# Lab 10: Connect to SaaS and Social Networks

[Lab 10: Connect to SaaS and Social Networks](#_unhd2nhxtatb)

[Overview](#_ot1mfjw9473q)

[Step 1: Import a project from Anypoint Exchange](#_ln2lzaa9urmt)

[Step 2: Add VM connectors for asynchronous messaging](#_e2ymm5td51dr)

[Step 3: Add the Salesforce Connector](#_malpqon5sogh)

[Step 4: Run the Application](#_9rgrsx1iut0)

[Step 5: Verify results](#_p8dw2ui3q004)

[Step 6: Deploy to CloudHub from Anypoint Studio](#_9unton446aog)

[Step 6a: Automated Deployment to CloudHub](#_t5n5x8h1wniw)

[Step 6m: Manually Deploy to CloudHub](#_363h8mi6833r)

[Step 7: Configure CloudHub App properties](#_k38rbei8kmf2)

[Step 8: Configure the Application Properties](#_mvpvgqkt6tzs)

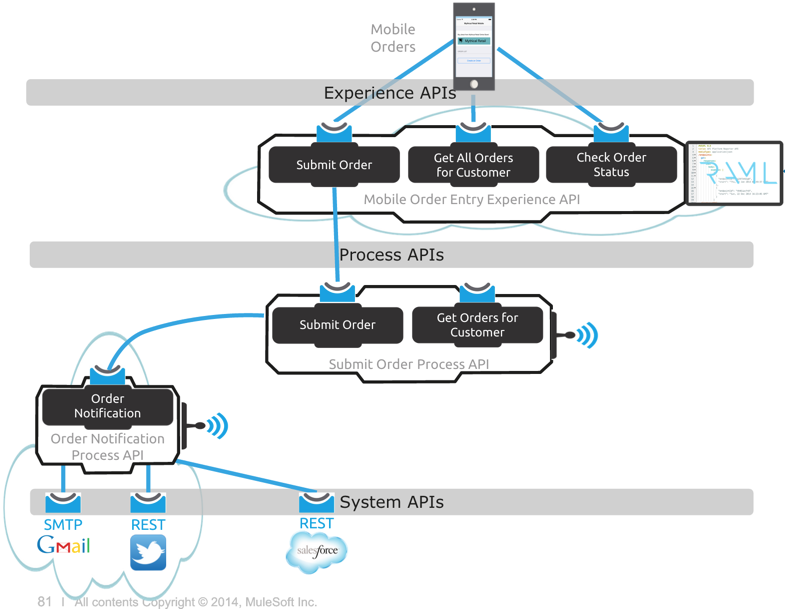
[Step 9: Test API on CloudHub](#_392gajid8vok)

[Summary](#_irq5eyedoeiu)

# Overview

We have successfully implemented our APIs for submitting,retrieving orders and checking order status. These are internal APIs that we will deploy to our on-premise environment.

We will now implement an API for broadcasting the order to a variety of Cloud-based applications using **Anypoint Connectors**. We will send the contact information to our SaaS CRM - **Salesforce.com**, we will send an email out using our **Google Mail** SMTP account and we will tweet about the order using **Twitter**.

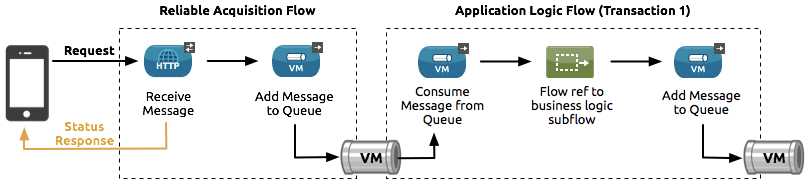


Because all the *Center of Gravity* of these applications is the cloud, it makes sense to deploy our API to the cloud as well using **CloudHub** - MuleSoft’s Integration-Platform-as-a-Service offering. Here we will see how the Anypoint Platform enables **true hybrid integration**, by doing orchestrations on the cloud with CloudHub, and hooking these in to the previous APIs/integrations we deployed on-premise with Mule ESB.

Instead of building this from the ground up, we will make use of the **Anypoint Exchange** for re-using an existing order notification project template. The template already has the logic to send emails using the GMail SMTP service and broadcast to social networking sites, Twitter in this example. We will download and import this template in Anypoint Studio and add the necessary customizations to store the customer information into our **Salesforce.com** contact list.

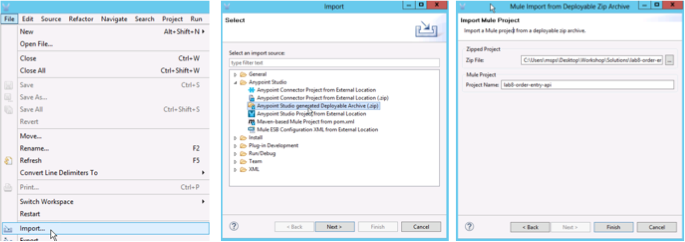
To make our project more resilient, we will utilize Mule’s **VM connectors** to call another flow through asynchronous messaging. We will add the logic in the existing order broadcast flow to send a message to a VM queue that will start another flow. The other flow will create a new Contact in Salesforce using the Salesforce.com Connector. This allows us to guarantee the delivery of the customer information to Salesforce by reliably storing the data in the VM queue first, so that in case there are failures in sending the data to Salesforce, the message can be recovered and retried.

<http://www.mulesoft.org/documentation/display/current/Reliability+Patterns>



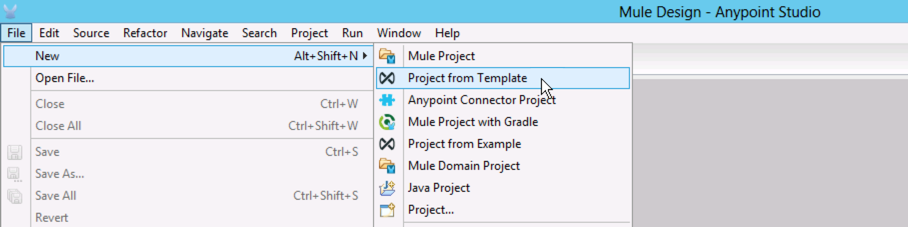
Let’s start implementing this.

|  |
| --- |
| **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\lab10-order-broadcast-api.zip |

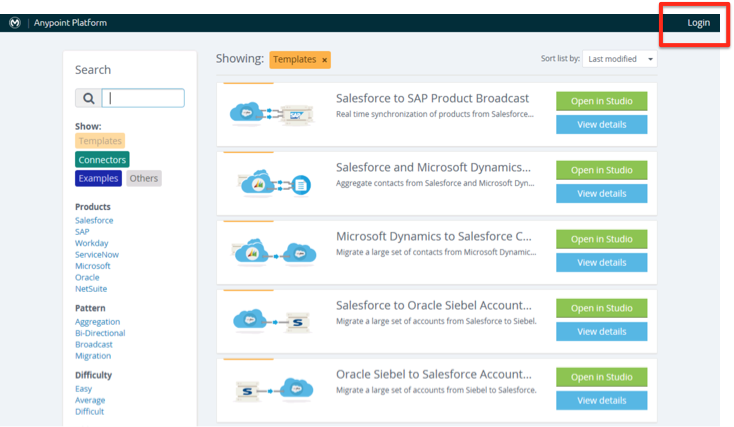


## Step 1: Import a project from Anypoint Exchange

Our first step is to create a new project based on the *Order Broadcast project template* located in **Anypoint Exchange**.

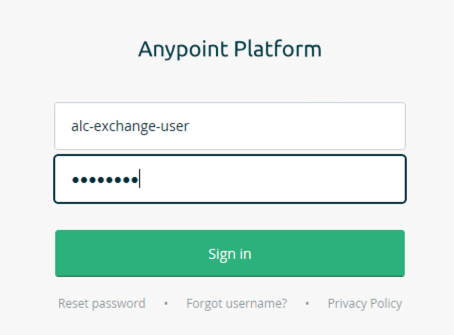


1. We can directly download a project template from Anypoint Exchange in Studio by doing [**File**] > [**New**] > [**Project From template**] as shown.

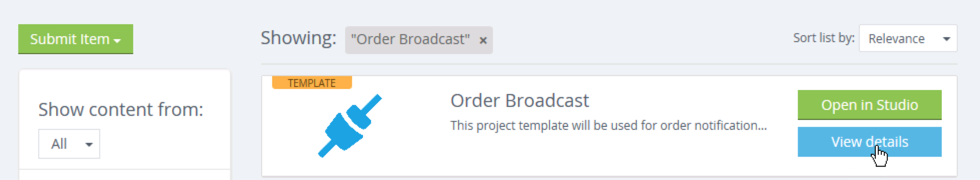


Here you will see the public Anypoint Exchange with all the available templates, connectors, and examples for you to download and use. We will be using an existing Order Broadcast template shared to us in a private library.

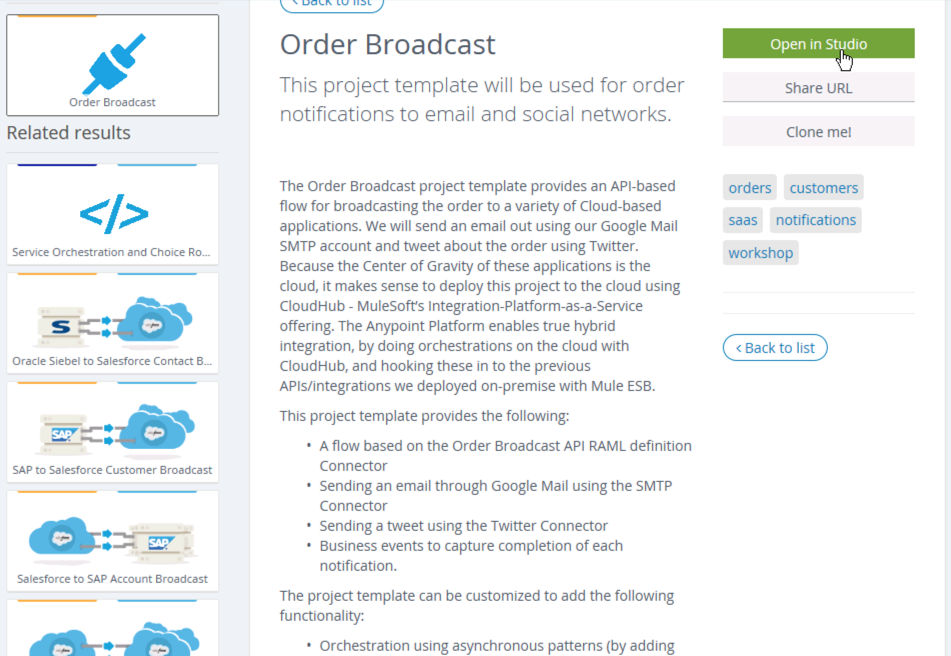
1. Click the **Login** link at the top-right corner.



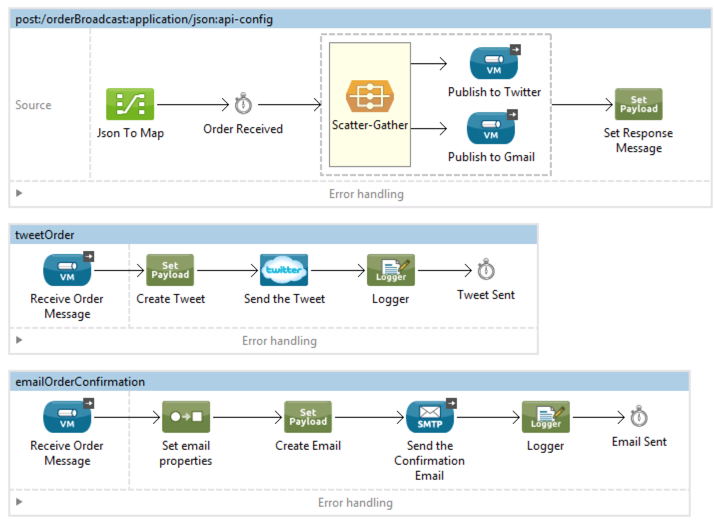
1. Login using the following credentials:
   1. **Username**: alc-exchange-user
   2. **Password**: Mule1379



1. You should see a Template called **Order Broadcast.**
2. If you don’t see the Order Broadcast template,
   1. type Order Broadcast on the Search box on the right.
   2. Click on **View Details .**

****

1. Review the details about the Order Broadcast template. You’ll see the following:.
   1. Title, summary and description of the template.
   2. Versions of the template you can download
2. Click on **Open in Studio** to download the template.

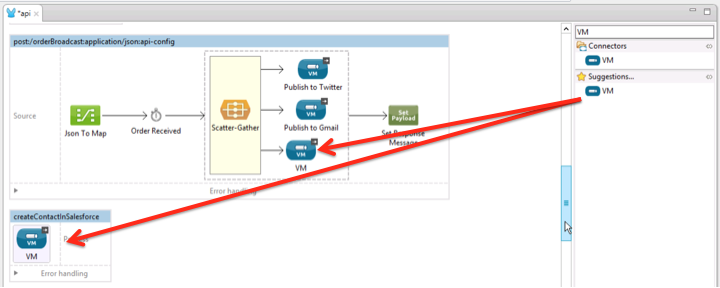


You’ll see a project with a flow that broadcasts to 2 sub-flows using VM connectors. The sub flows concurrently send messages to Twitter and GMail. We will add another flow to send the customer information to Salesforce. (Optional: Try running it.)

## 

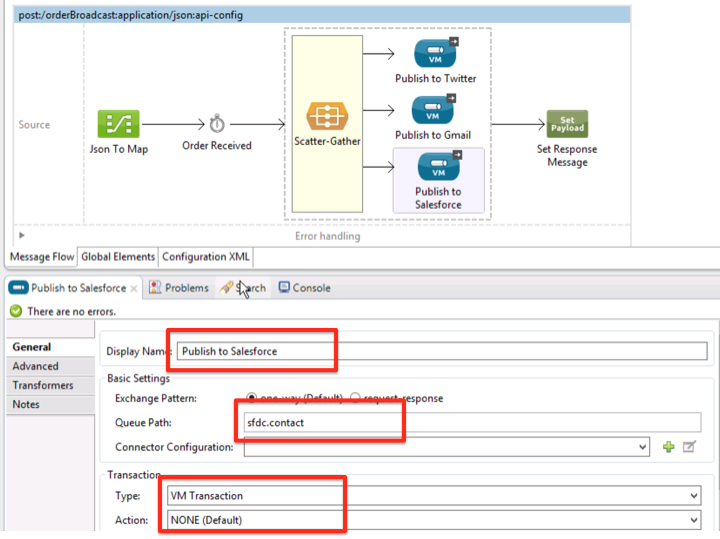
## 

## Step 2: Add VM connectors for asynchronous messaging

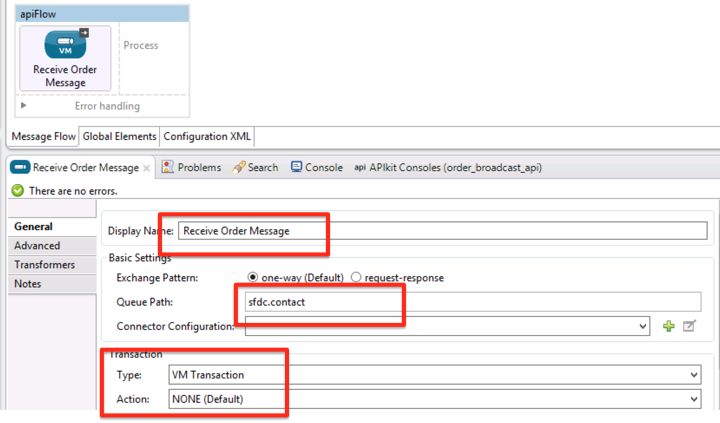


The first flow uses Scatter-Gather to execute routes concurrently instead of sequentially. Parallel execution of routes can greatly increase the efficiency of your application, in this case we are publishing a message to VM queues to call Twitter and Gmail, we will add another parallel path for Salesforce. Please see this link for more information on Scatter-Gather <http://www.mulesoft.org/documentation/display/current/Scatter-Gather>

1. Search forthe **VM** connectors in the toolbox.
2. Drag & drop a VM connector inside the **Scatter-Gather**
3. Drag and drop another VM connector under the main flow to create a new flow.
   1. This will trigger the new flow when a message arrives from the VM Connector we added to the first flow.



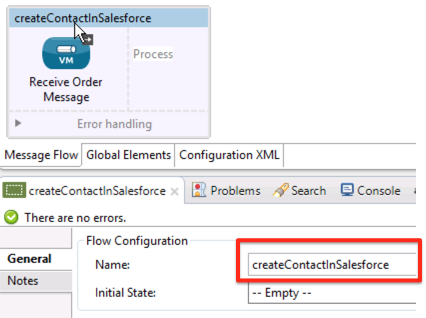
1. Click the VM connector ***inside the Scatter-Gather***
2. Change the **Display Name** to **Publish to Salesforce.**
3. Set the **Queue Path** to **sfdc.contact**.   
   **NOTE**: This will be the name of the VM queue where the message will be sent.
4. Set the **Transaction Type** to **VM Transaction**
5. Set the **Action** to **NONE.**



1. Click the VM Connector ***inside the second flow***
2. Change the display name to **Receive Order Message**.
3. Set Queue Path with **sfdc.contact**. **Note**: This is the same name as the internal queue we will be sending the message to, meaning this flow will be triggered when the previous flow sends a message.
4. Set the Transaction Type to **VM Transaction**
5. Set the Action to **NONE.**
6. Click **Save All**. 

## Step 3: Add the Salesforce Connector

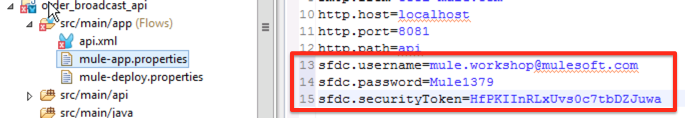
You will now implement the flow to add a new contact in Salesforce.



1. Rename the second flow to **createContactInSalesforce.**
2. Click the flow header
3. Change the name in the Flow Configuration.

****

1. Drag & drop the components you see above to implement the **createContactInSalesforce** flow
   * Data Mapper
   * Salesforce Connector (If asked to select a version, choose **Use Newest**)
   * Set Payload
   * Logger



Now add the **Salesforce** credentials to the project.

1. Open the mule-app.properties file to add the Salesforce credentials to the project’s properties file.
2. Copy and paste the following into the mule-app.properties file:

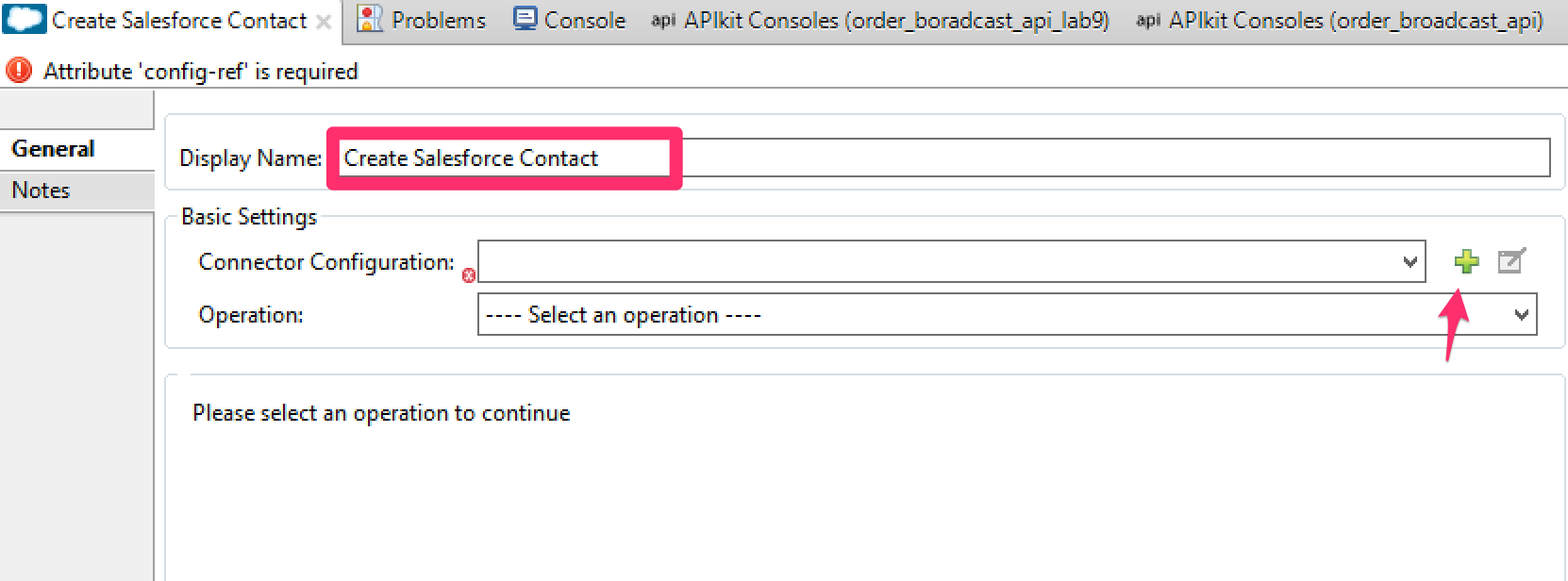
sfdc.username=[mule.workshop@mulesoft.com](mailto:mule.workshop@mulesoft.com)

sfdc.password=Mule1379

sfdc.securityToken=IwM3T9IMYRMKhfF9aGBoCBp7y

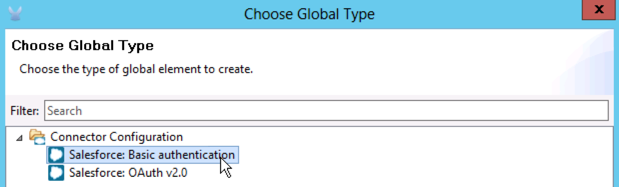
1. Save the mule-app.properties.

**Make sure you SAVE before proceeding!**

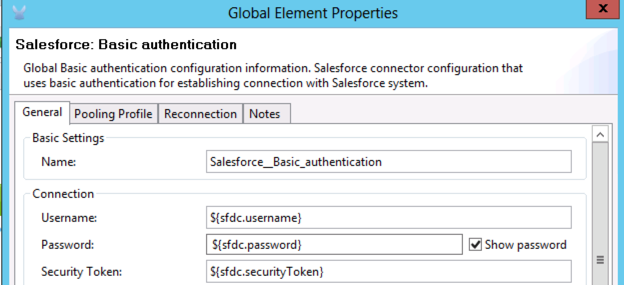


Configure the Salesforce Connector

1. Go back to the flow andclick the Salesforce Component
2. Set the *Display Name* to **Create Salesforce Contact**
3. Click the  icon by the *Connector Configuration* field to create a new **Global Salesforce Connector Configuration**.



1. Choose **Salesforce: Basic authentication**
2. Click **OK**.



1. Link the values in the mule-app.properties to the Salesforce config by entering the Spring properties for the credentials as shown above.
   * Username: ${sfdc.username}
   * Password: ${sfdc.password}
   * Security Token: ${sfdc.securityToken}

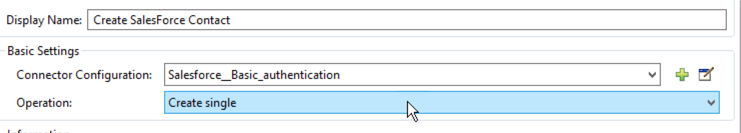
|  |
| --- |
| Optionally, you can create your own Salesforce.com Developer account. Please follow the [*Salesforce Developer Account Setup*](http://www.mulesoft.org/documentation/display/current/Salesforce+Developer+Account+Setup) instructions. |



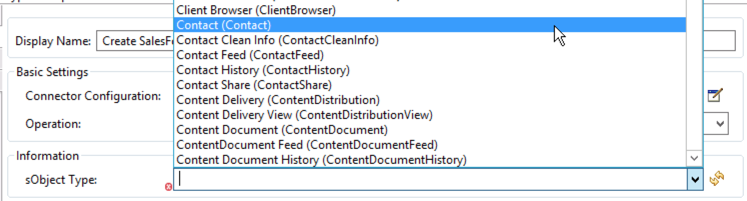
1. Click **Test Connection…**

|  |  |
| --- | --- |
| If connection test is **successful** | If you get an **error** |
|  |  |
| 1. Click **OK** 2. Click **Save All**. | 1. Go back to the Salesforce configuration and remove the **Security Token** setting |

Make sure you see **Test Connection successful** before proceeding!!!



1. Go back to the Salesforce connector in the flow
2. Click on the **Operation**
3. Select **Create Single**.

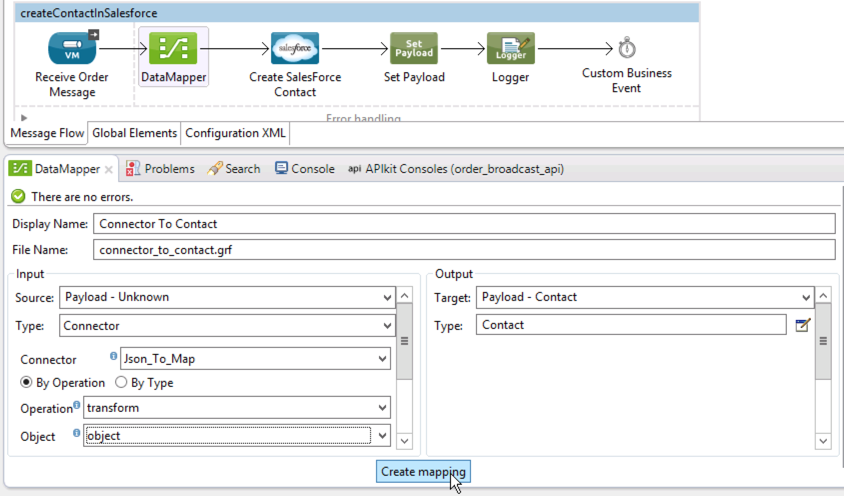


This will query the various objects from Salesforce and display it in the sObject Type list. These are the various Salesforce objects that you can use, including custom objects.

1. Choose the **Contact** sObject type.

|  |
| --- |
| **DOUBLE CHECK:** Make sure you choose **Contact** and not **Cont*r*act**! |

|  |
| --- |
| **TRIPLE CHECK:** Are you sure you selected **Contact** and not **Cont*r*act**! |



1. Click on the **DataMapper** transformer.

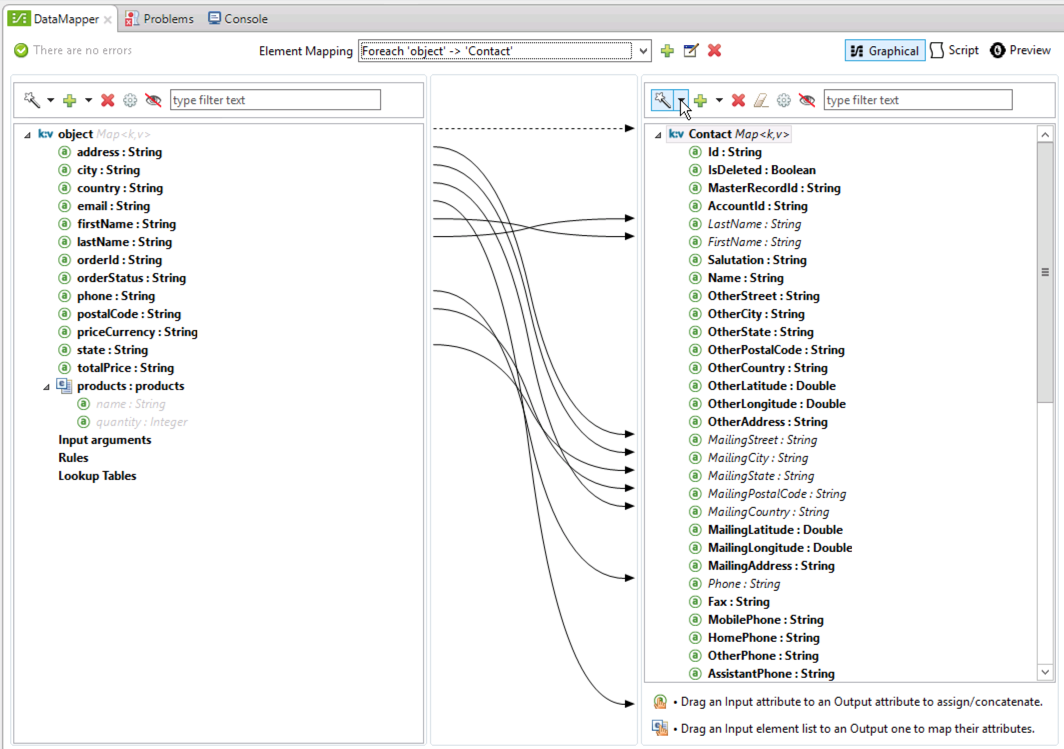
On the left side (**Input**):

1. Select **Type**: **Connector**
2. Select **Connector**: **Json\_to\_Map**. This points to the Data Mapper defined in the post:/orderBroadcast flow.
3. Select Operation: **transform**
4. Select object: **object**

|  |
| --- |
| **NOTE:** This will re-use the object from the previous data mapping in the main flow, which will be the object that will be sent to the queue. |

On the right side (**Output**), the **Contact** (not CONT**R**ACT) object should be automatically selected. DataSense has automatically introspected Salesforce and determined the type.

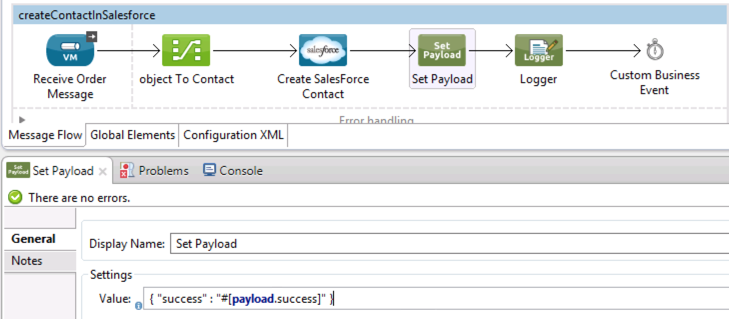
1. Click **Create Mapping**. You will see some fields automatically mapped.



1. Double-click the **DataMapper** tab to expand the window full-size so it’s easier for you to map.
2. Dragging the source (object) to the desired *Output* (Contact) field.

* **SOURCE => TARGET**
* address => MailingStreet
* city => MailingCity
* state => MailingState.
* postalCode => MailingPostalCode
* country => MailingCountry

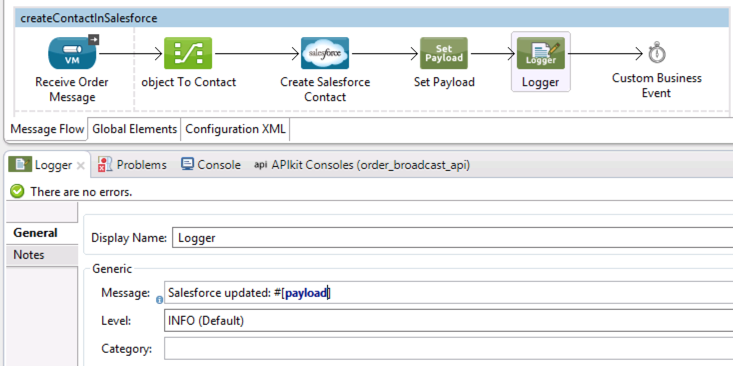
1. Double-click the **DataMapper** tab again to restore the window to normal size.



1. Double-click the **Set payload** component to define the response to be returned from this flow.
2. Set the *Value* to:

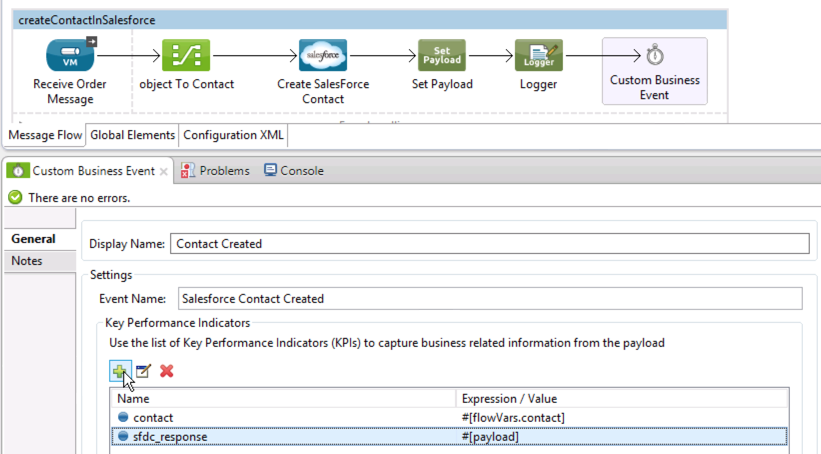
{ "success" : "#[**payload**.success]" }

This step sets the payload to the JSON string you just specified.



1. Click the **Logger** component and set the *Message* to:  
    Salesforce updated: #[payload]

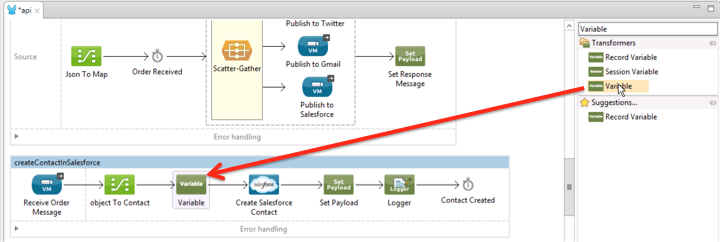
This configuration simply logs the JSON payload created in the **Set Payload** transformer.



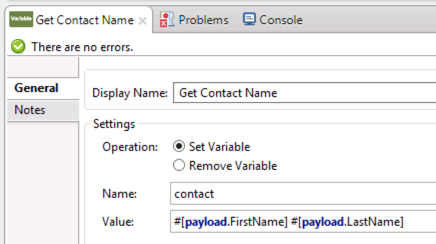
Let’s configure the custom business event that will collect information to be used for monitoring business transactions.

1. Click theCustombusiness eventcomponent
2. Set the **Display Name** to **Contact Created**.
3. Set the Event Name to **Salesforce Contact Created.**
4. Click on the icon to add the following Key Performance Indicators:

|  |  |
| --- | --- |
| Name | Value |
| contact | #[flowVars.contact] |
| sfdc\_response | #[payload] |

****

Lastly, you need to get the contact’s first name and last name and assign this to the **#[flowVars.contact]** variable. You will reference this in the custom business event. This needs to be retrieved before we call Salesforce so that we can assign the original name values to the custom business event.

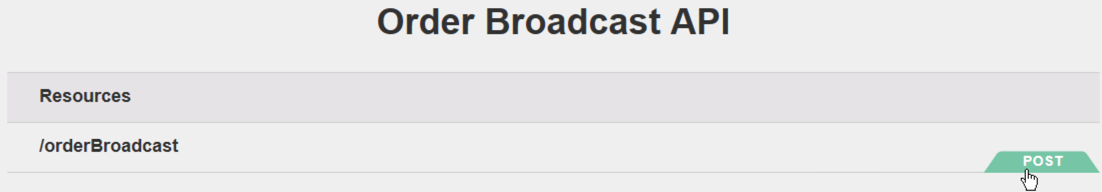


1. Select the **Variable** message processor you just added.
2. Set the Display name to **Get Contact Name**
3. Name the variable **contact** and specify the following expression for the value: **#[payload.FirstName] #[payload.LastName]**
4. Click **Save All**. 

## Step 4: Run the Application

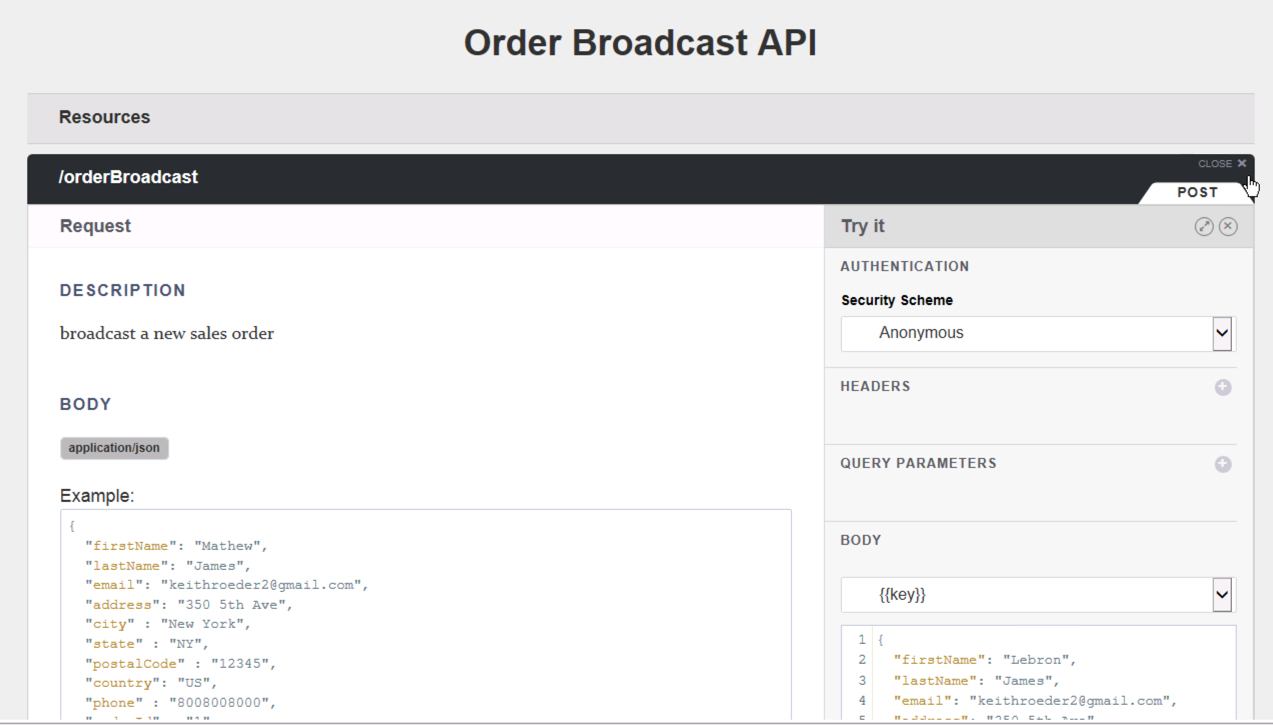
Let’s give our application a quick spin.

1. Right-click on the Project and select **Debug as Mule application.**

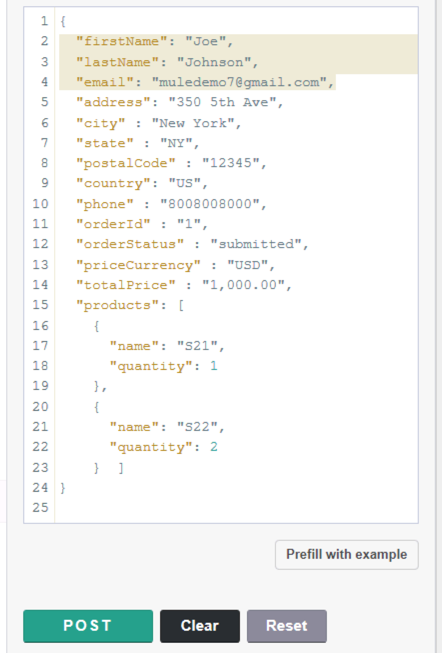


When the project has started, the APIkit Console will automatically appear.

1. Click on **POST**.

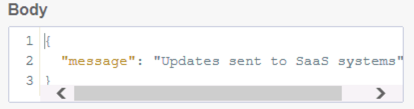


1. Explore the API
   * Review the **Request** to see what must be sent to the API
   * Review the **Response** to see what is returned.



Lets try the API. You are going to edit the prefilled data. The data was prefilled with the example data defined in the API

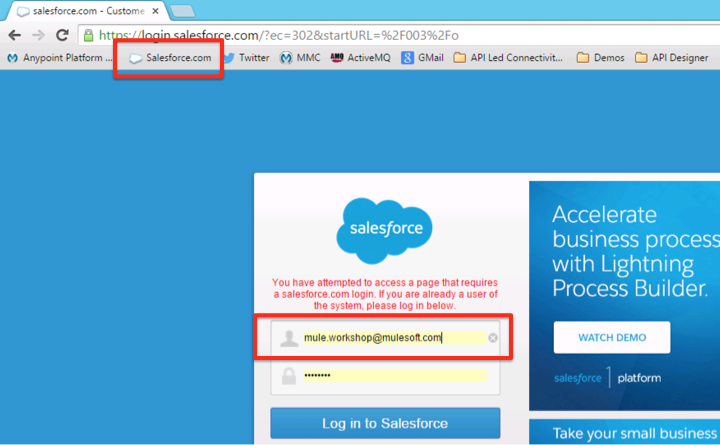
1. Change the following so you’ll be able to find your contact
   * First name
   * Last name
   * Email
2. Click **POST** to send the request to your Mule project.



1. Review the response body

{ “message”: “Updates sent to SaaS systems”}

## Step 5: Verify results



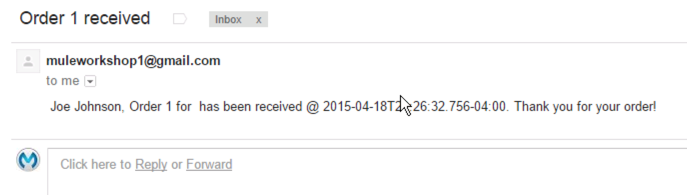
1. Login to Salesforce (Use the toolbar icon)

Username [**mule.workshop@mulesoft.com**](mailto:mule.workshop@mulesoft.com)

Password **Mule1379**

### 

1. Click on **Contacts**.
2. You should see your new Contact under Recent Contacts or by clicking **Go!** on the **New This Week** view.
3. Click your Contact to review the information you mapped.



1. Check your email to see your email order confirmation.



1. Open <https://twitter.com/muleworkshop> to see your tweet about the order (or click on the browser bookmark above.
2. Look for the welcome tweet.

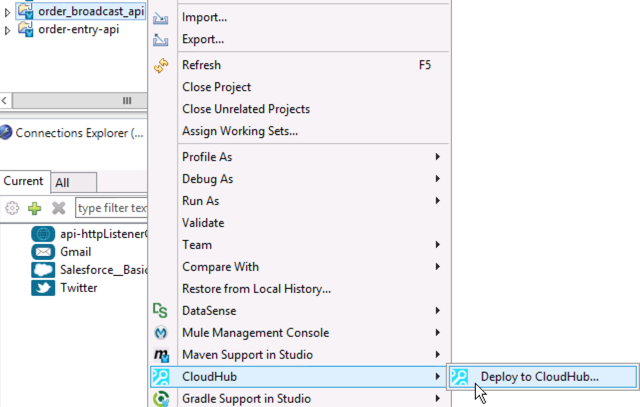
## Step 6: Deploy to CloudHub from Anypoint Studio

Now you are ready to deploy your Order Broadcast application to the Cloud using **CloudHub** - MuleSoft’s Integration-Platform-as-a-Service offering.

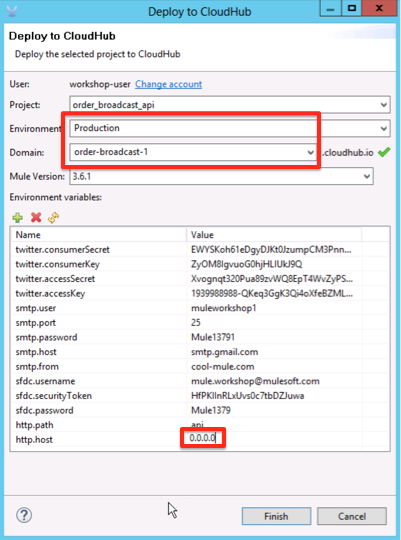
This document provides instructions for automated and manual deployment to CloudHub. Your instructor will tell you which approach to follow.

*If your instructor has created a ‘business group’ aka sub-org for all attendees in your workshop, then jump to Step 6m. The automated deployment option to CloudHub for business groups will be available 7/15/15.*

### Step 6a: Automated Deployment to CloudHub



1. Right click on the project
2. Click on CloudHub
3. Select **Deploy to CloudHub…**
4. Sign in with the Anypoint Platform account you created earlier.

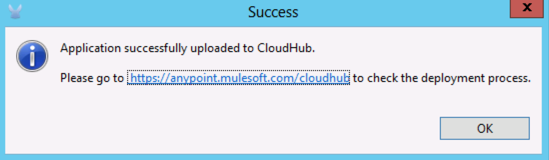


Specify the deployment information

1. Select **Production** as the Environment.
2. Specify a unique *Domain* name. This will be what you will use to access your application later on. Usually you will enter the project name. Since everyone in this class is using the same project name we need to add a qualifier to make it unique. Just add your initials or a random number to the end of the name.
3. In the **Environment Variables**, change the *http.host* from localhost to **0.0.0.0.** This will make the flow accessible from outside the CloudHub worker.

|  |
| --- |
| **DOUBLE CHECK**: Did you set http.host to 0.0.0.0? |

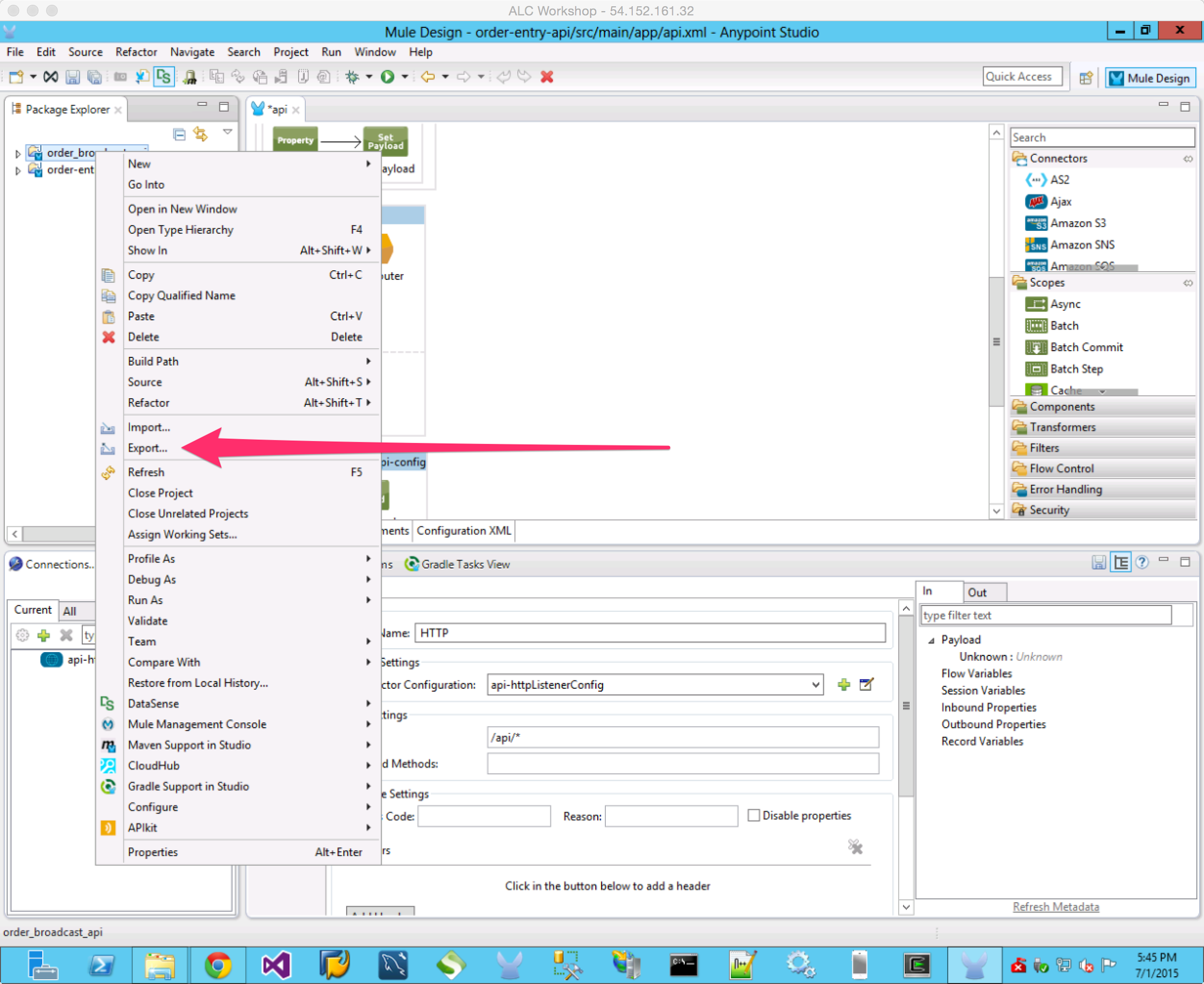
1. Leave the Mule Version as the default 3.6.1.



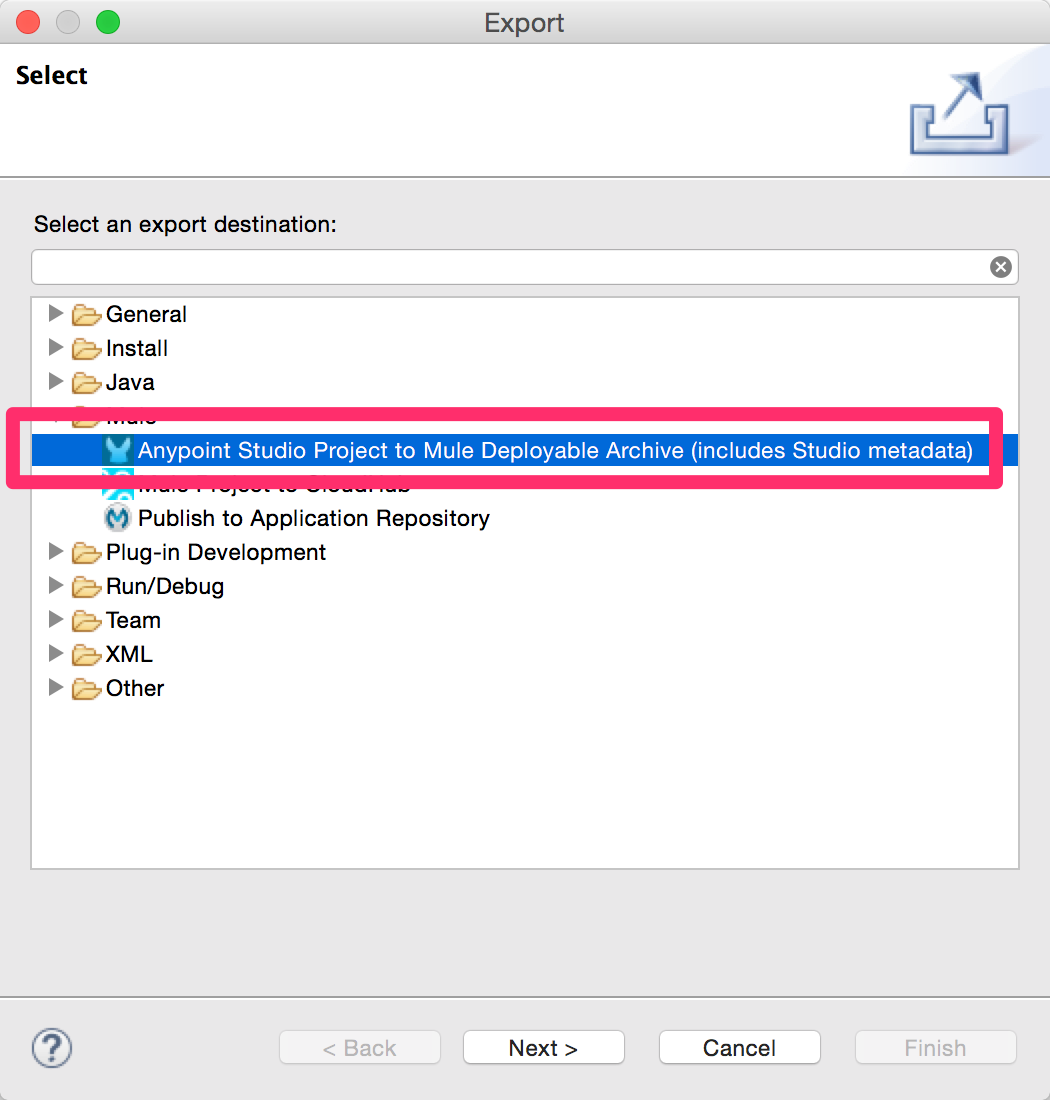
You will get a message indicating the application has been successfully uploaded to CloudHub.

### Step 6m: Manually Deploy to CloudHub

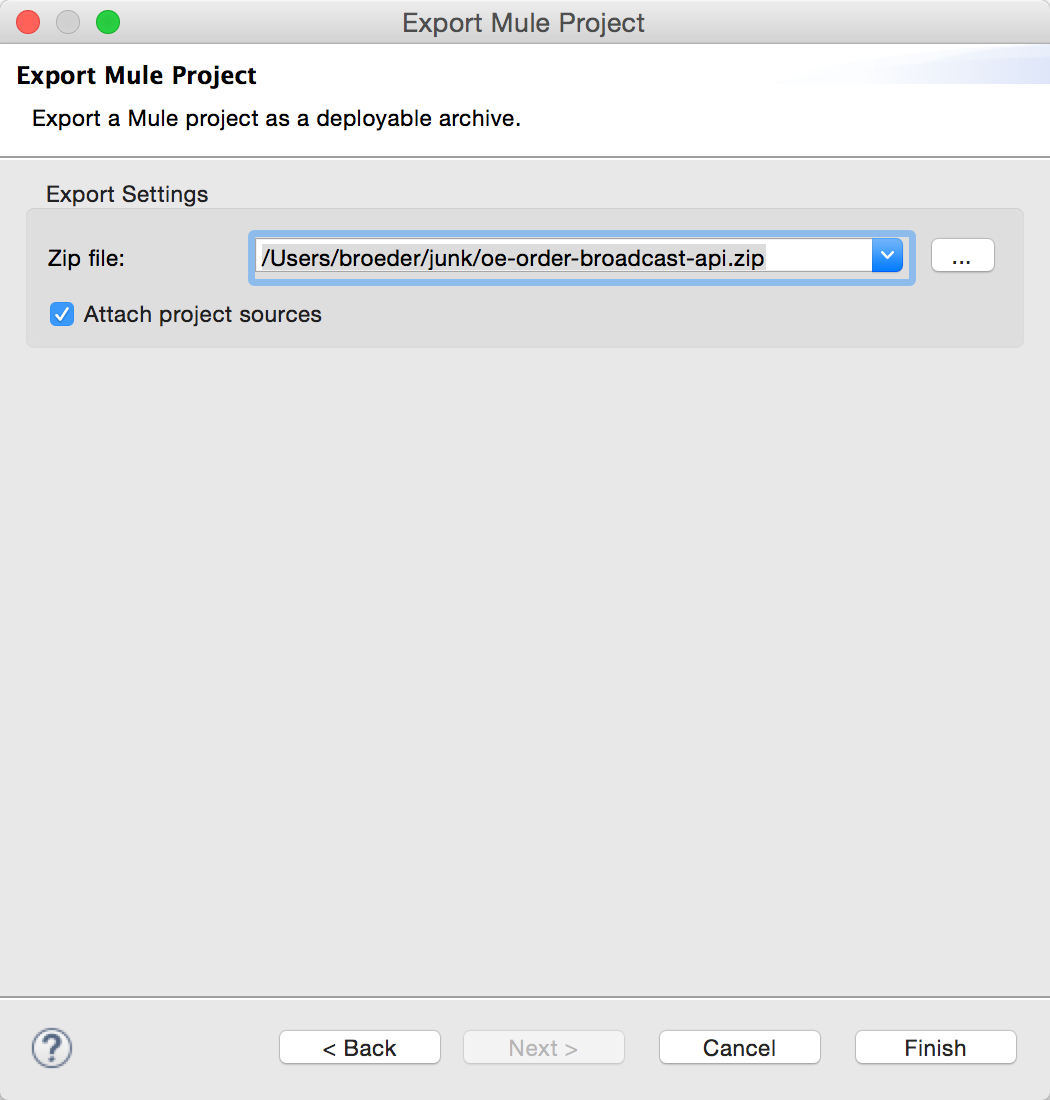
In this step you will export a deployment archive which you will then manually deploy.



1. Go to Anypoint Studio
2. Find the Project Explorer on the top left section of Studio
3. Click on the order-broadcast-api project
4. Right-click on the project
5. Select Export…

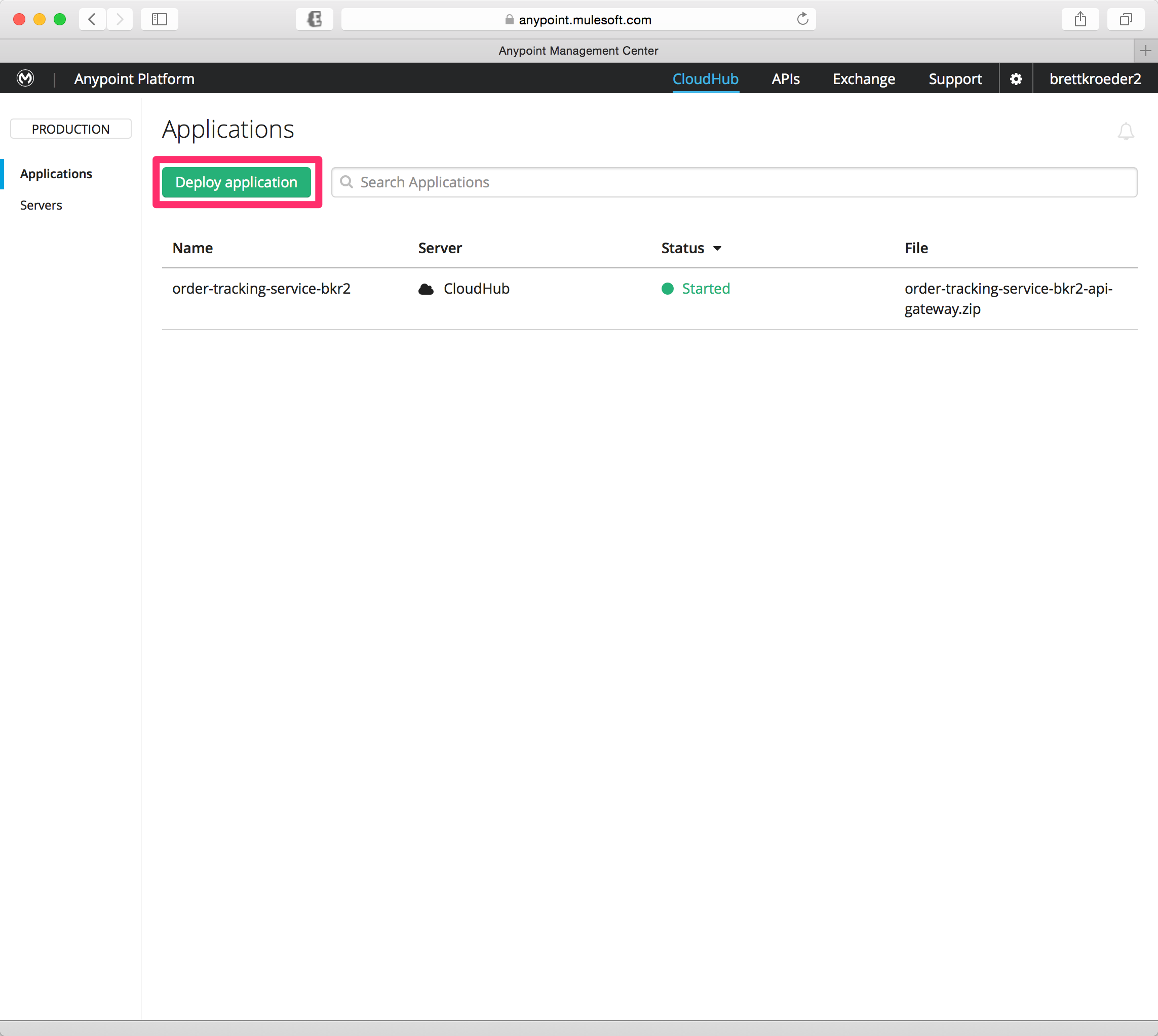


1. Select **Anypoint Studio Project to Mule Deployable Archive**
2. Click **Next**

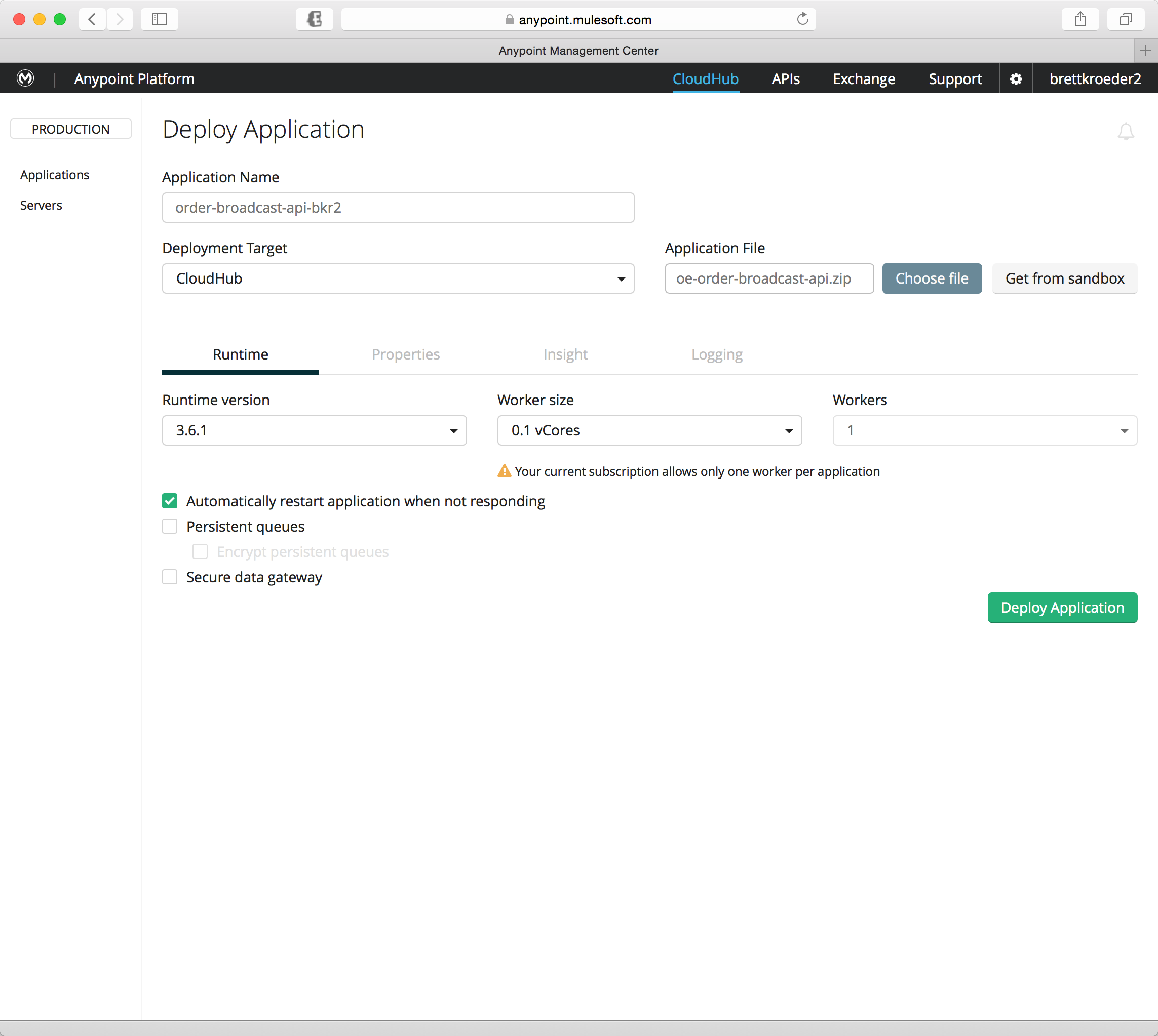


1. Set a destination for the zip file. Remember where you put the zip as you will need it in the next step.
2. Click **Next**

You now have a deployable archive.



1. Go to the **browser**
2. Go to the Anypoint Management Center (anypoint.mulesoft.com) if not already there.
3. Click on **CloudHub**
4. Click on **Deploy application**



Now you need to configure the deployment.

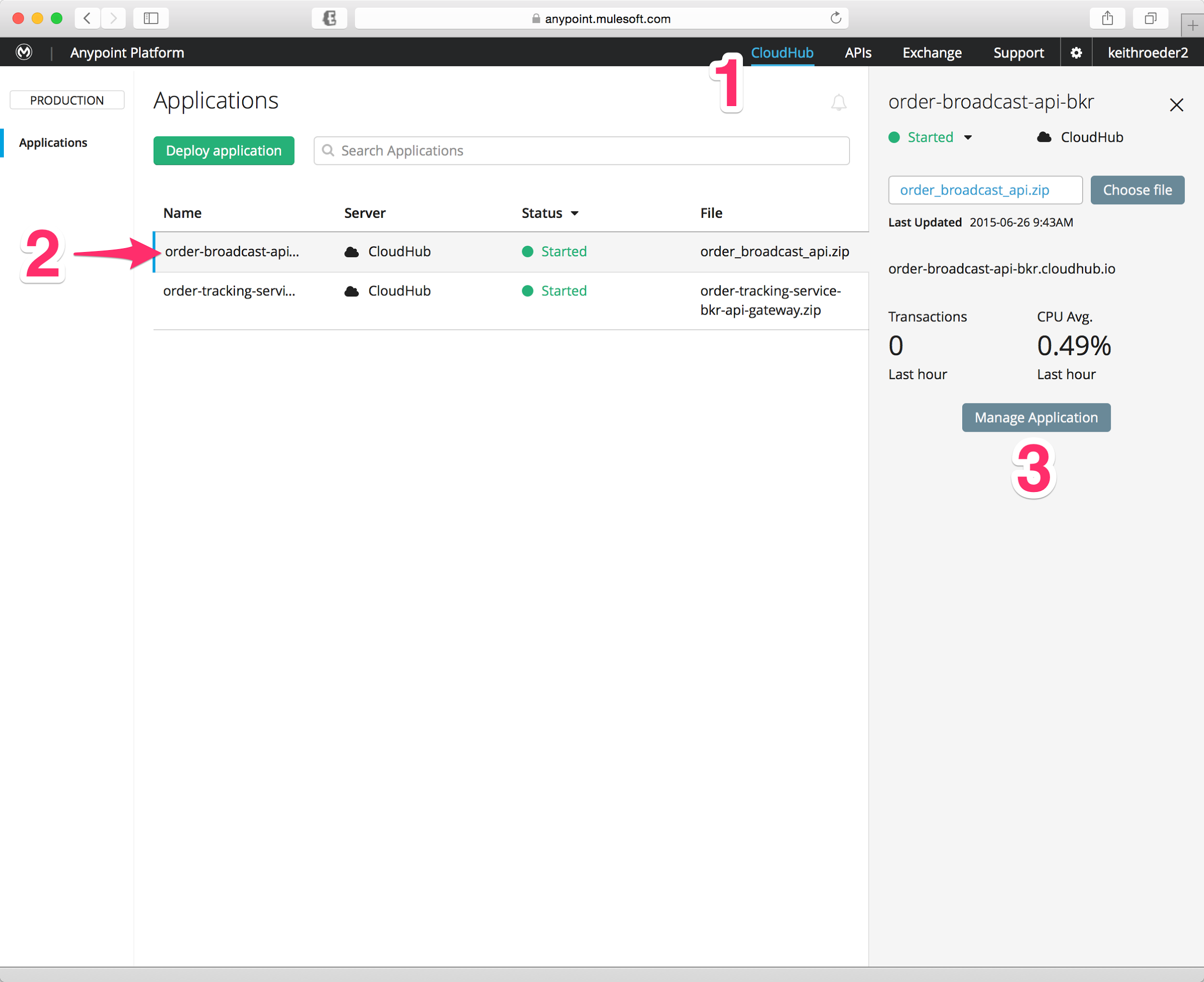
1. Set **Deployment Target** = **CloudHub**
2. Click on **Choose file**
3. **Select** the *deployable archive* you exported in the previous step.

## Step 7: Configure CloudHub App properties

If you exported a deployment archive and are deploying manually [skip to Step 8](#_mvpvgqkt6tzs).

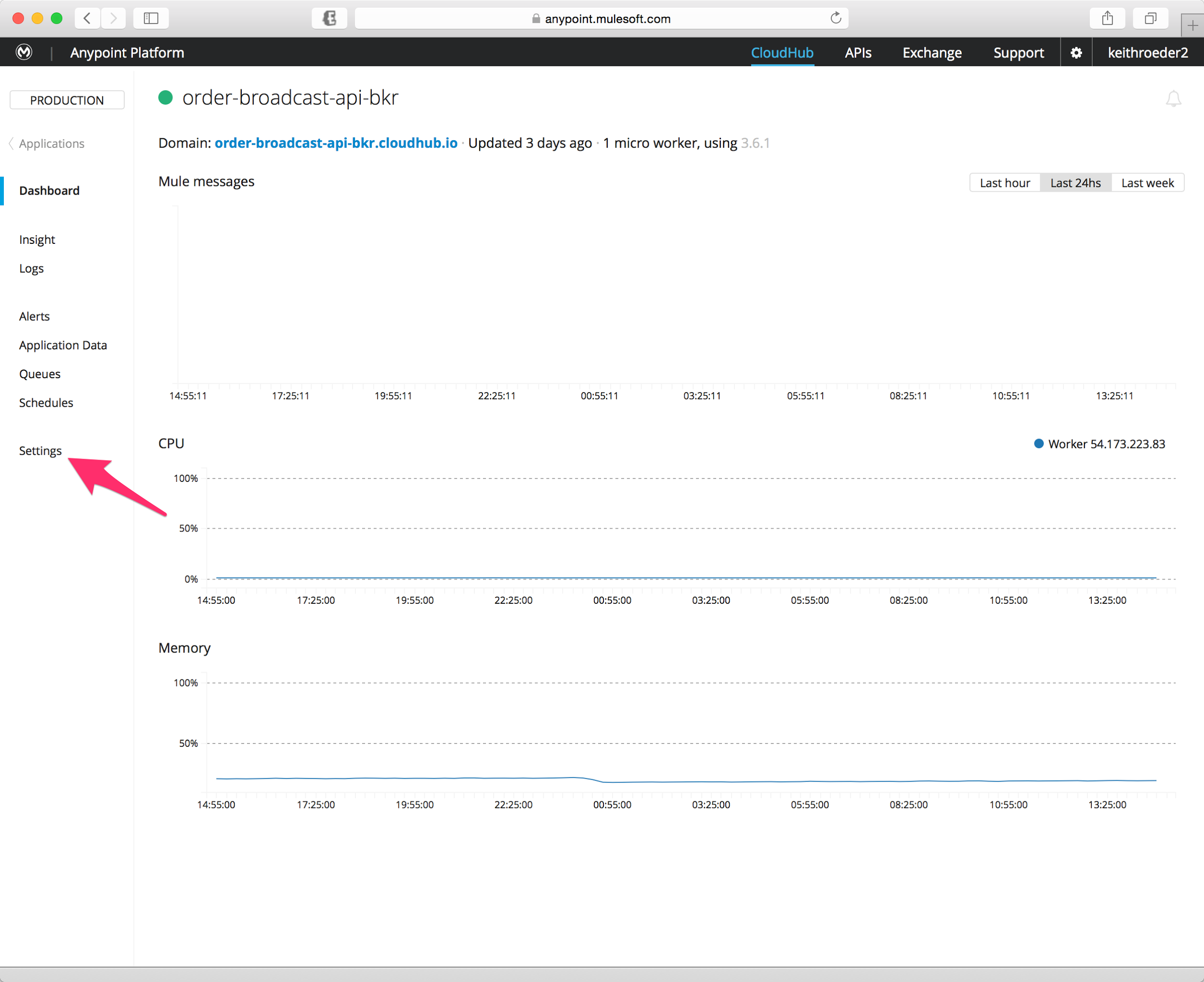
If not on the Anypoint Management Console.

1. Go to the browser
2. Click the link <https://anypoint.mulesoft.com>
3. Click on **Cloudhub**



You should see your **order-broadcast-api-XXX** application on the list of applications.

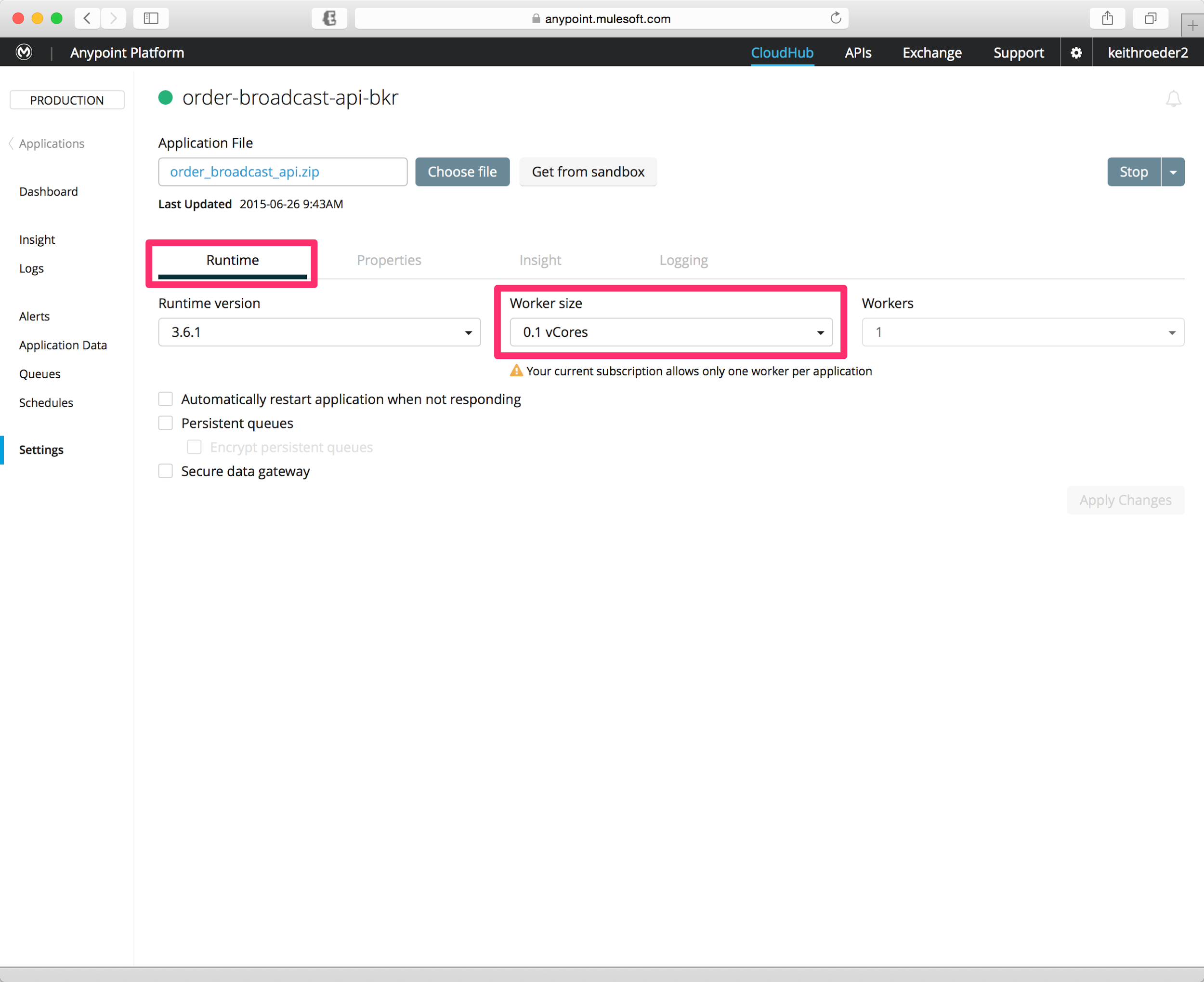
1. Click on your **order-broadcast-api-XXX** application (2)
2. Click on the **Manage Application** button (3)**.** This is where you configure your CloudHub application, including selecting the Mule version, enabling advanced features, configuring properties and logs and allocating workers.



You should now be on the Application Management page. On the left side of the page you see the different aspects of the application which you can manage/monitor.

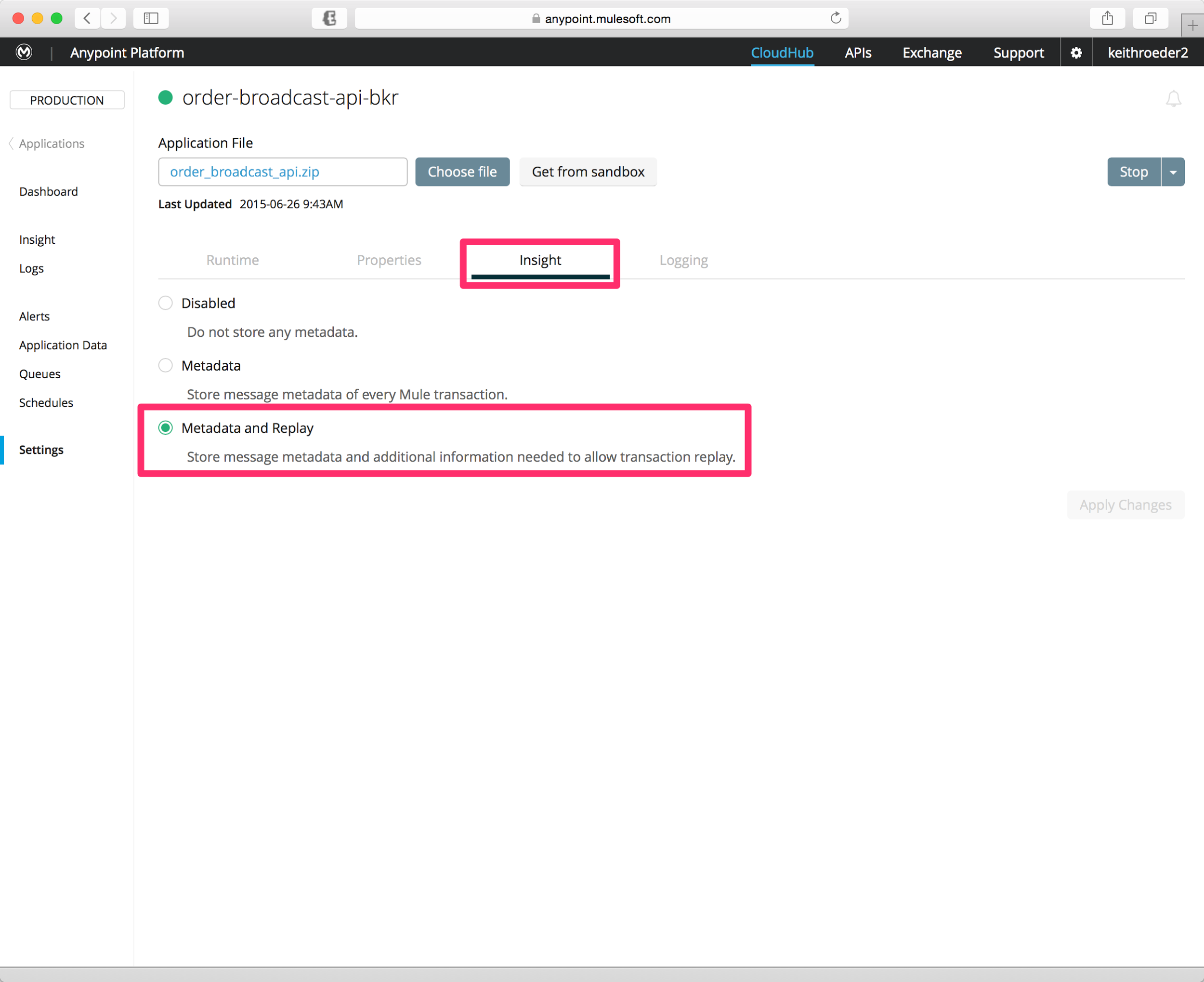
1. Click on the **Settings** option.

## Step 8: Configure the Application Properties

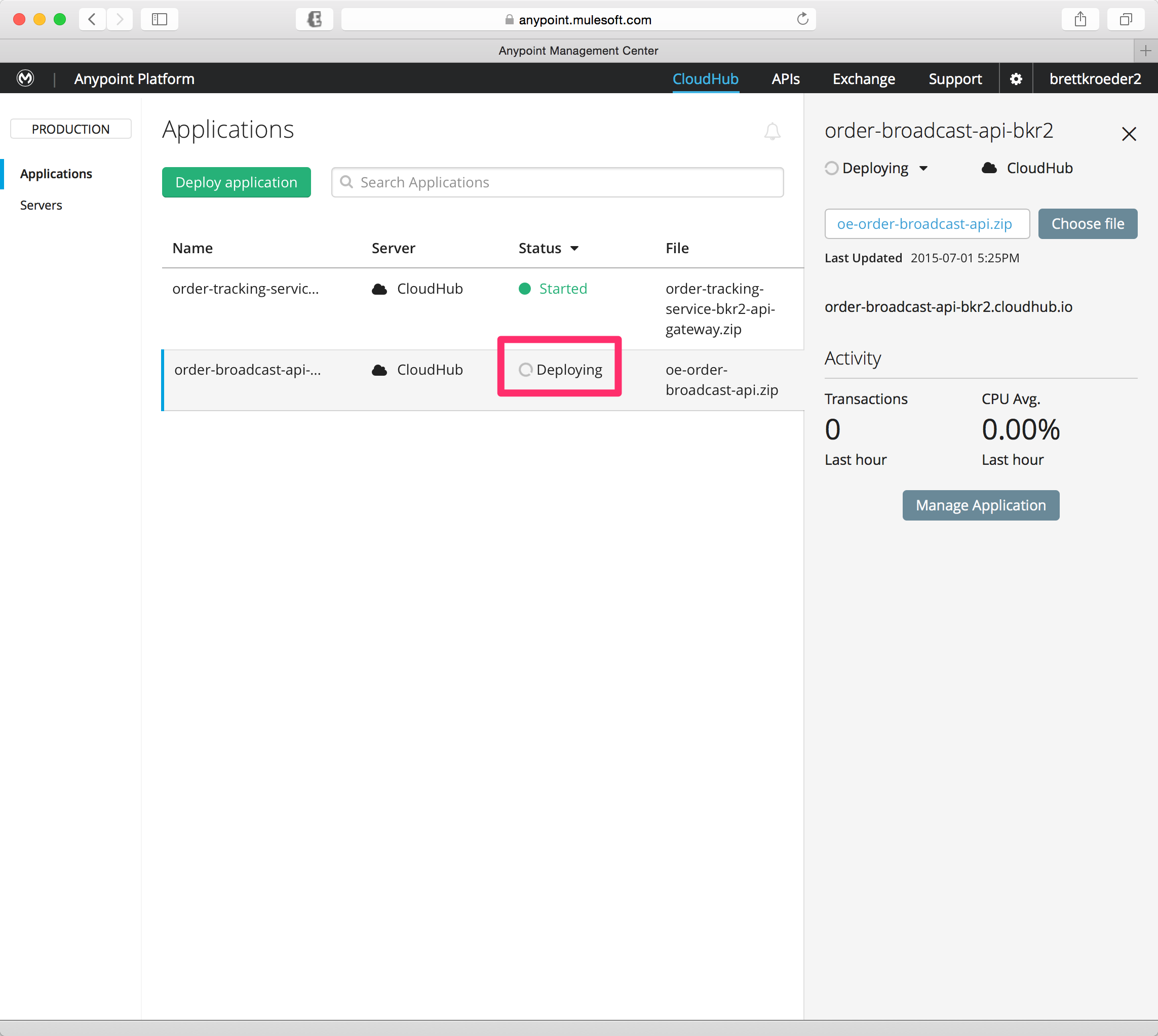


You should be sitting on the **Runtime** tab configuration.

1. Set the **Runtime version** to 3.6.1
2. Set the **Worker size** to 0.1 vCores



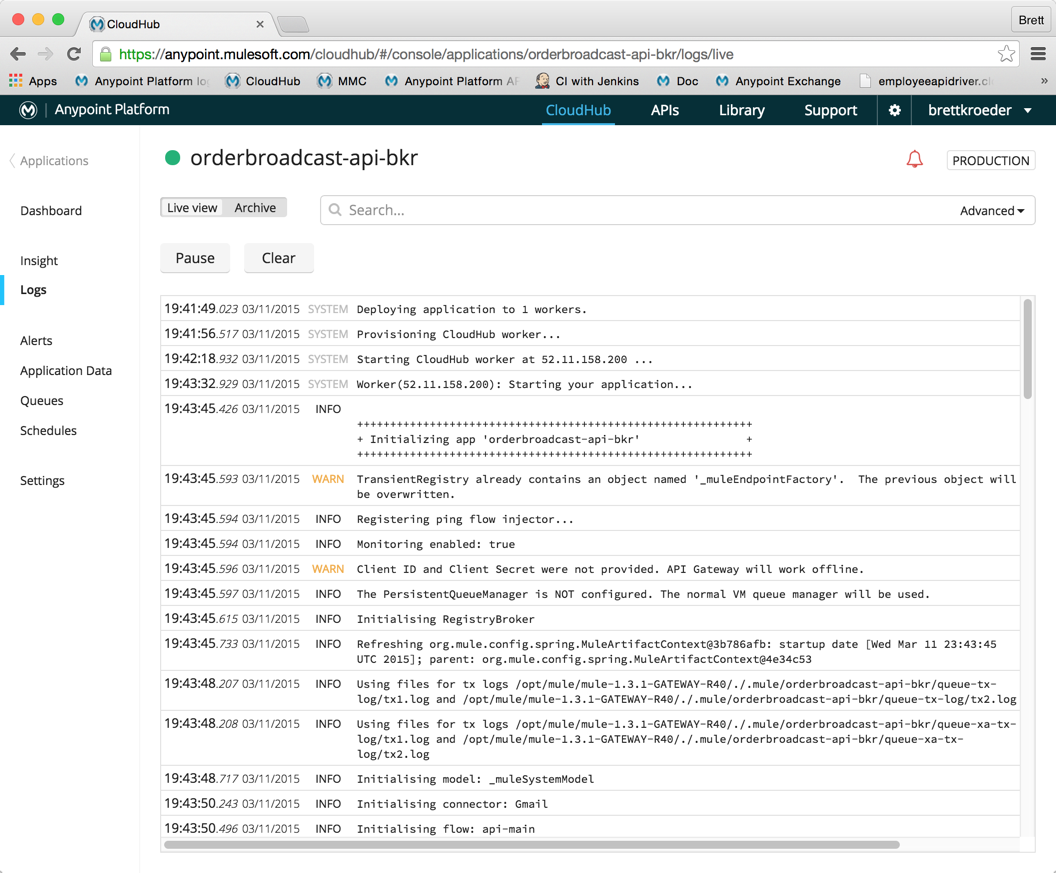
1. Click on the **Insight** tab
2. Select **Metadata and Replay** to enable KPI tracking.
3. If you did the automated deployment from Studio click **Apply Changes** in the bottom right.
4. If you are deploying manually click on **Deploy Application** in the bottom right.



You should now see your application deploying.

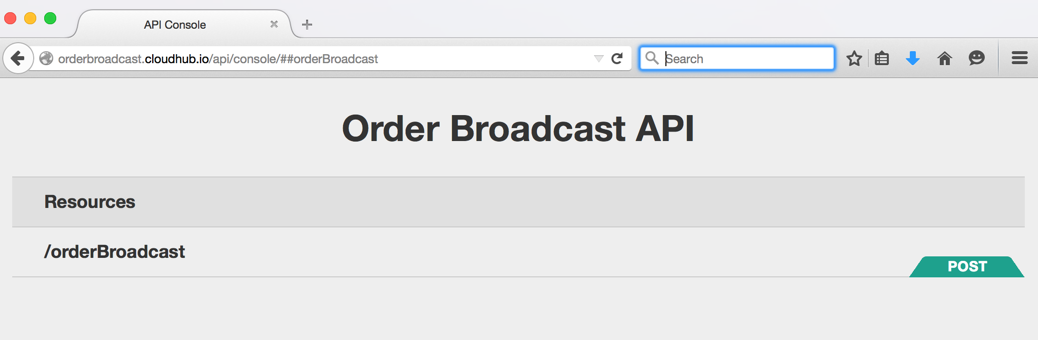
It takes about three minutes for the application to deploy so take a quick break!

Or you can click on the **Manage Application** and then **Logs** to watch the lines go by!



The icon next to your app name in the top left will turn green and you’ll see the application is started when deployment is complete.

## Step 9: Test API on CloudHub

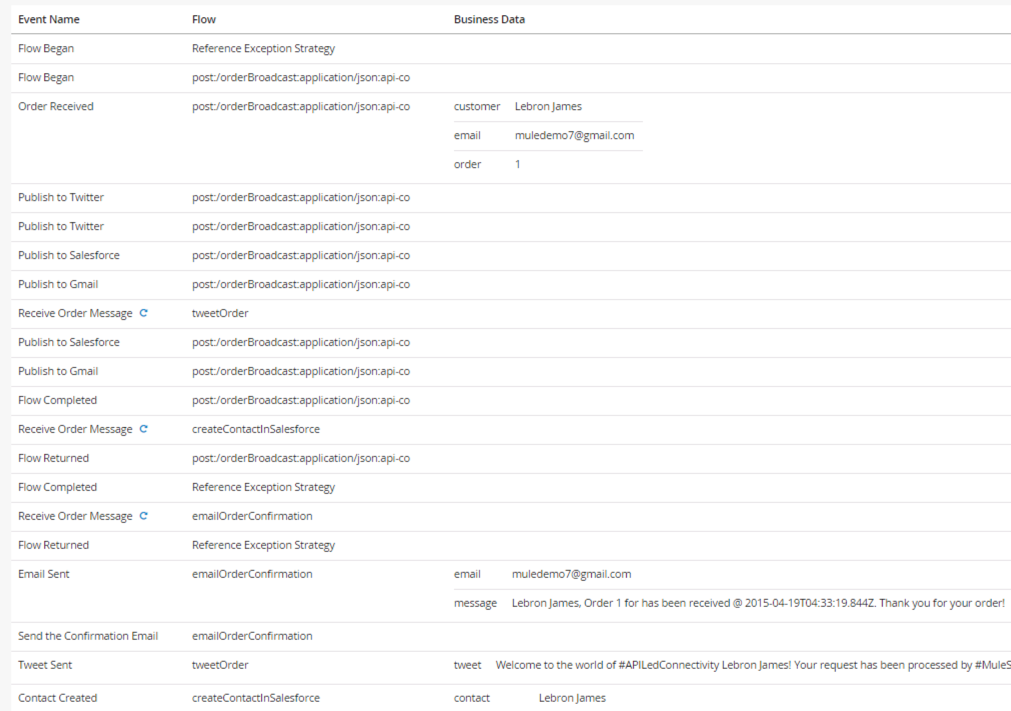


1. Open a new tab in your browser
2. Enter the your CloudHub domain URL to open the API console:

http://orderbroadcast-api-XXX.cloudhub.io/api/console

this opens the same console you’ve seen in Studio and API web site.

1. Click **POST** to run the same test you did a while ago.
2. Go back to to the CloudHub tab in your browser
3. Click on **Insight**
4. Expand one of the transactions and you will be able to see your custom business events along with the KPIs you added.



# Summary

In this lab you implemented the *Order Broadcast API* by completing the following steps

[Lab 10: Connect to SaaS and Social Networks](#_unhd2nhxtatb)

[Overview](#_ot1mfjw9473q)

[Step 1: Import a project from Anypoint Exchange](#_ln2lzaa9urmt)

[Step 2: Add VM connectors for asynchronous messaging](#_e2ymm5td51dr)

[Step 3: Add the Salesforce Connector](#_malpqon5sogh)

[Step 4: Run the Application](#_9rgrsx1iut0)

[Step 5: Verify results](#_p8dw2ui3q004)

[Step 6: Deploy to CloudHub from Anypoint Studio](#_9unton446aog)

[Step 6a: Automated Deployment to CloudHub](#_t5n5x8h1wniw)

[Step 6m: Manually Deploy to CloudHub](#_363h8mi6833r)

[Step 7: Configure CloudHub App properties](#_k38rbei8kmf2)

[Step 8: Configure the Application Properties](#_mvpvgqkt6tzs)

[Step 9: Test API on CloudHub](#_392gajid8vok)

[Summary](#_irq5eyedoeiu)

Through this lab, you saw how easy it is to consume cloud applications like Salesforce.com, Twitter and Gmail, allowing us to quickly create an API Layer on top of SaaS. We saw how to make use of asynchronous patterns using VM queues for flow-flow messaging and the scatter-gather for parallel processing. We utilized the **Anypoint Exchange** to accelerate our development efforts so we can re-use existing templates and best practices. Additionally, we have deployed this application to CloudHub, show how easy it is to get your integrations up and running on the cloud to orchestrate your SaaS applications. In the next lab, we will see how to tie this in with our on-premise integrations in a hybrid architecture.

Congratulations! You have finished Lab 10.

Please update the spreadsheet indicating you have completed Lab 10.