# Lab 11: Hybrid integration and Deployment

[Lab 11: Hybrid integration and Deployment](#_ooaxqb2s6ns4)

[Overview](#_scdppmli5w09)

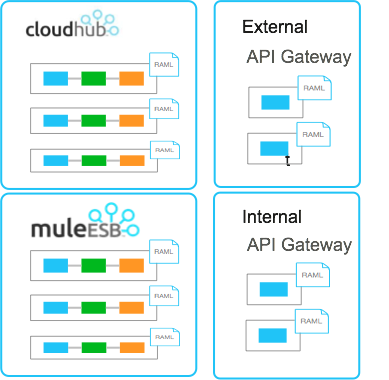
[Step 1: Call Cloud from On-Premise](#_pmzmh8mu91uz)

[Step 2: Deploy to Mule On-Premise through the management console](#_qo34hvwd5lv5)

[Step 3: Proxy your API in a Cloud-Based API Gateway](#_hn9gf0q9g9yh)

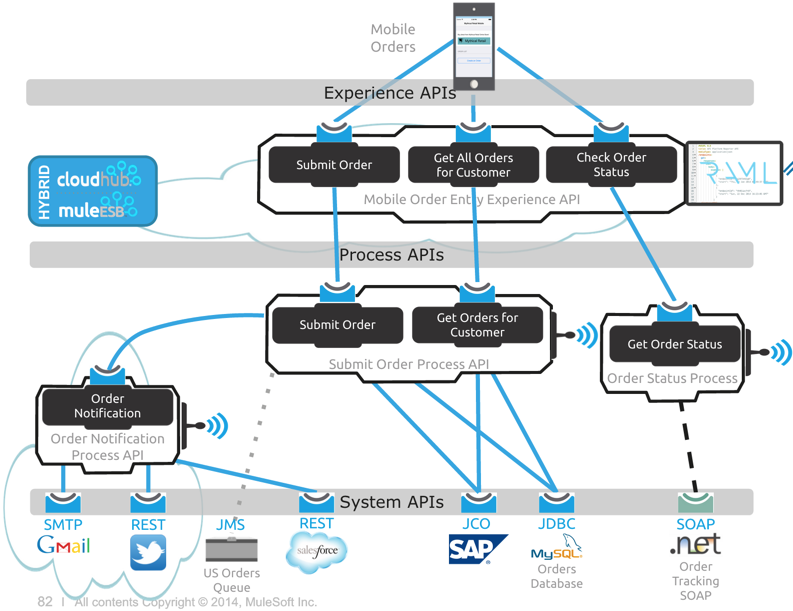
[Summary](#_eauu08fs6i0p)

# Overview



Let us now explore the flexibility and power of the Anypoint Platform in not only being able to integrate systems on-premise and on the cloud, but also providing a fully hybrid architecture between cloud and on-premise, as well as APIs and integration.

First, we will integrate the **Order Broadcast API** that you deployed on CloudHub to our existing **Order Entry API**. Then we will deploy our Order Entry API to Mule ESB running on-premise to facilitate the order submission and check order status processes.. Lastly, we will proxy the Order Entry API to our external API Gateway running on CloudHub.

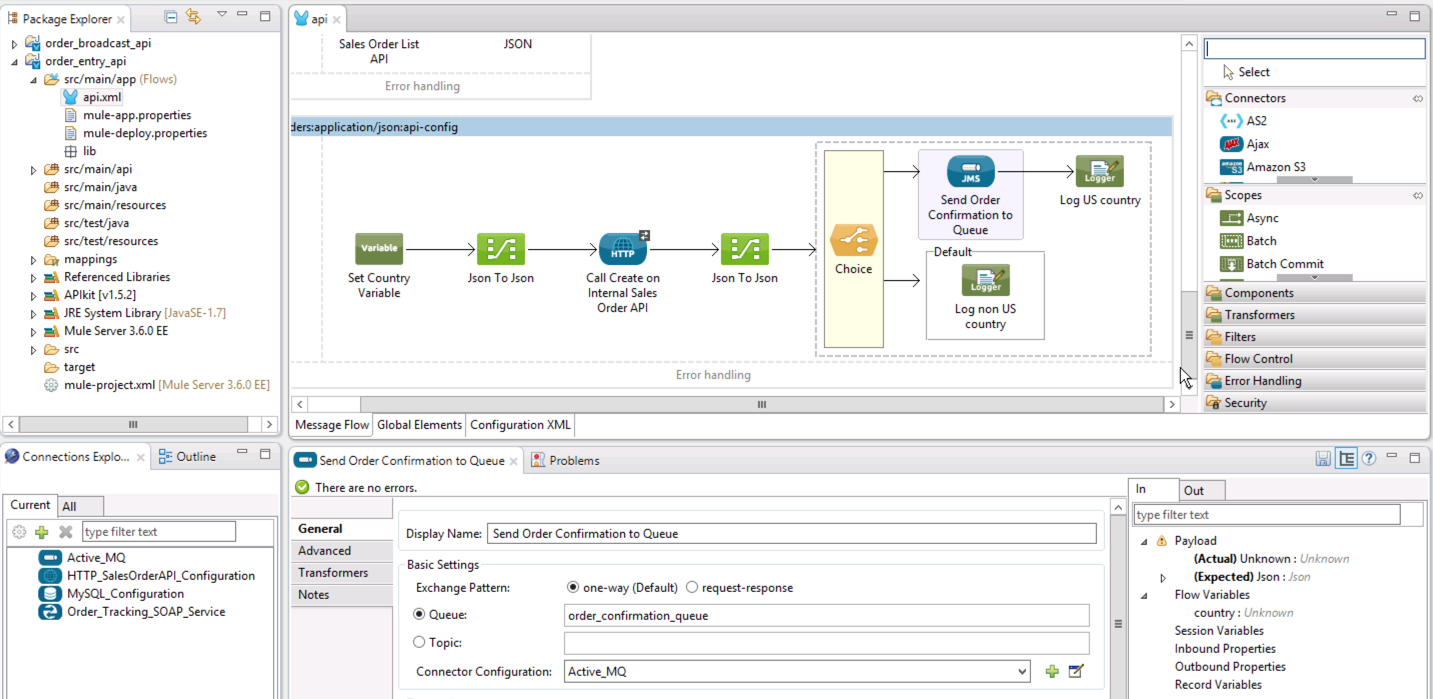


## 

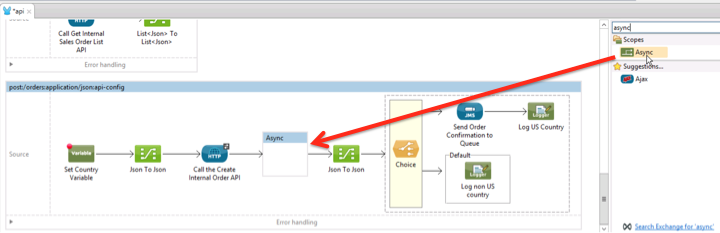
## 

## Step 1: Call Cloud from On-Premise

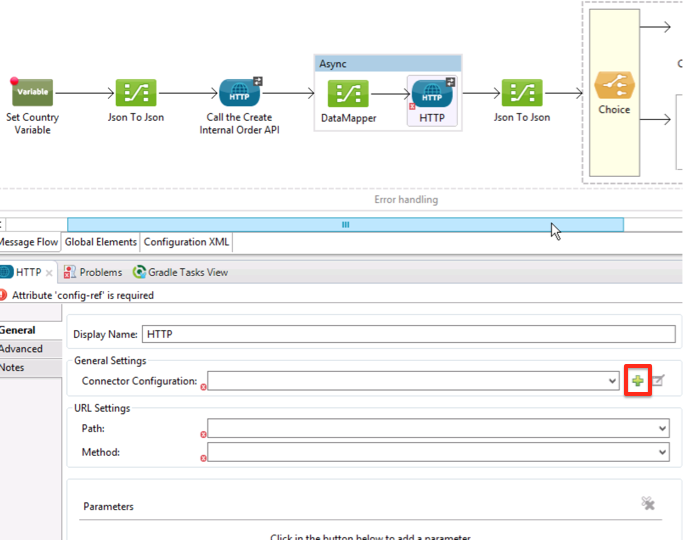
Let us now call the Order Broadcast API that we deployed on the Cloud from the Order Entry API that we will have running on-premise



1. In Anypoint Studio, open the **order\_entry\_api** project. If you recall, we are accepting orders through a RAML API, processing the order in our internal Sales Order system. We need to call the Order Broadcast API after we process the order.

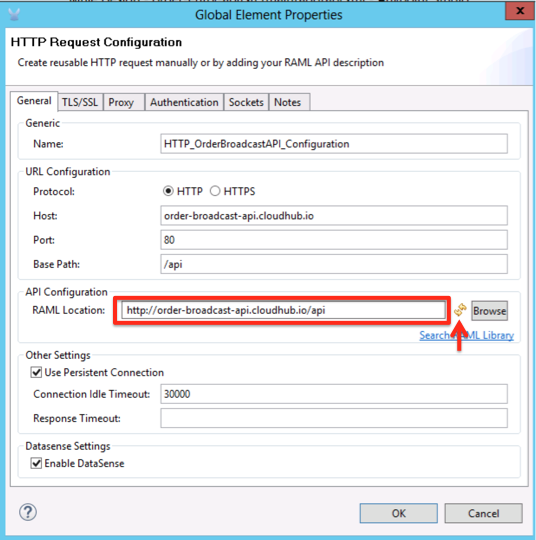


1. Add an **Async** Scope after the **Call the Create Internal Order API** .

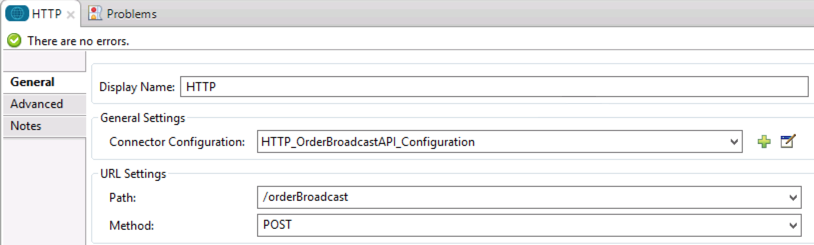


Inside the scope

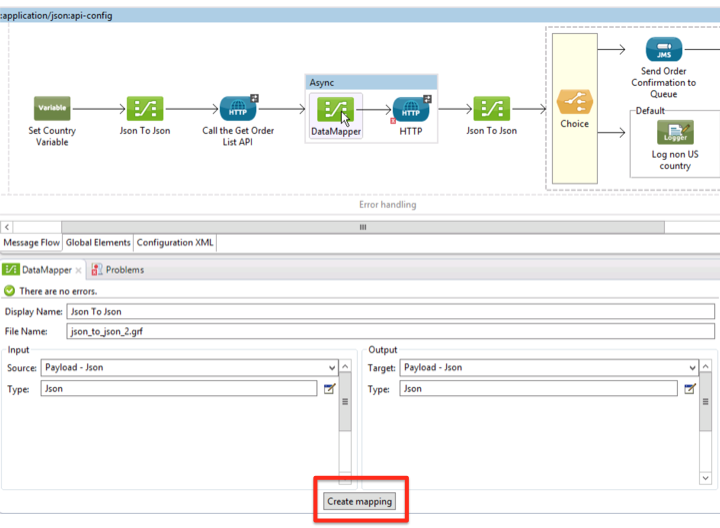
1. Add a **Data Mapper**
2. Add an **HTTP Connector**.
3. Click the icon to configure the HTTP Connector.



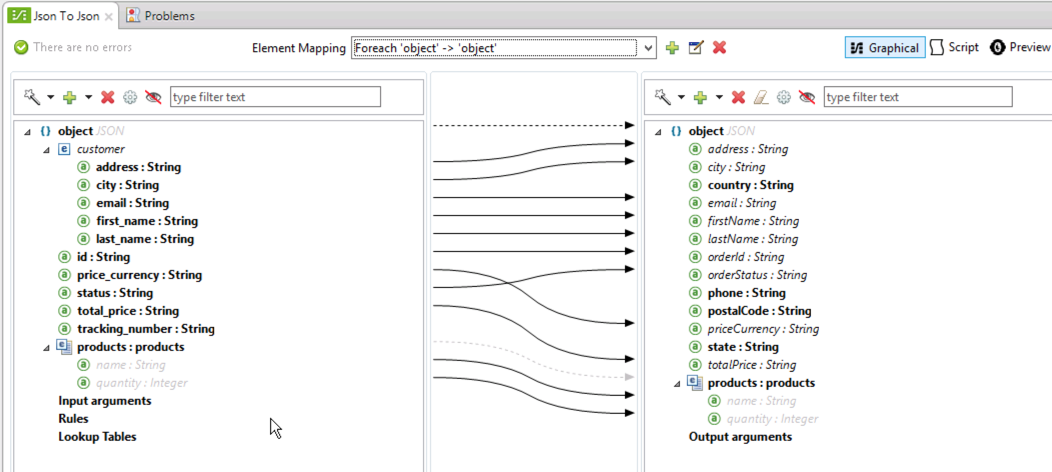
1. Name the HTTP Configuration **HTTP\_OrderBroadcastAPI\_Configuration**
2. Specify the RAML location for your Order Broadcast API. It should use the domain that you specified in lab 9 for your CloudHub deployment. Example: [**http://orderbroadcast.cloudhub.io/api/**](http://orderbroadcast.cloudhub.io/api/)**.**
3. Click the icon to populate the URL Configuration, Click **OK**.



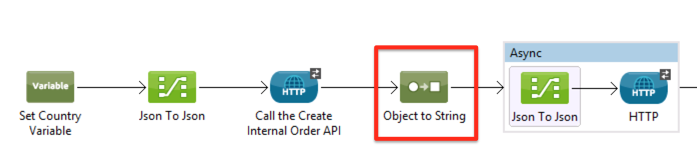
1. Your URL Settings should now be populated with the Path and Method for your Order Broadcast API.



1. Click the **Data Mapper** component and click **Create mapping**.



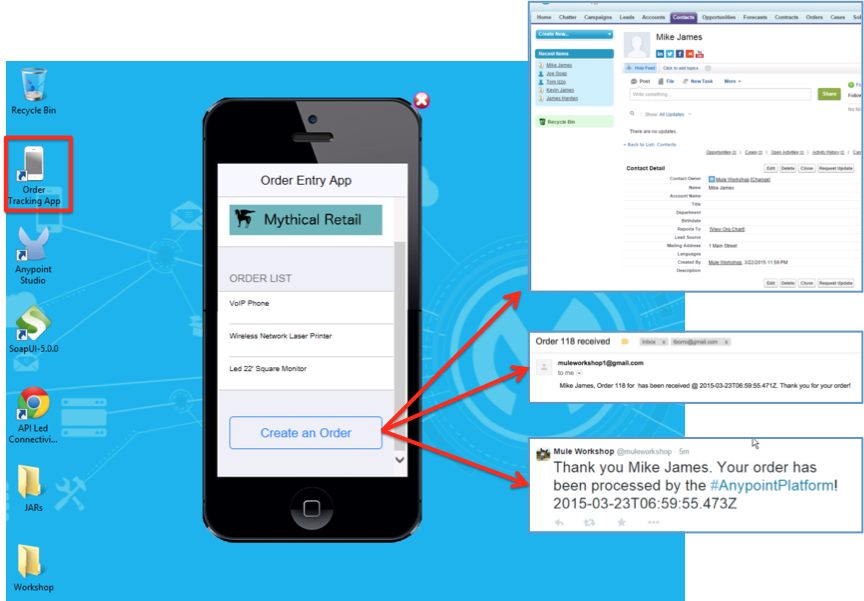
1. Map the fields as shown above.



1. Lastly, add an **Object to String** transformer before the Async scope.

This will allow us to serialize the stream coming from the previous HTTP endpoint to a String, so that it can be reused by 2 parallel threads (due to the addition of the Async scope).

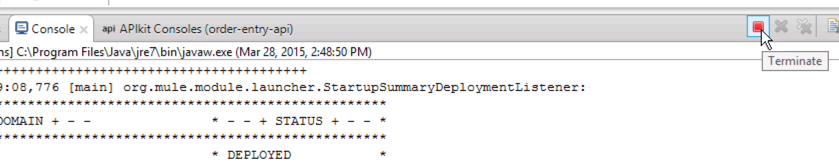
1. Click **Save All**. 



1. Debug your application and try your mobile application again. You should be able to submit orders that will be broad-casted to the SaaS based applications (Salesforce, Gmail and Twitter).

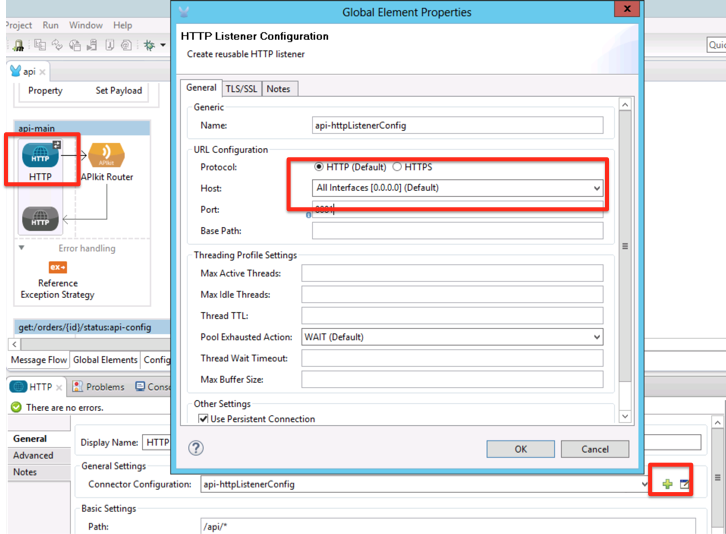
## Step 2: Deploy to Mule On-Premise through the management console

Now that our application is ready, we will deploy it to our on-premise integration platform.

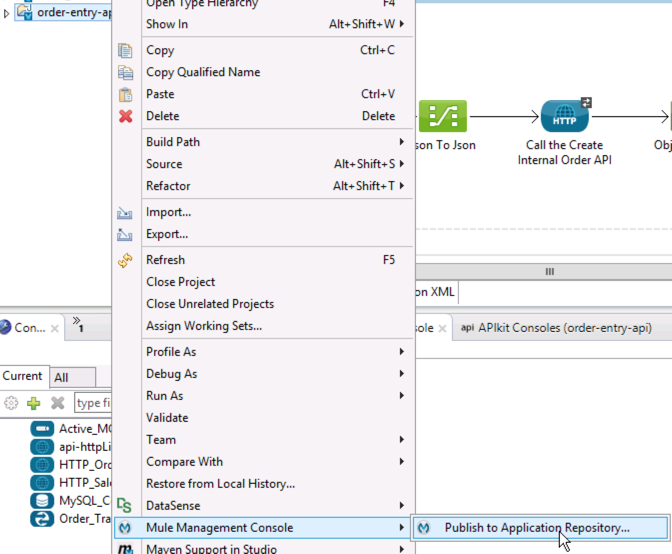


1. First, remember to stop the embedded Mule run-time on Anypoint Studio.

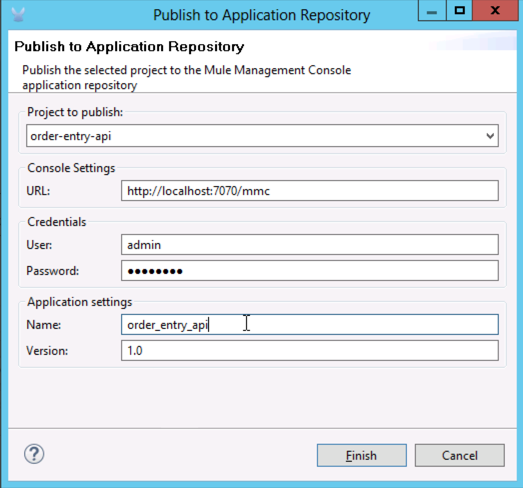
|  |
| --- |
| **NOTE**: Since the embedded run-time and the run-time where you will deploy this application are on the same machine, you should avoid having the application running in both run-times. One of them will not work because both are using the same ports. In real life, you will have your Mule instances running on other servers |



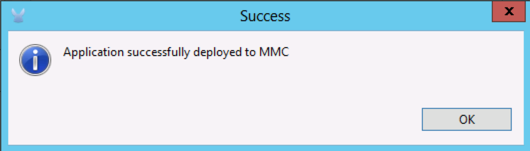
1. Modify the HTTP endpoint to use **0.0.0.0** as the host to allow you to access this API externally.



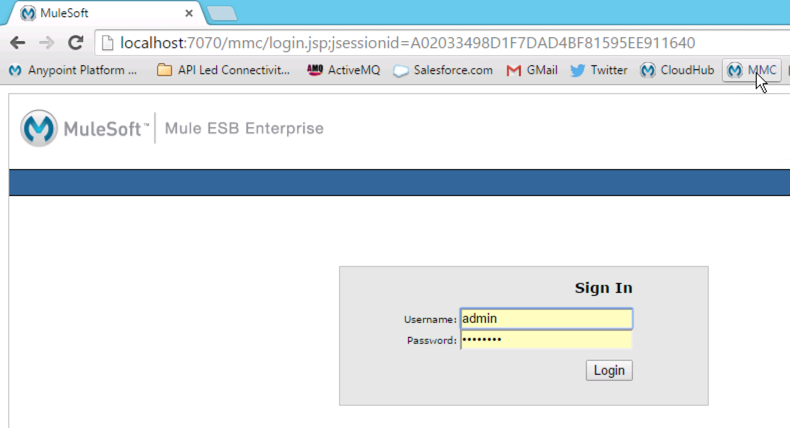
1. Right-click your project, choose **Mule Management Consol**e, and click **Publish to Application Repository.**

****

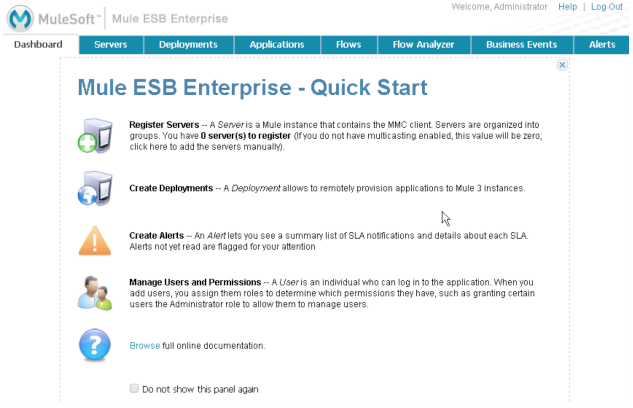
1. Enter the information above to publish your application to the Mule Management Console. The local Mule Management Console URL is: [**http://localhost:7070/mmc**](http://localhost:7070/mmc)**.** The password for the **admin** user is **Mule1379**.



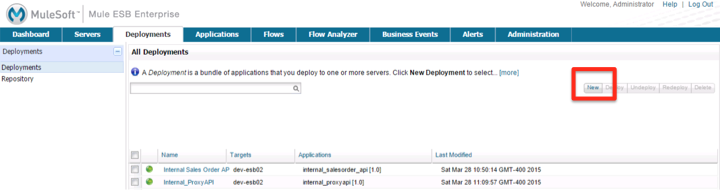
1. You should get a message confirming that your applications has been successfully deployed to MMC.



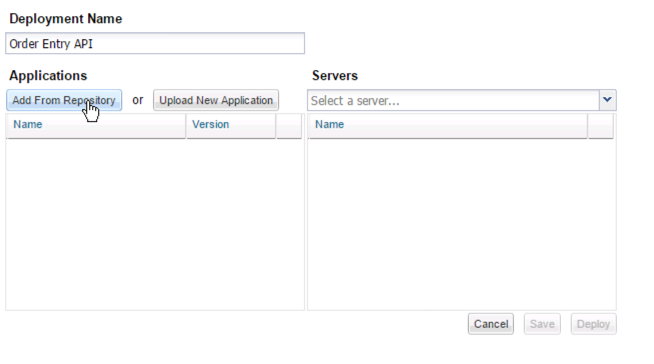
1. Go to http://localhost:7070/mmc/login.jsp and log in using the user **admin** and the password **Mule1379**.



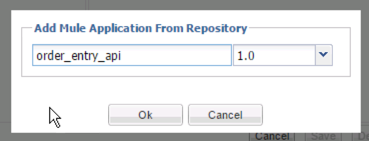
1. Feel free to glance over the Dashboard and Servers tabs to get a lay of the land. While in the Servers tab, click on the name of an actively running server to see detailed stats.



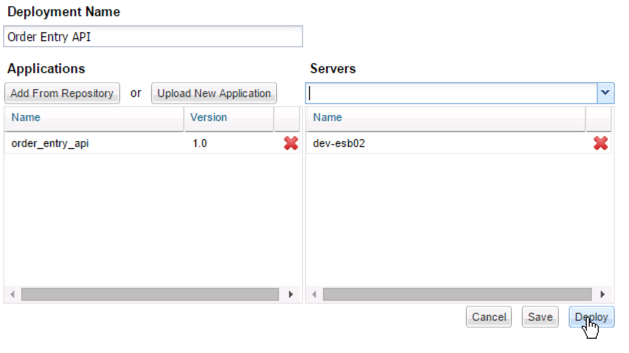
1. Click the Deployment tab and you will see 2 applications already deployed. We will add our Order Entry API by clicking the **New** button.



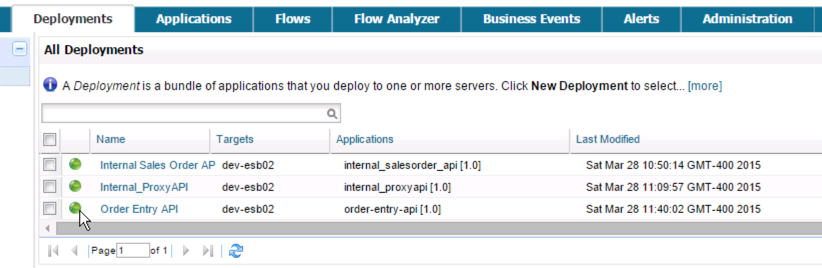
1. Call the deployment **Order Entry API**. Because we already published the Order Entry API app from Anypoint Studio to the Mule Management Console repository, you should be able to simply add this from the repository.



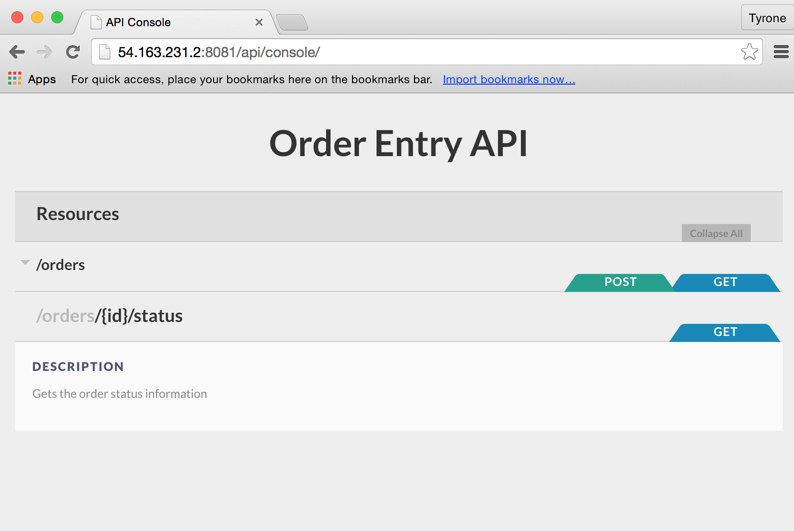
1. Locate the Order Entry API version 1.0 and click **Ok**.



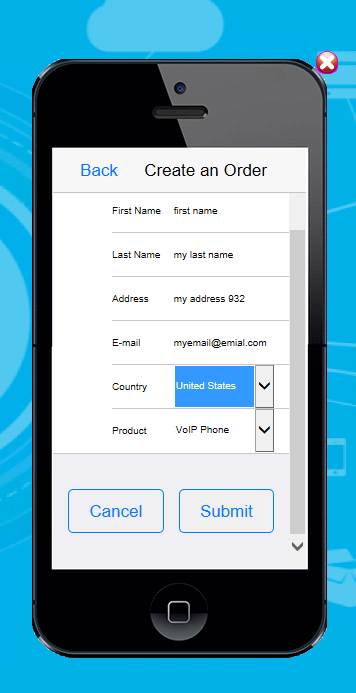
1. Choose the **dev-esb02** server. Your deployment package is defined.
2. Click **Deploy**.



1. Once it finishes you will see your deployment package with a green icon status.



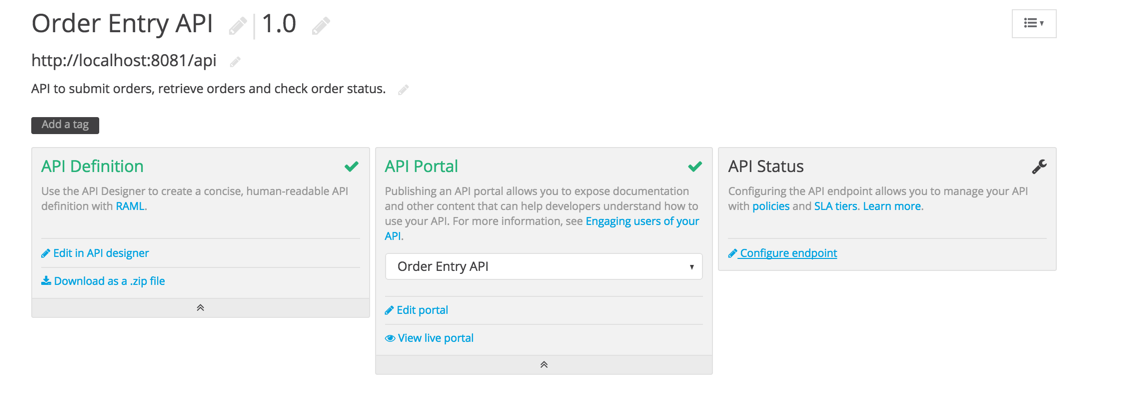
1. Open the web browser and type the URL: <http://localhost:8081/api/console>. You should see the console of your application. You should also be able to access it from outside by replacing localhost with your Workshop IP address for example: **http://54.163.231.2:8081/api/console.**



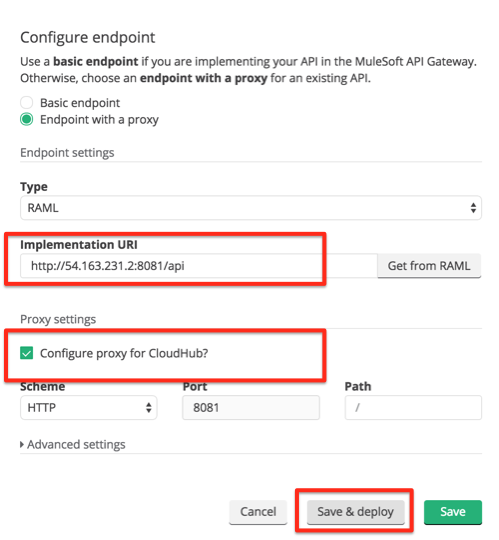
1. For posterity’s sake, try it from your Mobile Application again.

## Step 3: Proxy your API in a Cloud-Based API Gateway

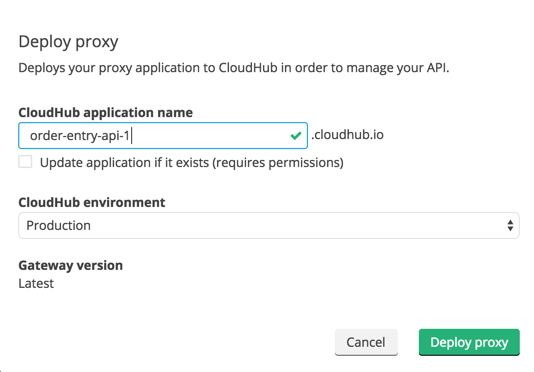
Let us now go back to our API Platform to create a proxy for our internal API and run this on our Cloud-based API Gateway. The gateway will host your API proxy application, which will route the requests to the internal order entry application that you just deployed. Once its running in the gateway, the Anypoint Platform can manage it, keep track of requests and ensure proper governance of our API to external consumers.



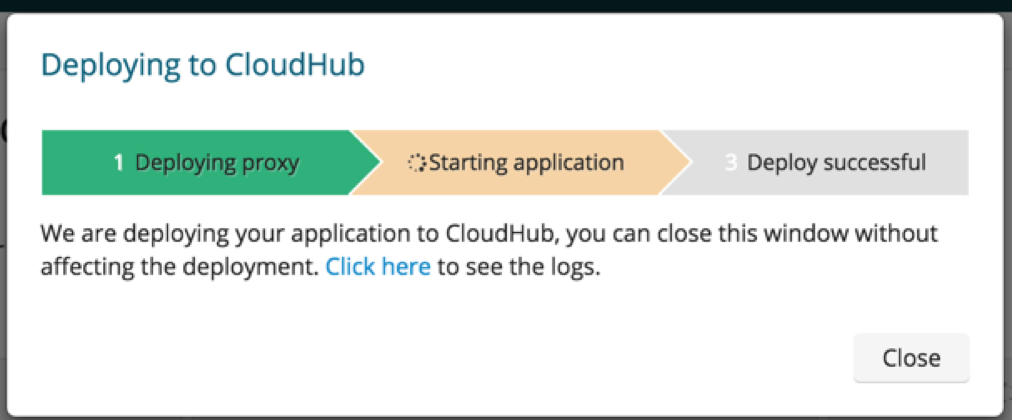
1. Log back in to Anypoint Platform at <http://anypoint.mulesoft.com> and go to the Order Entry API Administration.
2. Under **API Status**, click on: 



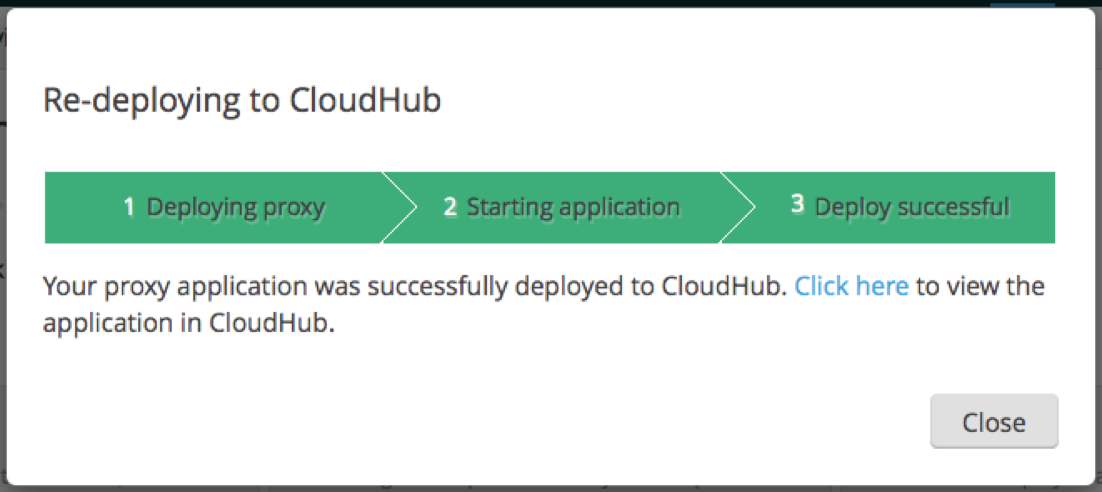
1. Replace the Implementation URI with the URI of your API running on your environment, using your Workshop IP address. For example: [**http://54.163.231.2:8081/api**](http://54.163.231.2:8081/api)
2. Check the box to **Configure proxy for Cloudhub.**
3. Click **Save and Deploy.**

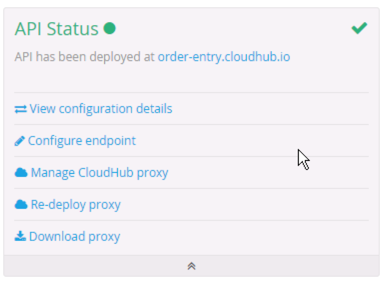
****

1. Set the Cloudhub application name with a unique name.
2. Click **Deploy Proxy**.



1. CloudHub will deploy your proxy on the API Gateway.

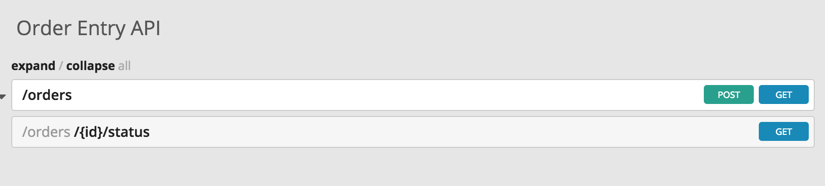




1. Back in the Anypoint Platform API Administration page, you should now see the API URL turn green with a green ball next to it, as shown. This indicates that your API is now being managed.

|  |
| --- |
| **Note**: You can manually deploy the proxy in CloudHub by clicking **Download proxy** and deploying it as a CloudHub application. See the [Deploying a CloudHub Application](http://www.mulesoft.org/documentation/display/current/Deploying+a+CloudHub+Application) doc for more information. |

Now that its being managed, you can apply policies to secure this, define SLAs for your consumers and keep track of requests using API Analytics, same as what you did during the API labs (Labs 1-3).



If you want to test the proxy, the API console will be accessible at:  
http://<*Your Cloudhub Domain>*cloudhub.io/console, for example <http://order-entry.cloudhub.io/console/>

## 

## Summary

In this lab you demonstrated how you completed the following steps:

[Step 1: Call the Order Broadcast API](#_4m0toxgmfskv)

[Step 2: Deploy to Mule On-Premise through the management console](#_qo34hvwd5lv5)

[Step 3: Proxy your API in a Cloud-Based API Gateway](#_hn9gf0q9g9yh)

|  |
| --- |
| **NOTE on Deployment**  Throughout the different phases in the development life-cycle, like QA, pre-production or production, your application may need to be configured differently as server names, credentials and other similar parameters will vary.  As a developer facing this kind of variability, your goal is to produce a single Mule application for all your environments and to externalize all the environment-specific configuration parameters. This is the key to reproducible deployments.  You can configure your Mule application to facilitate deployment to one of many different environments, both on-premise and in the cloud. To do so, you must complete the following macro steps:   1. In your application, create a properties file for each environment. 2. Configure a property placeholder in your application to look for the deployment environment upon launch. 3. Configure an environment variable to point to a specific environment during application deployment.   Consider externalizing other aspects of your configuration, like time-out values, polling frequencies, etc... even if they don't vary between environments. This will facilitate tuning and experimenting as the whole Mule application would become configurable through a single properties file. See the [doc](http://www.mulesoft.org/documentation/display/current/Deploying+to+Multiple+Environments) for more info. |

**Congratulations! You have successfully defined, implemented, connected and managed your API using an API Led Connectivity approach with MuleSoft’s Anypoint Platform.**