# Lab 8: Orchestrate REST and JMS Endpoints

[Lab 8: Orchestrate REST and JMS Endpoints](#_86maahp7gk5r)

[Overview](#_hmlhk16nx5wk)

[Step 1: Configure the RAML Consumer](#_pxqjtxd84ch4)

[Step 2: Create DataMapper transformations](#_7p0upyjbgcnf)

[Step 3: Add JMS to the API](#_ni3afidknyyt)

[Step 4: Run the API](#_subkunlz4qxi)

[Step 5: Debug the API](#_kgzed3suxrq3)

[Summary](#_knuzle5uc8kd)

## Overview

Let us now orchestrate our back-end systems to fulfill the order. We will post the order to our internal sales order system and send US orders outbound to a messaging queue.

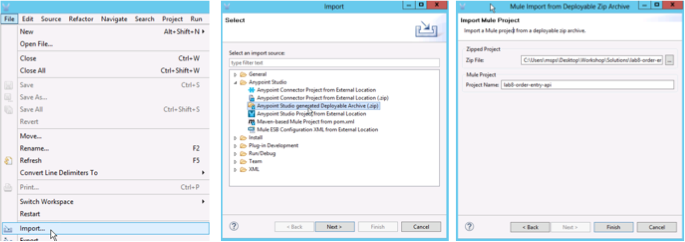


We will start implementing the Order Entry API’s **post:/orders** method by calling the post method of the **Internal Sales Order API** to process the order. This will persist the order information into a MySQL database and provide us with an order confirmation. We will use the same **RAML** consumer HTTP Configuration we defined previously.

We also have a JMS queue running on **ActiveMQ** that accepts orders coming from the United States. These orders will be processed by an external system that will be picking up messages from the JMS queue. Depending on the country, we will have a content-based routing rule. If the country equals to US, we will create a **JMS** message and send it to an **ActiveMQ** Queue with the content of the order creation confirmation. Otherwise, we will only log that the country was not US.

***Note:*** If you wish to skip this lab or need to catch up, you can open the completed solution by importing the following project:

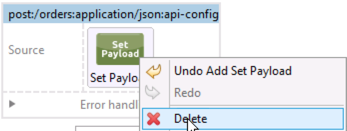
**C:\Users\msps\Desktop\Workshop\Solutions\lab8-order-entry-api.zip**



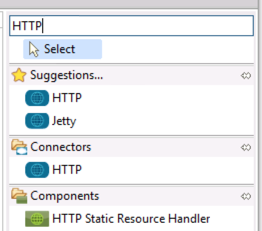
## Step 1: Configure the RAML Consumer

Let’s start implementing the **post:/orders** back-end flow by calling the **Internal Sales Order API.**

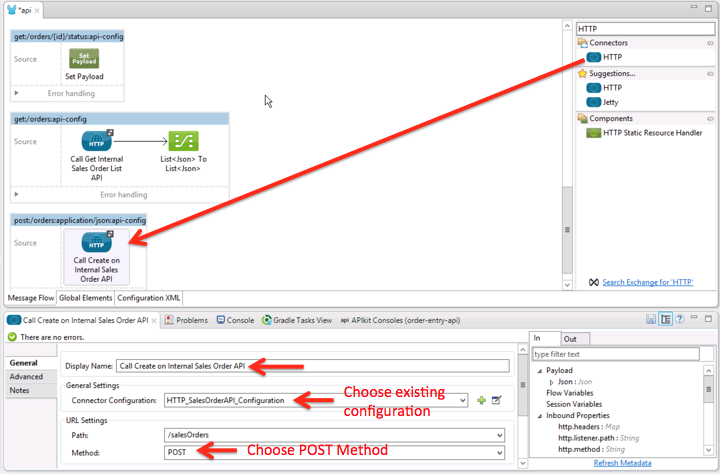
|  |
| --- |
| **NOTE**: Make sure you are in the **post:/orders:application/json:api-config flow** |



1. In Anypoint Studio, delete the **Set Payload** processor from the flow named **post:/orders:application/json:api-config**.



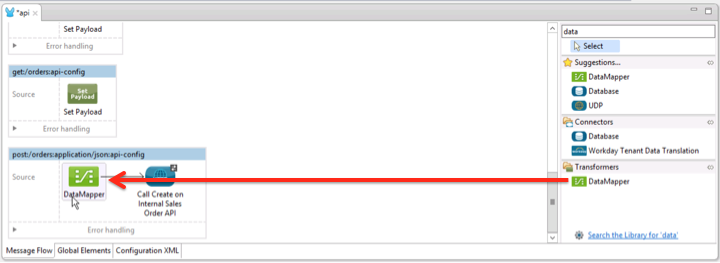
1. Search for HTTP in the component palette.



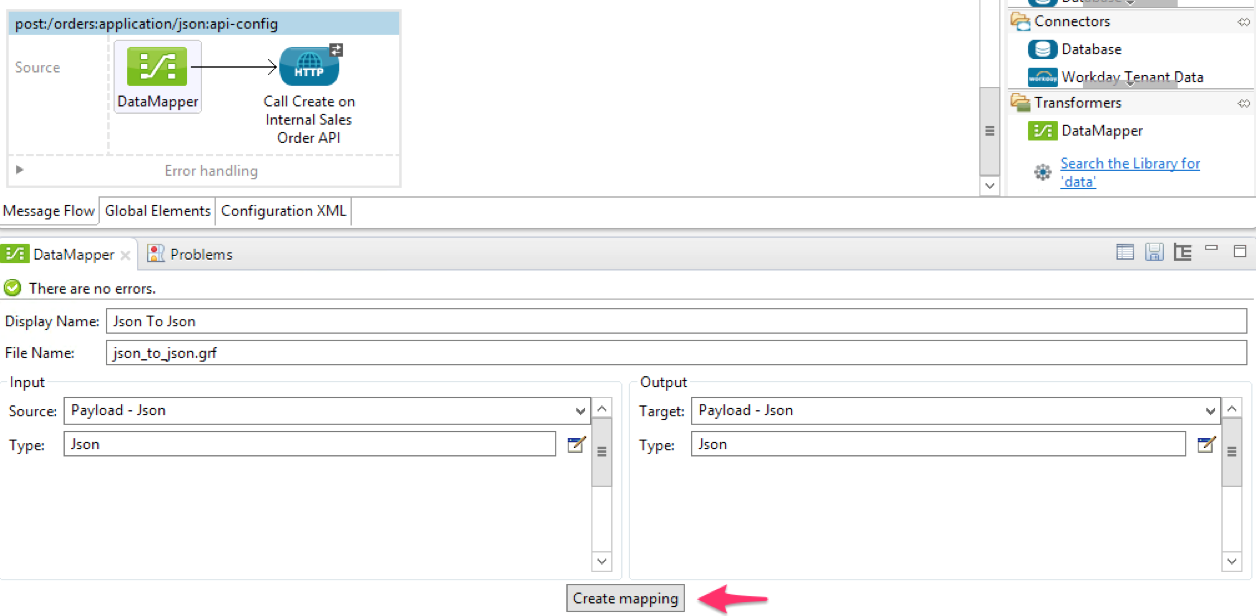
1. Click on the **HTTP Connector**
2. Drag it to the flow
3. Double-click on it to display it’s properties. You should see the component properties for you to configure in the **Mule Properties View**.
4. Add a meaningful name for the operation like Call Create on Internal Sales Order API in the **Display Name** field for the HTTP properties.
5. Let’s use the existing HTTP Configuration we created in the previous lab. Choose **HTTP\_SalesOrder\_API\_Configuration** from the Connector Configuration drop down list.
6. Now the **RAML** consumer populates the **Path** combo box with the resources defined in the API, in the this case: /**salesOrders**.
7. Click on the Method combo box
8. Select the POST method as shown.

## Step 2: Create DataMapper transformations

Now that we have configured the RAML consumer component, we need to create the necessary payload to call the API successfully. Since our incoming payload is not the same as the payload that the **Internal Sales Order API** needs, we need to make a transformation.

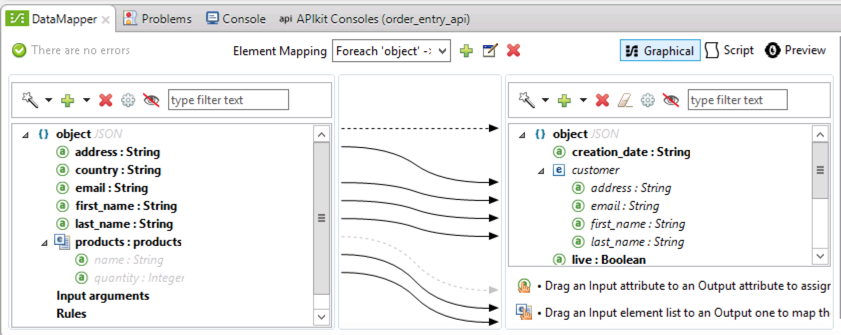


1. Search for the **DataMapper** component in the palette
2. Drag it right to the left of the RAML consumer processor
3. Double-click on it to display its properties.

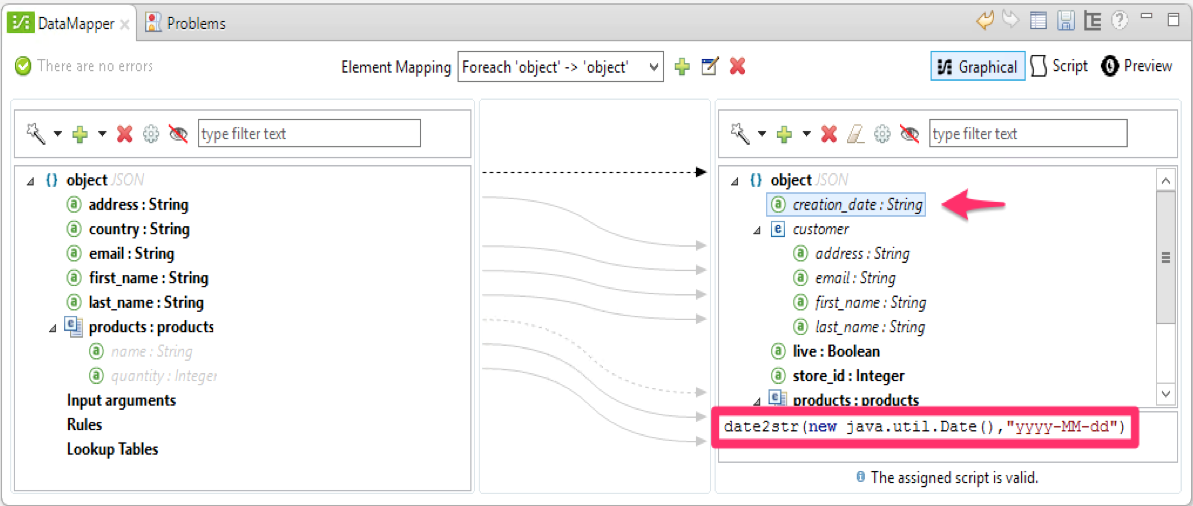


**DataMapper** knows the both the incoming payload format and the RAML consumer format because it DataSense read the metadata from both RAML definitions using **DataSense**.

1. Click on **Create mapping**.

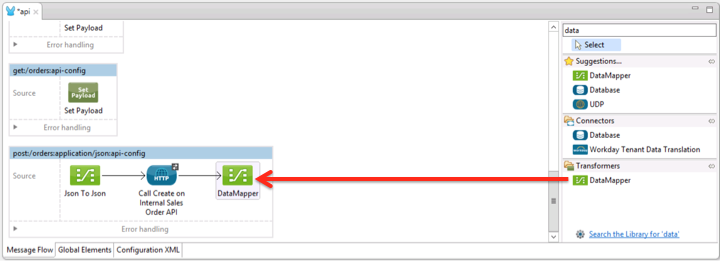


**DataMapper** has created a default mapping for us using property name similarities, which we can edit as we see fit. We can even set default values and call transformation functions. Since the Internal Sales Order API needs a **creation\_date** value and our API has not defined it, we can set the **creation\_date** to today using a default value.

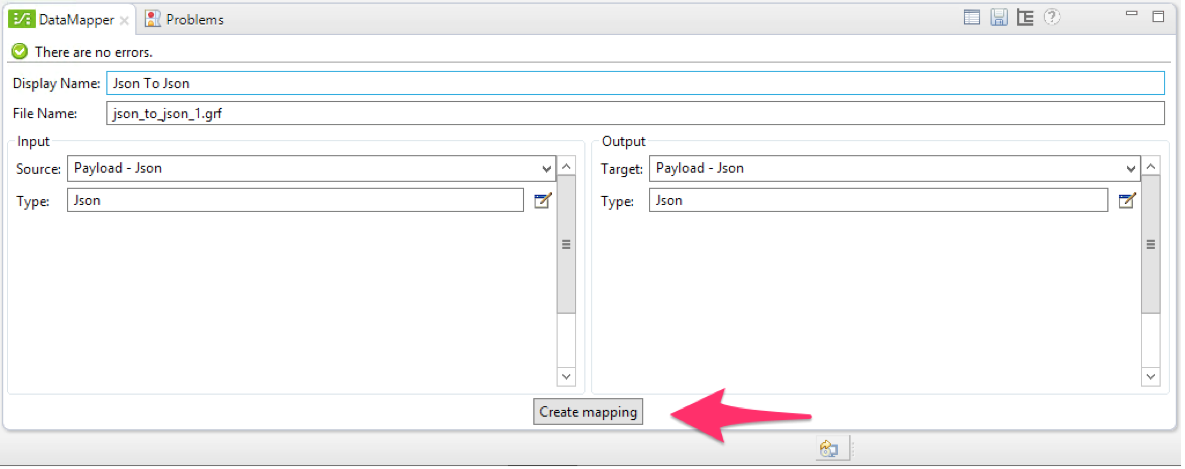


1. Click on the **creation\_date** property on the right pane
2. Add **date2str(new java.util.Date(),"yyyy-MM-dd")**as shown. Notice that we are creating a Date object and we are calling a transformation function called **date2str**to transform the date to a string value with a particular format.

We now need to transform the response coming out of the RAML consumer to the JSON object that we defined in our API.

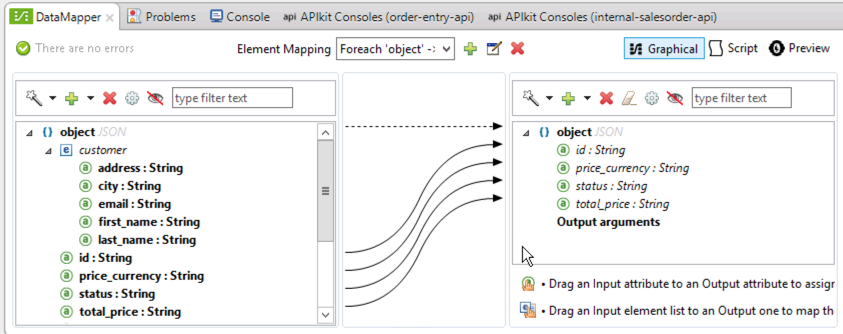


1. Drag another **DataMapper** component but this time drop it on the right side of the HTTP endpoint
2. Double-click on it to display its properties.



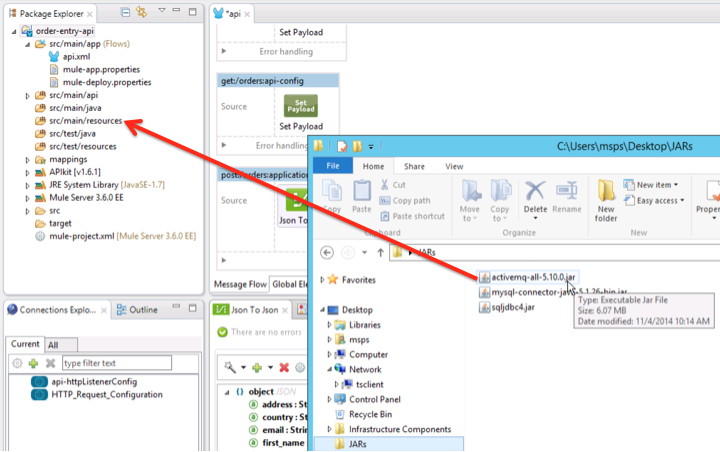
Once again, **DataMapper** knows the both formats.

1. Click on **Create mapping.** This time the automatic mapping created for us is sufficient so we don't need to do any additional mapping.

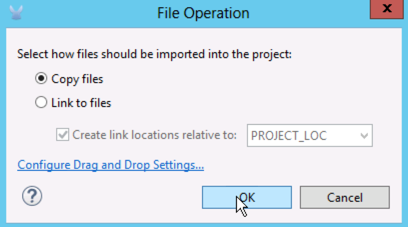
****

## Step 3: Add JMS to the API

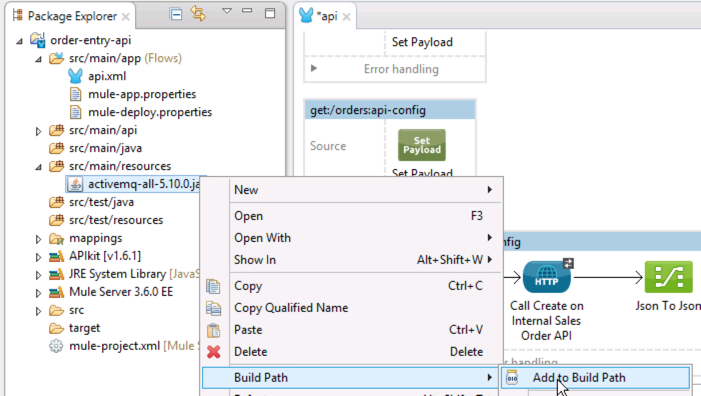
Before we start modifying the flow to add JMS support, we need to include the ActiveMQ jar library to our project.



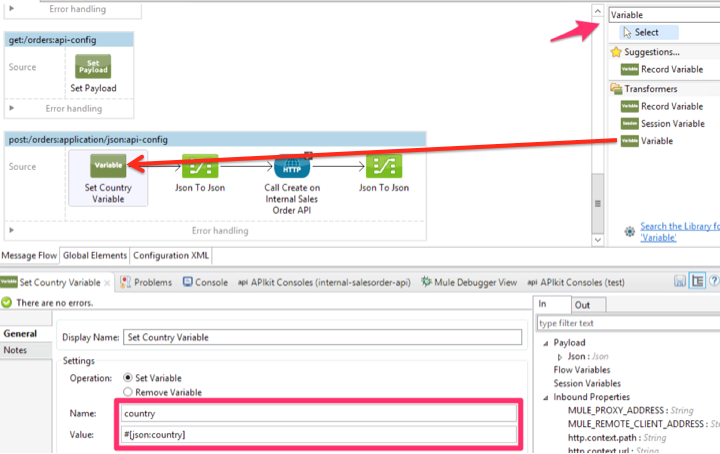
1. Copy the **ActiveMQ** jar file on the Desktop to the lib folder just by dragging it to the **src/main/resources** folder.
   1. The **ActiveMQ** jar file is located at C:\Users\msps\Desktop\JARs\activemq-all-5.10.0.jar



1. Select the option to Copy Files.



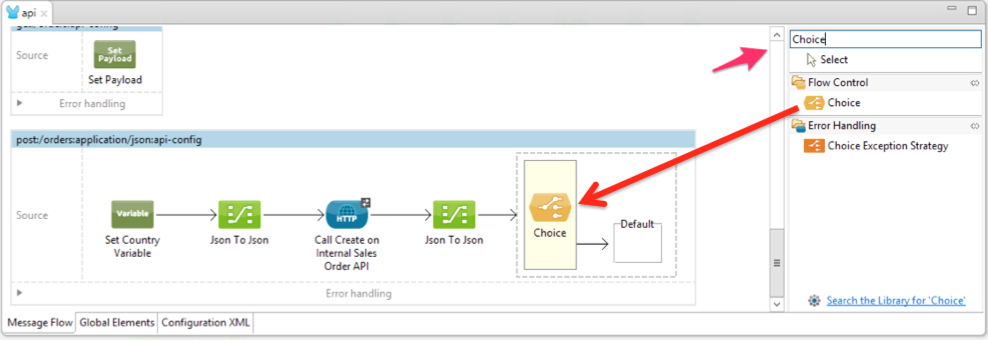
1. We need to add the jar file to the application classpath. Right click on the jar file
2. Select **Build Path**
3. Select **Add to Build Path**.



For our conditional routing, we need to define a variable that will contain the value of the country from the order. To do this, follow this steps as detailed in the screenshot:

1. Select the **Variable** **Component** and drag it at the beginning of the flow
2. Set the **Display Name** to Set Country Variable
3. Set **Name** to country
4. Set **Value** to #[json:country]

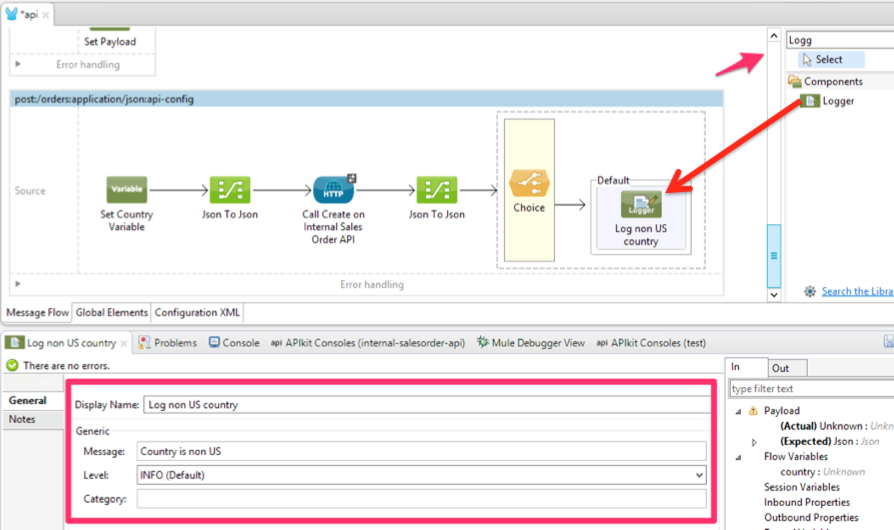
|  |
| --- |
| **NOTE**: There is no standard language currently for querying JSON data graphs in the same way XPATH can query XML documents. Mule provides a simple query syntax for working with JSON data in Java, called JsonPath.  The post:/orders flow receives the following JSON  {  "first\_name": "Brett",  "last\_name": "Roeder",  "email": "brettkroeder@mulesoft.com",  "country": "US",  "address": "350 5th Ave, Atlanta",  "products": [  {  "name": "S21",  "quantity": 1  }  ]  }  Given the above JSON, #[json:country]returns US  The importance of JsonPath is you do NOT need to convert the JSON to XML, a POJO, or some other format in order to work with it.  Check out the [JSON Module Reference](http://www.mulesoft.org/documentation/display/current/JSON+Module+Reference) for more information. |



Now that we have the **country variable**, lets add the routing part of the flow.

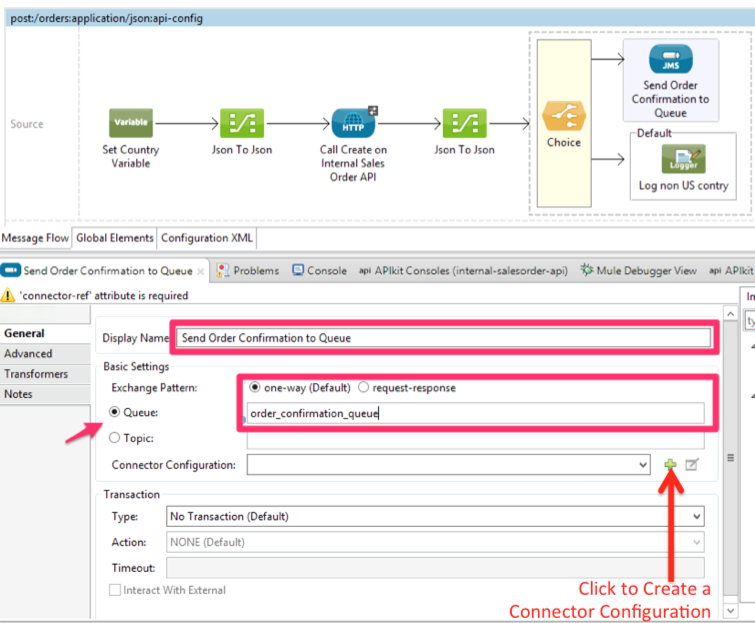
1. Select the Choice component
2. Drag it after the **DataMapper** at the right.

We will add the choice expression later after we define in the 2 conditional paths - one path for US orders, and another path for Non-US orders.



Now add a **Logger** for the non-US orders scenario which will be the Default path.

1. Select a **Logger** component
2. Drag it inside the **Default** scope in the **Choice** component.

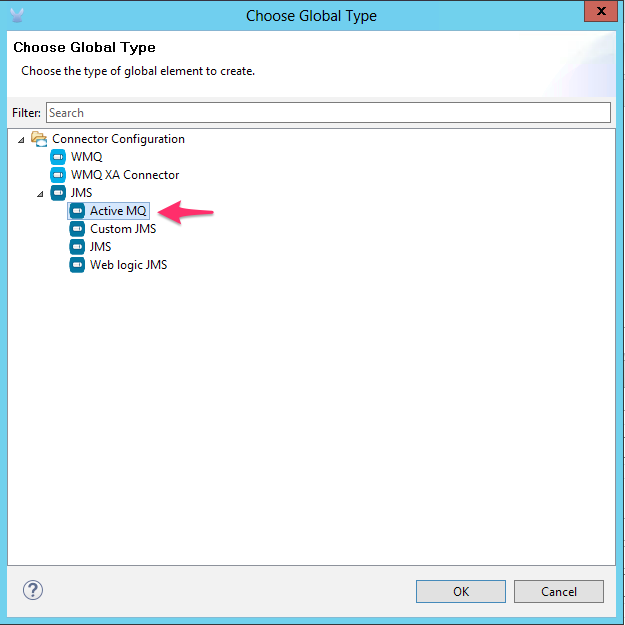


Now lets add the US route in our choice.

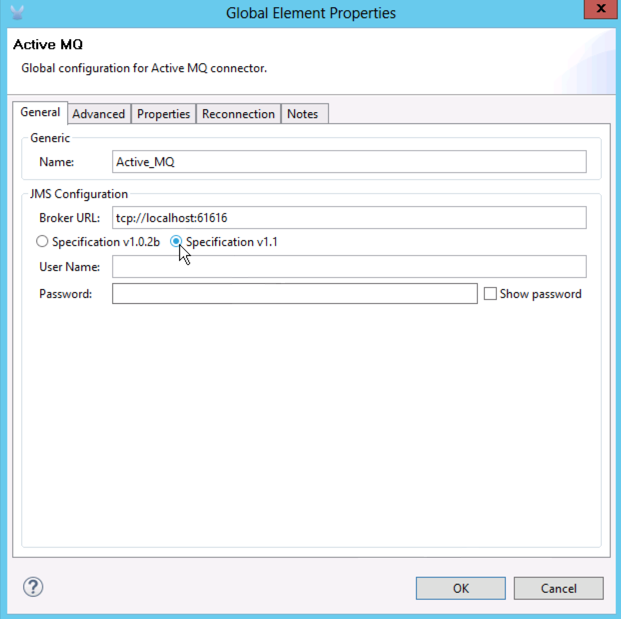
1. Search for **JMS** in the component palette,
2. Drag it inside the **Choice** component
3. Double-click on it to display its properties
4. Add a meaningful name like Send Order Confirmation to Queue.
5. In the **Basic Settings** section, click on **Queue**
6. Set the queue name to order\_confirmation\_queue. This is the name of queue that we will send US orders to.

We also need to create a **Connector Configuration** to configure the **ActiveMQ** server information.

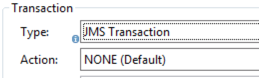
1. Click on **+** (as shown in the screenshot above).



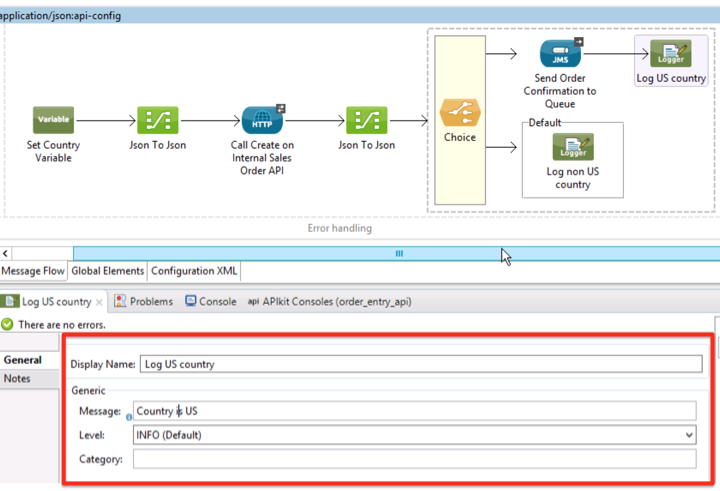
1. Choose **ActiveMQ**
2. Click OK



1. Choose the JMS Specification v1.1
2. Click **OK**.

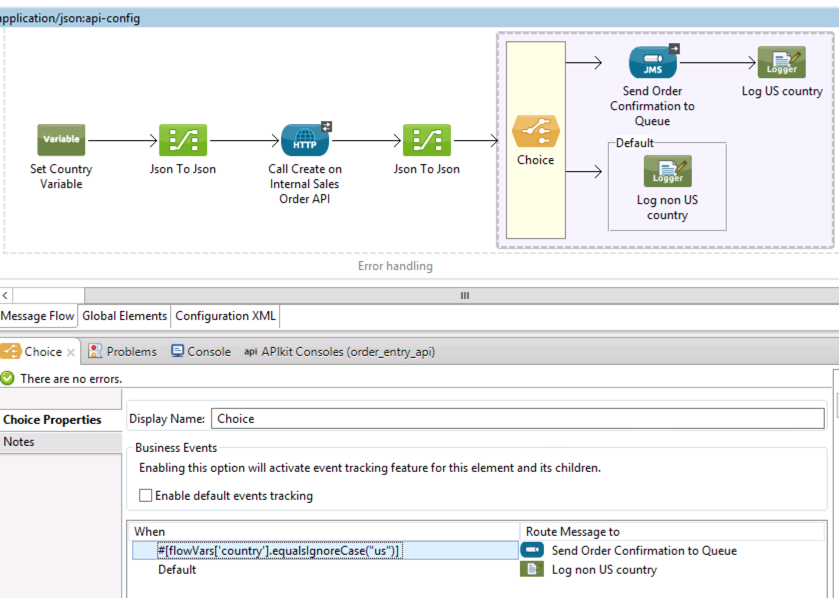


1. Finally, in **Transaction Settings** set the Type to **JMS Transaction**
2. Set the **Action** to **NONE**.



Lets also add a Logger in this route the same way we did it for the non US route.

1. Choose the Logger component
2. Drag it to the right of the **JMS** component
3. Configure the properties as shown above



At this point we have both routes in our choice configured but we have not yet configured the routing condition.

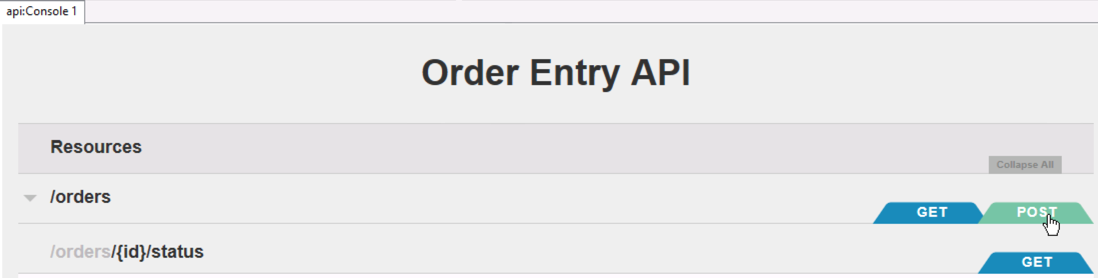
1. Double-click on the **Choice** component to open the **Properties** View. We will see that we have 2 choices, the JMS choice and the default one.
2. For the JMS choice, add an expression that will check if the incoming country is **US**. The expression we will use is the following [MEL](http://www.mulesoft.org/documentation/display/current/Mule+Expression+Language+MEL) ([Mule Expression Language](http://www.mulesoft.org/documentation/display/current/Mule+Expression+Language+MEL)) expression: #[flowVars['country'].equalsIgnoreCase("us")]

|  |
| --- |
| DOUBLE CHECK: Make sure there are no trailing spaces after the ] |

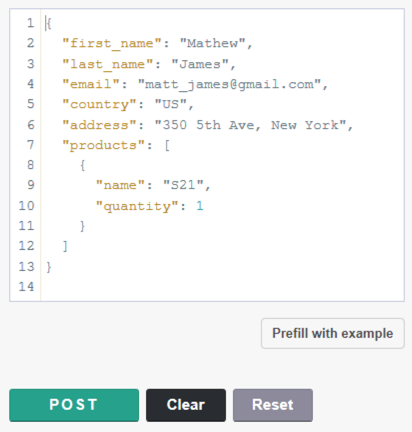
## Step 4: Run the API

Now it's time to run the API in **Anypoint Studio** to see how it all works together.

1. Start the application as done in the previous Lab. (Right-click the application, **Debug as > Mule Application** or simply click the Debug icon and choose **order\_entry\_api**:



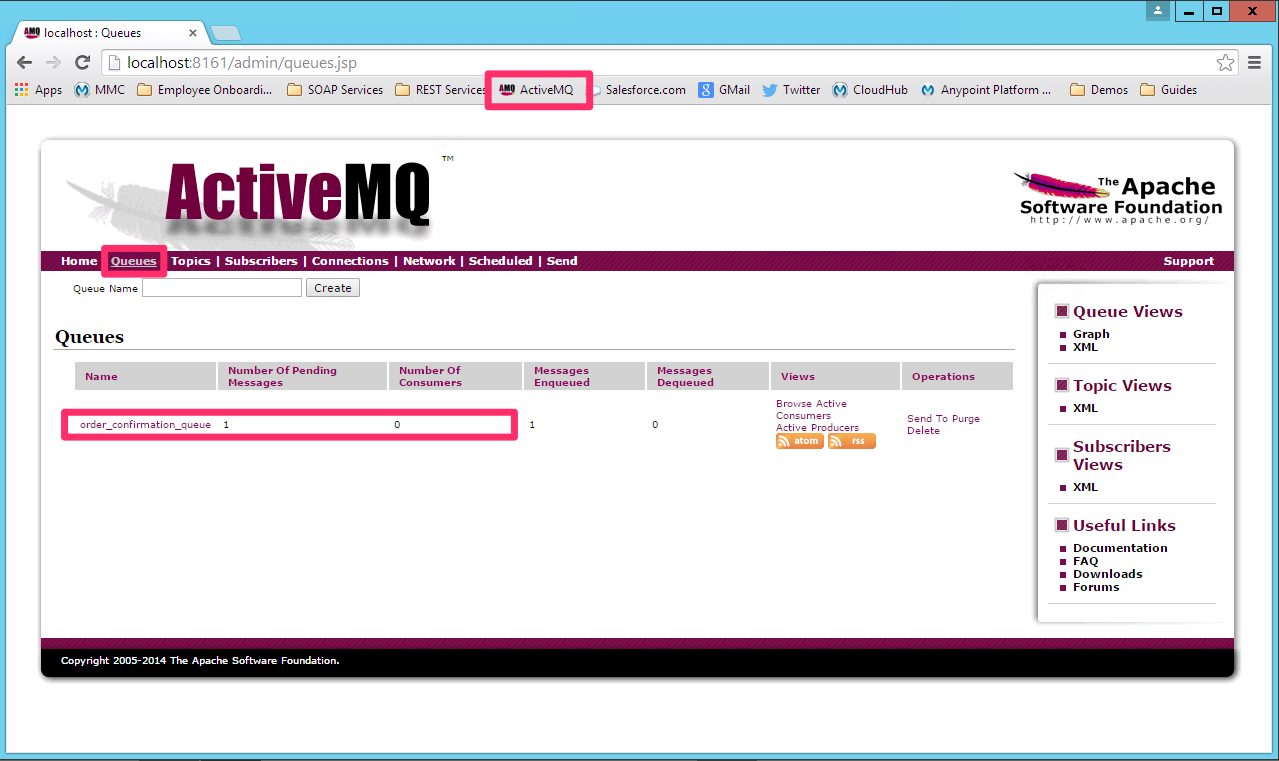
1. Once the **API Console View** opens in Anypoint Studio, click on the **POST** button on the /**orders** resource..



1. Under **Try it**, view the request body then click on the **POST** button below.



1. Once the request is served, you should see a **200** response code and the JSON response of our API.
2. Since this is US order and US orders are also sent to the JMS queue lets see it. In order to see the JMS message we just created, we will use **ActiveMQ Administration Console**.

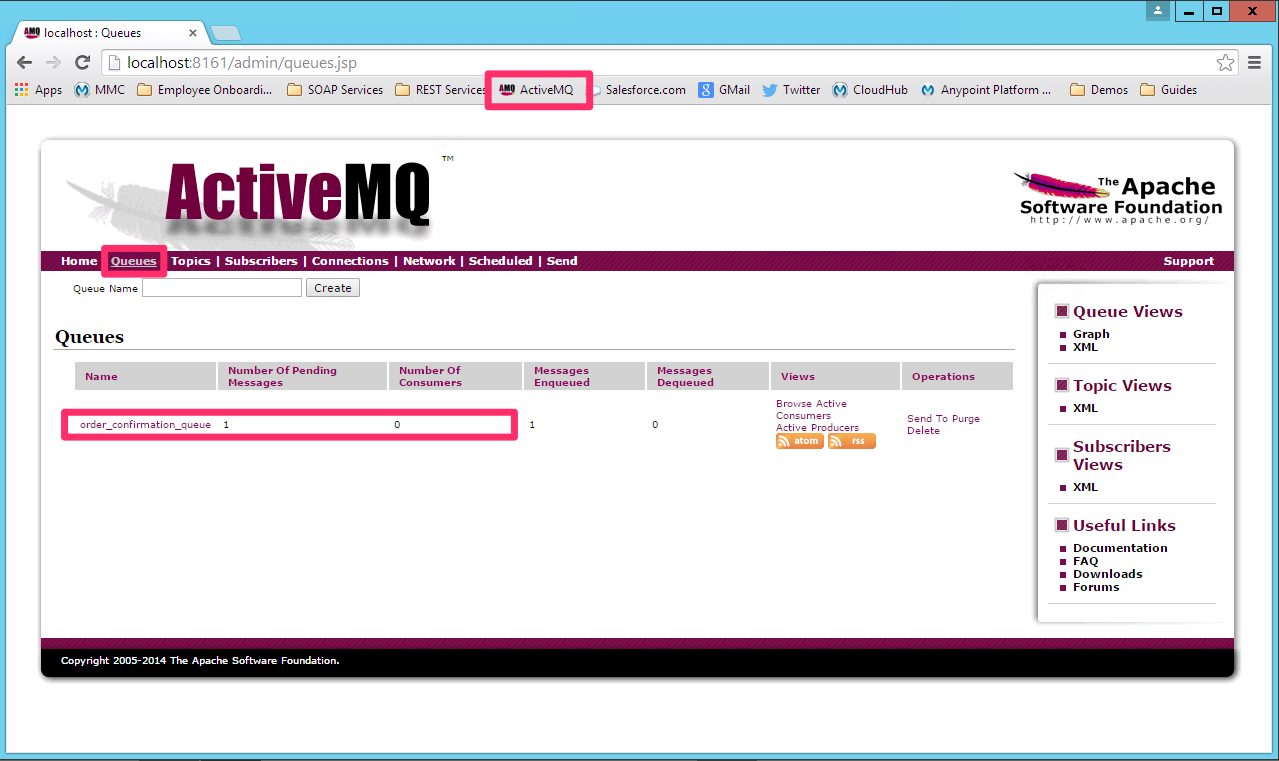


1. Open Google Chrome
2. Click on the **ActiveMQ** bookmark or by typing the following URL in Chrome:[**http://localhost:8161/admin/**](http://localhost:8161/admin/)

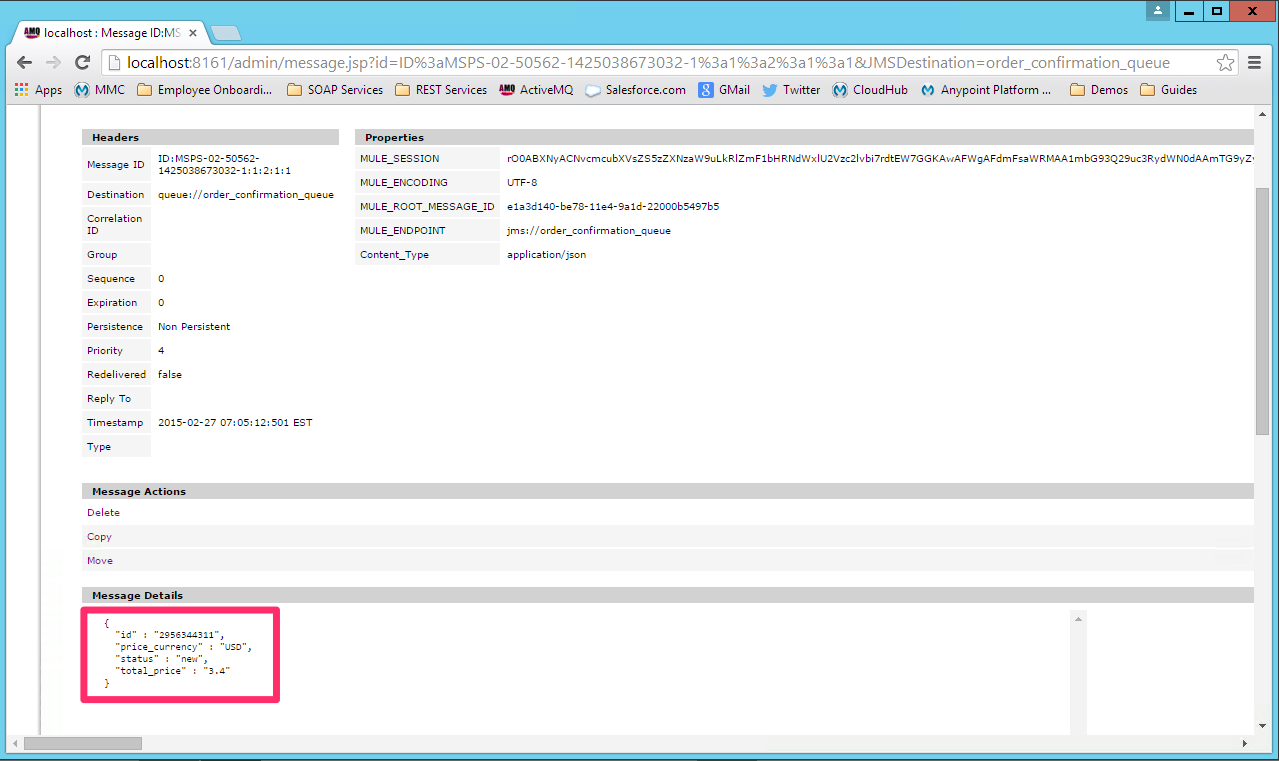
The username is **admin**

The password is **admin**.

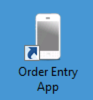
1. Click on the **Queues** link in the menu.You will a list of all the Queues created in **ActiveMQ** and you should also see a queue called **order\_confirmation\_queue** with active 1 message like.



1. Click on the **order\_confirmation\_queue**, you will be prompted a list of all the messages inside the queue. At this moment there should only be one message per order we created.



1. Click on any of the messages to view its detail.
2. Scroll down to see the order confirmation JSON message.
3. Additionally, we can check the logs in Studio to see the output of the Logger components we added in our flow.
4. Now change the country in the order to any country you like other than US and create the order. If you go back to the **ActiveMQ** console view, you will see that now there is ***NOT*** a new **JMS** message for that order.

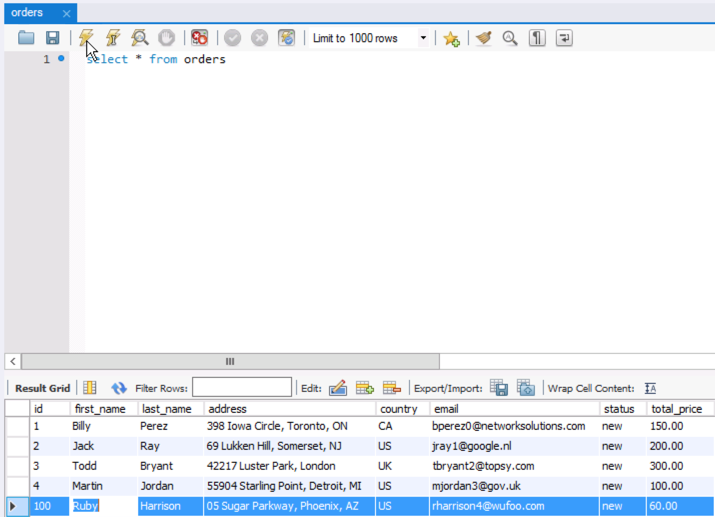


1. Test the app from the Mobile Order Entry Application as done is the previous lab. After creating a successful order, the Internal Sales Order API processes the order and persists it in our Order Entry database.



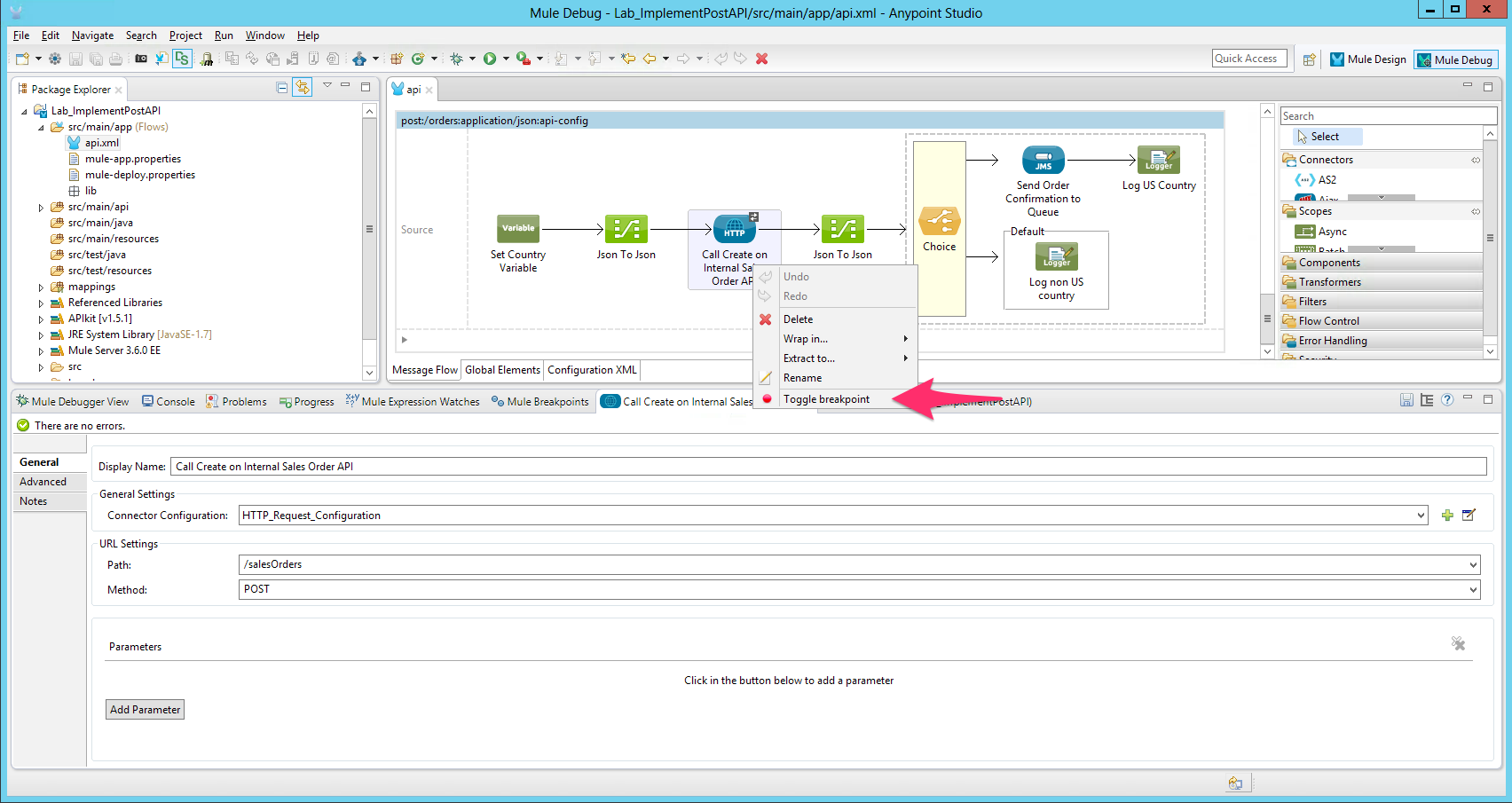
1. Optionally, click the MySQL Workbench

|  |
| --- |
| **NOTE**: If you cannot log in to the database because of a password vault error, you can clear the vault and enter username: **root** and password: **Mule1379** |



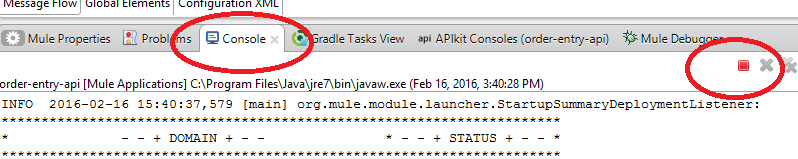
1. View the orders by executing **select \* from orders.**

## Step 5: Debug the API

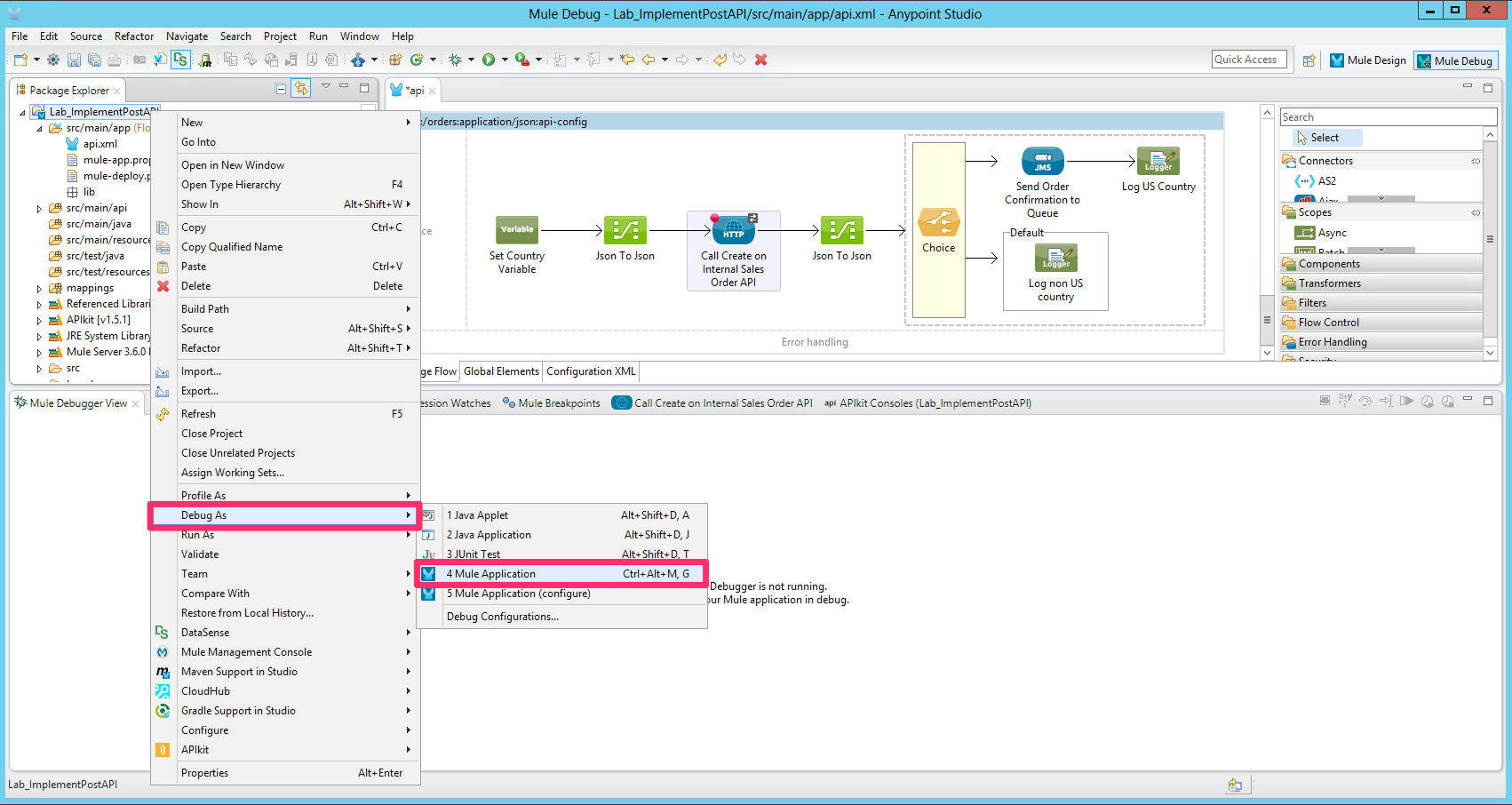


We need to set up the breakpoint in our flow.

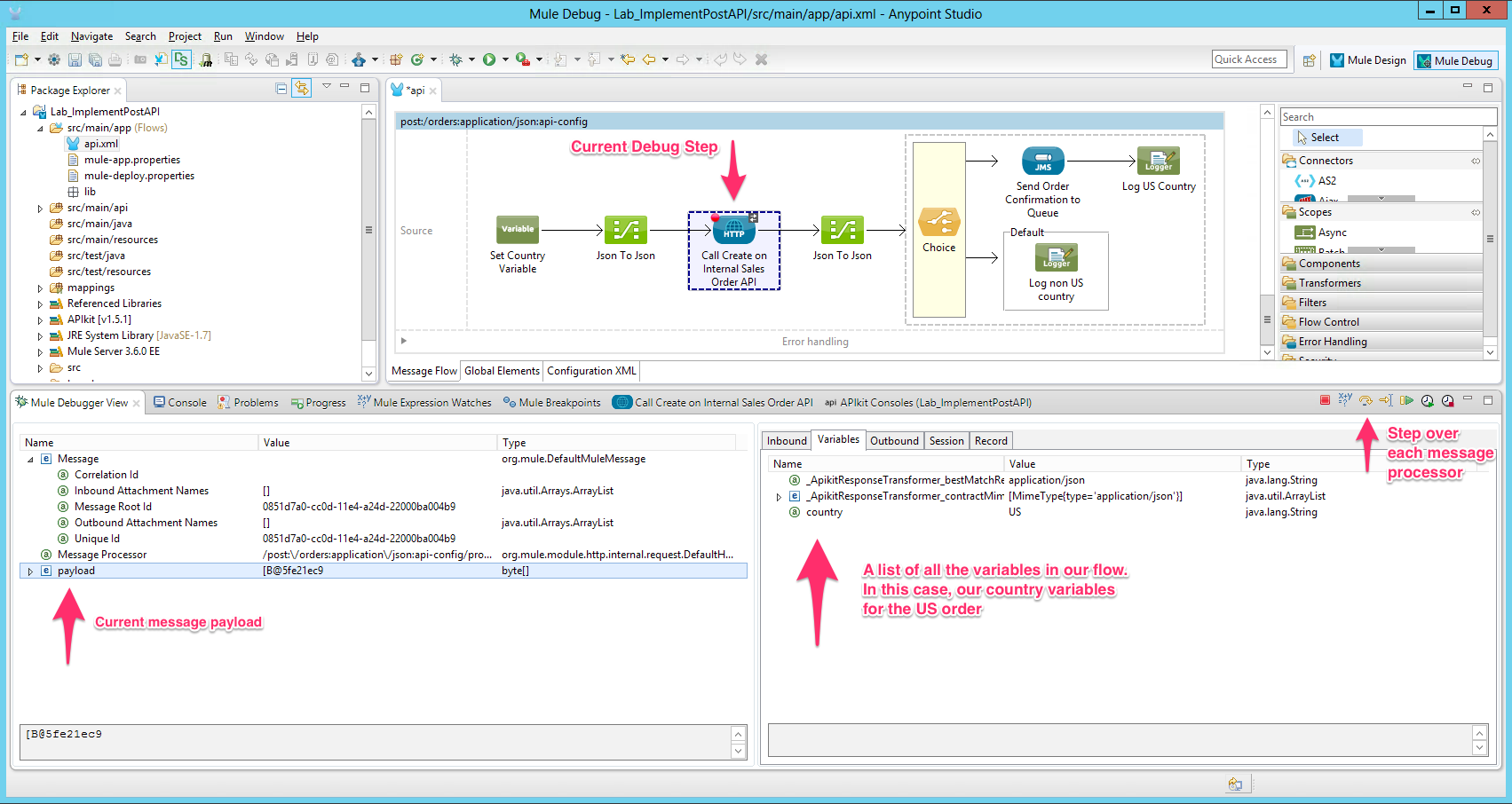
1. Right click above the HTTP Component (Call create on Internal Sales Order API)
2. Click on the **Toggle breakpoint** option from the menu.



1. If the API is running, stop it by clicking on the red square from the console tab. If the square is grey, it means it is already stopped.



1. Now lets start the API in Debug Mode. In order to restart the project in debug mode right click on the project name **>** **Debug As** **>** **Mule Application**.
2. Once the application is running, create a new order. You will see that application now its not responding. That's because the application has been stopped in the breakpoint we created.



1. Now, in the **Mule Debug View**, we can see a lot of useful information about our API at runtime:
   1. We can see the value for our message payload
   2. We can see the content for all the variables we created in our flow. Particularly, our **country** variable.
   3. We can also **Step Over** on each step of our flow and the view will refresh automatically with the new content of our flow.
   4. You can set [**Mule Expression Watches**](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-UsingMuleExpressionWatches)to test out script references and “play” with the payload.

|  |
| --- |
| NOTE: Check the [Studio Visual Debugger](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger) doc for more information   * [Using the Visual Debugger](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-UsingtheVisualDebugger) * [Evaluating Mule Expressions](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-EvaluatingMuleExpressions) * [Setting Breakpoint Parameters](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-SettingBreakpointParameters) * [Using Mule Expression Watches](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-UsingMuleExpressionWatches) * [Debugging Remotely](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-DebuggingRemotely) * [Go Further](http://www.mulesoft.org/documentation/display/current/Studio+Visual+Debugger#StudioVisualDebugger-GoFurther) |

## Summary

In this lab, you completed the following steps:

[Step 1: Configure the RAML Consumer](#_pxqjtxd84ch4)

[Step 2: Create DataMapper transformations](#_7p0upyjbgcnf)

[Step 3: Add JMS to the API](#_ni3afidknyyt)

[Step 4: Run the API](#_subkunlz4qxi)

[Step 5: Debug the API](#_kgzed3suxrq3)

These steps allows you to easily consume internal and external API's, allowing us to quickly create aconnectivity layer on top of System and Process APIs. In this case, we used the HTTP Connector to consume a REST RAML based API and transformed the data using Data Mapper. Additionally we added support for external systems to consumer events from our API using JMS and ActiveMQ. For this API we used a queue but we could change it to a topic and have a pub/sub model with just 1 click.

Click the links for more information on:

* [HTTP Request Connector](http://www.mulesoft.org/documentation/display/current/HTTP+Request+Connector) with the RAML Consumer
* [Data Mapper](http://www.mulesoft.org/documentation/display/current/Datamapper+User+Guide+and+Reference)
* [JMS Transport Reference](http://www.mulesoft.org/documentation/display/current/JMS+Transport+Reference)

Congratulations! You have finished Lab 8.

Please update the spreadsheet indicating you have completed Lab 8.