth.jsp	Dec 21, 2015, 7:54 AM	6 KB	JSP source file
WEB-INF	Today, 1:41 PM	--	Folder
classes	Today, 1:41 PM	--	Folder
lib	Feb 16, 2016, 3:28 PM	--	Folder
web.xml	Dec 21, 2015, 7:54 AM	8 KB	XML text

The **authenticationendpoint** web application uses a carbon component called `org.wso2.carbon.identity.application.authentication.endpoint.util`. This bundle includes a filter called the `org.wso2.carbon.identity.application.authentication.endpoint.util.filter.AuthenticationEndpointFilter`, which acts as the Front Controller.

When a request is made to the **authenticationendpoint** web application, based on the authentication protocol type identified by the request parameter 'type', the controller first forwards the request to the protocol based login url patterns defined. For example, if the request to the **authenticationendpoint** web application is initiated as a result of a SAML SSO authentication request, the controller will forward the request to the url pattern `/samlso_login.do`. If you look inside the **web.xml**, you will see that this url pattern is mapped to the **login.jsp** file. The request is finally forwarded to this **login.jsp** page.

Everything on the **authenticationendpoint** web application is customizable. You can customize it by adding JSP pages or modifying them and configuring the **web.xml** respectively.

The only restriction involved is that the content already sent back by the pages inside the default web app must be submitted to the Identity Server. Additionally, you must point to the correct location via the `<IS_HOME>/repository/conf/identity/application-authentication.xml` file.

### Customizing the login page

When a request comes to the default login page, you can see several parameters being passed in the address bar. For this customization, the focus is on the following two parameters:

- **sessionDataKey**: This is an identifier used by the Identity Server to maintain state information related to this particular request by the service provider.
- **relyingParty**: This is the value we gave for the "Issuer" field when we registered the SAML2 SSO service provider (e.g., [travelocity.com](#)). This value is used to display different login pages to different service providers.

When customizing the pages, ensure that the following is applied.

1. Form submissions should happen to the "commonauth" servlet as a POST.  
`<form id="form" name="form" action=".../commonauth" method="POST">`
2. Make sure to send back the "sessionDataKey" with the form submission, by using a hidden input field.

```
<input type="hidden" name="sessionDataKey"
value="<%>=request.getParameter("sessionDataKey")%>" />
```

### Using a JSP to redirect to SP relevant pages

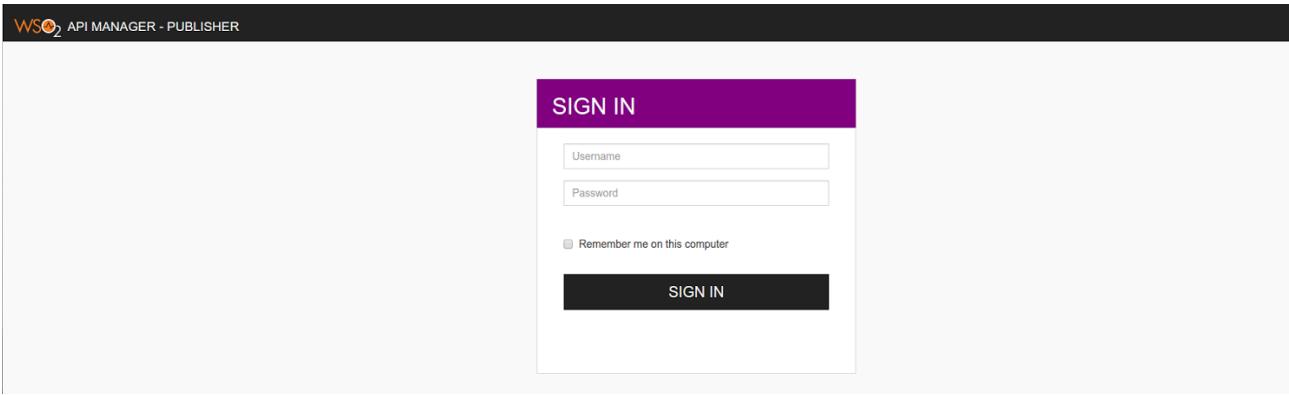
1. Rename the existing 'login.jsp' to 'default\_login.jsp'
2. Create a new file with the name 'login.jsp' including the following code.

```
<%
String relyingParty = request.getParameter("relyingParty");

if (relyingParty.equals("api_publisher")) {
 RequestDispatcher dispatcher =
 request.getRequestDispatcher("publisher_login.jsp");
 dispatcher.forward(request, response);
} else if (relyingParty.equals("api_store")) {
 RequestDispatcher dispatcher =
 request.getRequestDispatcher("store_login.jsp");
 dispatcher.forward(request, response);
}
else {
 RequestDispatcher dispatcher =
 request.getRequestDispatcher("default_login.jsp");
 dispatcher.forward(request, response);
}
%>
```

This code snippet forwards the request to a different login page by checking the value of relyingParty parameter.

3. Get the 'publisher\_login.jsp' from [here](#) and place it at the same level as 'login.jsp'. Also, download the contents of the 'css' folders from that same link and put them inside the respective folders in the authenticationendpoint.
4. Log in to the publisher web app again. You are presented with a different page.

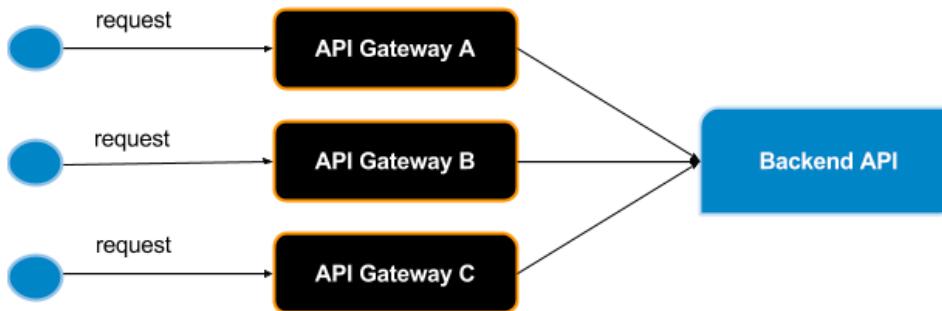


5. Follow steps 1 to 4 to configure the custom login page to the store web app.

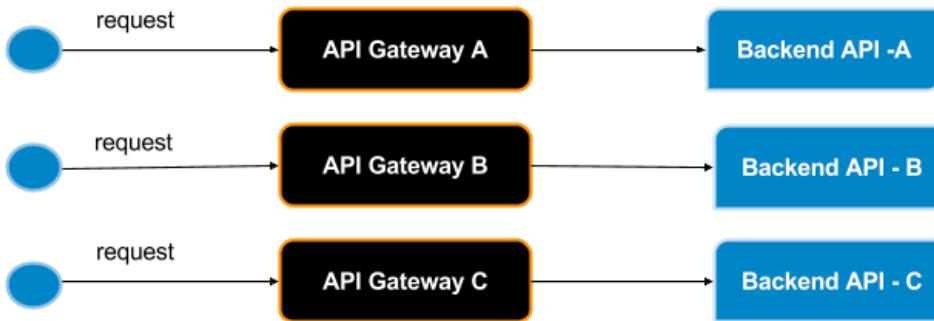
### API Gateways with Dedicated Backends

We can extend the [multiple gateway environments](#) feature by utilizing dynamic endpoint capabilities of WSO2 API Manager to have each gateway point to a different back-end endpoint. API Gateway is the actual runtime of the APIs that are developed and published from the API Publisher. WSO2 API Manager is capable of publishing APIs to different Gateways where API users connect to those API Gateways in order to do the actual API calls through the applications to which they are subscribed.

However, the API Publisher can only provide a single static endpoint for an API in the implementation. Therefore, the API call is directed to a single endpoint in whichever Gateway the API is deployed in, as depicted in the diagram below.



However, in most situations, you would want to have each Gateway proxying to a dedicated backend API. To provide that capability, WSO2 API Manager provides the ability to specify dynamic endpoint URLs at the time of specifying the API endpoint URL. This URL is resolved at runtime with the details (host and port) specified at the startup of each Gateway. Each gateway then points to a dedicated backend API, as depicted in the diagram below.



#### *Configuring dynamic endpoints*

Follow the steps below to configure a dynamic endpoint as the API endpoint.

1. Start the WSO2 API Manager server that includes the API Publisher component and create an API.
2. Go to the **Implement** tab of the API and replace the host and port of the API endpoint with `{uri.var.host}` and `{uri.var.port}` respectively, as shown below.

Managed API  
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type : \* ?

Load Balanced  Failover

Production Endpoint : \* ?    Test

Sandbox Endpoint : \* ?  Test

Show More Options

Message Mediation Policies

Enable Message Mediation  Check to select a message mediation policy to be executed in the message flow

3. Save and [publish](#) the API.
4. Navigate to the <API-M\_HOME>/repository/deployment/server/synapse-configs/sequences directory of each Gateway and create the following sequence.

```
<sequence xmlns="http://ws.apache.org/ns/synapse"
name="WSO2AM--Ext--In">
 <property name="uri.var.host"
expression="get-property('system','host') " />
 <property name="uri.var.port"
expression="get-property('system','port') " />
</sequence>
```

Java system properties are used at the server start-up process of each Gateway to resolve the variables that are defined as properties in this sequence.

Alternatively, you can resolve this host and port using a class mediator. To do that, follow the steps below as an alternative to step 4.

1. Create a java class extending the AbstractMediator class of org.synapse.core as shown below and create the JAR file out of it.

```

import org.apache.synapse.MessageContext;
import org.apache.synapse.mediators.AbstractMediator;

public class EnvironmentResolver extends AbstractMediator {

 @Override
 public boolean mediate(MessageContext messageContext) {

 String host = System.getProperty("environment.host");
 String port = System.getProperty("environment.port");

 messageContext.setProperty("uri.var.host", host);
 messageContext.setProperty("uri.var.port", port);

 return true;
 }

 @Override
 public boolean isContentAware(){
 return false;
 }
}

```

2. Add the created JAR file into the <API-M\_HOME>/repository/components/lib folder of each Gateway. You can download a sample JAR file [here](#).
3. Add the following sequence to the <API-M\_HOME>/repository/deployment/server/synapse-configs/sequences folder of each Gateway.

```

<sequence xmlns="http://ws.apache.org/ns/synapse"
name="WSO2AM--Ext--In">
 <class name="org.wso2.carbon.env.EnvironmentResolver"/>
</sequence>

```

**org.wso2.carbon.env.EnvironmentResolver** is the fully qualified name of the class that contains the code responsible for converting system variables into properties. It is a special class that needs to be extended from the **org.apache.synapse.mediators.AbstractMediator** class and requires overriding of the **mediate** function.

5. Execute the following command when starting up each Gateway to set the system variables at the server start up from within the <API-M\_HOME>/bin directory by replacing the following values.

<ip_of_backend_environment>	<port_of_backend_environment>
host IP of the Gateway	port where the Gateway is running in the dedicated machine or VM

```
./wso2server.sh -Dhost=<ip_of_backend_environment>
-Dport=<port_of_backend_environment>
```

If you have used the class mediator to configure API Gateways in step 4, use the command given below instead of the one above.

```
./wso2server.sh -Denvironment.host=<ip_of_backend_environment>
-Denvironment.port=<port_of_backend_environment>
```

Now the Gateways have started with the dedicated backend host/port combinations.

## 6. Invoke the API.

You receive the response from the API, which is sent through the dedicated backend, from the Gateway that this API is published.

## Securing OAuth Token with HMAC Validation

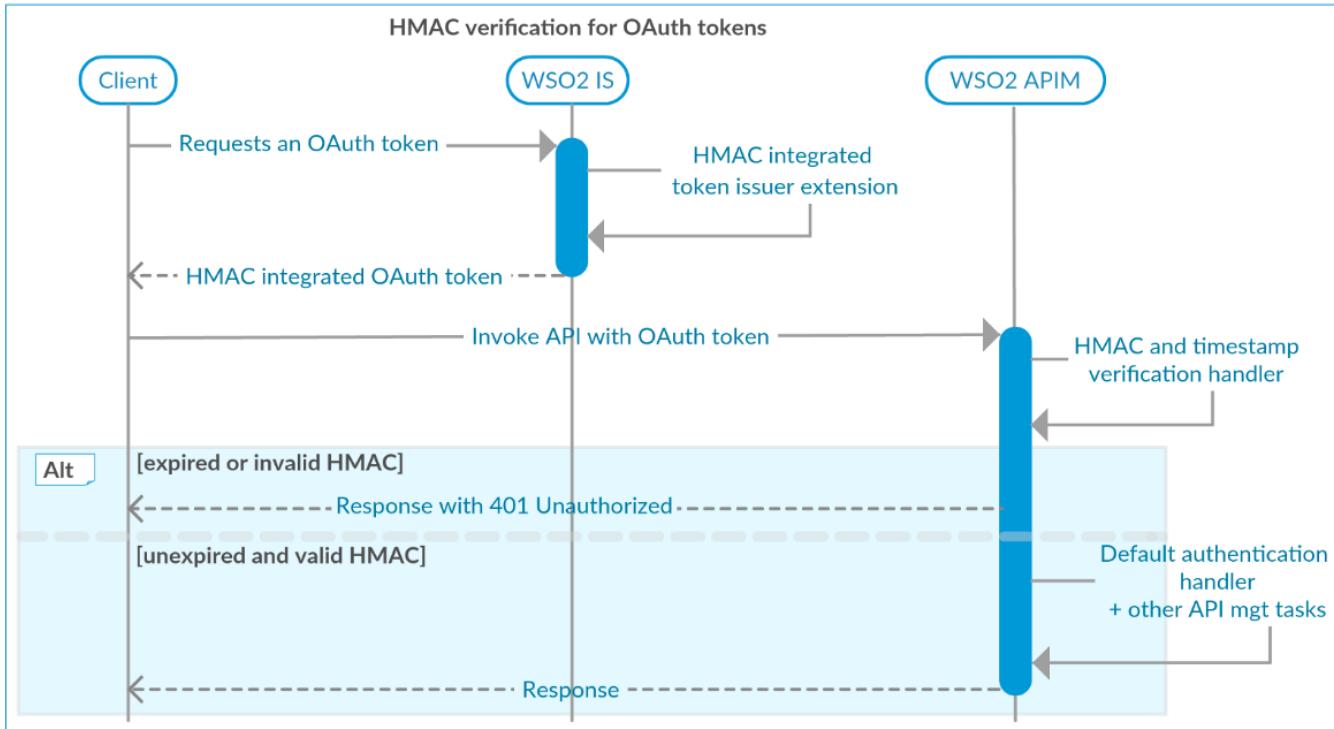
Implementing security measures in order to prevent the possible attacks is a need in using enterprise software. Keyed-Hash Message Authentication Code (HMAC) validation is such measure which involved a cryptographic hash function and used to verify both the data integrity and authentication of a Message as with any Message Authentication code. In this tutorial you will use the HMAC to validate the OAuth tokens created in WSO2 API Manager and WSO2 Identity Server.

- Preventing miss-use of OAuth Tokens
- WSO2 IS Extension - OAuth Token Generator Extension
- WSO2 API Manager extension - HMAC and timestamp verification handler

### Preventing miss-use of OAuth Tokens

In API Manager, the main use case of HMAC is preventing miss-use of expired OAuth tokens or randomly generated OAuth tokens. Stolen or randomly generated tokens can be used to employ DOS/DDOS attacks effectively.

If an attacker uses random tokens to send API requests, API Manager will try to verify the token and it will hit through the critical path of verification. This is a costly transaction and it can cause high latencies and instability in API Manager clusters. Implementation of this particular solution is done using extensions developed for standard extension points of WSO2 API Manager and WSO2 Identity Server.



### WSO2 IS Extension - OAuth Token Generator Extension

Engage the HMAC OAuth handler in order to do the Keyed-Hash Message Authentication Code (HMAC) validation by adding following into <IS\_HOME>/repository/conf/identity/identity.xml

```
<IdentityOAuthTokenGenerator>com.sample.lahiru.wso2.hmac.oauth</IdentityOAuthTokenGenerator>
```

More information on developing OAuth token generator extensions [here](#). Code for this particular solution can be found in [oauth-hmac-extension](#) GitHub repository.

This extension is responsible for enhancing the OAuth token with HMAC(Hash-based Message Authentication Code), so that above mentioned attacks will be less effective. Following two parts will be added to the token in addition to the default token created in WSO2 IS.

- HMAC
- Expiry timestamp

The format of the access token will be as follows thereafter. The token has 3 parts, delimited by “.”.

Part I—original access token issued from WSO2 Identity Server

Part II—Hex value for token expiry time

Part III—HMAC calculation of ('Part I' + '.' + 'Part II')

**Access token format : <Part I>.<Part II>.<Part III>**

E x a m p l e  
ba13cf7473cfbde970ae6e8b60973f64.0000015fc1ebabde.67830f2f2886256eb80faa9dab85c3d2c9be7db1

## WSO2 API Manager extension - HMAC and timestamp verification handler

You can engage this handler by adding following entry before **#foreach(\$handler in \$handlers)** line of **velocity\_template.xml** file located in <AM\_HOME>/repository/resources/api\_templates/ directory.

```
<handler
class="com.sample.lahiru.wso2.hmac.handler.HMACTokenValidatorHandler"/>
```

Refer [Writing Custom Handlers](#) for understanding how to develop and engage WSO2 API handler extensions. Find the code for APIM handler in GitHub in [oauth-hmac-extension](#).

This custom handler verifies the HMAC of the token before it tries to authenticate using default authentication handler, which will be an expensive operation usually. It will also make sure the token is not expired. These verification will avoid any API calls to WSO2 API Manager, in case of the token is expired or HMAC is invalid.

HMAC validation handler calculates the HMAC using Part I and Part II(See Access token format), extracted from the token and validates by comparing that value with HMAC value included in the token(Part III).

### Customizing User SignUp in API Store

WSO2 API Manager allows onboarding new users to the API store through a Self Sign-up page. The default sign-up page has set of mandatory and optional fields for user to provide details. However, there can be cases where one needs to customize the available fields by modifying available them or/and adding new fields.

This can be easily achieved in WSO2 API manager since the fields are loaded dynamically from the user claim attributes. This documentation explains how we can customize the default Sign-up page.

- [User Sign up Page](#)
- [Adding a new field to the User SignUp Form](#)
- [Modifying Existing Fields](#)

### User Sign up Page

By default API Store Sign-up looks as below.



Create your Account

Username \*  
 Characters left: 30

Password \*

Re-type Password \*

First Name \*

Last Name \*

Email \*

[Hide Additional Details](#)

Organization

Country

Land Phone

Mobile Phone

IM

URL

It has following fields by default.

Mandatory Fields	Additional Fields (Optional)
UserName	Organization
Password	Coutry
Re-type Password	Land Phone
First Name	Mobile Phone
LastName	IM
Email	URL

## Adding a new field to the User SignUp Form

If you want to add a new field to Store Sign-up page to be filled up when User Sign Up you can do this by adding a local claim through Management Console.

Here we are going to add a field called City. Follow the below steps to do so.

1. Start API Manager and go to Management Console (<https://localhost:9443/carbon/>)
2. Go to **Claims -> Add -> Add Local Claim** under **Main** tab.

The screenshot shows the WSO2 API Manager Management Console interface. On the left, there is a sidebar with tabs for Home, Identity, Claim, Monitor, Configure, and Jobs. Under the 'Claim' tab, there are options for 'Users and Roles', 'User Stores', and 'Claims'. The 'Claims' option has a sub-menu with 'Add' and 'List'. The 'Add' option is highlighted with a red box. The main content area shows the 'Add New Dialect/Claim' screen with three buttons: 'Add Claim Dialect', 'Add Local Claim' (which is also highlighted with a red box), and 'Add External Claim'.

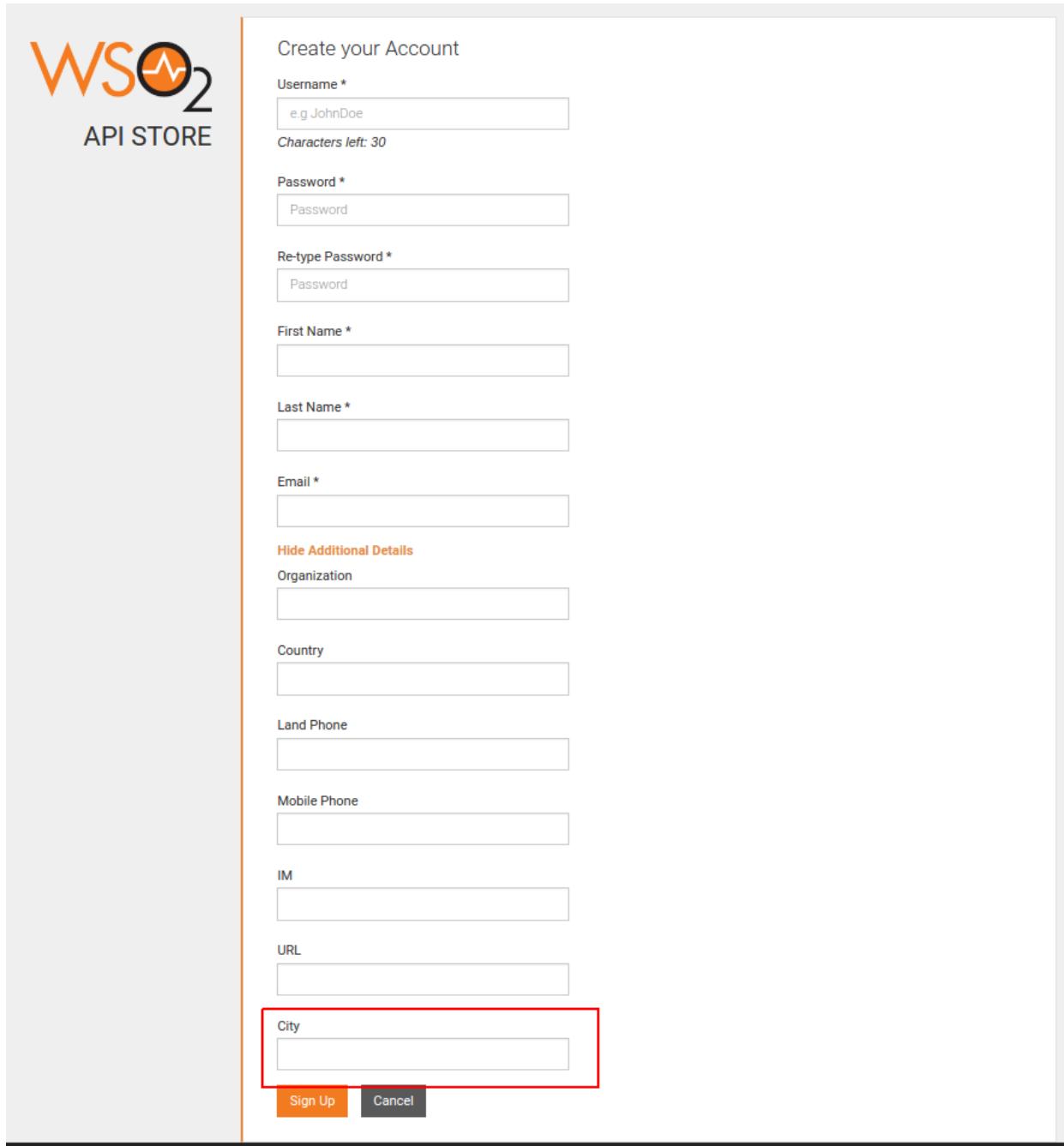
3. Fill the local claim details with below values for the new claim.

<b>Claim URI</b>	<a href="http://wso2.org/claims/city">http://wso2.org/claims/city</a>
<b>Display Name</b>	City
<b>Description</b>	City
<b>Mapped Attribute</b>	city
<b>Supported By Default</b>	select

The screenshot shows the 'Add Local Claim' configuration screen. It includes fields for 'Dialect URI' (set to 'http://wso2.org/claims'), 'Claim URI\*' (set to 'http://wso2.org/claims/city'), 'Display Name\*' (set to 'City'), 'Description\*', 'Mapped Attribute(s)\*' (with a table showing a mapping from 'User Store Domain Name: PRIMARY' to 'Mapped Attribute: city'), 'Regular Expression', 'Display Order', 'Supported by Default' (checkbox checked), 'Required' (checkbox unchecked), 'Read only' (checkbox unchecked), and 'Additional Properties'. At the bottom are 'Add' and 'Cancel' buttons.

The **claims** which are **Supported by Default**, are only displayed in the Sign-up page. Therefore when you are adding new claims make sure to check **Supported by Default** checkbox.

4. If you need this claim to be a required field (Mandatory field in Sign-up), make sure to check 'Required' checkbox.
5. After entering the values, click 'Add'.
6. Go to API Store Sign-up page and refresh. You should see the newly added field.



The image shows the 'Create your Account' form from the WSO2 API Store. The form is divided into two sections: 'Create your Account' and 'Hide Additional Details'. The 'Create your Account' section contains fields for Username, Password, Re-type Password, First Name, Last Name, and Email. The 'Hide Additional Details' section contains fields for Organization, Country, Land Phone, Mobile Phone, IM, URL, and City. A red box highlights the 'City' field, which is currently empty. At the bottom of the form are 'Sign Up' and 'Cancel' buttons.

Create your Account

Username \*

Characters left: 30

Password \*

Re-type Password \*

First Name \*

Last Name \*

Email \*

**Hide Additional Details**

Organization

Country

Land Phone

Mobile Phone

IM

URL

City

**Sign Up****Cancel**

## Modifying Existing Fields

By editing the existing claims mapped to the fields, you can modify the fields of User Sign Up.

Following steps guide you to make the **city** field **required**. Also change the display order of the field.

1. Start API Manager and go to Management Console (<https://localhost:9443/carbon/>)
2. Go to **Claims -> List** and click on <http://wso2.org/claims>. In the displaying list of claims.

3. Click Edit on City claim.

4. Select Required checkbox. To change the display order, change the display order of all the city claim to 4.

5. Now Access the API Store Sign-up page. You will see that the **City** field is re-ordered and marked as required.



Create your Account

Username \*  
 Characters left: 30

Password \*

Re-type Password \*

First Name \*

Last Name \*

City \*

Email \*

[Hide Additional Details](#)

Organization

Country

Land Phone

Mobile Phone

IM

URL

**Sign Up** **Cancel**

Like above you can modify the other existing fields in the User Sign Up Page by editing the claims.

## Working with Security

After you install the API-M, it is recommended to change the default security settings according to the requirements of your production environment. As the API-M is built on top of the WSO2 Carbon platform, some security configurations are inherited from the Carbon platform.

### Important!

If you are configuring your **production environment**, be sure to check the [Security Guidelines for Production Deployment](#) before applying any security configurations.

The following topics explain the platform-specific, and product-specific configurations:

- [API-M-specific security configurations](#)
- [WSO2 Carbon platform-based security configurations](#)
- [API Endpoint Security](#)

### **API-M-specific security configurations**

See the following topics:

- [Passing Enduser Attributes to the Backend Using JWT](#)
- [Dynamic SSL Certificate Installation](#)
- [Maintaining Logins and Passwords](#)
- [Saving Access Tokens in Separate Tables](#)
- [Configuring WSO2 Identity Server as the Key Manager](#)
- [Configuring a Third-Party Key Manager](#)
- [Enabling Role-Based Access Control Using XACML](#)
- [Encrypting OAuth Keys](#)
- [Provisioning Out-of-Band OAuth Clients](#)
- [Pass a Custom Authorization Token to the Backend](#)

### **WSO2 Carbon platform-based security configurations**

The following security configurations are common to all WSO2 products that are built on top of the WSO2 Carbon platform.

Configuration	Description
Configuring transport-level security	<p>WSO2 products support a variety of transports that make them capable of receiving and sending messages over a multitude of transport, and application-level protocols. By default, all WSO2 products are shipped with the HTTP transport. The transport receiver implementation of the HTTP transport is available in Carbon platform. The transport sender implementation comes from the Tomcat HTTP connector, which is configured in the &lt;API-M_HOME&gt;/repository/conf/tomcat/catalina-server.xml file.</p> <p>For more information on securing the HTTP transport, see <a href="#">Configuring transport level security</a> in the WSO2 Administration Guide.</p>
Configuring keystores	<p>A keystore is a repository that stores the cryptographic keys and certificates. These artifacts are used for encrypting sensitive information, and establishing trust between your server and outside parties that connect to your server.</p> <p>All WSO2 products come with a default keystore (wso2carbon.jks). In a production environment, it is recommended to replace it with one. You can also configure multiple keystores for different purposes.</p> <p>See the following in the WSO2 Administration Guide:</p> <ul style="list-style-type: none"> <li>• Learn <a href="#">how public key encryption and keystores are used</a>.</li> <li>• Learn <a href="#">how to create new keystores and replace the default one</a>.</li> <li>• Learn <a href="#">how configuration files should be updated</a> to use the relevant keystore for different purposes.</li> </ul>

To download a keystore in WSO2 API Manager, do the following:

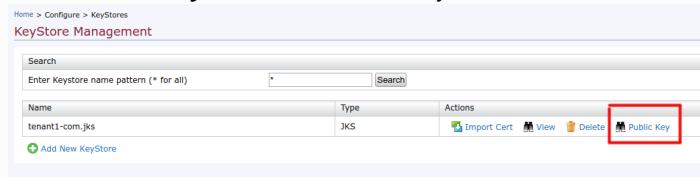
- Sign in to <https://<hostname>:9443/carbon> as the tenant admin and click on **Configure**.



- Select **Keystores**.



- Click **Public Key** to download the keystore for the selected tenant.



Securing sensitive passwords	As a secure vault implementation is available in all WSO2 products, you can encrypt the sensitive data (i.e., passwords in configuration files and passwords for mediation flows) using the Cipher tool. For more information, see the following sections. <ul style="list-style-type: none"> <li>• <a href="#">Working with Encrypted Passwords</a></li> <li>• <a href="#">Encrypting Secure Endpoint Passwords</a></li> </ul>
Enabling JAVA security manager	See <a href="#">Enabling JAVA security manager</a> in the WSO2 Administration Guide on how to prevent untrusted code from manipulating your system.

## API Endpoint Security

Look into the following topics under enabling endpoint security for the APIs.

- [Basic Auth](#)
- [Digest Auth](#)

## 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 end user 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 [JSON Web Token \(JWT\) Overview](#).

To authenticate end users, 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 JWT token contains the following information.

**JWT Header** : The header section declares that the encoded object is a JSON Web Token (JWT) and the JWT is in plaintext, that is not signed using any encryption algorithm.

#### JWT Claims set :

- "iss" - The issuer of the JWT
- "exp" - The token expiration time
- "http://wso2.org/claims/subscriber" - Subscriber to the API, usually the app developer
- "http://wso2.org/claims/applicationname" - Application through which API invocation is done
- "http://wso2.org/claims/apicontext" - Context of the API
- "http://wso2.org/claims/version" - API version
- "http://wso2.org/claims/tier" - Tier/price band for the subscription
- "http://wso2.org/claims/enduser" - Enduser of the app who's action invoked the API

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
- Customizing the JWT generation
- Changing the JWT encoding to Base64URL encoding
- Expiry time of the JWT

#### Configuring JWT

Before passing enduser attributes, you enable and configure the JWT implementation in the <API-M\_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.

Enable JWT in all Gateway and Key Manager nodes. For more information on setting up a distributed deployment of API Manager, see [Distributed Deployment of API Manager](#).

Element	Description
<EnableJWTGeneration>	Uncomment <EnableJWTGeneration> property and set this value to <b>true</b> to enable JWT generation.
<JWTHeader>	The name of the HTTP header to which the JWT is attached.

&lt;ClaimsRetrieverImplClass&gt;

By default, the <ClaimsRetrieverImplClass> parameter is commented out:

```
<ClaimsRetrieverImplClass>org.wso2.carbon.apimgt.impl.ClaimsRetrieverImpl
```

By default, the following are encoded to the JWT:

- subscriber name
- application name
- API context
- API version
- authorized resource owner name

In addition, you can also write your own class by extending the interface org.wso2.carbon.apimgt.impl.ClaimsRetrieverImpl. You need to implement the following methods of the interface:

Method	Description
void init() throws APIManagementException;	Used to perform initialization tasks. Is executed before the claims are encoded.
SortedMap<String, String> getClaims(String endUserName) throws APIManagementException;	Returns a sorted map of claims. The key is the attribute name and the value is the corresponding user attribute value. The order of encoding the claims is determined by the sorted map.
String getDialectURI(String endUserName);	The dialect URI to which the attribute names are mapped. If the getClaims method returns {email: "user1@wso2.com"}, and the dialect URI is /wso2.org/claims, the JWT will contain "email": "user1@wso2.com" as part of the claims.

&lt;ConsumerDialectURI&gt;

The dialect URI under which the user's claims are looked for. Only works when the consumer is a user.

The JWT token contains all claims defined in the <ConsumerDialectURI> element. If you want to include specific subset of users to be included in the JWT, simply uncomment this element after enabling the consumer.

&lt;SignatureAlgorithm&gt;

The signing algorithm used to sign the JWT. The general format of the JWT is {token header}{.}{token payload}{.}{token signature}. If you do not specify the algorithm, signing is turned off and the JWT looks as {token info}{.}{token payload}{.}.

This element can have only two values - the default value, which is SHA256withRSA, and

You can use TCPMon or API Gateway debug logs to capture JWT token header with enduserdetails. To enable gateway DEBUG logs for wire messages,

1. Go to the <APIM\_GATEWAY>/repository/conf directory and open the log4j.properties file with a text editor.
2. Edit the entries for the two loggers as follows (remove the # in order to enable debug logs):
 

```
log4j.logger.org.apache.synapse.transport.http.headers=DEBUG
log4j.logger.org.apache.synapse.transport.http.wire=DEBUG
```

### ***Customizing 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.keymgt.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.keymgt.APIConstants;
import org.wso2.carbon.apimgt.keymgt.dto.APIKeyValidationInfoDTO;
import org.wso2.carbon.apimgt.keymgt.token.JWTGenerator;
import org.wso2.carbon.apimgt.api.*;

import java.util.Map;

public class CustomTokenGenerator extends JWTGenerator {

 public Map<String, String>
populateStandardClaims(TokenValidationContext validationContext)
 throws APIManagementException {
 Map claims =
super.populateStandardClaims(keyValidationInfoDTO, apiContext,
version);
 boolean isApplicationToken =

keyValidationInfoDTO.getUserType().equalsIgnoreCase(APIConstants.ACCE
SS_TOKEN_USER_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 the <API-M\_HOME>/repository/components/lib directory.
3. Add your class in the <JWTGeneratorImpl> element of the <API-M\_HOME>/repository/conf/api-manager.xml file.

```
<JWTConfiguration>
 ...
 <JWTGeneratorImpl>org.wso2.carbon.test.CustomTokenGenerator</JWTGeneratorImpl>
 ...
</JWTConfiguration>
```

4. Set the <EnableJWTGeneration> element to **true** in the api-manager.xml file.
5. Restart the server.

#### **Changing the JWT encoding to Base64URL encoding**

The default JWT generator, org.wso2.carbon.apimgt.impl.token.JWTGenerator, encodes the value of the JWT using Base64 encoding. However, for certain apps you might need to have it in Base64URL encoding. To encode the JWT using Base64URL encoding, add the URLSafeJWTGenerator class in the <TokenGeneratorImpl> element in the <API-M\_HOME>/repository/conf/api-manager.xml file as shown below.

```
<JWTConfiguration>
 ...
 <JWTGeneratorImpl>org.wso2.carbon.apimgt.keymgt.token.URLSafeJWTGenerator</JWTGeneratorImpl>
 ...
</JWTConfiguration>
```

#### **Expiry time of the JWT**

JWT expiry time depends directly on whether caching is enabled in the Gateway Manager or Key Manager. The WSO2 API-M Gateway caching is enabled by default. However, if required, you can enable or disable the caching for the Gateway Manager or the Key Manager using the <EnableGatewayTokenCache> or <EnableKeyManagerTokenCache> elements respectively in the <API-M\_HOME>/repository/conf/api-manager.xml file. If caching is enabled for the Gateway Manager or the Key Manager, the JWT expiry time is the same as the cache expiry time by default.

The claims that are retrieved for the JWT Token generation are cached. The expiry time of these JWT claims can be set by setting the **JWTClaimCacheExpiry** in the api-manager.xml, under CacheConfigurations element:

```
<CacheConfigurations>
 ...
 <!-- JWT claims Cache expiry in seconds -->
 <JWTClaimCacheExpiry>900</JWTClaimCacheExpiry>
 ...
</CacheConfigurations>
```

## Dynamic SSL Certificate Installation

If you have a backend with a self-signed certificate (or a certificate which is not signed by a CA) you need to import it to the client-truststore and restart the server. This feature enables you to upload the backend certificate through API Publisher while creating or editing your API. Follow the steps below to add a new certificate to any endpoint. Note that this feature supports only **HTTP/REST** and **HTTP/SOAP** endpoints.

### Prerequisites

1. Ensure that you have downloaded the latest WUM update. For more details, see [Updating WSO2 Products in the WSO2 Administration Guide](#).
2. If you are an existing user, follow the instructions given below.
  - a. Run the scripts inside the <APIM\_WUM\_UPDATED\_PACK>/dbscripts/apimgt directory, according to your preferred database. For instructions on configuring databases, see [Set up the database](#). Verify that the table AM\_CERTIFICATE\_METADATA has been created in your database.
  - b. Open the <APIM\_HOME>/repository/conf/axis2/axis2.xml file. Add the following code under the **PassThroughHTTPSSLSSender** parameter.

```

<transportSender name="https"
class="org.apache.synapse.transport.passthru.PassThroughHttpSSLSSender">
 ...
 <!-- ===== -->
 <!-- Configuration for Dynamic SSL Profile loading. -->
 <!-- Configured for 5 mins. -->
 <!-- ===== -->
 <parameter name="dynamicSSLProfilesConfig">

 <filePath>repository/resources/security/sslprofiles.xml</filePath>
 <fileReadInterval>600000</fileReadInterval>
 </parameter>
</transportSender>
```

- c. If you use a different Trust Store/ Keystore configuration in the axis2.xml or carbon.xml files ,modify the KeyStore and TrustStore location in <APIM\_WUM\_UPDATED\_PACK>/repository/resources/security/sslprofiles.xml file accordingly. The sslprofiles.xml file is configured with the existing client-truststore.jks

This feature currently supports only the the following formats for keystores and certificates.

- Keystore : .jks
- Certificate : .crt

If you need to use a certificate in any other format, you can convert it using a standard tool before uploading.

### Adding a certificate

1. Log in to the API Publisher. [Create a new API](#) or edit an existing API.
2. Go to the **Implement** tab. Click **Manage Certificates** and click **Add New Certificate**

Endpoint: \* (specify at least one)

Production :	<a href="https://localhost:9443/am/sample/pizzashack/v1/a">https://localhost:9443/am/sample/pizzashack/v1/a</a>		Test
Sandbox :	<a href="https://localhost:9443/am/sample/pizzashack/v1/a">https://localhost:9443/am/sample/pizzashack/v1/a</a>		Test

[Manage Certificates](#)

You do not have any uploaded Certificates for above endpoints.

[Add New Certificate](#)

[Show More Options](#)

- Enter the following information and click **Upload**.

### Upload Endpoint Certificates

Alias:*	<input type="text" value="Production_Backend"/>
Endpoint:*	<input type="text" value="http://ws.cdyne.com"/>
Certificate:*	<input type="text" value="certificate.crt"/>
<a href="#">Upload</a> <a href="#">Close</a>	

Name	Description
Alias	Enter a name for your certificate.
Endpoint	Select an endpoint from the dropdown list
Certificate	Enter the location of your certificate file or click <b>Browse</b> to select through the UI

- The uploaded certificate aliases will be displayed.

Endpoint: \* (specify at least one)

Production :	<a href="http://ws.cdyne.com/phoneneverify/phoneneverify.asmx">http://ws.cdyne.com/phoneneverify/phoneneverify.asmx</a>		Test
Sandbox :	<a href="https://localhost:9443/am/sample/pizzashack/v1/a">https://localhost:9443/am/sample/pizzashack/v1/a</a>		Test

[Manage Certificates](#)

Endpoint	Alias
<a href="http://ws.cdyne.com">http://ws.cdyne.com</a>	Production_Backend

[Add New Certificate](#)

- You can repeat from step 2 to add a certificate to the sandbox endpoint.

## Upload Endpoint Certificates

X

Alias:*	Sandbox_Endpoint
Endpoint:*	https://localhost:9443
Certificate:*	certificate.crt
	<input type="button" value="Browse"/>
	<input style="border: 2px solid red;" type="button" value="Upload"/> <input type="button" value="Close"/>

You add only one certificate per endpoint. Make sure that your certificates have not expired.

### ***Deleting a certificate***

To delete a certificate, click the icon adjacent to the certificate, as shown below.

Endpoint	Alias	
http://ws.cdyne.com	Production_Backend	

## Maintaining Logins and Passwords

This section covers the following topics:

- Changing the super admin credentials
- Recovering a password
- Login in via multiple user attributes in API Store
- Setting up primary and secondary logins
- Setting up an e-mail login
- Setting up a social media login

### ***Changing the super admin credentials***

Follow the instructions below to change the default admin password:

#### 1. Change the user credentials in the following files.

- The <UserName> and <Password> values in <APIM\_HOME>/repository/conf/user-mgt.xml file

```
<UserManager>
 <Realm>
 <Configuration>
 ...
 <AdminUser>
 <UserName>admin</UserName>
 <Password>admin</Password>
 </AdminUser>
 ...
 </Realm>
 </UserManager>
```

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

To change the password from Management Console (<https://localhost:9443/carbon>), follow the steps in [Changing a Password](#) corresponding to API Manager.

- The `<APIM_HOME>/repository/conf/jndi.properties` file.

```
connectionfactory.TopicConnectionFactory =
amqp://admin:admin@clientid/carbon?brokerlist='tcp://localhost:5
672'
connectionfactory.QueueConnectionFactory =
amqp://admin:admin@clientID/test?brokerlist='tcp://localhost:567
2'
```

If you have **Configured API Manager Analytics**, when changing the super admin credentials you have to change credentials in `<APIM_HOME>/repository/conf/api-manager.xml` and `<APIM_HOME>/repository/conf/log4j.properties` as well.

- The `<APIM_HOME>/repository/conf/api-manager.xml` file.

```

<Analytics>
 <!-- Enable Analytics for API Manager -->
 <Enabled>true</Enabled>
 ...
 <DASServerURL>{tcp://localhost:7612}</DASServerURL>

 <!--DASAuthServerURL>{ssl://localhost:7712}</DASAuthServerURL-->
 <!-- Administrator username to login to the remote DAS
server. -->
 <DASUsername>${admin.username}</DASUsername>
 <!-- Administrator password to login to the remote DAS
server. -->
 <DASPassword>${admin.password}</DASPassword>
 ...
 <StatsProviderImpl>org.wso2.carbon.apimgt.usage.impl.APIUsageStatisticsRdbmsClientImpl</StatsProviderImpl>
 ...
 <DASRestApiURL>https://localhost:9444</DASRestApiURL>
 <DASRestApiUsername>${admin.username}</DASRestApiUsername>
 <DASRestApiPassword>${admin.password}</DASRestApiPassword>

</Analytics>

```

- The <APIM\_HOME>/repository/conf/log4j.properties file.

```

log4j.appender.DAS_AGENT.userName=admin
log4j.appender.DAS_AGENT.password=admin

log4j.appender.LOGEVENT.userName=admin
log4j.appender.LOGEVENT.password=admin

```

### 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](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>
```

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

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

### **Recovering a password**

See [How can I recover the admin password used to log in to the management console?](#)

### **Login in via multiple user attributes in API Store**

See [Authentication using multiple Attributes](#) in the WSO2 IS documentation.

### **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 access token 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 `<LoginConfig>` element in the `<APIM_HOME>/repository/conf/api-manager.xml` file.
  - a. 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:

```
<LoginConfig>
 <UserIdLogin primary="true">
 <ClaimUri></ClaimUri>
 </UserIdLogin>
 <EmailLogin primary="false">
 <ClaimUri>http://wso2.org/claims/emailaddress</ClaimUri>
 </EmailLogin>
</LoginConfig>
```

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 the 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>

 <CheckPermissionsRemotely>false</CheckPermissionsRemotely>
</AuthManager>
```

If you have set the `CheckPermissionRemotely` parameter as true, the permissions will be checked in the remote server set in `ServerURL`. If the parameter is set as false the permissions will be checked by the local server

**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.

#### Setting up an e-mail login

See [Email Authentication](#) in the WSO2 IS documentation.

- When setting up email login specify the complete username with tenant domain. If you are in super tenant mode username should be as follows. `<username>@<email>@carbon.super`  
**Example :** `admin@wso2.com@carbon.super`.
- When configuring `<DataPublisher>` section under `<ThrottlingConfiguration>` section in `<PRODUCT_HOME>/repository/conf/api-manager.xml`, specify the fully qualified username with tenant domain.  
**Example :** `<Username>admin@wso2.com@carbon.super</Username>`
- When specifying username in JMS Connection URL, under `<JMSSConnectionParameters>` in `<PRODUCT_HOME>/repository/conf/api-manager.xml`, "@" characters should be replaced by "!" character.  
**Example URL :**

```
<connectionfactory.TopicConnectionFactory><![CDATA[amqp://admin!wso2
.com!carbon.super:admin@clientid/carbon?failover='roundrobin'&cyclec
ount='2'&brokerlist='tcp://10.100.0.3:5682?retries='5'&connectdelay=
'50';tcp://10.100.0.3:5692?retries='5'&connectdelay='50']]></connec
tionfactory.TopicConnectionFactory>
```

### **Setting up a social media login**

You can auto-provision users based on a social network login by integrating the API Manager with WSO2 Identity Server. Refer [Log in to the API Store using Social Media](#) for more information.

Note that auto-provision users based on a social network login 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 the 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 is 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.

### **Saving Access Tokens in Separate Tables**

This feature has been deprecated as it is redundant. Although it was introduced as a security measure, a compromise in the database would result in a compromise in all its 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 the `<APIM_HOME>/repository/conf/identity/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/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/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 the '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/identity.xml file as shown in the following example:

### <APIM\_HOME>/repository/conf/identity/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

The **Key Manager** handles all clients, security and access token-related operations. For more information, see [Key Manager](#).

To configure WSO2 Identity Server as the Key Manager of the API Manager, see [Configuring WSO2 Identity Server as a Key Manager](#).

### Configuring a Third-Party Key Manager

The **Key Manager** handles all clients, security, and access token-related operations. In a typical API Manager production deployment, different components talk to the Key Manager component for achieving different tasks. The API Gateway connects with the Key Manager to check the validity of OAuth tokens, subscriptions, and API invocations. When a subscriber generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth App and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database. For more information, see [Key Manager](#).

The Key Manager decouples the OAuth client and access token management from the rest of its operations, so that you can plug in a third-party OAuth provider for managing OAuth clients and access tokens. When working with an external Key Manager, you need to extend the required Key Manager interface(s), which are explained below, based on your requirements.

- **Key Manager interface** - This interface handles functionalities of the API Store. It contains methods to create, update, get, and delete OAuth2 applications, to map the existing consumer keys and secrets, and to generate the application access tokens. For more information, see [Extending the Key Manager Interface](#).
- **Key Validation handler** - This interface handles functionalities of the Key Manager component. It contains methods to implement at API runtime to validate the token, subscriptions, and scopes, and also to generate JSON Web Tokens (JWTs). For more information, see [Extending Key Validation](#).

Let's see what basic steps you need to follow when writing a Key Manager implementation that acts as the bridge between a third-party OAuth provider and WSO2 API Manager.

In this guide, we explain how to integrate WSO2 API Store with an external Identity and Access Management (IAM) by using the **Surf OAuth Authorization Server**, which is an open source IAM, to manage the OAuth clients and tokens required by WSO2 API Manager. We have a sample client implementation that consumes APIs exposed by Surf OAuth.

This sample only extends the Key Manager interface.

Follow the instructions below to configure the third-party Key Manager:

- Step 1: Start the authorization server
- Step 2: Configure WSO2 API Manager
- Step 3: Run the sample

#### **Step 1: Start the authorization server**

1. Download the binary located [here](#) and deploy it in a Tomcat server.

Alternatively, you can build the OAuth Server from scratch and start the server by issuing the `mvn jetty:run` command in the `api-authorization-server-war` folder. Detailed steps for building and starting the server are provided [here](#).

**Tip:** The Surf OAuth web application that you just downloaded has the following customizations:

- The `apis.application.properties` file is copied to the classpath.
- All the URLs starting with `localhost` are replaced by the loop back IP (`127.0.0.1`).
- `org.surfnet.oaaas.noop.NoopAuthenticator` authenticator is set as the default authenticator.
- Token expiry time is increased to 99999 seconds. This ensures that the tokens issued for the web client last several months.

2. Move the web application to the ROOT context to ensure that the Surf Oauth web applications work on Tomcat.

```
rm -rf tomcat7/webapps/ROOT
mv tomcat7/webapps/surf-oauth tomcat7/webapps/ROOT
```

3. Access <http://127.0.0.1:8080/> to see the following page:

The screenshot shows the Surf OAuth 2.0 login interface. The main heading is "OAuth 2.0. But dead simple.". Below it, there is a descriptive text: "Can you imagine getting an OAuth2 compliant Authorization Server (and this client apparently;-) up in a matter of minutes? Wait and see. By the way, the Apis Authorization Server lets you authenticate against any possible backend of your choice and is totally agnostic as it comes to the flavor of your Resource Server." At the bottom left, there is a blue "Login" button.

The server is now up and running.

4. Follow the steps below to create a resource server.

- In Surf OAuth, click the **Resource Servers** link where all the OAuth clients are grouped together.
- Register a resource server representing WSO2 API Manager.
- Add two scopes named test and scope1 and save your changes.

You will use them when creating clients.

**Edit resource server**

Server name: WSO2 Resource Server

Description:

Scopes: test, scope

Thumbnail URL: http://your-thumbnail-url

Contact name: John Doe

Contact email address: john@doe.com

**Save changes** **Cancel**

The front end is now registered as a distinct client with the authorization server.

5. Follow the steps to create an OAuth Client.

- Click the **Access Tokens** link and note all the tokens issued for the web client.

These tokens are obtained at the time you sign in, by a Javascript client running on the browser. The same token is then used for subsequent operations.

Token	Resource server	Client	Scopes	Resource owner ID	Issue date	Valid until	Actions
57e21741-a133-4d16-a821-b20e83927cf6	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 6:42:16 AM	5/6/2015, 10:28:54 AM	<button>Delete</button>
06d66602-6a12-4d70-addf-f22d766d3b46	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 10:53:46 AM	5/6/2015, 2:40:25 PM	<button>Delete</button>
0d105cb9-8c7b-4172-a5a8-7520bdb5418	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 11:09:34 AM	5/6/2015, 2:56:13 PM	<button>Delete</button>
4154f2a7-b2a6-45af-aed7-35633b1c91aa	todo	authorization-server-admin-js-client	read,write	noop	5/5/2015, 2:03:59 PM	5/6/2015, 5:50:38 PM	<button>Delete</button>

- Pick an active access token from the above list.

You use it to create clients through WSO2 API Manager.

- Get a registration endpoint that is needed to register the client.

As Surf OAuth doesn't support a spec-compliant client registration yet, you can use an endpoint with similar capabilities. For example, as shown below, you can enable Developer Tools in Google Chrome to see the URL and the request:

The screenshot shows the WSO2 API Manager dashboard. On the left, there are navigation links for Resource Servers, Client Applications, Access Tokens, and Statistics. The main area has two sections: 'Resource servers' and 'Client applications'. In the 'Resource servers' section, there is one entry for 'WSO2 Resource Server' with 'test' as the client app, 'Contact: John Doe', 'Key: 6b3dce67-dfc4-44fd-8165-007cb19e4272', and 'Secret: 9358b2b6-8e17-451e-860d-973a5e5a91f'. In the 'Client applications' section, there is a message 'No clients yet. Add one.' Below the sections, a note says 'Powered by SURFnet. Fork me on [Github](#). Licensed under the [Apache License 2.0](#).'. A developer tools timeline is overlaid on the right, showing a network request for a client registration. The request URL is `http://localhost:8080/admin/resourceServer/101/client`, method is GET, status code is 200 OK, and the response body contains the JSON object from the screenshot above. The 'Request Headers' section shows the Authorization header with the value `bearer a8d6e8ae-a37a-4b83-acb3-b68f2d5ed84c`.

## Step 2: Configure WSO2 API Manager

- Build the demo.client available at <https://github.com/Rajith90/surf-oauth-demo/tree/v2.1.0>.
- Copy the JAR files that you built in to the <API-M\_HOME>/repository/components/lib directory.

If you are setting up a distributed environment, copy and paste the JAR files that you built in to the respective directories given below in the **Key Manager** node and the **Store** node respectively.

- API Key Manager - <API-M\_KEY\_MANAGER\_HOME>/repository/components/lib
- API Store - <API-M\_STORE\_HOME>/repository/components/lib

- Uncomment the <APIKeyManager> element in the /repository/conf/api-manager.xml file, which is in the API Key Manager and API Store and change the values based on your third-party implementation.

**Tip:** Be sure to replace the <RegistrationEndpoint> and <AccessToken> elements with the client registration endpoint and the access token you obtained earlier in step 7 and 6. ConsumerKey and Secret should be that of the created resource server. Also change the <hostname> in the <IntrospectionURL> accordingly.

The `nl.surfnet.demo.SurfOAuthClient` class, which is mentioned in the following example, extends the Key Manager interface.

### Example

```
<APIKeyManager>

<KeyManagerClientImpl>nl.surfnet.demo.SurfOAuthClient</KeyManagerClientImpl>
 <Configuration>
 <RegistrationEndpoint><Give the client registration endpoint you got in step 7></RegistrationEndpoint>
 <AccessToken><Give the access token you got in step 6></AccessToken>

 <IntrospectionURL>http://<hostname>:port/v1/tokeninfo</IntrospectionURL>
 <ConsumerKey>xxx</ConsumerKey>
 <ConsumerSecret>xxx</ConsumerSecret>
 </Configuration>
</APIKeyManager>
```

For a sample on Key Manager implementation, see the [WSO2 default Key Manager implementation](#).

#### **Step 3: Run the sample**

You have connected WSO2 API Manager with a third-part authorization server. Let's see how WSO2 API Manager creates OAuth clients at Surf OAuth when applications are registered in the API Store. In this guide, we use [Product APIs](#) to test invoke this process.

1. Start [WSO2 API Manager](#).
2. Sign in to the WSO2 API Store and create an application.

```
curl -k -X POST -c cookies
https://localhost:9443/store/site/blocks/user/login/ajax/login.jag -d
'action=login&username=admin&password=admin'
curl -k -X POST -b cookies
https://localhost:9443/store/site/blocks/application/application-add/
ajax/application-add.jag -d
'action=addApplication&application=SurfClientApp&tier=Unlimited&description=&callbackUrl='
```

3. Register an OAuth client of the type PRODUCTION in the authorization server. As shown below, you need to send the specific parameters required by the OAuth Server in JSON.

```
curl -k -X POST -b cookies
https://localhost:9443/store/site/blocks/subscription/subscription-ad/
ajax/subscription-add.jag -d
'action=generateApplicationKey&application=SurfClientApp&authorizedDo-
mains=ALL&keytype=PRODUCTION&validityTime=3600&jsonParams={"scopes": [
"test"], "contactName": "John Doe", "contactEmail": "john@doe.com"}'
```

4. Go to the **Client Applications** link in the Surf OAuth UI.

Note the newly created client listed there.

The screenshot shows the Surf OAuth UI interface. On the left, a sidebar has links for 'Resource Servers', 'Client Applications' (which is highlighted with a red box), 'Access Tokens', and 'Statistics'. The main area is divided into two sections: 'Resource servers' and 'Client applications'. The 'Resource servers' section shows one entry: 'WSO2 Resource Server' with 'SurfClientApp\_PRODUCTION' as the client app, 'test' as the scope, and contact information for 'John Doe' (key: 6b3dce67-dfc4-44fd-8165-007cb19e4272, secret: 9358b2b6-8e17-451e-860d-973a5e54a91f). The 'Client applications' section shows two entries: 'SurfClientApp\_PRODUCTION' (client ID: surfclientapp\_production, scope: test) and 'Connected to WSO2 Resource Server'.

You have now created an application and registered an OAuth Client corresponding to it.

5. Validate tokens by subscribing to a SurfClient application and obtaining a token.

- Sign in to the API Publisher and deploy the sample API (`PizzaShackAPI`) if you haven't done so already.

The screenshot shows the WSO2 API Publisher interface. The top navigation bar includes 'HOME / APIS', 'APIS' (selected), 'STATISTICS', and 'MANAGE SUBSCRIPTIONS'. Below the navigation is a large heading 'All APIs'. A message box says 'No APIs created yet. Click one of the buttons below to get started.' with buttons for 'New API...' and 'Deploy Sample API'. The 'Deploy Sample API' button is highlighted with a red box.

- Assuming you still have the OAuth client created earlier, subscribe to this API as follows:

```
curl -k -X POST -b cookies
https://localhost:9443/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d
'action=addAPISubscription&name=PizzaShackAPI&version=1.0.0&provider=admin&tier=Unlimited&applicationName=SurfClientApp'
```

Let's obtain a token from the OAuth Provider.

- c. Go to the **Edit** view of the OAuth client and make sure the `client_credentials` grant type is enabled, and a token expiration time is specified.

Scopes

test  
 openid

Thumbnail URL: http://your-thumbnail-url

Contact name: John Doe

Contact email address: john@doe.com

Redirect URIs: http://localhost:8080/playground2  
http://your-apps-uri/ +

Allow implicit grant

Allow client credentials grant

Use refresh tokens

Token expiration time: 3600

Additional attributes: key value +

Save changes Cancel

- d. Obtain a token.  
Replace <ConsumerKey:ConsumerSecret> with the Base64 encoded ConsumerKey:ConsumerSecret of the client application you just created.

```
curl -k -d "grant_type=client_credentials&scope=test" -H
"Authorization: Basic <ConsumerKey:ConsumerSecret>" -H
"Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/token
```

- e. Update the token endpoint in the <API-M\_GATEWAY\_HOME>/repository/deployment/server/synapse-configs/default/api/\_TokenAPI\_.xml file accordingly.  
f. Update the revoke endpoint in the <API-M\_GATEWAY\_HOME>/repository/deployment/server/

- `synapse-configs/default/api/_RevokeAPI_.xml` file accordingly.
- g. If you use the authorization code grant type to generate tokens, update the authorize endpoint in the `<api>` file accordingly.

The Token endpoint format for above e, f, and g steps is: `http://<surf-hostname>:port/v1/token`. A sample change done in `_TokenAPI_.xml` is as follows:

```

<api xmlns="http://ws.apache.org/ns/synapse"
name="_WSO2AMTokenAPI_" context="/token">
...
 <endpoint>
 <http
uri-template="http://<surf-hostname>:port/v1/token">
 <timeout>
 <duration>60000</duration>
 <responseAction>fault</responseAction>
 </timeout>
 </http>
 </endpoint>
...
</api>

```

- h. Invoke the API using the token obtained.

```

curl -k -H "Authorization: Bearer
02316379-8c19-4d72-94d1-6306ea2703a4"
"https://localhost:8243/pizzashack/1.0.0/menu"

```

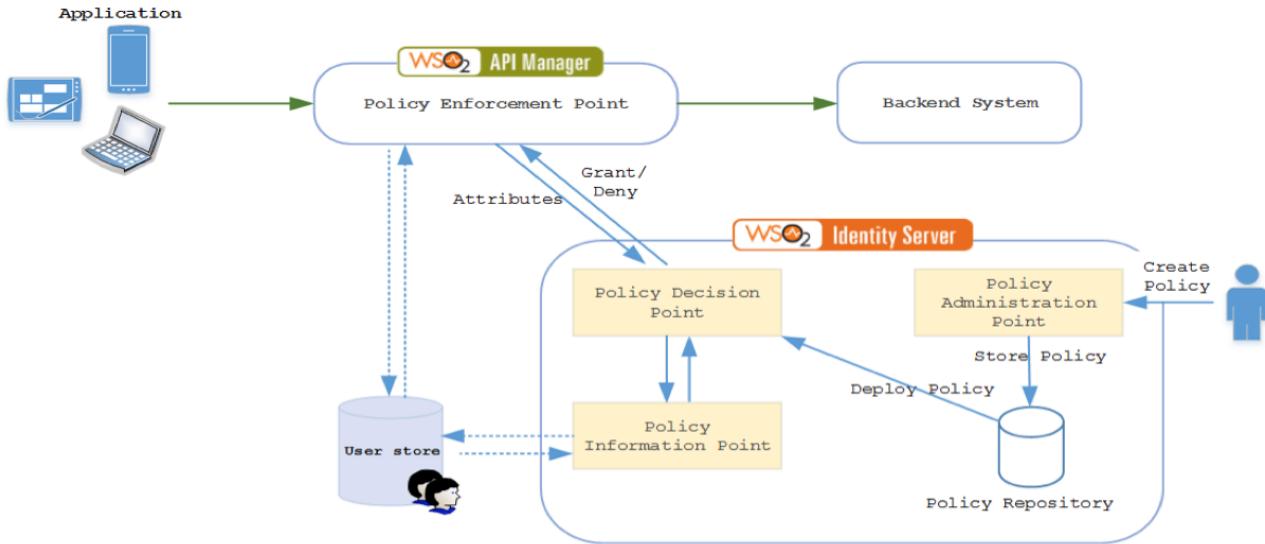
## Enabling Role-Based Access Control Using XACML

Many organizations expose their business capabilities through APIs. One of the key challenges is controlling access to these exposed APIs in such a way that all authorized users are able to access its APIs without any interruption, while at the same time making sure that any unauthorized users are kept out. In order to achieve this, parameters such as the user role can be used in determining whether to grant or deny access to an API for a given user. There are two ways to control access to users. 1. [OAuth 2.0](#) scope and 2. XACML. This section explains how an external eXtensible Access Control Markup Language (XACML) entitlement server can be integrated with WSO2 API Manager to provide role-based access control to APIs exposed via WSO2 API Manager. XACML is a declarative access control policy language based on XML that can provide a standardized way of validating authorization requests.

WSO2 API Manager provides the capability to authorize users based on [OAuth 2.0](#) tokens and this mechanism can be extended to provide role-based access control using [OAuth 2.0](#) scopes. However, as opposed to using [OAuth 2.0](#) scope to provide authorization, XACML provides a standardized way of validating authorization requests. Authorization policies can be written in a standardized way using XACML and can be stored and managed through a policy administration point (PAP). Since the policies are standardized, policies written to one XACML engine can be ported to another engine from a different vendor without any issue. Similarly, XACML provides more control on how access should be enforced as different parameters and possibilities can be evaluated. XACML also provides 'Obligations' and 'Advice' as part of the XACML response that can be used by the API Manager when enforcing the policy decision to implement fine-grained access control for APIs.

### How XACML is used with WSO2 API Manager

The diagram shown below depicts the scenario where WSO2 API Manager uses the XACML entitlement server to validate API requests that come into the API Manager. In this case, WSO2 Identity Server has been used as the XACML entitlement server.



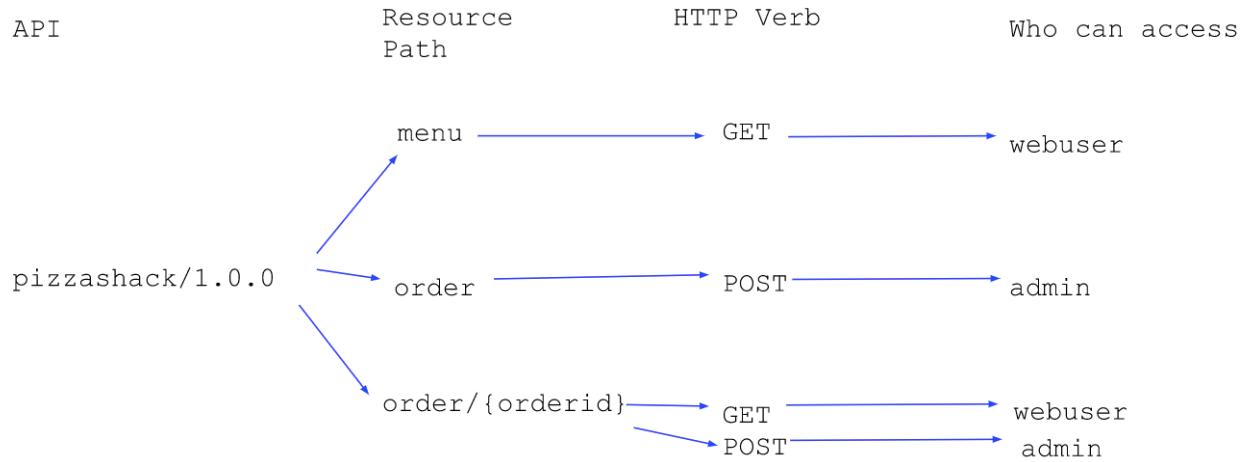
The process is initiated by an administrator who creates the XACML policies and adds them to the PAP. The created policies are stored in a policy repository and promoted to the policy decision point (PDP) by an authorized user. Once the policy is deployed, authorization requests are evaluated against this policy. There can be more than one policy deployed in the PDP.

The API Manager acts as the policy enforcement point (PEP). Whenever an API invocation comes to the API Manager, an authorisation request is sent to the PDP with the required attributes. In this case, it can be the name of the user, resource path and the HTTP verb. The PDP receives the request along with these attributes and evaluates the request against the existing policies deployed in the PDP. If the request requires more information, the PDP tries to obtain that information from a policy information point (PIP). In this case, the request from the API Manager can contain the username and the policy that is deployed requires the role of the user. In such a scenario, the PDP gets this information from the user store that is defined as a PIP. Once the PDP has the required information to evaluate the request, a response is sent back to the API Manager with its policy decision.

#### ***Enabling role-based access control***

The steps below demonstrate how WSO2 Identity Server, acting as a XACML entitlement server, can validate authentication requests from the API Manager based on a set of predefined XACML entitlement policies. This allows a standardized way of defining entitlement policies that can be enforced from WSO2 API Manager. For detailed information on XACML, see [XACML Architecture](#) in the WSO2 Identity Server documentation.

Let's take the following requirement in exposing an API via the API manager.



Based on the requirement, a single API is exposed to add or retrieve order information. Each member type (webuser or admin) is identified from the resource path. The operation (GET or POST) that needs to be performed is distinguished by the HTTP verb. Follow the steps below to implement this kind of role-based access control.

1. Let's start by creating the required users. First, you need to link both the API Manager and the Identity Server to the same user store in order to share users, roles and other related information. This can be done by linking the API manager with the LDAP user store within WSO2 Identity Server. For more information, see [Configuring an external LDAP or Active Directory Userstore](#). For this you can create a read write LDAP user store.

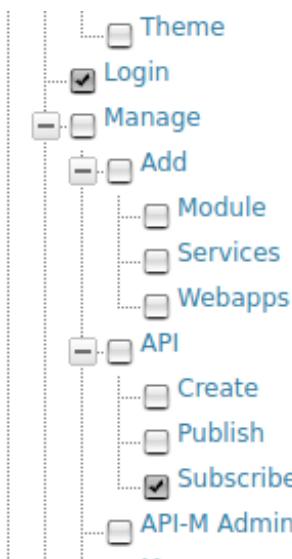
By default, in API Manager JDBCUserStore is enabled. When you are moving to the ReadWriteLDAPUserStore, make sure you have commented the configuration of JDBCUserStore and keep only one user store configuration <PRODUCT\_HOME>/repository/conf/user-mgt.xml in both nodes.

In an actual deployment, both these servers can [share the user store](#) of your organization.

2. Share the registry of both WSO2 API Manager and WSO2 Identity Server. Refer [Sharing the registry space](#) for the steps.
3. Start the WSO2 API Manager server and log in to the Management Console. Create user information with the following permission structure.

User	Role
api_user	webuser
api_admin	admin

4. When adding the webuser role, set the **Login** and **Subscribe** permissions from permission tree.



5. Start the WSO2 Identity Server and log in to its Admin Console.

Since API Manager and Identity Server run on the same server, offset the Identity Server by 1.

6. Under the **Entitlement** section, click **Policy Administration > Add New Entitlement Policy**.

The screenshot shows the WSO2 API Manager interface. On the left, there is a vertical navigation bar with tabs: Main, Monitor, Configure, and Tools. The Main tab is selected. Under the Main tab, the 'Identity' section is expanded, showing the following items:

- Users and Roles
  - Add
  - List
- User Stores
  - Add
  - List
- Claims
  - Add
  - List
- Service Providers
  - Add
  - List
  - Resident
- Identity Providers
  - Add
  - List
  - Resident

Below the Identity section, the 'Entitlement' section is partially visible, showing PAP and PDP options. The 'Policy Administration' link under PAP is highlighted with a red box.

Home > Entitlement > PAP > Policy Administration

## Policy Administration

[+ Add New Entitlement Policy](#)

Policy Type

ALL

Search Policy



7. You are redirected to a page listing all available policy editors. Select **Standard Policy Editor** from the list and add the values shown below in the policy editor. Refer [Creating a XACML Policy](#) in WSO2 Identity Server

for more information.

- a. **Entitlement Policy Name:** PizzaShackPolicy
- b. **Rule Combining Algorithm:** Deny unless Permit

When the rule combination algorithm is set to **Deny Unless Permit**, you need to set the permit criteria as a rule.

8. In the **Define Entitlement Rule(s)** area, set the following 2 rules to define the kind of requests and from which user they should be permitted.

- a. AdminGrant - grants full access to the admin user. Give the information below,

**Rule Name:** AdminGrant

**Conditions:** Under **Define your conditions by using followings....**, select drop down options as **Subject**, **is/are**, **at-least-one-member-of** in order and enter **admin** in the last field.

Click the icon next to **END** shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.

#### Create XACML Policy

The screenshot shows the 'Create XACML Policy' interface. At the top, there are fields for 'Entitlement Policy Name\*' (set to 'PizzaShackPolicy') and 'Rule Combining Algorithm' (set to 'Deny Unless Permit'). Below these are sections for 'Entitlement Policy Description' and a note: 'This Policy is going to evaluated, Only when followings are matched....'. A search bar and an 'END' button are also present. The main section is titled 'Define Entitlement Rule(s)'. It contains a 'Rule Name\*' field (set to 'AdminGrant'), a 'Rule Effect' dropdown (set to 'Permit'), and a note: 'Rule's conditions are evaluated Only when followings are matched....'. Below this are two condition definitions: one for 'Environment' (set to 'equals-with-regexp-m') and another for 'Subject' (set to 'equal admin'). The 'Subject' condition has a search icon and an 'END' button, with the search icon being highlighted with a red box. There is also a note: 'Define your obligations or advices for sending back to PEP...'. At the bottom, there is a table for 'Obligation Type' with columns 'Id' and 'Attribute Value', and an 'Add' button.

Select the attributes as given below. Note that this needs to be done for all the rules.

**Select Attribute ID:** Role

**Select Attribute Data Type:** String

**Entitlement Data Module:** Carbon Attribute Finder Module

Click on **Add** button after providing above values as shown below.

## Select Attribute Values

Select Attribute Id: Role  
 Select Attribute DataType: String  
 Entitlement Data module: Carbon Attribute Finder Module  
 Enter attribute search pattern:

**Attribute Values**      **Selected Attribute Values**

**Add** **Cancel**

- b. GetOrder- allows web users to get order information from the API. Give the information below,
- Rule Name:** GetOrder  
**Conditions:** Under Rule's conditions are evaluated..... , select drop down options as Action , is , equal in order and enter GET in the last field.  
 Under Define your conditions by using followings.... , select drop down options as Subject , is/are , at-least-one-member-of in order and enter webuser in the last field. Click the icon next to END shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.

## Create XACML Policy

Entitlement Policy Name\*: PizzaShackPolicy  
 Rule Combining Algorithm: Deny Unless Permit  
 Entitlement Policy Description:

This Policy is going to be evaluated, Only when followings are matched....

**Define Entitlement Rule(s)**

Rule Name\*: GetOrder  
 Rule Effect: Permit

Rule's conditions are evaluated Only when followings are matched....

Action is equal GET

Define your conditions by using followings....

Subject is/are at-least-one-member-of webuser

Define your obligations or advices for sending back to PEP...

Obligation Type:

Select the attributes as given below. Note that this needs to be done for all the rules.

Select Entitlement	Select Attribute Data	ID: Attribute	Role
Select Data	Module: Carbon	Type: Attribute	String Finder
			Module

Click on Add button after providing above values as shown below.

## Select Attribute Values

Select Attribute Id: Role  
 Select Attribute DataType: String  
 Entitlement Data module: Carbon Attribute Finder Module  
 Enter attribute search pattern:

**Attribute Values**      **Selected Attribute Values**

**Add** **Cancel**

9. Click Add once done.

Define Entitlement Rule(s)

Rule Name*	GetOrder	
Rule Effect	Permit	
Rule's conditions are evaluated Only when followings are matched....		
Action	is equal GET	
Define your conditions by using followings....		
Subject	is/are at-least-one-member webuser	
Define your obligations or advices for sending back to PEP...		
Obligation Type	Id	Attribute Value
Obligation		+ Add
<b>Add</b>		

10. The rules are added to the policy. Click **Finish** to save the policy.

Define Policy Obligations or Advices

Rule Id	Rule Effect	Action
AdminGrant	Permit	
GetOrder	Permit	

**Finish** **Cancel**

11. In the Policy Administration page, click **Publish to My PDP** to publish the policy to the PDP.

Publish Policy

Select policy publishing action	<input checked="" type="radio"/> Add Policy	<input type="radio"/> Update Policy	<input type="radio"/> Order Policy	<input type="radio"/> Enable Policy	<input type="radio"/> Disable Policy	<input type="radio"/> Delete Policy
Select policy Enable/Disable	<input checked="" type="radio"/> Publish As Enabled Policy <input type="radio"/> Publish As Disabled Policy					
Select policy version	<input checked="" type="radio"/> Use current policy version <input type="radio"/> Use older policy version					
Select policy order	<input checked="" type="radio"/> Use default policy order <input type="radio"/> Define policy order					
<b>Publish</b> <b>Cancel</b>						

Keep the default selected values in the Publish Policy window appears and select publish.

You can test the service by clicking **Try** option in front of the entitlement policy. It is the trylt tool developed for XACML in WSO2 Identity Server and you can access the same trylt tool by navigating to **Tools > XACML > Trylt**.

Tools

- SAML
- SAML Request Validator
- SAML Response Builder
- XACML
- Trylt

Trylt

Create Request Using Editor

Evaluation is done with one policy which policy id is **PizzaShackPolicy**

<input type="checkbox"/> Multiple Request	<input type="checkbox"/> Return Policy List	<input type="checkbox"/> Include In Result
Resource		<input type="checkbox"/> Include In Result
Subject Name		<input type="checkbox"/> Include In Result
Action Name		<input type="checkbox"/> Include In Result
Environment Name		<input type="checkbox"/> Include In Result

**Test Evaluate** **Create Request** **Clear** **Cancel**

Refer [Evaluating a XACML Policy](#) for more information on how to use the TryIt tool for XACML Policy evaluation.

## Policy Administration

The screenshot shows the 'Available Entitlement Policies' section. It lists a single policy named 'PizzaShackPolicy'. Below the list are several action buttons: 'Delete', 'Policy', 'Edit', 'Versions', 'Publish To My PDP' (which is highlighted with a red box), 'Try', and 'View Status'.

12. Download the [entitlement-1.0-SNAPSHOT.jar](#) and add it to the <API-M\_HOME>/repository/components/lib directory. This JAR file contains the `APIEntitlementCallbackHandler` class which passes the username, HTTP verb and the resource path to the XACML entitlement server. If you want to view the source code of the JAR, go [here](#).
13. Restart the server once the JAR file is added.
14. Now, you need to create a sequence containing the entitlement policy mediator that can be attached to each API required to authorize users with the entitlement server. Create an XML file with the following configuration and name it `EntitlementMediator.xml`.

```
<sequence xmlns="http://ws.apache.org/ns/synapse"
name="EntitlementMediator">
 <entitlementService xmlns="http://ws.apache.org/ns/synapse"
remoteServiceUrl="https://localhost:9444/services"
remoteServiceUserName="admin" remoteServicePassword="admin"
callbackClass="org.wso2.carbon.apimgt.gateway.sample.entitlement.APIEntitlementCallbackHandler"/>
</sequence>
```

The **Entitlement Mediator** intercepts requests and evaluates the actions performed by a user against an [eXtensible Access Control Markup Language \(XACML\)](#) policy. Here, WSO2 Identity Server is used as the XACML Policy Decision Point (PDP) where the policy is set, and WSO2 API Manager serves as the XACML Policy Enforcement Point (PEP) where the policy is enforced. Refer [Entitlement Mediator](#) for more information on parameters and usage of this mediator.

The attributes in the `<entitlementService>` element above should be modified according to the services endpoint configuration as follows.

```
remoteServiceUrl - Service url of WSO2 Identity Server, acting as the XACML entitlement server in this scenario.
remoteServiceUserName - Username
remoteServicePassword - Password used to connect to the service
```

15. Log in to the API Publisher and [create an API](#).
16. Attach the custom sequence to the inflow of the message as shown below.

The screenshot shows the WSO2 API Manager's 'Implement' tab. In the 'Managed API' section, the 'Endpoint Type' is set to 'HTTP/REST Endpoint'. Under 'Production Endpoint', the URL is 'http://localhost:9766/pizzashack-api-1.0.0/api/' with a 'Test' button. Under 'Sandbox Endpoint', the URL is 'http://localhost:9766/pizzashack-api-1.0.0/api/' with a 'Test' button. There are also options for 'Load Balanced' and 'Failover'. Below these fields is a 'Show More Options' link. The 'Message Mediation Policies' section follows, containing settings for 'Enable Message Mediation' (checked), 'In Flow' (set to 'EntitlementMediator' with an 'Upload In Flow' button), 'Out Flow' (set to 'None' with an 'Upload Out Flow' button), and 'Fault Flow' (set to 'None' with an 'Upload Fault Flow' button).

- Save, publish and test the API to make sure that the requests specified in the 2 rules defined in step 8 are accessible according to the user role specified. For example, the POST operation is only available to users with the role admin. If an anonymous user tries to access the POST operation, it should fail.

If you encounter an error stating "org.apache.axis2.transport.jms.JMSSender cannot be found by axis2\_1.6.1.wso2v16" when publishing the API, comment out the following JMSSender configuration in the <APIM\_HOME>/repository/conf/axis2/axis2\_blocking\_client.xml file and restart the server.

```
<!--transportSender name="jms"-->
 class="org.apache.axis2.transport.jms.JMSSender"/-->
```

- If you want to debug the entitlement mediator, enable debug logs in the Management Console for the org.wso2.sample.handlers.entitlement.APIENTitlementCallbackHandler class.

## Encrypting OAuth Keys

WSO2 API Manager allows you to encrypt any sensitive OAuth2.0 keys that are created. The API Manager encrypts access tokens, client secrets and authorization codes (this can be extended to any other OAuth2.0 keys if needed) using the primary keystore. The result is encoded in Base64 and stored in the database. The RSA algorithm is used by default and the key strength (1024, 2048, etc) is based on the private key strength of the primary keystore. If SymmetricEncryption is enabled, the API Manager uses the AES algorithm by default, or the algorithm specified for the SymmetricEncryption.Algorithm in the carbon.xml file.

**Symmetric Encryption** is a form of encryption where the same key is used to encrypt and decrypt the message along with a mathematical algorithm. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.

It is recommended to switch this configuration on/off **before any keys have been generated in your system**. Once token encryption is switched on, the system encrypts all sensitive OAuth2.0 data such as Access Tokens, Consumer Secrets, etc. When reading that information, the system assumes that they are in the encrypted format and attempts to decrypt them. Therefore, switching this configuration on **after** any keys are created would break the system, unless the data is converted back into plain text.

In order to encrypt the OAuth keys, change the following configurations.

1. In the <APIM\_HOME>/repository/conf/api-manager.xml file, set the <EncryptPersistedTokens> property to true.
2. In the <APIM\_HOME>/repository/conf/identity/identity.xml file, change the <TokenPersistenceProcessor> property to org.wso2.carbon.identity.oauth.tokenprocessor.EncryptionPersistenceProcessor.
3. Restart the server(s) after the above configuration changes are performed.

## Tip

- If you use a [Distributed API Manager](#) setup, the changes must be made on both the API Store and Key Manager nodes.
- If you use [WSO2 Identity Server \(WSO2 IS\)](#) as the Key Manager setup, you need to make changes in both WSO2 IS and WSO2 API Manager.

## Provisioning Out-of-Band OAuth Clients

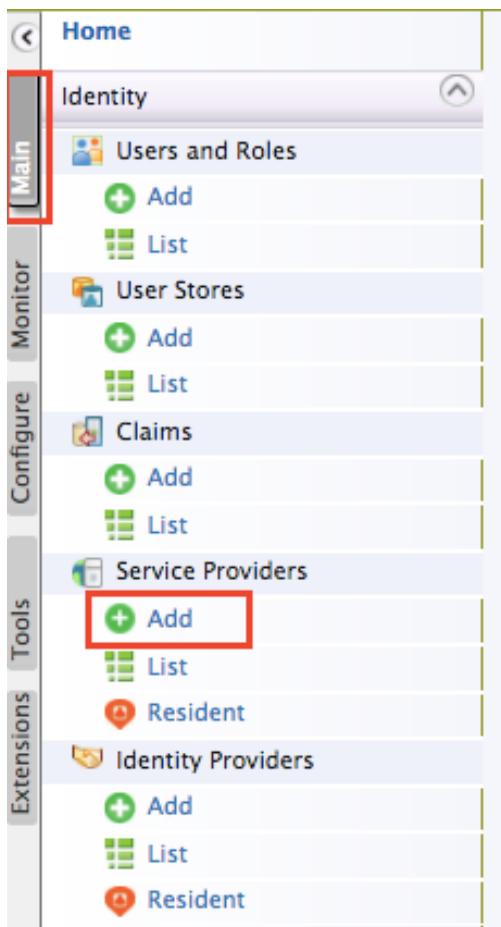
When an application [access token is generated](#), an OAuth client is created underneath. The consumer key and consumer secret shown under a key type actually belongs to the OAuth client. There can be situations where an OAuth client is created elsewhere, but needs to be associated with an application in the API Store. For instance, in an organization where WSO2 Identity Server is used as the authoritative server, OAuth clients may only be created through the Identity Server. Similarly, when a third party OAuth provider is used, users might want to use previously created OAuth clients with the API Manager. To achieve this, you can provision the OAuth clients created outside the API Store into the WSO2 API Manager (WSO2 APIM), thereby associating the OAuth client with an application in the API Store. Once the mapping is done, you can use it in the same way as an OAuth client created through the API Store.

Note that when you delete an application after Out-of-Band OAuth client is provisioned, the underlying OAuth client is not deleted. It just deletes the association of the OAuth client with the application. Therefore that OAuth client will be able to create an association to another application, which means we can map the same OAuth clients' keys to a new application created in the API Store.

The steps below describe how to provision OAuth clients created outside the API Store into the WSO2 APIM: In this example, we use a standalone API Manager instance and do this via the WSO2 APIM Management Console.

In a setup where WSO2 Identity Server is used as the Key Manager, this step is performed in the Identity Server.

1. Sign in to the WSO2 APIM Management Console (<https://<Server Host>:9443/carbon>) and click **Add** under **Service Providers**.



2. Enter the name of the service provider and click **Register**.

### Add New Service Provider

#### Basic Information

Service Provider Name:\*

⑦ A unique name for the service provider

Description:

⑦ A meaningful description about the service provider



3. Click **Configure** under **Inbound Authentication Configuration > OAuth/OpenId Connect Configuration** to add a new OAuth client.

**Service Providers**

**Basic Information**

Service Provider Name\*:  (?) A unique name for the service provider

Description:  (?) A meaningful description about the service provider

SaaS Application  (?) Applications are by default restricted for usage by users of the service provider's tenant. If this application is SaaS enabled it is opened up for all the users of all the tenants.

Claim Configuration  
 Role/Permission Configuration  
 Inbound Authentication Configuration  
 SAML2 Web SSO Configuration  
 OAuth/OpenID Connect Configuration

**Configure**

OpenID Configuration ✓  
 WS-Federation (Passive) Configuration ✓  
 WS-Trust Security Token Service Configuration  
 Kerberos KDC

Local & Outbound Authentication Configuration  
 Inbound Provisioning Configuration  
 Outbound Provisioning Configuration

**Update** **Cancel**

4. Provide a callback URL and click **Add**.

If you do not have a callback URL, you can clear the **Code** and **Implicit** authorization grant types and add the OAuth client.

**Register New Application**

**New Application**

OAuth Version\*  1.0a  2.0

Code  
 Implicit

Password

Allowed Grant Types  Client Credential  
 Refresh Token  
 SAML2  
 IWA-NTLM

**Add** **Cancel**

You have now created the OAuth client and are provided with the OAuth client key and OAuth client secret.

5. Enable the option to provide out-of-band keys by opening the <APIM\_HOME>/repository/deployment/server/jaggeryapps/store/site/conf/site.json file and changing the "mapExistingAuthApps" setting to true.

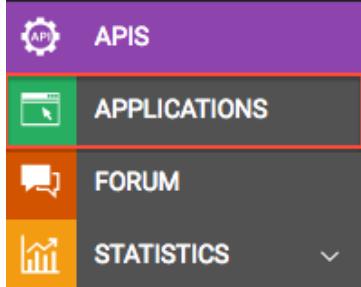
```
"mapExistingAuthApps" : true
```

Note that the ability to provision Out-of-Band Auth client will only be available for the applications that you created **after** doing this configuration.

6. Sign into the WSO2 API Store.

<https://<Server Host>:9443/store>

7. Click **Applications**.



8. Click on the respective application to view the subscriptions details for the application.

## Applications

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name	Tier	Workflow Status	Subscriptions	Actions
MyApp1	Unlimited	ACTIVE	0	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
DefaultApplication	Unlimited	ACTIVE	0	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>

Show 10 entries | Showing 1 to 2 of 2 entries

9. Provision an out-of-band OAuth client for the required environment.

The following steps explain how you can provision an out-of-band OAuth client for the production environment. If you wish to generate keys for your sandbox, you can follow the steps below using the

## Sandbox Keys tab.

a. Click **Production Keys**.

Notice that you now see a **Provide Keys** button for your application.

MyApp1

Details Production Keys Sandbox Keys Subscriptions

**No Keys Found**  
No keys are generated for this type in this application.

**Grant Types**  
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

<input checked="" type="checkbox"/> Refresh Token	<input checked="" type="checkbox"/> SAML2	<input type="checkbox"/> Implicit	<input checked="" type="checkbox"/> Password
<input checked="" type="checkbox"/> IWA-NTLM	<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code	

**Callback URL**

**Access token validity period**  
3600 Seconds.

**Generate keys**

**Map Existing OAuth Keys**  
If you already have an OAuth application registered with the IdP, you can use those keys without generating one.

**Provide Keys**

b. Click **Provide Keys**, paste the consumer key and consumer secret pair, which you received in step 4, and click **Save**.

APPLICATION LIST EDIT

MyApp1

Details Production Keys Sandbox Keys Subscriptions

**No Keys Found**  
No keys are generated for this type in this application.

**Provide Keys**

**Consumer Key**

**Consumer Secret**

**Save** Cancel

You have successfully provisioned an out-of-band OAuth client.

## Basic Auth

When you create an API using the API Publisher, you can specify the endpoint of the API backend implementation in the **Implement** tab as Production and Sandbox endpoints.

If this endpoint is secured, there is an option for you to set the Auth type and credentials for the endpoint under **Show More Options**.

The screenshot shows the 'Implement' tab of the WSO2 API Publisher. It includes fields for 'Production Endpoint' and 'Sandbox Endpoint' with their respective 'Test' buttons. Below these, a 'Show More Options' link is highlighted with a red box.

Here you can click the **Show More Options** link to select the endpoint security scheme. If you select **Secured**, you are prompted to select the authentication type for the endpoint and also to give its credentials.

Then select the endpoint authentication type according to the authentication scheme that is supported by your endpoint. If your endpoint supports basic authentication, you can select the **Basic Auth** option from the drop down list and give your credentials.

The screenshot shows the 'Implement' tab with 'Show Fewer Options' expanded. It includes fields for 'Endpoint Security Scheme' (set to 'Secured') and 'Endpoint Auth Type' (highlighted with a red box). Below these are 'Username' and 'Password' input fields.

To give more context on the above scenario, a secured endpoint is where we have access-protected resources. You

have to specify the username and the password when a request is sent to a secured endpoint. The endpoint authentication mechanism can either be Basic Authentication or Digest Authentication. They differ on how the credentials are communicated and how access is granted by the backend server.

The Endpoint Auth Type selected should match with the authentication mechanism supported by the secured endpoint.

**Basic Authentication** is the simplest mechanism used to enforce access controls to web resources. Here, the HTTP user agent provides the username and the password when making a request. The string containing the username and the password separated by a colon is Base64 encoded and sent in the authorization header when calling the backend when authentication is required.

If the user name and password is admin, the following header will be sent to the backend

Authorization: Basic YWRtaW46YWRtaW4= (where YWRtaW46YWRtaW4= is equivalent to Base64Encoded{admin:admin})

## Digest Auth

When you create an API using the API Publisher, you can specify the endpoint of the API backend implementation in the **Implement** tab as Production and Sandbox endpoints.

If this endpoint is secured, there is an option for you to set the Auth type and credentials for the endpoint under **Show More Options**.

The screenshot shows the 'Implement' tab of the WSO2 API Publisher interface. At the top, there are three tabs: 'Design' (blue), 'Implement' (yellow, currently active), and 'Manage' (grey). Below the tabs, there's a section titled 'Managed API' with a sub-section for 'Production and sandbox endpoints'. Under 'Managed API', it says 'Provide the production and sandbox endpoints of the API to be managed.' There are two main sections for endpoints:

- Production Endpoint:** A dropdown menu set to 'HTTP/REST Endpoint'. Below it are two checkboxes: 'Load Balanced' and 'Failover'.
- Sandbox Endpoint:** A dropdown menu set to 'http://ws.cdyne.com/phoneverify/phoneverify.asmx'. To its right is a 'Test' button.

At the bottom of the configuration area is a red-bordered button labeled 'Show More Options'.

Click **Show More Options** to select the endpoint security scheme. If you select **Secured**, you are prompted to select the authentication type for the endpoint and also to give its credentials.

Select the endpoint authentication type according to the authentication scheme that is supported by your endpoint. If your endpoint supports Digest authentication, you can select the **Digest Auth** option from the drop down list and give your credentials.

To give more context on the above scenario, a secured endpoint is where we have access-protected resources. You have to specify the username and the password when a request is sent to a secured endpoint. The endpoint authentication mechanism can either be Basic Authentication or Digest Authentication. They differ on how the credentials are communicated and how access is granted by the backend server.

The selected Endpoint Auth Type should match with the authentication mechanism supported by the secured endpoint.

**Digest Authentication** applies a hash function to the username and the password before sending them over the network. It is a process of applying MD5 cryptographic hashing with the usage of nonce values to prevent replay attacks. It is a simple challenge-response authentication mechanism that may be used by a server to challenge a client request and by a client to provide authentication information for the secured endpoint.

This approach is safer than Basic Authentication, which uses unencrypted base64 encoding instead of a hashing mechanism.

The following is the sample format of the header that will be sent to the backend when Digest Auth is specified as the endpoint auth type. The attributes added to the authorization header depends on the challenge header sent from the backend server.

```
Authorization: Digest username="Admin", realm="admin@wso2.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", uri="/dir/index.html",
qop=auth, nc=00000001, cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

## Working with Encrypted Passwords

All WSO2 products are shipped with a **Secure Vault** implementation that allows you to store encrypted passwords that are mapped to aliases. This approach allows you to use the aliases instead of the actual passwords in your configurations for better security. For example, some configurations require the admin username and password. If

the admin user's password is "admin", you could use `UserManager.AdminUser.Password` as the password alias. You will then map that alias to the actual "admin" password using Secure Vault. The WSO2 product will then look up this alias in Secure Vault during runtime, decrypt and use its password.

For more information on the Secure Vault implementation in WSO2 products, see [Carbon Secure Vault Implementation](#).

In all WSO2 products, Secure Vault is commonly used for encrypting passwords and other sensitive information in configuration files. When you use WSO2 API-M, you can encrypt sensitive information contained in synapse configurations (i.e., mediation flow) in addition to the information in configuration files. For more information, see the following topics:

- [Encrypting passwords in configuration files](#)
- [Encrypting passwords for mediation flow](#)
- [Using encrypted passwords in mediation flow](#)

#### *Encrypting passwords in configuration files*

To encrypt passwords in configuration files, you simply have to update the `cipher-text.properties` and `cipher-tool.properties` files that are stored in the `<API-M_HOME>/conf/security` directory and then run the Cipher tool that is shipped with the product. Go to the links given below to see instructions in the WSO2 administration guide:

- [Encrypting passwords using the automated process.](#)
- [Encrypting passwords using the manual process.](#)

The manual encryption process is relevant when the location of the configuration files, which contain the elements to be encrypted, cannot be specified using an xpath in the `cipher-tool.properties` file. The `log4j.properties` file and `jndi.properties` file are two such files which require the manual password encryption process.

- [Changing already encrypted passwords.](#)
- [Resolving already encrypted passwords.](#)

#### *Encrypting passwords for mediation flow*

**B e f o r e**

**y o u**

**b e g i n**

If you are using Windows, you need to have [Ant](#) (<http://ant.apache.org/>) installed before using the Cipher Tool.

The WSO2 API-M provides a UI that can be used for encrypting passwords and other sensitive information in synapse configurations. Follow the steps below.

1. If you are using the Cipher tool for the first time in your environment, you must first enable the Cipher tool by executing the `-Dconfigure` command with the cipher tool script:
  - a. Open a terminal and navigate to the `<API-M_HOME>/bin` directory.
  - b. Execute one of the following commands based on your OS:
    - On Linux: `./ciphertool.sh -Dconfigure`
    - On Windows: `./ciphertool.bat -Dconfigure`

If you are using the Cipher tool for the first time, this command first initializes the tool for your product.
2. When prompted, enter the primary key password, which is by default `wso2carbon`. Enter the password and proceed.
3. When prompted, enter the plain text password that you want to encrypt. Enter the following element as the password and proceed.

Enter Plain Text Value :admin

Now, you will receive the encrypted value.

### Example

Encrypted value is:

```
gaMpTzAccMScaH1lsZLXspml14HLI0M/srL5pB8jyknRKQ2zT7NuCvt1+qEkElRLgwlr0
hz3lkuE0KFuapXrCSs5pxfGMOLn4/k7dNs2SlwbsG8C++
zfJuft1Sl6cqvDRM55fQwzCPfybl713HvKu3oDaJ9VKgSbvHlQj6zqzg=
```

4. Start WSO2 API-M and sign in to the management console:
  - a. Open a terminal and navigate to the <API-M\_HOME>/bin directory.
  - b. Execute one of the following scripts:
    - On Windows: wso2server.bat --run
    - On Linux/Mac OS: sh wso2server.sh
  - c. Sign in to the management console.  
<https://<server-host>:9443/carbon>
5. Select **Browse** under **Resources** to access the registry browser and go to the /\_system/config/repository/components/secure-vault location.
6. Add the aliases and the encrypted value as a property.

## Encrypting passwords in api-manager.xml file

The api-manager.xml file does not contain any plaintext passwords. Therefore, encrypting the plaintext passwords in user-mgt.xml file as shown in [Encrypting Passwords with Cipher Tool](#) is sufficient. We recommend having the same admin user on all nodes in a clustered setup.

However if you need to use an alias to encrypt passwords in the <APIM\_HOME>/repository/conf/api-manager.xml file using the cipher tool, do the following.

To derive the alias, ignore the root XML element (i.e. <APIManager>) in the api-manager.xml file. Denote the subsequent XML elements separated by a dot (.), according to the hierarchy. You can derive the alias **AuthManager.Password** from the xml example given below.

```

<APIManager xmlns:svns="http://org.wso2.securevault/configuration">
 <DataSourceName>jdbc/WSO2AM_DB</DataSourceName>
 <GatewayType>Synapse</GatewayType>
 <EnableSecureVault>false</EnableSecureVault>
 <AuthManager>

 <ServerURL>https://localhost:${mgt.transport.https.port}${carbon.context}services/</ServerURL>
 <Username>${admin.username}</Username>
 <Password>${admin.password}</Password>
 .
 .
 </APIManager>

```

Following are some sample aliases derived for other passwords in the `api-manager.xml` file

```

ThrottlingConfigurations.DataPublisher.Password
ThrottlingConfigurations.PolicyDeployer.Password
ThrottlingConfigurations.JMSConnectionDetails.Password
Analytics.DASPassword.
Analytics.DASRestApiPassword

```

#### ***Using encrypted passwords in mediation flow***

To use the alias of an encrypted password in a mediation flow, you need to add the `{wso2:vault-lookup('alias')}` custom path expression when you define the mediation flow. For example, instead of hard coding the admin user's password as `<Password>admin</Password>`, you can encrypt and store the password using the `AdminUser.Password` alias as follows: `<Password>{wso2:vault-lookup('AdminUser.Password')}</Password>`.

This password in the mediation flow can now be retrieved by using the `{wso2:vault-lookup('alias')}` custom path expression to logically reference the password mapping.

#### **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.

Test: /test/1.0.0

The screenshot shows the 'Implement' tab selected in the top navigation bar. Under the 'Managed API' section, the 'Endpoint Type' is set to 'HTTP/REST Endpoint'. There are checkboxes for 'Load Balanced' and 'Failover'. The 'Production Endpoint' is set to 'http://appserver/resource' with a 'Test' button. The 'Sandbox Endpoint' is also set to 'http://appserver/resource' with a 'Test' button. A link 'Show Fewer Options' is visible. The 'Endpoint Security Scheme' dropdown is set to 'Secured' and is highlighted with a red box. The 'Endpoint Auth Type' dropdown is set to 'Basic Auth' and is also highlighted with a red box. The 'Credentials' section contains 'Username' and 'Password' fields, which are also highlighted with a red box.

## Cipher Tool

See [Encrypting Passwords with Cipher Tool](#) to understand how cipher tool can be used encrypt plain text passwords

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 the <APIM\_HOME>/repository/conf/api-manager.xml file to true. By default, the system stores passwords in configuration files in plain text because this value is set to false.
2. Run the cipher tool available in the <APIM\_HOME>/bin directory. If you are running Windows, it is the `ciphertool.bat` file. If you are using the default keystore, give **wso2carbon** as the primary keystore password when prompted.

```
sh ciphertool.sh -Dconfigure
```

When this command is issued, the Basic Authentication header which is set in the API definition xml file will be encrypted. For an example, see below for example of the same API when endpoint password is not encrypted and encrypted:

Not Encrypted	Encrypted
---------------	-----------

Here, the Basic authentication header is in bas464 encoded format and can be decoded to get the actual credentials of the endpoint.

```
<property name="Authorization" expression="fn:concat('Basic ', 'dGVzdDp0ZXN0MTIz')" scope="transport"/>
```

Here, the password is first looked up from the secret repository, and then set as a transport header.

```
<property name="password"
expression="wso2:vault-lookup('<api-identifier>')"/>
<property name="unpw"
expression="fn:concat('test',':',get-property('password'))"/>
<property name="Authorization" expression="fn:concat('Basic ',
base64Encode(get-property('unpw')))" scope="transport"/>
```

## Regular Expression Threat Protection for API Gateway

WSO2 API Manager provides pre-defined regex patterns to sanitize the request from SQL injection attacks. The attacks may depend on the API traffic at runtime. The API developers should identify the common attacks and select the appropriate restrictive measures. This feature extracts the data from XML, JSON payloads, Queryparam, URI path, headers and validates the content against pre defined regular expressions. If any predefined regex keyword is matched with the content, the API request is considered as a threat and it is blocked and rejected. This secures the backend resources from activities that make the system vulnerable. You can configure your own restriction patterns to thwart various attacks such as the following:

- Javascript Injection
- Server-side Include Injection
- XPath Injection
- Java Exception Injection
- XPath Abbreviated Syntax Injection

### Blacklisting patterns

We recommend the following patterns for blacklisting.

Name	Patterns
SQL Injection	. *' . *   . *ALTER . *   . *ALTER TABLE . *   . *ALTER VIEW . *   . *CREATE DATABASE . *   . *CREATE PROCEDURE . *   . *CREATE SCHEMA . *   . *create table . *   . *CREATE VIEW . *   . *DELETE . *   . *DROP DATABASE . *   . *DROP PROCEDURE . *   . *DROP . *   . *SELECT . *
Server-side Include Injection Attack	. *#include.*   . *#exec.*   . *#echo.*   . *#config.*
Java Exception Injection	. *Exception in thread.*
XPath Injection	. *' . *   . * o r . *   . * 1 = 1 . *   . *ALTER . *   . *ALTER TABLE . *   . *ALTER VIEW . *   . *CREATE DATABASE . *   . *CREATE PROCEDURE . *   . *CREATE SCHEMA . *   . *create table . *   . *CREATE VIEW . *   . *DELETE . *   . *DROP DATABASE . *   . *DROP PROCEDURE . *   . *DROP . *   . *SELECT . *
Javascript Exception	<\s*script\b[^>]*>[^<]+<\s*/\s*script\s*>

XPath Expanded Syntax Injection	/?(ancestor(-or-self)? descendant(-or-self)? following(-sibling))
------------------------------------------	-------------------------------------------------------------------

- Applying the Regular Expression Policy
- Editing the sequence through registry artifacts
- Add a custom sequence

#### Applying the Regular Expression Policy

You can apply the pre-defined Regular Expression Policy through the UI. Follow the instructions below to apply the regex\_policy in sequence.

1. Create an API or edit an existing API.
2. Go to **Message Mediation Policies** under the **Implement** tab.
3. Select **Enable Message Mediation**. Select `regex_policy` from the drop-down menu for **In Flow**.

The screenshot shows the 'Message Mediation Policies' section of the WSO2 API Manager interface. At the top, there's a note about specifying at least one endpoint. Below that, two endpoints are listed: 'Production' and 'Sandbox', each with a URL and a 'Test' button. Underneath the endpoints are links for 'Manage Certificates' and 'Show More Options'. The main configuration area starts with 'Enable Message Mediation' checked. Below it, there are three dropdown menus: 'In Flow' (set to 'regex\_policy'), 'Out Flow' (set to 'None'), and 'Fault Flow' (set to 'None'). Each dropdown has a corresponding 'Upload' button to its right. The 'In Flow' dropdown is specifically highlighted with a red box.

4. Click **Save and Publish** to save the changes.

Each request is sanitized through the regular expression threat protector. You can add or modify the regex patterns according to your requirement.

The `regex_policy` sequence is given below.

```

<sequence xmlns="http://ws.apache.org/ns/synapse" name="regex_policy">
 <log level="custom">
 <property name="IN_MESSAGE" value="Regular_expression_policy"/>
 </log>
 <property name="threatType" expression="get-property('threatType')"
value="SQL-Injection"/>
 <property name="regex" expression="get-property('regex')"
value=".+'.*|.*ALTER.*|.*ALTER TABLE.*|.*ALTER VIEW.*|
.*CREATE DATABASE.*|.*CREATE PROCEDURE.*|.*CREATE SCHEMA.*|.*create
table.*|.*CREATE VIEW.*|.*DELETE.*|.
DROP DATABASE.|.*DROP PROCEDURE.*|.*DROP.*|.*SELECT.*"/>
 <property name="enabledCheckBody"
expression="get-property('checkBodyEnable')" value="true"/>
 <property name="enabledCheckHeaders"
expression="get-property('enabledCheckHeaders')" value="true"/>
 <property name="enabledCheckPathParams"
expression="get-property('enabledCheckPathParams')" value="true"/>
 <class
name="org.wso2.carbon.apimgt.gateway.mediators.RegularExpressionProtector"
/>
 <filter source="get-property('threat')" regex=".error.*">
 <then>
 <sequence key="_threat_fault_"></sequence>
 </then>
 </filter>
</sequence>

```

If you need to validate only the request headers, you can disable the `enabledCheckBody` and `enabledCheckPathParams` properties by setting the value to `false`.

## Testing the default regex\_policy

You can test this feature by sending an SQL injection attack with the XML message body. The sample request and response will be as follows:

MessageResponse

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
<food>
 <name>Homestyle Breakfast</name>
 <price>drop table</price>
 <description>
 Two eggs, bacon or sausage, toast, and our ever-popular hash browns
 </description>
 <calories>950</calories>
</food>
</breakfast_menu>
```

```
<am:fault xmlns:am="http://wso2.org/apimanager">
 <am:code>400</am:code>
 <am:message>Bad Request</am:message>
 <am:description>SQL-Injection Threat detected in Payload</am:description>
</am:fault>
```

### ***Editing the sequence through registry artifacts***

To edit the existing sequence follow the steps below.

1. Log in to the Management Console.
2. Navigate to /\_system/governance/apimgt/customsequences/in/regex\_policy.xml
3. Edit the regex\_policy.xml file.
4. Go to the API Publisher and re-publish your API for the changes to take effect.

### ***Add a custom sequence***

You can also add custom sequences depending on the threats that you need to address. To add a custom sequence, do the following.

1. Create an xml file with your custome sequence, or edit and save the sequence given above.
2. Go to **Message Mediation Policies** as described in the previous section.
3. Click **Upload In Flow**.

#### Message Mediation Policies

Enable Message Mediation	<input checked="" type="checkbox"/> Check to select a message mediation policy to be executed in the message flow	
In Flow	None	<b>Upload In Flow</b>
Out Flow	None	<b>Upload Out Flow</b>
Fault Flow	None	<b>Upload Fault Flow</b>

4. Select and upload your custom sequence.



A sample sequence for server-side include injection attack is given below. Note that this example contains pre-defined regex patterns.

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="SSI_policy">
 <log level="custom">
 <property name="IN_MESSAGE" value="Regular_expression_policy"/>
 </log>
 <property name="threatType" expression="get-property('threatType')"/>
 value="SSI-Injection"/>
 <property name="regex" expression="get-property('regex')"/>
 value=".*#include.*|.*#exec.*|.*#echo.*|.*#config.*"/>
 <property name="enabledCheckBody"
 expression="get-property('checkBodyEnable')" value="true"/>
 <property name="enabledCheckHeaders"
 expression="get-property('enabledCheckHeaders')" value="true"/>
 <property name="enabledCheckPathParams"
 expression="get-property('enabledCheckPathParams')" value="true"/>
 <class
 name="org.wso2.carbon.apimgt.gateway.mediators.RegularExpressionProtector"
/>
 <filter source="get-property('threat')" regex=".error.*">
 <then>
 <sequence key="_threat_fault_"></sequence>
 </then>
 </filter>
</sequence>
```

Set the name of the sequence to **SSI-policy** or the name of the **policy** type. This name will be displayed in the UI once you have uploaded the In Flow sequence.

## Message Mediation Policies

Enable Message Mediation  In Flow  Out Flow  Fault Flow	<input checked="" type="checkbox"/> Check to select a message mediation policy to be executed in the message flow  <div style="border: 1px solid #ccc; padding: 5px; width: 300px;"> <b>SSI_policy</b> </div> <div style="border: 1px solid #ccc; padding: 5px; width: 300px;">           None         </div> <div style="border: 1px solid #ccc; padding: 5px; width: 300px;">           None         </div>
	<input type="button" value="Upload In Flow"/> <input type="button" value="Upload Out Flow"/> <input type="button" value="Upload Fault Flow"/>

Set the value of the `threatType` property to `SSI-injection` or the name of the `exception` type. It will be included in the response payload as shown below.

```
<am:fault xmlns:am="http://wso2.org/apimanager">
<am:code>400</am:code>
<am:message>Bad Request</am:message>
<am:description>SSI-Injection Threat detected in Query
Parameters</am:description>
</am:fault>
```

## Mutual SSL Support for API Gateway

In contrast to the usual one-way SSL authentication where a client verifies the identity of the server, in mutual SSL the server validates the identity of the client so that both parties trust each other. This builds a system that has a very tight security and avoids any requests made to the client to provide the username/password, as long as the server is aware of the certificates that belong to the client.

This section explains how to secure your backend by enabling mutual SSL between the API Gateway and your backend. To establish a secure connection with the backend service, API Manager needs to have the public key of the backend service in the truststore. Similarly, the backend service should have the public key of API Manager in the truststore.

- Export the certificates
- Configure API Manager to enable dynamic SSL profiles
- Test Mutual SSL between API Gateway and backend

### *Export the certificates*

1. Generate the keys for the backend. A sample command is given below.

```
keytool -keystore backend.jks -genkey -alias backend
```

The keystore will be generated in your target folder.

2. Export the certificate from the keystore. A sample command is given below.

```
keytool -export -keystore backend.jks -alias backend -file backend.crt
```

3. Import the generated backend certificate to the API Manager truststore file as shown below

```
keytool -import -file backend.crt -alias backend -keystore
<APIM_HOME>/repository/resources/security/client-truststore.jks
```

4. Export the public certificate from API Manager's keystore. The `<APIM_HOME>/repository/resources/security/wso2carbon.jks` file which is the default keystore shipped with WSO2 API Manager is used in this example. Use the command below to generate the certificate for the default keystore. Give the default password `wso2carbon` when prompted.

```
keytool -export -keystore wso2carbon.jks -alias wso2carbon -file
wso2PubCert.cert
```

To change the default keystore, generate a keystore file and copy it to the <APIM\_HOME>/repository/resources/security folder. After copying the keystore, generate the certificate as shown in step 2.

5. Import the generated certificate to your backend truststore.

```
keytool -import -file wso2PubCert.crt -alias wso2carbon -keystore
backend-truststore.jks
```

You have now successfully exported the certificates for mutual SSL.

#### ***Configure API Manager to enable dynamic SSL profiles***

To configure APIM for Dynamic SSL Profiles for HTTPS transport Sender, you need to create a new XML file <APIM\_HOME>/repository/deployment/server/multi\_ssl\_profiles.xml (this path is configurable) and copy the below configuration into it. This will configure client-truststore.jks as Trust Store for all connections to <localhost:port>.

```

<parameter name="customSSLProfiles">
 <!-- For SSL Handshake configure only trust store-->
 <profile>
 <servers>localhost:port</servers>
 <TrustStore>
 <Location>repository/resources/security/client-truststore.jks
 </Location>
 <Type>JKS</Type>
 <Password>wso2carbon</Password>
 </TrustStore>
 </profile>
 <!-- For Mutual SSL Handshake configure both trust store and key store-->
 <profile>
 <servers>10.100.5.130:9444</servers>
 <TrustStore>
 <Location>repository/resources/security/client-truststore.jks
 </Location>
 <Type>JKS</Type>
 <Password>wso2carbon</Password>
 </TrustStore>

 <KeyStore>
 <Location>repository/resources/security/wso2carbon.jks</Location>
 <Type>JKS</Type>
 <Password>xxxxxx</Password>
 <KeyPassword>xxxxxx</KeyPassword>
 </KeyStore>
 </profile>
</parameter>

```

To enable dynamic loading of this configuration, add below configurations to the Transport Sender configuration (`PassThroughHttpSSLSender`) of API Manager (`<APIM_HOME>/repository/conf/axis2.xml`). Set above file's path as the `filePath` parameter.

```

<parameter name="dynamicSSLProfilesConfig">
 <filePath>repository/deployment/server/multi_ssl_profiles.xml</filePath>
 <fileReadInterval>3600000</fileReadInterval>
</parameter>
<parameter name="HostnameVerifier">AllowAll</parameter>

```

Now both the backend service and ESB is configured to use default key stores and API Manager is configured to load dynamic SSL profiles. Restart API Manager.

You can start API Manager using the following options, to see the SSL debug logs.

```
-Djavax.net.debug=ssl:handshake
-Djavax.net.debug=all
-Djavax.net.debug=all:handshake:verbose
```

### **Test Mutual SSL between API Gateway and backend**

You can do the following to test your mutual SSL configurations

1. [Create an API](#)
2. [Subscribe to the API](#)
3. [Invoke the API](#)

## **Securing APIs**

APIs published on the WSO2 API Gateway are secured using OAuth2.0 by default. Any client application invoking a secure published API needs to have a valid subscription to the particular API and present a valid OAuth2.0 Access Token to the API Gateway. Please see steps 7 and 8 of the [quick start guide](#) to understand how you can subscribe an application to an API and how to get credentials for your application.

Once you have got the required credentials, namely the consumer key and consumer secret, for your application, you (application users) can get access tokens to invoke APIs that are subscribed to the particular application. To understand how you can get tokens for different types of applications, see [Token API](#).

### **Authentication**

HTTP Authorization header is the most common method of providing authentication information for REST APIs. The application needs to have the access token in an authorization header for the client application to authenticate the API that is being accessed. The format of the header is as follows.

[FormatExample](#)

### **Default Format**

```
Authorization : Bearer <access-token>
```

### **Format for customized authorization header**

```
<customized-authorization-header> : Bearer <access-token>
```

### **Example with default authorization header**

```
Authorization : Bearer NtBQkXoKElu0H1alfQ0DWfo6IX4a
```

### **Example with customized authorization header**

```
SERVER_AUTH : Bearer NtBQkXoKElu0H1alfQ0DWfo6IX4a
```

The string NtBQkXoKElu0H1a1fQ0DWfo6IX4a is a sample value of the access token that is being sent from the client application.

## Customizing the authorization header

If the value of the authorization header needs to be changed due to organizational policies, legacy APIs, or for any other reason, it can be done at three levels in WSO2 API Manager (WSO2 API-M) as described below.

- Server wide customization
- Tenant wide customization
- Per API customization

Server wide customization

Follow the instructions below to change the value of the authorization header at the server level configurations:

1. Navigate to the <API-M\_HOME>/repository/conf/api-manager.xml file and make the following changes.
  - a. Uncomment the <AuthorizationHeader> section and add the customized authorization header. You need to make this change on all the profiles of WSO2 API Manager.

[FormatExample](#)

```
<AuthorizationHeader>{authorization-header}</AuthorizationHeader>
```

```
<AuthorizationHeader>SERVER_AUTH</AuthorizationHeader>
```

- b. Add the custom authorization header to the list of allowed headers defined in the <corsConfiguration> <Access-Control-Allow-Headers> section.

[FormatExample](#)

```
<Access-Control-Allow-Headers>{authorization-header}</Access-Control-Allow-Headers>
```

```
<Access-Control-Allow-Headers>authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction,SERVER_AUTH</Access-Control-Allow-Headers>
```

2. Restart the WSO2 API Manager server to reload the changes.
3. If you have already published APIs, [sign in to the API Publisher](#) and republish those APIs.

▼ [Click here to test out the customized authorization header at the server level.](#)

**Before you begin**, deploy the sample PizzaShackAPI as explained in the [Quick Start Guide](#), as the following example is based on that API.

[Sample cURL Request](#)[cURL Request Format](#)[Sample Response](#)

```
curl -H "<customized-authorization-header>: Bearer <access-token>" -H
"accept: application/json"
"https://<server-IP>:<port>/pizzashack/1.0.0/menu" -k -v
```

```
curl -H "SERVER_AUTH: Bearer 3c536e3f-397c-3df9-a89c-9c40efedfa9e" -H
"accept: application/json"
"https://10.100.0.112:8243/pizzashack/1.0.0/menu" -k -v
```

```

> GET /pizzashack/1.0.0/menu HTTP/1.1
> Host: 10.100.0.112:8243
> User-Agent: curl/7.54.0
> accept: application/json
> SERVER_AUTH: Bearer 3c536e3f-397c-3df9-a89c-9c40efedfa9e
>
< HTTP/1.1 200 OK
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: GET
< Access-Control-Allow-Headers:
authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction,SERVER
_AUTH
< Content-Type: application/json
< Date: Fri, 19 Jan 2018 06:02:29 GMT
< Transfer-Encoding: chunked
<
[{"name": "BBQ Chicken Bacon", "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions in barbecue sauce", "price": "22.99", "icon": "/images/6.png"}, {"name": "Chicken Parmesan", "description": "Grilled chicken, fresh tomatoes, feta and mozzarella cheese", "price": "21.99", "icon": "/images/1.png"}, {"name": "Chilly Chicken Cordon Bleu", "description": "Spinash Alfredo sauce topped with grilled chicken, ham, onions and mozzarella", "price": "28.99", "icon": "/images/10.png"}, {"name": "Double Bacon 6Cheese", "description": "Hickory-smoked bacon, Julienne cut Canadian bacon, Parmesan, mozzarella, Romano, Asiago and and Fontina cheese", "price": "25.99", "icon": "/images/9.png"}, {"name": "Garden Fresh", "description": "Slices onions and green peppers, gourmet mushrooms, black olives and ripe Roma tomatoes", "price": "14.99", "icon": "/images/3.png"}, {"name": "Grilled Chicken Club", "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions topped with Roma tomatoes", "price": "15.99", "icon": "/images/8.png"}, {"name": "Hawaiian BBQ Chicken", "description": "Grilled white chicken, hickory-smoked bacon, barbecue sauce topped with sweet pine-apple", "price": "19.99", "icon": "/images/7.png"}, {"name": "Spicy Italian", "description": "Pepperoni and a double portion of spicy Italian sausage", "price": "14.99", "icon": "/images/2.png"}, {"name": "Spinach Alfredo", "description": "Rich and creamy blend of spinach and garlic Parmesan with Alfredo sauce", "price": "20.99", "icon": "/images/5.png"}, {"name": "Tuscan Six Cheese", "description": "Six cheese blend of mozzarella, Parmesan, Romano, Asiago and Fontina", "price": "22.99", "icon": "/images/4.png"}]

```

Note that the customized authorization header appears in the Store UI as well.

PizzaShackAPI - 1.0.0

#### Tenant wide customization

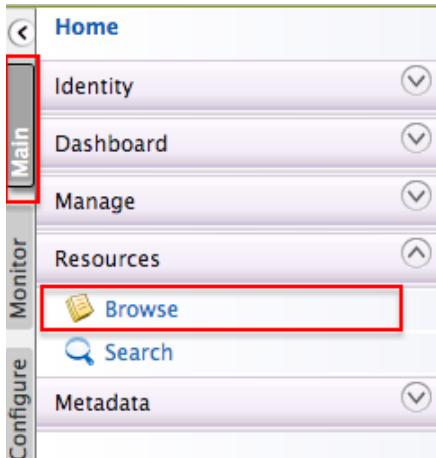
Each tenant can define their own name for the authorization header by following the instructions below:

The tenant wide authorization header customizations take precedence over the server wide authorization header customizations.

1. Sign in to the [WSO2 API Manager Management Console](https://<server-host>:9443/carbon) with your tenant credentials.  
<https://<server-host>:9443/carbon>

For information on creating a tenant, see [Managing Tenants](#).

2. Click **Main**, then click **Browse** which is under **Resources**.



3. Enter `/_system/config/apimgt/applicationdata/tenant-conf.json` as the location and click **Go** to access the `tenant-conf.json` file that is in the WSO2 Registry.
4. Click **Edit as text** to be able to edit the JSON file.
5. Add the following configuration with the customized authorization header to the JSON file and save the file.

[FormatExampleSample JSON](#)

```
"AuthorizationHeader" : "<tenant-authorization-header>"
```

```
"AuthorizationHeader" : "TENANT_AUTH"
```

```
{
 "EnableMonetization":false,
 "IsUnlimitedTierPaid":false,
 "ExtensionHandlerPosition":"bottom",
 "RESTAPIScopes":{
 "Scope":[
 {
 "Name":"apim:api_publish",
 "Roles":"admin,Internal/publisher"
 },
 {
 "Name":"apim:api_create",
 "Roles":"admin,Internal/creator"
 },
 {
 "Name":"apim:api_view",
 "Roles":"admin,Internal/publisher,Internal/creator"
 },
 {
 "Name":"apim:subscribe",
 "Roles":"admin,Internal/subscriber"
 },
 {
 "Name":"apim:tier_view",
 "Roles":"admin,Internal/publisher,Internal/creator"
 },
 {
 "Name":"apim:tier_manage",
 "Roles":"admin"
 },
 {
 "Name":"apim:bl_view",
 "Roles":"admin"
 },
 {
 "Name":"apim:bl_manage",
 "Roles":"admin"
 },
 {
 "Name":"apim:subscription_view",
 "Roles":"admin,Internal/creator"
 },
 {
 "Name":"apim:subscription_block",
 "Roles":"admin"
 }
]
 }
}
```

```

 "Roles" :"admin,Internal/creator"
 },
{
 "Name" :"apim:mediation_policy_view",
 "Roles" :"admin"
},
{
 "Name" :"apim:mediation_policy_create",
 "Roles" :"admin"
},
{
 "Name" :"apim:api_workflow",
 "Roles" :"admin"
}
]
},
"NotificationsEnabled": "false",
"Notifications": [
{
 "Type": "new_api_version",
 "Notifiers": [
{

```

"Class": "org.wso2.carbon.apimgt.impl.notification.NewAPIVersionEmailNotifier",

"ClaimsRetrieverImplClass": "org.wso2.carbon.apimgt.impl.token.DefaultClaimsRetriever",

"Title": "Version \$2 of \$1 Released",

"Template": "<html> <body> <h3 style='color:Black;'>We're happy to announce the arrival of the next major version \$2 of \$1 API which is now available in Our API Store.</h3><a href='https://localhost:9443/store'>Click here to Visit WSO2 API Store</a></body></html>"

}

]

}

]

],
"DefaultRoles": {
 "PublisherRole": {
 "CreateOnTenantLoad": true,
 "RoleName": "Internal/publisher"
 },
 "CreatorRole": {
 "CreateOnTenantLoad": true,
 "RoleName": "Internal/creator"
 },
 "SubscriberRole": {
 "CreateOnTenantLoad": true
 }
}

```
 } ,
 "AuthorizationHeader" : "TENANT_AUTH"
}
```

6. If you have already published APIs, sign in to the API Publisher using your tenant credentials and republish those APIs.

▼ [Click here to test out the customized authorization header at the tenant level.](#)

**Before you begin**, deploy the sample PizzaShackAPI as explained in the [Quick Start Guide](#), as the following example is based on that API.

#### Sample cURL Request

```
curl -H "<customized-authorization-header>: Bearer <access-token>" -H
"accept: application/json"
"https://<server-IP>:<port>/pizzashack/1.0.0/menu" -k -v
```

```
curl -H "TENANT_AUTH: Bearer 3c536e3f-397c-3df9-a89c-9c40efedfa9e" -H
"accept: application/json"
"https://10.100.0.112:8243/pizzashack/1.0.0/menu" -k -v
```

```

> GET /t/test.com/pizzashack/1.0.0/menu HTTP/1.1
> Host: 10.100.0.112:8243
> User-Agent: curl/7.54.0
> accept: application/json
> TENANT_AUTH: Bearer 57aca573-6fa8-3b03-a340-dd6628b44fe8
>
< HTTP/1.1 200 OK
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: GET
< Access-Control-Allow-Headers:
authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction,SERVER
_AUTH,TENANT_AUTH
< Content-Type: application/json
< Date: Fri, 19 Jan 2018 06:35:47 GMT
< Transfer-Encoding: chunked
<
[{"name": "BBQ Chicken Bacon", "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions in barbecue sauce", "price": "22.99", "icon": "/images/6.png"}, {"name": "Chicken Parmesan", "description": "Grilled chicken, fresh tomatoes, feta and mozzarella cheese", "price": "21.99", "icon": "/images/1.png"}, {"name": "Chilly Chicken Cordon Bleu", "description": "Spinash Alfredo sauce topped with grilled chicken, ham, onions and mozzarella", "price": "28.99", "icon": "/images/10.png"}, {"name": "Double Bacon 6Cheese", "description": "Hickory-smoked bacon, Julienne cut Canadian bacon, Parmesan, mozzarella, Romano, Asiago and and Fontina cheese", "price": "25.99", "icon": "/images/9.png"}, {"name": "Garden Fresh", "description": "Slices onions and green peppers, gourmet mushrooms, black olives and ripe Roma tomatoes", "price": "14.99", "icon": "/images/3.png"}, {"name": "Grilled Chicken Club", "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions topped with Roma tomatoes", "price": "15.99", "icon": "/images/8.png"}, {"name": "Hawaiian BBQ Chicken", "description": "Grilled white chicken, hickory-smoked bacon, barbecue sauce topped with sweet pine-apple", "price": "19.99", "icon": "/images/7.png"}, {"name": "Spicy Italian", "description": "Pepperoni and a double portion of spicy Italian sausage", "price": "14.99", "icon": "/images/2.png"}, {"name": "Spinach Alfredo", "description": "Rich and creamy blend of spinach and garlic Parmesan with Alfredo sauce", "price": "20.99", "icon": "/images/5.png"}, {"name": "Tuscan Six Cheese", "description": "Six cheese blend of mozzarella, Parmesan, Romano, Asiago and Fontina", "price": "22.99", "icon": "/images/4.png"}]

```

Note that the customized authentication header appears in the Store UI as well.

PizzaShackAPI - 1.0.0

**Version:** 1.0.0  
**By:** Jane Roe  
**Updated:** 19/Jan/2018 12:01:53 PM IST  
**Status:** PUBLISHED  
**Rating:** ★★★★★ 0

**Applications**: DefaultApplication  
**Tiers**: Unlimited

**Subscribe**

Overview API Console Documentation SDKs

**Notice**  
Please subscribe to the API to generate an access token. If you already have an access token, please provide it below.

Set Request Header **TENANT\_AUTH : Bearer** Access Token

Swagger (/swagger.json)

Per API customization

The API Publisher application allows an API Developer or Product Manager to specify the name of the authorization header when creating or modifying an API.

The customized authorization header defined per API takes precedence over the customized authorization headers that are defined server and tenant wide.

Follow the instructions below to add a customized authorization header for an API:

1. Sign in to the Publisher.  
<https://<server-host>:9443/publisher>
2. When creating or updating an API, define the customized authorization header in the **Manage** tab.  
UI without tenants UI with tenants

1 Design    2 Implement    3 Manage

**Configurations**

Make this the Default version:  No default version defined for the current API

Transports: \*  HTTPS  HTTP

Response Caching:

Authorization Header: **Authorization**

The screenshot shows the 'Manage' tab of the WSO2 API Manager interface. Under the 'Configurations' section, there are several settings:

- Make this the Default version:**  (No default version defined for the current API)
- Transports:**  HTTPS  HTTP
- Response Caching:** Disabled
- Subscriptions:** Available to current tenant only
- Authorization Header:** PER\_API\_AUTH (highlighted with a red border)

The subscription field appears only if you have any tenants.

### 3. Save and Publish the API.

Click here to test out the customized authorization header at the API level.

**Before you begin,** deploy the sample PizzaShackAPI as explained in the [Quick Start Guide](#), as the following example is based on that API.

#### Sample cURL Request

```
curl -H "<customized-authorization-header>: Bearer <access-token>" -H
"accept: application/json"
"https://<server-IP>:<port>/pizzashack/1.0.0/menu" -k -v
```

```
curl -H "PER_API_AUTH: Bearer 045d4999-8a9e-3897-945b-e7bea13d78fe" -H
"accept: application/json"
"https://10.100.0.112:8243/pizzashack/1.0.0/menu" -k -v
```

```

> GET /pizzashack/1.0.0/menu HTTP/1.1
> Host: 10.100.0.112:8243
> User-Agent: curl/7.54.0
> PER_API_AUTH: Bearer 045d4999-8a9e-3897-945b-e7bea13d78fe
> accept: application/json
>
< HTTP/1.1 200 OK
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: GET
< Access-Control-Allow-Headers:
authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction,SERVER
_AUTH,PER_API_AUTH
< Content-Type: application/json
< Date: Fri, 19 Jan 2018 08:23:31 GMT
< Transfer-Encoding: chunked
<
[{"name": "BBQ Chicken Bacon", "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions in barbecue sauce", "price": "22.99", "icon": "/images/6.png"}, {"name": "Chicken Parmesan", "description": "Grilled chicken, fresh tomatoes, feta and mozzarella cheese", "price": "21.99", "icon": "/images/1.png"}, {"name": "Chilly Chicken Cordon Bleu", "description": "Spinash Alfredo sauce topped with grilled chicken, ham, onions and mozzarella", "price": "28.99", "icon": "/images/10.png"}, {"name": "Double Bacon 6Cheese", "description": "Hickory-smoked bacon, Julienne cut Canadian bacon, Parmesan, mozzarella, Romano, Asiago and and Fontina cheese", "price": "25.99", "icon": "/images/9.png"}, {"name": "Garden Fresh", "description": "Slices onions and green peppers, gourmet mushrooms, black olives and ripe Roma tomatoes", "price": "14.99", "icon": "/images/3.png"}, {"name": "Grilled Chicken Club", "description": "Grilled white chicken, hickory-smoked bacon and fresh sliced onions topped with Roma tomatoes", "price": "15.99", "icon": "/images/8.png"}, {"name": "Hawaiian BBQ Chicken", "description": "Grilled white chicken, hickory-smoked bacon, barbecue sauce topped with sweet pine-apple", "price": "19.99", "icon": "/images/7.png"}, {"name": "Spicy Italian", "description": "Pepperoni and a double portion of spicy Italian sausage", "price": "14.99", "icon": "/images/2.png"}, {"name": "Spinach Alfredo", "description": "Rich and creamy blend of spinach and garlic Parmesan with Alfredo sauce", "price": "20.99", "icon": "/images/5.png"}, {"name": "Tuscan Six Cheese", "description": "Six cheese blend of mozzarella, Parmesan, Romano, Asiago and Fontina", "price": "22.99", "icon": "/images/4.png"}]

```

## Multi Factor Authentication (MFA) for Publisher and Developer Portals

Multi factor authentication provided a many-layered security for a user identity where even if one factor is compromised, an attacker cannot gain full access to the target due to the remaining authentication factors.

To configure MFA for Publisher and Store with WSO2 Identity Server, please follow the steps below:

1. Configure Single Sign On (SSO) with WSO2 Identity Server. Please see the documentation on [Configuring Identity Server as IDP for SSO](#).
2. Configure Multi Factor Authentication in WSO2 Identity Server. Please see [Configuring Multi-factor Authentication for WSO2 IS](#).

## Working with Throttling

The following sections explain how to work with throttling in the API Manager:

- [Introducing Throttling Use-Cases](#)
- [Setting Maximum Backend Throughput Limits](#)
- [Setting Throttling Limits](#)
- [Adding New Throttling Policies](#)
- [Managing Throttling](#)
- [Enforcing Throttling to an API](#)
- [Engaging a new Throttling Policy at Runtime](#)
- [Engaging Multiple Throttling Policies to a Single API](#)

### Introducing Throttling Use-Cases

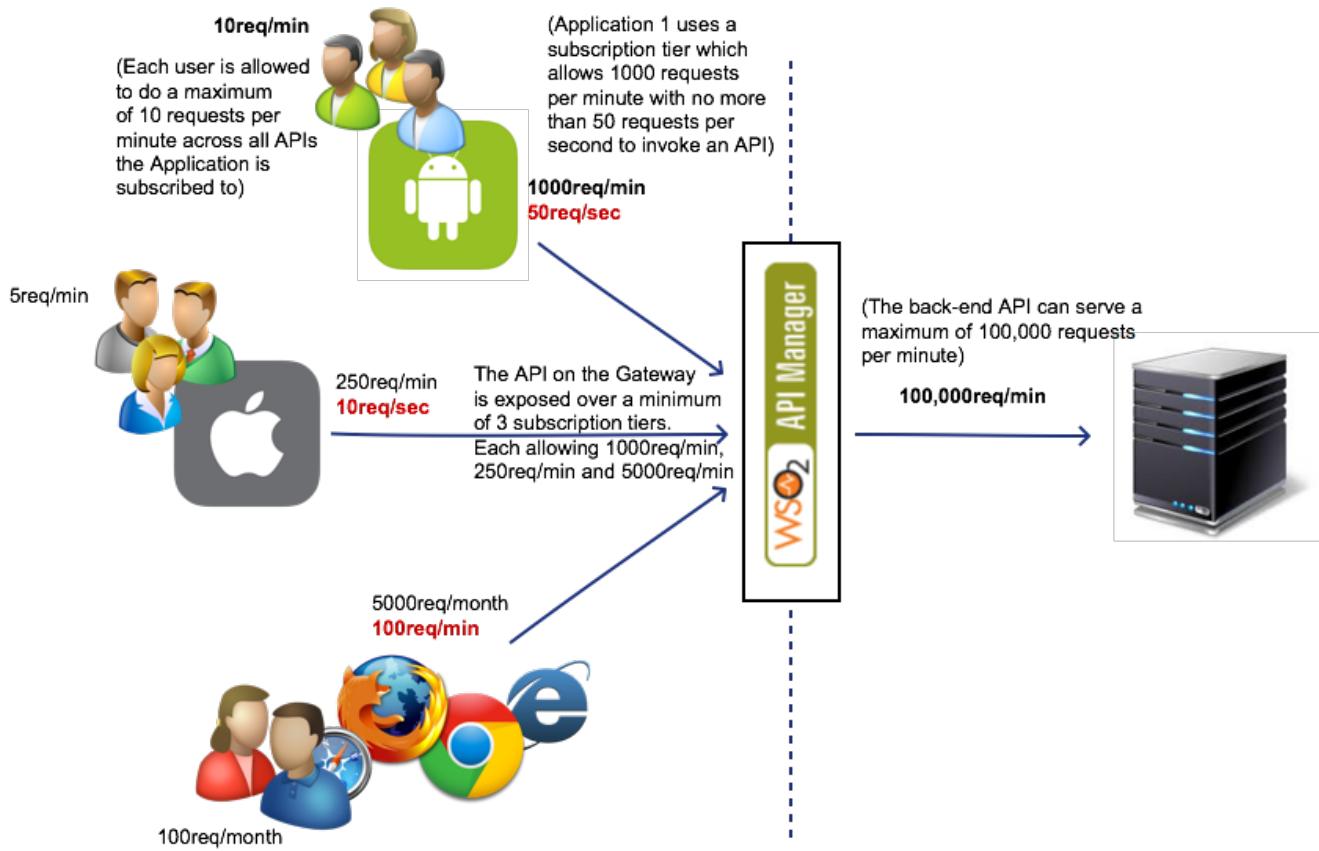
▼ [Click to see what throttling is...](#)

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 certain types of denial of service (DOS) attacks
- 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 API Gateway architecture model, which solves the API management problem, comprises the following:

- The back-end services/systems hosting the actual business logic
- The APIs in the API Gateway that proxy the back-end services
- The applications that consume the APIs in the API Gateway
- The users of the applications



The following sections describe the type of throttling policy applicable to each of the above areas and why the relevant stakeholders must consider each of them carefully.

- Implications on back-end services/systems
- Implications on the APIs in the Gateway
- Advanced throttling policies: API Publisher
- Implications on applications that consume APIs

#### ***Implications on back-end services/systems***

#### **Maximum backend throughput (Applies per API): API Publisher**

According to the API Gateway architecture, an API in the Gateway is actually a proxy to an actual service hosted within your organization, cloud, etc. This usually means that there is a physical capacity that your backend services can handle. Although you expose your API on defined limits (subscription tiers), as the number of applications that consume your API grows, the number of requests being served by your API rise, which in turn means that the number of requests served by your backend system rise as well. Therefore, although none of the applications may exceed their own allocated quotas, their combined load might hit the maximum capacity that can be handled by your backend system. To prevent your backend system from getting overloaded, the limits enforced by the **Maximum Backend Throughput** in the API act as a hard stop on the number of requests that your backend system can serve within a given time period. The counters maintained when evaluating the maximum backend throughput are shared across all nodes of the Gateway cluster and apply across all users using any application that accesses that particular API. For information on how to specify maximum backend throughput limits, see [Setting Maximum Backend Throughput Limits](#).

#### ***Implications on the APIs in the Gateway***

#### **Subscription tiers: API Publisher**

When an API Publisher publishes an API to be consumed by applications, s/he can choose to make the API available over different limits. For example, the **Gold** tier allows an application to access the API at 5000 requests per minute while a **Silver** tier allows an application to access the API at 2000 requests per minute. For information on how to define a throttling tier to an API, see [API-level throttling \(API publisher\)](#).

The subscription tiers are used to gain monetary value for the API; you can charge more from app developers who require larger quotas of your API's functionality and lesser from developers who require less. The limits can be enforced either by the number of requests over time (5000 req/min) or the amount of data bandwidth over time (500 mb/hour). The limits enforced by subscription tiers are applied across all users of the application that use that particular API and can be considered as a shared quota among all users of an application that access that API. When using a cluster of Gateway nodes, the counters maintained while evaluating the subscription tiers are shared across all nodes. For information on how to define a subscription tier to an API, see [Subscription-level throttling \(API publisher\)](#).

## Burst Control

Burst control limits are enforced for subscription tiers in order to distribute the load across the specified time period. For example, if you have a subscription tier that allows you to send 1000 requests per hour, you can ensure that a particular application does not consume the full quota of 1000 requests within the first 2 minutes by setting a burst control limit within the subscription tier allowing only a maximum of 25 requests per minute. Therefore, the time periods set for burst control limits must always be smaller than the time period specified for its corresponding subscription tier. Burst control limits can be set only to control the number of requests for a given period of time and does not allow you to control the data bandwidth for a given time period. The burst control limits are enforced for each individual Gateway node. Although the request counters are replicated across the cluster, since burst control time periods are usually quite small, the replication frequency can be quite high compared to the burst rate of incoming requests. Therefore, it is safe to assume that the burst control values are applied on a per-node basis. For information on how to define burst control limits, see [Rate limiting \(burst control\)](#).

### *Advanced throttling policies: API Publisher*

Advanced throttling policies allow an API Publisher to control access to his API resources using advanced rules. Advanced policies include the ability to apply limits by filtering requests based on the following properties and their combinations. The counters maintained when evaluating advanced throttling policies are shared across all nodes in the Gateway cluster.

- IP address and address range
- HTTP request headers
- JWT claims
- Query parameters

Let's look at how each of these can be important for serving requests through your APIs.

### **IP address and address range**

You can control/restrict access to your API or its selected resources for a given IP address or address range. For example, if you need to grant permission for internal applications to consume a larger quota of your API resource than your external consumers, you can define an advanced policy with higher limits for your internal IP address range and lower limits for the rest. For information on how to define IP throttling, see [Advanced throttling \(API publisher\)](#).

### **HTTP request headers**

Advanced policies allow you to apply limits to APIs by filtering requests based on HTTP headers. For example, assume you need to apply a special limit for JSON requests. To do that, you can filter JSON messages by using a policy that inspects the HTTP request headers and checks if the Content-Type header is application/json and apply a special limit for those requests while allowing a default value for the rest.

Here is a sample for configuring JWT claim condition by considering the "Content-Type" header.

**Header Condition Policy**

This configuration is used to throttle based on Headers.

Header Name:*	Content-Type
Param Value:*	application/json

**Add**

Invert Condition:

## Execution Policy

Request Count :	Request Count
Request Count: *	5
Time: *	1 Minute(s)

## JWT claims

A JWT claim contains meta information of an API request. It can include application details, API details, user claims, etc. Advanced throttling policies based on JWT claims allow you to filter requests by JWT claim values and apply limits for requests. For example, if you need to allow special limits for users in a specific user role, you can create an advanced policy that checks for a particular regular expression on the role claim of the user and apply special limits for the ones that match.

Here is a sample for configuring JWT claim condition by considering the version of the API ("<http://wso2.org/claims/version>").

**JWT Condition Policy**

This configuration is used to define JWT claims conditions

Claim Name:*	http://wso2.org/claims/version
Claim Value:*	1.0.0

**Add**

Invert Condition:

## Execution Policy

Request Count :	Request Count
Request Count: *	5
Time: *	1 Minute(s)

## Query parameters

Filtering based on query parameters almost always apply to HTTP GET requests when doing search type of operations. For example, if you have a search API with `category` as a query parameter, you can have different limits for searching different categories.

IP Condition

Header Condition

**Query Param Condition**

JWT Claim Condition

Query Param Condition Policy

This configuration use to throttle based on query parameters.

Param Name: \* category

Param Value: \* sales

Add

Invert Condition:

On

### Execution Policy

Request Count :	Request Count
Request Count: *	5
Time: *	1 Minute(s)

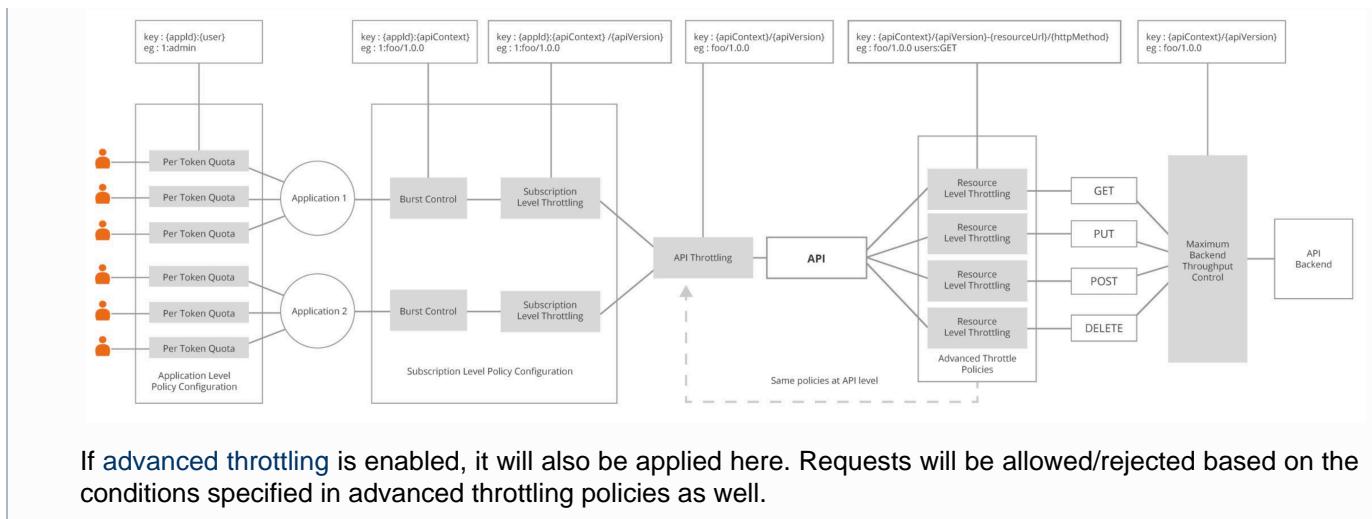
Eg : 'sales' category can be allocated with more requests than 'hr' category

#### *Implications on applications that consume APIs*

#### Per token quota: Application Developer

When an application developer subscribes their application to an API, they select a tier (limit) for their application to invoke the API. This limit applies across all users of the application when accessing the particular API. To ensure that a fair distribution of the quota is available among all the users, it is important to consider setting a per user quota for the application, since a user is identified by a token (in OAuth2.0, this limit is known as the per token quota). It is important to note that the limit enforced by this setting applies to a single user (token) accessing all APIs of the application. The counters maintained when evaluating a per token quota are shared across all nodes in the Gateway cluster. For information on how to define a throttling tier to an application, see [Application-level throttling \(application developer\)](#).

**The below diagram shows how throttle policies are applied at different levels.**



## Setting Maximum Backend Throughput Limits

The maximum backend throughput setting limits the total number of calls a particular API in the API Manager is allowed to make to the backend. While the [other throttling levels](#) define the quota the API invoker gets, they do not ensure that the backend is protected from overuse. The maximum backend throughput setting limits the quota the backend can handle. The counters maintained when evaluating the maximum backend throughput are shared across all nodes of the Gateway cluster and apply across all users using any application that accesses that particular API.

You set a maximum backend throughput using the **Manage** tab of the **API Publisher** when creating or editing an API. Under the **Throttling Settings** section, select the **Specify** option for the maximum backend throughput and specify the limits of the Production and Sandbox endpoints separately, as the two endpoints can come from two servers with different capacities.

Alternatively, you can go to the synapse configuration of the API , which is created at the point of creating the API, in the <API-M\_HOME>/repository/deployment/server/synapse-configs/default/api (for a tenant

directory, and specify the maximum backend throughput by modifying the synapse configuration. Maximum backend throughput limits are usually counted over a duration of 1 second, but you can increase the duration using the `productionUnitTime` and `sandboxUnitTime` properties in the API's synapse configuration. For example,

If you want to accept only 600 requests by the production endpoint within a minute of duration and 700 total requests within 5 minutes by sandbox endpoint you can modify the synapse configuration as below.

```

<handlers>
 <handler
 class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler"
 >
 <property name="apiImplementationType" value="ENDPOINT" />
 </handler>
 <handler
 class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
 <handler
 class="org.wso2.carbon.apimgt.gateway.handlers.throttling.ThrottleHandler">
 <property name="id" value="A"/>
 <property name="productionMaxCount" value="600" />
 <property name="productionUnitTime" value="60000" />
 <property name="sandboxMaxCount" value="700" />
 <property name="sandboxUnitTime" value="300000" />
 <property name="policyKey" value="gov:/apimgt/applicationdata/tiers.xml" />
 </handler>
 <handler
 class="org.wso2.carbon.apimgt.usage.publisher.APIMgtUsageHandler" />
 ...
</handlers>
```

Note that the duration is specified in milliseconds.

## Setting Throttling Limits

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 certain types of denial of service (DOS) attacks
- 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

You can define throttling in the API, application, resource and subscription levels. The final throttle 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 are 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.

### Different levels of throttling

It is possible to throttle requests for each tier based on the request count per unit time or the amount of data (bandwidth) transferred through the Gateway per unit time.

Request Count  Request Bandwidth

Request Count \*

Unit Time \*  Minute(s)

Let's take a look at the different levels of throttling:

- Subscription-level throttling (API publisher)
- Subscription-level throttling (API subscriber)
- Advanced throttling (API publisher)
- Application-level throttling (application developer)

### Subscription-level throttling (API publisher)

Subscription-level throttling tiers are also defined when managing APIs using the API Publisher portal.

The screenshot shows the API Publisher portal interface. At the top, there are three tabs: 'Design' (selected), 'Implement', and 'Manage'. Below these tabs, there are two main sections: 'Configurations' and 'Throttling Settings'.

**Configurations**

- Make this the Default Version:  (disabled)
- No default version defined for the current API
- Transports:  HTTPS  HTTP
- Response Caching:

**Throttling Settings**

- Maximum Backend Throughput:  Unlimited  Specify
- Subscription Tiers:  **Unlimited**: Allows unlimited requests
- Gold**: Allows 5000 requests per minute
- Silver**: Allows 2000 requests per minute
- Bronze**: Allows 1000 requests per minute

Subscription-level Throttling tiers

The default throttling tiers are as follows:

- **Bronze**: 1000 requests per minute
- **Silver**: 2000 requests per minute
- **Gold**: 5000 requests per minute
- **Unlimited**: Allows unlimited access (you can disable the Unlimited tier by editing the `<EnableUnlimitedTier>` element in `<ThrottlingConfigurations>` node of the `<API-M_HOME>/repository/conf/api-manager.xml` file)

In API Manager 2.0.0 onwards, **Advanced Throttling** is enabled by default with following configuration in `<API-M_HOME>/repository/conf/api-manager.xml`.

```
<ThrottlingConfigurations>
 <EnableAdvanceThrottling>true</EnableAdvanceThrottling>

<ThrottlingConfigurations>
```

If you are disabling **Advanced Throttling** in any case by setting the value of **<EnableAdvanceThrottling> false**, Advanced Throttling is disabled and basic Throttling mechanism is enabled thereafter. In such a scenario, if you want to disable the Unlimited Throttling tier of basic Throttling configurations, you need to disable it under **<TierManagement>** by setting **<EnableUnlimitedTier> to false**.

```
<TierManagement>
 <EnableUnlimitedTier>true</EnableUnlimitedTier>
</TierManagement>
```

It is also possible to specify a bandwidth per unit time instead of a number of requests. This can be done by an API Manager administrator. For information on editing the values of the existing tiers, defining new tiers and specifying a bandwidth per unit time, see [Adding a new subscription-level throttling tier](#).

**Note** that when you edit an API with active subscribers, certain things like tier changes do not get automatically reflected to the subscribers. For such changes to take effect, the subscribers should resubscribe to the API and regenerate the access token.

### **Rate limiting (burst control)**

With rate limiting, you can define tiers with a combination of, for example, a 1000 requests per day and 10 requests per second. Users are then throttled at two layers. Enforcing a rate limit protects the backend from sudden request bursts and controls the usage at a subscription and API level.

For instance, if there's a subscription level policy enforced over a long period, you may not want users to consume the entire quota within a short time span. Sudden spikes in usage or attacks from users can also be handled via rate limiting. You can define a spike arrest policy when the subscription level tier is created. For more information on using rate limiting in suscription tiers, refer [Adding a new subscription-level throttling tier](#).

**Spike Arrest Policy** is used to protect the API backend against large number of traffic spikes and DoS attacks. Unlike setting one definite throttling tier (Quota), it helps to limit the sudden increase of number of requests at any point in time.

As an example, if we specify a quota policy as 20 requests per minute, it is possible to send all 20 requests in first few seconds in one minute so that we cannot limit it. By defining a spike arrest policy as 10 requests per second, it equally scatter the the number of requests over the given one minute. Therefore by doing rate limiting we can protect the backend from sudden spikes and DoS attacks through spike arrest policy.

For each subscription level throttle key, a WS policy is created on demand. The request count is calculated and throttling occurs at the node level. If you are using a clustered deployment, the counters are replicated across the cluster.

### **Subscription-level throttling (API subscriber)**

After subscription-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 (out of those enabled for subscribers) they are interested in, as

shown below:

The screenshot shows the WSO2 API Publisher interface. On the left, there's a large red square icon with a white letter 'P'. To its right, the API details are listed:

- Version:** 1.0.0
- By:** admin
- Updated:** 22/Jul/2016 17:12:01 PM IST
- Status:** PUBLISHED
- Rating:** ★★★★☆☆

On the right, there's a section for subscription settings:

- Applications:** TestApp
- Tiers:** Bronze

A blue 'Subscribe' button is located at the bottom of this section, which is highlighted with a red border.

According to the selected tiers, the subscribers are granted a maximum number of requests to the API.

### Advanced throttling (API publisher)

Advanced throttling policies are applied when we are Publishing an API. It can be further divided into two levels according to the applicability. Those are,

1. API-Level Throttling
2. Resource-Level Throttling

#### *API Level Throttling*

API-level policies are defined when managing APIs using the API Publisher by selecting **Apply per API** under **Advanced Throttling** policies as shown below.

The screenshot shows the WSO2 API Manager interface in the 'Implement' tab. The top navigation bar has three tabs: 'Design' (blue), 'Implement' (white), and 'Manage' (light blue). The 'Implement' tab is active.

### Configurations

Make this the Default Version:  No default version defined for the current API

Transports: \*  HTTPS  HTTP

Response Caching:  Disabled

### Throttling Settings

Maximum Backend Throughput :  Unlimited  Specify

Subscription Tiers: \*  **Unlimited** : Allows unlimited requests  
 **Gold** : Allows 5000 requests per minute  
 **Silver** : Allows 2000 requests per minute  
 **Bronze** : Allows 1000 requests per minute

Advanced Throttling Policies:  **Apply to API**  Apply per Resource

**Unlimited**  
50KPerMin  
20KPerMin  
10KPerMin

each throttling setting.

This will specify the maximum throttling level per minute for the API .  
**Resource-Level Throttling**

An API is made up of one or more resources. Each resource handles a particular type of request and is similar to a method (function) in a larger API. You can use this method when handling a large number of request in resource level such as Financial transactions. For example, Imagine API have two resources and one resource take more request than other you do not need to throttle it in API level in that case you can use this. Resource-level throttling tiers are set to HTTP verbs of an API's resources. You can apply resource-level throttling thorough the **Manage** tab as shown below:

1 Design      2 Implement      3 Manage

### Configurations

Make this the Default Version:  No default version defined for the current API

Transports: \*  HTTPS  HTTP

Response Caching:

### Throttling Settings

Maximum Backend Throughput :  Unlimited  Specify

Subscription Tiers: \*  **Unlimited** : Allows unlimited requests  
 **Gold** : Allows 5000 requests per minute  
 **Silver** : Allows 2000 requests per minute  
 **Bronze** : Allows 1000 requests per minute

Advanced Throttling Policies:  Apply to API  Apply per Resource Select Policy per Resource

ⓘ Refer documentation for more information about each throttling setting.

### Select Policy per Resource

Resource	Policy
POST /order	<input checked="" type="checkbox"/> <b>Unlimited</b> 50KPerMin 20KPerMin 10KPerMin Unlimited
GET /menu	Unlimited
PUT /order/{orderId}	Unlimited
GET /order/{orderId}	Unlimited
DELETE /order/{orderId}	Unlimited

Close

Advanced Throttling tiers

The default throttling tiers are as follows:

- **10KPerMin**: 10,000 requests per minute
- **20KPerMin**: 20,000 requests per minute
- **50KPerMin**: 50,000 requests per minute
- **Unlimited**: Unlimited access (you can disable the Unlimited tier by editing the `<EnableUnlimitedTier>` element in `<ThrottlingConfigurations>` node of the `<APIM_HOME>/repository/conf/api-manage.xml` file)

It is also possible to specify a bandwidth per unit time instead of a number of requests. This can be done through the Admin Portal of API Manager. For information on editing the values of the existing tiers, defining new tiers and specifying a bandwidth per unit time, see [Adding a new advanced throttling policy](#).

### Application-level throttling (application developer)

Application-level throttling tiers are defined at the time an application is created in the API Store as shown below. The limits are restricted per token for a specific application.

Add Application

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name*	TestApp
Description	
Per Token Quota	Unlimited <ul style="list-style-type: none"> <li><b>Unlimited</b> Allows unlimited requests</li> <li>API request quota per access token. Allocated quota based APIs of the application.</li> </ul> <ul style="list-style-type: none"> <li>50PerMin Allows 50 request per minute</li> <li>20PerMin Allows 20 request per minute</li> <li>10PerMin Allows 10 request per minute</li> </ul>

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.

Application-level Throttling tiers

The default throttling levels are as follows:

- **10PerMin:** 10 requests per minute
- **20PerMin:** 20 requests per minute
- **50PerMin:** 50 requests per minute
- **Unlimited:** Unlimited access. The **Default Application**, which is provided out of the box has the tier set to Unlimited.

It is also possible to specify a bandwidth per unit time instead of a number of requests. This can be done through the Admin Portal of API Manager. For information on editing the values of the existing tiers, defining new tiers and specifying a bandwidth per unit time, see [Adding a new application-level throttling tier](#).

### Adding New Throttling Policies

WSO2 API Manager admins can add new throttling policies and define extra properties to the throttling policies. To get started, click the level of throttling that you want to add a new policy to:

- Adding a new advanced throttling policy
- Adding a new application-level throttling tier
- Adding a new subscription-level throttling tier

To make changes in the throttling configurations, set the `EnableAdvancedThrottling` parameter in `api-m`

anager.xml. This parameter is set to true by default. If you set it to false, you only see the available tiers.

Tier Name	Request Count	Unit Time (Seconds)	Stop On Quota Reach
Bronze	1	60	Yes
Gold	20	60	Yes
Silver	5	60	Yes

### **Adding a new advanced throttling policy**

You can add advanced throttling policies to both APIs and resources.

1. Sign in to the Admin Portal using the URL <https://localhost:9443/admin> and your admin credentials (admin/admin by default).
2. Click **Advanced Throttling** under the **Throttle Policies** section to see the set of existing throttling tiers.
3. To add a new tier, click **Add Tier**.

Name	Quota Policy	Quota	Unit Time
10KPerMin	requestCount	10000	1 min
20KPerMin	requestCount	20000	1 min
50KPerMin	requestCount	50000	1 min

4. Fill in the required details and click **Save**.

# Add Advanced Throttle Policy

## General Details

Tier Name: *	<input type="text" value="100KPerMin"/>
Tier Description: ?	<input type="text" value="Allows 100K requests per min"/>

## Default Limits

Request Count  Request Bandwidth

Request Count: *	<input type="text" value="100000"/>
Unit Time: *	<input type="text" value="1"/> Minutes(s)

## Conditional Groups



## Default Limits

Request Count  Request Bandwidth

Data Bandwidth: *	<input type="text"/>	KB
Unit Time:	<input type="text"/>	Minutes(s)

**Request Count** and **Request Bandwidth** are the two options for default limit. You can use the option according to your requirement. For example, If you're using API for File sharing, Data transmission you can use request bandwidth option and limit the data bandwidth for given time unit.

- To add throttling limits with different parameters to the conditions below, click **Add Conditional Group**.

Note that if you want to add a header, query param, or JSON Web Token (JWT) claim condition, you need to set the `<EnableHeaderConditions>`, `<EnableJWTClaimConditions>` or `<EnableQueryParamConditions>` element to true (depending on which condition you need) in the `repository/conf/api-manager.xml` file.

You can add Description about condition group by click **Sample description about condition group** under **Condition Group**.

### Conditional Groups

**Condition Group**  
Sample description about condition group

**IP Condition**

IP Condition Policy: Off

This configuration is used to throttle by IP address.

Request Count:	Request Count
Request Count: *	5
Time: *	1 Minute(s)

Condition	Description
IP Condition	Allows you to set a throttling limit for a specific IP address or a range of IP addresses.
Header Condition	Allows you to set a throttling limit to specific headers and parameters.
Query Param Condition	Allows you to set a throttling limit to specific query parameters.
JWT Claim Condition	Allows you to set a throttling limit to specific claims.

Conditional group Execution policy used only for that condition. For example, If you add IP condition and set request count as shown in above diagram then only 5 requests allow per 1 minute using that IP condition. Any request comes with false condition(Outside that condition) taken to the default limit.

6. Turn on the required condition and enter a condition and value.
7. Header condition and JWT claim condition values allow regex patterns to be defined. You can configure it to make either an exact match or a pattern match for the value using the regex values. For example,

**Condition Group**  
Sample description about condition group

0

IP Condition

Header Condition  ✓

Query Param Condition

JWT Claim Condition

Header Condition Policy

This configuration is used to throttle based on Headers.

Header Name: \* TestHeader

Param Value: \* ^{testvalue1|testvalue2|testvalue3}

Add

Invert Condition:

On

**Condition Group**  
Sample description about condition group

0

IP Condition

Header Condition  ✓

Query Param Condition

JWT Claim Condition  ✓

JWT Condition Policy

This configuration is used to define JWT claims conditions

Claim Name: \* http://wso2.org/claims/subscriber

Claim Value: \* ^{user1|user2|user3}

Add

Invert Condition:

On

Claim name : Name of the JWT Claim  
 Eg : "iss" - The issuer of the JWT, "http://wso2.org/claims/apicontext" - Context of the API, "http://wso2.org/claims/version" - API version

Claim value : Value to be checked in the corresponding claim. (allows regex patterns as well)

Invert condition : Whether to take the throttle decision based on the equality of the values.  
 Eg : If the claim name is "iss" value is "wso2" and "invert condition" is off - Requests having "wso2" as "iss" claim will be throttled.  
 If the claim name is "iss" value is "wso2" and "invert condition" is on - Requests not having "wso2" as "iss" claim will be throttled.

#### 8. Once done, click **Add**.

You have added a new advanced throttling policy. You can now apply it to an API or a resource.

You possibly can configure multiple conditional groups when defining a tier for advanced throttling policies. For example, Its possible to apply IP based throttling and query pram condition both in one advanced policy tier.

## Adding a new application-level throttling tier

Application-level throttling policies are applicable per access token generated for an application.

1. Sign in to the Admin Portal using the URL `https://localhost:9443/admin` and your admin credentials (admin/admin by default).
2. Click **Application Tiers** under the **Throttle Policies** section to see the set of existing throttling tiers.
3. To add a new tier, click **Add New Policy**.

Name	Quota Policy	Quota	Unit Time	Action
10PerMin	requestCount	10	1 min	
20PerMin	requestCount	20	1 min	
50PerMin	requestCount	50	1 min	

4. Fill in the required details and click **Save**.

## Add Application Level Policy

### General Details

Name *	100PerMin
Description	Allows 100 requests per min

### Quota Limits

Request Count  Request Bandwidth

Request Count\* 100

Unit Time \* 1 Minute(s) ▾

Save Cancel

You have added a new application-level throttling policy.

#### ***Adding a new subscription-level throttling tier***

1. Sign in to the Admin Portal using the URL `https://localhost:9443/admin` and your admin credentials.
2. Click **Subscription Policies** under the **Throttling Policies** section. The existing set of throttling tiers are displayed.
3. To add a new tier, click **Add New Policy**.

Name	Quota Policy	Quota	Unit Time	Rate Limit	Time Unit	Action	Action
Bronze	requestCount	1000	1 min	NA	NA		
Gold	requestCount	5000	1 min	NA	NA		
Silver	requestCount	2000	1 min	NA	NA		
Unauthenticated	requestCount	500	1 min	NA	NA		

When you are going to add a new Subscription level throttling tier, you can see the existing list of subscription tiers in **Subscription Tier List**. In this list, you will find a tier named **Unauthenticated** which have a request quota of 500. This is a subscription tier which automatically applied when the authentication type of your resources is '**None**'. That is, when you can invoke APIs without tokens. And this tier is not visible in the Throttling tier list of the application.

4. Fill in the details required by this form and click **Save** once you are done.

## Add Subscription Tier

### General Details

Name \*

Description

### Quota Limits

Request Count  Request Bandwidth

Request Count \*

Unit Time \*  Week(s)

### Burst Control (Rate Limiting)

Request Count  Request/s

### Policy Flags

Stop On Quota Reach

Billing Plan

### Custom Attributes

Add Custom Attribute

### Permissions

Roles \*

This tier is Allowed for above roles.

Allow  Deny

Save Cancel

Given below are the descriptions of the fields you find in the form:

Field	Description
Request Count/Request Bandwidth	The maximum number of requests/maximum bandwidth allowed to the API within the time period given in the next field.
Unit Time	Time within which the number of requests given in the previous field is allowed to the API. This can be defined in minutes, hours, days, weeks, months or years.

Burst Control (Rate Limiting)	You can define the request count/bandwidth per unit time on an addition layer by using rate limiting. This is usually a smaller number of requests/bandwidth for a shorter time span than what is enforced in the above fields. For instance, if there's a subscription level policy enforced over a long period, you may not want users to consume the entire quota within a short time span. Enforcing a rate limit protects the backend from sudden request bursts and controls the usage at a subscription and API level.
Stop On Quota Reach	This indicates the action to be taken when a user goes beyond the allocated quota. If the check box is selected, the user's requests are dropped and an error response (HTTP Status code 429) is given. If the check box is cleared, the requests are allowed to pass through.
Billing Plan	<p>This field only makes sense if you have API Monetization enabled. The available <b>billing plans</b> are <b>Free</b>, <b>Commercial</b>, and <b>Freemium</b>. An API is tagged/labelled as Free, Paid, or Freemium depending on its subscription <b>tiers</b> (e.g., Unlimited, Gold, etc.), which are the tiers selected when creating an API.</p> <ul style="list-style-type: none"> <li>• <b>Free</b> - If all subscription tiers are defined as Free, the API uses the <b>Free billing plan</b> and the API is labeled as Free in the Store.</li> <li>• <b>Paid</b> - If all subscription tiers are defined as Paid, the API uses the <b>Commercial billing plan</b> and the API is labeled as Paid in the Store.</li> <li>• <b>Freemium</b> - If the API has a combination of Free and Paid subscription tiers, the API uses the <b>Freemium billing plan</b> and the API is labeled as Freemium in the Store.</li> </ul> <p>This labeling happens on the API Store only if monetization has been enabled. For information on how to enable monetization and how to tag subscription tiers, see <a href="#">Configuring API Monetization Category Labels</a>.</p>
Custom Attributes	You can choose to display additional information about tiers using custom attributes, during custom implementations. The main objective of these fields are to provide more information regarding the tier to Application Developers at the time of API subscription. An example usage of custom attributes is API Monetization. See <a href="#">Enabling Monetization of APIs</a> for more information on practical usage of custom attributes in the subscription tier.
Permissions	You can allow or deny permission for specific roles. Once permission is denied to a role, the new subscription tier that you add here will not be available to that role in the API Store.

You have added a new subscription-level throttling policy.

## Managing Throttling

This section guides you through the following areas:

- IP Whitelisting
  - Creating the Advanced Throttling policy
  - Engage the policy with an API
- Blacklisting requests
  - Blacklisting PhoneVerification API
- Custom throttling

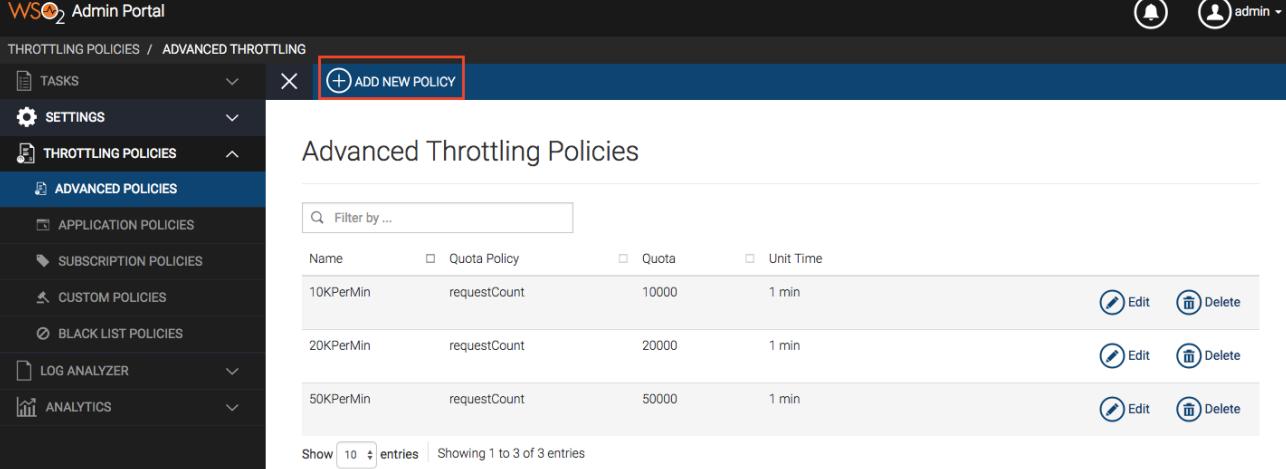
### IP Whitelisting

IP whitelisting is a way of configuring a filter to extract a particular set of known IP addresses and grant the access to the given assets for requests comes from those IPs only. With introducing Advanced Throttling in WSO2 API Manager you can achieve IP whitelisting via the features provided by Traffic Manager. For this we are creating an Advanced Throttling policy and attach it to the API.

### Creating the Advanced Throttling policy

1. Login to the admin portal of WSO2 API Manager ([https://<ip\\_address>:9443/admin](https://<ip_address>:9443/admin)).

2. Open **Throttling Policies** tab and navigate to **Advanced Throttling**.
3. Click **ADD NEW POLICY** to add a new Throttling tier.



The screenshot shows the WSO2 Admin Portal interface. The top navigation bar includes the WSO2 logo, a bell icon, and a user dropdown for 'admin'. The left sidebar has a tree structure with 'ADvanced Policies' selected, under which 'APPLICATION POLICIES', 'SUBSCRIPTION POLICIES', 'CUSTOM POLICIES', 'BLACK LIST POLICIES', 'LOG ANALYZER', and 'ANALYTICS' are listed. The main content area is titled 'Advanced Throttling Policies'. It features a search bar, a table with three rows of policy details, and pagination controls at the bottom.

Name	Quota Policy	Quota	Unit Time	Action
10KPerMin	requestCount	10000	1 min	
20KPerMin	requestCount	20000	1 min	
50KPerMin	requestCount	50000	1 min	

4. Fill the details as below and click **Add Conditional Group**.

### General Details

Name: *	<input type="text" value="whitelistIP"/>
Description:	<input type="text" value="Whitelisting"/>

### Default Limits

<input checked="" type="radio"/> Request Count <input type="radio"/> Request Bandwidth		
Request Count: *	<input type="text" value="1000"/>	
Unit Time: *	<input type="text" value="1"/>	Minutes(s)

### Conditional Groups

**Add Conditional Group**

5. Open the Conditional Group added and fill the details.

Property	Value
IP Condition Policy	Checked
IP Condition Type	Specific IP
IP Address	<IP_Address_to_be_whitelisted> E.g. 193.100.3.106

Invert Condition	Checked (If Invert Condition check then condition only apply to the IPs which not mention in IP Address above)
Request Count	0

Following is a example configuration.

Condition Group  
Sample description about condition group

IP Condition Policy

This configuration is used to throttle by IP address.

IP Condition Type: \* Specific IP

IP Address: \* 193.100.3.106

Invert Condition:

Execution Policy

Request Count : Request Count

Request Count: \* 0

Time: \* 5 Minute(s)

Save Cancel

In above configuration we are whitelisting a Specific IP.

You can whitelist a rang of IP as well by selecting **IP Range** for the IP Condition Type in the Conditional Group and specifying the range.

IP Condition Policy

On

This configuration is used to throttle by IP address.

IP Condition Type: \* IP Range

Start IP Address: \* 193.100.3.106

End IP Address: \* 193.100.3.125

Invert Condition:

6. Click **Save**.

## Advanced Throttling Policies

Name	(F) Quota Policy	Quota	Unit Time	
10KPerMin	requestCount	10000	1 min	<a href="#"></a> <a href="#"></a>
20KPerMin	requestCount	20000	1 min	<a href="#"></a> <a href="#"></a>
50KPerMin	requestCount	50000	1 min	<a href="#"></a> <a href="#"></a>
whitelistIP	requestCount	8	8 min	<a href="#"></a> <a href="#"></a>

Show  entries | Showing 1 to 4 of 4 entries

You have successfully created the policy. Now we should engage this policy to an API.

### Engage the policy with an API

1. Login to API Publisher [https://<IP\\_address>:9443/publisher](https://<IP_address>:9443/publisher).
2. Edit API and go to **Manage** tab.
3. Enable **Apply to API** under **Advance Throttling Policies** and select the newly created Throttling policy.

The screenshot shows the 'Manage' tab of the API Publisher interface. It includes sections for 'Configurations' (with options like 'Default version', 'Transports', and 'Response Caching') and 'Throttling Settings' (with options for 'Backend Throughput' and 'Subscription Tiers'). The 'Advanced Throttling Policies' dropdown is set to 'whitelist', which is highlighted with a red box.

4. Save and Publish the API.
- Now the API will be accessible only by the IP specified in the throttling policy.

Since it takes some time to deploy the policy, the first few requests from the IPs other than the white-listed IP/IPs will be passed through. After the policy is successfully deployed, non-white-listed IP access will be blocked.

### Blacklisting requests

By blacklisting requests, you can protect servers from common attacks and abuse by users. For example, if a malicious user misuses the system, all requests received from that particular user can be completely

blocked. Tenant administrative users can block requests based on the following parameters:

- Block calls to specific APIs
- Block all calls from a given application
- Block requests coming from a specific IP address
- Block a specific user from accessing APIs

To blacklist a request,

1. Log in to the Admin Portal using the URL `https://localhost:9443/admin` and your admin credentials.
2. Click **Black List** under the **Throttle Policies** section and click **Add Item**.

The screenshot shows the left sidebar with categories like Tasks, Settings, Throttling Policies (selected), Advanced Policies, Application Policies, Subscription Policies, and Custom Policies. Below these, a section for Black List Policies is shown, with the 'LOG ANALYZER' and 'ANALYTICS' sections visible. The main content area is titled 'Blacklisted Items' and displays a message: 'Blacklisted items are not defined'.

Select the item to black list, enter a value and click **Blacklist**.

## Select Item to Blacklist

Select Condition Type\*  API Context  Application  IP Address  User

Value : \*

Format : \${context}

Eg : test/1.0.0

**Blacklist** **Cancel**

Note that you have to use "/" always in front of the \${context} value when blacklisting the APIs with API context. E.g. **/test/1.0.0**. The sample provided in the product does not include "/" due to a known issue.

You can temporary on/off the blacklisting condition by enabling/disabling the **Condition status** that is auto enabled when a blacklisting condition is created.

### Blacklisted Items

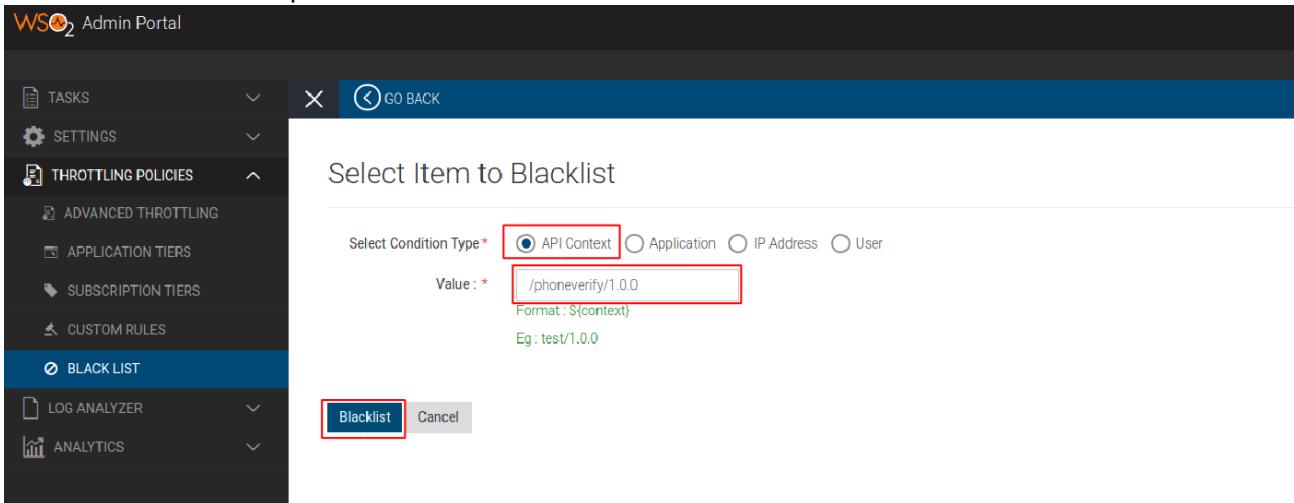
Condition ID	Condition Type	Condition Value	Condition Status	Actions
1	USER	testuser	<input checked="" type="checkbox"/>	

Show 10 entries | Showing 1 to 1 of 1 entries

### Blacklisting PhoneVerification API

As described above you can blacklist requests for APIs, by Applications, to IP Addresses and for Users. Let's see how we can blacklist the requests come to the [PhoneVerification API](#) that we published in Quick Start Guide.

1. Log in to the Admin Portal using the URL <https://localhost:9443/admin> and your admin credentials.
2. Click **Black List** under the **Throttle Policies** section and click **Add Item**.
3. Select **API Context** and provide the Context of PhoneVerification API with version as the **Value**.



4. Click **Blacklist**.
5. Now login to API Store using the URL <https://localhost:9443/store> and invoke the API.  
You will see the following response.

## Response Body

```
{
 "fault": {
 "code": 900805,
 "message": "Message blocked",
 "description": "You have been blocked from accessing the resource"
 }
}
```

## Response Code

403

## Response Headers

```
{
 "content-type": "application/json; charset=UTF-8"
}
```

### **Custom throttling**

Custom throttling allows system administrators to define dynamic rules for specific use cases, which are applied globally across all tenants. When a custom throttling policy is created, it is possible to define any policy you like. The Traffic Manager acts as the global throttling engine and is based on the same technology as WSO2 Complex Event Processor (CEP), which uses the [Siddhi query language](#). Users are therefore able to create their own custom throttling policies by writing custom Siddhi queries. The specific combination of attributes being checked in the policy need to be defined as the key (also called the key template). The key template usually includes a predefined format and a set of predefined parameters. It can contain a combination of allowed keys separated by a colon (:), where each key must start with the prefix \$. The following keys can be used to create custom throttling policies:

resourceKey, userId, apiContext, apiVersion, appTenant, apiTenant, appId

For example, the following sample custom policy allows the admin user to send 5 requests per minute to the Pizza Shack API.

#### Key Template

\$userId:\$apiContext:\$apiVersion

### Siddhi query

```

FROM RequestStream
SELECT userId, (userId == 'admin@carbon.super' and apiContext ==
'/pizzashack/1.0.0' and apiVersion == '1.0.0') AS isEligible ,
str:concat('admin@carbon.super',':','/pizzashack/1.0.0:1.0.0') as
throttleKey

INSERT INTO EligibilityStream;
FROM EligibilityStream[isEligible==true]#window.time(1 min)
SELECT throttleKey, (count(throttleKey) >= 5) as isThrottled group by
throttleKey
INSERT ALL EVENTS into ResultStream;

```

## Add Custom Rule

<p>Name *</p> <input type="text" value="Custom_Rule"/>
<p>Description</p> <div style="border: 1px solid #ccc; height: 60px; margin-top: 5px;"></div>
<p>Key Template * ?</p> <input type="text" value="\$userId:\$apiContext:\$apiVersion "/>
<p>Siddhi Query *</p> <div style="border: 1px solid #ccc; height: 150px; margin-top: 5px;"> <pre> FROM RequestStream SELECT userId, (userId == 'admin@carbon.super' and apiContext == '/pizzashack/1.0.0' and apiVersion == '1.0.0') AS isEligible , str:concat('admin@carbon.super',':','/pizzashack/1.0.0:1.0.0') as throttleKey  INSERT INTO EligibilityStream; FROM EligibilityStream[isEligible==true]#window.time(1 min) SELECT throttleKey, (count(throttleKey) &gt;= 5) as isThrottled group by throttleKey INSERT ALL EVENTS into ResultStream; </pre> </div>

[Show Sample](#)

 [Apply Rule](#)

 [Cancel](#)

As shown in the above Siddhi query, the throttle key must match the key template format. If there is a mismatch between the key template format and the throttle key, requests will not be throttled.

#### Enforcing Throttling to an API

**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 API Manager (WSO2 API-M), 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](#) to create and publish the PhoneVerification API and then the [Subscribe to an API](#) to subscribe to the 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. Sign in to the API Store and select the PhoneVerification API.

The screenshot shows the API Store interface for the PhoneVerification API version 1.0.0. On the left is a large red square icon with a white letter 'P'. To its right, the API title 'PhoneVerification - 1.0.0' is displayed. Below the title, there are several metadata fields: Version: 1.0.0, By: admin, Updated: 22/Jul/2016 17:12:01 PM IST, Status: PUBLISHED, and Rating: ★★★★☆. To the right of these fields is a section titled 'Applications' containing a dropdown menu set to 'DefaultApplication'. Below it is a 'Tiers' section with a dropdown menu set to 'Bronze'. At the bottom of this section is a blue 'Subscribe' button with a key icon. A red rectangular box highlights the 'Applications' and 'Tiers' sections.

2. Go to the Default Application, click the **Production Keys** tab and generate an access token. If you already have an access token for the application, you have to regenerate it after 1 hour.

## DefaultApplication

Details **Production Keys** Sandbox Keys Subscriptions

Show Keys

**Consumer Key**  
.....

**Consumer Secret**  
.....

**Grant Types**  
Application can use the following grant types to generate Access Tokens. Based on the application requirement, you can select one or more grant types.

<input checked="" type="checkbox"/> SAML2	<input checked="" type="checkbox"/> IWA-NTLM
<input checked="" type="checkbox"/> Client Credential	<input type="checkbox"/> Code

**Callback URL**  
.....

**Update**

**Generating Access Tokens**  
The following cURL command shows how to generate an access token using the password grant type.

```
curl -k -d "grant_type=password&username=Username&password=Password" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

In a similar manner, you can generate an access token using the client credential grant type with the following cURL command.

```
curl -k -d "grant_type=client_credentials" \
-H "Authorization: Basic Base64(consumer-key:consumer-secret)" \
https://192.168.1.6:8243/token
```

**Generate a Test Access Token**

**Access Token**  
.....

Above token has a validity period of **3600** seconds. And the token has (**am\_application\_scope default**) scopes.

**Scopes**  
No Scopes Found..

**Validity period**  
3600 Seconds.

**Re-generate**

Let's invoke this API.

- Click on the API, then go to its **API Console** tab and expand the GET method.

## PhoneVerification - 1.0.0

The screenshot shows the WSO2 API Manager interface for the 'PhoneVerification' API version 1.0.0. The top navigation bar includes tabs for 'Overview', 'API Console' (which is selected and highlighted with a red border), 'Documentation', and 'Forum'. Below the navigation, there's a summary section with details like Version: 1.0.0, By: admin, Updated: 22/Jul/2016 17:12:01 PM IST, Status: PUBLISHED, and Rating: ★★★★☆. To the right, there are dropdown menus for 'Applications' (Select Application...) and 'Tiers' (Unlimited), and a 'Subscribe' button. The main content area has sections for 'Try' (set to 'TestApp') and 'Using' (set to 'Production' with 'Key' selected). A 'Set Request Header' section shows 'Authorization : Bearer 4dd4bf2e-eae0-31a0-8850-6338213a8100'. At the bottom right, there's a link to 'Swagger (/swagger.json)' and options to 'Show/Hide', 'List Operations', and 'Expand Operations'.

4. Give values to the parameters and click **Try it out** to invoke the API.

This screenshot shows the 'Try it out' interface for the '/CheckPhoneNumber' endpoint. It displays two methods: 'GET /CheckPhoneNumber' and 'POST /CheckPhoneNumber'. Below the methods, a note indicates the base URL: '/phoneverify/1.0.0 , API VERSION: 1.0.0'. The 'Parameters' section lists two parameters: 'PhoneNumber' (value: '18006785432') and 'LicenseKey' (value: '0'). Both parameter input fields are highlighted with a red border. The 'Response Messages' section shows a single entry for '200' with a 'Try it out!' button, which is also highlighted with a red border. Below the response message table, there's another 'Try it out!' button for the POST method.

[ BASE URL: /phoneverify/1.0.0 , API VERSION: 1.0.0 ]

Note the response that appears in the API Console. As we used a valid phone number in this example, the response returns as 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>
 <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 />

```

Note that you subscribed to the API on the Bronze throttling tier. The Bronze tier allows you to make a 1000 calls to the API per minute. If you exceed your quota, you get a throttling error as shown below.

**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 the API using an unavailable resource name.

5. Go to the API's **Overview** page in the API Store and get the API's URL.

## PhoneVerification - 1.0.0



**Version:** 1.0.0  
**By:** admin  
**Updated:** 22/Jul/2016 16:59:07 PM IST  
**Status:** PUBLISHED

[Overview](#) [API Console](#) [Documentation](#) [Forum](#)

### Production and Sandbox Endpoints

Production and Sandbox URLs:

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

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

6. Install cURL or any other REST client.
7. Go to the command-line invoke the API using the following cURL command.

```
curl -k -H "Authorization :Bearer <access token in step 3>" '<API's URL in step 9>/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

Note that the PhoneVerification API's resource name is **CheckPhoneNumber**, but we use an undefined resource name as **CheckPhoneNum**. Here's an example:

```
curl -k -H "Authorization :Bearer 63cc9779d6557f4346a9a28b5cf8b53" 'https://localhost:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
```

8. Note that the call gets blocked by the API Gateway with a 'no matching resource' message. It doesn't reach your backend services as you are trying to access a REST resource that is not defined for the API.

```
[REDACTED] curl -k -H "Authorization :Bearer 63cc9779d6557f4346a9a28b5cf8b53" 'https://[REDACTED]:8243/phoneverify/1.0.0/CheckPhoneNum?PhoneNumber=18006785432&LicenseKey=0'
<ams:fault xmlns:ams="http://wso2.org/apimanager/security"><ams:code>900006</ams:code>
<ams:message>No matching resource found in the API for the given request</ams:message>
<ams:description>Access failure for API: /phoneverify/1.0.0, version: 1.0.0. Check the
```

You have seen how the API Gateway enforces throttling and resource access policies for APIs.

### Engaging a new Throttling Policy at Runtime

WSO2 API Manager allows you to control the number of successful requests to your API during a given period. You can enable for APIs in the CREATED and PUBLISHED state and also for published APIs at runtime. This feature protects your APIs, regulates traffic and access to the resources.

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

1. Write a new 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-ws-
 security-utility-1.0.xsd"
 xmlns:throttle="http://www.wso2.org/products/wso2commons/throttle"
 wsu:Id="WSO2MediatorThrottlingPolicy">
 <throttle:MediatorThrottleAssertion>

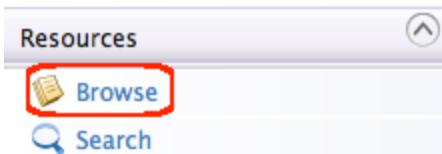
 <throttle:MaximumConcurrentAccess>1000</throttle:MaximumConcurrentAcc-
 ess>
 <wsp:Policy>
 <throttle:ID throttle:type="IP">other</throttle:ID>

 </wsp:Policy>
 </throttle:MediatorThrottleAssertion>
</wsp:Policy>
```

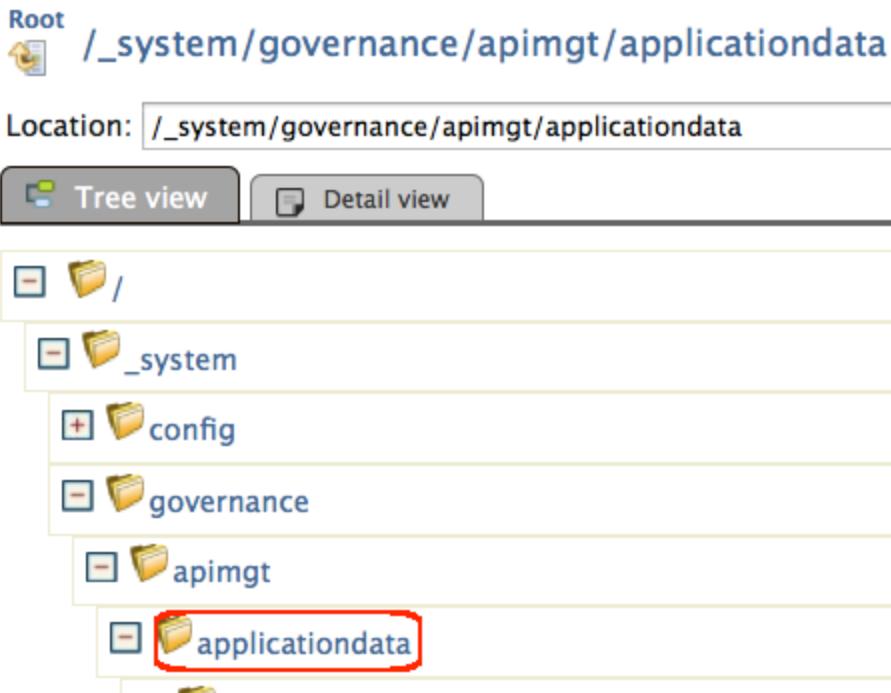
## Attributes

- Throttle policy - This section is used to specify the policy for throttling.
- Maximum concurrent accesses - The maximum number of messages that are served at a given time.
- Throttle assertion - Assertion for a concurrency-based policy.

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



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



- In the detail view, click **Add Resource**.

The screenshot shows the 'Browse' detail view for the 'applicationdata' folder. At the top, there's a 'Metadata' tab. Below it is a 'Properties' tab. Under 'Properties', there's a 'Entries' section containing three items: 'Add Resource' (highlighted with a red box), 'Add Collection', and 'Create Link'.

- Upload the policy file to the server as a registry resource.
- Open the synapse configuration file of a selected API you want to engage the policy, from the `<API-M_HOME>/repository/deployment/server/synapse-configs/default/api` directory.
- To engage the policy to a selected API, add it to your API definition. In this example, we add it to the login API under APIThrottleHandler.

```

<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.APIThrottle
Handler">
 <property name="id" value="A"/>
 <property name="policyKey"
value="gov:/apimgt/applicationdata/throttle.xml"/>
 </handler>
 <handler
class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensio
nHandler"/>
 </handlers>
</api>

```

**Note:** Be sure to specify the same path used in step 5 in the policy key of your API definition. Also, use the same tier name you selected when creating the API as the throttle id in the policy (example <throttle:ID throttle:type ="ROLE">Gold</throttle:ID>).

You have successfully engaged a throttling policy to an API at runtime, without restarting the server.

### Engaging Multiple Throttling Policies to a Single API

You can apply different throttling policies at the same time to a single API. This is called **multi-layer throttling**.

The following example shows how to have two throttling policies for a single API at a given time. The table below shows the throttling information of the two throttling policies.

Tier	throttle-I1	throttle-I2
free	300 per month	5 per 3 minutes
Silver	2000 per month	1 per 5 seconds
Gold - Unlimited	Unlimited	Unlimited

To engage the two throttling layers, you add two throttling tier definitions and engage them to the API using the steps below:

1. Go to the Synapse configuration file of the particular API located in < AM\_HOME>/repository/deployment/server/synapse-configs/default/api.
2. Copy the following content inside the <handlers> section in the API configuration.

```
<handler
 class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"/>
<handler
 class="org.wso2.carbon.apimgt.gateway.handlers.throttling.APIThrottleHandler">
 <property name="id" value="B"/>
 <property name="policyKey"
 value="gov:/apimgt/applicationdata/throttling-12.xml"/>
</handler>
<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>
```

3. Replace the existing content of the /\_system/governance/apimgt/applicationdata/tiers.xml file with following content.

**throttling-l1.xml**

```
<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="ROLE">Gold</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>20</throttle:MaximumCount>
 <throttle:UnitTime>60000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
 </wsp:Policy>
 <wsp:Policy>
 <throttle:ID throttle:type="ROLE">Silver</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>2000</throttle:MaximumCount>
 <throttle:UnitTime>2592000000</throttle:UnitTime>
 </wsp:Policy>
```

```
 </throttle:Control>
 </wsp:Policy>
</wsp:Policy>
<wsp:Policy>
 <throttle:ID throttle:type="ROLE">free</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>300</throttle:MaximumCount>

<throttle:UnitTime>2592000000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
</wsp:Policy>
</wsp:Policy>
<wsp:Policy>
 <throttle:ID
throttle:type="ROLE">Unauthenticated</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>60</throttle:MaximumCount>
 <throttle:UnitTime>60000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
</wsp:Policy>
```

```

 </wsp:Policy>
 </throttle:MediatorThrottleAssertion>
</wsp:Policy>

```

4. Create an XML as `throttling-12.xml` with the following content and add it to `/_system/governance/apimgt/applicationdata` registry location. The code adds two policies for each role (free, Silver, Gold) and engages them to the APIs with different keys. Both throttling layers execute in runtime sequentially.

#### throttling-12.xml

```

<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="ROLE">Gold</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>20</throttle:MaximumCount>
 <throttle:UnitTime>60000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
 </wsp:Policy>
 <wsp:Policy>
 <throttle:ID throttle:type="ROLE">Silver</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>1</throttle:MaximumCount>
 <throttle:UnitTime>5000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
 </wsp:Policy>
 <wsp:Policy>
 <throttle:ID throttle:type="ROLE">Free</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>5</throttle:MaximumCount>
 <throttle:UnitTime>180000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
 </wsp:Policy>
 <wsp:Policy>

```

```
<throttle:ID
throttle:type="ROLE">Unauthenticated</throttle:ID>
 <wsp:Policy>
 <throttle:Control>
 <wsp:Policy>

<throttle:MaximumCount>60</throttle:MaximumCount>
 <throttle:UnitTime>60000</throttle:UnitTime>
 </wsp:Policy>
 </throttle:Control>
 </wsp:Policy>
```

```
</wsp:Policy>
</throttle:MediatorThrottleAssertion>
</wsp:Policy>
```

## Working with 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 supports a range of different endpoint types, allowing the API Gateway to connect with advanced types of backends.

Endpoint Type	Description
HTTP endpoint	A REST service endpoint based on a URI template.
Address endpoint	The direct URL of the service.
Failover Group endpoint	<p>The endpoints that the service tries to connect to in case of a failure. Selecting the endpoint when the primary endpoint get failed happens in a round robin manner.</p> <p>Failover Group is a group of leaf endpoints(i.e, address endpoint, HTTP endpoint and WSDL endpoint). The failover group endpoint try to send the message to another endpoint when failure occur in current endpoint (while sending a message). Failover group ensures that a message is delivered as long as there is at least one active endpoint among the listed endpoints.</p>
Load Balance endpoint	The endpoints where the incoming requests are directed to in a round robin manner. They automatically handle fail-over as well.
Dynamic endpoint	The dynamic endpoint sends the message to the address specified in the <b>To</b> header. You can configure dynamic endpoints by setting mediation extensions with a set of conditions to dynamically change the <b>To</b> header. For details of configuring endpoints to change the default mediation flow, see <a href="#">Adding Mediation Extensions</a> .

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 <APIM\_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"> element to AllowAll in the <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.

### Configuring Load Balancing Endpoints

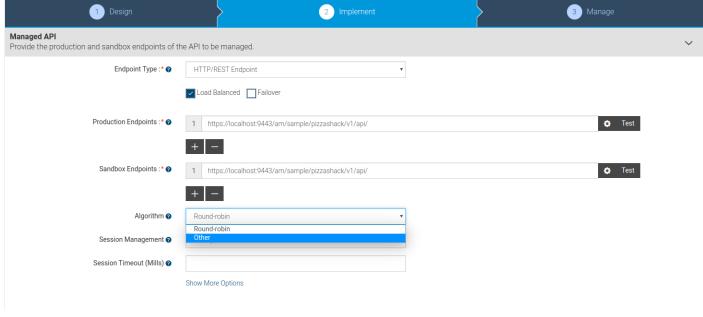
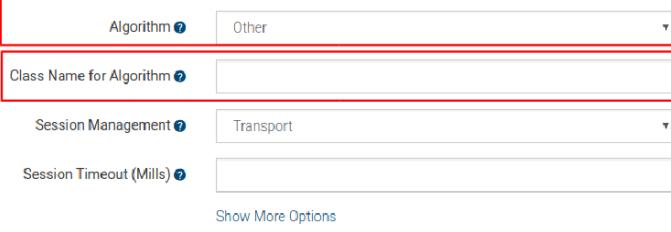
WSO2 API Manager provides configuring load balancing endpoints through API Publisher.

To configure load balanced endpoints go to the edit view of the API and navigate to the **implement** tab and click Load Balanced under endpoint type.

The screenshot shows the 'Implement' tab of the WSO2 API Publisher interface. Under 'Managed API', it says 'Provide the production and sandbox endpoints of the API to be managed.' The 'Endpoint Type' dropdown is set to 'HTTP/REST Endpoint'. The 'Load Balanced' checkbox is checked and highlighted with a red box. Below this, there are sections for 'Production Endpoints' and 'Sandbox Endpoints', each with a list box containing a single entry ('https://localhost:9443/am/sample/pizzashack/v1/api/'). There are '+' and '-' buttons next to each list box. Below these are dropdowns for 'Algorithm' (set to 'Round-robin') and 'Session Management' (set to 'Transport'), and a 'Session Timeout (Mills)' input field. A 'Show More Options' link is at the bottom.

Following are the other configurations that you need to do in order to specify a load balancing endpoint.

Configuration	Description
Production Endpoints	<p>The set of production endpoints can be specified here where the requests need to be load balanced. You can specify more than one endpoint by clicking on "+" sign and can delete the endpoints by clicking on "-" sign.</p> <p>Production Endpoints:</p> <ul style="list-style-type: none"> <li>1 https://localhost:9443/am/sample/pizzashack/v1/api/ <span style="float: right;">Test</span></li> <li>2 https://10.45.7.32:9443/am/sample/pizzashack/v1/api/ <span style="float: right;">Test</span></li> <li>3 https://10.45.7.33:9443/am/sample/pizzashack/v1/api/ <span style="float: right;">Test</span></li> </ul> <p style="text-align: center;">+   -</p> <p>Sandbox Endpoints:</p> <ul style="list-style-type: none"> <li>1 https://localhost:9443/am/sample/pizzashack/v1/api/ <span style="float: right;">Test</span></li> </ul> <p style="text-align: center;">+   -</p>
Sandbox endpoints	<p>The set of sandbox endpoints can be specified here where the requests need to be load balanced. You can specify more than one endpoint by clicking on "+" sign and can delete the endpoints by clicking on "-" sign</p>

Algorithm	<p>The load balancing algorithm is specified here.</p>  <p>The default is <b>Round Robin</b> Algorithm which has the className of <code>org.apache.synapse.endpoints.algorithms.RoundRobin</code>. If you select other from the dropdown list of Algorithms you need to specify the class name of the algorithm. Classnames of other algorithms can be found <a href="#">here</a>.</p> 
Session Management	<p>A session management method from the load balancing group. The possible values are as follows.</p> <p><b>None</b> - If this is selected, session management is not used.</p> <p><b>Transport</b> - If this is selected, session management is done on the transport level using HTTP cookies.</p> <p><b>SOAP</b> - If this is selected, session management is done using SOAP sessions.</p> <p><b>Client ID</b> - If this is selected, session management is done using an ID sent by the client.</p>
Session Timeout	The number of milliseconds after which the session would time out.

After completing the configuration click save and publish the API.

#### **Configuring Failover Group of Endpoints**

WSO2 API Manager provides configuring failover group endpoints through API Publisher.

To configure failover endpoints go to the edit view of the API and navigate to the **implement** tab and click **Failover** under endpoint type.

**Managed API**  
Provide the production and sandbox endpoints of the API to be managed.

Endpoint Type : \*  HTTP/REST Endpoint

Load Balanced  Failover

Production Endpoints : \*

Primary	https://localhost:9443/am/sample/pizzashack/v1/api/	
Failover 1	https://10.34.6.78:9443/am/sample/pizzashack/v1/api/	
Failover 2	https://10.34.6.79:9443/am/sample/pizzashack/v1/api/	

**Sandbox Endpoints : \***

Primary	https://localhost:9443/am/sample/pizzashack/v1/api/	
Failover 1	https://10.34.6.78:9443/am/sample/pizzashack/v1/api/	

**Show More Options**

At least one failover endpoint need to be added for production and sandbox (if you have specified) Endpoints.

You can specify more than one endpoint by clicking on "+" sign and can delete the endpoints by clicking on "-" sign.

After completing the adding endpoints, click save and publish the API.

#### ***Advanced Endpoint Configuration***

WSO2 API Manager provides controlling the production and sandbox endpoints with Advanced Endpoint Configuration.

To configure your endpoints with this feature, go to the edit view of the API, navigate to the **Implement** tab and click the cogwheel icon next to the endpoint you want to configure.

The Advanced Endpoint Configuration dialog box appears as below.

## Advanced Endpoint Configuration

**Endpoint Suspend State**

Error Codes [?](#)

A list of error codes. If these error codes are received from the endpoint, the endpoint will be suspended.

Initial Duration (ms) [?](#)  Max Duration (ms)

Factor

**Endpoint Timeout State**

Error Codes

Retries Before Suspension  Retry Delay(ms)

**Connection Timeout**

Action

Duration (ms)

**Save** **Close**

Following are the configurations that we can do with the Advanced Endpoint Configurations. You can do advanced configurations for both Production and Sandbox endpoints.

Endpoint Configuration	Description

Endpoint Suspend State	<p>If you want to configure the suspension of an Endpoint specifying error codes, maximum suspension time, suspension factors etc., you can use Endpoint Suspension State in Advanced Endpoint Configuration.</p> <p><b>Error Codes :</b> Error codes in the drop down list which need to make the endpoint suspension. If the selected error codes are received from the endpoint, the endpoint will be suspended. Specify the <a href="#">transport error codes</a> where the Endpoint Suspension should be triggered. You can select single or error codes here.</p> <p><b>Initial duration :</b> The time duration for which the endpoint will be suspended, when one or more suspend error codes are received from it for the first time.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>When creating (or updating) Failover endpoints through the Publisher UI (in the <b>Implement</b> tab), you need to go into this configuration box 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.</p> </div> <p><b>Max duration :</b> The maximum time duration for which the endpoint is suspended when suspend error codes are received from it.</p> <p><b>Factor:</b> The duration to suspend can vary from the first time suspension to the subsequent time. The factor value decides the suspense duration variance between subsequent suspensions.</p>
Endpoint Timeout state	<p>Configurations of retry, error codes and delays in terms of Endpoint timeout can be configured with Endpoint Timeout State in Advanced Endpoint Configuration.</p> <p><b>Error Codes :</b> A list of error codes. If these error codes are received from the endpoint, the request will be subjected to a timeout.</p> <p><b>Retries Before Suspension :</b> The number of re-tries in case of a timeout, caused by the above listed error codes.</p> <p><b>Retry Delay :</b> The delay between retries in milliseconds.</p>

Connection Timeout	<p>Duration and the Response Actions after a Connection Timeout can be configured here in Advanced Endpoint Configuration.</p> <p><b>Action :</b> Response Action to be performed after connection timeout. You can select from <b>Never Timeout</b>, <b>Discard Message</b>, <b>Execute Fault Sequence</b>. The default value is <b>Execute Fault Sequence</b>.</p> <p><b>Duration :</b> The time duration of Connection Timeout in milliseconds.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>if you want to change the Endpoint Connection Timeout duration globally affecting all APIs, do the following.</p> <ol style="list-style-type: none"> <li>1. Open <code>&lt;APIM_HOME&gt;/repository/conf/synapse.properties</code>. Change the value of the timeout as given below.</li> </ol> <pre style="border: 1px dashed #ccc; padding: 5px; margin-top: 10px;"><code>synapse.global_timeout_interval=30000</code></pre> <ol style="list-style-type: none"> <li>2. Open <code>&lt;ESB_HOME&gt;/repository/conf/passthru-http.properties</code> and change the socket timeout value.</li> </ol> <pre style="border: 1px dashed #ccc; padding: 5px; margin-top: 10px;"><code>http.socket.timeout=30000</code></pre> <p>Note that the <b>global timeout value</b> should be greater than the <b>connection timeout value</b> given for your API</p> </div>
--------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

For more information about endpoints and how to add, edit or delete them, see the [WSO2 ESB documentation](#).

## Analytics

This section explains how to configure and analyze statistics relating to APIs deployed in WSO2 API Manager, configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and to carry out detailed analysis of logs relating to the APIs. See the topics below for detailed information about how these monitoring activities are carried out.

- [Configuring APIM Analytics](#)
- [Analyzing APIM Statistics with Batch Analytics](#)
- [Managing Alerts with Real-time Analytics](#)
- [Analyzing Logs with the Log Analyzer](#)
- [Integrating with Google Analytics](#)
- [Monitoring with WSO2 Carbon Metrics](#)
- [Installing WSO2 APIM Analytics Features](#)
- [Purging Analytics Data](#)
- [Default Ports of WSO2 API-M Analytics](#)
- [Troubleshooting the Analytics Profile](#)
- [Updating WSO2 API Manager Analytics](#)

### Configuring APIM Analytics

This section explains how to configure analytics for WSO2 API Manager (WSO2 API-M). The API Manager

integrates with the [WSO2 Analytics platform](#) to provide reports, statistics and graphs on the APIs deployed in WSO2 API Manager. You can then configure alerts to monitor these APIs and detect unusual activity, manage locations via geo location statistics and carry out detailed analysis of the logs. WSO2 API Manager has an enhanced distribution of Analytics to cater to API Manager specific scenarios which is used here to configure APIM Analytics.

By default, WSO2 API Manager has a port offset of 0 (no port offset) and WSO2 API Manager Analytics has an offset of 1. Therefore, this guide assumes that you do not have any other carbon servers running on the same machine with port offsets of 0 or 1.

Click on the **Quick Setup** tab to set up analytics for quick demos and try-out scenarios, or click on the **Standard Setup** tab to set up analytics for a production environment.

WSO2 recommends using the API-M Analytics (powered by [WSO2 Data Analytics Server](#)) distribution to set up the minimum high availability deployment with API Manager. For configuration details, see [Minimum High Availability Deployment for WSO2 APIM Analytics](#).

### Quick Setup Standard Setup

This mode of setup is only recommended for non-critical demos and quick try-out scenarios. Once you run the servers you will notice that there are database files being created in a folder called **tmpStatDB** in the directory where you installed the two servers. These are H2 databases that hold the summarized data of your API Analytics. In a more standard deployment, this data is recorded in database servers such as MySQL, Oracle, etc. Therefore, this mode of operation is not recommended for production grade or other critical deployments.

1. Download the WSO2 API Manager and the WSO2 API-M Analytics distributions (zip files), and extract both files to the same directory (preferably an empty directory).
  - To download the WSO2 API Manager distribution, click **DOWNLOAD** and then click **DOWNLOAD Server** in the [WSO2 API Manager](#) page.
  - To download the WSO2 API-M Analytics distribution, click **DOWNLOAD** and then click **DOWNLOAD ANALYTICS** in the [WSO2 API Manager](#) page.
2. Take the following steps to install WSO2 APIM Analytics. Because this procedure is identical to installing WSO2 Data Analytics Server (DAS), these steps take you to the DAS documentation for details.
  - a. Ensure that you have met the [Installation Prerequisites](#).
  - b. Go to the installation instructions relevant to your operating system:
    - [Installing on Linux](#)
    - [Installing on Windows](#)
    - [Installing as a Windows Service](#)
    - [Installing as a Linux Service](#)
3. To enable Analytics, open the `<API-M_HOME>/repository/conf/api-manager.xml` file and set the `Enabled` property under Analytics to true as shown below. Save this change.

```
<Enabled>true</Enabled>
```

If you are working on a distributed (clustered) setup of API Manager, do the configurations instructed to be done in API Manager in Publisher, Store and Gateway nodes.

4. Share the WSO2AM\_STATS\_DB datasource between WSO2 API-M and WSO2 API-M Analytics as follows.
  - a. Open the <API-M\_HOME>/repository/conf/datasources/master-datasources.xml file and make sure that a configuration for the WSO2AM\_STATS\_DB datasource is included with your datasource configurations. The default configuration which is already available in master-datasources.xml file is as follows. It is configured for the in-built h2 database by default. You can change the datasource according to the database you use referring to [change statistics datasource](#).

```

<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:h2:../tmpStatDB/WSO2AM_STATS_DB;DB_CLOSE_ON_EXIT=FALSE
 ;LOCK_TIMEOUT=60000;AUTO_SERVER=TRUE</url>
 <username>wso2carbon</username>
 <password>wso2carbon</password>
 <defaultAutoCommit>false</defaultAutoCommit>
 <driverClassName>org.h2.Driver</driverClassName>
 <maxActive>50</maxActive>
 <maxWait>60000</maxWait>
 <testOnBorrow>true</testOnBorrow>
 <validationQuery>SELECT 1</validationQuery>
 <validationInterval>30000</validationInterval>
 </configuration>
 </definition>
</datasource>

```

If you are changing the datasource to a different database like MySQL, note that you do not need to run the database scripts against the created databases as the tables for the datasources are created at runtime.

- b. Open the <API-M\_ANALYTICS\_HOME>/repository/conf/datasources/stats-datasources.xml file and make sure that the same configuration for WSO2AM\_STATS\_DB in the <API-M\_HOME>/repository/conf/datasources/master-datasources.xml file (mentioned in the previous sub step) is added in it.
5. Open the <API-M\_HOME>/repository/conf/log4j.properties file. Add the DAS\_AGENT to the end of the root logger as shown below.

This configuration is required only if you need to analyze WSO2 API-M logs using the Log Analyzer. When you are disabling analytics, make sure to remove this configuration to avoid errors.

If you are working on a distributed (clustered) setup of API Manager, follow these steps to configure log analyzer in Gateway node.

```
log4j.rootLogger=<other loggers>, DAS_AGENT
```

6. Start the WSO2 API-M Analytics server, and then start the API Manager server. To start a WSO2 product server, navigate to the <PRODUCT\_HOME>/bin directory in your console and run one of the following scripts as relevant.
  - a. On Windows: wso2server.bat --run
  - b. On Linux/Mac OS: sh wso2server.sh

If Analytics is properly configured in WSO2 API Manager you will see the following log in the Analytics server during API Manager server startup.

```
INFO {org.wso2.carbon.databridge.core.DataBridge} - user admin connected
```

You can now start using the API Manager for its usual operations and the required Analytics functionality is enabled.

If you are configuring API-M Analytics with MSSQL and you get an error when you start the API-M Analytics server stating that a table cannot have more than one clustered index, follow the steps below.

1. Open the <API-M\_ANALYTICS\_HOME>/repository/components/features/org.wso2.carbon.analytics.spark.server\_VERSION/spark-jdbc-config.xml file.
2. Update the value for the <indexCreateQuery> element of the MySQL database as shown below.

```
<database name="Microsoft SQL Server">
<indexCreateQuery>CREATE INDEX {{TABLE_NAME}}_INDEX ON
{{TABLE_NAME}} ({{INDEX_COLUMNS}})</indexCreateQuery>
</database>
```

### **Downloading WSO2 API-M Analytics**

Follow the instructions below to download the binary distribution of WSO2 API-M Analytics.

The binary distribution contains the binary files for both MS Windows, and Linux-based operating systems. You can also download, and build the source code.

1. Go to the [WSO2 API Manager page](#).
2. Click **DOWNLOAD** and then click **DOWNLOAD ANALYTICS** to download the WSO2 API-M Analytics product pack.

### **Installing WSO2 API-M Analytics**

Take the following steps to install WSO2 APIM Analytics. Because this procedure is identical to installing WSO2 Data Analytics Server (DAS), these steps take you to the DAS documentation for details.

1. Ensure that you have met the [Installation Prerequisites](#).
2. Go to the installation instructions relevant to your operating system:
  - [Installing on Linux](#)
  - [Installing on Windows](#)
  - [Installing as a Windows Service](#)
  - [Installing as a Linux Service](#)

#### **Configuring WSO2 API Manager to publish statistics**

Follow the instructions below to do the required configurations for WSO2 API-M to publish statistics in the WSO2 API-M Analytics server.

To download the WSO2 API Manager distribution, click **DOWNLOAD** and then click **DOWNLOAD Server** in the [WSO2 API Manager page](#).

If you are working on a distributed (clustered) setup of API Manager, do the configurations instructed to be done in API Manager in Publisher, Store and Gateway nodes.

1. Open the <API-M\_HOME>/repository/conf/api-manager.xml file.
2. Under the <Analytics> sub element, set the Enabled parameter to true.
3. Configure the following parameters if required.

Parameter	Value	Description
<DASServerURL>	<protocol>://<hostname>:<port>/	The server URL of the remote DAS also be published to multiple receivers delimited by curly braces, whereas e.g., Three receivers \tcp://localhost:7612,tcp:13 Two receiver groups w {tcp://localhost:7612,tcp:13} For more information on configuring Data Agent.

<DASUsername>	A valid administrator username	The administrator user name to Manager.  <ul style="list-style-type: none"> <li>If you enable email user, Analytics admin user. e.g. must be demo@wso2.cc</li> <li>It is required to change the API-M is different to the user of the current AF</li> </ul>
<DASPassword>	The password of the username specified.	The administrator password to log Manager.  <p>It is required to change the API-M is different to the user of the current AF</p>
<DASRestApiURL>	<code>https://&lt;host&gt;:&lt;port&gt;</code>	The WSO2 DAS (WSO2 API-M Analytics) which are included under the REST API common to all the WSO2 API-M APIs
<DASRestApiUsername>	A valid administrator username	The administrator username to log
<DASRestApiPassword>	The password of the username specified.	The administrator password to log

#### 4. Save the changes.

##### *Configuring the Log Analyzer*

This configuration is required only if you want to analyze WSO2 API-M logs using the Log Analyzer. If you are working on a distributed (clustered) setup of API Manager, follow these steps below in Gateway node.

Follow the steps below to configure the Log Analyzer.

- Add `DAS_AGENT` to the end of the root logger in the `<API-M_HOME>/repository/conf/log4j.properties` file as shown below.

```
log4j.rootLogger=<other loggers>, DAS_AGENT
```

Then check and make sure that the following configuration is available in the same file. Modify the values for `userName`, `password` and/or `url` if required.

The values given below are the default values of the configuration.

```

DAS_AGENT is set to be a Custom Log Appender.
log4j.appenders.DAS_AGENT=org.wso2.carbon.analytics.shared.data.agents
.log4j.appenders.LogEventAppender
DAS_AGENT uses PatternLayout.
log4j.appenders.DAS_AGENT.layout=org.wso2.carbon.analytics.shared.data
.agents.log4j.util.TenantAwarePatternLayout
log4j.appenders.DAS_AGENT.columnList=%D,%S,%A,%d,%c,%p,%m,%H,%I,%Stack
trace
log4j.appenders.DAS_AGENT.userName=admin
log4j.appenders.DAS_AGENT.password=admin
log4j.appenders.DAS_AGENT.url=tcp://localhost:7612
log4j.appenders.DAS_AGENT.maxTolerableConsecutiveFailure=5
log4j.appenders.DAS_AGENT.streamDef=loganalyzer:1.0.0

```

2. To view log analytics on the API-M Admin portal, you have to set the DAS REST API configurations (DASRestApiURL, DASRestApiUsername, DASRestApiPassword). You can give the same previous value as <DASUsername> to <DASRestApiUsername> and <DASPassword> to <DASRestApiPassword>. The <DASRestApiURL> value should be in the following format, **<https://<DAS Hostname>:<DAS Https port>>**. Go to the <API-M\_HOME>/repository/conf/api-manager.xml file and change the value given below.

```
<DASRestApiURL>https://analytics.wso2.com:9444</DASRestApiURL>
```

3. If the API-M server and the DAS server run on two different hosts with their default certificates, set the HostNameVerifier to AllowAll in the Pass Through HTTP SSL Sender section of the <API-M\_HOME>/repository/conf/axis2/axis2.xml file as follows. If this setting is not changed, the javax.net.ssl.SSLException: Host name verification failed for host exception may occur for the API-M instance.

```

<transportSender name="https"
 class="org.apache.synapse.transport.passthru.PassThroughHttpSSLSSender"
 >
 <parameter name="non-blocking" locked="false">true</parameter>
 <parameter name="keystore" locked="false">
 <KeyStore>
 <Location>repository/resources/security/wso2carbon.jks</Location>
 <Type>JKS</Type>
 <Password>wso2carbon</Password>
 <KeyPassword>wso2carbon</KeyPassword>
 </KeyStore>
 </parameter>
 <parameter name="truststore" locked="false">
 <TrustStore>
 <Location>repository/resources/security/client-truststore.jks</Location>
 <Type>JKS</Type>
 <Password>wso2carbon</Password>
 </TrustStore>
 </parameter>
 <parameter name="HostnameVerifier">AllowAll</parameter>
</transportSender>

```

4. If the WSO2 API Manager was started before these configurations were done, restart it in order to apply the changes.

#### **Securing log4j properties file with a secure vault**

Using secure vaults allows you to avoid exposing passwords by having them in plain text in the log4j properties file. Follow the procedure below if you want to secure the log4j properties file with a secure vault.

1. If you have not already generated the secret-conf.properties file with default values, navigate to the <API-M\_HOME>/bin directory and execute the following command to generate it. This generates the secret-conf.properties file in the <API-M\_HOME>/repository/conf/security directory. If you have already generated this file, proceed to Step 2.  
sh ciphertool.sh -Dconfigure

Enter wso2carbon as the keystore password when the following appears in the console.

[Please Enter Primary KeyStore Password of Carbon Server : ]

2. Execute the following command from the <API-M\_HOME>/bin directory. This generates the encrypted value for the clear text password.

sh ciphertool.sh

- a. Enter wso2carbon when the following appears in the console.

[Please Enter Primary KeyStore Password of Carbon Server : ]

- b. Enter admin as the input value when the following appears in the log. (This value is entered based on the secret-conf.properties file you generated in step 1.)

[Enter Plain text value :]

[Please Enter value Again :]

The following output is displayed in the console.

Encryption is done Successfully

The encrypted value is:

```
MpfXhKP+iJSImA/KNa+DoOXCPQAYF3JLh1FNAdG6F3naWK+N1/WEWOJkFx4kK34i1VtKywNN9SiC
MRQGTFw+nqzK5/INGcFFdoxv491M/FJw8CyXKQ0JWdxw5QJPtrjJzvGp6Rj6xt4ysb6HdG5uNG+a
1E01qdmzGUYQ6oejIlk=
```

3. Open the <API-M\_HOME>/repository/conf/security/ cipher-text.properties file and add the following entry.

```
log4j.appenders.DAS_AGENT.password=MpfXhKP+iJSImA/KNa+DoOXCPQAYF3JLh1FNAdG6F3naWK+N1/WEWOJkFx4kK34i1VtKywNN9SiC
MRQGTFw+nqzK5/INGcFFdoxv491M/FJw8CyXKQ0JWdxw5QJPtrjJzvGp6Rj6xt4ysb6HdG5uNG+a
1E01qdmzGUYQ6oejIlk=
```

4. Open the <API-M\_HOME>/repository/conf/log4j.properties file and configure the password as log4j.appenders.DAS\_AGENT.password=secretAlias:log4j.appenders.DAS\_AGENT.password (as shown below).

```
log4j.appenders.DAS_AGENTt=org.wso2.carbon.analytics.shared.data.agent
s.log4j.appenders.LogEventAppender
log4j.appenders.DAS_AGENT.layout=org.wso2.carbon.analytics.shared.data
.agents.log4j.util.TenantAwarePatternLayout
log4j.appenders.DAS_AGENT.columnList=%D,%S,%A,%d,%c,%p,%m,%H,%I,%Stack
trace
log4j.appenders.DAS_AGENT.userName=admin
log4j.appenders.DAS_AGENT.password=secretAlias:log4j.appenders.DAS_AGEN
T.password
log4j.appenders.DAS_AGENT.url=tcp://localhost:7612
log4j.appenders.DAS_AGENT.maxTolerableConsecutiveFailure=5
log4j.appenders.DAS_AGENT.streamDef=loganalyzer:1.0.0
```

5. Start the WSO2 API Manager server by running one of the following commands from the <API-M\_ANALYTIC S\_HOME>/bin directory.

- On Windows: wso2server.bat --run
- On Linux/Mac OS: sh wso2server.sh

Enter wso2carbon when the following appears in the console.

[Enter KeyStore and Private Key Password :]

If you want to start the server as a background process, carry out the following steps before starting the

server.

1. Create a file named `password-tmp.txt` in the `<API-M_HOME>/repository/conf/datasources` directory. Add `wso2carbon` (the primary keystore password) to this file and save.

By default, the password provider assumes that both the private key and the keystore passwords are the same. If you want them to be different, the private key password should be entered in the second line of the file.

2. The keystore password is picked from the `password-tmp.txt` file. This file is automatically deleted from the file system when you start the server. Make sure to add this temporary file back whenever you start the server as a background process.

If you name the password file `password-persist.txt` instead of `password-tmp.txt`, then the file is not deleted once the server is started. Therefore, it is not required to provide the password in subsequent startups.

### **Configuring databases**

Configuring databases allow you to persist data relating to APIs, process them and analyze. Follow the procedure below to configure databases.

The following is a list of database versions that are compatible with WSO2 API-M Analytics.

- Postgres 9.5 and later
- MySQL 5.6
- MySQL 5.7
- Oracle 12c
- MS SQL Server 2012
- DB2

1. Open the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/analytics-datasources.xml` file. Note that two datasources named as `WSO2_ANALYTICS_EVENT_STORE_DB` and `WSO2_ANALYTIC_S_PROCESSED_DATA_STORE_DB` are configured by default to point to the H2 databases.
2. Create two database schemas in your database server (MySQL, Oracle, etc) for the two datasources, and change the configurations of those datasources to point to the relevant schemas. A sample configuration is given below.

The database user you provide here requires permissions to create tables.

Note that you do not need to run the database scripts against the created databases as the tables for the datasources are created at runtime.

```

<datasource>
 <name>WSO2_ANALYTICS_EVENT_STORE_DB</name>
 <description>The datasource used for analytics record
store</description>
 <definition type="RDBMS">
 <configuration>

 <url>jdbc:mysql://localhost:3306/stats_200?autoReconnect=true&rel
axAutoCommit=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>
 <defaultAutoCommit>false</defaultAutoCommit>
 </configuration>
 </definition>
</datasource>

```

- If you are using **Oracle**, its recommended to increase the DB block size as described in <http://www.oratable.com/ora-01450-maximum-key-length-exceeded/>, to avoid the error 'ORA-01450: maximum key length (6398) exceeded'.
- If you are using **DB2**, run [this script](#) before you start the WSO2 API-M Analytics server.
- If you are using **MySQL 5.7**, open `<API-M_ANALYTICS_HOME>/repository/conf/analytics/spark/spark-jdbc-config.xml` and configure the `stringType` property under the `typeMapping` element as follows.

```
<stringType>VARCHAR(100)</stringType>
```

If you are using **MSSQL**, add the `SendStringParametersAsUnicode` property to the database connection URL in the data source configuration in the `<API-M_ANALYTICS_HOME>/repository/conf/datasources/analytics-datasources.xml` file as shown below to avoid deadlock issues that are caused when the same table row is updated in two or more sessions at the same time.

```
<url>SQLSERVER_JDBC_URL;SendStringParametersAsUnicode=false</url>
```

3. Share the `WSO2AM_STATS_DB` datasource between WSO2 API-M and WSO2 API-M Analytics as follows.
  - a. Open the `<API-M_HOME>/repository/conf/datasources/master-datasources.xml` file and make sure that a configuration for the `WSO2AM_STATS_DB` datasource is included. The default configuration is as follows.

```

<datasource>
 <name>WSO2AM_STATS_DB</name>
 <description>The datasource used for setting statistics to API Manager</description>
 <jndiConfig>
 <name>jdbc/WSO2AM_STATS_DB</name>
 </jndiConfig>
 <definition type="RDBMS">
 <configuration>

 <url>jdbc:mysql://localhost:3306/WSO2AM_STATS_DB?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>
 <defaultAutoCommit>false</defaultAutoCommit>
 </configuration>
 </definition>
</datasource>

```

you need to enable analytics in publisher, store and gateway nodes. However, you need to add this datasource configuration in gateway nodes. Following table provides more information on Analytics usage of API Manager components in a distributed environment.

Component	Enable statistics	Events Published	Read statsDB
Gateway_Manager	YES only if accept request	YES only if accept request	NO
Gateway_worker	YES	YES	NO
Key Manager	NO	NO	NO
Publisher	YES	NO	YES
Store	YES	YES	YES
Traffic Manager	NO	NO	NO

You do not need to enable analytics in Key Manager and Traffic Manager nodes as those components do not read or publish statistics. Though gateway nodes publish events, they are not reading statistics database. Therefore you are not required to add the WSO2AM\_STATS\_DB datasource configuration in gateway nodes. Publisher node read statistics but not publishing events. Therefore you can disable event publisher initialization at

startup in publisher by setting **<SkipEventReceiverConnection>** value to true in **<PUBLISHER\_HOME>/repository/conf/api-manager.xml**. API Store node reads statistics and also publish events. Therefore we need to keep the statsource configuration for statsDB in Store node as well.

- b. Open the **<API-M\_ANALYTICS\_HOME>/repository/conf/datasources/stats-datasources.xml** file and make sure that the same configuration in the **<API-M\_HOME>/repository/conf/datasources/master-datasources.xml** file (mentioned in the previous sub step) is added in it.
4. Create a schema in your database server similar to the **WSO2AM\_STATS\_DB** datasource. Make sure that this datasource points to the relevant schema.

The database user you provide here requires permissions to create tables.

5. Download and copy the relevant database driver JAR file to the **<API-M\_ANALYTICS\_HOME>/repository/components/lib** directory.
6. Start the WSO2 API-M Analytics server.

## Troubleshooting

If you are configuring API-M Analytics with MSSQL and you get an error when you start the API-M Analytics server stating that a table cannot have more than one clustered index, follow the steps below.

1. Open the **<API-M\_ANALYTICS\_HOME>/repository/components/features/org.wso2.carbon.analytics.spark.server\_VERSION/spark-jdbc-config.xml** file.
2. Update the value for the **<indexCreateQuery>** element of the MySQL database as shown below.

```
<database name="Microsoft SQL Server">
 <indexCreateQuery>CREATE INDEX {{TABLE_NAME}}_INDEX ON
 {{TABLE_NAME}} ({{INDEX_COLUMNS}})</indexCreateQuery>
</database>
```

3. Restart the server for the above changes to take effect.

### Purging Analytics Data

You can remove historical data in API Manager Analytics by data purging. By purging data, you can achieve high performance in data analysis without removing analyzed summary data. When purging data, only the stream data generated by API Manager is purged. Refer [Purging Analytics Data](#) for more information.

### Analyzing APIM Statistics with Batch Analytics

The following topics cover how statistics relating to the APIs deployed in the WSO2 API Manager are analyzed via APIM - Analytics.

- [Introducing the WSO2 API Manager Statistics Model](#)
- [Viewing API Statistics](#)

- Using Geolocation Based Statistics

## Introducing the WSO2 API Manager Statistics Model

- Introduction
- API Manager usage publisher
- Workflow executor
- Data analyzer
- API Manager event streams
- API Manager summarized tables (APIM Analytics internal storage)
- API Manager statistics

### Introduction

This section describes and illustrates the API Manager statistic publishing and generating model. It describes the internal components of API Manager, external analyzer information and other data retrieval components. API Manager generates events based on the API Manager invocation pattern and publishes them to all the listening event analyzers. The analyzer is responsible for the accumulation of all events and generates summary data based on the defined summarisation logic. After the summarized data is generated, the API Manager Dashboard can retrieve statistics from the analyzer data-source to the UI via the API Manager analytics client.

### API Manager usage publisher

The internal API Manager component listens to the API Manager invocations and its behavior. Based on the request and responses, the event is generated and published to all the event receivers. This publisher publishes the following event streams,

- org.wso2.apimgt.statistics.request
- org.wso2.apimgt.statistics.response
- org.wso2.apimgt.statistics.fault
- org.wso2.apimgt.statistics.throttle

### Workflow executor

This component publishes the org.wso2.apimgt.statistics.workflow event stream containing work-flow related event data.

### Data analyzer

The data analyzer is a summary data generator based on the received event. WSO2 API-M uses WSO2 APIM Analytics to perform analytics tasks. APIM Analytics uses Apache Spark as a processing language. The <APIM-ANALYTICS\_HOME>/repository/deployment/server/carbonapps/org\_wso2\_carbon\_analytics\_apim-1.0.0.car CAR file (which is deployed in the APIM Analytics server by default) contains details of all summarized data and their destination.

A Carbon Application (**C-App**) or a **CAR file** is a collection of artifacts deployable on a WSO2 product instance. These artifacts are usually JAVA-based or XML configurations, designed differently for each product in the WSO2 Carbon platform. You can deploy these artifacts to generate services.

A single WSO2 product can have numerous artifacts such as Axis2 services, dataservices, synapse configurations, endpoints, proxy services, mediators, registry resources, BPEL workflows etc. Usually, these artifacts are created in a development environment and then moved one by one into staging/production environments. Manually configuring artifacts to build up the entire solution this way is a time-consuming task. Instead, you can bundle configuration files and artifacts in a C-App and port Web service based solutions across environments more easily. C-Apps allow you to export your entire solution as a single archive file.

## API Manager event streams

API-M 2.1.0 provides six types of event streams as listed below.

### *org.wso2.apimgt.statistics.request*

This stream tracks information for the API request.

<b>org.wso2.apimgt.statistics.request</b>			
meta_clientType : Client type	STRING	:	Meta information of
consumerKey : application	STRING	:	Consumer key of API invoked client
context : request	STRING	:	API context depending on the user's
api_version : [API Provider +---+API Name]	STRING	:	API synapse artifact contained name
api : resourcePath	STRING	:	API Name
request	STRING	:	API resource URL pattern of API
method : [e.g.:GET/POST]	STRING	:	HTTP Verb of API request
version : request	STRING	:	API version
requestTime : userId	LONG	:	Request count (e.g. 1)
tenantDomain : hostName	STRING	:	API request hit time in APIM
hostName : apiPublisher	STRING	:	API invoked end user name
apiPublisher : applicationName	STRING	:	Tenant domain of API provider
applicationName : applicationId	STRING	:	API Manager server host
applicationId : userAgent	STRING	:	API provider
userAgent : tier	STRING	:	Name of the client application
tier : assigned to the request	STRING	:	ID of the client application
assigned to the request : throttledOut	BOOL	:	User agent of the user
throttledOut : clientip	STRING	:	Name of the throttling policy
clientip : applicationOwner	STRING	:	Describes whether this is a throttled
applicationOwner : the Application	STRING	:	request or not
the Application : clientip	STRING	:	IP Adress of the
clientip : applicationOwner	STRING	:	Name of the Owner of
applicationOwner : the Application	STRING	:	the Application

### *org.wso2.apimgt.statistics.response*

This stream tracks information for the API response. It includes the time taken for the response to get back, the received time etc.

**org.wso2.apimgt.statistics.response**

meta_clientType :	STRING	:	Meta information of Client type
consumerKey :	STRING	:	Consumer key of the client application invoking the API
context :	STRING	:	API context depending on the user's request
api_version :	STRING	:	API synapse artifact contained name [API Provider +---+API Name]
api :	STRING	:	API Name
resourcePath :	STRING	:	API resource URL pattern of API request
resourceTemplate:	STRING	:	API resource URL template of the API request
method :	STRING	:	HTTP Verb of API request [e.g.:GET/POST]
version :	STRING	:	API version
response :	INT	:	Response count (e.g. 1)
responseTime :	LONG	:	Total time taken for request/response flow[serviceTime+backendTime]
serviceTime :	LONG	:	Time taken to serve the API request in APIM side
backendTime :	LONG	:	Time taken process the request at the backend
username :	STRING	:	API invoked end user name
eventTime :	LONG	:	Timestamp of response event published
tenantDomain :	STRING	:	Tenant domain of API provider
hostname :	STRING	:	API Manager server hostname
apiPublisher :	STRING	:	API provider
applicationName :	STRING	:	Name of the client application
applicationId :	STRING	:	ID of the client application
cacheHit :	BOOL	:	Describes if response caching is enabled or not
responseSize :	LONG	:	Response message size in bytes
protocol :	STRING	:	Protocol used to send the response (HTTP/HTTPS) and the port
responseCode :	STRING	:	HTTP Response Code
destination :	STRING	:	URL of the Desination IP

**org.wso2.apimgt.statistics.fault**

This stream contains the fault API invocations. It includes the API with back end errors, timeout etc.

**org.wso2.apimgt.statistics.fault**

meta_clientType :	STRING	:	Meta information of Client Type
consumerKey :	STRING	:	Consumer key of the client
application invoking the API			
context :	STRING	:	API context depending on the user's request
api_version :	STRING	:	API version
api :	STRING	:	API Name
resourcePath :	STRING	:	API resource url pattern of API request
method :	STRING	:	HTTP Verb of API request [e.g.:GET/POST]
version :	STRING	:	API version
errorCode :	STRING	:	HTTP error code
errorMessage :	STRING	:	Description of error message
requestTime :	LONG	:	API request time in millisecond
userId :	STRING	:	API invoked end user name
tenantDomain :	STRING	:	Tenant domain of API provider
hostName :	STRING	:	API Manager server host
apiPublisher :	STRING	:	API provider
applicationName :	STRING	:	Name of the client application
applicationId :	STRING	:	ID of the client application
protocol :	STRING	:	Protocol used to send the response (HTTP/HTTPS) and the port

**org.wso2.apimgt.statistics.throttle**

This stream contains the API invocation with throttle information. Throttling can happen due to any of the following reasons:

- The application limit has exceeded.
- The resource limit has exceeded.
- The API limit has exceeded.
- The hard level limit has exceeded.

**org.wso2.apimgt.statistics.throttle**

meta_clientType	:	STRING	:	meta information
of Client Type				
accessToken	:	STRING	:	Access token of the request
userId	:	STRING	:	API invoked end user name
tenantDomain	:	STRING	:	Tenant domain of API provider
api	:	STRING	:	API Name
api_version	:	STRING	:	API synapse artifact contained
name [API Provider +---+API Name]				
context	:	STRING	:	API context depending on the
user's request				
apiPublisher	:	STRING	:	API provider
throttledTime	:	LONG	:	The timestamp which throttle out
event triggers				
applicationName	:	STRING	:	Name of the client application
applicationId	:	STRING	:	ID of the client application
subscriber	:	STRING	:	Name of the
subscriber of the Application				
throttledOutReason	:	STRING	:	The reason describing why the
request has been throttled out				

**org.wso2.apimgt.statistics.workflow**

This event stream creates events based on the API Manager workflow and publishes them to the analyzer.

**org.wso2.apimgt.statistics.workflow**

meta_clientType	:	STRING	:	meta information
of Client Type				
workflowReference	:	STRING	:	Holds the workflow reference ID
workflowStatus	:	STRING	:	Status of the workflow e.g.:
CREATED, APPROVED, REJECTED, REGISTERED				
tenantDomain	:	STRING	:	Tenant domain of subscriber who
triggers the workflow in APIStr				
workflowType	:	STRING	:	Type of the workflow
e.g.:AM_APPLICATION_CREATION,				
createdTime	:	LONG	:	The workflow was creation time in
milliseconds				
updatedTime	:	LONG	:	The last updated time of the
workflow in milliseconds				

**org.wso2.apimgt.statistics.execution.time**

This stream contains information relating to API invocation including time stamps and the time taken by the API at different stages of invocation (e.g., time taken to backend, time taken to mediation flow, response time etc.).

**org.wso2.apimgt.statistics.execution.time**

meta_clientType	:	STRING	:	meta information of Client Type
api	:	STRING	:	API Name
api_version	:	STRING	:	API Version
tenantDomain	:	STRING	:	Tenant domain of subscriber who triggers the workflow in APIStore
apiPublisher	:	STRING	:	API Provider
apiResponseTime	:	LONG	:	Total time taken for request/response flow
context	:	STRING	:	API context depending on the user's request
securityLatency	:	LONG	:	Time taken for authentication
throttlingLatency	:	LONG	:	Time taken for throttling the request/response
requestMediationLatency	:	LONG	:	Time taken to mediate the request
responseMediationLatency	:	LONG	:	Time taken to mediate the response
backendLatency	:	LONG	:	Time taken by the backend to return the response
otherLatency	:	LONG	:	Time taken to process tasks other than mentioned above
eventTime	:	LONG	:	Timestamp of the sent event

**API Manager summarized tables (APIM Analytics internal storage)**

Note that these summarized tables are stored in the APIM Analytics internal storage. Both the C-Apps that API Manager deploy on APIM Analytics first stores the summary data into these tables. There are additional columns in some of these tables containing the composition column of other columns. Those columns types are of facet type, which is used to support the aggregation function on the APIM Analytics REST API. All the columns are indexed in order to search using Apache Lucene and supports the APIM Analytics REST API. When API Manager is configured with the RDBMS client, all these tables are replicated in the external RDBMS except for the facet columns.

- **API\_REQUEST\_SUMMARY**

This table contains the summary data of the request event stream.

**API\_REQUEST\_SUMMARY table schema**

```
CREATE TEMPORARY TABLE API_REQUEST_SUMMARY_FINAL USING CarbonAnalytics
OPTIONS (tableName "API_REQUEST_SUMMARY",
 schema "api string -i,
 api_version string -i,
 version string -i,
 apiPublisher string -i,
 consumerKey string -i,
 userId string -i,
 context string -i,
 max_request_time long -i,
 total_request_count int -i,
 hostName string -i,
 year int -i,
 month int -i,
 day int -i,
 time string -i,
 key_api_facet facet -i,
 key_userId_facet facet -i,
 api_version_userId_facet facet -i,
 api_version_userId_apiPublisher_facet facet -i,
 api_version_userId_context_facet facet -i",
 primaryKeys
 "api,api_version,version,apiPublisher,consumerKey,userId,context,host
Name,year,month,day"
);
```

▼ Expand to find the table of descriptions for each column

```

api : API Name
api_version : API synapse artifact contained name [API Provider
+---+API Name]
version : API version
apiPublisher : API provider
context : API context depending on the user's request
consumerKey : Consumer key of the client application invoking the
API
userId : End user name invoked by the API
max_request_time: Time of the latest API request occurrence
total_request_count: Total request count for the requests coming
for same API
hostname : APIM server hostname
year : The year of initial API request occurred of the batch of
API requests
month : The month of initial API request occurred of the batch of
API requests
day : The date of API initial request occurred of the batch of
API requests
time : The time of API initial request occurred of the batch of
API requests

```

- **API\_VERSION\_USAGE\_SUMMARY**

This table contains the summary data for API Manager usage. It is also derived from the request event table.

#### VERSION\_USAGE\_SUMMARY table schema

```

CREATE TEMPORARY TABLE API_VERSION_USAGE_SUMMARY_FINAL USING
CarbonAnalytics OPTIONS (tableName "API_VERSION_USAGE_SUMMARY",
schema "api string -i,
version string -i,
apiPublisher string -i,
context string -i,
total_request_count int -i,
hostName string -i,
year int -i,
month int -i,
day int -i,
time string -i,
max_request_time long -i,
api_version_context_facet facet -i",
primaryKeys
"api,version,apiPublisher,context,hostName,year,month,day"
);

```

▼ Expand to find the table of descriptions for each column

```

api : API Name
version : API version
apiPublisher : API provider
context : API context depending on the user's request
total_request_count: Total request count of an API version
hostname : APIM server hostname
year : The year of initial API request occurred of the batch of
API requests
month : The month of initial API request occurred of the batch of
API requests
day : The date of API initial request occurred of the batch of
API requests
time : The time of API initial request occurred of the batch of
API requests

```

- **API\_Resource\_USAGE\_SUMMARY**

This table contains the summarized data for API Manager usage by resources and it is also derived from request event table.

#### **API\_Resource\_USAGE\_SUMMARY table schema**

```

CREATE TEMPORARY TABLE API_Resource_USAGE_SUMMARY_FINAL USING
CarbonAnalytics OPTIONS (tableName "API_Resource_USAGE_SUMMARY",
schema "api string -i,
version string -i,
apiPublisher string -i,
consumerKey string -i,
resourcePath string -i,
context string -i,
method string -i,
total_request_count int -i,
hostName string -i,
year int -i,
month int -i,
day int -i,
time string -i,
max_request_time long -i,
key_api_method_path_facet facet -i,
api_version_context_method_facet facet -i",
primaryKeys
"api,version,apiPublisher,consumerKey,context,resourcePath,method,hos
tName,year,month,day"
);

```

▼ Expand to find the table of descriptions for each column

```

api : API Name
version : API version
apiPublisher : API provider
context : API context depending on the user's request
consumerKey : Consumer key of the client application invoking the
API
resourcePath : API resource url pattern of API request
method : HTTP Verb of API request [eg:GET/POST]
total_request_count :Total request count for a particular API
resource pattern
hostname : API Manager server hostname
year : The year of initial API request occurred of the batch of
API requests
month : The month of initial API request occurred of the batch of
API requests
day : The date of API initial request occurred of the batch of
API requests
time : The time of API initial request occurred of the batch of
API requests

```

- **API\_RESPONSE\_SUMMARY**

This table contains the summarized data from API responses. It is derived from the response event table.

#### **API\_RESPONSE\_SUMMARY table schema**

```

CREATE TEMPORARY TABLE API_RESPONSE_SUMMARY_FINAL USING
CarbonAnalytics OPTIONS (tableName "API_RESPONSE_SUMMARY",
schema "api_version string -i,
apiPublisher string -i,
context string -i,
serviceTime int -i,
total_response_count int -i,
hostName string -i,
year int -i,
month int -i,
day int -i,
time string -i,
max_request_time long -i,
api_version_context_facet facet -i",
primaryKeys
"api,version,apiPublisher,context,hostName,year,month,day"
);

```

▼ Expand to find the table of descriptions for each column

```

api_version : API synapse artifact contained name [API Provider
+---+API Name]
apiPublisher : API provider
context : API context depending on the user's request
consumerKey : Consumer key of the client application invoking the
API
serviceTime : Total time taken to serve the batch of API requests
in APIM side
total_response_count: Total response count for the API requests for
a specific API
hostname : API Manager server hostname
year : The year of initial API request occurred of the batch of
API requests
month : The month of initial API request occurred of the batch
of API requests
day : The date of API initial request occurred of the batch of
API requests
time : The time of API initial request occurred of the batch of
API requests

```

- **API\_FAULT\_SUMMARY**

This table contains the summarized data of faulty API invocations and is derived from the fault event stream.

#### **API\_FAULT\_SUMMARY table schema**

```

CREATE TEMPORARY TABLE API_FAULT_SUMMARY_FINAL USING CarbonAnalytics
OPTIONS (tableName "API_FAULT_SUMMARY",
schema "api string -i,
version string -i,
apiPublisher string -i,
consumerKey string -i,
context string -i,
total_fault_count int -i,
hostName string -i,
year int -i,
month int -i,
day int -i,
time string -i,
max_request_time long -i,
consumerKey_api_facet facet -i,
api_version_apiPublisher_context_facet facet -i",
primaryKeys
"api,version,apiPublisher,consumerKey,context,hostName,year,month,day"
);

```

▼ Expand to find the table of descriptions for each column

```

api : API Name
version : API version
apiPublisher : API provider
consumerKey : Consumer key of the client application invoking the
API
context : API context depending on the user's request
total_fault_count: Total faulty API request count for a specific
API
hostname : APIM server hostname
year : The year of initial API request occurred of the batch of
API requests
month : The month of initial API request occurred of the batch of
API requests
day : The date of API initial request occurred of the batch of
API requests
time : The time of API initial request occurred of the batch of
API requests

```

- **API\_DESTINATION\_SUMMARY**

This table contains the summarized data of the API destinations and is derived from the destination event stream.

#### **API\_DESTINATION\_SUMMARY table schema**

```

CREATE TEMPORARY TABLE API_DESTINATION_SUMMARY_FINAL USING
CarbonAnalytics OPTIONS (tableName "API_DESTINATION_SUMMARY",
schema "api string -i,
version string -i,
apiPublisher string -i,
context string -i,
destination string -i,
total_request_count int -i,
hostName string -i,
year int -i,
month int -i,
day int -i,
time string -i,
max_request_time long -i,
api_version_context_dest_facet facet -i",
primaryKeys
"api,version,apiPublisher,context,destination,hostName,year,month,day"
);

```

▼ Expand to find the table of descriptions for each column

```

api : API Name
version : API version
apiPublisher : API provider
context : API context depending on the user's request
destination : API endpoint hostname
total_request_count : Total request count for the requests coming
for same API to same destination
hostname : APIM server hostname
year : The year of initial API request occurred of the batch of
API requests
month : The month of initial API request occurred of the batch
of API requests
day : The date of API initial request occurred of the batch of
API requests
time : The time of API initial request occurred of the batch of
API requests

```

- **API\_LAST\_ACCESS\_TIME\_SUMMARY**

This table contains the summary data of the last access times of the API and is derived from the request event stream.

#### API\_LAST\_ACCESS\_TIME\_SUMMARY table schema

```

CREATE TEMPORARY TABLE API_LAST_ACCESS_TIME_SUMMARY_FINAL USING
CarbonAnalytics OPTIONS (tableName "API_LAST_ACCESS_TIME_SUMMARY",
schema "tenantDomain string -i,
apiPublisher string -i,
api string -i,
version string -i,
userId string -i,
context string -i,
max_request_time long -i",
primaryKeys "tenantDomain,apiPublisher,api"
);

```

▼ Expand to find the table of descriptions for each column

```

api : API Name
version : API version
apiPublisher : API provider
context : API context depending on the user's request
userId : API invoked end user name
tenantDomain : Tenant domain of API provider
max_request_time: Time of the latest API request occurrence

```

- **API\_THROTTLED\_OUT\_SUMMARY**

This table contains the summary of the throttle out API invocation data. It is derived from the throttle out event table and request table.

**API\_THROTTLED\_OUT\_SUMMARY table schema**

```

CREATE TEMPORARY TABLE THROTTLED_OUT_FINAL_SUMMARY USING
CarbonAnalytics OPTIONS (tableName "API_THROTTLED_OUT_SUMMARY",
 schema "api string -i,
 api_version string -i,
 context string -i,
 apiPublisher string -i,
 applicationName string -i,
 tenantDomain string -i,
 year int -i,
 month int -i,
 day int -i,
 week int -i,
 time string -i,
 success_request_count int -i,
 throttleout_count int -i,
 max_request_time long -i,
 api_year_month_week_day_facet facet -i,
 applicationName_facet facet -i",
 primaryKeys
"api,api_version,context,apiPublisher,applicationName,tenantDomain,year,month,day"
);

```

## ▼ Expand to find the table of descriptions for each column

api	: API Name
api_version	: API synapse artifact contained name [API Provider +---+API Name]
apiPublisher	: API provider
context	: API context depending on the user's request
applicationName	: Name of the client application
tenantDomain	: Tenant domain of API provider
throttleout_count	: Total throttled out API request count for the particular API
success_request_count	: Total successful API request count for the particular API
year	: The year of initial API request occurred of the batch of API requests
month	: The month of initial API request occurred of the batch of API requests
day	: The date of API initial request occurred of the batch of API requests
time	: The time of API initial request occurred of the batch of API requests

**API Manager statistics**

API statistics are provided in both the API Publisher and the API Store. Apart from the number of subscriptions per API, all other statistical dashboards require an instance of WSO2 Data Analytics Server installed. For information on

the available statistics and how to view them, see [Viewing API Statistics](#).

## Viewing API Statistics

API statistics are provided in both the API Publisher and the API Store. For instructions on how to set up Analytics, see [Configuring APIM Analytics](#). Once Analytics is set up, follow the instructions below to view statistics through the API Publisher and API Store.

First, [invoke a few APIs](#) to generate traffic and see the statistics.

The following gadgets on the API Manager statistical dashboards, display real runtime statistics even when Analytics is not set up (as described in [Configuring APIM Analytics](#)).

- Published APIs Over Time
- Applications Created Over Time
- Developer Signups Over Time
- Subscriptions Created Over Time
- API subscriptions

The other gadgets you see on the API Manager statistical dashboards without setting up Analytics, are just samples and are not based on real runtime statistics of your server.

Please note that our data summarization logic summarizes the data on a **per day** basis.

The sections below explain how to access the statistical dashboards:

- [API Publisher statistics](#)
- [API Store statistics](#)
- [Admin Portal Statistics](#)

### API Publisher statistics

The gadgets that display publisher statistics can only be viewed via users that have permission to create APIs. For more information, see [Managing Users and Roles](#).

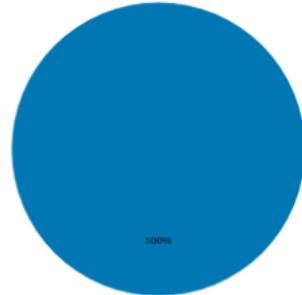
Log in to the API Publisher. Anyone who can create and/or publish APIs can view the API-level usage and subscription statistics by clicking on a selected API and referring to its **Versions** and **Users** tabs.

## PizzaShackAPI - 1.0.0

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

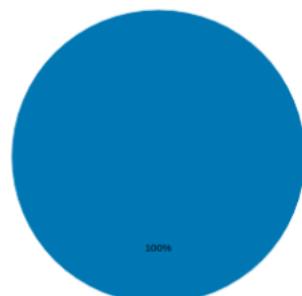
## API Subscriptions by Versions

Version	Number of API calls
1.0.0	4



## API Usage by Versions

Version	Number of API calls
1.0.0	325



## PizzaShackAPI - 1.0.0

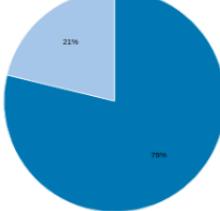
Overview   Lifecycle   Versions   Docs   **Users**

Active Subscriptions

Name	Date of User Registration
admin	12/19/2017, 12:23:19 PM
JohnDoe	12/19/2017, 12:28:58 PM

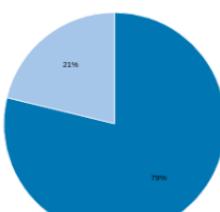
Usage by Current Subscribers (v-1.0.0)

Subscriber	Number of API calls
JohnDoe@carbon.super	256
admin@carbon.super	69

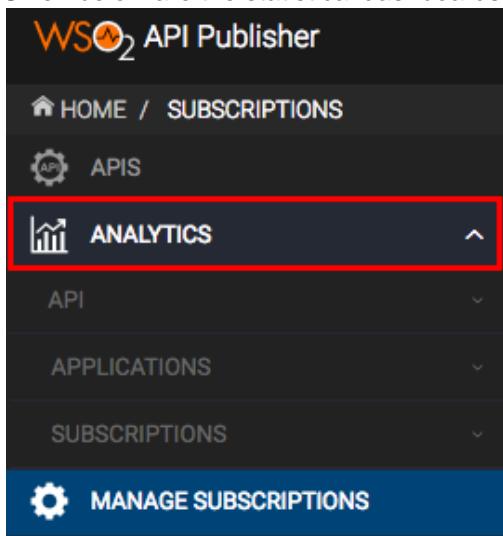


Usage by Current Subscribers (Across All Versions)

Subscriber	Number of API calls
JohnDoe@carbon.super	256
admin@carbon.super	69



Given below are the statistical dashboards that are available from the **Analytics** menu.



**API**

- CREATED APIs OVER TIME
- API USAGE
- TOP API USERS
- API LAST ACCESS TIMES
- USAGE BY RESOURCE PATH
- USAGE BY DESTINATION
- API USAGE COMPARISON
- API THROTTLED REQUESTS
- FAULTY INVOCATIONS
- API LATENCY
- API USAGE ACROSS GEO LOCATIONS
- API USAGE ACROSS USAGE AGENT

**APPLICATIONS**

- APP THROTTLED REQUESTS
- APPLICATIONS CREATED OVER TIME

**SUBSCRIPTIONS**

- API SUBSCRIPTIONS
- DEVELOPER SIGNUPS OVER TIME
- SUBSCRIPTIONS CREATED OVER TIME

In each of the dashboards, you can choose to view all APIs or if you are an API creator, only the APIs that you have created. You can also select the time period for which you wish to view the statistics.



Several examples of usage and performance statistics are given below:

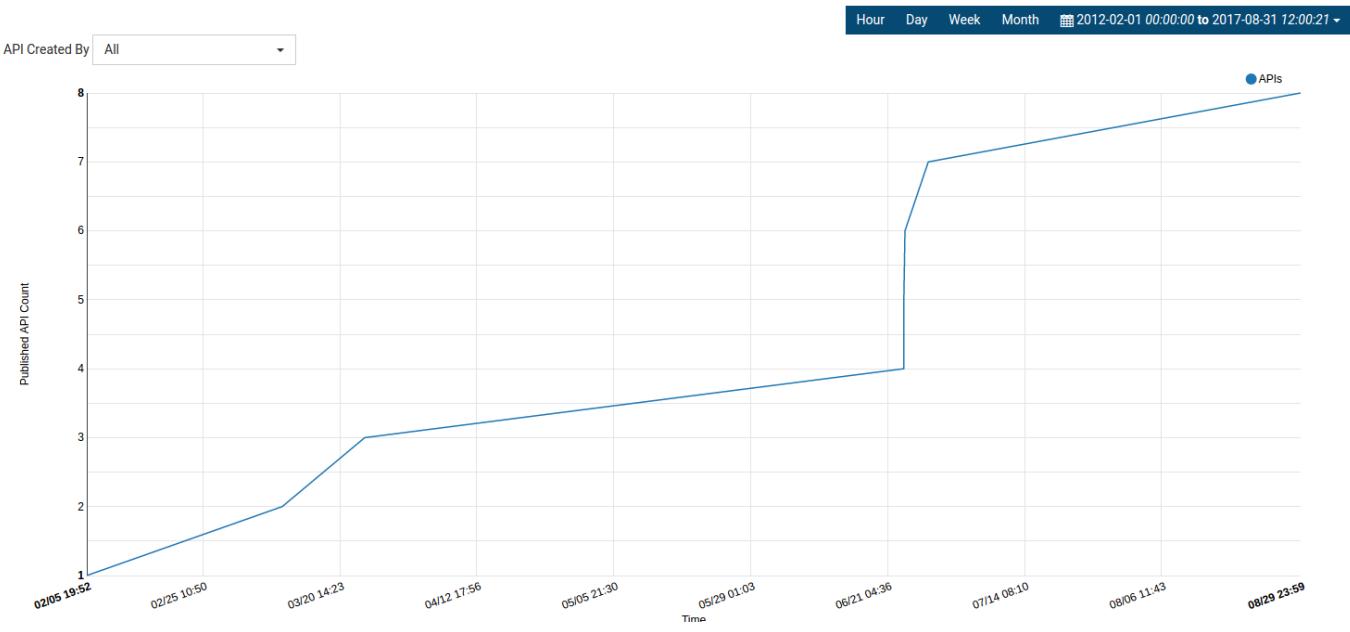
- [Created APIs Over Time](#)
- [API Usage](#)
- [API Last Access Times](#)
- [Usage by Resource Path](#)

- Usage by Destination
- API Usage Comparison
- API Throttled Requests
- Faulty Invocations
- API Latency Time
- API Usage Across Geo Locations
- API Usage Across User Agent
- App Throttled Requests
- Applications Created Over Time
- API Subscriptions
- Developer Signups Over Time
- Subscriptions Created Over Time
- API Usage per Application
- Top Users per Application
- Resource Usage per Application
- Faulty Invocations per Application
- Availability of APIs

#### Created APIs Over Time

The number of APIs published over a given time period.

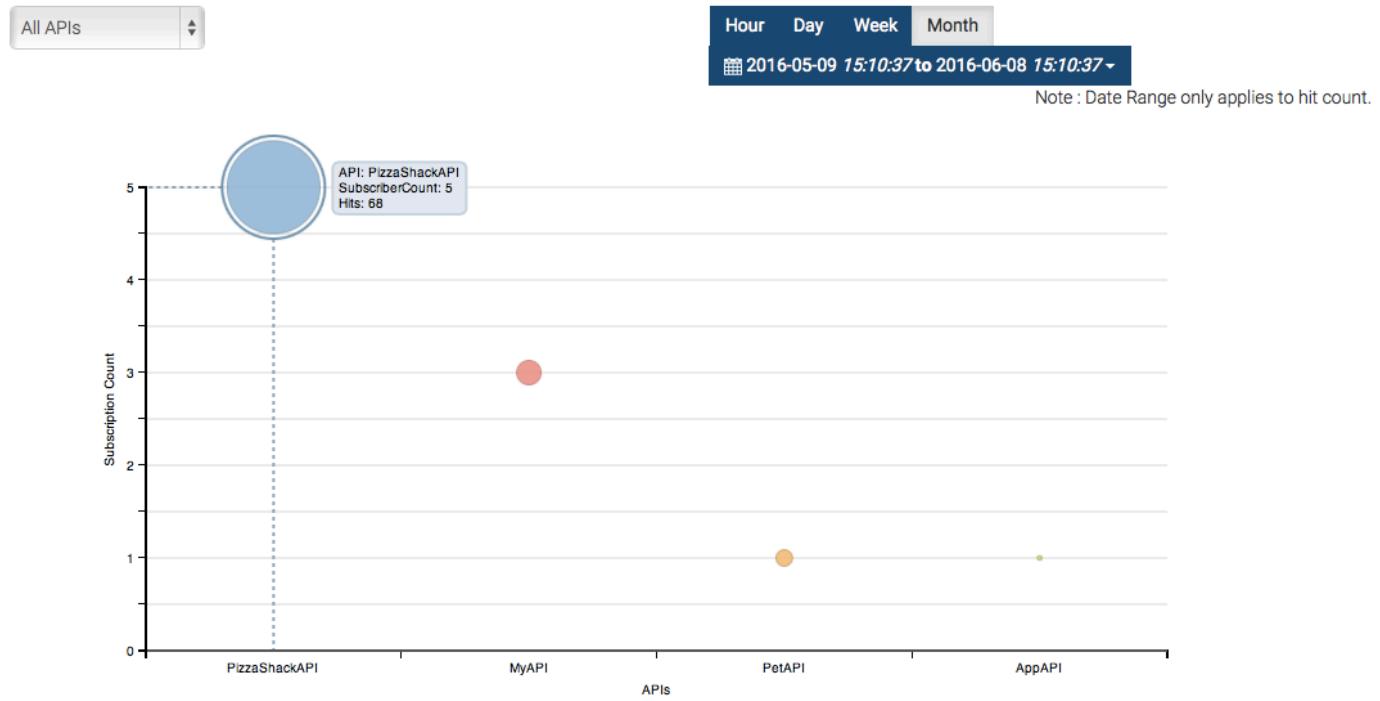
#### Created APIs Over Time



#### API Usage

The number of subscriptions of each API with a graphical view of the count.

## Overall API Usage (Across All Versions)



### API Last Access Times

A tabular representation of the last access times of an API, according to the version and the accessed subscriber.

## API Last Access Times (Across All Versions)

All APIs ▼

Hour Day Week Month

2016-05-09 15:05:13 to 2016-06-08 15:05:13 ▾

Filter by ...

API	Version	Subscriber	Access Time (GMT+05:30)
AppAPI (admin)	1.0.0	admin@carbon.super	June 8, 2016 2:48:00 PM GMT+05:30
PizzaShackAPI (admin)	1.0.0	sub1@carbon.super	June 8, 2016 2:31:00 PM GMT+05:30
MyAPI (admin)	1.0.0	admin@carbon.super	June 8, 2016 2:24:00 PM GMT+05:30
PetAPI (admin)	1.0.0	admin@carbon.super	June 8, 2016 12:44:00 PM GMT+05:30

Show  entries Showing 1 to 4 of 4 entries

### Usage by Resource Path

The number of invocations made by resources for an API, represented in a tabular view.

## API Usage by Resource Path

All APIs ▾

Hour Day Week Month 2014-05-10 00:00:00 to 2016-11-30 11:03:31 ▾

Filter by ...

api	(F)	version	resourcePath	method	Hits
api2		v1.2.0	/*	GET	1
api2		v1.2.0	/context	GET	1
mockapi		v2.0.0	/menu	GET	1
pizza		v1.0.0	/menu	GET	3
pizza		v1.0.0	/order	POST	1

Show 10 entries | Showing 1 to 5 of 5 entries

## Usage by Destination

The number of times an API was accessed based on the destination, represented in a tabular view.

## API Usage by Destination

All A ▾

Hour Day Week Month 2016-05-09 15:05:53 to 2016-06-08 15:05:53 ▾

Filter by ...

API	VERSION	CONTEXT	DESTINATION ADDRESS	NO OF ACCESS
MyAPI	1.0.0	/myAPI/1.0.0	https://localhost:9443/am/sample/pizzashack/v1/api/	9

Show 10 entries | Showing 1 to 1 of 1 entries

## API Usage Comparison

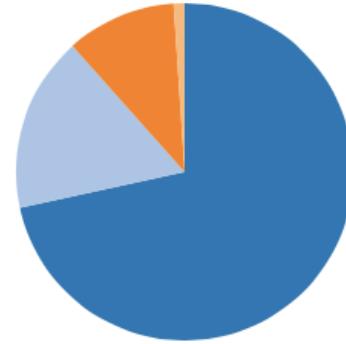
The number of invocations made for an API represented as a combination of all resources and all versions of the API.

## API Usage Comparison (Across All Versions)

All APIs

Hour Day Week Month

2016-05-09 15:06:11 to 2016-06-08 15:06:11

 Filter by ...

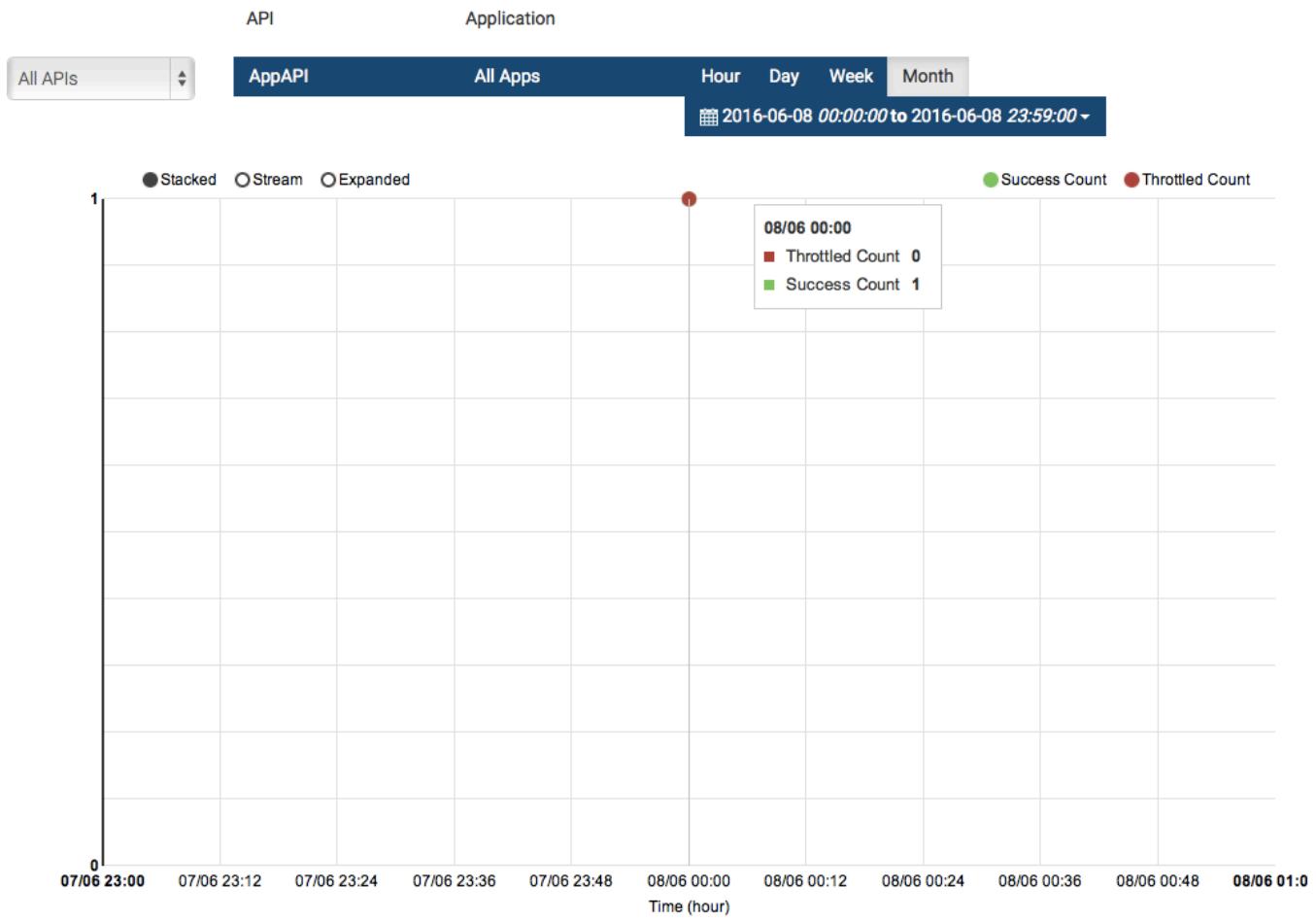
API	Hits
PizzaShackAPI	68
MyAPI	16
PetAPI	10
AppAPI	1

Show 10 entries | Showing 1 to 4 of 4 entries

API Throttled Requests

The total count of the successful request count and throttled request count towards an API over time.

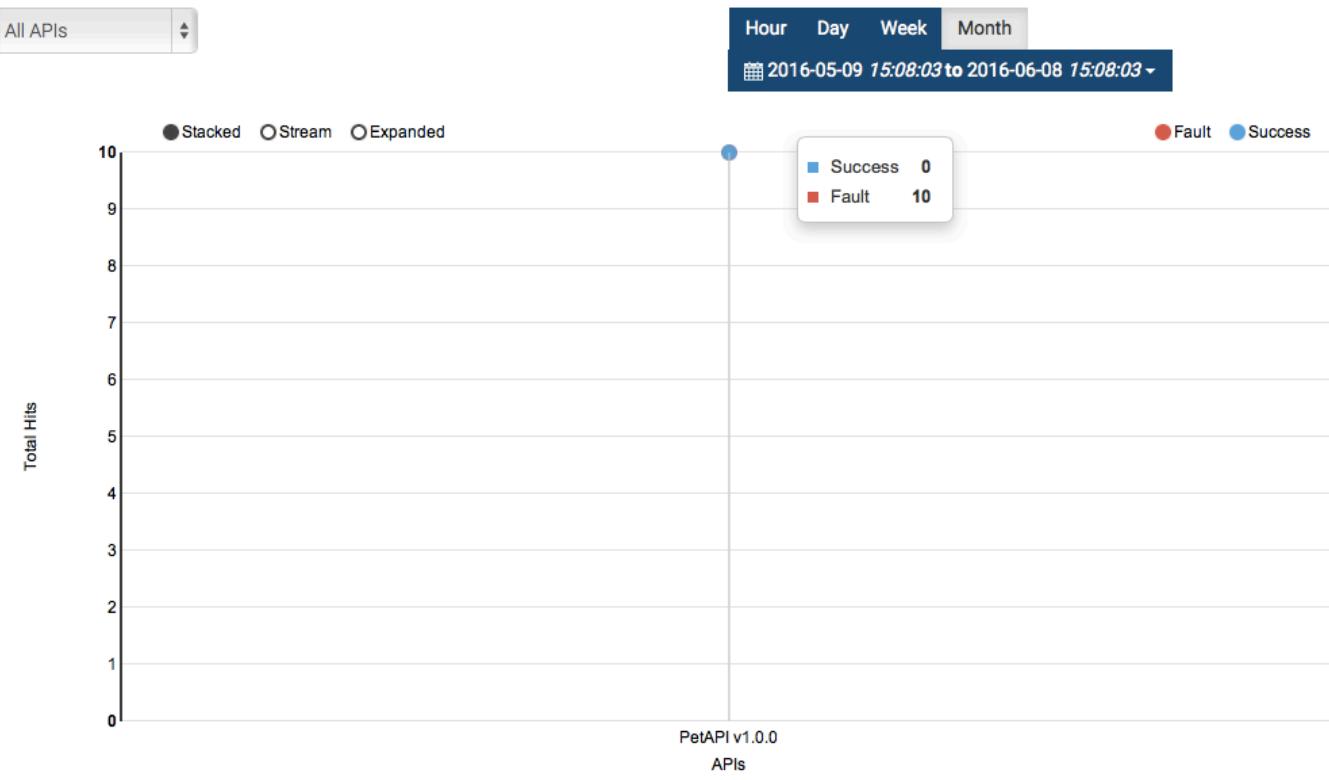
## API Throttled Out Requests



### Faulty Invocations

A successful invocation is when an API receives the expected response. If it results in any kind of error response, that invocation is considered a faulty invocation. The total number of invocations of an API represented as a combination of successful and faulty invocations is denoted in the faulty invocations graph.

## Faulty Invocations

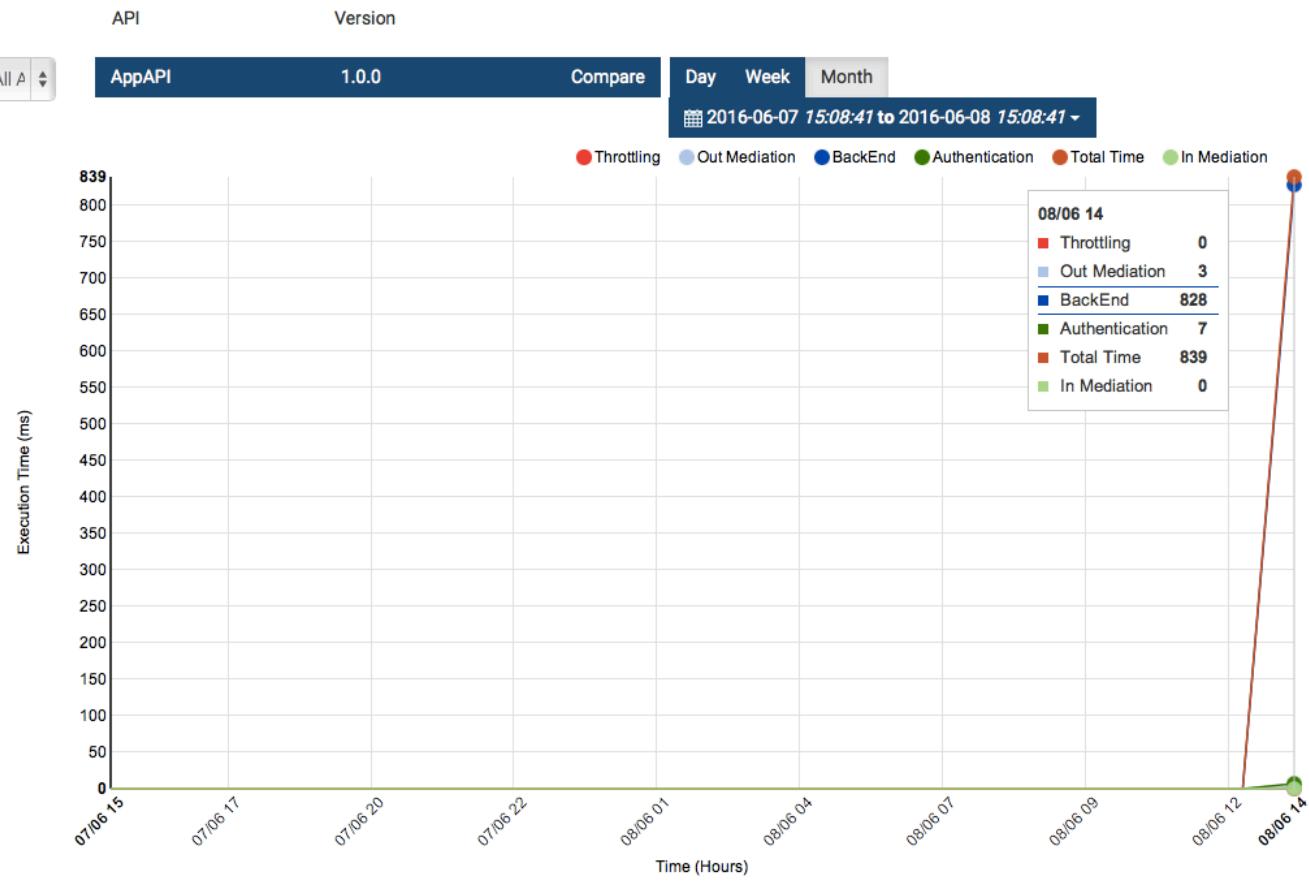


API Latency Time

The execution time of the APIs represented as a combination of throttling, In mediation, Out mediation, backend response time, and authentication time.

You can also see a comparison view of the latencies.

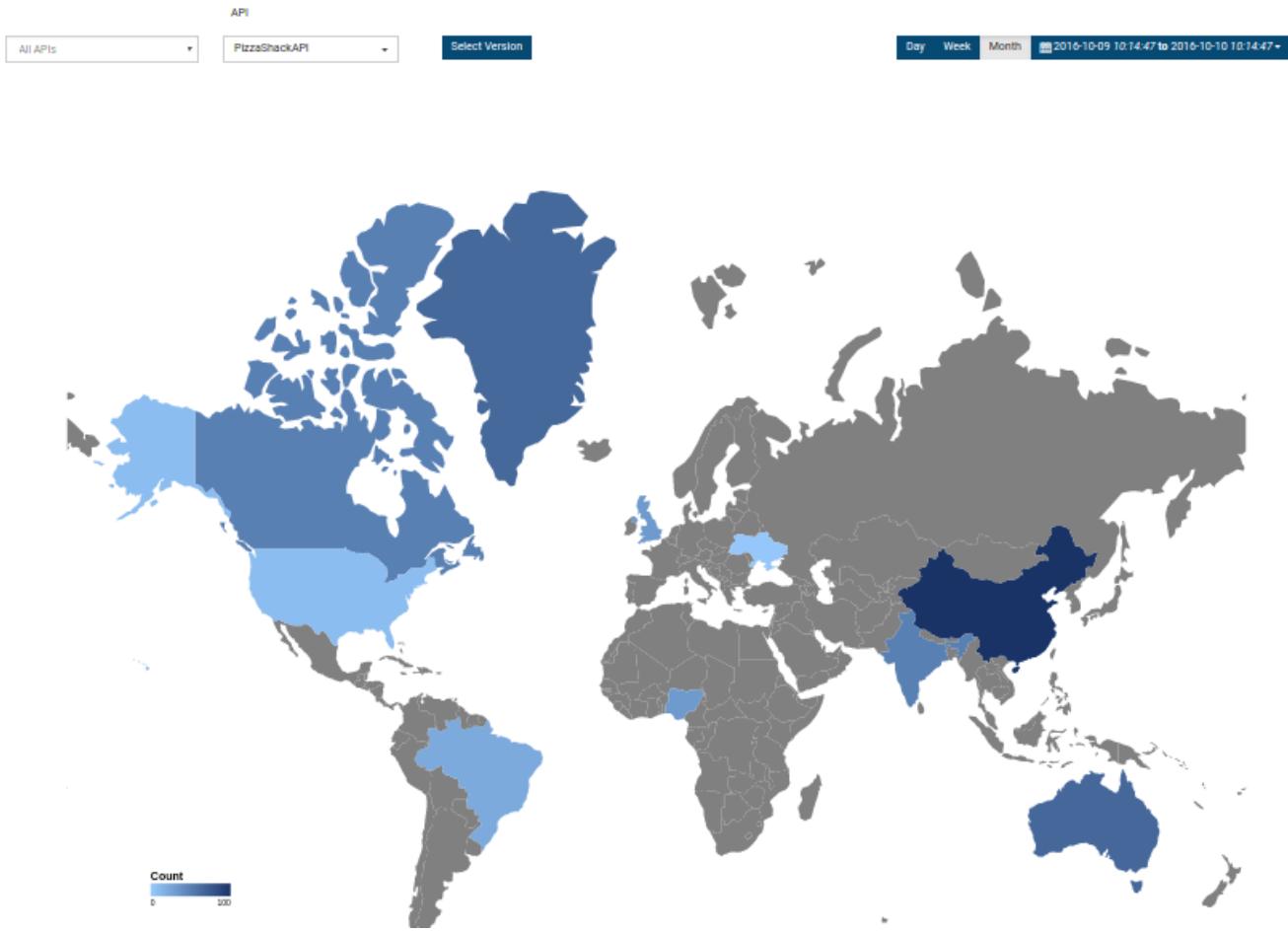
## API Latency BreakDown



### API Usage Across Geo Locations

The data script that updates statistics related to geo locations is executed once a day. Therefore, at a given time, some of the statistics generated within the last 24 hours may not be displayed in this gadget.

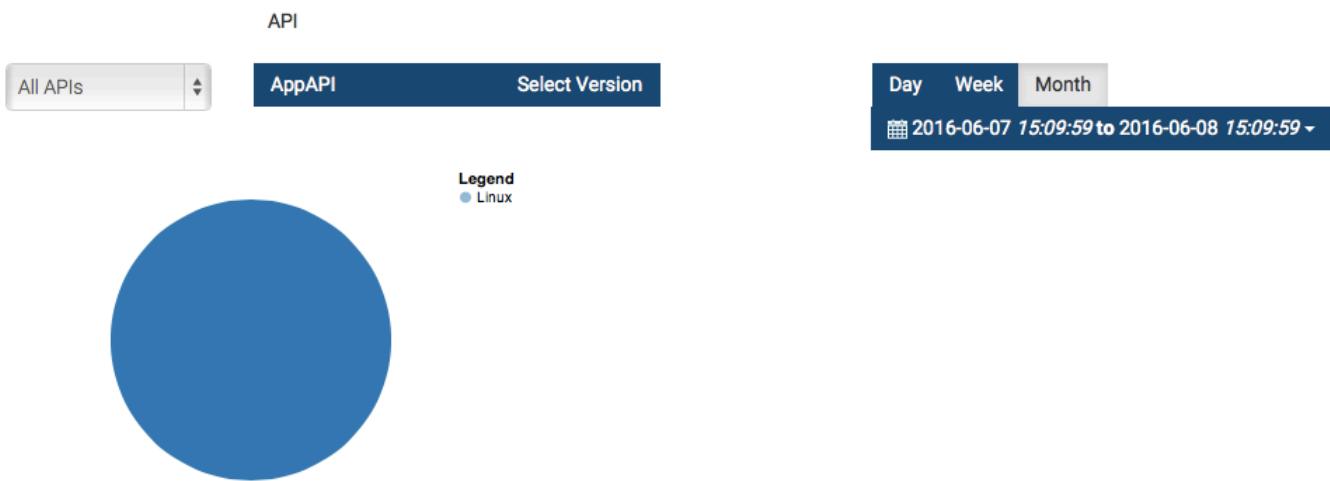
## API Usage Across Geo Locations



API Usage Across User Agent

The proportional distribution of the usage (invocation) of each API differentiated by the user agent HTTP header received in requests towards the API.

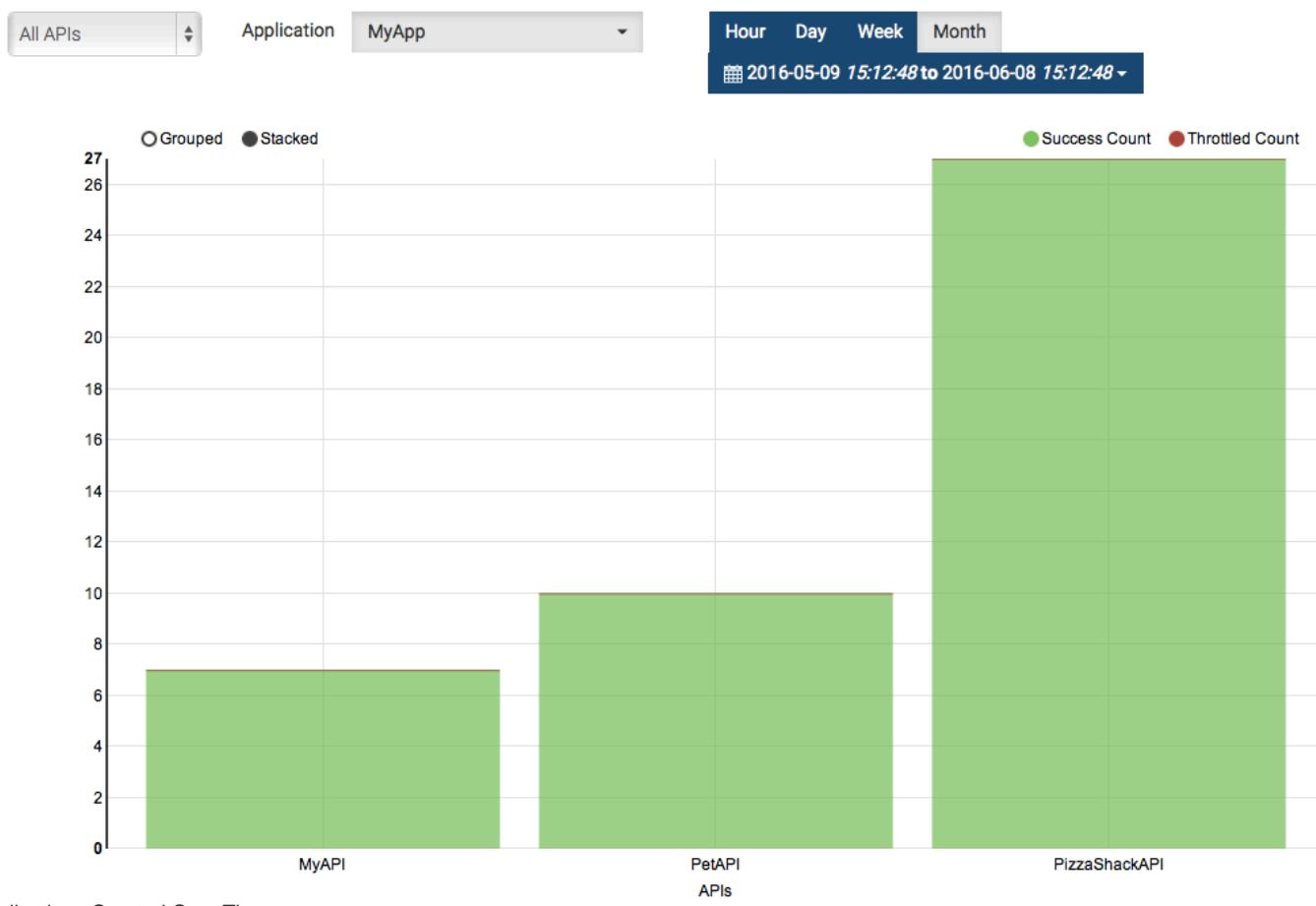
## API Usage Across Usage Agent



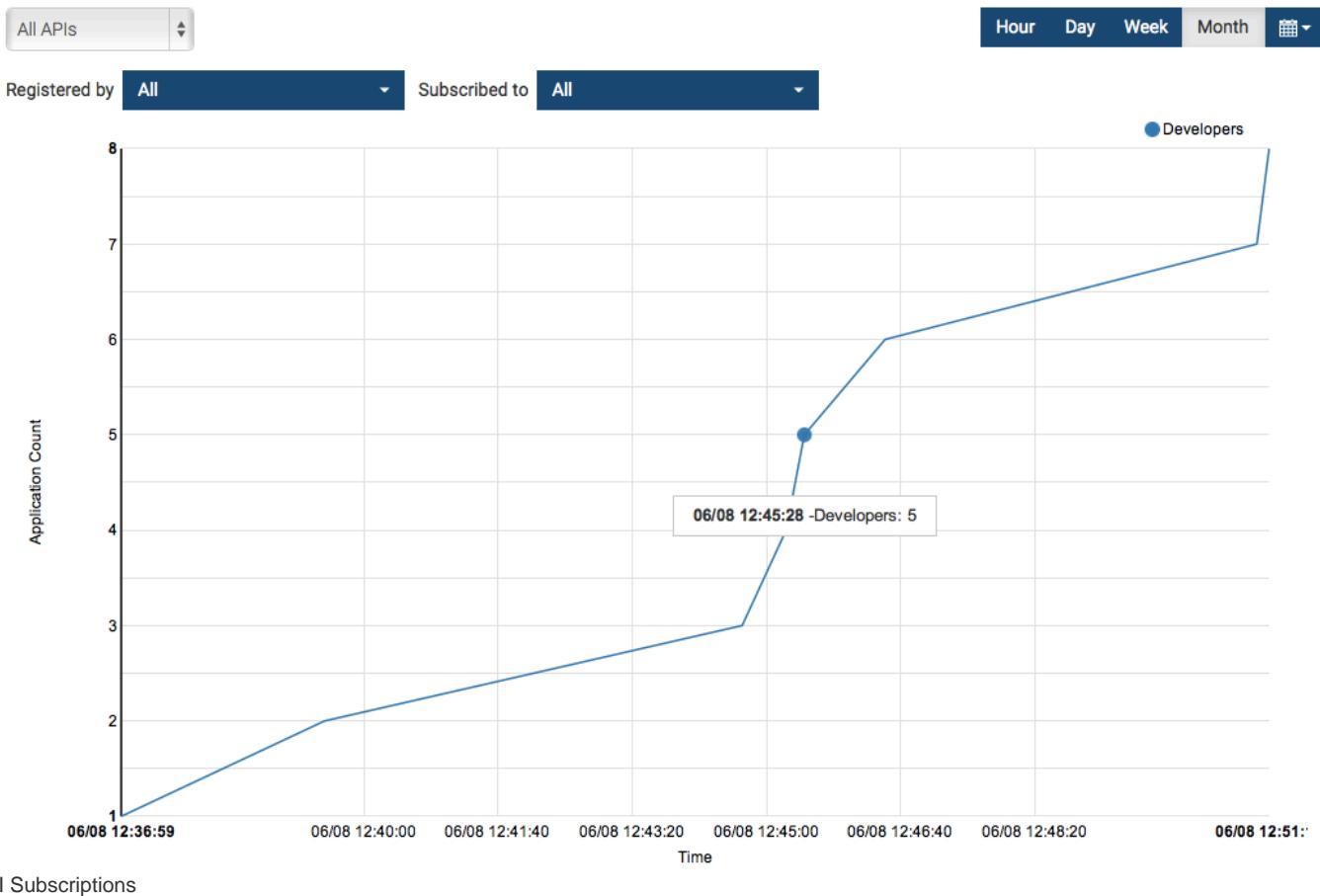
App Throttled Requests

The successful request count and throttled request count of each API invoked by each application.

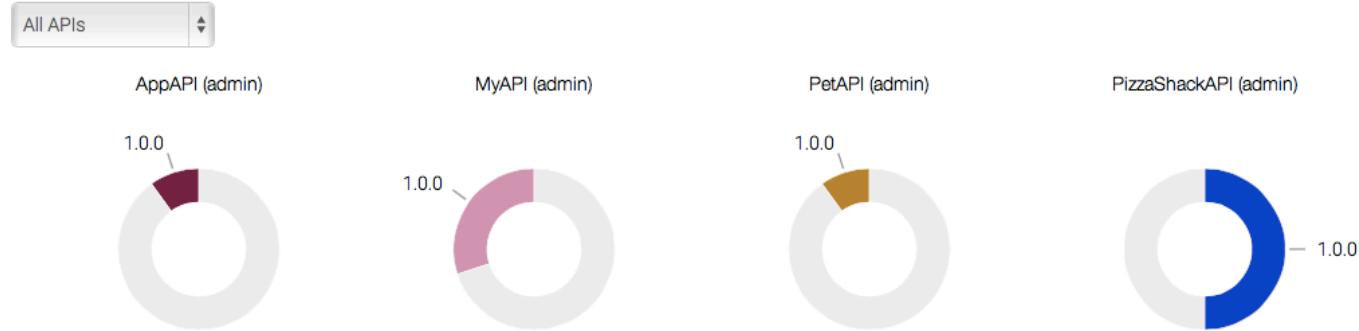
## Application Throttled Out Requests



## App Registrations



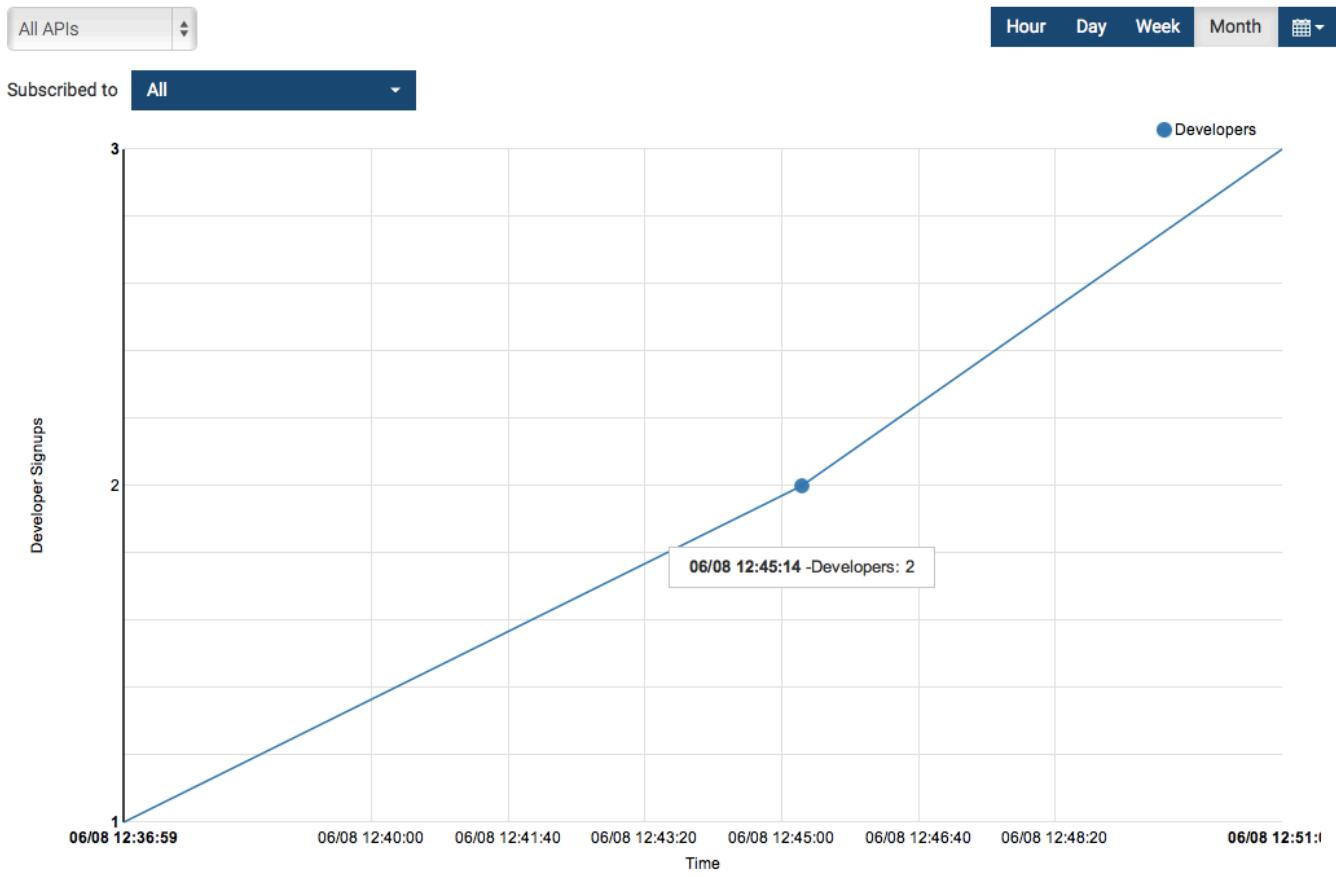
## Overall API Subscriptions (Across All Versions)



### Developer Signups Over Time

The number of developers who signed up to the API Store over time.

## Developer Signups



Subscriptions Created Over Time

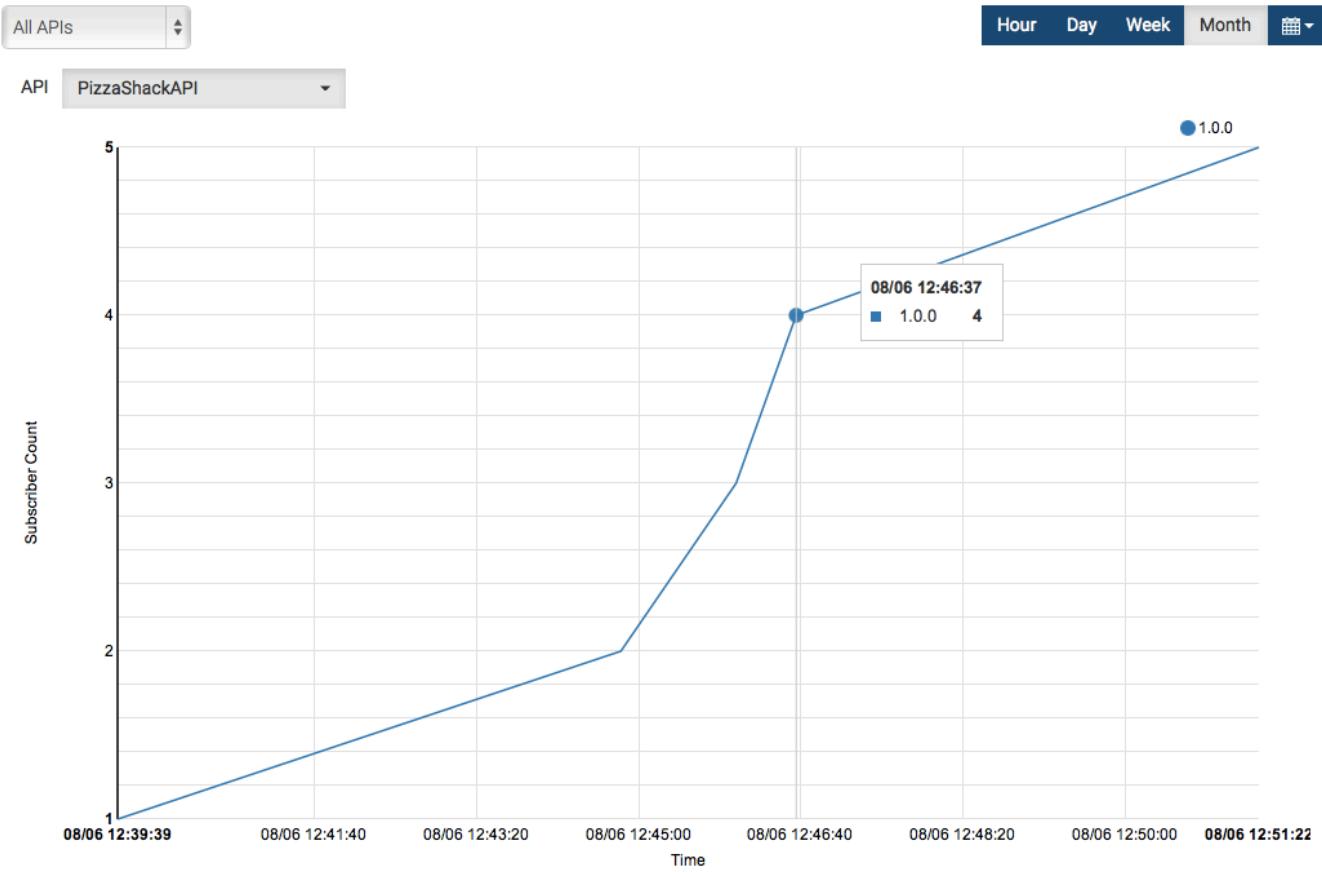
The number of subscriptions created for an API over a given period.

First, select the API for which you wish to view subscriptions.

### API Subscription Over Time

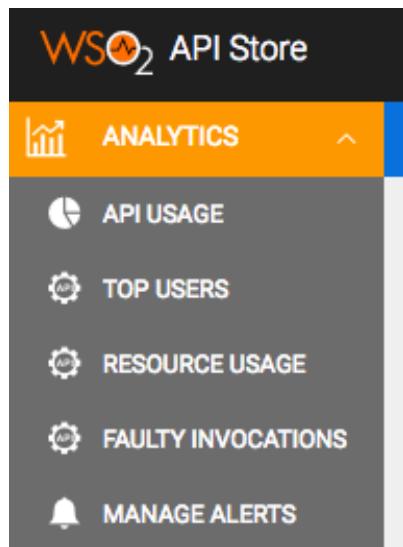
All APIs	▼	Hour Day Week Month	2012-02-01 00:00:00 to 2017-12-21 17:38:15 ▾
----------	---	---------------------	----------------------------------------------

## API Subscription per app



## API Store statistics

Log in to the API Store. You can self-subscribe to the store. Next, click the **Statistics** menu.



Given below are the statistical dashboards that are available:

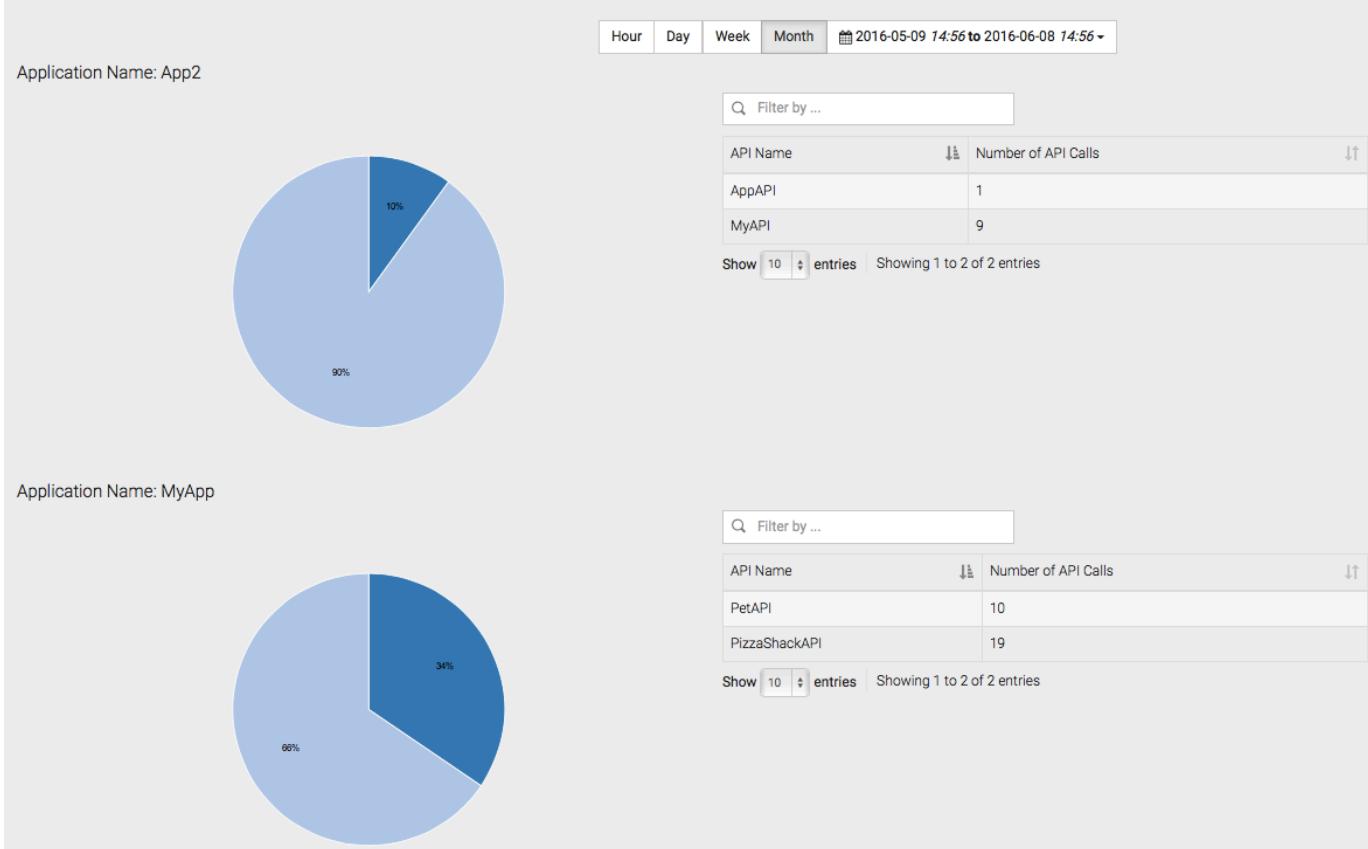
- **API Usage:** The usage of the API per application.
- **Top Users:** Users who make the most API invocations per application and the number of registered users per application.
- **Resource Usage:** The usage of an API and from which resource path per application.

- **Faulty Invocations:** The 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.

Several examples of usage and performance statistics are given below:  
API Usage per Application

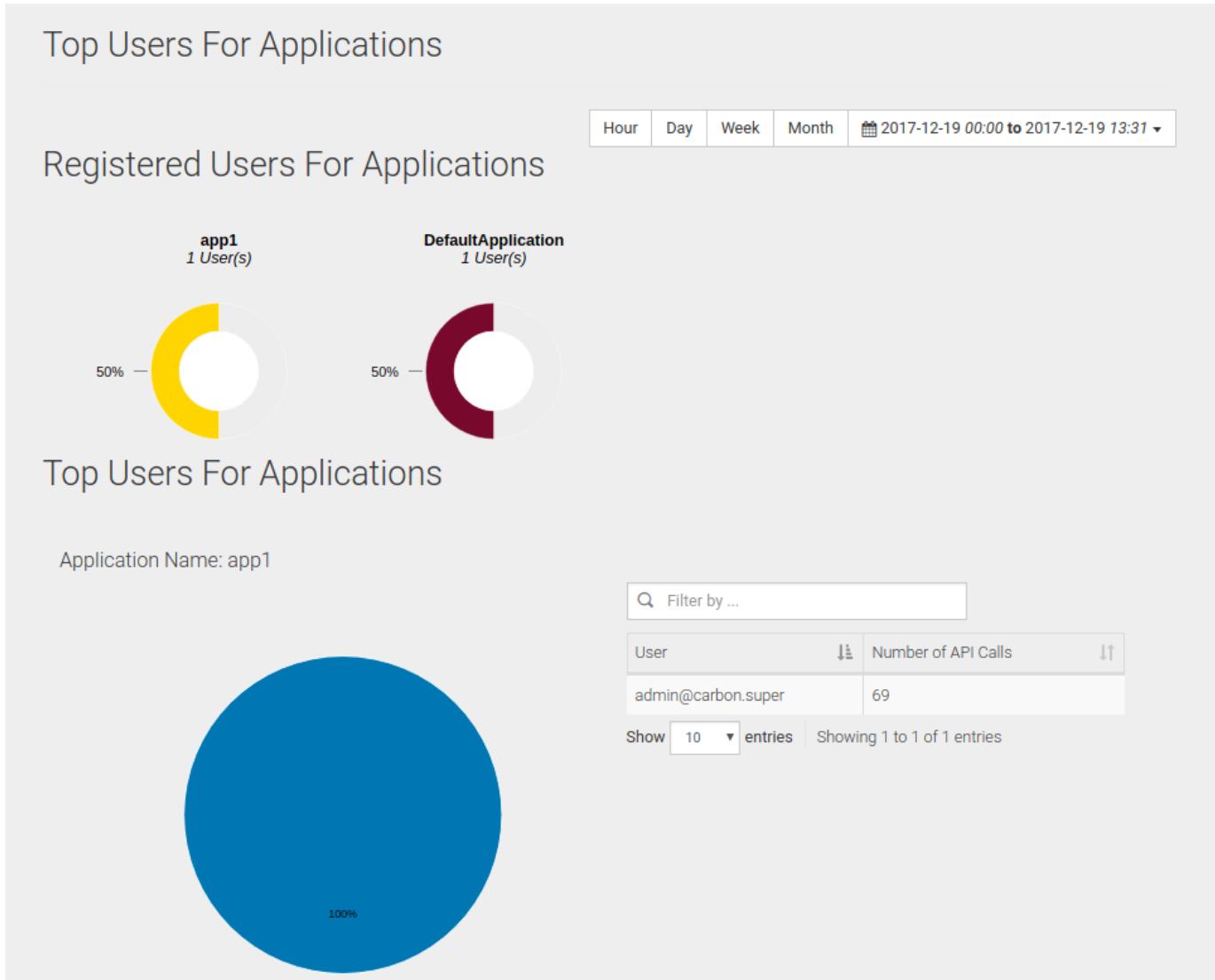
The number of invocations made for each API by each application.

### API Usage per Application



### Top Users per Application

The users who made the largest number of API calls per application.



The statistics for the registered users for applications takes the number of users shared across each application in to consideration. To enable application sharing among users within the same organization, see [Sharing Applications and Subscriptions](#).

Only users who have generated access tokens using the password grant type are considered in these statistics.

### Resource Usage per Application

The usage of resources of the APIs by each application.

## API Usage from Resource Path

Application Name	API Name	Resource Usage
App2	AppAPI	/* (POST)
App2	MyAPI	/hello (GET)
App2	MyAPI	/hello (POST)
MyApp	PetAPI	/hello (GET)
MyApp	PetAPI	/hello (POST)
MyApp	PetAPI	/hello (PUT)
MyApp	PizzaShackAPI	/menu (GET)
MyApp	PizzaShackAPI	/order (POST)

Show 10 entries Showing 1 to 8 of 8 entries 1

### Faulty Invocations per Application

The total number of invocations made by each application that are unsuccessful (faulty).

## Faulty Invocations per Application

Application Name	API Name	Faulty API Invocation Count
MyApp	PetAPI	10
Show 10 entries Showing 1 to 1 of 1 entries <span style="float: right;">1</span>		

## Admin Portal Statistics

Log in to the Admin Portal (<https://localhost:9443/admin>). API Availability is the only statistical view that exists in the Admin Portal. Admin users can view API Availability statistics by navigating to **ANALYTICS > API AVAILABILITY**.

Availability of APIs

The status of the APIs (all API versions) represented in a tabular view.

Api Version	Status
admin-PizzaShackAPI(v1.0.0)	Available
admin-GeoAPIv1.0.0	Available
Show 10 entries Showing 1 to 2 of 2 entries <span style="float: right;">1</span>	

<b>Status</b>	<p>This indicates the status of the API. There are two possible values; <b>Available</b> and <b>Limited</b>.</p> <p><b>Available</b> - This status indicates that the API has traffic with normal successful invocations. By default, if an API receives successful invocations for at least one out of five invocations within 30000 milliseconds, the status of the API becomes <b>Available</b>.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"><p>Note that only the APIs that have traffic are represented in this tabular representation.</p></div> <p><b>Limited</b> - If an API receives an alert due to one of the reasons indicated in <a href="#">Availability of APIs (health monitoring)</a>, the API status changes to <b>Limited</b>.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"><p>For more information on how to view the generated alerts, see <a href="#">Viewing Alerts</a>.</p></div>
---------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The availability of APIs statistics is directly related to the [Availability of APIs \(health monitoring\)](#) alert type. You can edit the default configurations of the numbers set as parameters to customize generating alerts by navigating to **SETTINGS > ANALYTICS** and by going to the **Edit** view of **HealthAvailabilityPerMinAlert** as shown below.

# Edit Configurations

Template Name:	HealthAvailabilityPerMinAlert	▼
Monitors the API Availability		
Configuration Name	HealthAvailabilityPerMin	
Description	Monitoring API health	
<b>Parameter Configurations</b>		
Number of continuous responses	5	
Number of responses that should fail to pass the lower percentile		
Number of continuous response time failures	5	
number of minutes that responses should fail to pass the lower percentile		
Time interval(in milliseconds) for API availability status change	300000	
Time duration taken to recheck and change the availability of an API		
Cache Time-out	300	
Cache time-out value in seconds		
Severity Level	2	
Severity level of the alert:(1:severe,2:moderate,3:mild)		

**Save Configuration**

The parameter configurations of this alert type are given below.

<b>Number of continuous responses</b>	Considering the aspect of generating the alerts, number of responses that should fail to pass the lower percentile. The default value is 5 responses.
<b>Number of continuous response time failures</b>	The number of minutes that responses should fail to pass the lower percentile for the alert to be generated. The default value is 5 minutes.
<b>Time interval</b>	The time duration taken to recheck and change the availability of an API. The availability of the API is rechecked after this time interval and the new status is retrieved if the behaviour changes. The default value is 300 seconds (300000 milliseconds).
<b>Cache time-out</b>	The cache time-out value in seconds.
<b>Severity level</b>	The severity level of the alert, which you can specify as 1, 2 or 3 where 1 is severe, 2 is moderate and 3 is mild.

## Using Geolocation Based Statistics

Geolocation based statistics are used to carry out detailed monitoring of geographic locations. The following sections explain how WSO2 API Manager is integrated with WSO2 Analytics - APIM to create alerts based on

statistics generated for selected geographic locations.

- Configuring Geolocation Based Statistics
- Writing a Custom Geolocation Provider
- Updating Geo Location Data Set

### Configuring Geolocation Based Statistics

Follow the procedure below in order to configure WSO2 API Manager to receive geolocation based alerts.

Note that we are using MySQL in this documentation for configuring the GEO\_LOCATION\_DATA database.

1. Download the Geolocation data from [here](#).
2. Unzip the file you downloaded.

If you have Geo Location Dataset already dowloaded you can update that dataset by following [Updating Geo Location Data Set](#).

3. Create the GEO\_LOCATION\_DATA database by executing one of the scripts in the Geolocation Data/dbscripts directory. In this example, mysql.sql is executed.

This can be done using the [MySQL Workbench](#).

For detailed instructions to run the database script, see [MySQL Documentation - The Workbench Scripting Shell](#).

4. Restore data to the BLOCKS and LOCATION tables by importing data from BLOCKS.csv and LOCATION.csv in .Geolocation Data/data directory of the extracted zip using below commands.

- **Importing Geolocation Data/data/LOCATION.csv**

```
mysqlimport -u root -p --ignore-lines=2 --fields-terminated-by=,
--fields-optionally-enclosed-by=''' --local GEO_LOCATION_DATA
<path_to_folder_location>/GeolocationData/data/LOCATION.csv
```

- **Importing Geolocation Data/data/BLOCKS.csv**

```
mysqlimport -u root -p --ignore-lines=2 --fields-terminated-by=,
--fields-optionally-enclosed-by=''' --local GEO_LOCATION_DATA
<Extracted_location>/GeolocationData/data/BLOCKS.csv
```

For more information, see [MySQL Documentation - Data Export and Import](#).

5. Check whether your imported dataset is properly working using executing following query in MySQL Command Line.

```
SELECT loc.country_name,loc.subdivision_1_name FROM BLOCKS block , LOCATION loc
WHERE block.network_blocks = '<Network_part_of_ip4>' AND
<Long_value_of_publilc_IP> BETWEEN block.network AND block.broadcast AND
block.geoname_id=loc.geoname_id;
```

**Example query :**

```
SELECT loc.country_name,loc.subdivision_1_name FROM BLOCKS block , LOCATION loc
WHERE block.network_blocks = '221.192' AND 3720398641
```

```
BETWEEN block.network AND block.broadcast AND block.geoname_id=loc.geoname_id;
```

6. Download a JDBC provider depending on the database you are using (MySQL in this example), and copy it to the <APIM\_ANALYTICS\_HOME>/repository/components/lib directory.
7. Configure datasource in the <APIM\_ANALYTICS\_HOME>/repository/conf/datasources/geolocation-datasources.xml file as follows.

```

<datasources-configuration
 xmlns:svns="http://org.wso2.securevault/configuration">
 <datasources>
 <datasource>
 <name>GEO_LOCATION_DATA</name>
 <description>The datasource used for Geo location
 database</description>
 <jndiConfig>
 <name>jdbc/GEO_LOCATION_DATA</name>
 </jndiConfig>
 <definition type="RDBMS">
 <configuration>

 <url>jdbc:mysql://localhost:3306/GEO_LOCATION_DATA</url>
 <username>wso2carbon</username>
 <password>wso2carbon</password>

 <driverClassName>com.mysql.jdbc.Driver</driverClassName>
 <maxActive>50</maxActive>
 <maxWait>60000</maxWait>
 <testOnBorrow>true</testOnBorrow>
 <validationQuery>SELECT 1</validationQuery>
 <validationInterval>30000</validationInterval>
 <defaultAutoCommit>false</defaultAutoCommit>
 </configuration>
 </definition>
 </datasource>
 </datasources>
</datasources-configuration>

```

8. Log into the WSO2 API Manager Admin Portal using the [https://localhost:<SERVER\\_PORT>/admin](https://localhost:<SERVER_PORT>/admin) URL.
9. Under **Settings => Analytics**, click **Configure Alerts** to open the **Alert Configurations** page.

## Alert Configurations

Scenario Type	Description	Status	Actions
RequestSummarizer	Responsible for summarizing incoming traffic	Active	Deactivate
FrequentTierLimitHitting	Frequent hitting of tier limits	Active	Deactivate
AbnormalRequestCountDetection	Detects abnormal request count	Active	Deactivate
AbnormalResponseAndBackendTimeDetection	Detects abnormal backend time and response time	Active	Deactivate
RequestPatternChangeDetection	Detection of request pattern changes	Active	Deactivate
MarkovStateClassifier	Clasifier configurations for Request Patterns	Active	Deactivate
UnusualIPAccessTemplate	Detecting accesses from new or rarely used hostnames	Active	Deactivate
HealthAvailabilityPerMinAlert	Monitoring API health	Active	Deactivate
ConfigureAccessToken	Configure access token to analyze data	Active	Deactivate
AbnormalTierUsageAlert	Abnormal tier usage alerts	Active	Deactivate
RequestPerApi	Request per API percentile values	Active	Deactivate
RequestStatGenerator	To generate request statistics	Active	Deactivate
ResponsePerApiStatGenerator	Response per API percentile values	Active	Deactivate
ResponseTime	Response time percentile values	Active	Deactivate
ResponseStatGenerator	Response percentile values	Active	Deactivate

## GEO Location configurations

Scenario Type	Description	Status	Actions
APIM_GEO_LOCATION_STATS	To generate Geo locations-based statistics	Inactive	

10. In the **Geo Location Configurations** section, click **Activate** for APIM\_GEO\_LOCATION\_STATS. This opens the **Edit Configuration** page with the default configuration for geolocation statistics as shown below.

### Edit Configurations

Template Name:	<input type="text" value="APIM_GEO_LOCATION_STATS"/>
	To generate Geo locations-based statistics
Configuration Name	<input type="text" value="APIM_GEO_LOCATION_STATS"/>
Description	<input type="text" value="To generate Geo locations-based statistics"/>
<input type="button" value="Save Configuration"/>	

Modify parameter values as required.

11. Click **Save Configuration**.

The script that summarizes geolocation based statistics runs every day at 2300 hours. Due to this, statistics may not be visible immediately when you enable geolocation based statistics. To update these statistics immediately, follow the steps below.

1. Log in to the WSO2 APIM Analytics Management Console (<https://<hostname>:9444/carbon> assuming the port offset is 1).
2. Click the **Main** tab.
3. In the **Batch Analytics** section, click **Scripts**.
4. Click **Execute** for the **APIMAnalytics-APIM\_GEO\_LOCATION\_STATS-APIM\_GEO\_LOCATION\_STATS-batch1** script.

## Writing a Custom Geolocation Provider

Each Geolocation Resolver implementation in WSO2 Analytics is inherited from the `org.wso2.carbon.analytics.apim.spark.geolocation.api.LocationResolver` abstract class has the following methods.

- **getLocation**: This contains the Geolocation Resolving implementation. Only this method needs to be implemented for this scenario.
- **init**: This contains the Geolocation Resolver implementation

To customize the default Geolocation Resolver extension, you should override the `getLocation()` method with your custom implementation. For example, the following class is a sample implementation of the Geolocation Resolving service It returns the `Location` according to the IP of the Geolocation API that provided through the configuration on each IP resolving through the UDF.

```
/*
 * Copyright (c) 2016, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
 *
 * WSO2 Inc. licenses this file to you under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except
 * in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
package com.wso2.carbon.analytics.apim.spark.geolocation.impl;

import org.wso2.carbon.analytics.apim.spark.geolocation.api.Location;
import org.wso2.carbon.analytics.apim.spark.geolocation.api.LocationResolver;
import org.wso2.carbon.analytics.apim.spark.geolocation.exception.GeoLocationResolverException;

public class CustomLocationResolver extends LocationResolver {
 private String restUrl;
 private String username;
 private String password;
```

```

@Override
public void init() throws GeoLocationResolverException {
}

public Location getLocation(String ip) throws
GeoLocationResolverException {
 return null;
}

public String getRestUrl() {
 return restUrl;
}

public void setRestUrl(String restUrl) {
 this.restUrl = restUrl;
}

public String getUsername() {
 return username;
}

public void setUsername(String username) {
 this.username = username;
}

public String getPassword() {
 return password;
}

public void setPassword(String password) {
 this.password = password;
}
}

```

Compile as a jar file and copy into  
<Product-home>/repository/components/lib folder.  
Configure the above class in geolocation.xml under  
<Product-home>/repository/conf/etc as following way.

```

<GeoLocation>
 <Implementation
class="com.wso2.carbon.analytics.apim.spark.geolocation.impl.CustomLocatio
nResolver">
 <Property
name="restUrl">http://localhost:80080/geolocation/service</Property>
 <Property name="username">admin</Property>
 <Property name="password">admin</Property>
 </Implementation>
 <Cache>
 <enabled>true</enabled>

```

```
<IpResolveCacheCount>10000</IpResolveCacheCount>
</Cache>
</GeoLocation>
```

## Updating Geo Location Data Set

Follow the procedure below in order to update your existing Geo Location dataset to use in [Configuring Geolocation Based Statistics](#).

1. Download the latest CSV file from [here](#).
2. Download the geoip-2-csv-converter from <https://github.com/maxmind/geoip2-csv-converter/releases> according to your operating system.

Prepare the database entries

1. Unzip the latest CSV file and the geoip-2-csv-converter you have downloaded in the steps above.
2. Run `update-geolocation-data.sh` file using the command below.

```
sh update-geolocation-data.sh
```

- Enter the path to the extracted GeoLite2-City-Blocks-IPv4 directory which you downloaded first, as the response for **Enter path to GeoLite2-City-Blocks-IPv4 directory:**  
E.g : /<PATH\_TO>/GeoLite2-City-CSV\_20171107
- Enter the path to geoip2-csv-converter directory as the response for **Enter path to geoip2-csv-converter home directory:**  
E.g : /<PATH\_TO>/geoip2-csv-converter-v1.0.0
- After executing the script, you will find the `final.csv` file inside your current directory.

```
** get first column form original
** change column name to 'network_cidr'
** Extract ip address data
** change column name to 'network_blocks'
** extract entries from original
cut:
/home/chamalee/xxx/geoip2-csv-converter-v1.0.0/GeoLite2-City-Blo
cks-IPv4-converted.csv: No such file or directory
** change column name to 'network'
** change column name to 'broadcast'
** merge csv files
```

3. Shut down both APIM and APIM-Analytics servers if you are running them already.
4. Truncate BLOCKS and LOCATION tables from the GEO\_LOCATION\_DATA database.

Alternatively you can drop the tables in the GEO\_LOCATION\_DATA database and create new tables.

## Importing Data

1. Import the created `final.csv` file into BLOCKS table. Use the command given below.

```

load data local infile '[PATH_TO_FINAL.CSV]/final.csv' into table
BLOCKS
 fields terminated by ','
 enclosed by ""
 lines terminated by '\n'
 (network_cidr, network, broadcast, geoname_id,
 registered_country_geoname_id, represented_country_geoname_id,
 is_anonymous_proxy, is_satellite_provider, postal_code, latitude,
 longitude, network_blocks);

```

- Import the GeoLite2-City-Locations-en.csv file located inside the extracted geoip-2-csv-converter directory (e.g geoip-2-csv-converterGeoLite2-City-CSV\_2017110) into LOCATION table. Use the command given below.

```

load data local infile
'[PATH_TO_GeoLite2-City-Locations-en]/GeoLite2-City-Locations-en.csv'
into table LOCATION
 fields terminated by ','
 enclosed by ""
 lines terminated by '\n'
 (geoname_id, locale_code, continent_code, continent_name,
 country_iso_code, country_name, subdivision_1_iso_code,
 subdivision_1_name, subdivision_2_iso_code, subdivision_2_name,
 city_name, metro_code, time_zone);

```

- Restart WSO2 API Manager and WSO2 APIM-Analytics servers.

You have now updated the Geo Location Data Set.

## Managing Alerts with Real-time Analytics

WSO2 API Manager uses WSO2 Analytics to create alerts for different purposes. Common scenarios that require alerts include the sudden failure of one or more APIs, a sudden increase in the response time of one or more APIs and the change in the pattern of API resource access.

The following topics explain the different alert types that can be created and how to configure WSO2 API Manager to use them.

- [Alert Types](#)
- [Configuring Alerts](#)
- [Subscribing for Alerts](#)
- [Viewing Alerts](#)

### Alert Types

WSO2 APIM currently supports the following alert types.

- [Abnormal response time](#)
- [Abnormal backend time](#)
- [Abnormal request counts](#)
- [Abnormal resource access pattern](#)

- Unseen source IP address
- Frequent tier limit hitting (tier crossing)
- Abnormal API usage
- Availability of APIs (health monitoring)

### Abnormal response time

<b>Reason for triggering</b>	If there is a sudden increase in the response time of a specific API resource.
<b>Indication</b>	Slow WSO2 API Manager runtime, or slow backend.
<b>Description</b>	If the response time of a particular API resource (e.g., GET /API1/1.0/user/1) of a tenant, lies outside the Xth percentile value, an alert is sent. Default percentile value is 95%. Here, it is assumed that the response time of an API resource follows a normal distribution. Percentile value gets calculated daily by default.

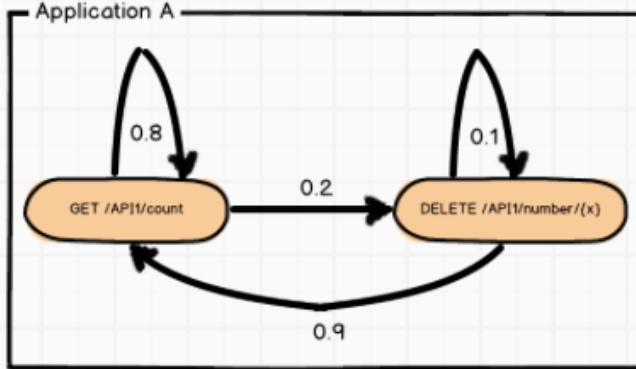
### Abnormal backend time

<b>Reason for triggering</b>	If there is a sudden increase in the backend time corresponding to a particular API resource.
<b>Indication</b>	Slow backend
<b>Description</b>	An alert is sent if the backend time of a particular API resource (e.g., GET /calc/1.0/numbers) of a tenant lies outside the Xth percentile value. Default percentile value is 95%. Here, it is assumed that the corresponding backend time of an API resource follows a normal distribution. The percentile value gets calculated daily by default.

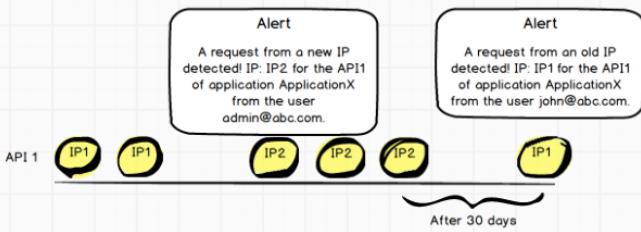
### Abnormal request counts

<b>Reason for triggering</b>	If there is a sudden spike or a drop in the request count within a period of one minute by default for a particular API resource.
<b>Indication</b>	These alerts can be considered indications of high traffic, suspicious acts or the malfunction of client applications etc.
<b>Description</b>	An alert is sent if the number of requests received by a particular API resource (e.g., GET /calc/1.0/numbers) of a tenant of a particular application within the last minute lies outside the Xth and Yth percentile values. The default percentile values are 95% and 5%. Here, it is assumed that the request counts received by an API resource follows a normal distribution. Percentile value (a per minute average request count value) gets calculated daily by default.

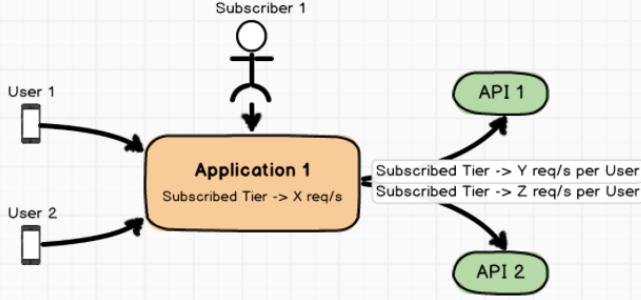
## Abnormal resource access pattern

<b>Reason for triggering</b>	If there is a change in the resource access pattern of a user who uses a particular application.
<b>Indication</b>	These alerts can be considered as indications of suspicious activities done by one or more users in your application.
<b>Description</b>	<p>A Markov Chain model is built for each application to learn its resource access pattern. For the purpose of learning the resource access patterns, no alerts are sent during the first 500 (default) requests. After learning the normal pattern of a specific application, WSO2 Analytics performs a real time check on a transition done by a specific user, and sends an alert if it is identified as an abnormal transition. For a transition to be considered valid, it has to occur within 60 minutes by default, and it should be by the same user.</p>  <pre> graph LR     subgraph Application_A [Application A]         GET[GET /API1/count] -- "0.2" --&gt; DELETE[DELETE /API1/number/{x}]         GET -- "0.8" --&gt; GET         DELETE -- "0.1" --&gt; DELETE         DELETE -- "0.9" --&gt; GET     end   </pre> <p>The above diagram depicts an example where a Markov Chain model is created during the learning curve of the system. Two states are recorded against Application A and the arrows show the directions of the transitions. Each arrow carries a probability value that stands for the probability of a specific transition taking place. Assume that the following two consecutive events are received by the application from user <a href="#">john@abc.com</a>.</p> <ol style="list-style-type: none"> <li>1. DELETE /API1/number/1</li> <li>2. DELETE /API1/number/3</li> </ol> <p>The above transition has happened from the DELETE /API1/number/{x} state to itself. According to the Markov chain model learnt by the system, the probability of this transition occurring is very low. Therefore, an alert is sent.</p>

## Unseen source IP address

<b>Reason for Triggering</b>	If there is either a change in the request source IP for a specific API of an application, or if the request is from an IP used before 30 days (default).
<b>Indication</b>	These alerts can be considered as indications of suspicious activities carried out by a user over an API of an application.
<b>Description</b>	 <p>The first 500 requests are used only for learning purposes by default and therefore, no alerts are sent during that time. However, the learning would continue even after the first 500 requests. This means, even if you receive continuous requests from the newly detected IP2, you are alerted only once.</p>

### Frequent tier limit hitting (tier crossing)

<b>Reason for Triggering</b>	<p>This alert is triggered in the following scenarios.</p> <ul style="list-style-type: none"> <li>• If a particular application is throttled for reaching a subscribed tier limit more than the specified number of times during a defined period (10 times within a day by default).</li> <li>• If a particular user of an application is throttled for reaching a subscribed tier limit of a specific API more than the specified number of times during a defined period (10 times within a day by default).</li> </ul> 
<b>Indication</b>	These alerts indicate that you need to subscribe to a higher tier.

### Abnormal API usage

<b>Reason for Triggering</b>	If there is a drastic reduction in API usage by a specific user for a given API.
<b>Indication</b>	These alerts indicate the failure of the application that is using the altered API.
<b>Description</b>	For the purpose of detecting abnormal API usage, it is assumed that API requests are normally distributed. The mean and the variance are the two main properties of a normal distribution. These are calculated per user for each application. Instead of using all the past requests, you can define a time period based on which the mean and variance should be calculated. The default time period is 30 days.

### Availability of APIs (health monitoring)

These alerts are triggered for the reasons specified in the tables below.

<b>Reason for Triggering</b>	The response time of an API is greater than the upper percentile value specified for the same (which is 95 by default). This should occur continuously for a specified number of times (5 times by default).
<b>Indication</b>	The response time is too high.
<b>Reason for Triggering</b>	The request count of an API per minute is less than the lower percentile value specified for the same (which is 5 by default). This should occur 5 times (i.e. 5 minutes) continuously in order to trigger the error.
<b>Indication</b>	The request count per minute is normal, but the response count per minute is low.
<b>Reason for Triggering</b>	The response status code is greater than or equal to 500, but less than 600. This should occur continuously for a specified number of times ( 5 by default) in order to trigger an alert.
<b>Indication</b>	A server side error has occurred.

Refer [Viewing Availability Of APIs](#) for more information on API status changing over Availability of APIs.

### Configuring Alerts

All alerts are configured globally by system administrators. The steps below explain how to change the default parameter values for alerts.

Before you begin, make sure you [configure API Manager Analytics](#) so that you can see the alert configurations shown in the screenshots below in the admin portal.

1. Log into the WSO2 API Manager Admin Portal using the `https://localhost:<SERVER_PORT>/admin` URL.
2. Click **Settings** to expand that section and then click **Analytics**. This opens the **Alert Configurations** page as

shown below.

#### Alert Configurations

Scenario Type	Description	Status	Actions
RequestSummarizer	Responsible for summarizing incoming traffic.	Active	Deactivate
FrequentTierLimitHitting	Detects frequent hitting of the tier limit	Active	Deactivate
AbnormalRequestCountDetection	Detects abnormal request count	Active	Deactivate
AbnormalResponseAndBackendTimeDetection	Detects abnormal backend time and response time	Active	Deactivate
RequestPatternChangeDetection	Change the configuration of request pattern detection	Active	Deactivate
MarkovStateClassifier	Classifier configurations for Request Patterns	Active	Deactivate
UnusualIPAccessTemplate	To detect accesses from new or rarely used hostnames	Active	Deactivate
HealthAvailabilityPerMinAlert	To monitor the API health	Active	Deactivate
ConfigureAccessToken	Configure access token to analyse data	Inactive	
AbnormalTierUsageAlert	This is the CEP stream for Abnormal Tier Usage	Active	Deactivate
RequestPerApi	To create RequestPerApi percentile values	Active	Deactivate
RequestStatGenerator	To generate request stats	Active	Deactivate
ResponsePerApiStatGenerator	To create Response per api percentile values	Active	Deactivate
ResponseTime	To create response time percentile values	Inactive	
ResponseStatGenerator	To create response percentile values	Inactive	

#### GEO Location configurations

Scenario Type	Description	Status	Actions
APIM_GEO_LOCATION_STATS	To generate Geo locations-based statistics	Inactive	

The **Status** column indicates whether an alert is active or inactive.

- If you want to activate a currently inactive alert, click **Activate** in the relevant row. As a result, the **Edit Configuration** page opens where you can configure the alert. e.g., if you click **Activate** for the ConfigureAccessToken alert, the following page opens.

## Edit Configurations

Template Name:	<input type="text" value="ConfigureAccessToken"/>
	Configure access token to analyse data
Configuration Name	<input type="text" value="ConfigureAccessToken"/>
Description	<input type="text" value="Configure access token to analyse data"/>

#### Parameter Configurations

Lower percentile	<input type="text" value="0.10"/>
	Lower percentile (value between 0 and 1) to calculate minimum time difference
Upper percentile	<input type="text" value="0.95"/>
	Upper percentile (value between 0 and 1) to calculate maximum time difference

**Save Configuration**

- If you want to deactivate a currently active alert, click **Deactivate** in the relevant row and click **Yes** in the message box that appears to confirm whether you want to deactivate the alert.
- If you want to edit an alert, click **Edit** for the required alert to open the **Edit Configurations** page.

The parameters you have to configure depends on the alert type you select for this parameter. Click on the relevant tab below to view descriptions for alert-specific parameters.

[Request Summarizer](#)  
[Frequent Tier Limit Hitting](#)  
[Abnormal Request Count Detection](#)

[Abnormal Response and Backend Time Detection](#)  
[Request Pattern Change Detection](#)  
[Markov State Classifier](#)

[Unusual IP Access Template Health Availability](#)  
[Per Minute Alert Configuration](#)  
[Access Token Abnormal Tier Usage Alert](#)

[Request Per API Request Stat Generator](#)  
[Response Per API Stat Generator](#)  
[Time Response Stat Generator](#)

Parameter Name	Description	Default Value
Time Interval	The time intervals at which the request summarization of requests takes place. This value is expressed in seconds. e.g., If the time interval is 20, 20 seconds should elapse after the last request summarization before this operation is repeated.	60

Parameter Name	Description	Default Value
Time Interval	The time duration for which the number of tier crossings should be calculated.	1 day
Alert Suppression Period in Minutes	The number of minutes to wait after the alert is sent in order to send it again.	10
No of Tier Crossings	The average number of times a user gets throttled out of the application for reaching its subscribed tier limit. This is calculated for the time interval specified in the <b>Time Interval</b> parameter.	10
Severity Level	The severity level assigned to the alert. The available levels are as follows. <ul style="list-style-type: none"> <li>• 1: Severe</li> <li>• 2: Moderate</li> <li>• 3: Mild</li> </ul>	0

Parameter Name	Description	Default Value
Alert Suppression Period in Minutes	The number of minutes to wait after the alert is sent in order to send it again.	10
Severity Level	The severity level assigned to the alert. The available levels are as follows. <ul style="list-style-type: none"> <li>• 1: Severe</li> <li>• 2: Moderate</li> <li>• 3: Mild</li> </ul>	0

Parameter Name	Description	Default Value

<b>Severity Level of Abnormal Response Time</b>	The severity level assigned to an alert generated for an abnormal response time. The possible levels are as follows. <ul style="list-style-type: none"><li>• 1: Severe</li><li>• 2: Moderate</li><li>• 3: Mild</li></ul>	2
<b>Severity Level of Abnormal Backend-Response Time</b>	The severity level assigned to an alert generated for an abnormal response time but he backend. The possible levels are as follows. <ul style="list-style-type: none"><li>• 1: Severe</li><li>• 2: Moderate</li><li>• 3: Mild</li></ul>	0

Parameter Name	Description	Default Value
<b>Regular API Transitions</b>	The number of API transitions to be considered when calculating the probability value of the current transition. If a value greater than 1 is specified for this parameter, the last API transitions are also considered to determine whether the current transition is an abnormal transition. e.g., If 3 is specified for this parameter, the probability value is the mean of the transition value of the previous two transitions and the current transition.	1
<b>Request Count</b>	The number of requests required from a specific consumer key in order to learn the pattern within the application. Alerts are not generated until the request count reaches this number .	500
<b>Probability Threshold</b>	If an API transition has a probability value equal to the value calculated as follows, it is considered a suspicious transition and an alert is sent. $1 - \text{Probability Threshold}$	0.95
<b>Alert Suppression Period</b>	Once an alert is generated for a request pattern change, a similar alert is not sent again until the time interval specified for this parameter has passed. This value is expressed in milliseconds.	$30 * 60 * 1000$
<b>Severity Level</b>	The severity level assigned to the alert. The available levels are as follows. <ul style="list-style-type: none"><li>• 1: Severe</li><li>• 2: Moderate</li><li>• 3: Mild</li></ul>	1

Parameter Name	Description	Default Value
<b>Transition Period</b>	The time interval within which the transition by the user should occur. If the time specified for this parameter is exceeded, the transition is considered invalid and an alert is sent.	60 min

Parameter Name	Description	Default Value

<b>Maximum Days Between Last Access</b>	The maximum number of days that should elapse between two occurrences IP access. Once the number of days specified for this parameter has passed since the IP was last accessed, an alert is sent.	30
<b>Severity Level</b>	The severity level assigned to the alert. The available levels are as follows. <ul style="list-style-type: none"> <li>• 1: Severe</li> <li>• 2: Moderate</li> <li>• 3: Mild</li> </ul>	

Parameter Name	Description	Default Value
<b>Number of Continuous Responses</b>	The number of responses that should fail per minute after the lower percentile value is reached in order to generate an alert. The lower percentile is calculated by the Spark script which runs in the background when WSO2 Analytics - APIM run. This value varies depending on the available data.	5
<b>Number of Continuous Response Time Fails</b>	The time interval in minutes during which responses should continuously fail after the lower percentile value is reached in order to generate an alert. The lower percentile is calculated by the Spark script which runs in the background when WSO2 Analytics - APIM run. This value varies depending on the available data.	5
<b>Time interval(in milliseconds) for API availability status change</b>	The time interval during which the availability status of the API should be rechecked and updated.	300000
<b>Severity Level</b>	The severity level assigned to the alert. The available levels are as follows. <ul style="list-style-type: none"> <li>• 1: Severe</li> <li>• 2: Moderate</li> <li>• 3: Mild</li> </ul>	2

Parameter Name	Description	Default Value
<b>Lower Percentile</b>	The lower percentile value. If the time interval between two consecutive renewals of an access token is less than this value, the subsequent renewal is identified as an abnormal renewal, and an alert is generated.	0.10
<b>Upper Percentile</b>	The upper percentile value. If the time interval between two consecutive renewals of an access token is more than this value, the subsequent renewal is identified as an abnormal renewal, and an alert is generated.	0.95

Parameter Name	Description	Default Value
<b>Percentile</b>	The percentile value to calculate the threshold.	0 .05
<b>Alert Start Date</b>	The starting date from which the alert should be activated.	The current date is default.
<b>Days Considered for Percentile Calculation</b>	The number of days before the Alert Start Date that should be considered for the percentile calculation. e.g., If the Alert Start Date is 01/12/2017, and the Days Considered for Percentile Calculation is 30, the time period 01/11/2017 - 30/11/2017 is considered when calculating the percentile.	30
<b>Days Considered for Abnormal Tier Availability Calculation</b>	The number of days before the Alert Start Date that should be considered for the abnormal tier availability calculation. e.g., If the Alert Start Date is 01/12/2017, and the Days Considered for Abnormal Tier Availability Calculation is 30, the time period 01/11/2017 - 30/11/2017 is considered when calculating the abnormal tier availability.	5
<b>Severity Level</b>	The severity level assigned to the alert. The available levels are as follows. <ul style="list-style-type: none"> <li>• 1: Severe</li> <li>• 2: Moderate</li> <li>• 3: Mild</li> </ul>	2

Parameter Name	Description	Default Value
<b>Lower Percentile</b>	If the number of requests received by an application within a minute is less than this percentile value, it is identified as an abnormal request count and an alert is sent.	0 .05

Parameter Name	Description	Default Value
<b>Upper Percentile</b>	The upper percentile value used to calculate the number of requests per minute.	0 .98
<b>Lower Percentile</b>	The lower percentile value used to calculate the number of requests per minute.	0 .05

Parameter Name	Description	Default Value

<b>Lower Percentile</b>	The lower percentile value used to calculate the number of requests per minute for each API.	0 . 05
-------------------------	----------------------------------------------------------------------------------------------	--------

Parameter Name	Description	Default Value
<b>Upper Percentile</b>	The upper percentile value to calculate the response time	0 . 95

Parameter Name	Description	Default Value
<b>Upper Percentile Response Time</b>	If the time duration taken by the API to respond is greater than this percentile value, it indicates that the API is slow to respond and an alert is sent.	0.95
<b>Upper Percentile Backend Time</b>	If the time duration taken by the backend to respond is greater than this percentile value, it indicates that the backend is slow to respond and an alert is sent.	0.95

6. Edit the email address, username, password and other relevant properties in the <APIM-ANALYTICS\_HOME>/repository/conf/output-event-adapters.xml file, to point the mail transport sender that is enabled by default in the product to a valid SMTP configuration as shown in the example below.

```
<adapterConfig type="email">
 <property key="mail.smtp.from">email-address</property>
 <property key="mail.smtp.user">user-name</property>
 <property key="mail.smtp.password">password</property>
 <property key="mail.smtp.host">smtp.gmail.com</property>
 <property key="mail.smtp.port">587</property>
 <property key="mail.smtp.starttls.enable">true</property>
 <property key="mail.smtp.auth">true</property>
 <!-- Thread Pool Related Properties -->
 <property key="maxThread">100</property>
 <property key="keepAliveTimeInMillis">20000</property>
 <property key="jobQueueSize">10000</property>
</adapterConfig>
```

In gmail [account security settings](#) you may have to enable the Allow less secure apps option in order to connect the account to WSO2 products.

## Subscribing for Alerts

You can subscribe to events as a system administrator or as a API publisher/subscriber. These users can subscribe to any of the alert types listed in the **Manage Alert Types** page specific to them. For more information about different types of alerts and their importance, see [Alert Types](#).

Click on the relevant tab to view the required procedure.

System Administrators APIM Publisher/Subscriber

A system administrator is allowed to select one or more alert types to subscribe for, as well as specify a list of email addresses to which the alerts should be sent. Follow the procedure below to carry out the tasks mentioned above using the Admin Portal of WSO2 API Manager.

1. Log into the WSO2 API Manager Admin Portal using the `https://localhost:<SERVER_PORT>/admin` URL.
2. In **ANALYTICS** menu, Click **MANAGE ALERT TYPES** to open the **Manage Alert Types** page.

### Manage Alerts

<input type="checkbox"/> <b>Abnormal Response Time</b>	This alert gets triggered if there is a sudden increase in the response time of a particular API resource. These alerts could be treated as an indication of a slow WSO2 API Manager runtime or a slow backend.
<input type="checkbox"/> <b>Abnormal Backend Time</b>	This alert should get triggered if there is a sudden increase in the backend time corresponding to a particular API resource. These alerts could be treated as an indication of a slow backend. In technical terms, if the backend time of a particular API resource (eg. GET /calc/1.0/numbers) of a tenant lies outside the Xth percentile value, we will send an alert out. Default percentile value is 95%. Here, we safely assume that the corresponding backend time of an API resource follows a normal distribution. Percentile value gets calculated daily by default.
<input type="checkbox"/> <b>Abnormal Request Count</b>	This alert gets triggered if there is a sudden spike or a drop in the request count within a period of one minute by default for a particular API resource. These alerts could be treated as an indication of a possible high traffic, suspicious activity, possible malfunction of the client application, etc.
<input type="checkbox"/> <b>Abnormal Resource Access</b>	This alert gets triggered if there is a change in the resource access pattern of a user of a particular Application. These alerts could be treated as an indication of a suspicious activity made by a user over your application.
<input type="checkbox"/> <b>Unseen Source IP Access</b>	This alert gets triggered if there is either a change in the request source IP for a particular API of an application or if the request is from an IP used before a time period of 30 days (default). These alerts could be treated as an indication of a suspicious activity made by a user over an API of an application.
<input type="checkbox"/> <b>Abnormal Access Token Renewal</b>	This alert gets triggered if there is a change in the pattern of renewing access tokens of an application by a user. These alerts could be treated as an indication of a stolen access token.
<input type="checkbox"/> <b>Tier Crossing</b>	This alert gets triggered if at least one of the two cases below are satisfied; if a particular application gets throttled out for hitting the subscribed tier limit of that application, more than 10 times (by default) within a day (by default) or if a particular user of an application, gets throttled out for hitting the subscribed tier limit of a particular API, more than 10 times (by default) within a day (by default)
<input type="checkbox"/> <b>Abnormal API Usage</b>	This alert gets triggered if there is a drastic reduction in API usage for a given API for a given user. These types of alerts should be treated as an indication of a failure of the application using the altered API.
<input type="checkbox"/> <b>Health Availability</b>	This alert gets triggered if at least one of the three cases below are satisfied; Response time of an API > Response time upper percentile of that particular API or Request count of an API per minute > Request count per minute lower percentile or Response status code $\geq 500$ (By Default) AND Response status code $< 600$ (By Default)

Email:

Type an Email and press Enter

3. Select the relevant check boxes based on the alert types to which you want to subscribe.
4. Under **Email list**, enter the list of email addresses that should receive alerts. The email addresses should be those of system administrators. Each email address can be separated with a comma or you can type Email address and press Enter.
5. Click **Save** to save the information.

An API Manager publisher/subscriber can enable/disable alert types based on the alerts that he/she wants to receive individually, as well as specify a list of email addresses to which the alerts should be sent.

1. Log into the API publisher with the username and password of a user with permission to publish using WSO2 API Manager.
2. In **ANALYTICS** menu, Click **MANAGE ALERT TYPES** to open the **Manage Alert Types** page.

## Manage Alert Types

<input type="checkbox"/> <b>Abnormal Response Time</b>	This alert type gets triggered if there is a sudden increase in the response time of a particular API resource. These alerts could be treated as an indication of a slow WSO2 API Manager runtime or a slow backend.
<input type="checkbox"/> <b>Abnormal Backend Time</b>	This alert type gets triggered if there is a sudden increase of the backend time corresponding to a particular API resource. These alerts could be treated as an indication of a slow backend. In technical terms, if the backend time of a particular API resource (eg: GET /calc/1.0/numbers) of a tenant lies outside the Xth percentile value, we will send an alert out. Default percentile value is 95%. Here, we safely assume that the corresponding backend time of an API resource follows a normal distribution. Percentile value gets calculated daily by default.
<input type="checkbox"/> <b>Abnormal API Usage</b>	This alert gets triggered if there is a drastic reduction in API usage for a given API for a given user. These types of alerts should be treated as an indication of a failure of the application using the altered API.
<input type="checkbox"/> <b>Health Availability</b>	This alert type gets triggered if at least one of the three cases below are satisfied: Response time of an API > Response time upper percentile of that particular API or Request Count of an API per minute > Request count per minute lower percentile or Response status code >= 500 (By Default) AND Response status code < 600 (By Default)

Email:   
Type an Email and press Enter

**Save** **Cancel**

3. Select the relevant check boxes based on the alert types to which you want to subscribe.
4. Under **Email list**, enter the list of email addresses that should receive alerts.
5. Click **Save** to save the information.

## Viewing Alerts

The following procedure to view alerts is relevant only for system administrators. System administrators, API publishers as well as API subscribers receive alerts via notification emails if they have subscribed to one or more alert type.

Follow the procedure below to view alerts that were generated for the APIs deployed in your WSO2 API Manager installation.

1. Access the WSO2 API Manager Admin Portal by using the following URL, and log in using your credentials.  
[https://<API-M\\_HOST>:<API-M\\_port>/admin](https://<API-M_HOST>:<API-M_port>/admin)

The screenshot shows the WSO2 API Manager Admin Portal. The top navigation bar includes the WSO2 logo, a search bar, and a user icon labeled 'admin'. The left sidebar has a 'TASKS' tab selected, showing sub-options like 'APPLICATION CREATION', 'SUBSCRIPTIONS CREATION', 'APPLICATION REGISTRATION', 'API STATE CHANGE', 'SETTINGS', 'THROTTLING POLICIES', 'LOG ANALYZER', and 'ANALYTICS'. The main content area is titled 'Approval Tasks' and contains a message: 'No tasks assigned to the login user or no connectivity with BPS engine.'

2. Once the Admin Portal opens, click the following icon in the top right corner of the view.



This opens the **Alerts History** page as shown in the example below. The **All Alerts** value is selected in the **Alert Type** field by default as shown in the example below. Therefore, all the alerts that were generated in your WSO2 API Manager installation are displayed by default in this page.

## Alerts History

**Alert Type**

All Alerts

Filter by ...

Alert Timestamp	Type	Message	Severity
Mon Jul 11 2016 15:24:34 GMT+0530 (IST)	Abnormal Request Count	Abnormal request count spike detected during last minute by user_id:admin@carbon.super using application DefaultApplication owned by admin for http POST method of resource template /http://www.mocky.io/v2/577f72de100000250162fc59 in api :admin--API2:vv1.0.0, abnormal request count:32.	moderate

Show 10 entries

3. If you want to filter the alerts displayed in the page by a specific alert type, select the required alert type in the **Alert Type** field as shown in the example below.

## Alerts History

**Alert Type**

Abnormal Request Count

Filter by ...

Alert Timestamp	Type	Message	Severity
Mon Jul 11 2016 15:24:34 GMT+0530 (IST)	Abnormal Request Count	<b>User ID</b> admin@carbon.super <b>Application Name</b> DefaultApplication <b>Application Owner</b> admin <b>API</b> admin--API2:vv1.0.0 <b>Resource Template</b> /http://www.mocky.io/v2/577f72de100000250162fc59 <b>Method</b> POST <b>Reason</b> spike Abnormal request count detected during last minute.	moderate

Show 10 entries

## Analyzing Logs with the Log Analyzer

The Log Analytics in WSO2 API Manager helps you to view your API related statistics and helps you to analyze it. Follow the procedure below to view reports in WSO2 Log Analyzer.

- Start the WSO2 API-M Analytics server. Then start the WSO2 API Manager server by running one of the following commands from the <API-M\_ANALYTICS\_HOME>/bin directory.
  - On Windows: wso2server.bat --run
  - On Linux/Mac OS: sh wso2server.sh

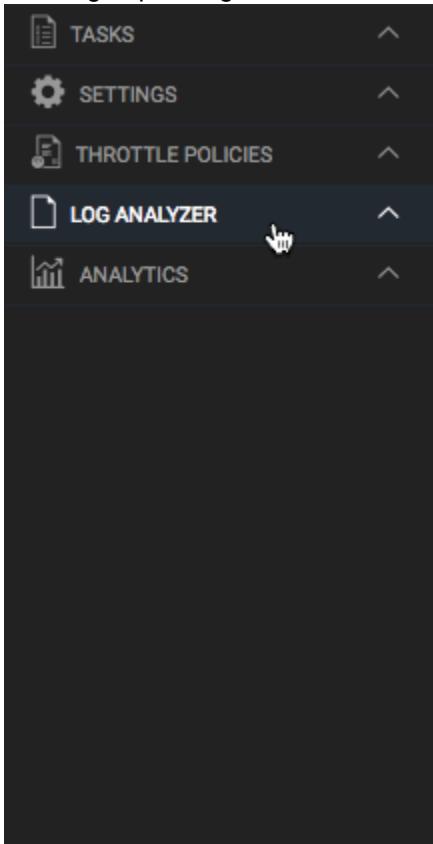
To update the Log Analyzer with the most recent information, execute the APIM\_LOGANALYZER\_SCRIPT script as follows. This script is automatically run every 15 minutes based on the cron job.

- Log into the API-M Analytics Management Console using the https://<ANALYTICS-APIM\_

Host\_Name> : <ANALYTICS-APIM\_Port\_Name>/carbon/ URL.

- b. In the **Main** tab, click **Scripts** to open the **Available Analytics Scripts** page.
- c. Click **Execute** for the APIM\_LOGANALYZER\_SCRIPT script to update the Log Analyzer.

2. Access the WSO2 API Manager Admin Portal using the `https://<Host_Name>:<Port_Name>/admin` URL. Log into the portal using your credentials. As default credentials use both username and password as "admin".
3. In the left navigator, click **Log Analyzer** to expand the **Log Analyzer** section. Then click on one of the following depending on the information you want to view.



- **LIVE LOG VIEWER:** To view logs that are currently being generated. For more information about how to use the gadgets in this page, see [Viewing Live Logs](#).
- **OVERVIEW:** To view the overall statistics for each log category generated. For more information about how to use the gadgets in this page, see [Analyzing the Log Overview](#).
- **APPLICATION ERRORS:** To view detailed information about different categories of errors that have occurred. For more information about how to use the gadgets in this page, see [Analyzing Application Errors](#).
- **API DEPLOYMENT STATS:** To view overall statistics relating to all the APIs that are deployed in WSO2 API Manager. For more information about how to use the gadgets in this page, see [Analyzing API Deployment Statistics](#).
- **LOGIN ERRORS:** To view overall statistics for login errors that have occurred for your WSO2 API Manager installation. For more information about how to use the gadgets in this page, see [Analyzing Login Errors](#).
- **NUMBER OF API FAILURES:** To view statistics relating to instances where the APIs deployed in your WSO2 API Manager installation have failed. For more information about how to use the gadgets in this page, see [Analyzing the Number of API Failures](#).
- **ACCESS TOKEN ERRORS:** To view detailed statistics relating to access token errors. For more information about how to use the gadgets in this page, see [Analyzing Access Token Errors](#).

### **Configuring Log Analyzer for reverse proxy enabled admin portal**

If you have enabled a reverse proxy for admin portal of API Manager, follow the below additional steps to configure Log Analyzer.

1. Configure API Manager to work with reverse proxy as described in [Adding a Reverse Proxy Server](#).

Documentation describe how to apply reverse proxy to store and publisher applications. You can follow the same for admin portal(not WSO2 carbon management console) also.

2. Add two entries in your Load Balancer level for **portal** and **shindig apps**. Example Nginx configuration is shown below.

Note that, at the moment you have to use a context with no prefixes/suffixes as mapping for these two apps. These entries are internal but needed for functionality of admin portal.

```
Location /portal {
 index index.html;
 proxy_set_header X-Forwarded-Host $host;
 proxy_set_header X-Forwarded-Server $host;
 proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
 proxy_pass https://localhost:9443/portal;
 proxy_redirect https://localhost:9443/portal
https://myhost/portal;
 proxy_cookie_path /portal /portal;

}
```

3. Navigate to API Manager node where you are accessing admin portal and open file <APIM\_HOME>/repository/deployment/server/jaggeryapps/portal/store/carbon.super/fs/gadget/gadget-commons/js/gadget-utils.js file.
4. Comment out the following line.

```
{ name: "APIM", svrUrl: "https://" + location.hostname + ":" +
location.port + "/admin/modules/la/log-analyzer-proxy.jag" }
```

5. Add a new line representing reverse proxy URL for admin portal which replaces **location.hostname** + ":" + **location.port** in above commented out line.

If your admin portal URL is <https://myhost/apimanager/admin> after applying reverse proxy new line would look like below.

```
{ name: "APIM", svrUrl:
"https://myhost/apimanager/admin/modules/la/log-analyzer-proxy.jag" }
```

6. Final configuration will look like below.

```
//{ name: "APIM", svrUrl: "https://" + location.hostname + ":" +
location.port + "/admin/modules/la/log-analyzer-proxy.jag" }
{ name: "APIM", svrUrl:
"https://myhost/apimanager/admin/modules/la/log-analyzer-proxy.j
ag" }
```

7. In the same APIM instance, open up <APIM\_HOME>/repository/deployment/server/webapps/shindig/WEB-INF/web.xml and navigate to following section.

```
<context-param>
<param-name>system.properties</param-name>
<param-value>
<![CDATA[
shindig.host=
shindig.port=
aKey=/shindig/gadgets/proxy?container=default&url=]]>
</param-value>
</context-param>
```

Here you have to set values as per your load balancer configuration as **shindig.host** and **shindig.port**. Enter Server hostname as **host** and server HTTPS port as **port**.

If you use the same load balancer configuration used in this documentation, resulting configuration will look like below.

```
<context-param>
<param-name>system.properties</param-name>
<param-value>
<![CDATA[
shindig.host=myhost
shindig.port=443
aKey=/shindig/gadgets/proxy?container=default&url=
]]>
</param-value>
</context-param>
```

8. Save the configurations and start up API Manager.

## Viewing Live Logs

- [Introduction](#)
- [Purpose](#)
- [Recommended action](#)

### Introduction

The Live Log Viewer in the API-M Admin Portal views the latest 100 logs collected from a WSO2 API Manager at any given time. The following is a sample log viewed in the Live Log Viewer. The logs are displayed in the order in which they are generated since the initial page loading. The latest log appears at the bottom. The information displayed on this page gets refreshed in every 5 seconds time.

## Filtered Logs for Tenant

```
Welcome to the live log viewer.

Fri, 08 Jul 2016 09:31:33 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:01:33,802+0530]
Fri, 08 Jul 2016 09:31:33 GMT INFO org.wso2.carbon.registry.core.jdbc.EmbeddedRegistryService
Configured Registry in 3ms
Fri, 08 Jul 2016 09:31:56 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:01:56,885+0530]
Fri, 08 Jul 2016 09:32:13 GMT INFO org.wso2.carbon.databridge.core.DataBridge user admin connected
Fri, 08 Jul 2016 09:32:13 GMT INFO org.apache.synapse.rest.API Initializing API: admin--API2:v1.0.0
Fri, 08 Jul 2016 09:32:13 GMT INFO org.wso2.carbon.mediation.dependency.mgt.DependencyTracker
API : admin--API2:v1.0.0 was added to the Synapse configuration successfully
Fri, 08 Jul 2016 09:32:12 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:02:12,990+0530]
Fri, 08 Jul 2016 09:32:12 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:02:12,877+0530]
Fri, 08 Jul 2016 09:32:12 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:02:12,743+0530]
Fri, 08 Jul 2016 09:32:44 GMT INFO org.wso2.carbon.identity.oauth2.dao.TokenMgtDAO
Thread pool size for session persistent consumer : 100
Fri, 08 Jul 2016 09:32:44 GMT INFO org.wso2.carbon.identity.oauth.config.OAuthServerConfiguration
The default OAuth token issuer will be used. No custom token generator is set.
Fri, 08 Jul 2016 09:32:44 GMT INFO org.wso2.carbon.identity.oauth.config.OAuthServerConfiguration
The default Identity OAuth token issuer will be used. No custom token generator is set.
Fri, 08 Jul 2016 09:32:59 GMT INFO org.apache.synapse.core.axis2.TimeoutHandler
This engine will expire all callbacks after : 120 seconds, irrespective of the timeout action, after the specified or optional timeout
Fri, 08 Jul 2016 09:32:59 GMT WARN org.apache.synapse.transport.http.access.AccessConfiguration
Error loading properties from file: access-log.properties
Fri, 08 Jul 2016 09:32:59 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:02:59,289+0530]
Fri, 08 Jul 2016 09:33:02 GMT INFO org.wso2.andes.kernel.AndesRecoveryTask Running DB sync task.
Fri, 08 Jul 2016 09:33:37 GMT INFO org.wso2.carbon.core.services.util.CarbonAuthenticationUtil
'admin@carbon.super [-1234]' logged in at [2016-07-08 15:03:37,578+0530] from IP address
$!
```

### Purpose

This gadget allows you to view logs online with detailed information in the order in which they were generated.

### Recommended action

To view logs for specific activities, check the Live Log Viewer immediately after you carry them out.

### Analyzing the Log Overview

To analyze the Log Overview,

Login to admin portal ([https://<ip\\_address>:<port>/admin](https://<ip_address>:<port>/admin)).  
In the left navigation, Click **OVERVIEW** under **LOG ANALYZER**.

- Introduction
- Purpose
- Recommended Action

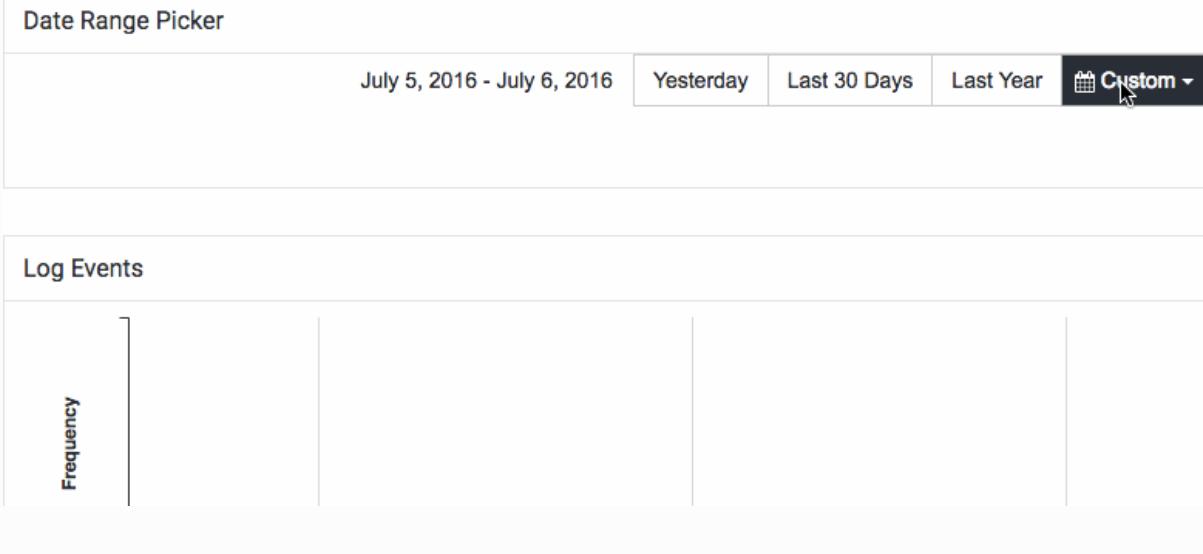
At any given time, this page displays the statistics for a selected time interval.

- If you want to view statistics for a pre-defined time interval, click on the relevant time interval (e.g., **Last 30 Days**).

Date Range Picker

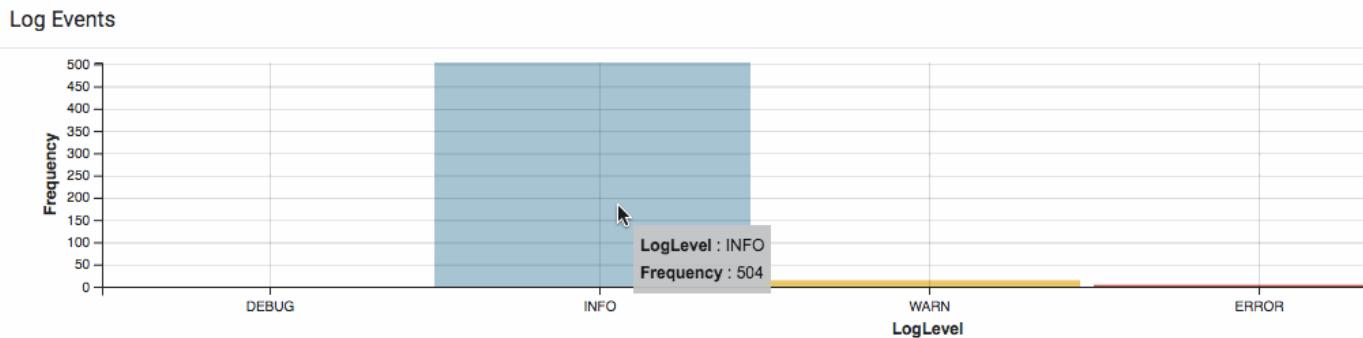
July 6, 2016 - July 8, 2016 Yesterday Last 30 Days Last Year Custom ▾

- If you want to define a custom time interval, click **Custom** and select the start and end dates of the required time interval in the calendar that appears. Then click **Apply**.



## Introduction

This page displays the overall statistics for all the available types of log events (i.e. **INFO**, **DEBUG**, **ERROR**, **WARN** and **FATAL**) that were created during the selected time interval. The information displayed allows you to understand the overall health of the API Manager installation. The exact count for each log event type can be viewed by moving the cursor over the relevant bar. The following is an example of this report.



## Purpose

This gadget allows you to:

- Check the count for each type of log event at different time intervals in order to identify any correlation between frequency of their occurrence and time.
- Compare the count for different types of log events.

## Recommended Action

- If the count for log events of **ERROR** and **FATAL** types are particularly high at a specific time, carry out further investigations for unusual occurrences (e.g., API failure corresponding to the same time interval(s)).
- If the count for log events of **ERROR**, **WARN** and **FATAL** is always high, recheck the configurations for your WSO2 API Manager installation and do the necessary changes to improve the overall health of your setup.
- Compare the count for different type of log events different times and identify any patterns relating to the correlation between the occurrence of log events and time. When major deviations from these patterns are identified, carry out further investigations to identify the causes (e.g., increase/decrease in the load handled).

by WSO2 API Manager.).

## Analyzing Application Errors

To analyze the Application Errors Statistics,

Login to admin portal ([https://<ip\\_address>:<port>/admin](https://<ip_address>:<port>/admin)).

In the left navigation, Click **APPLICATION ERRORS** under **LOG ANALYZER**.

- [Introduction](#)
- [Purpose](#)
- [Recommended action](#)

At any given time, this page displays the statistics for a selected time interval.

- If you want to view statistics for a pre-defined time interval, click on the relevant time interval (e.g., **Last 30 Days**).



- If you want to define a custom time interval, click **Custom** and select the start and end dates of the required time interval in the calendar that appears. Then click **Apply**.



### Errors Distribution

Filtering Type [Message](#)



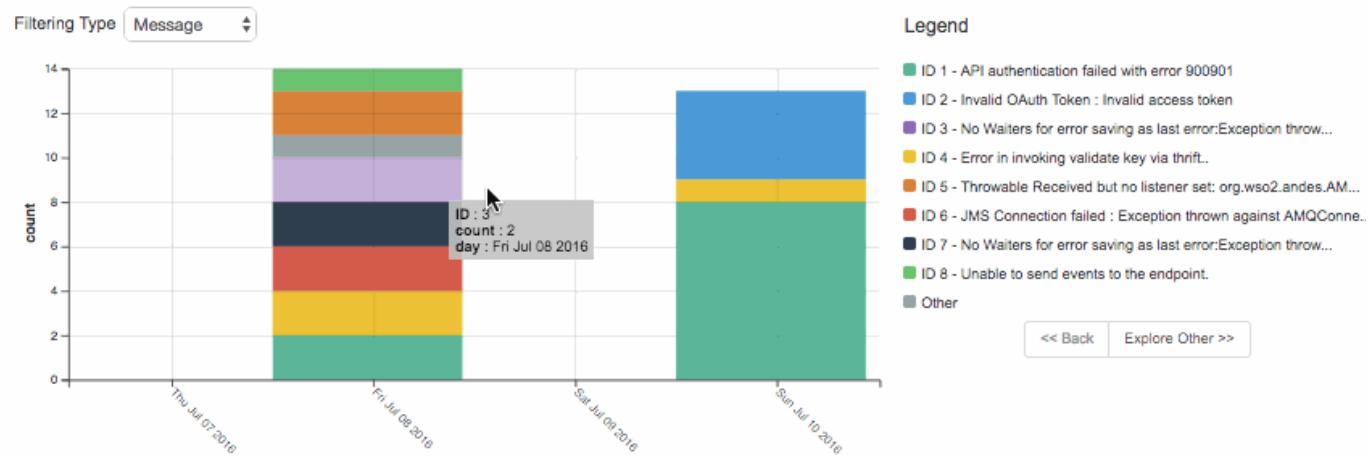
### Legend

- █ ID 1 - No Waiters for error saving as last error:Exception throw...
- █ ID 2 - Unable to send events to the endpoint.
- █ ID 3 - Error in invoking validate service via Thrift

## Introduction

This page displays information relating to application errors that have occurred in the WSO2 API Manager.  
Errors Distribution

## Errors Distribution

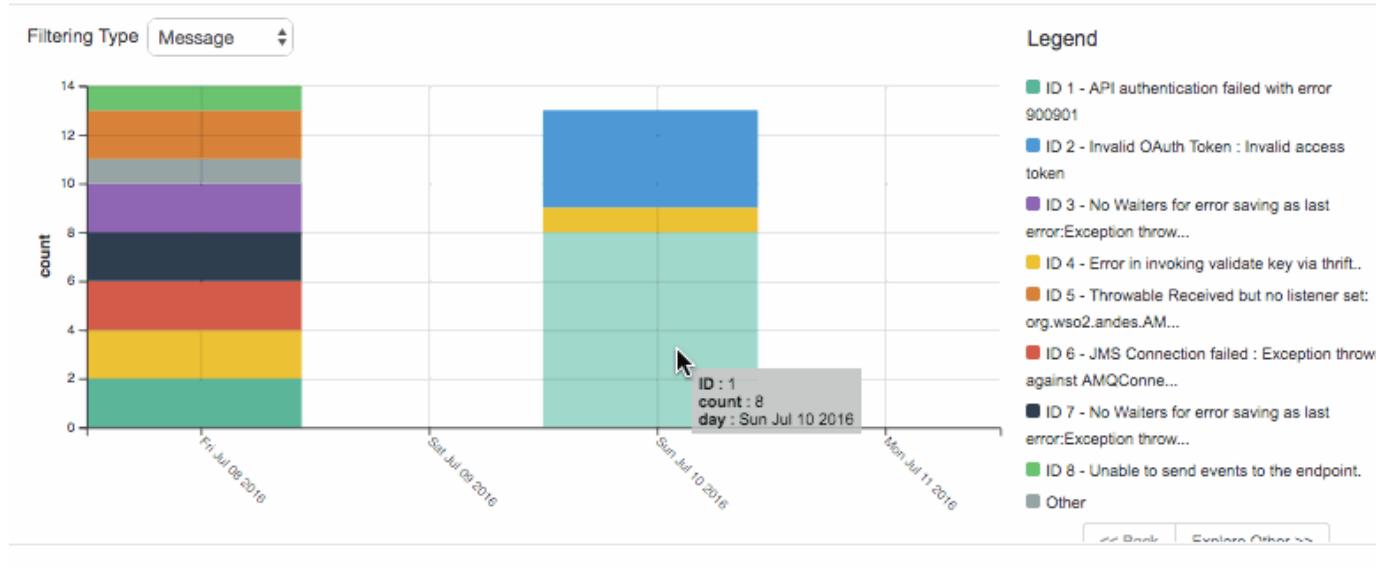


This gadget displays the count of application errors for each day/time over the selected time interval in a bar chart as shown in the example below. Each bar provides a breakdown of errors based on their exception category. If you move the cursor over a specific category, the following information is displayed as demonstrated above.

- ID:** The ID of the exception category.
- Count:** The number of times an exception belonging to the exception category has occurred.
- Day:** The date on which the exception occurred.

Filtered Messages

## Errors Distribution



## Filtered Messages

Search: 

Timestamp	Message	Class	Action
2016-07-10 16:31:17.166	API authentication failed with error 900901	org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler	<a href="#">View</a>
2016-07-10 16:31:21.478	API authentication failed with error 900901	org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler	<a href="#">View</a>
2016-07-10 16:31:27.712	API authentication failed with error 900901	org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler	<a href="#">View</a>
2016-07-10 16:31:43.614	API authentication failed with error 900901	org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler	<a href="#">View</a>
2016-07-10	API authentication failed with error 900901	org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler	<a href="#">View</a>

If you click on a coloured block representing an exception category, the **Filtered Messages** section is populated with details of all the individual occurrences of that exception category as shown above. At a give time, it displays only messages that belong to a selected category. You can sort the records in this table in the ascending/descending order based on the time stamp.

[Log viewer](#)

## Filtered Messages

Filtered Messages			
Timestamp	Message	Class	Action
2016-07-08 13:26:47.750	JMS Connection failed : Exception thrown against AMQConnection: Host: 10.100.5.73 Port: 5672 Virtual Host: carbon Client ID: clientid Active session count: 1: org.wso2.andes.AMQDisconnectedException: Server closed connection and reconnection not permitted. - shutting down worker tasks	org.wso2.carbon.apimgt.jms.listener.utils.JMSTaskManager	 View
2016-07-08 18:28:00.862	JMS Connection failed : Exception thrown against AMQConnection: Host: 10.100.5.73 Port: 5672 Virtual Host: carbon Client ID: clientid Active session count: 1: org.wso2.andes.AMQDisconnectedException: Server closed connection and reconnection not permitted. - shutting down worker tasks	org.wso2.carbon.apimgt.jms.listener.utils.JMSTaskManager	 View
Show <input type="button" value="10"/> entries   Showing 1 to 2 of 2 entries <span style="float: right;">1</span>			

## Log Viewer

### No content to display

Please click on a View button from the above table to access all the log events in the time period surrounding an event.

If you want to view more details about an individual exception occurrence displayed in the **Filtered Messages** section, click **View** on the relevant row. As a result, the **Log Viewer** section displays the log in which the exception was recorded, including the 100 rows that were logged before the exception as well as the 100 rows that were logged after the exception. Different log levels are highlighted in different colours (i.e. **ERROR** level in red, **WARN** level in yellow, and **INFO** level in blue).

## Purpose

This page allows you to:

- Identify the exception categories that have occurred for your APIM Manager installation for different time intervals.
- Compare counts for different exception categories during selected time intervals.
- Check detailed information relating to each exception to identify its cause.
- Identify related logs for each exception to identify its cause.

## Recommended action

- Observe the most frequently occurring exception categories. Identify their causes and take corrective action (e.g., configuration corrections etc.)
- Compare counts for exception categories for different time intervals. If the count is high for any exception

category at specific time intervals, check whether any unusual activity has taken place during that time (e.g., system downtime).

## Analyzing Access Token Errors

To analyze the statistics of Access Token Errors,

Login to admin portal ([https://<ip\\_address>:<port>/admin](https://<ip_address>:<port>/admin)).

In the left navigation, Click **ACCESS TOKEN ERRORS** under **LOG ANALYZER**.

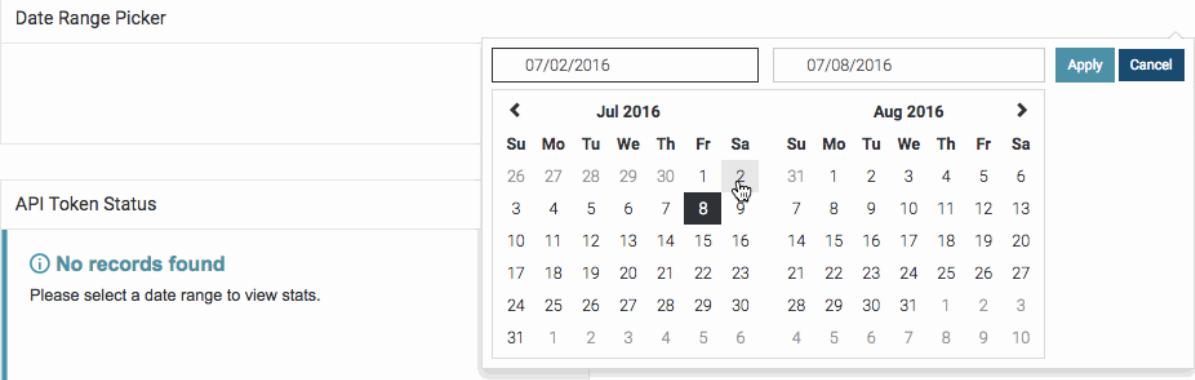
- [Introduction](#)
- [Purpose](#)
- [Recommended action](#)

At any given time, this page displays the statistics for a selected time interval.

- If you want to view statistics for a pre-defined time interval, click on the relevant time interval (e.g., **Last 30 Days**).



- If you want to define a custom time interval, click **Custom** and select the start and end dates of the required time interval in the calendar that appears. Then click **Apply**.

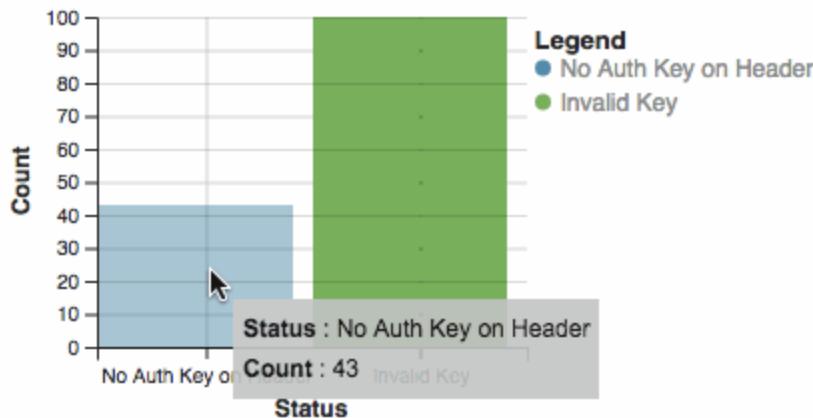


## Introduction

This page displays information about the access token violations that have taken place during a selected time interval.

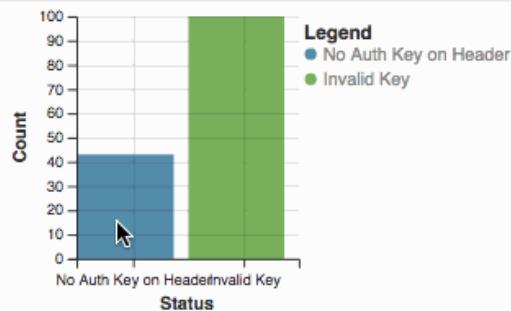
- The **API Token Status** gadget provides a comparison of the count for each access token violation type that has taken place during the selected time interval in a bar chart. Each violation type is displayed with a specific status in the relevant bar. Move the cursor over the relevant bar as demonstrated below to view the exact count for each violation type.

## API Token Status



- The **API Key Status** gadget at a given time displays all the records for a selected API token status in a table. To populate this table, click on the required status in the **API Token Status** gadget as demonstrated below.

## API Token Status



## API Key Status

**i** No content to display

Please click on an error category from the above chart to view the log events.

The records in this table can be sorted in the ascending or the descending order by the timestamp as demonstrated below.

## API Key Status

Search: 

Timestamp	Status	Content	Actions
Thu, 07 Jul 2016 12:15:14 GMT	INVALID_KEY	API authentication failure due to Invalid Credentials	View
Thu, 07 Jul 2016 12:15:13 GMT	INVALID_KEY	API authentication failure due to Invalid Credentials	View
Thu, 07 Jul 2016 12:15:12 GMT	INVALID_KEY	API authentication failure due to Invalid Credentials	View
Thu, 07 Jul 2016 12:14:51 GMT	INVALID_KEY	API authentication failure due to Invalid Credentials	View

- The **Log Viewer** gadget displays the detailed log for a selected record in the **API Key Status** table as demonstrated below.

## API Key Status

Timestamp	Status	Content	Actions
Thu, 07 Jul 2016 12:19:52 GMT	AUTH_FAIL_MISSING_INFO	API authentication failure due to Missing Credentials	View
Thu, 07 Jul 2016 12:20:06 GMT	AUTH_FAIL_MISSING_INFO	API authentication failure due to Missing Credentials	View
Thu, 07 Jul 2016 12:20:13 GMT	AUTH_FAIL_MISSING_INFO	API authentication failure due to Missing Credentials	View
Thu, 07 Jul 2016 12:20:14 GMT	AUTH_FAIL_MISSING_INFO	API authentication failure due to Missing Credentials	View

## Log Viewer

### No content to display

Please click on a View button from the above table to access all the log events in the time period surrounding an event.

## Purpose

This gadget allows you to:

- View the overall count of different errors that have occurred relating to application errors during a selected time interval.
- View detailed information for each occurrence under a selected access token error category to investigate the reason for its occurrence.
- View the 100 logs before and after an error to carry out further investigations about the reason for its occurrence.

## Recommended action

- Compare the access token error count for different time periods. If the count is particularly high for a specific time interval, check for unusual events that have occurred during that time interval (e.g., system downtime).
- Click on individual error categories and view all the records for that category in the **API Key Status** table. View the details of the individual errors and identify any common issues that cause those errors (e.g., a specific user requiring a change in his/her credentials).
- If an error is unique and its cause cannot be identified by checking detailed information in the **API Key Status** table, view other logs that were generated immediately before and after the error in the **Log Viewer** g

adget.

## Analyzing API Deployment Statistics

To analyze the API Deployment Statistics,

Login to admin portal ([https://<ip\\_address>:<port>/admin](https://<ip_address>:<port>/admin)).

In the left navigation, Click **API DEPLOYMENT STATS** under **LOG ANALYZER**.

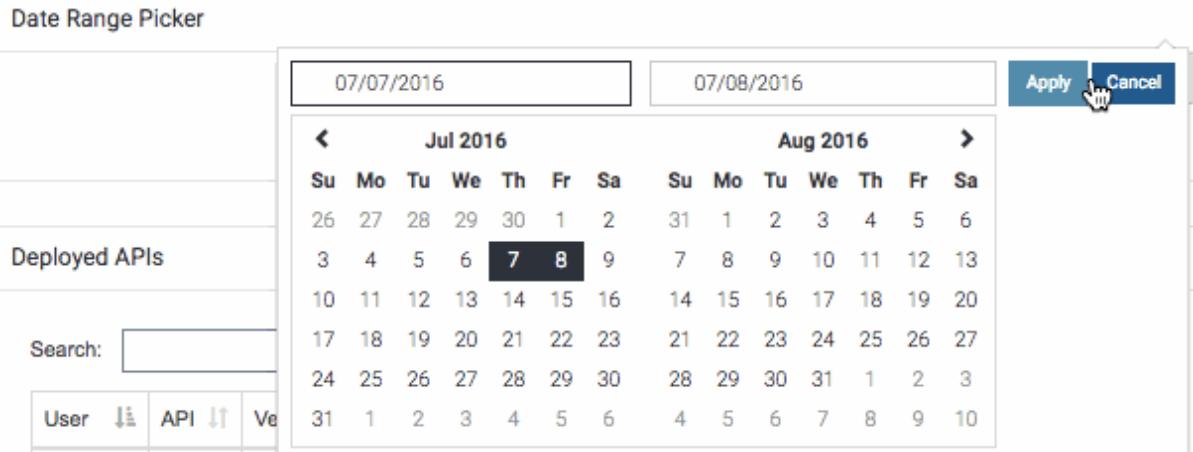
- **Introduction**
- **Purpose**
- **Recommended Action**

At any given time, this page displays the statistics for a selected time interval.

- If you want to view statistics for a pre-defined time interval, click on the relevant time interval (e.g., **Last 30 Days**).



- If you want to define a custom time interval, click **Custom** and select the start and end dates of the required time interval in the calendar that appears. Then click **Apply**.



## Introduction

This page shows the artifacts that were deployed as well as the artifacts that were deleted during the selected time period. It also indicates the number of times each artifact was deployed/deleted. In each gadget, you can search for a specific API and sort the APIs in the ascending/descending order by the available fields

### Deployed APIs

User	API	Version	Frequency
manager	API4	vv1.0.0	1
manager	API5	vv2.0.0	1
admin	API	vv1.0.0	3
admin	API3	vv1.0.0	1
admin	API2	vv1.0.0	4

### Deleted APIs

User	API	Version	Frequency
admin	API3	vv1.0.0	1
manager	API5	vv2.0.0	1

Show 10 entries | Showing 1 to 2 of 2 entries

1

## Purpose

This page allows you to:

- View statistics for all the APIs deployed in your WSO2 API Manager installation.
- Check the status for each API (i.e. whether it is available for use or deleted).
- Understand the extent to which each API is used by checking the frequency with which they were deployed/deleted.

## Recommended Action

Compare the frequency with which different APIs are deployed/deleted to identify the most frequently used APIs.

## Analyzing Login Errors

To analyze the Login Errors,

Login to admin portal ([https://<ip\\_address>:<port>/admin](https://<ip_address>:<port>/admin)).

In the left navigation, Click **LOGIN ERRORS** under **LOG ANALYZER**.

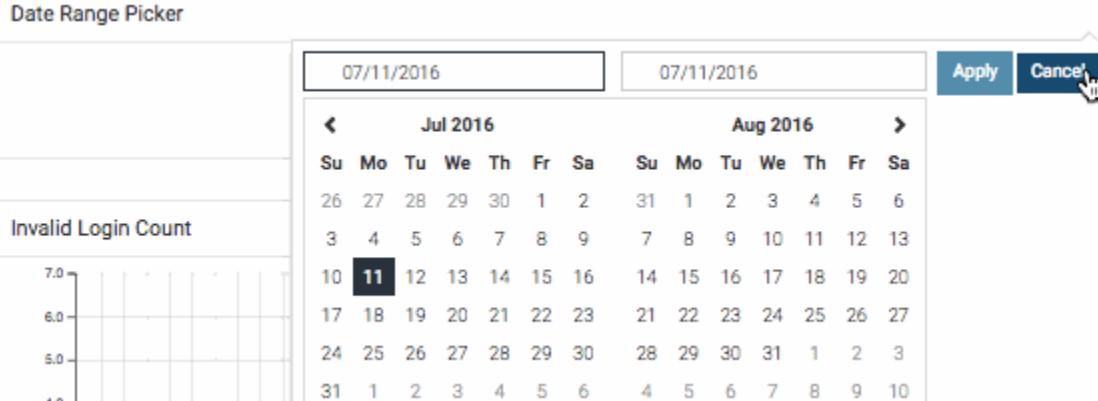
- [Introduction](#)
- [Purpose](#)
- [Recommended action](#)

At any given time, this page displays the statistics for a selected time interval.

- If you want to view statistics for a pre-defined time interval, click on the relevant time interval (e.g., **Last 30 Days**).



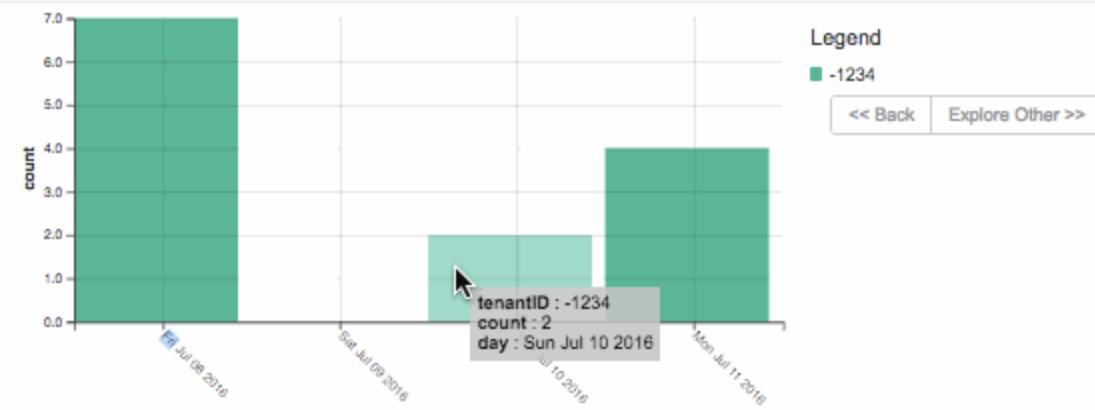
- If you want to define a custom time interval, click **Custom** and select the start and end dates of the required time interval in the calendar that appears. Then click **Apply**.



## Introduction

This page indicates the number of login attempt failures that have occurred during the selected time interval in a bar chart as shown in the example below. The exact count for a specific unit of time (e.g., day, hour, etc.) can be viewed by moving the cursor over the relevant bar.

## Invalid Login Count



## Purpose

This page allows you to identify the time periods during which login errors have occurred to understand what caused them.

## Recommended action

Compare the count for failed login errors at different time intervals. If the count is particularly high during specific time intervals, check for any unusual occurrences that may have taken place during that time (e.g., system downtime). If a high count for login errors is observed for all time intervals, check the relevant configurations in WSO2 API Manager (e.g., configuration of users and user roles).

## Analyzing the Number of API Failures

To analyze the number of API failures,

Login to admin portal ([https://<ip\\_address>:<port>/admin](https://<ip_address>:<port>/admin)).

In the left navigation, Click **NUMBER OF API FAILURES** under **LOG ANALYZER**.

- [Introduction](#)
- [Purpose](#)
- [Recommended Action](#)

At any given time, this page displays the statistics for a selected time interval.

- If you want to view statistics for a pre-defined time interval, click on the relevant time interval (e.g., **Last 30 Days**).

Date Range Picker

July 6, 2016 - July 8, 2016   Yesterday   Last 30 Days   Last Year   Custom

- If you want to define a custom time interval, click **Custom** and select the start and end dates of the required time interval in the calendar that appears. Then click **Apply**.



## Introduction

This report indicates the number of API failures that have occurred during the selected time interval. Each API is represented by a different colour. To check the exact failure count for an API on a specific date, move the cursor over the relevant colour block as demonstrated below.



## Purpose

This page allows you to:

- Check the API failure count at different time intervals, and identify any correlations that may exist between API failure and time.
- Compare the number of failures for different APIs and identify APIs with highest number of failures and identify the corrective action that needs to be taken.

## Recommended Action

- Identify time intervals with the highest API failure counts and investigate further to find the causes.
- Identify APIs with the highest number of failures and investigate further by checking logs relating to these APIs to find the causes, and take corrective action (e.g., correct the message formats).

## Integrating with Google Analytics

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.

The screenshot shows the 'Resources' browser interface. At the top, there are buttons for 'Browse' (which is highlighted with a red box) and 'Search'. Below that, the 'Root' path is shown with a folder icon and a '/' symbol. A 'Location' input field contains '/'. There are two tabs: 'Tree view' (selected) and 'Detail view'. The main area displays a tree structure of configuration files under the root:

- /
- \_system
  - config
  - governance
  - apimgt
    - applicationdata
    - customsequences
    - externalstores
    - statistics
      - ga-config.xml

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 WSO2 API Manager Content editor interface. At the top, there are buttons for 'Display as text' (disabled), 'Edit as text' (highlighted with a red box), 'Upload', and 'Download'. Below this, there are two radio buttons: 'Plain Text Editor' (selected) and 'Rich Text Editor'. The main area contains XML configuration code for Google Analytics tracking. Several parts of the code are highlighted with red boxes: the 'Enabled' attribute in the <Enabled> tag, the <TrackingID> tag, and its value 'UA-XXXXXXX-X'. At the bottom of the editor are 'Save Content' and 'Cancel' buttons.

```

<!--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-XXXXXXX-X</TrackingID>

</GoogleAnalyticsTracking>

```

- If you want to enable tracking for tenants, log in to the management console with a tenant's credentials, click **Source View**, and then add the following parameter to the org.wso2.carbon.mediation.registry.WS02Registry registry definition near the top (repeat this step for each tenant):
 

```
<parameter name="cachableDuration">15000</parameter>
```

The following screen shot illustrates this change:

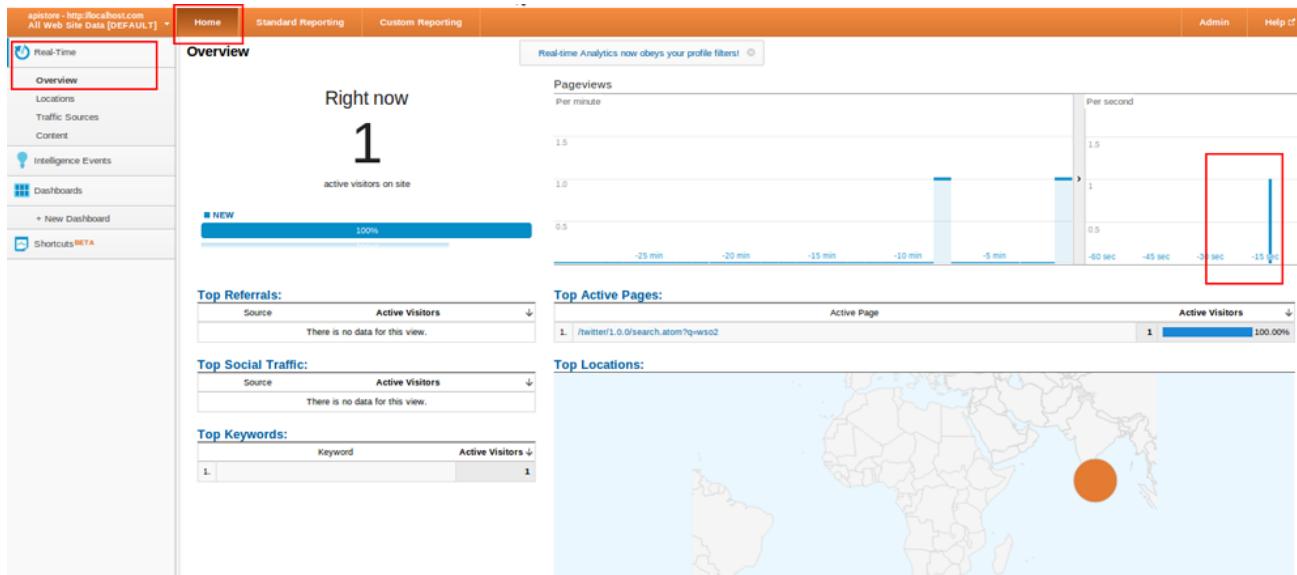
The screenshot shows the WSO2 API Manager Service Bus Configuration page. On the left, there is a navigation sidebar with sections like Identity, Main, Monitor, Configure, Tools, Extensions, and Resources. Under 'Configure', 'Source View' is highlighted with a red box. The main content area is titled 'Service Bus Configuration' and contains an 'ESB Configuration' editor. The editor shows XML code for a sequence. A specific line of code, containing a parameter declaration, is highlighted with a red box:
 

```
<parameter provider="org.wso2.carbon.mediation.registry.WS02Registry" name="cachableDuration">15000</parameter>
```

- 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**.
- Invoke the above API using the embedded **WSO2 REST Client** (or any third-part rest client such as cURL).

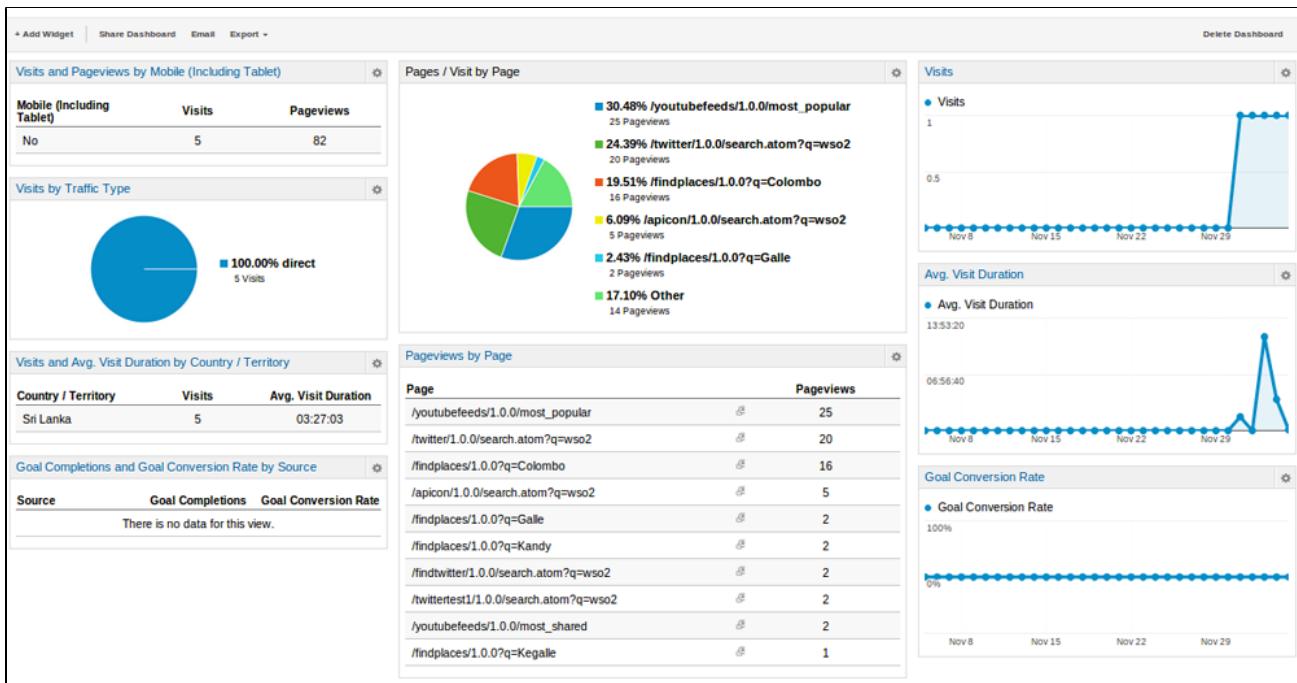
### Real-time statistics

- 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.

## Monitoring with WSO2 Carbon Metrics

WSO2 Carbon Metrics provides an API for WSO2 Carbon Components to use the [Metrics library](#). WSO2 API Manager is shipped with JVM Metrics to monitor statistics of your API Manager server using Java Metrics. The following sections provide a detailed description of how Carbon Metrics is used in API Manager for monitoring.

- Enabling Metrics and Storage Types
- Configuring Metrics Properties
- Using JVM Metrics

## Enabling Metrics and Storage Types

Given below are the configurations that should be in place for your API Manager server in order to use the metrics feature. You need to first enable metrics for your server and then enable the required storage types (reporters) that are used for storing the metrics data. See the following topics for instructions:

- Enabling metrics
- Configuring the storage of metrics
- Sample configuration

### Enabling metrics

To enable metrics for your product, set the `Enabled` parameter under the `Metrics` element to `true` in the `<APIM_HOME>/repository/conf/metrics.xml` file. Alternatively, you can enable metrics at the time of starting the API Manager server by using the following command:

```
-Dmetrics.enabled=true
```

### Configuring the storage of metrics

WSO2 API Manager is configured by default to store the information from metrics in the following reporters: JMX, CSV and JDBC. These reporters are configured in the `metrics.xml` file (stored in the `<APIM_HOME>/repository/conf` directory). You can disable metrics for individual reporters by setting the `Enabled` parameter to `false`.

If you set the `Enabled` parameter under the `metrics` element to `false` in the `metrics.xml` file, metrics is disabled for all the reporters and it is not possible to enable metrics for individual reporters.

See the following topics for information on configuring each of the available storage types.

- JMX
- CSV
- JDBC

### JMX

The following parameters in the `metrics.xml` file can be used to configure a JMX storage for metrics data.

Element Name	Description	Type	Default Value	Mandatory/Optional
<b>Enabled</b>	This parameter specifies whether metrics monitoring is enabled for JMX or not.	Boolean	true	Mandatory

### CSV

The following parameters in the `metrics.xml` file can be used to configure a CSV storage for metrics data.

Element Name	Description	Type	Default Value	Mandatory/Optional
<b>Enabled</b>	This parameter specifies whether metrics monitoring is enabled for CSV or not.	Boolean	false	Mandatory

<b>Location</b>	The location where the CSV files are stored.	String	<code> \${carbon.home}/repository/logs/metrics/</code>	
<b>PollingPeriod</b>	The time interval between polling activities that are carried out to update the metrics dashboard based on latest information. For example, if the polling period is 60, polling would be carried out every 60 milliseconds.	Integer	60	

## JDBC

The following parameters in the `metrics.xml` file can be used to configure a JDBC storage for metrics data.

Element Name	Description	Type	Default Value
<b>Enabled</b>	This parameter specifies whether metrics monitoring is enabled for JDBC or not.	Boolean	true
<b>DataSourceName</b>	The name of the datasource used.	String	<code>jdbc/WSO2MetricsI</code>
<b>PollingPeriod</b>	The time interval between polling activities that are carried out to update the metrics dashboard based on latest information. For example, if the polling period is 60, polling would be carried out every 60 seconds.	Integer	60
<b>ScheduledCleanup</b>	This element contains parameters relating to scheduled cleanup. The possible values are <code>Enabled</code> , <code>ScheduledCleanupPeriod</code> and <code>DaysToKeep</code> . Scheduled cleanup involves scheduling a task to clear metric data in the database after a specified time interval. This is done to avoid excessive memory usage.		
<b>ScheduledCleanup/Enabled</b>	This parameter specifies whether scheduled cleanup is enabled or not.	Boolean	true

<b>ScheduledCleanup/ScheduledCleanupPeriod</b>	The number of seconds that should elapse after a cleanup task before the next clean-up task is carried out.	Integer	86400
<b>ScheduledCleanup/DaysToKeep</b>	The number of days during which the scheduled clean-up task should be carried out.	Integer	

If you have enabled JDBC, then you also need to specify a datasource configuration to be used to create the connection between WSO2 API Manager and the JDBC data storage system. The `metrics-datasources.xml` file is used for configuring this datasource for metrics.

Parameters that can be configured for a datasource are as follows:

XML element	Attribute	Description
<code>&lt;datasources-configuration&gt;</code>	<code>xmlns</code>	The root element. The namespace is specified as: <code>xmlns:svns="://org.wso2.securevault/configuration"</code>
<code>&lt;providers&gt;</code>		The container element for the datasource providers.
<code>&lt;provider&gt;</code>		The datasource provider, which should implement <code>org.wso2.carbon.datasource.common.spi.DataSourceReader</code> . The datasources follow a pluggable approach in providing datasource type implementations using this approach.
<code>&lt;datasources&gt;</code>		The container element for the datasources.
<code>&lt;datasource&gt;</code>		The root element of a datasource.
<code>&lt;name&gt;</code>		Name of the datasource.
<code>&lt;description&gt;</code>		Description of the datasource.
<code>&lt;jndiConfig&gt;</code>		The container element that allows you to expose this datasource as a JNDI datasource.
<code>&lt;name&gt;</code>		The JNDI resource name to which this datasource should be bound.
<code>&lt;environment&gt;</code>		The container element in which you specify the following properties: <ul style="list-style-type: none"> <li><code>java.naming.factory.initial</code>: Selects the registry service provider as the initial context.</li> <li><code>java.naming.provider.url</code>: Specifies the location of the registry when the registry is being used as the initial context.</li> </ul>
<code>&lt;definition&gt;</code>	<code>type</code>	The container element for the data source definition. Set the type attribute to RDBMS, or to custom if you're creating a custom type "RDBMS" data source reader expects a "configuration" element with sub-elements listed below.
<code>&lt;configuration&gt;</code>		The container element for the RDBMS properties.

<url>		The connection URL to pass to the JDBC driver to establish the connection.
<username>		The connection user name to pass to the JDBC driver to establish connection.
<password>		The connection password to pass to the JDBC driver to establish connection.
<driverClassName>		The class name of the JDBC driver to use.
<maxActive>		The maximum number of active connections that can be allocated from this pool at the same time.
<maxWait>		Maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception.
<testOnBorrow>		Specifies whether objects are validated before being borrowed from the pool. If the object fails to validate, it is dropped from the pool, and the attempt to borrow another. When set to true, the validationQuery parameter must be set to a non-null string.
<validationQuery>		The SQL query used to validate connections from this pool before returning them to the caller. If specified, this query does not have to return any data, it just can't throw a SQLException. The default value is null. Example values are SELECT 1(mysql), select 1 from dual(oracle), SELECT 1(MS Sql Server).
<validationInterval>		To avoid excess validation, only run validation at most at this frequency (interval time in milliseconds). If a connection is due for validation and has been validated previously within this interval, it will not be validated again. The default value is 30000 (30 seconds).

## Sample configuration

Shown below is a sample `metrics.xml` file with the default configurations specifying the types of storages enabled for metrics data. See the above topics for instructions.

### ▼ The default configurations in the `metrics.xml` file

```
-->

<!--
 This is the main configuration file for metrics
-->

<Metrics xmlns="http://wso2.org/projects/carbon/metrics.xml">

 <!--
 Enable Metrics
 -->
 <Enabled>false</Enabled>
 <!--
```

```

Metrics reporting configurations
-->

<Reporting>
 <JMX>
 <Enabled>true</Enabled>
 </JMX>
 <CSV>
 <Enabled>false</Enabled>
 <Location>${carbon.home}/repository/logs/metrics/</Location>
 <!-- Polling Period in seconds -->
 <PollingPeriod>60</PollingPeriod>
 </CSV>
 <JDBC>
 <Enabled>true</Enabled>
 <!-- Source of Metrics, which will be used to
 identify each metric in database -->
 <!-- Commented to use the hostname
 <Source>Carbon</Source>
 -->
 <!--
 JNDI name of the data source to be used by the JDBC
 Reporter.
 This data source should be defined in a
 *-datasources.xml
 file in conf/datasources directory.
 -->
 <DataSourceName>jdbc/WSO2MetricsDB</DataSourceName>
 <!-- Polling Period in seconds -->
 <PollingPeriod>60</PollingPeriod>
 <ScheduledCleanup>
 <!--
 Schedule regular deletion of metrics data older than
 a set number of days.
 It is strongly recommended that you enable this job
 to ensure your metrics tables do not get extremely
 large. Deleting data older than seven days should be
 sufficient.
 -->
 <Enabled>true</Enabled>
 <!-- This is the period for each cleanup operation in
 seconds -->
 <ScheduledCleanupPeriod>86400</ScheduledCleanupPeriod>
 <!-- The scheduled job will cleanup all data older than
 the specified days -->
 <DaysToKeep>7</DaysToKeep>
 </ScheduledCleanup>

```

```

 </JDBC>
</Reporting>
</Metrics>

```

## Configuring Metrics Properties

The `<APIM_HOME>/repository/conf/metrics.properties` file specifies properties that correspond to the gauges in the **Metrics Dashboard**. See the topic on [using JVM metrics](#) for details on using the metrics dashboard for JVM metrics. The level defined for a property in this file determines the extent to which the relevant gauge in the dashboard should be updated with information. The different levels that can be defined for properties are as follows:

Level	Description
<b>Off</b>	Designates no informational events.
<b>Info</b>	Designates informational metric events that highlight the progress of the application at coarse-grained level.
<b>Debug</b>	Designates fine-grained informational events that are most useful to debug an application.
<b>Trace</b>	Designates finer-grained informational events than the DEBUG.
<b>All</b>	Designates all the informational events.

If no specific level is configured for a property in the `metrics.properties` file, the metrics root level applies. The root level is defined as shown in the following example in the `metrics.properties` file.

```
metrics.rootLevel=OFF
```

If you want to change the current root level, you can also use the following command.

```
-Dmetrics.rootLevel=INFO
```

The levels in `metrics.properties` file can be configured to any hierarchy. However, if the level defined for an individual property is different to the level defined for its parent in the hierarchy, the level defined for the individual property overrules that of the parent. For example, if we have `metric.level.jvm.memory=INFO` in the `<APIM_HOME>/repository/conf/metrics.properties` file, all metrics under `jvm.memory` have `INFO` as the configured level. However, if you have `metric.level.jvm.memory.heap=TRACE`, the `TRACE` level would apply for the `metric.level.jvm.memory.heap` property even though it is a child property of `jvm.memory`.

The properties that are included in this file by default are as follows:

- [JVM's direct and mapped buffer pools](#)
- [Class loading](#)
- [GC](#)
- [Memory](#)
- [Operating system load](#)
- [Threads](#)

### JVM's direct and mapped buffer pools

### Class loading

Property	Default Level	Description
metric.level.jvm.class-loading	INFO	The gauge showing the number of classes currently loaded for the JVM.

## GC

Property	Default Level	Description
metric.level.jvm.gc	DEBUG	The gauge for showing garbage collection and memory usage. Monitoring this allows you to identify memory leaks and memory thrash, which have a negative impact on performance.

## Memory

Property	Default Level	Description
metric.level.jvm.memory	INFO	The gauge for showing the used and committed memory in WSO2 API Manager.
metric.level.jvm.memory.heap	INFO	The gauge for showing the used and committed heap in WSO2 API Manager.
metric.level.jvm.memory.non-heap	INFO	The gauge for showing the used code cache and used CMS Perm Gen in WSO2 API Manager.
metric.level.jvm.memory.total	INFO	The gauge for showing the total memory currently available for the JVM.
metric.level.jvm.memory.pools	OFF	The gauge for showing the used and available memory for JVM in the memory pool.

## Operating system load

Property	Default Level	Description
metric.level.jvm.os	INFO	The gauge for showing the current load imposed by the JVM on the operating system.

## Threads

Property	Default Level	Description
metric.level.jvm.threads	OFF	The parent property of all the gauges relating to the JVM thread pool. The metric level defined for this property applies to all the rest of the properties in this table. The metric level set via this property to a child property can be overruled if a different level is set for it.

metric.level.jvm.threads.count	DEBUG	The gauge for showing the number of active and idle threads currently available in the JVM thread pool.
metric.level.jvm.threads.daemon.count	DEBUG	The gauge for showing the number of active daemon threads currently available in the JVM thread pool.
metric.level.jvm.threads.blocked.count	OFF	The gauge for showing the number of threads that are currently blocked in the JVM thread pool.
metric.level.jvm.threads.deadlock.count	OFF	The gauge for showing the number of threads that are currently deadlocked in the JVM thread pool.
metric.level.jvm.threads.new.count	OFF	The gauge for showing the number of new threads generated in the JVM thread pool.
metric.level.jvm.threads.runnable.count	OFF	The gauge for showing the number of runnable threads currently available in the JVM thread pool.
metric.level.jvm.threads.terminated.count	OFF	The gauge for showing the number of threads terminated from the JVM thread pool since you started running the WSO2 API Manager instance.
metric.level.jvm.threads.timed_waiting.count	OFF	The gauge for showing the number of threads in the Timed_Waiting state.
metric.level.jvm.threads.waiting.count	OFF	The gauge for showing the number of threads in the waiting state in the JVM thread pool. One or more other threads are required to perform certain actions before these threads can proceed with their actions.

### Viewing API specific Handler Metrics

Follow the below steps to view API specific metrics for handlers which are exposed through Dropwizard. These are timer metrics which allow us to see how much time is taken for each handler.

1. Copy the **JMX Service URL** displayed in the console at API Manager server startup.

```
INFO - JMXServerManager JMX Service URL :
service:jmx:rmi://localhost:11111/jndi/rmi://localhost:9999/jmxrmi
```

2. Go to <JDK\_HOME>/bin. JDK\_HOME is the directory where the JDK(Java Development Kit) is installed. Find the **jconsole** executable file in the directory and run it on the command/shell prompt.

You can type **jconsole** in the command/shell prompt to run the file.

3. In the prompted window, enter the JMX Service URL copied before as a new connection. Type **admin** in both username and password fields and click **Connect**.
4. Now **invoke an API** in WSO2 API Manager.

Handler Metrics will only be visible when you invoke an API.

5. Go to the **MBeans** tab in the open **jconsole** window. You can view the implemented metrics by using Dropwizard Metrics Library which starts with the **org.wso2.am** prefix.

## Using JVM Metrics

JVM metrics are Java metrics enabled by default in WSO2 products for the purpose of monitoring general statistics related to the server performance. Follow the procedure below to view the JVM metrics dashboard for a WSO2 product.

For detailed instructions to enable/disable JVM metrics and to configure metrics, see [Enabling Metrics and Storage Types](#).

1. Log into the Management Console of the WSO2 product. Click **Monitor -> Metrics -> JVM Metrics** to open the **View Metrics** page.
2. Specify the source for the JVM metrics by selecting a value from the drop-down list for the **Source** parameter in the top panel.



3. Specify the time interval for which the statistics should be displayed in the dashboard by selecting a value from the following drop-down list in the top panel.



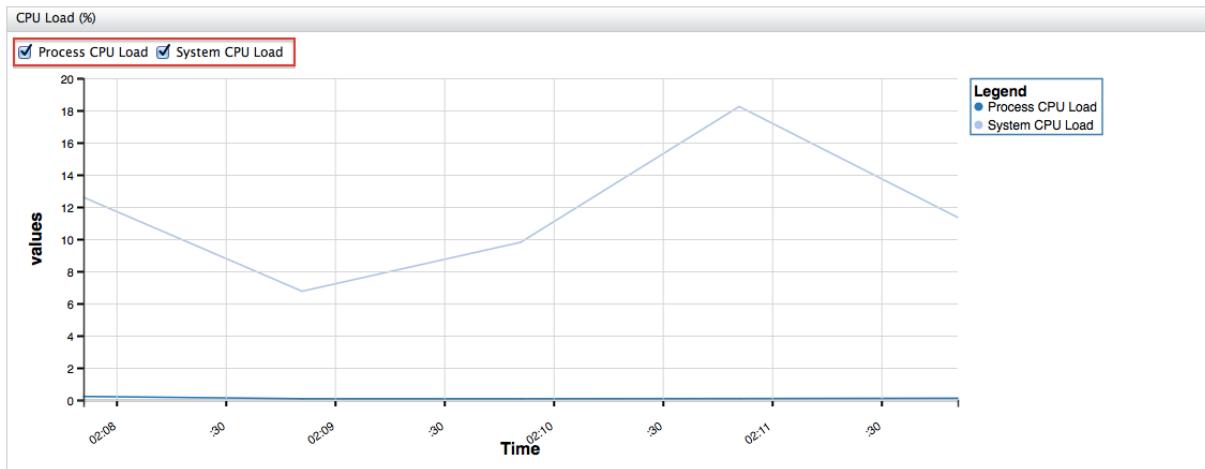
4. Click the required buttons opposite **Views** in the top panel to select the types of information you want to view in the dashboard, and refresh the web page.

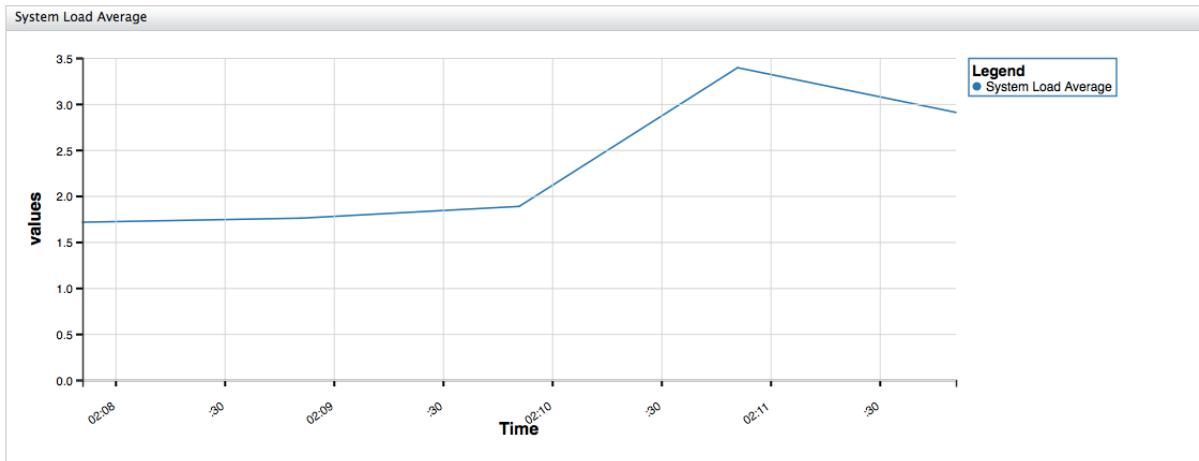


Statistics corresponding to each button can be viewed as follows:

- **CPU**

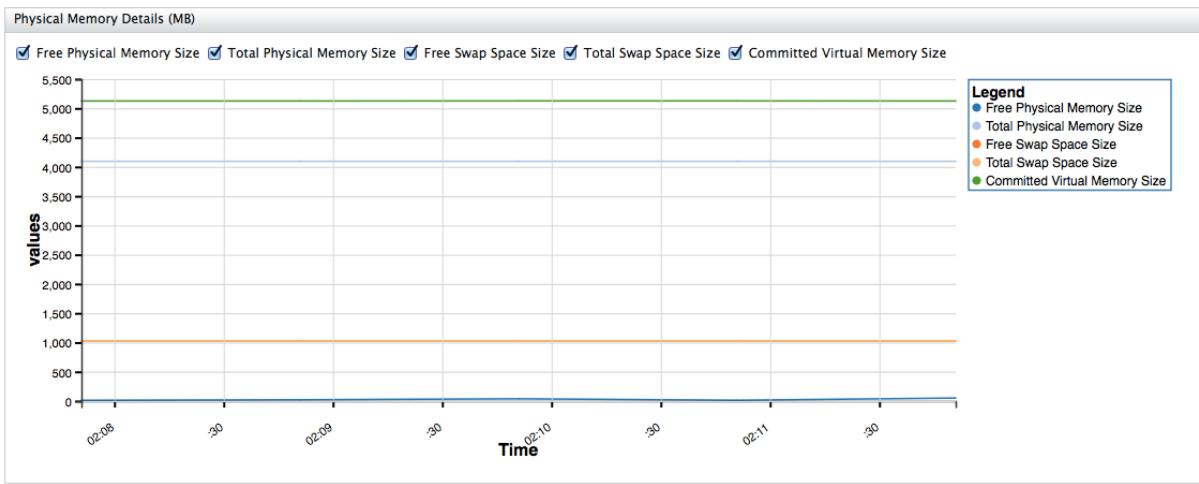
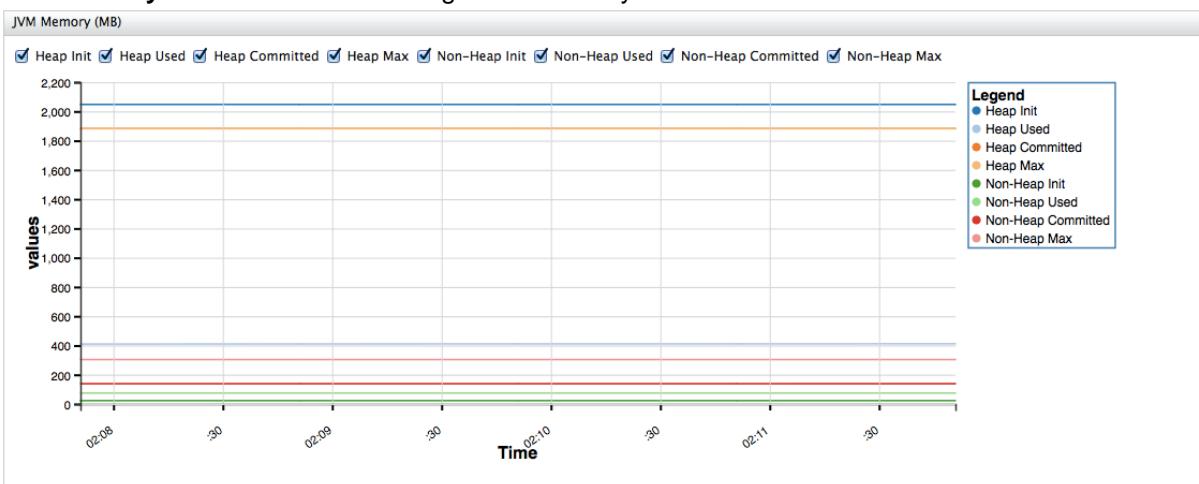
Click this button to view statistics relating to the CPU as shown below.





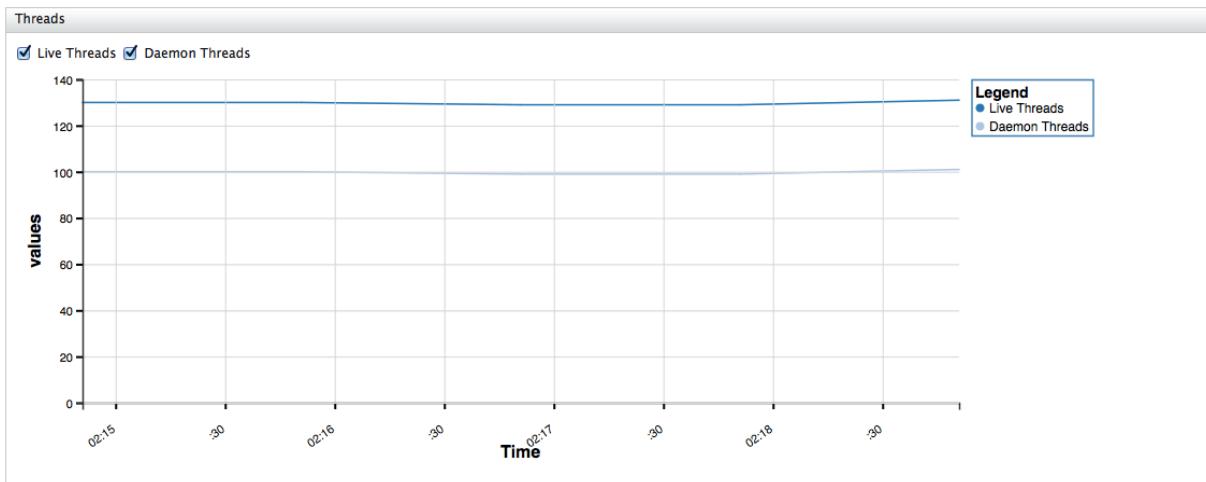
- **Memory**

Click **Memory** to view statistics relating to the memory as shown below.



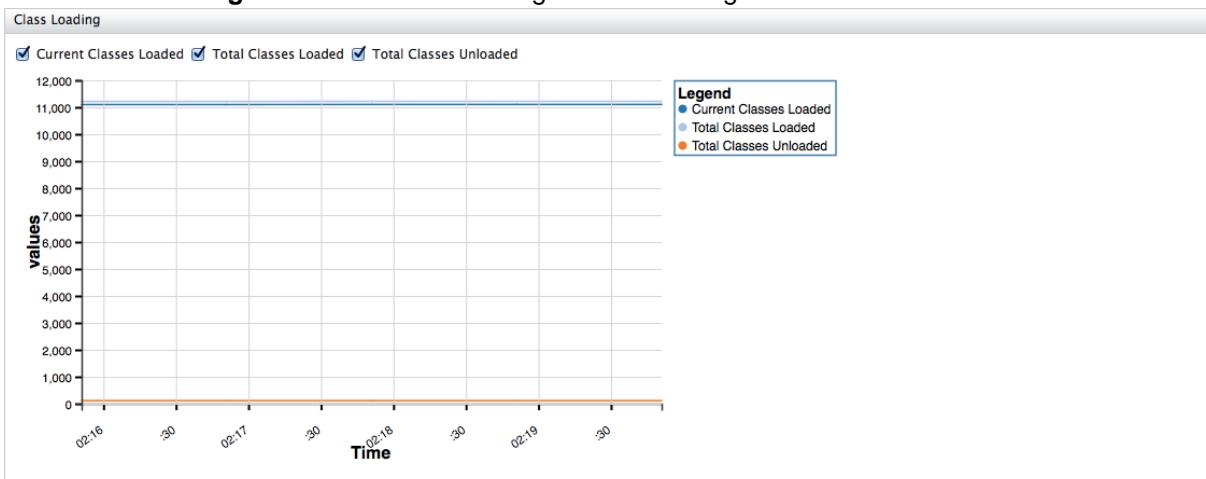
- **Threading**

Click **Threading** to view statistics relating to threading as shown below.



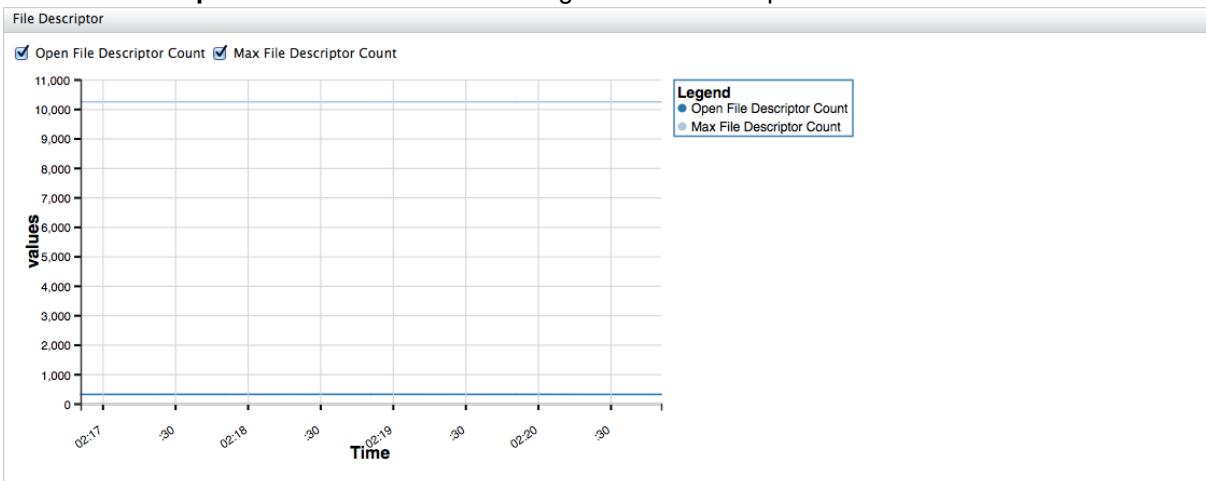
- **Class Loading**

Click **Class Loading** to view statistics relating to class loading as shown below.



- **File Descriptor**

Click **File Descriptor** to view information relating to the file descriptor count as shown below.



## Installing WSO2 APIM Analytics Features

- [Introduction](#)
- [Getting P2 repositories](#)
- [Installing required features in WSO2 DAS](#)

## Introduction

Each WSO2 product is a collection of reusable software units called features. A single feature is a list of components and/or other features. This section describes how to install the specific features required in order to use the WSO2 APIM Analytics features in WSO2 Data Analytics Server or any other WSO2 product that performs Analytics.

This procedure can be followed for WSO2 DAS, WSO2 ESB Analytics and WSO2 IS Analytics. APIM Analytics features are installed by default in WSO2 APIM Analytics.

## Getting P2 repositories

The following steps are required if you are adding the repository by uploading it as a file. They are not required if you provide a link to the repository as described in the next section. For detailed information about adding features and repositories, see [WSO2 Administration Guide - Installing Features](#).

1. Download the latest p2 repository from [here](#).
2. Unzip the p2 repository (p2-repo.zip) into a local directory in your machine.

## Installing required features in WSO2 DAS

Follow the steps below to install the required features in WSO2 DAS.

1. Download WSO2 DAS and start the server. For instructions, see the [Installation Guide](#).
2. Log into the Management Console.
3. Click **Configure**, and then click **Features**.
4. Click **Repository Management**, and then click **Add Repository**.
5. Enter the details as shown below to add the created P2 repository.

Home > Configure > Features

**Feature Management**

Add Repository

You can add a new local repository or a remote repository

Name: *	WSO2 P2 Repository
Location:	<input checked="" type="radio"/> URL <a href="http://product-dist.wso2.com/p2/carbon/releases/wilkes/">http://product-dist.wso2.com/p2/carbon/releases/wilkes/</a> <input type="radio"/> Local <a href="/Users/rukshaniweerasinha/Downloads/p2-repo2">/Users/rukshaniweerasinha/Downloads/p2-repo2</a>

**Add** **Cancel**

Parameter Name	Value
<b>Name</b>	WSO2 P2 Repository
<b>Location (URL)</b>	<a href="http://product-dist.wso2.com/p2/carbon/releases/wilkes/">http://product-dist.wso2.com/p2/carbon/releases/wilkes/</a>

6. Click **Add** to add the repository.
7. Click the **Available Features** tab.
8. In the **Repository** parameter, select the **WSO2 P2 Repository** repository you previously added.
9. Click **Find Features**. Once all the features are listed, select the following features.

<input type="checkbox"/> Analytics Spark	1.3.0	<a href="#">More Info.</a>
<input checked="" type="checkbox"/> Analytics Spark Geolocation UDF	1.0.3	<a href="#">More Info.</a>
<input checked="" type="checkbox"/> Analytics Spark scripts Common	1.0.3	<a href="#">More Info.</a>
<input checked="" type="checkbox"/> Analytics Spark Useragent UDF	1.0.3	<a href="#">More Info.</a>
<input type="checkbox"/> Analytics Util	1.0.2	<a href="#">More Info.</a>
<input type="checkbox"/> Analytics Web Service	1.3.0	<a href="#">More Info.</a>
<input type="checkbox"/> Analytics Web Service Server	1.3.0	<a href="#">More Info.</a>
<input checked="" type="checkbox"/> APIM Analytics Server Core	2.1.2	<a href="#">More Info.</a>
<input checked="" type="checkbox"/> APIM Analytics Server UDF	2.1.2	<a href="#">More Info.</a>

- Analytics Spark Geolocation UDF
- Analytics Spark scripts Common
- Analytics Spark Useragent
- APIM Analytics Server Core
- APIM Analytics Server UDF

If API Manager Analytics features are being installed in an Analytics product where IS Analytics features are also installed, the version of the Analytics Spark Geolocation UDF feature selected should be 1.0.2.

If you cannot find these features, retry with one of the following suggestions:

- Clear the **Group features by category** check box, and then click **Find Features** again.
- Try adding a more recent P2 repository. The repository you added could be deprecated.
- Check the **Installed Features** to see whether the feature is already installed.

10. Once the features are selected, click **Install** to proceed with the installation.

11. Once the installation is completed, restart the product server.

## Purging Analytics Data

Data purging is an option to remove historical data in WSO2 API Manager Analytics. This is important since it is not possible to delete tables or table data in WSO2 API Manager Analytics. By purging data, you can achieve high performance in data analysis without removing analyzed summary data. Only data from the stream data generated by API Manager is purged. This data is contained in the following tables:

- ORG\_WSO2\_APIMGT\_STATISTICS\_WORKFLOW
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERMINUTEREQUEST
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERHOURREQUEST
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERDAYREQUEST
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERMINUTERESPONSE
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERHOURRESPONSE
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERDAYRESPONSE
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERMINUTEEXECUTIONTIMES
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERHOURExecUTIONTIMES
- ORG\_WSO2\_APIMGT\_STATISTICS\_PERDAYEXECUTIONTIMES
- ORG\_WSO2\_APIMGT\_STATISTICS\_THROTTLE
- ORG\_WSO2\_APIMGT\_STATISTICS\_FAULT
- ORG\_WSO2\_CARBON\_IDENTITY\_OAUTH\_TOKEN\_ISSUANCE
- LOGANALYZER

Make sure you do not purge data in tables other than those mentioned above because that deletes your summarized historical data.

There are two ways to purge data in API Manager Analytics:

- [Using the Management Console](#)
- [Using the global method](#)

#### **Using the Management Console**

To schedule data purging via the Management Console, follow the steps below:

1. Login to the WSO2 API-M Analytics Management Console.
2. In the **Main** tab, click **Data Explorer** to open the **Data Explorer** page.
3. Select the required table in the **Table Name** field, and click **Schedule Data Purging** as shown below.

This opens the **Schedule Data Purging** dialog box.

4. In the **Schedule Data Purging** dialog box, set the time and days within which you want to purge data as shown below. Then click **Save**.

Do not purge data that is less than 2 days old as it can result in a data loss.

5. Repeat these steps for all 6 tables mentioned above to schedule all of them to be cleared regularly.

#### **Using the global method**

This action affects all tenants.

1. Open the <API-M\_ANALYTICS\_HOME>/repository/conf/analytics/analytics-config.xml file.
2. Change the contents under the <analytics-data-purging> property as shown below:

```

<analytics-data-purging>
 <purging-enable>true</purging-enable>
 <cron-expression>0 0 12 * * ?</cron-expression>
 <purge-include-table-patterns>
 <table>.*</table>
 </purge-include-table-patterns>
 <data-retention-days>365</data-retention-days>
</analytics-data-purging>

```

3. Save your changes.

## Default Ports of WSO2 API-M Analytics

Given below are the specific ports used by WSO2 API-M Analytics

### *Ports inherited from WSO2 BAM*

- 7712 - Thrift SSL port for secure transport, where the client is authenticated to use WSO2 API-M Analytics.
- 7612 - Thrift TCP port where WSO2 API-M Analytics receives events from clients.

### *Ports used by the Spark Analytics Engine*

The Spark Analytics engine is used in 3 separate modes in WSO2 API-M Analytics as follows.

- Local mode
- Cluster mode
- Client mode

Default port configurations for these modes are as follows.

For more information on these ports, go to [Apache Spark Documentation](#).

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

Description	Port number
spark.ui.port	4041
spark.history.ui.port	18081
spark.blockManager.port	12001
spark.broadcast.port	12501
spark.driver.port	13001
spark.executor.port	13501
spark.filesServer.port	14001
spark.replicaClassServer.port	14501

- **P o r t s        a v a i l a b l e        f o r        t h e        c l u s t e r        m o d e**  
The following ports are available only for the cluster mode.

Description	Port number
spark.master.port	7078
spark.master.rest.port	6067
spark.master.webui.port	8082
spark.worker.port	11001
spark.worker.webui.port	11501

## Troubleshooting the Analytics Profile

The functionality of the API-M Analytics profile is similar to that of WSO2 DAS. Therefore, for detailed instructions to troubleshoot the API-M Analytics profile, see the [WSO2 DAS Troubleshooting Guide](#).

## Updating WSO2 API Manager Analytics

For detailed information about how to apply WUM updates to API Manager Analytics, see the [WSO2 Admin Guide - Updating WSO2 Products](#).

### Best practices

This section covers the best practices to follow when applying WUM updates to API Manager Analytics. The main factors to consider are as follows.

- The following guidelines applicable only if you replace your binary pack of WSO2 API Manager Analytics with a new pack to which the WUM updates are applied. For more information about this method of applying WUM updates, see [WSO2 Admin Guide - Applying Wum Updates Manually](#). If you follow this method, you need to move the WSO2 API Manager Analytics pack that is updated with the WUM updates to the production environment before applying these best practices. For more information, see [WSO2 Admin Guide - Apply WUM Updated Products Using Automation](#).
- These guidelines are not applicable if you apply extracted WUM updates using automation. For more information, see [WSO2 Admin Guide - Apply the Extracted WUM Updates Using Automation](#).

- The unique node ID
- Indexing data and the indexing queue
- Shard allocation

#### *The unique node ID*

The `my-node-id.dat` file is created in the `<APIM-ANALYTICS_HOME>/repository/conf/analytics` directory. If you do not create it, it is automatically created at server start-up. This file contains a unique ID for the Analytics node that helps to identify it. The databases are updated with the entries in this file. Therefore, when you apply WUM updates to nodes, use the unique ID in this file to keep track of the nodes.

#### *Indexing data and the indexing queue*

When WSO2 API Manager Analytics uses Lucene for indexing, the indexing related information is stored in the `<APIM-ANALYTICS_HOME>/repository/data` directory. The data that requires to be indexed goes through a queue that is known as the indexing queue. Once the data is indexed, it is stored in the `index_data` directory.

Before you discard the old DAS instance, you must take a backup of the `<APIM-ANALYTICS_HOME>/repository/data` directory and copy it to the DAS binary pack that is updated with the WUM updates.

#### *Shard allocation*

Information relating to the Shard allocation is maintained in the <APIM-ANALYTICS\_HOME>/repository/conf/analytics/local-shard-allocation-config.conf file. This file stores the shard number along with its state (that can be INIT or NORMAL). The INIT is the initial state. Usually this state cannot be seen from outside. This is because the INIT state changes to the NORMAL state once the server starts. If the indexing node is running, the state of shards should be NORMAL and not INIT. The NORMAL state denotes that the indexing node has started indexing.

Before you discard the old API Manager Analytics instance, you must take a backup of the <APIM-ANALYTICS\_HOME>/repository/conf/analytics/local-shard-allocation-config.conf file, and copy it back to the same location in the Analytics binary pack that is updated with the WUM updates.

## 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](#)
- [Message Flow in the API Manager Gateway](#)
- [Updating WSO2 API Manager](#)
- [Accessing API Manager by Multiple Devices Simultaneously](#)

### Product Profiles

When a WSO2 product starts, it starts all components, features and related artifacts bundled with it. Multi-profile support allows you to run the product on a selected profile so that only the features specific to that profile along with common features start up with the server.

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 can still access the API Publisher web application.

## OSGI Bundle

OSGI bundle is a tightly coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any). In OSGi, a bundle is the primary deployment format. Bundles are applications that are packaged in JARs, and can be installed, started, stopped, updated, and removed.

- [API-M Profiles](#)
- [Starting an API-M profile](#)
- [How multi-profiling works](#)

### API-M Profiles

The following are the different profiles available in WSO2 API Manager.

Profile	Command Option with Profile Name	Description
---------	----------------------------------	-------------

Gateway manager	-Dprofile=gateway-manager	<p>Only starts the components related to the API Gateway.</p> <p>You use this 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.</p>
Gateway worker	-Dprofile=gateway-worker -DworkerNode=true	<p>Only starts the components related to the API Gateway.</p> <p>You use this when the API Gateway acts as a worker node in a cluster. This profile starts the backend features for data processing and communicating with the manager node.</p>
Key Manager	-Dprofile=api-key-manager	Only starts the features relevant to the Key Manager component of the API Manager.
Traffic Manager	-Dprofile=traffic-manager	<p>Only starts the features relevant to the Traffic Manager component of the API Manager.</p> <p>The Traffic Manager helps users to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests.</p>
API Publisher	-Dprofile=api-publisher	Only starts the front end/backend features relevant to the API Publisher.
Developer Portal (API Store)	-Dprofile=api-store	Only starts the front end/backend features relevant to the Developer Portal (API Store).

#### Starting an API-M profile

1. Perform the following configurations on the profile.

Gateway Manager	Gateway Worker	Key Manager	Traffic Manager	Publisher	Store
-----------------	----------------	-------------	-----------------	-----------	-------

When using this profile, make sure you configure the Gateway Manager as described in the [Starting the Gateway Manager](#) section.

When using this profile, make sure you configure the Gateway Worker as described in the [Starting the Gateway Worker](#) section.

Carryout the following configurations on the Key Manager node before starting the Key Manager.

- a. Open the <API-M\_HOME>/repository/conf/axis2/axis2.xml file and remove the configuration section that starts with <transportSender name="ws" class="org.wso2.carbon.websocket.transport.WebsocketTransportSender">.
- b. In the <API-M\_HOME>/repository/conf/api-manager.xml file, set the values of the following elements to false.

```
<ThrottlingConfigurations>
...
<DataPublisher>
<Enabled>true</Enabled>
</DataPublisher>
...
</ThrottlingConfigurations>
```

```
<ThrottlingConfigurations>
...
<JMSConnectionDetails>
<Enabled>true</Enabled>
</JMSConnectionDetails>
...
</ThrottlingConfigurations>
```

When using this profile, make sure you configure the Traffic Manager as described in the [Configuring the Traffic Manager](#) section.

Carryout the following configurations on the Publisher node before starting the Publisher.

- Set the values of the following elements to false in the `<API-M_HOME>/repository/conf/api-manager.xml` file.

```
<ThrottlingConfigurations>
...
<DataPublisher>
<Enabled>true</Enabled>
</DataPublisher>
...
</ThrottlingConfigurations>
```

```
<ThrottlingConfigurations>
...
<JMSConnectionDetails>
<Enabled>true</Enabled>
</JMSConnectionDetails>
...
</ThrottlingConfigurations>
```

- Open the `<API-M_HOME>/repository/conf/axis2/axis2.xml` file and remove the configuration section that starts with `<transportSender name="ws" class="org.wso2.carbon.websocket.transport.WebsocketTransportSender">`.

Carryout the following configurations on the Store (Developer Portal) node before starting the Store.

- Before starting the server, set the values of the following elements to false in the `<API-M_HOME>/rep`

ository/conf/api-manager.xml file.

```
<ThrottlingConfigurations>
...
<DataPublisher>
<Enabled>true</Enabled>
</DataPublisher>
...
</ThrottlingConfigurations>
```

```
<ThrottlingConfigurations>
...
<JMSSConnectionDetails>
<Enabled>true</Enabled>
</JMSSConnectionDetails>
...
</ThrottlingConfigurations>
```

- b. Open the <API-M\_HOME>/repository/conf/axis2/axis2.xml file and remove the configuration section that starts with <transportSender name="ws" class="org.wso2.carbon.websocket.transport.WebsocketTransportSender">.

## 2. Carryout the following configuration changes on **all the profiles with the exception of the Gateway profiles**

- a. Open the axis2.xml file and comment out the following.

```
<transportSender name="ws"
class="org.wso2.carbon.websocket.transport.WebsocketTransportSender">
<parameter name="ws.outflow.dispatch.sequence"
locked="false">outflowDispatchSeq</parameter>
<parameter name="ws.outflow.dispatch.fault.sequence"
locked="false">outflowFaultSeq</parameter>
</transportSender>
```

- b. Delete the WebSocketInboundEndpoint.xml file from the <API-M\_HOME>/repository/deployment/server/synapse-configs/default/inbound-endpoints directory
3. Make sure to keep the following required web apps and remove the unnecessary web apps from each node from the <API-M\_HOME>/repository/deployment/server/webapps directory.

This step is mandatory for the Traffic Manager node, but is optional for all other nodes.

The following are the **web apps required for each node**:

Profile	Required web apps
---------	-------------------

Gateway Manager	am#sample#pizzashack#v1 (only needed if the Pizzashack sample API is used) api#am#admin#v0.11 (only needed if you want to perform administrative tasks through Gateway Manager) authenticationendpoint
Gateway Worker	am#sample#pizzashack#v1 (only needed if the Pizzashack sample API is used) api#am#admin#v0.11 (only needed if you want to perform administrative tasks through Gateway Manager)
Traffic Manager	shindig (shindig web app is used for the WSO2 CEP Dashboard)
Key Manager	authenticationendpoint client-registration#v0.11 oauth2 throttle#data#v1
API Publisher	am#sample#pizzashack#v1 (only needed if the Pizzashack sample API is used) api#am#publisher#v0.11 authenticationendpoint
API Store (Developer Portal)	api#am#store#v0.11 authenticationendpoint

4. Run the required API-M profile.  
Execute one of the following commands based on your OS:

You can start **only one profile** at a time.

OS	Command & Example
Windows	<pre>&lt;PRODUCT_HOME&gt;/bin/wso2server.bat -Dprofile=&lt;preferred-profile&gt; --run</pre> <p>For example to start the API Store execute the following command.</p> <pre>&lt;API-M_HOME&gt;/bin/wso2server.bat -Dprofile=api-store --run</pre>
Linux/Solaris	<pre>sh &lt;PRODUCT_HOME&gt;/bin/wso2server.sh -Dprofile=api-store</pre> <p>For example to start the API Store execute the following command.</p> <pre>sh &lt;API-M_HOME&gt;/bin/wso2server.sh -Dprofile=api-store</pre>

#### 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 the `<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 the `<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 the `<PRODUCT_HOME>/repository/components/default/configuration/org.eclipse.equinox.simpleconfigurator/bundles.info` file by default. This file contains references to all bundles in the `<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.

- Product-specific ports
- Disabling HTTP Transports

### 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

WSO2 Enterprise Integrator (WSO2 EI) uses the following ports to access the management console:

- 9443 - HTTPS servlet transport for the **ESB** runtime (the default URL of the management console is <https://localhost:9443/carbon>)
- 9445 - HTTPS servlet transport for the **EI-Business Process** runtime (the default URL of the management console is <https://localhost:9445/carbon>)
- 9444 - Used for the **EI-Analytics** management console

### LDAP server ports

Provided by default in the WSO2 Carbon platform.

- 10389 - Used in WSO2 products that provide an embedded LDAP server

### KDC ports

- 8000 - Used to expose the Kerberos key distribution center server

### JMX monitoring ports

WSO2 Carbon platform uses TCP ports to monitor a running Carbon instance using a JMX client such as JConsole. By default, JMX is enabled in all products. You can disable it using `<PRODUCT_HOME>/repository/conf/etc/j`

`mx.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](#) | [BPS](#) | [Data Analytics Server](#) | [Complex Event Processor](#) | [Elastic Load Balancer](#) | [ESB](#) | [Enterprise Integrator](#) | [Identity Server](#) | [Message Broker](#) | [Machine Learner](#) | [Storage Server](#) | [Enterprise Mobility Manager](#) | [IoT Server](#)

### API Manager

- 5672 - Used by the internal Message Broker.
- 7611 - Authenticate data published when Thrift data publisher is used for throttling.
- 7612 - Publish Analytics to the API Manager Analytics server.
- 7711 - Port for secure transport when Thrift data publisher is used for throttling.
- 7711 + Port offset of the APIM Analytics Server - Thrift SSL port for secure transport when publishing analytics to the API Manager Analytics server.
- 8280, 8243 - NIO/PT transport ports.
- 9611 - Publish data to the Traffic Manager. Required when binary data publisher for throttling.
- 9711 - Authenticate data published to the Traffic Manager. Required when binary data publisher for throttling.
- 10397 - Thrift client and server ports.
- 9099 - Web Socket ports.

If you change the default API Manager ports with a port offset, most of its ports will be changed automatically according to the offset except a few exceptions described in the [APIM Manager documentation](#).

### BPS

- 2199 - RMI registry port (datasources provider port)

### Data Analytics Server

Given below are the specific ports used by WSO2 DAS.

Ports inherited from WSO2 BAM

WSO2 DAS inherits the following port configurations used in its predecessor, [WSO2 Business Activity Monitor](#)

(BAM).

- 7711 - Thrift SSL port for secure transport, where the client is authenticated to use WSO2 DAS.
- 7611 - Thrift TCP port where WSO2 DAS receives events from clients.

Ports used by the Spark Analytics Engine

The Spark Analytics engine is used in 3 separate modes in WSO2 DAS as follows.

- Local mode
- Cluster mode
- Client mode

Default port configurations for these modes are as follows.

For more information on these ports, go to [Apache Spark Documentation](#).

**• Ports available for all modes**

The following ports are available for all three modes explained above.

Description	Port number
spark.ui.port	4040
spark.history.ui.port	18080
spark.blockManager.port	12000
spark.broadcast.port	12500
spark.driver.port	13000
spark.executor.port	13500
spark.filesServer.port	14000
spark.replicaClassServer.port	14500

**• Ports available for the cluster mode**

The following ports are available only for the cluster mode.

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

### **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

## **Enterprise Integrator**

Integration runtime ports

- 9443 - HTTPS servlet transport (the default URL of the management console is <https://localhost:9443/carbon>)

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

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

EI-Analytics runtime ports

- 9444 - Management console port
- 9161 - Cassandra port on which Thrift listens to clients
- 7712 - Thrift SSL port for secure transport, where the client is authenticated to DAS
- 7612 - Thrift TCP port to receive events from clients to DAS

EI-Business Process runtime ports

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

EI-Broker runtime ports

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

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

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

## **Identity Server**

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

## **Message Broker**

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

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

## **Machine Learner**

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

- 4040 - The default port for Spark UI.

## **Storage Server**

Cassandra:

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

HDFS:

- 54310 - Port used to connect to the default file system.
- 54311 - Port used by the MapRed job tracker
- 50470 - Name node secure HTTP server port
- 50475 - Data node secure HTTP server port
- 50010 - Data node server port for data transferring
- 50075 - Data node HTTP server port
- 50020 - Data node IPC server port

## **Enterprise Mobility Manager**

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

A n d r o i d :

The ports to open are 5228, 5229 and 5230. GCM/FCM typically only uses 5228, but it sometimes uses 5229 and 5230.

GCM/FCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.

iOS:

- 5223 - TCP port used by devices to communicate to APNs servers
- 2195 - TCP port used to send notifications to APNs
- 2196 - TCP port used by the APNs feedback service
- 443 - TCP port used as a fallback on Wi-Fi, only when devices are unable to communicate to APNs on port 5223

The APNs servers use load balancing. The devices will not always connect to the same public IP address for notifications. The entire 17.0.0.0/8 address block is assigned to Apple, so it is best to allow this range in the firewall settings.

API Manager:

The following WSO2 API Manager ports are only applicable to WSO2 EMM 1.1.0 onwards.

- 10397 - Thrift client and server ports
- 8280, 8243 - NIO/PT transport ports

## **IoT Server**

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

Default ports

<b>8243</b>	HTTPS gateway port.
-------------	---------------------

<b>9443</b>	HTTPS port for the core profile.
<b>8280</b>	HTTP gateway port.
<b>9763</b>	HTTP port for the core profile.
<b>1886</b>	Default MQTT port.
<b>9445</b>	HTTPS port for the analytics profile.
<b>9765</b>	HTTP port for the analytics profile.
<b>1039</b>	HTTP port for the analytics profile

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

	<b>Android</b>
<b>5228</b>	The ports to open are 5228, 5229 and 5230. Google Cloud Messaging (GCM) and Firebase Cloud Messaging (FCM) typically only uses 5228, but it sometimes uses 5229 and 5230.
<b>5229</b>	GCM/FCM does not provide specific IPs, so it is recommended to allow the firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.
<b>5230</b>	
	<b>iOS</b>
<b>5223</b>	Transmission Control Protocol (TCP) port used by devices to communicate to APNs servers.
<b>2195</b>	TCP port used to send notifications to APNs.
<b>2196</b>	TCP port used by the APNs feedback service.
<b>443</b>	TCP port used as a fallback on Wi-Fi, only when devices are unable to communicate to APNs on port 5223.  The APNs servers use load balancing. The devices will not always connect to the same public IP address for notifications. The entire 17.0.0.0/8 address block is assigned to Apple, so it is best to allow this range in the firewall settings.

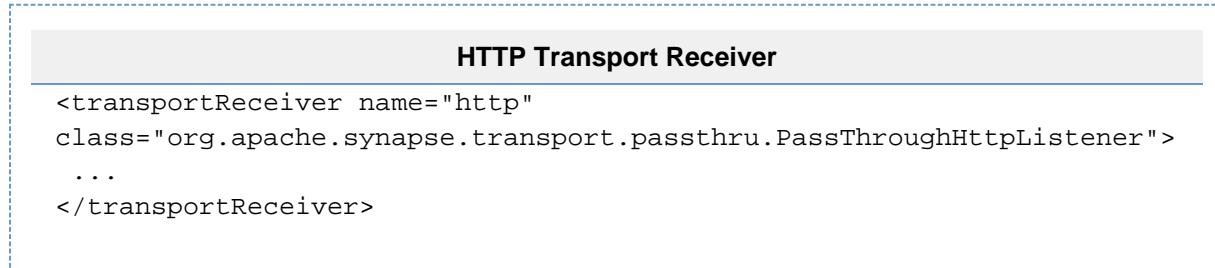
### *Disabling HTTP Transports*

API Manager has two HTTP transports. See below for instructions on how to disable the following:

1. Passthru (API Traffic) Transport
2. Servlet (UI Traffic and Admin service access) Transport

[Disabling Passthru Transport](#)[Disabling Servlet Transport](#)

1. Open the <API-M\_HOME>/repository/conf/axis2/axis2.xml file.
2. Locate the Transport receiver http as shown below:



3. Comment out the http transport receiver section.

1. Open the <API-M\_HOME>/repository/conf/tomcat/catalina-server.xml file.
2. Locate the Connector with the port 9763 as shown below:

#### HTTP Transport Receiver

```
<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
port="9763"
...
/>
```

3. Comment out the http connector section.

The Server needs to be restarted for these changes to be effective.

### 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 port 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 offset in the Ports section of <PRODUCT\_HOME>/repository/conf/carbon.xml. E.g., <Offset>3</Offset>

When you offset the server's port, it automatically changes all ports it uses. However, you are also able to manually adjust the ports for the Thrift client and Thrift server if needed.

### Changing the Thrift client and server ports

The port offset specified earlier in the carbon.xml file affects the ports of the Thrift client and server as well (the default port is 10397). However, since Thrift is run as a separate server within WSO2 servers, it is possible to adjust the ports manually in the <APIM\_HOME>/repository/conf/api-manager.xml file. By default, the <ThriftClientPort> and <ThriftServerPort> elements are commented out. If you want to adjust those ports manually, first uncomment the elements and change the Thrift ports separately. For example,

```
<KeyValidatorClientType>ThriftClient</KeyValidatorClientType>
<ThriftClientPort>10399</ThriftClientPort>
<ThriftClientConnectionTimeOut>10000</ThriftClientConnectionTimeOut>
<ThriftServerPort>10399</ThriftServerPort>
<ThriftServerHost>localhost</ThriftServerHost>
<EnableThriftServer>true</EnableThriftServer>
```

If you specify the Thrift client and server ports manually, the port offset specified in the carbon.xml file has no effect on those two ports and the value that is set manually is used instead.

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. When the API Gateway receives API invocation calls, it contacts the API Key Manager service for verification (given that [caching](#) is not enabled at the Gateway level). 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 the 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 which listens for workflow callbacks. This service configuration can be found at <APIM\_HOME>/repository/deployment/server/synapse-configs/default/proxy-services/WorkflowCallbackService.xml. Open this file and change the port value of the <address uri> accordingly.

For example,

```
<address
uri="https://localhost:9445/store/site/blocks/workflow/workflow-listener/a
jax/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/exceptions occur in the system, the API Manager throws XML-based error responses to the client by default. To change the format of these error responses, you change the relevant XML file in the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/sequences directory. The directory includes multiple XML files, named after the type of errors that occur. You must select the correct file.

For example, to change the message type of authorization errors, open the <APIM\_HOME>/repository/deployment/server/synapse-configs/default/sequences/\_auth\_failure\_handler.xml file and change application/xml to something like application/json .

```
<sequence name="_auth_failure_handler_"
xmlns="http://ws.apache.org/ns/synapse">
<property name="error_message_type" value="application/json"/>
<sequence key="_cors_request_handler_"/>
</sequence>
```

Similarly, to change the error messages of throttling errors (e.g., quota exceeding), change the \_throttle\_out\_handler\_.xml file; resource mismatch errors, the \_resource\_mismatch\_handler\_.xml file, etc.

- API handlers error codes
- Sequences error codes
- Transport error codes
- Custom error messages

Given below are some error codes and their meanings.

#### ***API handlers error codes***

Error code	Error Message	Description	Example
700700	API blocked	This API has been blocked temporarily. Please try again later or contact the system administrators.	Invoke an API which is in the <b>BLOCKED</b> lifecycle state
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.	Invoke an API exceeding the tier limit
900801	Hard limit exceeded	Hard throttle limit has been reached	Invoke an API exceeding the hard throttle limit
900802	Resource level throttle out	Message is throttled out because resource level has exceeded	Sending/Receiving messages beyond authorized resource level
900803	Application level throttle out	Message is throttled out because application level is exceeded	Sending/Receiving messages beyond authorized application level
900900	Unclassified authentication failure	An unspecified error has occurred	Backend service for key validation is not accessible when trying to invoke an API
900901	Invalid credentials	Invalid authentication information provided	Using an older access token after an access token has been renewed.
900902	Missing credentials	No authentication information provided	Accessing an API without <b>Authorization: Bearer</b> header
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 and user accesses tokens. See <a href="#">Access Tokens</a> .	Invoke an API with application token, where the resource only allows application user 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.	Invoke an API resource that is not available
900907	The requested API is temporarily blocked	Happens when the API user is blocked.	Invoke API resource with a subscription that has been blocked by the API publisher
900908	Resource forbidden	The user invoking the API has not been granted access to the required resource.	Invoke an unsubscribed API
900909	The subscription to the API is inactive	The status of the API has changed to an inaccessible/unavailable state.	Invoke an API resource with a subscription that has not yet been approved by the administrator.
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.	Invoke API resource with an access token that is not generated to be used with the resource's scope.

102511	Incomplete payload	The payload sent with the request is too large and the client is unable to keep the connection alive until the payload is completely transferred to the API Gateway	Sending a large PDF file with the POST request
--------	--------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------

The error codes 900903 (Access token expired) and 900904 (Access token inactive) are deprecated from API Manager 1.9.0 onwards. Alternatively, error code 900901 will be sent when the token is invalid or inactive.

#### Sequences error codes

Error code	Description
900901	Production/sandbox key offered to the API with no production/sandbox endpoint
400	Server cannot process the request due to an error in the request sent by the client
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.

The HTTP Status Codes and the corresponding error codes from the error responses are given below.

HTTP Status Code	Error Code
401	900901, 900902, 900905, 900907, 900909
403	900906, 900908, 900910
429	900800
500	900900
503	700700, 900801

#### Transport error codes

Error Code	Detail
101000	Receiver input/output error sending
101001	Receiver input/output error receiving
101500	Sender input/output error sending
101501	Sender input/output error receiving
101503	Connection failed

101504	Connection timed out (no input was detected on this connection over the maximum period of inactivity)
101505	Connection closed
101506	NHTTP protocol violation
101507	Connection canceled
101508	Request to establish new connection timed out
101509	Send abort
101510	Response processing failed

If the HTTP PassThrough transport is used, and a connection-level error occurs, the error code is calculated using the following equation:

$$\text{Error code} = \text{Base error code} + \text{Protocol State}$$

There is a state machine in the transport sender side, where the protocol state changes according to the phase of the message.

Following are the possible protocol states and the description for each:

Protocol State	Description
REQUEST_READY(0)	Connection is at the initial stage ready to send a request
REQUEST_HEAD(1)	Sending the request headers through the connection
REQUEST_BODY(2)	Sending the request body
REQUEST_DONE(3)	Request is completely sent
RESPONSE_HEAD(4)	The connection is reading the response headers
RESPONSE_BODY(5)	The connection is reading the response body
RESPONSE_DONE(6)	The response is completed
CLOSING(7)	The connection is closing
CLOSED(8)	The connection is closed

Since there are several possible protocol states in which a request can time out, you can calculate the error code accordingly using the values in the table above. For example, in a scenario where you send a request and the request is completely sent to the backend, but a timeout happens before the response headers are received, the error code is calculated as follows:

In this scenario, the base error code is CONNECTION\_TIMEOUT(101504) and the protocol state is REQUEST\_DONE(3).

Therefore,

$$\text{Error code} = 101504 + 3 = 101507$$

These Transport error codes are used in [Advanced Configurations of Endpoints](#).

#### **Custom error messages**

To send a custom message with a custom HTTP status code, you execute an additional sequence that can generate a new error message. You then override the message body, HTTP status code and other values.

The following steps demonstrate how to override a throttled-out message's HTTP status code as a custom error message:

1. Go to <APIM\_HOME> /repository/deployment/server/synapse-configs/default/sequences directory and create the file convert.xml as follows.

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="convert">
 <payloadFactory media-type="xml">
 <format>
 <am:fault xmlns:am="http://wso2.org/apimanager">
 <am:code>$1</am:code>
 <am:type>Status report</am:type>
 <am:message>Runtime Error</am:message>
 <am:description>$2</am:description>
 </am:fault>
 </format>
 <args>
 <arg evaluator="xml" expression="$ctx:ERROR_CODE"/>
 <arg evaluator="xml" expression="$ctx:ERROR_MESSAGE"/>
 </args>
 </payloadFactory>
 <property name="RESPONSE" value="true"/>
 <header name="To" action="remove"/>
 <property name="HTTP_SC" value="555" scope="axis2"/>
 <property name="NO_ENTITY_BODY" scope="axis2" action="remove"/>
 <property name="ContentType" scope="axis2" action="remove"/>
 <property name="Authorization" scope="transport" action="remove"/>
 <property name="Access-Control-Allow-Origin" value="*"
 scope="transport"/>
 <property name="Host" scope="transport" action="remove"/>
 <property name="Accept" scope="transport" action="remove"/>
 <property name="X-JWT-Assertion" scope="transport"
 action="remove"/>
 <property name="messageType" value="application/json"
 scope="axis2"/>
 <send/>
</sequence>
```

Alternatively, you can use the **Source View** of the API Management Console as follows to edit the synapse configuration:

- Start the API Manager and log in to the Management Console. (<https://<Server Host>:9443/carbon>).
- Go to **Manager -> Source View**.
- Copy the content of the sequence in convert.xml, paste it as a new sequence in the source view and update it.

2. Check the logs to see whether there are issues in the deployment. If the deployment is successful, you see a message like the following in the system logs:

[2015-04-13 09:17:38,885] INFO - SequenceDeployer Sequence named 'convert' has been deployed from

file : <APIM\_HOME>/repository/deployment/server/synapse-configs/default/sequences/convert.xml

3. Invoke the API until the throttling limit exceeds and the new requests get throttled out.

```
curl -v -H "Authorization: Bearer <Access_Token>"
http://localhost:8280/<API_name>/<context>/<version>
```

4. Note that you get following response:

```
* About to connect() to 127.0.0.1 port 8280 (#0)
* Trying 127.0.0.1...
* Adding handle: conn: 0x17a2db0
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x17a2db0) send_pipe: 1, recv_pipe: 0
* Connected to 127.0.0.1 (127.0.0.1) port 8280 (#0)
> GET /testam/sanjeewa/1.0.0 HTTP/1.1
> User-Agent: curl/7.32.0
> Host: 127.0.0.1:8280
> Accept: */*
> Authorization: Bearer 7f855a7d70aed820a78367c362385c86
>
< HTTP/1.1 555
< Access-Control-Allow-Origin: *
< Content-Type: application/json
< Date: Mon, 13 Apr 2015 05:30:12 GMT
* Server WSO2-PassThrough-HTTP is not blacklisted
< Server: WSO2-PassThrough-HTTP
< Transfer-Encoding: chunked
<
* Connection #0 to host 127.0.0.1 left intact
{ "fault":{ "code":"900800", "type":"Status report", "message": "Runtime Error", "description": "Message throttled out" }}
```

WSO2 API Manager has the following default fault sequences located in <APIM\_HOME> /repository/deployment/server/synapse-configs/default/sequences directory.

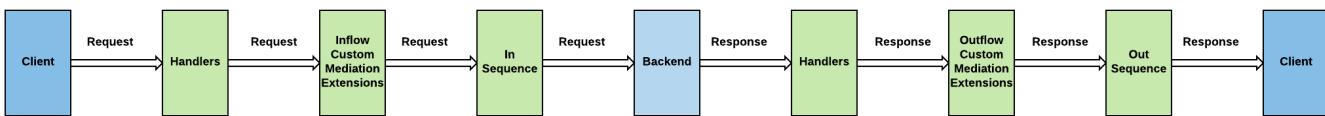
Fault Sequence	Description
fault.xml	This is the primary fault sequence that gets invoked when an error occurs during the execution of an API resources
main.xml	This sequence is called when the endpoint being called does not exist
auth_failure_handler.xml	This sequence is called when an API authentication error is encountered
production_key_error.xml	This sequence is called when a Production key is used to invoke an API that does not have a Production endpoint defined

<code>sandbox_key_error.xml</code>	This sequence is called when a Sandbox key is used to invoke an API that does not have a Sandbox endpoint defined
<code>throttle_out_handler.xml</code>	This sequence is called when a given request to an API gets throttled out
<code>token_fault.xml</code>	This sequence is called when there is an error in invoking the token API
<code>resource_mismatch_handler.xml</code>	This sequence is called when a matching resource cannot be found by the gateway to the corresponding resource being invoked

The default sequences can also be customized as shown in the section above.

## Message Flow in the API Manager Gateway

The Gateway of an API Manager deployment is responsible for the main business functionality of serving API traffic. The following diagram illustrates the message flow in the Gateway at a very high level.



- The handlers
- Mediation extensions
- In sequence and out sequence

### *The handlers*

The handlers are request and response interceptors. The list of API handlers in APIM 2.1.0 are as follows:

- APIMgtLatencyStatsHandler
- CORSRequestHandler
- APIAuthenticationHandler
- ThrottleHandler
- APIMgtUsageHandler
- APIMgtGoogleAnalyticsTrackingHandler
- APIManagerExtensionHandler

Each handler performs a specific task as mentioned in the table below. Note that some handlers are functional both at the inflow and outflow of messages.

Handler	Inflow	Outflow
APIMgtLatencyStatsHandler	Publish request and response latencies, if analytics is enabled	Publish request and response latencies, if analytics is enabled
CORSRequestHandler	Set CORS Headers	Set CORS Headers
APIAuthenticationHandler	Request authentication	N/A
ThrottleHandler	Request throttling	N/A
APIMgtUsageHandler	Publish request data, if analytics is enabled	N/A

APIManagerGoogleAnalyticsTrackingHandler	Publish data to Google Analytics, if Google Analytics is configured	N/A
APIManagerExtensionHandler	Execute custom mediation sequences at inflow	Execute custom mediation sequences at outflow

### Mediation extensions

Mediation extensions are the custom mediation logic that can be executed in either the inflow or the outflow. For more details on how to configure mediation extensions, see [Adding Mediation Extensions](#).

#### In sequence and out sequence

The in sequence and out sequence carry the main business logic of the request flow. The in sequence handles sending the request from the client to the backend, while the out sequence routes the response sent from the backend to the client.

### Updating WSO2 API Manager

WSO2 introduces the [WSO2 Update Manager \(WUM\)](#), which is a command-line utility that allows you to get the latest updates that are available for a particular product release. These updates include the latest bug fixes and security fixes that are released by WSO2 after a particular product version is released. Therefore, you do not need to wait and upgrade to the next product release to get these bug fixes. For information on updating WSO2 API Manager with the latest available patches (issued by WSO2) using the WUM tool, see [Updating WSO2 Products](#) in the WSO2 Administration Guide.

#### Persisting Index data

The indexing related information of WSO2 API Manager is stored in the <DAS\_HOME>/solr/data directory. Once the data is indexed, it is stored in the index directory. Refer [Add Apache Solr-Based Indexing](#) for more information.

Before you discard the old API Manager instance,

You must take a backup of the <APIM\_HOME>/repository/data directory and copy it to the API Manager binary pack in the <APIM\_HOME>/repository/data directory that is updated with the WUM updates.

#### Persisting WSO2CarbonDB

To avoid conflicts that can be occurred in the update process, it is recommended to persist the local H2 databases as well.

Before you discard the old API Manager instance,

Take a backup of <AM\_HOME>/repository/database/WSO2CARBON\_DB.h2.db and replace it to the API Manager binary pack in the <AM\_HOME>/repository/database directory that is updated with the WUM updates.

If you are using the existing local H2 database for WSO2MetricsDB as well,

Take a backup of <AM\_HOME>/repository/database/WSO2METRICS\_DB.h2.db and replace it to the API Manager binary pack in the <AM\_HOME>/repository/database directory that is updated with the WUM updates.

For more information on run time and configuration artefact directories of API Manager refer [Common Runtime and Configuration Artifacts](#).

Refer [Updating WSO2 API Manager Analytics](#) in order to update WSO2 API Manager Analytics binary distribution.

## Accessing API Manager by Multiple Devices Simultaneously

When there are many users who use production deployment setups, accessing API Manager by multiple devices is more important. According to the architecture, if we logged out from one device and revoke the access token, then all the calls made with that token thereafter will get authentication failures. In this case Applications should be smart enough to detect that authentication failure and should request for a new access token.

This will be a guide for you if you create client applications having API Manager underlying. And note that, you need to use [Password Grant](#) type in this scenario

- [Issue in having multiple access tokens](#)
- [Recommended Solution](#)
- [How this should work](#)

### ***Issue in having multiple access tokens***

Once user log in to the application, the user may need to provide username and password. We can use that information (username and password) and consumer key, consumer secret pair to receive new token once the authentication failure is detected. We should handle this from client application side. If we allowed users to have multiple tokens at the same time, that will cause security related issues and finally users will end up with thousands of tokens that the user cannot even maintain. And also those affects to the usage of metering and statistics.

### ***Recommended Solution***

The recommended solution for this issue is having only one active user token at a given time. We need to make the client application aware about error responses sent from the API Manager Gateway. And use the refresh token Approach. When you request a user token you will get refresh token along with the token response, so that you can use that for refreshing the access token.

### ***How this should work***

Let's assume that same user is logged in to WSO2 API Manager from desktop and tablet. At that time client should provide username and password both when they are login into desktop and tablet apps. Then you can generate token request with the username - password and consumer key - consumer secret pair. So this request will be kept in memory until the user close or logout from application (We do not persist this data to anywhere so that there is no security issue.).

Then, when the user log out from the desktop or the application on the desktop, it decides to refresh the OAuth token first, and the user will be prompted to enter their username and password on the tablet, since the tablet has revoked or inactivated the OAuth token. But in here, you should not prompt username and password because the client is already provided them and you have the token request in memory. Once we detect authentication failure from the tablet, it will immediately send token generation request and get new token. Hence, the user will not aware about what happen underline.

## Developer Guide

This section provides instructions and information regarding the various developer functions of WSO2 API Manager for users. This may include methods of working with the available administration and user features available in the API Manager. The topics in this section are listed as follows:

- [Working with the Source Code](#)
- [Java Documentation](#)
- [WSO2 Admin Services](#)
- [Product APIs](#)
- [Working with Audit Logs](#)
- [Enabling Authentication Session Persistence](#)
- [Enabling Monetization of APIs](#)
- [Using the Registry REST API](#)

## Working with the Source Code

The source code of all WSO2 products as well as the scripts that are used for building WSO2 products are maintained in GitHub repositories. If you are a developer, you can easily clone the source code from these Git repositories, and if required, you can do modifications and build a customized product on your own. For more information, see the following:

- [WSO2 GitHub Repositories](#)
- [Using Maven to Build WSO2 Products](#)
- [Contributing to the Code Base](#)

## Java Documentation

The following Java documentation describes all the classes, interfaces, and methods of the API Manager, which you can use to create custom classes: <http://product-dist.wso2.com/javadocs/api-manager/2.1.0/>

## 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:

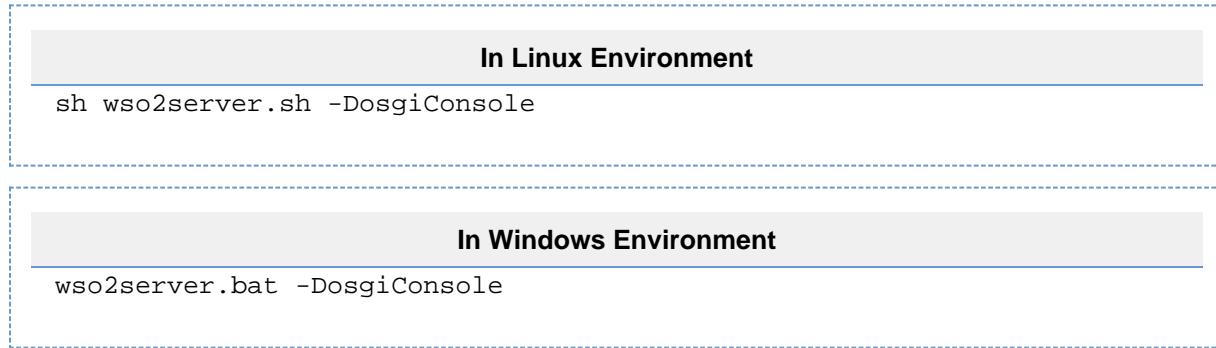
- Service component: provides the actual service
- UI component: provides the Web user interface to the service
- Service stub: provides the interface to invoke the service generated from the service WSDL

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 using the [OSGi](#) console.

1. Set the <HideAdminServiceWSDLs> element to false in the <PRODUCT\_HOME>/repository/conf/carbon.xml file.
2. Go to <PRODUCT\_HOME>/bin/ folder and start the WSO2 product as follows,



3. When the server is started, hit the enter/return key several times to get the OSGI shell in the console.
4. In the OSGI shell, type: osgi> listAdminServices
5. The list of admin services of your product are listed. For example:

```

osgi> listAdminServices
Admin services deployed on this server:
1. ModuleAdminService, ModuleAdminService, https://mgt.esb.wso2.com:8243/services/ModuleAdminService
2. EventStatisticsAdminService, EventStatisticsAdminService, https://mgt.esb.wso2.com:8243/services/EventStatisticsAdminService
3. ThemeMgtService, ThemeMgtService, https://mgt.esb.wso2.com:8243/services/ThemeMgtService
4. RemoteUserRealmService, RemoteUserRealmService, https://mgt.esb.wso2.com:8243/services/RemoteUserRealmService
5. Document, null, https://mgt.esb.wso2.com:8243/services/Document
6. ListMetadataService, ListMetadataService, https://mgt.esb.wso2.com:8243/services/ListMetadataService
7. PropertiesAdminService, PropertiesAdminService, https://mgt.esb.wso2.com:8243/services/PropertiesAdminService
8. AndesEventAdminService, AndesEventAdminService, https://mgt.esb.wso2.com:8243/services/AndesEventAdminService
9. RemoteUserStoreManagerService, RemoteUserStoreManagerService, https://mgt.esb.wso2.com:8243/services/RemoteUserStoreManagerService
10. LogViewer, LogViewer, https://mgt.esb.wso2.com:8243/services/LogViewer
11. SearchAdminService, SearchAdminService, https://mgt.esb.wso2.com:8243/services/SearchAdminService
12. ws-xacml, https://mgt.esb.wso2.com:8243/services/ws-xacml
13. CustomMeteringService, CustomMeteringService, https://mgt.esb.wso2.com:8243/services/CustomMeteringService
14. UserAdmin, UserAdmin, https://mgt.esb.wso2.com:8243/services/UserAdmin
15. UserStoreConfigAdminService, UserStoreConfigAdminService, https://mgt.esb.wso2.com:8243/services/UserStoreConfigAdminService
16. TaskAdmin, TaskAdmin, https://mgt.esb.wso2.com:8243/services/TaskAdmin
17. RegistryCacheInvalidationService, RegistryCacheInvalidationService, https://mgt.esb.wso2.com:8243/services/RegistryCacheInvalidationService
18. ServerAdmin, ServerAdmin, https://mgt.esb.wso2.com:8243/services/ServerAdmin
19. RelationAdminService, RelationAdminService, https://mgt.esb.wso2.com:8243/services/RelationAdminService
20. ServiceAdmin, ServiceAdmin, https://mgt.esb.wso2.com:8243/services/ServiceAdmin
21. ConfigServiceAdmin, ConfigServiceAdmin, https://mgt.esb.wso2.com:8243/services/ConfigServiceAdmin
22. WebappAdmin, WebappAdmin, https://mgt.esb.wso2.com:8243/services/WebappAdmin
23. EventPublisherAdminService, EventPublisherAdminService, https://mgt.esb.wso2.com:8243/services/EventPublisherAdminService
24. EventBrokerService, EventBrokerService, https://mgt.esb.wso2.com:8243/services/EventBrokerService
25. StatisticsAdmin, StatisticsAdmin, https://mgt.esb.wso2.com:8243/services/StatisticsAdmin
26. KeyStoreAdminService, KeyStoreAdminService, https://mgt.esb.wso2.com:8243/services/KeyStoreAdminService
27. MessageProcessorAdminService, MessageProcessorAdminService, https://mgt.esb.wso2.com:8243/services/MessageProcessorAdminService
28. OAuth2TokenValidationService, OAuth2TokenValidationService, https://mgt.esb.wso2.com:8243/services/OAuth2TokenValidationService
29. APIGatewayAdmin, APIGatewayAdmin, https://mgt.esb.wso2.com:8243/services/APIGatewayAdmin
30. APIKeyMgtProviderService, APIKeyMgtProviderService, https://mgt.esb.wso2.com:8243/services/APIKeyMgtProviderService
31. ProvisioningAdminService, ProvisioningAdminService, https://mgt.esb.wso2.com:8243/services/ProvisioningAdminService
32. MediationSecurityAdminService, MediationSecurityAdminService, https://mgt.esb.wso2.com:8243/services/MediationSecurityAdminService
33. EventSimulatorAdminService, EventSimulatorAdminService, https://mgt.esb.wso2.com:8243/services/EventSimulatorAdminService
34. IdentityApplicationManagementService, IdentityApplicationManagementService, https://mgt.esb.wso2.com:8243/services/IdentityApplicationManagementService

```

6. To see the service contract of an admin service, select the admin service's URL and then paste it in your browser with **?wsdl** at the end. For example:  
<https://localhost:9443/services/RemoteUserStoreManagerService?wsdl>

In products like WSO2 ESB and WSO2 API Manager, the port is 8243 (assuming 0 port offset). However, you should be accessing the Admin Services via the management console port, which is 9443 when there is no port offset.

7. Note that the admin service's URL appears as follows in the list you discovered in step 6:

RemoteUserStoreManagerService, RemoteUserStoreManagerService,  
<https://<host IP>:9443/services/RemoteUserStoreManagerService/>

After discovering admin service you can restart the server without -DosgiConsole

### **Invoking an admin service**

Admin services are secured using common types of security protocols such as HTTP basic authentication, WS-Security username token, and session based authentication to prevent anonymous invocations. For example, the UserAdmin Web service is secured with the HTTP basic authentication. To invoke a service, you do the following:

1. Authenticate yourself and get the session cookie.
2. Generate the client stubs to access the back-end Web services.

To generate the stubs, you can write your own client program using the Axis2 client API or use an existing tool like [SoapUI](#) (4.5.1 or later) or wsdl2java.

The wsdl2java tool, which comes with WSO2 products by default hides all the complexity and presents you with a proxy to the back-end service. The stub generation happens during the project build process within the Maven POM files. It uses the Maven ant run plug-in to execute the wsdl2java tool.

You can also use the Java client program given [here](#) to invoke admin services. All dependency JAR files that you need to run this client are found in the /lib directory.

---

## Authenticate the user

The example code below authenticates the user and gets the session cookie:

```

import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
import org.wso2.carbon.authenticator.stub.AuthenticationAdminStub;
import
org.wso2.carbon.authenticator.stub.LoginAuthenticationExceptionException;
import
org.wso2.carbon.authenticator.stub.LogoutAuthenticationExceptionException;

import org.apache.axis2.context.ServiceContext;
import java.rmi.RemoteException;

public class LoginAdminServiceClient {
 private final String serviceName = "AuthenticationAdmin";
 private AuthenticationAdminStub authenticationAdminStub;
 private String endPoint;

 public LoginAdminServiceClient(String backEndUrl) throws AxisFault {
 this.endPoint = backEndUrl + "/services/" + serviceName;
 authenticationAdminStub = new AuthenticationAdminStub(endPoint);
 }

 public String authenticate(String userName, String password) throws
RemoteException,
LoginAuthenticationExceptionException {

 String sessionCookie = null;

 if (authenticationAdminStub.login(userName, password, "localhost"))
{
 System.out.println("Login Successful");

 ServiceContext serviceContext = authenticationAdminStub.

_getServiceClient().getLastOperationContext().getServiceContext();
 sessionCookie = (String)
serviceContext.getProperty(HTTPConstants.COOKIE_STRING);
 System.out.println(sessionCookie);
 }

 return sessionCookie;
 }

 public void logOut() throws RemoteException,
LogoutAuthenticationExceptionException {
 authenticationAdminStub.logout();
 }
}

```

## Generate the client stubs

After authenticating the user, give the retrieved admin cookie with the service endpoint URL as shown in the sample below. The Remote user management service name is RemoteUserStoreManagerService. You can find its URL (e.g., <https://localhost:9443/services/RemoteUserStoreManagerService>) 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.um.ws.api.stub.RemoteUserStoreManagerServiceStub;
import
org.wso2.carbon.um.ws.api.stub.RemoteUserStoreManagerServiceUserStoreExceptionException;

import java.rmi.RemoteException;

public class RemoteUserStoreServiceAdminClient {

 private final String serviceName = "RemoteUserStoreManagerService";
 private RemoteUserStoreManagerServiceStub
remoteUserStoreManagerServiceStub;
 private String endPoint;

 public RemoteUserStoreServiceAdminClient(String backEndUrl, String
sessionCookie) throws AxisFault {
 this.endPoint = backEndUrl + "/services/" + serviceName;
 remoteUserStoreManagerServiceStub = new
RemoteUserStoreManagerServiceStub(endPoint);
 //Authenticate Your stub from sessionCookie
 ServiceClient serviceClient;
 Options option;

 serviceClient =
remoteUserStoreManagerServiceStub._getServiceClient();
 option = serviceClient.getOptions();
 option.setManageSession(true);

 option.setProperty(org.apache.axis2.transport.http.HTTPConstants.COOKIE_ST
RING, sessionCookie);
 }

 public String[] listUsers()
 throws RemoteException,
RemoteUserStoreManagerServiceUserStoreExceptionException {
 return remoteUserStoreManagerServiceStub.listUsers("*", 100);
 }
}

```

The following sample code lists the users in the APIM server:

```
import org.apache.axis2.AxisFault;
```

```

import
org.wso2.carbon.authenticator.stub.LoginAuthenticationException;
import
org.wso2.carbon.authenticator.stub.LogoutAuthenticationException;
import
org.wso2.carbon.um.ws.api.stub.RemoteUserStoreManagerServiceUserStoreException;
import
java.rmi.RemoteException;

public class AdminServiceClientManager {
 public static void main (String[] args) {

 System.setProperty("javax.net.ssl.trustStore",
"/repository/resources/security/client-truststore.jks");
 System.setProperty("javax.net.ssl.trustStorePassword",
"wso2carbon");

 try {
 LoginAdminServiceClient loginAdminServiceClient =
 new LoginAdminServiceClient("https://localhost:9443");
 String sessionId =
loginAdminServiceClient.authenticate("admin", "admin");
 RemoteUserStoreServiceAdminClient
remoteUserStoreServiceAdminClient = new

RemoteUserStoreServiceAdminClient("https://localhost:9443", sessionId);
 String[] users = remoteUserStoreServiceAdminClient.listUsers();

 if(users != null){
 System.out.println("Listing user names of Carbon
server..... ");
 for(String user : users){
 System.out.println("User Name : " + user);
 }
 }

 loginAdminServiceClient.logOut();

 } catch (AxisFault axisFault) {
 axisFault.printStackTrace();
 throw new RuntimeException(axisFault);
 } catch (RemoteException e) {
 e.printStackTrace();
 throw new RuntimeException(e);
 } catch (LoginAuthenticationException e) {
 e.printStackTrace();
 throw new RuntimeException(e);
 } catch (RemoteUserStoreManagerServiceUserStoreException
e) {
 e.printStackTrace();
 throw new RuntimeException(e);
 }
 }
}

```

```

 } catch (LogoutAuthenticationException e) {
 e.printStackTrace();
 throw new RuntimeException(e);
 }

 }
}

```

The complete maven project can be found at: [org.wso2.carbon.sample.admin.service.invoker.zip](#)

## Product APIs

The following topics consist of a list of RESTful APIs, Token APIs, and deprecated APIs exposed by WSO2 API Manager.

- [RESTful APIs](#)
- [Token API](#)
- [Deprecated APIs](#)

- RESTful APIs - These are the APIs exposed from the API Publisher, API Store, and Admin Portal, which can be used to create and manage APIs.
- Token API - These APIs can be used to generate and renew user and application access tokens.
- Deprecated APIs - These APIs are deprecated and will be unsupported in a future release.

## RESTful APIs

The following topics list the APIs exposed from the API Publisher, API Store, and Admin Portal, which you can use to create and manage APIs. You can consume APIs directly through their UIs or, an external REST client like cURL, or the [WSO2 REST client](#).

- [Publisher APIs](#)
- [Store APIs](#)
- [Admin APIs](#)

## 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.

The topics below explain how to generate access tokens and authorize them. WSO2 API Manager supports the following common [authorization grant types](#) and you can also define additional types.

- [Kerberos OAuth2 Grant](#)
- [Refresh Token Grant](#)
- [Authorization Code Grant](#)
- [NTLM Grant](#)
- [Password Grant](#)
- [SAML Extension Grant](#)
- [Client Credentials Grant](#)
- [Implicit Grant](#)

For more information on Token APIs, see the following topics.

- [Revoking access tokens](#)
- [Configuring the token expiration time](#)
- [Token Persistence](#)

## 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 the Revoke API using a utility like cURL. The Revoke API's endpoint URL is <http://localhost:8280/revoke>.

You can also revoke refresh tokens. For more information, see [Revoking a refresh token](#).

You can use any of the following cURL command options to revoke an access token:

### Option 1

The parameters required to invoke the following API are as follows:

- token - The token to be revoked
- <base64 encoded (clientId:clientSecret)> - Use a base64 encoder (e.g., <https://www.base64encode.org/>) to encode your client ID and client secret using the following format: <clientId>:<clientSecret> Thereafter, enter the encoded value for this parameter.

#### FormatExampleResponse

```
curl -k -v -d "token=<REFRESH_TOKEN_TO_BE_REVOKED>" -H "Authorization: Basic <base64 encoded (clientId:clientSecret)>" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/revoke
```

```
curl -k -v -d "token=a0d210c7a3de7d548e03f1986e9a5c39" -H "Authorization: Basic OVRRNVJLZWfhVGZGeUpRSkRzam9aZmp4UkhjYTpDZnJ3ZXRa19ZOTdSSzFTZWlWQWxlaXdVVmth" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/revoke
```

You receive an empty response with the HTTP status as 200. The following HTTP headers are returned:

Note that if you use an invalid access token, you still receive an empty response with the HTTP status as 200 but only the following HTTP headers are returned:

```
Revokedaccesstoken: a0d210c7a3de7d548e03f1986e9a5c39
Authorizeduser: admin@carbon.super
Revokedrefreshtoken: 5e87a8235cd4d066e15c4c989f5ecf94
Content-Type: text/html
Pragma: no-cache
Cache-Control: no-store
Date: Tue, 23 Aug 2016 19:28:52 GMT
Transfer-Encoding: chunked
```

```
Content-Type: text/html
Pragma: no-cache
Cache-Control: no-store
Date: Tue, 23 Aug 2016 19:31:45 GMT
Transfer-Encoding: chunked
```

## Option 2

The parameters required to invoke the following API are as follows:

- token - The token to be revoked
- <base64 encoded (clientId:clientSecret)> - Use a base64 encoder (e.g., <https://www.base64encode.org/>) to encode your client ID and client secret using the following format: <clientId>:<clientSecret> Thereafter, enter the encoded value for this parameter.
- token\_type\_hint= This parameter is optional. If you do not specify this parameter, then WSO2 Identity Server (WSO2 IS) will search in both key spaces (access and refresh) and if it finds a matching token then it will be revoked. Therefore, if this parameter is not specified the token revocation process takes longer. However, if you specify this parameter then WSO2 IS only searches in the respective token key space, hence the token revocation process is much faster.

FormatExample

```
curl -k -v -d
"token=<REFRESH_TOKEN_TO_BE_REVOKED>&token_type_hint=<access_token_or_refresh_token>" -H "Authorization: Basic <base64 encoded (clientId:clientSecret)>" -H Content-Type: application/x-www-form-urlencoded https://localhost:8243/revoke
```

```
curl -k -v -d
"token=1d18ec65-6151-3499-9352-68afe64299c3&token_type_hint=access_token"
-H "Authorization: Basic OVRRNVJLZWfhVGZGeUpRSkRzam9aZmp4UkhjYTpDZnJ3ZXRa19ZOTdSSzFTZWlWQWxlaXdVVm
th" -H "Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/revoke
```

## 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 the <APIM\_HOME>/repository/conf/identity/identity.xml file.

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 the <APIM\_HOME>/repository/conf/identity/identity.xml file.

**Tip:** Ideally, the time stamp skew should not be larger than the token's life span. We recommend you to set it to zero if the nodes in your cluster are synchronized.

When a user access token expires, the user can try regenerating the token.

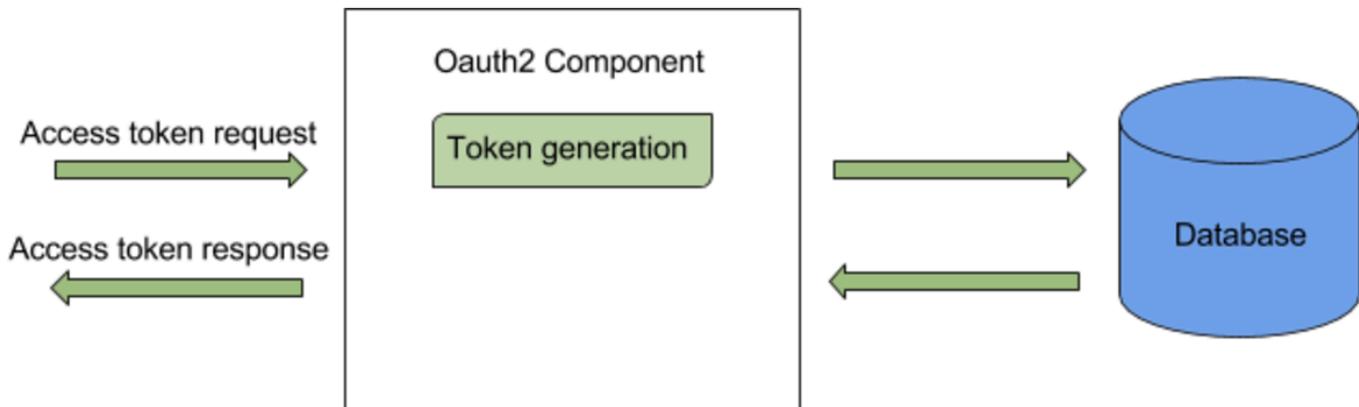
### Token Persistence

The OAuth2 component in WSO2 API Manager (WSO2 API-M) has two implementations that you can use to handle token persistence in the database, which are namely synchronous and asynchronous token persistence. The following sections guide you through the difference between these two approaches and how to configure them in a production environment.

- [Synchronous token persistence \(When PoolSize = 0\)](#)
- [Asynchronous token persistence \(When PoolSize > 0\)](#)

The Synchronous or Asynchronous behavior is governed by the `PoolSize` property under `SessionDataPersist` element in the `identity.xml` file.

### **Synchronous token persistence (When PoolSize = 0)**



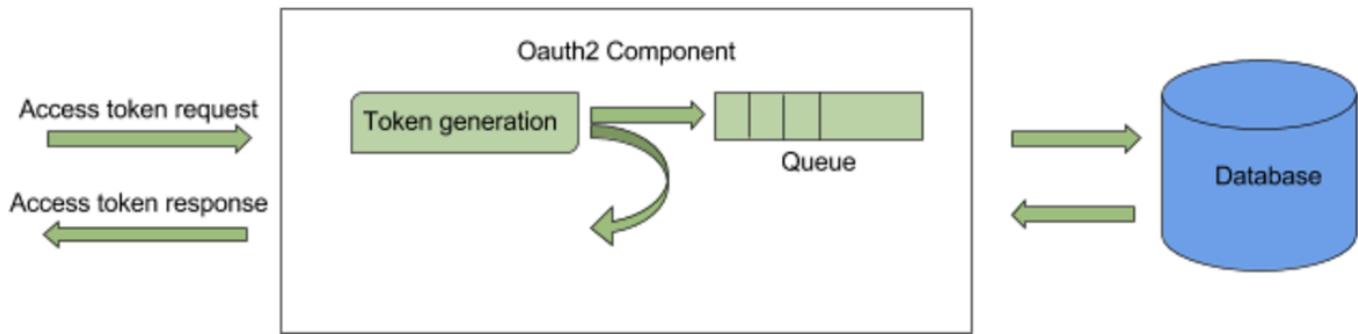
The flow of synchronous token persistence is as follows:

1. The client sends an access token request.
2. The OAuth2 component in WSO2 API-M checks for an existing active access token for the given `client/user scope`. It first checks the cache and if an active token is not found, it then checks the database.
3. If an active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component creates a new access token and persists it in the database using the same thread. Once it is persisted, the new token is returned to the client.

## Enabling synchronous token persistence

To enable synchronous token persistence, follow the steps in the [Enabling Authentication Session Persistence tutorial](#) and **set the `<poolsize>` property to 0**.

### **Asynchronous token persistence (When PoolSize > 0)**



The flow of asynchronous token persistence is as follows:

1. The client sends an access token request.
2. The OAuth2 component in WSO2 API-M checks for an existing active access token for the given client / user / scope. It first checks the cache and if an active token is not found, it then checks the database.
3. If an active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component creates a new access token and adds it to a persisting queue.
5. Once the token is added to the queue, the token is returned to the client.
6. Thereafter, there are background threads that consume the above queue and they in-turn will persist these tokens to the DB.

## Enabling asynchronous token persistence

To enable asynchronous token persistence, follow the steps in the [Enabling Authentication Session Persistence](#) tutorial and **set the <poolszie> property to a value higher than 0**. The value provided for the <poolszie> property determines the number of threads in the thread pool that are used to consume the token persisting queue.

The main difference between synchronous and asynchronous token persistence is that the OAuth2 component in the synchronous token persistence implementation waits for the access token to be persisted in the database before returning it to the client.

### Recovery flow for token persistence

This section explains the recovery flow triggered in WSO2 API Manager for exceptional cases that may occur in a production environment caused by the client application mishandling the CON\_APP\_KEY constraint that is explained below.

- CON\_APP\_KEY constraint
- Asynchronous token persistence
- Synchronous token persistence

CON\_APP\_KEY constraint

CONSTRAINT	CON_APP_KEY	UNIQUE	(CONSUMER_KEY ,
AUTHZ_USER , USER_TYPE , TOKEN_STATE , TOKEN_STATE_ID , TOKEN_SCOPE)			)

As seen in the code snippet above for a given set of consumer key, user, and scope values, there can be only one ACTIVE access token. The CON\_APP\_KEY constraint in the IDN\_OAUTH2\_ACCESS\_TOKEN table enforces this by allowing only one active access token for a given set of consumer key, user, and scope values. This constraint may be violated in a scenario where two or more identical token requests come from the same application.

The above scenario is unlikely, because in practice an application is usually designed to handle this

situation using scopes, or in the case of a multi-threaded client, there is usually a separate thread to acquire access tokens so that other threads can retrieve from it.

Asynchronous token persistence

## Flow

For instance, if the violation mentioned above occurs with two nodes of a cluster receiving identical access token requests, the flow of the asynchronous token persistence is as follows:

1. The client sends an access token request.
2. The OAuth2 component in both nodes of WSO2 API-M checks for an existing active access token for the given client/user/scope. Both nodes first check the cache and if an active token is not found, it checks the database.
3. If an existing active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component in **both nodes** creates a new access token and adds it to the persisting queue.
5. After adding it to the queue, the access token is returned to the client.
6. However, the background threads that consume the persisting queue in both servers (nodes) attempt to persist the token to the database. One of the servers succeed and successfully persist the access token to the database, and the other server receives an error due to violation of the CON\_APP\_KEY constraint. The violation is due to the fact that the same access token was already persisted by the first server in the cluster and is currently active.

## Recovery flow

To handle this situation, WSO2 API Manager has a recovery flow for token persistence that does the following:

- As both access tokens were returned to the client, there must be a database entry for both tokens.
- As the access token that the second node is attempting to persist is not allowed due to the violation, the recovery flow takes the latest entry in the database, which is the active access token persisted by the first node, and marks it as INACTIVE.
- The access token that is received from the second node is now saved as an ACTIVE access token in the database. Therefore, one of the access tokens returned to the client is an INACTIVE token.

**Tip:** If the client application is not designed to handle the CONN\_APP\_KEY constraint violation using scopes, you can avoid the situation described above and avoid any invalid tokens by using synchronous token persistence. To do this, set the <PoolSize> property in the <API-M\_HOME>/repository/conf/identity/identity.xml file to 0.

Synchronous token persistence

## Flow

The flow of the synchronous token persistence when receiving two identical access token requests is as follows:

1. The client sends an access token request.
2. The OAuth2 component in both nodes of WSO2 API-M checks for an existing active access token for the given client/user/scope. Both nodes first check the cache and if an active token is not found, the database is checked.
3. If an existing active access token is found, the token is returned to the client.
4. Alternatively, if an existing access token is not found, the OAuth2 component in **both nodes** creates a new access token and persists the access token to the database using the same thread.
5. Either one of the nodes persist the token successfully and returns it to the client, but the other node receives an error due to violation of the CON\_APP\_KEY constraint.

## Recovery flow

The process flow now moves on to the recovery flow described above in order to handle the CON\_APP\_KEY constraint violation and is executed as follows:

- As the same thread is being used, the OAuth2 component in the second node checks the database again for an ACTIVE access token.
- As there is now an ACTIVE token, which was persisted by the first node, the second node now returns the access token persisted by the first node to the client.
- Both access token requests receive the same access token.

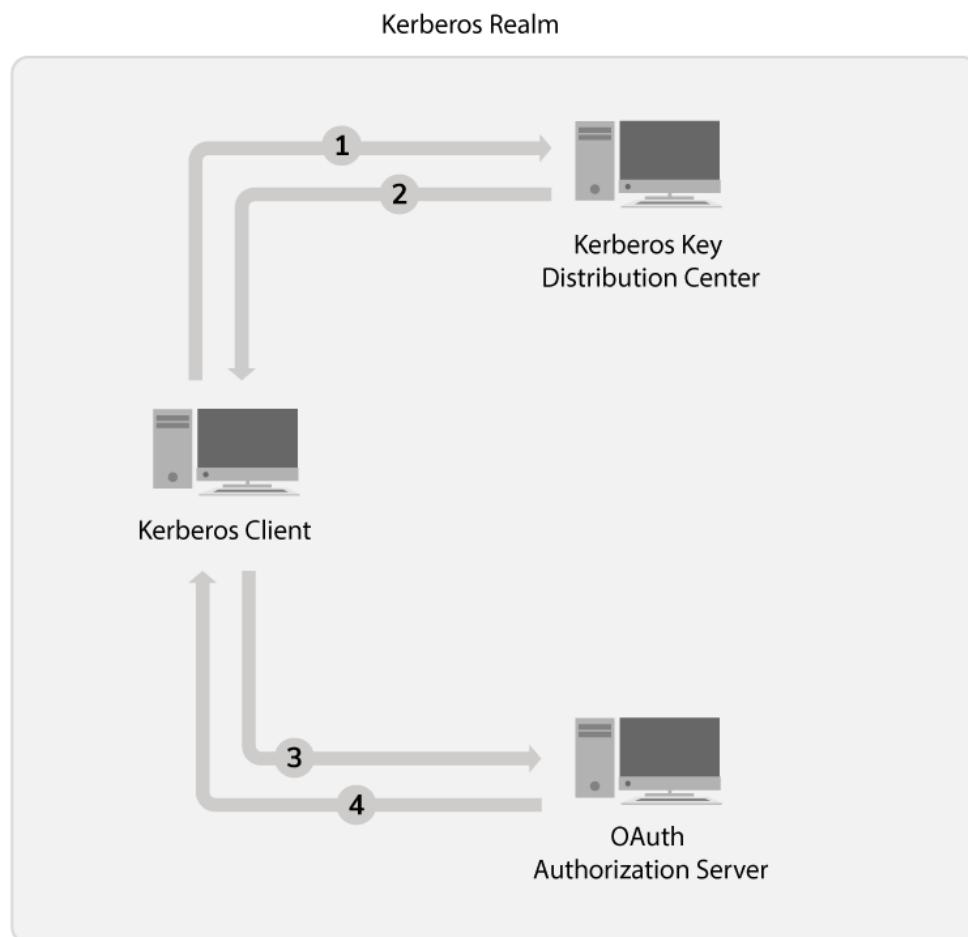
### Kerberos OAuth2 Grant

Kerberos is a security protocol that has support built into various operating systems and open-source distributions (e.g., Ubuntu, Windows, RedHat, Open Solaris, etc). In addition, a majority of browsers support some Kerberos functions as well. As WSO2 API Manager uses the OAuth 2.0 protocol, the Kerberos OAuth2 grant type allows organizations to exchange a Kerberos ticket for an OAuth 2.0 token. Thereby, allowing organizations to re-use their existing Kerberos infrastructure, while easier adopting OAuth 2.0 within these organizations.

- [Kerberos OAuth2 grant flow](#)
- [Configuring Kerberos Grant with API Manager](#)

### *Kerberos OAuth2 grant flow*

The following section describes the flow involved in exchanging a Kerberos ticket for an OAuth2 token.



1. The Kerberos client requests the Kerberos Service Ticket from the Kerberos Key Distribution Center (KDC) to invoke the service.  
The Kerberos Key Distribution Center can be any Kerberos Server.
2. The Kerberos Key Distribution Center sends a response with the Kerberos Service Ticket.  
If the client and the requested service is valid, the Key Distribution Center (KDC) sends a Kerberos ticket

encrypted with the service owners private key. The API handles the exchanging of the Ticket Granting Ticket (TGT), Service Granting Ticket (SGT), and all other low level Kerberos details.

3. The Kerberos client requests the OAuth2 token.  
The message format of the OAuth2 token request should be as follows:

#### cURL Request Format

You can use one of the following two cURL commands to request for the OAuth2 token.

```
curl -v -X POST -H "Authorization: Basic
<base64-encoded-client-id>:<client-secret-value>" -k -d
"grant_type=kerberos&kerberos_realm=<kerberos-realm>&kerberos_token=<
kerberos-token>&scope=<scope>" -H
"Content-Type:application/x-www-form-urlencoded"
https://localhost:8243/token
```

```
curl -u <client-id>:<client-secret> -k -d
"grant_type=kerberos&kerberos_realm=<kerberos-realm>&kerberos_token=<
kerberos-token>&scope=<scope>" -H
"Content-Type:application/x-www-form-urlencoded"
https://localhost:8243/token
```

The “scope=my\_scope” is an optional parameter that you can add to the string in the token request body.

#### Example

```
grant_type=kerberos&scope=my_scope&kerberos_realm=example.com&kerbero
s_token=YII1...
```

#### Example

```
POST /token HTTP/1.1
Host: idp.example.com:8243
Content-Type: application/x-www-form-urlencoded
Authorization: Basic
MW91TDJmTzZTeGxmRDJMRHcxMjVjVG8wdlFrYTp1VUV0bTg5dFk2UVp1WlVtcVpmTDkyQ
kRGZUFh
grant_type=kerberos&kerberos_realm=example.com&kerberos_token=YII1...
```

4. The Kerberos client receives the OAuth2 token.  
The Kerberos Grant validates the received token with the provided Identity Provider (IDP) credentials and if it is a valid token, it issues an OAuth2 token to the client.

### Example

```
{
 "access_token": "636ce45f-c7f6-3a95-907f-d1f8aca28403",
 "refresh_token": "831271d9-16ba-3bad-af18-b9f6592a8677",
 "scope": "my_scope",
 "token_type": "Bearer",
 "expires_in": 521
}
```

### **Configuring Kerberos Grant with API Manager**

Follow the instructions below to configure Kerberos Grant with WSO2 API Manager:

1. Download the Keberos-grant JAR ([kerberos-grant-1.0.0.jar](#)).
2. Copy the JAR into the <API-M\_HOME>/repository/components/lib directory.
3. Add following entry under <SupportedGrantTypes> in the <API-M\_HOME>/repository/conf/identity/identity.xml file.

```
<SupportedGrantType>
 <GrantTypeName>kerberos</GrantTypeName>

 <GrantTypeHandlerImplClass>org.wso2.carbon.identity.oauth2.grant.kerberos.ExtendedKerberosGrant</GrantTypeHandlerImplClass>

 <GrantTypeValidatorImplClass>org.wso2.carbon.identity.oauth2.grant.kerberos.KerberosGrantValidator</GrantTypeValidatorImplClass>
</SupportedGrantType>
```

4. Create a file named jaas.conf in the <API-M\_HOME>/repository/conf/identity directory with the following content.

```
Server {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=false
 storeKey=true
 useTicketCache=false
 isInitiator=false;
};

Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useTicketCache=false;
};
```

5. Copy the following JARs into the <API-M\_HOME>/repository/components/dropins directory.
  - [org.wso2.carbon.identity.application.authenticator.iwa-5.3.0.jar](#)
  - [org.wso2.carbon.identity.idp.metadata.saml2\\_1.0.1.jar](#)
6. Configure OAuth2 for your client application with the Kerberos grant type.
  - a. Start the WSO2 API-M server by navigating to the <API-M\_HOME>/bin directory in your console and running one of the following scripts based on your OS.

- On Windows: `wso2server.bat --run`
  - On Linux/Mac OS: `sh wso2server.sh`
- b. Sign into the API Store.  
<https://<hostname>:9443/store>
- c. Click **Applications** and click on the name of the application that you want to configure the OAuth2 with the Kerberos grant type.
- d. Generate the Production Keys.
- i. Click **Production Keys**.
  - ii. Click on the **Kerberos** checkbox as shown in the screenshot.

- iii. Click **Generate Keys** to generate the keys.
- e. Generate the Sandbox Keys.
- i. Click **Sandbox Keys**.
  - ii. Click on the **Kerberos** checkbox.
  - iii. Click **Generate Keys** to generate the keys.
7. Configure the Service Principal Name (SPNName) and Service Principal Password (SPNPassword).

A **service principal name (SPN)** is a unique identifier of a **service** instance. SPNs are used by Kerberos authentication to associate a **service** instance with a **service** account. This allows a client application to request that the **service** authenticate an account even if the client does not have the account **name**.

- a. Sign in to the WSO2 API-M Management Console.  
<https://<Server-Host>:9443/carbon>
- b. Navigate to the **Main** menu, click **Add** under the **Identity Provider** menu.
- c. Add a new Identity Provider (IDP).

The IDP name should be the name of the realm. Based on this example, it should be example.com). An identity provider is needed here to manage the KDC Service. It provides access to an identity stored in a **Kerberos** authentication server.

- **Identity Provider Name:** example.com
- **Alias:** `https://192.168.53.12:9443/oauth/token`
- **Server Principal Name:** `HTTP/idp.example.com@EXAMPLE.COM`

8. Invoke the token endpoint using the message format discussed in step 3.

Note that for users to be counted in the **Registered Users for Application statistics** which takes the number of users shared each of the Application, they should have to generate access tokens using **P password Grant type**.

#### Refresh Token Grant

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. With this grant type, the refresh token acts as credentials that are issued to the client by the authorization server. Issuing a refresh token is optional. If the authorization server issues a refresh token, it is included when issuing an access token. Refresh tokens are issued for all other grant types other than the **implicit grant** as recommended by the OAuth 2.0 specification.

**Tip:** Be sure to keep the refresh token private, similar to the access token as this token issues access tokens without user interactions.

- Generating a new access token and refresh token
- Revoking a refresh token

#### Generating a new access token and refresh token

To use this grant type, you need a refresh token, using which you can get a new access token and a refresh token. This can be done by issuing a REST call to the Token API through a REST client like cURL, with the following parameters:

- The Token API URL is <https://localhost:8243/token>, assuming that both the client and the Gateway are run on the same server.
- payload: "grant\_type=refresh\_token&refresh\_token=<retoken>". Replace the <retoken> value with the refresh token generated in the previous step.
- 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>" -H
"Authorization: Basic
SVpzSWk2SERiQjV1OFZLZFpBblVpX2ZaM2Y4YTpHbTBiSjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVz
hh" -H "Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/token
```

You receive a response similar to the following:

```
{
 "scope": "default",
 "token_type": "Bearer",
 "expires_in": 3600,
 "refresh_token": "7ed6bae2b1d36c041787e8c8e2d6cbf8",
 "access_token": "b7882d23f1f8257f4bc6cf4a20633ab1"
}
```

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. You can configure an expiration time for the refresh token by setting it in the `<RefreshTokenValidityPeriod>` element in the `<APIM_HOME>/repository/conf/identity/identity.xml` file.

### ***Revoking a refresh token***

After issuing an access token and refresh token, a user or an admin can revoke it in case of theft or a security violation. You can do this by calling the Revoke API using a utility like cURL. The Revoke API's endpoint URL is <http://localhost:8280/revoke>.

#### **Option 1**

The parameters required to invoke the following API are as follows:

- `<refresh_token_to_be_revoked>` - The token to be revoked
- `<base64 encoded (clientId:clientSecret)>` - Use a base64 encoder (e.g., <https://www.base64encode.org/>) to encode your client ID and client secret using the following format: `<clientId>:<clientSecret>`. Thereafter, enter the encoded value for this parameter.

FormatExampleResponse

```
curl -k -v -d "token=<refresh_token_to_be_revoked>" -H "Authorization:
Basic <base64 encoded (clientId:clientSecret)>" -H "Content-Type:
application/x-www-form-urlencoded" https://localhost:8243/revoke
```

```
curl -k -v -d "token=c8e8eec2-0092-3ac6-b23f-ef7492f345a6" -H
"Authorization: Basic
OVRRNVJLZWFnVGZGeUpRSkRzam9aZmp4UkhjYTpDZnJ3ZXRu19ZOTdSSzFTZwlWQWx1aXdVVm
th" -H "Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/revoke
```

```

> Host: localhost:8243
> User-Agent: curl/7.50.2
> Accept: */
> Authorization: Basic
YjNtTzdkQ2h3UHBfdTVHOFN6cVBzSDVTRnZRYTo4OG16bGFAejc2T2RlekJSNDBwcmZBa2ZNUj
Bh
> Content-Type: application/x-www-form-urlencoded
> Content-Length: 42
>
< HTTP/1.1 200 OK
< X-Frame-Options: DENY
< RevokedRefreshToken: c8e8eec2-0092-3ac6-b23f-ef7492f345a6
< Cache-Control: no-store
< X-Content-Type-Options: nosniff
< AuthorizedUser: admin@carbon.super
< Pragma: no-cache
< RevokedAccessToken: c7febbd3-5f35-3727-ae5f-5a8492b04f93
< X-XSS-Protection: 1; mode=block
< Content-Type: text/html
< Date: Thu, 02 Nov 2017 12:57:58 GMT
< Transfer-Encoding: chunked

```

## Option 2

The parameters required to invoke the following API are as follows:

- <refresh\_token\_to\_be\_revoked> - The token to be revoked.
- <base64 encoded (clientId:clientSecret)> - Use a base64 encoder (e.g., <https://www.base64encode.org/>) to encode your client ID and client secret using the following format: <clientId>:<clientSecret> Thereafter, enter the encoded value for this parameter.
- token\_type\_hint - This parameter is optional. If you do not specify this parameter, then WSO2 Identity Server (WSO2 IS) will search in both key spaces (access and refresh) and if it finds a matching token then it will be revoked. Therefore, if this parameter is not specified the token revocation process takes longer. However, if you specify this parameter then WSO2 IS will only search in the respective token key space, hence the token revocation process is much faster.

FormatExampleResponse

```

curl -k -v -d
"token=<refresh_token_to_be_revoked>&token_type_hint=<access_token_or_refr
esh_token>" -H "Authorization: Basic <base64 encoded
(clientId:clientSecret)>" -H Content-Type:
application/x-www-form-urlencoded https://localhost:8243/revoke

```

```
curl -k -v -d
"token=4ed29669-a457-3f83-af1e-180cad271cca&token_type_hint=refresh_token"
-H "Authorization: Basic
OVRRNVJLZWFnVGZGeUpRSkRzam9aZmp4UkhjYTpDZnJ3ZXRu19ZOTdSSzFTZWlWQWx1aXdVVm
th" -H "Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/revoke
```

```
> POST /revoke HTTP/1.1
> Host: localhost:8243
> User-Agent: curl/7.50.2
> Accept: */*
> Authorization: Basic
YjNtTzdkQ2h3UHBfdTVHOFN6cVBzSDVTRnZRYTo4OG16bGFAejc2T2RlekJSNDBwcmZBa2ZNUj
Bh
> Content-Type: application/x-www-form-urlencoded
> Content-Length: 72
>
< HTTP/1.1 200 OK
< X-Frame-Options: DENY
< RevokedRefreshToken: 4ed29669-a457-3f83-af1e-180cad271cca
< Cache-Control: no-store
< X-Content-Type-Options: nosniff
< AuthorizedUser: admin@carbon.super
< Pragma: no-cache
< RevokedAccessToken: 23562997-bbc7-353f-a650-16558b7147bc
< X-XSS-Protection: 1; mode=block
< Content-Type: text/html
< Date: Thu, 02 Nov 2017 12:59:41 GMT
< Transfer-Encoding: chunked
```

Note that for users to be counted in the Registered Users for Application statistics which takes the number of users shared each of the Application, they should have to generate access tokens using [Password Grant type](#).

### Authorization Code Grant

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
SVpzSWk2SERiQjVlOFZLZFpBblVpX2ZaM2Y4YTpHbTBiSjZvV1Y4ZkM1T1FMTGxDNmpzbEFDVz
hh
Content-Type:
application/x-www-form-urlencoded
grant_type=authorization_code&code=Spxl0BeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcbs
```

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 the `<APIM_HOME>/repository/conf/identity/application-authentication.xml` file. The default configuration has a relative path, which works in a standalone API Manager setup:

```
<AuthenticationEndpointURL>/authenticationendpoint/login.do</Authenticatio
nEndpointURL>
<AuthenticationEndpointRetryURL>/authenticationendpoint/retry.do</Authenti
cationEndpointRetryURL>
```

### Try Authorization Code Grant

The steps below show how access tokens are generated for the authorization code grant type.

## Before you begin,

The following instructions use the sample playground webapp. For instructions on how to set up the sample webapp, see [Setting up the Sample Webapp](#).

1. Log in to the API Manager Store and create a new application.

Add Application

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name\* Test\_Webaapp  
Characters left: 58

Per Token Quota Unlimited Allows unlimited requests  
This feature allows you to assign an API request quota per access token. Allocated quota will be shared among all the subscribed APIs of the application.

Description

Add Cancel

2. Go to the **Production Keys** tab.

3. Add the Callback URL of your playground app, select **Code** Grant type click **Generate Keys**.

By default the implicit and code grant type selection checkboxes are disabled in the UI. You need to enter the callback URL first to enable selecting the code grant type.

Details    Production Keys    **Sandbox Keys**    Subscriptions

**ⓘ No Keys Found**  
No keys are generated for this type in this application.

**Grant Types**  
The application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

Refresh Token     SAML2     Implicit     Password  
 IWA-NTLM     Client Credential     **Code**

**Callback URL**

**Scopes**  
No Scopes Found..

**Access token validity period**  
3600 Seconds

**Generate keys**

4. Go to the playground app and click Import Photos.



5. Give the information in the table below and click **Authorize**.

Field	Sample Value
Authorization Grant Type	Authorization Code
Client Id	Consumer Key obtained for your application

Scope	The scope you have selected for your application
Callback URL	The callback URL of your application
Authorize Endpoint	<a href="https://localhost:9443/oauth2/authorize">https://localhost:9443/oauth2/authorize</a>

WSO2 OAuth2 Playground

Authorization Grant Type : Authorization Code

Client Id : jf9BnQjUdVU179N0cCP7dj7SKYca

Scope : openid

Callback URL : https://10.100.5.83:9443/playground2/oauth2client

Authorize Endpoint : https://localhost:9443/oauth2/authorize

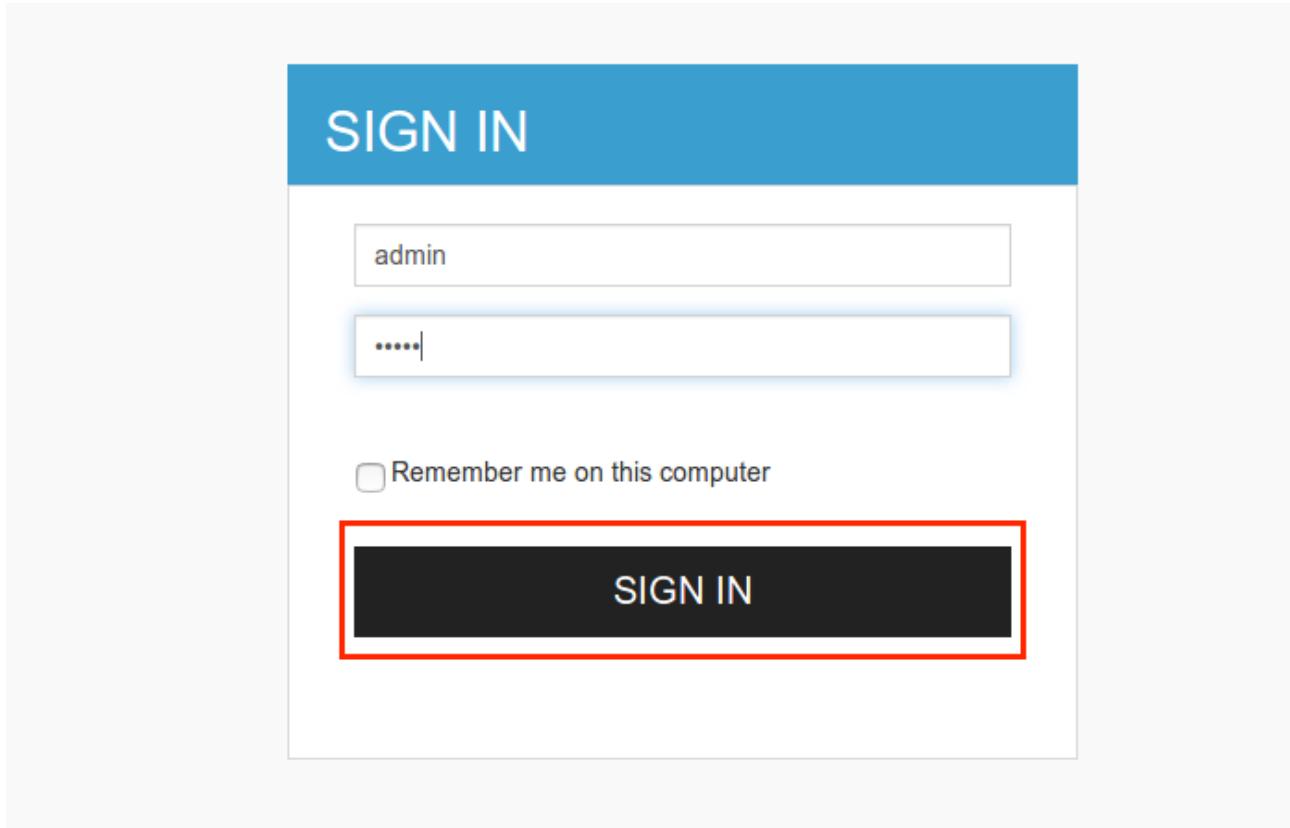
Logout Endpoint :

Session Iframe Endpoint :

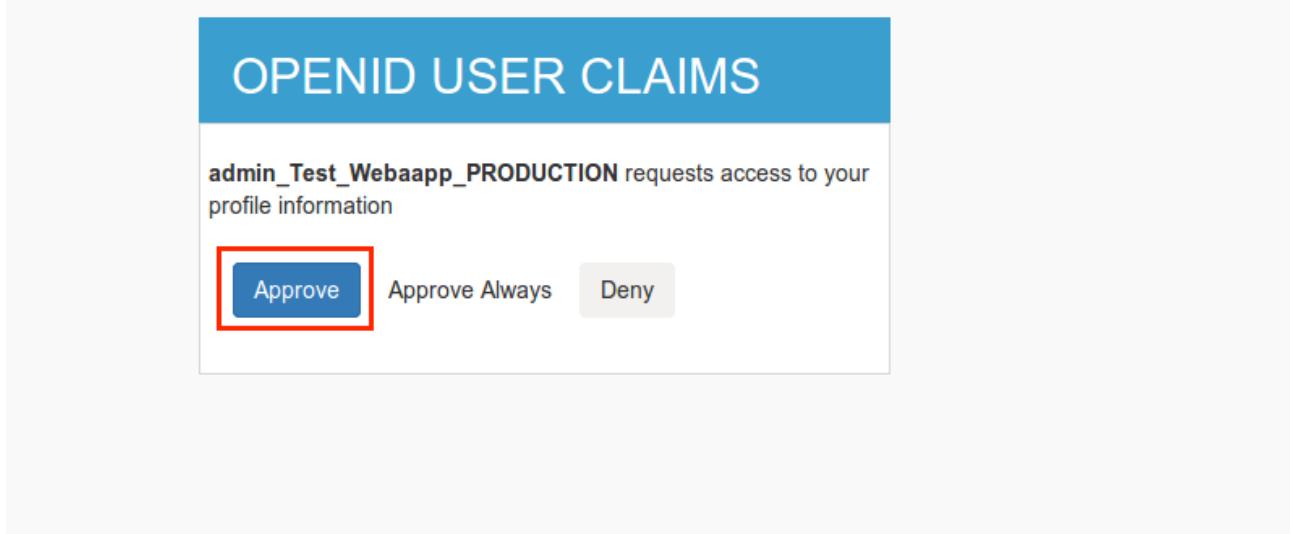
Use PKCE  Yes  No

**Authorize**

6. The playground application redirects to the login page. Enter your username and password and click **Sign In**.



7. Click Approve to provide access to your information.



You will receive the access token as follows

The screenshot shows the WSO2 OAuth2 Playground interface. At the top, there is a navigation bar with a 'Home' tab. Below it, the title 'WSO2 OAuth2 Playground' is displayed. A form area contains an 'Access Token :' input field containing the value '64499dea-05ba-347d-a8b5-afeac51c1f5f', which is highlighted with a red border. Below this is an 'Introspection Endpoint :' input field and a 'Get TokenInfo' button.

Note that for users to be counted in the [Registered Users for Application statistics](#) which takes the number of users shared each of the Application, they should have to generate access tokens using [P password Grant type](#).

## NTLM Grant

**NTLM** is the successor of the authentication protocol in Microsoft LAN Manager (LANMAN), an older Microsoft product, and attempts to provide backwards compatibility with LANMAN. You can obtain an access token to your API in an API Manager instance running on **Windows** by providing a valid NTLM token as an authorization grant. The steps are given below:

Invoking the Token API to generate tokens

1. Get a valid consumer key and consumer secret pair. Initially, you generate these keys through the API Store by clicking the **Generate Keys** button on the **Production Keys** tab of the application.
2. Combine the consumer key and consumer secret keys in the format consumer-key:consumer-secret and encode the combined string using base64 (<http://base64encode.org>). In order to generate an access token with NTLM, you must have an NTLM token.
3. Generate an NTLM token by running the sample provided in the <APIM\_HOME>/samples/NTLMGrantClient directory. See the **Readme.txt** in the same folder for instructions.
4. Invoke the token API in the following manner to get an access token. The value of the **windows\_token** in the following command is the NTLM token that you generated in the previous step.

```
curl -k -d "grant_type=iwa:ntlm&windows_token=<give the NTLM token you got in step 3>" -H "Authorization: Basic <give the string you got in step2>" -H "Content-Type: application/x-www-form-urlencoded" https://localhost:8243/token
```

Note that for users to be counted in the [Registered Users for Application statistics](#) which takes the number of users shared each of the Application, they should have to generate access tokens using [P password Grant type](#).

## Password Grant

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 API Store by clicking **Generate Keys** on the **Production Keys** tab of the application.
- A running API Gateway instance (typically an API Manager instance should be running). For instructions on API Gateway, see [Components](#).
- If the Key Manager is on a different server than the API Gateway, change the server URL (host and ports) of the Key Manager accordingly in the `<APIKeyManager><ServerURL>` element of the `<AM_HOME>/repository/conf/api-manager.xml` file.
- If you have multiple Carbon servers running on the same computer, change the port with an offset to avoid port conflicts.

#### 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: `wU62DjlyDBnq87G1Bwp1fqvmAbAa:ksdSdoefDDP7wpaElfqvmjDue`. And here's the string encoded from the example: `d1U2MkRqbH1EQm5xODdHbEJ3cGxmcXZtQWJBYTprc2RTZG91ZkREUDD3cGFFbGZxdm1qRHV1`. The encoded string should be used in the header of the cURL command.

2. Access the Token API by using a REST client such as 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 `https://localhost:8243/token`
- payload - "grant\_type=password&username=<username>&password=<password>&scope=<scope>". Replace the <username> and <password> values as appropriate.

**Tip:** <scope> is optional.

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.

- 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
d1U2MkRqbH1EQm5xODdHbEJ3cGxmcXZtQWJBYTprc2RTZG91ZkREUDD3cGFFbGZxdm1qRHV1" -H
"Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/token
```

You receive a response similar to the following:

```
Response:
{
 "scope": "default",
 "token_type": "Bearer",
 "expires_in": 3600,
 "refresh_token": "ca5a51f18b2edf4eaa9e4b871e42b58a",
 "access_token": "f2c66f146278aaaf6513b585b5b68d1d"
}
```

Instead of using the Token API, you can generate access tokens from the API Store's UI.

Note that for users to be counted in the [Registered Users for Application statistics](#) which takes the number of users shared each of the Application, they should have to generate access tokens using [Password Grant type](#).

#### SAML Extension Grant

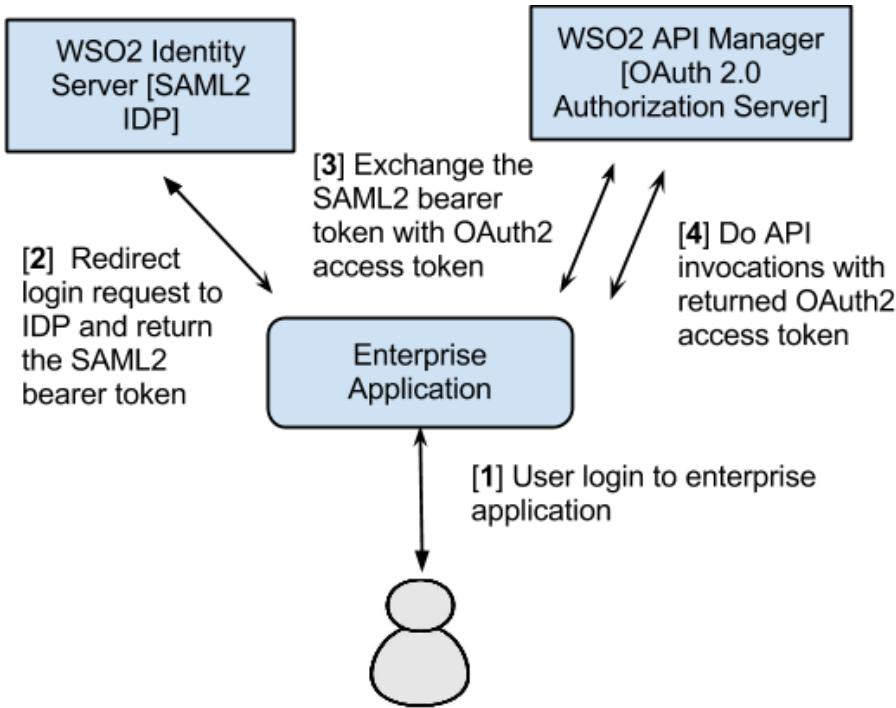
SAML 2.0 is an XML-based protocol. It uses security tokens containing assertions to pass information about an end-user between a SAML authority and a SAML consumer. A SAML authority is an identity provider (IdP) and a SAML consumer is a service provider (SP).

Enterprise applications that have SAML2 based SSO infrastructures sometimes need to consume OAuth-protected resources through APIs. However, these apps prefer to use the existing trust relationship with the IdP, even if the OAuth authorization server is entirely different from the IdP. The API Manager leverages this trust relationship by exchanging the SAML2.0 token to an OAuth token with the authorization server. It acts as the OAuth authorization server.

When SAML beare token is used, the roles of the user can be retrieved from either the user store or the SAML assertion. When **checkRolesFromSamlAssertion** system property is set to true, the roles will be checked from the SAML assertion, not the user store. Refere the step below to set this property:

1. Set the property -DcheckRolesFromSamlAssertion=true in the <API-M\_HOME>/bin/wso2server.(sh|bat) file.
2. Restart the server.

The diagram below depicts the above with **WSO2 Identity Server** as the IdP.



The steps of the above diagram are explained below:

**Step [1]:** User initiates a login call to an enterprise application

**Step [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.

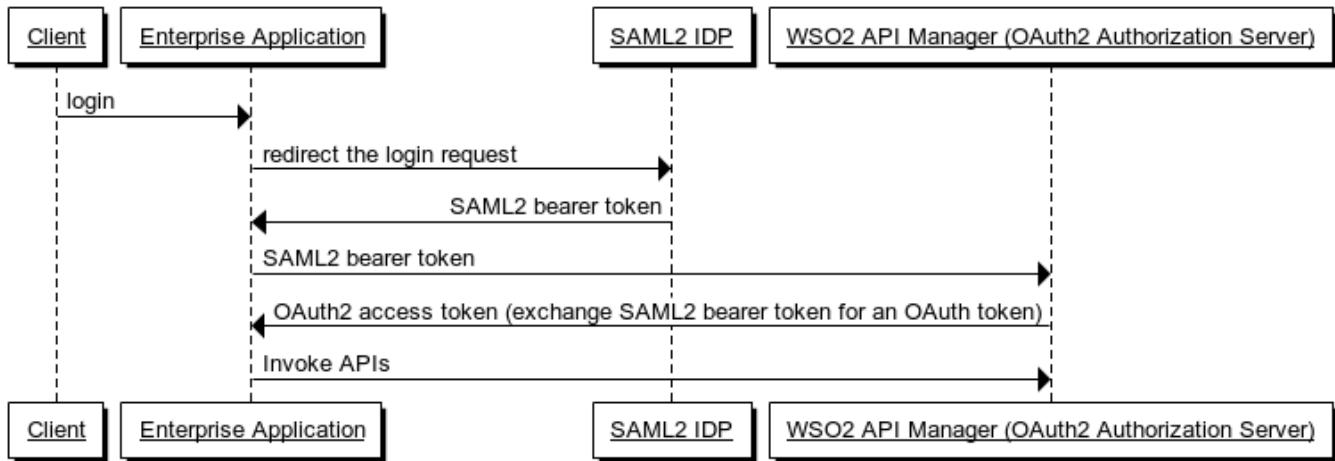
**Step [3]:**

- 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.

**Step [4]:** User does API invocations through the API Manager by setting it as an Authorization header with the returned OAuth2 access token.

- Configuring the token exchange
- Invoking the Token API to generate tokens

A sequence diagram explaining the above flow would be as follows:



### Configuring the token exchange

Before you begin, make sure you have the following:

- A valid user account in the API Store.
- A valid consumer key and consumer secret. Initially, these keys must be generated in the API Store by clicking the **Generate Keys** button on the **Production Keys** tab of the application.
- A running API Gateway instance.
- If the Key Manager is on a different server than the API Gateway, change the server URL (host and ports) of the Key Manager accordingly in the `<ServerURL>` element of the `<APIKeyManager>` section in the `<API-M_HOME>/repository/conf/api-manager.xml` file.
- A valid SAML2 assertion. For instructions on how to configure WSO2 API Manager with SAML2, see [Configuring API Manager for SSO](#)

In this example, **WSO2 Identity Server 5.3.0** is used as the IdP to get a SAML token and the API Manager is used as the OAuth server.

1. Sign in to the API Manager's management console (<https://localhost:9443/carbon>) using admin/admin credentials.

If you are using a tenant to create the Identity Provider, use the credentials of tenant admin to log into the API Manager's Management Console.

2. Click **Main > Identity Providers > Add**.



3. Provide the following values to configure the IdP:

- Under **Basic Information**
  - **Identity Provider Name:** Enter a unique name for the IdP.
  - **Identity Provider Public Certificate:** The certificate used to sign the SAML assertion. [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
wookiekeystore.jks -alias wookie
```

- **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/samlss/> if you have offset the default port, which is 9443.

If you are in tenant mode, append the tenant domain to the SSO URL as a query parameter as below.

<https://localhost:9443/samlss?tenantDomain=<tenantDomain>>

## Add New Identity Provider

**Basic Information**

Identity Provider Name: <sup>*</sup>	<input type="text" value="IS"/> <small>⑦ Enter a unique name for this identity provider</small>
Display Name:	<input type="text"/> <small>⑦ Specify the identity provider's display name</small>
Description:	<input type="text"/> <small>⑦ A meaningful description about the identity provider</small>
Federation Hub Identity Provider:	<input type="checkbox"/> <small>⑦ Check if this points to a federation hub identity provider</small>
Home Realm Identifier:	<input type="text"/> <small>⑦ Enter the home realm identifier for this identity provider</small>
Identity Provider Public Certificate:	<input type="button" value="Choose File"/> keystore1.cer ⑦ Upload identity provider's public certificate in PEM format
Alias:	<input type="text" value="https://localhost:9443/oauth2/token"/> ⑦ If the resident identity provider is known by an alias at the federated identity provider specify it

**SAML2 Web SSO Configuration**

Enable SAML2 Web SSO	<input checked="" type="checkbox"/> <small>⑦ Specifies if SAML2 Web SSO is enabled for this identity provider</small>
Default	<input checked="" type="checkbox"/> <small>⑦ Specifies if SAML2 Web SSO is the default</small>
Service Provider Entity Id: <sup>*</sup>	<input type="text" value="TestSP"/> <small>⑦ Enter the service provider's entity identifier value</small>
Select Mode	<input checked="" type="radio"/> Manual Configuration <input type="radio"/> Metadata File Configuration ⑦ Select the input method for SAML configuration
Identity Provider Entity Id: <sup>*</sup>	<input type="text" value="TestSP"/> <small>⑦ Enter identity provider's entity identifier value</small>
SSO URL: <sup>*</sup>	<input type="text" value="https://localhost:9444/samlsso/"/>

Next, let's register a service provider.

- Sign in to the management console of the Identity Server and click **Main > Service Providers > Add**.



- Choose to edit the service provider that you just registered and click **Inbound Authentication Configuration > SAML2 Web SSO Configuration**.

## Service Providers

**Basic Information**

Service Provider Name: <sup>*</sup>	IS
<small> ⓘ A unique name for the service provider</small>	
Description:	<small> ⓘ A meaningful description about the service provider</small>
SaaS Application	<input type="checkbox"/> <small> Applications are by default restricted for usage by users of the service provider's tenant. If this application is SaaS enabled it is opened up for all the users of all the tenants.</small>
<ul style="list-style-type: none"> <li><input checked="" type="checkbox"/> Claim Configuration</li> <li><input checked="" type="checkbox"/> Role/Permission Configuration</li> <li><input type="checkbox"/> Inbound Authentication Configuration</li> <li><input checked="" type="checkbox"/> SAML2 Web SSO Configuration</li> <li><input checked="" type="checkbox"/> OAuth/OpenID Connect Configuration</li> <li><input checked="" type="checkbox"/> OpenID Configuration</li> <li><input checked="" type="checkbox"/> WS-Federation (Passive) Configuration</li> <li><input checked="" type="checkbox"/> WS-Trust Security Token Service Configuration</li> <li><input checked="" type="checkbox"/> Kerberos KDC</li> </ul>	
<ul style="list-style-type: none"> <li><input checked="" type="checkbox"/> Local &amp; Outbound Authentication Configuration</li> <li><input checked="" type="checkbox"/> Inbound Provisioning Configuration</li> <li><input checked="" type="checkbox"/> Outbound Provisioning Configuration</li> </ul>	
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

6. Provide the following values to configure the SP and click **Update**:

- **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](https://localhost:9443/store/jagg/jaggery_acs.jag).
- **Enable Response Signing:** true
- **Enable Audience Restriction:** true
- **Audience:** URL of the token API. E.g., <https://localhost:9443/oauth2/token>.

## Register New Service Provider

**Select Mode**

- Manual Configuration
- Metadata File Configuration
- URL Configuration

**Manual Configuration**

Issuer *	<input type="text" value="TestSP"/>
Assertion Consumer URLs *	<input type="text" value="https://localhost:9443/store/jagg/jaggery_acs"/> <span>Add</span>
Default Assertion Consumer URL *	<input type="button" value="---Select---"/>
NameID format	<input type="text" value="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress"/>
Certificate Alias	<input type="text" value="wso2carbon.cert"/>
Response Signing Algorithm *	<input type="text" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
Response Digest Algorithm *	<input type="text" value="http://www.w3.org/2000/09/xmldsig#sha1"/>
<input checked="" type="checkbox"/> Enable Response Signing	
<input type="checkbox"/> Enable Signature Validation in Authentication Requests and Logout Requests	
<input type="checkbox"/> Enable Assertion Encryption	
<input type="checkbox"/> Enable Single Logout	
SLO Response URL	<input type="text" value="https://localhost:9443/saml2/logout"/> <small>⑦ Single logout response accepting endpoint</small>
SLO Request URL	<input type="text" value="https://localhost:9443/saml2/logout?logout=1"/> <small>⑦ Single logout request accepting endpoint</small>
<input type="checkbox"/> Enable Attribute Profile	
<input type="checkbox"/> <i>Include Attributes in the Response Always</i>	
<input checked="" type="checkbox"/> Enable Audience Restriction	
Audience	<input type="text" value="https://localhost:9443/oauth2/token"/> <span>Add</span>
<input type="checkbox"/> Enable Recipient Validation	
Recipient	<input type="text"/> <span>Add</span>

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 developed by WSO2 to create the 64-bit, URL-encoded SAML assertion.

### Invoking the 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.

- Let's create a SAML2 assertion using the same command-line client that you used in the previous section.
2. Download the command-line tool from [here](#) and extract the ZIP file.
  3. Go to the extracted folder using the command line and execute the following command. We assume that both the client and the API Gateway run on the same server. Therefore, the Token API URL is `https://localhost:8243/token`.

[FormatExample](#)

#### Format

```
java -jar SAML2AssertionCreator.jar <Identity_Provider_Entity_Id>
<saml-subject> <saml-recipient> <saml-audience>
<Identity_Provider_JKS_file> <Identity_Provider_JKS_password>
<Identity_Provider_certificate_alias>
<Identity_Provider_private_key_password>
```

#### Example

```
java -jar SAML2AssertionCreator.jar localhost admin
https://localhost:9443/oauth2/token
https://localhost:9443/oauth2/token
/home/user/wso2am-2.1.0/repository/resources/security/wso2carbon.jks
wso2carbon wso2carbon wso2carbon
```

The arguments are as follows:

- <Identity\_Provider\_Entity\_Id> - This is the value of the `saml:Issuer`, which is a unique identifier of the identity provider.
- <saml-subject> - This is the value of the name ID, which is found in the `saml:Subject -> saml:NameId`
- <saml-recipient> - This is the value of the subject confirmation data recipient, which is found in the `saml:Subject -> saml:SubjectConfirmation -> saml:SubjectConfirmationData.Recipient`
- <saml-audience> - This is the value that is added to the `saml:AudienceRestriction` element of the token. This argument can take multiple values separated by commas. Each value is added as a `saml:Audience` element within `saml:AudienceRestriction`.
- <Identity\_Provider\_JKS\_file> - Pointer to the Java Key Store (JKS) file to be used for credentials.
- <Identity\_Provider\_JKS\_password> - The JKS password.
- <Identity\_Provider\_certificate\_alias> - The alias of the public certificate.
- <Identity\_Provider\_private\_key\_password> - The password of the private key that is used for signing.

This command returns a SAML2 assertion XML string and a base64-URL encoded assertion XML string. You now have a SAML2 assertion.

4. Access the Token API using a REST client such as curl. For example, the following cURL command

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=<
base64-URL_encoded_assertion>&scope=PRODUCTION" -H "Authorization:
Basic <base64_encoded_consumer-key:consumer_secret>" -H "Content-Type:
application/x-www-form-urlencoded" https://localhost:8243/token
```

Note that for users to be counted in the [Registered Users for Application statistics](#) which takes the number of users shared each of the Application, they should have to generate access tokens using **P**assword Grant type.

### Client Credentials Grant

Client credentials can be used when the authorization scope is limited to the protected resources belonging to the client. Client credentials are used as an authorization grant when the client requests access to protected resources based on an authorization previously arranged with the authorization server. The client application requests an access token from the authorization server, authenticating the request with its client key and client secret. If the client is successfully authenticated, an access token is returned.

Please refer to the [WSO2 IS documentation](#) for a detailed explanation on this grant type with the use of a sequence diagram.

#### ***Invoking the Token API to generate the tokens***

1. Get a valid consumer key and consumer secret pair. Initially, you generate these keys through the API Store by clicking **Generate Keys** on the **Production Keys** tab of the application.
2. Combine the consumer key and consumer secret keys in the format consumer-key:consumer-secret and encode the combined string using base64 (<http://base64encode.org>).
3. Use the following sample cURL command to obtain the access token.

FormatExampleResponse

```
curl -k -d "grant_type=client_credentials" -H "Authorization: Basic
<Base64-encoded-client_key:client_secret>" -H "Content-Type:
application/x-www-form-urlencoded"
https://localhost:<https-port>/token -v
```

```
curl -k -d "grant_type=client_credentials" -H "Authorization: Basic
cEJ6dUlaaEdwaGZRbWRjVVgwbG5lRmlpdXh3YTo0U0pnV19qTU56aGpIU284OGJuZVhtT
nFNMjRh" -H "Content-Type: application/x-www-form-urlencoded"
https://localhost:8243/token -v
```

```

> POST /token HTTP/1.1
> Host: localhost:8243
> User-Agent: curl/7.54.0
> Accept: */*
> Authorization: Basic
cEJ6dUlaaEdwaGZRbWRjVVgwbG5lRmlpdXh3YTo0U0pnV19qTU56aGpIU2840GJuZVhtT
nFNMjRh
> Content-Type: application/x-www-form-urlencoded
> Content-Length: 29
< HTTP/1.1 200 OK
< X-Frame-Options: DENY
< Cache-Control: no-store
< X-Content-Type-Options: nosniff
< Pragma: no-cache
< X-XSS-Protection: 1; mode=block
< Content-Type: application/json
< Date: Thu, 18 Jan 2018 12:54:32 GMT
< Transfer-Encoding: chunked
{
 "access_token": "4c27f899-6f9c-3217-b974-3ceb5a409ac3",
 "scope": "am_aplication_scope default",
 "token_type": "Bearer",
 "expires_in": 723
}

```

**Tip:** We use the Client Credentials grant type to regenerate access tokens for an application through the API Store. Therefore, you should enable this grant type to the application. To do that, go to the API Store, click the application name from under the **APPLICATIONS** menu, click the **Production Keys** tab, and select the **Client Credentials** check box under **Grant Types**.

#### Grant Types

The application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

- |                                                       |                                              |                                              |                                              |
|-------------------------------------------------------|----------------------------------------------|----------------------------------------------|----------------------------------------------|
| <input checked="" type="checkbox"/> Refresh Token     | <input checked="" type="checkbox"/> SAML2    | <input checked="" type="checkbox"/> Implicit | <input checked="" type="checkbox"/> Password |
| <input checked="" type="checkbox"/> Client Credential | <input checked="" type="checkbox"/> IWA-NTLM | <input checked="" type="checkbox"/> Code     | <input type="checkbox"/> JWT Bearer          |

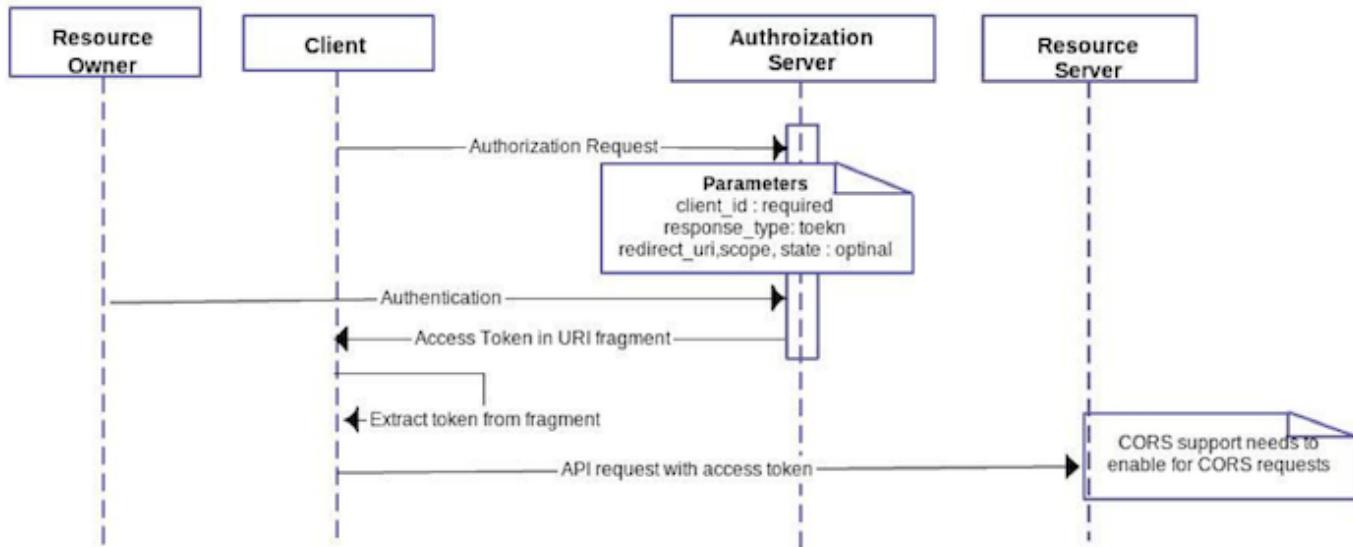
Note that for users to be counted in the [Registered Users for Application statistics](#) which takes the number of users shared each of the Application, they should have to generate access tokens using [Password Grant type](#).

#### Implicit Grant

Implicit grant type is used to obtain access tokens if your application (client) is a mobile application or a browser based app such as a JavaScript client. Similar to authorization code grant, implicit grant type is also based in redirection flow but the redirection URI includes the access token in the URI fragment. Therefore, the client application is capable of interacting with the resource owner user agent to obtain the access token from the redirection URI which is sent from the authorization server.

The implicit grant type does not require client authentication, and relies on the presence of the resource owner and the registration of the redirection URI. The resource owner needs to authenticate with the authorization server to obtain the access token. Because the access token is encoded into the redirection URI, it may be exposed to the resource owner and other applications residing in the same device.

The diagram below depicts the flow of Implicit Grant.



1. The client requests for the access token with the client ID and grant type, with optional parameters.
2. Since the resource owner authenticates directly with the authorization server, his/her credentials will not be shared with the client.
3. The Authorization Server sends the Access token through a URI fragment to the client.
4. The client extracts the token from the fragment and sends the API request to the Resource Server with the access token.

The refresh token will not be issued for the client with this grant, as the client type is public. Also note that, the implicit grant does not include client authentication because it does not make use of client secret.

Invoking the Token API to generate tokens

In this example we use the WSO2 Playground, which is hosted as a web application, to obtain the access token with implicit grant.

## Before you begin,

The following instructions use the sample playground webapp. For instructions on how to set up the sample webapp, see [Setting up the Sample Webapp](#).

1. Login to WSO2 API Manager Store and create an application as below.

Add Application

An application is a logical collection of APIs. 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. The DefaultApplication is pre-created and allows unlimited access by default.

Name*	<input type="text" value="playgroundApp"/>
Per Token Quota	<input type="text" value="Unlimited"/> Allows unlimited requests
Description	<input type="text" value="Sample web app"/>
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

2. Go to production keys tab in the created application, Add <http://localhost:8080/playground2/oauth2client> as the **callback URL**, select **implicit** from the grant type list and click Generate Keys.

By default the implicit and code grant type selection checkboxes are disabled in the UI. You need to enter the callback URL first to enable selecting the implicit grant type.

3. Go to playground app <http://wso2is.local:8080/playground2/index.jsp> and click **import photos**.



4. Give the information in the table below and click **Authorize**.

Field	Sample Value
Authorization Grant Type	Implicit
Client Id	Consumer Key obtained for your application
Scope	The scope you have selected for your application
Callback URL	The callback URL of your application
Authorize Endpoint	<a href="https://localhost:9443/oauth2/authorize">https://localhost:9443/oauth2/authorize</a>

Home

## WSO2 OAuth2 Playground

Authorization Grant Type : **Implicit**

Client Id : **DGkyfw4Baar9WnCfhhCw5sTmKg4a**

Scope : **openid**

Callback URL : **http://localhost:8080/playground2/oauth2client**

Authorize Endpoint : **https://localhost:9443/oauth2/authorize**

**Authorize**

- The playground application redirects to the login page. Enter your username and password and click **Sign In**.

SIGN IN

admin

.....

Remember me on this computer

**SIGN IN**

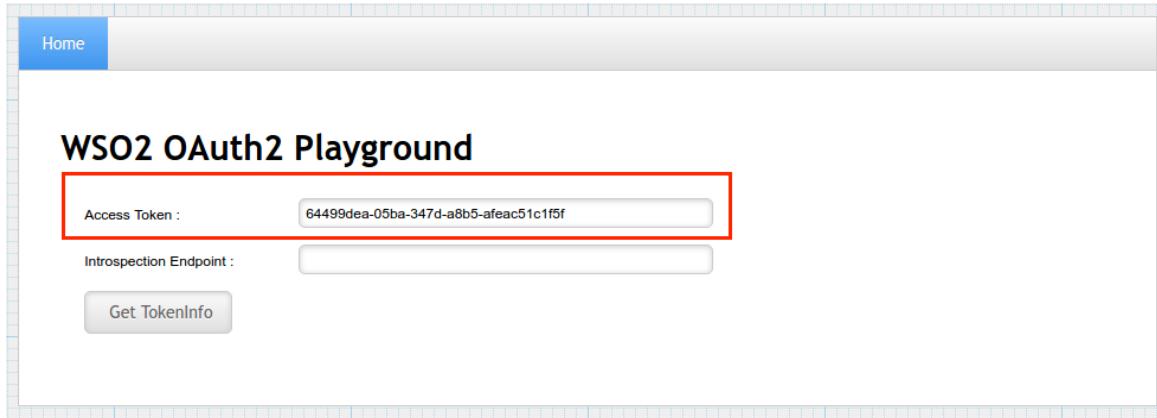
- Click Approve to provide access to your information.

OPENID USER CLAIMS

admin\_playgroundApp\_PRODUCTION requests access to your profile information

**Approve**   **Approve Always**   **Deny**

- You will receive the access token as follows



Note that for users to be counted in the [Registered Users for Application statistics](#) which takes the number of users shared each of the Application, they should have to generate access tokens using [Password Grant type](#).

## Deprecated APIs

The following APIs are deprecated and will be unsupported in a future release. They are provided below for reference by existing users.

- [Publisher APIs](#)
- [Store APIs](#)

### Publisher APIs

**The following Publisher APIs are deprecated** and will be unsupported in a future release. They are provided below for reference by existing users. For a complete list of the currently supported Publisher APIs, go to <https://docs.wso2.com/display/AM2xx/apidocs/publisher>.

- [Login](#)
- [Logout](#)
- [Add API](#)
- [Add API with Path Parameter](#)
- [Update API](#)
- [Publishing an API to external Store](#)
- [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](#)
- [Analytics related APIs](#)

**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 response is a JSON message.

## Login

<b>Description</b>	Log in to API Publisher web application.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a>
<b>URI Parameters</b>	action=login&username=xxx&password=xxx
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -c cookies <a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a> -d 'action=login&username=admin&password=admin'

## Logout

<b>Description</b>	Log out from API Publisher web application.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a>
<b>URI Parameters</b>	?action=logout
<b>HTTP Methods</b>	GET
<b>Example</b>	curl -b cookies <a href="http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag">http://localhost:9763/publisher/site/blocks/user/login/ajax/login.jag</a> ?action=logout

## Add API

<b>Description</b>	Add a new API.												
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>												
<b>URI Parameters</b>	Given below are the parameters that you can pass with an Add-API call. Mandatory ones are marked												
	<table border="1"> <thead> <tr> <th>Parameter name</th> <th>Syntax</th> </tr> </thead> <tbody> <tr> <td>Action*</td> <td>action=addAPI</td> </tr> <tr> <td>Name*</td> <td>name=xxx</td> </tr> <tr> <td>Context*</td> <td>context=/xxx</td> </tr> <tr> <td>Version*</td> <td>version=x.x.x</td> </tr> <tr> <td>API visibility*</td> <td>visibility=&lt;public private restricted&gt; The default is public. If you select restricted , mention to which roles as follows: You can read more about <b>API visibility</b> from <a href="#">here</a> .</td> </tr> </tbody> </table>	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 <b>API visibility</b> from <a href="#">here</a> .
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 <b>API visibility</b> from <a href="#">here</a> .												

Thumbnail image	<ul style="list-style-type: none"> <li>To add a thumbnail image as a file object, create the object and pass it with the</li> <li>To add a thumbnail image as a URL of the image, pass the URL with the <code>x-wso2-thum</code></li> </ul>
Description	description=xxx
Tags	tags=x,y,z
Resources*	<p>resourceCount=0&amp;resourceMethod-0=GET&amp;resourceMethodAuthType-0=Applicati</p> <ul style="list-style-type: none"> <li><code>resourceMethod</code> can take any one of the following values: GET, POST, DEL</li> <li><code>resourceMethodAuthType</code> can take any one of the following values: Applicati</li> <li><code>resourceMethodThrottlingTier</code> can take any one of the following default tem/governance/apimgt/applicationdata/tiers.xml registry location</li> </ul>
Resources as Swagger	<p>Instead of adding resources directly as above, you can add resources, including sc</p> <div style="border: 1px dashed #ccc; padding: 10px;"> <pre>swagger={"paths" : { "/CheckPhoneNumber" : {"post" : {"x-ws-read_number" : "true", "x-throttling-tier" : "Unlimited", "responses" : {"200" : {"description" : "Phone Number", "schema" : {"type" : "string"}, "x-ws-read_number" : "true"}, "201" : {"description" : "License Key", "schema" : {"type" : "string"}, "x-ws-read_number" : "true"}}, "x-ws-read_number" : "true", "roles" : "admin", "key" : "read_number"}}}</pre> </div> <p>In the above code, note that you have one resource path defined with the URL pat POST, PUT etc.) You can have multiple similar resource paths to a single API and i</p> <p>For more information of the Swagger objects used in this example, see the <a href="#">Swagger</a></p> <ul style="list-style-type: none"> <li><b>x-wso2-scopes:</b> The list of scope elements that you want to define. Each element <ul style="list-style-type: none"> <li><b>description:</b> Scope description</li> <li><b>roles:</b> Allowed roles</li> <li><b>name:</b> Scope Name</li> <li><b>key:</b> Scope Key</li> </ul> </li> <li><b>x-auth-type:</b> Authentication type of the method.</li> <li><b>x-throttling-tier:</b> Throttling tier of the method.</li> <li><b>x-scope:</b> OAuth scope of the method. This must be one of the list of element yo</li> </ul> <p>The following image shows the WSO2-specific parameters we describe here. Also :</p> <p>The screenshot shows the 'Resources' section of the WSO2 API Manager. A specific resource entry is highlighted with a yellow background. The 'x-wso2-scopes' parameter is listed under the 'Scope' section. Below it, there is an 'Add Scopes' button. At the bottom of the screen, there are tabs for 'PUT /CheckPhoneNumber + Summary' and a row of parameters: x-auth-type (Application &amp; Application User), x-throttling-tier (Unlimited), and x-scope (+ Scope).</p>

Endpoints*	<p>This example adds an HTTP production endpoint: <code>endpoint_config={ "producs", " actionSelect":"fault", " actionDuration":60000} } , "endpoi</code></p> <p>To give advanced endpoint configurations, add the JSON implementation inside "config".</p> <p>You add sandbox endpoints in the same way. The only difference is that instead of "url" you use "uri".</p> <p>If you want to add other types of endpoints, follow the examples below. <b>Note</b> that the following examples are for the production endpoint type.</p> <ul style="list-style-type: none"> <li><b>For address endpoints:</b> <code>endpoint_config={"production_endpoints":[{"url":" http://somehost.com:8080"}]}</code></li> <li><b>For failover endpoints:</b> <code>endpoint_config={"production_endpoints":[{"url":" http://failover1.endpoint.com ", "config":null}, {"url":" http://failover2.endpoint.com ", "config":null}], "algoCombo":"org.apache.synapse.endpoints.algorithms.Failover", "sessionManagement":"simpleClientSessionManagement"}</code></li> <li><b>For load balanced endpoints:</b> <code>endpoint_config" {"production_endpoints": [{"url": " http://somehost.com:8080", "config":null}], "algoCombo":"org.apache.synapse.endpoints.algorithms.RoundRobin", "sessionManagement":" simpleClientSessionManagement"}</code></li> </ul>
Endpoint security scheme	<p><code>endpointType=&lt;secured nonsecured&gt;</code></p> <p>The default is non-secured but if you select 'secured', you must pass the credentials: <code>username=&lt;the username&gt;&amp; epPassword=&lt;the password&gt;</code></p>
Make default version	<p>To mark this version of the API as the <b>default version</b> from a group of versions, give it the <b>Default Version</b> option.</p> <p>The <b>Default Version</b> option means that you make this version the default in a group of versions. For example, if you mark <a href="http://host:port/youtube/2.0">http://host:port/youtube/2.0</a> as the default version when the API has three versions, both URLs will be available.</p> <p>If you mark any version of an API as the default, you get two API URLs in its <b>Overview</b> page, one for each of the two URLs.</p> <p>If you mark an unpublished API as the default, the previous default, published API will still be available.</p>
Tier Availability*	<code>tiersCollection=&lt;Gold,Silver,Bronze,Unlimited&gt;</code>
Transports	<p><code>http_checked=http&amp;https_checked=https</code></p> <p>Both are selected by default. If you want to set only the HTTP <b>transport</b>, leave the <code>https_checked</code> argument out.</p>
Sequences	If you want to engage a custom sequence to the API, give <code>inSequence=&lt;sequence name&gt;</code> in the registry.
Response caching	<p><code>responseCache=&lt;enabled disabled&gt;</code></p> <p>It is disabled by default but if you enable it, pass the response cache timeout as follows: <code>responseCache=enabled&amp;cacheTime=10000</code>.</p> <p>See <a href="#">Configuring Caching</a> for more information.</p>
Subscriptions	<p>By default, subscription is allowed to the current tenant only.</p> <p>Add the argument <code>subscriptions=all_tenants</code> to <b>enable subscriptions</b> to the API. For example, <code>?&amp;subscriptions=all_tenants</code>.</p> <p>See <a href="#">API visibility and subscription</a> for more information.</p>
Business information	Add a section like this: <code>bizOwner=&lt;name&gt;&amp;bizOwnerMail=&lt;e-mail address&gt;</code>

<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a> -d "action=phone number&tags=phone,mobile,multimedia&endpointType=nonsecured&tiersCollection=Gold,Bronze=Application&resourceMethodThrottlingTier-0=Unlimited&uriTemplate-0=/*&default_version_checked='endpoint_config={"production_endpoints":{"url":" <a href="http://ws.cdyne.com/phoneneverify/phoneneverify.asmx">http://ws.cdyne.com/phoneneverify/phoneneverify.asmx</a> "}}

Add API with Path Parameter

<b>Description</b>	Add a new API with path parameter
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
<b>URI Parameters</b>	action=addAPI&name=xxx&context=/xxx&version=1.0.0&visibility=xxx&thumbUrl=&description=xxx&-d 'swagger=xxx'
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a> -d '{"x-auth-type": "Application%20%26%20User", "x-throttling-tier": "Unlimited", "response_type": "string", "description": "Phone Number", "in": "path"}}, "swagger": "2.0", "x-wso2-security": "key": "read_number"}, "info": {"title": "SampleApi", "version": "1.0.0"}' -d 'endpoint_config={"production_endpoint_type": "http"}'

Update API

<b>Description</b>	Update an existing API
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
<b>URI Parameters</b>	<b>Parameters are same as in Add API</b> except that <b>action = updateAPI</b> and you can only update the provider, tags, endpointType, endpoint_config (can change the endpoint URL etc.) http_checked, https_checked add new resources. See example below.
<b>HTTP Methods</b>	POST
<b>Example</b>	<b>Update API :</b> curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a> -d '{"x-auth-type": "Application%20%26%20User", "x-throttling-tier": "Unlimited", "response_type": "string", "description": "Phone Number", "in": "path"}}, "swagger": "2.0", "x-wso2-security": "key": "read_number"}, "info": {"title": "SampleApi", "version": "1.0.0"}' -d 'endpoint_config={"production_endpoint_type": "http"}'

Publishing an API to external Store

<b>Description</b>	Publish an API to external store
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-external/ajax/external.jag">http://localhost:9763/publisher/site/blocks/item-external/ajax/external.jag</a>

<b>URI Parameters</b>	<p>action=updateExternal&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;externalAPIStores=&lt;external-store-1&gt;::&lt;external-store-2&gt;::&lt;external-store-3&gt;::...::&lt;external-store-n&gt;</p> <p>:: sign is used to separate the list of API stores</p>
<b>HTTP Methods</b>	POST
<b>Example</b>	<pre>updateExternal : curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-external/ajax/externalAPIStores?action=updateExternal&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=admin&amp;externalAPIStores=exstore1::exstore2::exstore3</pre> <p>This API can be used to unpublish an API from a given API store as well. If we remove the provider parameter again, that API will get unpublished from the external stores which are listed in the externalAPIStores parameter. For example, if you want to remove the API from exstore2 and keep it published in exstore3, then you need to use.</p> <pre>curl -X POST -b cookies http://localhost:9763/publisher/site/blocks/item-external/ajax/externalAPIStores?action=updateExternal&amp;name=sampleAPI&amp;version=v1&amp;provider=admin@wso2.com&amp;externalAPIStores=exstore1::exstore3</pre> <p>If you want to remove the API from all the stores, provide '::' as the externalAPIStores parameter.</p>

#### Get All APIs

<b>Description</b>	Lists all the created APIs.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a>
<b>URI Parameters</b>	?action=getAllAPIs
<b>HTTP Methods</b>	GET
<b>Example</b>	curl -b cookies <a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a> ?action=getAllAPIs

#### Get an API

<b>Description</b>	Get details of a specific API.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a>
<b>URI Parameters</b>	action=getAPI&name=xxx&version=xxx&provider=xxx
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag">http://localhost:9763/publisher/site/blocks/listing/ajax/item-list.jag</a> -d "action=getAPI&name=PhoneVerification&version=1.0.0&provider=admin"

#### Remove an API

<b>Description</b>	Remove an API.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag</a>
<b>URI Parameters</b>	action=removeAPI&name=xxx&version=xxx&provider=xxx
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/remove.jag</a> -d "action=removeAPI&name=PhoneVerification&version=1.0.0&provider=admin"

Copy an API

<b>Description</b>	Copy an API to a newer version.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag">http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag</a>
<b>URI Parameters</b>	action=createNewAPI&provider=xxx&apiName=xxx&version=xxx&newVersion=xxx
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag">http://localhost:9763/publisher/site/blocks/overview/ajax/overview.jag</a> -d "action=createNewAPI&provider=admin&apiName=PhoneVerification&version=1.0.0&newVersion=2.0.0"

Check Older Version

<b>Description</b>	Does older version of API exist.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a>
<b>URI Parameters</b>	action=isAPIOlderVersionExist&provider=xxx&name=xxx&version=xxx
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a> -d "action=isAPIOlderVersionExist&provider=admin&name=PhoneVerification&version=1.0.0"

Change API Status

<b>Description</b>	Change the API's status.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a>
<b>URI Parameters</b>	action=updateStatus&name=xxx&version=1.0.0&provider=apiCreateName&status=PUBLISHED&pu
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies ' <a href="http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag">http://localhost:9763/publisher/site/blocks/life-cycles/ajax/life-cycles.jag</a> ' -d "action=updateStatus&name=PhoneVerification&version=1.0.0&provider=admin&status=PUBLISHED"

Add/Update an API Document

<b>Description</b>	Add a new API document.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</a>
<b>URI Parameters</b>	<p><b>Add Document:</b> action=addDocumentation&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx Note that docVisibility is applicable only if you have enabled it. See <a href="#">API documentation visibility</a>.</p> <p><b>Add Document file:</b> action=addDocumentation&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx&amp;sourceType=file</p> <p><b>Update Document:</b> action=addDocumentation&amp;mode=xxx&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx&amp;sourceType=inline</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The docType parameter can have values such as How To, Samples, Public Forum etc. If the sourceType is inline, the docType is ignored.</p> <p>The sourceType can be inline, file or URL.</p> </div>
<b>HTTP Methods</b>	POST
<b>Example</b>	<p><b>Add Document:</b> curl -X POST -b cookies "action=addDocumentation&amp;provider=xxx&amp;apiName=xxx&amp;version=xxx&amp;docName=xxx&amp;summary=xxx&amp;sourceType=inline"</p> <p><b>Add Document file:</b> curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</a> -F "docName=testDoc2" -F "docType=how to" -F "sourceType=file" -F "docUrl=" -F "summary=testing"</p> <p><b>Update Document:</b> curl -X POST -b cookies "action=addDocumentation&amp;mode=Update&amp;provider=admin&amp;apiName=PizzaShackAPI&amp;version=1.0.0&amp;docName=xxx&amp;summary=xxx&amp;sourceType=inline&amp;docType=other&amp;newType=primary&amp;docUrl=&amp;docLocation="</p>

#### Remove an API Document

<b>Description</b>	Remove an API document.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</a>
<b>URI Parameters</b>	action=removeDocumentation&provider=xxx&apiName=xxx&version=xxx&docName=xxx&docType=xxx
<b>HTTP Methods</b>	POST
<b>Example</b>	curl -X POST -b cookies <a href="http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag">http://localhost:9763/publisher/site/blocks/documentation/ajax/docs.jag</a> "action=removeDocumentation&provider=admin&apiName=PizzaShackAPI&version=1.0.0&docName=xxx&docType=xxx"

#### Get all Throttling Tiers

<b>Description</b>	Get the throttling tiers that can be applied to APIs
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?</a>
<b>URI Parameters</b>	action=getTiers
<b>HTTP Methods</b>	GET
<b>Example</b>	curl -b cookies <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?</a> action=getTiers

## Check if API Exists

<b>Description</b>	Check if an API by a given name exists in the API Publisher
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
<b>URI Parameters</b>	action=isAPINameExist&apiName=<name of the API>
<b>HTTP Methods</b>	GET
<b>Example</b>	curl -b cookies " <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINamePhoneVerification">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=isAPINamePhoneVerification</a> "

## Validate Roles

<b>Description</b>	Check if the user logged in user is any one in a given list of users
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag</a>
<b>URI Parameters</b>	action=validateRoles&roles=<list of roles>
<b>HTTP Methods</b>	GET
<b>Example</b>	curl -b cookies " <a href="http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=validateRoles&amp;roles=admin">http://localhost:9763/publisher/site/blocks/item-add/ajax/add.jag?action=validateRoles&amp;roles=admin</a> "

## Analytics related APIs

Before using the following analytics related APIs, ensure to configure Analytics for API-M. For more information, see [Configuring APIM Analytics](#).

- Get List of API Creators
- Get Subscriber Count
- Get API Usage By Resource Path
- Get API Usage By Destination
- Get API Usage by Provider
- Get API and Application Throttling Data
- Get API Response Fault Count

## Get List of API Creators

<b>Description</b>	Get the list of all the API creators.
<b>URI</b>	<a href="http://localhost:9763/publisher/site/blocks/stats/api-usage-user/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/api-usage-user/ajax/stats.jag</a>
<b>Request Headers</b>	<pre> "Content-Type" "Cookie" "JSESSIONID=29FC66CF81BED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8A Path=/publisher/; Secure; HttpOnly" </pre>
<b>HTTP Methods</b>	POST

<b>Payload</b>	action=getAPIUsageByUser&currentLocation=/publisher/site/pages/all-statistics.jag&fromDate=2014-01-01T00:00:00Z&toDate=2014-01-01T23:59:59Z
<b>Example</b>	curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIUsageByUser&currentLocation=/publisher/site/pages/all-statistics.jag&fromDate=2014-01-01T00:00:00Z&toDate=2014-01-01T23:59:59Z'
<b>Sample Response</b>	<html><head><title>API Usage Statistics</title></head><body><div style="text-align: center;"><h1>API Usage Statistics</h1><hr/><table border="1"><thead><tr><th>API Name</th><th>Version</th><th>Last Accessed</th><th>Usage Count</th></tr></thead><tbody><tr><td>buzzwordapi</td><td>1.0.0</td><td>2014-01-01T00:00:00Z</td><td>1</td></tr></tbody></table><hr/><p>Powered by WSO2 API Manager</p></div></body></html>

## Get Subscriber Count

Description	Get the number of subscribers.
URI	<code>http://localhost:9763/publisher/site/blocks/stats/api-subscriptions/ajax/stats.jag</code>
Request Headers	<code>"Content-Type": "application/x-www-form-urlencoded"</code> <code>"JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8A"</code> <code>Path=/publisher/; Secure; HttpOnly"</code>
HTTP Methods	POST
Payload	<code>action: getSubscriberCountByAPIs</code> <code>currentLocation:/publisher/site/pages/all-stats</code> <code>apiFilter:allAPIs</code>
Example	<code>curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getSubscriberCountByAPIs&amp;currentLocation=/publisher/site/pages/all-stats&amp;apiFilter:allAPIs'</code>
Sample Response	<code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <b>Date :</b> M <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>&lt;</code> <code>{"error": false, "usage": [{"apiName": ["buzzwordapi", "1.0.0", "sabra.wso2.com-AT-sabraorg"], "cou</code>

## Get API Usage By Resource Path

Description	Get the API usage based on the resource path.
URI	<a href="http://localhost:9763/publisher/site/blocks/stats/api-usage-resource-path/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/api-usage-resource-path/ajax/stats.jag</a>
Request Headers	"Content-Type" "C o o k i e" "JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8A Path=/publisher/; Secure; HttpOnly"

## Get API Usage By Destination

Description	Get the API usage based on the destination
URI	<a href="http://localhost:9763/publisher/site/blocks/stats/api-usage-destination/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/api-usage-destination/ajax/stats.jag</a>
Request Headers	"Content-Type": "application/x-www-form-urlencoded" "JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8A; Path=/publisher/; Secure; HttpOnly"
HTTP Methods	POST
Payload	action:getAPIUsageByDestination currentLocation:/publisher/site/pages/all-stats fromDate:2014-05-10 toDate:2016-12-16 apiFilter:allAPIs
Example	curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIUsageByDestination' stats.jag'

## Get API Usage by Provider

## Get API and Application Throttling Data

Description	Get the throttling related data related to the APIs and applications.
URI	<a href="http://localhost:9763/publisher/site/blocks/stats/api-throttledcounts/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/api-throttledcounts/ajax/stats.jag</a>
Request Headers	"Content-Type" "Cookie" "JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8A Path=/publisher/; Secure; HttpOnly"

HTTP Methods	POST
Payload	<pre> action : getThrottleDataOfAPIAndApplication currentLocation : /publisher/site/pages/all-stats fromDate : 2014-05-10 toDate : 2016-12-16 apiFilter : all APIs apiName : appNames: </pre>
Example	<pre>curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getThrottleBlocks/stats/api-throttledcounts/ajax/stats.jag'</pre>
Sample Response	<pre> &lt;                                             F r &lt;                                             Date : &lt;                                             F r &lt; JSESSIONID=AA2ADAFAD614FDDCE276E1E63EA836BBEBF098FA685A4FE6B8BB9110FC1E4E Path = /publisher/ ; { "error" : false } JSESSIONID=AA2ADAFAD614FDDCE276E1E63EA836BBEBF098FA685A4FE6B8BB9110FC1E4E Secure; HttpOnly' -H "Content-type: application/x-www-form-urlencoded" -d 'action=getThrottleDataOfAPIAndApplication/stats/api-throttledcounts/ajax/stats.jag {"error" : false, "usage" : {"result" : [{"apiName" : "Cedum", "apiPublisher" : "__all_providers__@andday", "timeUnitMilli" : 86400000}]} </pre>

## Get API Response Fault Count

Description	Get the response fault count of APIs.
URI	<a href="http://localhost:9763/publisher/site/blocks/stats/faulty-invocations/ajax/stats.jag">http://localhost:9763/publisher/site/blocks/stats/faulty-invocations/ajax/stats.jag</a>
Request Headers	"Content-Type" "Cookie" "JSESSIONID=29FCD6CF81BED3701B2F0FD00A7D14B6574F6BF4AF4F4A4D3E6DA7CE1DB8A Path=/publisher/; Secure; HttpOnly"
HTTP Methods	POST
Payload	action:getThrottleDataOfAPIAndApplication currentLocation:/publisher/site/pages/all-stats fromDate:2014-05-10 toDate:2016-12-16 apiFilter:allAPIs
Example	curl -v -b cookies -XPOST -H "Content-type: application/x-www-form-urlencoded" -d 'action=getAPIResults.jag'

<b>Sample Response</b> <pre>&lt; &lt; {"error": false, "usage": [{"apiName": "Cedum", "version": "1.0.0", "context": "/t/anuruddha/demo/1."}]}</pre>	<span style="font-size: 2em;">D a t e :</span> <span style="font-size: 1.5em;">X - X</span>	<span style="font-size: 1.5em;">F r</span>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------	--------------------------------------------

## Store APIs

The following Store APIs are deprecated and will be unsupported in a future release. They are provided below for reference by existing users. For a complete list of the currently supported Store APIs, go to <https://docs.wso2.com/display/AM2xx/apidocs/store/>.

- Login
- Logout
- User Signup
- Search APIs
- Get all Paginated Published APIs
- Add an Application
- Update an Application
- Get Applications
- Get an Application by Name
- Remove an Application
- Generate an Application Key
- Update an Application Key
- Add a Subscription
- List Subscriptions
- List Subscriptions by Application
- List Subscriptions by API
- Remove a Subscription
- Delete an OAuth Application
- Provision an Out-of-Band OAuth Client
- Clean Partially Created Keys
- Get all Documentation
- Get the Contents of a File Document
- Add an API Comment
- Get all Endpoint URLs
- Get all Available Tiers
- Update Grant Types

**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	<a href="http://localhost:9763/store/site/blocks/user/login/ajax/login.jag">http://localhost:9763/store/site/blocks/user/login/ajax/login.jag</a>
URI Parameters	action=login&username=<username>&password=<password>
HTTP Methods	POST
Example	<pre>curl -X POST -c cookies http://localhost:9763/store/site/blocks/user/login/ajax/login.jag -d 'action=login&amp;username=admin&amp;password=admin'</pre>

## Logout

Description	Log out from API Store.
URI	<a href="http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout">http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout</a>
URI Parameters	?action=logout
HTTP Methods	GET
Example	<pre>curl -b cookies http://localhost:9763/store/site/blocks/user/login/ajax/login.jag?action=logout</pre>

## User Signup

Description	Add a new API Consumer.
URI	<a href="http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag">http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag</a>
URI Parameters	action=addUser&username=<username>&password=<password>&allFieldsValues=<first>
HTTP Methods	POST
Example	<pre>curl      'http://localhost:9763/store/site/blocks/user/sign-up/ajax/user-add.jag' 'action=addUser&amp;kimhill=username&amp;password=kimhill1234&amp;allFieldsValues=Pasadena,California USA kim@abcnetwork.com 0016269934122 0016269934134 kim'</pre>

## Search APIs

Description	Search for APIs using a given query.
URI	<a href="http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag">http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag</a>
URI Parameters	action=searchAPIs&query=<query>&start=<number>&end=<number>
	The <b>start</b> and <b>end</b> parameters determine the range of APIs you want to retrieve. For example, if start=1 and end=3, the first 3 APIs that appear in the search results will be returned. <b>Note</b> that both 0 and 1 represent the first API in the search results, so start=0 and start=1 both means the same.
HTTP Methods	POST

Example	<pre>curl -X POST -b cookies "http://localhost:9763/store/site/blocks/search/api-search/ajax/search.jag" -d "action=searchAPIs&amp;query=test&amp;start=0&amp;end=3"</pre>
---------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Get all Paginated Published APIs

Description	Get a list of all published APIs in paginated form so that browsing is easier.
URI	<a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag">http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag</a>
URI Parameters	<p><code>action=getAllPaginatedPublishedAPIs</code>, <code>tenant</code>, <code>start</code>, <code>end</code>, <code>returnAPITags</code> (optional)</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.</p> <p>The <code>returnAPITags</code> parameter is optional. If <code>returnAPITags=true</code>, the system makes a call to the registry and returns the tags of each API in the response.</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&amp;tenant=carbon.super&amp;start=1&amp;end=100"</pre>

Please note that the `getAllPublishedAPIs` API is now deprecated. You can get the same functionality from `getAllPaginatedPublishedAPIs`.

Add an Application

Description	Add a new application.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag">http://localhost:9763/store/site/blocks/application/application-add/ajax/application-add.jag</a>
URI Parameters	<code>action=addApplication&amp;application=xxx&amp;tier=xxx&amp;description=xxx</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&amp;application&gt;NewApp1&amp;tier=Unlimited&amp;description='</pre>

Update an Application

Description	Update an existing application.
-------------	---------------------------------

URI	<a href="http://localhost:9763/store/site/blocks/application/application-update/ajax/application-update.jag">http://localhost:9763/store/site/blocks/application/application-update/ajax/application-update.jag</a>
URI Parameters	action=updateApplication&applicationOld=<application_name>&applicationNew=<
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/b 'action=updateApplication&applicationOld=NewApp1&applicationNew=NewApp2&tie

## Get Applications

Description	Get list of applications.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag">http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag</a>
URI Parameters	?action=getApplications
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/store/site/blocks/application/application-list/ajax/

## Get an Application by Name

Description	Get details of a single application by name.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag">http://localhost:9763/store/site/blocks/application/application-list/ajax/application-list.jag</a>
URI Parameters	?action=getApplicationByName&applicationName=\$APP_NAME
HTTP Methods	GET
Example	curl -b cookies http://localhost:9763/store/site/blocks/application/application-list/ajax/

## Remove an Application

Description	Remove an existing application.
URI	<a href="http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag">http://localhost:9763/store/site/blocks/application/application-remove/ajax/application-remove.jag</a>
URI Parameters	action=removeApplication&application=<application_name>
HTTP Methods	POST
Example	curl -X POST -b cookies http://localhost:9763/store/site/blocks/application/application-remove/ajax/ -d "action=removeApplication&application=NewApp2"

## Generate an Application Key

Description	Generate the key and secret values for a new application.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action=generateApplicationKey&application=<app_name>&keytype=<PRODUCTION S domains from which requests are allowed to the APIs>&validityTime=<time>& tokenScope is given in the request when your API has Auth scopes defined. See <a href="#">OAuth scopes</a> .
HTTP Methods	POST

## Update an Application Key

Description	Update the key and secret values for an application.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action=updateClientApplication&application=<app_name>&keytype=<PRODUCTION S authorizedDomains=<The domains from which requests are allowed to the APIs>&validityTime=<time>& tokenScope is given in the request when your API has Auth scopes defined. See <a href="#">OAuth scopes</a> .
HTTP Methods	POST

## Add a Subscription

Description	Add a new API subscription.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	<ul style="list-style-type: none"> <li><b>By application name:</b> action=addAPISubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx</li> <li><b>By application ID:</b> action=addSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx&amp;id=xxx</li> </ul>
HTTP Methods	POST

Example	<ul style="list-style-type: none"> <li><b>By application name:</b> curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax -d 'action=addAPISubscription&amp;name=TestAPI&amp;version=1.0.0&amp;provider=admin&amp;ame=DefaultApplication'</li> <li><b>By application ID:</b> curl -X POST -b cookies http://localhost:9763/store/site, bscription-add/ajax/subscription-add.jag -d 'action=addSubscription&amp;name=TestAPI&amp;version=1.0.0&amp;provider=admin&amp;tier=G'</li> </ul>
---------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## List Subscriptions

Description	List all applications with active subscriptions, along with the access key information of each application.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag</a>
URI Parameters	<p>action=getAllSubscriptions, selectedApp (optional)</p> <p>You can give an application's name in the <b>selectedApp</b> parameter. The API then returns the given application's subscribed APIs and access key information. If you do not specify this parameter, only the first application in the retrieved application list will contain subscribed API details, in addition to the access key information.</p>
HTTP Methods	GET
Examples	<ol style="list-style-type: none"> <li>curl -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions</li> <li>curl -b cookies 'http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag?action=getAllSubscriptions &amp;selectedApp=NewApp1'</li> </ol>

## List Subscriptions by Application

Description	List all API subscriptions of a given application.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag</a>
URI Parameters	action=getSubscriptionByApplication&app=<application_name>
HTTP Methods	GET
Example	<pre>curl -b cookies 'http://localhost:9763/store/site/blocks/subscription/subscription-list.jag?action=getSubscriptionByApplication&amp;app=NewApp1'</pre>

## List Subscriptions by API

Description	List all subscriptions of a given API.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag">http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag</a>
URI Parameters	action=getSubscriptionByAPI&apiName=xxx&apiVersion=xxx&provider=xxx
HTTP Methods	GET
Example	curl 'http://localhost:9763/store/site/blocks/subscription/subscription-list/ajax/subscription-list.jag'

Remove a Subscription

Description	Remove an API subscription.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag">http://localhost:9763/store/site/blocks/subscription/subscription-remove/ajax/subscription-remove.jag</a>
URI Parameters	<ul style="list-style-type: none"> <li><b>By application name:</b> action=removeSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx</li> <li><b>By application Id:</b> action=removeSubscription&amp;name=xxx&amp;version=xxx&amp;provider=xxx</li> </ul>
HTTP Methods	POST
Example	<ul style="list-style-type: none"> <li><b>By application Name:</b> curl -X POST -b cookies http://localhost:9763/store/site/subscription-remove/ajax/subscription-remove.jag -d 'action=removeSubscription&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=ltApplication'</li> <li><b>By application Id:</b> curl -X POST -b cookies http://localhost:9763/store/site/subscription-remove/ajax/subscription-remove.jag -d 'action=removeSubscription&amp;name=PhoneVerification&amp;version=1.0.0&amp;provider=ltApplication'</li> </ul>

Delete an OAuth Application

Description	Deletes an OAuth application in a <a href="#">third-party Authorization Server</a> . If you delete it through the API Store UI, only the mapping that is maintained in the API Manager side will be deleted.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action= <b>deleteAuthApplication</b> &consumerKey=<application_key>
HTTP Methods	POST
Example	curl -k -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d 'action=deleteAuthApplication&consumerKey=4lHddsxCtpFa2zJE1EbBpJy_NIQa'

Provision an Out-of-Band OAuth Client

Description	Provisions an OAuth client that was created out-of-band.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action= <b>mapExistingOAuthClient</b> &application=<APPLICATION NAME>&keytype=PRODUCTION&duration=1n<authorized_domains> = The domains from which requests are allowed to the APIs.
HTTP Methods	POST
Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag?application=Appl&amp;keytype=PRODUCTION&amp;callbackUrl=google.com&amp;authorizedDomains=ALL&amp;version=1&amp;jsonParams={"username": "admin", "key_type": "PRODUCTION", "client_secret": "ynEJLwvDyfHkPQGKzqBZCQ"}&amp;duration=1n</pre>

## Clean Partially Created Keys

Description	Cleans any partially created keys from the API Manager database, before adding a new subscription. Partially created keys can remain in the API Manager databases when an OAuth application of a <a href="#">third-party authorization server</a> gets deleted using the API Store UI. It only deletes the mapping that is maintained in the API Manager side.
URI	<a href="http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action= <b>cleanUpApplicationRegistration</b> &applicationName=xxx&keyType=PRODUCTION/SAND BOX'
HTTP Methods	POST
Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag -d 'action=cleanUpApplicationRegistration&amp;applicationName=DefaultApplication&amp;keyType=PRODUCTION'</pre>

[Get all Documentation](#)

Description	Get all documents create for a given API
URI	<a href="http://localhost:9763/store/site/blocks/api/listing/ajax/list.jag">http://localhost:9763 /store/site/blocks/api/listing/ajax/list.jag</a>
URI Parameters	action=getAllDocumentationOfApi&name=<API_name>&version=x.x.x&provider=<API_provider>
HTTP Methods	GET
Example	<code>curl -b cookies "http://localhost:9763/store/site/blocks/api/listing/ajax/rification&amp;version=1.0.0&amp;provider=admin"</code>

## Get the Contents of a File Document

Description	Get the contents of a file that is attached to API documentation of type 'File'
URI	<a href="http://localhost:9763/store/site/blocks/api/documentation/view/ajax/file-docs.jag">http://localhost:9763/store/site/blocks/api/documentation/view/ajax/file-docs.jag</a>
URI Parameters	<pre>action=getFileDocumentByFilePath&amp;filePath = &lt;file_path&gt;</pre> <p>&lt;file_path&gt; - Get the file path using <a href="#">getAllDocumentationOfApi</a></p>
HTTP Methods	GET
Example	<pre>curl -b cookies "http://localhost:9763/store/site/blocks/api/documentation/view/ajax/file-docs.jag?action=getFileDocumentByFilePath&amp;filePath=/registry/resource/_system/governance/apimgt/applicationdata/provider/admin/host/1.0.0/documentation/files/test.txt"</pre>

Add an API Comment

Description	Add a comment to an API.
URI	<a href="http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag">http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag</a>
URI Parameters	action=addComment&name=xxx&version=xxx&provider=xxx&comment=xxx
HTTP Methods	POST
Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/comment/comment-add/ajax/comment-add.jag 'action=addComment&amp;name=CalculatorAPI&amp;version=1.0&amp;provider=admin&amp;comment=the comment'</pre>

Get all Endpoint URLs

Description	Get all the endpoint URLs of the API Gateway environments configured for an API.
URI	<a href="http://localhost:9763/store/site/blocks/api/api-info/ajax/api-info.jag">http://localhost:9763/store/site/blocks/api/api-info/ajax/api-info.jag</a>
URI Parameters	action=getAPIEndpointURLs&name=xxx&version=x.x.x&provider=xxx
HTTP Methods	POST
Example	<pre>curl -X POST -b cookies http://localhost:9763/store/site/blocks/api/api-info/ajax/api-info.jag -d 'action=getAPIEndpointURLs&amp;name=CalculatorAPI&amp;version=1.0&amp;provider=admin'</pre>

## Get all Available Tiers

Description	Get all the tiers available in the deployment.
URI	<a href="http://localhost:9763/store/site/blocks/item-add/ajax/add.jag">http://localhost:9763/store/site/blocks/item-add/ajax/add.jag</a>
URI Parameters	action=getTiers
HTTP Methods	GET

## Update Grant Types

Description	Edit default grant types and add new grant types
URI	<a href="https://localhost:9443/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag">https://localhost:9443/store/site/blocks/subscription/subscription-add/ajax/subscription-add.jag</a>
URI Parameters	action=updateClientApplication&application=<Application_Name>&keytype=<Typ
HTTP Methods	GET

Example

```
curl -b cookie http://localhost:9763/store/site/blocks/item-add/ajax/add.jag?action=getTiers
```

To create a list of the grant types to be encoded

1. Write a JSON string with the required grant types.

```
{ "grant_types": "refresh_token,urn:ietf:params:oauth:grant-type
```

2. Encode them with a URL encoder.
3. Use the encoded value for the `jsonParams` parameter as shown in the sample cURL command below.

You can also invoke these APIs using mutual SSL authentication. Follow the instructions below to enable this:

1. Go to `<APIM_HOME>/repository/conf/tomcat/catalina-server.xml` and set the `clientAuth` attribute to `want`.

```
<Connector
 protocol="org.apache.coyote.http11.Http11NioProtocol"
 port="9443"
 ...
 clientAuth="want"
 ...
/>
```

2. For each Store API, attach the X509Certificate and pass the MutualAuthUserName parameter in the header.
3. Ensure that both client and the server have each other's certificates in the trust store.

## Working with Audit Logs

Auditing is a primary requirement when it comes to monitoring production servers. For examples, DevOps need to have a clear mechanism for identifying who did what, and to filter possible system violations or breaches. Audit logs or audit trails contain a set of log entries that describe a sequence of actions that occurred over a period of time. Audit logs allow you to trace all the actions of a single user, or all the actions or changes introduced to a certain module in the system etc. over a period of time. For example, it captures all the actions of a single user from the first point of logging in to the server.

Audit logs are enabled by default in WSO2 API Manager (WSO2 API-M) via the following configurations, which are in the <API-M-HOME>/repository/conf/log4j.properties file.

```
Configure audit log for auditing purposes
log4j.logger.AUDIT_LOG=INFO, AUDIT_LOGFILE
log4j.appenders.AUDIT_LOGFILE=org.apache.log4j.DailyRollingFileAppender
log4j.appenders.AUDIT_LOGFILE.File=${carbon.home}/repository/logs/audit.log
log4j.appenders.AUDIT_LOGFILE.Append=true
log4j.appenders.AUDIT_LOGFILE.layout=org.wso2.carbon.utils.logging.TenantAwarePatternLayout
log4j.appenders.AUDIT_LOGFILE.layout.ConversionPattern=[%d] %P%5p - %x %m %n
log4j.appenders.AUDIT_LOGFILE.layout.TenantPattern=%U%@%D [%T] [%S]
log4j.appenders.AUDIT_LOGFILE.threshold=INFO
log4j.additivity.AUDIT_LOG=false
```

The audit logs that get created when running WSO2 API-M are stored in the audit.log file, which is located in the <API-M\_HOME>/repository/logs directory.

### *Audit log actions*

In WSO2 API-M, audit logs can be enabled for the following user actions in the Publisher and Store.

#### Publisher

Action	Sample Format
Sign in to the Publisher	[2017-06-07 22:26:22,506] INFO - 'devona@carbon.super [-1234]' logged in at [2017-06-07 22:26:22,506]

Create an API	[2017-06-07 22:28:06,027] INFO - {"performedBy": "admin", "action": "created", "typ": "API", "info": "{\"provider\": \"carbon.super\", \"version\": \"1.0\", \"name\": \"Test API\", \"description\": \"A test API\", \"category\": \"test\", \"label\": \"Test API\", \"type\": \"rest\", \"url\": \"http://localhost:8280/test/api\", \"status\": \"active\", \"lastUpdated\": \"2017-06-08T10:15:55.369Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-08T10:15:55.369Z\"}"}
Update an API	[2017-06-08 10:22:49,657] INFO - {"performedBy": "admin", "action": "updated", "typ": "API", "info": "{\"provider\": \"carbon.super\", \"version\": \"1.0\", \"name\": \"Test API\", \"description\": \"A test API\", \"category\": \"test\", \"label\": \"Test API\", \"type\": \"rest\", \"url\": \"http://localhost:8280/test/api\", \"status\": \"active\", \"lastUpdated\": \"2017-06-08T10:15:55.369Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-08T10:22:49.657Z\"}"}
Delete an API	[2017-06-08 10:15:55,369] INFO - {"performedBy": "admin", "action": "deleted", "typ": "API", "info": "{\"provider\": \"carbon.super\", \"version\": \"1.0\", \"name\": \"Test API\", \"description\": \"A test API\", \"category\": \"test\", \"label\": \"Test API\", \"type\": \"rest\", \"url\": \"http://localhost:8280/test/api\", \"status\": \"active\", \"lastUpdated\": \"2017-06-08T10:15:55.369Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-08T10:15:55.369Z\"}"}

## Store

Action	Sample Format
Sign in to the Store	[2017-06-07 22:34:54,684] INFO - 'admin@carbon.super [-1234]' logged in
Sign up via the Store	[2017-06-07 22:55:34,054] INFO - Initiator : admin@carbon.super   Action
Create an application	[2017-06-07 22:40:17,625] INFO - {"performedBy": "admin", "action": "create", "typ": "Subscription", "info": "{\"application\": \"Test Application\", \"status\": \"active\", \"lastUpdated\": \"2017-06-07T22:40:17.625Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-07T22:40:17.625Z\"}"}
Update an application	[2017-06-07 22:44:25,931] INFO - {"performedBy": "admin", "action": "update", "typ": "Subscription", "info": "{\"application\": \"Test Application\", \"status\": \"active\", \"lastUpdated\": \"2017-06-07T22:44:25.931Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-07T22:44:25.931Z\"}"}
Delete an application	[2017-06-07 22:45:59,093] INFO - {"performedBy": "admin", "action": "delete", "typ": "Subscription", "info": "{\"application\": \"Test Application\", \"status\": \"active\", \"lastUpdated\": \"2017-06-07T22:45:59.093Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-07T22:45:59.093Z\"}"}
Subscribe to an application	[2017-06-07 22:36:48,826] INFO - {"performedBy": "admin", "action": "created", "typ": "Subscription", "info": "{\"application\": \"Test Application\", \"status\": \"active\", \"lastUpdated\": \"2017-06-07T22:36:48.826Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-07T22:36:48.826Z\"}"}
Unsubscribe from an application	[2017-06-07 22:38:08,277] INFO - {"performedBy": "admin", "action": "deleted", "typ": "Subscription", "info": "{\"application\": \"Test Application\", \"status\": \"active\", \"lastUpdated\": \"2017-06-07T22:38:08.277Z\", \"lastModifiedBy\": \"admin\", \"lastModifiedAt\": \"2017-06-07T22:38:08.277Z\"}"}

## Enabling Authentication Session Persistence

This topic is regarding sessions in the WSO2 API Manager (WSO2 API-M) and the process of enabling session persistence for these sessions. This is particularly useful when the remember me option is selected when logging into either the service provider or the WSO2 API-M.

Uncomment the following configuration in the <API-M\_HOME>/repository/conf/identity/identity.xml file, under the the Server and JDBCersistenceManager elements to enable authentication session persistence.

```

<SessionDataPersist>
 <Enable>true</Enable>
 <Temporary>false</Temporary>
 <PoolSize>100</PoolSize>
 <SessionDataCleanUp>
 <Enable>true</Enable>
 <CleanUpTimeout>20160</CleanUpTimeout>
 <CleanUpPeriod>1140</CleanUpPeriod>
 </SessionDataCleanUp>
 <OperationDataCleanUp>
 <Enable>true</Enable>
 <CleanUpPeriod>720</CleanUpPeriod>
 </OperationDataCleanUp>
</SessionDataPersist>

```

The following table describes the elements of the configurations mentioned above.

Configuration element	Description
Enable	This enables the persistence of session data. Therefore, this must be configured to true if you wish to enable session persistence.
Temporary	Setting this to true enables persistence of temporary caches that are created within an authentication request.
PoolSize	To improve performance, OAuth2 access tokens are persisted asynchronously in the database using a thread pool. This value refers to the number of threads in that thread pool.
SessionDataCleanUp	This section of the configuration is related to the cleaning up of session data.
Enable	Selecting true here enables the cleanup task and ensures that it starts running.
CleanUpTimeOut	This is the timeout value (in minutes) of the session data that is removed by the cleanup task. The default value is 2 weeks.
CleanUpPeriod	This is the time period (in minutes) that the cleanup task would run. The default value is 1 day.
OperationDataCleanUp	This section of the configuration is related to the cleaning up of operation data.

**Note:** If Single Sign-On is to work, you must enable at least one of the two configurations mentioned in this topic.

### Related Topics

- See [Configuring Single Sign-on with SAML2](#) for more information

## Enabling Monetization of APIs

WSO2 API Manager allows you to manage, govern and monetize APIs. You can specify throttling tiers and list the APIs that are available in the Gateway using the API Store. Subscribers who subscribe to the listed APIs and invoke them are billed based on the throttling tiers. To define a billing plan you need to define a new throttling tier in WSO2

API Manager.

#### **Subscription workflow**

To activate the subscription using throttling tiers, you need to use the WSO2 API Manager Subscription workflow. In the default workflow, subscriptions are automatically active. But when integrating with the billing engine, subscriptions are active if and only if the billing engine accepts the user. This behavior can be achieved by extending the default subscription workflow. When a subscriber tries to subscribe to an API after the subscription workflow is configured, they will be redirected to the billing engine for authentication with the API manager workflow details. Then the billing engine accepts the user or signs them up as a new user to the system. After the user enrollment is done by the billing engine it will redirect them back to the WSO2 API manager and activate the subscription.

To configure the default subscription workflow, follow the instructions given in [Implementing Workflow Extensions](#).

#### **Enabling and disabling billing related developer subscriptions**

A developer can subscribe to an API, invoke it and get invoiced for it but avoid the bill payment. To stop this from happening there needs to be a mechanism that address this issue. With WSO2 API Manager subscriptions can be enabled and disabled at the publisher's will. If the user doesn't pay, the publisher can disable the subscription and re-enable it once the payment is made.

To enable/disable subscriptions through the UI,

1. Sign in to the API Publisher.  
`https://<hostname>:9443/publisher` (e.g.: `https://localhost:9443/publisher`).
2. In the **HOME** menu, click **MANAGE SUBSCRIPTIONS**.
3. To disable the subscription of a particular API, click **Block**. Now, an API Store user is unable to invoke that API until it is re-enabled.

## Manage Subscriptions

<input type="text"/> Filter by ...					
User	Application	API	Still Subscribed?		
admin	DefaultApplication	WorldBank-1.0.0	Yes	Production & Sandbox	<input checked="" type="radio"/> Block
admin	NewApplication	WorldBank-1.0.0	Yes	Production & Sandbox	<input checked="" type="radio"/> Block
admin	NewApplication2	WorldBank-1.0.0	Yes	Production & Sandbox	<input checked="" type="radio"/> Block

Show  entries | Showing 1 to 3 of 3 entries

In **Manage Subscriptions** subscriptions of the APIs only visible for the particular user that API belongs. For example, User1 created 2 APIs and User2 created 1 API, Then User1 can only view the subscriptions for the APIs belongs to him.

## Controlling over exposure

Throttling tiers contain a Quota Reach Option, which might result in the gateway getting flooded with requests. You can use the Hard Level Throttling option to define the maximum number of requests per minute. For more information, see [Setting Maximum Backend Throughput Limits](#).

#### **Configuring API Manager Analytics**

You have to configure WSO2 API Manager Analytics in order to the billing engine to retrieve data. The API invocation related events are published to WSO2 APIM Analytics, which then persists all the events and makes them available in internal tables. You may also want summarized data sets based on the raw event data to help the billing engine work efficiently.

For details on how to configure WSO2 API Manager Analytics, see [Configuring APIM Analytics](#).

#### ***Data retrieving model for monetization***

Following are the events that are sent to WSO2 DAS:

- org.wso2.apimgt.statistics.request
- org.wso2.apimgt.statistics.response
- org.wso2.apimgt.statistics.fault
- org.wso2.apimgt.statistics.throttle
- org.wso2.apimgt.statistics.workflow

For more information about the event details please refer [Introducing the WSO2 API Manager Statistics Model](#).

The sections below describe how to monetize your API

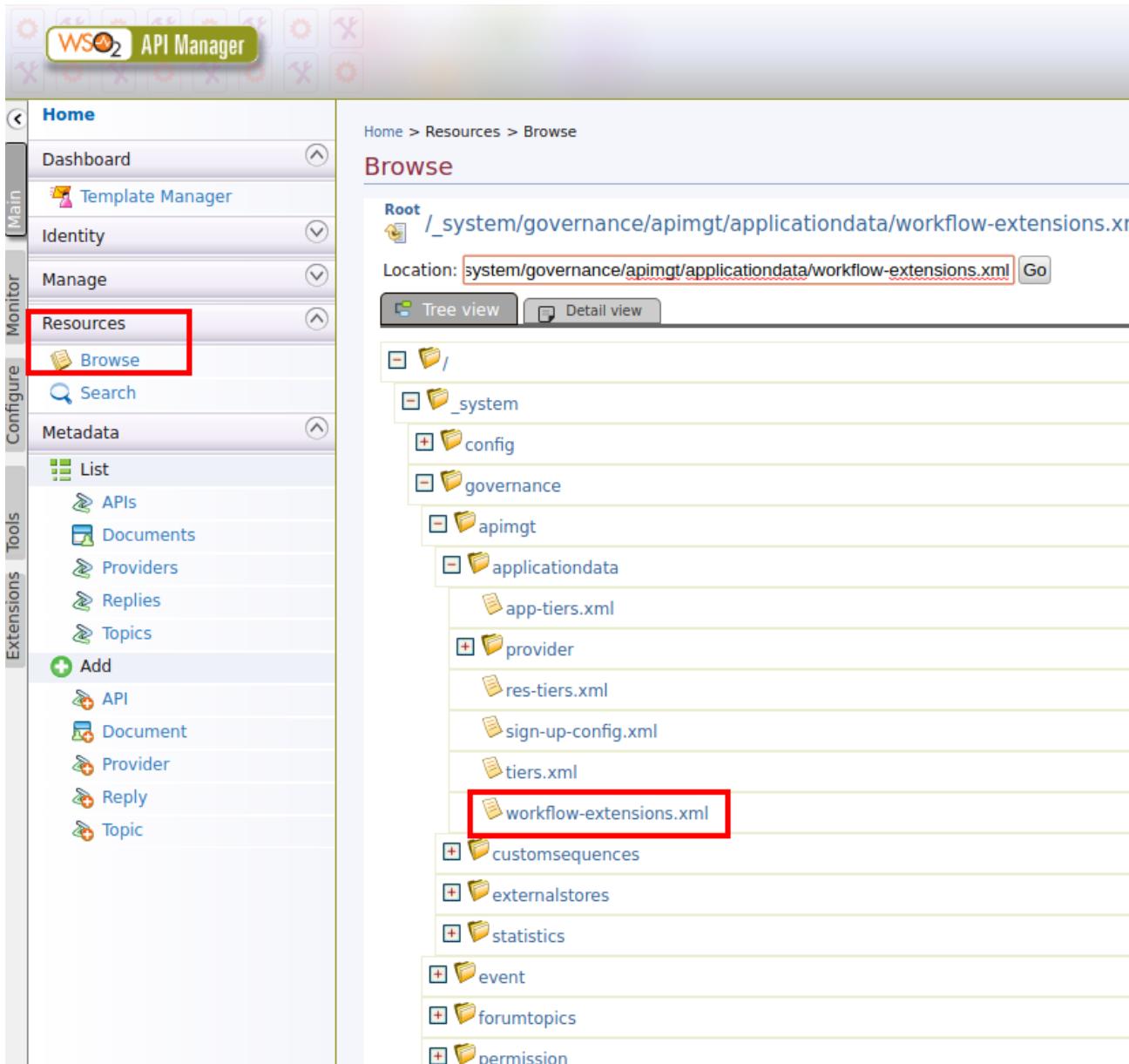
- [Implement workflow extensions](#)
- [Deploying the extended workflow](#)
- [Configuring the billing engine](#)
- [Install additional cApp](#)
- [Configuring WSO2 API Manager](#)

#### **Implement workflow extensions**

To implement the default subscription workflow, start a new maven project and create a new class named SubscriptionBillingWorkflow. You need to implement the methods as per the source code [here](#). If you need to modify the existing workflow (or even write a new workflow, you can create a new project and a new class. This class name must be the name you use in step 4.), you could do so within the implemented methods in the above code. Once you have implemented the workflow proceed to deploying the workflow extention which is explained in the below section.

#### **Deploying the extended workflow**

1. Build the maven project (`subs-billing-workflow` in the source). Or you can download the built jar file from [here](#).
2. Copy and paste the built jar into the `<API-M home>/repository/component/lib` folder (If you already start the server then restart it before proceeding further).
3. Log in to the WSO2 API Manager web console (<https://localhost:9443/carbon>) and browse the resources.



4. Open the `/_system/governance/apimgt/applicationdata/workflow-extensions.xml` file. Replace the "SubscriptionCreation" tag with the value given in the example below

```
<SubscriptionCreation
executor="org.wso2.sample.apimgt.workflow.SubscriptionBillingWorkflow
"/>
```

## Configuring the billing engine

1. Create a mysql database 'billing'.
2. Get the sample billing engine provide for WSO2 API Manager from [here](#).
3. Deploy the war in the Tomcat container.
4. Locate the deploy webapp and edit the `<apim-billing-engine-home>/WEB-INF/classes/datasource.properties` file as shown below

- url=jdbc:mysql://localhost:3306/billing
- username=root
- password=pass

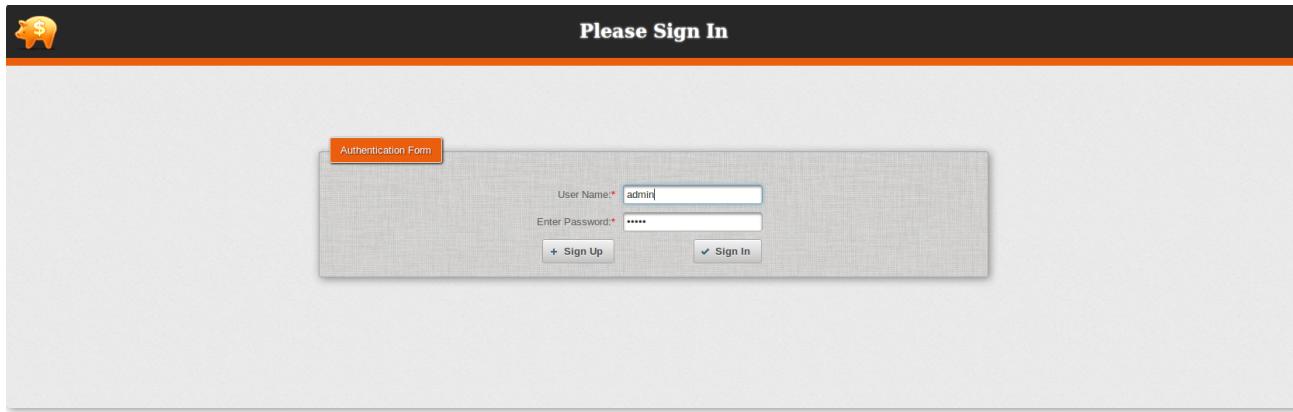
- driverClassName=com.mysql.jdbc.Driver
- dialect=org.hibernate.dialect.MySQL5InnoDBDialect
- apimStoreUrl=https://localhost:9443/
- apimUserName=admin
- apimPassword=admin
- dasUrl=https://localhost:9444/
- dasUserName=admin
- dasPassword=admin
- jksPath=<extracted\_apim-billing-engine-home>/WEB-INF/classes/wso2-jks/wso2carbon.jks

5. Navigate to the main page of the billing engine at <http://localhost:8080/apim-billing-engine-1.2.0/app/main>.
6. Sign up as a user.

**Registration Form**

Country	Sri Lanka	
Credit Card Number *	1234567891011112	
Visa	<input checked="" type="radio"/>	
Payment Type *	Master Card <input type="radio"/>	
Amex	<input type="radio"/>	
Expiration Date *	02-17	
CVC *	234	
First Name: *	Shani	
Last Name: *	Ranasinghe	
Email: *	shani@wso2.com	
Company Name: *	WSO2	
Address 1 *	Colombo	
Address 2		
Address 3		
User Name: *	Administrator	<span>(i) User Name 'Administrator' is available</span>
Enter Password: *	*****	
Repeat Password: *	*****	
<a href="#">Back to Sign In</a>		<a href="#">Sign Up</a>

7. Log in using the newly created user. This is when the tables are created in the billing database that you created in step 1.



You can define the following two different types of billing plans based on the WSO2 API Manager billing model.

- **Subscription-based Usage Plan:** Charges based on the subscription fee and fee per additional request.

Plan Name:*	Diamond	<span>User Name 'Diamond' is available</span>
Enter Quota:*	50	
Enter Fee:*	60	
Enter Additional Fee:*	0.2	

[Back to Home](#) [Save Plan](#)

- **Request-based Usage Plan:** Charge a fixed fee per request.

Usage Plan Name:*	Platinum	<span>User Name 'Platinum' is available</span>
Charge Per Unit:*	0.02	\$

[Back to Home](#) [Save Usage Plan](#)

## Install additional cApp

You can use this additional cApp to generate additional summary data used to feed the billing engine.

A Carbon Application (**C-App**) or a **CAR file** is a collection of artifacts deployable on a WSO2 product instance. These artifacts are usually JAVA-based or XML configurations, designed differently for each product in the WSO2 Carbon platform. You can deploy these artifacts to generate services.

A single WSO2 product can have numerous artifacts such as Axis2 services, dataservices, synapse configurations, endpoints, proxy services, mediators, registry resources, BPEL workflows etc. Usually, these artifacts are created in a development environment and then moved one by one into staging/production environments. Manually configuring artifacts to build up the entire solution this way is a time-consuming task. Instead, you can bundle configuration files and artifacts in a C-App and port Web service based solutions across environments more easily. C-Apps allow you to export your entire solution as a single archive file.

1. Download the cApp from [APIM\\_Billing\\_2.1.0.car](#)
2. Login to WSO2 API-M Analytics web console and navigate to **Main > Carbon Applications**.
3. Click **Add** and upload the downloaded cApp. This deploys the cApp.

You can verify whether the CApp is deployed properly by navigating to **Main > Carbon Applications > List** in WSO2 API-M Analytics web console.

## Configuring WSO2 API Manager

1. Define the data source for the billing engine. Since the workflow extension is used, define the billing engine user details data source configuration in the `<API-M home>/repository/conf/datasources/master-datasources.xml` file, as given in the example below.

```

<datasource>
 <name>BILLING_DB</name>
 <description>The datasource used for registry and user
manager</description>
 <jndiConfig>
 <name>jdbc/BILLING_DB</name>
 </jndiConfig>
 <definition type="RDBMS">
 <configuration>

<url>jdbc:mysql://localhost:3306/billing?autoReconnect=true</url>
 <username>root</username>
 <password>pass</password>
 <driverClassName>com.mysql.jdbc.Driver</driverClassName>
 <maxActive>50</maxActive>
 <maxWait>60000</maxWait>
 <testOnBorrow>true</testOnBorrow>
 <validationQuery>SELECT 1</validationQuery>
 <validationInterval>30000</validationInterval>
 <defaultAutoCommit>false</defaultAutoCommit>
 </configuration>
 </definition>
</datasource>

```

2. To configure the billing engine URL open the <API-M home>/repository/conf/api-manager.xml file and add the following configuration under <APIManager>.

```

<billingEngineUrl>http://localhost:8080/apim-billing-engine-1.2.0/app
/main</billingEngineUrl>

```

3. Since WSO2 API Manager is configured for monetization we can enable the API status as free or premium. To enable this flag edit the registry location log in to the management console and open the \_system/config/apimgt/applicationdata/tenant-conf.json file. Make the changes given below:
  - To enable monetization set the "EnableMonetization" property to true.
  - To define if the unlimited tier is paid, set the IsUnlimitedTierPaid property to true.
4. Copy and paste the mysql jar to /repository/component/lib since the extension workflow uses the mysql connector.

#### **Test the system**

1. Create a subscription tier in the Admin Portal. For instructions on creating a tier, see [Adding a new subscription-level throttling tier](#).

The screenshot shows the 'Add Subscription Tier' page in the WSO2 API Manager. The left sidebar has a tree structure with 'POLICIES', 'THROTTLING', 'SUBSCRIPTION TIERS' (which is selected), and 'RULES'. The main content area has the following sections:

- General Details**: Name is set to 'Diamond' and Description is 'Allowed 50 requests per minute, subscription fee 60\$. Charge 0.2\$ per extra request'.
- Quota Limits**: Request Count is set to 50 and Unit Time is 1 Minute(s).
- Burst Control (Rate Limiting)**: Request Count is set to 10 and Unit is Request/s.
- Policy Flags**: Stop On Quota Reach is unchecked and Billing Plan is set to Commercial.
- Custom Attributes**: A table with two rows: 'subscription\_fee' (value: 60) and 'charge\_per\_extra' (value: 0.2). Each row has a delete icon.
- Permissions**: This section is currently empty.

2. Deploy the sample API in the API Publisher. More information about deploying the sample API, see [Deploy sample API](#).
3. Edit the sample API. Go to the **Manage** tab and select the new tier you have created in the previous step. Click **Publish**.

For details on editing APIs, see [Create and Publish an API](#).

4. Login to the API Store.
5. Subscribe to the sample API using the Default Application or create a new application and subscribe. You are re-directed to the billing engine.

6. Sign up using the billing engine. You are re-directed to the Store with a successful subscription.
7. Create a new billing plan according to your subscription tier, as shown below.

8. Invoke the sample API from the API Store.
9. Go to the billing engine and generate an invoice as shown below,

The screenshot shows the 'My Account Home' page. At the top, there is a navigation bar with a piggy bank icon and the text 'My Account Home'. Below this, the heading 'Account for' is displayed. A central callout box contains several buttons: '+ Define Standard Plan', '+ Define Usage Plan', '✓ List Plan', '✓ Invoices' (which is highlighted with a red border), and '✓ View Invoices'. The background of the page is light gray.

The screenshot shows the 'Invoices for' page. At the top, there is a navigation bar with a piggy bank icon and the text 'My Account Home'. Below this, the heading 'Invoices for' is displayed. A central callout box contains two dropdown menus: 'Select Plan: Diamond' and 'Select Cycle: March'. Below these is a 'Back to Home' button and a 'Generate Invoice' button (which is highlighted with a red border). The background of the page is light gray.

Select the correct plan and the month in order to generate the invoice.

10. Your invoice is generated based on usage.



Invoice #: 80  
Created: 2016/03/22  
Due: 2016/04/22

WSO2, Inc.  
Palm Grove, Colombo  
Sri Lanka, TX 12345

Plan Type	STANDARD
Payment Method	#
Plan name	Diamond
Request count	100
Throttle count	50
Fee per Success Request	0.0
Fee per Throttled Request	0.2

Item	Price
Subscription Fee	\$60.0
Success Request Fee	\$0.0
Throttle Request Fee	\$10.0
<b>Total:</b>	<b>\$70.0</b>

## Using the Registry REST API

The registry REST API can be used to perform CRUD operations on registry resources. This is not packed with WSO2 API Manager by default. Follow the steps below to use the registry REST API with WSO2 API Manager.

1. Download the [registry REST API webapp](#).
2. Copy the webapp to <API-M\_HOME>/repository/deployment/server/webapps.
3. Invoke the registry REST API. For an example, to get the content of the `app-tiers.xml` file, in the registry path `_system/governance/apimgt/applicationdata`, the following curl command can be used:  
[Format Sample](#)

```
curl -X GET -H "Authorization: Basic
<base64_encoded_username:password>" -H "Content-Type:
application/json" -H "Cache-Control: no-cache"
"https://<hostname>:<port>/resource/1.0.0/artifact/_system/governance
/apimgt/applicationdata/app-tiers.xml" -k
```

```
curl -X GET -H "Authorization: Basic YWRtaW46YWRtaW4=" -H
"Content-Type: application/json" -H "Cache-Control: no-cache"
"https://localhost:9443/resource/1.0.0/artifact/_system/governance/ap
imgt/applicationdata/app-tiers.xml" -k
```

For a complete reference of the available REST API operations, go to [Resources with REST API](#).

# FAQ

- About WSO2 API Manager
  - What is WSO2 API Manager?
  - What is the open source license of the API Manager?
  - 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?
  - What is the default communication protocol of the API Manager?
- Installation and start up
  - What are the minimum requirements to run WSO2 API Manager?
  - What Java versions are supported by the API Manager?
  - Which MySQL database script should I use?
  - 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?
  - I don't want all the components of the API Manager up when I start the server. How do I start up only selected ones?
- Deployment
  - Where can I look up details of different deployment patterns and clustering configurations of the API Manager?
  - What are the container technologies that are supported in API Manager?
  - What is the recommended way to manage multiple artifacts in a product cluster?
  - Is it recommended running multiple WSO2 products on a single server?
  - Can I install features of other WSO2 products to the API Manager?
  - How can I continue to use my email address as the username in a distributed API-M deployment?
  - 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 ?
  - Why are the changes I did to the resource parameter Response Content Type of a published API not reflected in the API Store after saving?
  - How can I add more features to the API Manager server and extend its functionality?
  - How do I change the pass-through transport configurations?
  - If I want to extend the default API Manager server by installing new features, how can I do it?
  - How can I preserve the CDATA element tag in API responses?
- 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 enable self signup to the API Store?
  - 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?
  - Is there a way to recover a forgotten password for a API store user?
  - 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?
  - How can I manage session timeouts for the management console?
  - How can I add the authentication headers to the message going out of the API Gateway to the backend?
  - Can I give special characters in the passwords that appear in the configuration files?

- How to protect my product server from security attacks caused by weak ciphers such as the Logjam attack (Man-in-the-Middle attack)?
- Troubleshooting
  - Why do I get the following warning:  
org.wso2.carbon.server.admin.module.handler.AuthenticationHandler - Illegal access attempt while trying to authenticate APIKeyValidationService?
  - How can I fix the javax.net.ssl.SSLException: hostname in certificate didn't match: <ip address> != <localhost> exception?
  - How can I fix the javax.net.ssl.SSLException: Received fatal alert: unknown\_ca error that I get when invoking the methods of an API via the API Console?
  - 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?
  - 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?
  - Why do I get the following error: Gateway Failures?
  - How can I capture the state of a system?
- General questions
  - Can I implement an API facade with the API Manager?
  - How can I write automated test scripts for the API Manager?
  - Does WSO2 API-M support HTTP pipelining?

## 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 the [introduction](#).

### *What is the open source license of the API Manager?*

Apache Software License Version 2.0

### *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](mailto:dev@wso2.org) and [architecture@wso2.org](mailto: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 [our support page](#).

### *What is the default communication protocol of the API Manager?*

The default communication protocol is Thrift.

## Installation and start up

### **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?**

Oracle JDK versions 1.7.\*/1.8.\*.

### **Which MySQL database script should I use?**

From Carbon kernel 4.4.6 onward your product is shipped with two scripts for MySQL as follows (click [here](#) to see if your product is based on this kernel version or newer):

- `mysql.sql` : Use this script for MySQL versions prior to version 5.7.
- `mysql5.7.sql` : Use this script for MySQL 5.7 and later versions.

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

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

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

### **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, see [Using Puppet Modules to Set up WSO2 API-M](#).

### **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.

### **I don't want all the components of the API Manager up when I start the server. How do I start up only selected ones?**

Even though the API Manager bundles all components together, you can select which component/s you want to start by using the `-Dprofile` command at product startup. See [Product Profiles](#) for more information.

## Deployment

**Where can I look up details of different deployment patterns and clustering configurations of the API Manager?**

See [Deploying WSO2 API Manager](#).

**What are the container technologies that are supported in API Manager?**

OpenShift, Docker, Kubernetes and Mesos are supported.

**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 running 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 continue to use my email address as the username in a distributed API-M deployment?**

To enable using your email (e.g., `admin@wso2.com`) as your username when deploying WSO2 API-M and WSO2 Identity Server (WSO2 IS), while doing master configurations, do the following.

Go to `<API-M_HOME>/repository/conf/api-manager.xml`. In the DataPublisher section, under ThrottlingConfiguration section, specify the username as follows: `admin@wso2.com@carbon.super`. The `api-manager.xml` file accepts only configurations for the super tenant.

```
<Username>admin@wso2.com@carbon.super</Username>
```

For more details, see [Using Email Address as the Username](#).

**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.

**Why are the changes I did to the resource parameter Response Content Type of a published API not reflected in the API Store after saving?**

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.

**How can I add more features to the API Manager server and extend its functionality?**

You can install any WSO2 component to the API Manager. See the [Installing Features](#) section in the WSO2 Carbon docs for more information.

**How do I change the pass-through transport configurations?**

If you have enabled the pass-through transport, you can change its default configurations by adding the following under the `<transportReceiver name="https" class="org.apache.synapse.transport.passthru.PassThroughHttpSSLListener">` element in the `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml` file. Be sure to **stop the server** before editing the file.

If you are using JDK 1.7.\* or 1.8.\*, 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>
```

**If I want to extend the default API Manager server by installing new features, how can I do it?**

See [Working with Features](#) in the WSO2 Carbon documentation.

**How can I preserve the CDATA element tag in API responses?**

Set the `javax.xml.stream.isCoalescing` property to `false` in the `<APIM_HOME>/XMLInputFactory.properties` file. Here's an example:

```
<XacuteResponse xmlns="http://aaa/xI">
 <Rowset>
 <Row>
 <outxml><![CDATA[<inSequence>
 <send>
 <endpoint>
 <address uri="http://localhost:8080/my-webapp/echo" />
 </endpoint>
 </send>
 </inSequence>]]></outxml>
 </Row>
 </Rowset>
</XacuteResponse>
```

***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 enable self signup to the API Store?***

See [how to enable self signup](#).

***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 **Account Recovery and Credentials Management** feature (Available under User Management category) to the API Manager and configure it. For information, see [User Account Locking and Account Disabling](#) page in the Identity Server documentation. For more information on installing features, see [Feature Installation](#) documentation.

***Is there a way to recover a forgotten password for a API store user?***

In order to recover a password of a store user, you need to enable the password recovery feature. By default the feature will not work, as an email server is not configured. Follow the steps below to properly configure the forgotten password feature.

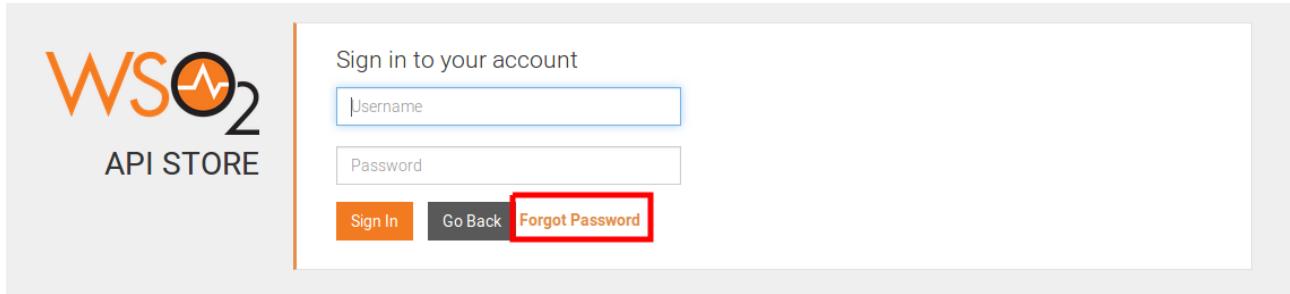
1. Open the <APIM\_HOME>repository/conf/identity folder and open the identity-mgt.properties file. Change the value of the following property to **false**.

```
Captcha.Verification.Internally.Managed=false
```

2. Open the <APIM\_HOME>repository/conf/axis2/axis2.xml and uncomment the following tag and configure a mail server. Refer to [MailTo Transport](#) for more information.

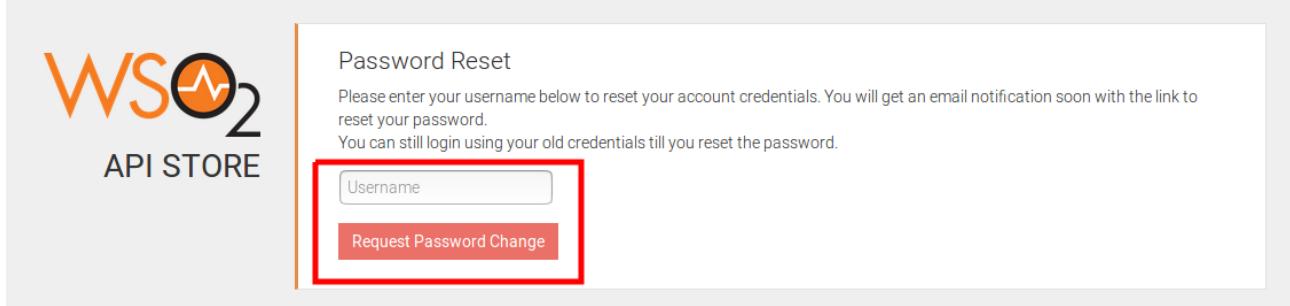
```
<transportSender name="mailto"
class="org.apache.axis2.transport.mail.MailTransportSender">
```

3. Now you have the configured set up. Now a user can click on the forgot password link on the login page of the API Store.



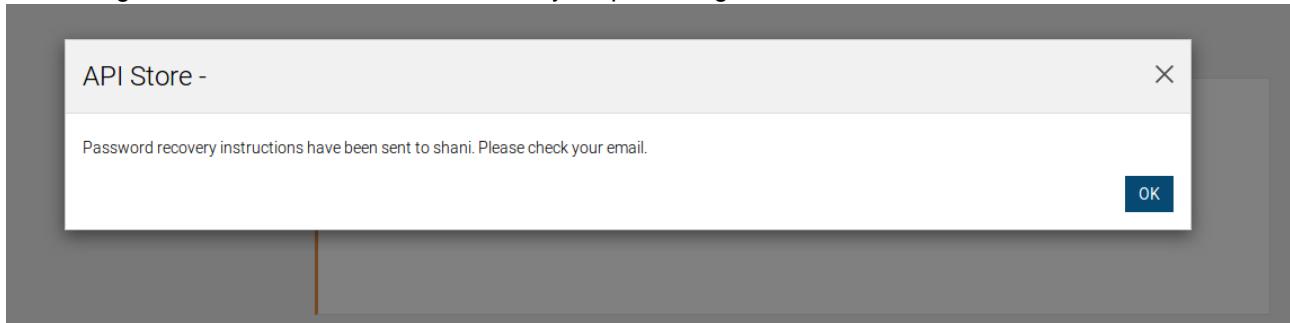
The screenshot shows the WSO2 API Store login page. It features a logo on the left and a central form titled "Sign in to your account". The form contains fields for "Username" and "Password", and buttons for "Sign In", "Go Back", and "Forgot Password". The "Forgot Password" button is highlighted with a red box.

4. You need to have a user account with an email configured for this feature to work. Once you click on the Forgot Password link you will be redirected to add the username of the user you are trying to recover the password of.



The screenshot shows the WSO2 API Store password reset page. It features a logo on the left and a central form titled "Password Reset". The form contains a "Username" field and a "Request Password Change" button. Both the "Username" field and the "Request Password Change" button are highlighted with a red box.

5. Once you add the username and click on Request Password Change, you will receive a notification mentioning that instructions have been sent to your pre configured email address.

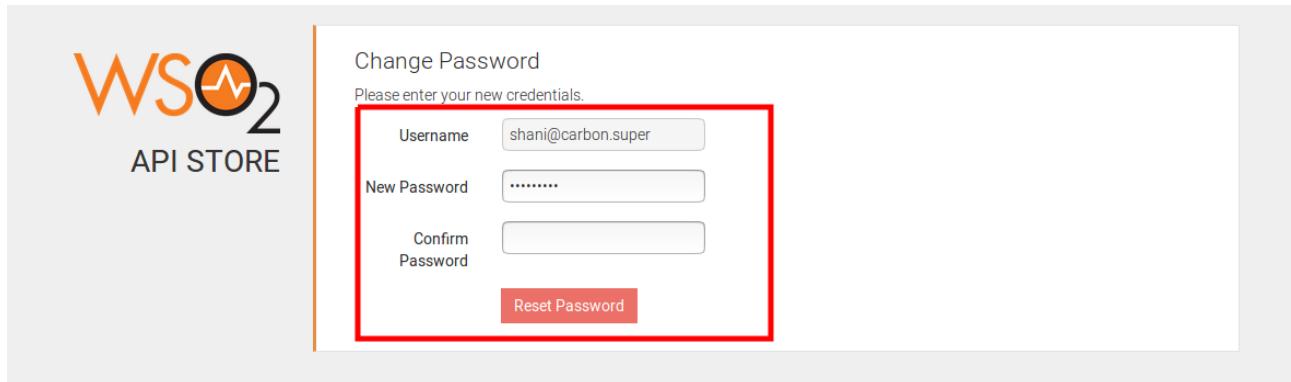


6. You would receive an email with the following information. Click on the link given.

Hi shani  
 We received a request to change the password on the shani account associated with this e-mail address.  
 If you made this request, please click the link below to securely change your password:  
<https://localhost:9443/store/site/pages/reset.jag?confirmation=ccdebe60-eba0-42b8-b562-af8ed24cd542&userstoredomain=PRIMARY&id=shani&tenantdomain=carbon.super>  
 If clicking the link doesn't seem to work, you can copy and paste the link into your browser's address window.  
 If you did not request to have your shani password reset, simply disregard this email and no changes to your account will be made.  
 Best Regards,  
 WSO2 Carbon Team  
<http://www.wso2.com>

If you need to change the template of the email, i.e the email link, wording etc, you need to open the <APIM\_HOME>/repository/conf/email/email-admin-config.xml file and change the template.

7. Once you click on the link, you will be redirected to a change password page, where you can change your password.



- Now you can use the changed password to sign in.

***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, do the following:

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>
```

Go to the **Resources -> Browse** menu in the management console to open the registry and update the credentials in `/_system/governance/apimgt/applicationdata/sign-up-config.xml` registry location.

***How can I recover the admin password used to log in to the management console?***

Use <APIM\_HOME>/bin/chpasswd.sh script.

***How can I manage session timeouts for the management console?***

To configure session timeouts, see [Configuring the session time-out](#).

***How can I add the authentication headers to the message going out of the API Gateway to the backend?***

Uncomment the <RemoveOAuthHeadersFromOutMessage> element in the <APIM\_HOME>/repository/conf/api-manager.xml file and set its value to false.

***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](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>
```

**How to protect my product server from security attacks caused by weak ciphers such as the Logjam attack (Man-in-the-Middle attack)?**

You can disable weak ciphers as described in [Disable weak ciphers](#) in the WSO2 Carbon documentation.

## 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 `<APIKeyValidator>` 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/samlsso</Parameter>
 <Parameter
 name="NameIDPolicyFormat">urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</Parameter>
 <Parameter name="ISAuthnReqSigned">false</Parameter>
 <!--<Parameter
 name="AssertionConsumerServiceURL">https://localhost:9443/acs</Parameter>-->
 </Config>
 </Authenticator>
```

**How can I fix the `javax.net.ssl.SSLException: hostname in certificate didn't match: <ip addrees> != <localhost>` exception?**

### Reason for occurrence

The `javax.net.ssl.SSLException: hostname in certificate didn't match: <ip addrees> != <localhost>`

<localhost> exception is a very common exception that occurs whenever the WSO2 product server is accessed using a different IP address (e.g., <https://10.100.0.77:9443/publisher>) except localhost (e.g., <https://localhost:9443/publisher>).

The reason that the latter mentioned exception occurs is because the self-signed certificate that is shipped with WSO2 products is configured using the hostname as localhost, as a result, Apache Shindig does not allow any other HTTP requests that originate from other hostnames/IP addresses other than localhost.

### Overcoming the issue

You have to create and add a certificate for the IP/domain name in order to overcome this issue. Follow the instructions below:

In the following instructions, assume that you are attempting to add a self-signed certificate for the domain 'foo.com'.

#### Step 1 - Create a self-signed Java KeyStore file and include your domain as the Common Name (CN)

1. Open a terminal and type the following command to generate a KeyStore.

```
keytool -genkey -alias test.foo.com -keyalg RSA -keystore foo.jks
-keysize 2048
```

2. Specify a preferred KeyStore password when prompted.

```
Enter keystore password: <keystore_password>
Re-enter new password: <keystore_password>
```

3. Enter the first name and last name as \*.foo.com and fill out the other information accordingly when prompted.

#### Example

```
What is your first and last name?
[Unknown]: <new_host_name>
What is the name of your organizational unit?
[Unknown]:
What is the name of your organization?
[Unknown]: WSO2
What is the name of your City or Locality?
[Unknown]: Mountain View
What is the name of your State or Province?
[Unknown]: CA
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=*.foo.com, OU=Unknown, O=WSO2, L=Mountain View, ST=CA, C=US
correct?
[no]: yes
```

4. Specify a preferred private Key password when prompted.

```
Enter key password for <keystore_password>
 (RETURN if same as keystore password): <key_password>
Re-enter new password: <key_password>
```

- <key\_password> - Enter the key password that you provided in [step 1.2](#).

This will generate a KeyStore with a private key and a public certificate with CN as \*.foo.com

## Step 2 - Configure the SSL KeyStore

Follow the instructions to configure the WSO2 product with the generated KeyStore:

1. Copy the generated self-signed keystore, namely foo.jks, which was created in [step 1](#), into the <PRODUCT\_HOME>/repository/resources/security directory.
2. Export the public certificate from the keystore and import that certificate to the client--truststore.jks file.
  - a. Navigate to the <API-M\_HOME>/repository/resources/security directory.
  - b. Export the public certificate from the primary KeyStore.

```
keytool -export -alias test.foo.com -file test.foo.com -keystore
foo.jks -storepass <KEYSTORE_PASSWORD_GIVEN_ABOVE>
```

- c. Import the certificate to the client--truststore.jks file.

```
keytool -import -alias test.foo.com -file test.foo.com -keystore
client-truststore.jks -storepass wso2carbon
```

## Step 3 - Update the KeyStoreFile and KeyStorePass parameters of the Tomcat HTTPS connector

1. Change the keystoreFile and keystorePass parameter of the Server.Service.Connector configuration with regard to port 9443 in the <API-M\_HOME>/repository/conf/tomcat/catalina-server.xml file as follows, in order to locate the new SSL KeyStore.

```

<Connector protocol="org.apache.coyote.http11.Http11NioProtocol"
 port="9443"
 bindOnInit="false"
 sslProtocol="TLS"
 sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"
 maxHttpHeaderSize="8192"
 acceptorThreadCount="2"
 maxThreads="250"
 minSpareThreads="50"
 disableUploadTimeout="false"
 enableLookups="false"
 connectionUploadTimeout="120000"
 maxKeepAliveRequests="200"
 acceptCount="200"
 server="WSO2 Carbon Server"
 clientAuth="false"
 compression="on"
 scheme="https"
 secure="true"
 SSLEnabled="true"
 compressionMinSize="2048"
 noCompressionUserAgents="ozilla, traviata"
 compressableMimeType="text/html, text/javascript, application/x-javascript, application/javascript, application/xml, text/css, application/xslt+xml, text/xsl, image/gif, image/jpg, image/jpeg"
 keystoreFile="${carbon.home}/repository/resources/security/foo.jks"
 keystorePass=<KEYSTORE_PASSWORD_GIVEN_ABOVE>
 URIEncoding="UTF-8" />
```

2. Restart the server for the changes to be applicable.

#### Step 4 - Configure the new key store

Update the <Password>, <KeyAlias>, <KeyPassword> values under the <KeyStore> field in the <API-M\_HOME>/repository/conf/carbon.xml file based on your new key store configuration.

```

<KeyStore>
 <!-- Keystore file location-->

 <Location>${carbon.home}/repository/resources/security/foo.jks</Location>
 <!-- Keystore type (JKS/PKCS12 etc.)-->
 <Type>JKS</Type>
 <!-- Keystore password-->
 <Password><KEYSTORE_PASSWORD></Password>
 <!-- Private Key alias-->
 <KeyAlias><NAME_OF_THE_ALIAS></KeyAlias>
 <!-- Private Key password-->
 <KeyPassword><KEY_PASSWORD></KeyPassword>
</KeyStore>
```

**How can I fix the javax.net.ssl.SSLException: Received fatal alert: unknown\_ca error that I get when invoking the methods of an API via the API Console?**

The root cause of this issue is because the default pack is not shipped with a CA-signed certificate. When using the API Console, the web browser sends an HTTPS request to the API Gateway. As the certificate on the Gateway is not CA-signed, the browser does not accept it.

To resolve this issue, first access the Gateway URL via a new browser tab of the same browser and accept the certificate from the browser.

**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](#).

**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>
```

**Why do I get the following error: Gateway Failures?**

The Gateway Failures UI error occurs when the ServerURL, username, password and/or GatewayEndpoint is incorrect. This can be rectified by checking and correcting the latter mentioned gateway configurations under <Environments> in the <API-M>/repository/conf/api-manager.xml file.

If you are using the API-M instance, which you used as the first instance in this tutorial - [Publish through Multiple API Gateways](#), to tryout other tutorials, you will face the above mentioned error, because you updated the environments configurations in that pack by adding two API Gateway environments under the <Environments> element, and commenting the <environment> element that comes by default. To overcome this error in the latter mentioned API-M pack, simply, uncomment the default configuration and delete the newly added configuration under <Environments> in the <API-M>/repository/conf/api-manager.xml file.

#### ***How can I capture the state of a system?***

You can use the tool named Carbon Dump. At the time of an error you can use Carbon Dump (carbondump.sh) to collect all the necessary data (i.e., heap and thread dumps) from a running WSO2 API Manager instance in order to carryout a head dump and thread stack analysis. For more information on using this tool, see [Capturing the state of the system](#) in the Administration guide.

### **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.

In addition, 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](#).

#### ***Does WSO2 API-M support HTTP pipelining?***

No, currently WSO2 API-M does not support [HTTP pipelining](#).

## Sample Scenarios

- API Development
- API Governance
- API Lifecycle Management
- API Rate Limiting
- API Rate Monetization
- API Security Sample
- API Versioning

### API Development

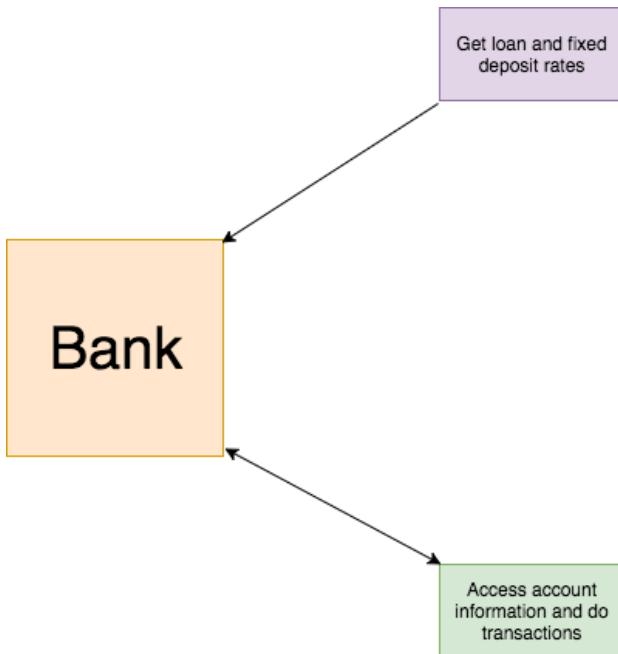
- Developer Optimized APIs Development - Sample Documentation
- Managing Public, Partner vs Private APIs Sample Documentation
- Ownership, permission and collaborative API development - Sample Documentation

#### Developer Optimized APIs Development - Sample Documentation

##### *Sample Use Case*

- Business APIs can be accessed by different parties via different devices. Providing an optimized and personalized experience for these different user stories is key to the success of digital transformation.
- The same API can be accessed by different clients. E.g., mobile devices, PC, TV etc.
- The API developer should be able to optimize API output by identifying the client
- Optimization can be a composition of the multiple backend or stripdown.
- APIs are prioritized based on the client

##### *Business Scenario*



For example, let's take ABC organization which is a bank which is providing financial facilities such as loans, fixed deposits, etc. For these financial services they have a requirement to produce the loan and fixed deposit rates. They have an online facility for clients to log in and check their account information and perform transactions. Their requirement extends to developing a mobile banking solution as well.

##### *How this Business Scenario can be Achieved Using WSO2 API Manager?*

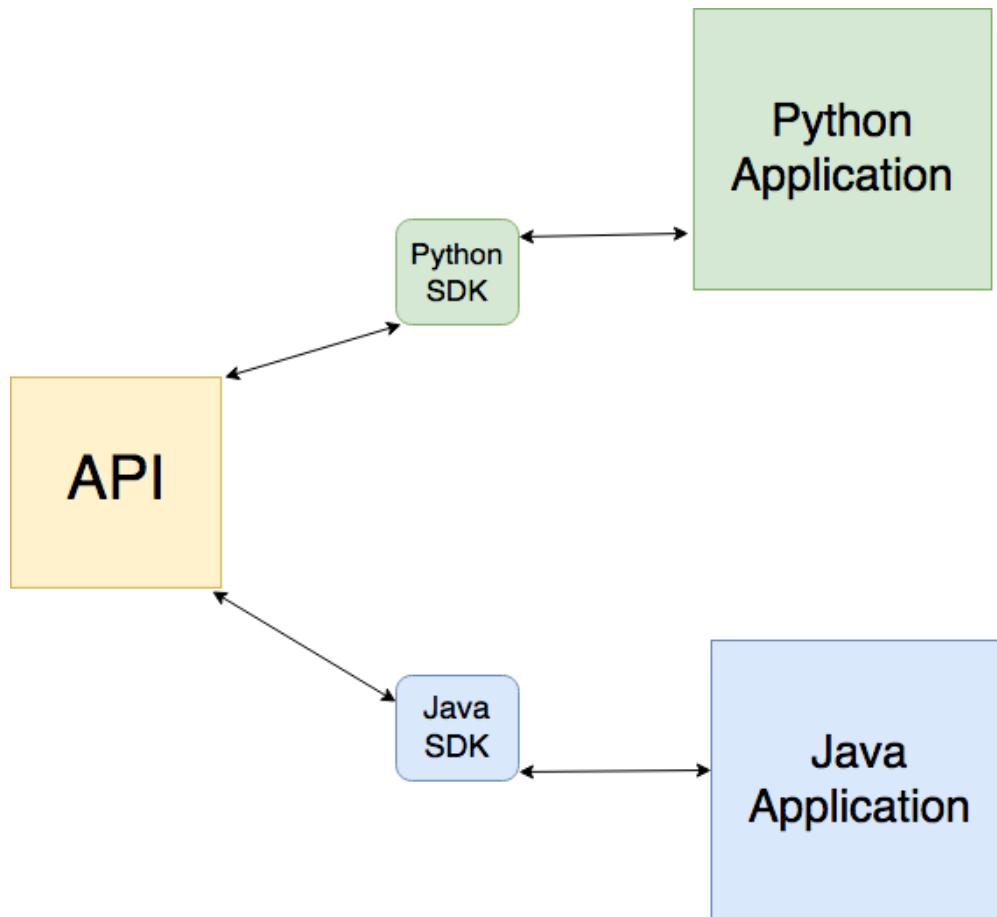
For the above sample business scenario, the organization will need an API to

1. Expose the exchange rates
2. Send clients' relevant account information
3. Grant access to the online banking solution as well as the mobile banking solution

In the future, they will need more and more functionality exposed through APIs, but as for now, let's consider only the above mentioned three APIs and consider how we could achieve this using WSO2 API Manager.

#### ***Business APIs accessed by different parties via different devices***

WSO2 API Manager provides an SDK feature that generates SDKs for a variety of programming languages. This particular feature will address the first part of the sample scenario, i.e. "Business APIs can be accessed by different parties via different devices", where different SDKs can address the needs of the different types of consumers involved.



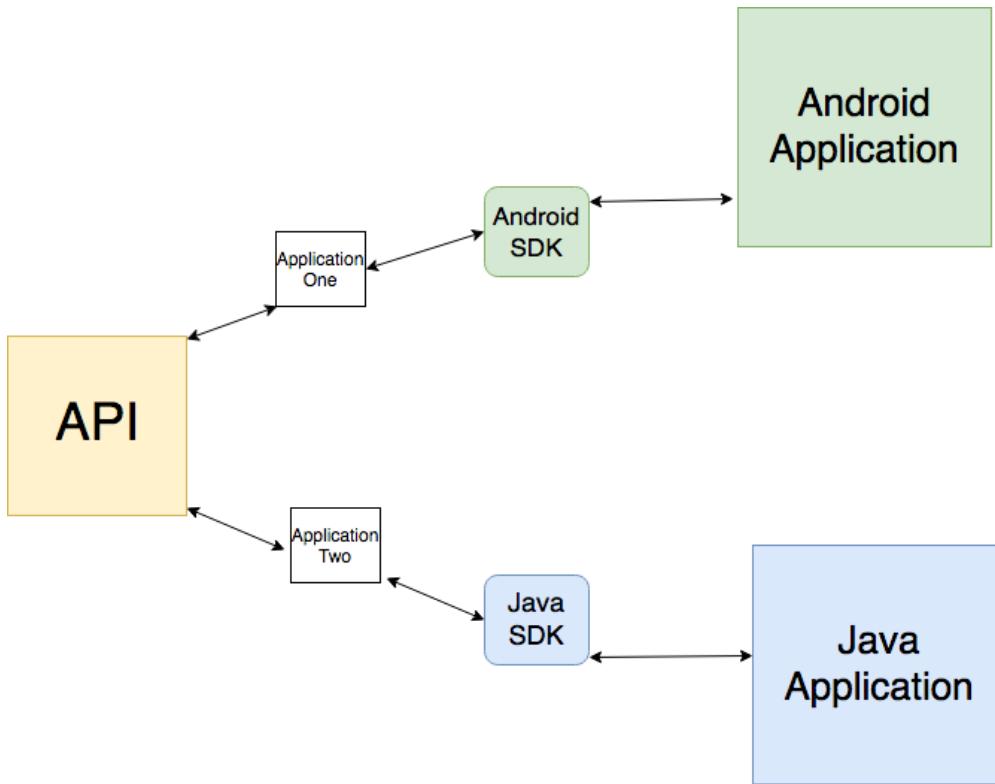
Here the API consumers can create different clients using the programming language they prefer. For an example, let's consider two parties that need an application that consumes the APIs using Python and Java. They can develop two applications using the SDK's for Python and Java separately. The SDK, which is automatically generated through WSO2 API Manager, will handle the API invocation logic while the application developer only needs to focus on the business logic of the client and call the method with the respective parameters in the SDK.

#### ***Same API accessed by different clients***

In WSO2 API Manager, clients can create separate Applications for the same API and access the API. In this example, different users can get bank loan and fixed deposit rates by subscribing to the Applications that provide relevant APIs which the bank will be exposing to the public.

Furthermore, the client may be accessing this using a mobile device. For this they can develop the client application

using android SDK. same as for a java application they can get the java SDK and develop the client application by calling the SDK methods.



#### *API developer optimizing the API output by identifying the client*

API developers can use the headers sent by the client and detect the type of the client (e.g., if the client is a mobile device or not). For example, consider a client who is accessing the banks API's and performing online account balance checking. In this scenario, if the API developer needs to restrict the mobile device clients from accessing all of the account information, instead restrict access to a select set of data of the account, and expose all the data to desktop application clients, that can be achieved by having a custom sequence in the API definition. For more information on adding custom sequences, see [Adding Mediation Extensions](#).

Consider a scenario where the developer needs to restrict sending the address element for the mobile device clients.

```

<xml>
<name>Chris Smith</name>
<address>No 4, Street One, San Francisco.</address>
<mobile>0718123456</mobile>
</xml>

```

The following custom sequences can be added to the API definition to achieve this. In the IN-sequence, it reads the User-Agent header and store it in a custom property in API Manager

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="custom-in">
 <description> Description</description>
 <property name="ClientAccessDevice" expression="$trp:User-Agent" />
</sequence>
```

In the custom OUT-sequence, a filter is introduced to read that property and send the custom payload to the back end as follows,

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="custom-out">
 <description>Custom out sequence to remove Address
element</description>
 <filter source="get-property('ClientAccessDevice')" regex="iphone">
 <then>
 <log level="custom">
 <property name="ClientAccessDevice" value="then" />
 </log>
 <script language="js">
 var payload = mc.getPayloadXML();
 delete payload.address;
 mc.setPayloadXML(payload);
 </script>
 </then>
 <else>
 </else>
 </filter>
 <property name="Client user device" expression="$trp:User-Agent" />
 </sequence>
```

The above API can be tested using the following CURL commands.

### **Mobile Device Client**

```
curl -X GET --header 'User-Agent: iphone' 'Accept: application/json'
--header 'Authorization: Bearer <key>'
'https://10.100.5.168:8243/information/1.0.0/getCustomerInfo'
```

Output for the mobile devices will be as follows,

```
<xml>
<name>Chris Smith</name>
<mobile>0718123456</mobile>
</xml>
```

## Desktop Device Client

```
curl -X GET --header 'User-Agent: chrome' 'Accept: application/json'
--header 'Authorization: Bearer <key>'
'https://10.100.5.168:8243/information/1.0.0/getCustomerInfo'
```

Output for the desktop devices will be as follows,

```
<xml>
<name>Chris Smith</name>
<address>No 4, Street One, San Francisco.</address>
<mobile>0718123456</mobile>
</xml>
```

## *Optimization can be a composition of the multiple backend or stripdown*

This is currently identified as a gap in the API manager 2.1.0 implementation and a new feature addressing this via API composition is being developed in WSO2 Carbon 5.0 based release of the API Manager.

## *Client based prioritization of the APIs*

This is currently identified as a gap in the API Manager 2.1.0 implementation, and will be addressed in the new features that will be delivered in the WSO2 Carbon 5.0 based release of the API Manager.

## Managing Public, Partner vs Private APIs Sample Documentation

### Usecase

- Enable the internal use of APIs
- Help pick and choose subset of those to be used with partners
- Enabling building API ecosystems with partners that can unlock partnerships that cross industries (book a hotel and a car when you book a flight)
  - Connect with APIs from partners
  - Enable partners to connect with ours
- A further subset to be exposed as public APIs. Many of the same APIs used internally and with partners can be used as public APIs to drive additional business and help obtain new customers.

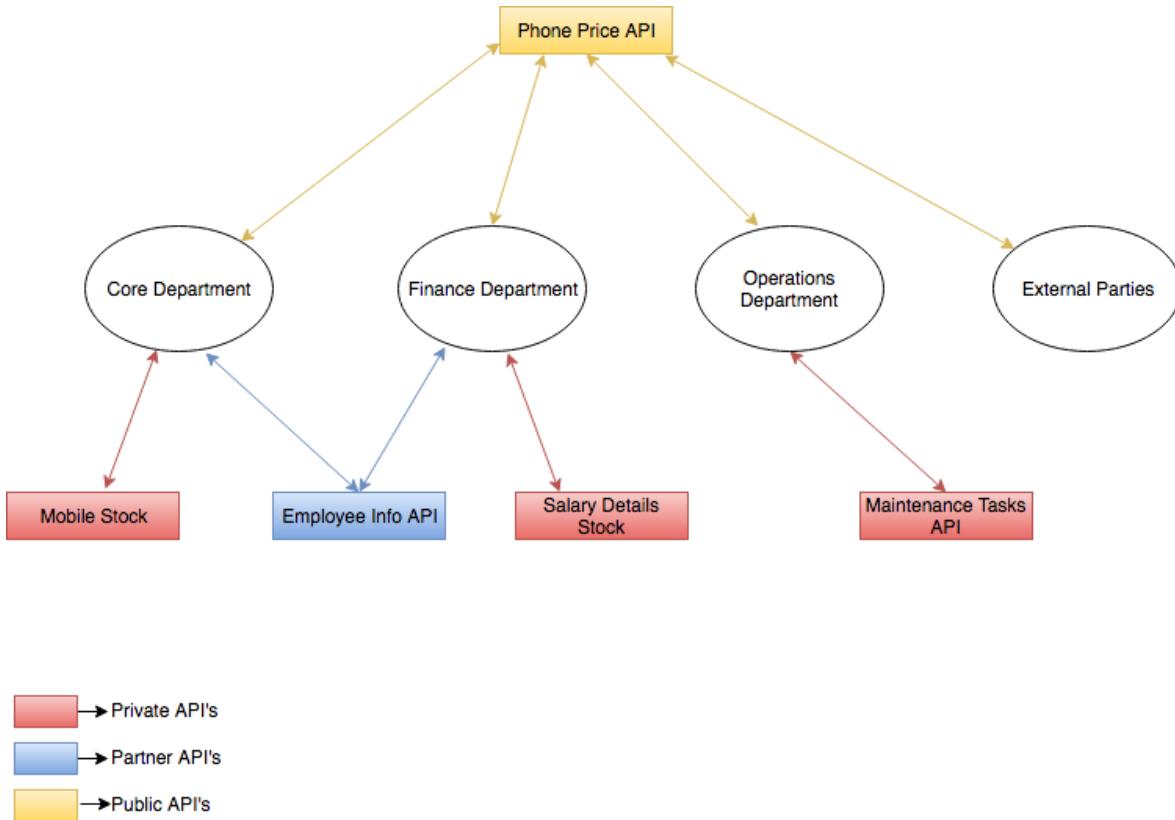
The sample scenario is mentioned below.

### Business story

- ABC organisation is an organisation that has separate departments for finance and operations other than their core business department which is mobile phone manufacturing.
- The core business department is called `department_core` and the finance department is called `depratment_finance` and operations department is named `department_operations`.
- The finance department handles the salaries of employees. The core department is responsible for maintaining the mobile phone manufacturing stock. The operations department handles maintenance work of the organization.

Description of each API is as follows.

1. Employee\_info\_API - Used by core and finance departments.
2. Mobile\_stock\_API - An API used to get current mobile stock details. Used by the core department.
3. Salary\_details\_API - An API used to get the salary of employees. Used by the finance department.
4. Maintenance\_Task\_API - An API used to get maintenance tasks required for the day.



### Business use cases

1. For the above mentioned business scenario, department\_finance needs to get the salary of each and every employee. This data is private to the Finance Department.
2. The department\_core needs to know details about stocks such as the current stock price. This is private to the Core Department.
3. Both Finance and Core departments need to know the employee details of employees working in Core department. This data is only provided to the Core and Finance departments, and the Operations department should not be able to access this data.
4. The department\_core and the public needs to know the current prices of the mobile phones.
5. The operations department needs to know the maintenance tasks required for the day.
6. When a public user gets a phone price of a phone in Brand "A" and model "B", that user should be able to predict the prices of the pouches for the relevant model phone.

### Sample Business Use Cases Related to Sample One

#### API Development - Managing Public, Partner vs Private APIs

The following define how the business story is related to a real world business scenario

- Using some of the APIs internally.
- Share a subset of those APIs with partners if needed.
- This will facilitate to develop an API ecosystems with partners. It will be helpful to maintain partnerships across industries.
- This enables to connect with partners while allowing them to connect with our APIs.
- Moreover, some APIs can be exposed as public APIs to expand the business operations.

### **How this Business Scenario can be Achieved Using WSO2 API Manager**

In API manager we need to create

- Three different tenants for the three departments with users that can create APIs
- An API visible only to the tenant relevant of finance department to get employee salary details.
- An API visible only to the tenant relevant of core department to get current mobile stock details
- An API visible only to the tenant relevant of operations department to get required maintenance task for the day.
- An API subscription visible only to the finance and core departments but restricted to operations departments, to get the employee details of employees working in the core departments.
- An API publicly visible, to get mobile phone prices by exposing a api to mobile phone prices. This API should be tagged as “mobile”. An API to get the prices of the mobile pouches. This should also have the tag “mobile”. In this scenario when an application client calls the mobile phone prices it will call another API to the API's which has the same tag and get the relevant results. (When a user buys a mobile, matching pouches are predicted to buy.)

Depending on the situation, we can specify the "Visibility on Store" as Public (the API is accessible to everyone) or Restricted by roles (the API is visible only to specific user roles in the tenant store that you specify).

### **Running the sample to populate the sample data**

- Download the [wso2am-2.1.0-updateX-sample-scenarios](#).
- Unzip and Copy sample-scenarios folder to <APIM\_HOME> folder.
- Start the wso2am-2.1.0-updateX distribution.
- Go to <API-M\_HOME>/sample-scenarios. Execute the `run.sh` file. Enter the scenario number as 1, when prompted.

### **User credentials needed for login**

User	Username	Password
Finance department user	<a href="mailto:chris@finance.abc.com">chris@finance.abc.com</a>	123123
Manufacturing department user	<a href="mailto:alex@core.abc.com">alex@core.abc.com</a>	123123
Maintenance department user	<a href="mailto:sam@operations.abc.com">sam@operations.abc.com</a>	123123

The screenshots below show how to create tenants and how to control API visibility.

Created APIs.

The screenshot shows the WSO2 API Store interface. On the left, there's a sidebar with icons for APIS, APPLICATIONS, FORUM, and ANALYTICS. The main area is titled "APIs" and displays five API entries:

- Employee\_info\_API**: Version 1.0.0, created by admin, rated 5 stars.
- Maintenance\_ask\_API**: Version 1.0.0, created by admin, rated 5 stars.
- Mobile\_stock\_API**: Version 1.0.0, created by admin, rated 5 stars.
- Phone\_prices\_API**: Version 1.0.0, created by admin, rated 5 stars.
- Salary\_details\_API**: Version 1.0.0, created by admin, rated 5 stars.

## Created Tenants

Tenants List (Number of tenants : 3)

Domain	Email	Created Date	Active	Edit
dpt.finance.com	john@dpt.finance.com	2017/11/06 12:42:43	<input checked="" type="checkbox"/>	Edit
dpt.core.com	tom@dpt.core.com	2017/11/06 12:42:45	<input checked="" type="checkbox"/>	Edit
dpt.opt.com	bob@dpt.opt.com	2017/11/06 12:42:45	<input checked="" type="checkbox"/>	Edit

The screenshot below shows that the user Chris can subscribe to the Salary\_details\_API since Chris belongs to the finance department.

The screenshot shows the details for the "Salary\_details\_API - 1.0.0". The API icon is a pink square with a white letter "S". The details panel shows:

- Version: 1.0.0
- By: john@finance.abc.com
- Updated: 09/Nov/2017 19:54:34 PM IST
- Status: PUBLISHED
- Rating: 5 stars

On the right, there's a "Subscribe" button with a bell icon.

The screenshot below shows that Chris can not subscribe to Mobile\_stock\_API because Chris does not belong to the core manufacturing department.

The screenshot shows the details for the "Mobile\_stock\_API - 1.0.0". The API icon is a red square with a white letter "M". The details panel shows:

- Version: 1.0.0
- By: tom@core.abc.com
- Updated: 09/Nov/2017 19:54:34 PM IST
- Status: PUBLISHED
- Rating: 5 stars

On the right, there's a "Subscribe" button with a bell icon.

The screenshot below shows that user Alex can subscribe to the Mobile\_stock\_API since Alex belongs to the core manufacturing department.

Mobile\_stock\_API - 1.0.0

Version: 1.0.0  
By: tom@core.abc.com  
Updated: 09/Nov/2017 19:54:34 PM IST  
Status: PUBLISHED  
Rating: ★★★★☆

Applications: DefaultApplication  
Tiers: Unlimited

[Subscribe](#)

The screenshots below shows that both Chris and Alex can subscribe to Employee\_info\_API since both of them are given privilege to access the Employee\_info\_API.

Employee\_info\_API - 1.0.0

Version: 1.0.0  
By: admin  
Updated: 09/Nov/2017 20:09:57 PM IST  
Status: PUBLISHED  
Rating: ★★★★☆

Applications: DefaultApplication  
Tiers: Unlimited

[Subscribe](#)

Salary\_details\_API - 1.0.0

Version: 1.0.0  
By: john@finance.abc.com  
Updated: 09/Nov/2017 19:54:34 PM IST  
Status: PUBLISHED  
Rating: ★★★★☆

Applications: DefaultApplication  
Tiers: Unlimited

[Subscribe](#)

You can invoke and check the API's giving results after subscribing to the relevant API's and generating the keys.

### Ownership, permission and collaborative API development - Sample Documentation

#### Use case

- Each business unit is responsible for their APIs and data. Some of them contain sensitive data. They need to consider security and controllability while providing value to the other business units.
- Ability to restrict API (and its development) by business units e.g., APIs developed by financial business unit should be able to restrict (view/edit) to other business units
- Selected APIs should be able to be shared with different business units (edit and view)

#### Business Story

- ABC organisation is an organisation that has separate departments for finance and operations other than their core business department which is mobile phone manufacturing(manufacturing department).
- The core business department is called department\_core and the finance department is called department\_finance and operations department is named department\_operations.
- The finance department handles the salaries of employees. The core department is responsible for

maintaining the mobile phone manufacturing stock. The operations department handles maintenance work of the organization

#### **How this business scenario can be achieved Using WSO2 API Manager ?**

In WSO2 API manager we need to create

- Three different tenants([finance.abc.com](#), [core.abc.com](#) and [operations.abc.com](#)) for the three departments with users(chris, alex and sam respectively) that can [create APIs](#)
- An [API subscription](#) can be performed to the tenant relevant of finance department to get employee salary details. ([Salary\\_details\\_API](#))
- An API subscription can be performed to the tenant relevant of core department to get current mobile stock details. ([Mobile\\_stock\\_API](#))
- An API subscription can be performed to the tenant relevant of operations department to get required maintenance task for the day. ([Maintenance\\_ask\\_API](#))
- An [API subscription only visible](#) to finance and manufacturing departments but restricted to operations departments to get the number of employees working in the core departments. ([Employee\\_info\\_API](#))(From this we can share/restrict the consumption of the API in store, But we can not share/restrict edit of the api in publisher. This has been identified as a GAP in API Manager 2.1.0)

#### **Running the sample to populate the sample data**

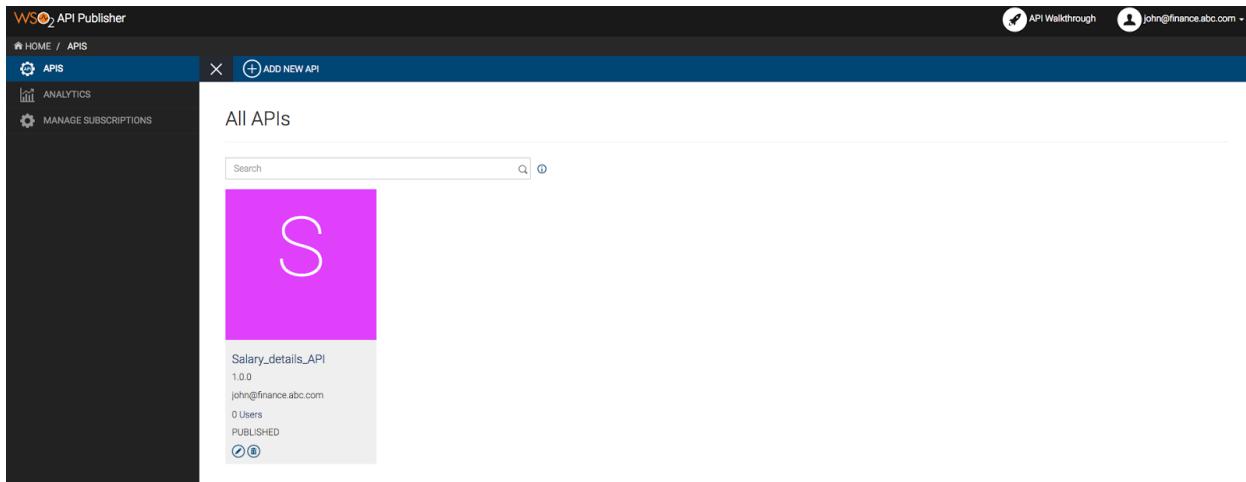
- Start the wso2am-2.2.0-updateX distribution.
- Go to <API-M\_HOME>/sample-scenarios. Execute the `run.sh` file. Enter the scenario number as 2, when prompted.

#### **User credentials needed for login to API Manager instance which the sample data is populated**

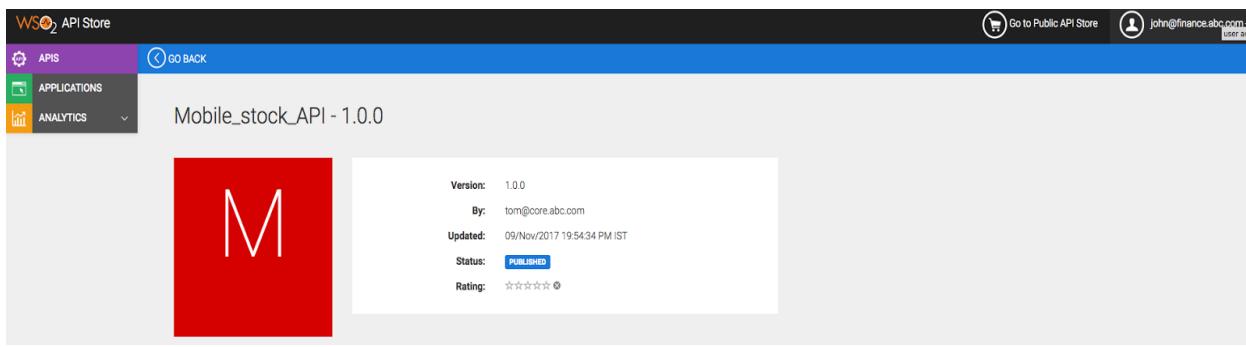
User	Username	Password
Finance department user	<a href="mailto:chris@finance.abc.com">chris@finance.abc.com</a>	123123
Manufacturing department user	<a href="mailto:alex@core.abc.com">alex@core.abc.com</a>	123123
Maintenance department user	<a href="mailto:sam@operations.abc.com">sam@operations.abc.com</a>	123123

The screenshot below shows that user Chris can subscribe to the [Salary\\_details\\_API](#) since Chris belongs to the finance department.

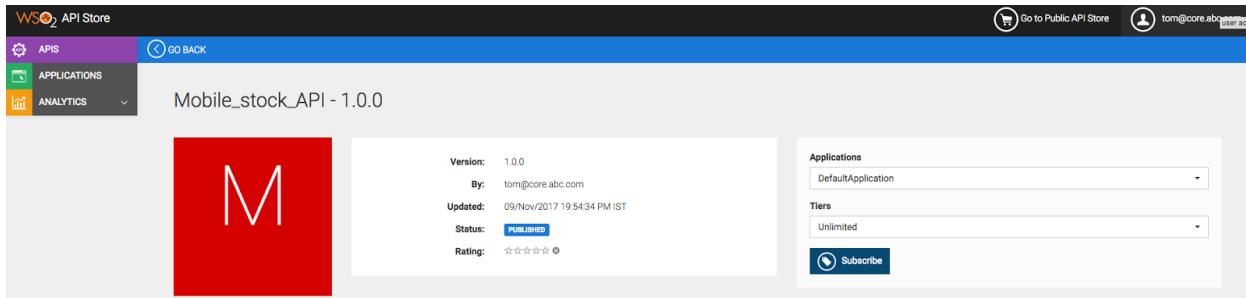
The screenshot below shows that Chris can see the API in the publisher view and develop it.



The screenshot below shows that Chris can not subscribe to Mobile\_stock\_API since Chris does not belong to the core manufacturing department.



The screenshot below shows that user **Alex** can subscribe to the Mobile\_stock\_API since Alex belongs to the core manufacturing department.



The screenshot below shows that Alex can develop the Mobile\_stock\_API since Alex also belongs to core manufacturing department.

All APIs

Mobile\_stock\_API  
1.0.0  
tom@core.abc.com  
0 Users  
PUBLISHED  
Edit Delete

The screenshots below show that both Chris and Alex can subscribe to Employee\_info\_API since both of them are given privilege to access the Employee\_info\_API.

Employee\_info\_API - 1.0.0

Version: 1.0.0  
By: admin  
Updated: 09/Nov/2017 20:09:57 PM IST  
Status: PUBLISHED  
Rating: ★★★★★

Applications: DefaultApplication  
Tiers: Unlimited  
Subscribe

Salary\_details\_API - 1.0.0

Version: 1.0.0  
By: john@finance.abc.com  
Updated: 09/Nov/2017 19:54:34 PM IST  
Status: PUBLISHED  
Rating: ★★★★★

Applications: DefaultApplication  
Tiers: Unlimited  
Subscribe

## API Governance

### Usecase

- Control and track the broader operational characteristics of how APIs get exposed.
- Manage and maintain policy characteristics such as metering, SLAs, availability and performance.
- Policy management specific to different partners and developers.
- People and persona-driven governance models (who can do what and when).
- Dependency analysis; track which services fuel which APIs and which APIs fuel which apps.

## Business story

- Assume that there is an organization that needs to get data related to their API usage. They need to give more traffic to the API's that are accessed frequently. Out of those APIs, they need to give more traffic to a particular resource in a particular API.
- This organization has a separate unit that needs traffic management policies to manage their API's separately.
- There are users that only need to invoke the APIs. They are not offered API developer capabilities.
- They need to know how the services are consumed by the end users.

## Business use cases

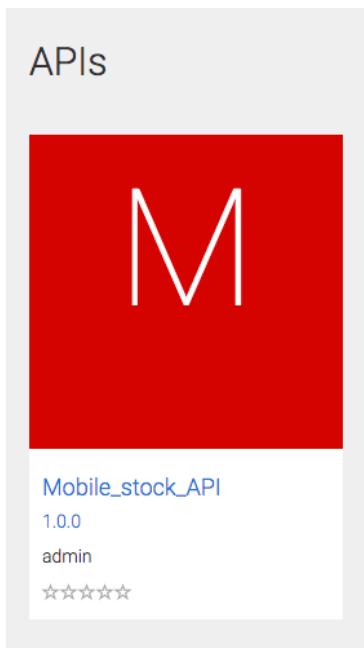
- ABC company is a mobile phone manufacturing company. Assume they had a requirement to publish mobile phone stock availability through an API and they need to give more traffic availability to this API's `getStock` resource.
- They have a separate finance department to which they need to allocate less traffic since this API is only accessed by the finance department.
- They have users that do not have privilege to develop the APIs and only have rights to consume the APIs.
- They need to monitor the API usage, to manage the traffic allocations of the APIs.

## How this business scenario is achieved using WSO2 API Manager

- Place the `wso2am-analytics-2.2.0-updatex` pack in the same location as the `wso2am-2.2.0-updatex` pack.
- Custom advanced throttling policies for the APIs.
- A separate tenant is required for the finance department and custom advanced throttling policies are required for that tenant.
- Two APIs; one for the super tenant, that exposes the mobile phone prices and the other for the tenant created for the finance department, which is a private API that retrieves salary details of the employees.
- Engage the new advanced throttle policies with the two newly created APIs above.
- Create a user who does not have permission to the Publisher.
- Invoke the APIs and check the analytics graphs to check the API usability statistics.

Below are the screenshots that show the old and new APIs with their respective lifecycle states.

Created API for super tenant



Created API for finance department

The screenshot shows the 'APIs' section of the WSO2 API Manager. A large pink placeholder image with a white letter 'S' is visible. Below it, the details for the 'Salary\_details\_API' are listed:

- Name:** Salary\_details\_API
- Version:** 1.0.0
- Owner:** John@finance.abc.com
- Rating:** ★★★★☆

Engaged advance throttle policy for the stock GET request in super tenant

The screenshot shows the 'Select Policy per Resource' dialog box. It lists a single resource entry:

Resource	Policy
GET /stock/{id}	100KPerMin

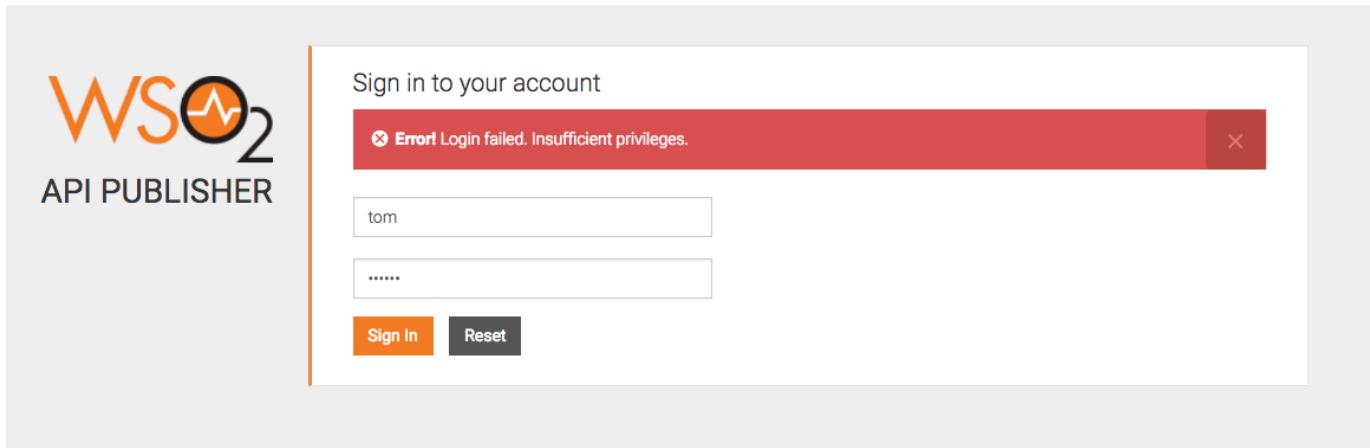
Below the table, there are settings for 'Response Caching' (Disabled) and 'Subscriptions' (Available to current tenant only). A 'Close' button is at the bottom right.

Alex can log in to the Store

The screenshot shows the 'APIs' section again. A large red placeholder image with a white letter 'M' is visible. Below it, the details for the 'Mobile\_stock\_API' are listed:

- Name:** Mobile\_stock\_API
- Version:** 1.0.0
- Owner:** admin
- Rating:** ★★★★☆

Alex cannot log in to the Publisher since it has been restricted



Advanced throttling policy engaged to finance department

A screenshot of the WSO2 Admin app interface. A modal dialog box titled "Select Policy per Resource" is open. It contains a table with two columns: "Resource" and "Policy". The resource listed is "GET /stock/{id}" and the policy selected is "5KPerMin". There are "Close" and "Manage" buttons at the bottom right of the dialog. The background shows a list of resources like "Salary\_d" and "Configurations".

Add advanced throttling policies in the Admin app

# Edit Advanced Throttle Policy

## General Details

Tier Name: \* ?

100KKBPerMin

Tier Description: ?

Allows 100000 kilo bytes per minute

## Default Limits

Request Count  Request Bandwidth

Data Bandwidth: \* ?

100000

KB

Unit Time: \* ?

1

Minutes(s)

## Conditional Groups



Add Conditional Group

# Edit Advanced Throttle Policy

## General Details

Tier Name: \* ?

100KPerMin

Tier Description: ?

Allows 100000 requests per minute

## Default Limits

Request Count  Request Bandwidth

Request Count: \* ?

100000

Unit Time: \* ?

1

Minutes(s)



## Conditional Groups



Add Conditional Group

## Advanced Throttling Policies

 Filter by ...

Name	<input type="checkbox"/> Quota Policy	<input type="checkbox"/> Quota	<input type="checkbox"/> Unit Time	
100KKBPerMin	bandwidthVolume	100000 KB	1 min	<a href="#"></a> Edit <a href="#"></a> Delete
100KPerMin	requestCount	100000	1 min	<a href="#"></a> Edit <a href="#"></a> Delete

After users start invoking APIs, statistics appear in the Publisher, as shown below:

## API Usage by Resource Path

All APIs Hour Day Week Month 2017-10-23 00:00:00 to 2017-11-23 13:51:59

Filter by ...

api	version	resourcePath	method	Hits
Mobile_stock_API	1.0.0	/stock/{id}	GET	41

Show 10 entries | Showing 1 to 1 of 1 entries

## API Last Access Times (Across All Versions)

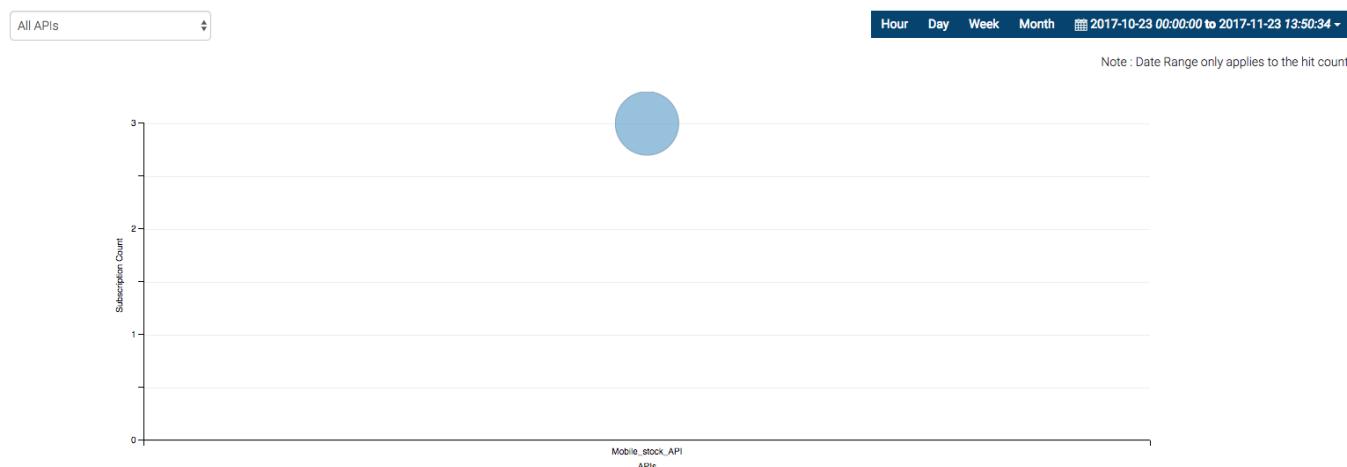
All APIs Hour Day Week Month 2017-10-23 00:00:00 to 2017-11-23 13:51:59

Show 10 entries

API	Version	Subscriber	Access Time (GMT+05:30)
Mobile_stock_API (admin)	1.0.0	tom@carbon.super	11/23/2017, 1:49 pm

Showing 1 to 1 of 1 entries 1

## Overall API Usage (Across All Versions)



## Running the sample to populate the sample data

- Start the wso2am-analytics-2.2.0-updateX distribution.
- Start wso2am-2.2.0-updateX, after starting the APIM analytics node.
- Go to <API-M\_HOME>/sample-scenarios. Execute the run.sh file. Enter the scenario number as 9, when prompted.

## User credentials needed for log in

User	Username	Password
Super tenant	admin	admin
Store only	alex	123123
Finance department user	chris@finance.abc.com	123123

## References

<https://wso2.com/library/articles/2016/09/article-generating-insights-with-wso2-api-manager-analytics/#apistore>

# API Lifecycle Management

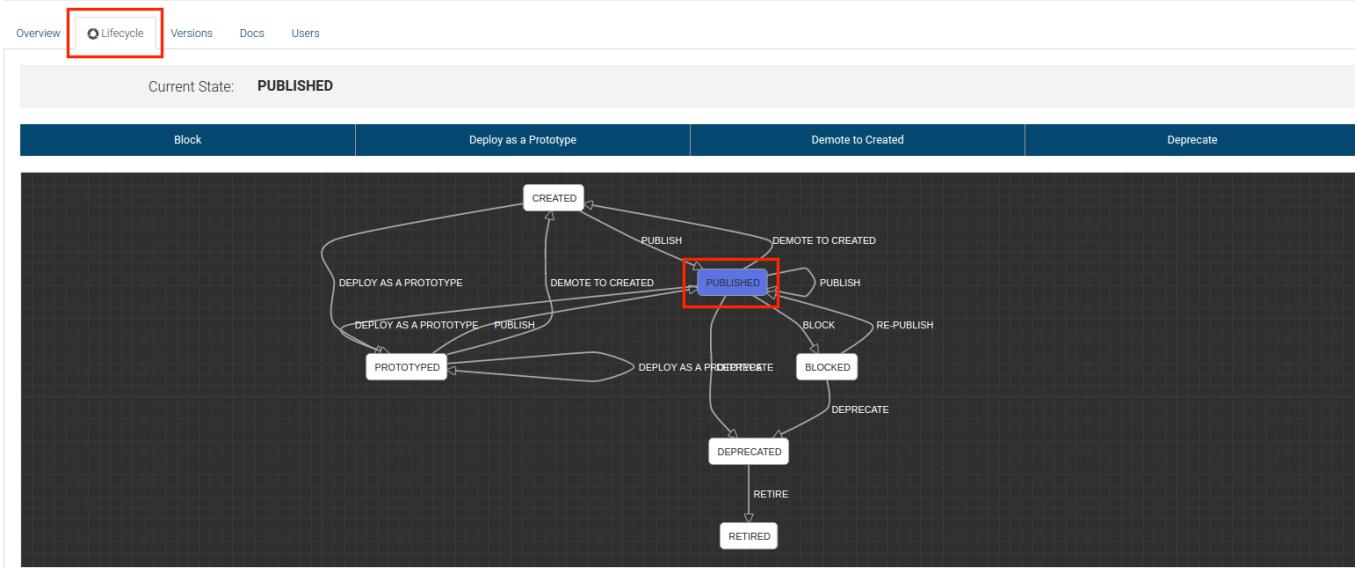
## Usecase

- Ability to run full lifecycle API Management from the inception stage of an API until it's retirement.
- Notification mechanisms for informing developers on API changes.
- Application lifecycle management mechanisms in sync with API lifecycle management.
- Introduce and execute organization specific lifecycle states.

## Business story

The API lifecycle is one of the key factors in API development. An API lifecycle has predefined states. These states represent the stages that an API has in the process of starting to develop an API until its retirement. The following is a diagram that describes the current states of an API lifecycle.

PizzaShackAPI - 1.0.0



## API lifecycle states

State	Description
PUBLISHED	The API is ready to be used by users in the API Store.
CREATED	The API has been created, but it is not available for usage.
PROTOTYPED	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.
BLOCKED	The API is temporarily blocked from being used. A publisher can publish the API from the BLOCKED state.

DEPRECATED	The old version of the API is DEPRECATED when a newer version of the API is created and PUBLISHED.
RETIRED	The API is no longer in use and has been moved to the RETIRED state.

### Business use cases

Assume that ABC is a mobile phone manufacturing company that needs the following:

- Employee salary details
- Employee personal details
- Available stock details
- Produce sales promotions
- Stop offering promotions when they capture the market

How can this business scenario be achieved using WSO2 API Manager?

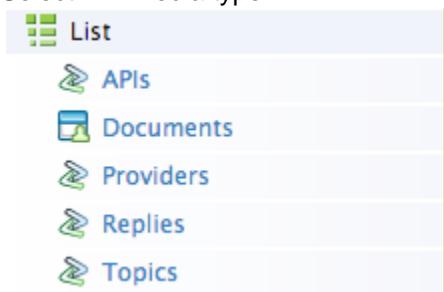
Separate APIs are required for the following scenario:

- API needs to get employee salary details while in the PUBLISHED state.
- API to the is still not published which is in created state to get employee personal details in CREATED state
- Old API to get available Stock details in DEPRECATED STATE
- A New API version of the API which is to get available Stock details PUBLISHED state
- API to produce sales\_promotions which will be in PUBLISHED state and BLOCKED state when there are no promotions available.
- API developers must get notifications when a specific API is promoted to published state. This point is by subscribing to the lifecycle state change notification. How to subscribe will be explained below.

For details on how to enable notifications, see [Enabling Notifications](#).

Follow the steps below to configure API lifecycle state notifications.

1. Login to carbon console (<https://<hostname>:9443/carbon>).
2. Select API media type.



3. Select the API from the list.

Provider	Name	Version	Lifecycle Status	Actions
admin	Mobile_stock_API	1.0.0	APILifeCycle / Deprecated	
admin	Employee_Info_API	1.0.0	APILifeCycle / Created	
admin	Sales_promotions_API	2.0.0	APILifeCycle / Blocked	
admin	Sales_promotions_API	1.0.0	APILifeCycle / Retired	
admin	Mobile_stock_API	2.0.0	APILifeCycle / Rejected	
admin	Salary_details_API	1.0.0	APILifeCycle / Rejected	

4. Go to the API resource and add a subscription to Change\_LC\_State.

5. Now you will be getting notifications for the API you have subscribed.

The API to produce sales promotions will be in REJECTED state once the organization decides there will be no longer promotions available in future (since we need to represent the sales promotions when the API is in the BLOCKING state as well, we are creating a new API sales\_promotions\_2 to represent the REJECTED state).

#### Running the sample to populate the sample data

1. Go to <API-M\_HOME>/sample-scenarios. Execute the run.sh file. Enter the scenario number as 7, when prompted.

After running the sample two APIs in different states (PUBLISHED and BLOCKED) will be created. Note that both APIs have the same name.

The screenshot shows the WSO2 API Manager interface. At the top, there is a search bar with the text "Sales\_promotions\_API" and a magnifying glass icon. Below the search bar, there are two large, bold letters "S" on a pink background, representing the API names. Underneath each letter, there is a card for an API version:

- Sales\_promotions\_API**  
1.0.0  
admin  
0 Users  
PUBLISHED
- Sales\_promotions\_API**  
2.0.0  
admin  
0 Users  
BLOCKED

#### Related Links

- [Customize the API Life Cycle](#)
- [Eventing and Notifications with WSO2 Governance Registry](#)
- [Adding a Subscription](#)

## API Rate Limiting

### Usecase

- For monetization purposes, to enforce limits on an Application based on its subscriptions
- Enforce fair usage policy among an application's users
- Allow privileged rate limits based on location, device type, user credentials, etc.
- Enforce a peak limit on back-end services to prevent total outages.

### Business Story

- ABC organization has an API to expose their mobile phone prices. This organization currently exposes prices to the sales agents and they have a large number of third party sales agents.
- They need to limit the total number of API calls to 50PerMin for this mobile price API.
- They need to limit the number of requests to 5000, when specific third party agent calls the mobile price API.
- They need to allow API calls coming from a specific location (IP address - 192.168.1.1 to 192.168.1.100) and they need to restrict specific device types such as mobiles.
- Even though they allow a specific number of API calls for a given period of time, they need to limit the API calls that come at a given point in time to restrict the burst requests and to prevent their back end service.

### Business Use Cases

- For the above mentioned business story, ABC organization needs to have an API manager solution to expose their mobile price API.
- They need to limit the number of API calls that invokes by a specific third party agent
- They need to limit the number of API calls that invokes by all the agents in a given time frame

- They need to control the number of requests from specific devices, locations
- They need to control the number of requests at a given time to limit the burst requests.

How can this Business Scenario be achieved using WSO2 API Manager ?

In WSO2 API Manager we need to,

- Create an API to expose their mobile phone prices.
- Create an Application throttle policy to manage the API calls that come from all the third party agents subscribed to that particular application.
- Create a Subscription throttle policy to limit the number of API calls from a specific third party agent.
- Add a Burst Control (Rate Limiting) to the subscription policy to prevent the burst requests that can come in a given time frame.

Running the sample to populate the sample data

- Start wso2am-2.2.0-updateX.
- Go to <API-M\_HOME>/sample-scenarios. Execute the run.sh file. Enter the scenario number as 10, when prompted.

User credentials needed for log in

User	Username	Password
Super tenant	admin	admin

The screenshots below show how this usecase can be implemented.

**Edit Application Level Policy**

**General Details**

Name \*: 50PerMin  
Description: Allows 50 request per minute

**Quota Limits**

Request Count  Request Bandwidth  
Request Count \*: 50  
Unit Time \*: 1 Minute(s)

**Save** **Cancel**

**Add Conditional Group**

**Condition Group** **Custom Conditional Group**

- IP Condition** ✓
- Header Condition** ✓
- Query Param Condition**
- JWT Claim Condition**

**IP Condition Policy** **On**

This configuration is used to throttle by IP address.

IP Condition Type: \* IP Range  
Start IP Address: \* 192.168.1.1 End IP Address: \* 192.168.1.100  
Invert Condition:

**Execution Policy**

Request Count : Request Count  
Request Count: \* 100000  
Time: \* 1 Minute(s)

## Edit Advanced Throttle Policy

### General Details

Tier Name: \* 100KPerMin  
Tier Description: \* Allows 100000 requests per minute

### Default Limits

Request Count  Request Bandwidth  
Request Count: \* 100000  
Unit Time: \* 1 Minutes(s)

### Conditional Groups

**Add Conditional Group**

**Condition Group** **Custom Conditional Group**

## API Rate Monetization

### Usecase

- Defining, enabling and monitoring revenue for APIs.
- Defining monetization tiers and throttling traffic based on these tiers.
- Support for different API subscription models.
  - By call volume
  - By bandwidth consumption
  - By time of day usage
- For backends that are also monetized, create reports and compare against invoices from backend vendors.
- Offer a free trial version and request payment if they wish to continue the service for a long time

### Business Story

- ABC organization has an API to expose their mobile phone prices. Assume this organization exposes prices to the sales agents and they have big number of third party sales agents.
- They need to gain revenue from third party agents based on their subscription models. As an example let's use the following subscription models
- They need to charge for the,
  - Number of request counts
  - Bandwidth consumption

### How can this Business Scenario be achieved using WSO2 API Manager ?

WSO2 API Manager has the following subscription tiers defined by default. You can also add custom subscription tiers.

Name	Quota Policy	Quota	Unit Time	Rate Limit	Time Unit		
Bronze	requestCount	1000	1 min	NA	NA		
Gold	requestCount	5000	1 min	NA	NA		
Silver	requestCount	2000	1 min	NA	NA		
Unauthenticated	requestCount	500	1 min	NA	NA		

We can connect to a billing engine and connect the API manager node to the billing engine to charge according to the API usage and bandwidth consumption. For more details, see [Enabling Monetization of APIs](#). By connecting [WSO2 API Manager to the WSO2 API Manager Analytics server distribution](#), we can identify the API usages along with the billing engine outcomes.

You can create a free subscription tier for free trial version for users, through the approach given above. Once they exceed the tier limit, they can be offered the priced subscriptions which will gain revenue.

## API Security Sample

### Usecase

- Prevent misuse or abuse of information or of any application resources exposed by an API
- Have the ability to distinguish between internal, partner and public use of APIs via security controls and audits

- Ability to trace back which apps are using what APIs (hence data or resources) with which user credentials, permissions and roles
- Ability to support multiple security standards and protocols
  - Embed custom security algorithms globally or for selected services
- Ability to enforce both authentication (are you a valid user) and authorization (are you permitted to perform this action) on APIs

### Business Story

- ABC organization needs to manage their salary details among a set of permitted employees in there finance department.
- The organization is a mobile phone manufacturing company and they have to expose their mobile prices to the public.
- The management needs to secure the services using a custom method.
- The organization has a requirement to back-track who has used the services, and restrict the service usage for employees.

### Business Use Cases

- ABC organization has an API to manage employee salary details. This API should only be accessed by a specific set of people, for a given time period to prevent data misuse.
- This organization needs to secure there APIs for internal, partner and public APIs.
- Let this organization needs to check who did changes to their APIS, and who invoked the APIs and etc..
- Further they have a requirement to add custom security standards.
- They need the users of their system to be authenticated to facilitate that only the permitted set of users are using the system and when the API invocation happens they need to authorize the users whether they are permitted to access the APIs.

### How can this Business Scenario be achieved using WSO2 API Manager?

- We need to create and API to get the salary details of the employees
- We need to have separate tenants to manage their APIs through tenants. For more details, see [Managing Public, Partner vs Private APIs](#)
- We need to enable audit logs to trace the API creations and API invocations.  
For example, to enable custom security algorithms we can use the [Kerberos OAuth2 Grant](#).
- We can authorize the users through API Manager access tokens and we can use scopes to authorise the API consumers when consuming the APIS.

### How to run the sample to populate the above mentioned sample data

- Start wso2am-2.2.0-updateX.
- Go to <API-M\_HOME>/sample-scenarios. Execute the run.sh file. Enter the scenario number as 4, when prompted.

### User credentials needed for log in

User	Username	Password
Super tenant	admin	admin

The audit logs are printed in the <API-M\_HOME>/repository/logs/audit.log file, when you run the sample.

After invoking and creating the APIs the audit logs will be similar to the sample shown below.



```
[2017-12-22 11:48:03,227] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:03,226+0530]
[2017-12-22 11:48:03,341] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:03,339+0530]
[2017-12-22 11:48:09,390] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:09,389+0530]
[2017-12-22 11:48:09,398] INFO - Initiator : admin@carbon.super | Action : Add User | Target : tom | Data : { Roles :Internal/subscriber, } | Result : Success
[2017-12-22 11:48:09,437] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:09,436+0530]
[2017-12-22 11:48:09,458] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:09,457+0530]
[2017-12-22 11:48:10,396] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:10,395+0530]
[2017-12-22 11:48:10,819] Chris@finance.abc.com [1] [AM] INFO - Initiator : Chris | Action : create | Target : 0 | Data : { Chris_Integration_Test_App } | Result : Success
[2017-12-22 11:48:10,941] Chris@finance.abc.com [1] [AM] INFO - Initiator : Chris | Action : update | Target : 1 | Data : { Chris_Integration_Test_App } | Result : Success
[2017-12-22 11:48:11,132] INFO - Initiator : admin | Action : create | Target : 0 | Data : { admin_Integration_Test_App } | Result : Success
[2017-12-22 11:48:11,138] INFO - Initiator : admin | Action : update | Target : 2 | Data : { admin_Integration_Test_App } | Result : Success
[2017-12-22 11:48:12,437] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:12,436+0530]
[2017-12-22 11:48:12,709] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:12,709+0530]
[2017-12-22 11:48:12,813] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:12,813+0530]
[2017-12-22 11:48:12,880] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:12,879+0530]
[2017-12-22 11:48:12,909] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:12,909+0530]
[2017-12-22 11:48:13,116] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,116+0530]
[2017-12-22 11:48:13,207] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,207+0530]
[2017-12-22 11:48:13,234] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,234+0530]
[2017-12-22 11:48:13,293] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,293+0530]
[2017-12-22 11:48:13,342] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,341+0530]
[2017-12-22 11:48:13,425] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,425+0530]
[2017-12-22 11:48:13,484] INFO - 'admin@carbon.super [-1234]' logged in at [2017-12-22 11:48:13,484+0530]
[2017-12-22 11:48:13,503] ERROR - Illegal access attempt at [2017-12-22 11:48:13,0502] from IP address 127.0.0.1 while trying to authenticate access to service EventProcessorAdminService
[2017-12-22 11:48:13,509] INFO - 'admin@carbon.super [-1234]' logged in
```

```

at [2017-12-22 11:48:13,509+0530]
[2017-12-22 11:48:13,594] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:13,593+0530]
[2017-12-22 11:48:14,449] Chris@finance.abc.com@finance.abc.com [1] [AM]
INFO -
{"performedBy":"Chris","action":"created","typ":"API","info":"{\\"provider\\\":\\\"Chris-AT-finance.abc.com\\\",\\\"name\\\":\\\"Salary_details_API\\\",\\\"context\\\":\\\"\\\\\\t\\\\\\finance.abc.com\\\\\\salaries\\\\\\1.0.0\\\",\\\"version\\\":\\\"1.0.0\\\"}"}
[2017-12-22 11:48:14,911] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:14,910+0530]
[2017-12-22 11:48:15,003] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:15,003+0530]
[2017-12-22 11:48:15,175] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:15,175+0530]
[2017-12-22 11:48:15,629] INFO -
{"performedBy":"admin","action":"created","typ":"API","info":"{\\"provider\\\":\\\"admin\\\",\\\"name\\\":\\\"Mobile_stock_API\\\",\\\"context\\\":\\\"\\\\\\stocks\\\\\\1.0.0\\\",\\\"version\\\":\\\"1.0.0\\\"}"}
[2017-12-22 11:48:15,689] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:15,689+0530]
[2017-12-22 11:48:15,722] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:15,722+0530]
[2017-12-22 11:48:15,808] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:15,808+0530]
[2017-12-22 11:48:18,412] INFO -
{"performedBy":"admin","action":"created","typ":"Application","info":"{\\"tier\\\":\\\"Unlimited\\\",\\\"name\\\":\\\"Application_one\\\",\\\"callbackURL\\\":null}"}
[2017-12-22 11:48:18,507] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:18,507+0530]
[2017-12-22 11:48:18,512] INFO - Initiator : admin | Action : create |
Target : 0 | Data : { admin_Application_one_PRODUCTION } | Result : Success
[2017-12-22 11:48:18,514] INFO - Initiator : admin | Action : update |
Target : 3 | Data : { admin_Application_one_PRODUCTION } | Result : Success
[2017-12-22 11:48:18,520] INFO - Initiator : admin | Action : update |
Target : 3 | Data : { admin_Application_one_PRODUCTION } | Result : Success
[2017-12-22 11:48:18,671] INFO -
{"performedBy":"admin","action":"updated","typ":"Application","info":"{\\"Generated keys for application\\\":\\\"Application_one\\\"}"}
[2017-12-22 11:48:18,720] INFO -
{"performedBy":"admin","action":"created","typ":"Subscription","info":"{\\"application_name\\\":\\\"Application_one\\\",\\\"tier\\\":\\\"Unlimited\\\",\\\"provider\\\":\\\"admin\\\",\\\"api_name\\\":\\\"Mobile_stock_API\\\",\\\"application_id\\\":2}"}
[2017-12-22 11:48:18,732] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:18,732+0530]
[2017-12-22 11:48:18,884] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:18,884+0530] from IP address
[2017-12-22 11:48:19,823] INFO -
{"performedBy":"admin","action":"created","typ":"Application","info":"{\\"tier\\\":\\\"Unlimited\\\",\\\"name\\\":\\\"Application_two\\\",\\\"callbackURL\\\":null}"}
[2017-12-22 11:48:19,848] INFO - Initiator : admin | Action : create |
Target : 0 | Data : { admin_Application_two_PRODUCTION } | Result : Success
[2017-12-22 11:48:19,851] INFO - Initiator : admin | Action : update |
Target : 4 | Data : { admin_Application_two_PRODUCTION } | Result : Success

```

```
[2017-12-22 11:48:19,854] INFO - Initiator : admin | Action : update |
Target : 4 | Data : { admin_Application_two_PRODUCTION } | Result : Success
[2017-12-22 11:48:19,888] INFO -
{"performedBy":"admin","action":"updated","typ":"Application","info":"\\"G
enerated keys for application\\":\\"Application_two\\""}
[2017-12-22 11:48:19,904] INFO -
{"performedBy":"admin","action":"created","typ":"Subscription","info":"\\"
application_name\\":\\"Application_two\\",\\"tier\\":\\"Unlimited\\",\\"provider\\"
```

```
:\"admin\", \"api_name\": \"Mobile_stock_API\", \"application_id\":3}"]
[2017-12-22 11:48:19,908] INFO - 'admin@carbon.super [-1234]' logged in
at [2017-12-22 11:48:19,908+0530]
```

When analysing the audit logs

- This log show that a user has been created

```
[2017-12-22 11:48:09,398] INFO - Initiator : admin@carbon.super |
Action : Add User | Target : tom | Data : { Roles
:Internal/subscriber, } | Result : Success
```

- This log show that an application has been created

```
[2017-12-22 11:48:10,819] Chris@finance.abc.com [1] [AM] INFO -
Initiator : Chris | Action : create | Target : 0 | Data : {
Chris_Integration_Test_App } | Result : Success
```

- This log show the API creation of the Salary details API

```
[2017-12-22 11:48:14,449] Chris@finance.abc.com@finance.abc.com [1]
[AM] INFO -
{"performedBy": "Chris", "action": "created", "typ": "API", "info": "{\"provider\": \"Chris-AT-finance.abc.com\", \"name\": \"Salary_details_API\", \"context\": \"/\\\\\\t\\\\\\finance.abc.com\\\\\\salaries\\\\\\1.0.0\", \"version\": \"1.0.0\"}"}
```

These data can be view in WSO2 API Manager Analytics server. For more details, see [API Governance](#).

Assume that the GET resource in the Salary API should be restricted for admin role users. Follow the steps below to restrict the resource for a particular role.

- Create a scope named `new_scope`. Assign scope to the admin role.

The screenshot shows the 'Resources' section of the WSO2 API Manager interface. A new scope named 'new\_scope' is being created for the 'admin' role. The 'Scopes' table shows the new scope entry. Below the table, there is a 'GET /salary/(id)' endpoint listed under 'Application & Application User' with a rate limit of '5KPerMin'. A link to '+Summary' is also present.

- When you execute the sample, it invokes the API without this scope.

```
{
 "fault": {
 "code": 900910,
 "message": "The access token does not allow you to access the requested resource",
 "description": "Access failure for API: /salariesSecure/1.0.0, version: 1.0.0 status: (900910) - The access token does not allow you to access the requested resource"
 }
}
```

You will see the following error in the audit log if the access token does not allow you to access the

requested resource.

WARN - APIAuthenticationHandler API authentication failure

- Executing the sample also invokes the GET resource and return the response as shown below.

Code	Details
200	Response body
<pre>{   "id": 2,   "fixed": 10000,   "allowance": 5000,   "empId": "2" }</pre>	
Response headers	
<code>content-type: application/json</code>	
Responses	
Code	Description
200	

## API Versioning

### Use case

- Ability to retire old APIs and introduce new versions of APIs to enhance functionality.
- Ensure that API updates don't break when they are upgraded, versioned or moved between environments, geographical locations, data centers and the cloud.
- [Alpha and beta testing](#) with old vs new APIs
- Ability to notify about the availability of the new API version to the consumers of the older version.
- Enforcing a grace period to upgrade to the new version of the API

Transferring contracts with app developers to newer versions.

### Business Story

API versioning is a key functionality that needs to be addressed once the users, exposed to a specific API for a long time, need to add, remove or change the API. When doing this there may be users who use the old API. Hence while introducing the new API the old API should also be supported for a specific time period. The time period is offered to guide the users and allow them to switch to the new API.

### Business Use Cases

- ABC company is a mobile phone manufacturing company. They have a requirement to publish mobile phone prices through an API.
- When the industry grows with the prices, they need to publish some additional data such as a rating, user reviews of the mobile phones, etc.
- Users of the old API should know that there is a new API version released, and they need to be notified.

### How can this Business Scenario be achieved using WSO2 API Manager ?

- We need an API to expose the mobile phone details of ABC company which will become the old API, as soon as the new version is created.
- We need a new version of the above mentioned API.
- We need the old API to be in the PUBLISHED state and then once the new API version is created the old API should be DEPRECATED. The new version of the API should be in the PUBLISHED state.
- Subscribers of the old API needs to get notified when the new API version is published.

Below are the screenshots that shows the old and new APIs with there lifecycle states.

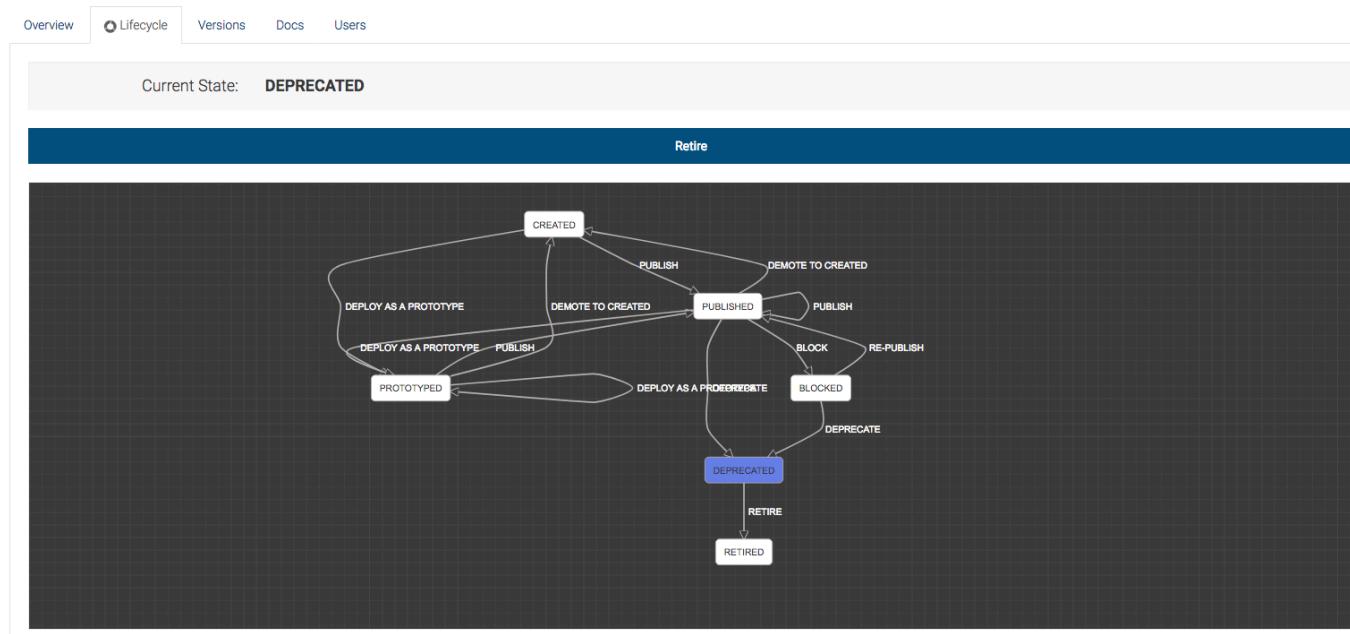
## All APIs

The screenshot shows the 'All APIs' page with a search bar at the top. Below the search bar are two API entries:

- Mobile\_stock\_API** (version 1.0.0): Admin, 0 Users, DEPRECATED. Includes edit and delete icons.
- Mobile\_stock\_API** (version 2.0.0): Admin, 0 Users, PUBLISHED. Includes edit and delete icons.

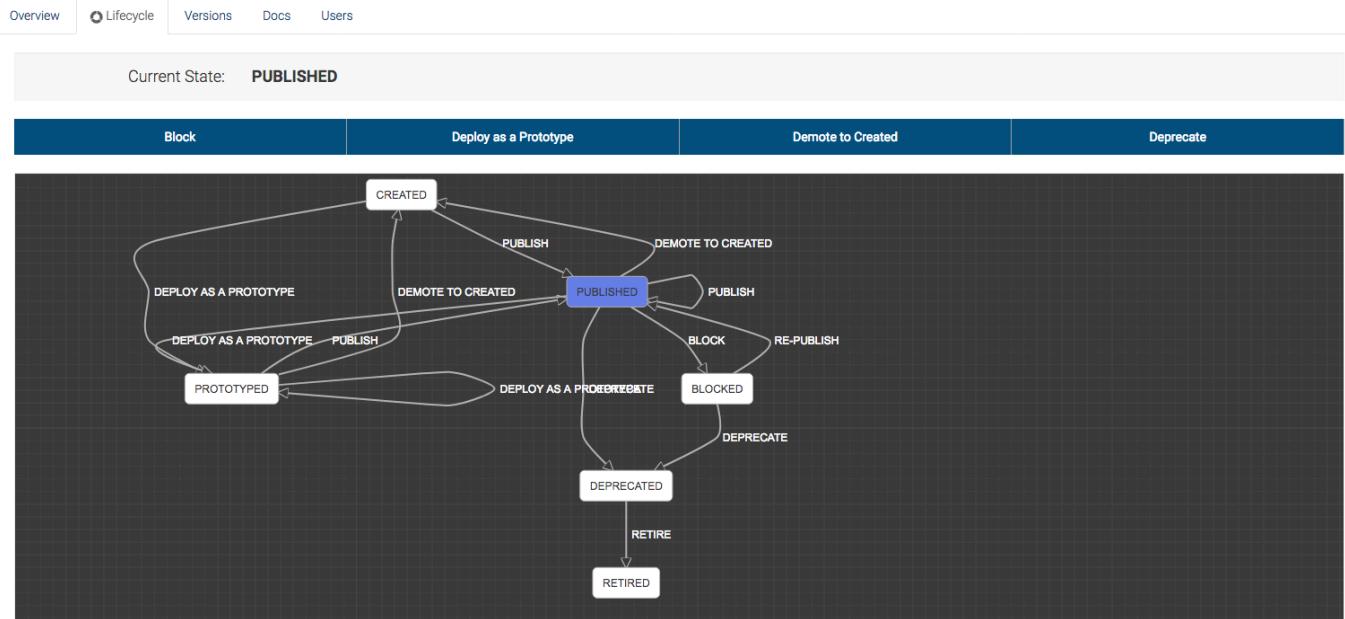
The old API is changed to the DEPRECATED state

Mobile\_stock\_API - 1.0.0



The new API is published

## Mobile\_stock\_API - 2.0.0



Users can configure to get notifications for the newly created API. For more details, see [Enabling Notifications](#).

Enabling grace period to upgrade to a new version can be achieved by using the notifications when a new version of an API is created. At that time, when a new version of an API is created, a notification should be sent to existing subscribers to notify that the API will be deprecated in X months and retired in Y months. Then API Publisher have to perform those actions (deprecation and retiring) in X and Y months time period respectively.

### Running the sample to populate the sample data

- Start wso2am-2.2.0-updateX.
- Go to <API-M\_HOME>/sample-scenarios. Execute the run.sh file:

```
bash run.sh
```

Enter the scenario number, when prompted.