Workshop – SALA A

Construindo plataformas de Negócios usando o Marketplace de APIs.

Updated Nov, 2018

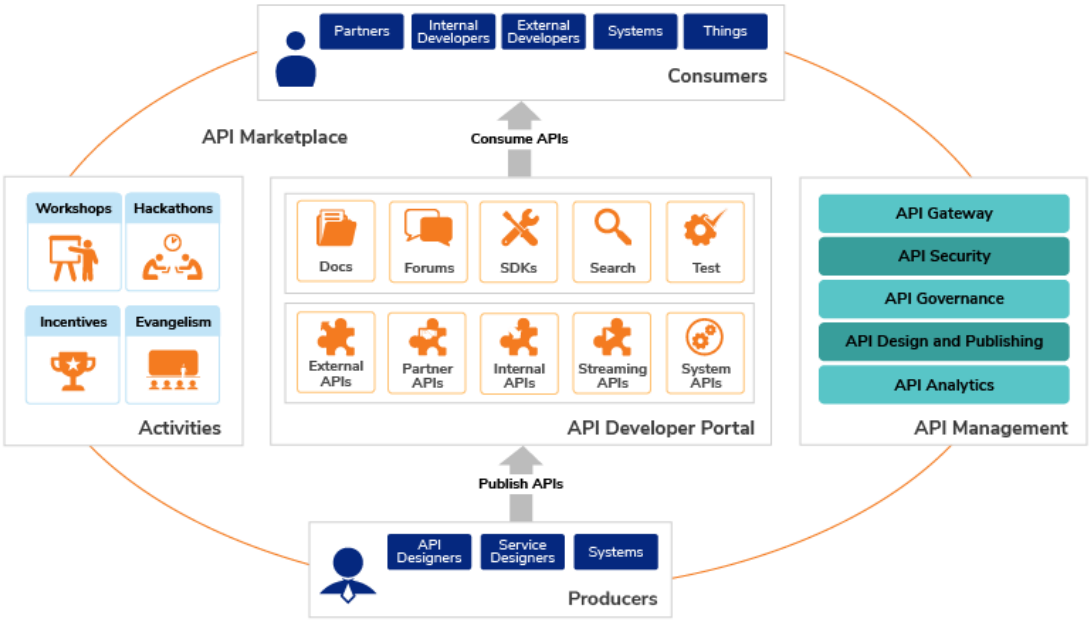
Sumário

APIM Introdução

API Management é a tecnologia predominante para a maioria das organizações que desejam se tornar Digital Driven. Cada vez mais organizações utilizam o gerenciamento de API como um passo importante para se tornar digital.

Um API Marketplace, no entanto, vai além do simples gerenciamento de APIs; um mercado envolve a tecnologia, os negócios e os aspectos humanos do gerenciamento de API, garantindo que as APIs alcancem o objetivo a que se destinam: o consumo e o uso de APIs. Assim, o Marketplace de APIs permite a conexão entre produtores e consumidores de APIs e garante que essa conexão funcione.

Como parte deste workshop, vamos dar uma olhada no que significa API Management e como isso se relaciona com o conceito de um Marketplace corporativo. Analisaremos as vantagens da criação de um Marketplace de API corporativo e as etapas para o lançamento de um Marketplace eficaz de APIs corporativas. Para ilustrar ainda mais esse conceito, abordaremos os aspectos práticos do Marketplace de API da empresa Pizzashark.



Preparação do Ambiente

Instalação do JAVA

Instale Oracle Java SE Development Kit (JDK) 1.8.* e configure a variável de ambiente JAVA_HOME

```
$ /usr/libexec/java_home -V

vi ~/.bash_profile

# Settings for JAVA_HOME

JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home

export JAVA_HOME

$ java -version

java version "1.8.0_191"

Java(TM) SE Runtime Environment (build 1.8.0_191-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Instalação do API-M 2.6.0

Copie o pacote baixado no diretório /opt/wso2/summit2018

```
$ cd /summit2018

$ unzip -x wso2am-2.6.0.zip
```

Instalação do APIM Analytics

Copie o pacote baixado no diretório /opt/wso2/summit2018

$ cd /opt/wso2/summit2018

$ unzip -x wso2am-analytics-2.6.0.zip

Instalação do API-M CLI

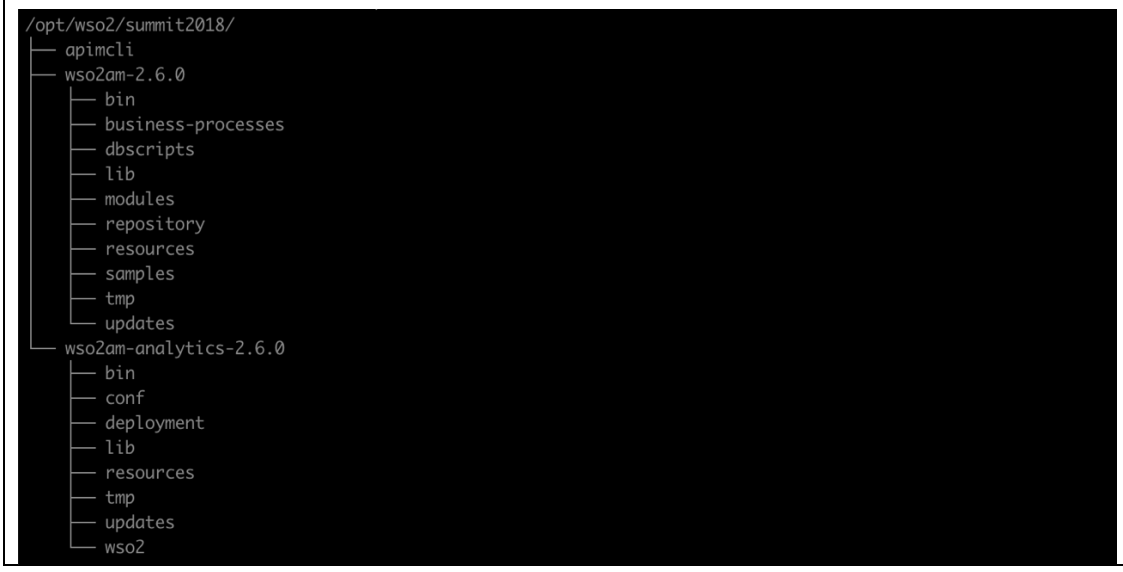
Copie o pacote baixado no diretório /opt/wso2/summit2018

```
$ tar xvf apimcli-1.2.0-macosx-x64.tar
x apimcli/
x apimcli/LICENSE
x apimcli/README.html
x apimcli/apimcli
```

Instalação do API M Developer Studio e extras (Postman, VisualStudio, Curl)

Basta executar o pacote baixado.

Seu diretório deve estar assim.



Configurando o Analytics

```
$ cd /opt/wso2/summit2018/wso2am-2.6.0/repository/conf/datasources

$ vi master-datasources.xml

Procure por

<datasource>

        <name>WSO2AM_STATS_DB</name>

E altere a linha URL.

<configuration>

                <url>jdbc:h2:$/opt/wso2/summit2018/wso2am-analytics-
2.6.0/wso2/worker/database/WSO2AM_STATS_DB;AUTO_SERVER=TRUE</url>

...
```

Vá para o diretório /opt/wso2/summit2018/wso2am-2.6.0/repository/conf e edite o arquito api-manager.xml

```
$ vi api-manager.xml

Procure por Analytics e altere a tag Enabled.

   <Analytics>

      <!-- Enable Analytics for API Manager -->

      <Enabled>true</Enabled>
```

Iniciando o API-M Analytics

```
$ cd /opt/wso2/summit2018/wso2am-analytics-2.6.0/bin

$ worker.sh

Output:
```

```
INFO {org.wso2.carbon.kernel.internal.CarbonStartupHandler} - WSO2 API Manager Analytics Server started in 5,461 sec
```

Iniciando o API-M All-in-one profile

$ cd /opt/wso2/summit2018/wso2am-2.6.0/bin

$ ./wso2server.sh

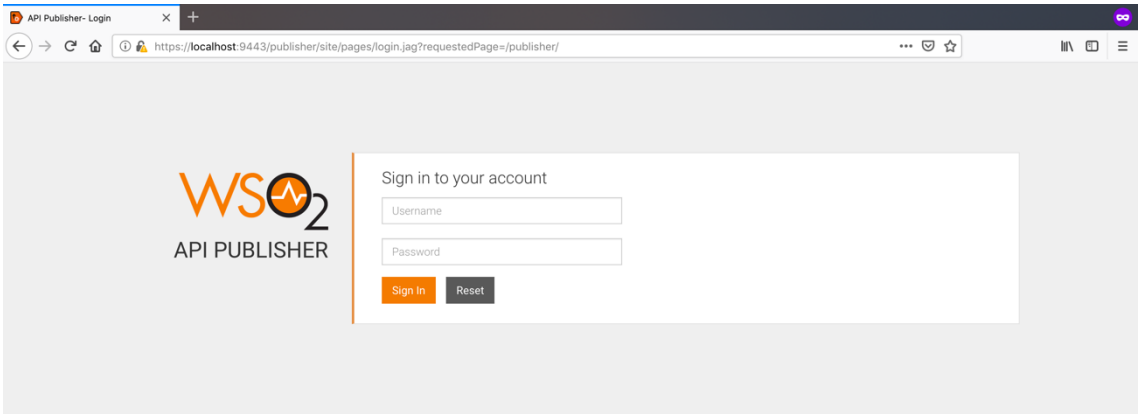
Output:

```
...
INFO - StartupFinalizerServiceComponent Server :  WSO2 API Manager-2.6.0
INFO - StartupFinalizerServiceComponent WSO2 Carbon started in 45 sec
INFO - CarbonUIServiceComponent Mgt Console URL  : https://your.ip:9443/carbon/
INFO - CarbonUIServiceComponent API Store Default Context : https://your.ip:9443/store
INFO - CarbonUIServiceComponent API Publisher Default Context : https://your.ip:9443/publisher
...
```

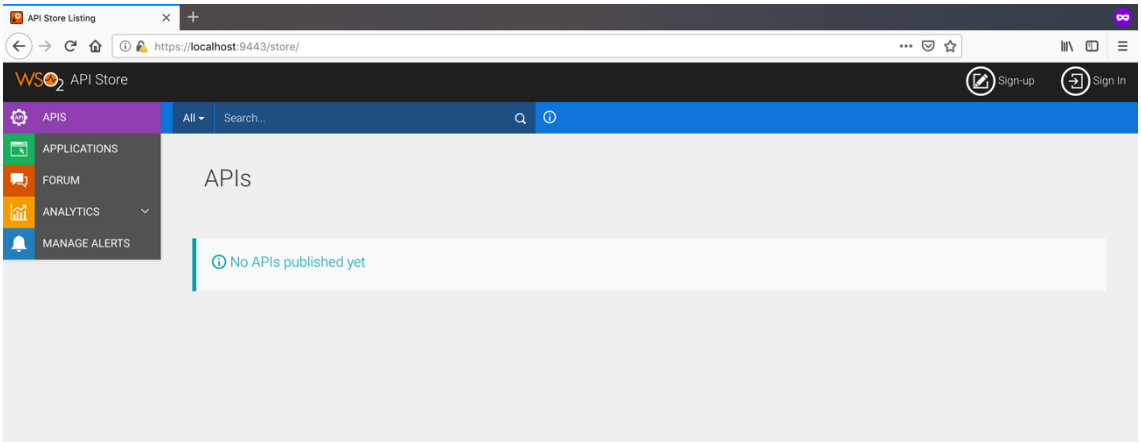
Neste momento você será capaz de acessar as consoles abaixo em seu browser:

Usuário e senha padrão: admin/admin

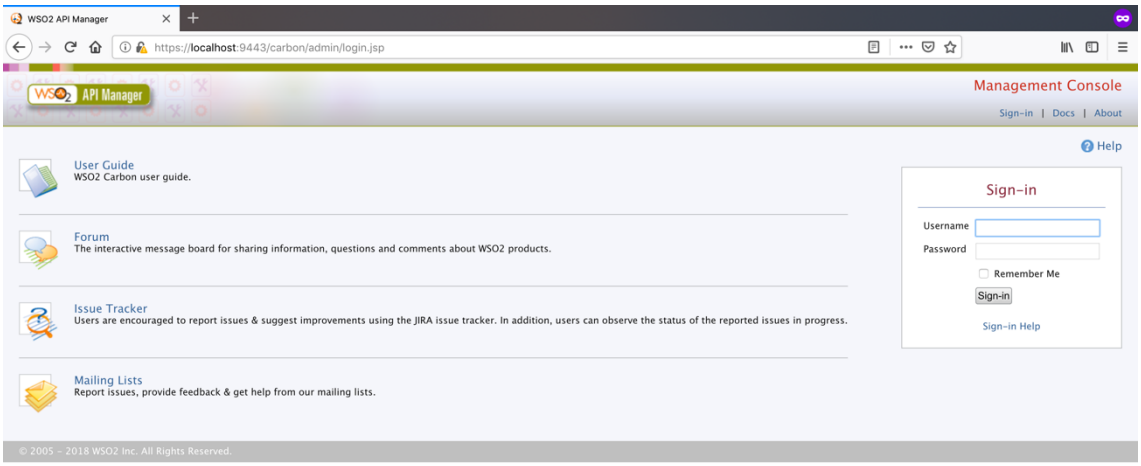
URL: https://localhost:9443/publisher



URL: https://localhost:9443/store



URL: https://localhost:9443/carbon



Configurando o APIM Tooling:

Edite o arquivo /Applications/Eclipse.app/Contents/Info.plist Procure pela chave <key>Eclipse</key> Adicione a linha: <string>-vm</string><string>/Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/bin/java</string>

Demo 1 – API Quick-Start

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.

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

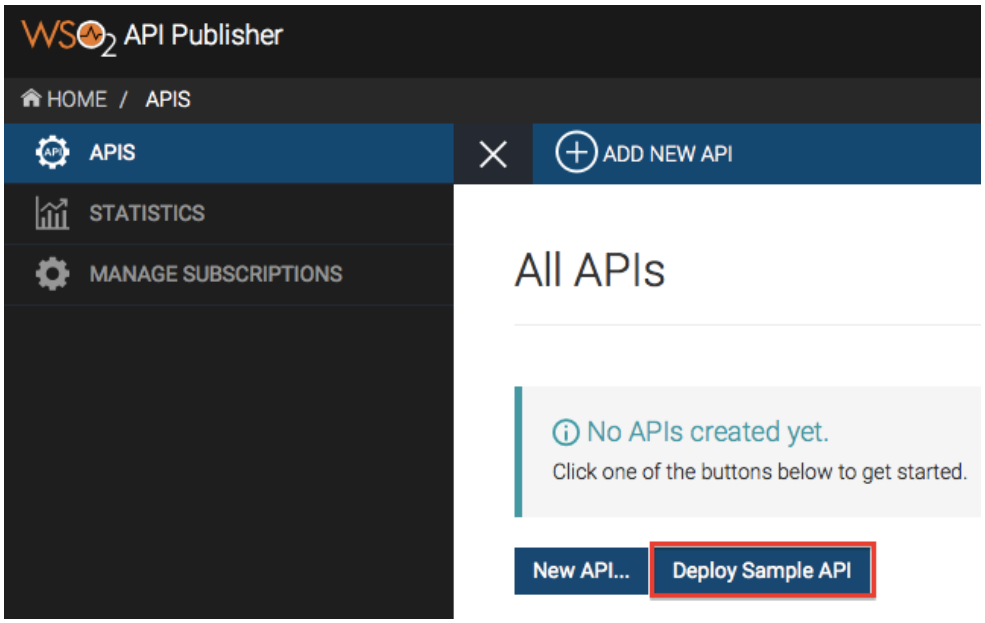
Invoking your first API

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

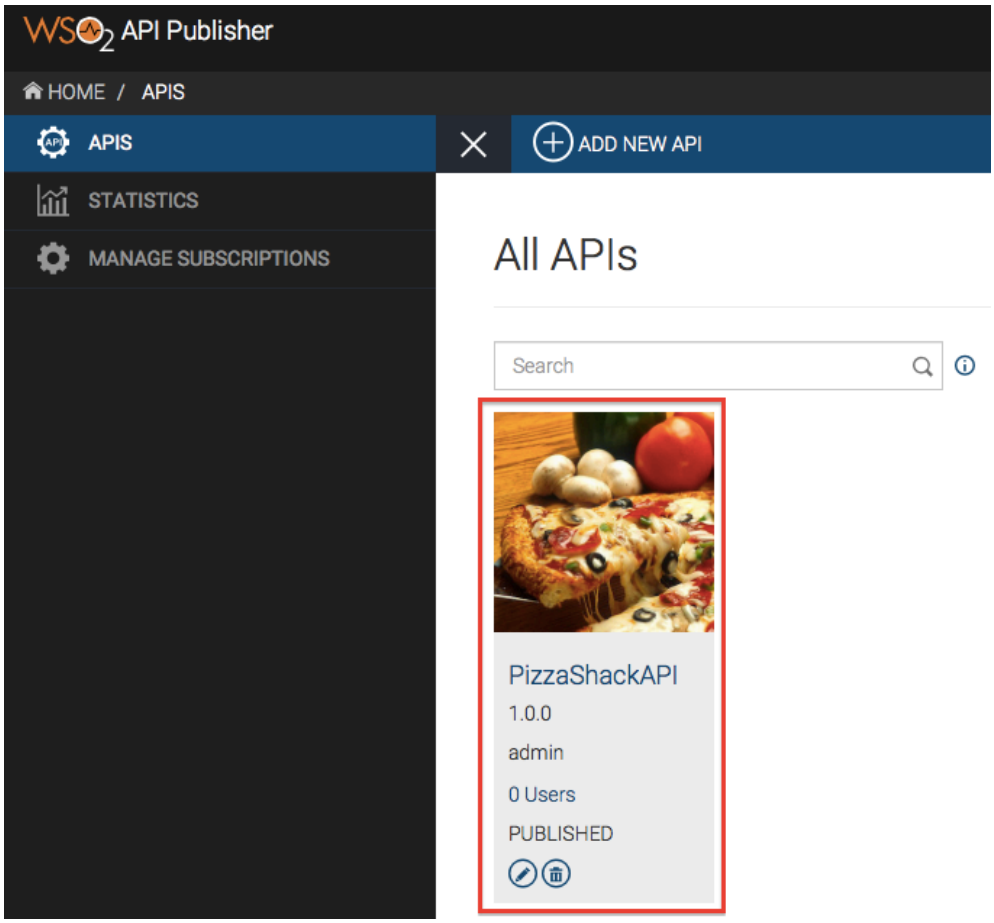
1. Open the API Publisher (https://<hostname>:9443/publisher) and sign in with **admin/admin** credentials.
2. Exit from API creation tutorial by clicking the close icon(X) on top right corner.

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

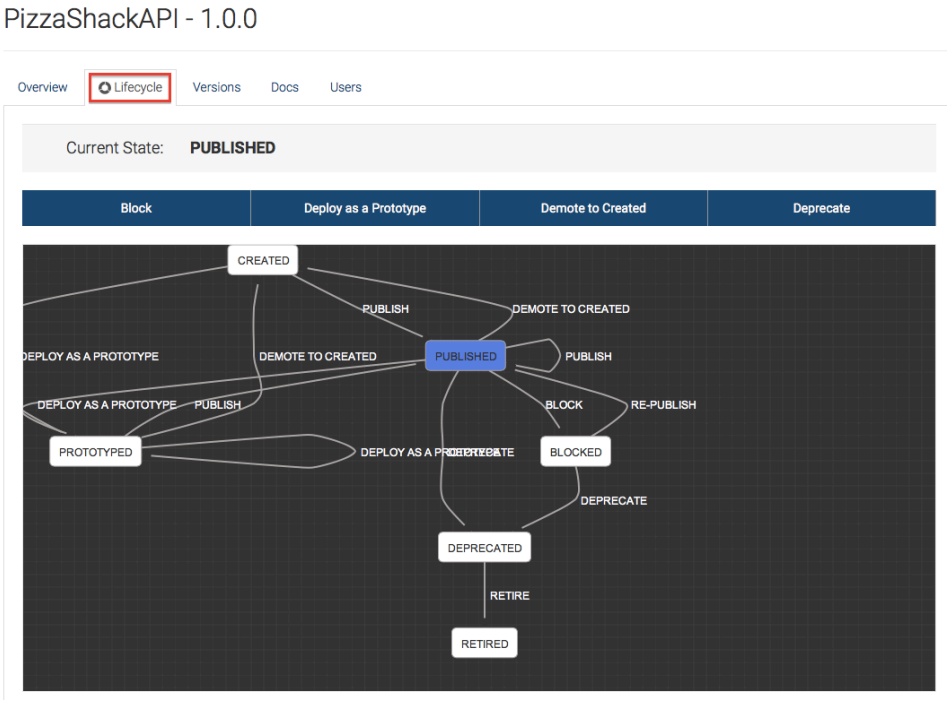
   This **Deploy Sample API** option is available only when there are no APIs in API Publisher. If you have already created aAPI, this option will not be available.



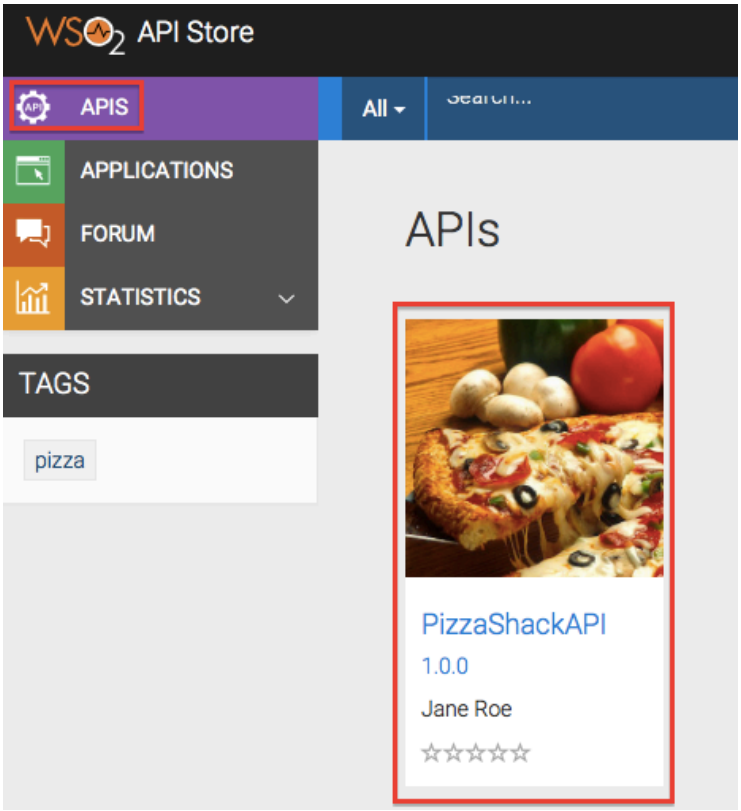
4. Click `PizzaShackAPI` to open it.



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

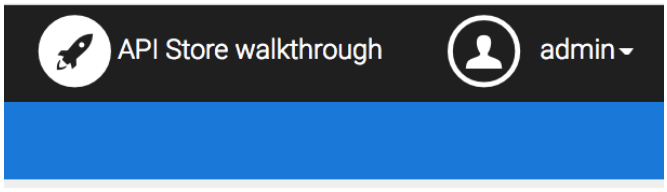


6. Sign in to the API Store (`https://<hostname>:9443/store`) with the **`admin/admin`** credentials and click on the `PizzaShackAPI` API.

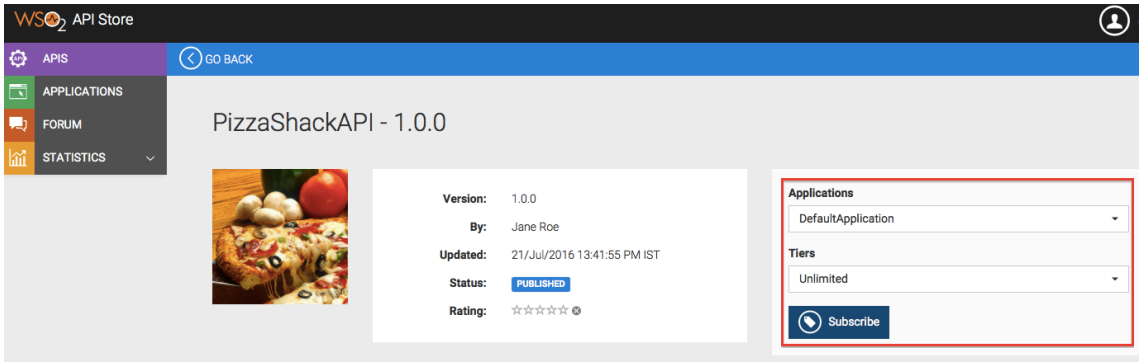


API Store Walkthrough

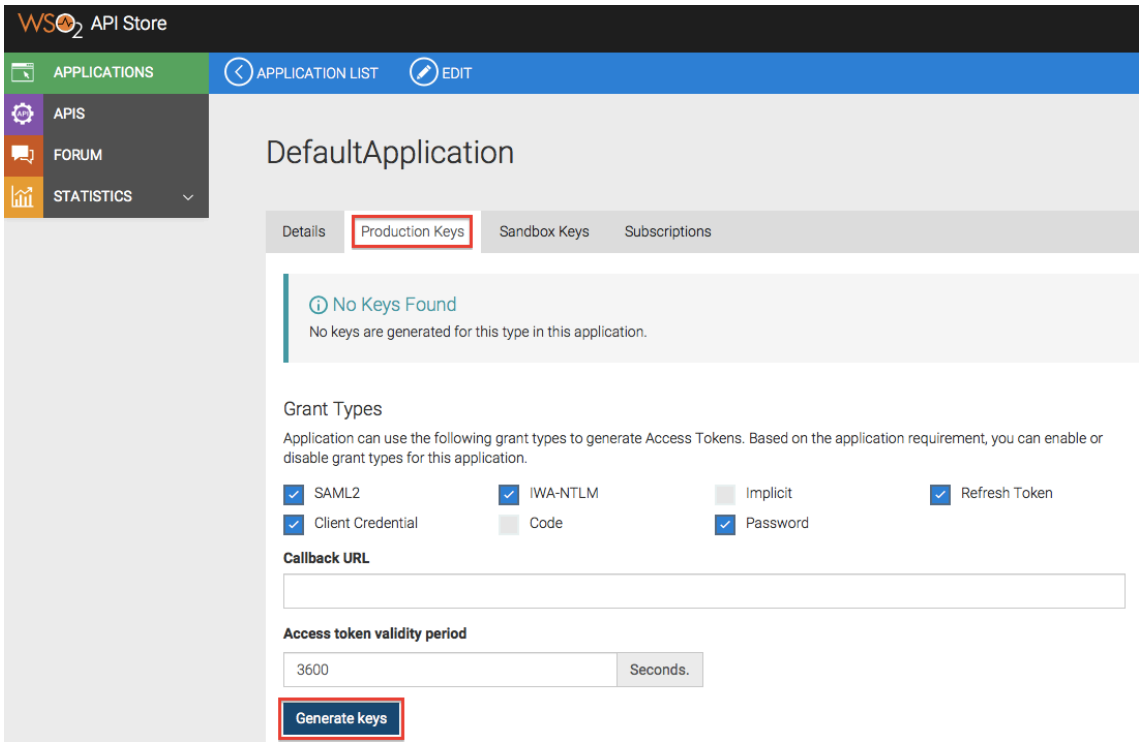
You can click "API Store walkthrough" to view the interactive tutorial to invoke the API.



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

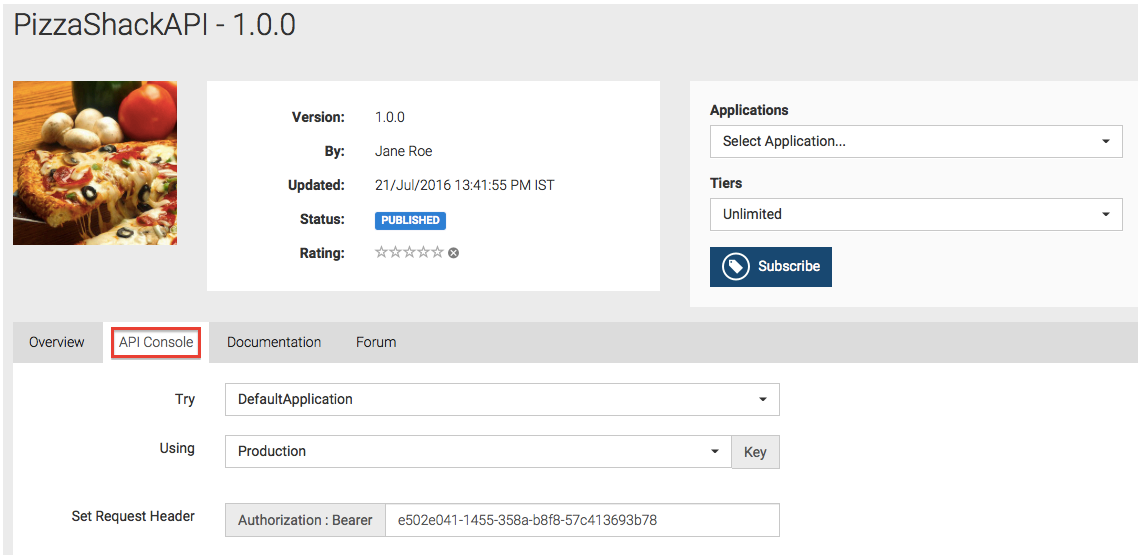


8. When the subscription is successful, click **View Subscriptions** on the information message that appears. Click the **Production Keys** tab and click **Generate Keys** to generate an access token to invoke the API.

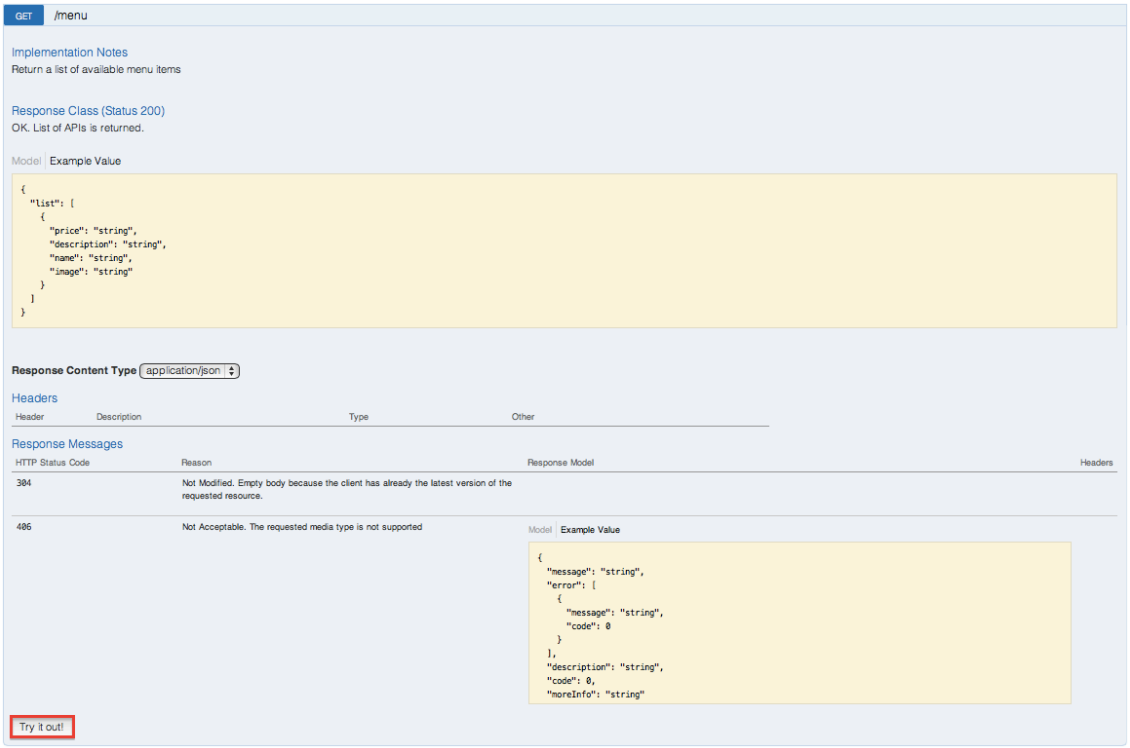


You have now successfully subscribed to an API. Let's invoke the API using the integrated Swagger-based API Console.

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



Expand the GET method (which retrieves the menu) and click **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 barbeque 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",
```


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

Demo 2 – 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:

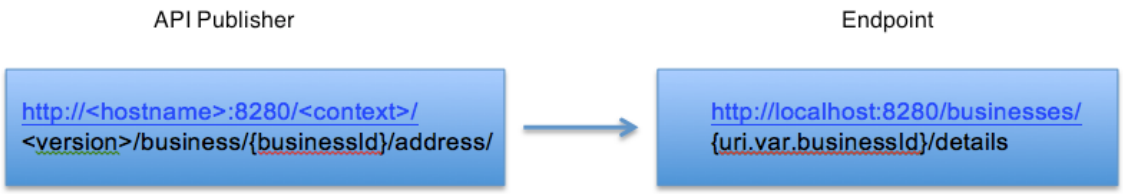
1. The URL pattern of the APIs in the Publisher is `http://<hostname>:8280/<context>/<version>/<API resource>`.

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.

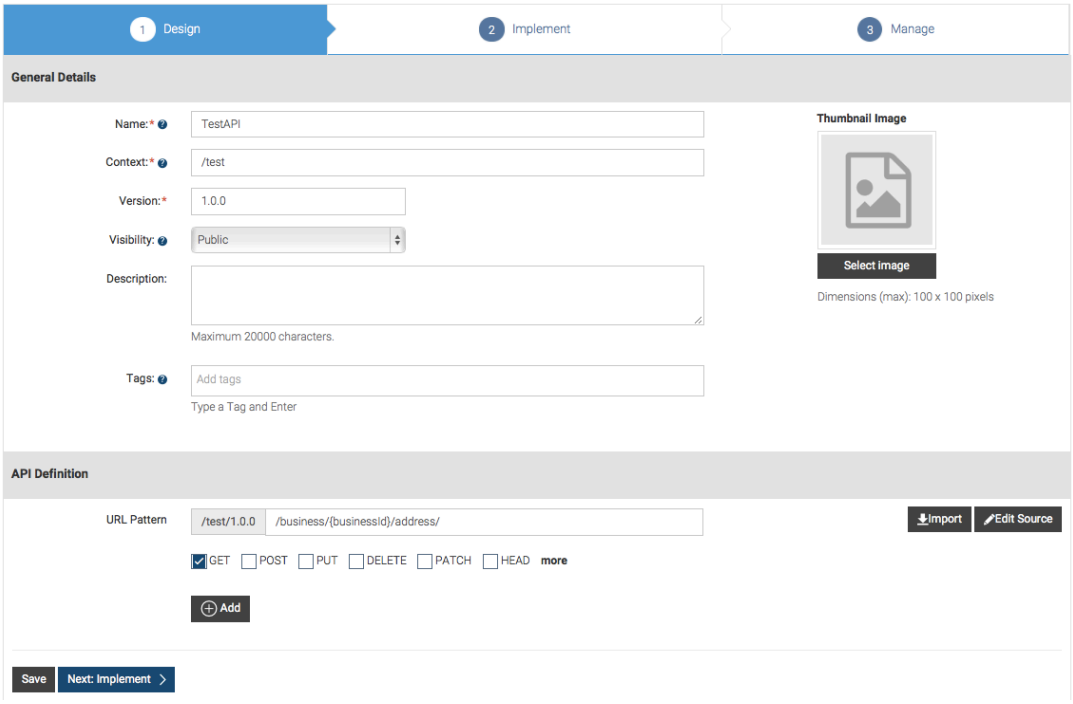
$ cp Response_API.xml /opt/wso2/summit2018/wso2am-2.6.0/repository/deployment/server/synapse-configs/default/api

Output from APIM Server:

INFO - API Initializing API: Response_API
INFO - DependencyTracker API : Response_API was added to the Synapse configuration successfully
INFO - APIDeployer API named 'Response_API' has been deployed from file : /opt/wso2/summit2018/wso2am-2.6.0/repository/deployment/server/synapse-configs/default/api/Response_API.xml

1. Log in to the API Publisher, design a new API with the following information, click **Add** and then click **Next: Implement >**.

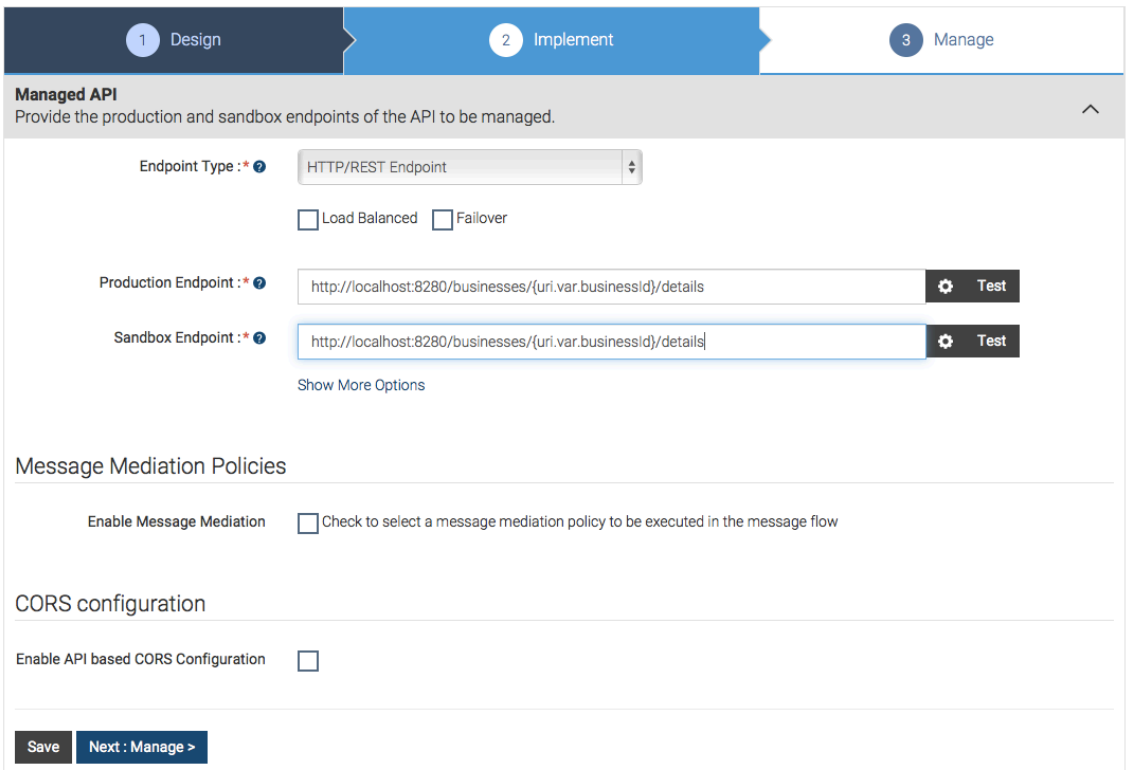
Field		Sample value
Name		TestAPI
Context		/test
Version		1.0.0
Visibility		Public
Resources	URL pattern	/business/{businessId}/address/
	Request types	GET

2. 

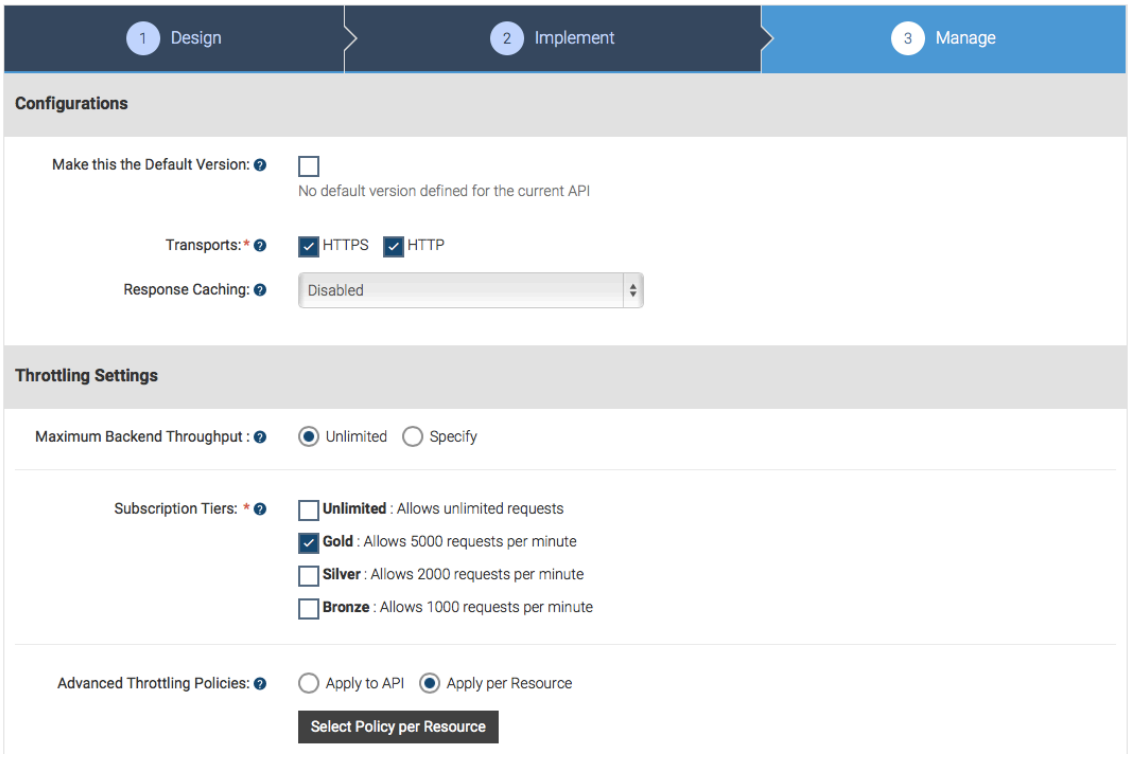
3. The **Implement** tab opens. Give the information in the table below.

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`

4.



5. Click **Next: Manage >** to go to the `Manage` tab, select the `Gold` tier and publish the API.



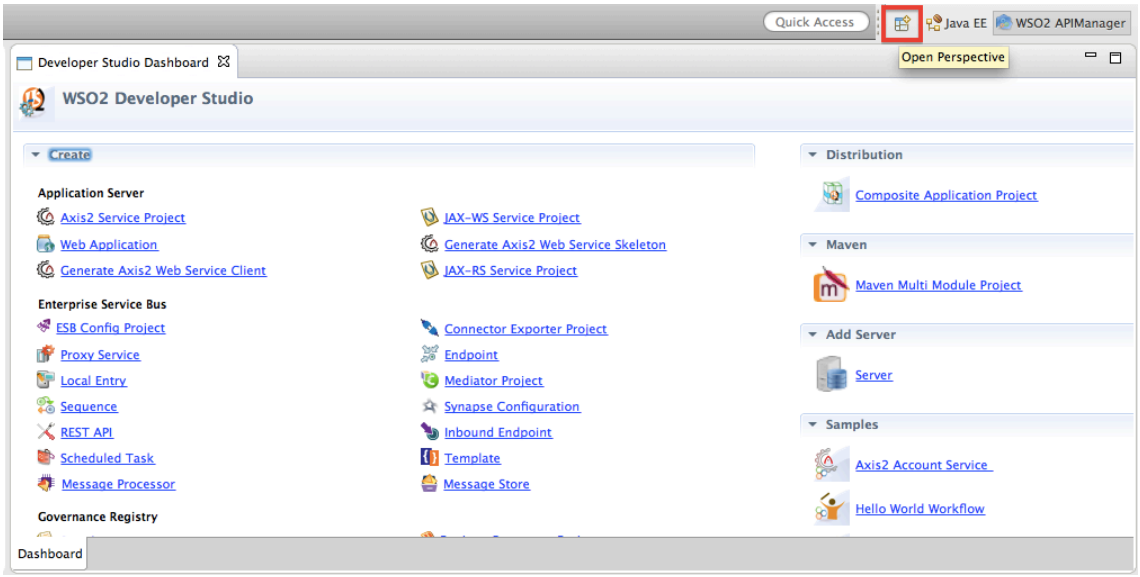
As the API's resource is appended to its endpoint by Synapse at runtime, let's write a custom sequence to remove this appended resource.

6. Copy the the following to a text editor and save the file in XML format (e.g., `TestSequence.xml`).

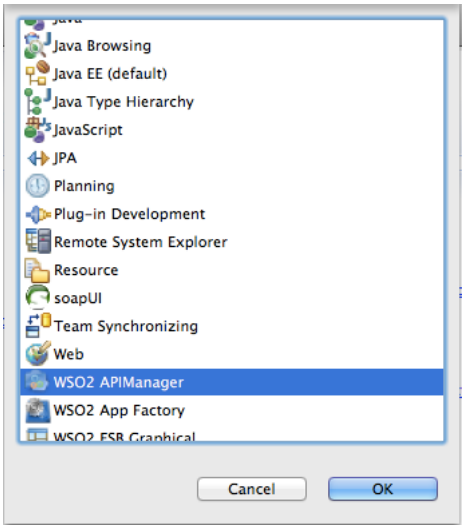
```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="TestSequence">
    <property name="REST_URL_POSTFIX" scope="axis2" action="remove"/>
</sequence>
```

7. 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.

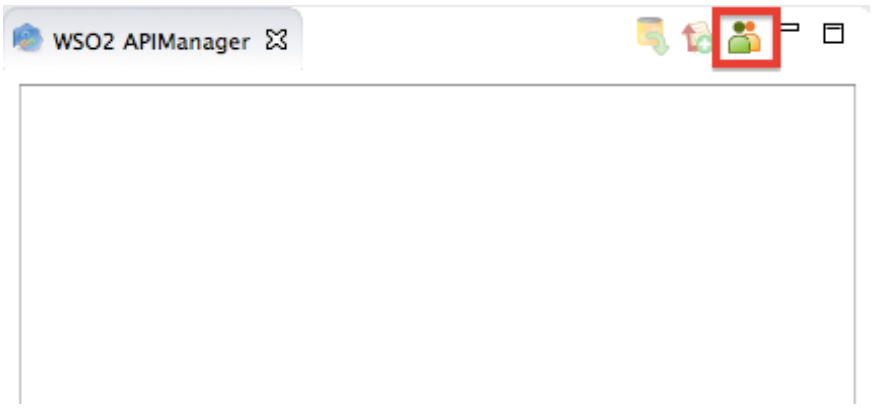
8. Click **Window > Open Perspective > Other** to open the Eclipse perspective selection window. Alternatively, click the **Open Perspective** icon shown below at the top right corner.



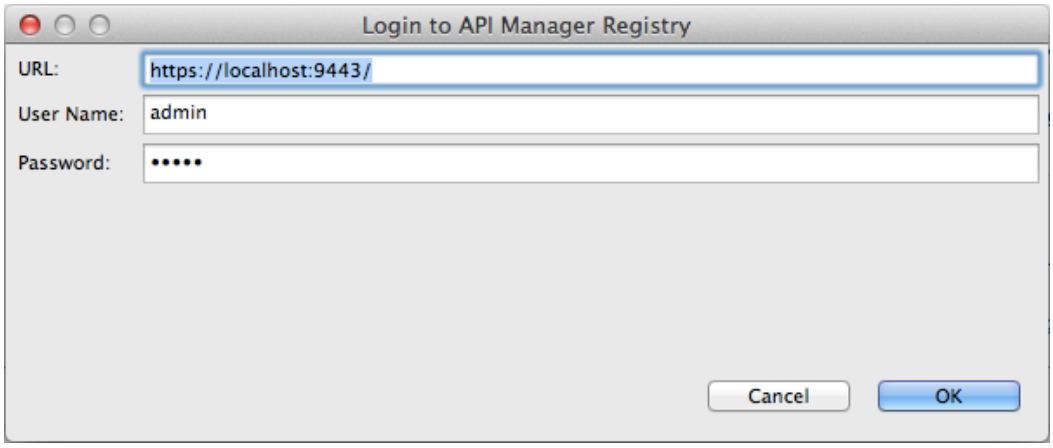
9. On the dialog box that appears, click **WSO2 APIManager** and click **OK**.



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

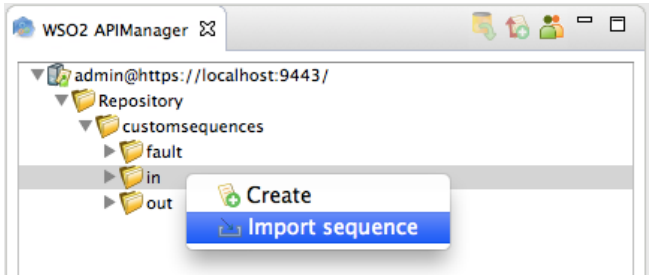


11. On the dialog box that appears, enter the URL, username and password of the Publisher server.



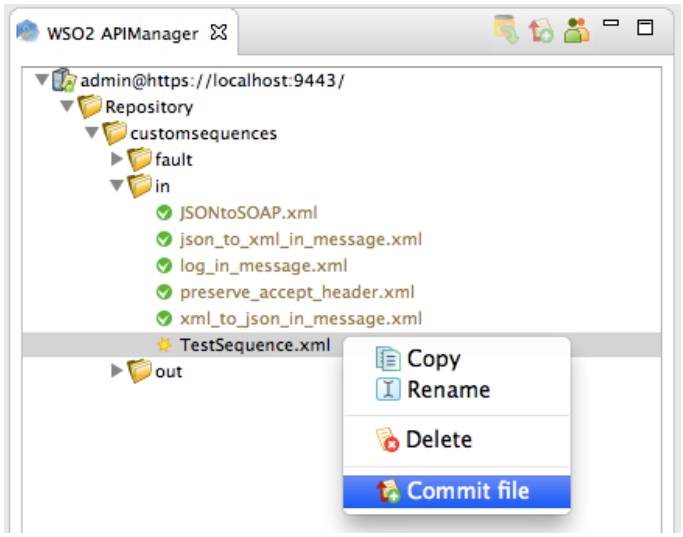
12. On the tree view that appears, expand the folder structure of the existing API.

13. Right-click on the `in` sequence folder and click **Import Sequence** to import the sequence you create above.

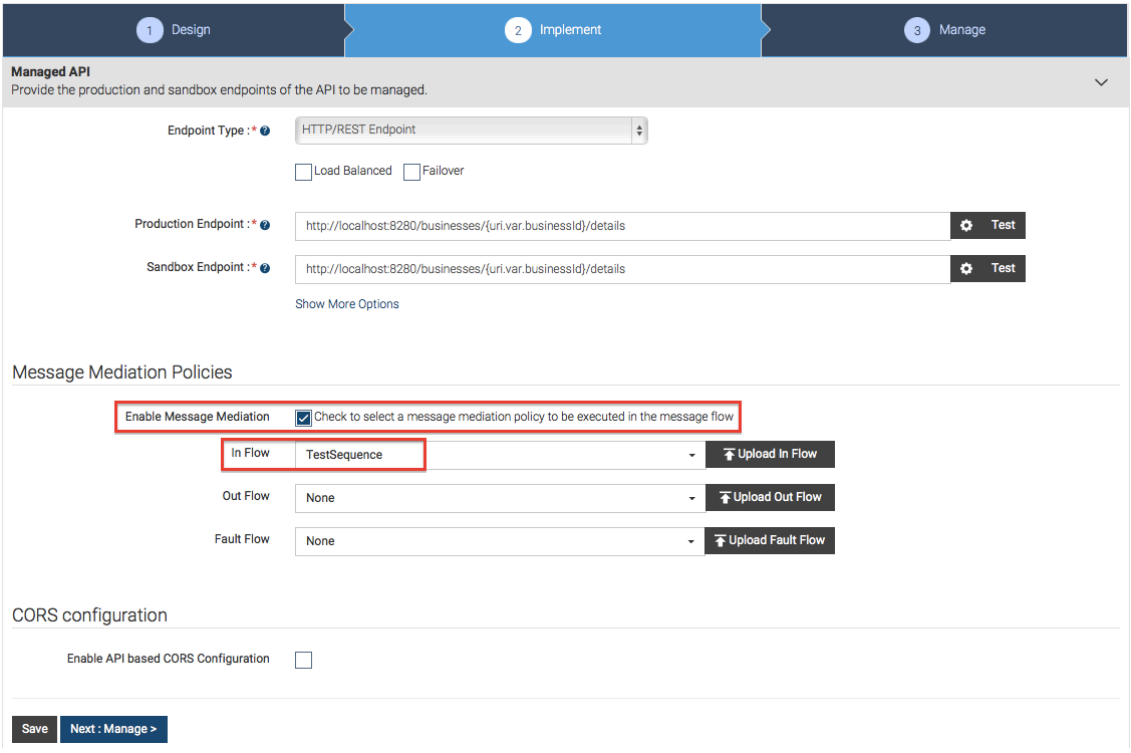


14. Browse to the `TestSequence.xml` file you created in step 4.

15. 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.



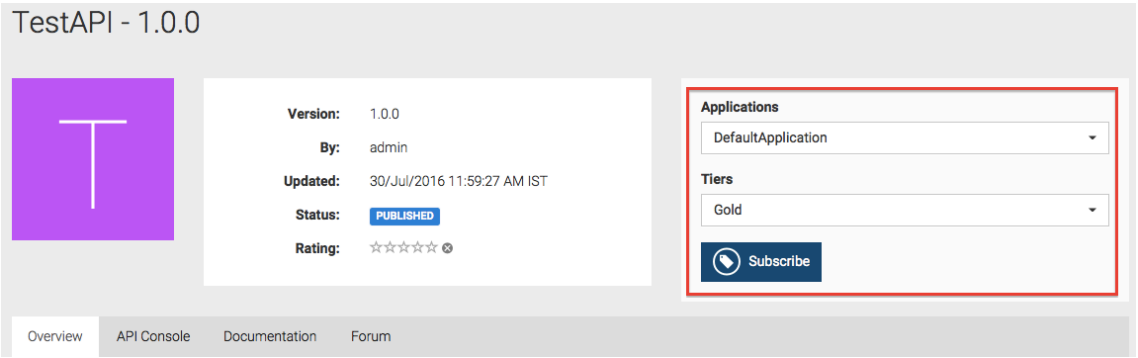
16. Log back into the API Publisher, click **Edit** and go to the **Implement** tab. Select the **Enable Message Mediation** check box and engage the `In` sequence that you created earlier.



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**.

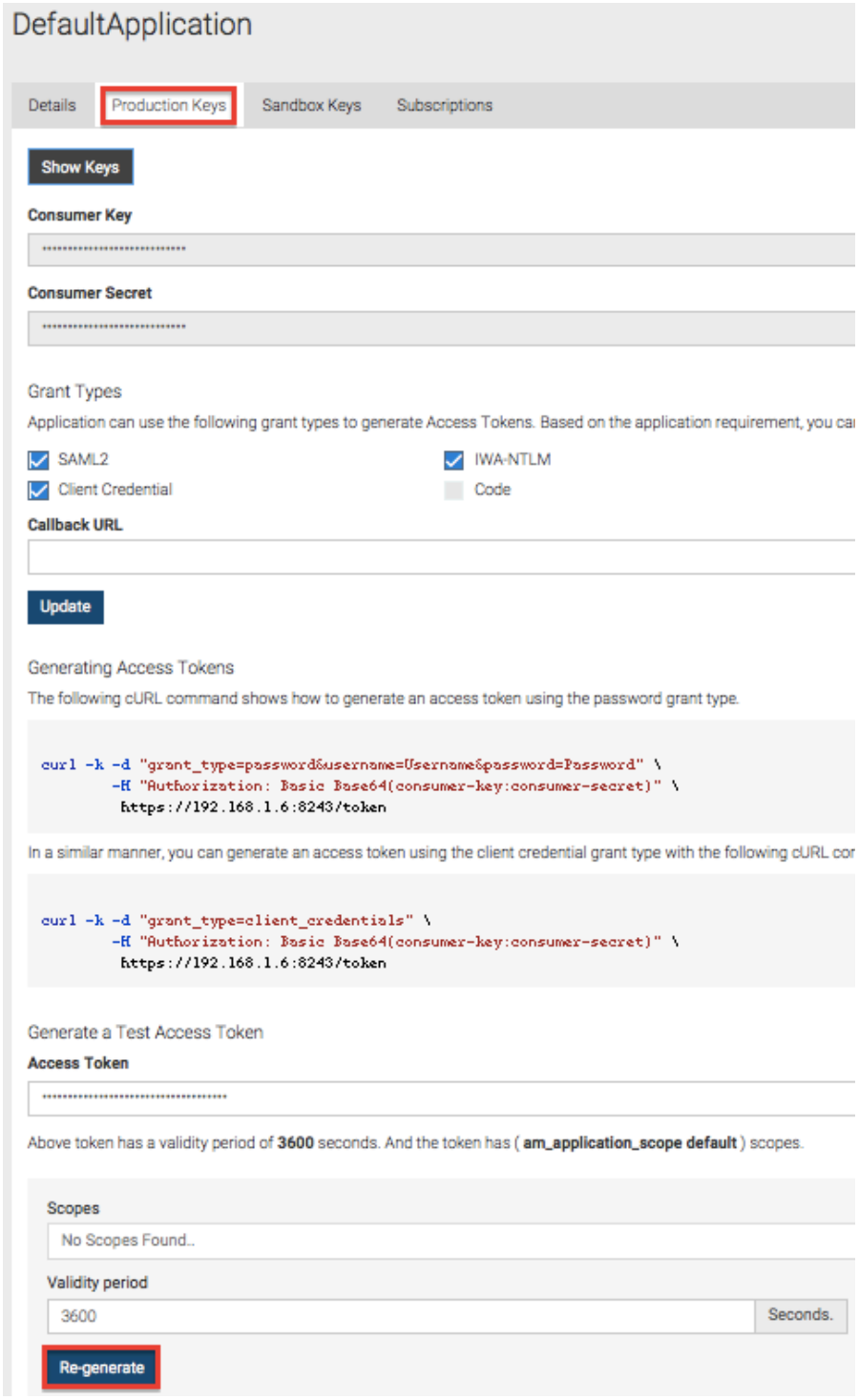
17. Save and Publish the API.
    You have created an API. Let's subscribe to the API and invoke it.

18. Log in to the API Store and subscribe to the API.

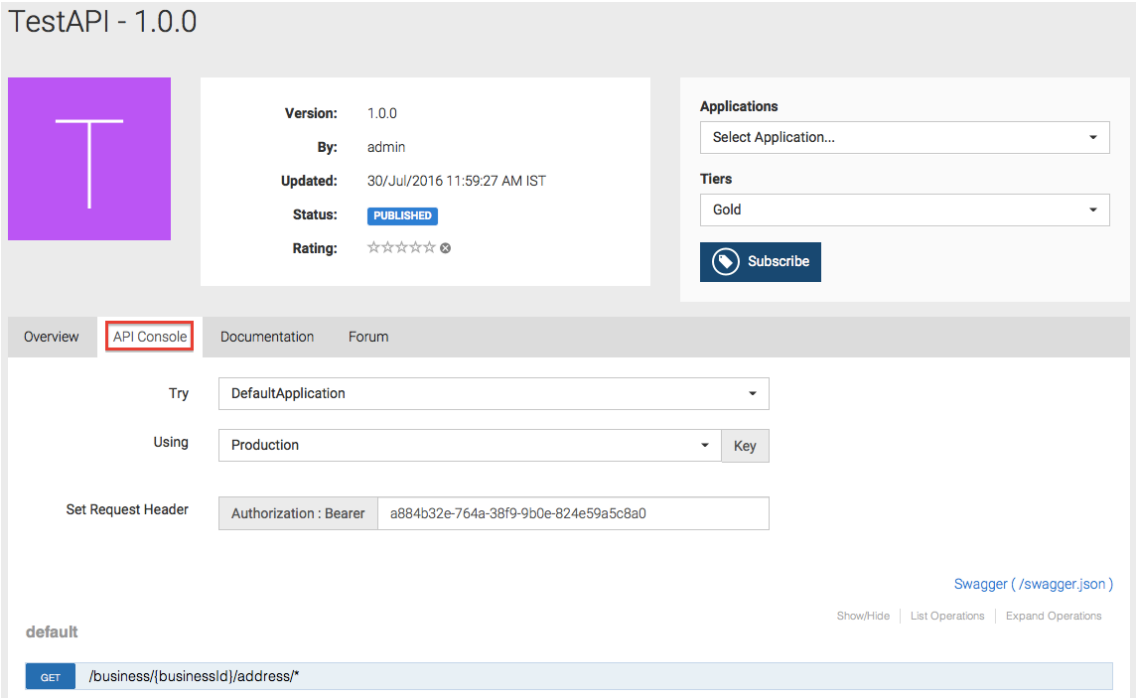


19. Click the **View Subscriptions** button when prompted. The **Subscriptions** tab opens.

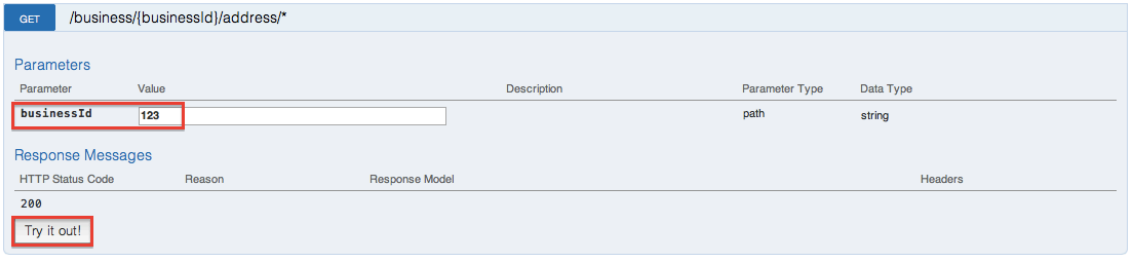
20. Click the **Production Keys** tab and click **Generate Keys** to create an application access token. If you have already generated a token before, click **Regenerate** to renew the access token.



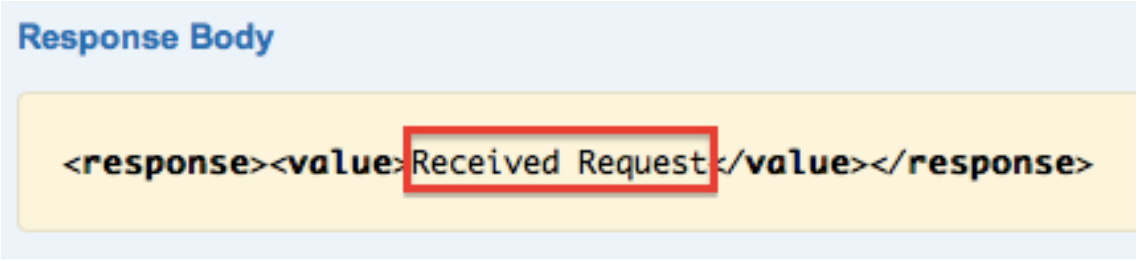
21. Click the **API Console** tab of your API.



22. Note that the `businessId` is added in the UI as a parameter. Give a `businessId` and click **Try it out** to invoke the API.



23. Note the response that you get. According to the mock backend used in this tutorial, you get the response `Received Request`.

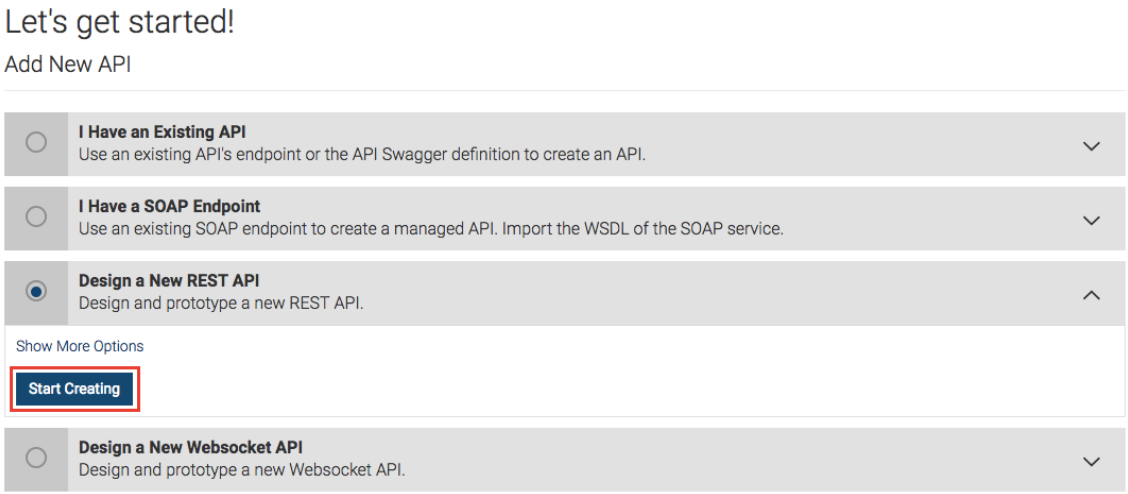


In this tutorial, you mapped the URL pattern of the APIs in the Publisher with the endpoint URL pattern of a sample backend.

Demo 3 – 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. Typically, you create APIs with inline scripts 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.

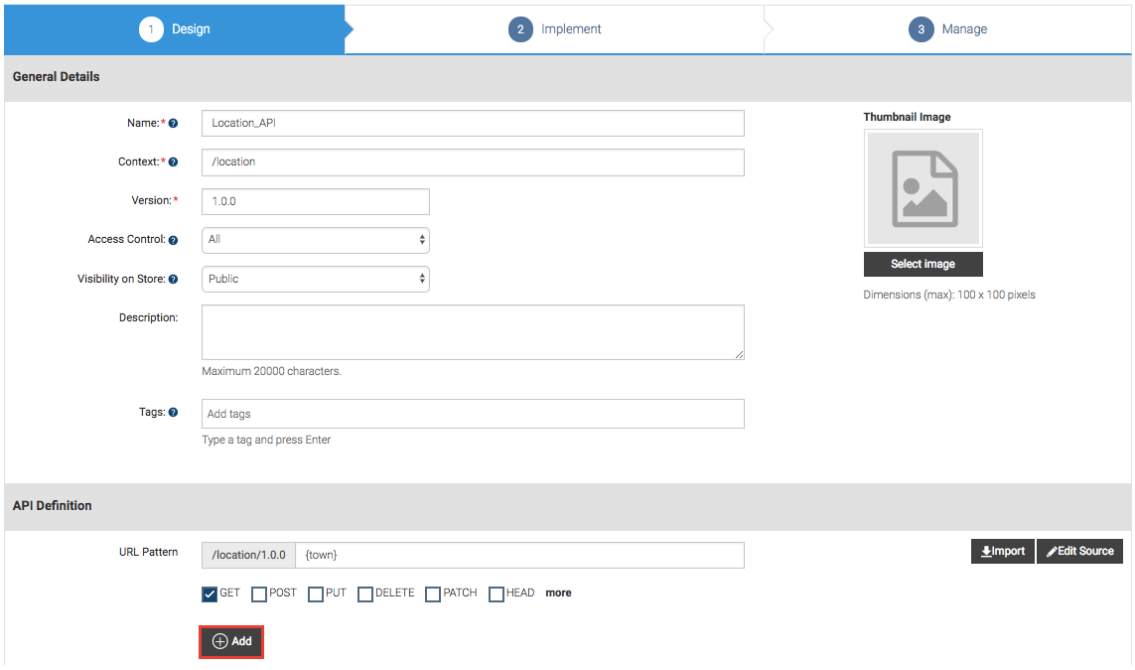
1. Sign in to the API Publisher using `admin` as the username and password. `https://<hostname>:9443/publisher` (e.g., `https://localhost:9443/publisher`).
2. Select the option to design a new REST API and click **Start Creating**.



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

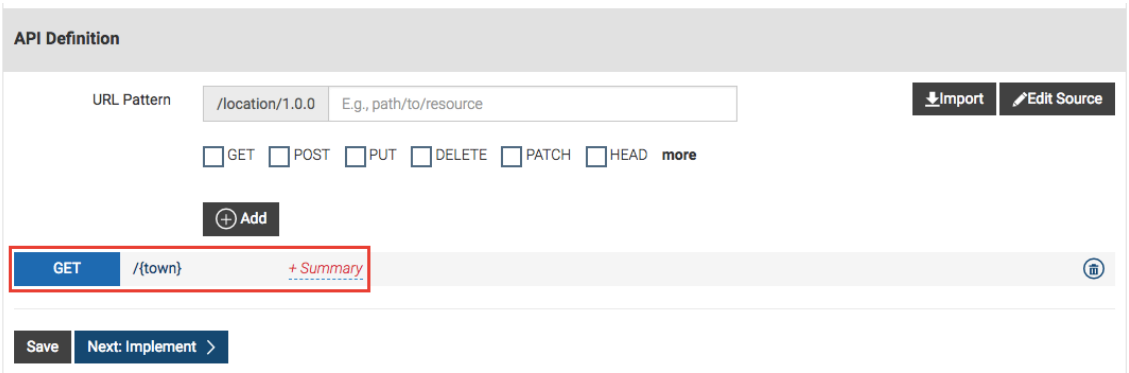
4. Design API



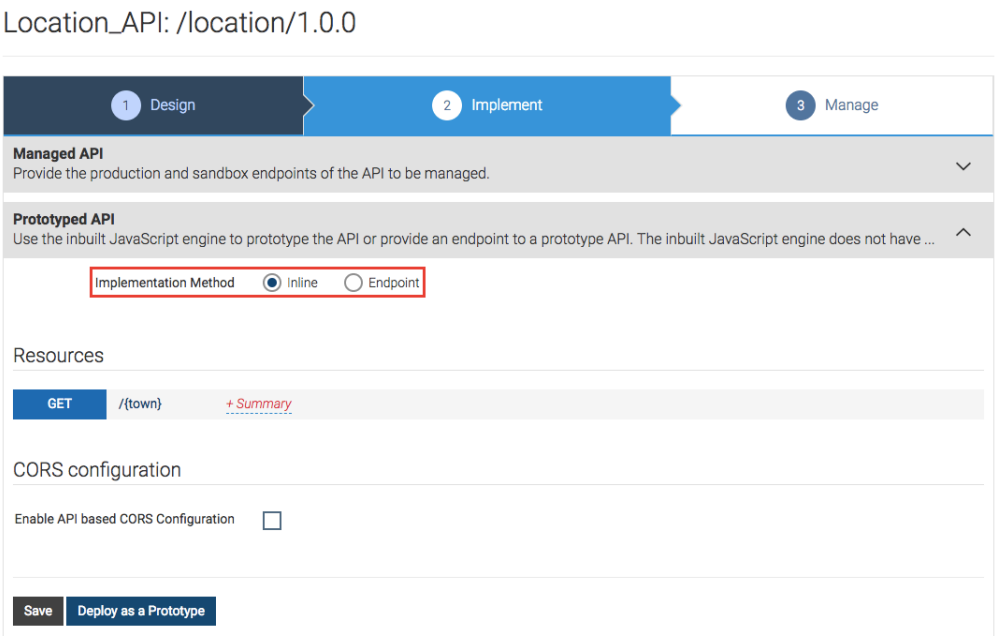
5. After the resource is added, expand its `GET` method and note that a parameter by the name `town` is added under the resource. You use it to pass the payload to the backend. Once done, click **Next: Implement >**.

   To specify multiple parameters in the API resource, separate the parameters with a forward slash.

```
{param1}/{param2}
```



6. In the **Prototyped API** section under the **Implement** tab, select the implementation method as **Inline**.



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

7. 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+'"}');
```

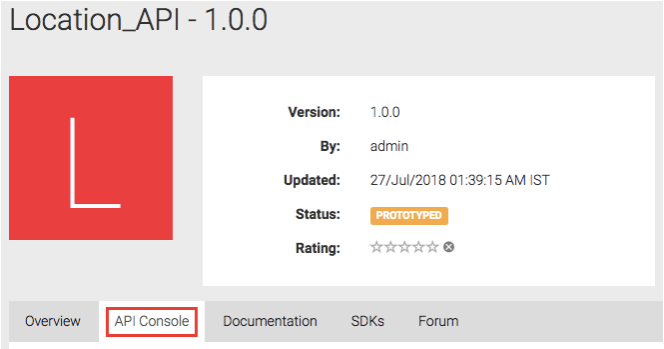
8. 

9. Click **Deploy as a Prototype**.

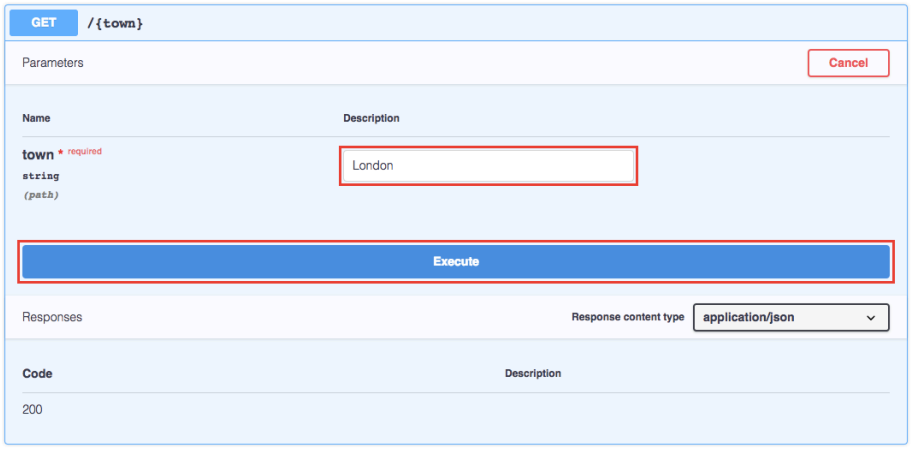
10. When prompted, choose to open the newly published API in the API Store.

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.

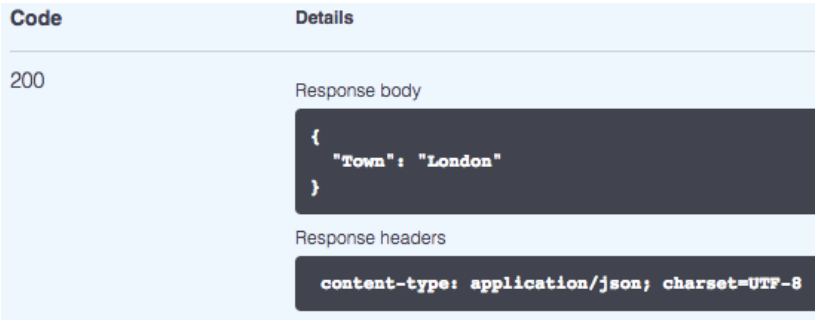
11. The Location API opens. Click the **API Console** tab.



12. Expand the `GET` method, click **Try it out**. Give any value for the town (e.g. London) and click **Execute** to invoke the API.



13. Note the payload you gave as a JSON output in the response.



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.

Demo 4 – 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:

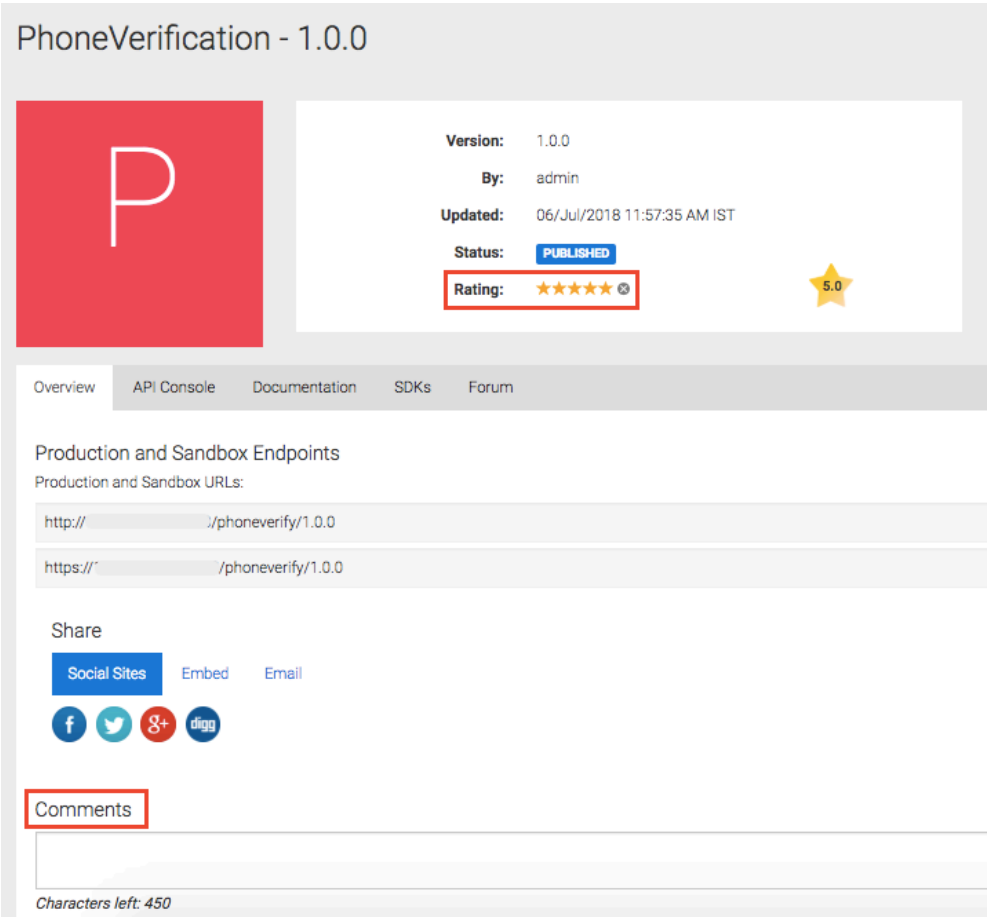
Clause	Syntax
By the API's name	As this is the default option, simply enter the API's name and search.
By the API provider	**provider:xxxx**. For example, provider:admin Provider is the user who created the API.
By the API version	**version:xxxx**. For example, version:1.0.0 A version is given to an 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:/checkphonenumber. 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 `checkphonenumber`,

Clause	Syntax
	then `/checkphonenumber` 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.

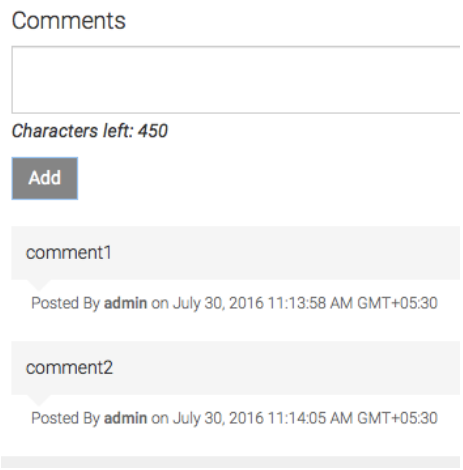
Rate and comment

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:

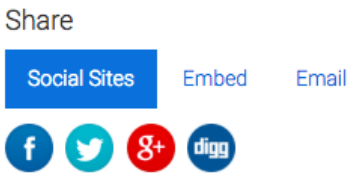


3. Add a rating and a comment. Note that the comments appear sorted by the time they were entered, alongside the author's name.



Share on social media/e-mail

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

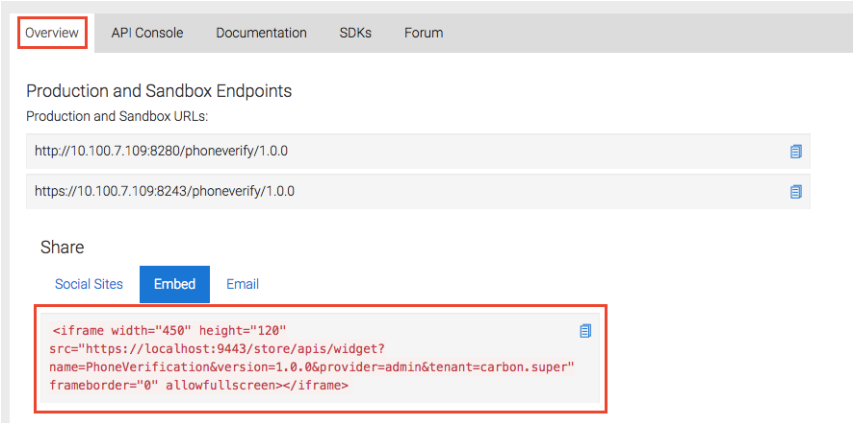


Embed an API widget

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 are embedded in a web page.

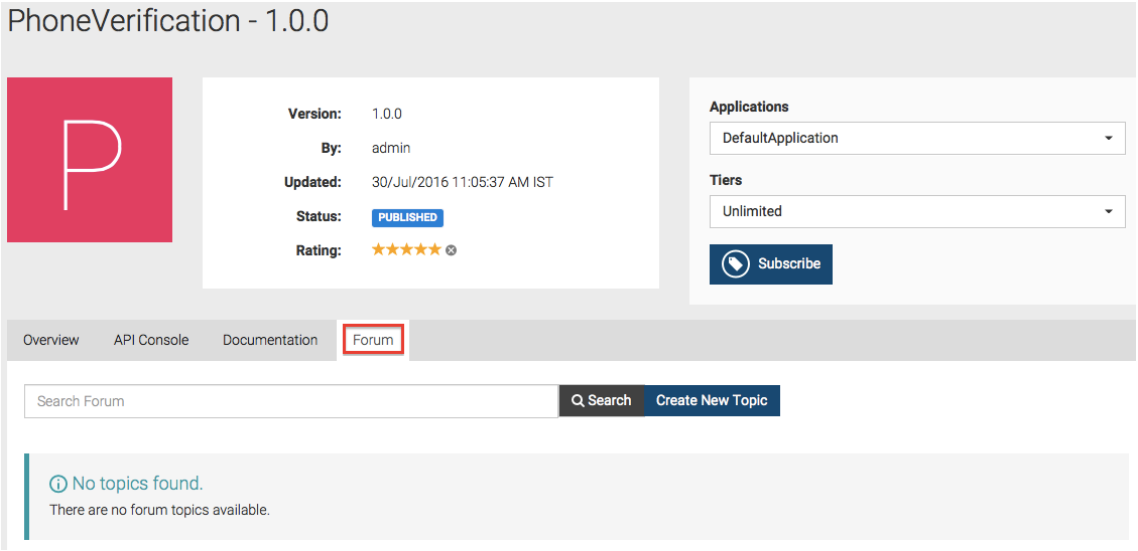
1. Log in to the API Store and click on a published API.

2. Note the **Embed** tab under the API's sharing options.



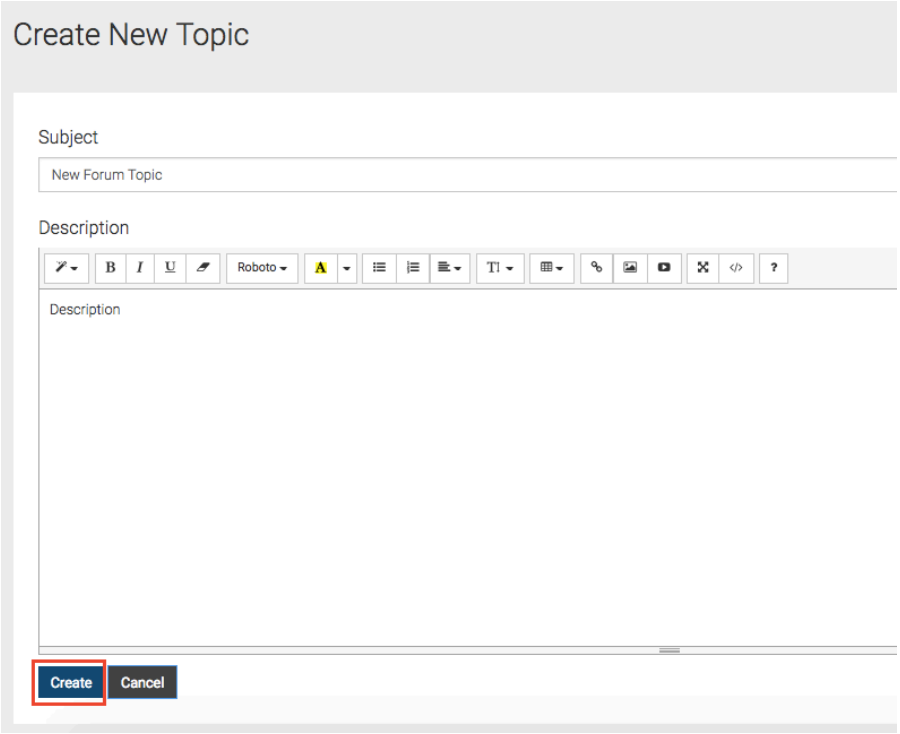
Participate in the fórum

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.



Click **Create New Topic** to start a new topic in the Forum.

3. Enter the Subject and Description of your topic and click Create.



Demo 5 – Adding Internationalization and Localization (para casa)

The API Manager comes with two Web interfaces as API Publisher and API Store. The following steps show an example of how to localize the API Publisher UI. Same instructions apply to localize the API Store.

Changing the browser settings

1. Follow the instructions in your Web browser's user guide and set the browser's language to a preferred one. For example, in Google Chrome, you set the language using the **Settings -> Show advanced settings -> Languages** menu.
2. Set the browser's encoding type to UTF-8.

Introduction to resource files

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.
3. 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.
4. 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

To localize the API publisher to Spanish, first localize the strings defined in jaggery files. Create a new file by the name **locale_{lolcaleCode}.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/publisher`).

3. Note that the UI is now changed to Spanish.

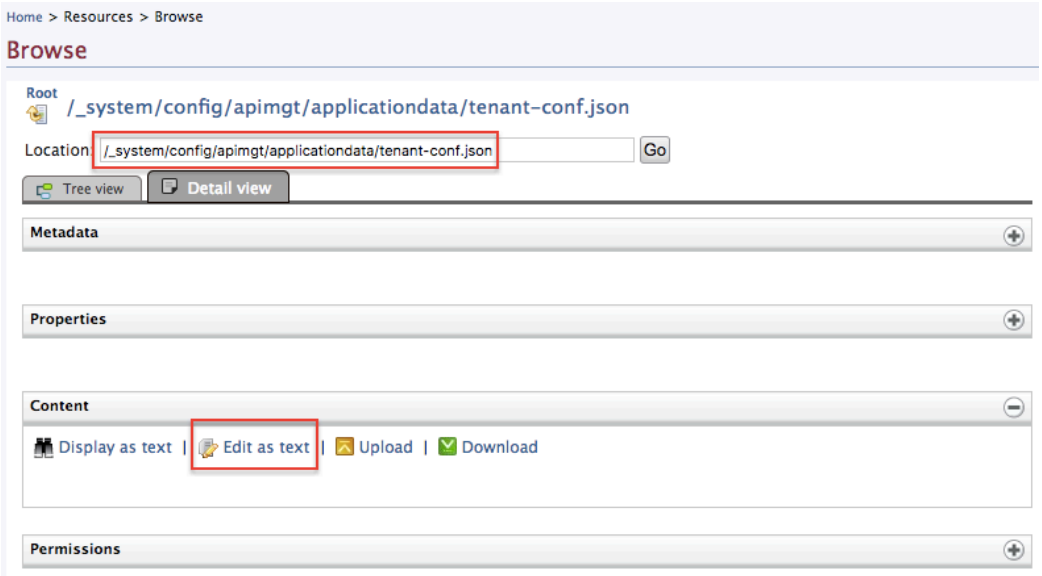
Demo 6 – Configuring API Monetization Category Labels

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,

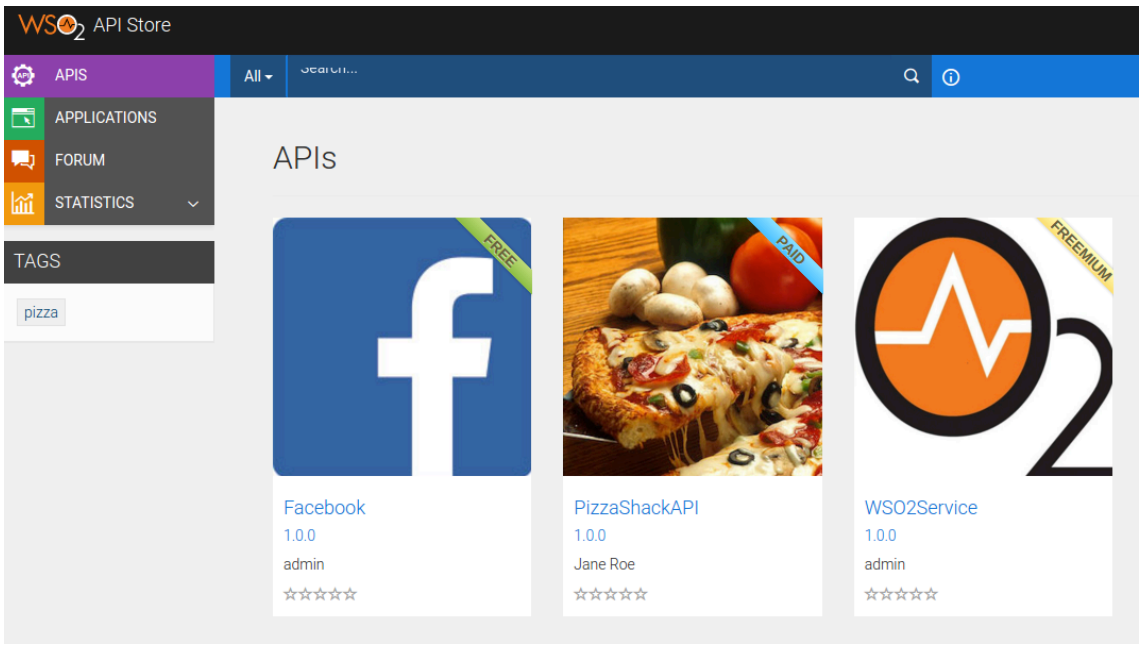
   - To define the unlimited tier as **paid**, set the `IsUnlimitedTierPaid` property to true.
   - To define the unlimited tier as **free**, set the `IsUnlimitedTierPaid` property to false.

   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.



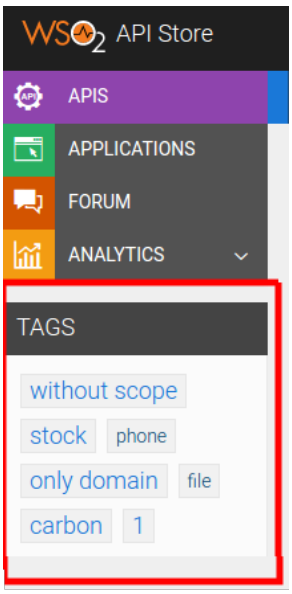
Related links
For more information on Monetization of APIs, see Enabling Monetization of APIs-.

~~Demo 7 – Configure Multiple Tenants~~

Demo 7 – Customizing the API Store

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).

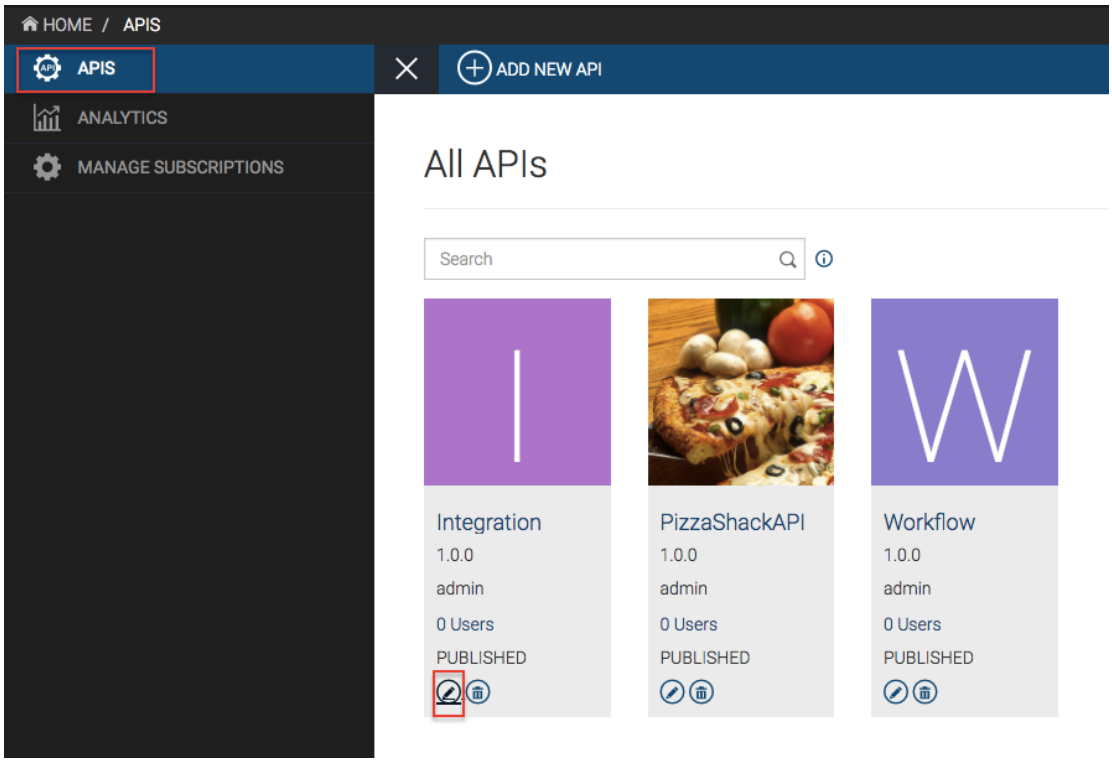
- The **tag name** should **not have a suffix or prefix**, but it **can have spaces**.

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.

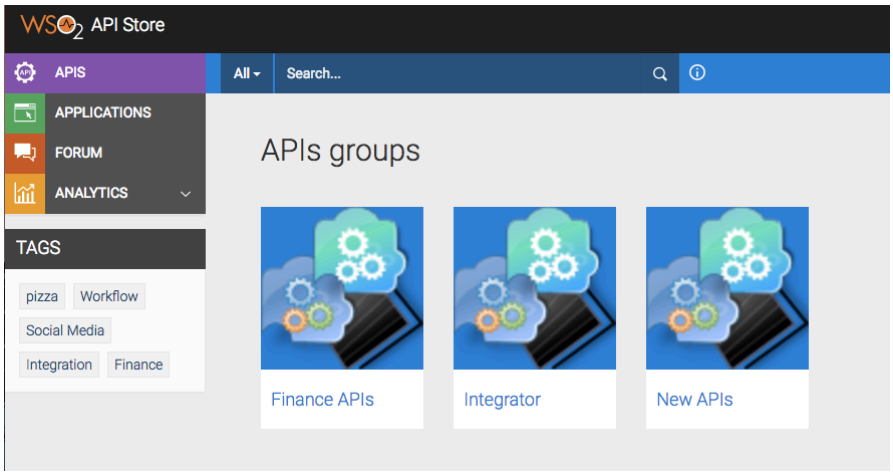


c. Add a group name to the APIs that you wish to group.

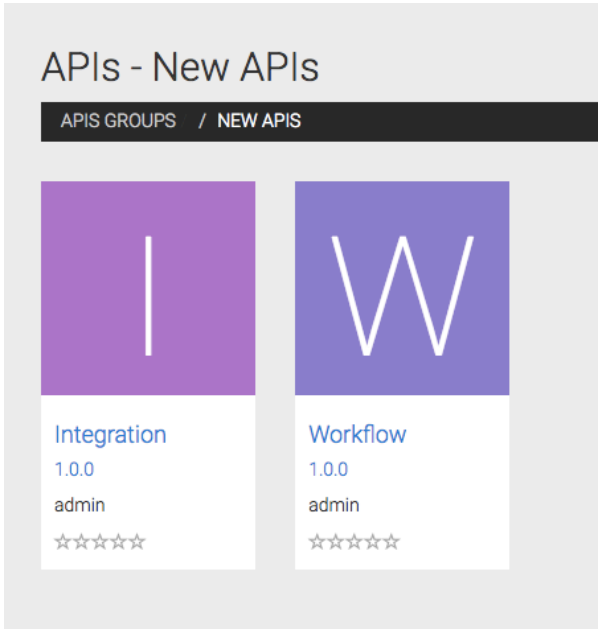
   For example add the "APIs groups-group" tag to the Workflow and Integration APIs.

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.



If you wish, you can click on a group to see the APIs that belong to a specific group.

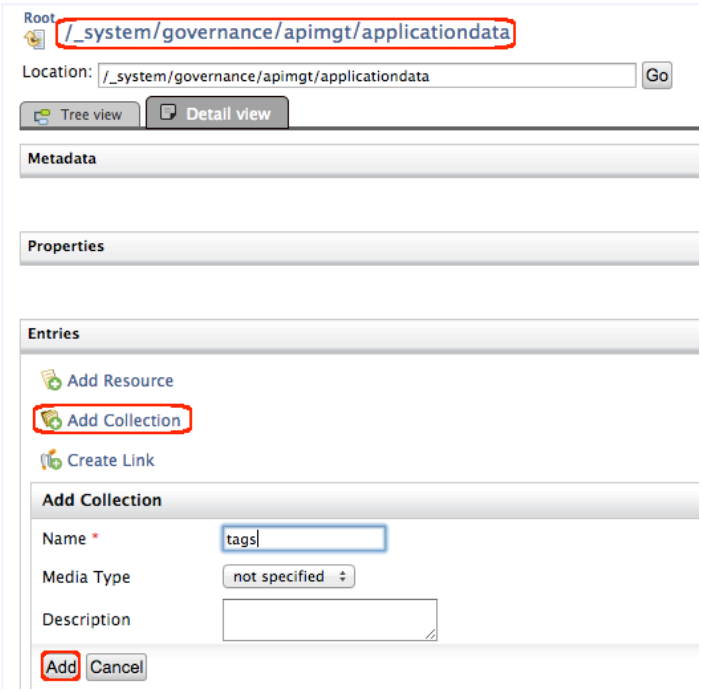


Customizing the API group (optional)

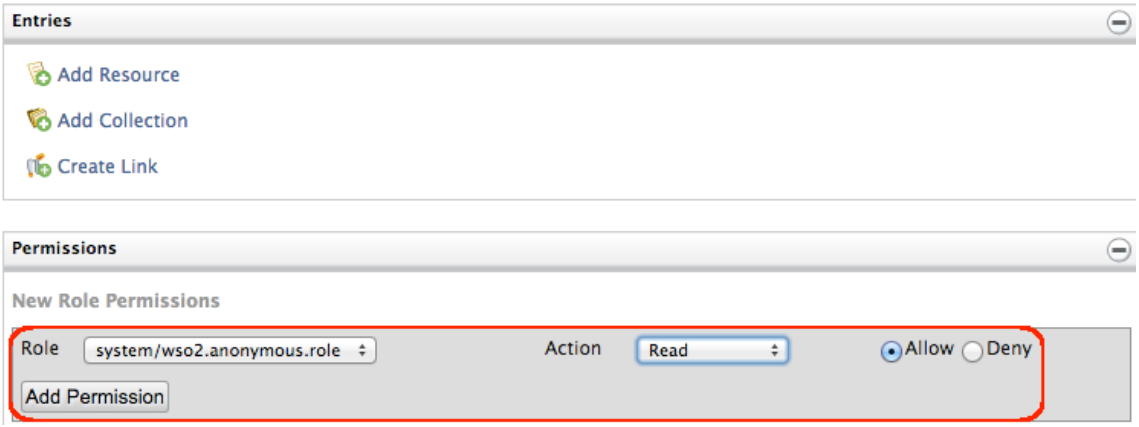
If you want to change the descriptions and the thumbnail images that come by default, do the following:

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/applicationdata`.



3. Give read permission to the `system/wso2.anonymous.role` role.



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.

Demo 8 – Migrating the APIs and Applications to a Different Environment

WSO2 API Manager (WSO2 API-M) allows you to maintain multiple environments running on the same WSO2 API-M version. This feature allows you to import and export applications or APIs between your environments. For example, if you have an application or API running in the development environment, you can import it and export to the test(QA) environment. Thereby, APIs and applications do not have to be created from scratch in different environments.

Getting Started

After running the CLI tool make sure to add an environment before you start working with the import/export CLI commands, because all APIs and applications need to be imported or exported to a specific environment.

Running the CLI tool

Step 1 - Deploy the API import/export tool

1. Download the latest version of WSO2 API Manager from http://wso2.com/products/api-manager/.

2. Start WSO2 API Manger.

3. Download the latest WSO2 API import/export tool (`api-import-export-2.6.0-v1.war`) from here.

4. Note that the import/export tool attached is specific to this version of WSO2 API Manager.

5. Copy the downloaded `api-import-export-2.6.0-v1.war` file to the `<API-M_HOME>/repository/deployment/server/webapps` folder.

   The file is automatically deployed as hot deployment is enabled.

Step 2- Run the CLI tool

1. Navigate to the API Management Tooling page - https://wso2.com/api-management/tooling/

2. Click **Download** under **CLI**.

3. Select a generated archive suitable for your platform (i.e., Mac, Windows, Linux) and extract it the CLI tool that you downloaded to a desired location and `cd` into it.

4. Navigate to the working directory where the executable CLI Tool resides.

5. Execute the following command to start the CLI tool.

6. ./apimcli

7. Add the location of the extracted folder to your system's `$PATH` variable to be able to access the executable from anywhere.

8. For further instructions execute the following command.

```
apimcli –help
```

Global flags for CLI tool

The following are some global flags that you can use with the CLI tool.

```
  --verbose
       Enable verbose logs (Provides more information on execution)
  --insecure, -k
      Allow connections to SSL sites without certs
  --help, -h
      Display information and example usage of a command
```

Adding an environment

You can add environments by either manually editing the `$HOME/.wso2apimcli/main_config.yaml` file or by running the following CLI command.

```
$ apimcli add-env
```

The directory structure for the configuration files (`$HOME/.wso2apimcli`) will be created upon execution of the `apimcli` command.

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to add an environment.

```
apimcli add-env -n <environment-name> \
                        --registration <registration-endpoint> \
                        --apim <API-Manager-endpoint> \
                        --token <token-endpoint> \
                        --import-export <endpoint-for-environment> \
                        --admin <admin-REST-API-endpoint> \
                        --api_list <API-listing-REST-API-endpoint> \
                        --app_list <application-listing-REST-API-
endpoint>
```

3. **Flags:**

   Required flags:

   --name, -n

   There are no short flags for the following flags.

   ```
   --registration
   --apim
   --token
   --import-export
   --admin
   --api_list
   --app_list
   ```

   Removing an environment

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to remove an environment.

```
apimcli remove-env -n <environment-name>
apimcli remove-env --name <environment-name>
```

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.

2. Run the following CLI command to list the environments.
   There are no flags for the following CLI command.

```
apimcli list envs
```

Migrating APIs to different environments

Exporting an API

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.
2. Run the following CLI command to export an existing API as a `.zip` archive.

```
apimcli export-api -n <API-name> -v <version> -r <provider> -e
<environment> -u <username> -p <password> -k
apimcli export-api --name <API-name> --version <version> --provider
<provider> --environment <environment> --username <username> --password
<password> --insecure
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

Required flags:

```
--name, -n
--version, -v
--provider, -r
--environment, -e
--insecure, -k : This allows connections to SSL sites without
certificates
```

Optional flags:

```
--username, -u
--password, -p
```

Importing an API

When you import an API, regardless the status of the imported API it will be added with the created state and you need to sign in to the Publisher and publish the API.

You can use the archive created in the previous section to import APIs to an API Manager instance.

1. Make sure that WSO2 API Manager is started and that the CLI import/export tool is running.

   For Secure Endpoint Enabled APIs:

   If you have enabled secure endpoints when creating the API and your username or/and password differs in the two environments, please follow the steps below before importing the API.

   1. Unzip the .zip archive created in the previous section.

   2. Go to the `<API-name-version>/Meta-information` directory and open the `api.json` file.
   For example, `PhoneVerification_1.0.0/Meta-information` directory and open the `api.json` file.

   3. Modify the `endpointUTPassword` with your endpoint password and save the `api.json` file.

   4. Compress the `PhoneVerification_1.0.0` folder to a folder named `myExportedAPI`.

- Run the following CLI command to import an API.

```
apimcli import-api -f <environment>/<file> -e <environment> -u
<username> -p <password> -k
apimcli import-api --file <environment>/<file> --environment
<environment> --username <username> --password <password> --insecure
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

Required flags:

`--file, -f : The file path of the exported API. For example, if your file path is /Users/kim/.wso2apimcli/exported/apis/dev/PhoneVerification_1.0.0.zip.`, then you need to enter `dev/PhoneVerification_1.0.0.zip` as the value for this flag.

`--environment, -e : The environment to which you what to import the API to.`

`--insecure, -k : This allows connections to SSL sites without certificates`

Optional flags:

`--username, -u`

`--password, -p`

You must add a parameter named `--preserve-provider` to the CLI 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 CLI command. Here's an example:

```
apimcli import-api -k -f <environment>/<file> -e <environment> -u
<username> -p <password> --preserve-provider <preserve_provider>
apimcli import-api --insecure --file <environment>/<file> --environment
<environment> -u <username> -p <password> --preserve-
provider=<preserve_provider>
```

The username and password are optional flags. You will be prompted to enter your credentials if you do not provide these flags.

Flags:

Required flags:

`--insecure, -k : This allows connections to SSL sites without certificates`

`--file, -f`

`--environment, -e : The environment to which you what to import the API to.`

`--preserve-provider (This does not have a short flag)`

Optional flags:

`--username, -u`

`--password, -p`

The `--preserve-provider` flag 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 `--preserve-provider` 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 `--preserve-provider` 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.

Rename the `<lastAccessTimeLocation>` element in the `<API-M_2.6.0_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/components/org.wso2.carbon.registry/indexing/lastaccesstime` registry path to `/_system/local/repository/components/org.wso2.carbon.registry/indexing/lastaccesstime_1.`

1. Shut down the API Manager 2.6.0, backup and delete the `<API-M_2.6.0_HOME>/solr` directory.

For more information, see Upgrading the API Manager to 2.6.0.

Demo 9 – Microgateway Quick Start

Before you begin,

1. Download the Microgateway distribution and extract it.
2. Append the full path of the `/bin` folder of the extracted Microgateway distribution to the PATH environment variable.
   If you are unable to append the path, you can alternatively run the Microgateway commands by navigating to the `<MICROGW_HOME>/bin` folder and running the commands as `./micro-gw` instead.

   Let's go through the main use case of the Microgateway:

   Deploy and subscribe to the sample API

1. Open the API Publisher (`https://<hostname>:9443/publisher`) and sign in with `admin/admin` credentials.
2. Close the interactive tutorial that starts automatically if you are a first-time user.
3. Click the **Deploy Sample API** button. It deploys a sample API called `PizzaShackAPI` into the API Manager.
   The **Deploy Sample API** option is available only when there are no APIs in API Publisher. If you have already created a API, this option is not available.
4. Click `PizzaShackAPI` to open it.
5. Go to the **Lifecycle** tab and note that the **State** is `PUBLISHED`. The API is already published to the API Store.
6. Sign in to the API Store (`https://<hostname>:9443/store`) with the `admin/admin` credentials and click the `PizzaShackAPI` API.
7. Select the default application and an available tier, and click **Subscribe**.
8. When the subscription is successful, click **View Subscriptions** on the information message that appears.

   You have now successfully subscribed to an API. Let's generate a distribution for this API.

   Generate a distribution for a single API

   It is possible to create a Microgateway distribution for a single API that is in the **Published** state.

   For details on how to create a Microgateway distribution for a group of APIs, see Grouping APIs with Labels.

1. Navigate to a preferred workspace folder using the command line.
   This location is used to run the Microgateway commands and to generate Microgateway artifacts.

2. Set up a project using the command given below,

   ```
   micro-gw setup <project_name> -a <API_name> -v <version>
   ```

3. When the tool requests for the username and password, use `admin` in both instances. Use the default values for the other parameters by pressing **Enter**.

4. Here is an example:

```
$ micro-gw setup pizzashack-project -a PizzaShackAPI -v 1.0.0
Enter Username:
admin
Enter Password for admin:

Enter APIM base URL [https://localhost:9443/]:

Enter Trust store location:
[lib/platform/bre/security/ballerinaTruststore.p12]

Enter Trust store password: [ use default? ]

Setting up project pizzashack-project is successful.
```

5. When the above command is issued, the tool connects with API Manager REST APIs and retrieves the API specified above. The source artifacts are generated in the current folder location.

6. The folder structure looks similar to the following,

```
.
└── pizzashack-project
    ├── conf
    │   └── deployment-config.toml
    ├── src
    │   ├── extension_filter.bal
    │   ├── listeners.bal
    │   ├── PizzaShackAPI_1_0_0.bal
    │   └── policies
    │       ├── application_10PerMin.bal
    │       ├── application_20PerMin.bal
    │       ├── ...
    │       ├── subscription_Bronze.bal
    │       ├── subscription_Gold.bal
    │       ├── ...
    │       ├── subscription_Unauthenticated.bal
```

```
    │       └── throttle_policy_initializer.bal
    └── target
```

7. If you re-run the setup command while you already have a project named `pizzashack-project` in the current working directory, you receive an error similar to the following:

```
$ micro-gw setup pizzashack-project -a PizzaShackAPI -v 1.0.0

[micro-gw: Project name `pizzashack-project` already exist. use -f
or --force to forcefully update the project directory., Run 'micro-
gw help' for usage.]
```

8. As suggested in the above output, if you need to override the current project, run the setup command with the `-f (--force)` option.

```
$ micro-gw setup pizzashack-project -a PizzaShackAPI -v 1.0.0  -f

Enter Password for admin:

Setting up project pizzashack-project is successful.
```

9. Build the microgateway distribution for the project using the following command:

```
micro-gw build <project_name>
```

10. Here is an example:

```
$ micro-gw build pizzashack-project

Build successful for the project - pizzashack-project
```

11. Once the above command is executed, the generated source files are built and a Microgateway distribution is created in the `target` folder.

12. The name of the distribution will have the format `micro-gw-<project_name>.zip`. In the above example, the name will be `micro-gw-pizzashack-project.zip`.
13. Next, unzip the `micro-gw-pizzashack-project.zip` and run the `gateway` shell script inside the `bin` folder of the extracted zip using the following command:

```
bash gateway
```

14. The Microgateway will now start for the Pizzashack API. For example,

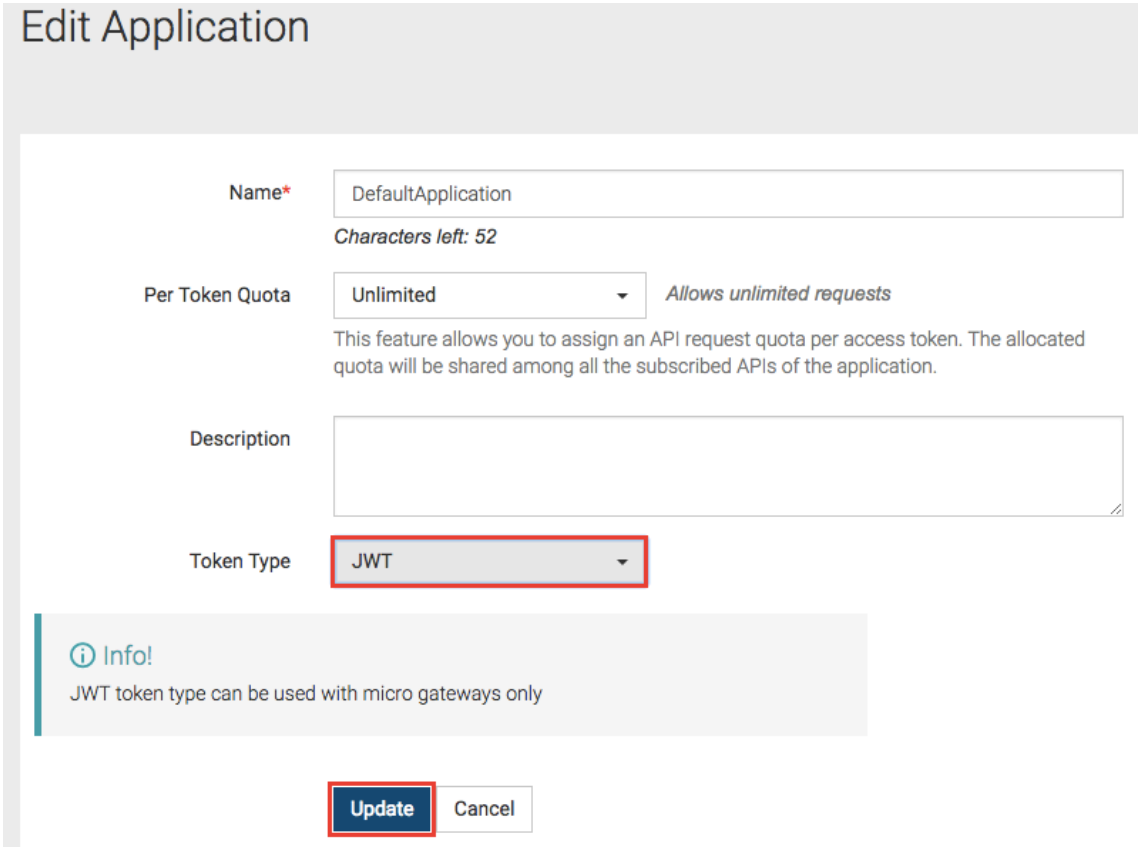
```
micro-gw-pizzashack-project/bin$ bash gateway

ballerina: HTTP access log enabled
ballerina: initiating service(s)
in '/home/user/workspace/pizzashack-project/target/micro-gw-
pizzashack-project/exec/pizzashack-project.balx'
ballerina: started HTTPS/WSS endpoint 0.0.0.0:9096
ballerina: started HTTPS/WSS endpoint 0.0.0.0:9095
ballerina: started HTTP/WS endpoint 0.0.0.0:9090
```

You have successfully generated and started a Microgateway distribution for the Pizzashack API.

Generate a JWT token and invoke the API

Once you start the Microgateway, you can use a JWT or an OAuth2 token to invoke the API. For the invocation URL, you can use
either `https://localhost:9095/` (HTTPS)
or `http://localhost:9090/` (HTTP).
A signed JWT token is a self-contained access token that can be validated by the Microgateway itself without connecting to any external party. In contrast, a typical OAuth2 token needs to be validated by connecting to the Key Manager. This is one of the key advantages of the Microgateway, if you need to run it in offline mode.

1. Log in to the API Store, click **Applications** and choose to edit the Default Application. Set the token type to JWT.



2. Click the **Production Keys** tab and click **Generate Keys** to generate an access token to invoke the API.

   If you have already generated keys before, click **Regenerate**.

3. Invoke the API using the JWT token using the following command:

   Format

```
curl -k -i -H "Authorization: Bearer <JWT_token>" <API_url>
```

   For example,

   Example

```
curl -k -i -H "Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IlVCX0JReTJIRlYzRU1UZ3E2
NFEtMVZpdFliRSJ9.eyJhdWQiOiJodHRwOlwvXC9vcmcud3NvMi5hcGltZ3RcL2dhdGV
3YXkiLCJzdWIiOiJhZG1pbiIsImFwcGxpY2F0aW9uIjp7ImlkIjoxLCJuYW1lIjoiRGV
mYXVsdEFwcGxpY2F0aW9uIiwidGllciI6IlVubGltaXRlZCJ9LCJzY29wZSI6ImFtX2F
wcGxpY2F0aW9uX3Njb3BlIGRlZmF1bHQiLCJpc3MiOiJodHRwczpcL1wvbG9jYWxob3N
```

```
0OjgyNDNcL3Rva2VuIiwia2V5dHlwZSI6IlBST0RVQ1RJT04iLCJzdWJzY3JpYmVkQVB
JcyI6W3sibmFtZSI6IlBpenphU2hhY2tBUEkiLCJjb250ZXh0IjoiXC9waXp6YXNoYWN
rXC8xLjAuMCIsInZlcnNpb24iOiIxLjAuMCIsInB1Ymxpc2hlciI6ImFkbWluIiwic3V
ic2NyaXB0aW9uVGllciI6IlVubGltaXRlZCIsInN1YnNjcmliZXJUZW5hbnREb21haW4
iOiJjYXJib24uc3VwZXIifV0sImV4cCI6MTUyODgwMjg3ODExMCwiaWF0IjoxNTI4ODA
yODgwMTEwLCJqdGkiOiJlZmQxZGVhZC1iNWI3LTQ0M2MtOTIyMC1iYzJjZTJjY2MwYjE
ifQ==.b8P2uPRkVai0O7PcbvbjANLuHlJQzX1eHplweDpE6ItbEHRTkN2U_h6b39tz14
dKUmigzASinj5LheUWGB7gEDRqlc39ckhRX2qpolQpITZvpzYo8ky9AcxlJXLxrfPwgd
ht36zfIQwlPN_s2A5nY7c9pDBMu0OOOlYpmK81SrtipFSTAyPiRg5VyY3n-4POnjkEF-
LQKCCTq7ef0uLOFTcoCT-gqNsXeKqt15suCYj5QMHJ8VP5bKsKZy9-
1o9oFNlwc1QE0qE01fPuGuz-
4J22OvkHyrasbjhhGaaDgdpdERl9ElUDuL0C9AdX6Fb1sz54gnAiU3RUBK3RQUDK7Q==
" https://localhost:9095/pizzashack/1.0.0/menu
```

You receive a response similar to the following:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jul 2018 16:52:38 GMT
Content-Type: application/json
Transfer-Encoding: chunked
server: WSO2 Carbon Server

[{"name":"BBQ Chicken Bacon","description":"Grilled white chicken,
hickory-smoked bacon and fresh sliced onions in barbeque
sauce","price":"13.99","icon":"/images/6.png"},{"name":"Chicken
Parmesan","description":"Grilled chicken, fresh tomatoes, feta and
mozzarella
cheese","price":"9.99","icon":"/images/1.png"},{"name":"Chilly
Chicken Cordon Bleu","description":"Spinash Alfredo sauce topped
with grilled chicken, ham, onions and
mozzarella","price":"17.99","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":"16.99","icon":"/images/9.png"},{"name":"Garden
Fresh","description":"Slices onions and green peppers, gourmet
mushrooms, black olives and ripe Roma
tomatoes","price":"9.99","icon":"/images/3.png"},{"name":"Grilled
Chicken Club","description":"Grilled white chicken, hickory-smoked
bacon and fresh sliced onions topped with Roma
tomatoes","price":"13.99","icon":"/images/8.png"},{"name":"Hawaiian
BBQ Chicken","description":"Grilled white chicken, hickory-smoked
bacon, barbeque sauce topped with sweet pine-
apple","price":"27.99","icon":"/images/7.png"},{"name":"Spicy
Italian","description":"Pepperoni and a double portion of spicy
Italian
```

```
sausage","price":"13.99","icon":"/images/2.png"},{"name":"Spinach
Alfredo","description":"Rich and creamy blend of spinach and garlic
Parmesan with Alfredo
sauce","price":"27.99","icon":"/images/5.png"},{"name":"Tuscan Six
Cheese","description":"Six cheese blend of mozzarella, Parmesan,
Romano, Asiago and Fontina","price":"11.99","icon":"/images/4.png"}]
```

Demo 10 – Deploying the API Microgateway in Kubernetes

1. Start the API Manager and log in to the API Publisher (`https://<hostname>:9443/publisher`) using `admin` as the username and password.
2. Create an API with the following details:

Field	Sample value
Name	hello_world - v1
Context	`/hello/v1`
Version	1.0.0
Access Control	All
Visibility on Store	Public
Production URL	http://bk.test.com
Tier Availability	Gold, Unlimited

3. You can create an application that supports JWT tokens, subscribe to the API and get a JWT token to invoke the API.

4. Create a `deployment.toml` file enabling Kubernetes deployment, service and config map resources.

   The config map is used to copy the `micro-gw.conf` file.

```
[kubernetes]
  [kubernetes.kubernetesDeployment]
    enable = true
    #name = ''
    #labels = ''
    #replicas = ''
    #enableLiveness = ''
    #initialDelaySeconds = ''
    #periodSeconds = ''
    #livenessPort = ''
    #imagePullPolicy = ''
    #image = ''
```

```
    #env = ''
    #buildImage = ''
    #copyFiles = ''
    #dockerHost = ''
    #dockerCertPath = ''
    #push = ''
    #username = ''
    #password = ''
    #baseImage = ''
    #singleYAML = ''
  [kubernetes.kubernetesService]
    enable = true
    #name = ''
    #labels = ''
    serviceType = 'NodePort'
    #port = ''
  [kubernetes.kubernetesConfigMap]
    enable = true
    ballerinaConf = '/home/user/wso2am-micro-gw-toolkit-
2.5.0/resources/conf/micro-gw.conf'
    #[[kubernetes.kubernetesConfigMap.configMaps]]
    #    name = ''
    #    mountPath = ''
    #    readOnly = false
    #    data = ['']
```

Let's create a project called `k8s_project` and provide the `deployment.toml` file as an input.

5. Navigate to the `wso2am-micro-gw-toolkit-2.5.0/bin` directory and run the following command,

```
./micro-gw setup k8s_project -a hello_world -v v1 --deployment-config deployment.toml
```

6. This commands creates the following folders under the `k8s_project` folder.

```
├── k8s_project
│   ├── conf
│   │   └── deployment-config.toml
│   ├── src
│   │   ├── extension_filter.bal
│   │   ├── hello_world_v1.bal
│   │   ├── listeners.bal
│   │   └── policies
│   │       ├── application_10PerMin.bal
```

```
│   │   ├── application_20PerMin.bal
│   │   ├── application_50PerMin.bal
│   │   ├── subscription_Bronze.bal
│   │   ├── subscription_Gold.bal
│   │   ├── subscription_Silver.bal
│   │   ├── subscription_Unauthenticated.bal
│   │   └── throttle_policy_initializer.bal
│   ├── target
│   └── temp
│       └── hashes.json
```

7. Build the project using the following command,

```
./micro-gw build k8s_project
```

8. This generates the following Kubernetes resources.

```
├── k8s_project
│   └── docker
│       └── Dockerfile
├── k8s_project_config_map.yaml
├── k8s_project_deployment.yaml
└── k8s_project_svc.yaml
```

9. The docker image to be deployed in Kubernetes is created in your local registry. You can find the image `k8s_project:latest` when you execute the docker images command.

10. Deploy the docker image in a Kubernetes environment.
    You can push the docker image to the docker hub or to a private docker registry. If you change the docker image name, you need to change the image name in the `k8s_project_deployment.yaml` file.

11. You can also SCP the image to the Kubernetes nodes as follows:

a. Save the docker image to a `tar` file. For example,

```
docker save k8s_project:latest > image.tar
```

b. SCP the image to the Kubernetes nodes. For example,

```
scp -I <identity file> image.tar username@K8s_NODE_IP:
```

c. Load the docker image in the Kubernetes nodes. This needs to be executed in the Kubernetes nodes. For example,

```
docker load < image.tar
```

12. Deploy the API Microgateway in Kubernetes by deploying the Kubernetes resources using the following command,

```
kubectl create -f k8s_project/target/kubernetes/
```

13. Access the API in HTTPS using the following details:
The `NodePort` service type has been used in Kubernetes. For that service type, you can access the API using any of the Kubernetes node IP addresses and node ports.

`https://<Any_Kubernetes_Node_IP>:<NodePort>/hello/v1/check`

You can use the `kubectl get services` command to list down the services that run on Kubernetes.

Node port - 32616

URL - `https://<Any_Kubernetes_Node_IP>:32616/hello/v1/check`

Headers - `Authorization Bearer <JWT_TOKEN>`

Method - `GET`

As JWT is a self-contained access token, the Microgateway does not need to connect to the Key Manager. However, if you are using an Oauth2 access token, point the Microgateway to the Key Manager using the Key Manager details in the `micro-gw.conf` configuration file of the Microgateway. If you are running Key Manager in Kubernetes, you can provide the Key Manager `serverUrl` as shown below. The `serverUrl` has to be accessible from the Microgateway pods.

```
[keyManager]
```

```
serverUrl="https://localhost:9443"
username="admin"
password="admin"
tokenContext="oauth2"
timestampSkew=5000
```