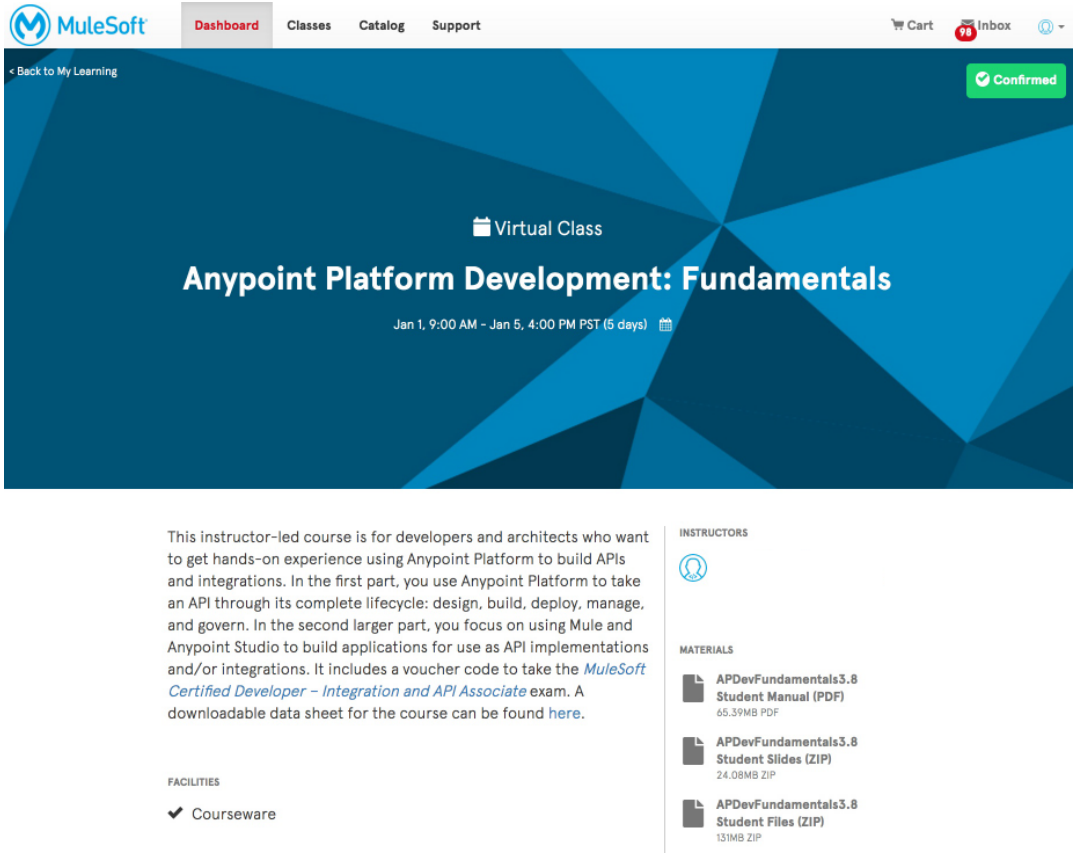
Anypoint Platform Development: Fundamentals

Student Manual

Mule runtime 3.9
December 5, 2017

Table of Contents

Introducing the Course



Objectives:

- Describe the course format.
- Download the course files.
- Make sure your computer is set up for class.
- Review the course outline.

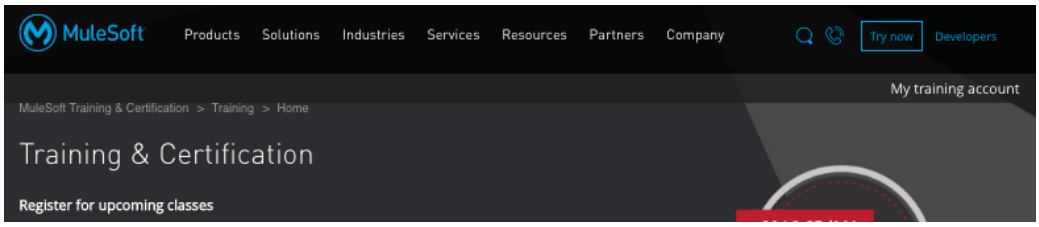
Walkthrough: Set up your computer for class

In this walkthrough, you make sure your computer is set up correctly so you can complete the class exercises. You will:

- Download the course files from the MuleSoft Training Learning Management System.
- Make sure you have JDK 1.8 and that it is included in your PATH environment variable.
- Make sure Anypoint Studio starts successfully.
- Install Postman (if you did not already).
- Make sure you have an active Anypoint Platform account.
- Make sure you have a Salesforce developer account and an API security token.

Download student files

1. In a web browser, navigate to http://training.mulesoft.com.
2. Click the My training account link.

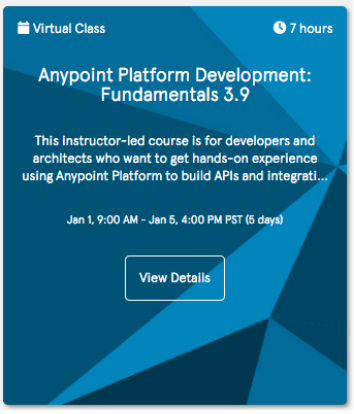


3. Log in to your MuleSoft training account using the email that was used to register you for class.

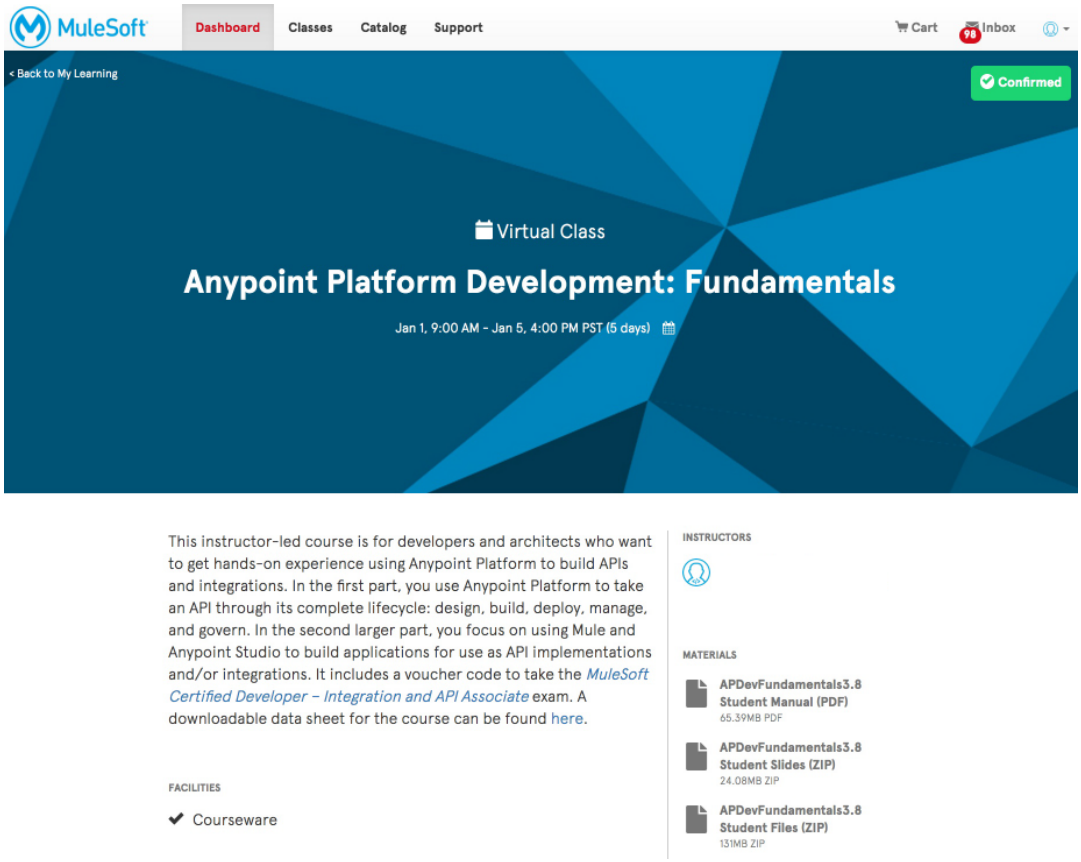
   Note: If you have never logged in before and do not have a password, click the Forgot your password link, follow the instructions to obtain a password, and then log in.

4. On the My Learning page, locate the card for your class.

   Note: If you do not see your event, locate the Current and Completed buttons under the My Learning tab, click the Completed button, and look for your event here.



5. Click the event's View Details button.
6. Locate the list of course materials on the right side of the page.



7. Click the student files link to download the files.
8. Click the student manual link to download the manual.
9. Click the student slides link to download the slides.
10. On your computer, locate the student files ZIP and expand it.
11. Open the course snippets.txt file.

    Note: Keep this file open. You will copy and paste text from it during class.

Make sure you have JDK 1.8

12. On your computer, open Terminal (Mac) or Command Prompt (Windows) or some other command-line interface.
13. Type java –version and press enter.

```
java -version
```

14. Look at the output and check if you have Java 1.8.

```
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

Note: If you have an older version of the JDK installed or have no version at all, go to http://www.oracle.com/technetwork/java/javase/downloads/index.html and download the correct version of JDK 1.8 for your operating system. Install and then confirm with java -version in a command-line interface again.

Note: JDK 1.9 is NOT supported.

Make sure you have Java in your PATH environment variable

15. In a command-line interface, type $PATH (Mac) or %PATH% (Windows) and press enter.
    - Mac: $PATH
    - Windows: %PATH%
16. After installing the correct JDK version, add or update an environment variable named JAVA_HOME that points to the installation location and then add JAVA_HOME/bin to your PATH environment variable.

    Note: For instructions on how to set or change environment variables, see the following instructions for PATH: http://docs.oracle.com/javase/tutorial/essential/environment/paths.html.

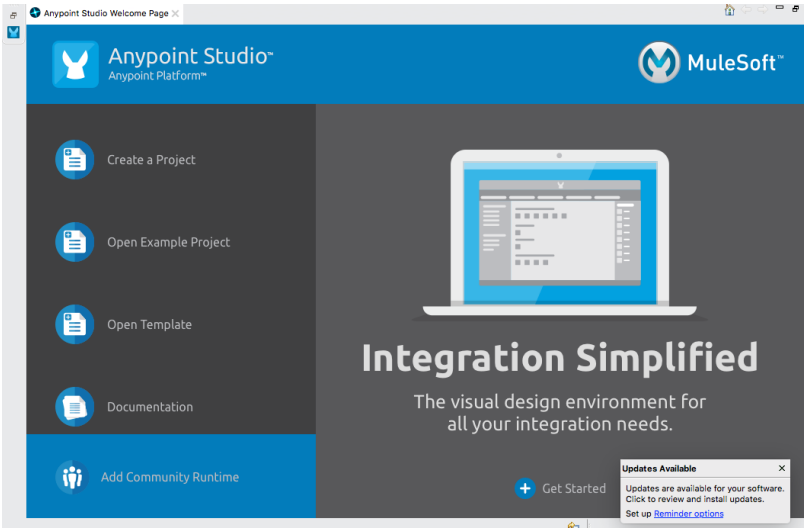
Start Anypoint Studio

17. In your computer's file browser, navigate to where you installed Anypoint Studio and open it.

    Note: If you do not have Anypoint Studio, you can download it from https://www.mulesoft.com/lp/dl/studio.
18. In the Workspace Launcher dialog box, look at the location of the default workspace; change the workspace location if you want.
19. Click OK to select the workspace; Anypoint Studio should open.

    Note: If you cannot successfully start Anypoint Studio, make sure the JDK and Anypoint Studio are BOTH 64-bit or BOTH 32-bit and that you have enough available memory (at least 4GB available) to run Anypoint Studio.

20. If you get a Welcome Page, click the X on the tab to close it.
21. If you get an Updates Available pop-up in the lower-right corner of the application, click it and install the available updates.

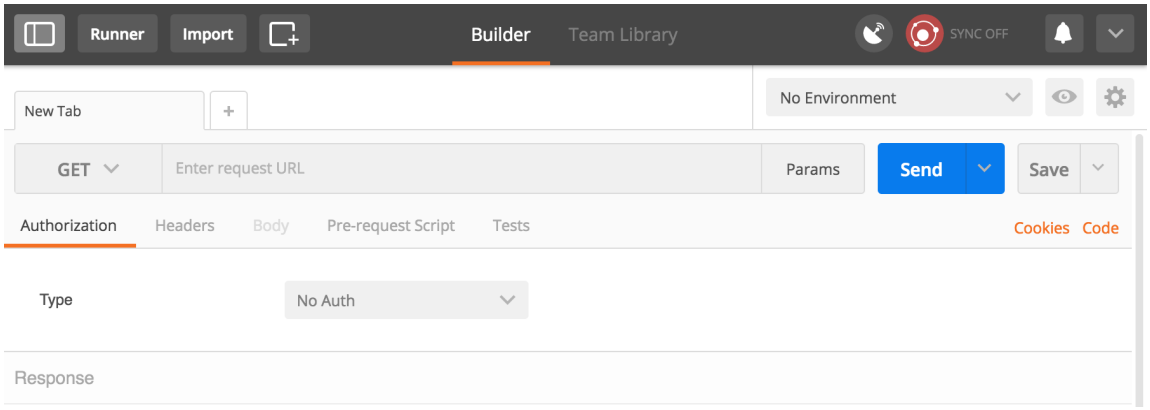


Open Postman

22. Open Postman.

    Note: If you do not have Postman or another REST API client installed, download Postman now from https://www.getpostman.com and install it.

23. Leave Postman open; you will use it throughout class.



Make sure you have an active Anypoint Platform account

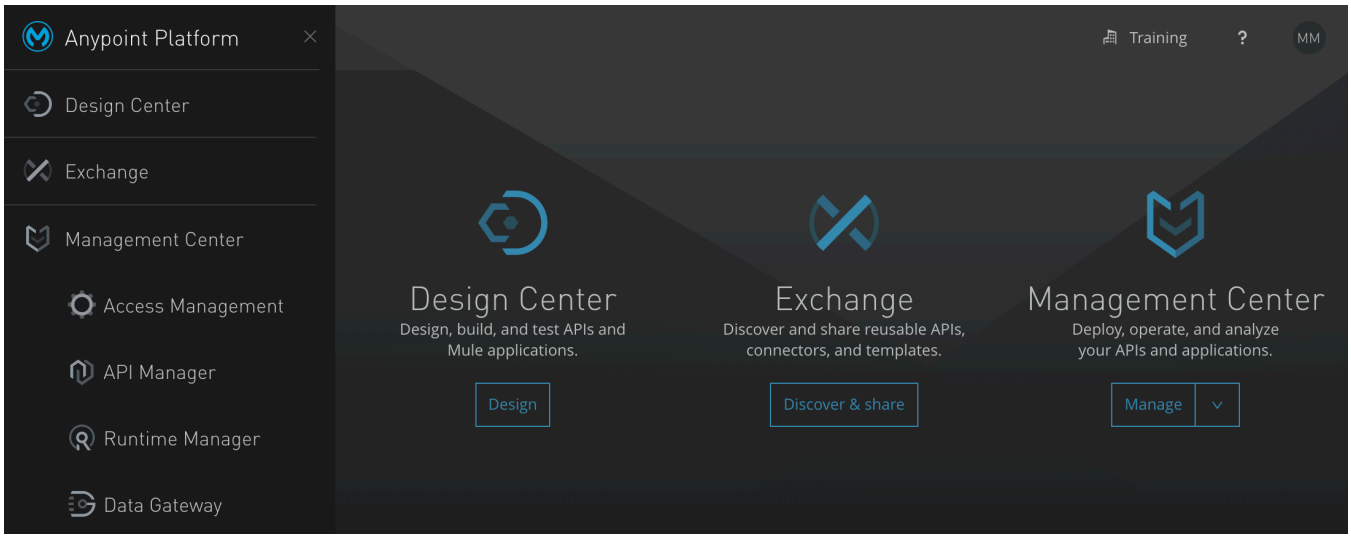
24. In a web browser, navigate to http://anypoint.mulesoft.com/ and log in.

    Note: You can use a trial account or your company account (if you already have one). If you do not have an account, sign up for a free, 30-day trial account now.

25. Click the menu button located in the upper-left in the main menu bar.

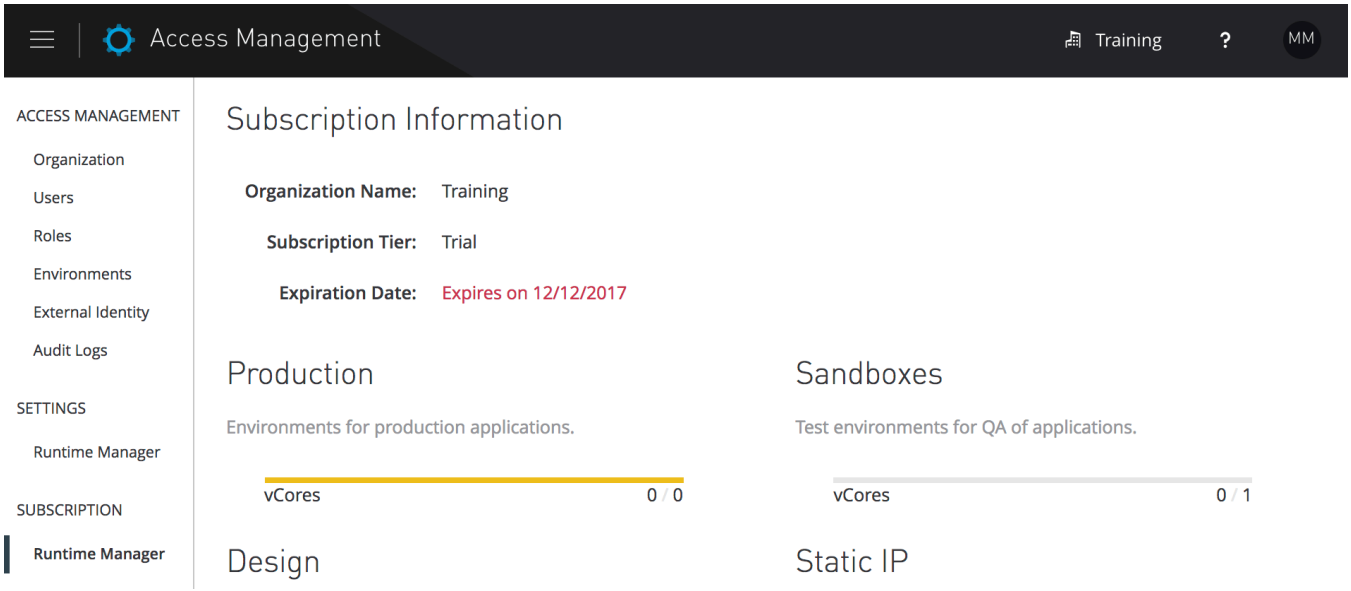
26. In the menu that appears, select Access Management.

    Note: This will be called the main menu from now on.



27. In the left-side navigation, click the Runtime Manager link under Subscription.
28. Check your subscription level and if it is a trial account, make sure it is not expired.

    Note: If your trial is expired or will expire during class, sign out and then sign up for a new trial account now.

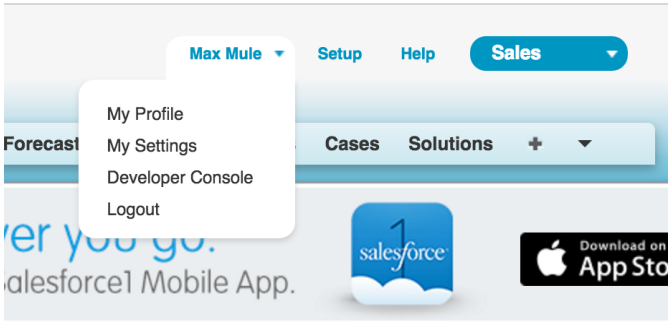


Make sure you have a Salesforce developer account and an API security token

29. In the same or another web browser tab, navigate to http://salesforce.com and log in.

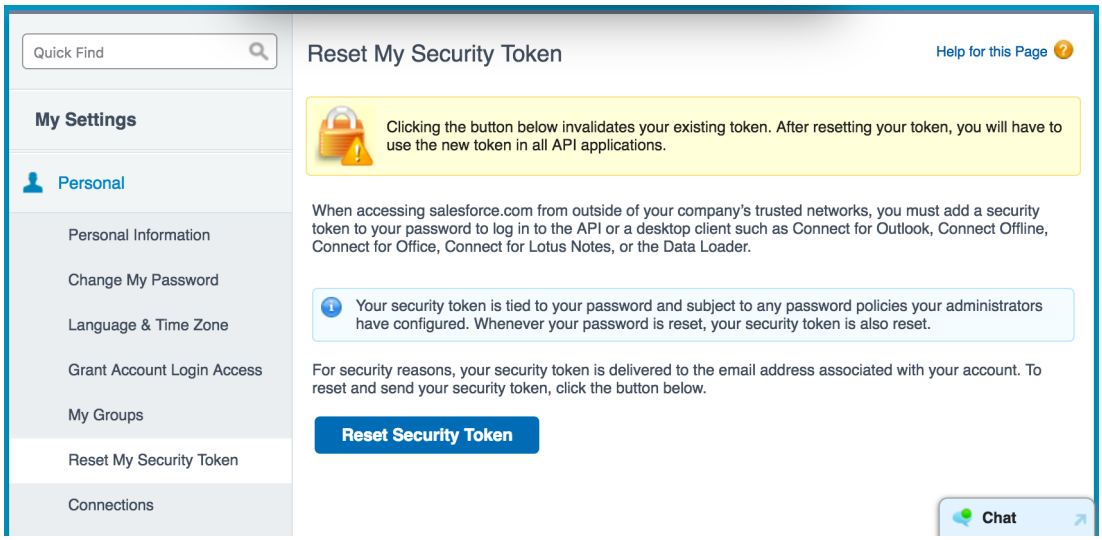
    Note: If you did not sign up for a free developer account yet, go to http://developer.salesforce.com/ and sign up for one now. You will want to use a free developer account and not your company account (if you already have one) for class. You will use the API to add new fictitious accounts to it and will probably not want to add those to your real data.

30. In Salesforce, click your name at the top of the screen and select My Settings.

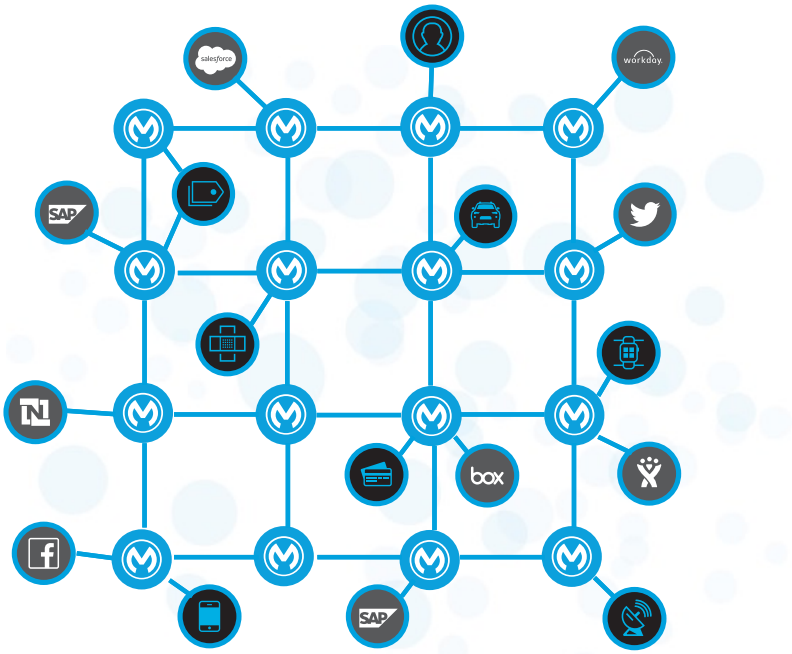


31. In the left-side navigation, select Personal > Reset My Security Token.
32. If you did not already request a security token, click the Reset Security Token button.

    Note: A security token will be sent to your email in a few minutes. You will need this token to make API calls to Salesforce from your Mule applications.



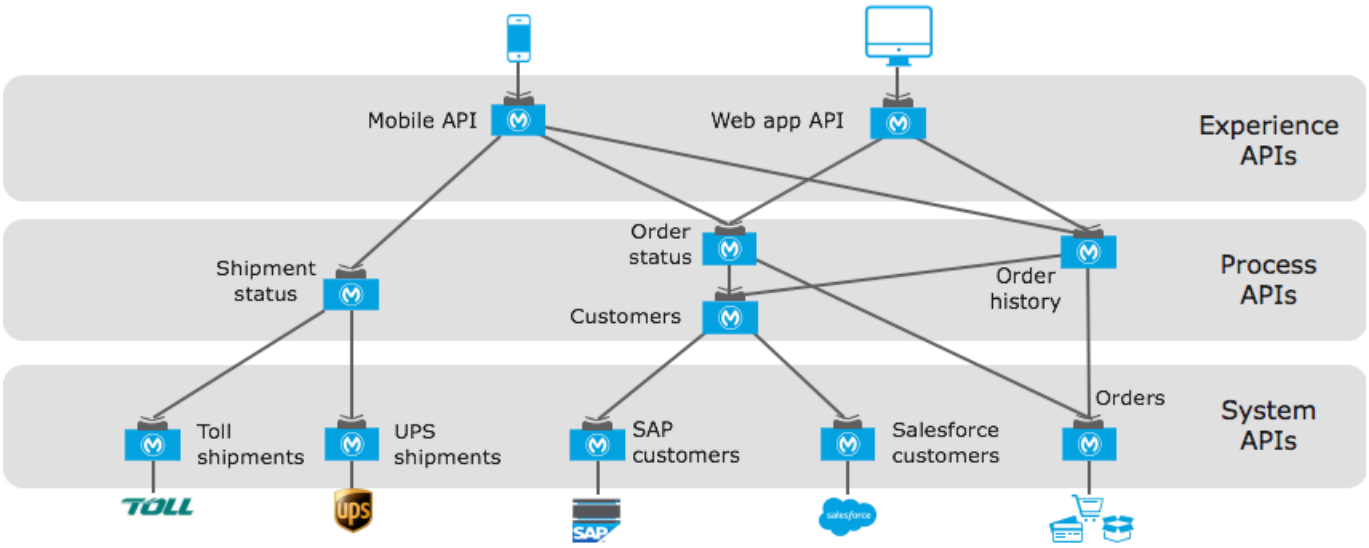
PART 1: Building Application Networks with Anypoint Platform



At the end of this part, you should be able to:

- Describe and explain the benefits of application networks & API-led connectivity.
- Use Anypoint Platform as a central repository for the discovery and reuse of assets.
- Use Anypoint Platform to build applications to consume assets and connect systems.
- Use Anypoint Platform to take an API through its complete development lifecycle.

Module 1: Introducing Application Networks and API-Led Connectivity



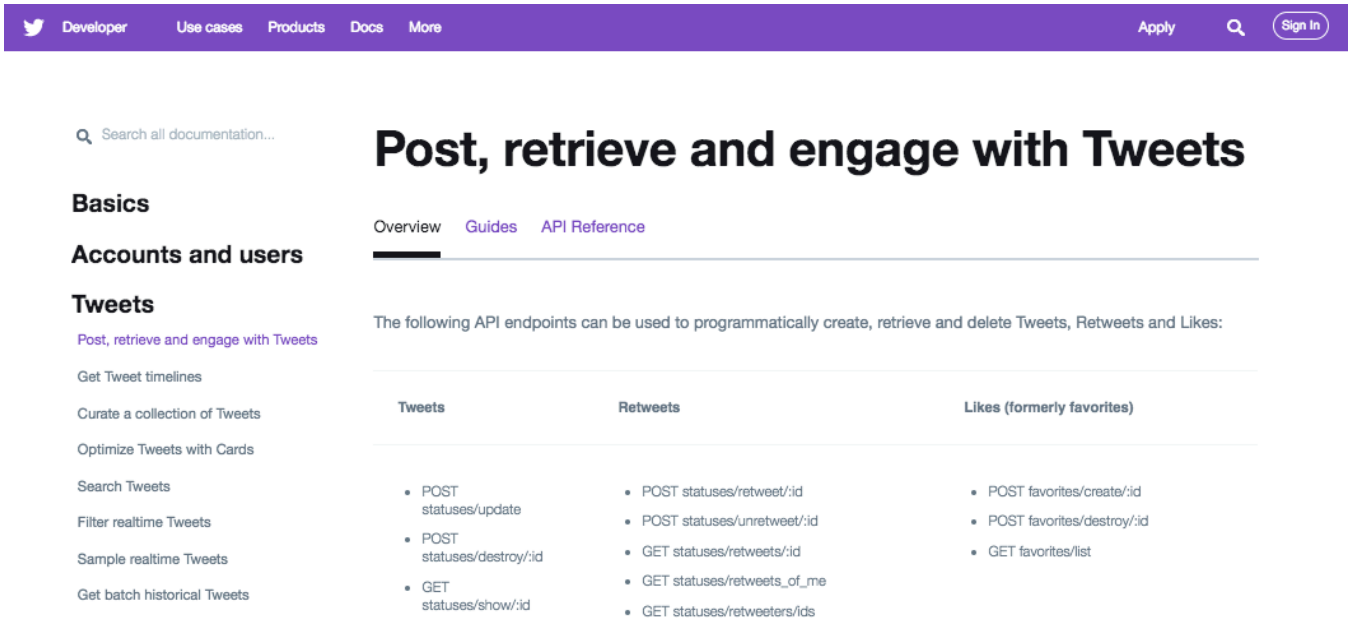
At the end of this module, you should be able to:

- Explain what an application network is and its benefits.
- Describe how to build an application network using API-led connectivity.
- Explain what web services and APIs are.
- Explore API directories and references.
- Make calls to secure and unsecured APIs.

Walkthrough 1-1: Explore an API directory and an API reference

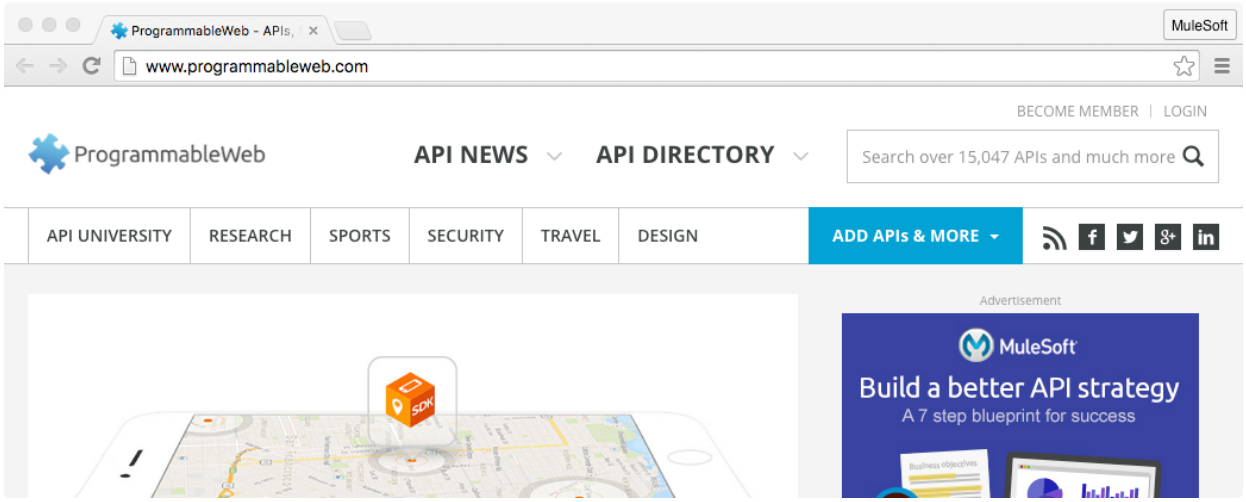
In this walkthrough, you make calls to a RESTful API. You will:

- Browse the ProgrammableWeb API directory.
- Explore the API Reference for the Twitter API.

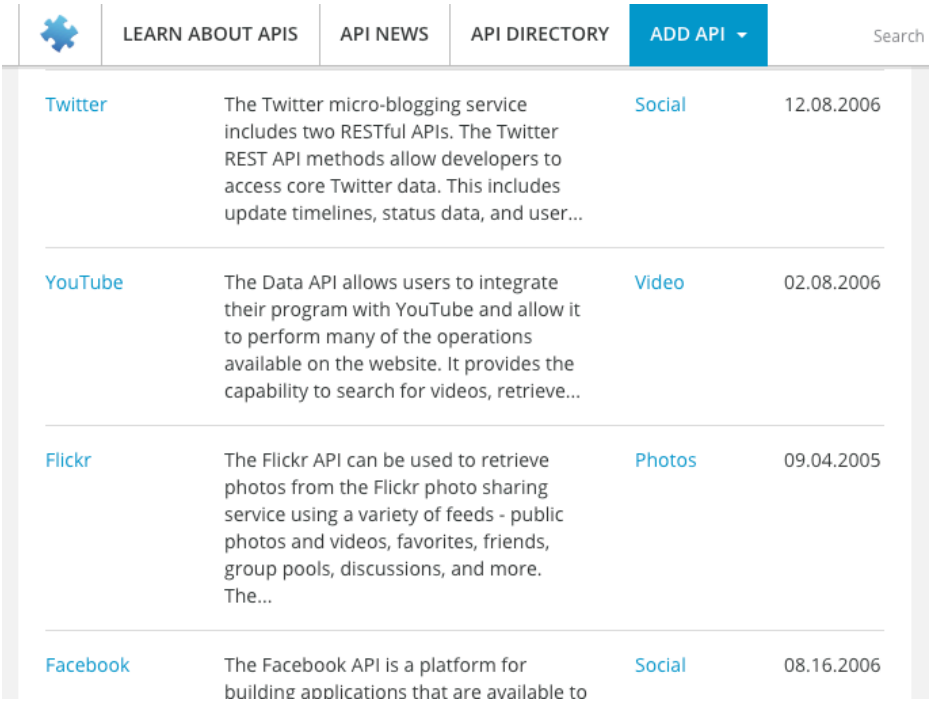


Browse the ProgrammableWeb API directory

1. In a web browser, navigate to http://www.programmableweb.com/.
2. Click the API directory link.



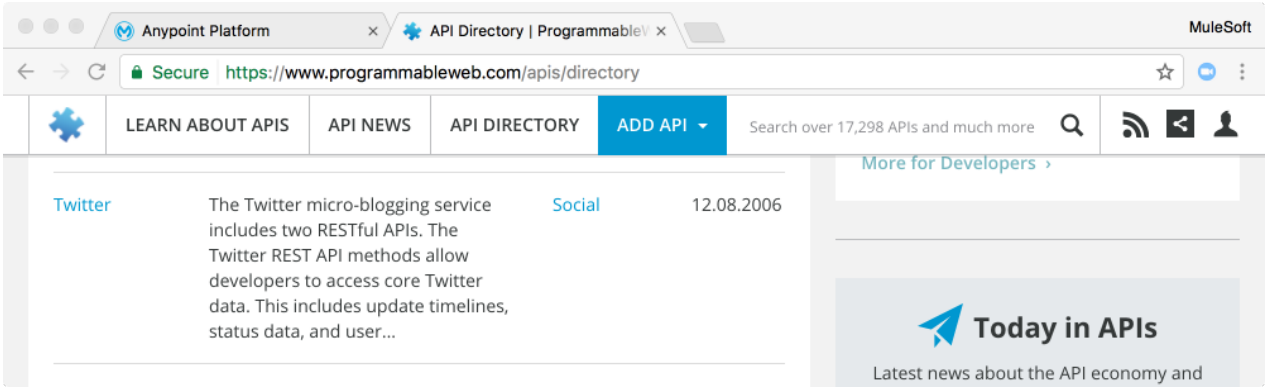
3. Browse the list of popular APIs.



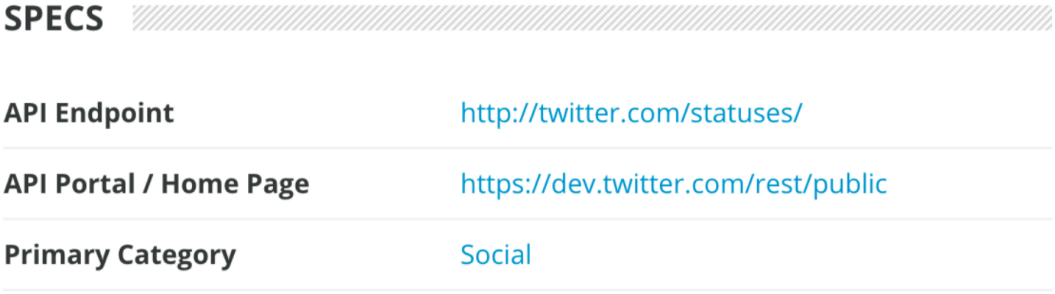
Explore the API reference for the Twitter API

4. Scroll down and click the link for the Twitter API.

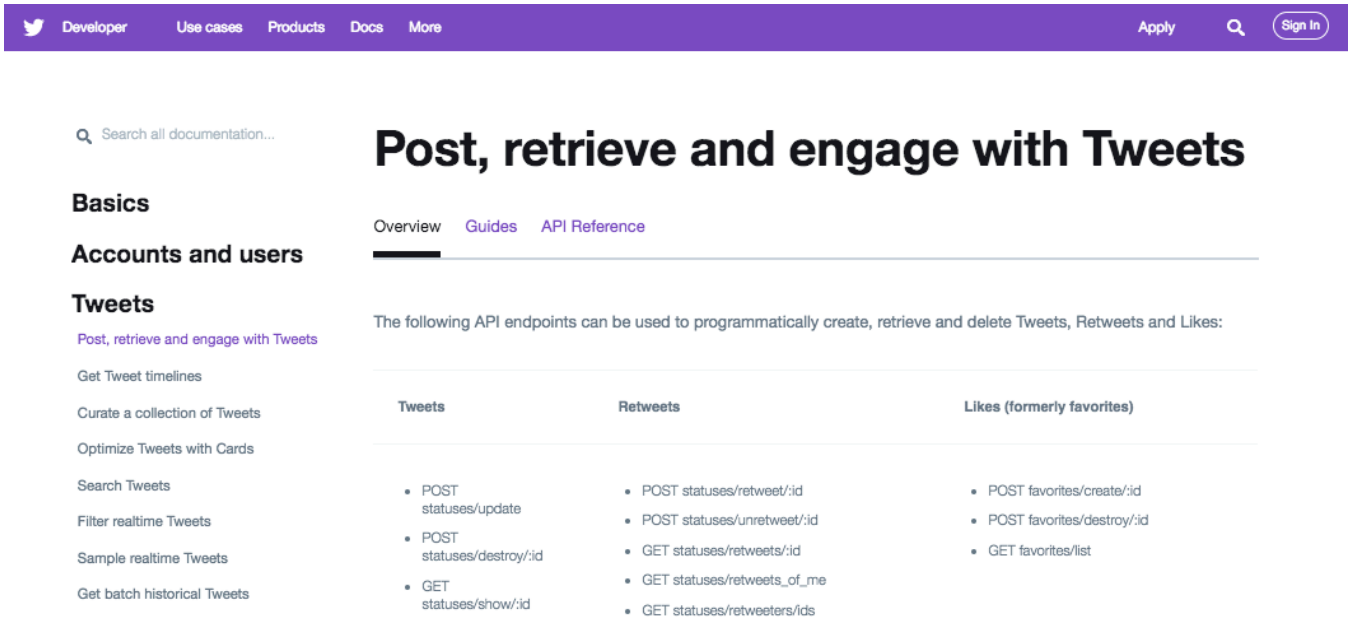
   Note: If Twitter is no longer displayed on the main page, search for it.



5. In the Specs section, click the API Portal / Home Page link.

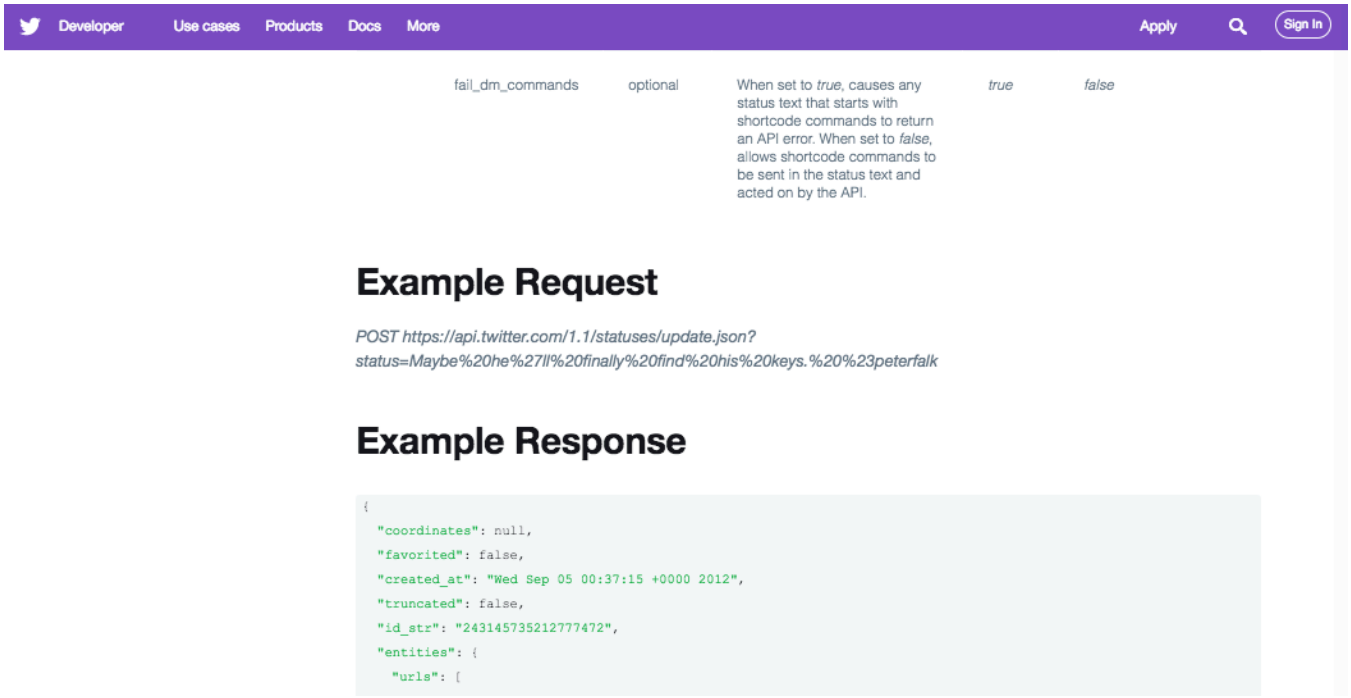


6. In the new browser tab that opens, click Tweets in the left-side navigation.
7. In the left-side navigation, click Post, retrieve and engage with Tweets.



8. Browse the list of requests you can make to the API.
9. Select the API Reference tab beneath the title of the page.

10. Review the information for POST statuses/update including parameters, example request, and example response.

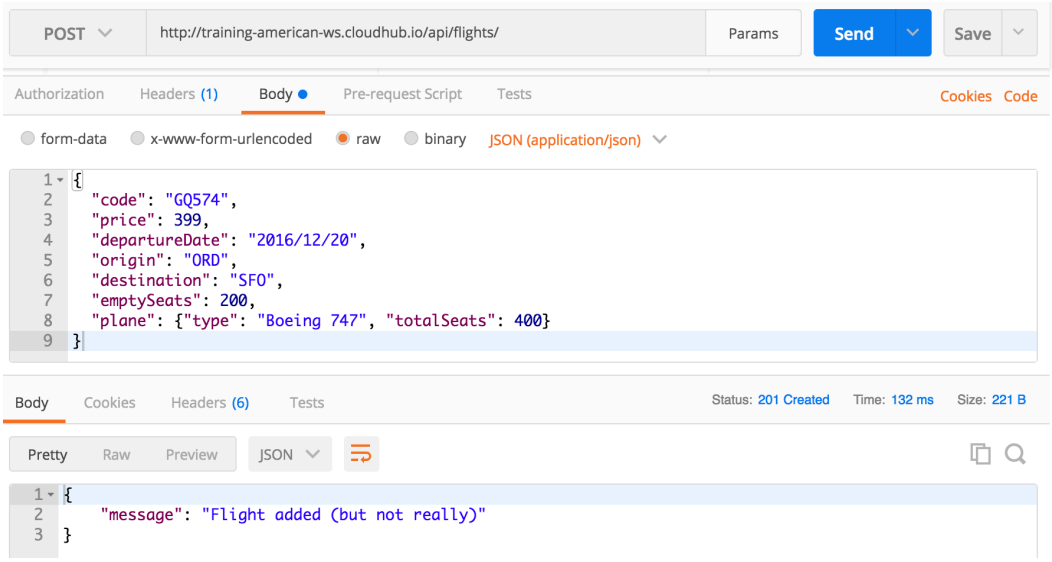


11. Close the two browser tabs.

Walkthrough 1-2: Make calls to an API

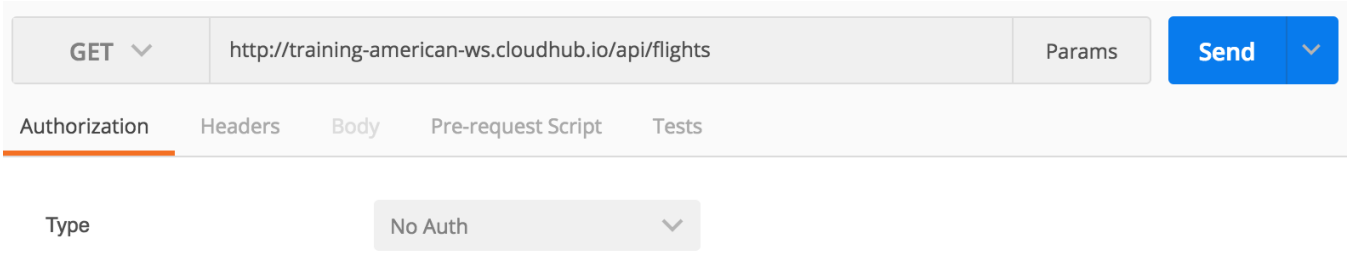
In this walkthrough, you make calls to a RESTful API. You will:

- Use Postman to make calls to an unsecured API (an implementation).
- Make GET, DELETE, POST, and PUT calls.
- Use Postman to make calls to a secured API (an API proxy).

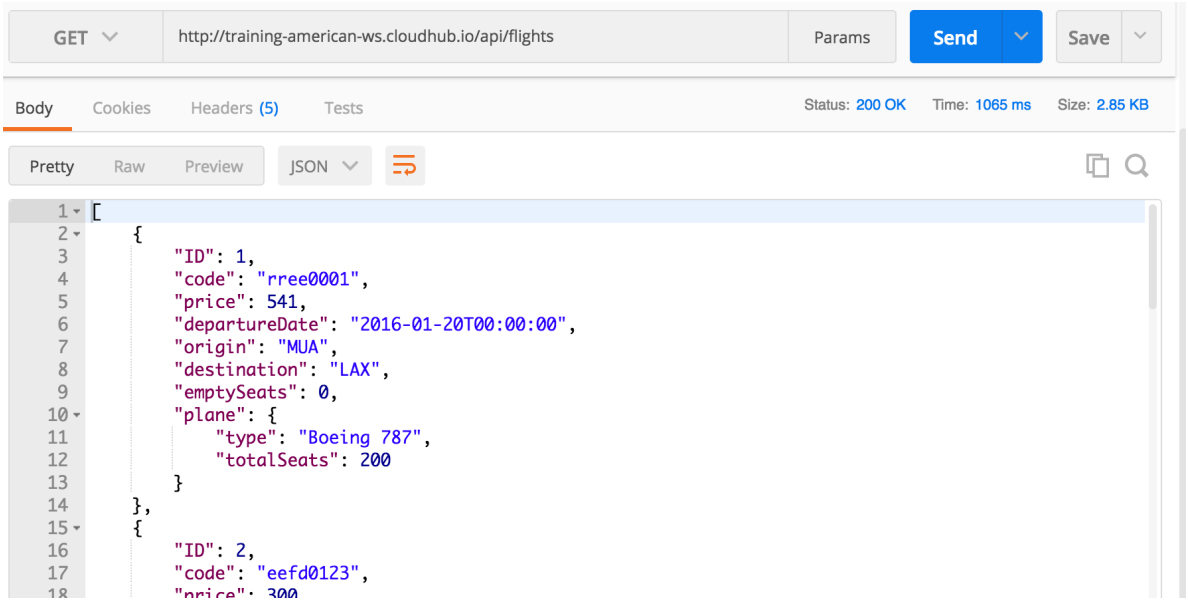


Make GET requests to retrieve data

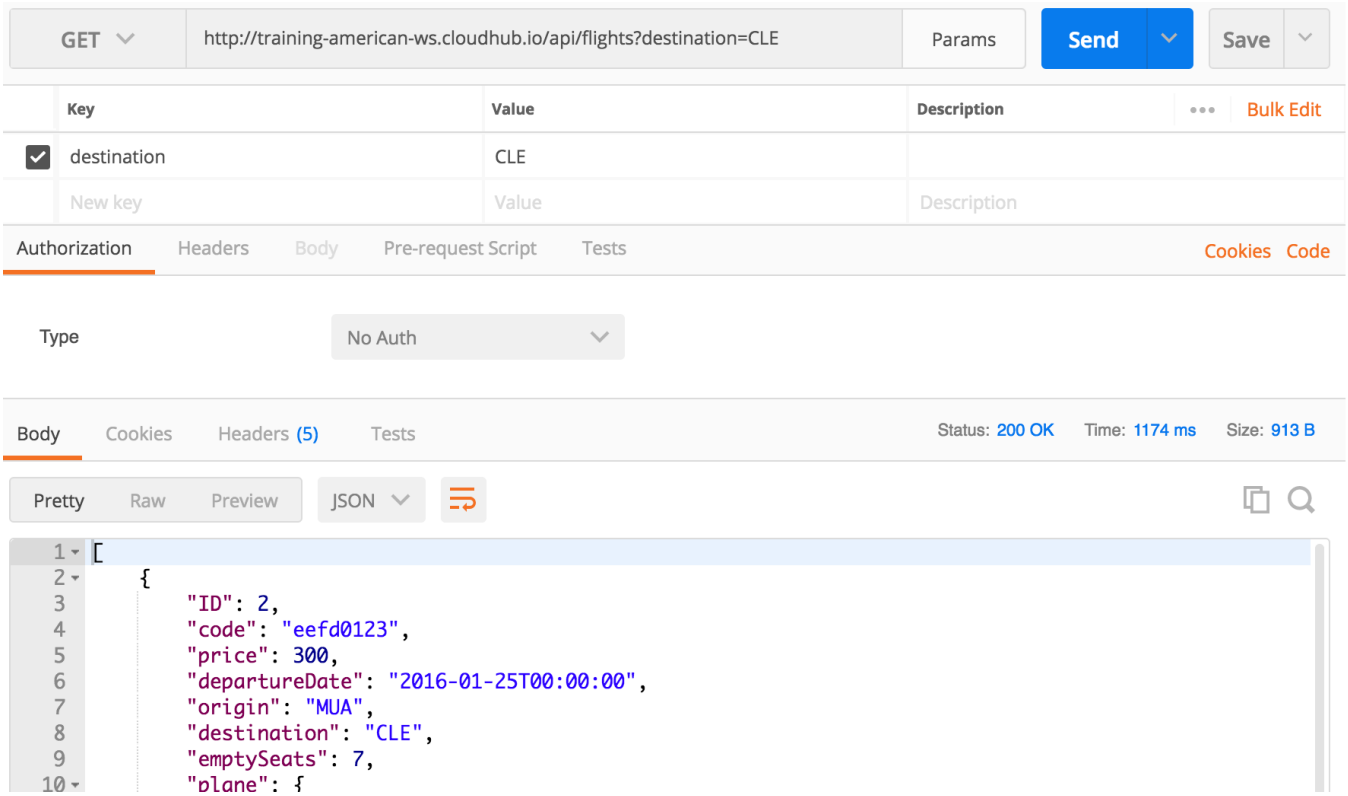
1. Return to or open Postman.
2. Make sure the method is set to GET.
3. Return to the course snippets.txt file.
4. Copy the URL for the American Flights web service: http://training-american-ws.cloudhub.io/api/flights.
5. Return to Postman and paste the URL in the text box that says Enter request URL.



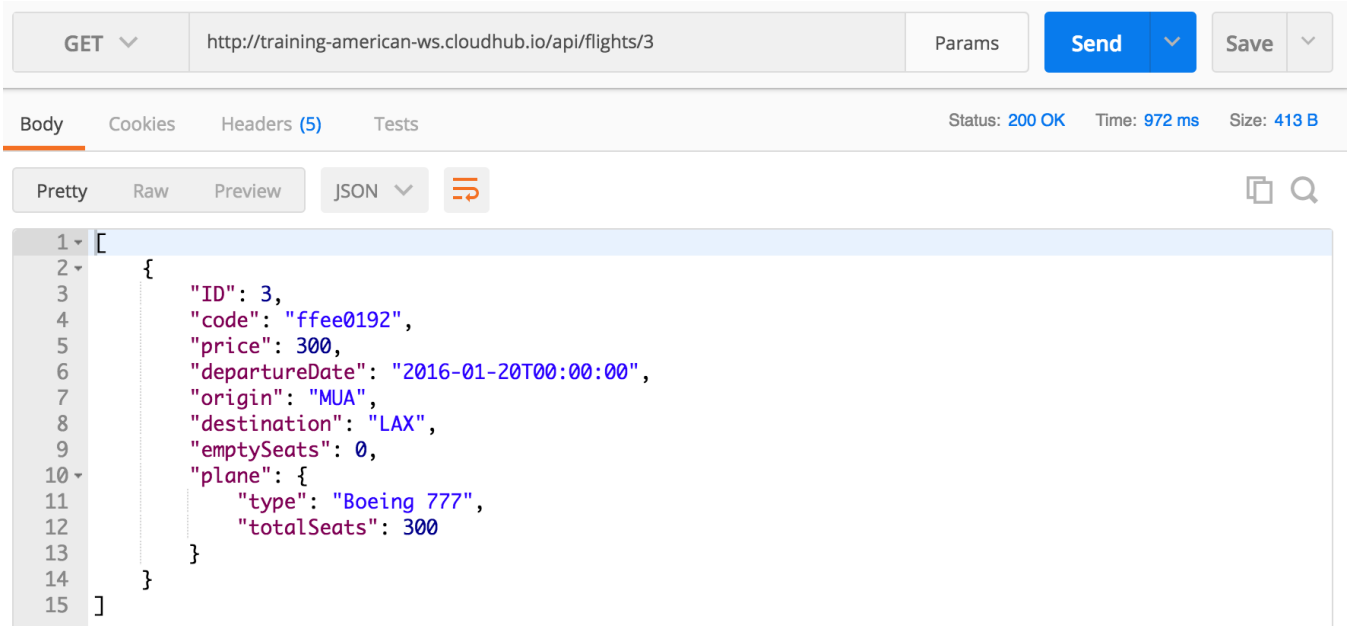
6. Click the Send button; you should get a response.
7. Locate and click the return HTTP status code of 200.
8. Review the response body containing flights to SFO, LAX, and CLE.



9. Click the Params button next to the URL.
10. In the area that appears, set the key to destination and the value to CLE.
11. Click the Send button; you should get just flights to CLE returned.



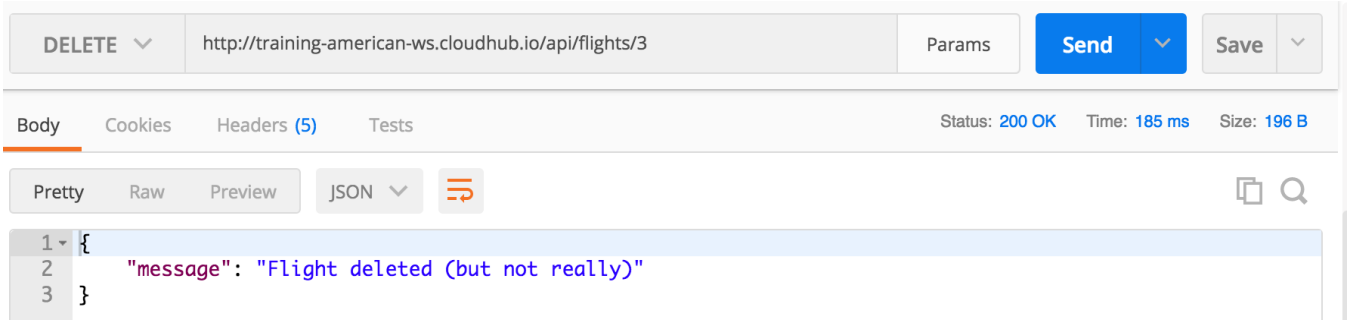
12. Click the X next to the parameter to delete it.
13. Change the request URL to use a uri parameter to retrieve the flight with an ID of 3: http://training-american-ws.cloudhub.io/api/flights/3
14. Click the Send button; you should see only the flight with that ID returned.



Make DELETE requests to delete data

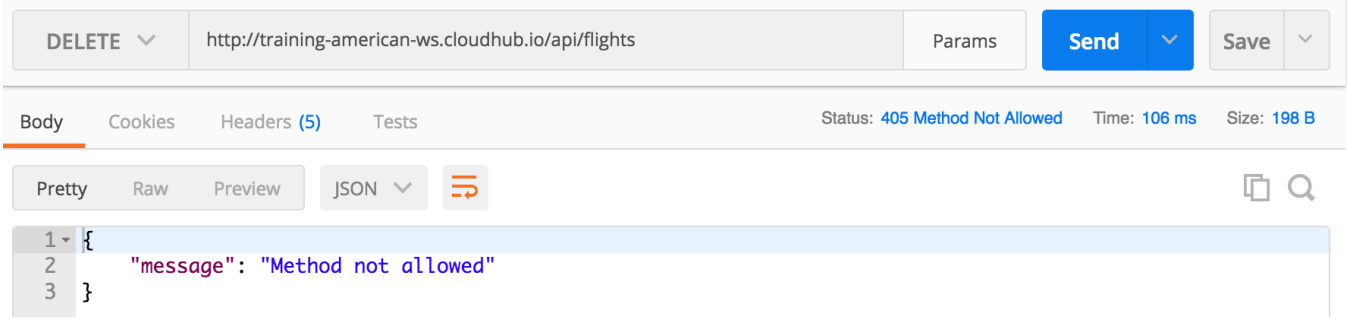
15. Change the method to DELETE.
16. Click the Send button; you should see a 200 response with a message that the Flight was deleted (but not really).

    Note: The database is not actually modified so that its data integrity can be retained for class.



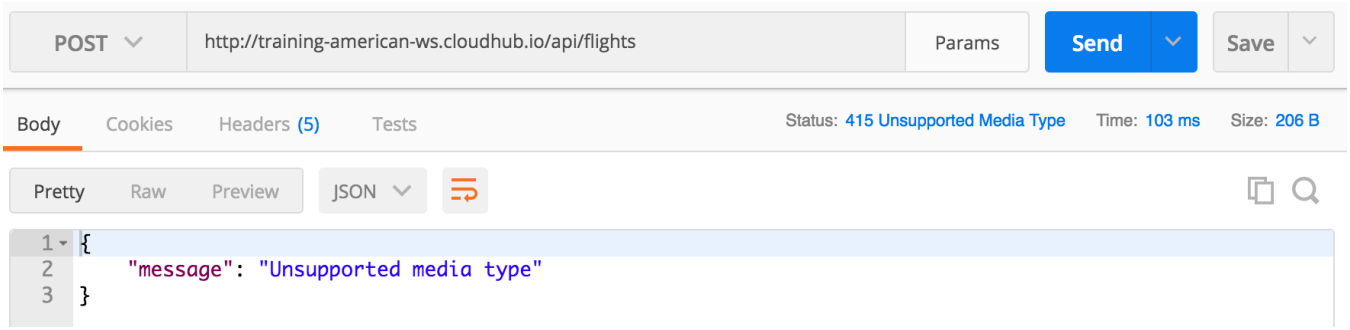
17. Remove the URI parameter from the request: http://training-american-ws.cloudhub.io/api/flights.

18. Click the Send button; you should get a 405 response with a message of method not allowed.

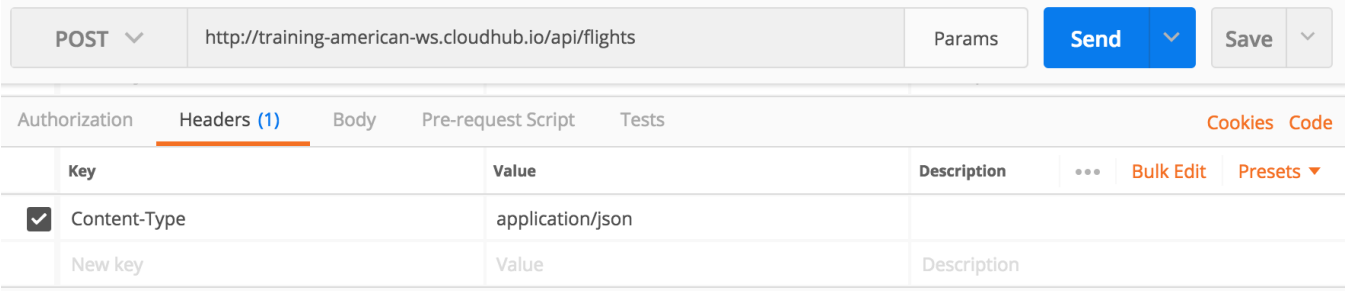


Make a POST request to add data

19. Change the method to POST.
20. Click the Send button; you should get a 415 response with a message of unsupported media type.

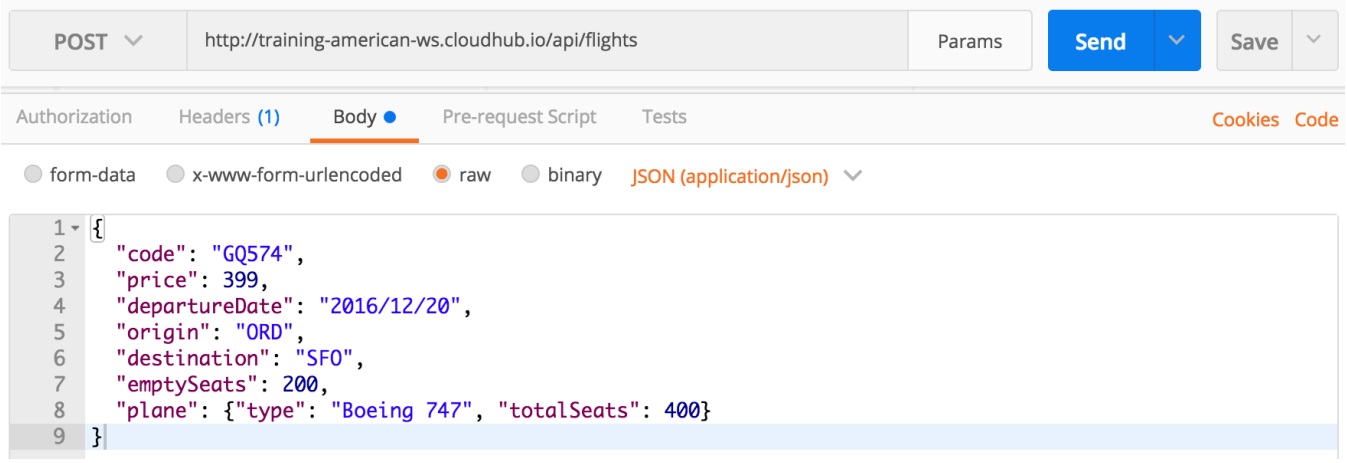


21. Click the Headers link under the request URL.
22. Click in the Headers key field, type C, and then select Content-Type.
23. Click in the Value field, type A, and then select application/json.

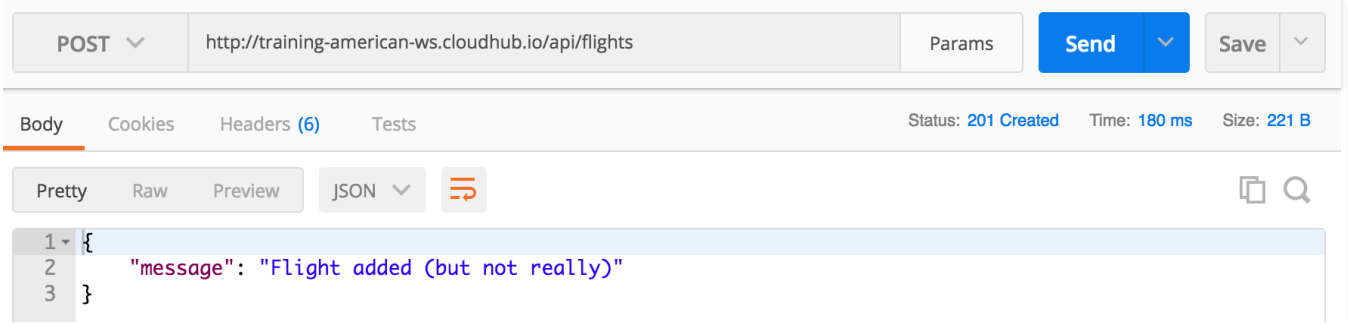


24. Click the Body link under the request URL.
25. Select the raw radio button.
26. Return to the course snippets.txt file and copy the value for American Flights API post body.

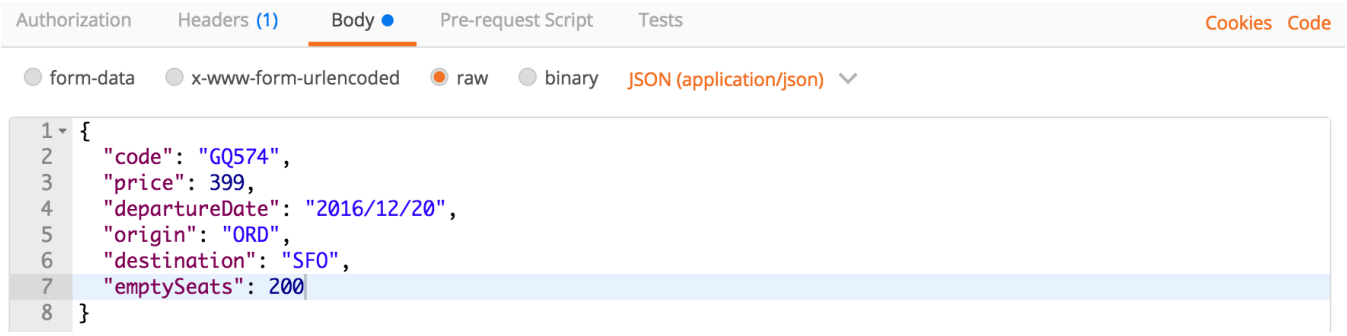
27. Return to Postman and paste the code in the body text area.



28. Click the Send button; you should see a 201 response with the message Flight added (but not really).

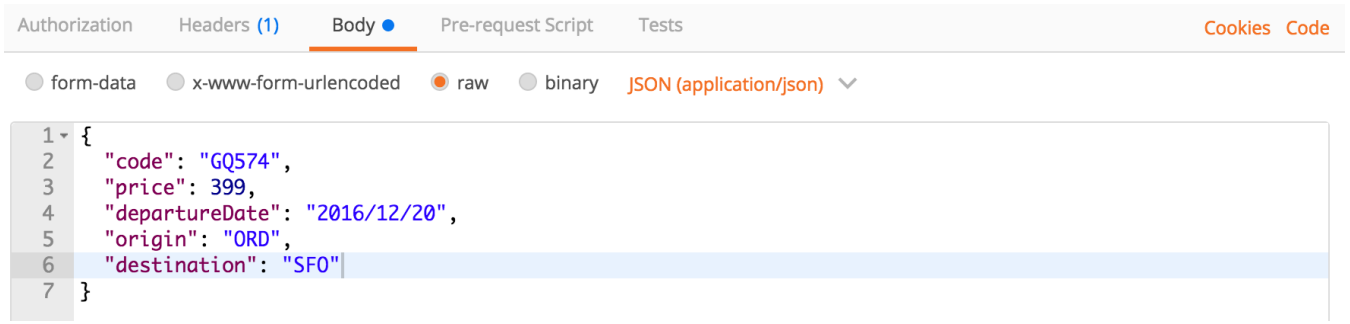


29. Return to the request body and remove the plane field and value from the request body.
30. Remove the comma after the emptySeats key/value pair.

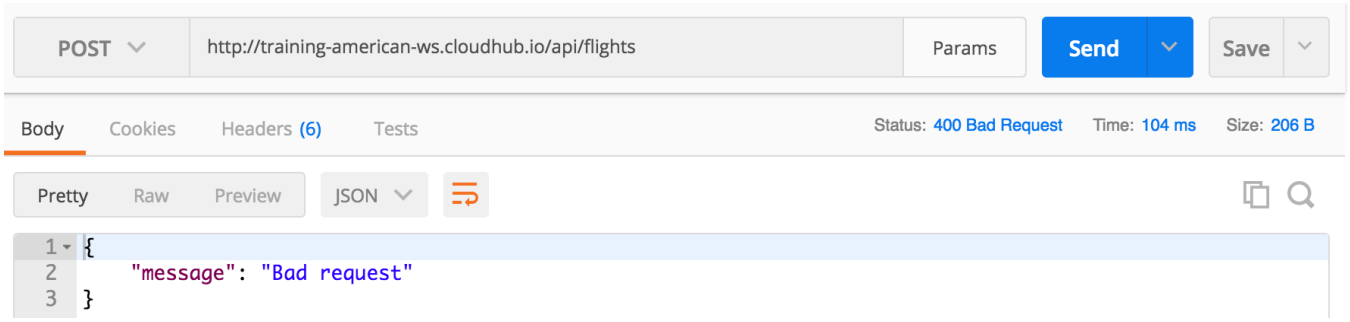


31. Send the request; the message should still post successfully.

32. In the request body, remove the emptySeats key/value pair.
33. Delete the comma after the destination key/value pair.



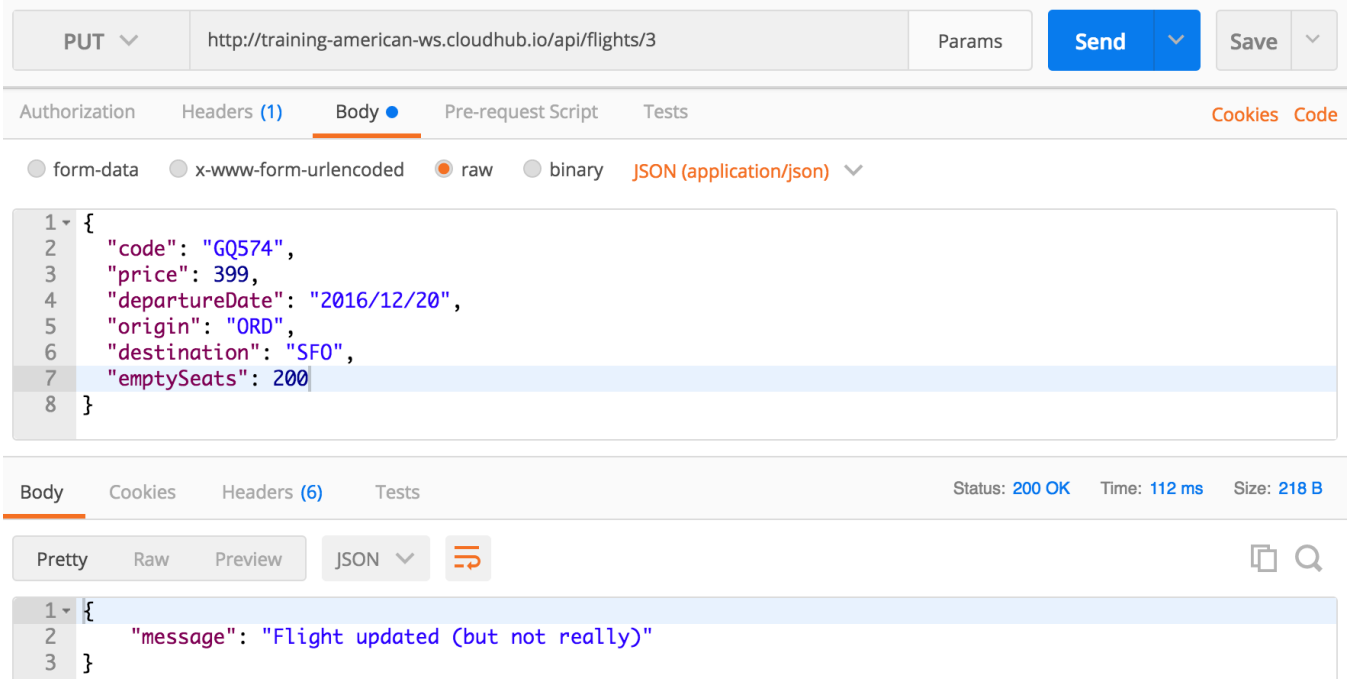
34. Send the request; you should see a 400 response with the message Bad request.



Make a PUT request to update data

35. Change the method to PUT.
36. Add a flight ID to the URL to modify a particular flight.
37. Click the Send button; you should get a bad request message.
38. In the request body field, press Cmd+Z or Ctrl+Z so the emptySeats field is added back.

39. Send the request; you should see the response Flight updated (but not really).

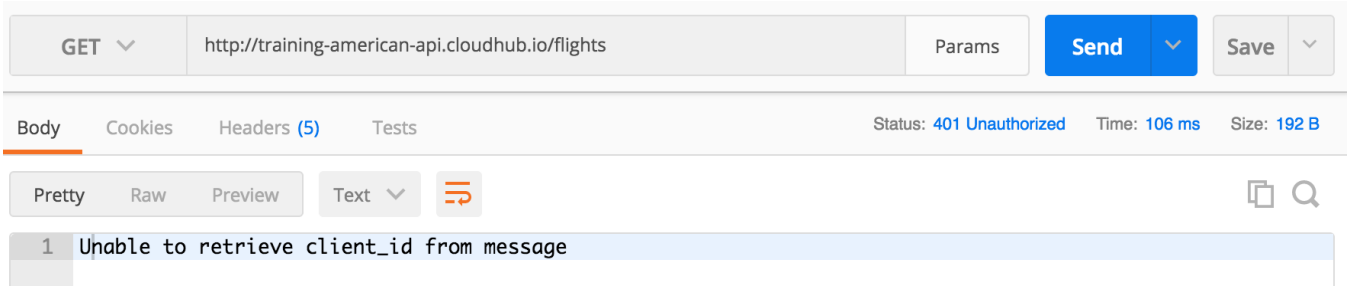


Make a request to a secured API

40. Change the method to GET.
41. Change the request URL to http://training-american-api.cloudhub.io/flights/3.

    Note: The -ws in the URL has been changed to -api and the /api removed.

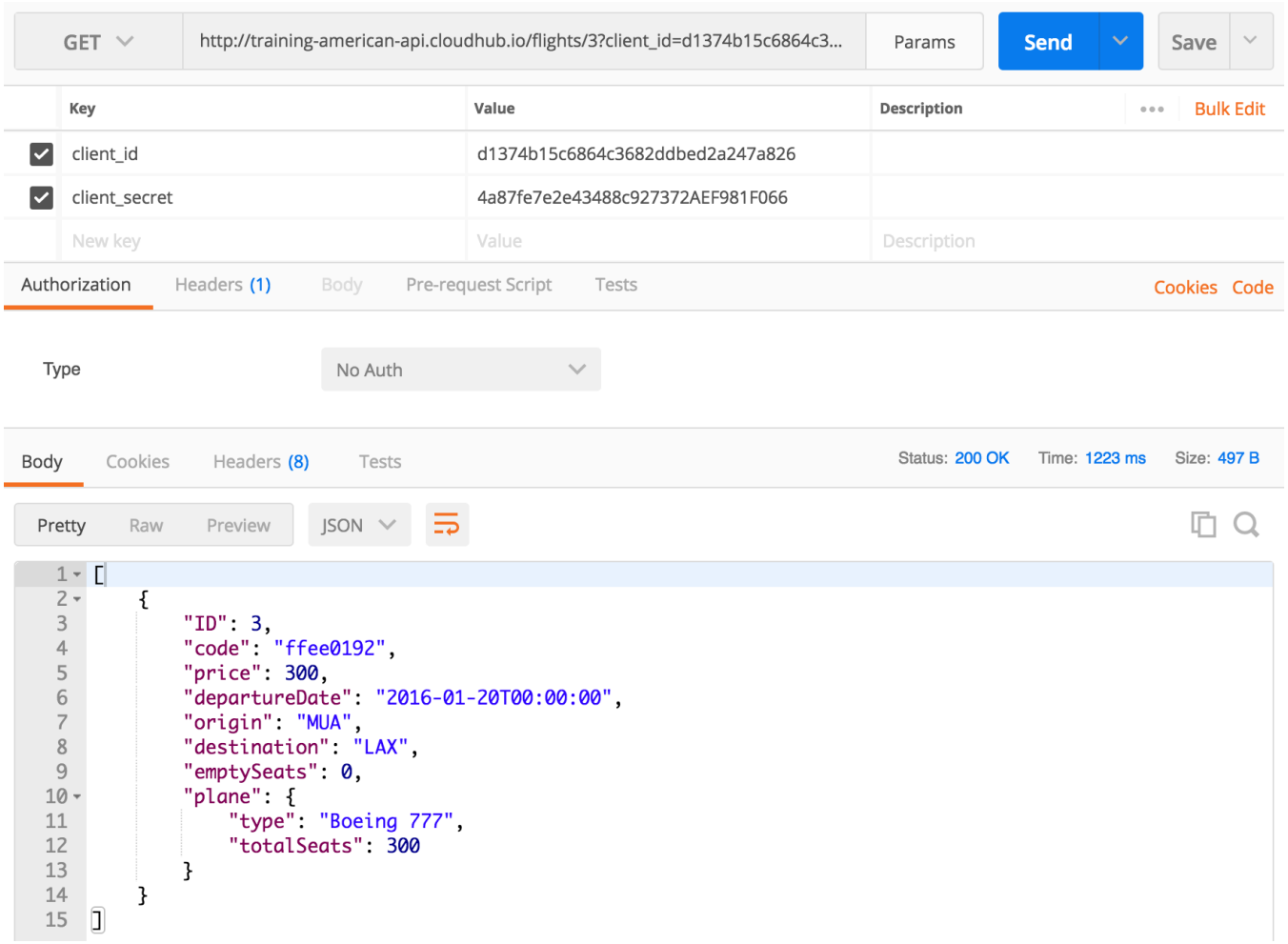
42. Click the Send button; you should get a message about a missing client_id.



43. Return to the course snippets.txt file and copy the value for the American Flights API client_id.
44. Return to Postman and add a request parameter called client_id.
45. Set client_id to the value you copied from the snippets.txt file.
46. Return to the course snippets.txt file and copy the value for the American Flights API client_secret.
47. Return to Postman and add a request parameter called client_secret.

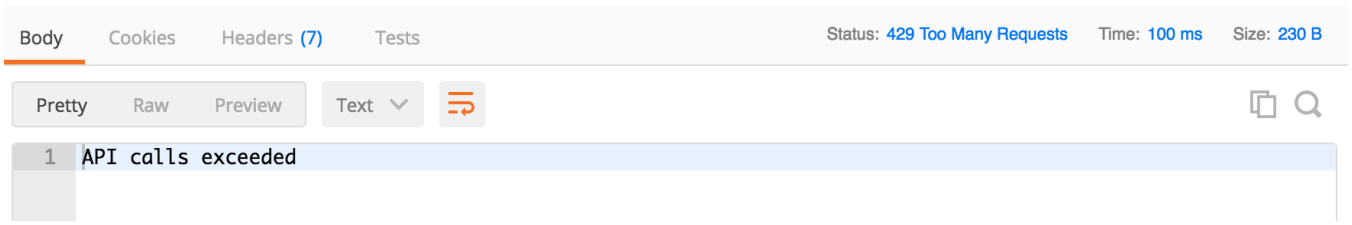
48. Set client_secret to the value you copied from the snippets.txt file.
49. Click the Send button; you should get data for flight 3 again.

    Note: The API service level agreement (SLA) for the application with this client ID and secret has been set to allow three API calls per minute.

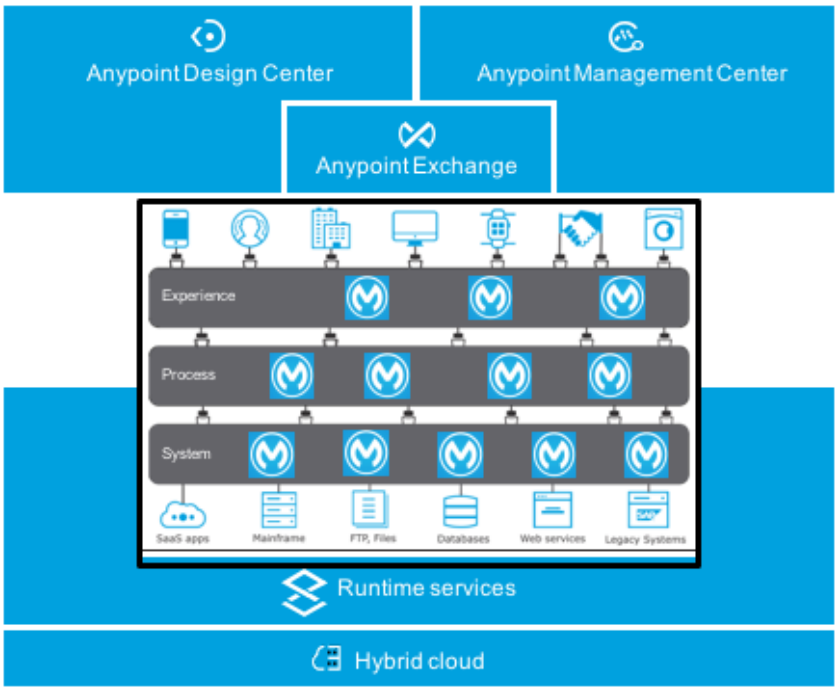


50. Click the Send button three more times; you should get a 429 response and an API calls exceeded message.

    Note: The API service level agreement (SLA) for the application with this client ID and secret has been set to allow three API calls per minute.



Module 2: Introducing Anypoint Platform



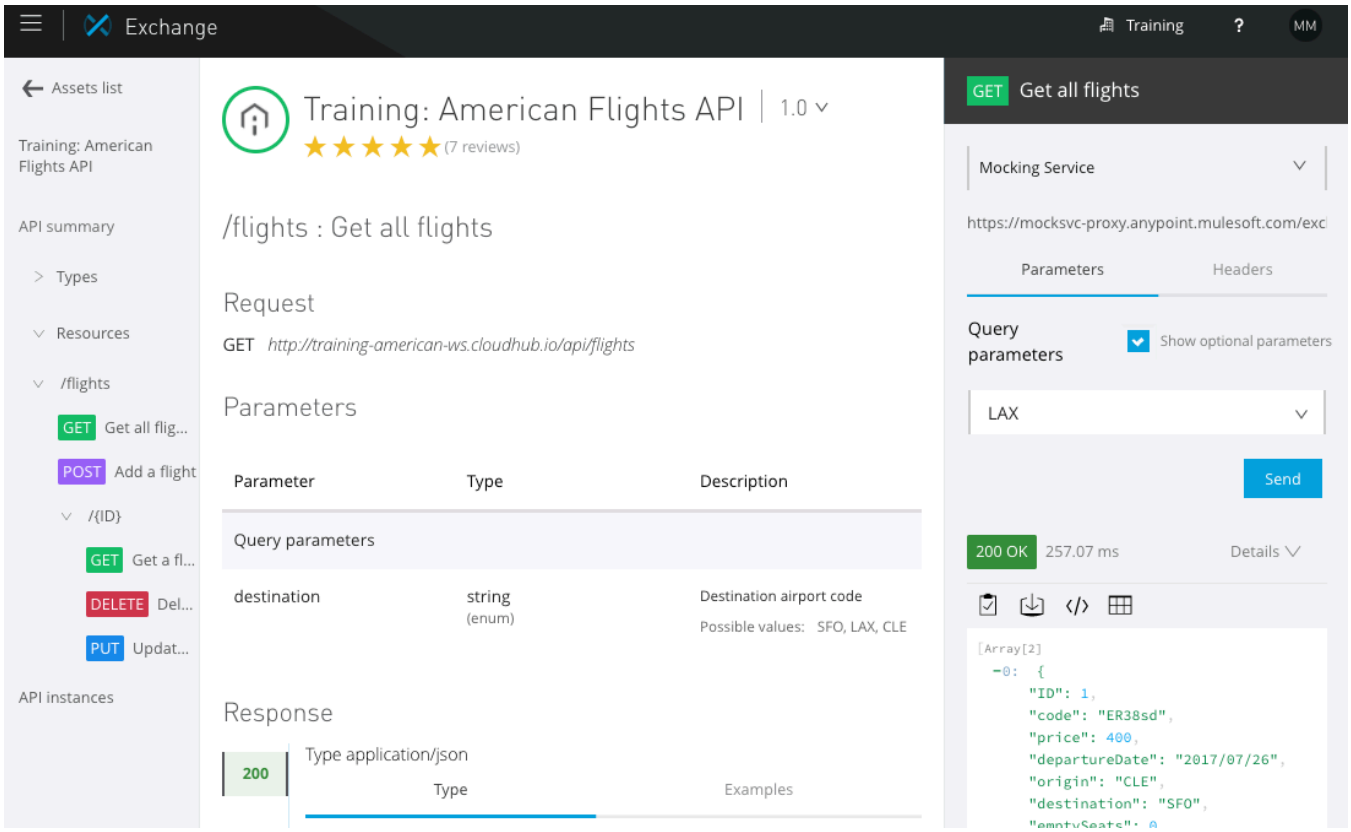
At the end of this module, you should be able to:

- Identify all the components of Anypoint Platform.
- Describe the role of each component in building application networks.
- Navigate Anypoint Platform.
- Locate APIs and other assets needed to build integrations and APIs in Anypoint Exchange.
- Build basic integrations to connect systems using flow designer.

Walkthrough 2-1: Explore Anypoint Platform and Anypoint Exchange

In this walkthrough, you get familiar with the Anypoint Platform web application. You will:

- Explore Anypoint Platform.
- Browse Anypoint Exchange.
- Review an API portal for a REST API in Exchange.
- Discover and make calls to the Training: American Flights API.



Return to Anypoint Platform

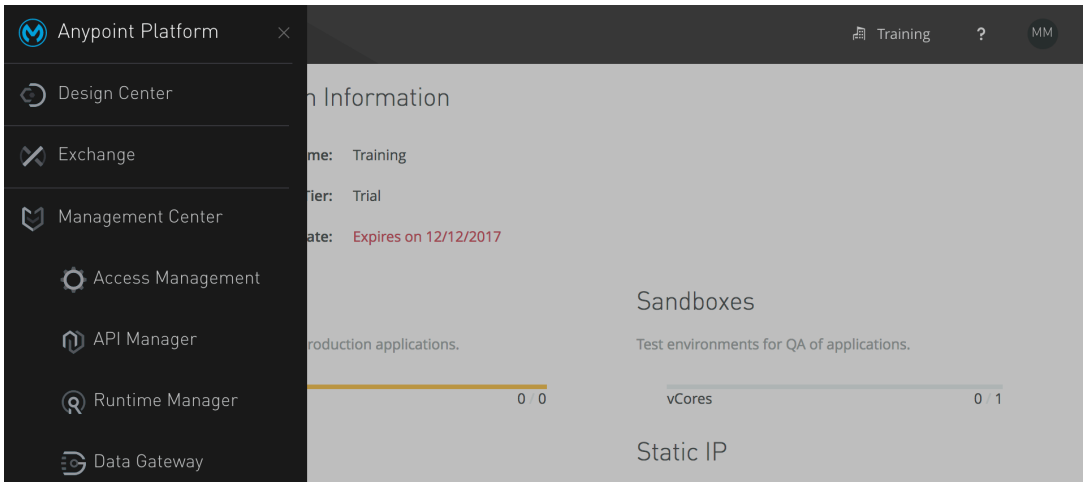
1. Return to Anypoint Platform in a web browser.

   Note: If you closed the browser window or logged out, return to https://anypoint.mulesoft.com and log in.

2. Click the menu button located in the upper-left in the main menu bar.

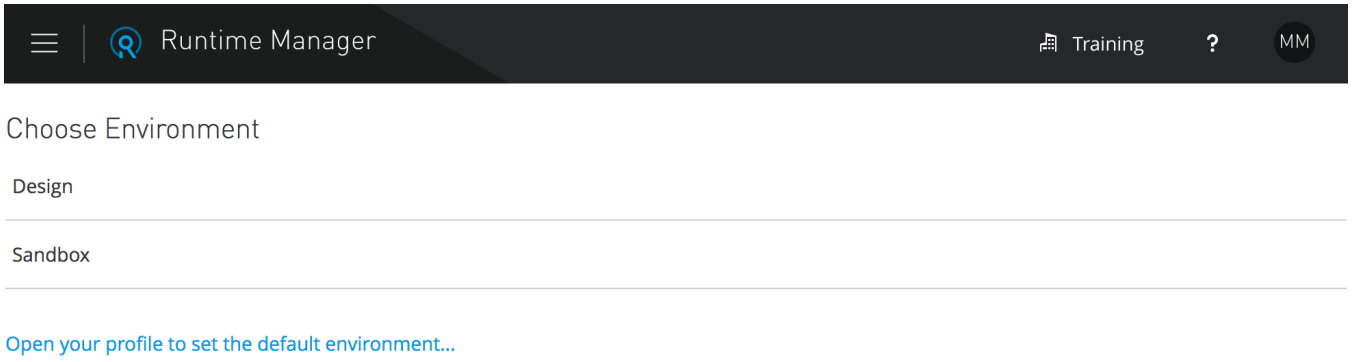
3. In the menu that appears, select Anypoint Platform; this will return you to the home page.

   Note: This will be called the main menu from now on.



Explore Anypoint Platform

4. In the main menu, select Access Management.
5. In the main menu, select Design Center.
6. In the main menu, select Runtime Manager.
7. If you get a Choose environment page, select Design.

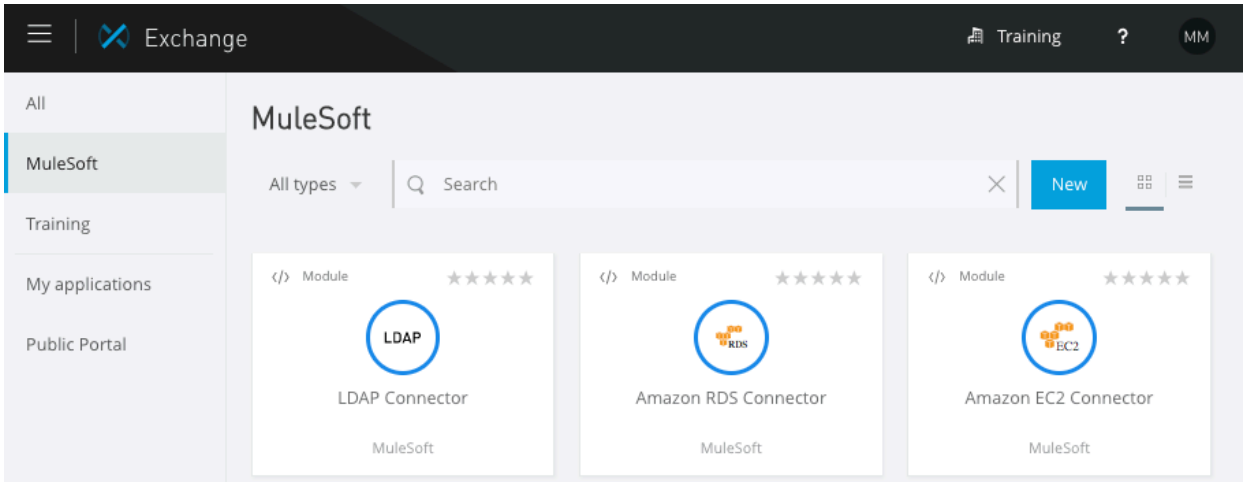


8. In the main menu, select API Manager.

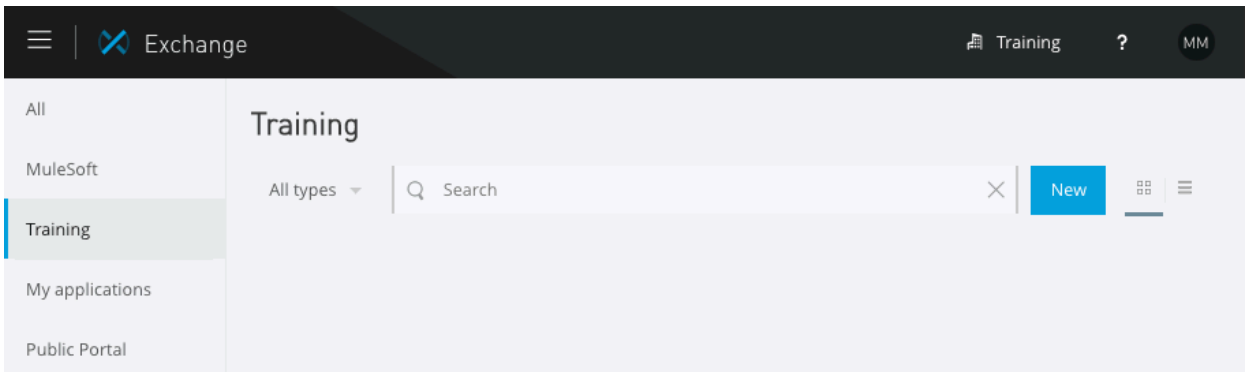
Explore Anypoint Exchange

9. In the main menu, select Exchange.

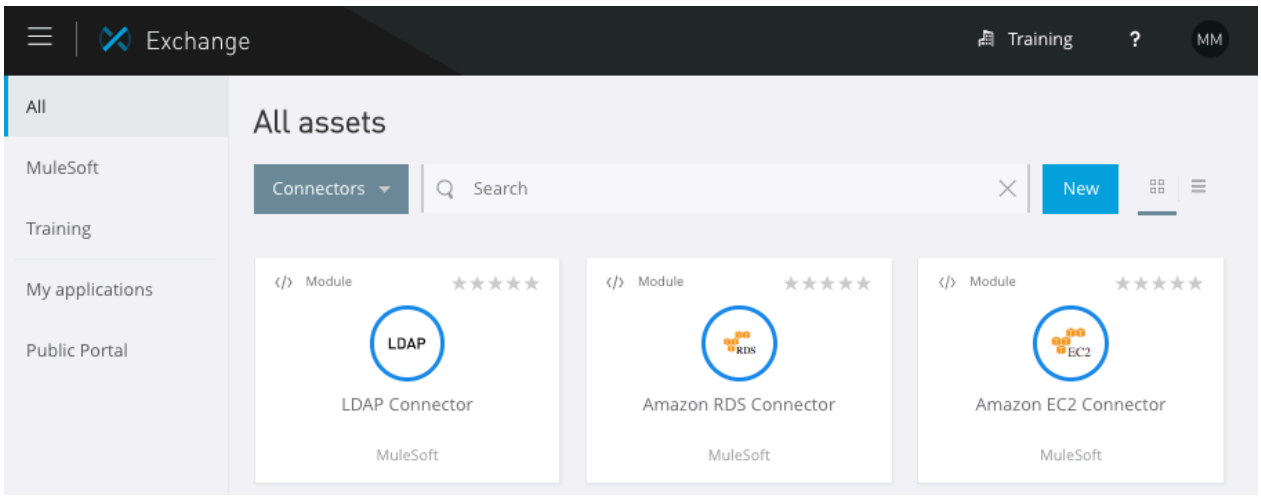
10. In the left-side navigation, click MuleSoft; you should see all the content in the public Exchange.



11. In the left-side navigation, click the name of your organization beneath MuleSoft (Training in the screenshots); you should now see only the content in your private Exchange, which is currently empty.

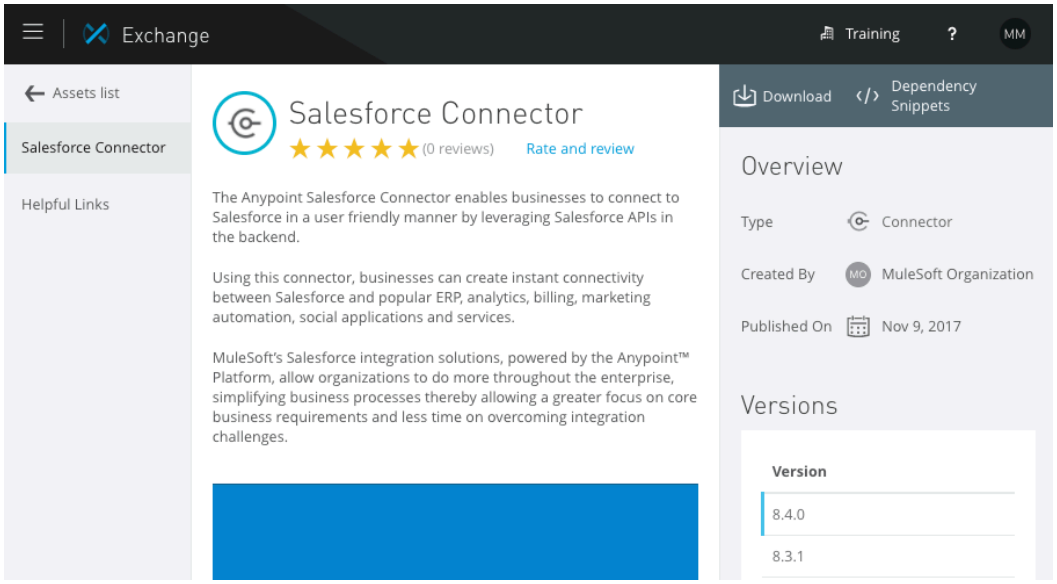


12. In the left-side navigation, click All.
13. In the types menu, select Connectors.



14. Click one of the connectors and review its information.
15. In the left-side navigation, click the Assets list link.
16. Select the Salesforce connector (or search for it if it is not shown) and review its details.

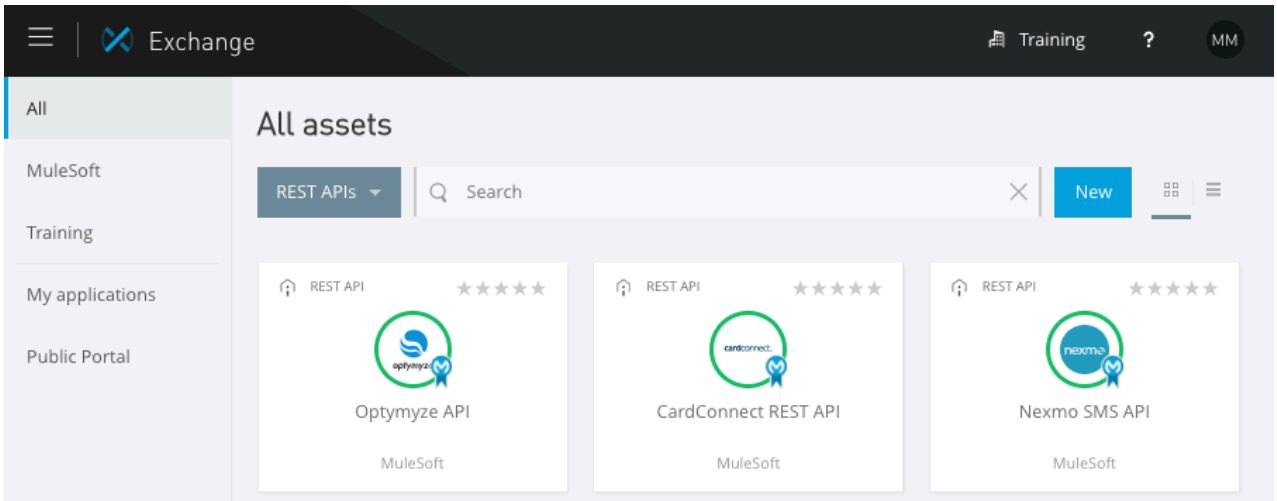
    Note: The Salesforce connector is used in the Development Fundamentals course.



17. In the left-side navigation, click Assets list.
18. In the types menu, select Templates (and remove anything from the search field).

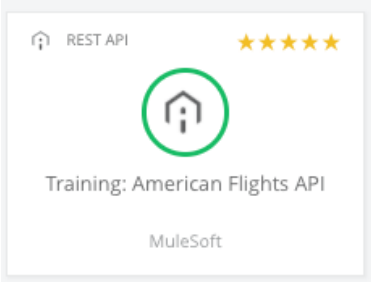
Browse REST APIs in Anypoint Exchange

19. In the types menu, select REST APIs.
20. Browse the APIs.

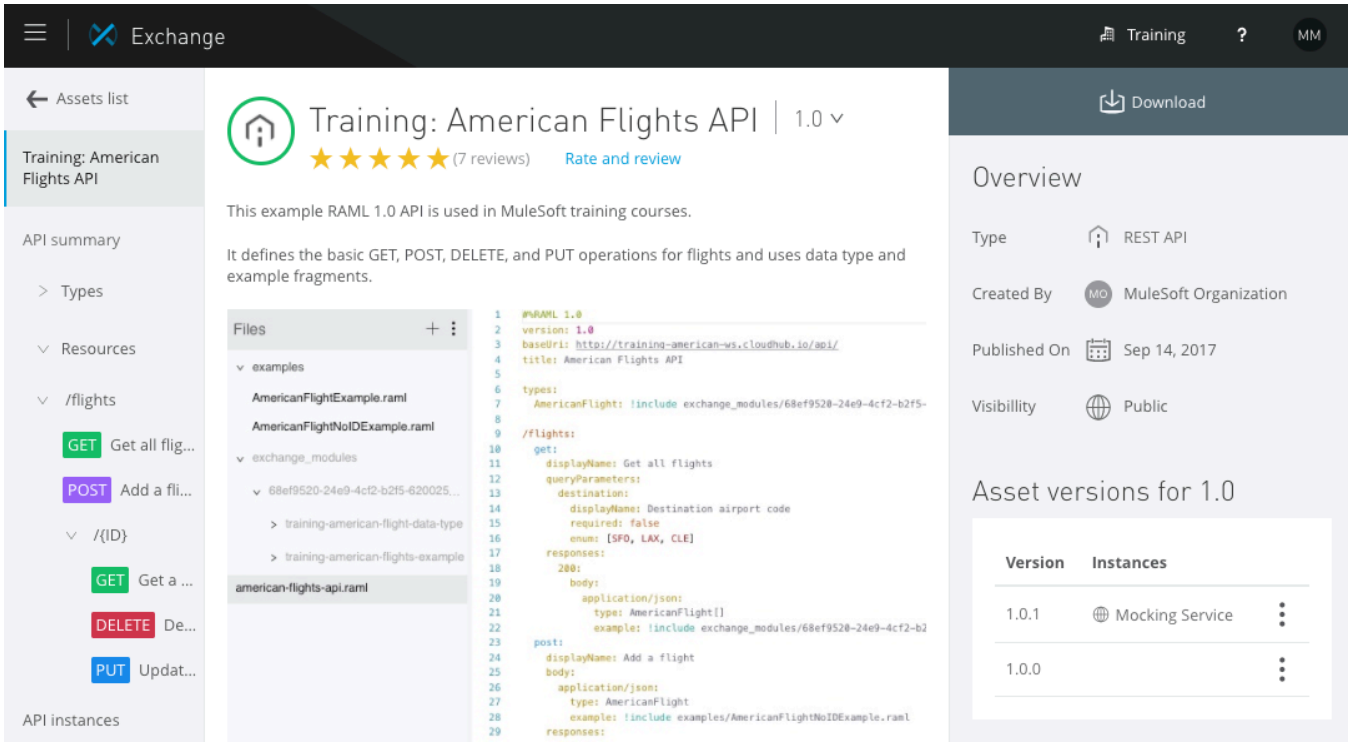


Discover and review the API portal for the Training: American Flights API

21. Locate and click the Training: American Flights API.

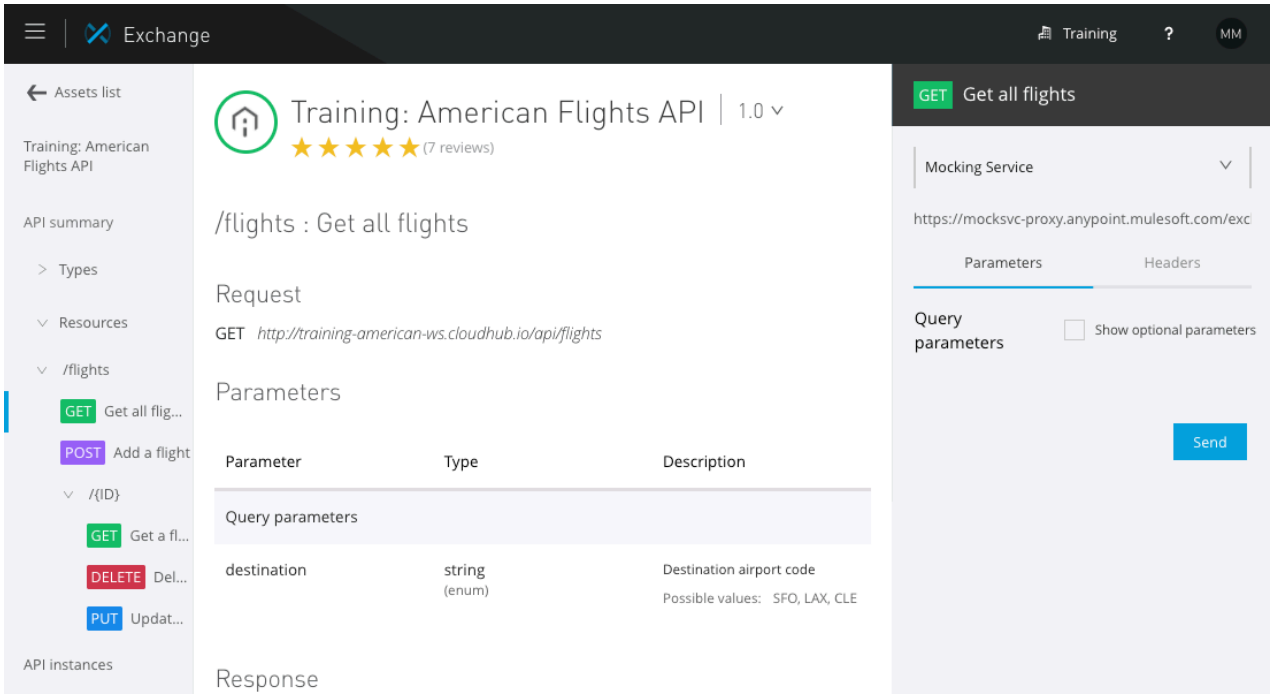


22. Review the API portal.

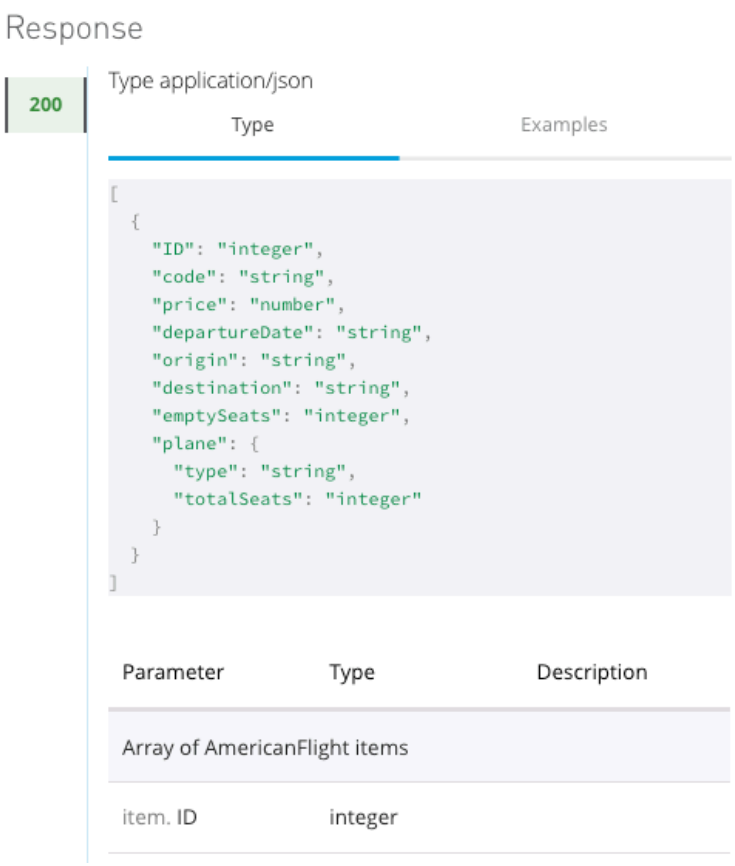


23. In the left-side navigation, expand and review the list of available resources.
24. Click the GET link for the /flights resource.

25. On the /flights: Get all flights page, review the information for the optional query parameter.



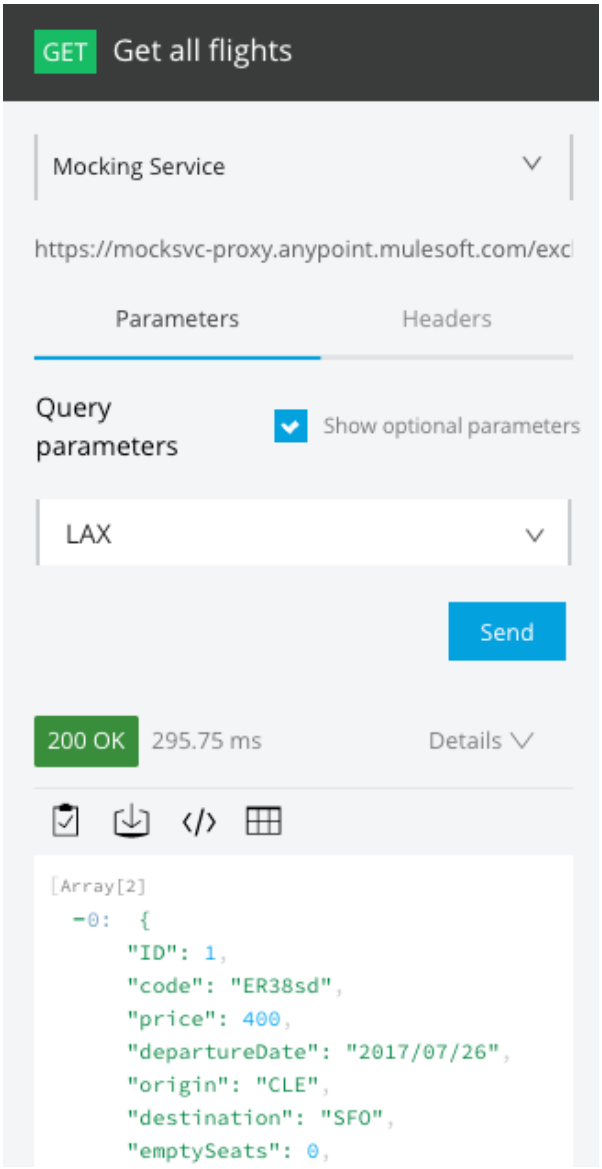
26. Scroll down and locate the type and details for the data returned from a call.



27. Select the Examples tab; you should see an example response.

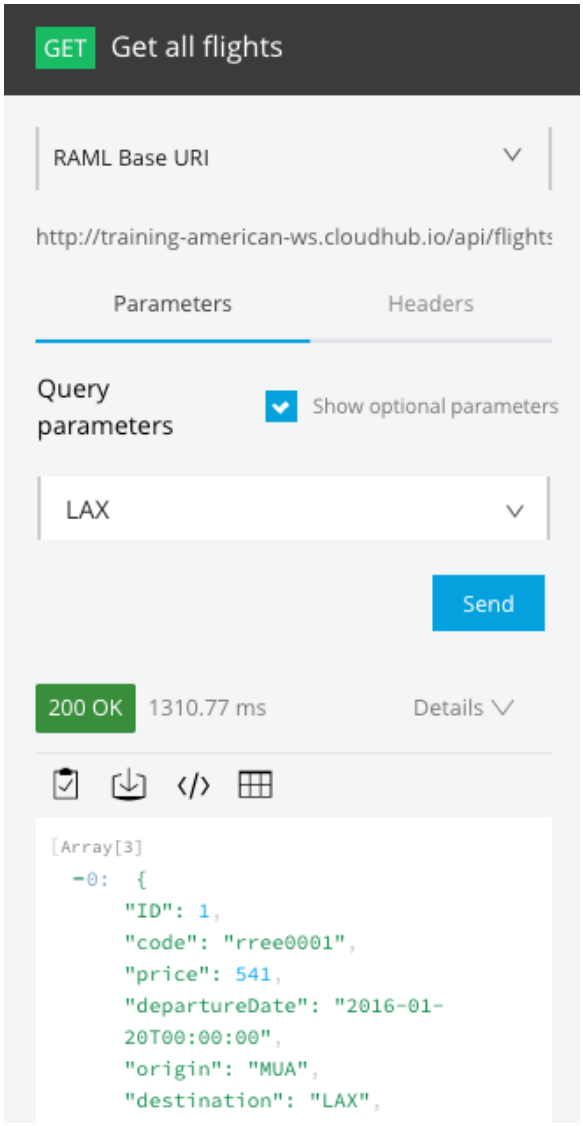
Use the API console to make calls to the Training: American Flights API

28. Scroll up and locate the API console on the right-side on the page.
29. Select to show optional query parameters and select a value.
30. Select a destination and click Send; you should get the example data returned using the mocking service.



31. In the API console, change the API instance from Mocking Service to RAML Base URI.

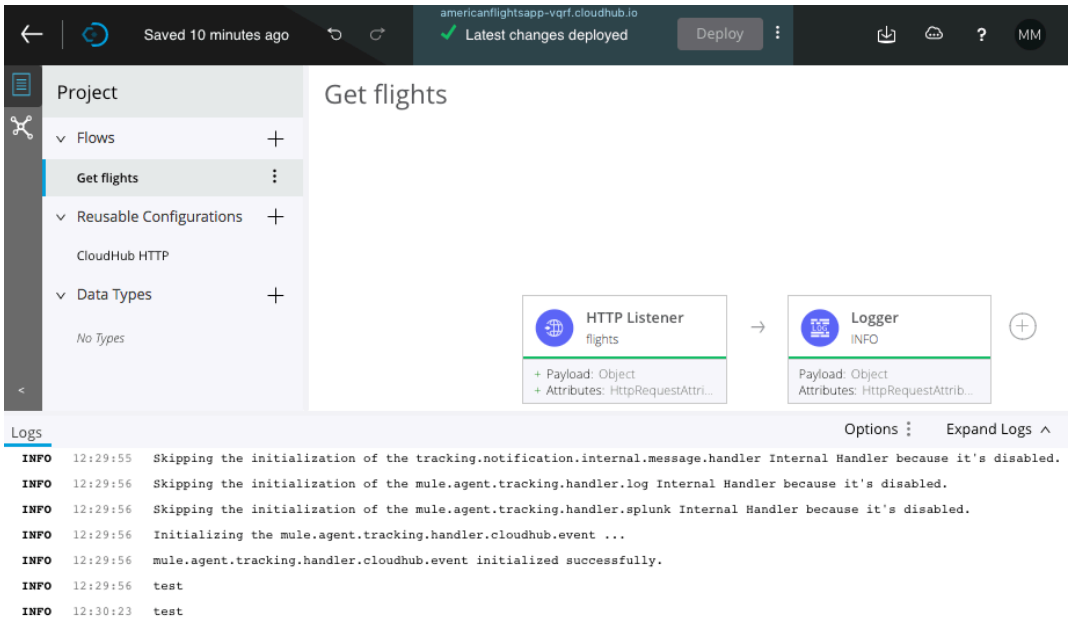
32. Click Send again; you should get results from the actual API implementation for the destination you selected.



Walkthrough 2-2: Create a Mule application with flow designer

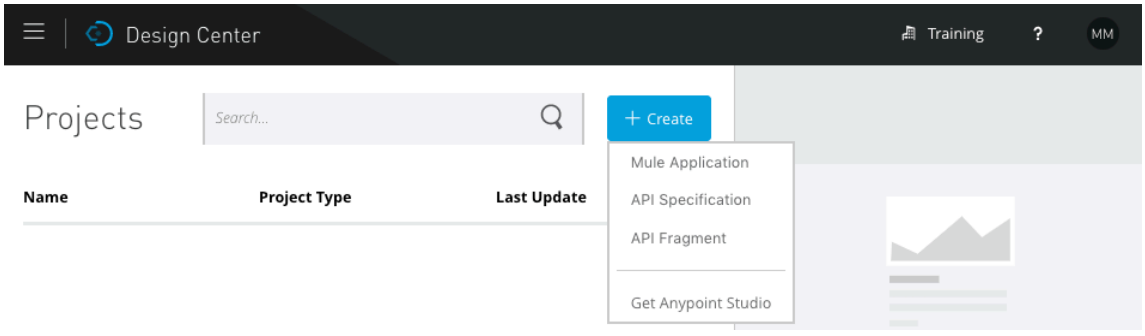
In this walkthrough, you build, run, and test a basic Mule application with flow designer. You will:

- Create a new Mule application project in Design Center.
- Create an HTTP trigger for a flow in the application.
- Add a Logger component.
- Deploy, run, and test the application.
- View application information in Runtime Manager.



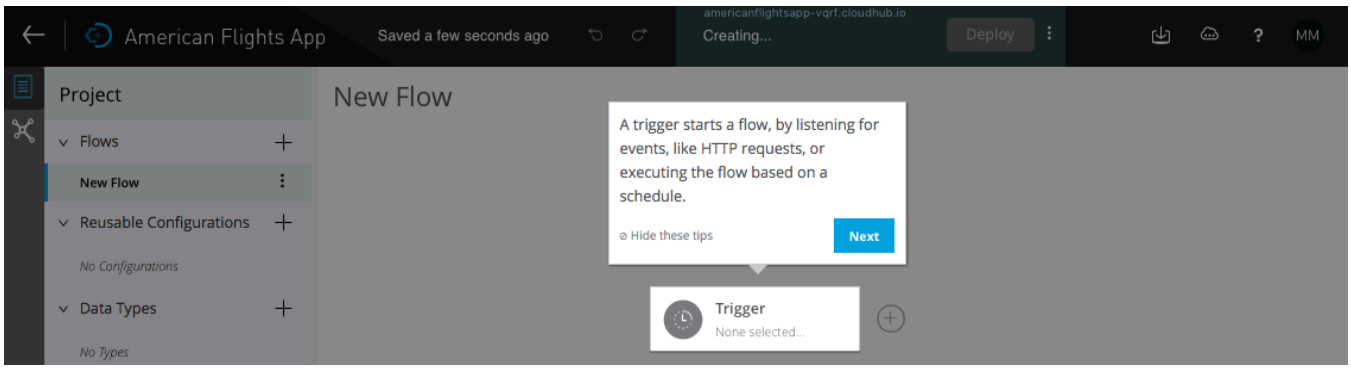
Create a Mule application project in Design Center

1. Return to Anypoint Platform.
2. In the main menu, select Design Center.
3. Click the Create button and select Mule Application.

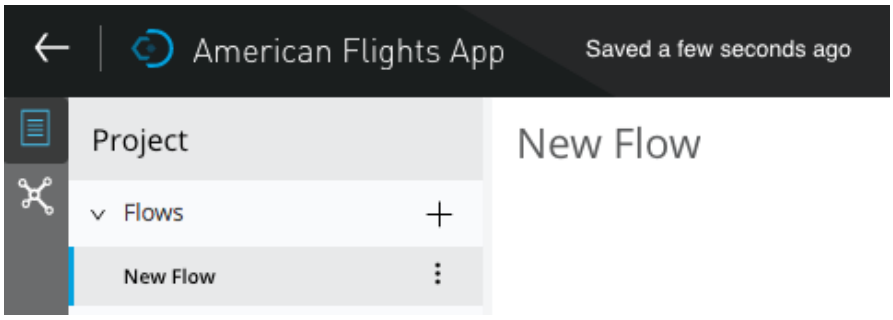


4. In the New Mule Application dialog box, set the project name to American Flights App.

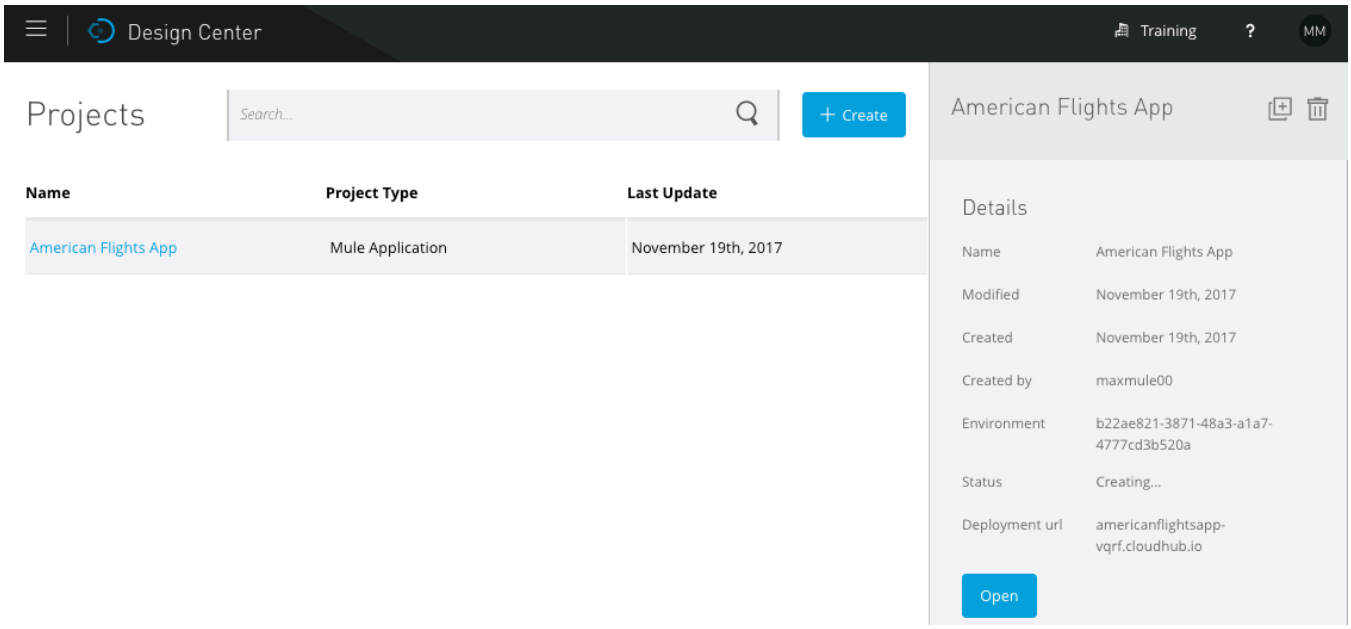
5. Click Create; flow designer should open.



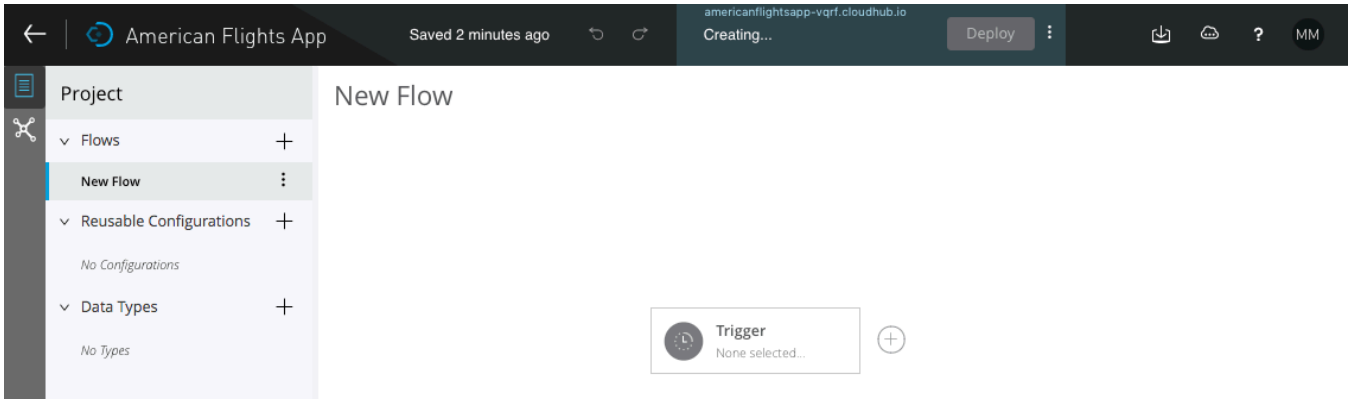
6. In the pop-up box in flow designer, click Hide these tips.
7. Click the arrow icon in the upper-left corner; you should return to the Design Center.



8. In the Design Center project list, click the row containing the American Flights App; you should see information about the project displayed on the right side of the page.

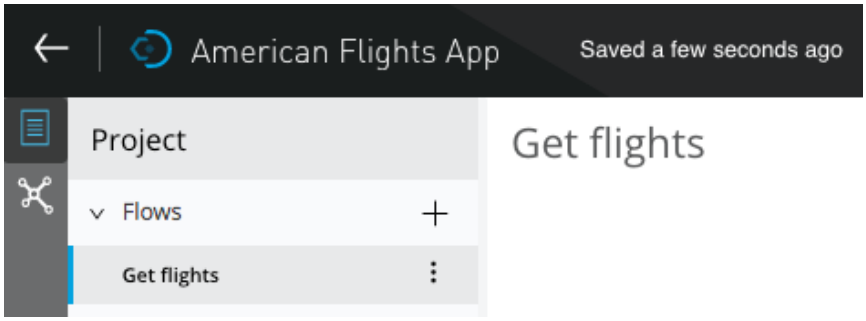


9. Click the Open button or click the American Flights App link in the project list; the project should open in flow designer.



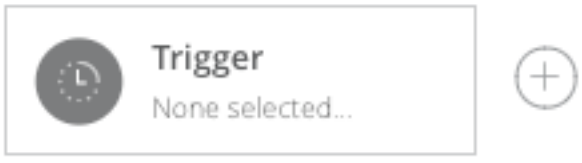
Rename the flow

10. Locate New Flow in the project explorer.
11. Click its option menu and select Rename.
12. In the Rename Flow dialog box, set the name to Get flights and click OK.

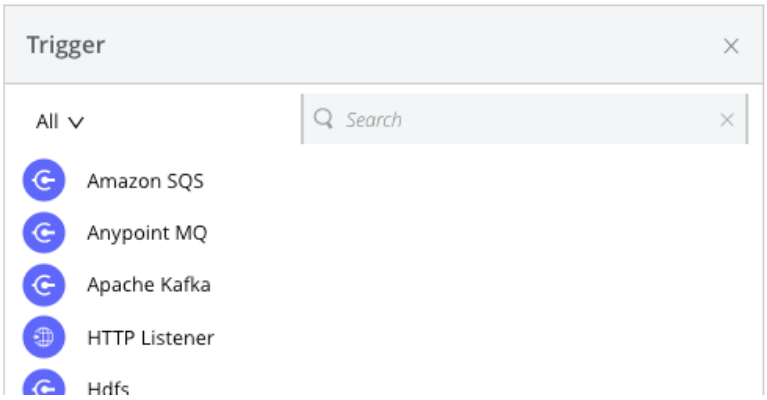


Create an HTTP trigger for a flow in the application

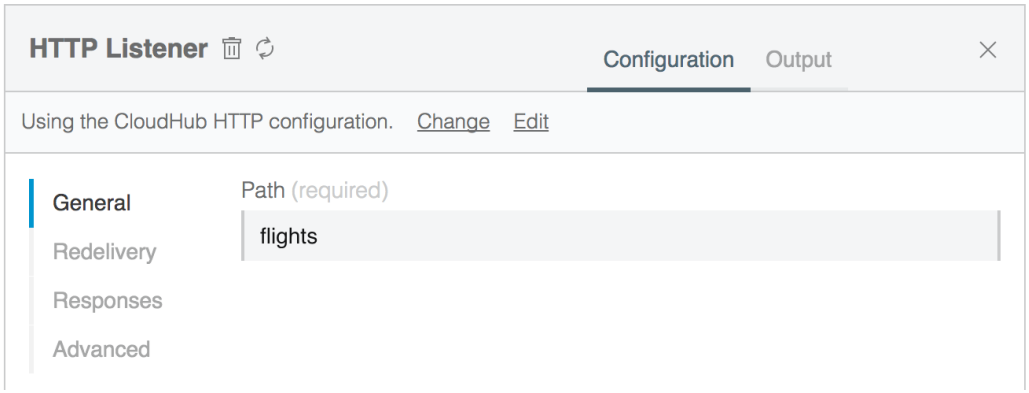
13. In flow designer, click the Trigger card.



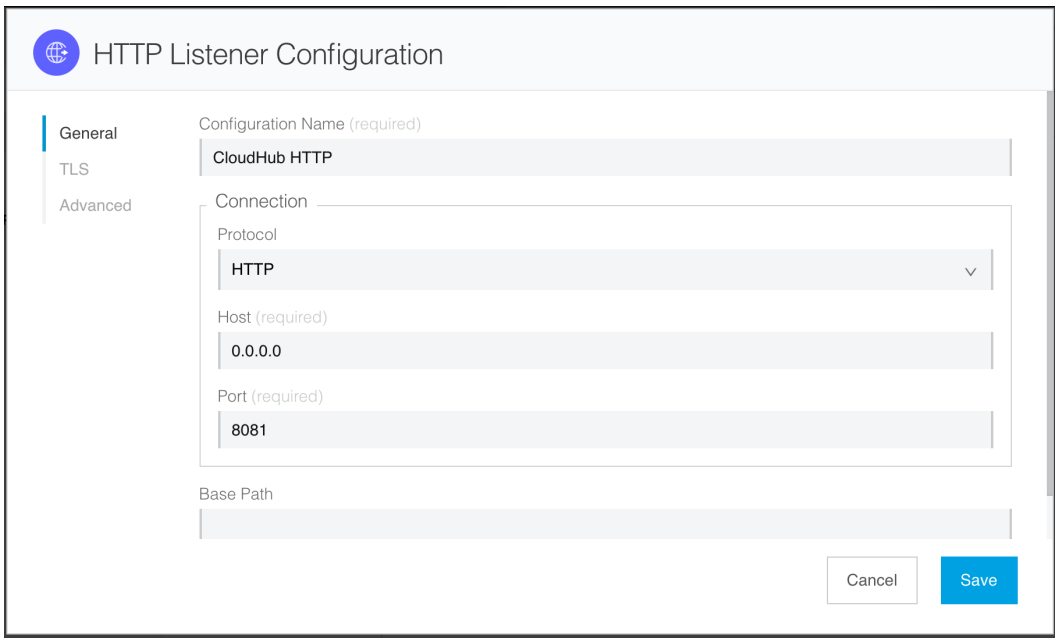
14. In the Trigger card, select HTTP Listener.



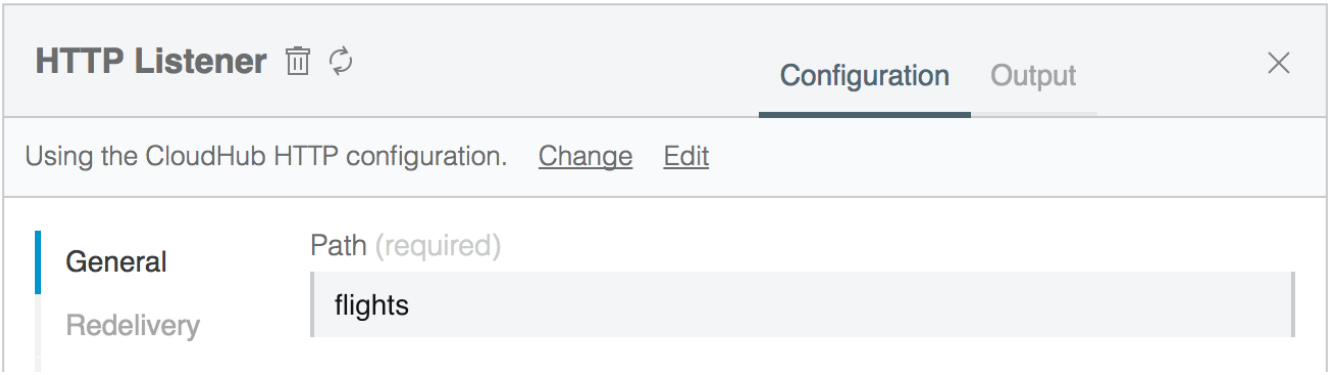
15. In the HTTP Listener dialog box, set the path to flights.



16. Click the Edit link for the CloudHub HTTP configuration.
17. In the HTTP Listener Configuration dialog box, review the information and click Cancel.

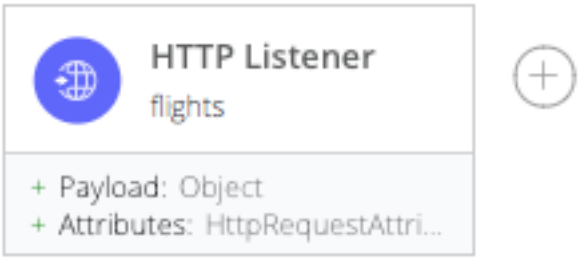


18. In the HTTP Listener dialog box, click the close button in the upper-right corner.

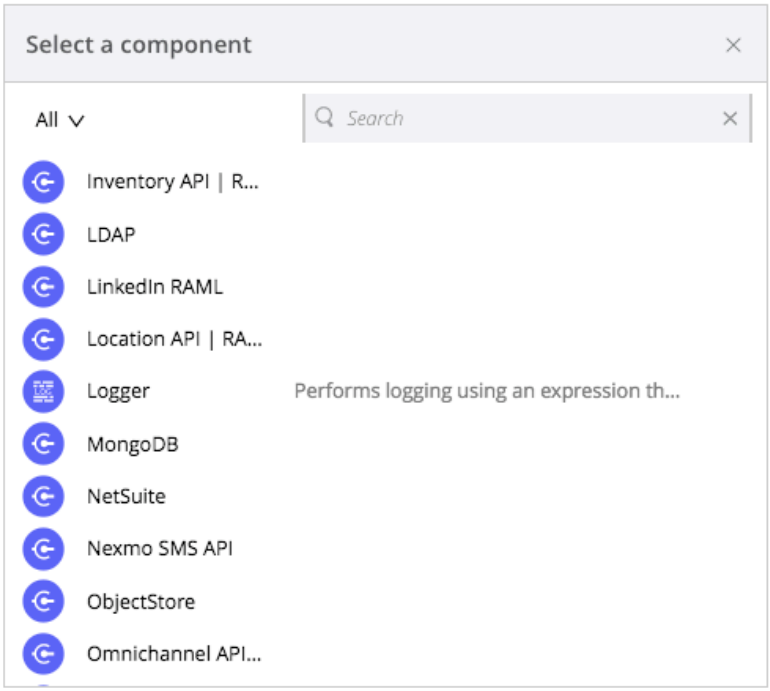


Add a Logger

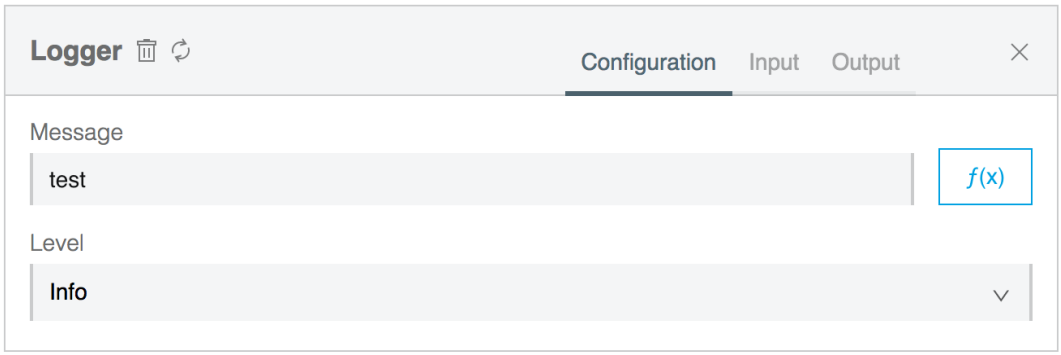
19. Click the add button next to the HTTP Listener card.



20. In the Select a component dialog box, select Logger.



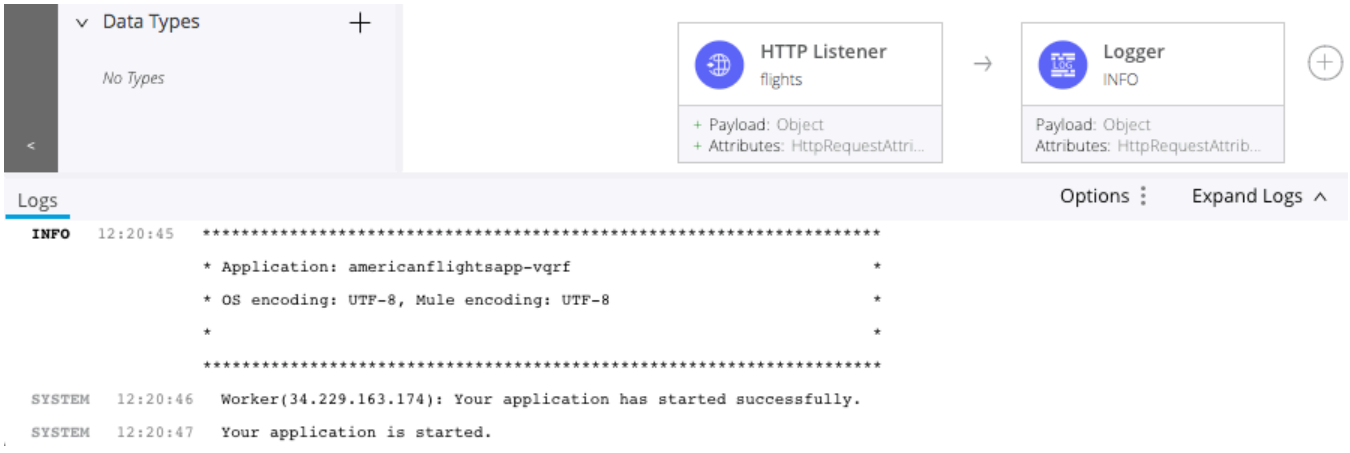
21. In the Logger dialog box, set the message to test.



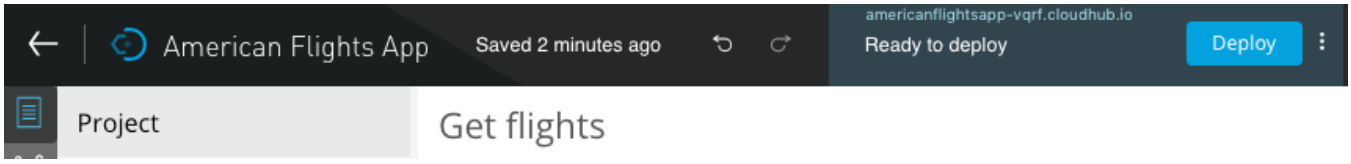
22. Close the card.
23. Notice that there are gray lines across both cards.

Deploy the application

24. Click the Logs tab located in the lower-left corner of the window; you should see that your application is already started.



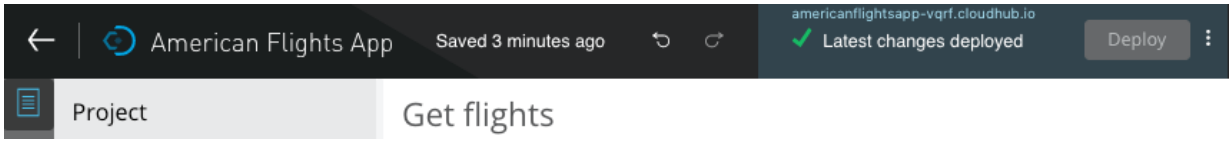
25. Locate the application status in the main menu bar; it should say Ready to deploy.



26. Look at the generated URL for the application.

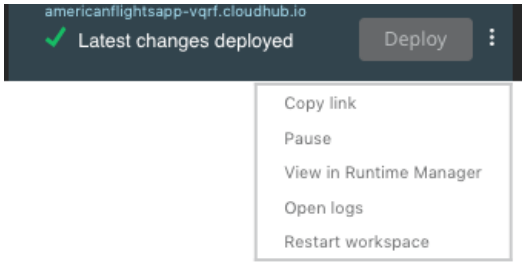
    Note: The application name is appended with a four-letter suffix to guarantee that it is unique across all applications on CloudHub.

27. Click the Deploy button.
28. Watch the application status; it should change from Ready to Deploying then to Latest changes deployed.



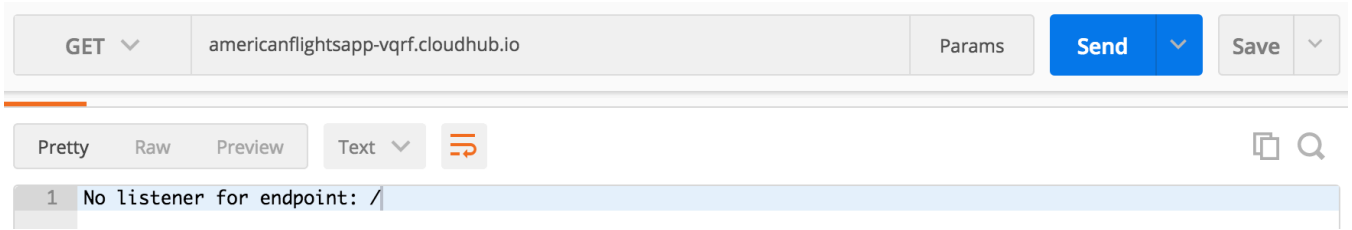
    Note: If your application fails to deploy, look at the messages in the Logs panel. If there is no message about incorrect syntax, try restarting the workspace by clicking the options menu in the application status area and selecting Restart workspace.

29. Click the options menu in the application status area and select Copy link.

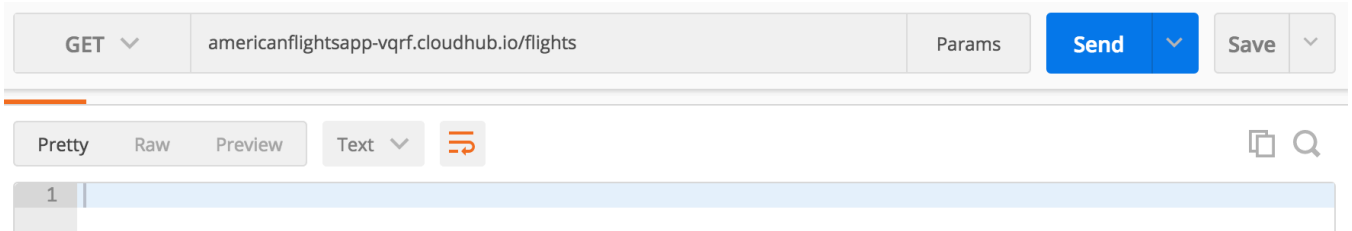


Test the application

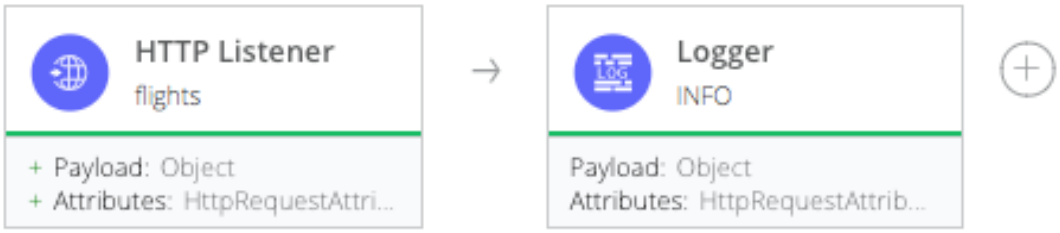
30. Return to Postman, paste the copied link, and click Send; you should get a No listener for endpoint: / message.



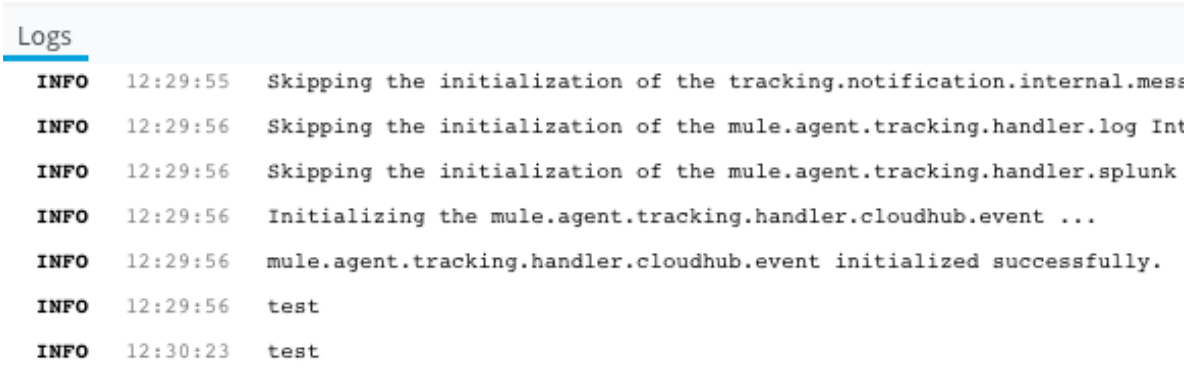
31. Add /flights to the path and click Send; you should get a 200 response with no body.



32. Click Send again to make a second request.
33. Return to flow designer.
34. Notice that there are now green lines across both cards.

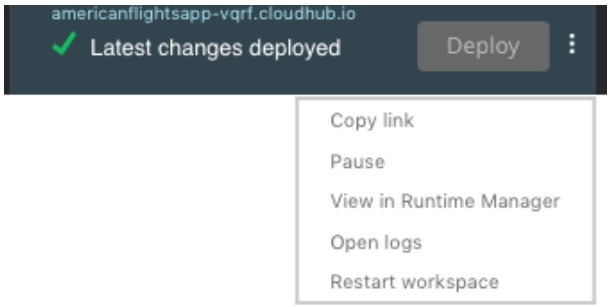


35. Look at the logs; you should see your Logger message displayed twice.

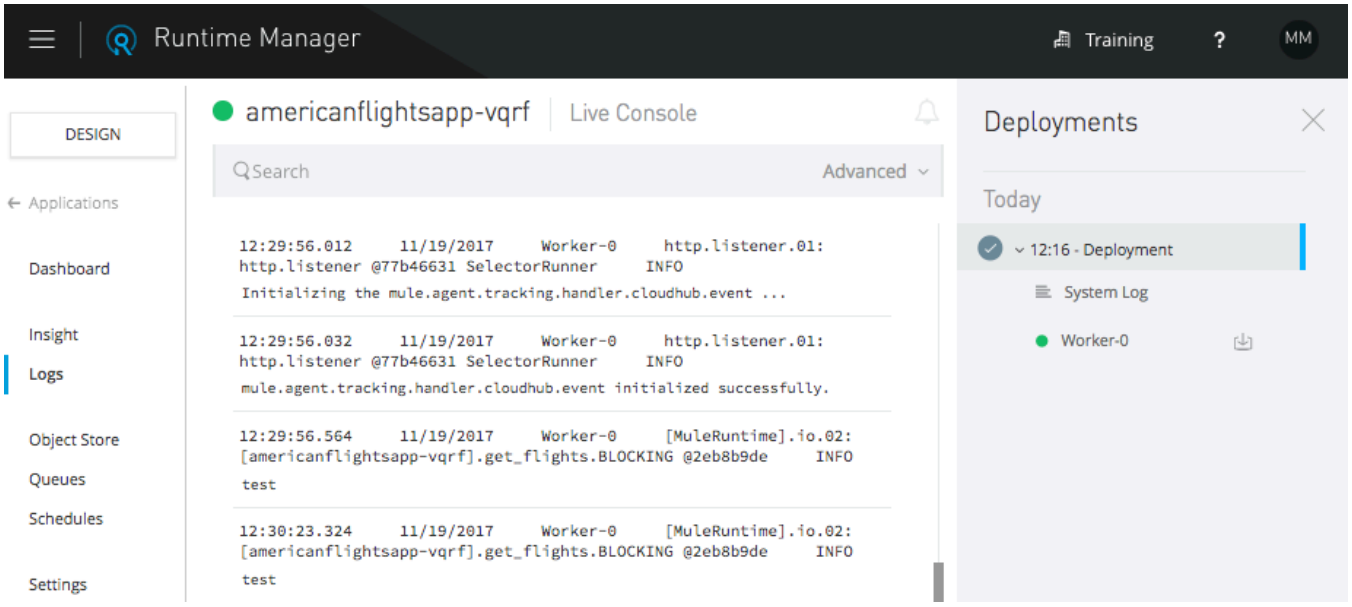


View the application in Runtime Manager

36. Click the options menu in the application status area and select View in Runtime Manager; Runtime Manager should open in a new tab.

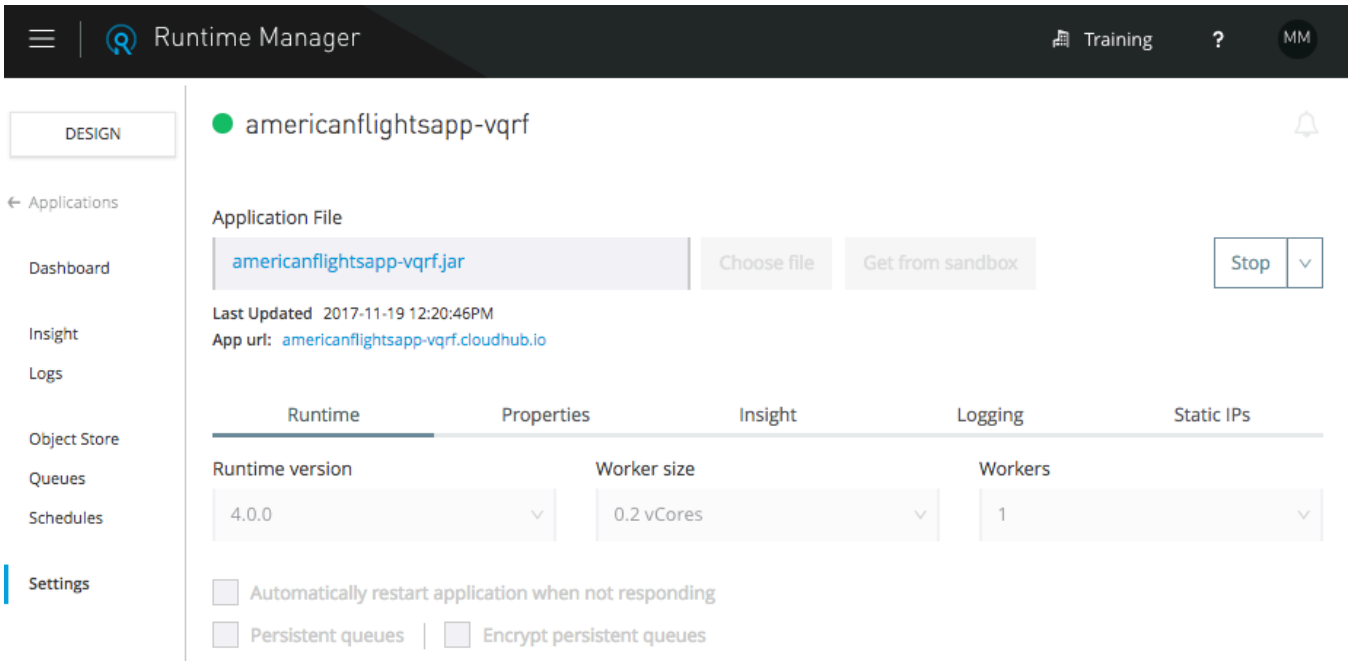


37. In the new browser tab that opens with Runtime Manager, review the application log file; you should see your test log messages.



38. In the left-side navigation, click Settings.
39. Review the settings page and locate the following information for the application:

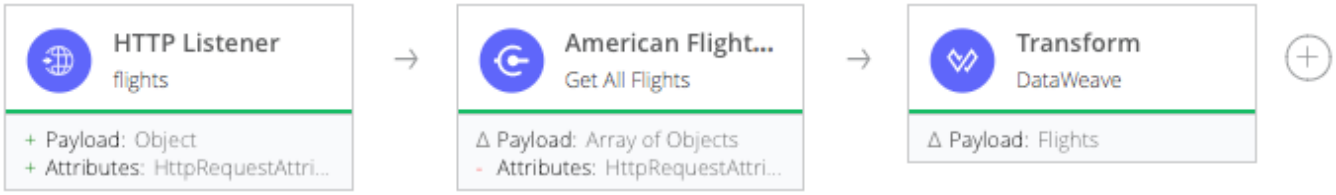
- To which environment it was deployed
- To what type of Mule runtime it was deployed
- To what size worker it was deployed



Walkthrough 2-3: Create an integration application with flow designer that consumes an API

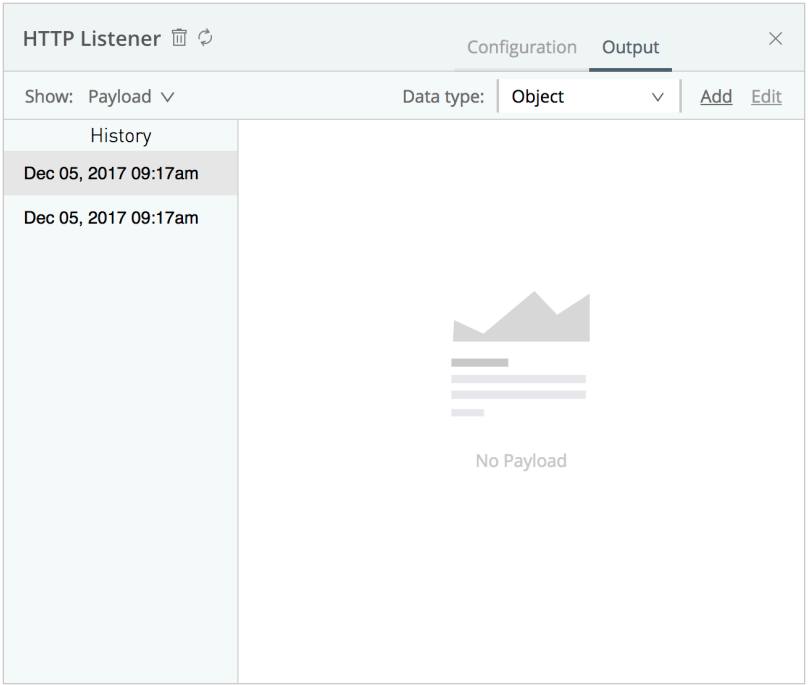
In this walkthrough, you build an integration application to consume an API from Anypoint Exchange. You will:

- Examine Mule event data for calls to an application.
- Use the American Flights API in Anypoint Exchange to get all flights.
- Transform data returned from an API to another format.

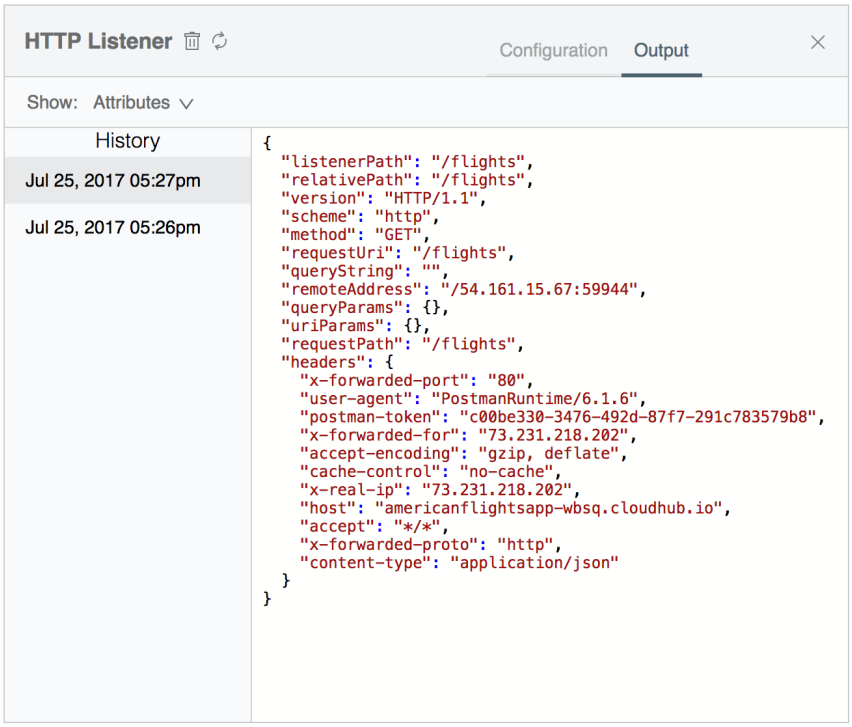


Review Mule event data for the calls to the application

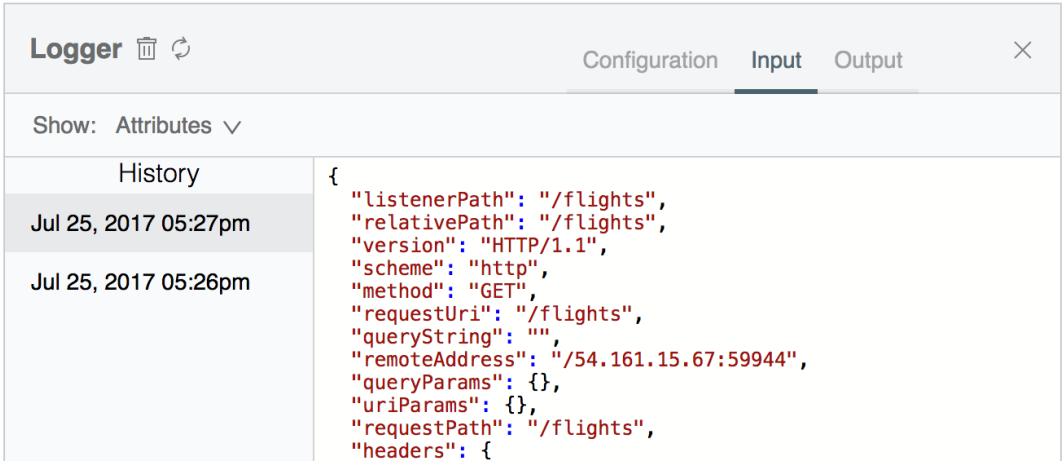
1. Return to the American Flights App in flow designer.
2. Click the title bar of the Logs panel to close it.
3. Expand the HTTP Listener card.
4. Click the Output tab.
5. Locate the Show drop-down menu that currently has Payload selected.



6. Locate your two calls to the application in the History panel; there should be no message payload for either call.
7. Change the Show drop-down menu to Attributes.
8. Review the attributes for the Mule event leaving the HTTP Listener processor.



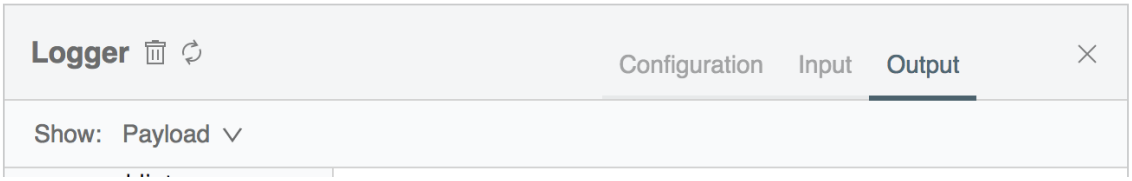
9. Close the card.
10. Open the Logger card.
11. Click the Input tab.
12. Review the payload and attributes values for the two calls.



13. Click the Output tab and review the payload and attributes values for the calls.

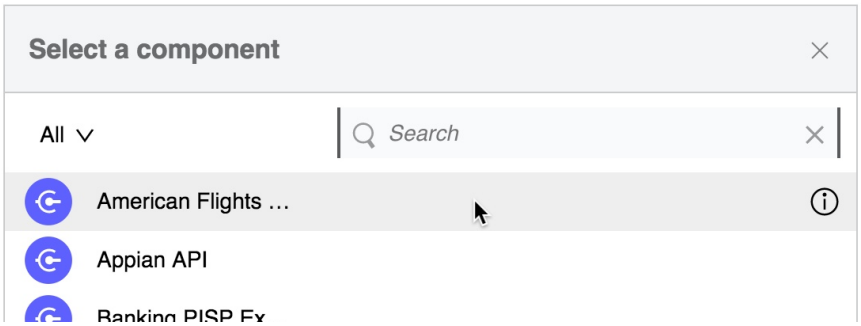
Delete a card

14. Click the Remove button at the top of the card.

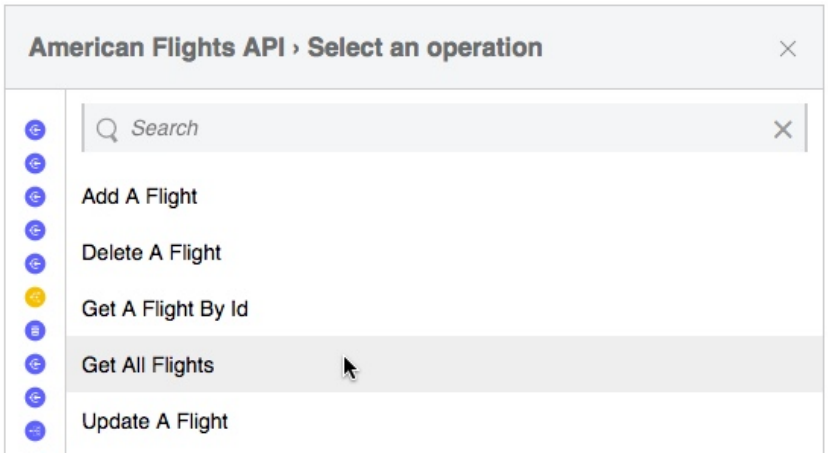


Use the American Flights API in Anypoint Exchange to get all flights

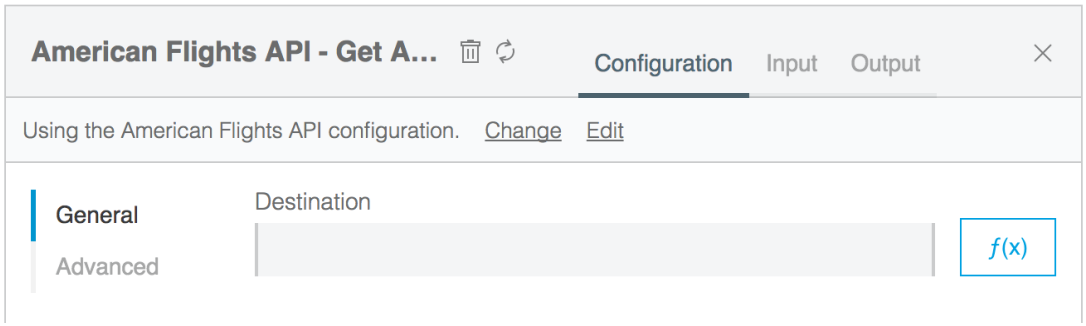
15. Click the Add button next to the HTTP Listener card.
16. In the Select a component dialog box, select the American Flights API.



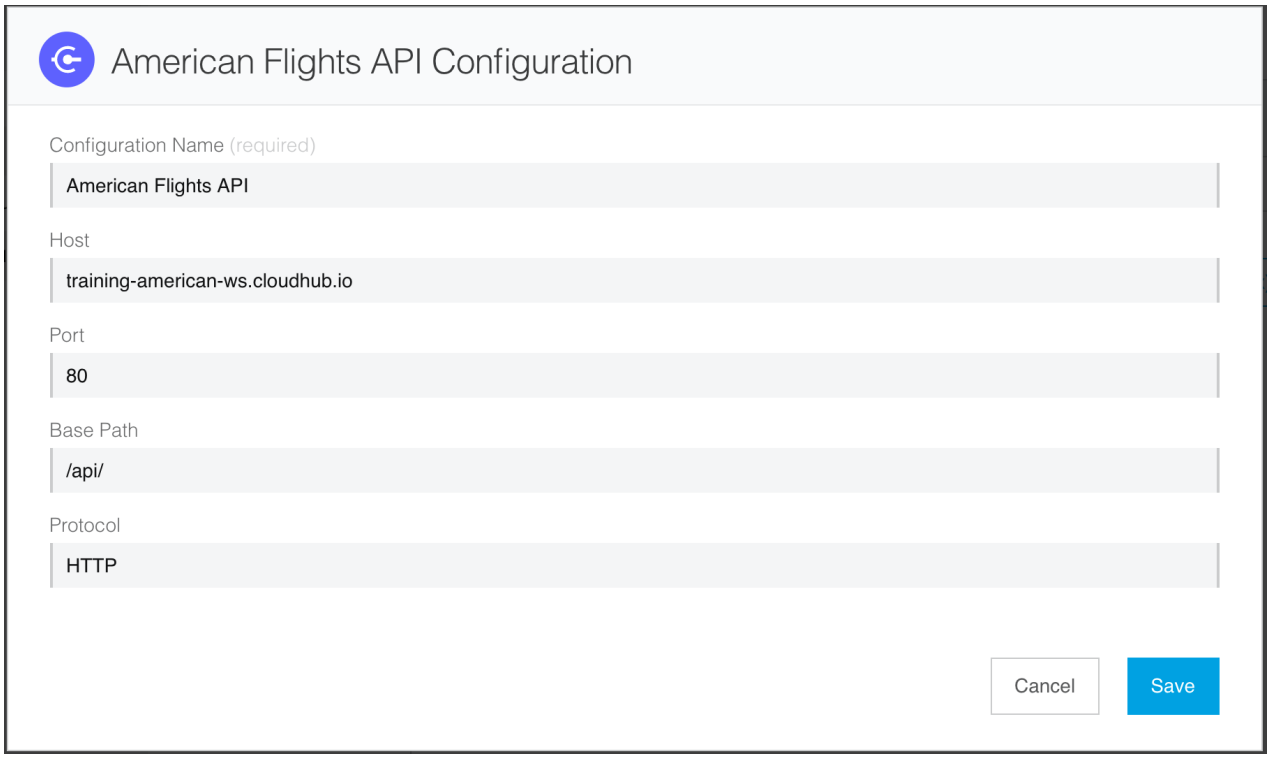
17. In the American Flights API > Select an operation dialog box, select Get All Flights.



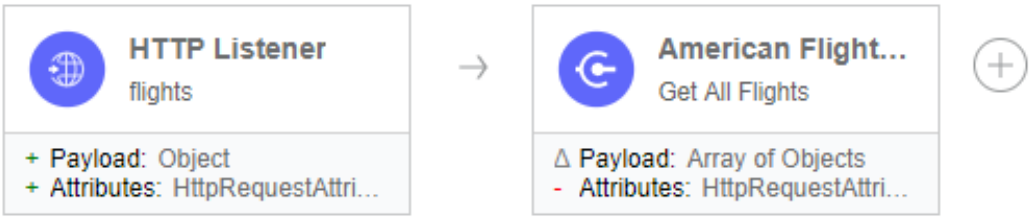
18. In the Get All Flights dialog box, click the Edit link for the American Flights API configuration.



19. Review the information and click Cancel.



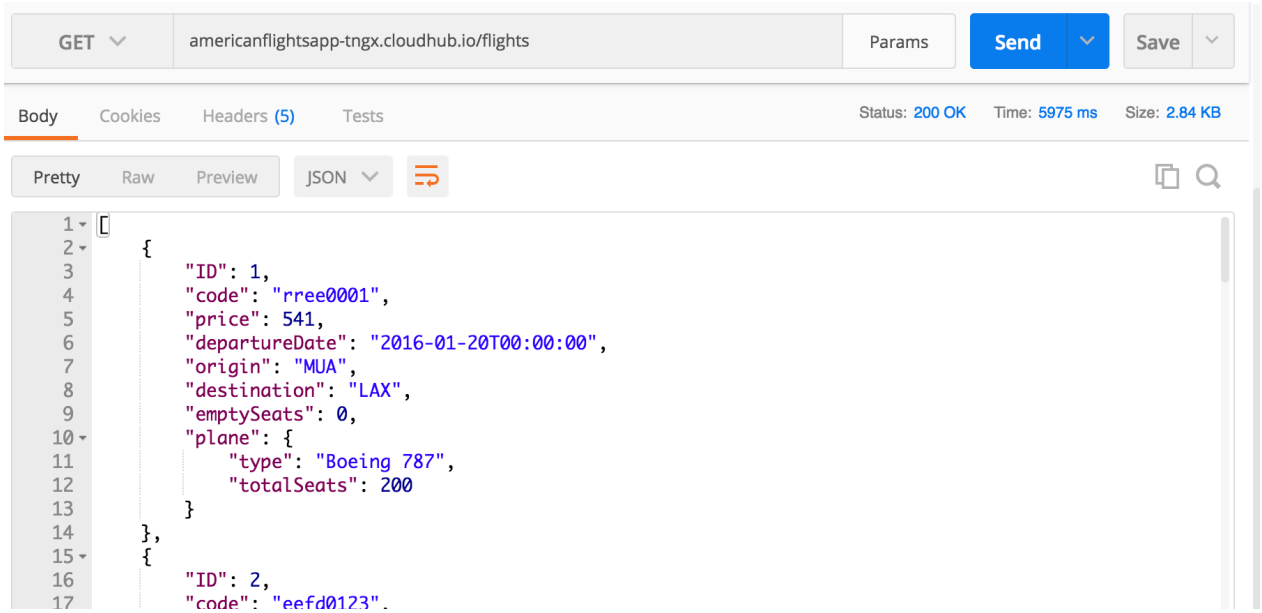
20. Close the American Flights API card.



21. Click the Deploy button in the main menu bar.

Test the application

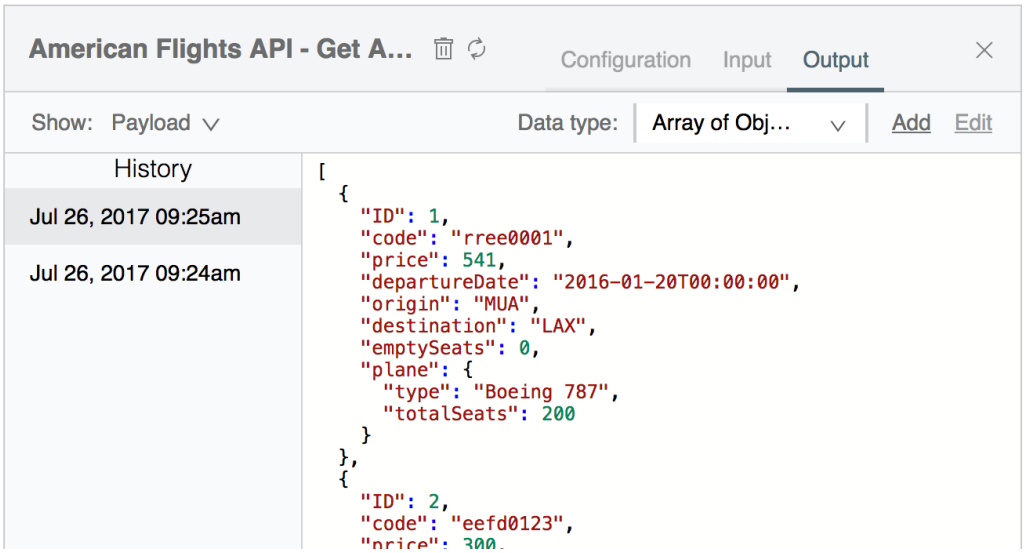
22. Return to Postman and click Send; you should see flight data.



23. Click Send again to make a second request.

Review Mule event data

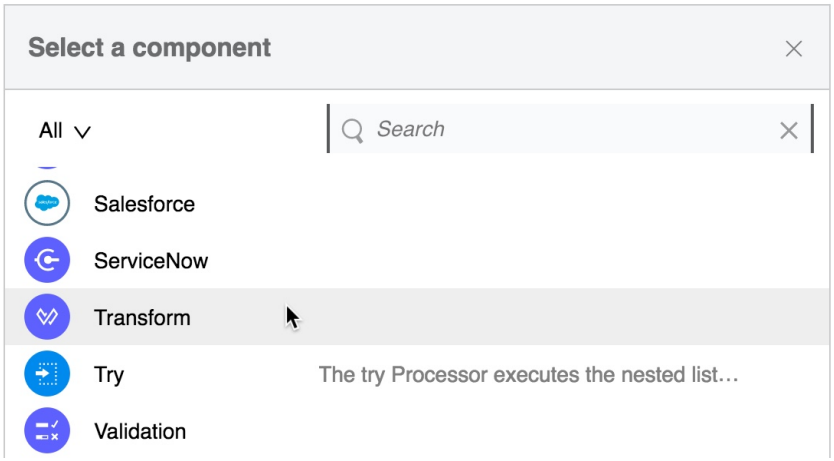
24. Return to flow designer and open the American Flights API card.
25. Click the Input tab and examine the Mule event data.
26. Click the Output tab; you should see payload data.



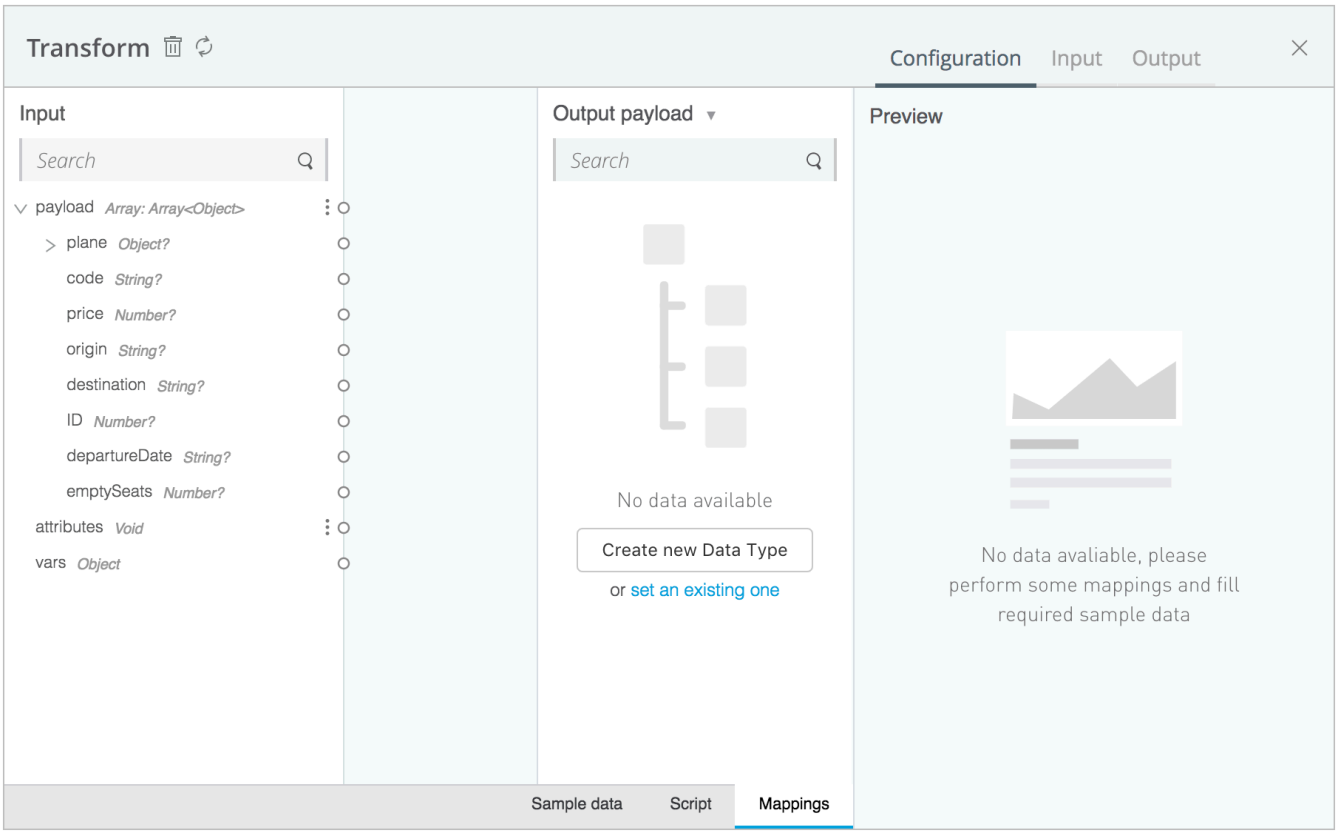
27. Close the card.

Add and configure a component to transform the data

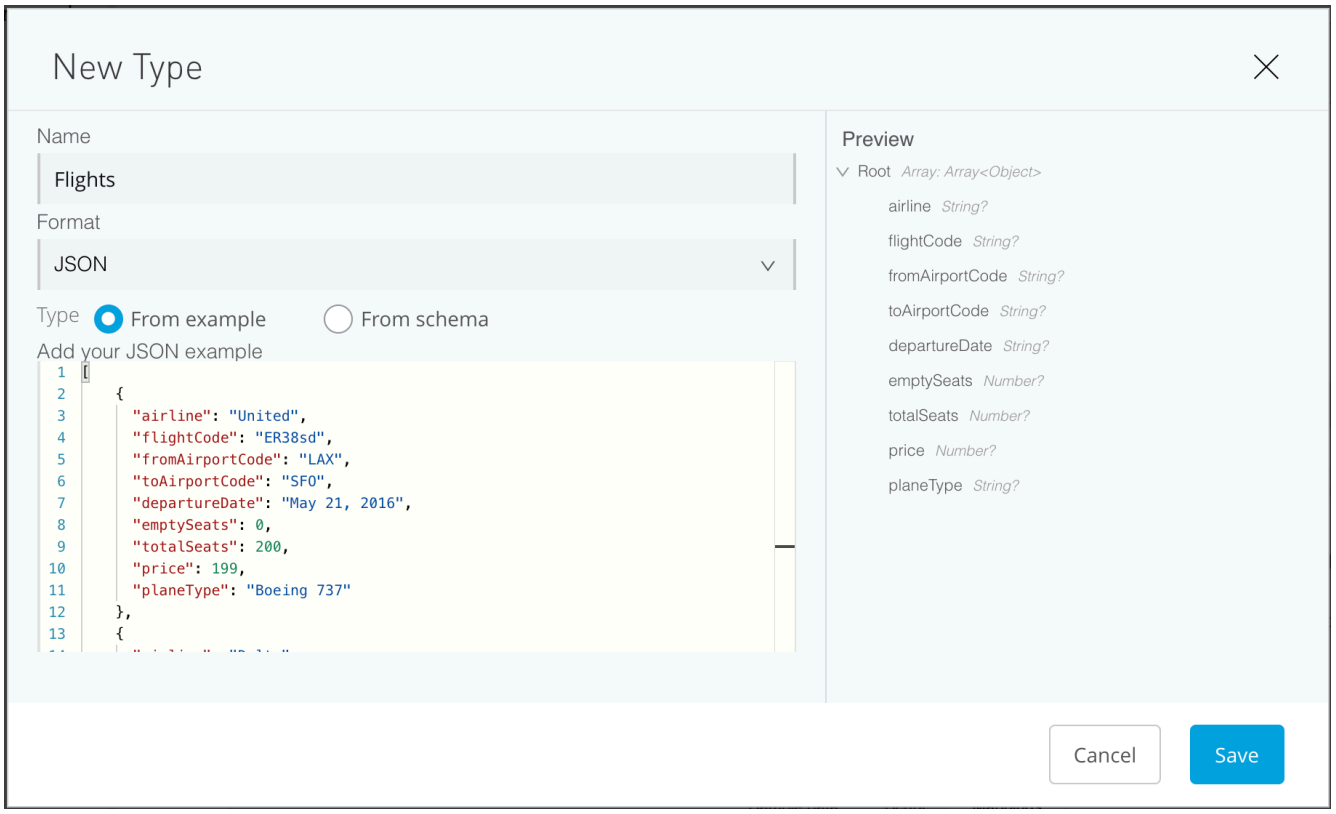
28. Click the add button in the flow.
29. In the Select a component dialog box, select Transform.



30. In the Transform card, look at the Mule event structure in the input section.
31. In the output section, click the Create new Data Type button.

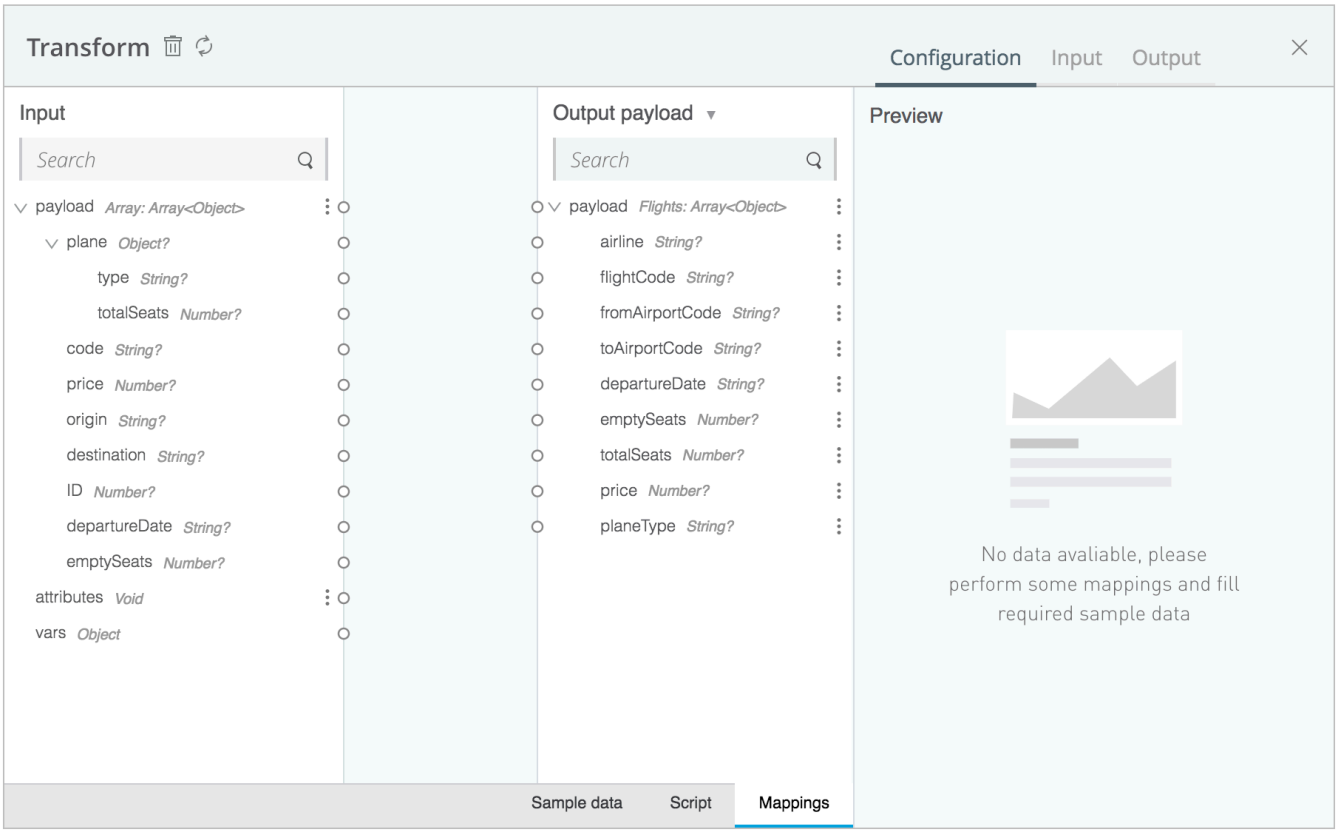


32. In the New Type dialog box, set the following values:
    - Name: Flights
    - Format: JSON
    - Type: From example
33. In the computer's file explorer, return to the student files folder and locate the flights-example.json file in the examples folder.
34. Open the file in a text editor and copy the code.
35. Return to flow designer and paste the code in the section to add your JSON example.



36. Click Save.

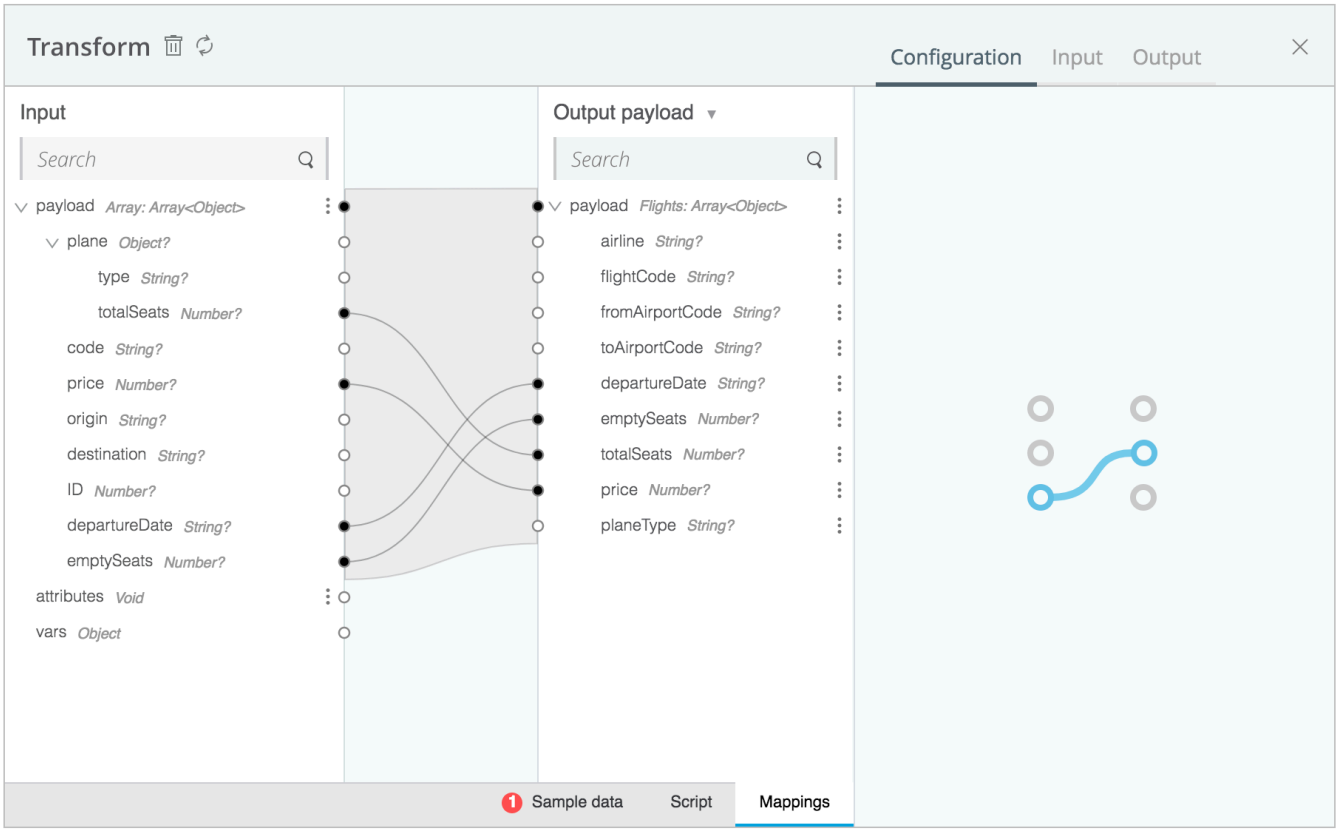
37. In the input section, expand the plane object.



Create the transformation

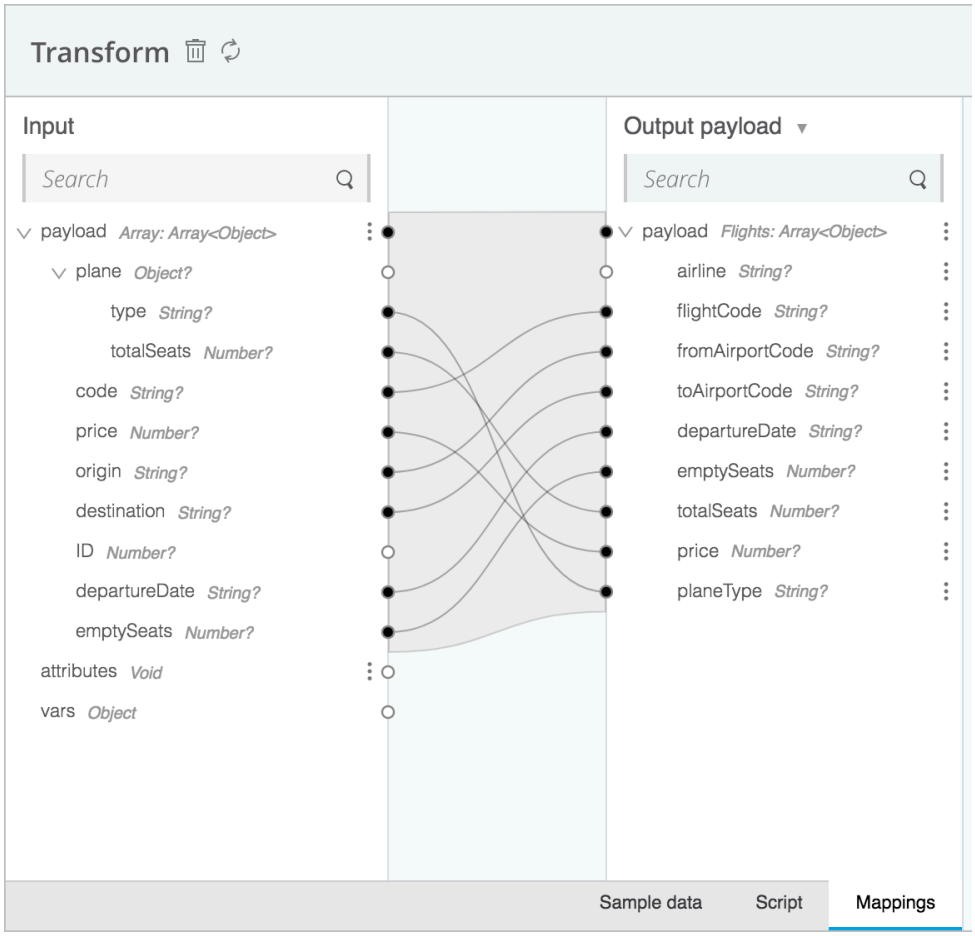
38. Map fields with the same names by dragging them from the input section and dropping them on the corresponding field in the output section.

    - price to price
    - departureDate to departureDate
    - plane > totalSeats to totalSeats
    - emptySeats to emptySeats

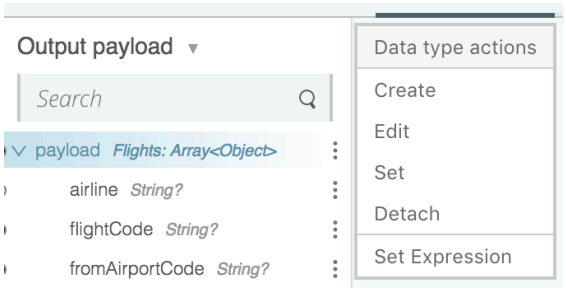


39. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

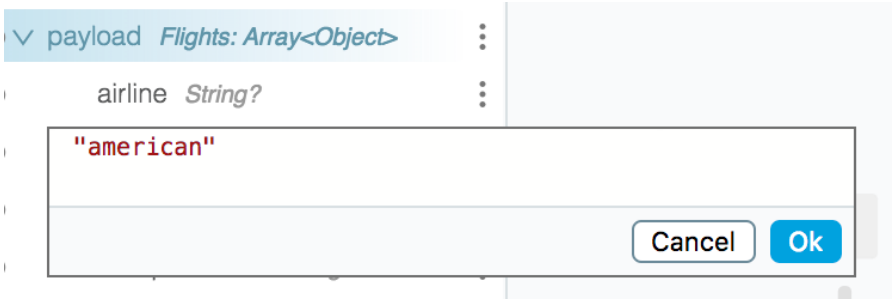
- plane > type to planeType
- code to flightCode
- origin to fromAirport
- destination to toAirport



40. In the output section, click the options menu for the airline field and select Set Expression.

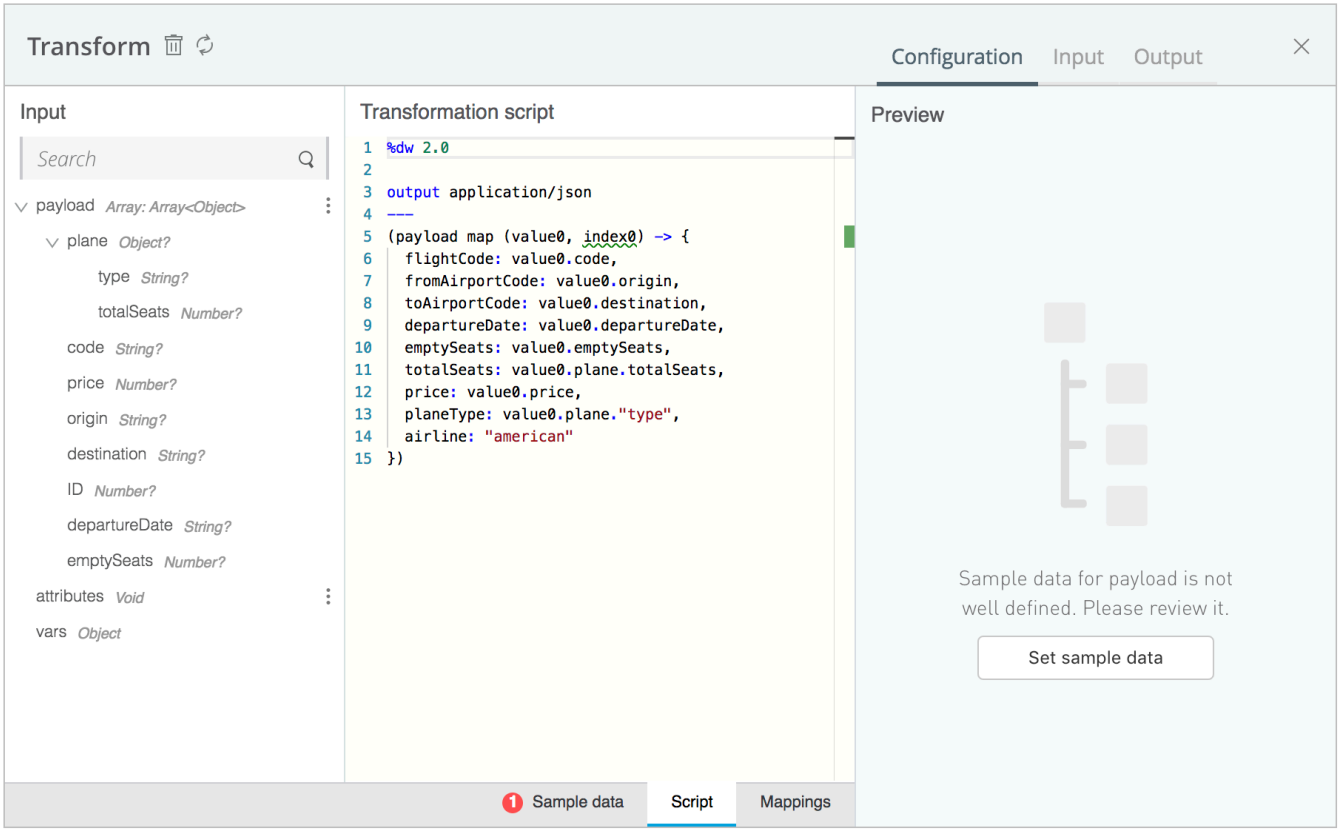


41. Change the value from null to "american" and click OK.



42. Click the Script tab at the bottom of the card; you should see the DataWeave expression for the transformation.

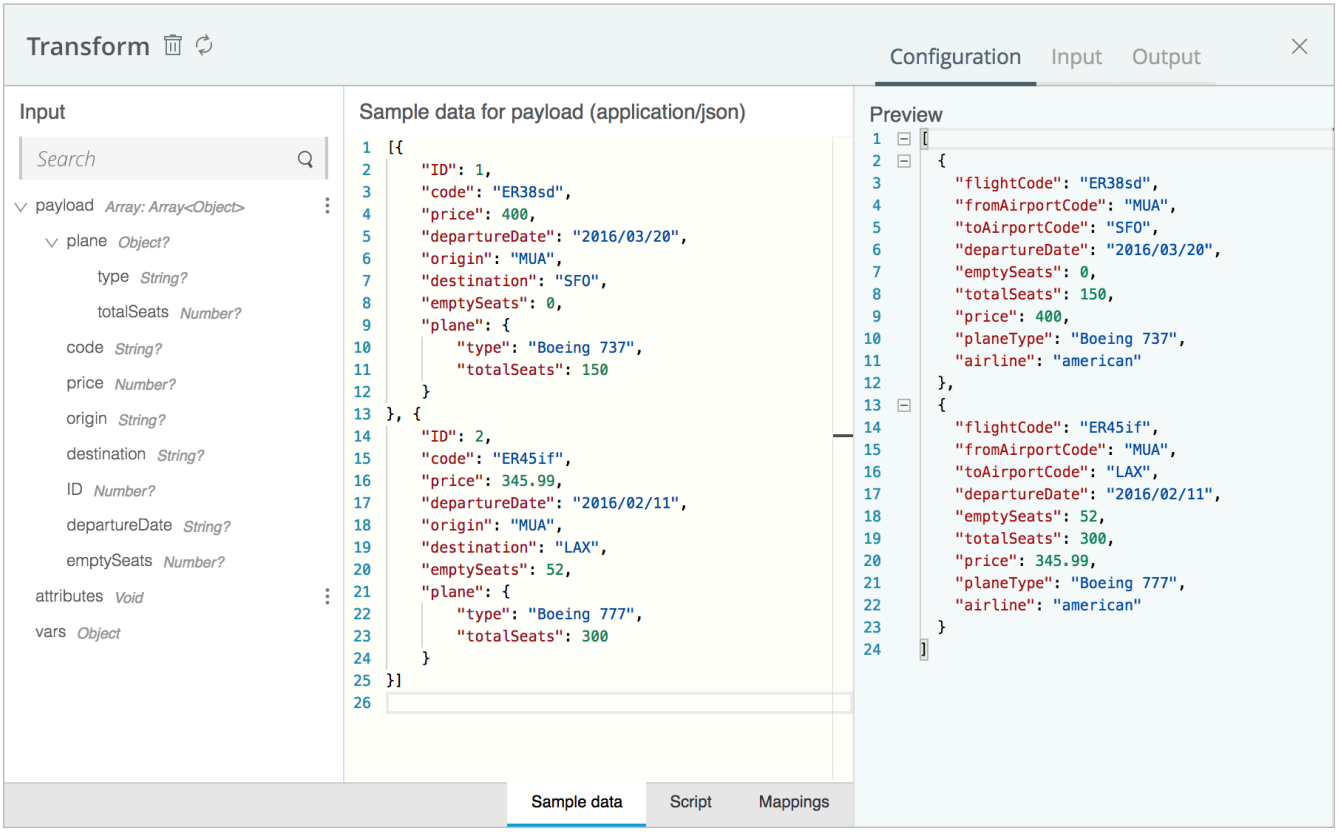
    Note: You learn to write DataWeave 1.0 expressions later in this course. You can also learn to write DataWeave expressions in the Flow Design course and the DataWeave course.



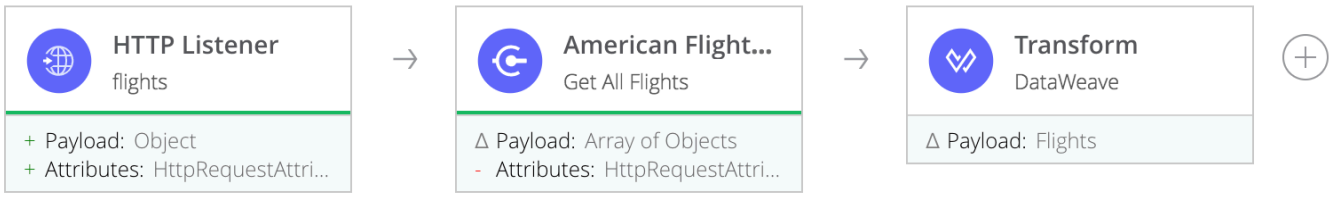
Add sample data

43. Click the Set sample data button in the preview section.
44. In the computer's file explorer, return to the student files folder and locate the american-flights-example.json file in the examples folder.

45. Open the file in a text editor and copy the code.
46. Return to flow designer and paste the code in the sample data for payload section.

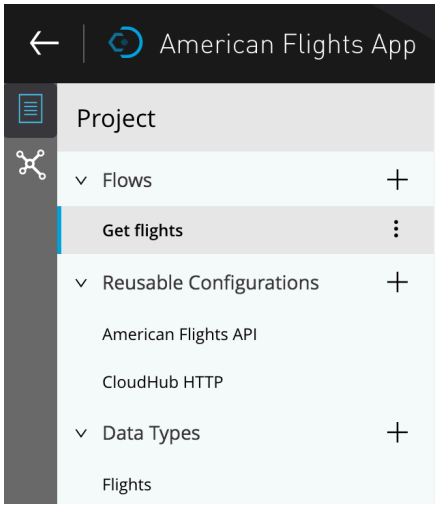


47. Look at the preview section, you should see a sample response for the transformation.
48. Close the card.



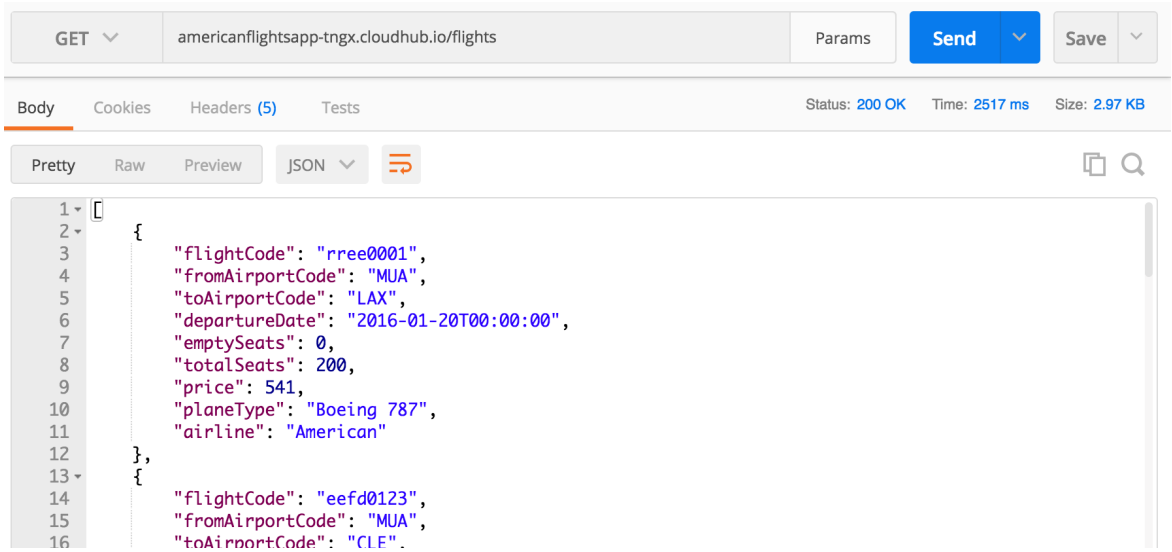
Locate the data type and configuration definitions

49. Locate the connector configurations and the new Flight data type in the project explorer.



Test the application

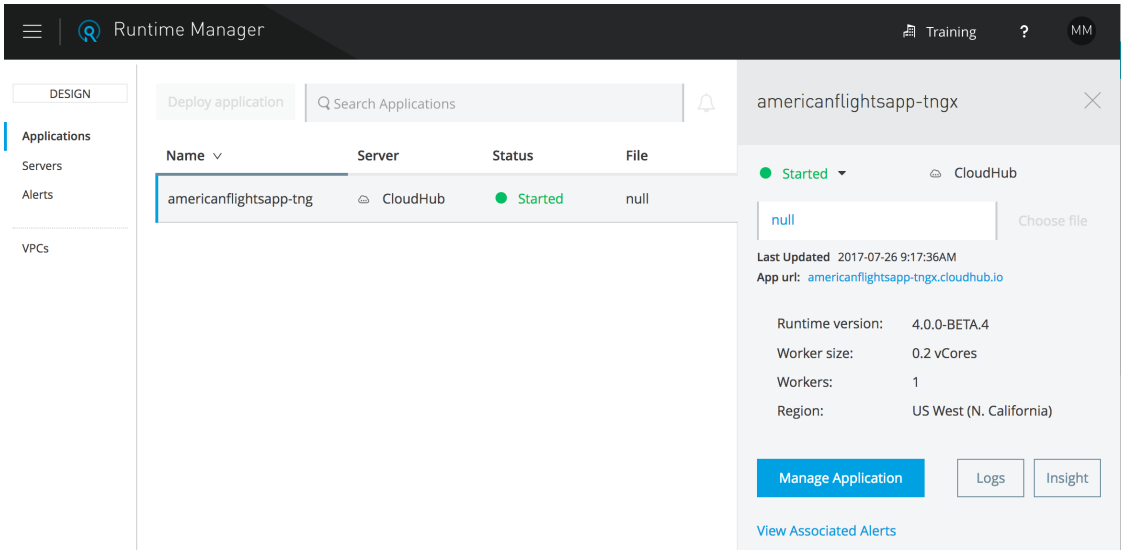
50. Deploy the project.
51. Return to Postman and click Send to make another request to http://americanflightsapp-xxxx.cloudhub.io/flights; you should see all the flight data as JSON again but now with a different structure.



Stop the application

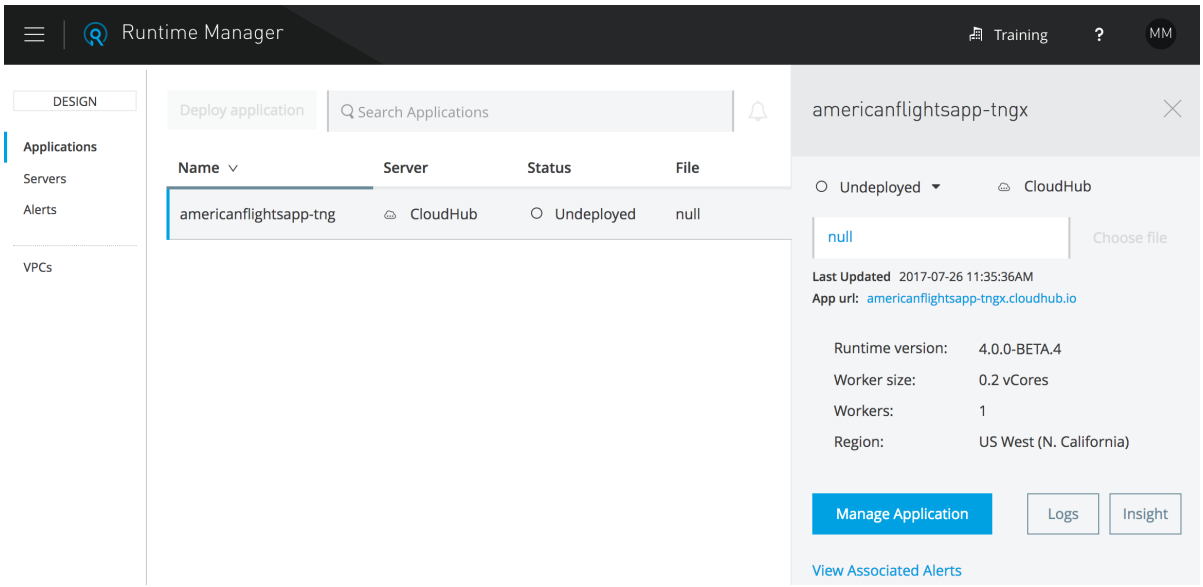
52. Return to Runtime Manager.
53. In the left-side navigation, click Applications.

54. Select the row with your application; you should see information about the application displayed on the right side of the window.

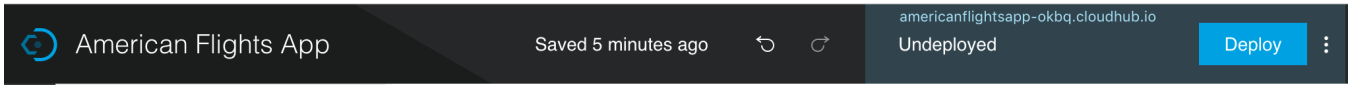


55. Click the drop-down menu button next to Started and select Stop; the status should change to Undeployed.

Note: You can deploy it again from flow designer when or if you work on the application again.

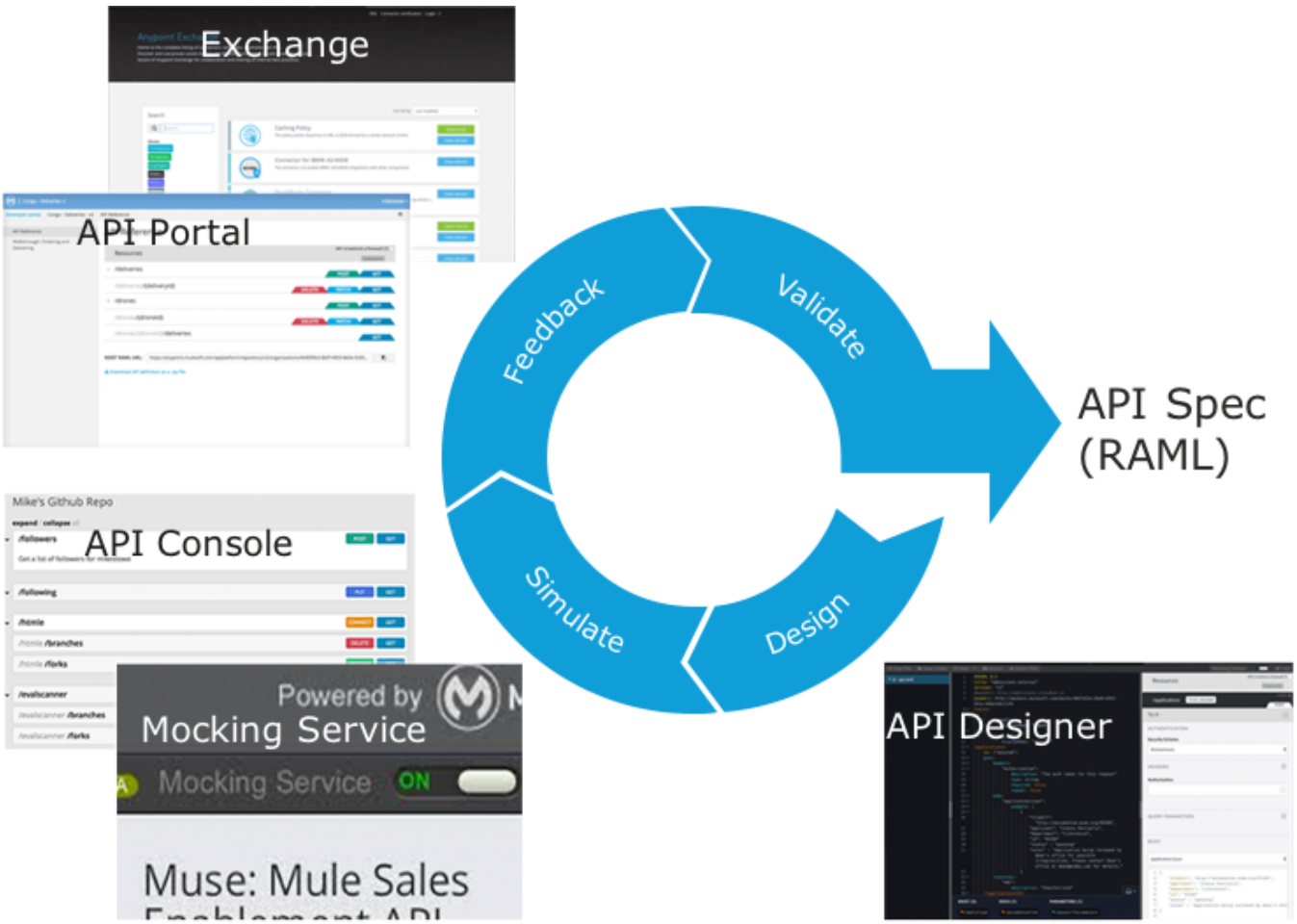


56. Close Runtime Manager.
57. Return to flow designer; you should see the application is undeployed.



58. Return to Design Center.

Module 3: Designing APIs



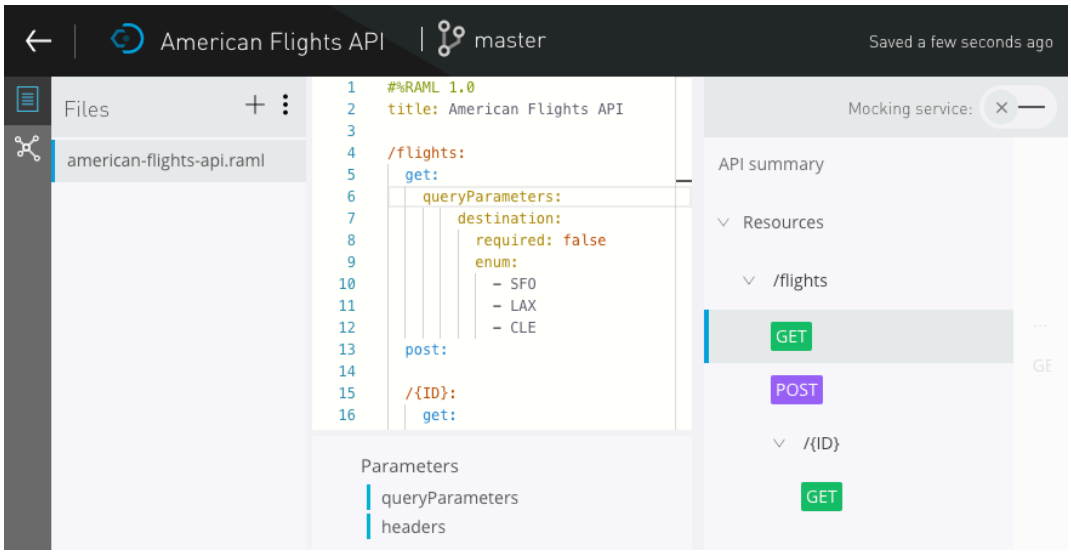
At the end of this module, you should be able to:

- Define APIs with RAML, the Restful API Modeling Language.
- Mock APIs to test their design before they are built.
- Make APIs discoverable by adding them to the private Anypoint Exchange.
- Create public API portals for external developers.

Walkthrough 3-1: Use API designer to define an API with RAML

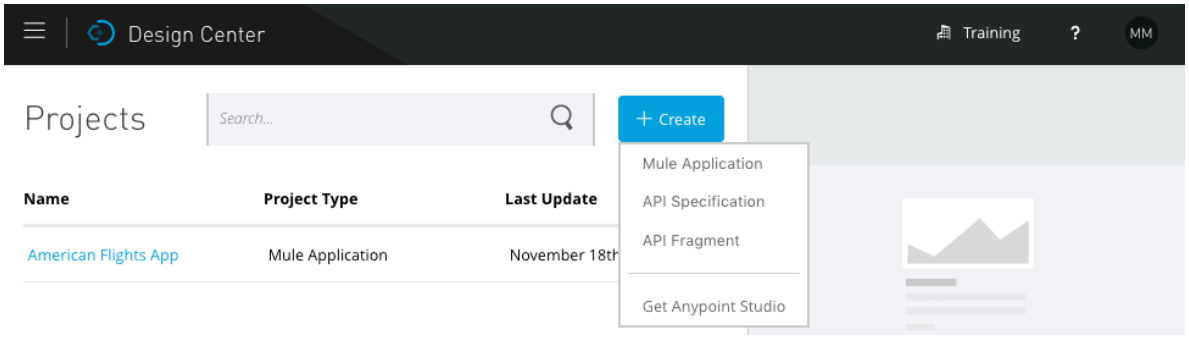
In this walkthrough, you create an API definition with RAML using API designer. You will:

- Define resources and nested resources.
- Define get and post methods.
- Specify query parameters.
- Interact with an API using the API console.



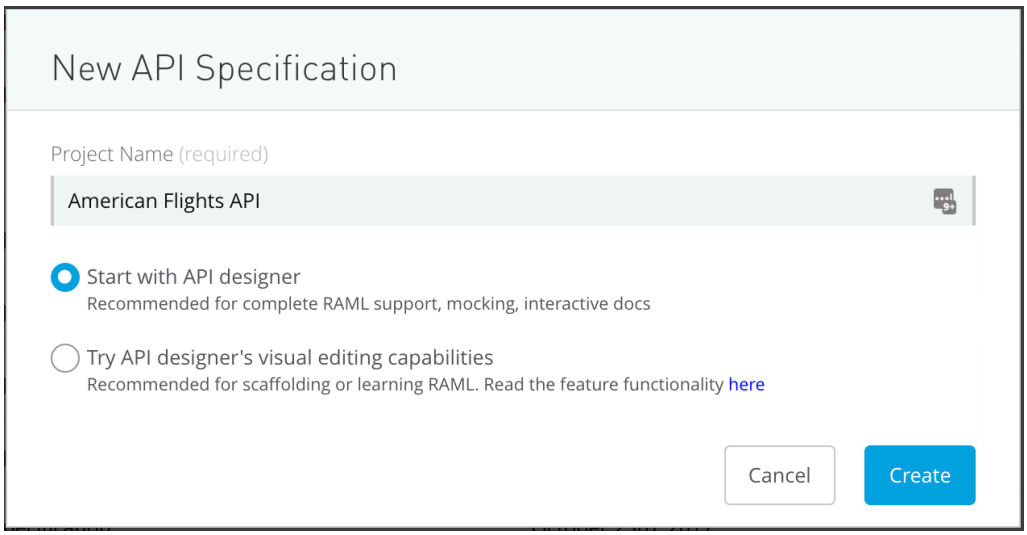
Create a new Design Center project

1. Return to Design Center.
2. Click the Create button and select API Specification.

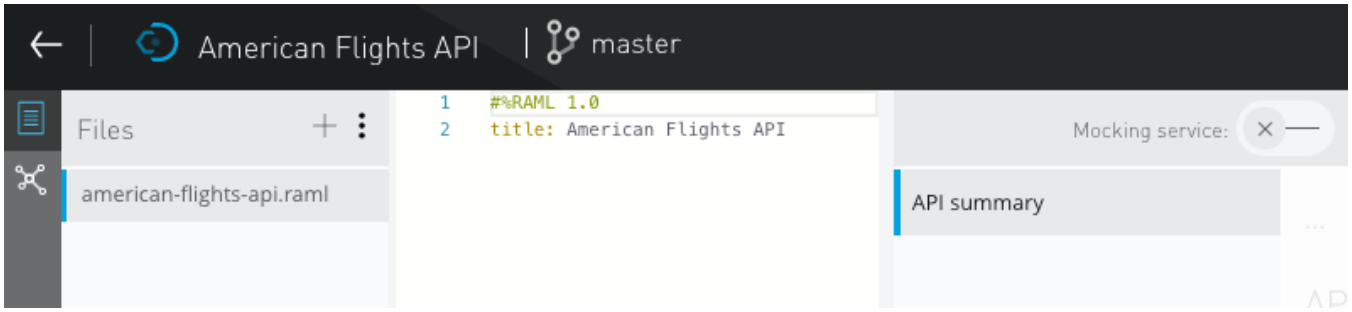


3. In the New API Specification dialog box, set the project name to American Flights API.

4. Select Start with API designer and click Create; API designer should open.



5. In API designer, review the three sections of API designer: the file browser, the editor, and the API console.



Add a RAML resource

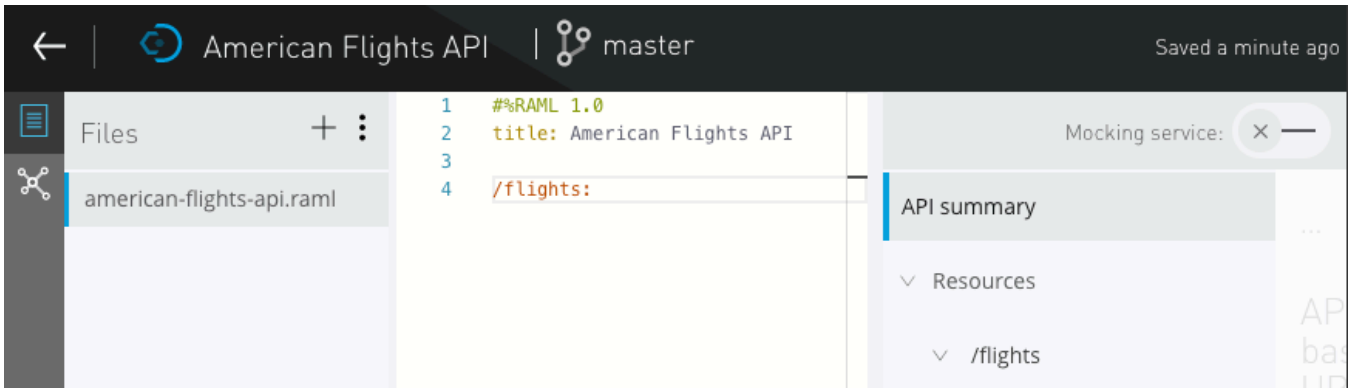
6. In the editor, place the cursor on a new line of code at the end of the file.
7. Add a resource called flights.

```
1   #%RAML 1.0
2   title: American Flights API
3
4   /flights:
```

View the API console

8. Look at the API console on the right side of the window; you should see summary information for the API.

   Note: If you do not see the API console, click the arrow located in the upper-right of the right edge of the web browser window.

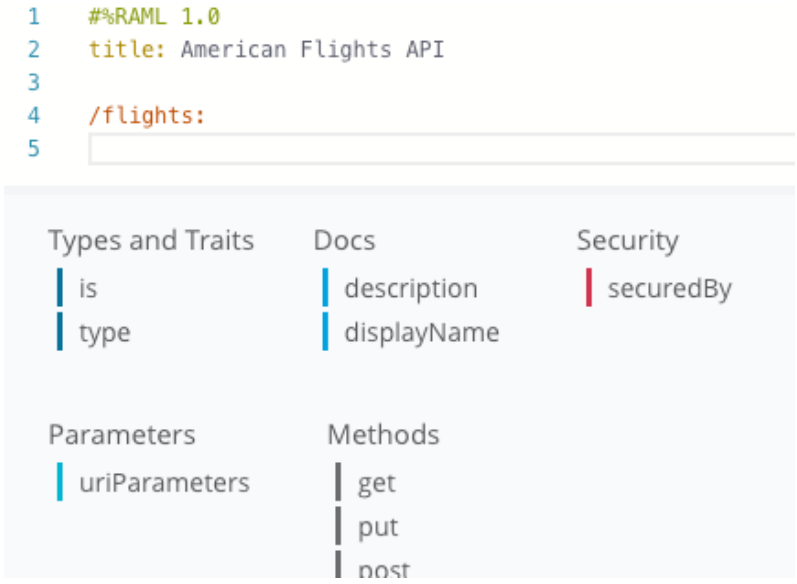


Add RAML methods

9. In the editor, go to a new line of code and look at the contents of the API designer shelf.

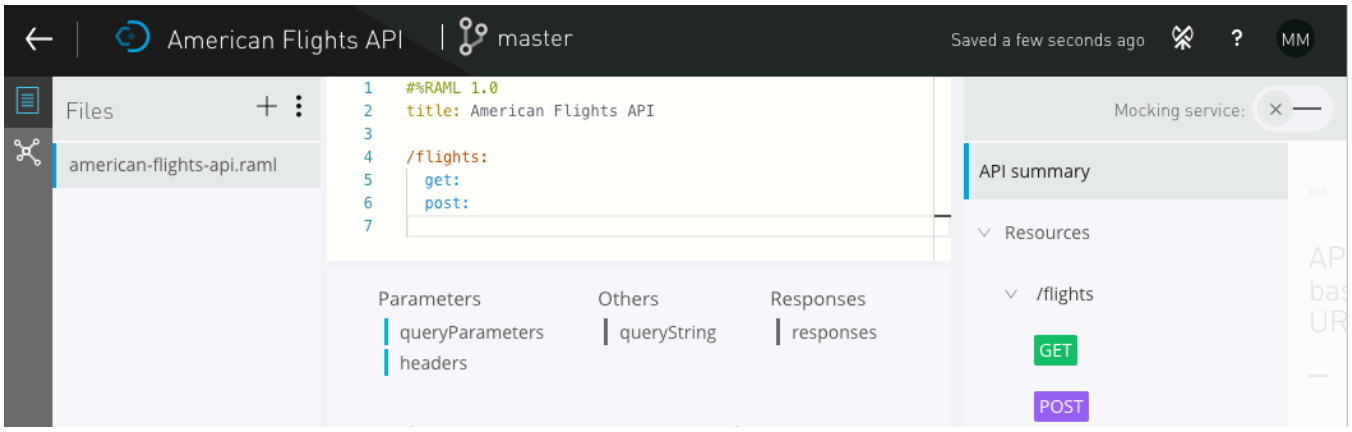
   Note: If you don't see the API designer shelf, it is either minimized or there is an error in your code. To check if it is minimized, go to the bottom of the web browser window and look for an arrow. If you see the arrow, click it to display the shelf.

10. Indent by pressing the Tab key; the contents in the API designer shelf should change.



11. Click the get method in the shelf.

12. Look at the API console; you should see a GET method for the flights resource.
13. In the editor, backspace so you are indented the same amount as the get method.
14. Click the post method in the shelf.
15. Look at the API console; you should see GET and POST methods for the flights resource.

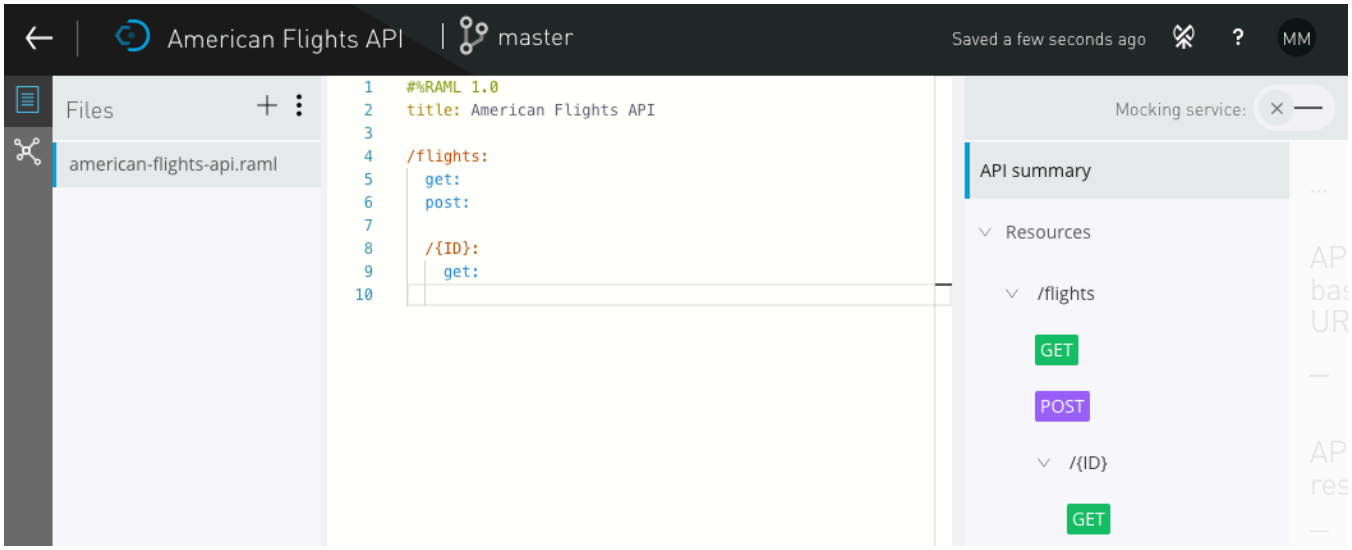


Add a nested RAML resource

16. In the editor, backspace and then go to a new line.
17. Make sure you are still under the flights resource (at the same indentation as the methods).
18. Add a nested resource for a flight with a particular ID.

```
/{ID}:
```

19. Add a get method to this resource.
20. Look at the API console and expand the /{ID} resource; you should see the nested resource with a GET method.



Add an optional query parameter

21. In the editor, indent under the /flights get method (not the /flights/{ID} get method).
22. In the shelf, click the queryParameters parameter.
23. Add a key named destination.

```
#%RAML 1.0
title: American Flights API

/flights:
  get:
    queryParameters:
      destination:
  post:

  /{ID}:
```

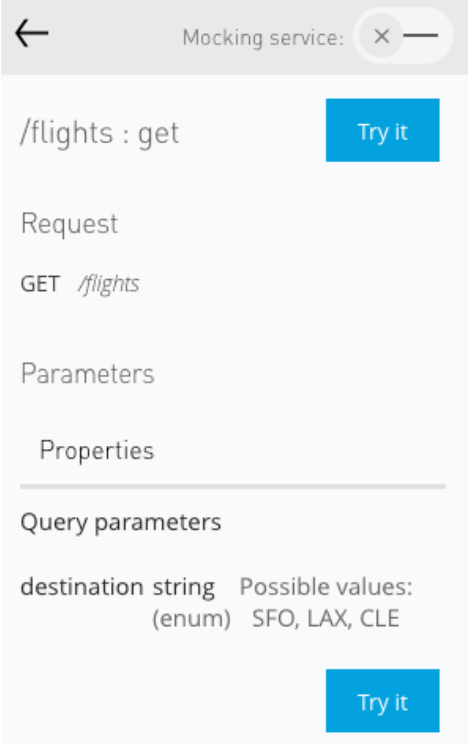
24. Indent under the destination query parameter and look at the possible parameters in the shelf.
25. In the shelf, click the required parameter.
26. In the shelf, click false.
27. Go to a new line of code; you should be at the same indent level as required.
28. In the shelf, click the enum parameter.
29. Set enum to a set of values including SFO, LAX, and CLE.

```
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
```

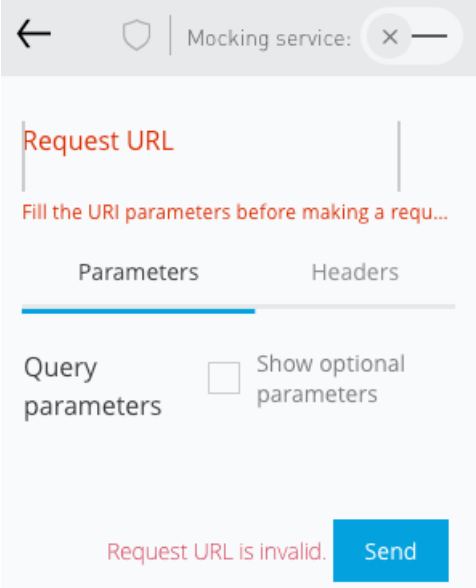
Try to call an API method using the API console

30. In the API console, click the GET method for the /flights resource.

31. Review the information.



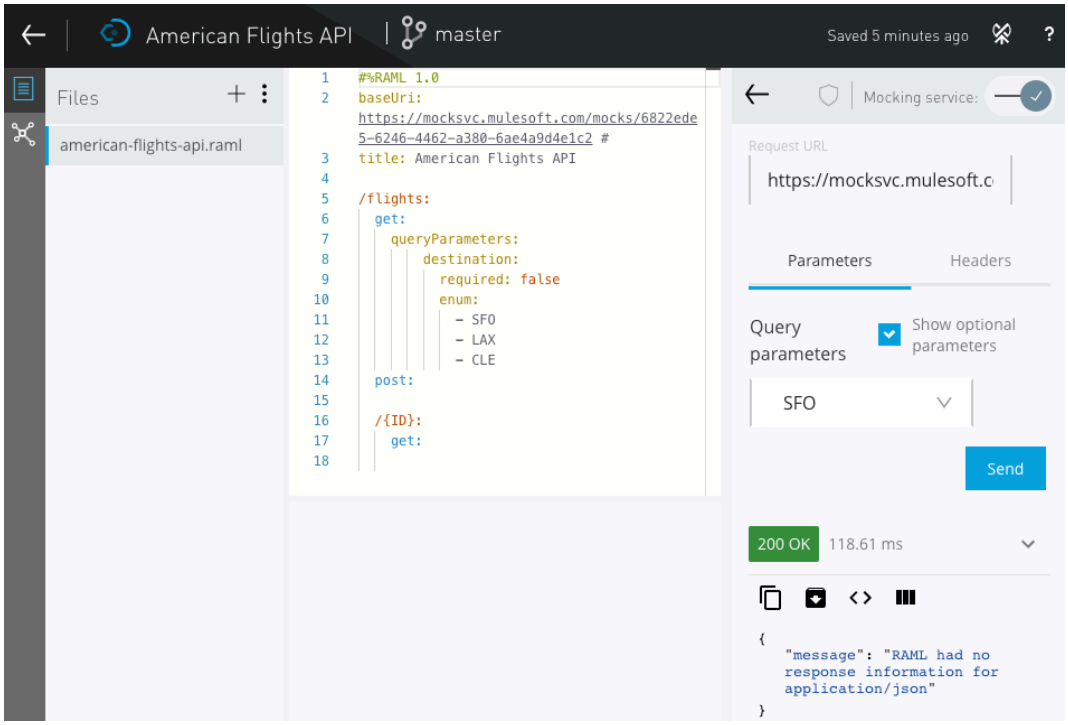
32. Click the Try it button; you should get a message that the Request URL is invalid; a URL to call to try the API needs to be added to the RAML definition.



Walkthrough 3-2: Use the mocking service to test an API

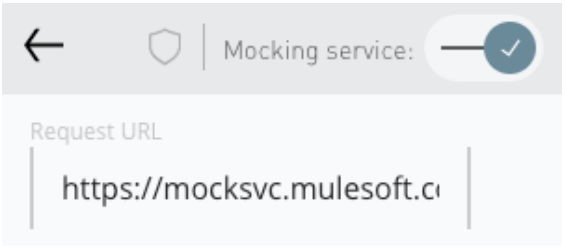
In this walkthrough, you test the API using the Anypoint Platform mocking service. You will:

- Turn on the mocking service.
- Use the API console to make calls to a mocked API.



Turn on the mocking service

1. Return to API designer.
2. Locate the Mocking Service slider in the menu bar at the top of the API console.
3. Click the right-side of the slider to turn it on.

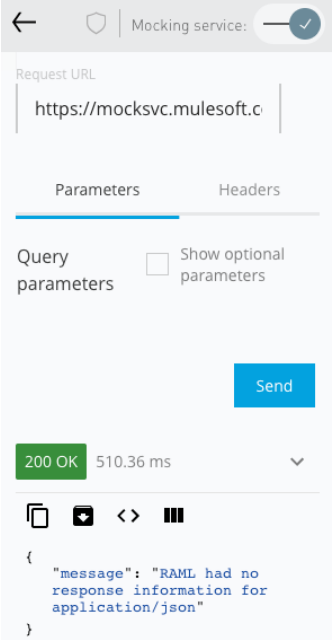


4. Look at the baseUri added to the RAML definition in the editor.

```
1  #%RAML 1.0
2  baseUri: https://mocksvc.mulesoft.com/mocks/6822ede5-6246-4462-a380-6ae4a9d4e1c2 #
3  title: American Flights API
```

Test the /flights:get resource

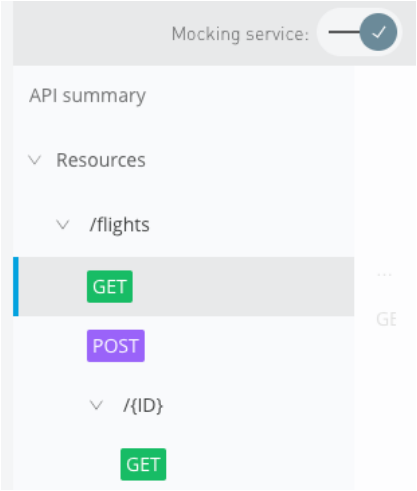
5. In API console, click the Send button for the flights GET method; you should get a 200 status code, a content-type of application/json, and a general RAML message placeholder.



6. Select the Show optional parameters checkbox.
7. In the text field for the code query parameter, enter SFO and click Send; you should get the same response.

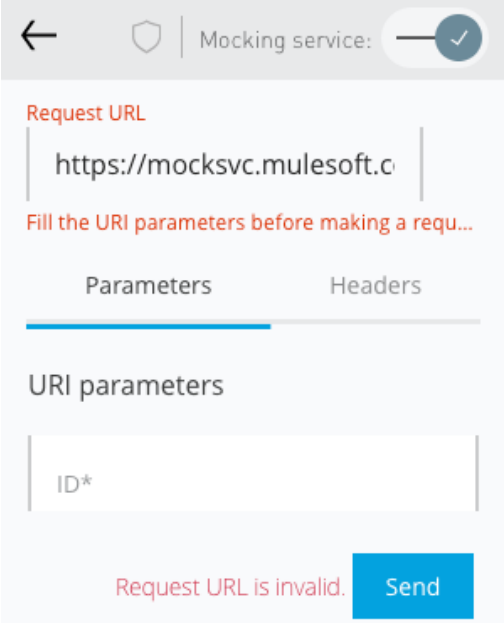
Test the /flights/{ID} resource

8. Click the back arrow at the top of API console twice to return to the resource list.

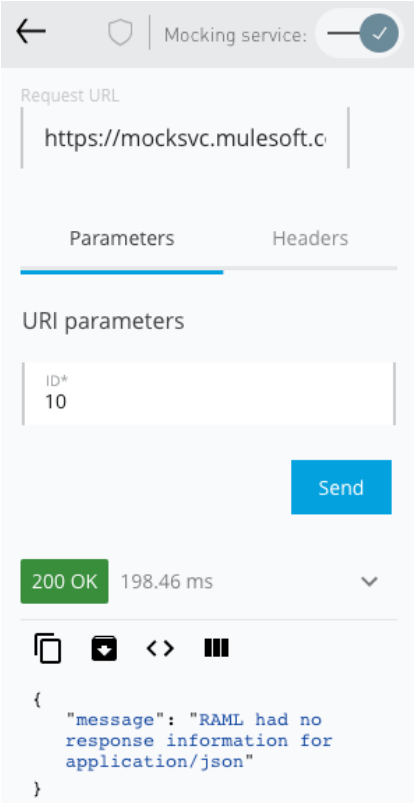


9. Click the GET method for the /{ID} nested resource.

10. Click Try it; you should see a message next to the Send button that the request URL is invalid.

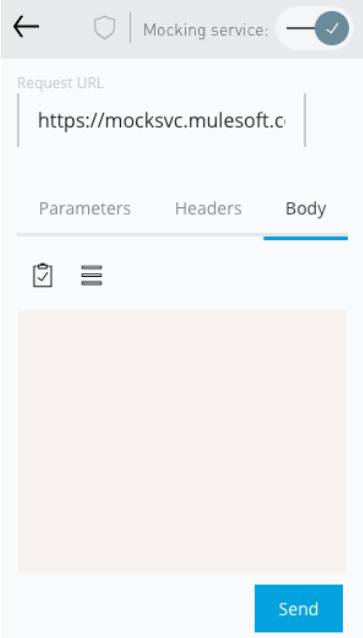


11. In the ID text box, enter a value of 10.
12. Click the Send button.
13. Look at the response; you should get the same default response with a 200 status code, a content-type of application/json, and the general RAML message placeholder.

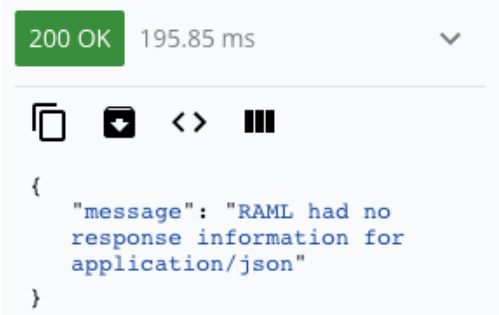


Test the /flights:post resource

14. Click the back arrow at the top of API console twice to return to the resource list.
15. Click the POST method.
16. Click Try it.
17. Select the Body tab; it should not have any content.



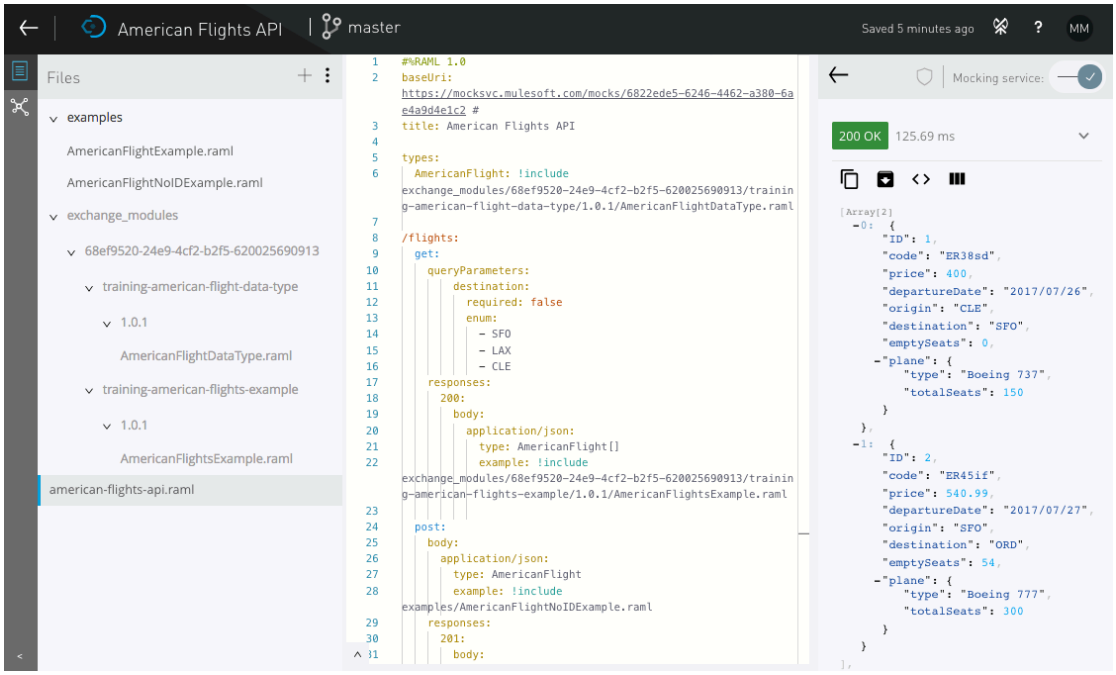
18. Click the Send button.
19. Look at the response; you should get the same generic 200 status code response.



Walkthrough 3-3: Add request and response details

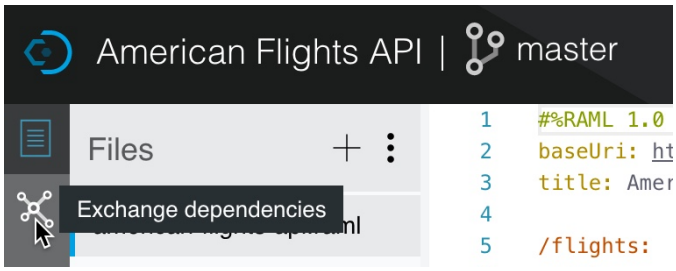
In this walkthrough, you add information about each of the methods to the API specification. You will:

- Use API fragments from Exchange in an API.
- Add a data type to be used by resources in an API.
- Specify data types for GET and POST method requests and responses.
- Add example JSON requests and responses.
- Create new files and folders in an API project and import a file.
- Test an API and get example responses.

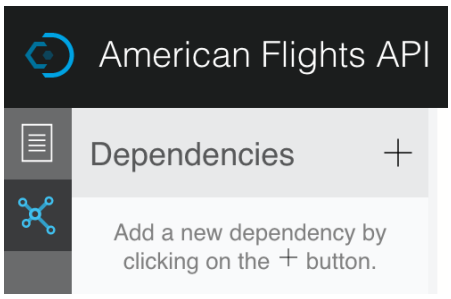


Add data type and example fragments from Exchange

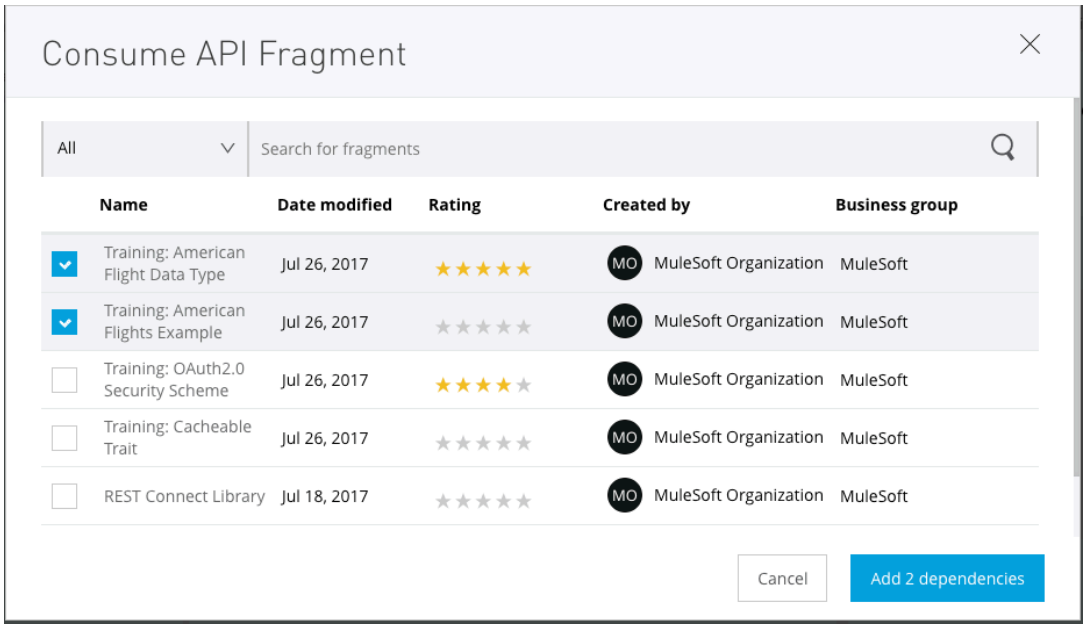
1. Return to API designer.
2. In the file browser, click the Exchange dependencies button.



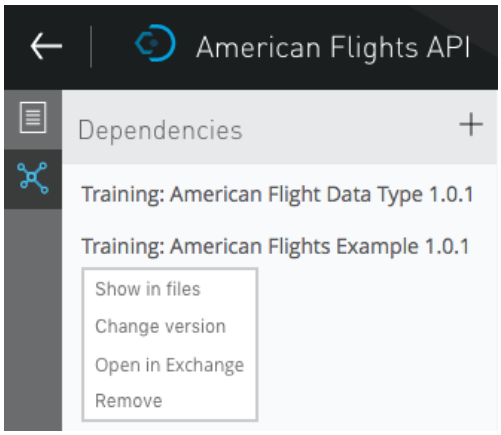
3. Click the Add button.



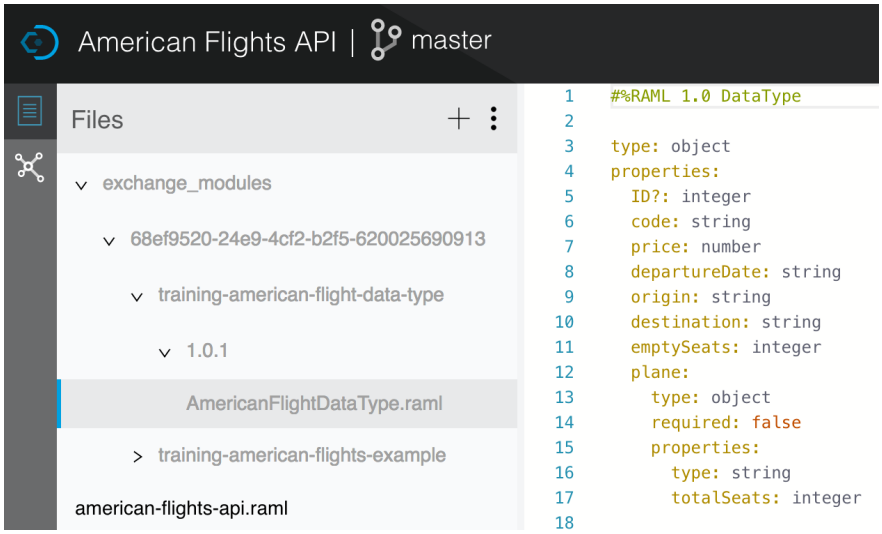
4. In the Consume API Fragment dialog box, select the Training: American Flight Data Type and the Training: American Flights Example.



5. Click the Add 2 dependencies button.
6. In the dependencies list, click the Training: American Flight Data Type and review the menu options.



7. Click the Files button (above the Exchange dependencies button).
8. Expand the exchange_modules section until you see AmericanFlightDataType.raml.
9. Click AmericanFlightDataType.raml and review the code.



10. In the file browser, click the options menu button next to AmericanFlightsDataType.raml and select Copy path to clipboard.

Define an AmericanFlight data type for the API

11. Return to american-flights-api.raml.
12. Near the top of the code above the /flights resource, add a types element.
13. Indent under types and add a type called AmericanFlight.
14. Add the !include keyword and then paste the path you copied.

Note: You can also add the path by navigating through the exchange_modules folder in the shelf.

```
#%RAML 1.0
baseUri: https://mocksvc.mulesoft.com/mocks/3220238a-17c6-4e31-b5fa-413e8129e12b #
title: American Flights API

types:
  AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/tr

/flights:
  get:
```

Specify the /flights:get method to return an array of AmericanFlight objects

15. Go to a new line of code at the end of the /flights get method and indent to the same level as queryParameters.
16. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

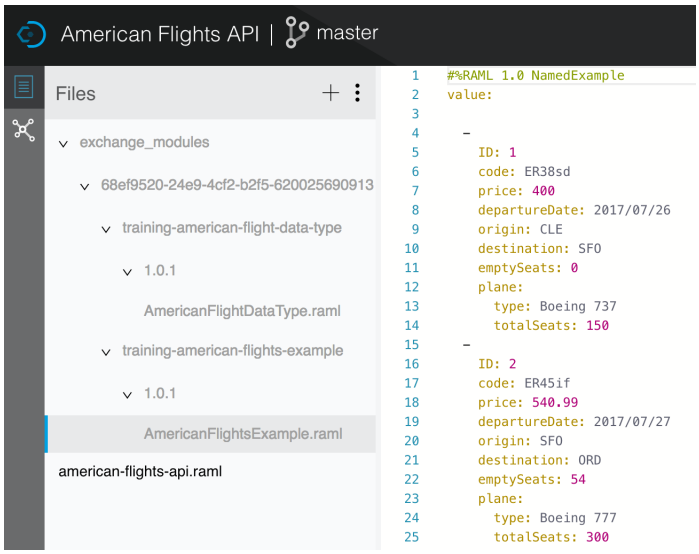
```
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
```

17. Set the type to be an array of AmericanFlight objects: AmericanFlight[].

```
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight[]
```

Add an example response for the /flights:get method

18. In the fie browser, locate AmericanFlightsExample.raml in exchange_modules and review the code.



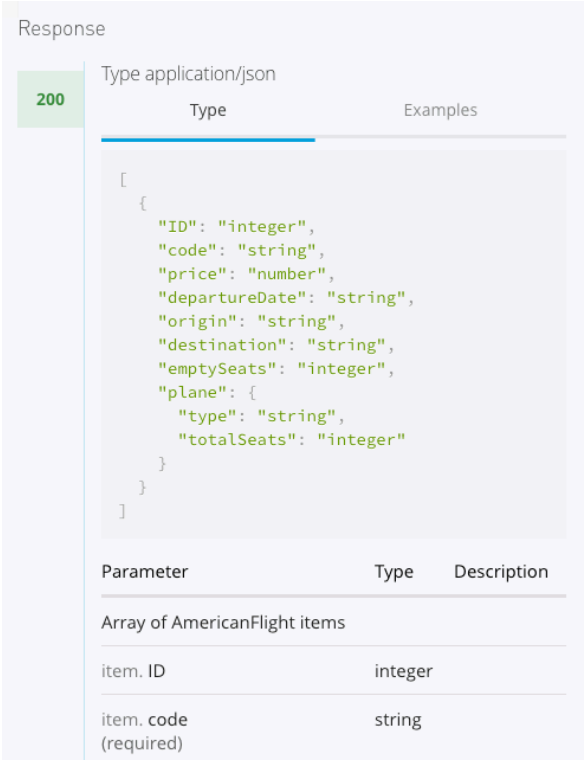
19. In the file browser, click the options menu next to AmericanFlightsExample.raml and select Copy path to clipboard.
20. Return to american-flights-api.raml.
21. In the editor, go to a new line after the type declaration in the /flights:get 200 response (at the same indentation as type).
22. In the shelf, click example.
23. Add the !include keyword and then paste the path you copied.

    Note: You can also add the path by navigating through the exchange_modules folder in the shelf.

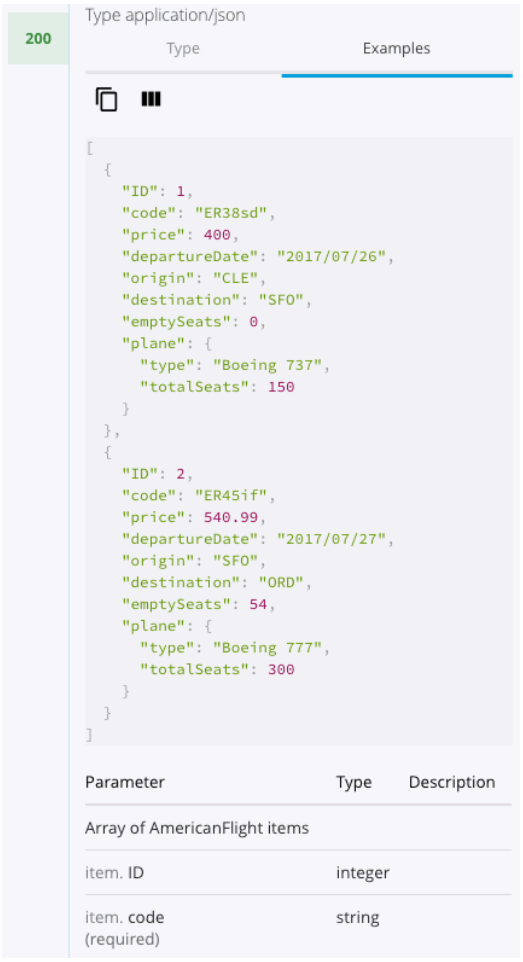
```
17      responses:
18        200:
19          body:
20            application/json:
21              type: AmericanFlight[]
22              example: !include exchange_modules/86f74f12-decb-452c
23    post:
```

Review and test the /flights:get resource in API console

24. In API console, click the /flights:get method.
25. Look at the type information in the response information.



26. Click the Examples tab; you should see the example array of AmericanFlight objects.



27. Click the Try it button and click Send; you should now see the example response with two flights.

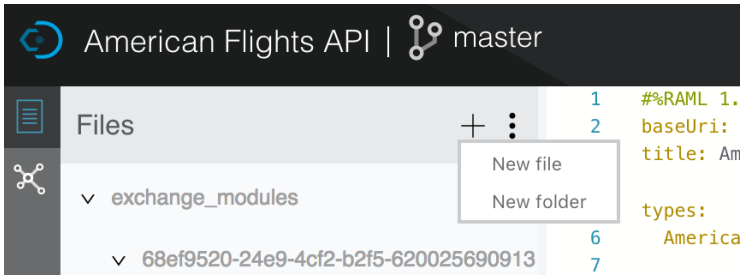
Specify the /{ID}:get method to return an AmericanFlight object

28. In the editor, indent under the /{ID} resource get method.
29. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

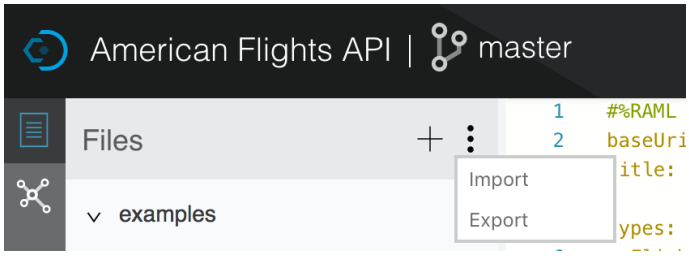
```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
```

Define an example response for the /flights:get method in a new folder

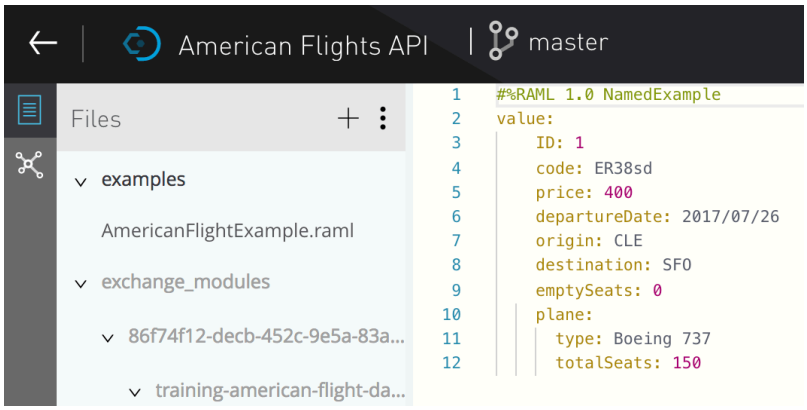
30. In the file browser, click the add button and select New folder.



31. In the Add new folder dialog box, set the name to examples and click Create.
32. In the file browser, click the menu button and select Import.



33. In the Import dialog box, leave File or ZIP selected and click the Choose File button.
34. Navigate to your student files and select the AmericanFlightExample.raml file in the examples folder.
35. In the Import dialog box, click Import; you should see the new file in API designer.
36. Review the code.
37. In the file browser, drag AmericanFlightExample.raml into the examples folder.



Add an example response for the /{ID}:get method

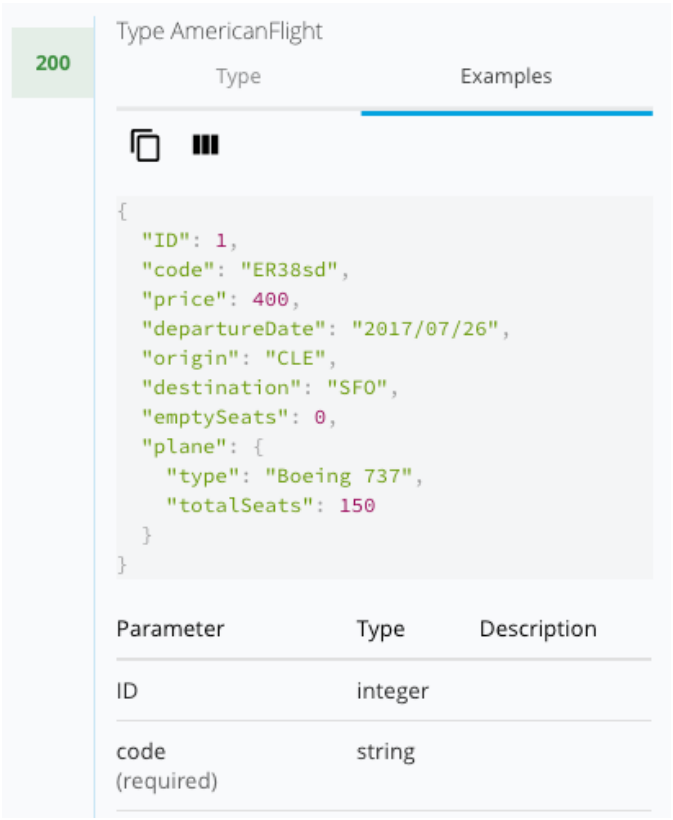
38. Return to american-flights-api.raml.
39. In the editor, go to a new line after the type declaration in {ID}:/get (at the same indentation).

40. In the shelf, click example.
41. Add an include statement and include the example in examples/AmericanFlightExample.raml.

```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
            example: !include examples/AmericanFlightExample.raml
```

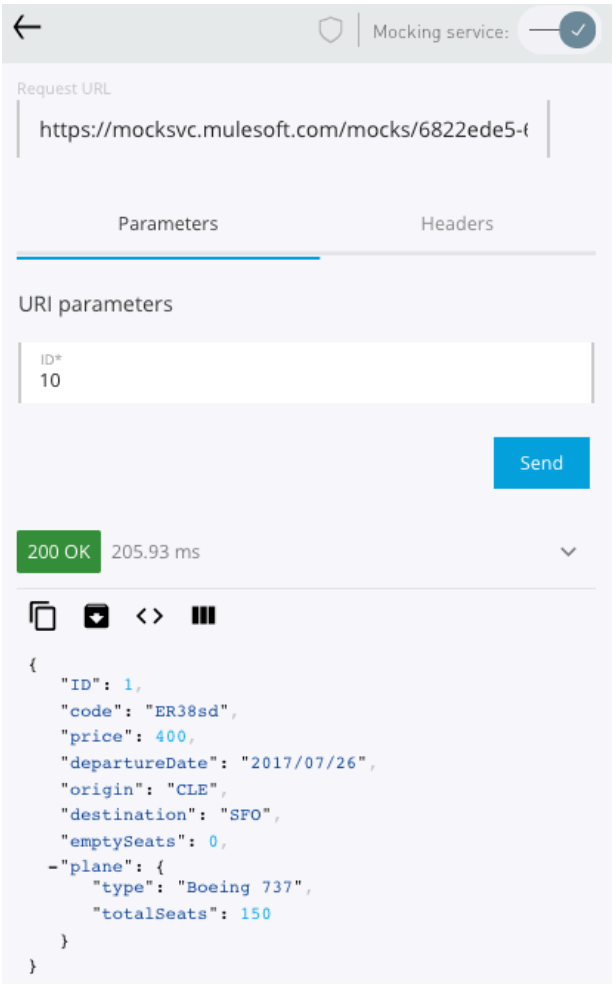
Review and test the /{ID}:get resource in API console

42. In API console, return to the /{ID}:get method; you should now see the response will be of type AmericanFlight.



43. Select the Examples tab; you should see the example AmericanFlightExample data.

44. Click the Try it button, enter an ID, and click Send; you should now see the example flight data returned.



Specify the /flights:post method request to require an AmericanFlight object

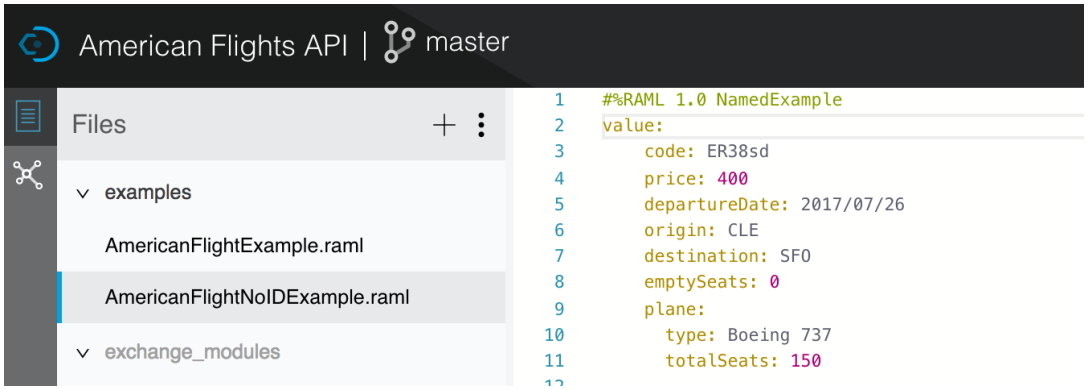
45. In the editor, indent under the /flights post method.
46. In the shelf, click body > application/json > type > AmericanFlight.

```
24    post:
25      body:
26        application/json:
27          type: AmericanFlight
```

Define an example request body for the /flights:post method

47. Return to AmericanFlightExample.raml and copy all the code.
48. In the file browser, click the add button next to the examples folder and select New file.

49. In the Add new file dialog box, set the following values:
    - Version: RAML 1.0
    - Type: Example
    - File name: AmericanFlightNoIDExample.raml
50. Click Create.
51. Delete any code in the new file and then paste the code you copied.
52. Delete the line of code containing the ID.



53. Return to american-flights-api.raml.
54. In the post method, go to a new line under type and add an example element.
55. Use an include statement to set the example to examples/AmericanFlightNoIDExample.raml.

```
post:
  body:
    application/json:
      type: AmericanFlight
      example: !include examples/AmericanFlightNoIDExample.raml
```

Specify an example response for the /flights:post method

56. Go to a new line of code at the end of the /flights:post method and indent to the same level as body.

57. Add a 201 response of type application/json.

```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
29    responses:
30      201:
31        body:
32          application/json:
33
```

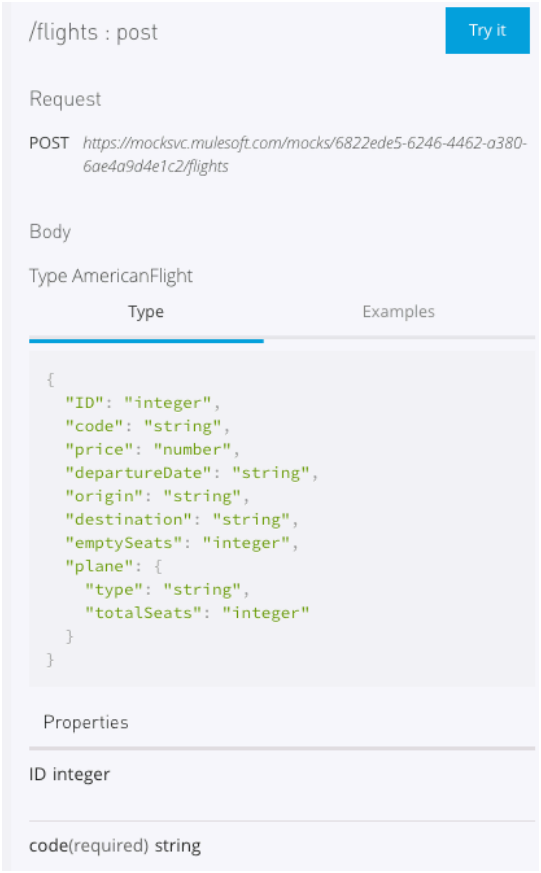
58. In the shelf, click example.
59. Indent under example and add a message property equal to the string: Flight added (but not really).

```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
29    responses:
30      201:
31        body:
32          application/json:
33            example:
34              message: Flight added (but not really)
```

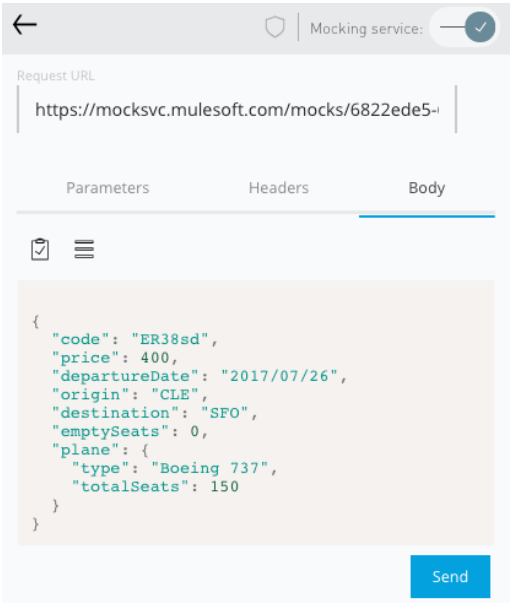
Review and test the /flights:post resource in API console

60. In API console, return to the /flights:post method.

61. Look at the request information; you should now see information about the body - that it is type AmericanFlight and it has an example.

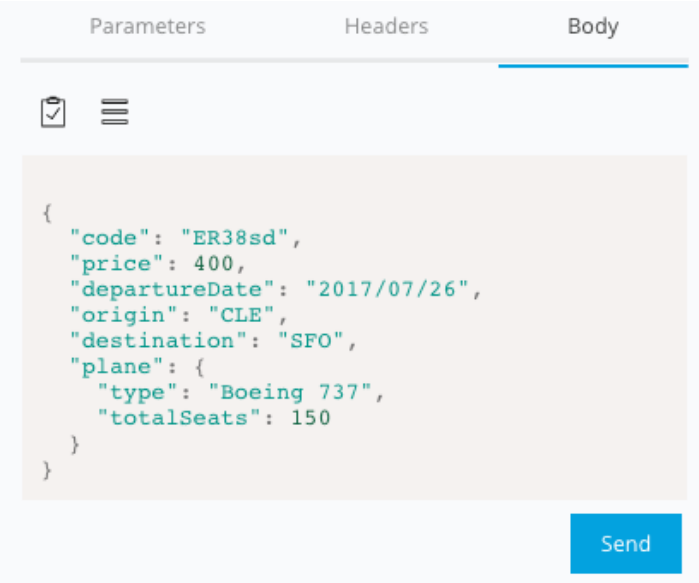


62. Click the Try it button and select the Body tab again; you should now see the example request body.

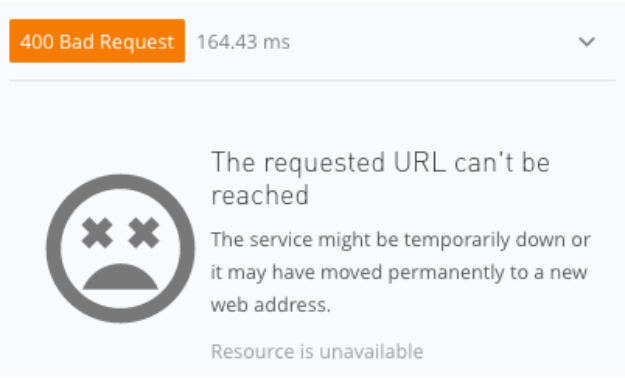


63. Click the Send button; you should now see a response with a 201 status code and the example message.

64. In the body, remove the emptySeats properties.



65. Click Send again; you should get a 400 Bad Request message.

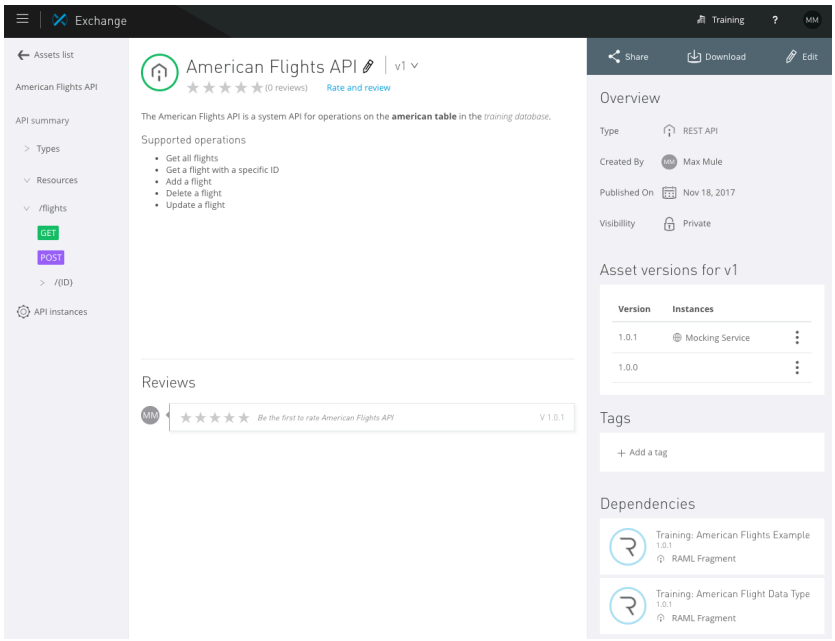


66. Add the emptySeats property back to the body.

Walkthrough 3-4: Add an API to Anypoint Exchange

In this walkthrough, you make an API discoverable by adding it to Anypoint Exchange. You will:

- Publish an API to Exchange from API designer.
- Review an auto-generated API portal in Exchange and test the API.
- Add information about an API to its API portal.
- Create and publish a new API version to Exchange.



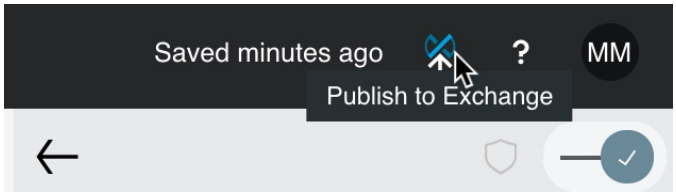
Remove the baseUri by turning off the mocking service

1. Return to API designer.
2. Click the slider to turn off the mocking service; the RAML code should no longer have a baseUri.

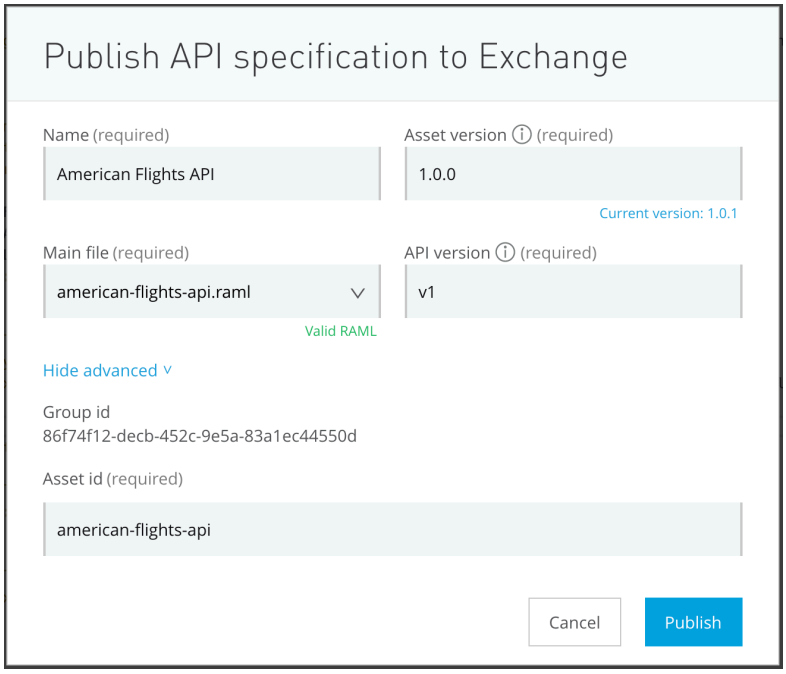
```
1    #%RAML 1.0
2    title: American Flights API
3
```

Publish the API to Anypoint Exchange from API designer

3. Click the Publish to Exchange button.



4. In the Publish API specification to Exchange dialog box, leave the default values:
    - Name: American Flights API
    - Main file: american-flights-api.raml
    - Asset version: 1.0.0
    - API version: v1
5. Click the Show advanced link.
6. Look at the ID values.
7. Click Publish.



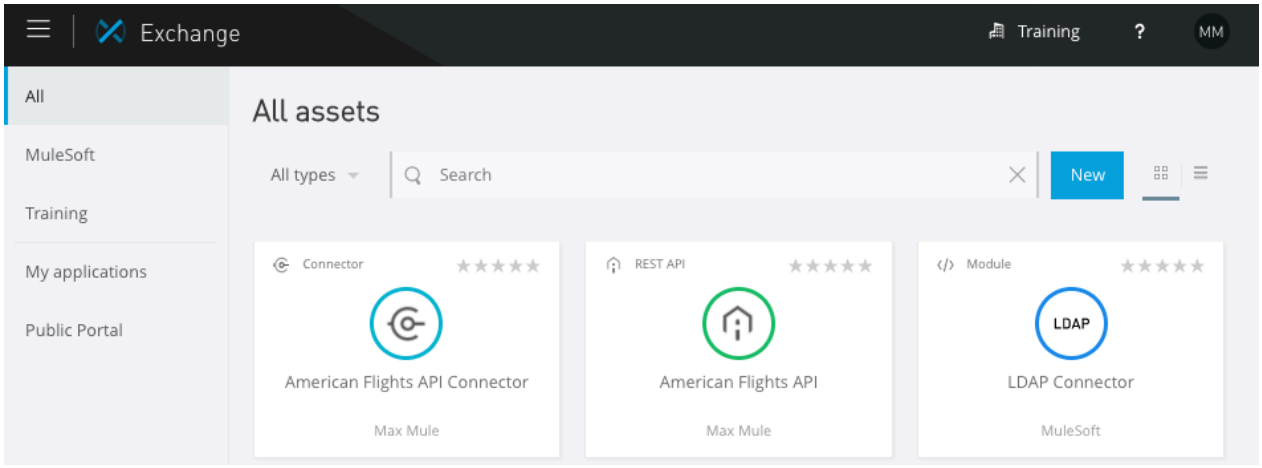
8. Click the Publish button.
9. In the Publish API specification to Exchange dialog box, click Done.
10. Look at the RAML file; you should see a version has been added.

```
1   #%RAML 1.0
2   version: v1
3   title: American Flights API
```

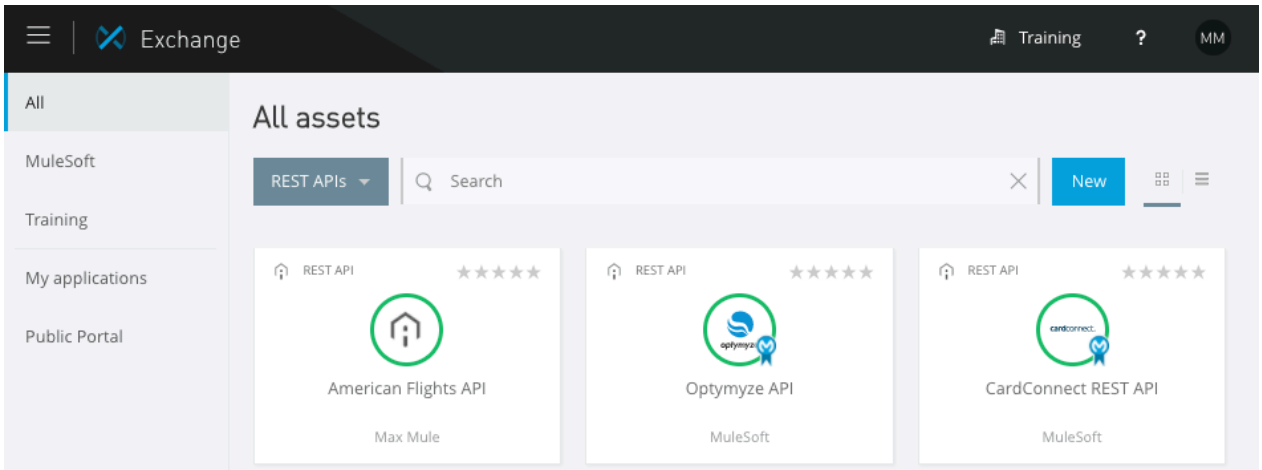
Locate your API in Anypoint Exchange

11. Return to Design Center.

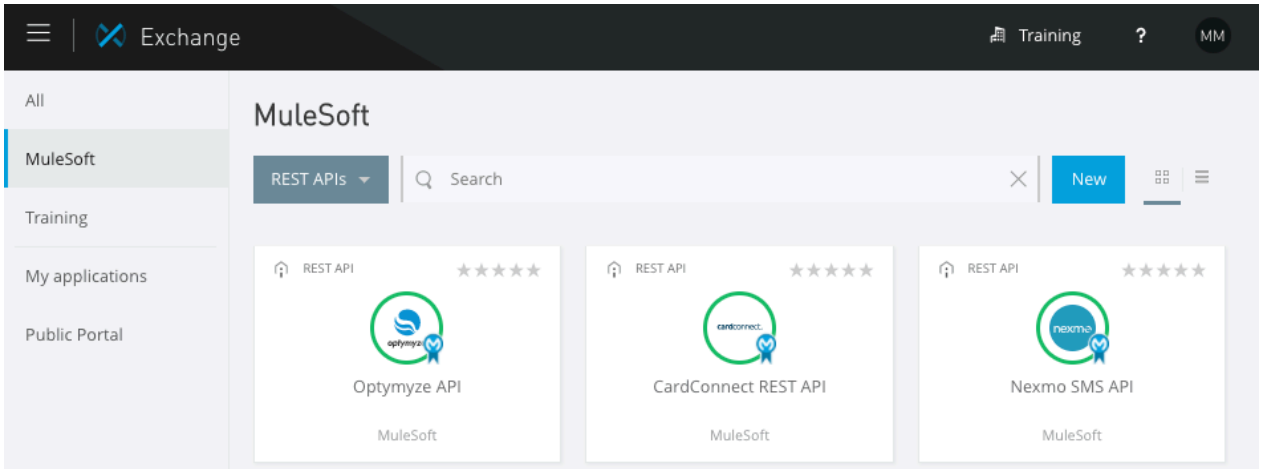
12. In the main menu, select Exchange; you should see your American Flights API.



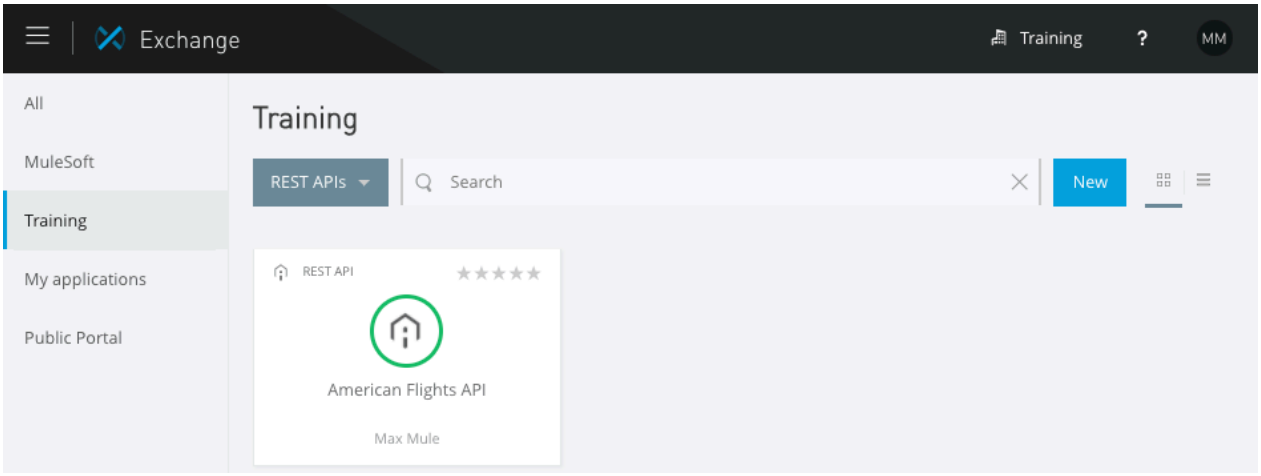
13. In the types drop-down menu, select REST APIs; you should still see your API.



14. In the left-side navigation, click MuleSoft; you should not see your American Flights API in the public Exchange (just the Training: American Flights API).

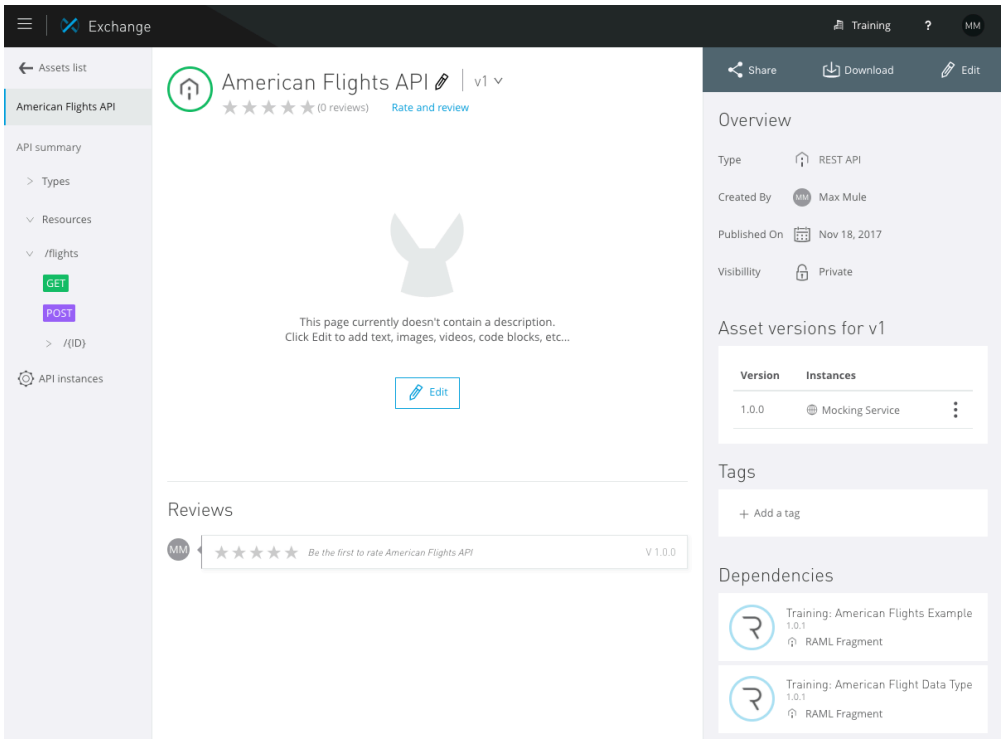


15. In the left-side navigation, click the name of your organization (Training in the screenshots); you should see your American Flights API in your private Exchange.



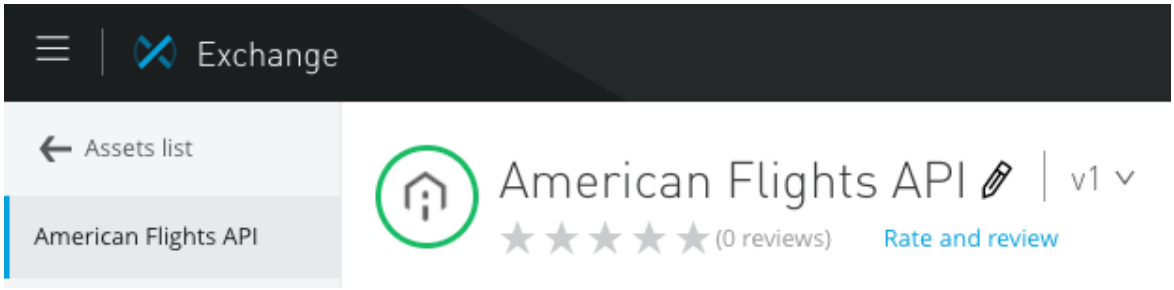
Review the auto-generated API portal

16. Click the American Flights API.
17. Review the page; you should see that as the creator of this API, you can edit, review, share, download, and add tags to this version.

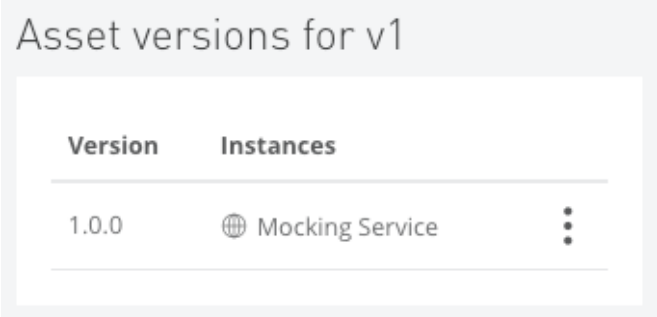


18. Locate the API dependencies in the lower-right corner.

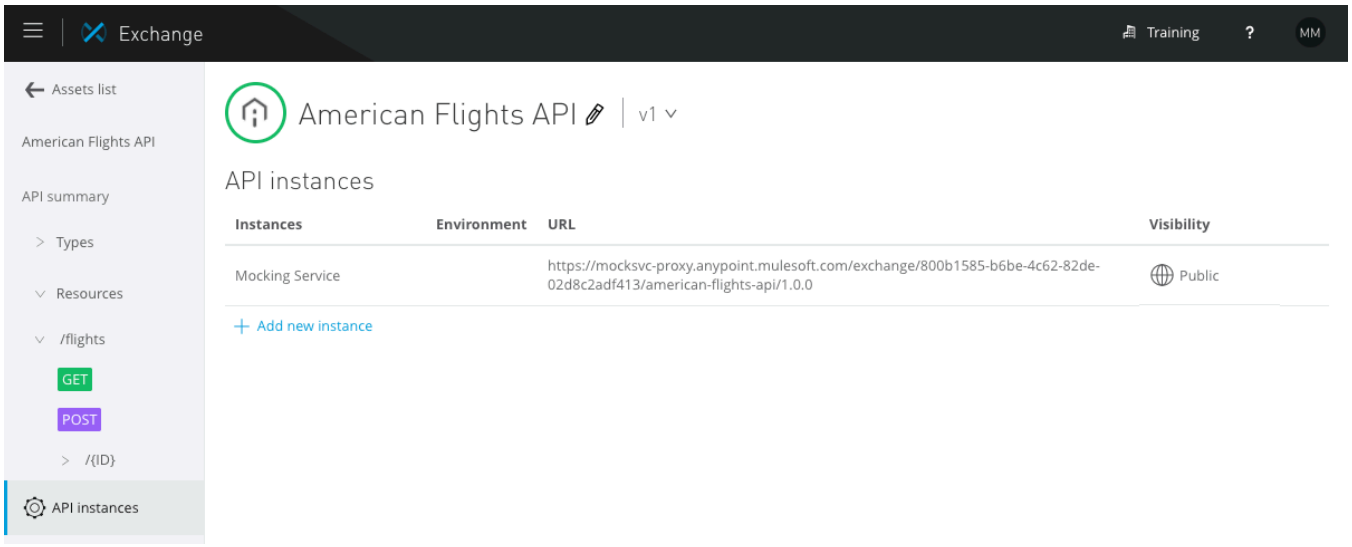
19. Locate the API version (v1) next to the name of the API at the top of the page.



20. Locate the asset versions for v1; you should see one version (1.0.0) of this API specification (v1) has been published and there is one API instance for it and that uses the mocking service.

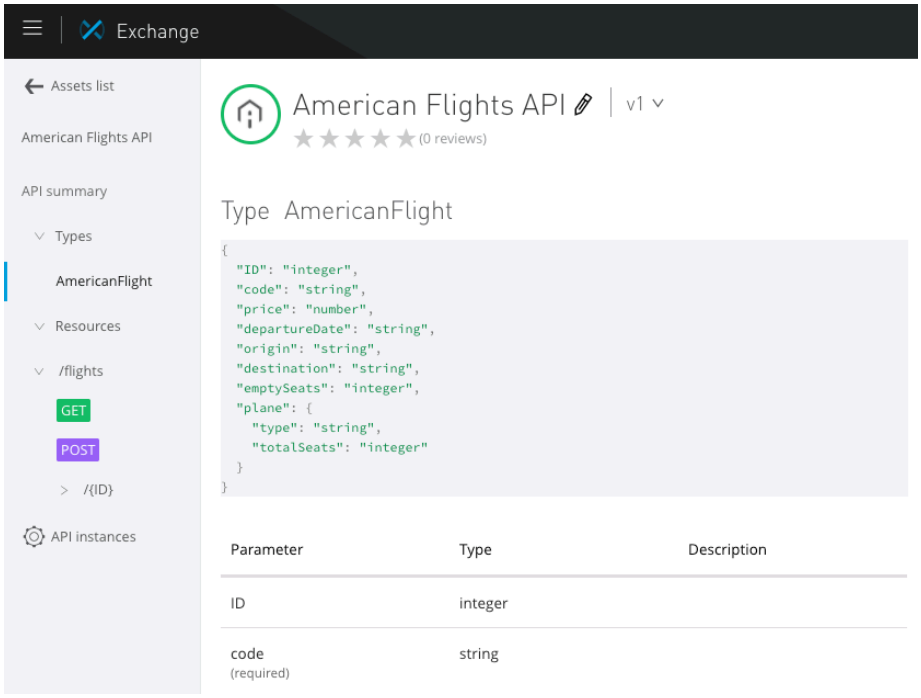


21. In the left-side navigation, click API instances; you should see information for the API instance generated from the API specification using the mocking service.



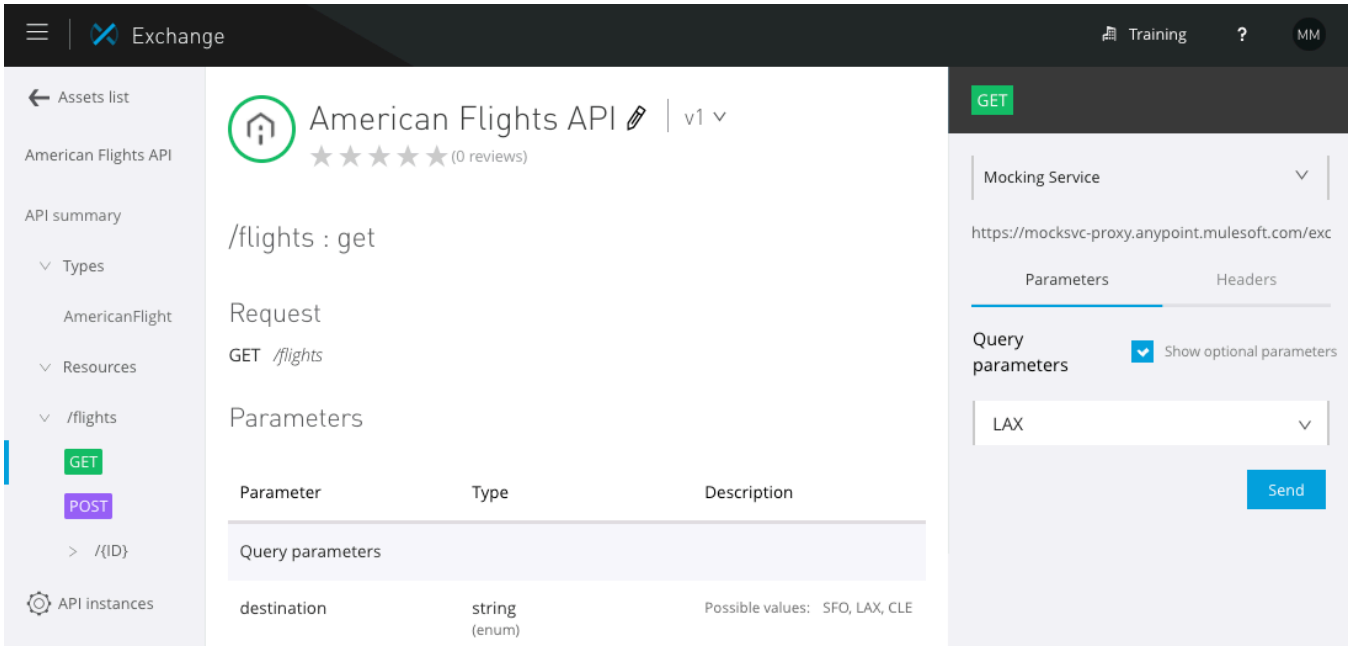
22. In the left-side navigation, expand Types.

23. Click AmericanFlight and review the information.

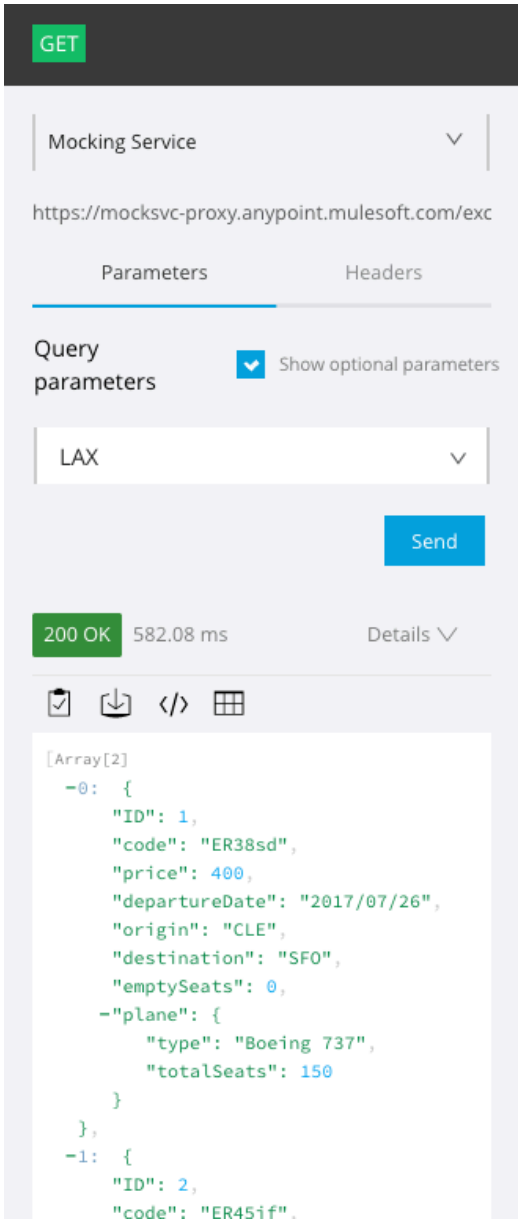


Test the API in its API portal in Exchange

24. In the left-side navigation, click the /flights GET method; you should now see an API Reference section to test the method on the right side of the page.
25. In the API Reference, select to show optional query parameters.
26. Click the destination drop-down and select a value.



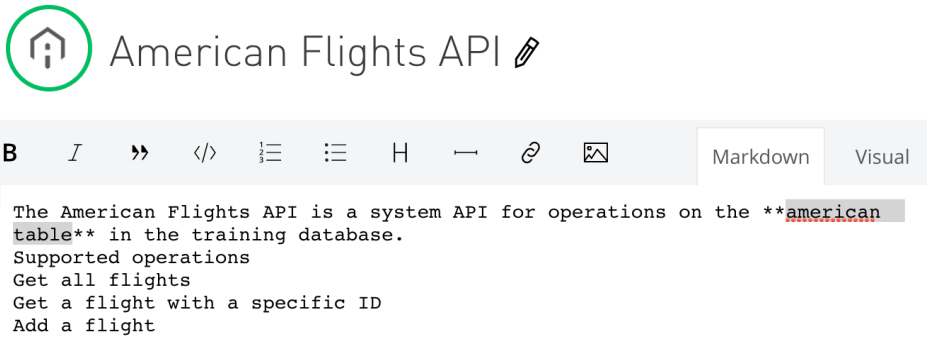
27. Click Send; you should get a 200 response and the example data displayed – just as you did in API console in API designer.



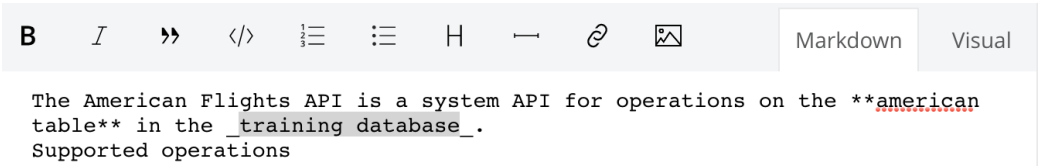
Add information about the API

28. In the left-side navigation, click the name of the API: American Flights API.
29. Click one of the Edit buttons for the API.
30. Return to the course snippets.txt file and copy the text for the American Flights API description text.
31. Return to the editor in Anypoint Exchange and paste the content.

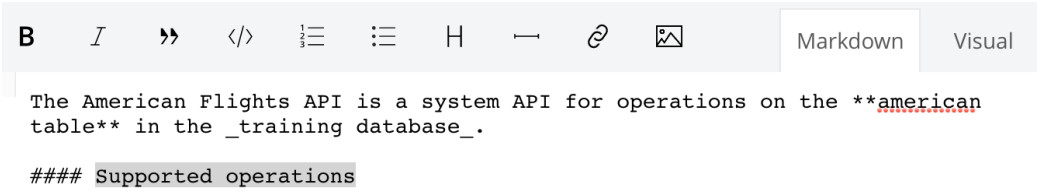
32. Select the words american table and click the strong button (the B).



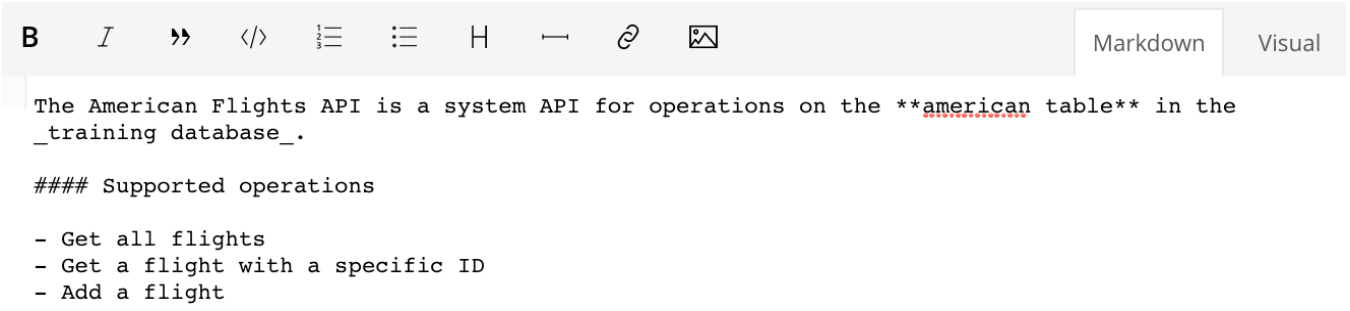
33. Select the words training database and click the emphasis button (the I).



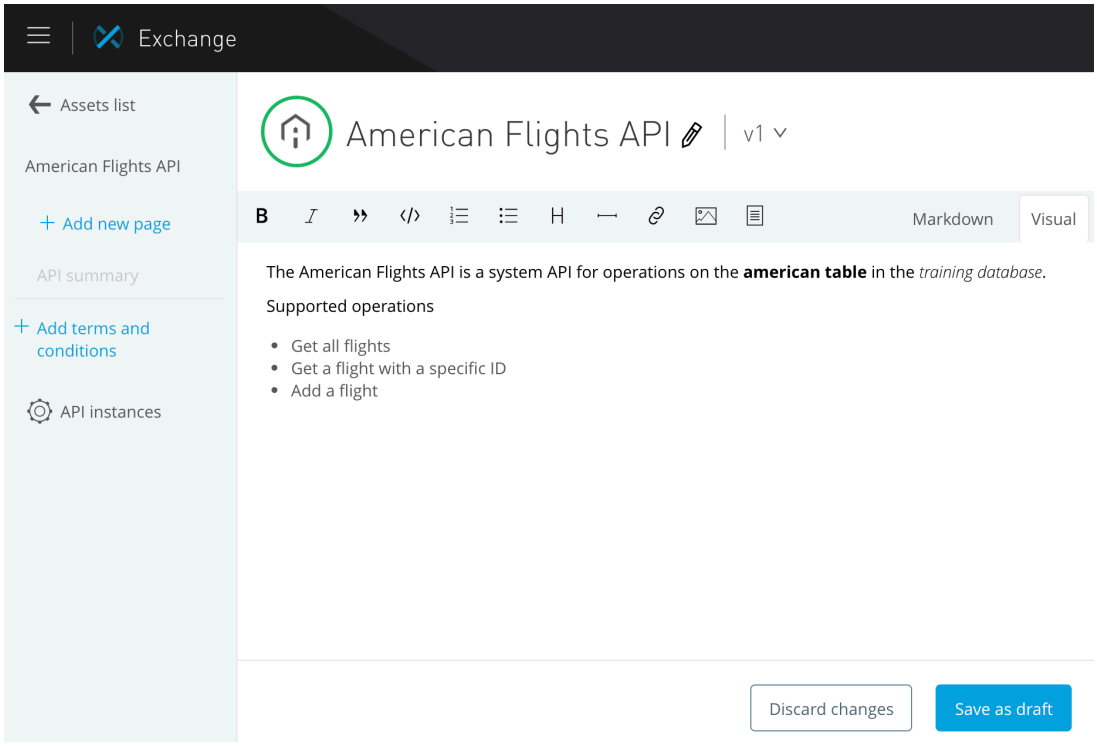
34. Select the words Supported operations and click the heading button four times (the H).



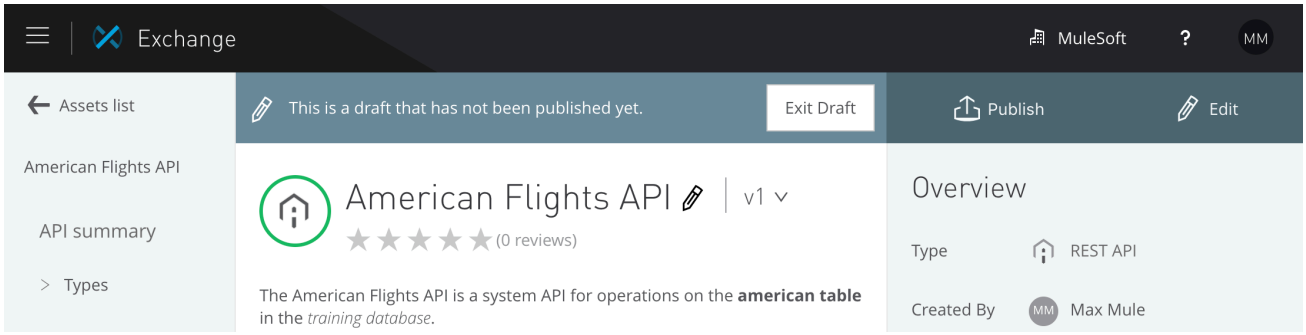
35. Select Get all flights and click the bulleted list button.
36. Repeat for the other operations.



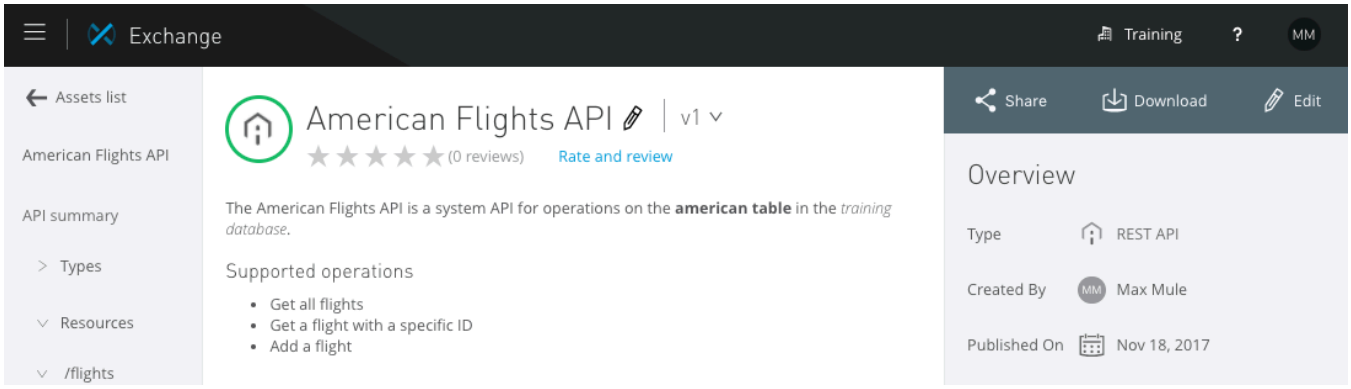
37. Click the Visual tab in the editor toolbar and view the rendered markdown.



38. Click the Save as draft button; you should now see buttons to exit the draft or publish it.



39. Click the Publish button; you should now see the new information about the API in the API portal.

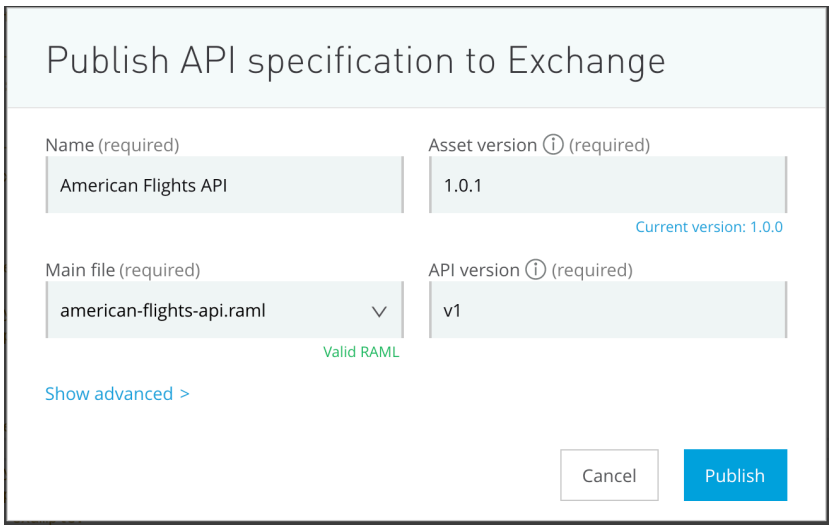


Modify the API and publish the new version to Exchange

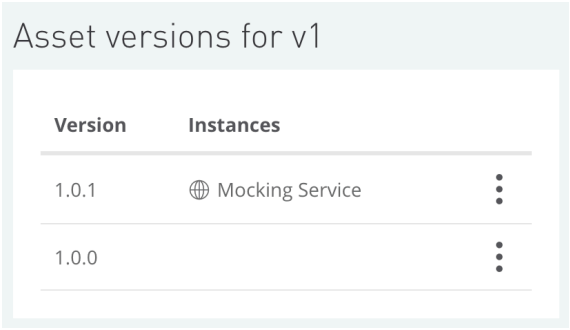
40. Return to your American Flights API in API designer.
41. Return to the course snippets.txt file and copy the American Flights API - /{ID} DELETE and PUT methods.
42. Return to american-flights-api.raml and paste the code after the {ID}/get method.
43. Fix the indentation.
44. Review the code.

```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
            example: !include examples/AmericanFlightExample.raml
  delete:
    responses:
      200:
        body:
          application/json:
            example:
              message: Flight deleted (but not really)
  put:
    body:
      application/json:
        type: AmericanFlight
        example: !include examples/AmericanFlightNoIDExample.raml
    responses:
      200:
        body:
          application/json:
            example:
              message: Flight updated (but not really)
```

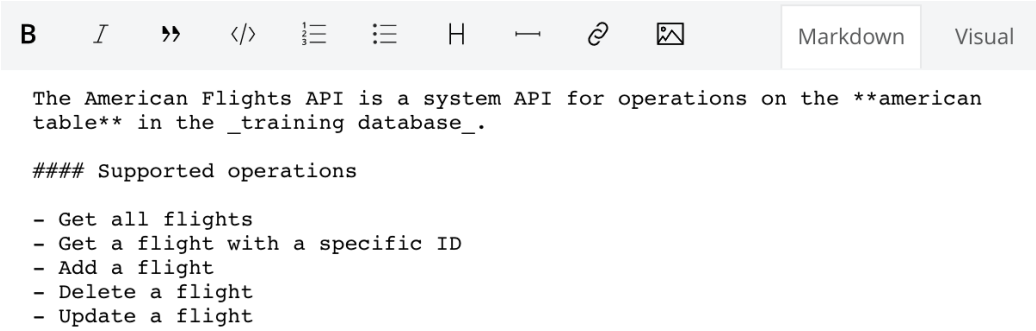
45. Click the Publish to Exchange button.
46. In the Publish API specification to Exchange dialog box, look at the asset version.



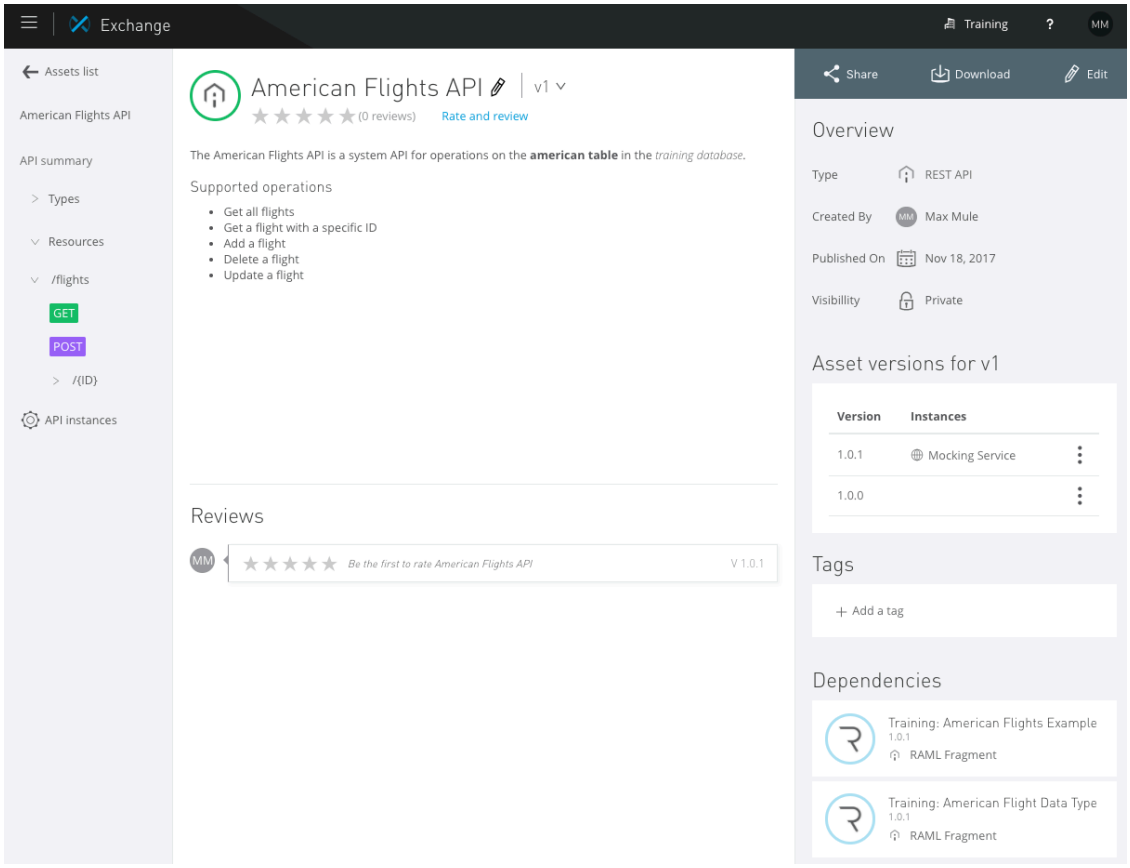
47. Click Publish.
48. In the Publish API specification to Exchange dialog box, click Exchange; Exchange should open in a new browser tab.
49. Locate the asset versions listed for the API; you should see both asset versions of the API listed with an associated API instance using the mocking service for the latest version.



50. Click the Edit button.
51. Add two new operations that delete a flight and update a flight.



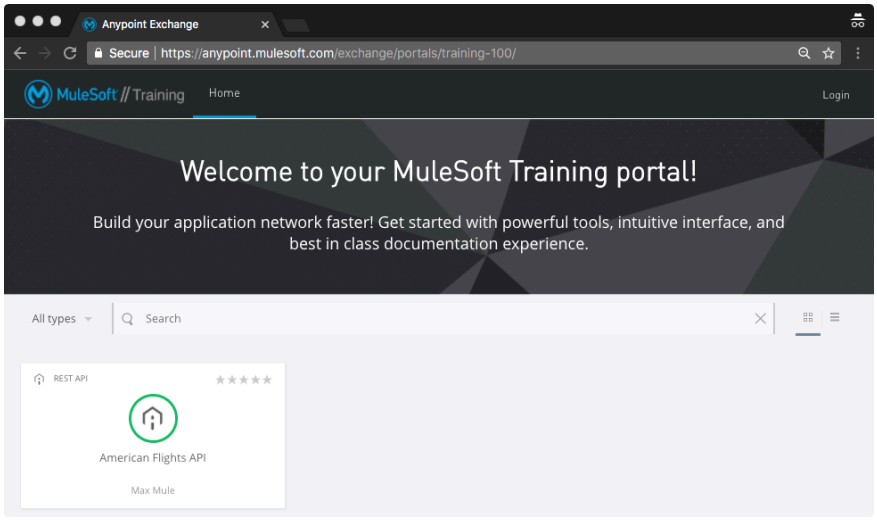
52. Click Save as draft and then Publish.



Walkthrough 3-5: Share an API

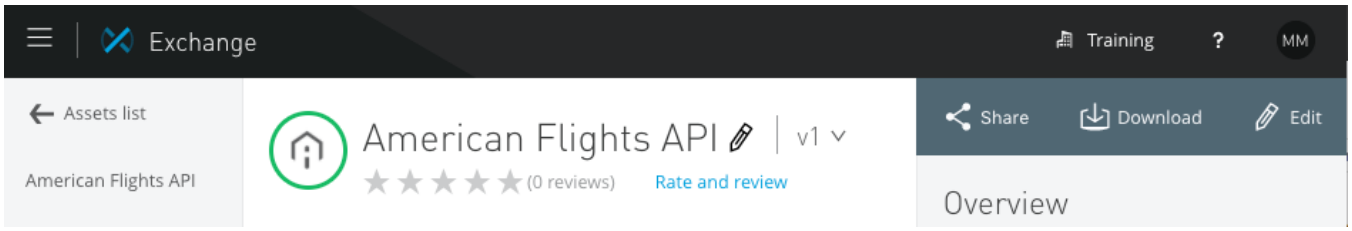
In this walkthrough, you share an API with both internal and external developers to locate, learn about, and try out the API. You will:

- Share an API within an organization using the private Exchange.
- Create a public API portal.
- Customize a public portal.
- Explore a public portal as an external developer.

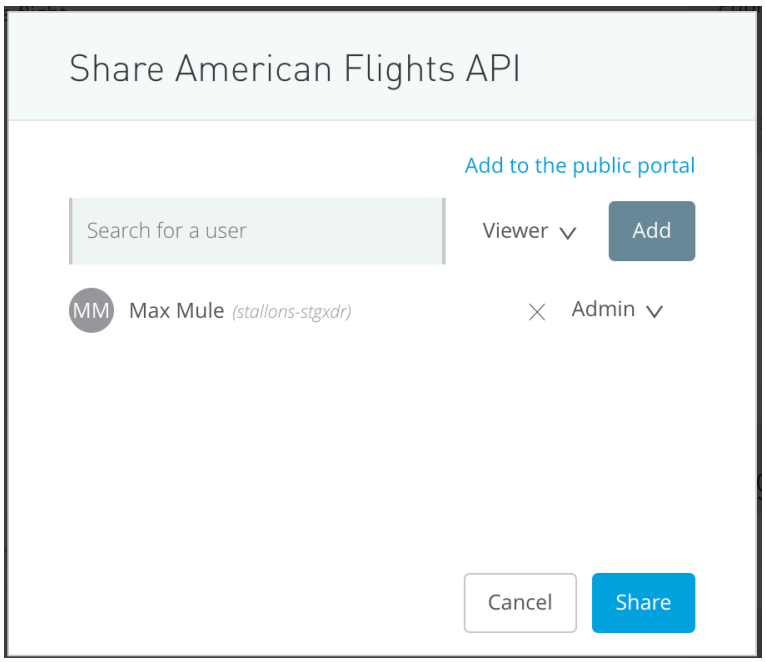


Share the API in the private Exchange with others

1. Return to your American Flights API in Exchange.
2. Click Share.



3. In the Share American Flights API dialog box, you should see that you are an admin for this API.

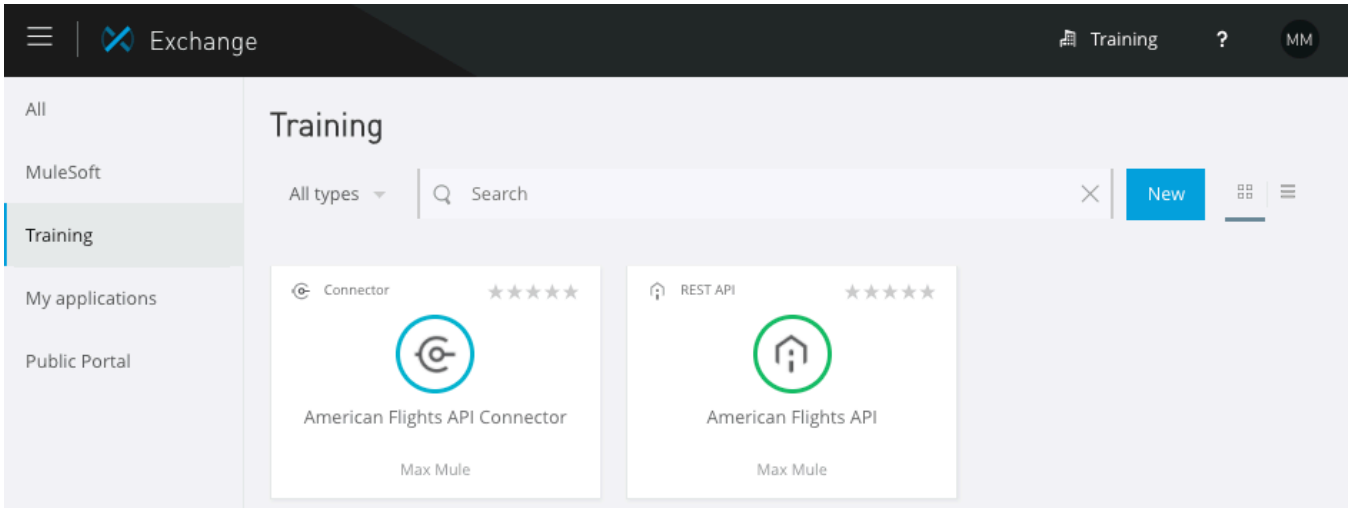


4. Open the Viewer drop-down menu located next to the user search field; you should see that you can add additional users to be of type Viewer, Contributor, or Admin.

   Note: In the future, you will also be able to share an asset with an entire business group.

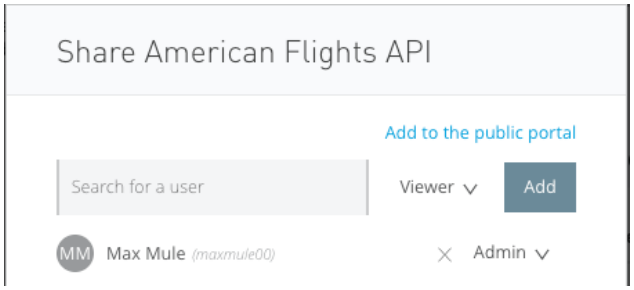
5. Click Cancel.
6. In the left-side navigation, click Assets list.
7. In the left-side navigation, click the name of your organization.
8. Click your American Flights API.

   Note: This is how users you share the API with will find the API.

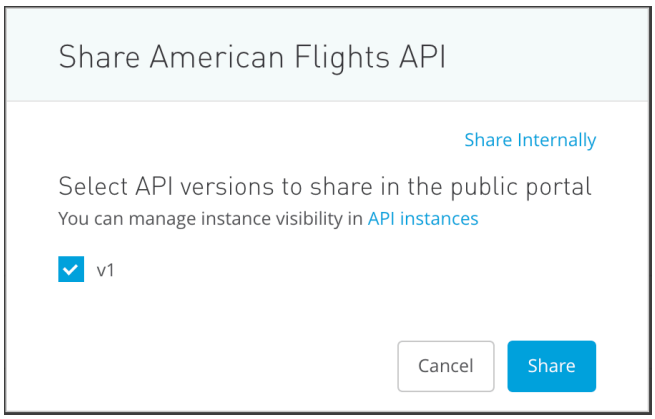


Create a public API portal

9. Click the American Flights API.
10. Click the Share button for the API again.
11. In the Share American Flights API dialog box, click Add to the public portal.

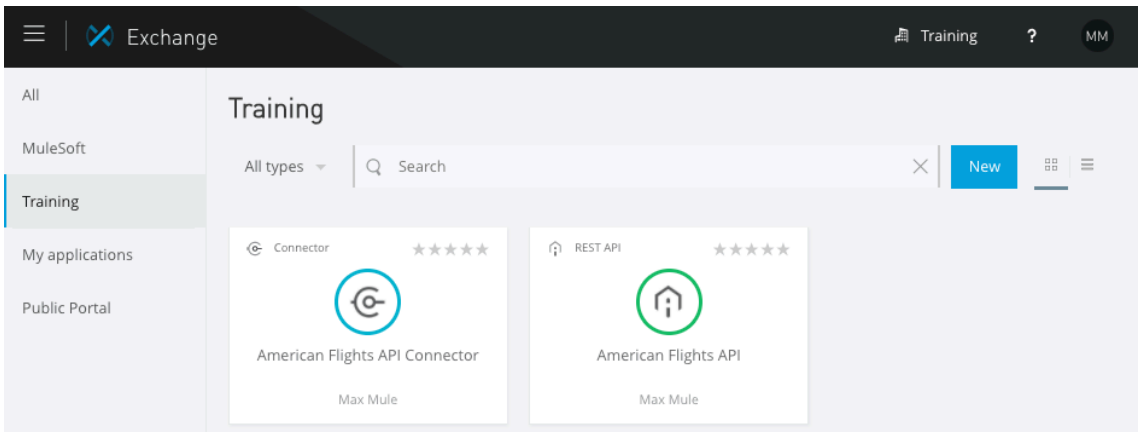


12. In the new dialog box, select to share v1 of the API.
13. Click Share.

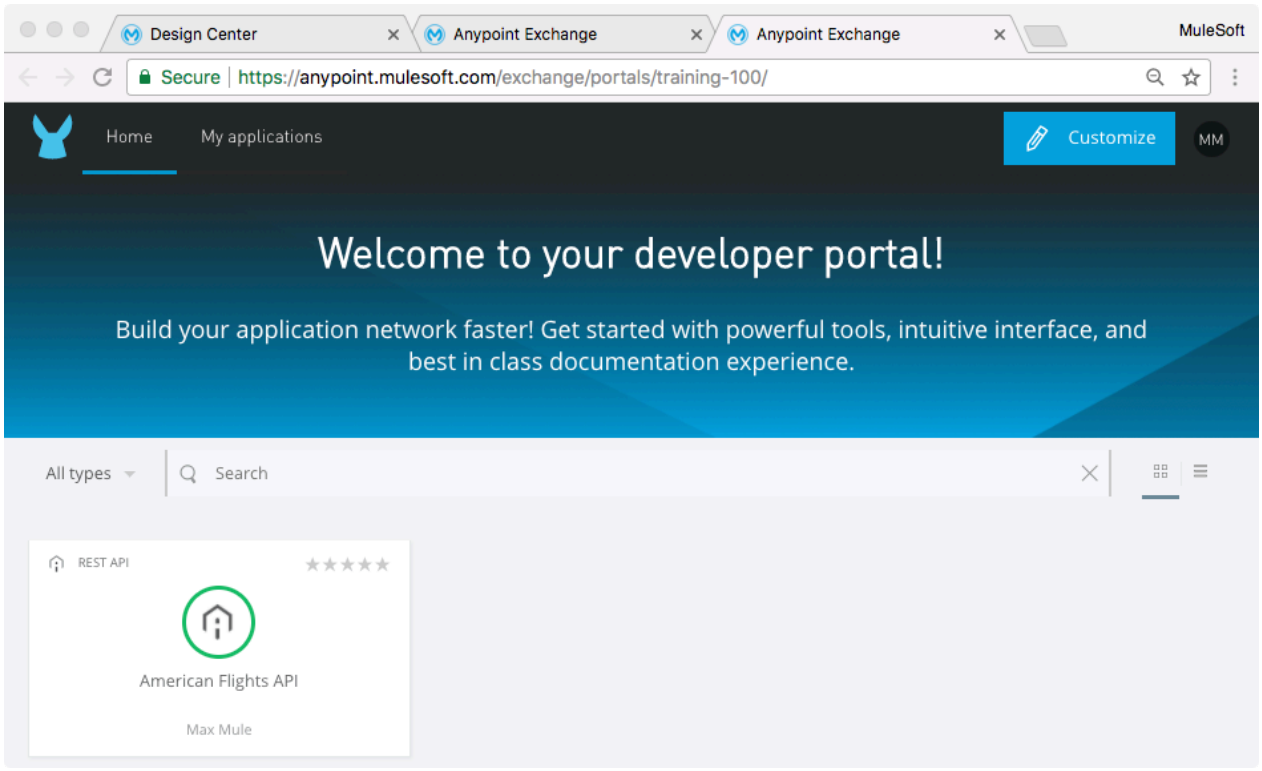


Explore the API in the public portal

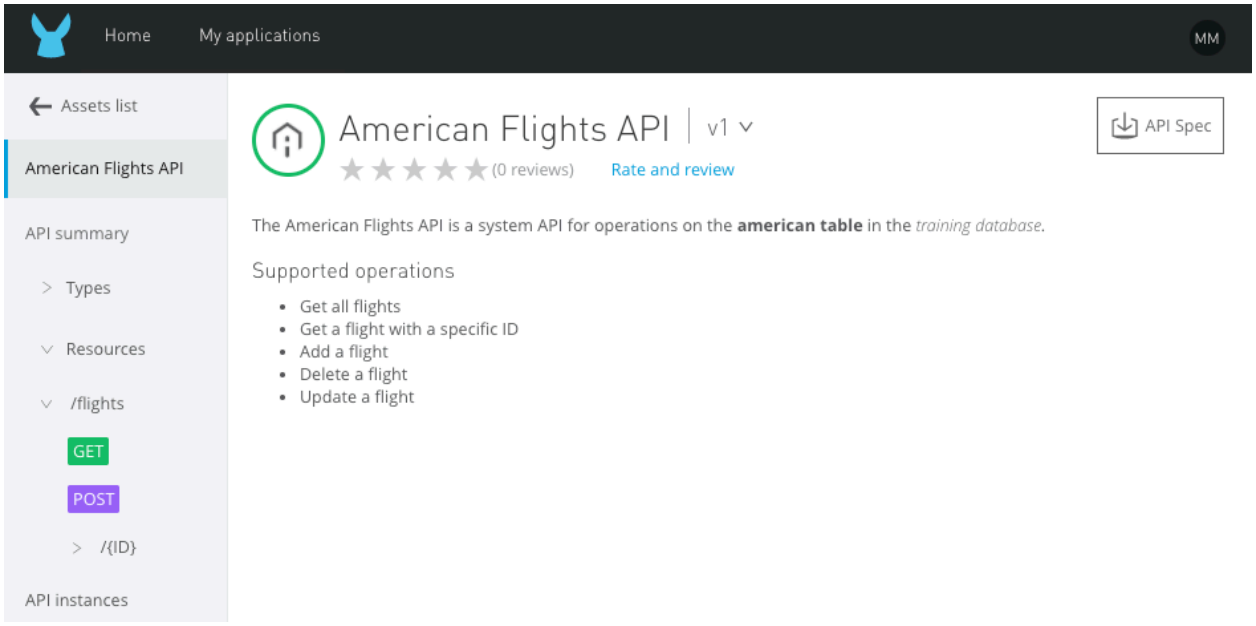
14. In the left-side navigation, click Assets list.
15. In the left-side navigation, click Public Portal.



16. Review the public portal that opens in a new browser tab.
17. Look at the URL for the public portal.

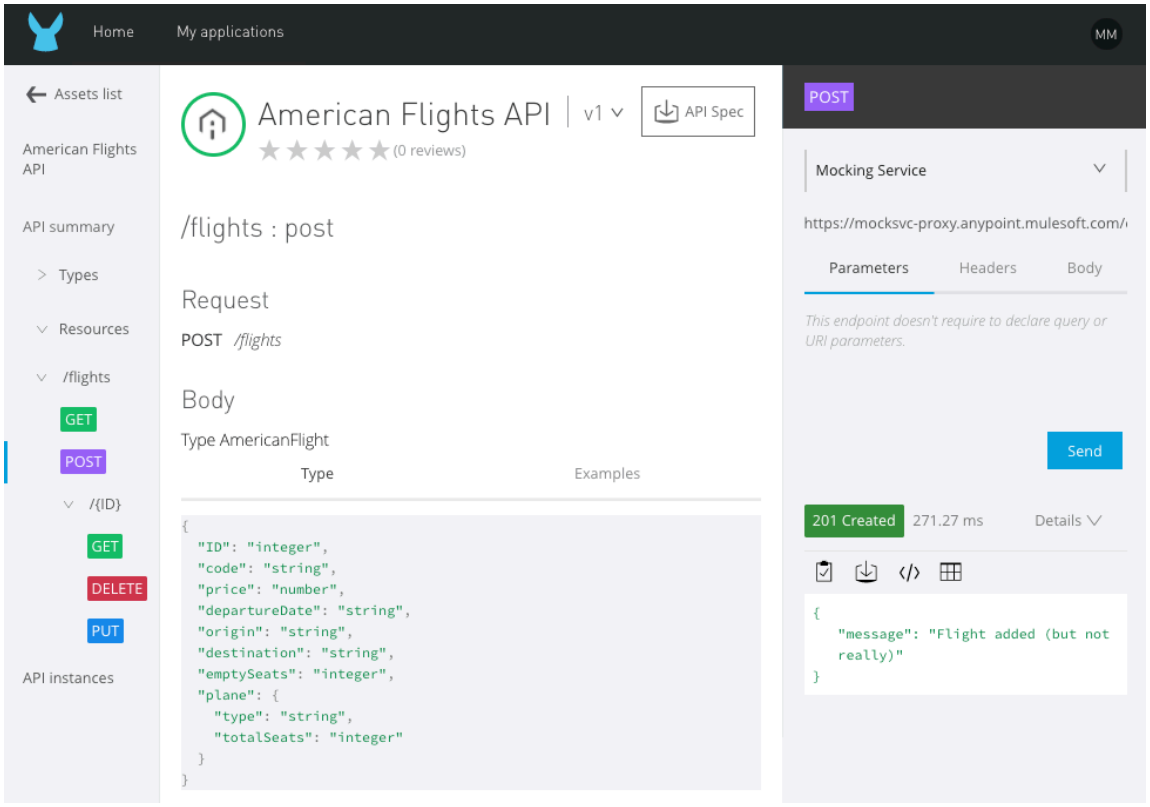


18. Click your American Flights API and review the auto-generated public API portal.



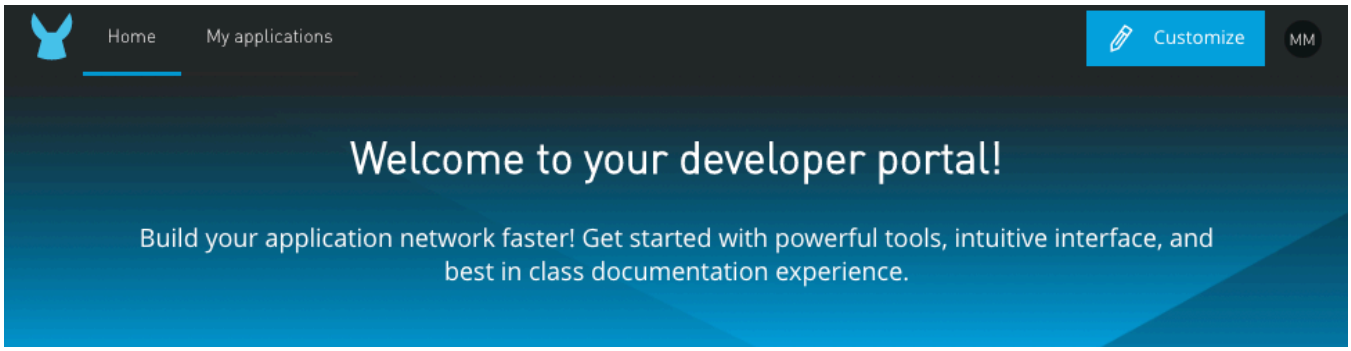
19. In the left-side navigation, click the POST method for the flights resource.
20. Review the body and response information.

21. In the API Reference section, click Send; you should get a 201 response with the example data returned from the mocking service.

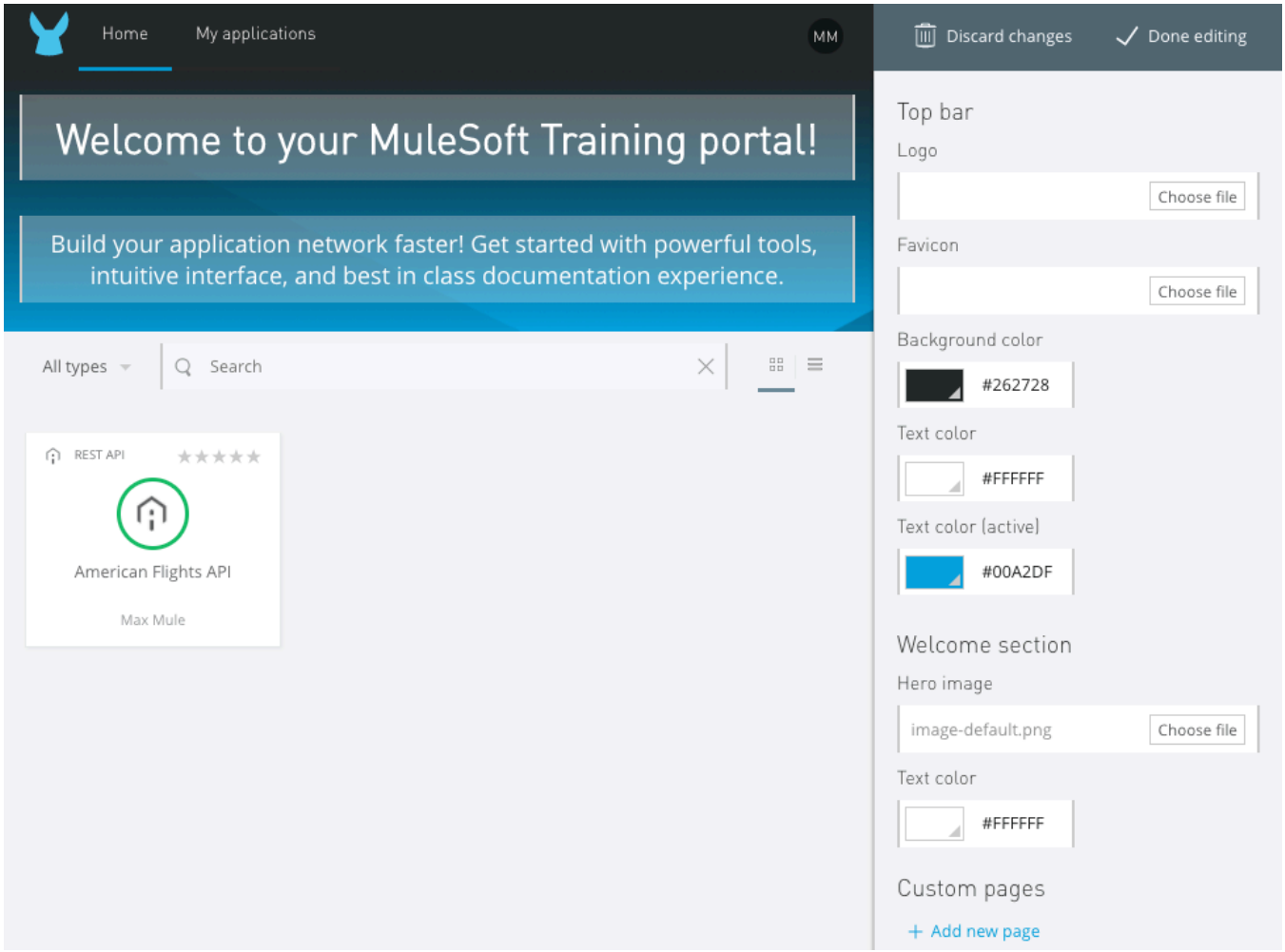


Customize the public portal

22. In the left-side navigation, click Assets list.
23. Click the Customize button.

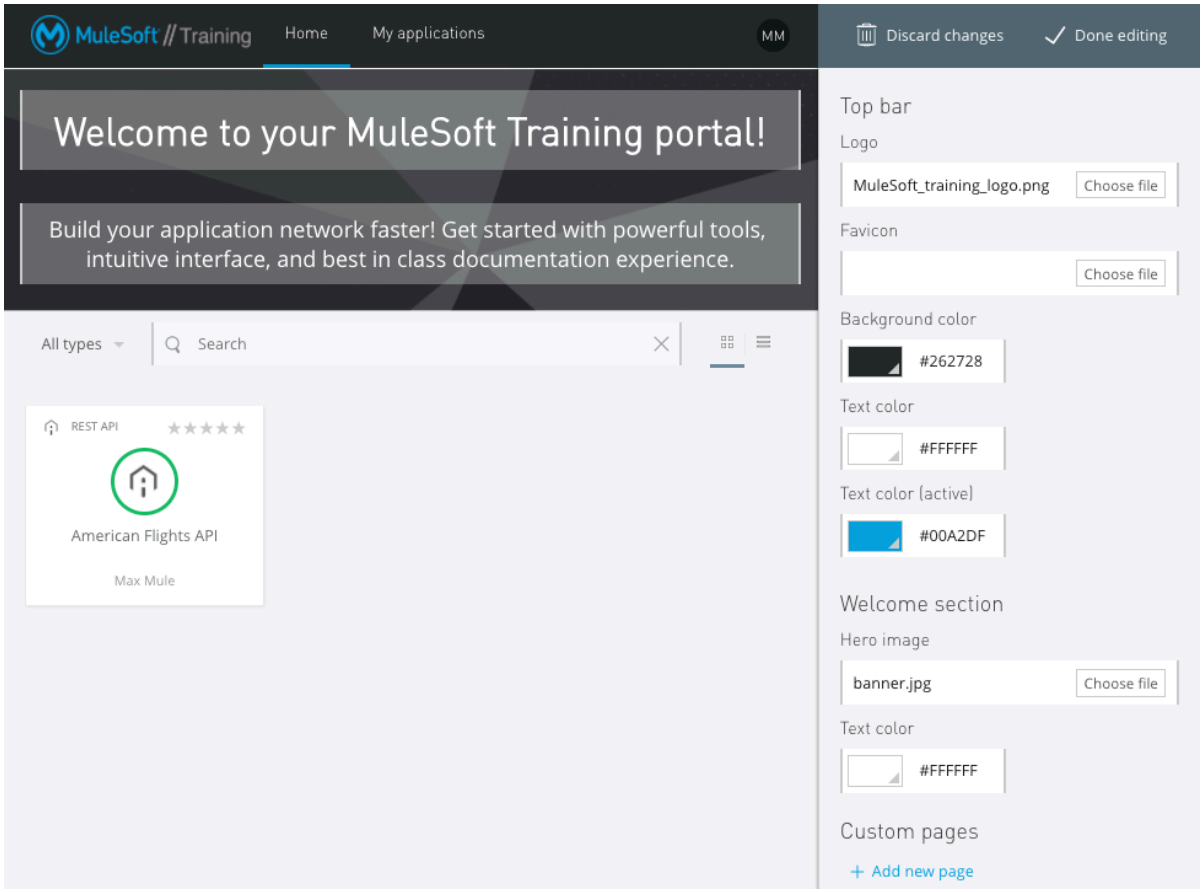


24. Change the text to Welcome to your MuleSoft Training portal!



25. In the logo field, click Choose file.
26. In the file browser dialog box, navigate to the student files and locate the MuleSoft_training_logo.png file in the resources folder and click Open.
27. Locate the new logo in the preview.
28. In the hero image field, click Choose file.
29. In the file browser dialog box, navigate to the student files and locate the banner.jpg file in the resources folder and click Open.

30. Review the new logo and banner in the preview.

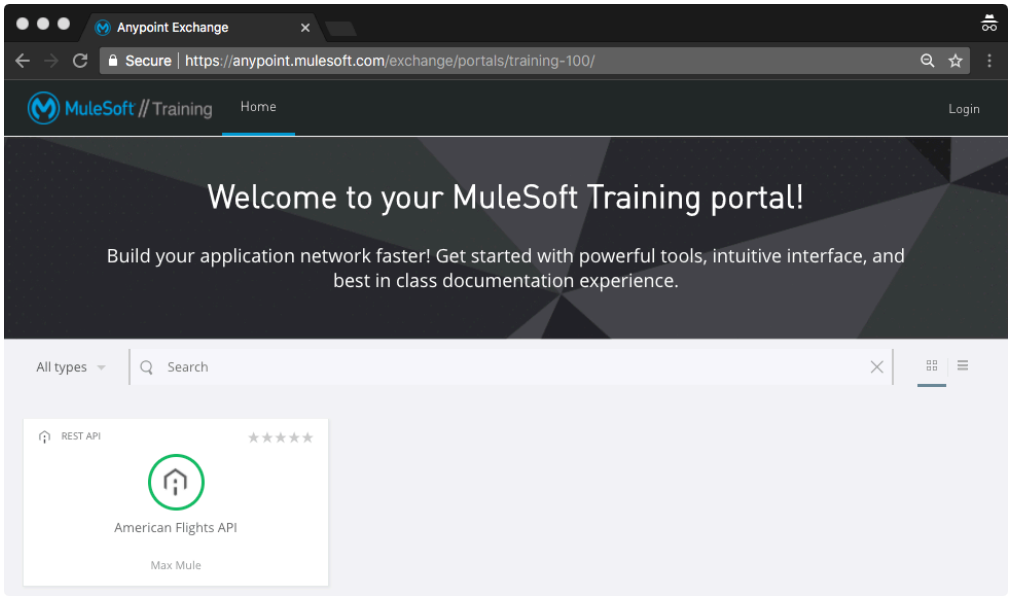


31. Change any colors that you want.
32. Click the Done editing button.
33. In the Publish changes dialog box, click Yes, publish; you should see your customized public portal.

Explore the public portal as an external developer

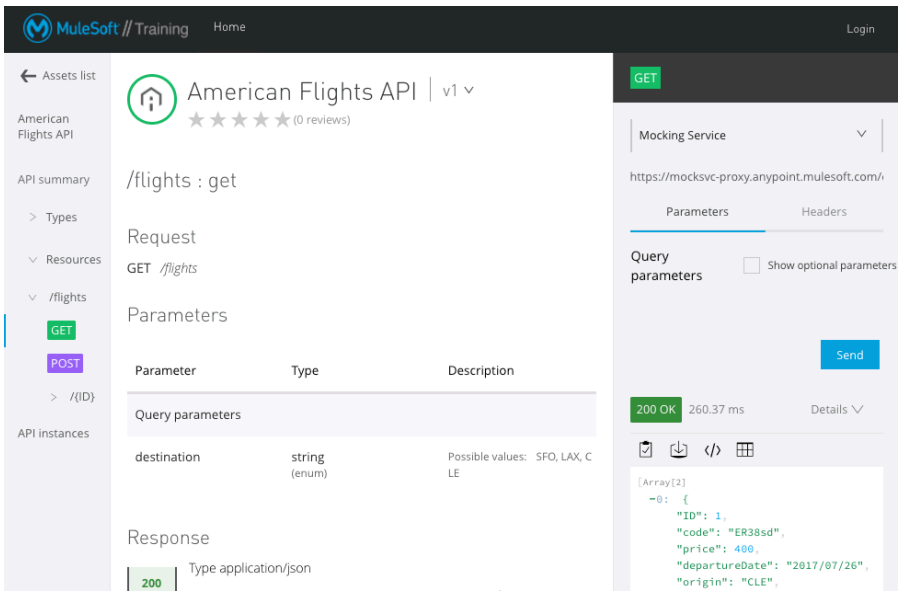
34. In the browser, copy the URL for the public portal.
35. Open a new private or incognito window in your browser.

36. Navigate to the portal URL you copied; you should see the public portal (without the customize button).

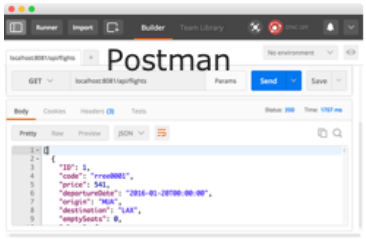


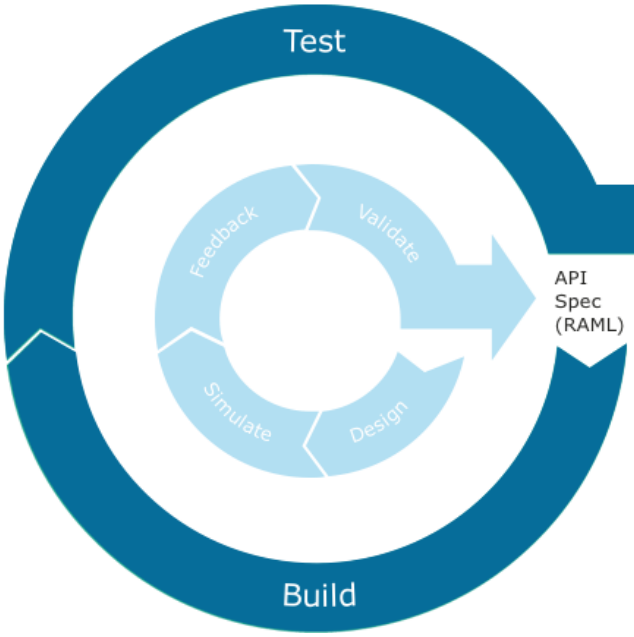
37. Click the American Flights API.
38. Explore the API portal.
39. Make a call to one of the resource methods.

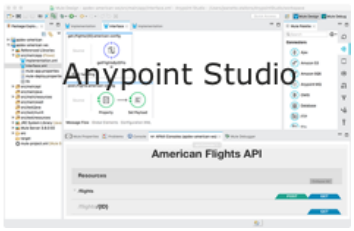
    Note: As an anonymous user, you can make calls to an API instance that uses the mocking service but not managed APIs.



Module 4: Building APIs







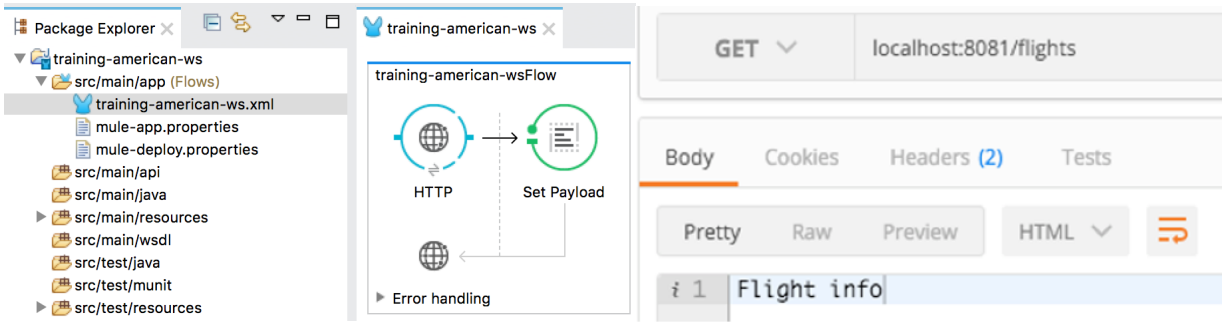
At the end of this module, you should be able to:

- Use Anypoint Studio to build, run, and test Mule applications.
- Use a connector to connect to databases.
- Use the graphical DataWeave editor to transform data.
- Create RESTful interfaces for applications from RAML files.
- Connect API interfaces to API implementations.

Walkthrough 4-1: Create a Mule application with Anypoint Studio

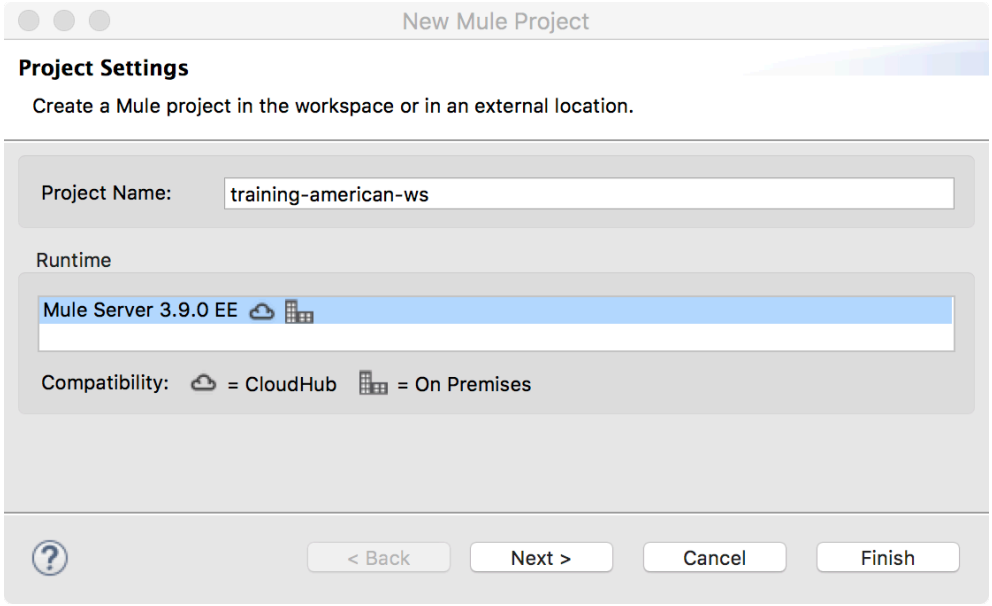
In this walkthrough, you build a Mule application. You will:

- Create a new Mule project with Anypoint Studio.
- Add a connector to receive requests at an endpoint.
- Set the message payload.
- Run a Mule application using the embedded Mule runtime.
- Make an HTTP request to the endpoint using Postman.



Create a Mule project

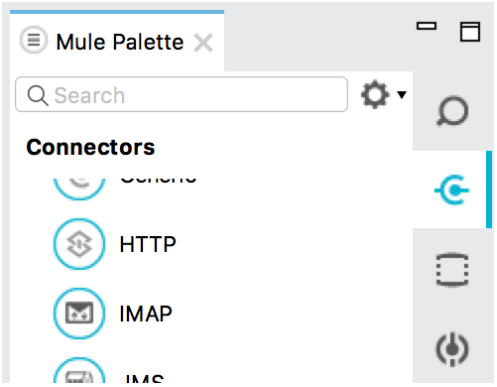
1. Open Anypoint Studio.
2. Select File > New > Mule Project.
3. In the New Mule Project dialog box, set the Project Name to training-american-ws.
4. Ensure the Runtime is set to the latest version of Mule.



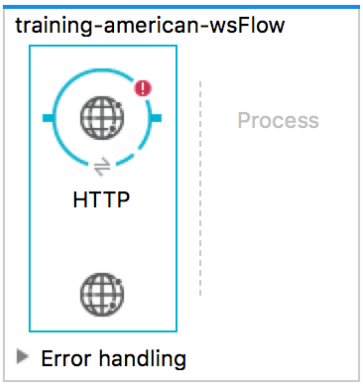
5. Click Finish.

Create an HTTP connector endpoint to receive requests

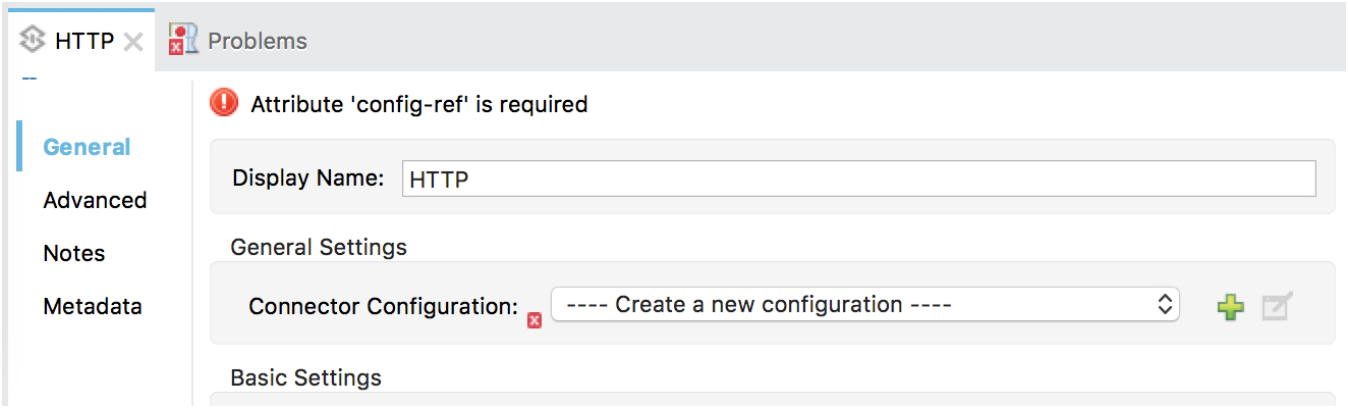
6. In the Mule Palette, select the Connectors tab.



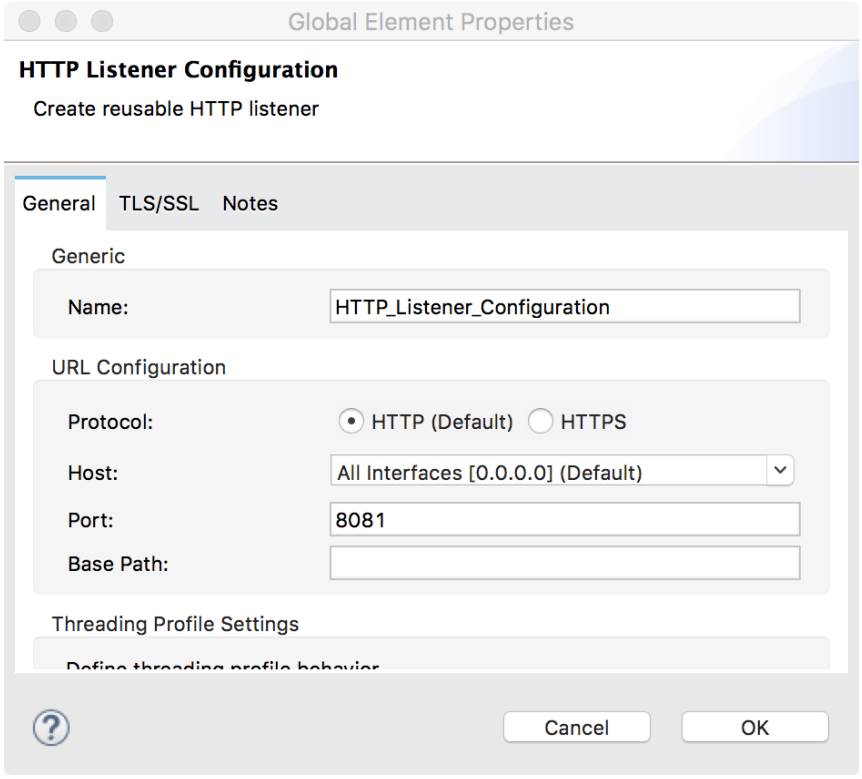
7. Drag an HTTP connector from the Mule Palette to the canvas.



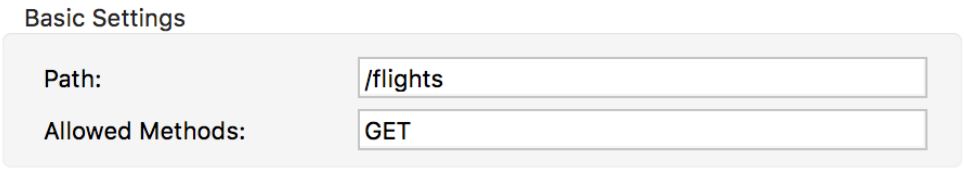
8. Double-click the HTTP endpoint.
9. In the HTTP Properties view that opens at the bottom of the window, click the Add button next to connector configuration.



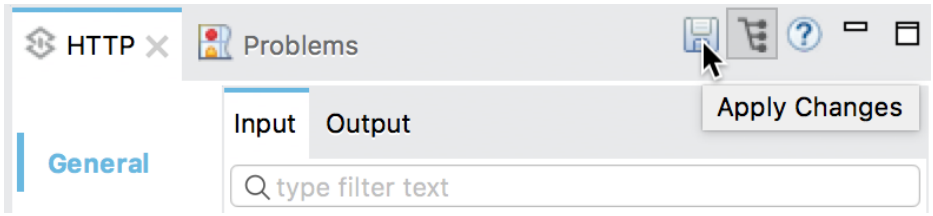
10. In the Global Element Properties dialog box, look at the default values and click OK.



11. In the HTTP properties view, set the path to /flights.
12. Set the allowed methods to GET.

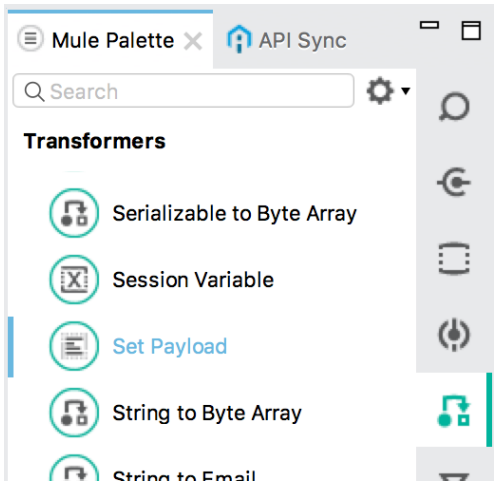


13. Click the Apply Changes button; the errors in the Problems view should disappear.

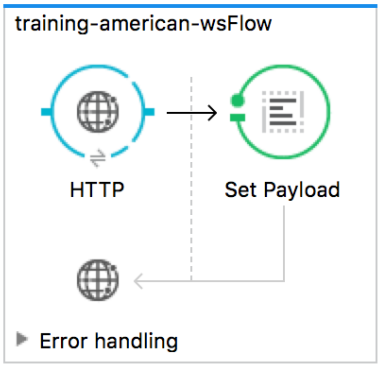


Display data

14. In the Mule Palette, select the Transformers tab.

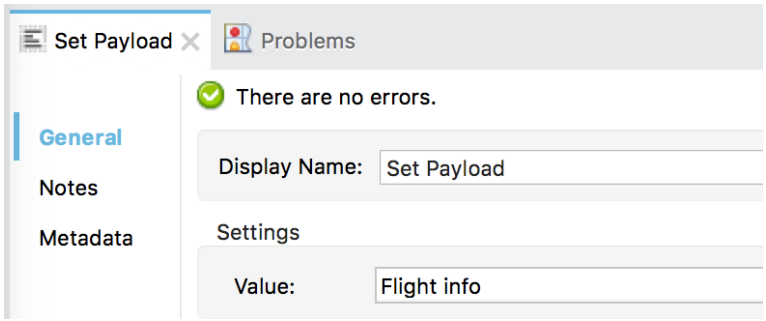


15. Drag a Set Payload transformer from the Mule Palette into the process section of the flow.



Configure the Set Payload transformer

16. In the Set Payload properties view, set the value field to Flight info.



17. Click the Configuration XML link at the bottom of the canvas and examine the corresponding XML.

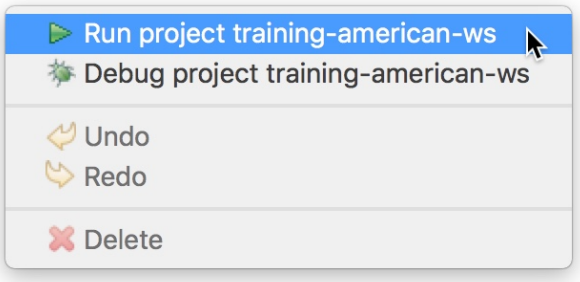
*training-american-ws

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns="http://www.mulesoft.or
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfram
http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/current
http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current
    <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081" d
    <flow name="training-american-wsFlow">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedMe
        <set-payload value="Flight info
" doc:name="Set Payload"/>
    </flow>
</mule>
```

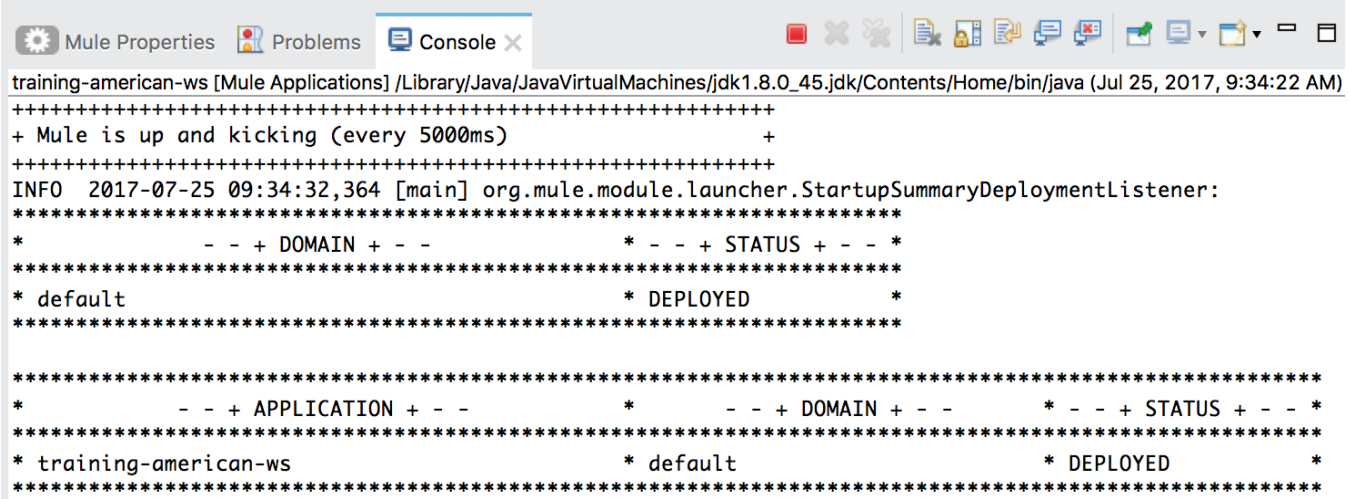
18. Click the Message Flow link to return to the canvas.
19. Click the Save button or press Cmd+S or Ctrl+S.

Run the application

20. Right-click in the canvas and select Run project training-american-ws.

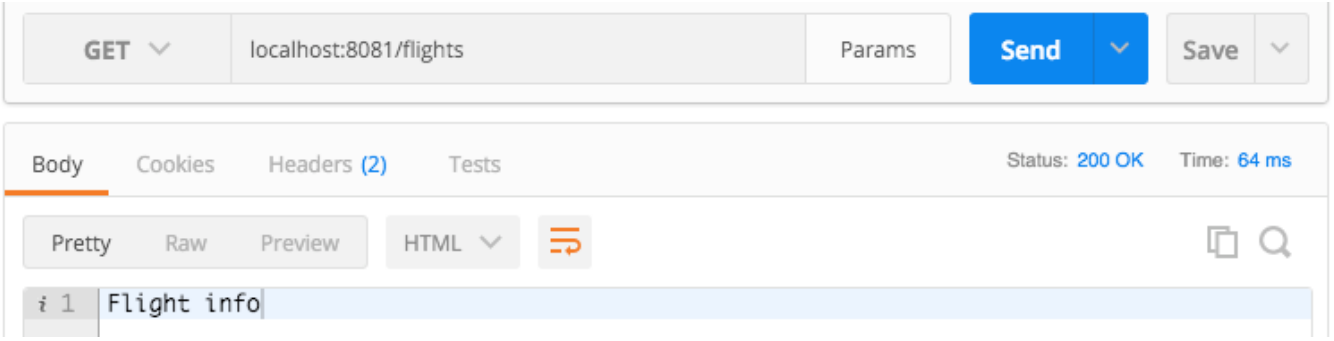


21. Watch the Console view; it should display information letting you know that both the Mule runtime and the training-american-ws application started.

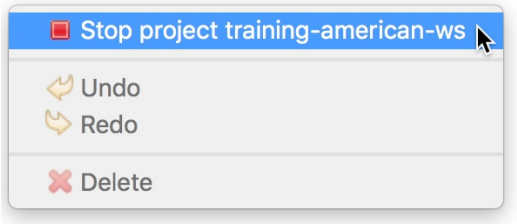


Test the application

22. Return to Postman.
23. Make sure the method is set to GET and that no headers or body are set for the request.
24. Make a GET request to http://localhost:8081/flights; you should see Flight info displayed.



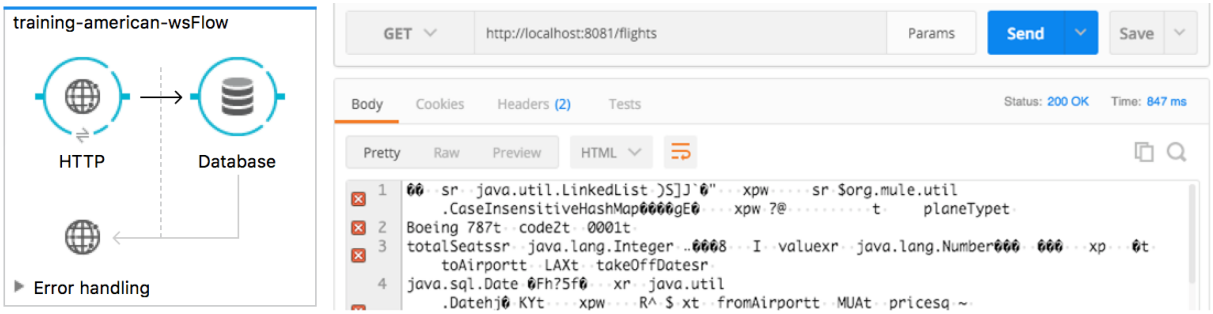
25. Return to Anypoint Studio.
26. Right-click in the canvas and select Stop project training-american-ws.



Walkthrough 4-2: Connect to data (MySQL database)

In this walkthrough, you connect to a database and retrieve data from a table that contains flight information. You will:

- Add a Database connector endpoint.
- Configure a Database connector that connects to a MySQL database (or optionally an in-memory Derby database if you do not have access to port 3306).
- Configure the Database endpoint to use that Database connector.
- Write a query to select data from a table in the database.



Locate database information

1. Return to the course snippets.txt file and locate the MySQL and Derby database information.

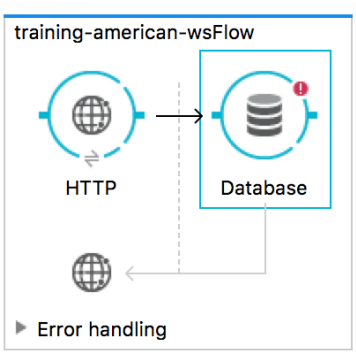
```
* MySQL database
db.host = mudb.mulesoft-training.com
db.port = 3306
db.user = mule
db.password = mule
db.database = training
American table: american
American table version2: flights
Account table: accounts
Account list URL:  http://mu.mulesoft-training.com/essentials/accounts/show
or http://localhost:9090/essentials/accounts/show.html

* Derby database
driverName: org.apache.derby.jdbc.ClientDriver
url: jdbc:derby://localhost:1527/memory:training
```

Note: The database information you see may be different than what is shown here; the values in the snippets file differ for instructor-led and self-study training classes.

Add a Database connector endpoint

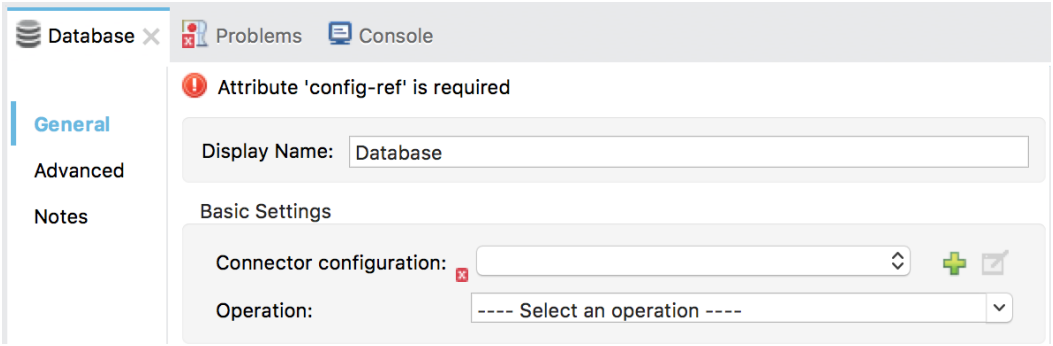
2. Return to Anypoint Studio.
3. Right-click the Set Payload message processor and select Delete.

4. In the Mule Palette, select the Connectors tab.
5. Drag a Database connector to the process section of the flow.

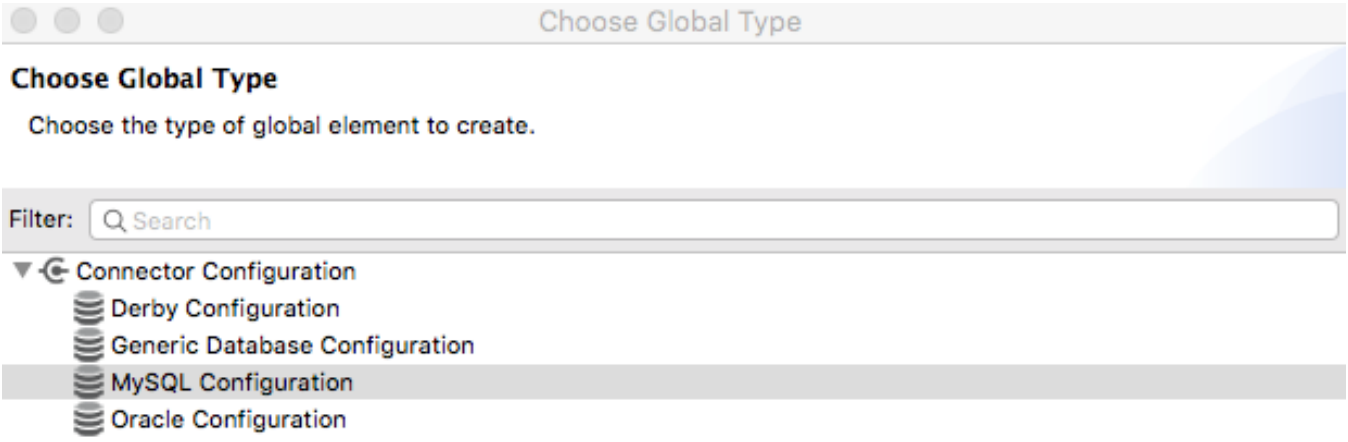


Option 1: Configure a MySQL Database connector (if you have access to port 3306)

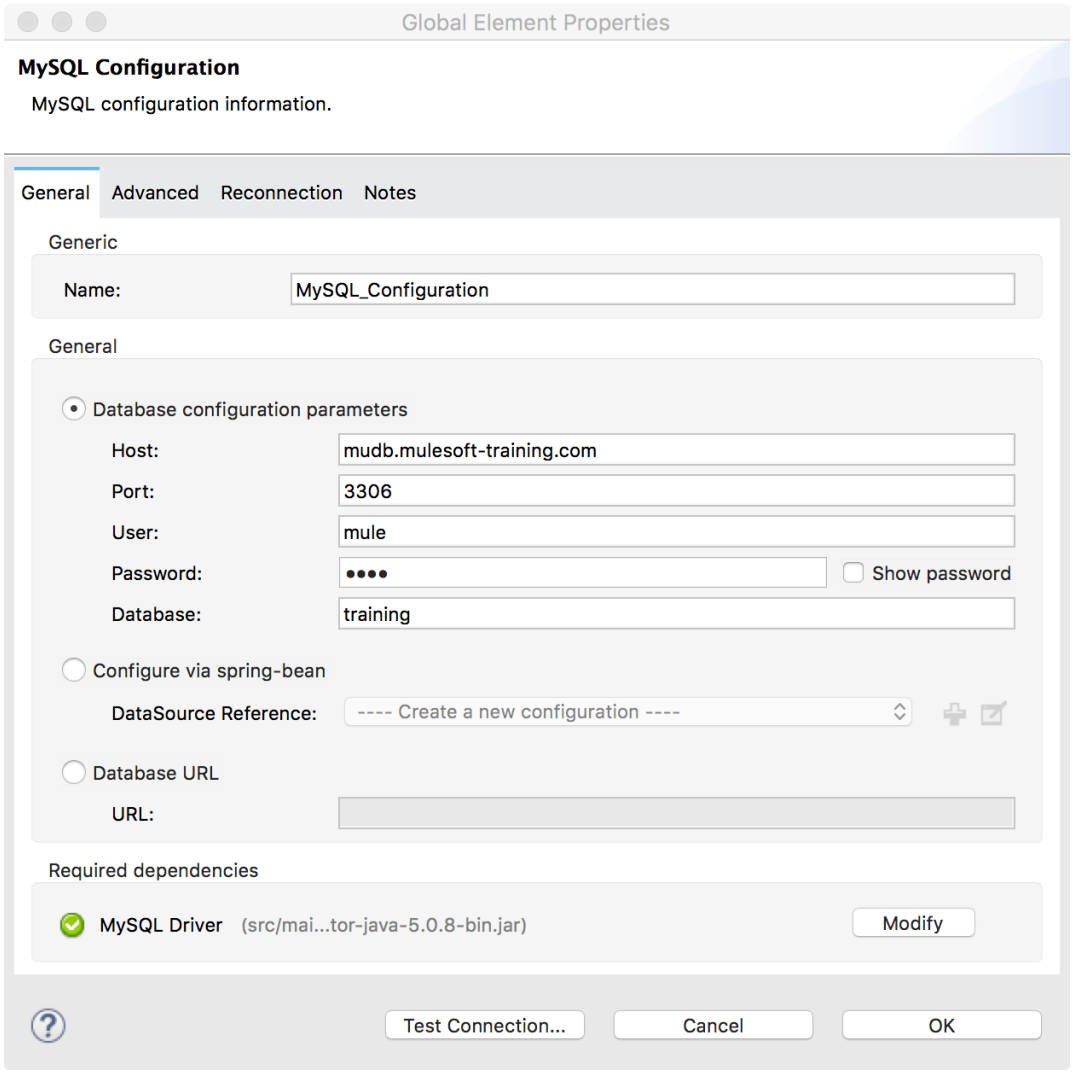
6. Double-click the Database endpoint.
7. In the Database properties view, click the Add button next to connector configuration.



8. In the Choose Global Type dialog box, select Connector Configuration > MySQL Configuration and click OK.

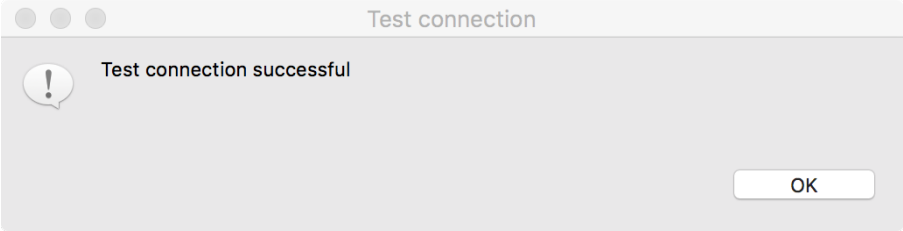


9. In the Global Element Properties dialog box, set the server, port, user, password, and database values to the values listed in the course snippets.txt file.
10. Under Required dependencies, click the Add File button next to MySQL Driver.
11. Navigate to the student files folder, select the MySQL JAR file located in the jars folder, and click Open.



12. Back in the Global Element Properties dialog box, click the Test Connection button; you should get a successful test dialog box.

    Note: Make sure the connection succeeds before proceeding.



    Note: If the connectivity test fails, make sure you are not behind a firewall restricting access to port 3306. If you cannot access port 3306, use the instructions in the next section for option 2.

13. Click OK to close the dialog box.
14. Click OK to close the Global Element Properties dialog box.

Option 2: Configure a Derby Database connector (if no access to port 3306)

15. In a command-line interface, use the cd command to navigate to the folder containing the jars folder of the student files.
16. Run the mulesoft-training-services.jar file.

```
java –jar mulesoft-training-services-X.X.X.jar
```

    Note: Replace X.X.X with the version of the JAR file, for example 1.5.0.

    Note: The application uses ports 1527, 9090, 9091, and 61616. If any of these ports are already in use, you can change them when you start the application as shown in the following code.

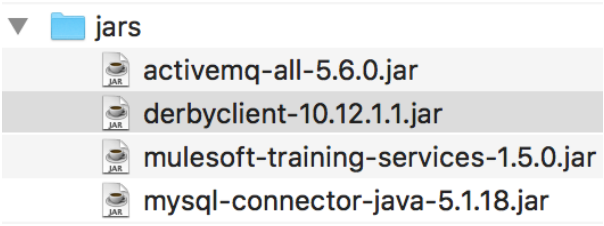
```
java -jar mulesoft-training-services-X.X.X.jar --database.port=1530 --
ws.port=9092 --spring.activemq.broker-url=tcp://localhost:61617 --
server.port=9193
```

17. Look at the output and make sure all the services started.

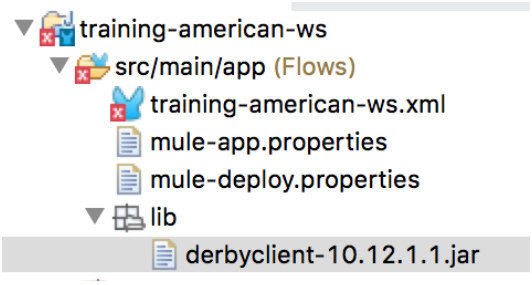
```
1. java
- Message Broker started
- American database started
- American flights database ready
- Delta flights web service started
- Order web service started
- Accounts REST API published
- American flights API published
- Banking REST API published
- JMS API published
- United flights web service started

┌──────────────────────────────────────────────────────────────────────────────┐
| MuleSoft Training Services - Press CTRL-C to terminate this application      |
├──────────────────────┬───────────────────────────────────────────────────────┤
| Welcome page         | http://localhost:9090                                 |
| American database URL | jdbc:derby://localhost:1527/memory:training          |
| American REST API    | http://localhost:9090/essentials/american/flights     |
| American REST API RAML | http://localhost:9090/essentials/american/flights-api.raml |
| United REST service  | http://localhost:9090/essentials/united/flights       |
| Delta SOAP service   | http://localhost:9191/essentials/delta                |
| Delta SOAP WSDL      | http://localhost:9191/essentials/delta?wsdl           |
| Order SOAP service   | http://localhost:9191/advanced/orders                 |
| Order SOAP WSDL      | http://localhost:9191/advanced/orders?wsdl            |
| Accounts API         | http://localhost:9090/essentials/accounts/api         |
| Accounts form        | http://localhost:9090/essentials/accounts/show.html   |
| JMS broker URL       | tcp://localhost:61616 (topic name: apessentials)      |
| JMS form             | http://localhost:9090/essentials/jmsform.html         |
| Banking API          | http://localhost:9090/api/...                         |
| Banking API RAML     | http://localhost:9090/api/banking-api.raml            |
└──────────────────────┴───────────────────────────────────────────────────────┘
```

Note: When you want to stop the application, return to this window and press Ctrl+C.

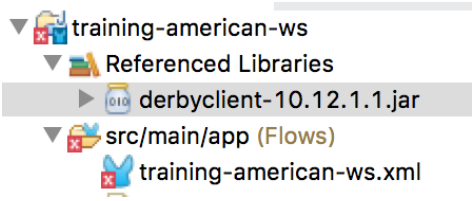
18. In the computer's file explorer, return to the student files folder and locate the derbyclient.jar file in the jars folder.



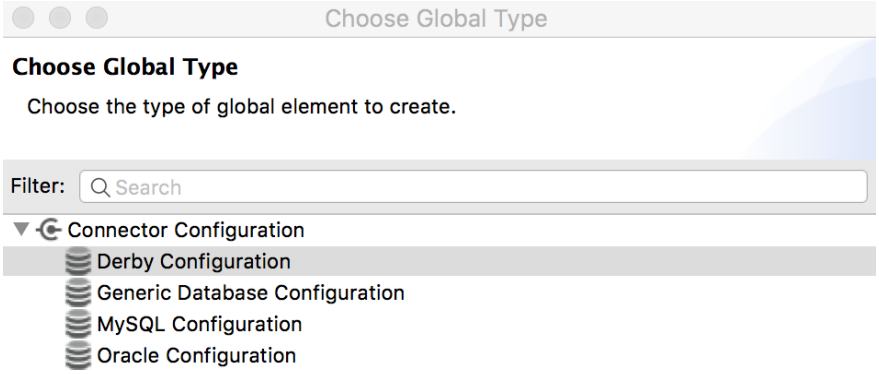
19. Copy and paste or drag this JAR file into the src/main/app/lib folder of the project in Anypoint Studio.



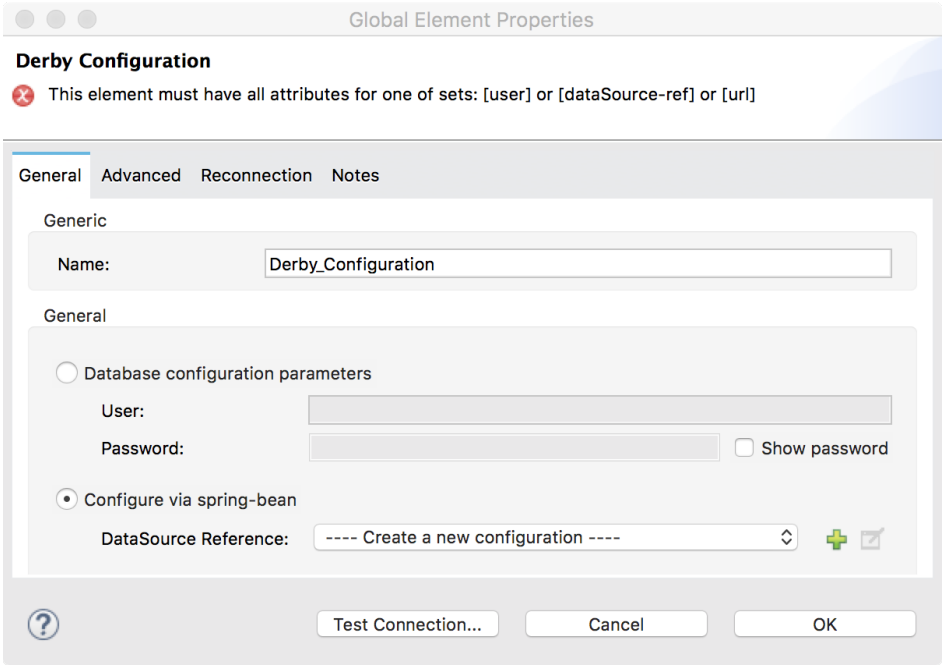
20. Right-click the JAR file and select Build Path > Add to Build Path; you should now see the JAR file in the project's Referenced Libraries.



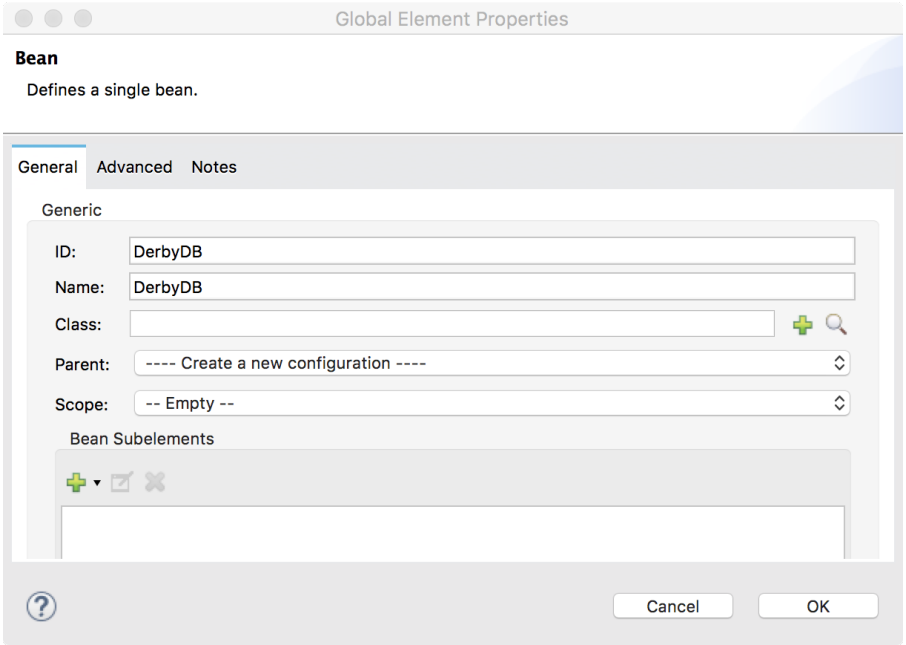
21. Double-click the Database endpoint in the canvas.
22. In the Database properties view, click the Add button next to connector configuration.
23. In the Choose Global Type dialog box, select Connector Configuration > Derby Configuration and click OK.



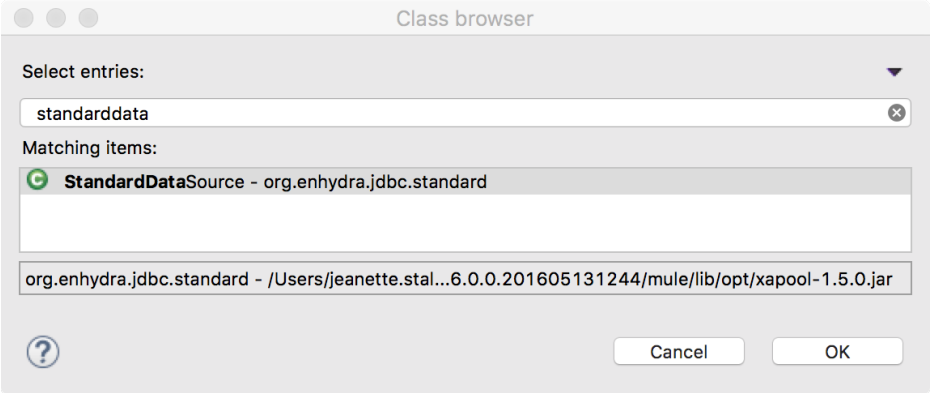
24. In the Global Element Properties dialog box, select Configure via spring-bean.
25. Click the Add button next to DataSource Reference.



26. On the Bean page of the Global Element Properties dialog box, set the ID and name to DerbyDB.



27. Click the Browse for Java class button next to class.
28. In the Class browser dialog box, start typing StandardDataSource and select the matching item for StandardDataSource – org.enhydra.jdbc.standard.



29. Click OK.
30. On the Bean page of the Global Element Properties dialog box, click the Add button under Bean Subelements and select Add Property.
31. In the Property dialog box, set the following values:

    - Name: driverName
    - Value: org.apache.derby.jdbc.ClientDriver

    Note: You can copy this value from the course snippets.txt file.

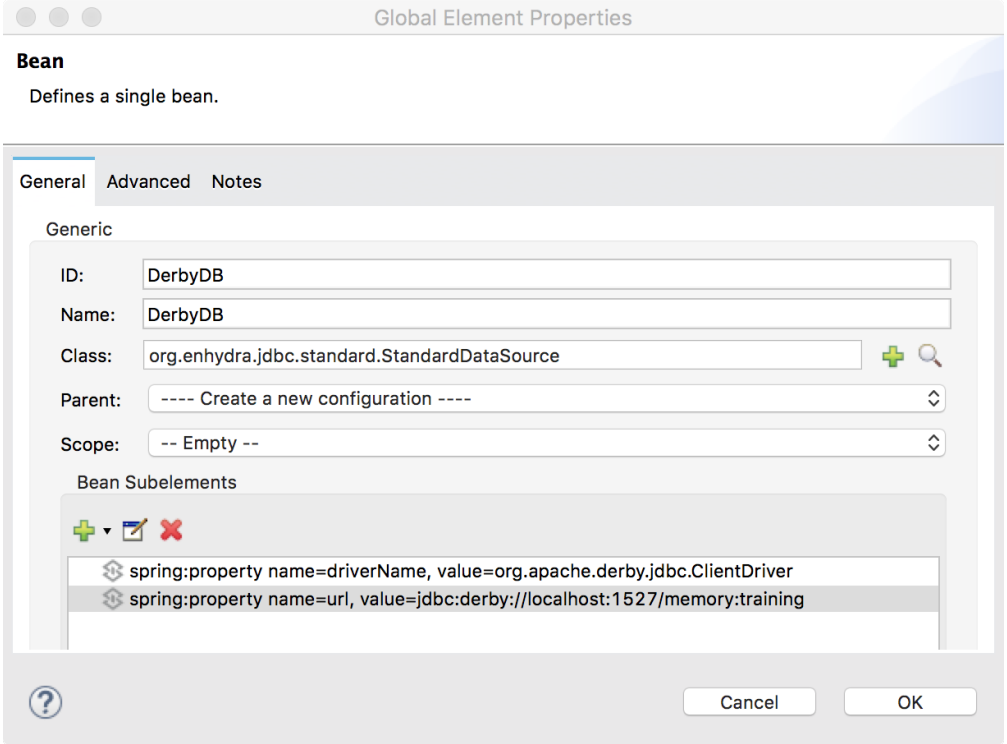
32. Click Finish.
33. Click the Add button under Bean Subelements again and select Add Property.
34. In the Property dialog box, set the following values:

    - Name: url
    - Value: jdbc:derby://localhost:1527/memory:training

    Note: You can copy this value from the course snippets.txt file.

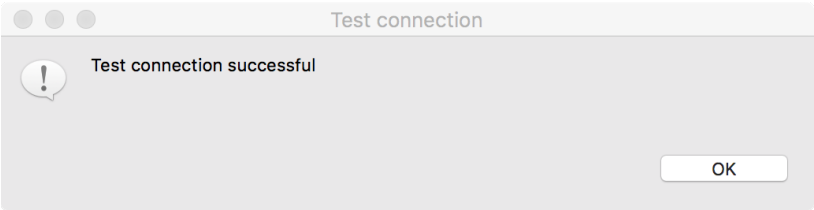
    Note: If you changed the database port when you started the mulesoft-training-services application, be sure to use the new value here.

35. Click Finish.
36. On the Bean page of the Global Element Properties dialog box, click OK.



37. On the Derby Configuration page of the Global Element Properties dialog box, click Test Connection; you should get a successful test dialog box.

    Note: Make sure the connection succeeds before proceeding.

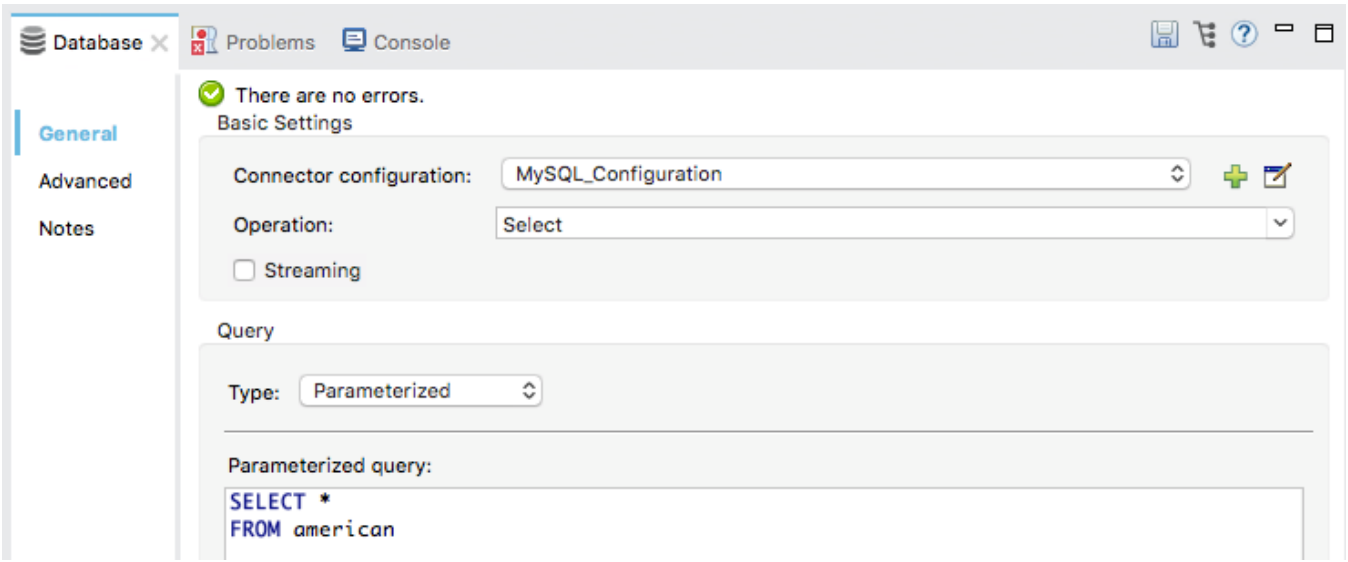


38. Click OK to close the dialog box.
39. Click OK to close the Global Element Properties dialog box.

Write a query to return all flights

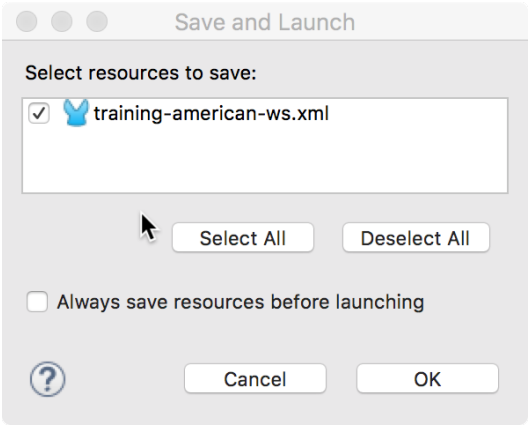
40. In the Database properties view, set the operation to Select.
41. Add a query to select all records from the american table.

```
SELECT *
FROM american
```

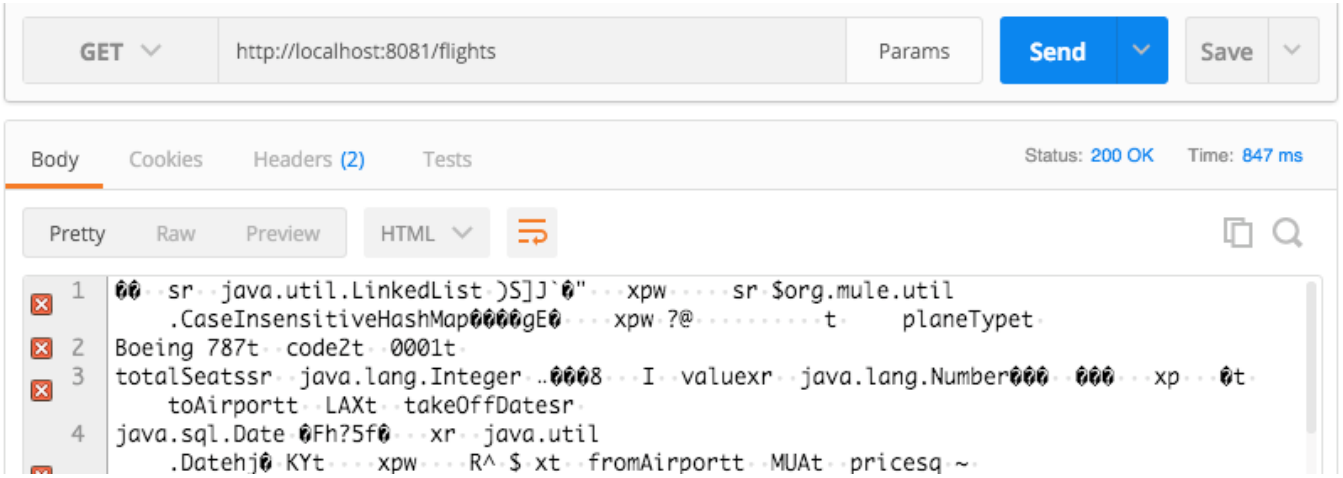


Test the application

42. Run the project.
43. In the Save and launch dialog box, select Always save resources before launching and click OK.



44. Watch the console, and wait for the application to start.
45. Once it has started, return to Postman.
46. In Postman, make another request to http://localhost:8081/flights; you should get some garbled plain text displayed – the tool's best representation of Java objects.

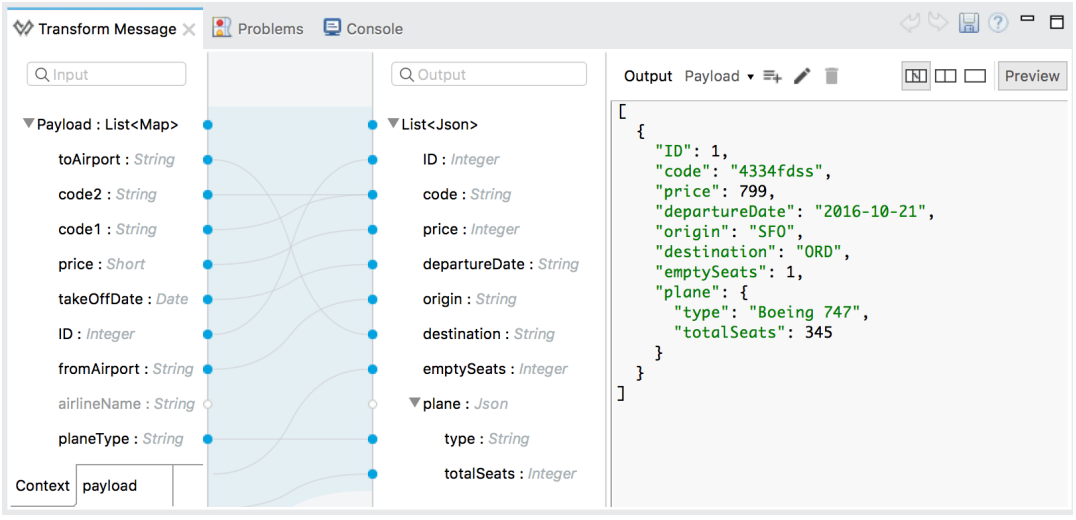


47. Return to Anypoint Studio.
48. Stop the project.

Walkthrough 4-3: Transform data

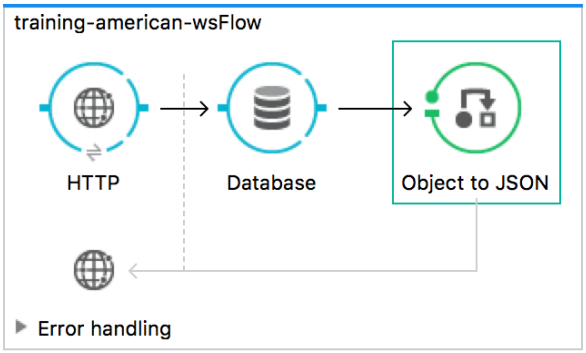
In this walkthrough, you transform and display the account data into JSON. You will:

- Use the Object to JSON transformer.
- Replace it with a Transform Message component.
- Use the DataWeave visual mapper to change the response to a different JSON structure.



Add an Object to JSON transformer

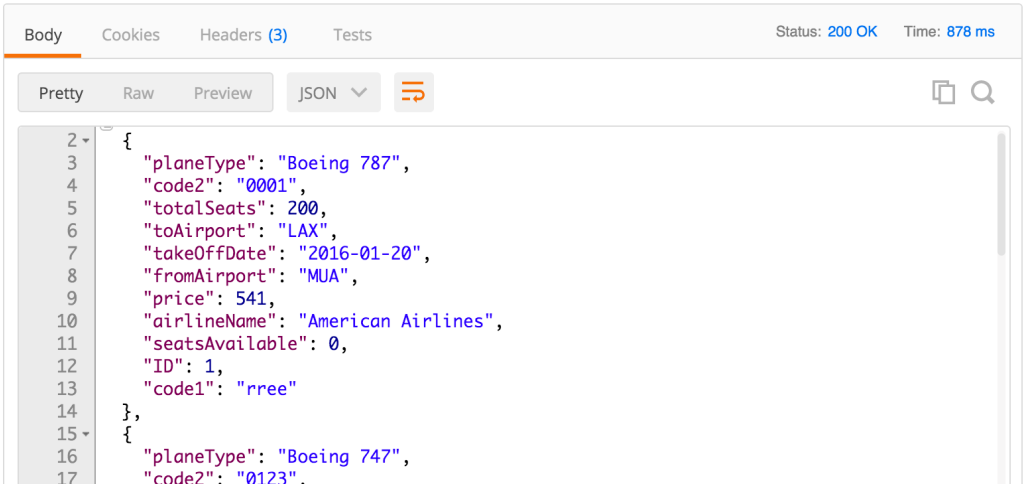
1. In the Mule Palette, select the Transformers tab.
2. Drag an Object to JSON transformer from the Mule Palette and drop it after the Database endpoint.



Test the application

3. Run the project.

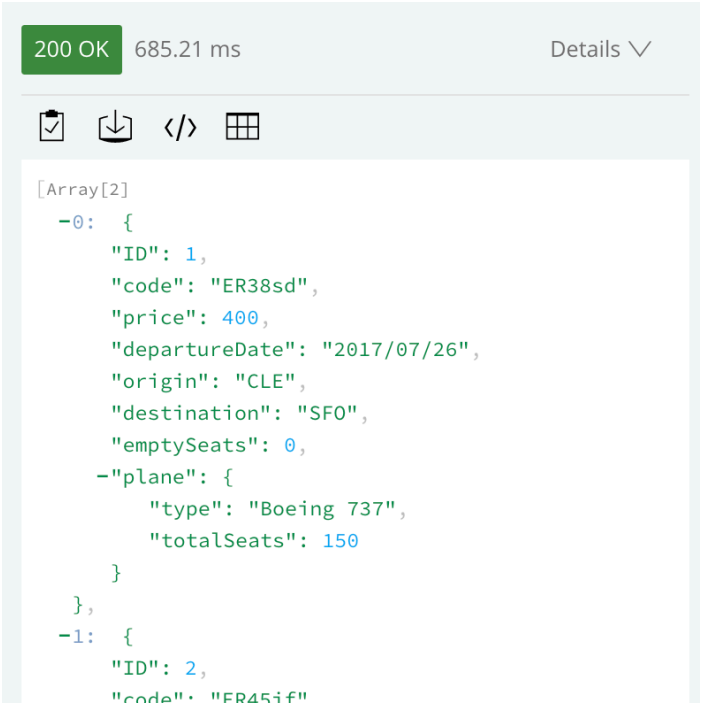
4. In Postman, send the same request; you should see the American flight data represented as JSON.



Note: If you are using the local Derby database, the properties will be uppercase instead.

Review the data structure to be returned by the American flights API

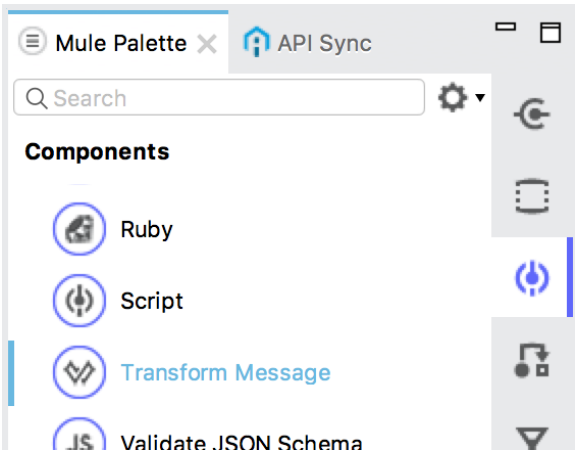
5. Return to your American Flights API in Exchange.
6. Look at the example data returned for the /flights GET method.



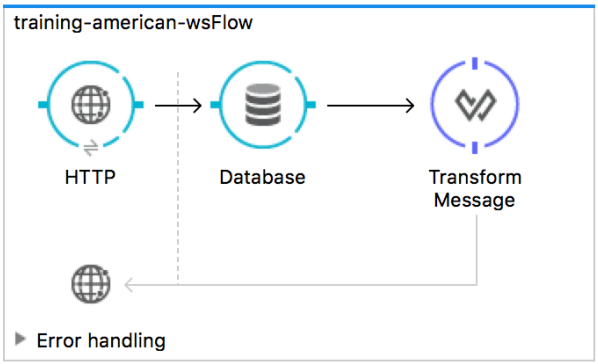
7. Notice that the structure of the JSON being returned by the Mule application does not match this JSON.

Add a Transform Message component

8. Return to Anypoint Studio and stop the project.
9. Right-click the Object to JSON transformer and select Delete.
10. In the Mule Palette, select the Components tab.



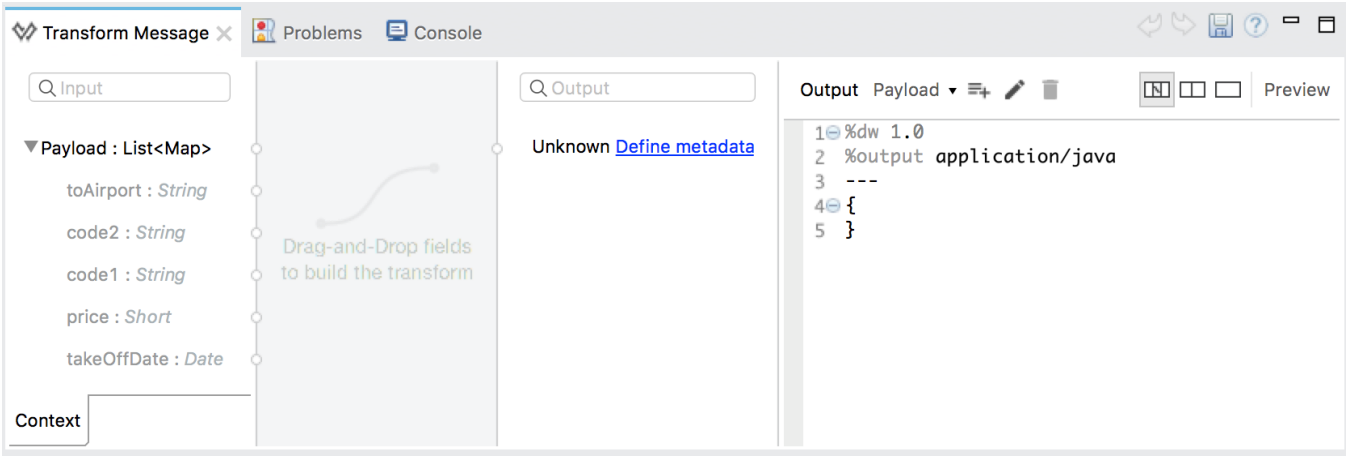
11. Drag a Transform Message component from the Mule Palette and drop it after the Database endpoint.



Review metadata for the transformation input

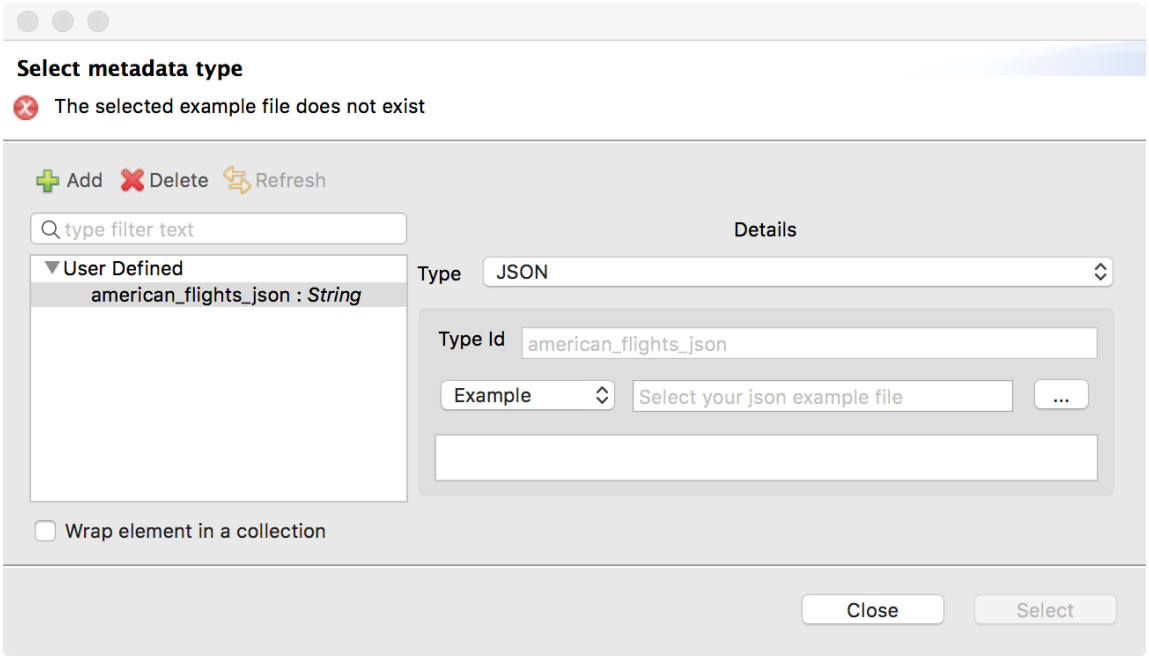
12. Double-click the Transform Message component in the canvas.

13. In the Transform Message properties view, look at the input section and review the payload metadata; it should match the data returned by the Database endpoint.



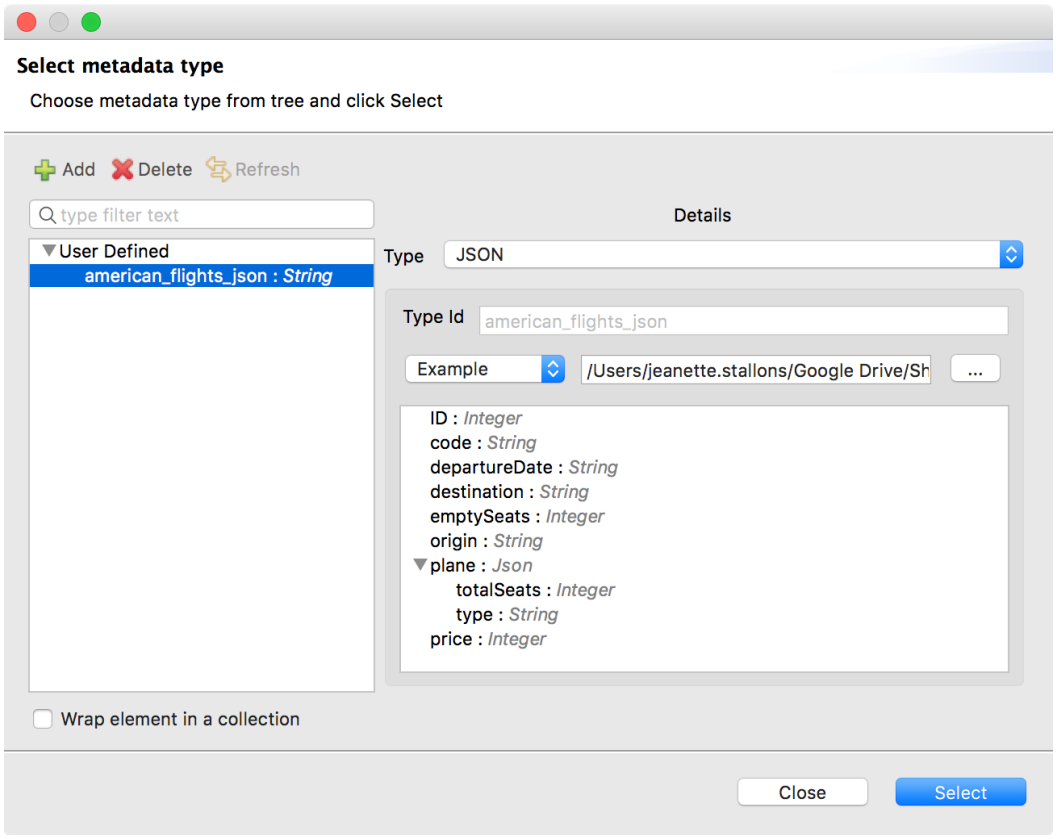
Add metadata for the transformation output

14. Click the Define metadata link in the output section.
15. In the Select metadata type dialog box, click the Add button.
16. In the Create new type dialog box, set the type id to american_flights_json.
17. Click Create type.
18. Back in the Set metadata type dialog box, set the type to JSON.
19. Change the Schema selection to Example.

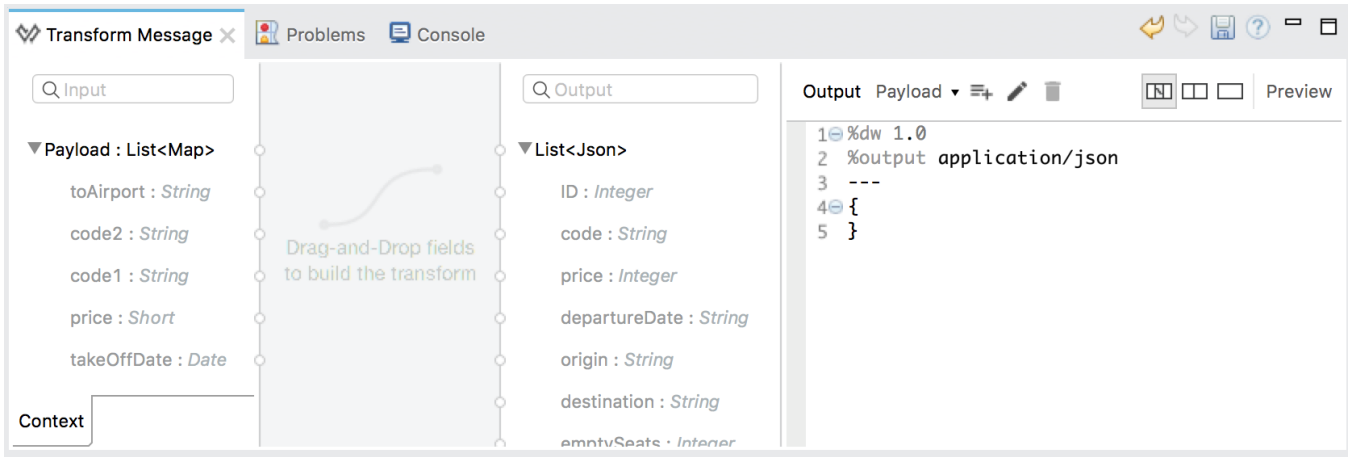


20. Click the browse button and navigate to the course student files.

21. Select american-flights-example.json in the examples folder and click Open; you should see the example data for the metadata type.



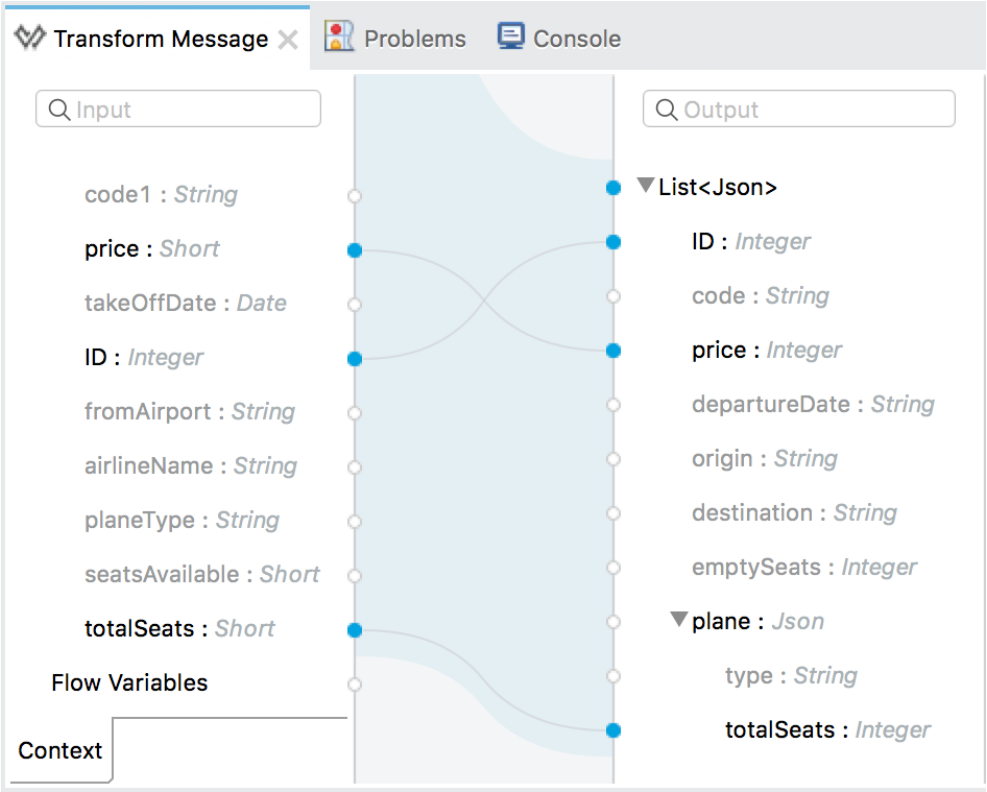
22. Click Select; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

23. Map fields with the same names by dragging them from the input section and dropping them on the corresponding field in the output section.

    - ID to ID
    - price to price
    - totalSeats to plane > totalSeats

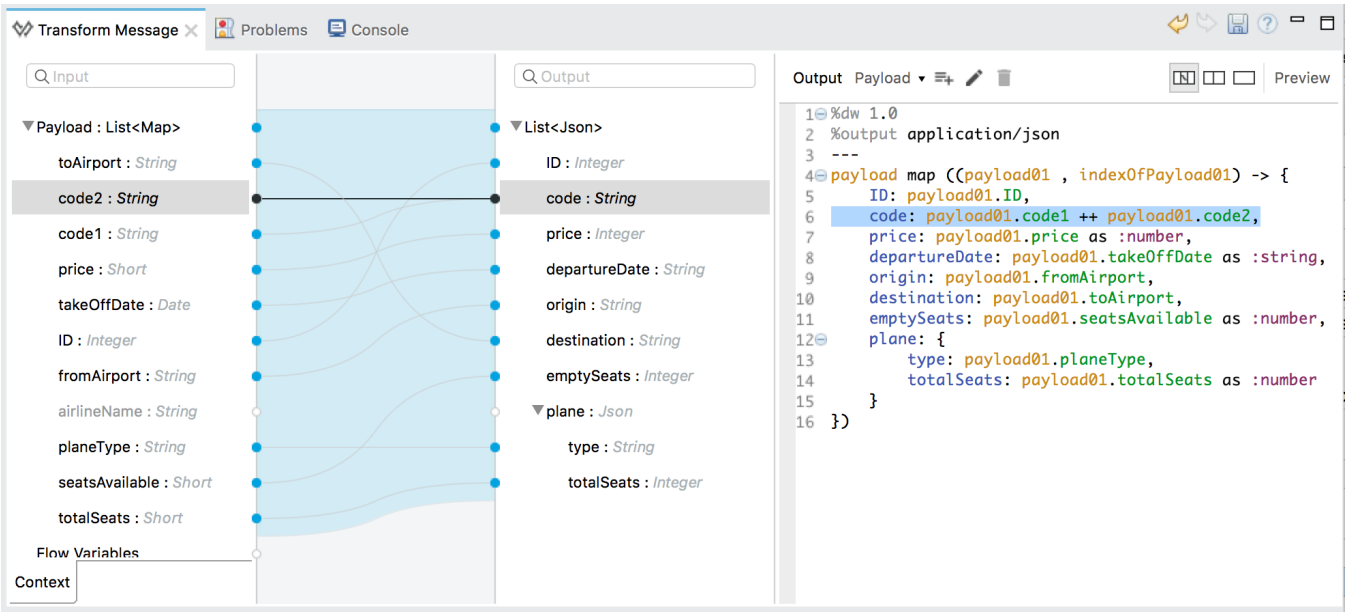


24. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

    - toAirport to destination
    - takeOffDate to departureDate
    - fromAirport to origin
    - seatsAvailable to emptySeats
    - planeType to plane > type

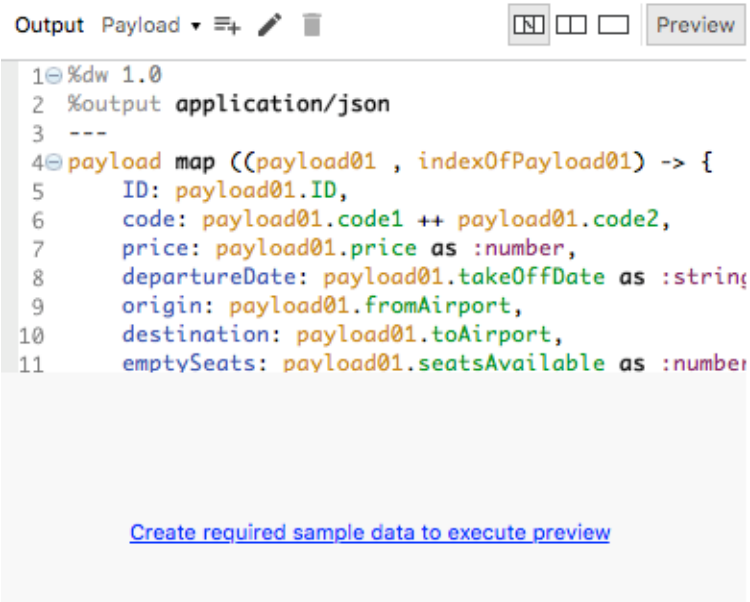
25. Concatenate two fields by dragging them from the input section and dropping them on the same field in the output section.

    - code1 to code
    - code2 to code

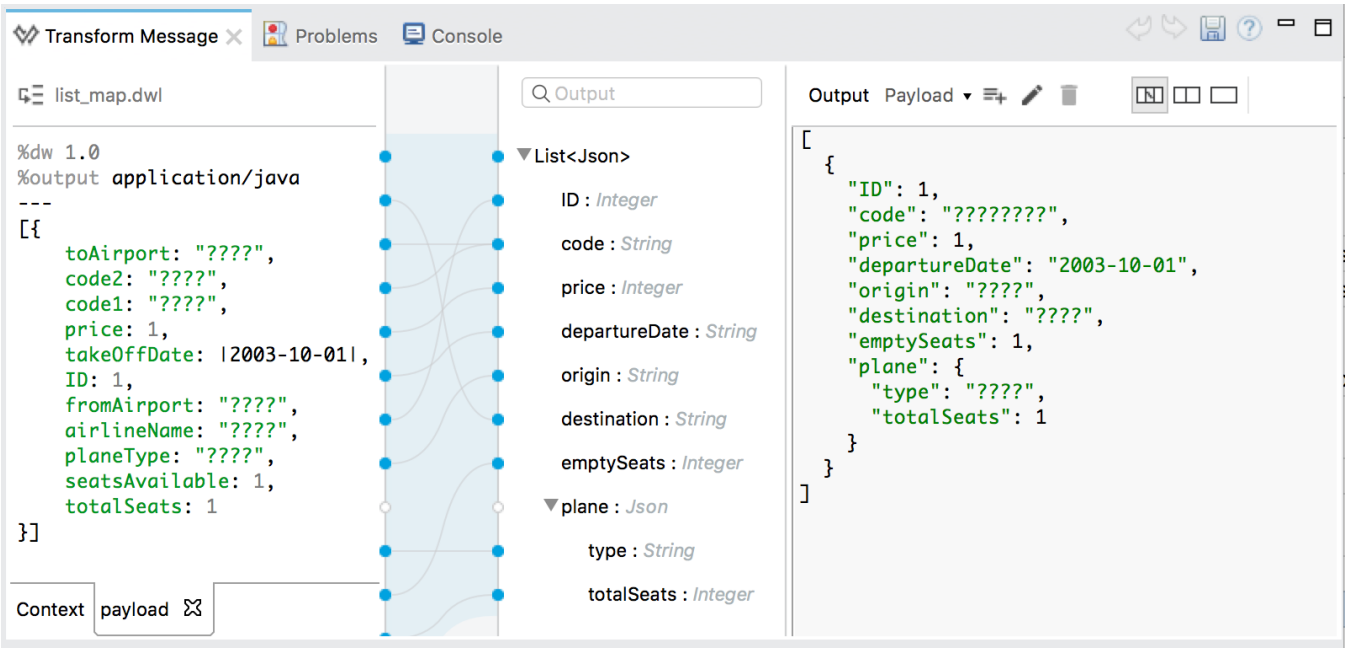


Add sample data

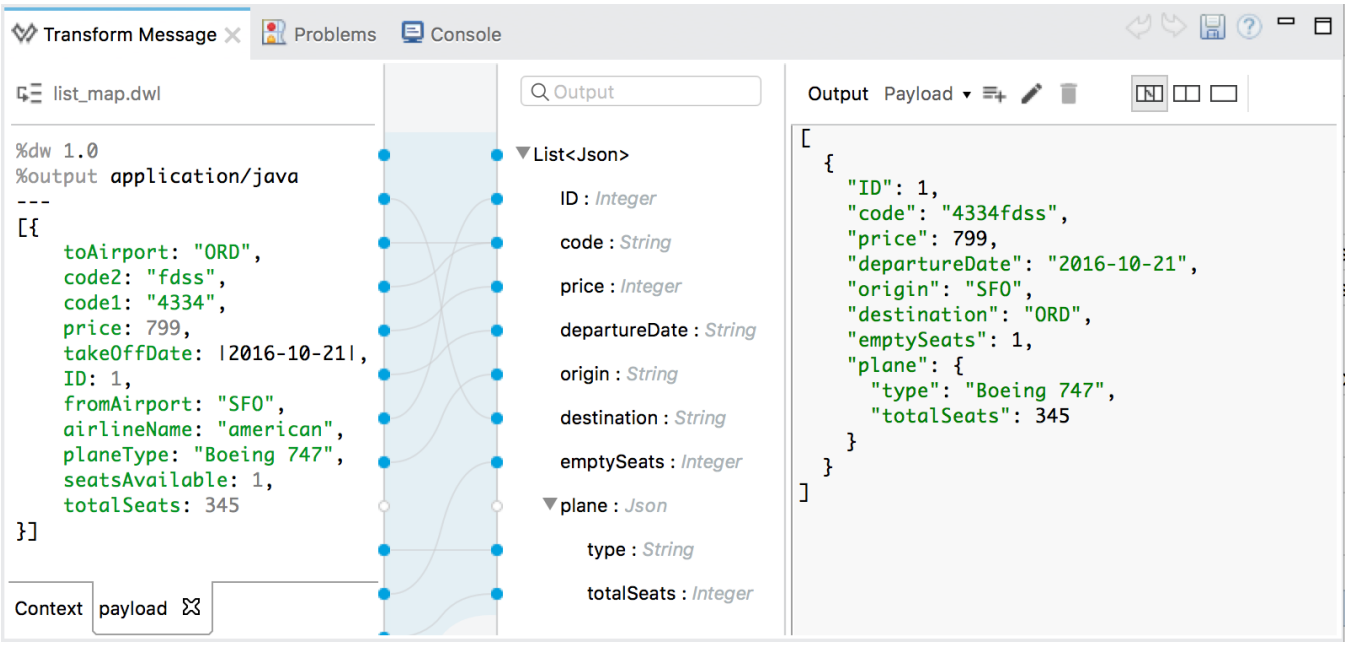
26. Click the Preview button in the output section.
27. In the preview section, click the Create required sample data to execute preview link.



28. Look at the input section, you should see a new tab called payload with sample data generated from the input metadata.
29. Look at the output section, you should see a sample response for the transformation.

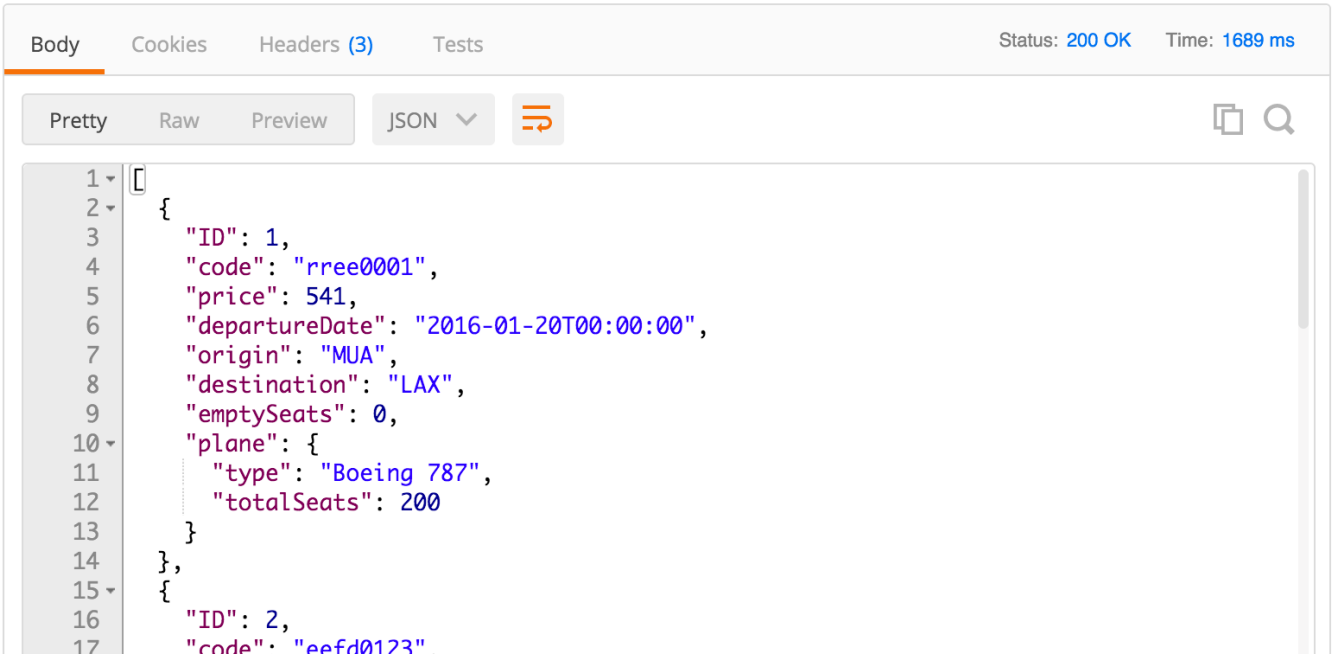


30. In the input section, replace all the ???? with sample values.
31. Look at the output section, you should see the sample values in the transformed data.



Test the application

32. Run the project.
33. In Postman, make another request to http://localhost:8081/flights; you should see all the flight data as JSON again but now with a different structure.



Try to retrieve information about a specific flight

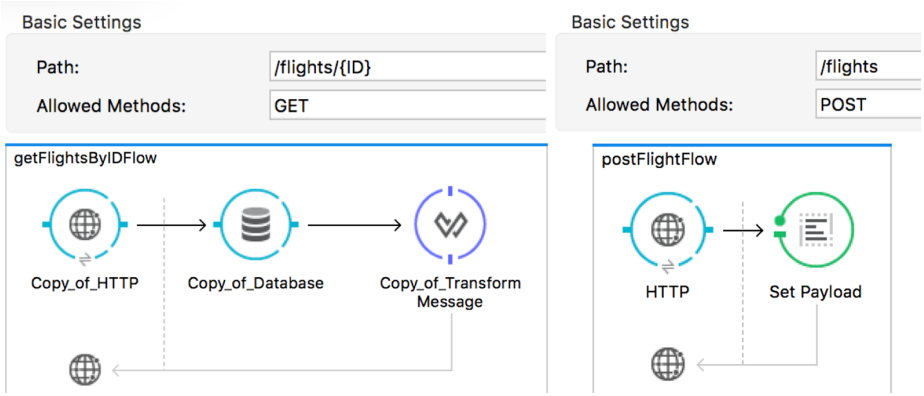
34. Add a URI parameter to the URL to make a request to http://localhost:8081/flights/3; you should get a 404 response with a no listener or resource not found message.

35. Return to Anypoint Studio.
36. Look at the console; you should get a no listener found for request (GET)/flights/3.
37. Stop the project.

Walkthrough 4-4: Create a RESTful interface for a Mule application

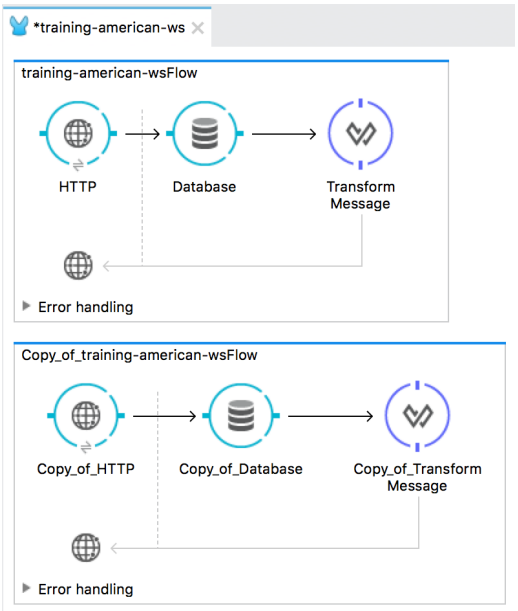
In this walkthrough, you continue to create a RESTful interface for the application. You will:

- Route based on path.
- Add a URI parameter to a new HTTP Listener endpoint path.
- Route based on HTTP method.



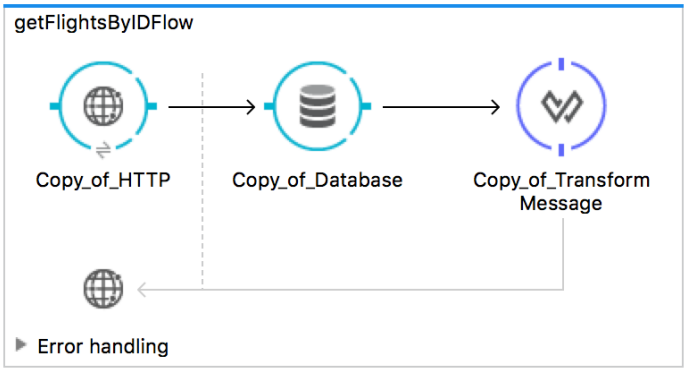
Make a copy of the existing flow

1. Return to training-american-ws.xml.
2. Click the flow in the canvas to select it.
3. From the main menu bar, select Edit > Copy.
4. Click in the canvas beneath the flow and select Edit > Paste.



Rename the flows

5. Double-click the name of the first flow.
6. In the Properties view, change its name to getFlightsFlow.
7. Change the name of the second flow to getFlightsByIDFlow.



Note: If you want, change the name of the message source and message processors.

Specify a URI parameter for the new HTTP Listener endpoint

8. Double-click the HTTP Listener endpoint in getFlightsByIDFlow.
9. Change the path to have a URI parameter called ID.

Modify the Database endpoint

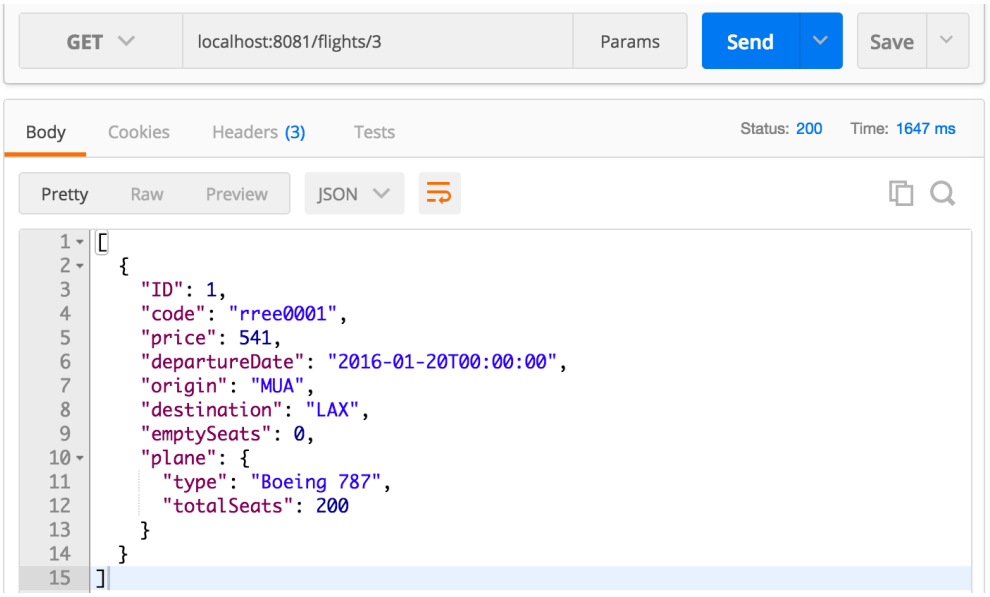
10. Double-click the Database endpoint in getFlightsByIDFlow.
11. Modify the query WHERE clause, to select flights with the ID equal to 1.

```
SELECT *
FROM american
WHERE ID = 1
```

Test the application

12. Run the project.

13. In Postman, make another request to http://localhost:8081/flights/3; you should see details for the flight with an ID of 1.



Modify the database query to use the URI parameter

14. Return to the course snippets.txt file and copy the SQL expression for American Flights API.
15. Return to Anypoint Studio and stop the project.
16. In the Database properties view, replace the existing WHERE clause with the value you copied.

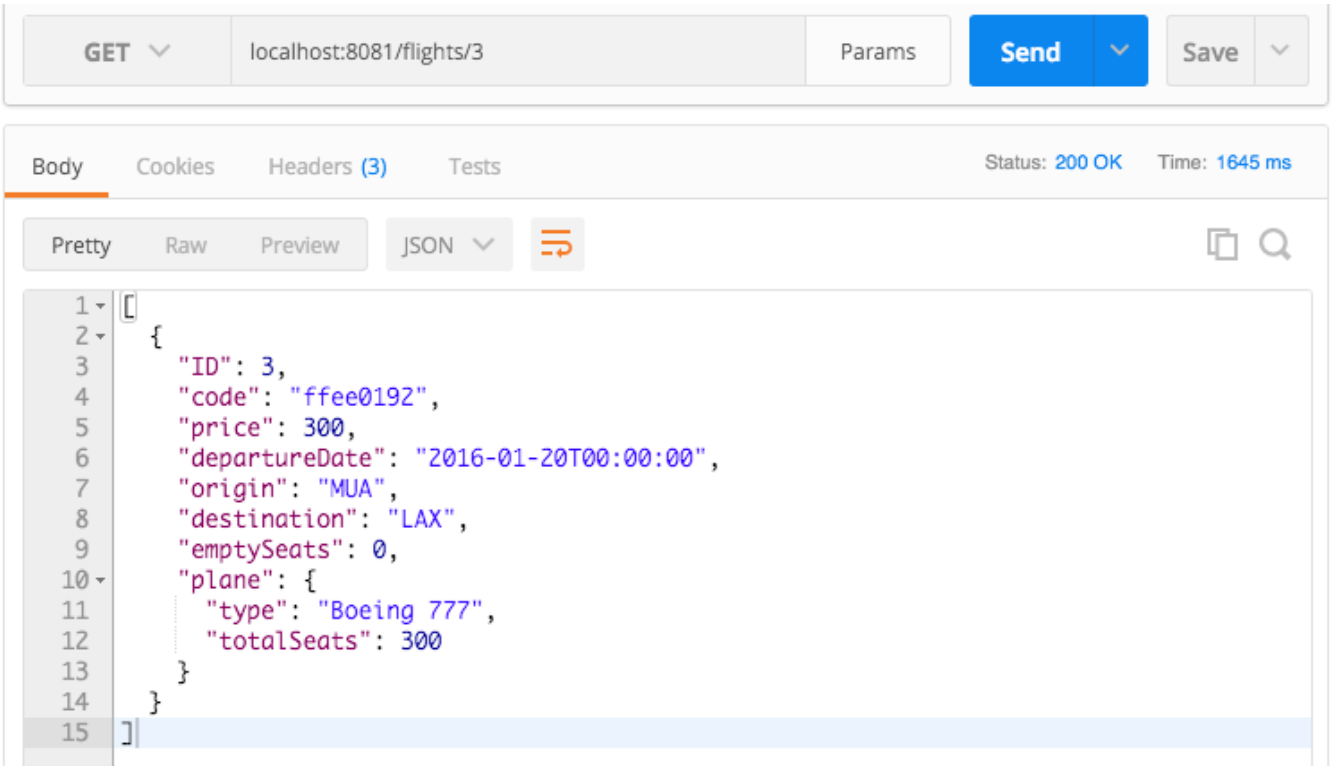
```
SELECT *
FROM american
WHERE ID = #[message.inboundProperties.'http.uri.params'.ID]
```

Note: You learn about reading and writing properties and variables in a later module in the Development Fundamentals course.

Test the application

17. Run the project.

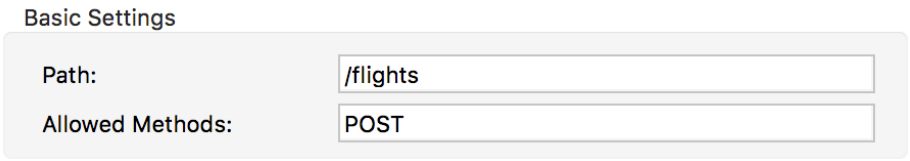
18. In Postman, make another request to http://localhost:8081/flights/3; you should now see the info for the flight with an ID of 3.



19. Return to Anypoint Studio and stop the project.

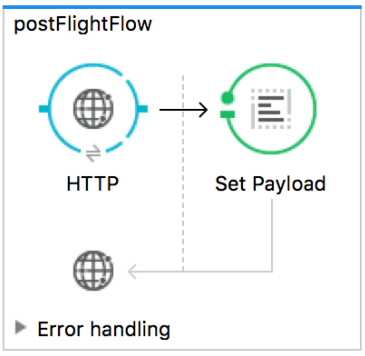
Make a new flow to handle post requests

20. In the Mule Palette, select the Connectors tab.
21. Drag out an HTTP connector from the Mule Palette and drop it in the canvas below the two existing flows.
22. Change the name of the flow to postFlightFlow.
23. Double-click the HTTP Listener endpoint.
24. In the HTTP properties view, set the path to /flights and the allowed methods to POST.



25. In the Mule Palette, select the Transformers tab.

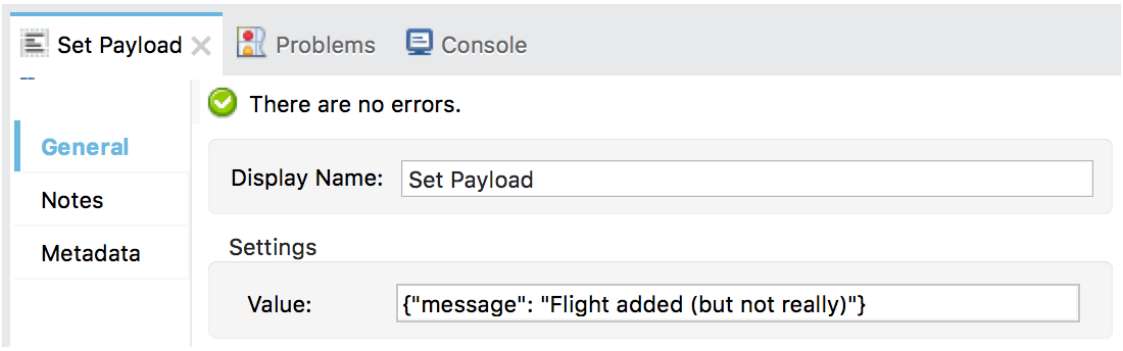
26. Drag out a Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



27. Double-click the Set Payload processor.
28. Return to the course snippets.txt file and copy the American Flights API - /flights POST response example.

```
{"message": "Flight added (but not really)"}
```

29. Return to Anypoint Studio and in the Set Payload properties view, set value to the value you copied.

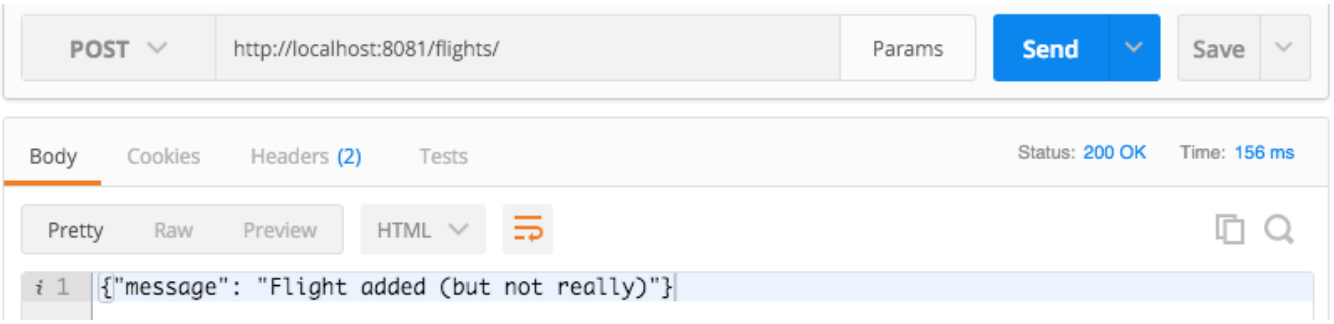


Note: This flow is just a stub. For it to really work and add data to the database, you would need to add logic to insert the request data to the database.

Test the application

30. Run the project.
31. In Postman, change the request type from GET to POST.
32. Remove the URI parameter from the request URL: http://localhost:8081/flights.

33. Send the request; you should now see the message the flight was added – even though you did not send any flight data to add.

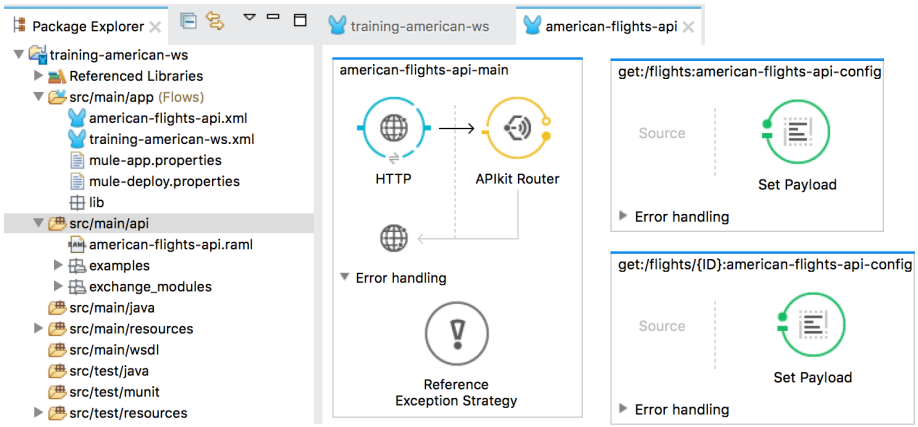


34. Return to Anypoint Studio and stop the project.

Walkthrough 4-5: Use Anypoint Studio to create a RESTful API interface from a RAML file

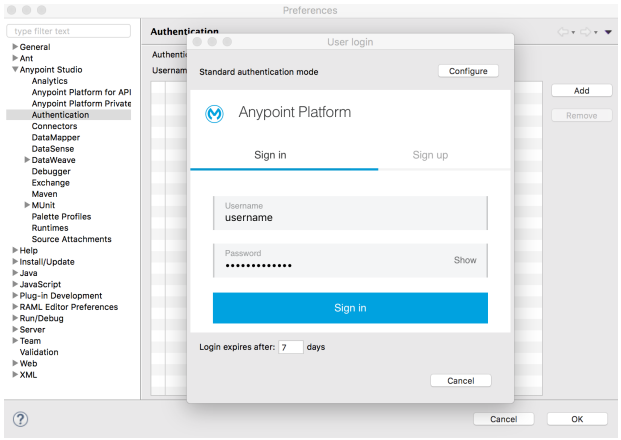
In this walkthrough, you generate a RESTful interface from the RAML file. You will:

- Add Anypoint Platform credentials to Anypoint Studio.
- Import an API from Design Center into an Anypoint Studio project.
- Use APIkit to generate a RESTful web service interface from an API.
- Test a web service using Postman.



Add Anypoint Platform credentials to Anypoint Studio

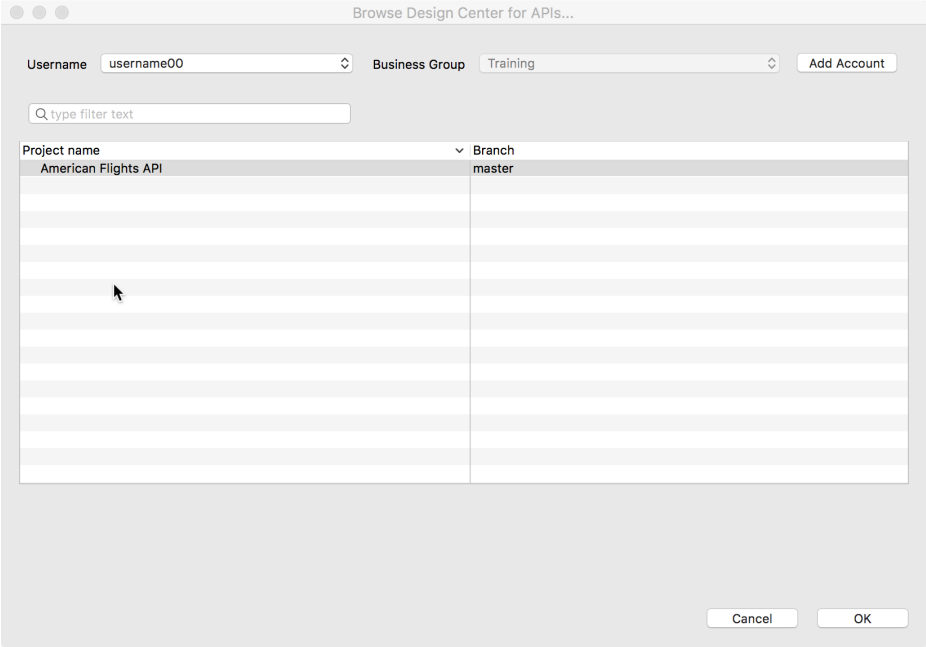
1. In Anypoint Studio, right-click training-american-ws and select Anypoint Platform > Configure Credentials.
2. In the Authentication page of the Preferences dialog box, click the Add button.
3. In the Anypoint Platform Sign In dialog box, enter your username & password and click Sign In.



4. On the Authentication page, make sure your username is listed and selected.
5. Click OK.

Add an API from Design Center to the Anypoint Studio project

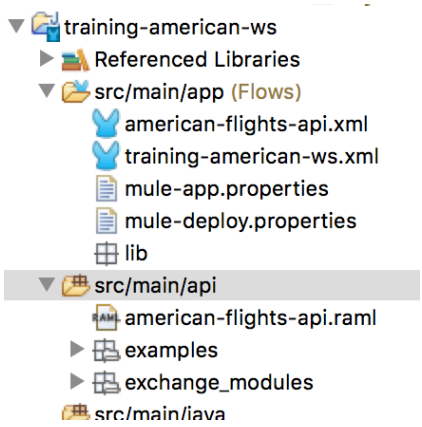
6. In the Package Explorer, locate the src/main/api folder; it should not contain any files.
7. Right-click the folder (or anywhere in the project in the Package Explorer) and select Anypoint Platform > Import from Design Center.
8. In the Browse Design Center for APIs dialog box, select the American Flights API and click OK.



9. In the Override files dialog box, click Yes.

Locate the API files added to the project

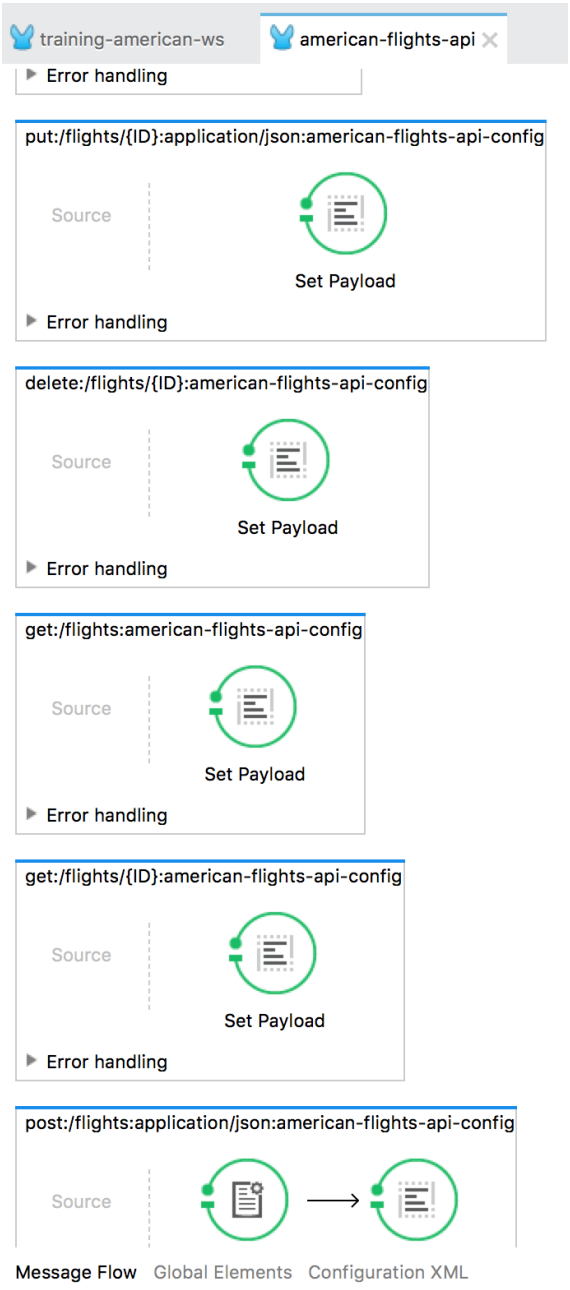
10. In the Package Explorer, locate and expand the src/main/api folder; it should now contain files.



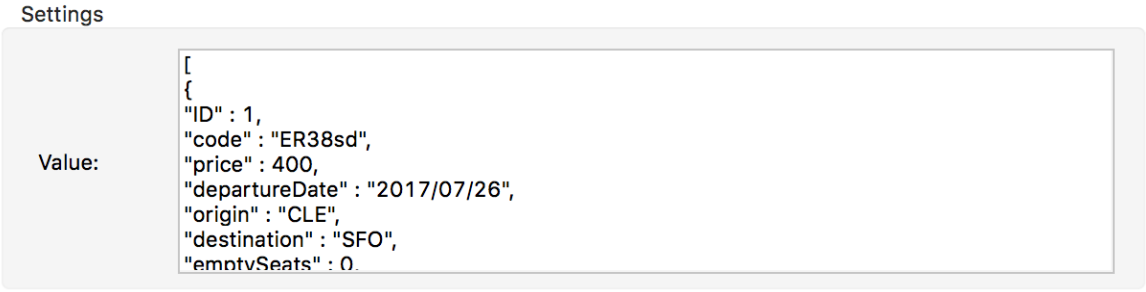
Examine the XML file created

11. Examine the generated american-flghts-api.xml file and locate the following five flows:

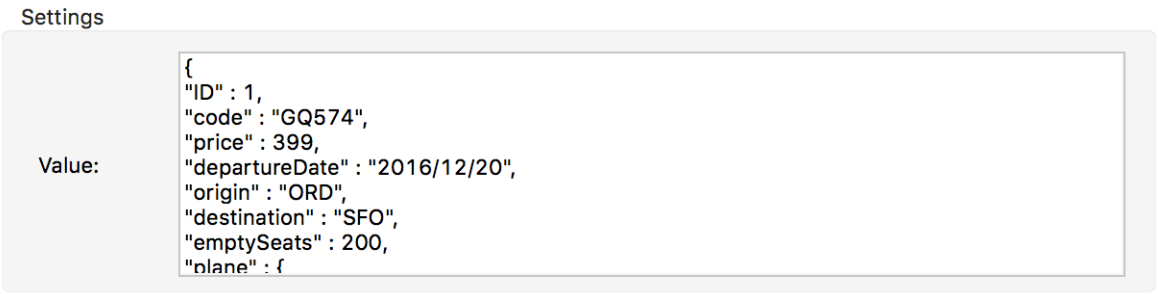
- get:/flights
- get:/flights/{ID}
- post:/flights
- delete:/flights/{ID}
- put:/flights/{ID}



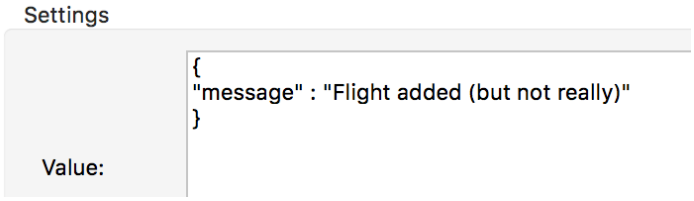
12. In the get:/flights flow, double-click the Set Payload transformer and look at the value in the Set Payload properties view.



13. In the get:/flights/{ID} flow, double-click the Set Payload transformer and look at the value in the Set Payload properties view.



14. In the post:/flights flow, double-click the Set Payload transformer and look at the value in the Set Payload properties view.

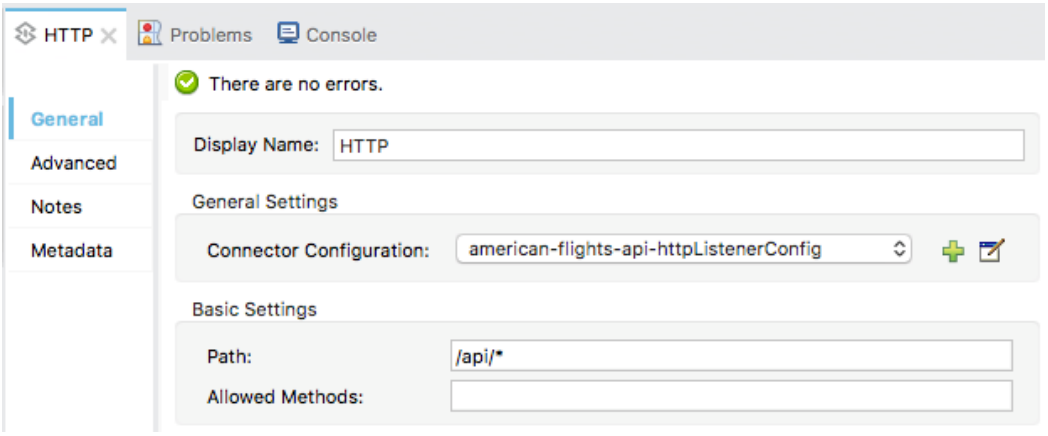


Examine the main flow and the new HTTP Listener endpoint

15. Locate the american-flights-api-main flow.
16. Double-click its HTTP Listener endpoint.

17. In the HTTP properties view, notice that the path is set to /api/*.

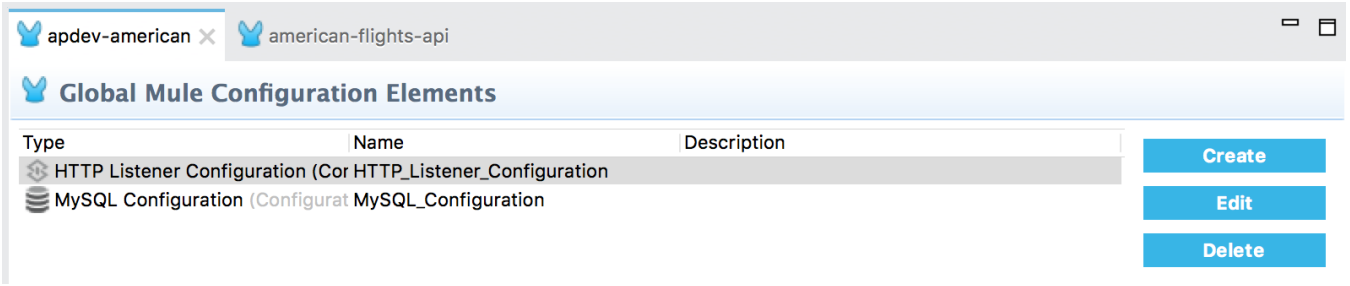
    *Note: The * is a wildcard allowing any characters to be entered after /api/.*



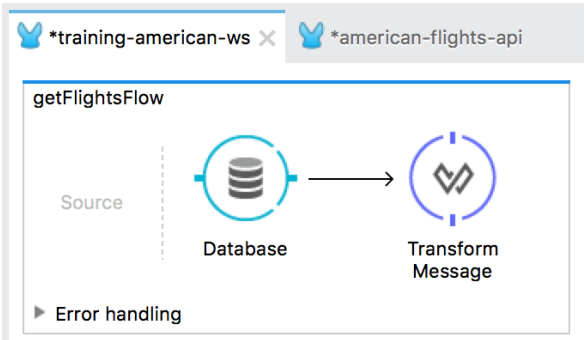
18. Click the Edit button for the connector configuration; you should see that the same port 8081 is used as the HTTP listener you created previously.
19. Click OK.

Remove the other HTTP configuration and listeners

20. Return to training-american-ws.xml.
21. In the Global Elements view, select the HTTP Listener and click Delete.



22. Return to the Message Flow view.
23. Right-click the HTTP Listener endpoint in getFlightsFlow and select Delete.

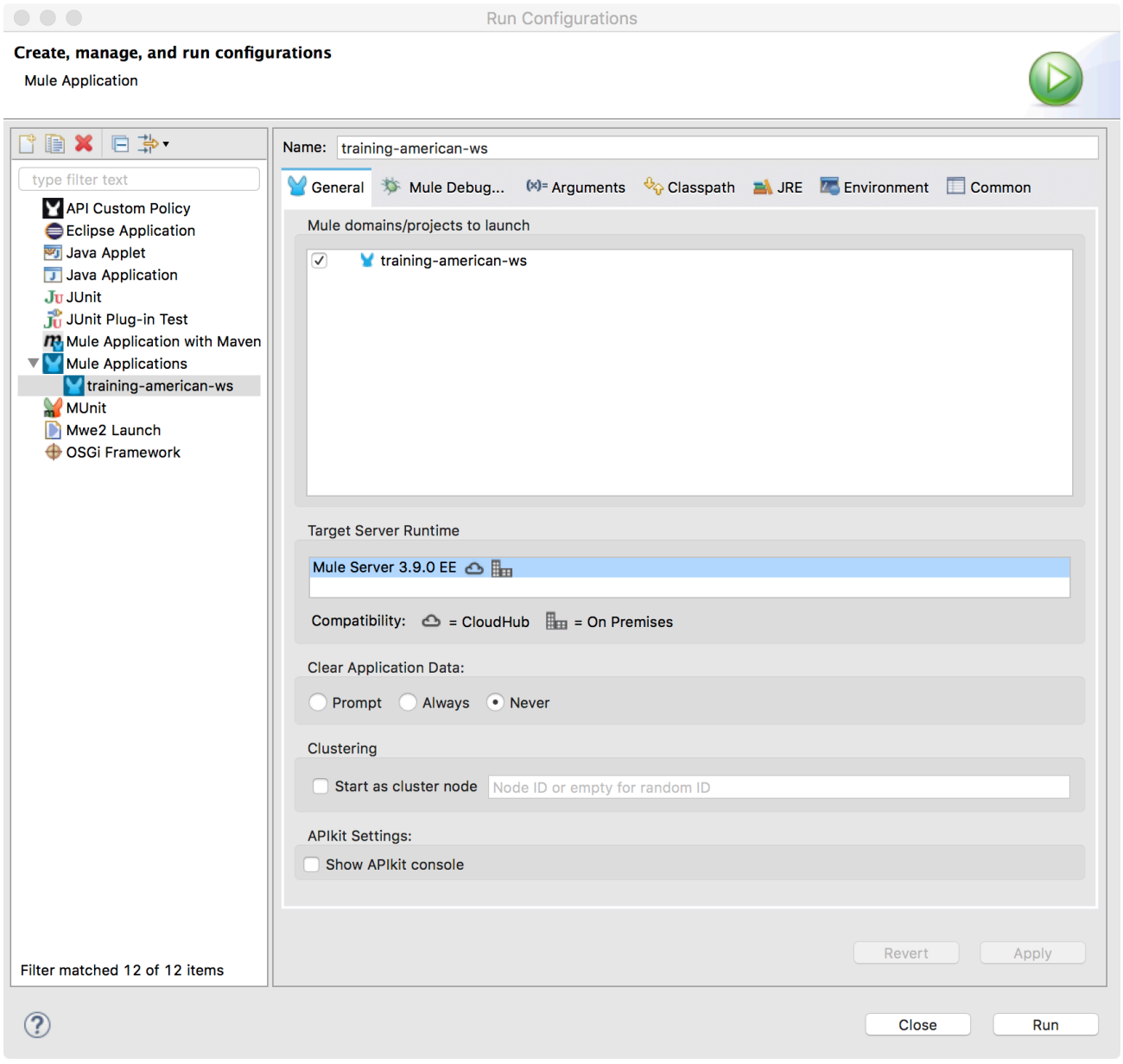


24. Delete the other two HTTP Listener endpoints.

Disable the APIkit Consoles view

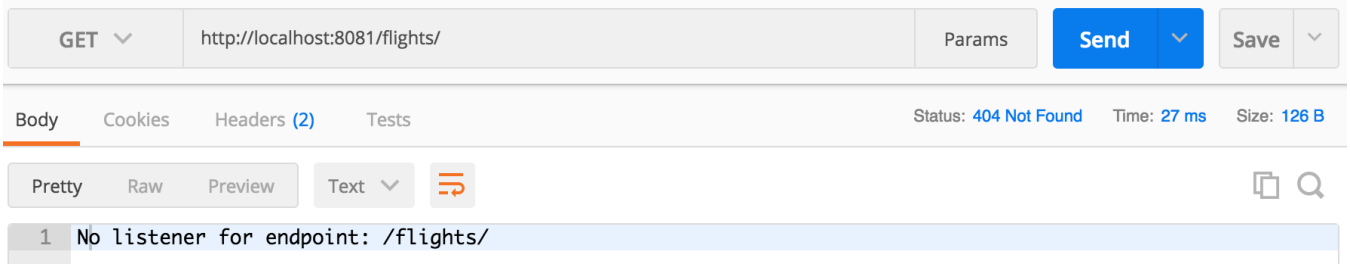
25. In the main menu, select Run > Run Configurations.
26. In the Run Configurations dialog box for training-american-ws, uncheck the Show APIkit console option located at the bottom of the General tab settings.
27. Click Run.

Note: If you want to use the APIkit Consoles view, you need to add a baseUri to the RAML file in order for the Try-it functionality of the API console to work.

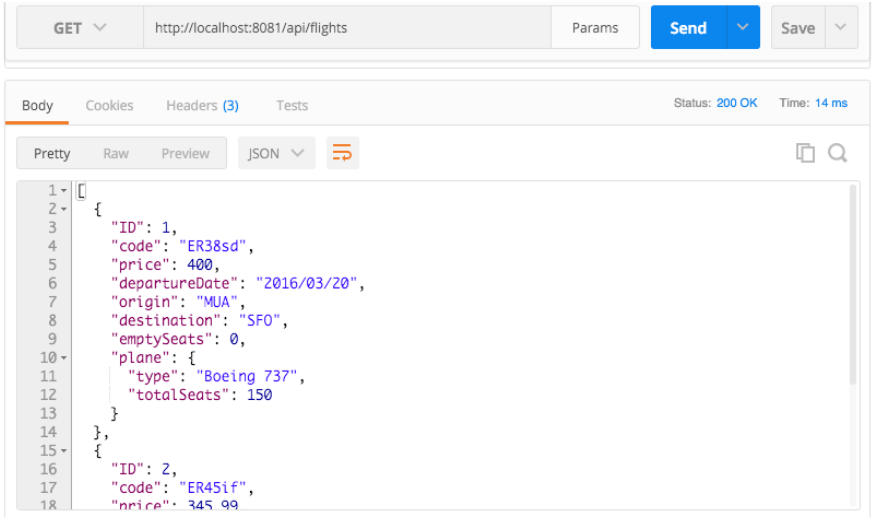


Test the web service using Postman

28. In Postman, make a GET request to http://localhost:8081/flights; you should get a 404 response with a message that the resource was not found because there is no longer a listener for that endpoint.



29. Change the URL to http://localhost:8081/api/flights and send the request; you should see the example data returned.

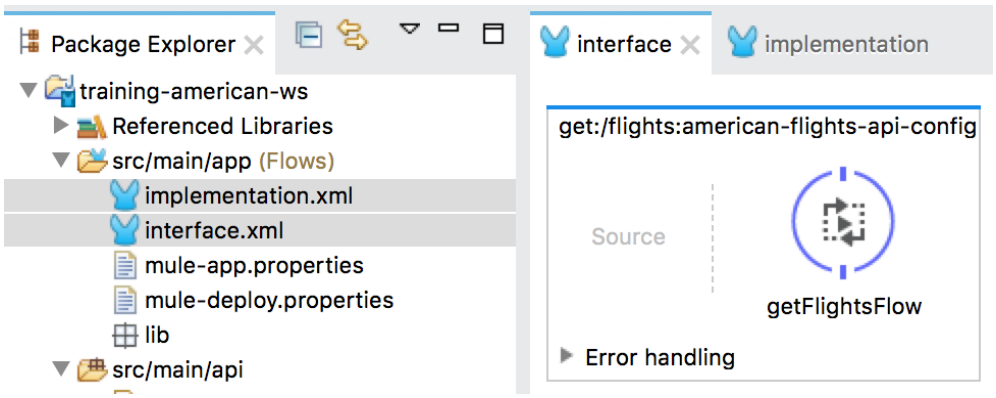


30. Make a request to http://localhost:8081/api/flights/3; you should see the example data returned.
31. Return to Anypoint Studio and stop the project.

Walkthrough 4-6: Implement a RESTful web service

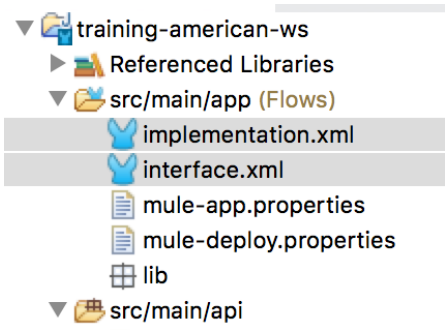
In this walkthrough, you wire the RESTful web service interface up to your back-end logic. You will:

- Pass a message from one flow to another.
- Call the backend flows.
- Create new logic for the nested resource call.
- Test the web service using Postman.



Rename the configuration files

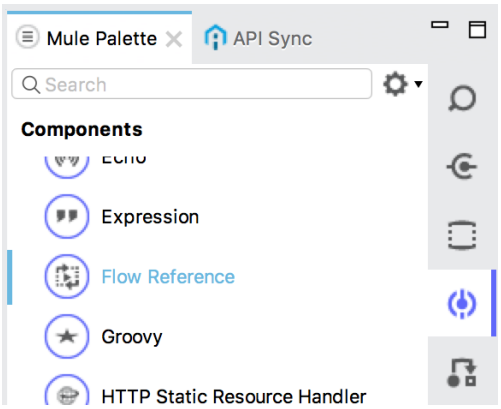
1. Right-click american-flights-api.xml in the Package Explorer and select Refactor > Rename.
2. In the Rename Resource dialog box, set the new name to interface.xml and click OK.
3. Right-click training-american-ws.xml and select Refactor > Rename.
4. In the Rename Resource dialog box, set the new name to implementation.xml and click OK.



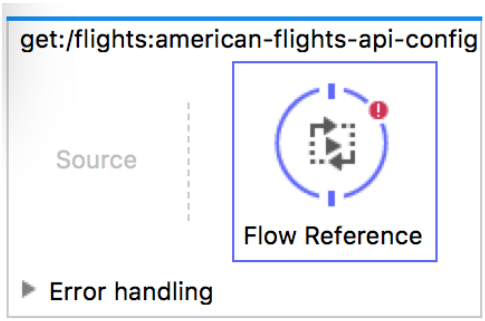
Set logic for the /flights resource

5. Return to interface.xml.
6. Delete the Set Payload transformer in the get:/flights flow.

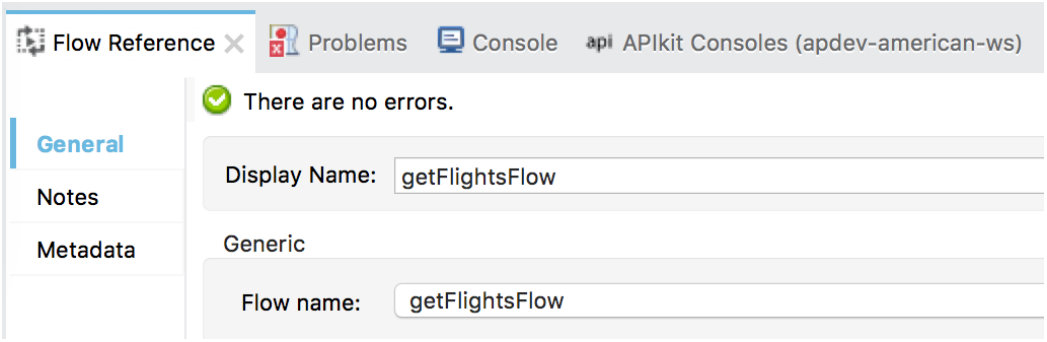
7. In the Mule Palette, select the Components tab.



8. Drag a Flow Reference component from the Mule Palette and drop it into the process section of the flow.



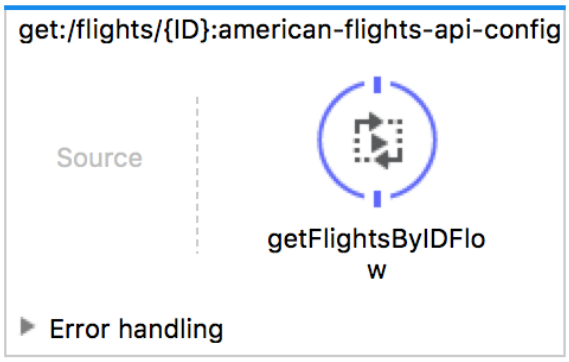
9. In the Flow Reference properties view, select getFlightsFlow for the flow name.



Set logic for the /flights/{ID} resource

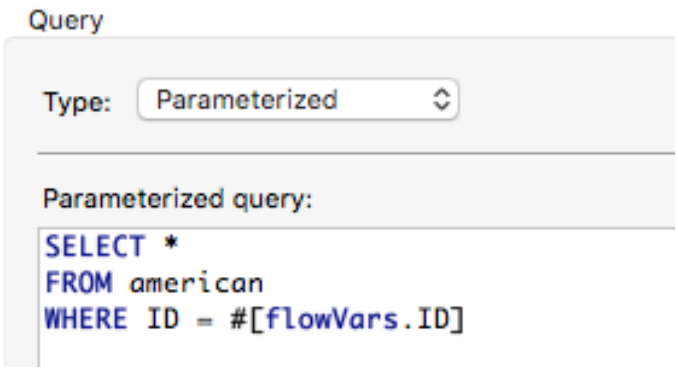
10. Delete the Set Payload transformer in the get:/flights/{ID} flow.
11. Drag a Flow Reference component from the Mule Palette and drop it into the flow.

12. In the Flow Reference properties view, select getFlightsByIDFlow for the flow name.



13. Return to implementation.xml.
14. Double-click the Database endpoint in getFlightsByIDFlow.
15. Change the query to use a variable instead of a query parameter.

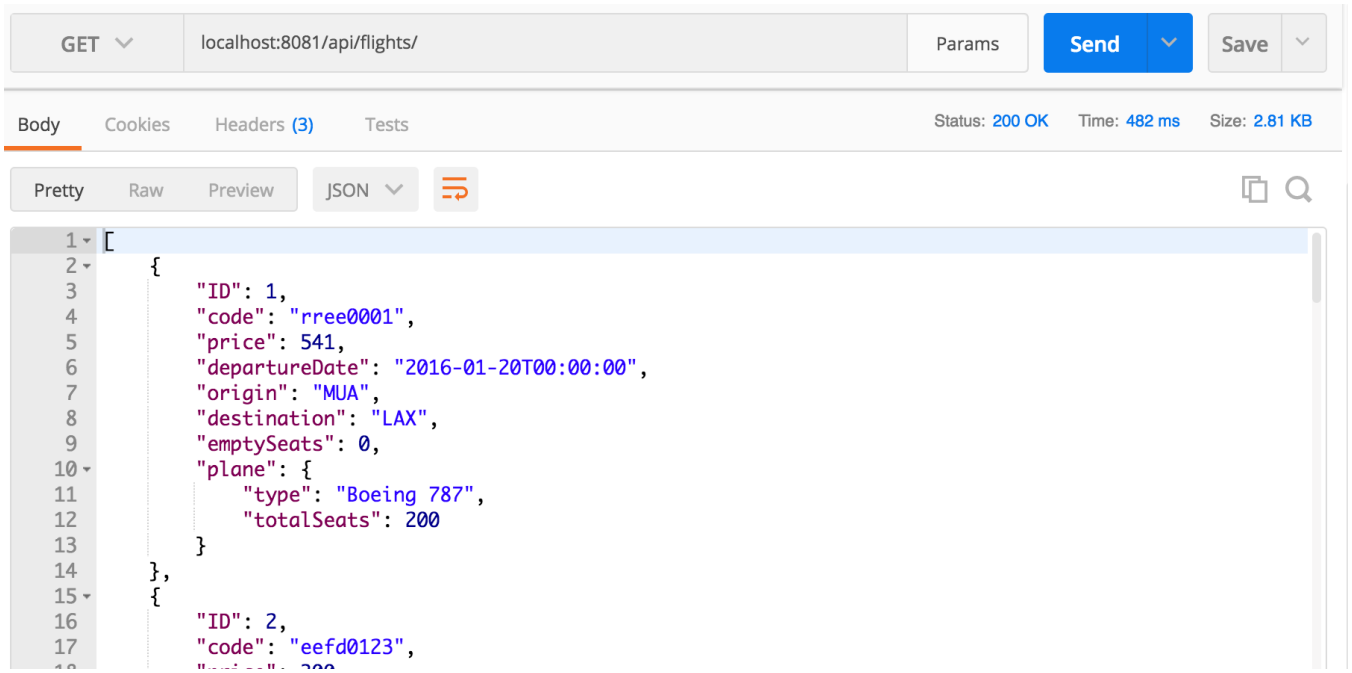
```
WHERE ID = #[flowVars.ID]
```



Note: You learn about the different types of variables in later modules in the Development Fundamentals course.

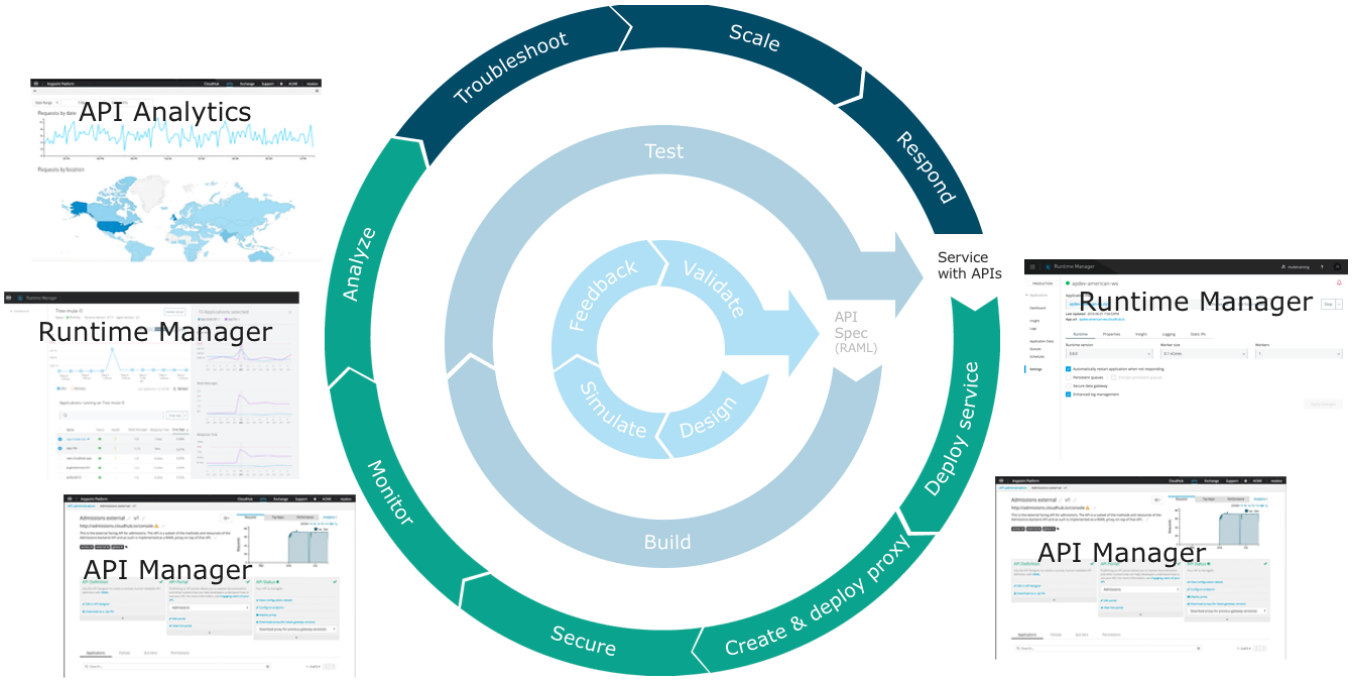
Test the web service using Postman

16. Run the project.
17. In Postman, make a request to http://localhost:8081/api/flights; you should now get the data for all the flights from the database instead of the sample data.



18. Make a request to http://localhost:8081/api/flights/3; you should now get the data that flight from the database instead of the sample data.
19. Return to Anypoint Studio and stop the project.

Module 5: Deploying and Managing APIs



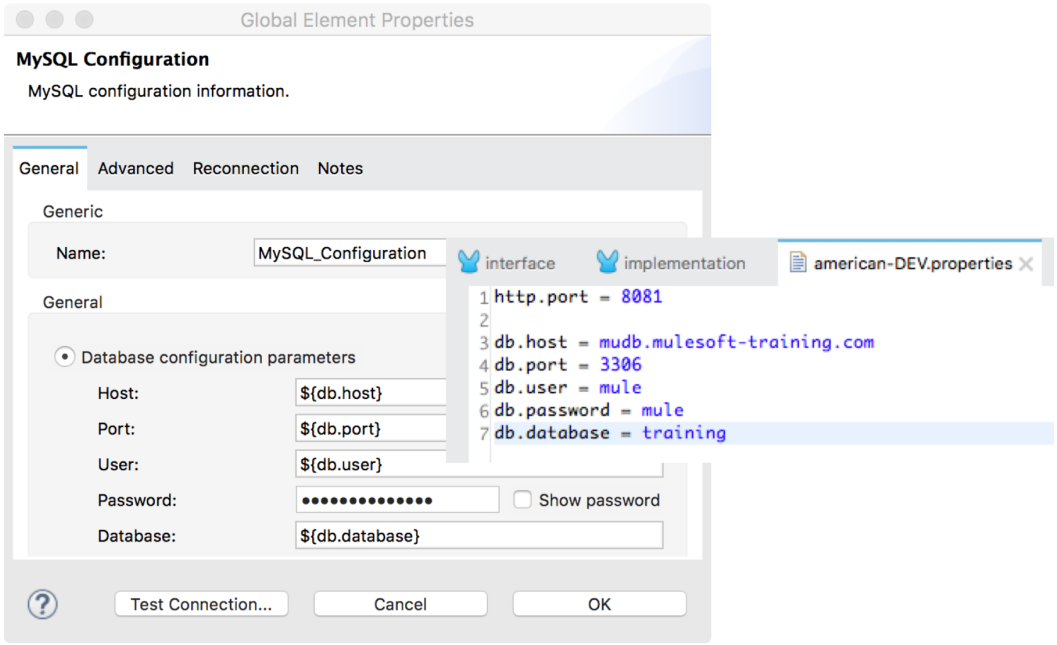
At the end of this module, you should be able to:

- Describe the options for deploying Mule applications.
- Use properties in Mule applications so they can be easily moved between environments.
- Deploy Mule applications to CloudHub.
- Use API Manager to create and deploy API proxies.
- Use API Manager to restrict access to API proxies.

Walkthrough 5-1: Prepare an API for deployment using properties

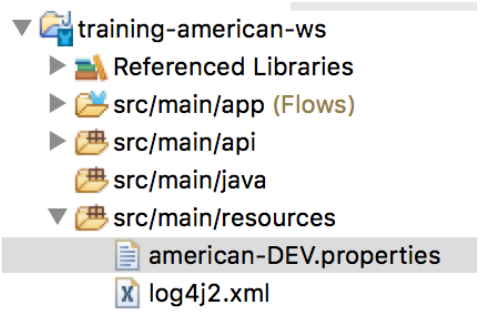
In this walkthrough, you introduce properties into your API implementation. You will:

- Create a properties file for an application.
- Create a Properties Placeholder global element to specify the properties file.
- Define and use Database connector properties.
- Specify a properties file dynamically.



Create a properties file

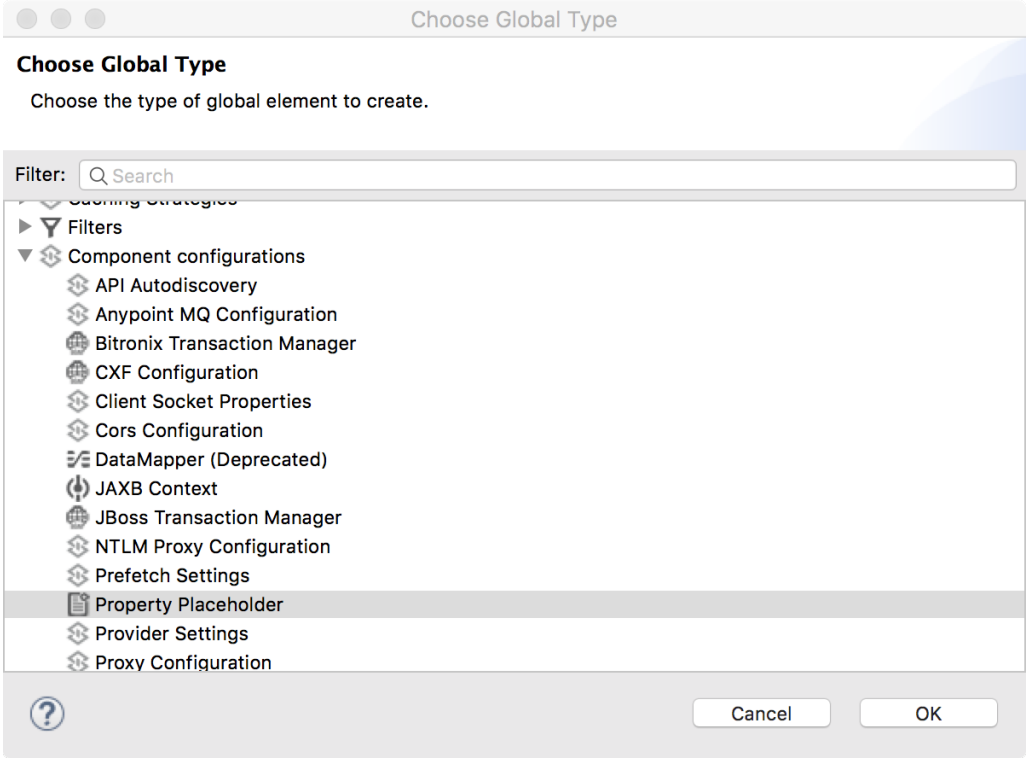
1. Right-click the src/main/resources folder in the Package Explorer and select New > File.
2. Set the file name to american-DEV.properties and click Finish.



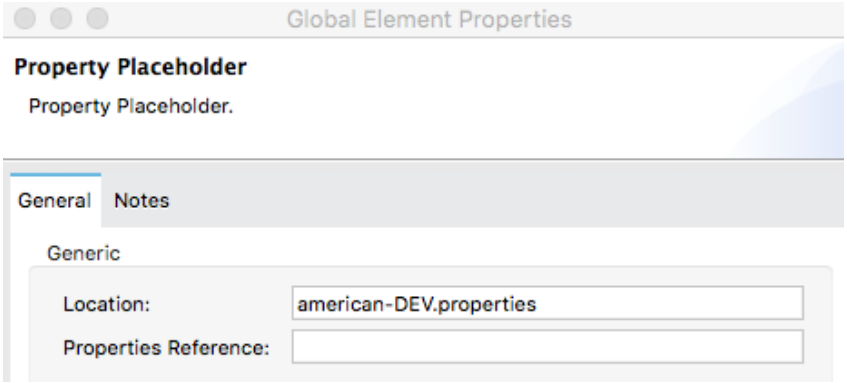
Create a Properties Placeholder global element

3. Return to interface.xml and go to its Global Elements view.

4. In the Global Elements view, click Create.
5. In the Choose Global Type dialog box, select Component configurations > Property Placeholder and click OK.



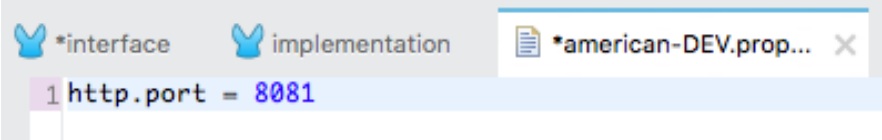
6. In the Global Element Properties dialog box, set the location to american-DEV.properties and click OK.



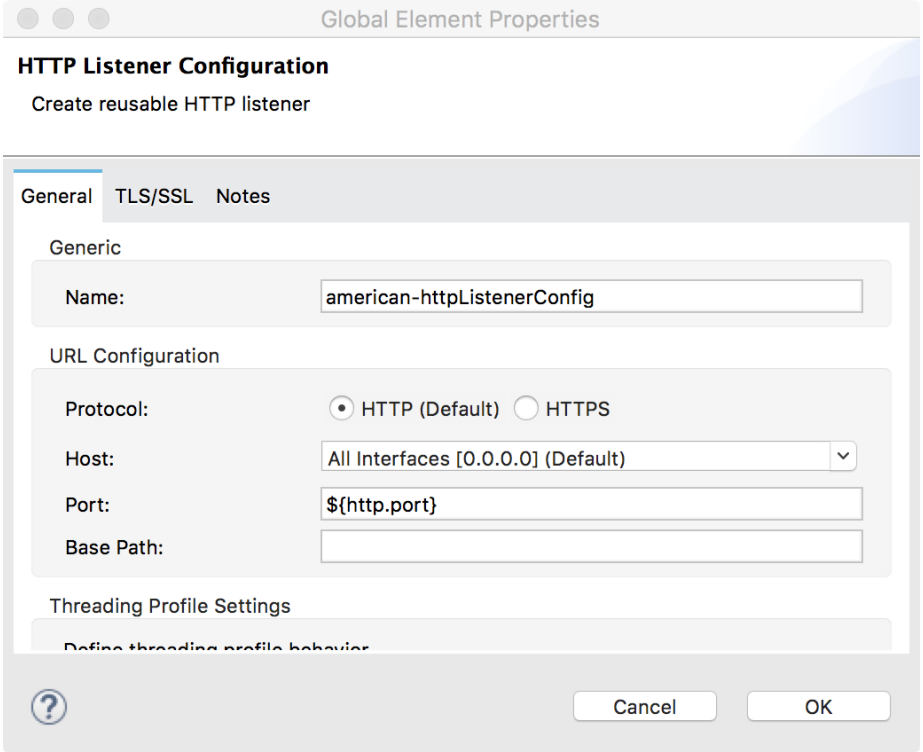
Parameterize the HTTP Listener port

7. Return to american-DEV.properties.

8. Create a property called http.port and set it to 8081.



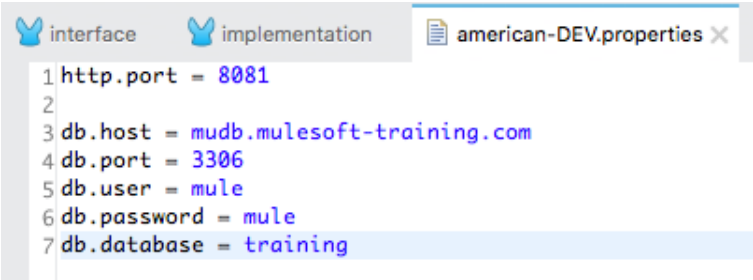
9. Save the file.
10. Return to the Global Elements view in interface.xml.
11. Double-click the HTTP Listener Configuration global element.
12. Change the port from 8081 to the application property, ${http.port}.
13. Click OK.



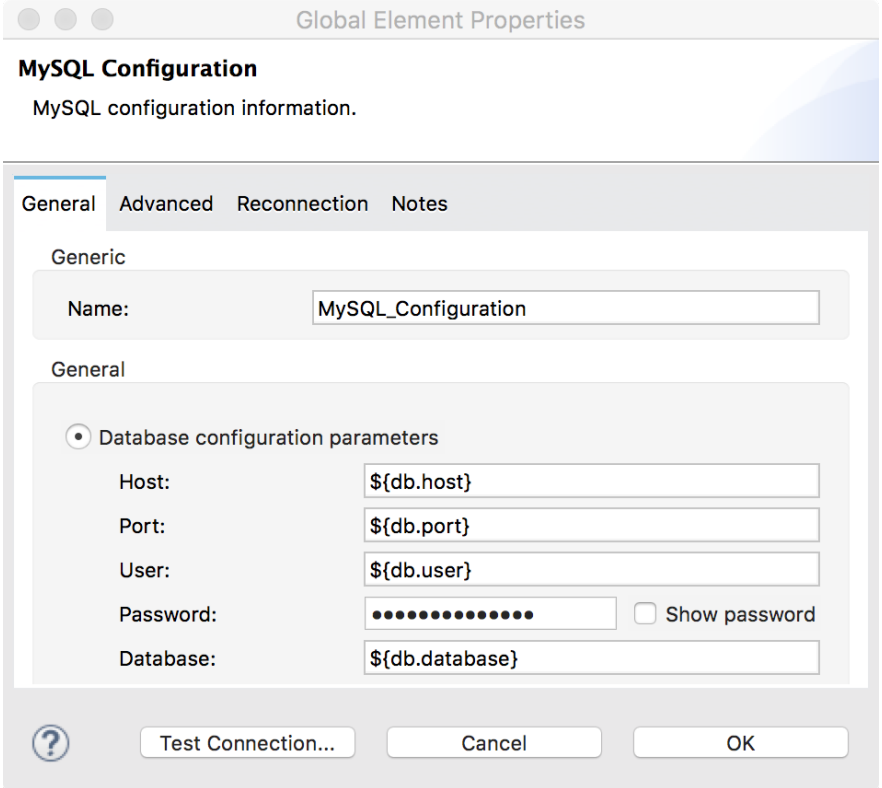
Parameterize the database credentials

14. Return to the course snippets.txt file and copy the database parameters (the five starting with db).

15. Return to american-DEV.properties and paste the values.



16. Save the file.
17. Return to the Global Elements view in implementation.xml.
18. Double-click the Database Configuration global element.
19. Change the values to use the application properties.



20. Click Test Connection and make sure it succeeds.

    Note: If your connection fails, click OK and then go back and make sure you saved american-DEV.properties.

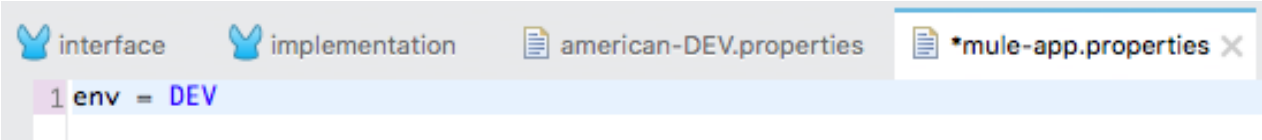
21. Click OK.
22. In the Global Element Properties dialog box, click OK.
23. Switch to the Message Flow view.

Test the application

24. Run the project.
25. In Postman, make a request http://localhost:8081/api/flights and confirm you still get data.

Define an environment property value in mule-app.properties

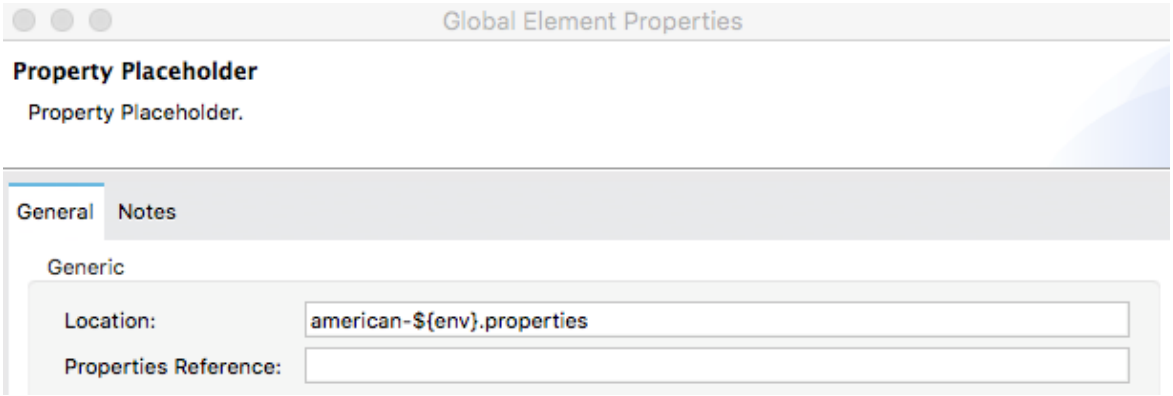
26. Return to Anypoint Studio and stop the project.
27. Open mule-app.properties located in the src/main/app folder.
28. Define a property called env and set it to DEV.



29. Save the file.

Use the environment property in the Property Placeholder

30. Return to the Global Elements view in interface.xml.
31. Double-click the Property Placeholder to edit it.
32. In the Global Element Properties dialog box, change the location to american-${env}.properties and click OK.



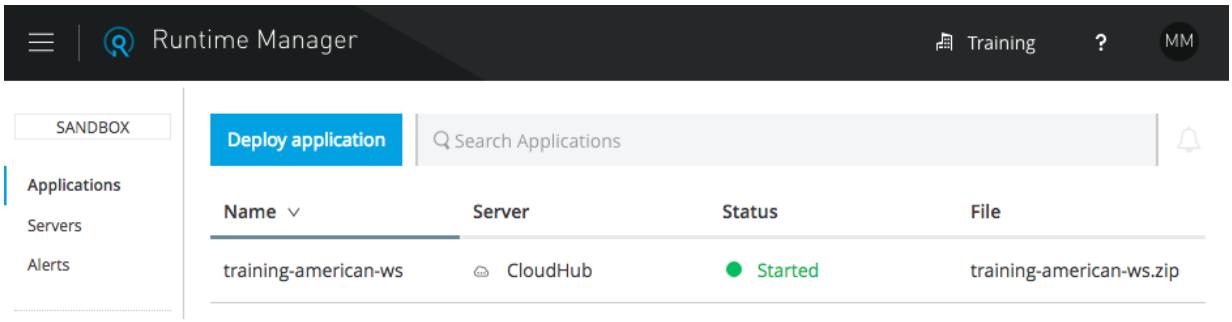
Test the application

33. Return to the Message Flow view in interface.xml.
34. Run the project.
35. In Postman, make a request http://localhost:8081/api/flights and confirm you still get data.
36. Return to Anypoint Studio and stop the project.

Walkthrough 5-2: Deploy an application to CloudHub

In this walkthrough, you deploy and run your application on CloudHub. You will:

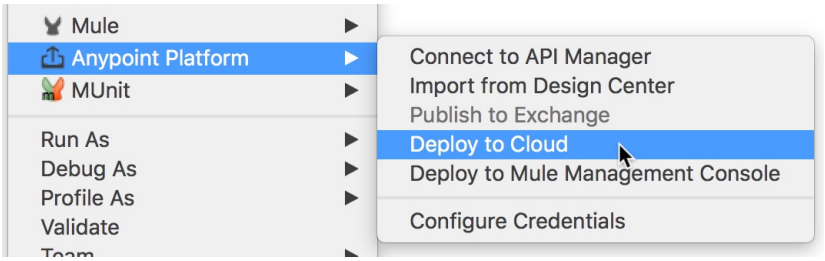
- Deploy an application from Anypoint Studio to CloudHub.
- Run the application on its new, hosted domain.
- Make calls to the web service.
- Update the API implementation deployed to CloudHub.



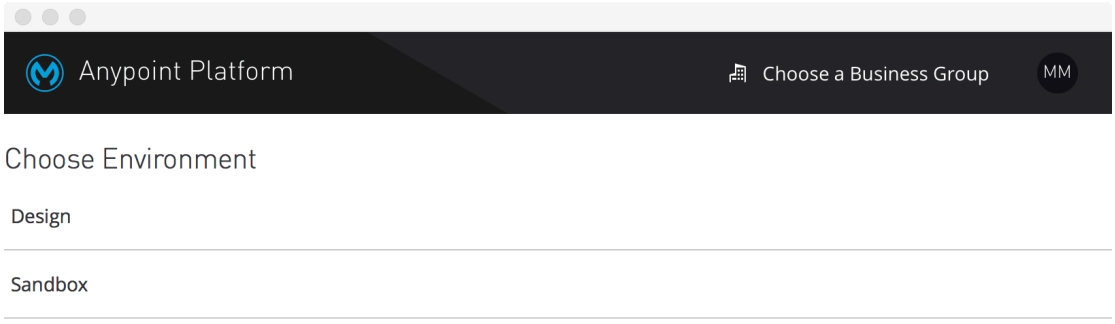
Note: If you do not have a working application at this point, import the training-american-ws-WT5.1 solution into Anypoint Studio and work with that project.

Deploy the application to CloudHub

1. Return to the training-american-ws project in Anypoint Studio.
2. In the Package Explorer, right-click the project and select Anypoint Platform > Deploy to Cloud.

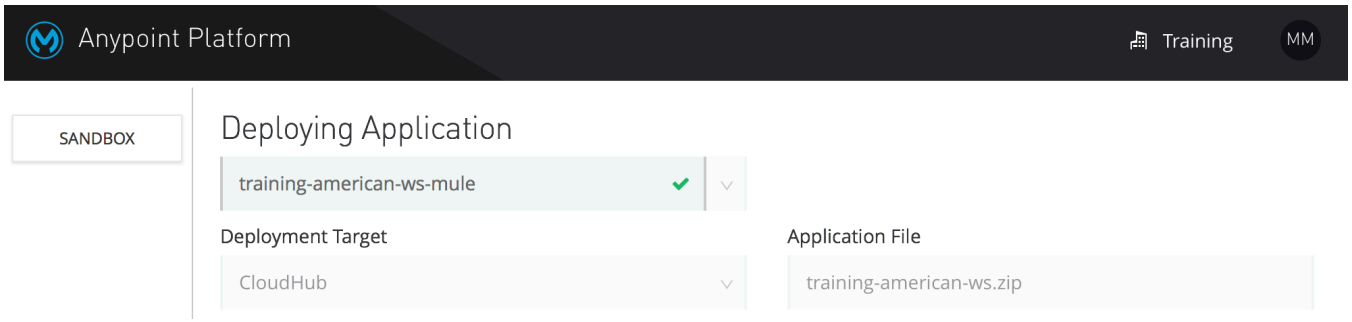


3. In the Choose Environment dialog box, select Sandbox.



4. At the top of the Anypoint Platform dialog box, set the application name to training-american-ws-*{your-lastname}* so it is a unique value.

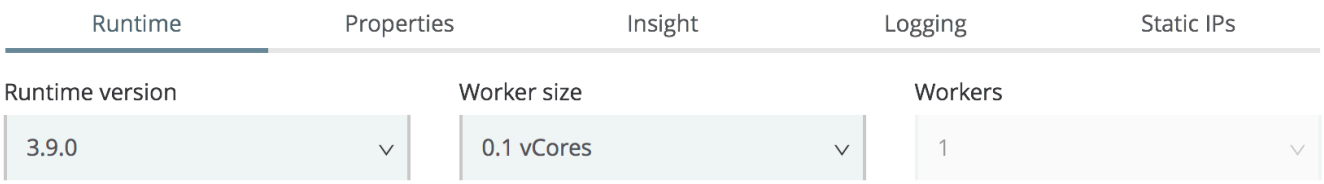
   Note: This name will be part of the URL used to access the application on CloudHub. It must be unique across all applications on CloudHub. The availability of the domain is instantly checked and you will get a green check mark if it is available.



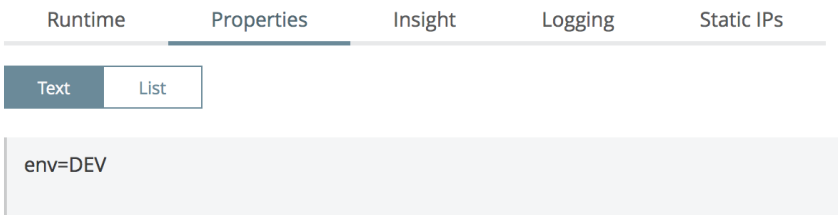
5. Make sure the runtime version is set to the version your project is using.

   Note: If you don't know what version it is using, look at the Package Explorer and find a library folder with the name of the server being used, like Mule Server 3.9.0 EE.

6. **Change the worker size to 0.1 vCores.**

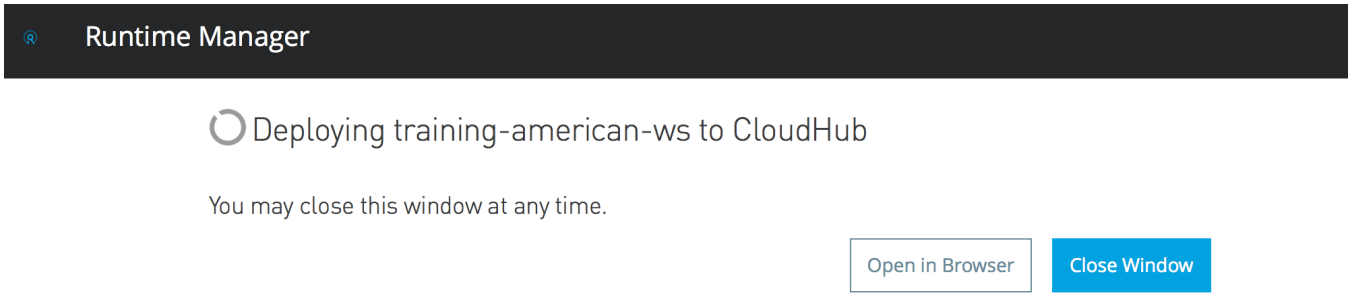


7. Click the Properties tab; you should see the env variable set to DEV.

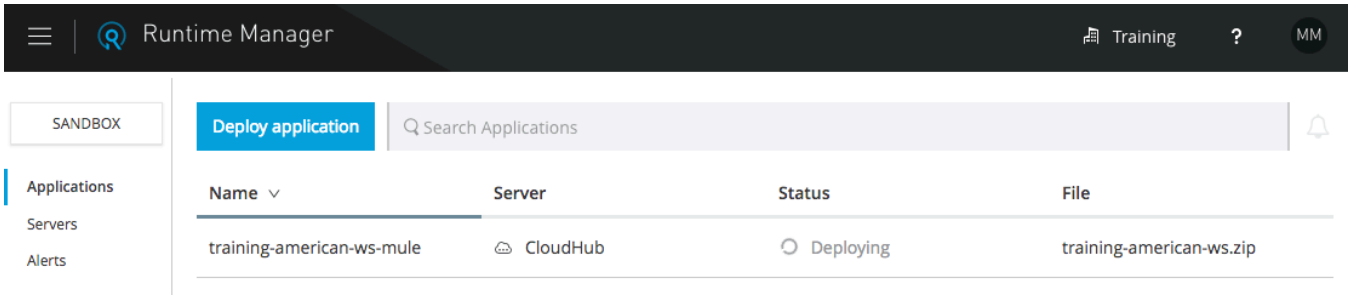


8. Click the Deploy Application button.

9. Click the Open in Browser button.

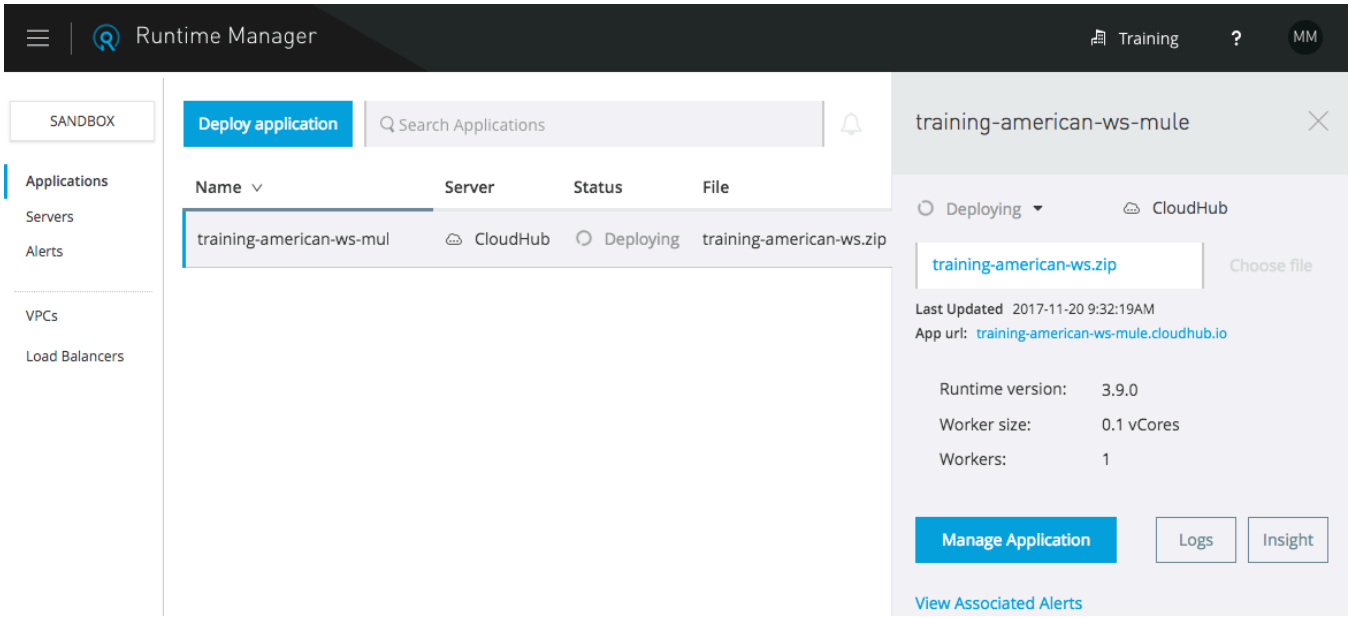


10. In the Anypoint Platform browser window that opens, locate the status of your deployment in the Runtime Manager.



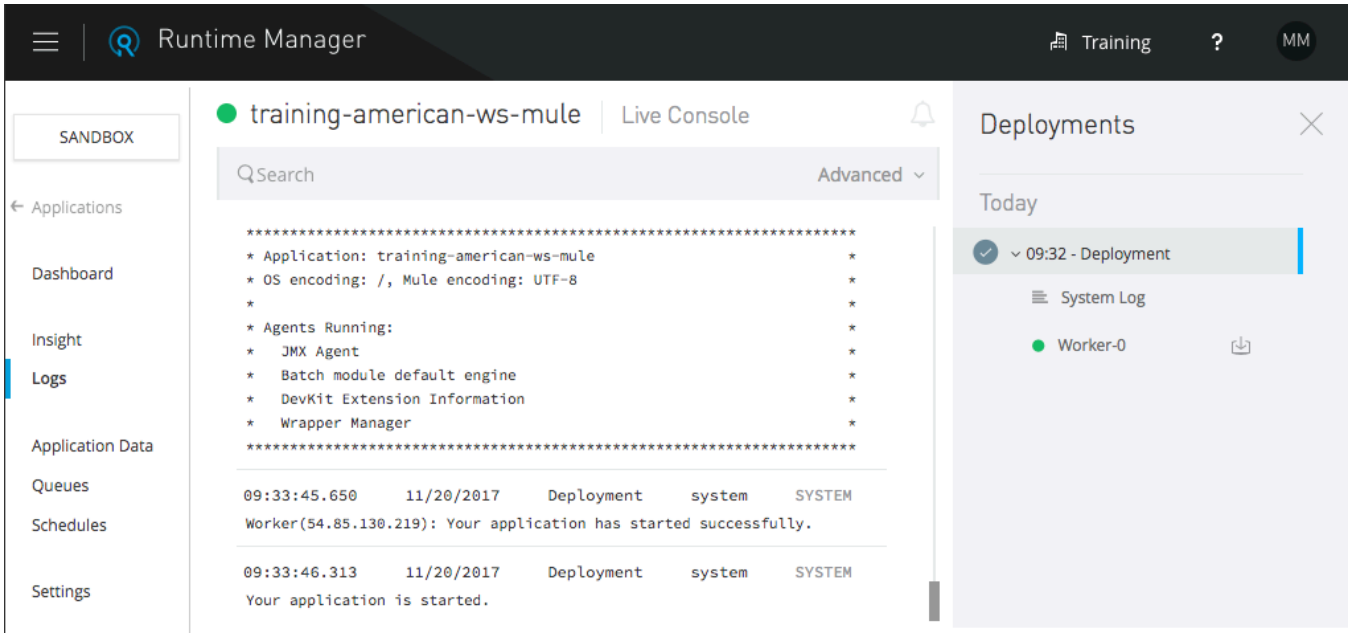
Watch the logs and wait for the application to start

11. Click in the row of the application (not on its name); you should see information about the application appear on the right side of the window.



12. Click the Logs button.

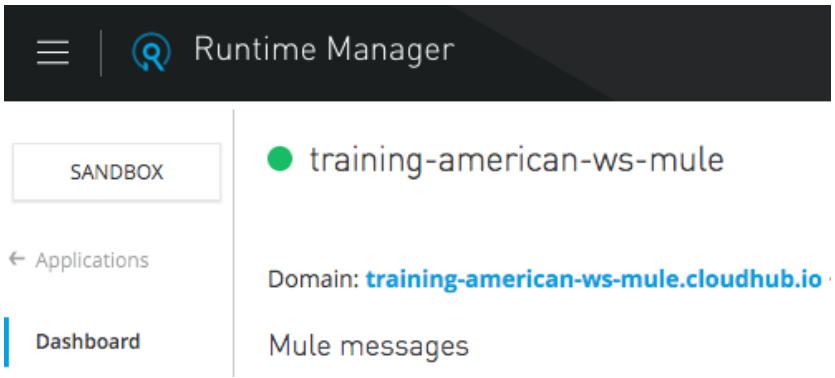
13. Watch the logs as the application is deployed.
14. Wait until the application starts (or fails to start).



Note: If your application did not successfully deploy, read the logs to help figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them, and then redeploy.

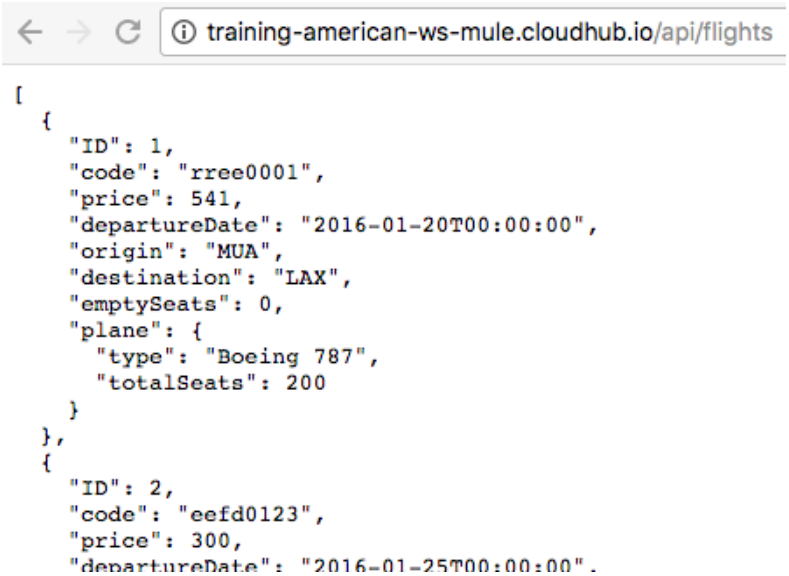
Test the application

15. In the left-side navigation, click Dashboard.
16. Locate the link for the application on its new domain: training-american-ws-*{lastname}*.cloudhub.io.



17. Click the link; a request will be made to that URL in an new browser tab and you should get a message that there is no listener for that endpoint.

18. Modify the path to http://training-american-ws-{*lastname*}.cloudhub.io/api/flights; you should see the flights data.



Note: If you are using the local Derby database, your application will not return results when deployed to CloudHub. You will update the application with a version using the MySQL database in the next section so it works.

19. Add a query parameter called destination to the URL and set it equal to SFO.
20. Send the request; you should still get all the flights.

Note: You did not add logic to the application to search for a particular destination. You will deploy an application with this additional functionality implemented next.

Note: If you wanted to test this API implementation from Exchange, you could by either 1) setting the baseUri in the API specification to the URL of the API implementation and then republishing the API specification or 2) clicking API instances in Exchange and adding a new API instance with this URL. Typically, however, you want to govern and manage access to an API. You will create a proxy to do this in the next walkthrough.

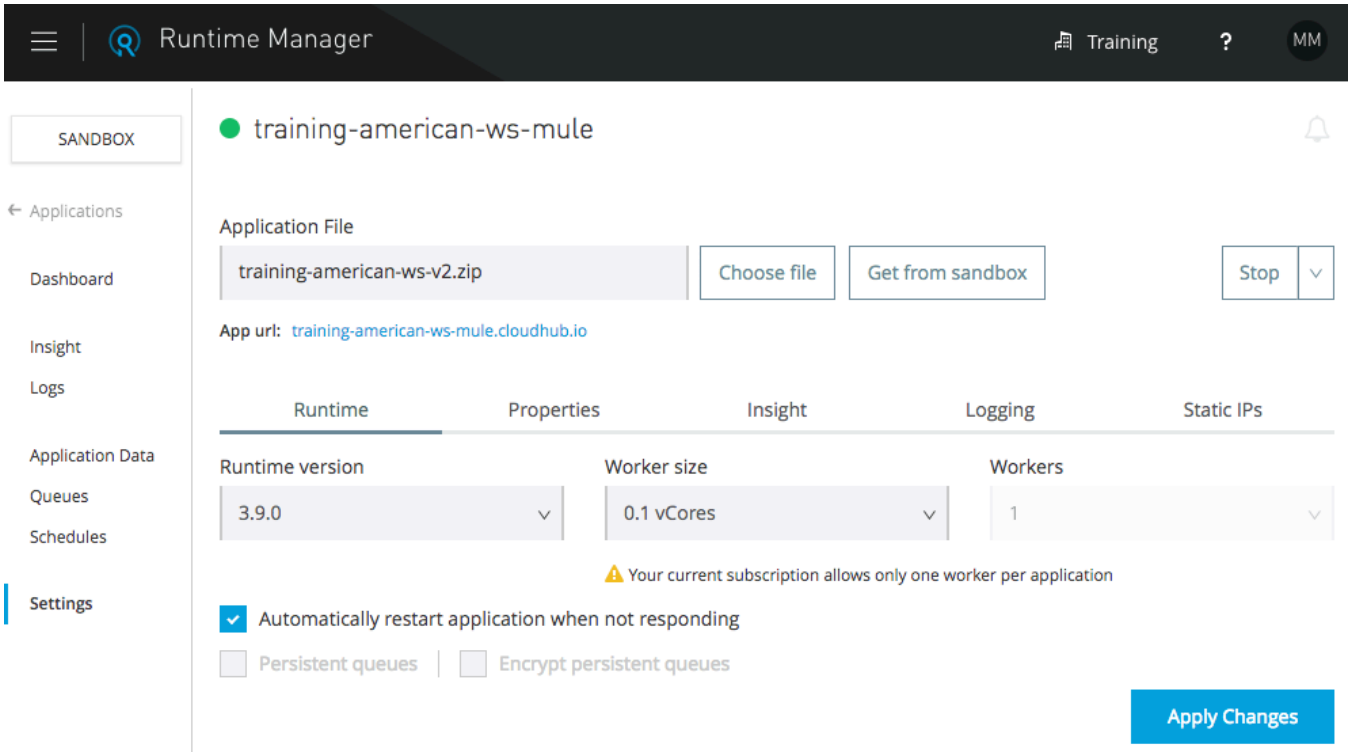
21. Leave this browser tab open.

Update the API implementation deployed to CloudHub

22. Return to the browser tab with Runtime Manager.
23. In the left-side navigation, click Settings for the training-american-ws-{*lastname*} application.
24. Click the Choose file button.
25. Browse to the resources folder in the course student files.
26. Select training-american-ws-v2.zip and click Open.

    Note: This updated version of the application adds functionality to return results for a particular destination. You will learn to do this later in the Development Fundamentals courses.

27. Click the Apply Changes button.



28. Wait until the application is uploaded and then redeploys successfully.

    Note: Because this can take some time for trial accounts, your instructor may move on with the next topic and then come back to test this later.

29. Close the browser tab with Runtime Manager.

Test the updated application

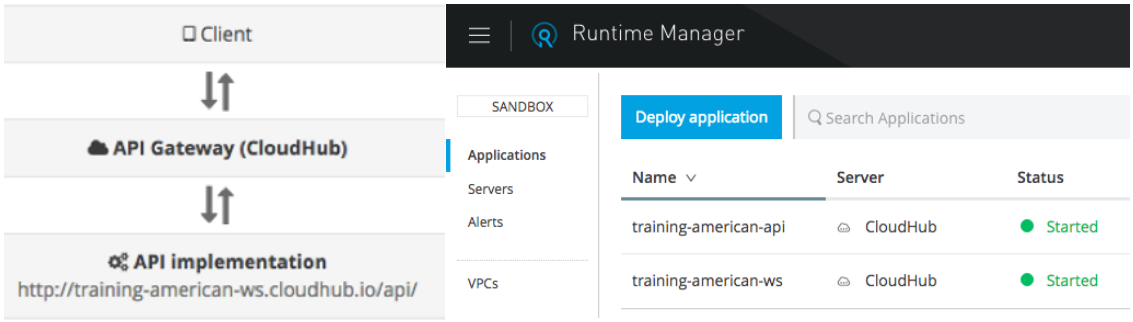
30. Return to the browser tab making a request to the API implementation on CloudHub with a destination of SFO and refresh it; you should now get only flights to SFO.

31. Close this browser tab.

Walkthrough 5-3: Create and deploy an API proxy

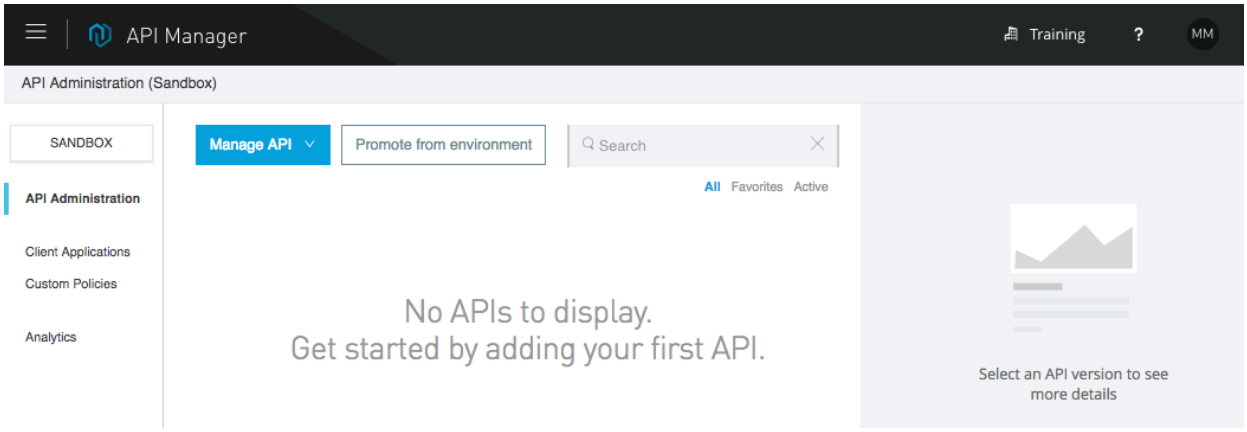
In this walkthrough, you create and deploy an API proxy for your API implementation on CloudHub. You will:

- Add an API to API Manager.
- Use API Manager to automatically create and deploy an API proxy application.
- Set a label and consumer endpoint for a proxy so requests can be made to it from Exchange.
- Make calls to the API proxy from API portals for both internal and external developers.
- View API request data in API Manager.

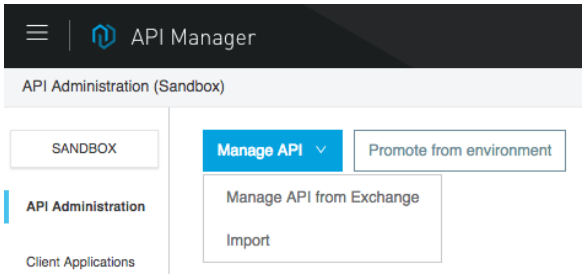


Create and deploy a proxy application

1. Return to the browser tab with Runtime Manager.
2. In the main menu, select API Manager; you should see no APIs listed.

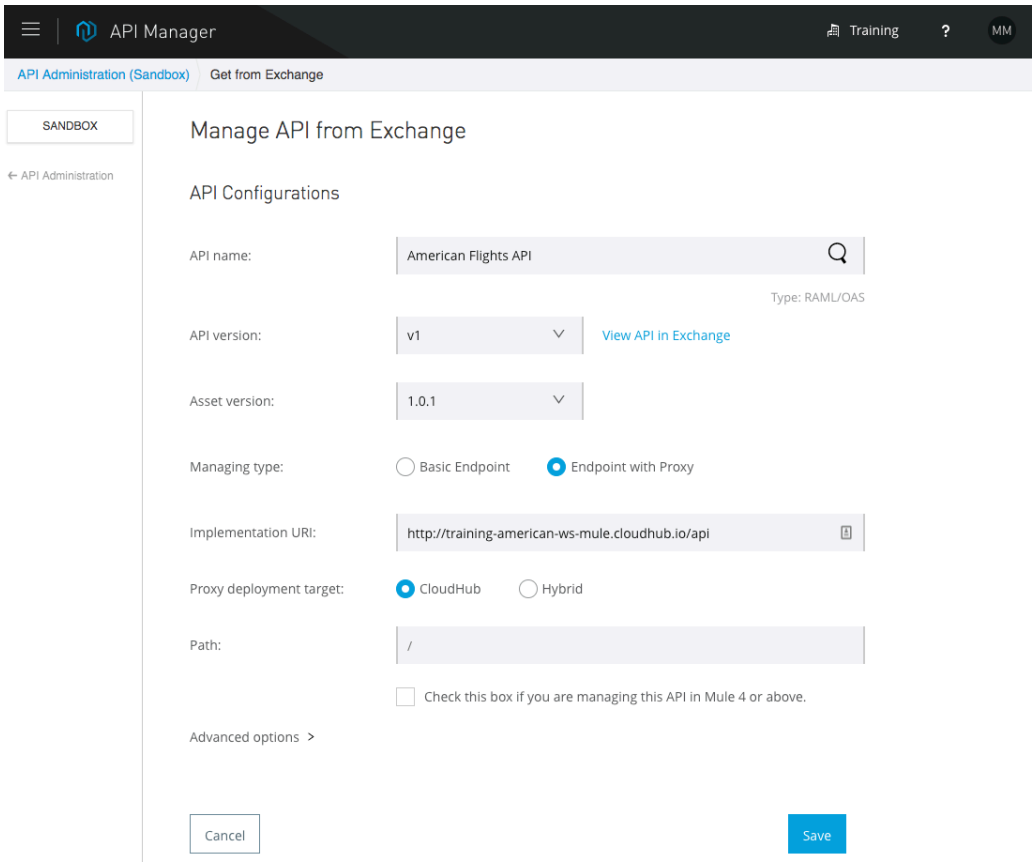


3. Click the Manage API button and select Manage API from Exchange.



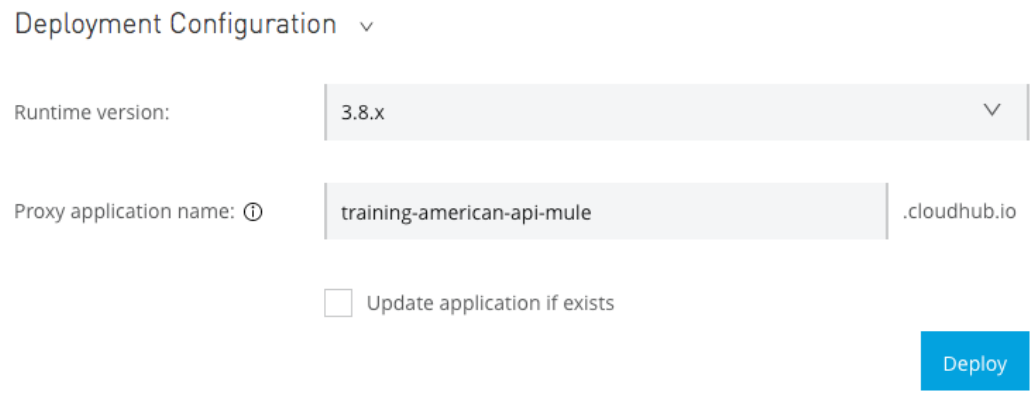
4. For API name, start typing American in the text field and then select your American Flights API in the drop-down menu that appears.
5. Set the rest of the fields to the following values:

   - API version: v1
   - Asset version: 1.0.1
   - Managing type: Endpoint with Proxy
   - Implementation URI: http://training-american-ws-{*lastname*}.cloudhub.io/api
   - Proxy deployment target: CloudHub

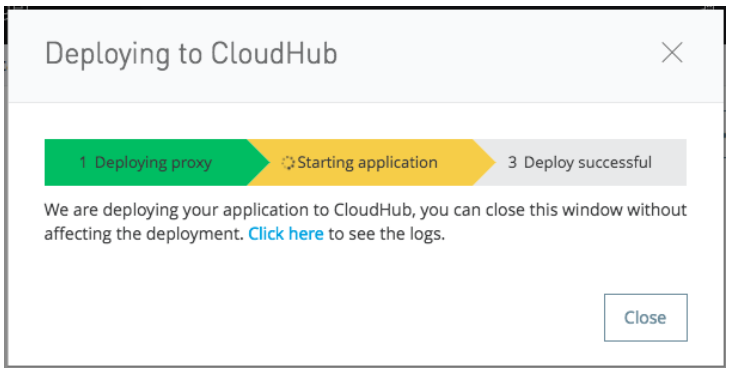


6. Click Save.

7. In the Deployment Configuration section, set the following values:
   - Runtime version: 3.8.x (or a later value)
   - Proxy application name: training-american-api-{*lastname*}

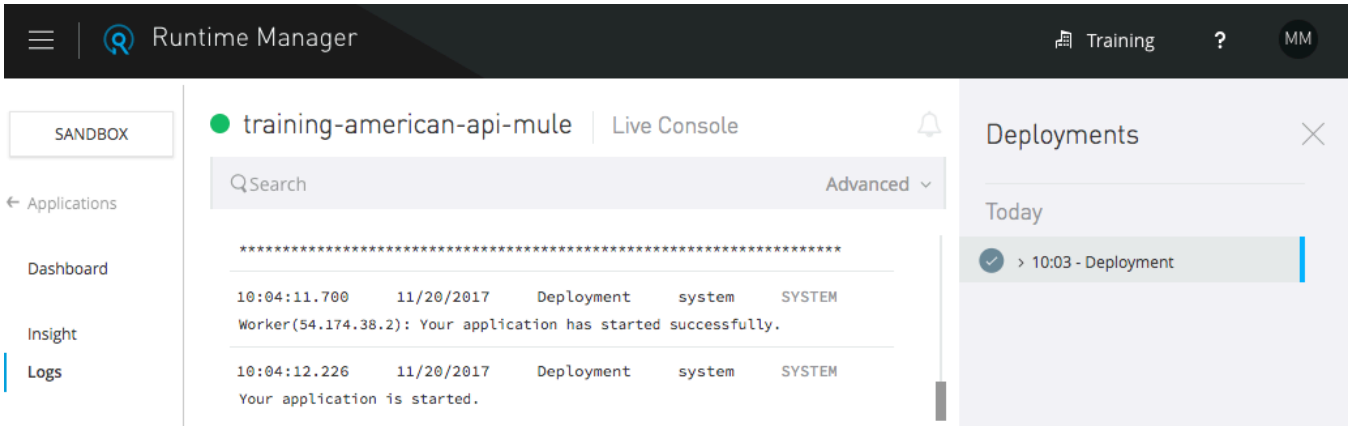


8. Click Deploy.
9. In the Deploying to CloudHub dialog box, click the Click here link to watch the logs.

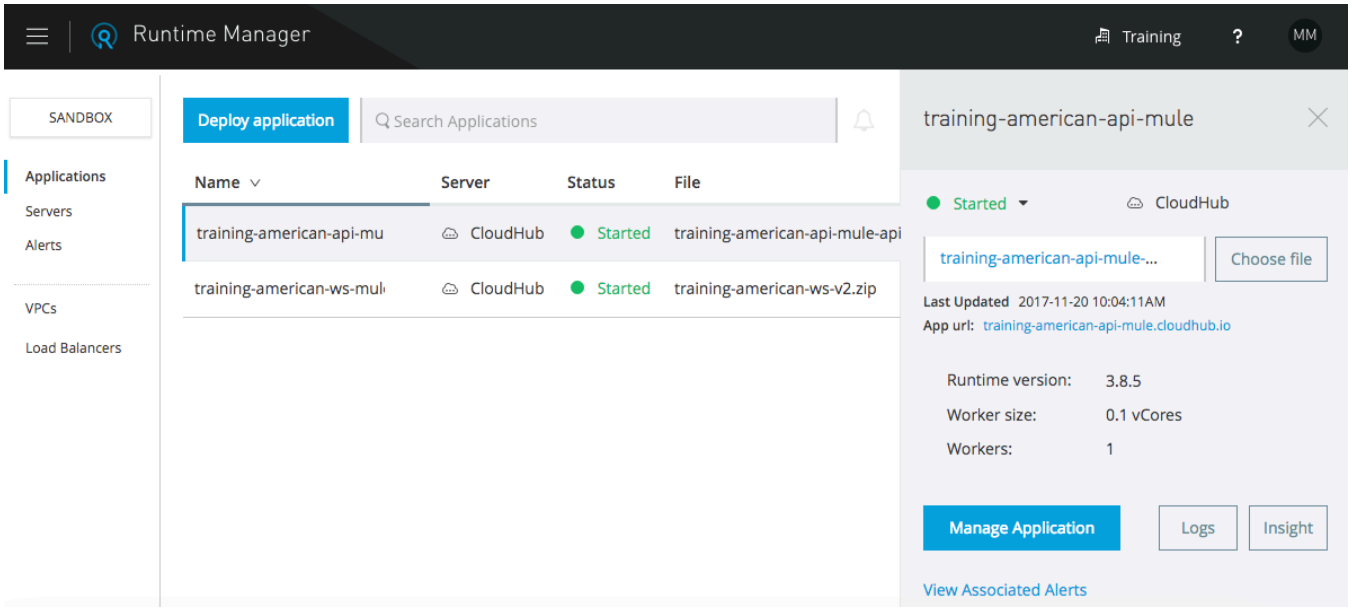


10. In the new browser tab that opens, watch the logs in Runtime Manager.
11. Wait until the proxy application starts.

    Note: If it does not successfully deploy, read the logs to help figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them, and then redeploy.



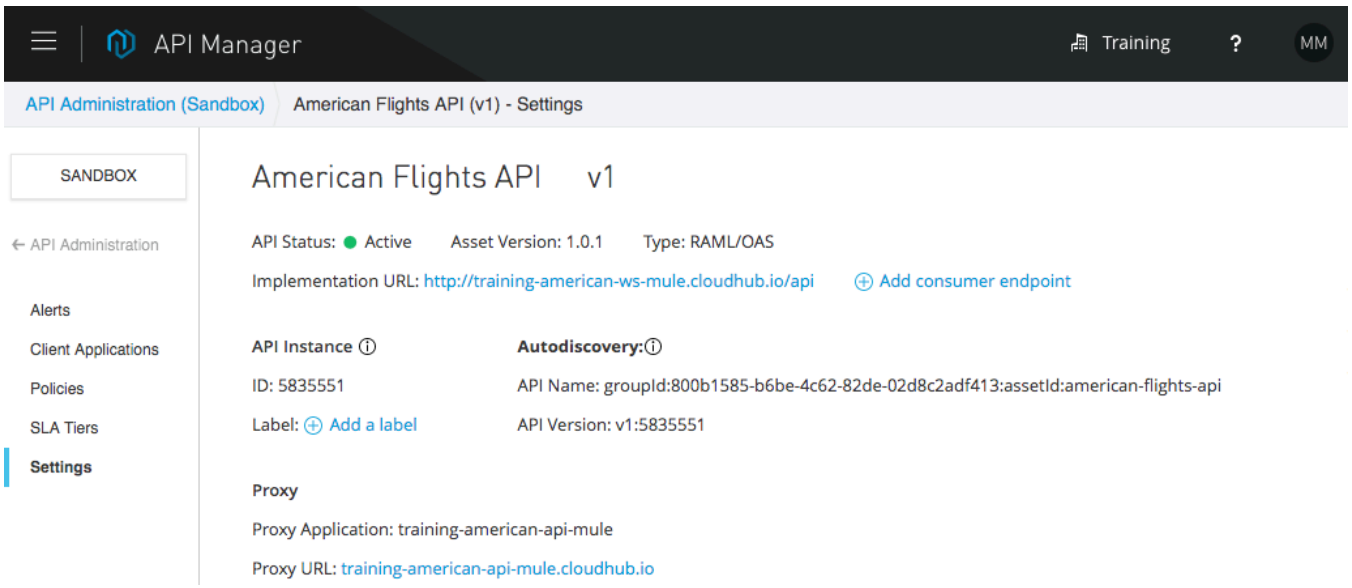
12. In the left-side navigation, click Applications; you should see the proxy application.
13. Click the row for the proxy and review its information in the right section of the window.



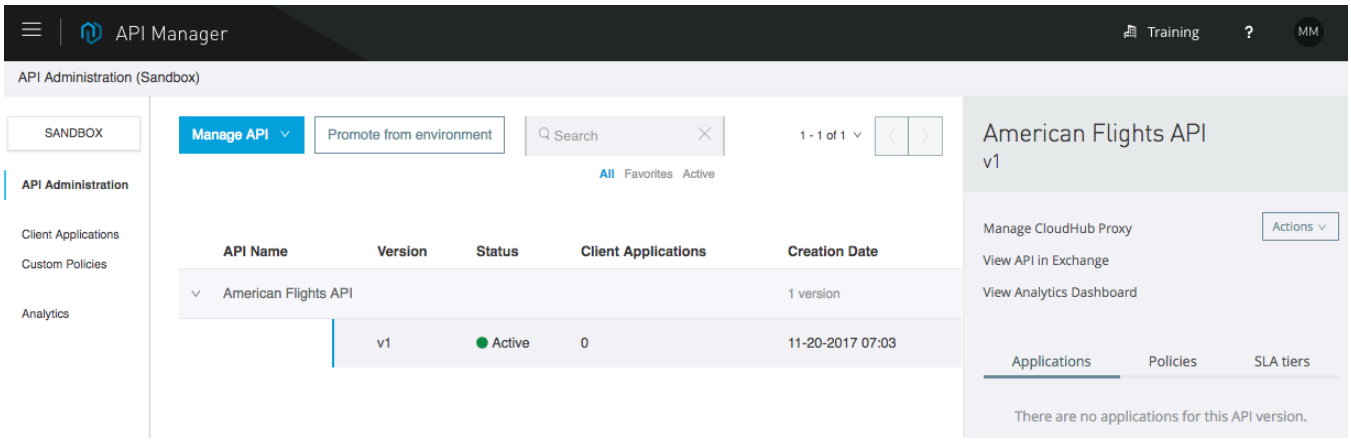
14. Close the browser tab.

View API details in API Manager

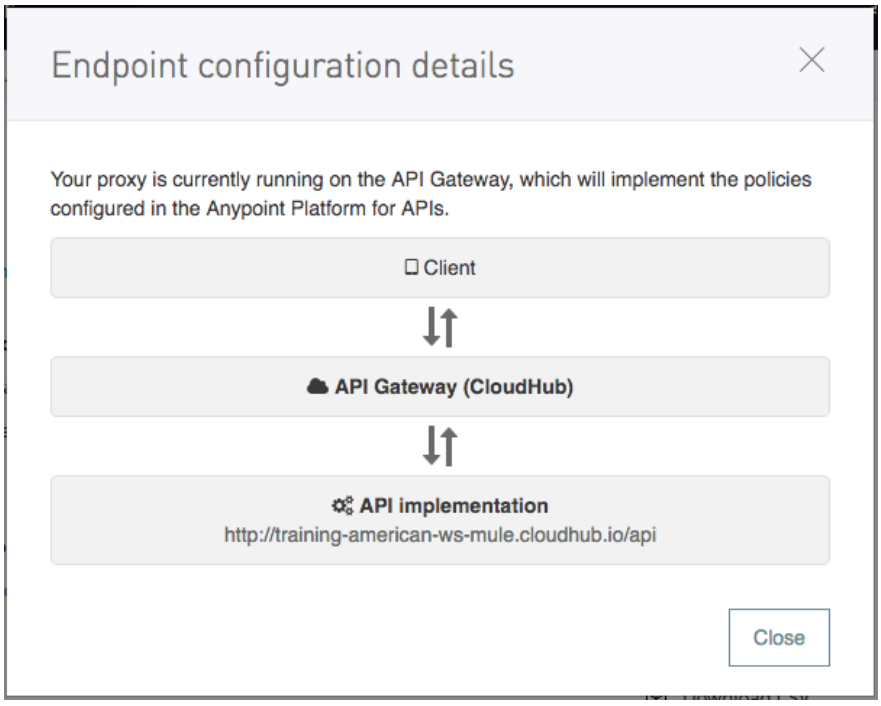
15. Return to the tab with API Manager and click the Close button in the Deploying to CloudHub dialog box.
16. Review the API proxy information at the top of the page.



17. In the left-side navigation, click the API Administration link; you should now see your American Flights API listed.
18. Click in the row for the v1 version – but not on the v1 link.
19. Review the API version info that appears on the right side of the window; you should see there are no policies, SLA tiers, or client applications.

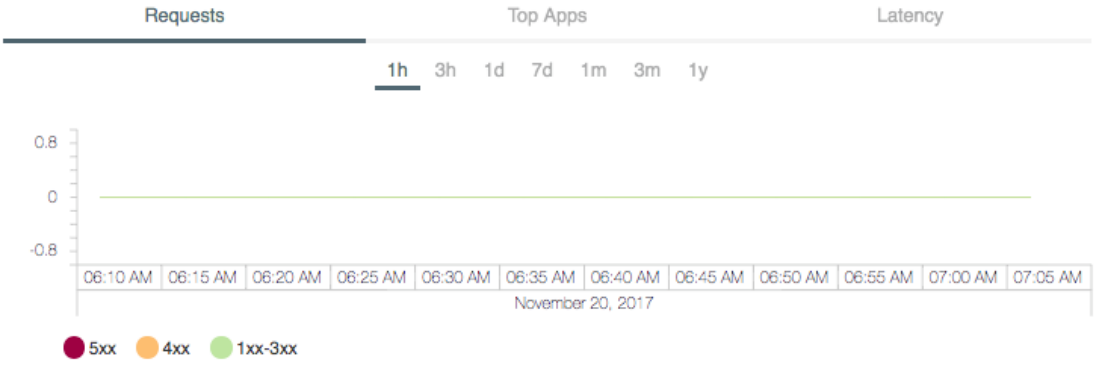


20. In the API list, click the v1 link for the API; you should be returned to the Settings page for the API.
21. Locate and review the links on the right side of the page.
22. Click the View configuration details link.



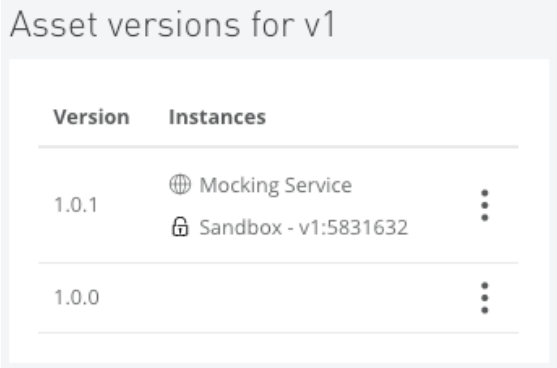
23. In the Endpoint configuration details dialog box, click Close.

24. On the Settings page, look at the requests graph; you should not see any API requests yet.



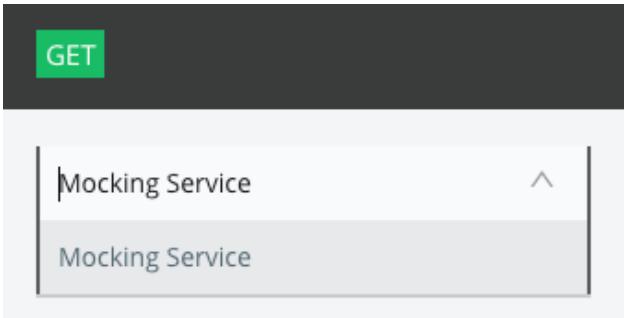
View the new API proxy instance in Exchange

25. Return to the browser tab with Exchange.
26. Return to the home page for your American Flights API.
27. Locate the API instances now associated with asset version 1.0.1; you should see the Mocking Service instance and now the new proxy.

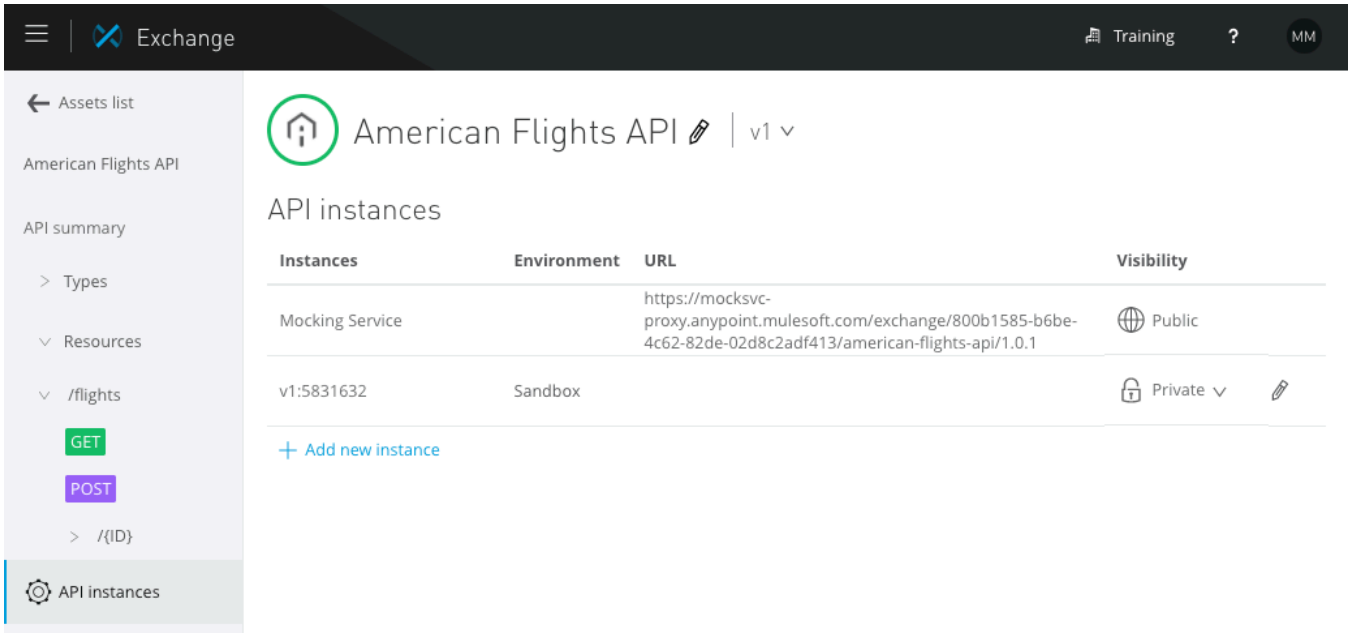


Note: You may need to refresh your browser page.

28. Click the GET method for the flights resource.
29. In the API console, click the drop-down arrow next to Mocking Service; you should NOT see the API proxy as a choice.

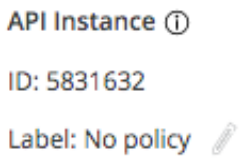


30. In the left-side navigation, click API instances; you should see that the new proxy instance does not have a URL.



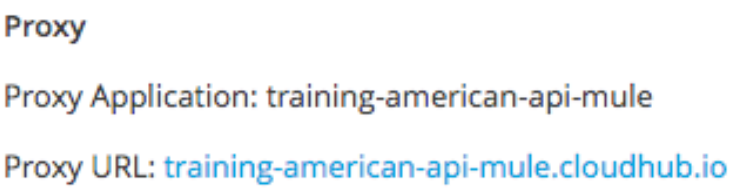
Set a friendly label for the API instance in API Manager

31. Return to the browser tab with API Manager.
32. On the Settings page for your American Flights API, click the Add a label link.
33. Set the label to No policy and press Enter/Return.



Set a consumer endpoint for the proxy in API Manager

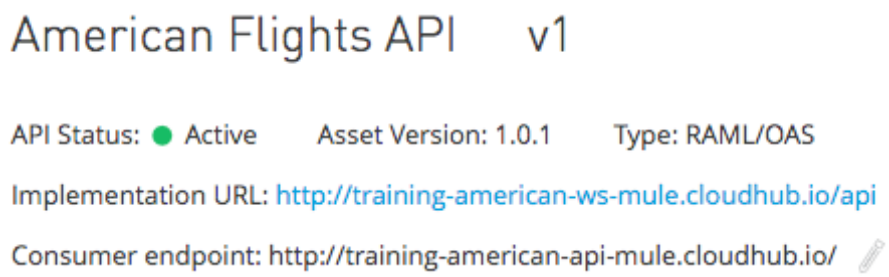
34. Locate the proxy URL.



35. Right-click it and copy the link address.

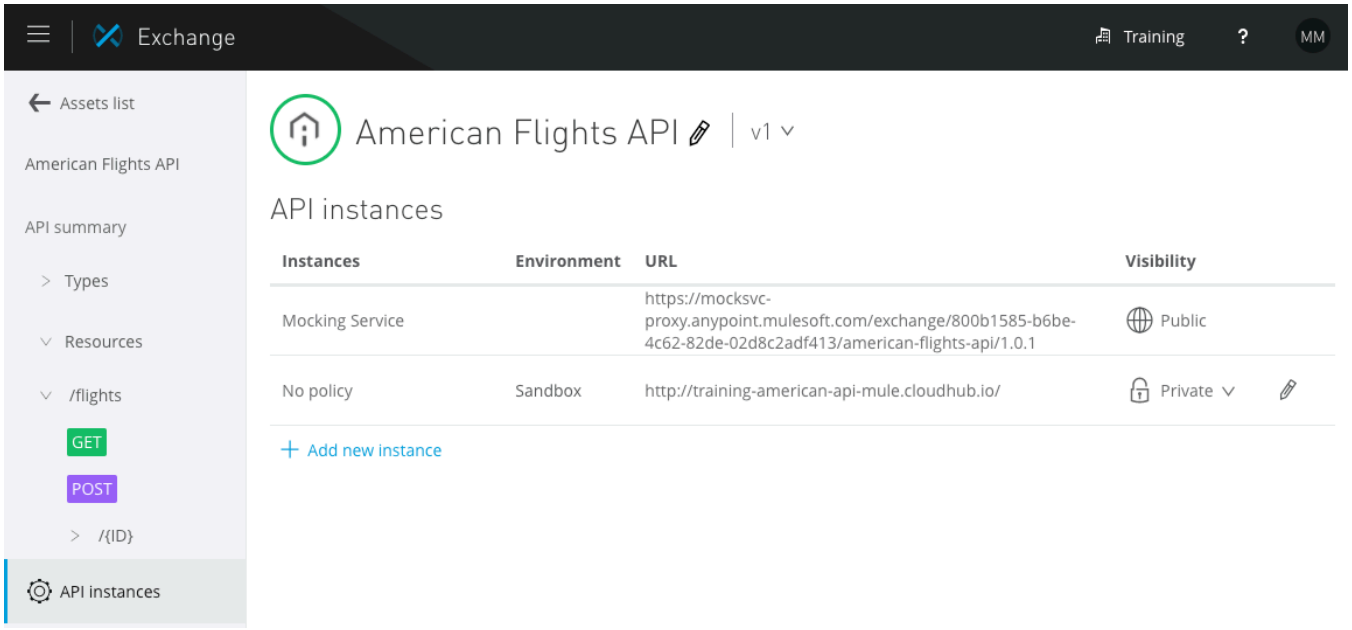
36. Click the Add consumer endpoint link.

37. Paste the value of the proxy URL.
38. Press Enter/Return.



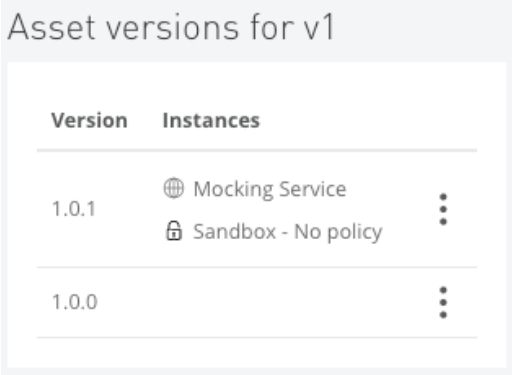
Make requests to the API proxy from Exchange

39. Return to the browser tab with Exchange.
40. Refresh the API instances page for your American Flights API; you should see the new label and the URL.

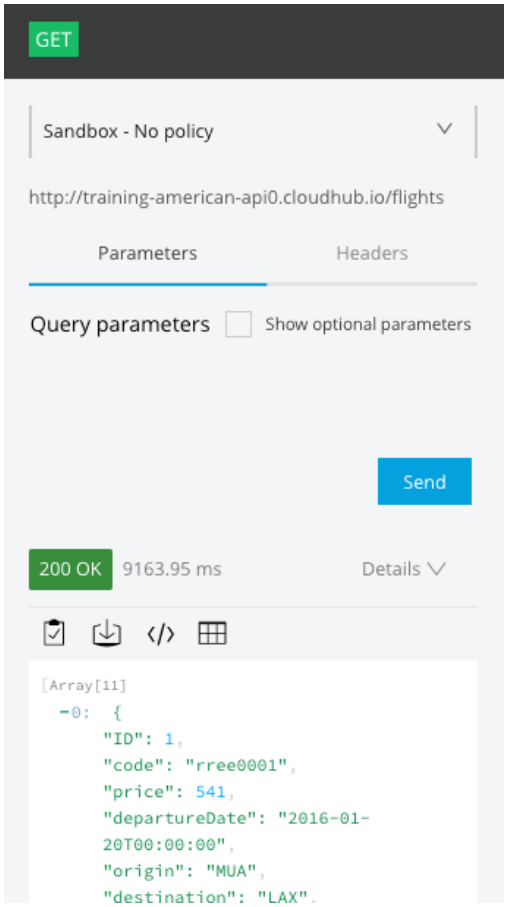


41. In the left-side navigation, click the name of the API to return to its main page.

42. Locate the API instances now associated asset version 1.0.1; you should see the new label for the API instance.



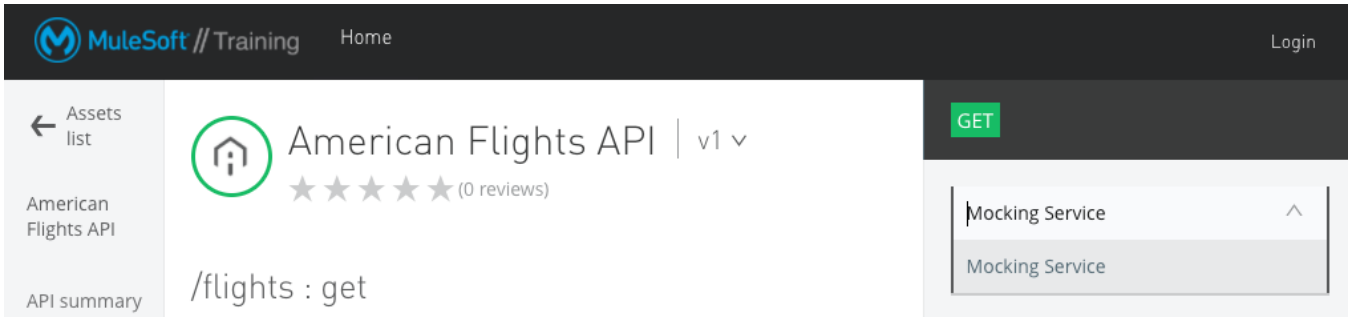
43. Click the GET method for the flights resource.
44. In the API console, click the drop-down arrow next to Mocking Service; you should now see your API proxy instance as a choice.
45. Select the Sandbox - No policy instance.
46. Click the Send button; you should now see the real data from the database, which contains multiple flights.



47. Make several more calls to this endpoint.
48. Make calls to different methods.

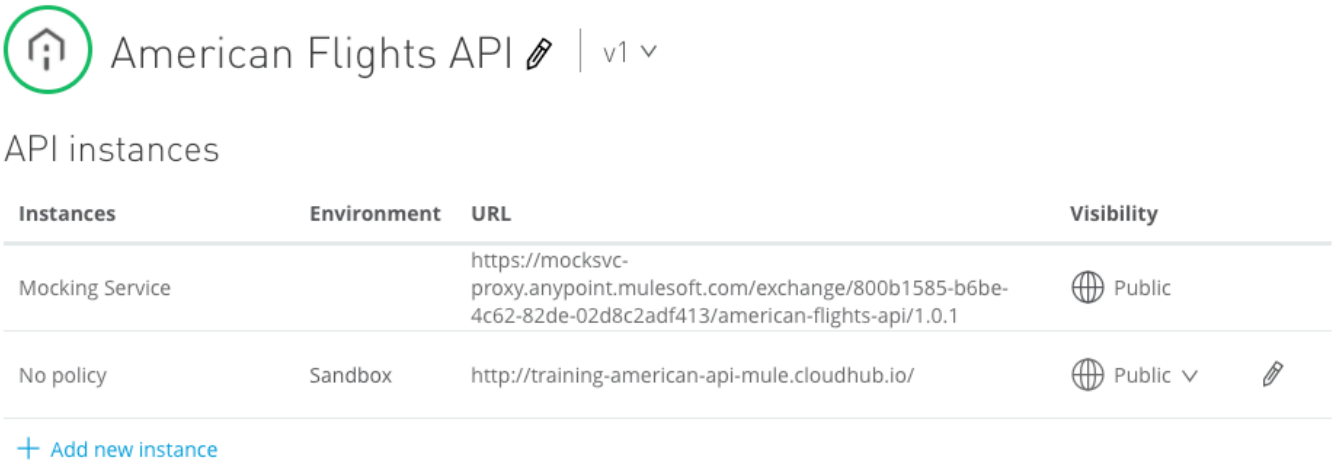
Make requests to the API proxy from the public portal

49. Return to the public portal in the private/incognito window.
50. Click the GET method for the flights resource.
51. In the API reference section, click the drop-down arrow next to Mocking Service; you should NOT see your API proxy instance as a choice.



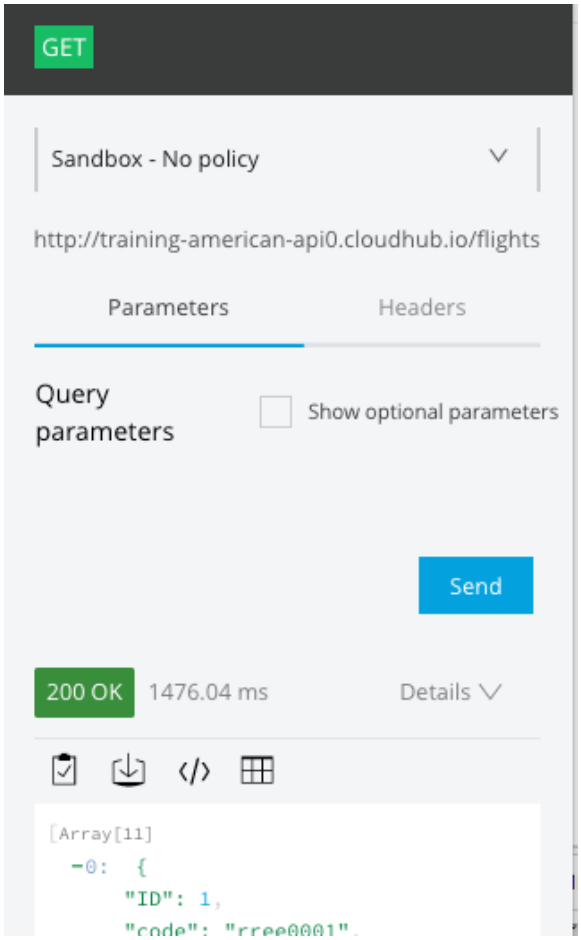
Make an API instance visible in the public portal

52. Return to the browser with Exchange.
53. In the left-side navigation, click API instances.
54. Change the visibility of the No policy instance from private to public.



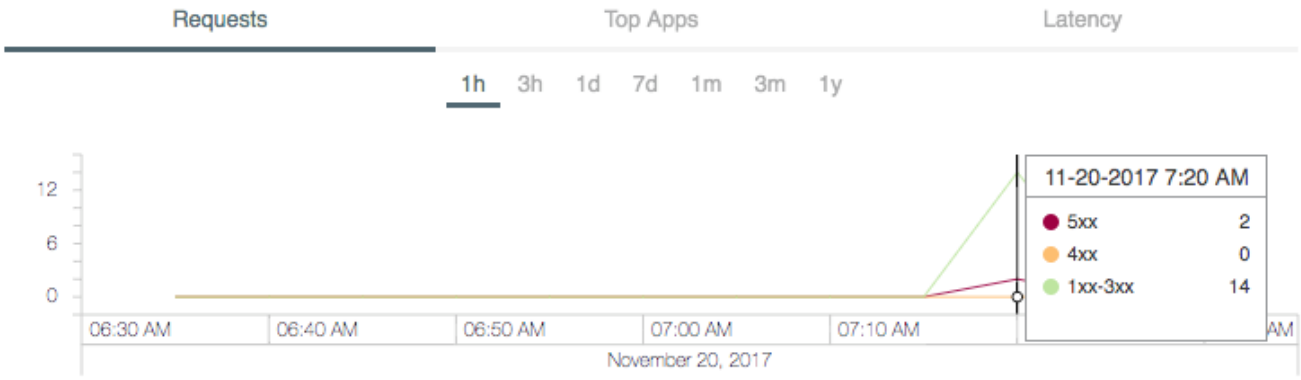
55. Return to the public portal in the private/incognito window.
56. Refresh the page.

57. In the API console, change the API instance from Mocking Service to Sandbox - No policy.
58. Click Send; you should get data.

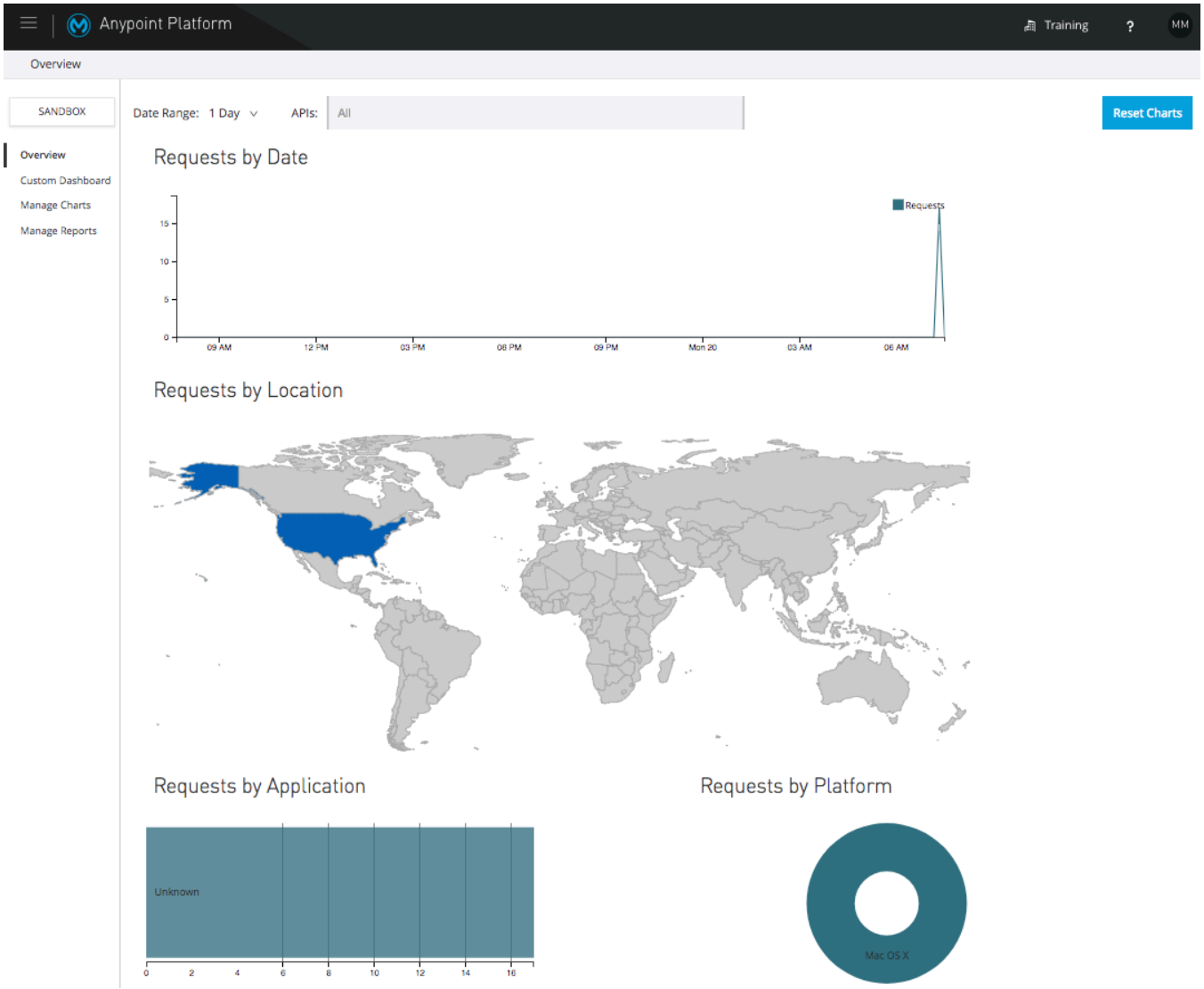


Look at the API request data

59. Return to the browser tab with API Manager.
60. Refresh the Settings page for you American Flights API.
61. Look at the Request chart again; you should now see data for some API calls.



62. Click the View Analytics Dashboard link located in the upper-right corner.
63. Review the data in the dashboard.

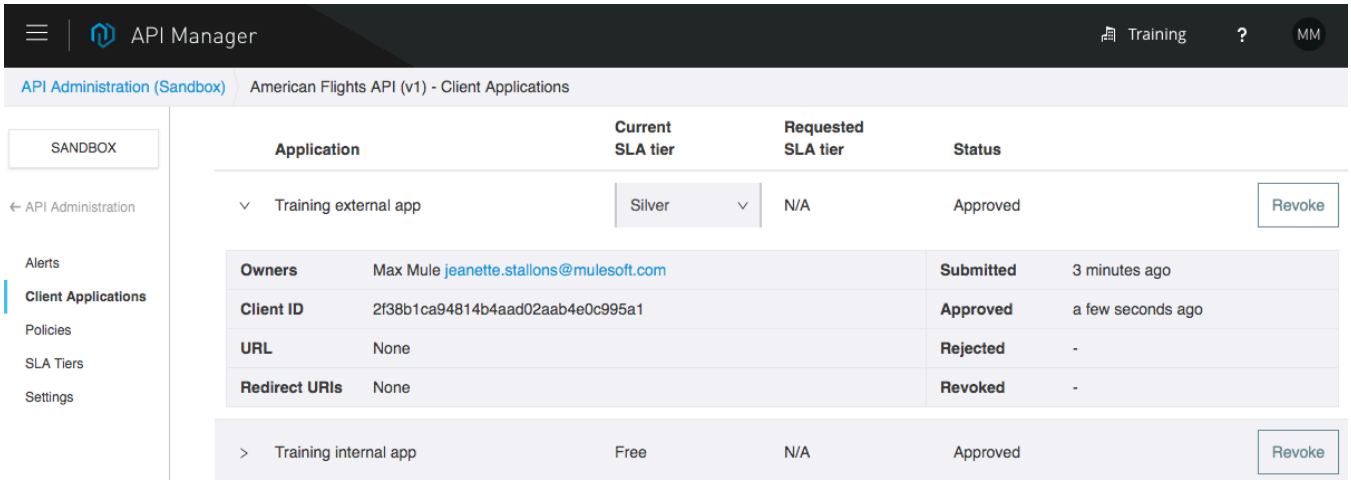


64. Close the browser tab.

Walkthrough 5-4: Restrict API access with policies and SLAs

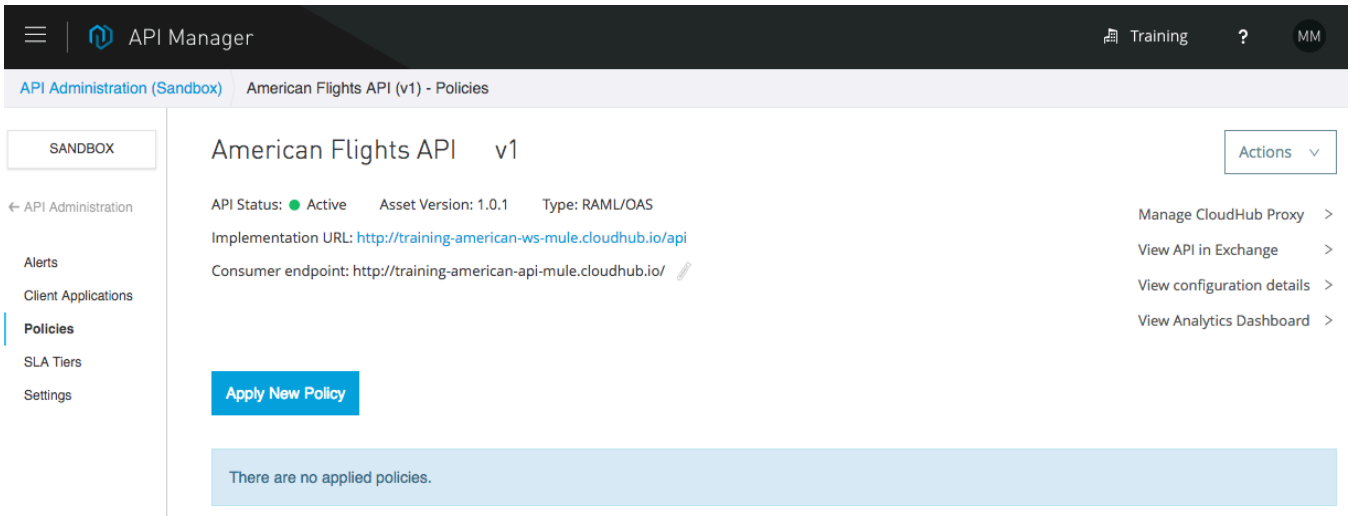
In this walkthrough, you govern access to the API proxy. You will:

- Add and test a rate limiting policy.
- Add SLA tiers, one with manual approval required.
- Add and test a rate limiting SLA policy.
- Request application access to SLA tiers from private and public API portals.
- Approve application requests to SLA tiers in API Manager.



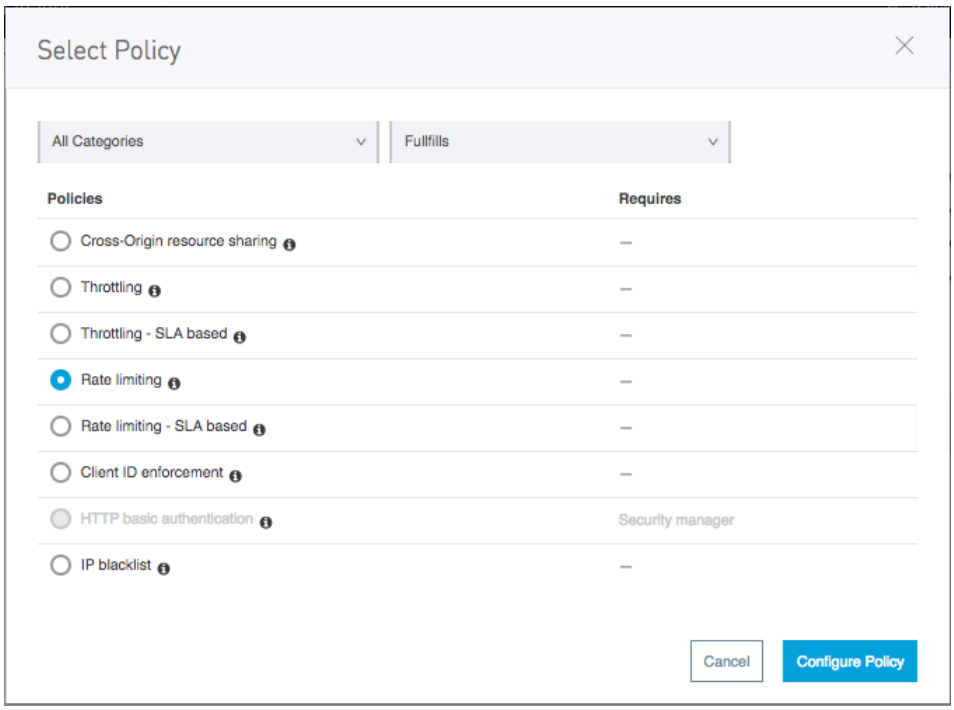
Create a rate limiting policy

1. Return to the Settings page for your American Flights API in Anypoint Manager.
2. In the left-side navigation, click the Policies link.



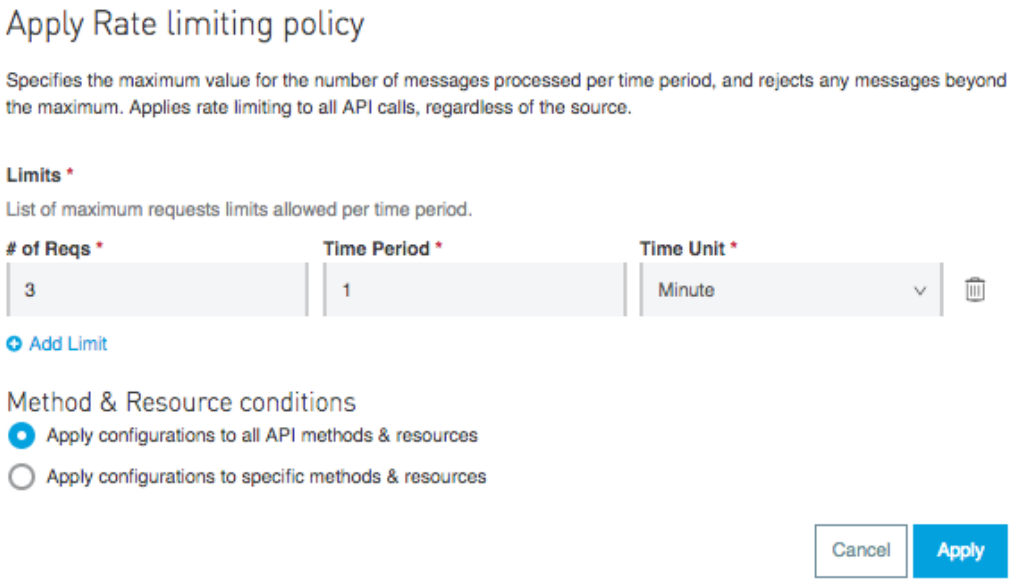
3. Click the Apply New Policy button.

4. In the Select Policy dialog box, select Rate limiting.

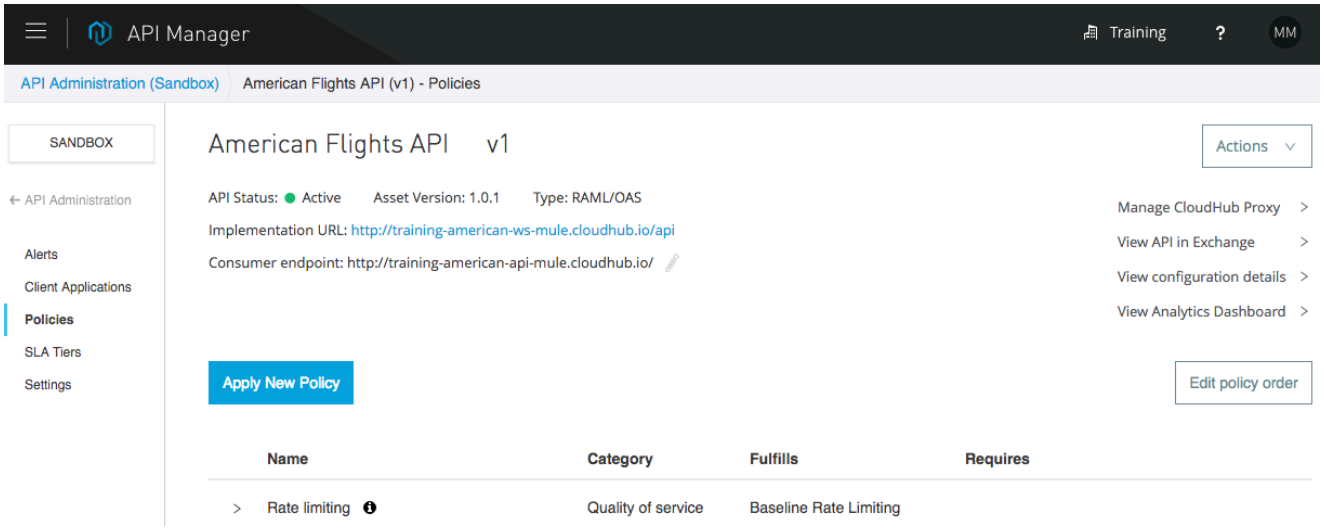


5. Click Configure Policy.
6. On the Apply Rate limiting policy page, set the following values and click Apply:

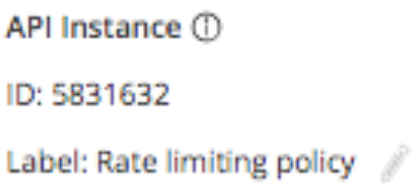
   - # of Reqs: 3
   - Time Period: 1
   - Time Unit: Minute
   - Methods & Resource conditions: Apply configurations to all API methods & resources



7. Click Apply; you should see the policy listed for your API.



8. In the left-side navigation, click Settings.
9. Change the API instance label to Rate limiting policy.

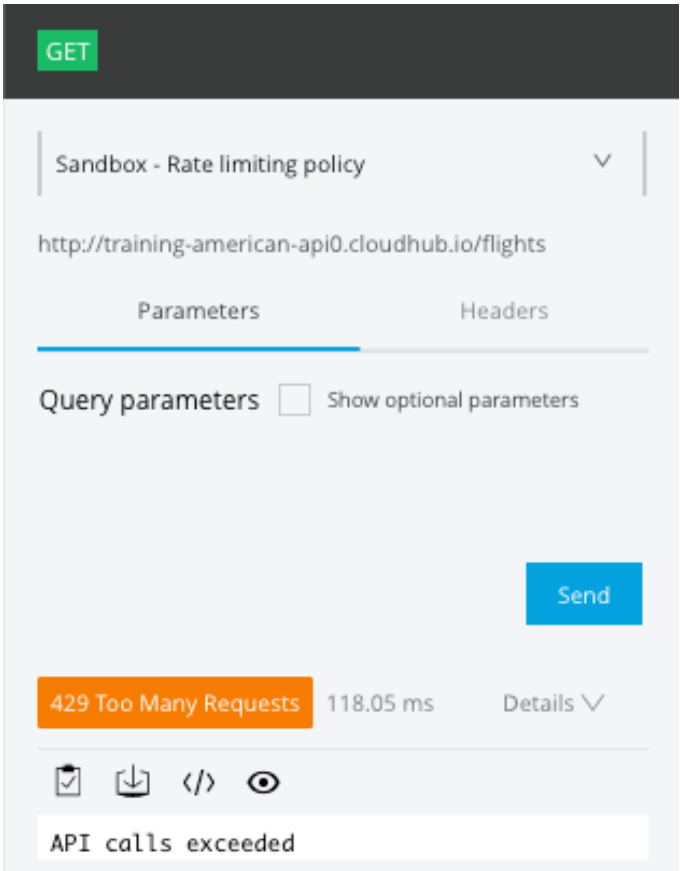


Test the new rate limiting policy

10. Return to the browser tab with your American Flights API in Exchange.
11. Return to the page with the API console for the flights:/GET resource.
12. Select the Sandbox – Rate limiting policy API instance.

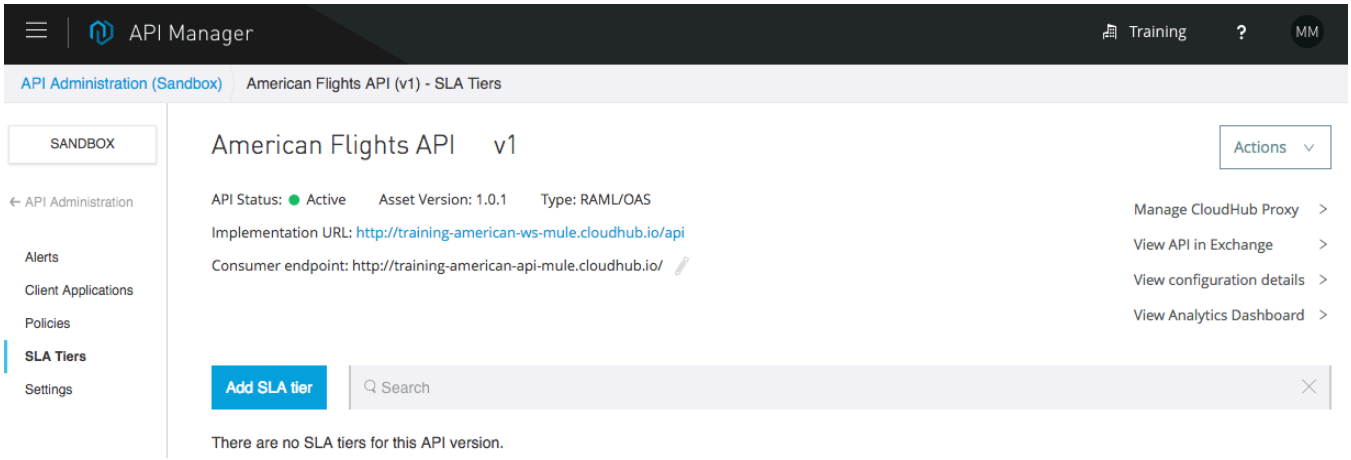
    Note: You may need to refresh the page to see the new label for the API instance.

13. Press Send until you get a 429 Too Many Requests response.



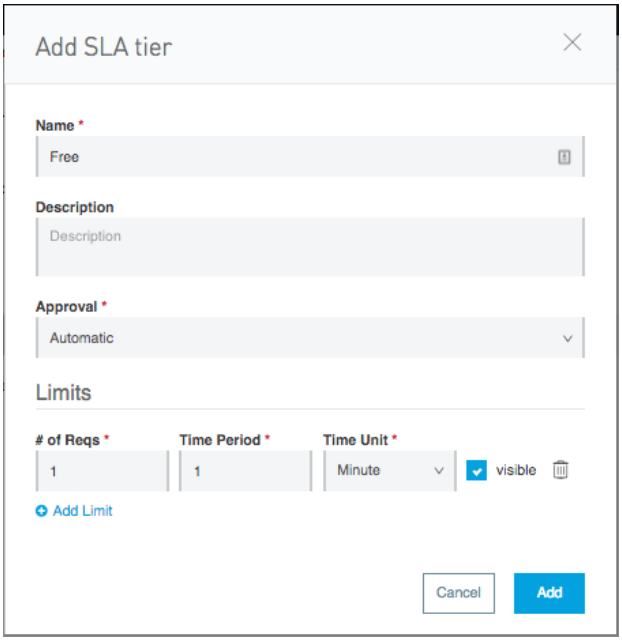
Create SLA tiers

14. Return to the browser tab with your American Flights API in API Manager.
15. In the left-side navigation, click the SLA Tiers link.
16. Click the Add SLA tier button.



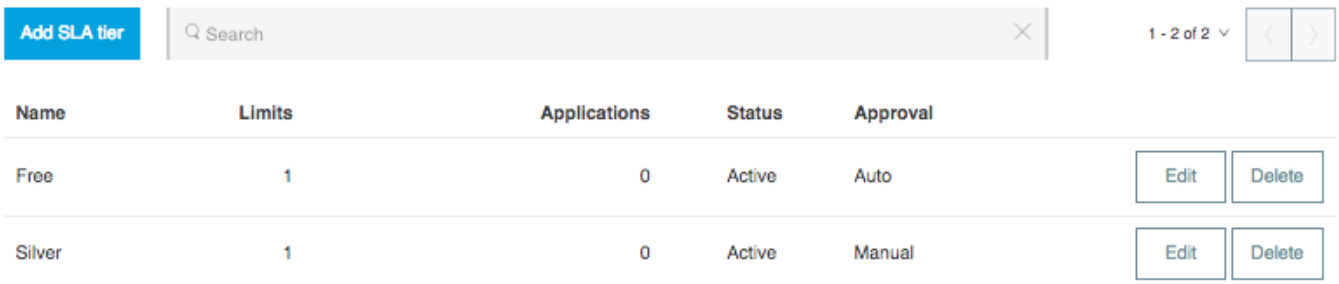
17. In the Add SLA tier dialog box, set the following values:

- Name: Free
- Approval: Automatic
- # of Reqs: 1
- Time Period: 1
- Time Unit: Minute



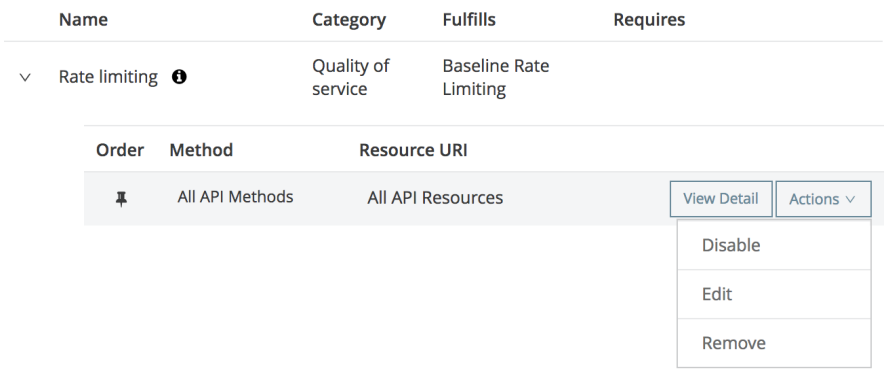
18. Click the Add button.
19. Create a second SLA tier with the following values:

- Name: Silver
- Approval: Manual
- # of Reqs: 1
- Time Period: 1
- Time Unit: Second

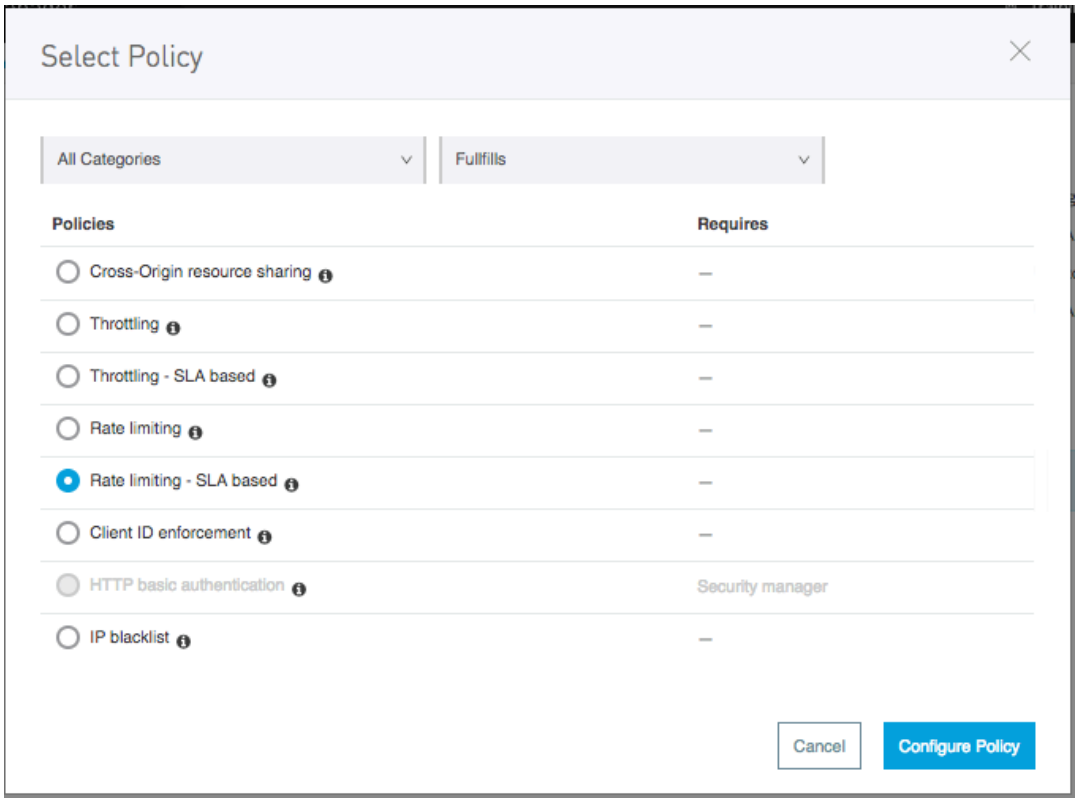


Change the policy to rate limiting – SLA based

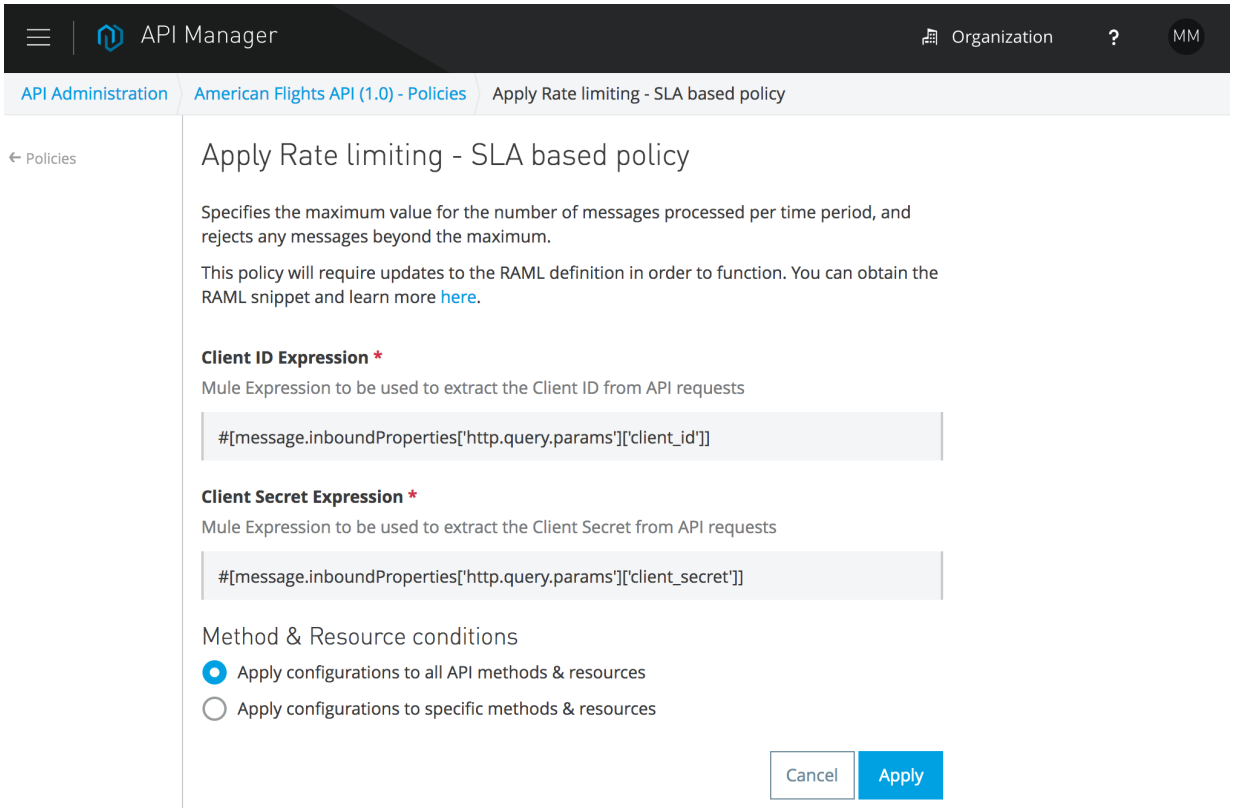
20. In the left-side navigation, click the Policies link.
21. Expand the Rate limiting policy.
22. Click the Actions button and select Remove.



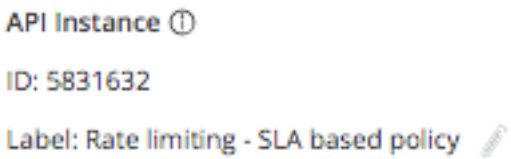
23. In the Remove policy dialog box, click Remove.
24. Click the Apply New Policy button.
25. In the Select Policy dialog box, select Rate limiting - SLA based.
26. Click Configure Policy.



27. On the Apply Rate limiting – SLA based policy page, look at the expressions and see that a client ID and secret need to be sent with API requests as query parameters.



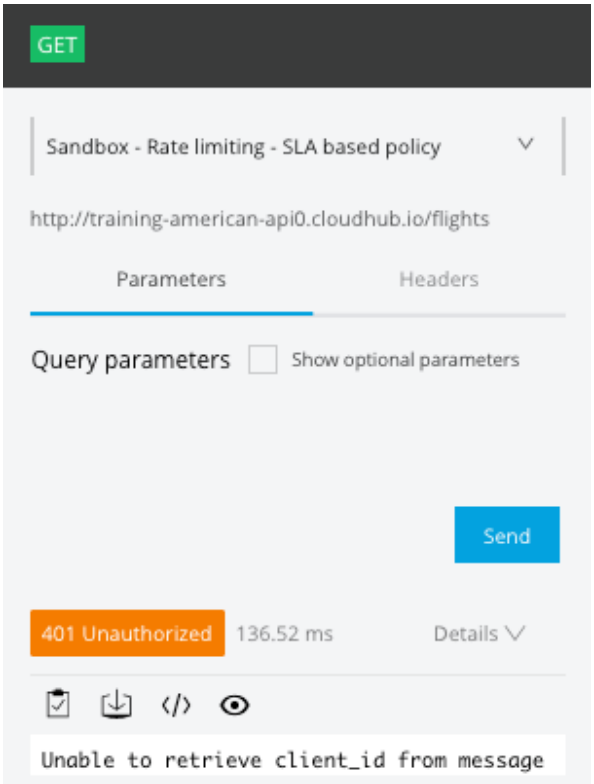
28. Click Apply.
29. In the left-side navigation, click Settings.
30. Change the API instance label to Rate limiting – SLA based policy.



Test the rate limiting – SLA based policy in Exchange

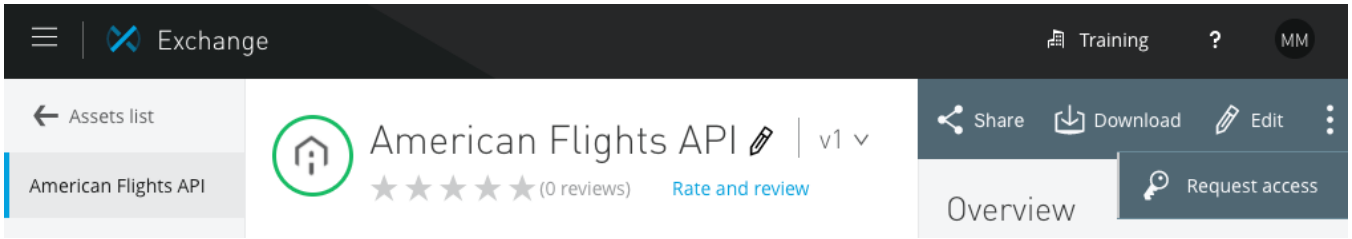
31. Return to the browser tab with your API in Exchange.
32. Refresh the page and select to make a call to the Sandbox – Rate limiting – SLA based policy.

33. Click Send; you should get a 401 response with a message that a client_id could not be retrieved from the message.

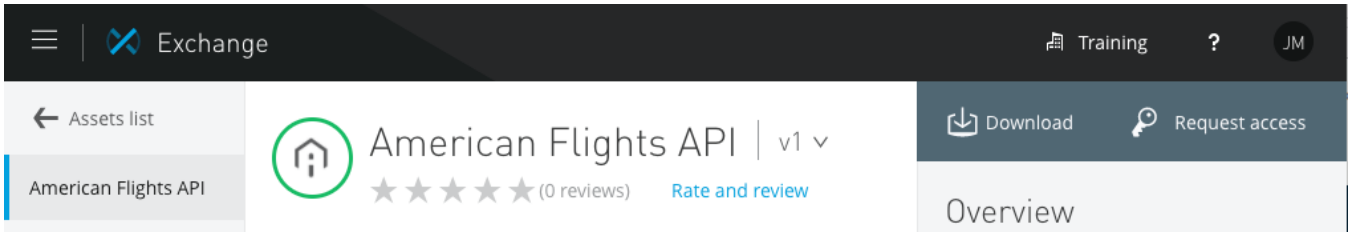


Request access to the API as an internal consumer

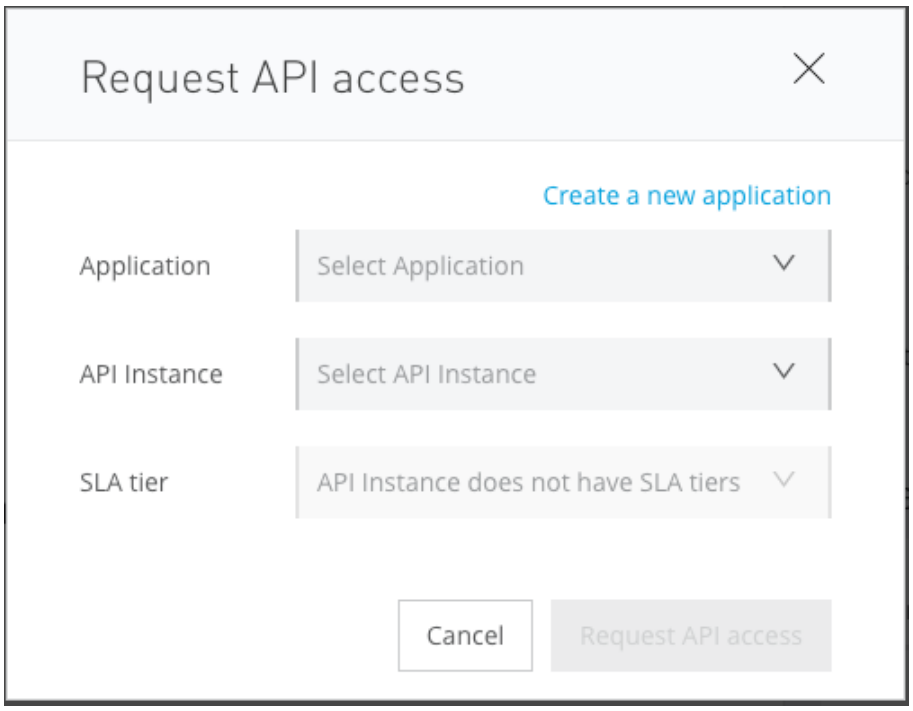
34. In the left-side navigation, click the name of the API to return to its home page.
35. Click the more options button in the upper-right corner and select Request access.



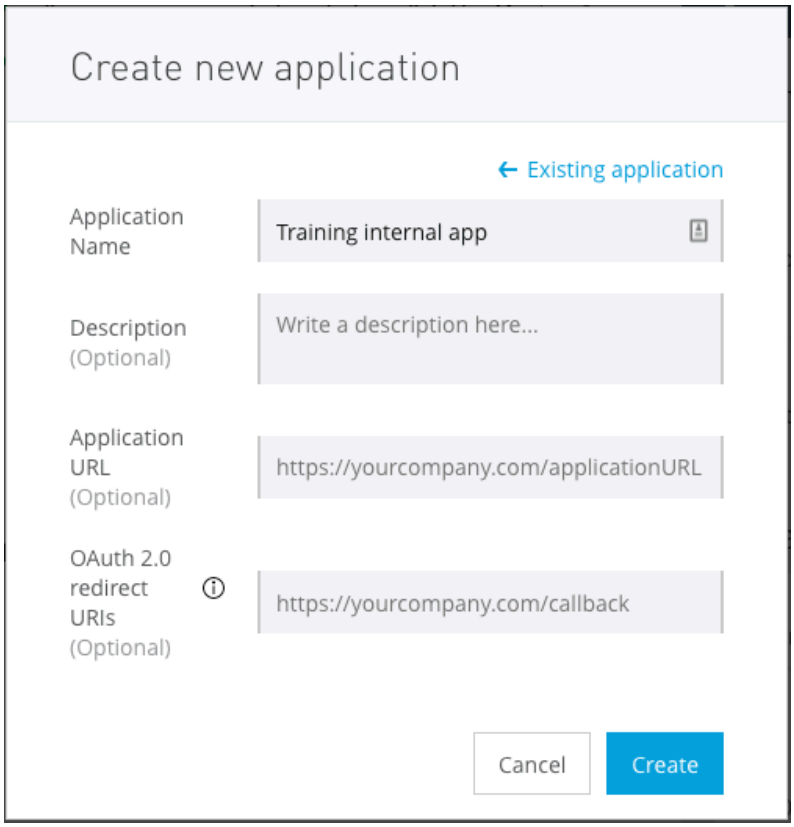
Note: Other internal users that you shared the API with that do not have Edit permissions will see a different menu.



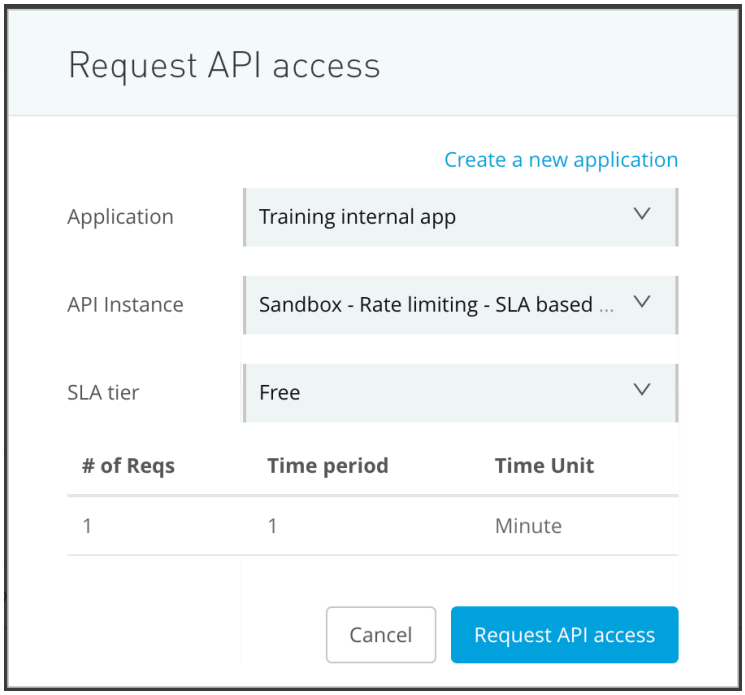
36. In the Request API access dialog box, click the Create a new application link.



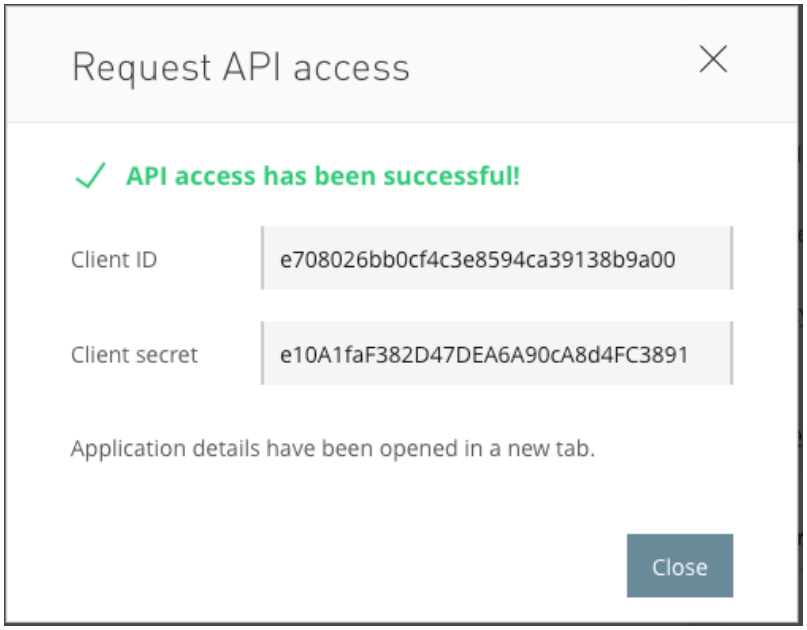
37. In the Create new application dialog box, set the name to Training internal app and click Create.



38. In the Request API access dialog box, set the API instance to Sandbox – Rate limiting SLA - based policy.
39. Set the SLA tier to Free.



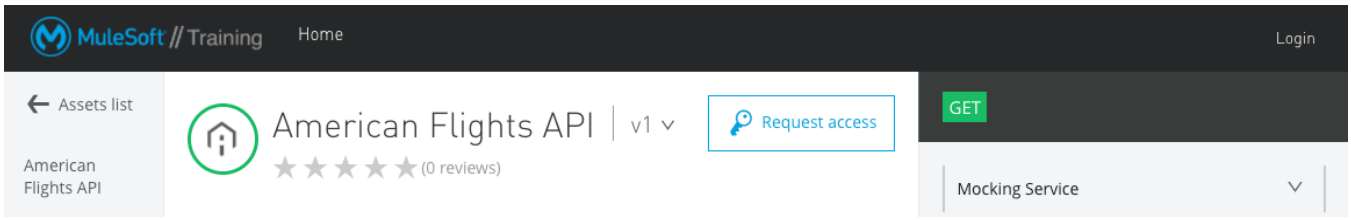
40. Click Request API access.
41. In the Request API access dialog box, view the assigned values for the client ID and client secret.



42. Click Close.

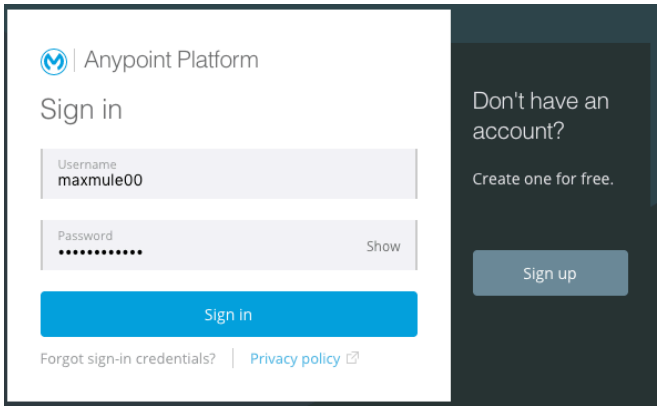
Request access to the API as an external consumer

43. Return to the public portal in the private/incognito window.
44. Refresh the page for the American Flights API; you should now see a Request access button.



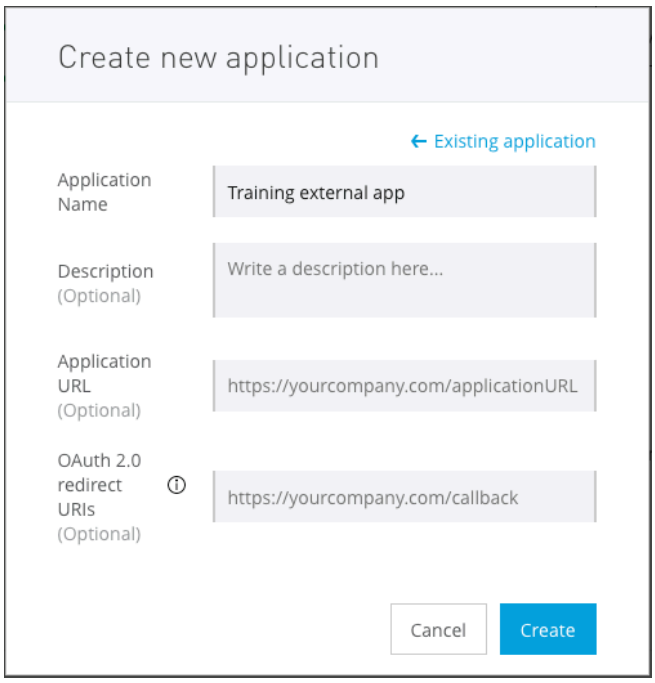
45. Click the Request access button; you should get a page to sign in or create an Anypoint Platform account.
46. Enter your existing credentials and click Sign in.

    Note: Instead of creating an external user, you will just use your existing account.

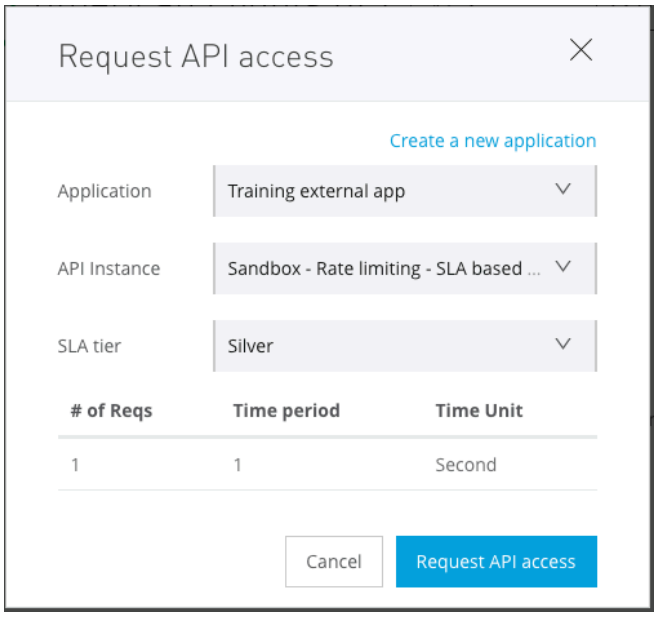


47. Back in the public portal, click the Request access button again.
48. In the Request API access dialog box, click the Create a new application button

49. In the Create new application dialog box, set the name to Training external app and click Create.

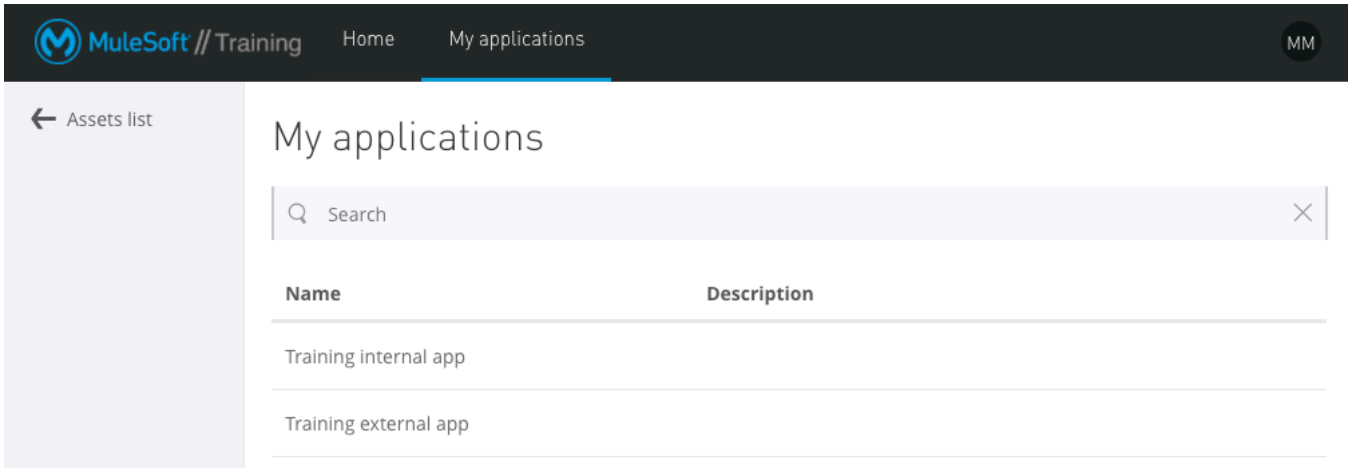


50. Click Create.
51. In the Request API access dialog box, set the API instance to Sandbox – Rate limiting SLA-based policy.
52. Set the SLA tier to Silver.

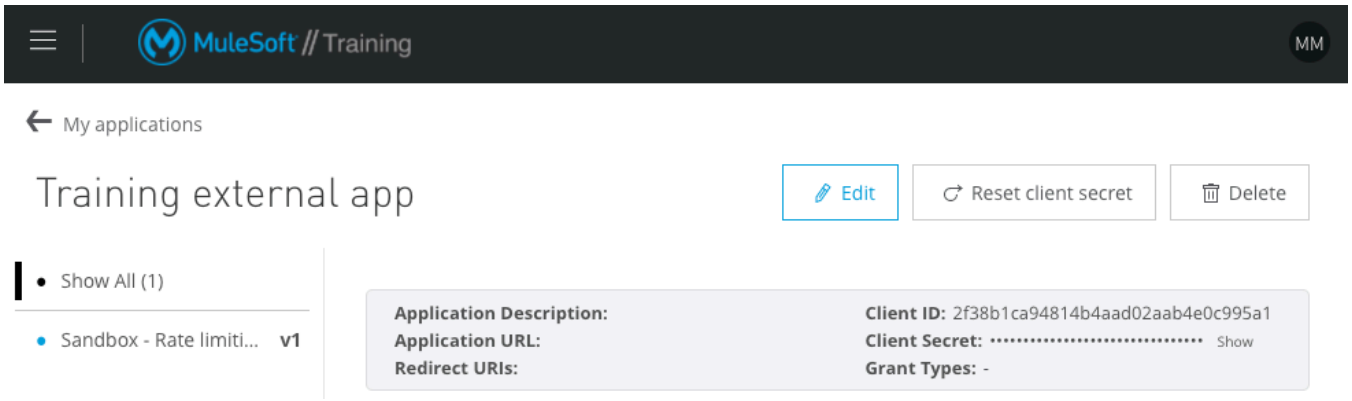


53. Click Request API access.
54. In the Request API access dialog box, click Close.

55. In the portal main menu bar, right-click My applications and select to open it in a new tab; you should see the two applications you created.



56. Click the link for Training external app; you should see what APIs the application has access to, values for the client ID and secret to access them, and request data.

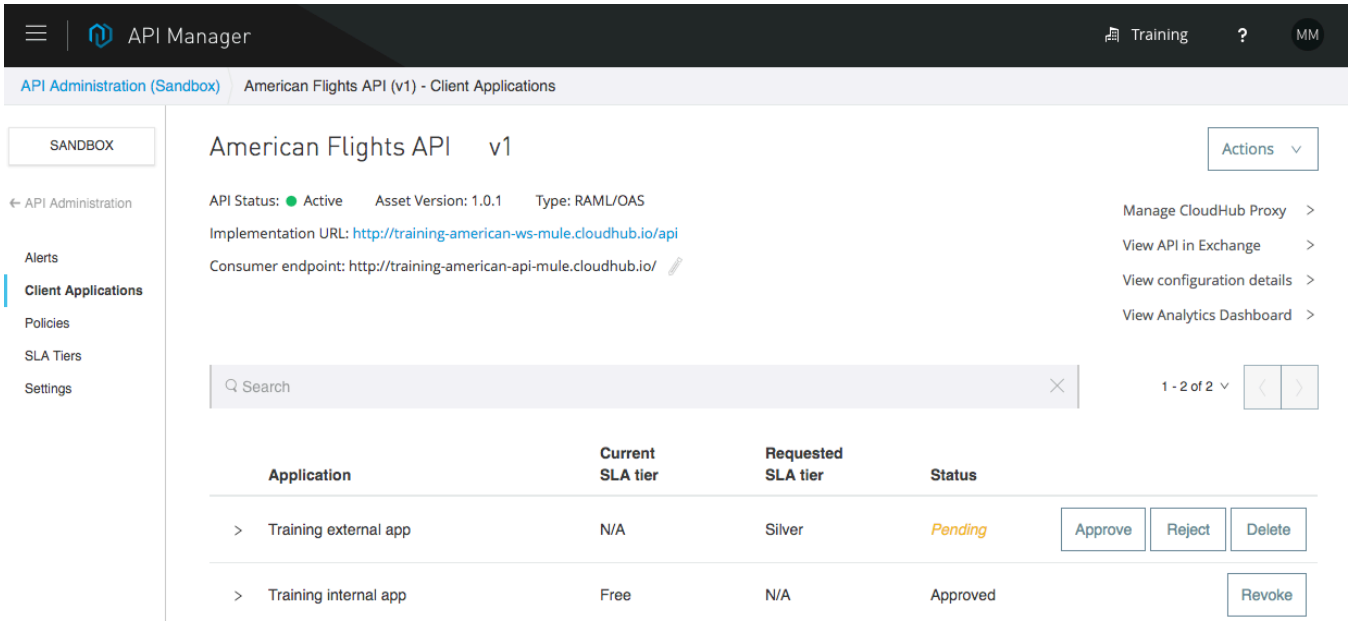


57. Leave this page open in a browser so you can return to it and copy these values.

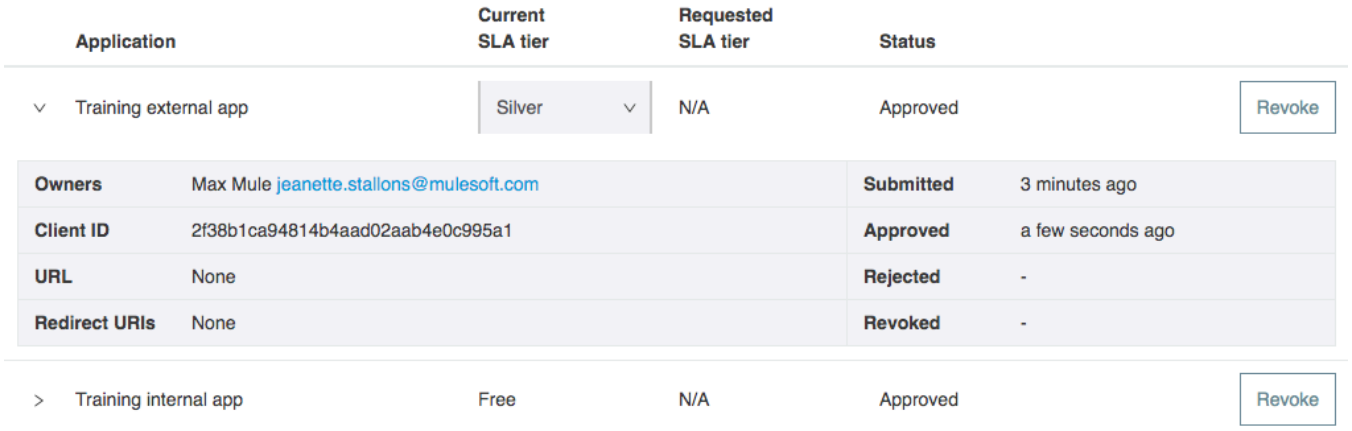
Approve the application requests

58. Return to the browser window and tab with the Settings page for American Flights API (v1) in API Manager.

59. In the left-side navigation, click Client Applications; you should see the two applications that requests access to the API.



60. Click the Approve button for the application requesting Silver tier access.
61. Expand the Training external app row and review its information.

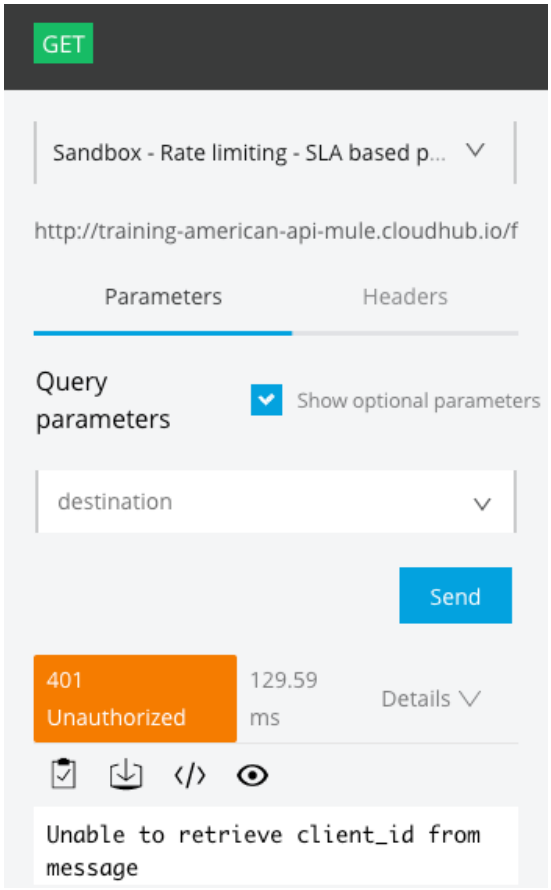


Try to test the rate limiting – SLA based policy from an API portal

62. Return to the browser window and tab with the API console in the public portal.
63. Try again to make a call to the Sandbox – Rate limiting – SLA based policy; you should still get a 401 response with a message that a client_id could not be retrieved from the message.

    Note: There is currently no way to pass the client authentication parameters needed to call this API from an API portal; you will add this in the next walkthrough. You could, however, use

another tool like Postman to make calls to the API, explicitly passing client_id and client_secret query parameters as you did in module 2.

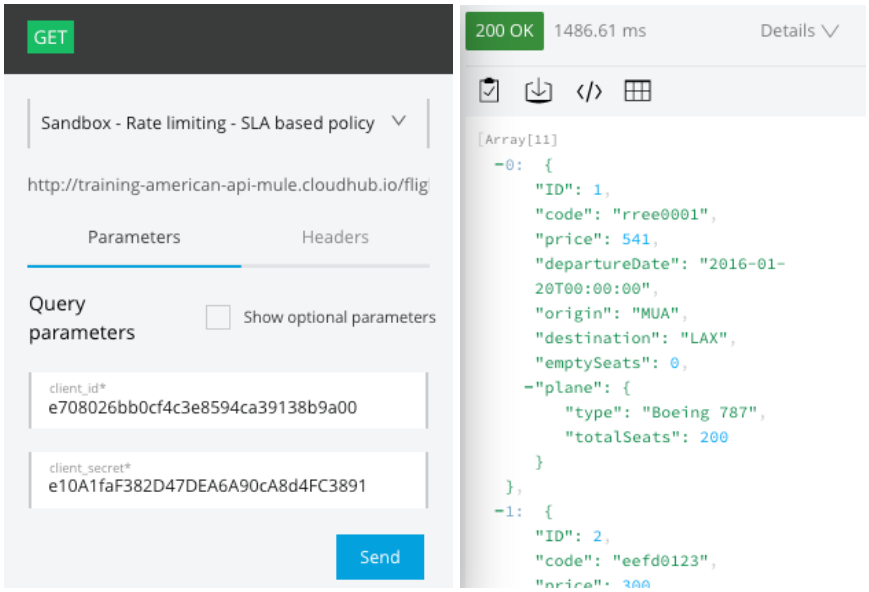


64. Close this browser window.

Walkthrough 5-5: Make calls to an API with a client ID based policy from API portals

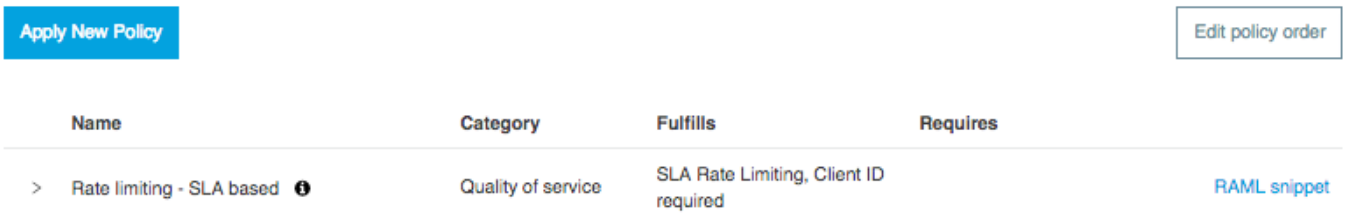
In this walkthrough, you add the ability to make calls to APIs with client ID enforcement policies form internal and public API portals. You will:

- Add authentication query parameters to an API specification.
- Update a managed API to use a new version of an API specification.
- Call a governed API with client credentials from API portals.



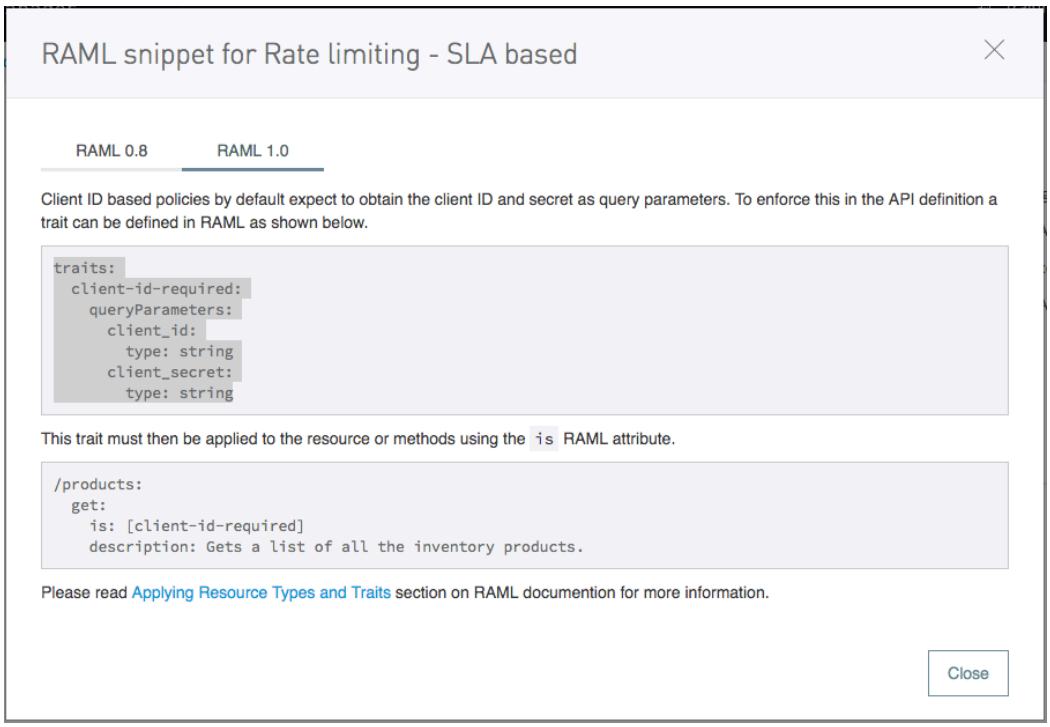
Add authentication query parameters to the API specification

1. Return to the browser tab with the Settings page for American Flights API (v1) in API Manager.
2. In the left-side navigation, click the Policies link.
3. Click the RAML snippet link for the rate limiting – SLA based policy.



4. In the RAML snippet for Rate limiting – SLA based dialog box, select RAML 1.0.

5. Copy the value for the traits.



6. Click Close.
7. Return to the browser tab with your API in Design Center.
8. Go to a new line after the types declaration and paste the traits code you copied.

```
1   #%RAML 1.0
2   version: v1
3   title: American Flights API
4
5   types:
6     AmericanFlight: !include exchange_modu
7
8   traits:
9     client-id-required:
10      queryParameters:
11        client_id:
12          type: string
13        client_secret:
14          type: string
15
16  /flights:
```

9. Go to a new line after the flights resource declaration and add is: [].

```
16  /flights:
17    is: []
18    get:
```

10. Place the cursor inside the array brackets and in the shelf, click client-id-required.

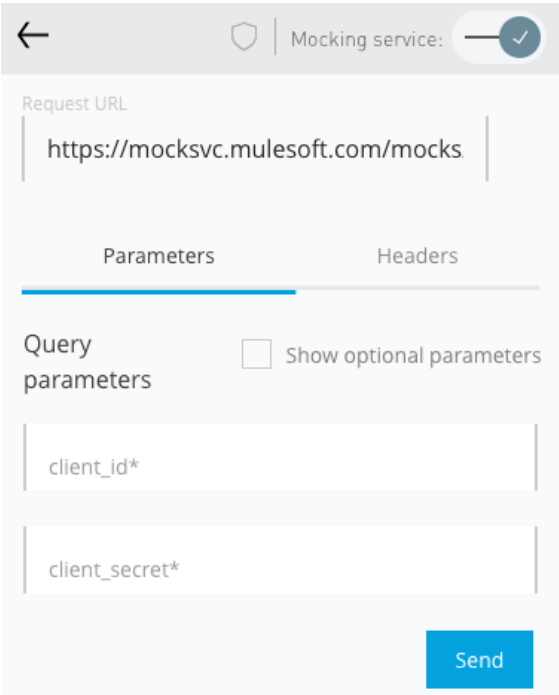
```
16  /flights:
17    is: [client-id-required]
18    get:
```

11. Repeat this process so the trait is applied to all methods of the {ID} resource as well.

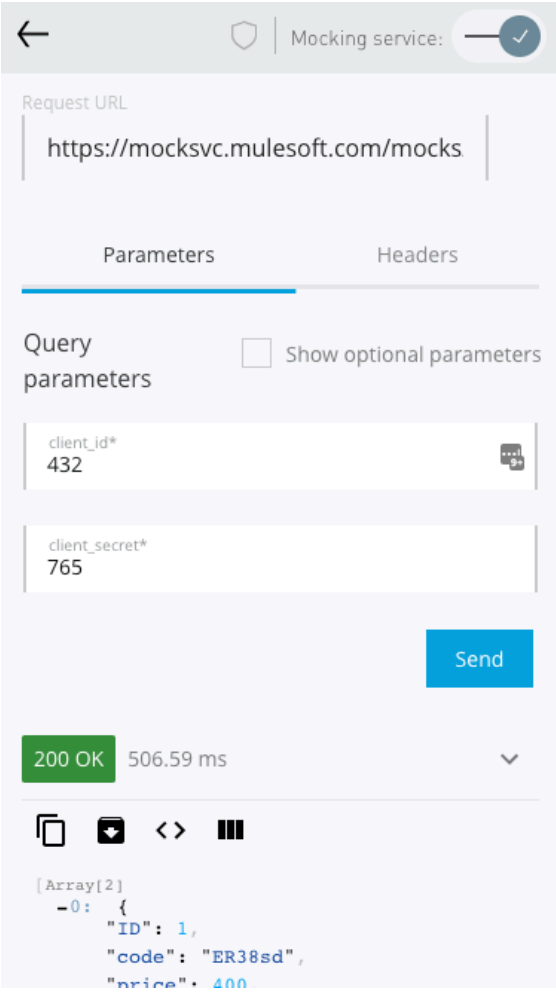
```
45  /{ID}:
46    is: [client-id-required]
47    get:
```

Test the authentication query parameters in API console

12. In the API console, turn on the mocking service.
13. Select to try any one of the resources; you should now see text fields to enter client_id and client_secret query parameters.

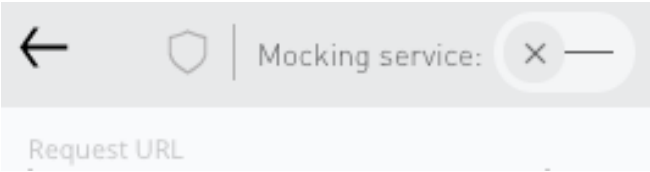


14. Enter any values for the client_id and client_secret and click Send; you should get a 200 response with the example results.



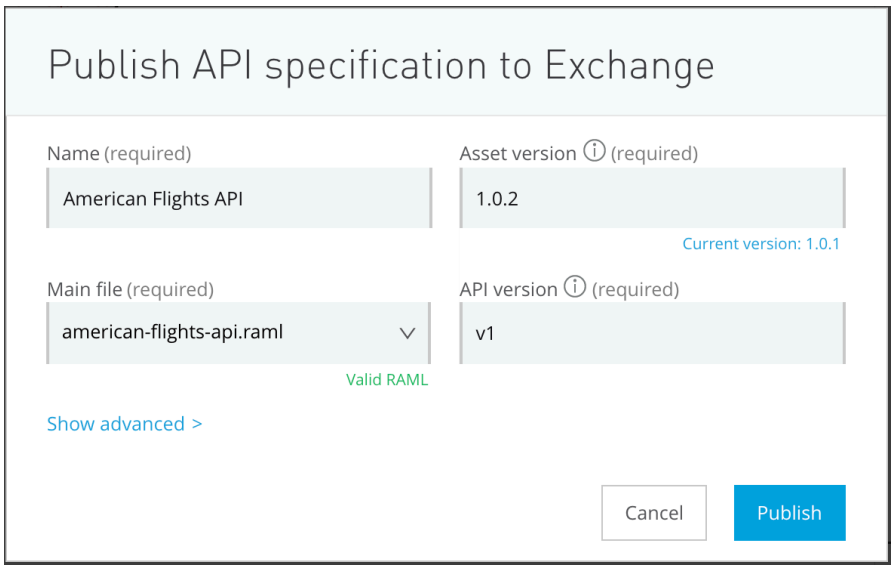
Publish the new version of the API to Exchange

15. Turn off the mocking service.



16. Click the Publish to Exchange button.

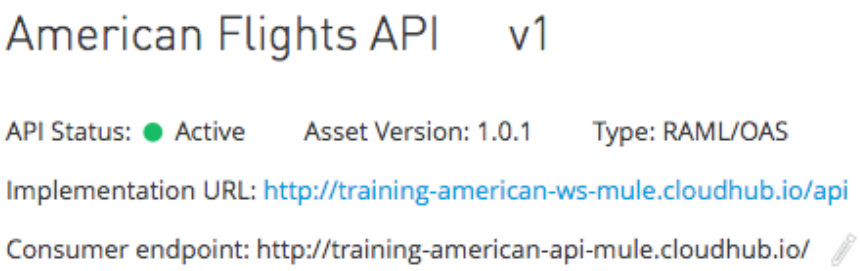
17. In the Publish API specification to Exchange dialog box, note the asset version and click Publish.



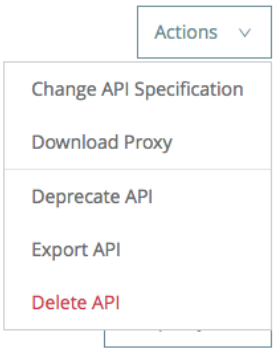
18. After the API is published, click Done in the Publish API specification to Exchange dialog box.

Update the managed API instance to use the new version of the API specification

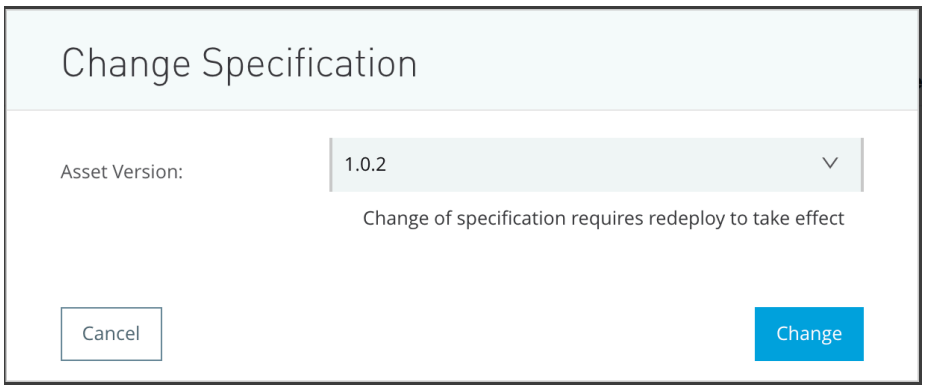
19. Return to browser tab with American Flights API (v1) in API Manager.
20. Locate the asset version displayed at the top of the page; you should see 1.0.1.



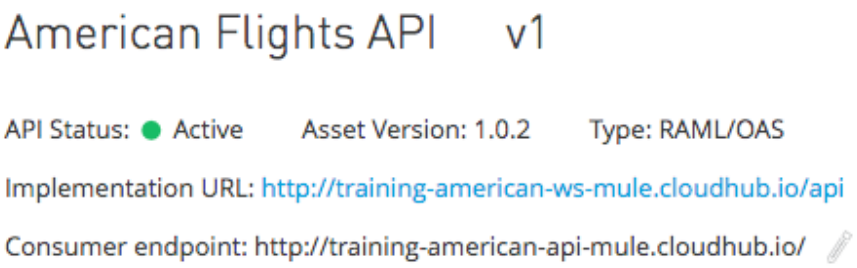
21. Click the Actions button in the upper-right corner and select Change API Specification.



22. In the Change Specification dialog box, select the latest asset version, 1.0.2.
23. Click Change.

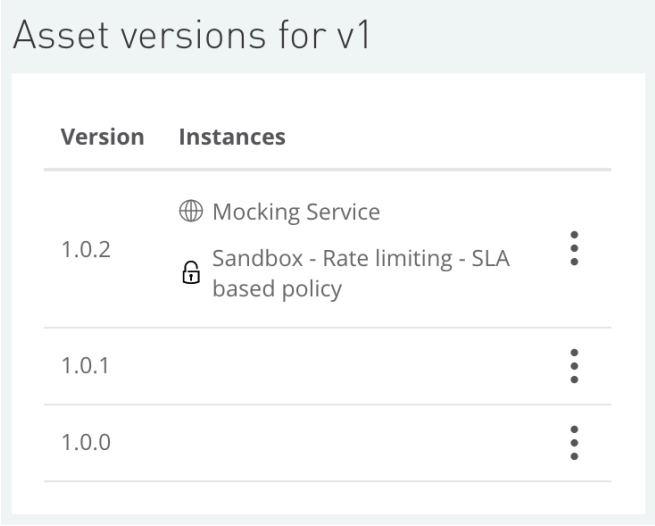


24. Wait until the API Status is green again indicating that the new proxy application has been redeployed.

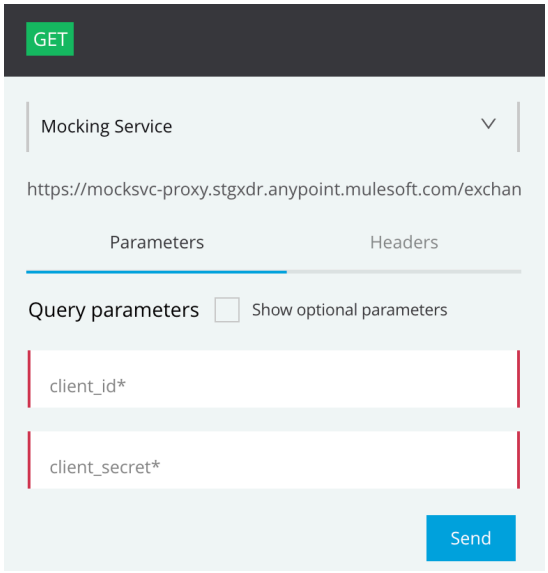


Test the rate limiting – SLA based policy in an API portal with random client authentication values

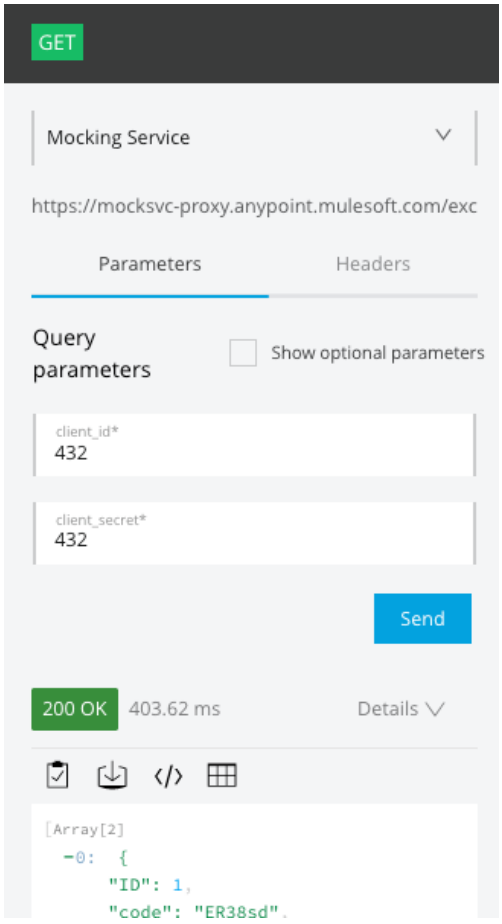
25. Return to the browser tab with Exchange.
26. Return to the home page for the API; you should see the new asset version listed.



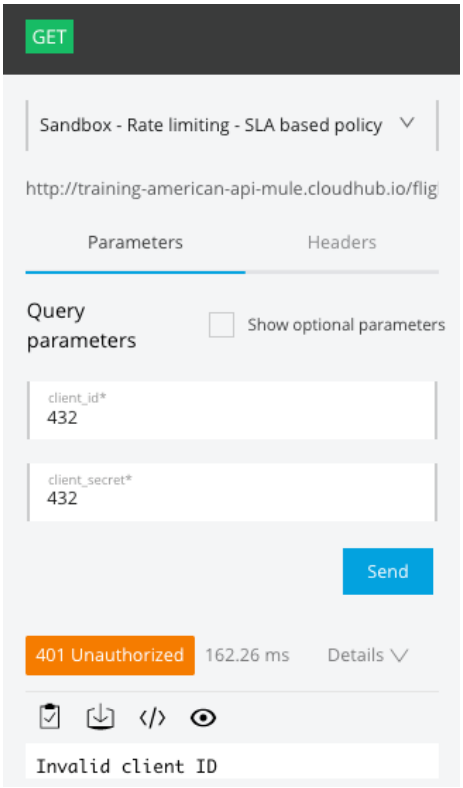
27. Click the GET method for the flights resource; you should now see required text boxes to enter the required client_id and client_secret values.



28. With the Mocking Service endpoint selected, enter any values for the client_id and client_secret.
29. Click Send; you should get results.

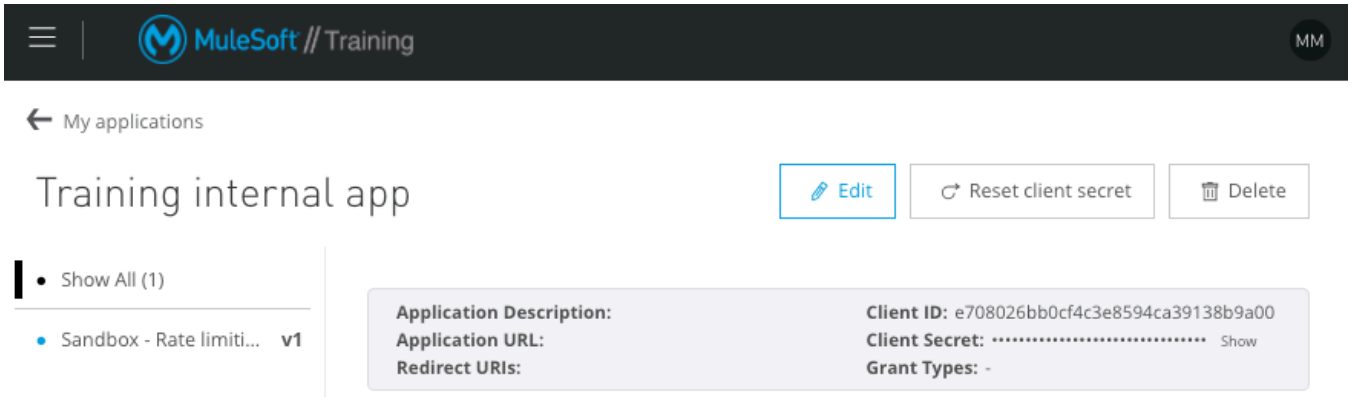


30. Change the API instance to Sandbox – Rate limiting – SLA based policy and leave the random values for the authentication parameters.
31. Click Send; you should now get a 410 response with a message that the client ID is invalid.

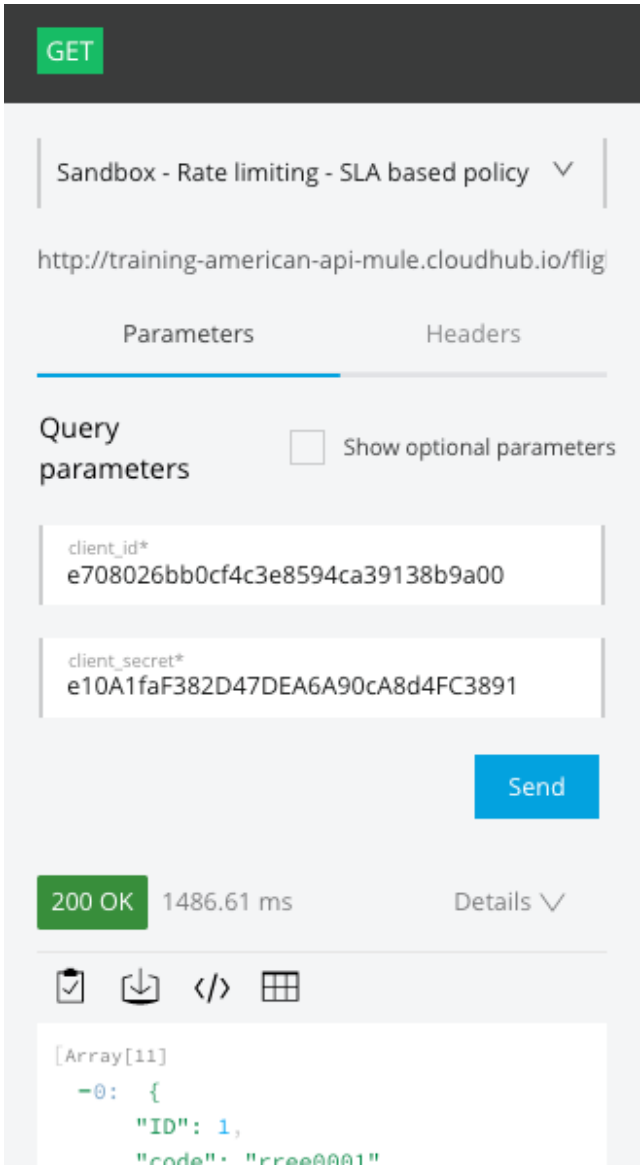


Test the rate limiting – SLA based policy in an API portal with assigned client authentication values

32. In the left-side navigation, click Assets list.
33. In the left-side navigation, click Public Portal.
34. In the main menu, select My applications.
35. Click the Training internal application.



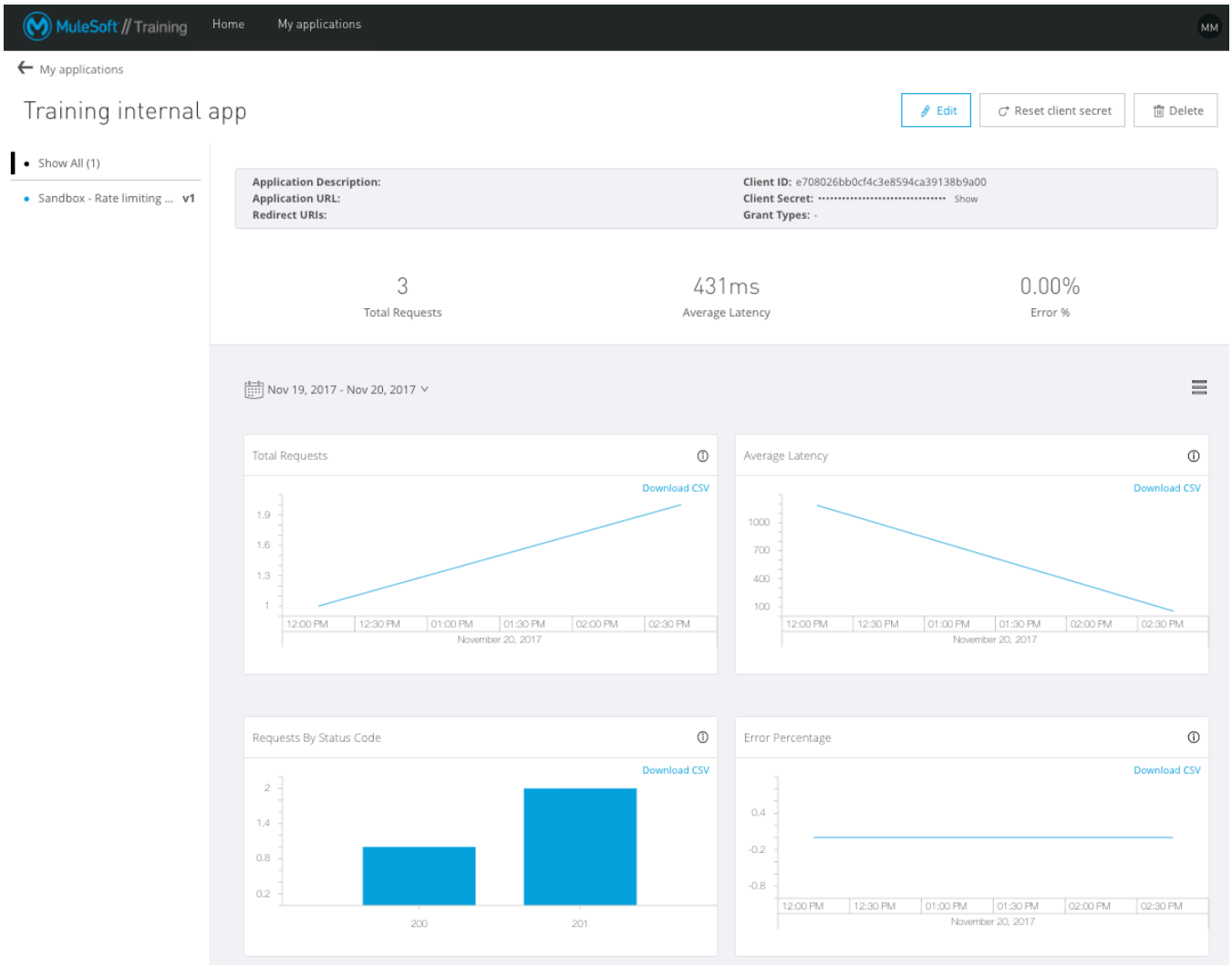
36. Copy the value of its client ID.
37. Right-click Home in the main menu bar and select to open the link in a new browser tab.
38. In that new tab, click the American Flights API.
39. Click the GET method for the flights resource.
40. In the API console, change the API instance to Sandbox – rate limiting – SLA based policy.
41. In the client_id field, paste the value of the client ID.
42. Return to the other browser tab and copy the value of the client secret.
43. Return to the other browser tab and paste the value of the client secret.
44. Click Send; you should now get a 200 response with live results.



45. Make multiple calls to this method.

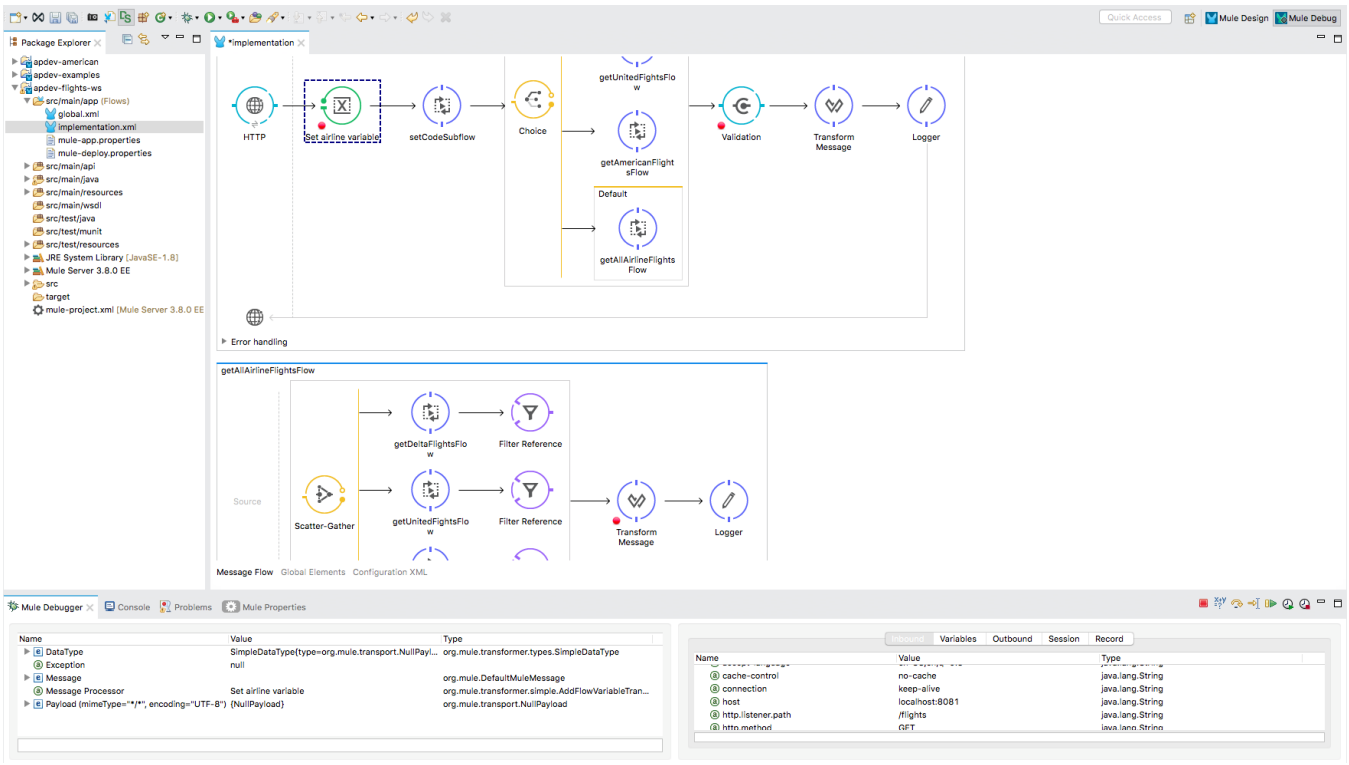
Review request data

46. Return to the other browser tab with the application information.
47. Change the date selection to Last Day; you should see data about your requests.



48. Close all the browser tabs with Anypoint Platform.

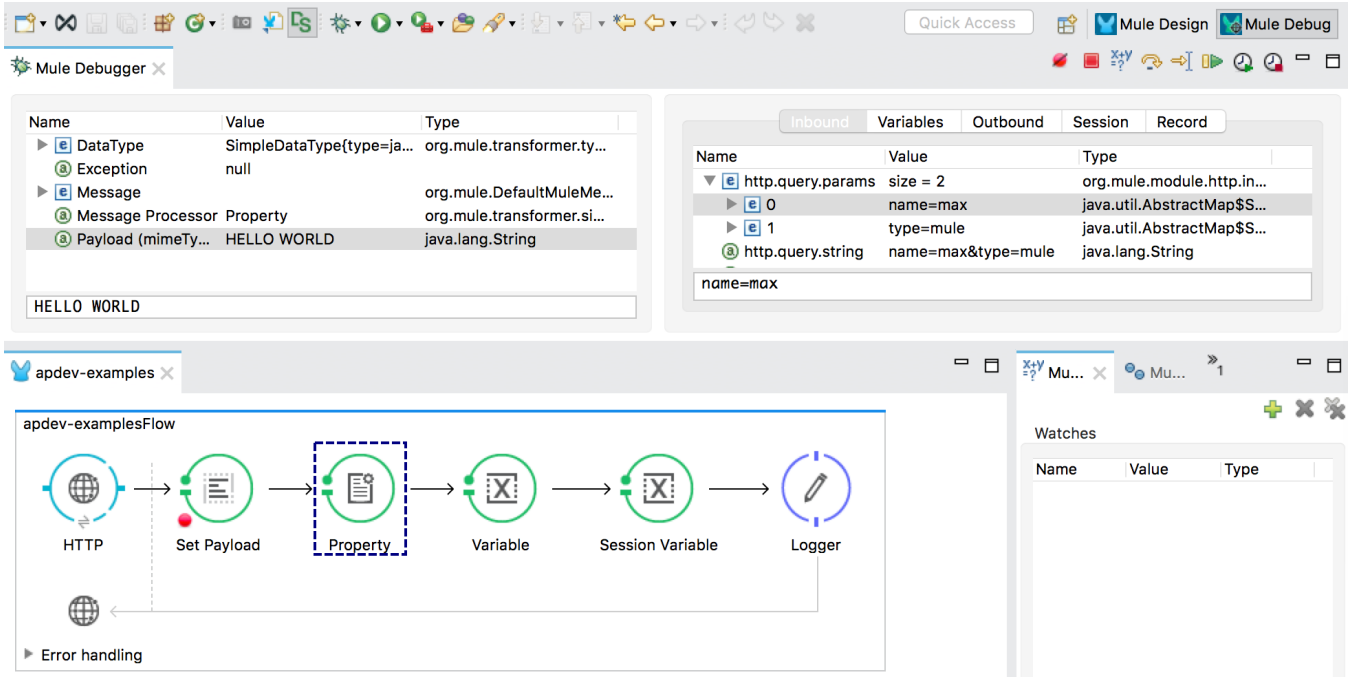
PART 2: Building Applications with Anypoint Studio



At the end of this part, you should be able to:

- Debug Mule applications.
- Read and write message payloads, properties, and variables using the Mule Expression Language.
- Structure Mule applications using flows, subflows, in-memory message queues, properties files, and configuration files.
- Connect to web services, SaaS applications, files, polled resources, JMS queues, and more.
- Route, filter, and validate messages and handle message exceptions.
- Write DataWeave expressions for more complicated transformations.
- Process individual records in a collection and synchronize data in databases to SaaS applications.

Module 6: Accessing and Modifying Mule Messages



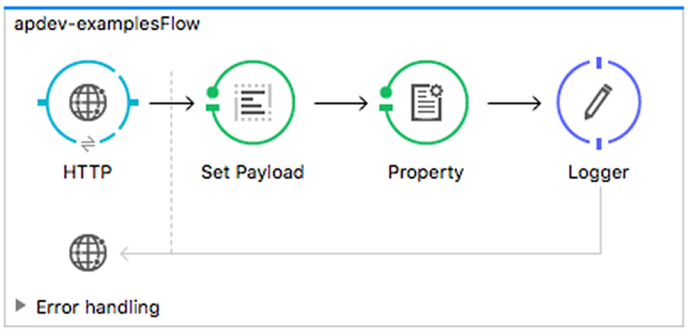
At the end this module, you should be able to:

- Log message data.
- Debug Mule applications.
- Read and write message properties.
- Write expressions with Mule Expression Language (MEL).
- Create variables.

Walkthrough 6-1: Set and log message data

In this walkthrough, you create a new project to use in the next two modules for learning about Mule messages and Mule applications. You will:

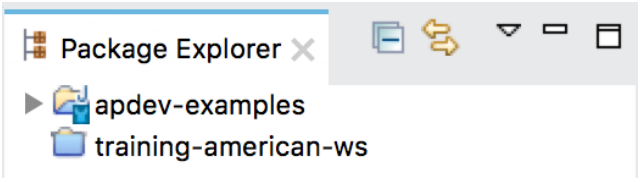
- Create a new Mule project with an HTTP Listener endpoint.
- Set the message payload and an outbound property.
- Use a Logger to view message data in the Anypoint Studio console.
- Review message data in Postman.




```
  http.request.path=/hello
  http.request.uri=/hello
  http.scheme=http
  http.uri.params=ParameterMap{[]}
  http.version=HTTP/1.1
  postman-token=675b8e19-012d-f66e-f796-aa3f
  user-agent=Mozilla/5.0 (Macintosh; Intel M
OUTBOUND scoped properties:
  qpname=max
SESSION scoped properties:
}
```

Create a new Mule project

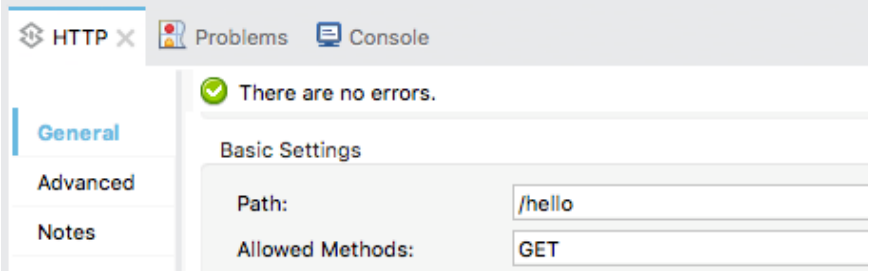
1. Return to Anypoint Studio.
2. Right-click training-american-ws and select Close Project.
3. Select File > New > Mule Project.
4. Set the Project Name to apdev-examples and click Finish.



Create an HTTP connector endpoint to receive requests

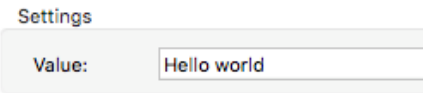
5. Drag an HTTP connector from the Mule Palette to the canvas.
6. In the HTTP properties view, click the Add button next to connector configuration.
7. In the Global Element Properties dialog box, look at the default values and click OK.
8. In the HTTP properties view, set the path to /hello.

9. Set the allowed methods to GET.



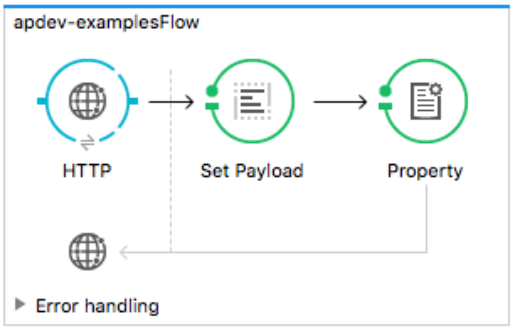
Set the message payload

10. Drag a Set Payload transformer from the Mule Palette into the process section of the flow.
11. In the Set Payload properties view, set the value field to Hello world.

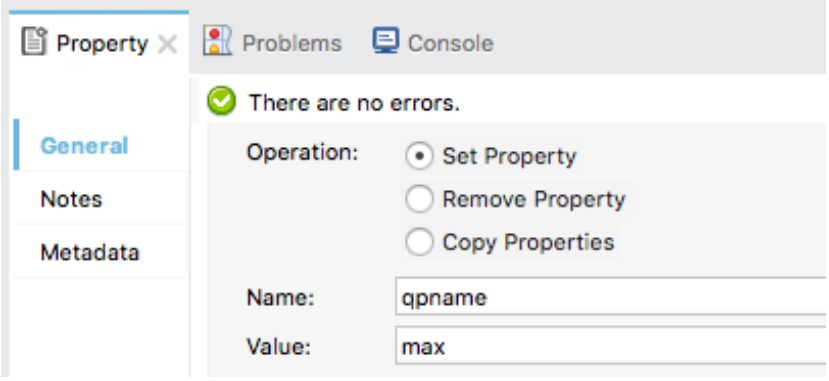


Set an outbound message property

12. Drag a Property transformer from the Mule Palette into the flow.

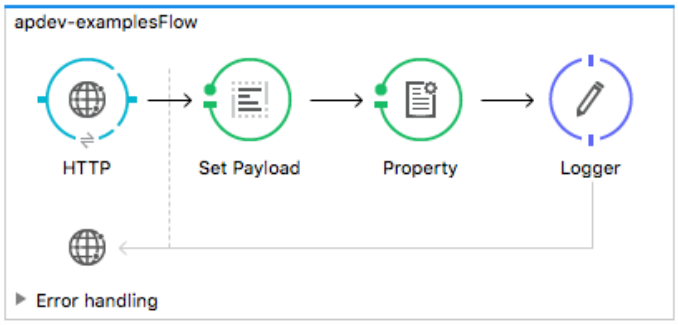


13. In the Properties view for the Property transformer, select Set Property.
14. Set the name to qpname and the value to max.



Add a Logger

15. Drag a Logger component from the Mule Palette and drop it at the end of the flow.

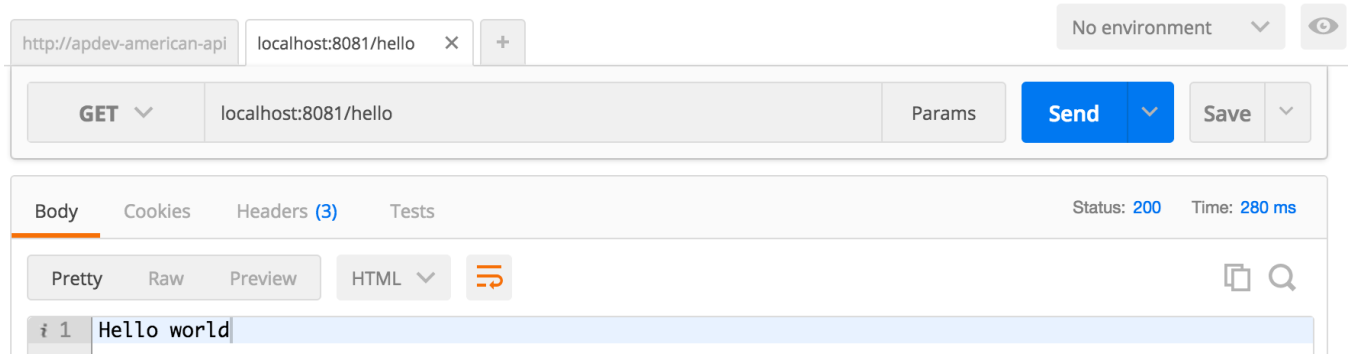


Run the application and review message data

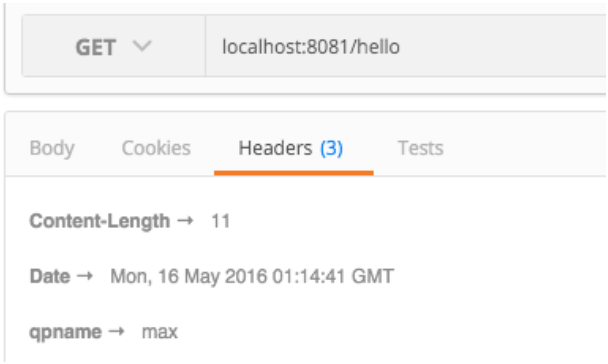
16. Run the project.
17. Return to Postman and click the button to create a new tab.

    Note: You are adding a new tab so that you can keep the request to your American API saved in another tab for later use. Optionally, you could also save it to a Postman collection.

18. In the new tab, make a GET request to http://localhost:8081/hello; you should see Hello world displayed.

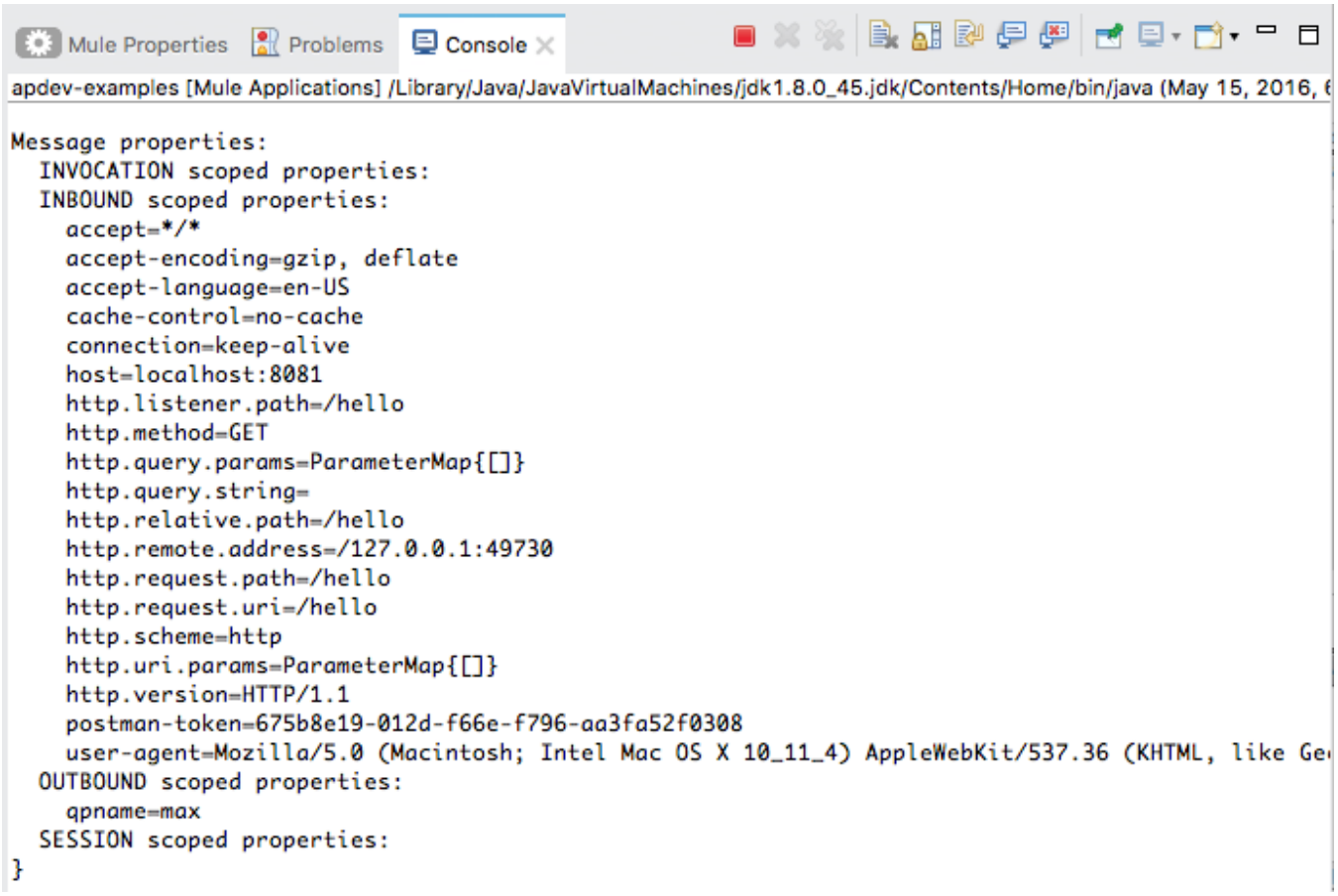


19. Click the Headers link in the response; you should see the qpname property.



View message data in the Anypoint Studio console

20. Return to Anypoint Studio and look at the console.
21. Locate the data displayed by using the Logger.
22. Find where the data type of the payload is specified.
23. Review the message inbound properties.
24. Review the message outbound properties.
25. Locate the other two scopes of properties that have no values set.

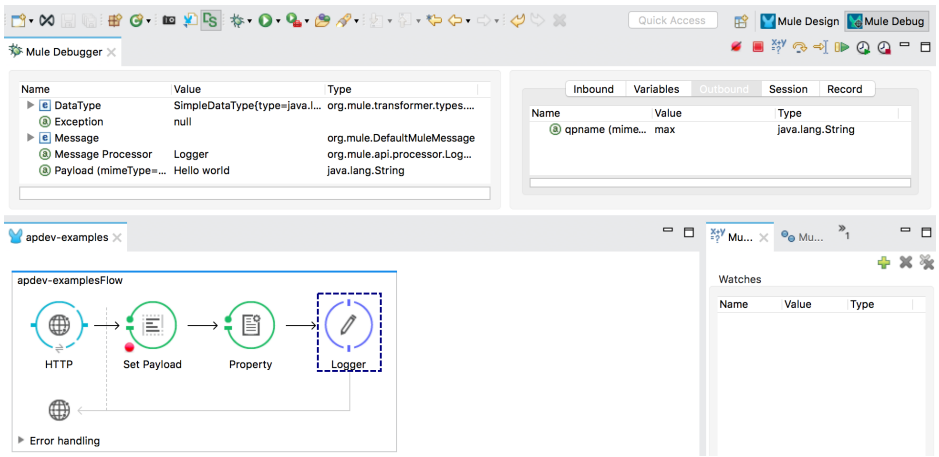


26. Stop the project.

Walkthrough 6-2: Debug a Mule application

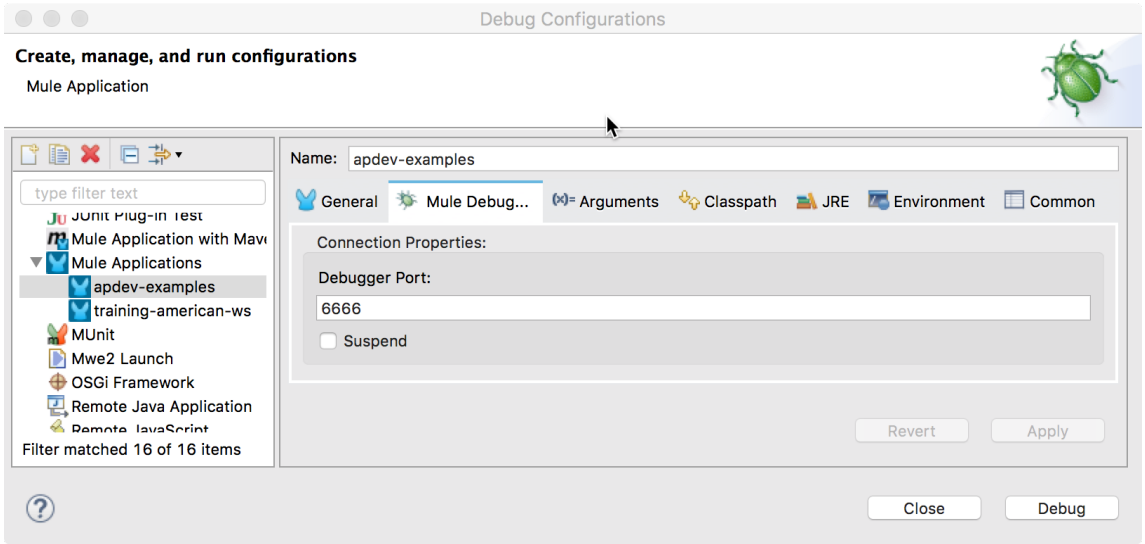
In this walkthrough, you debug and step through the code in a Mule application. You will:

- Locate the port used by the Mule Debugger.
- Add a breakpoint, debug an application, and step through the code.
- Use the Mule Debugger to view message properties.
- Pass query parameters to a request and locate them in the Mule Debugger.



Locate the port used by the Mule Debugger

1. Return to the apdev-examples project in Anypoint Studio.
2. In the main menu bar, select Run > Debug Configurations.
3. Click apdev-examples in the left menu bar under Mule Applications; you should see that the apdev-examples project is selected to launch.
4. Click the Mule Debug tab; you should see the debug port is set to 6666 for the project.

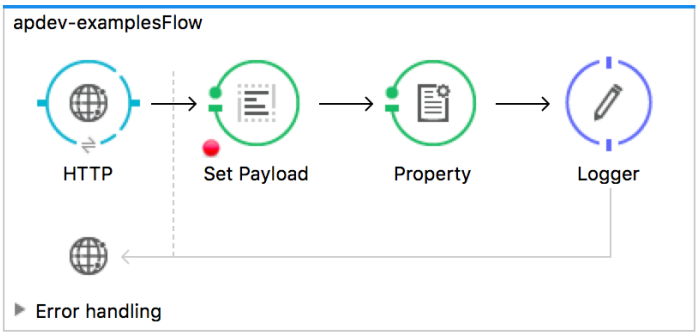


Note: If you know you have another program using port 6666 like McAfee on Windows, change this to a different value. Otherwise, you can test the debugger first and come back and change the value here later if there is a conflict.

5. Click Close.

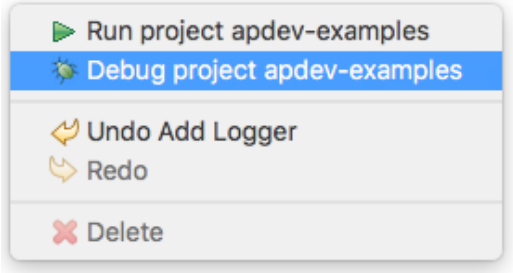
Debug the application

6. Right-click the Set Payload component and select Toggle breakpoint.

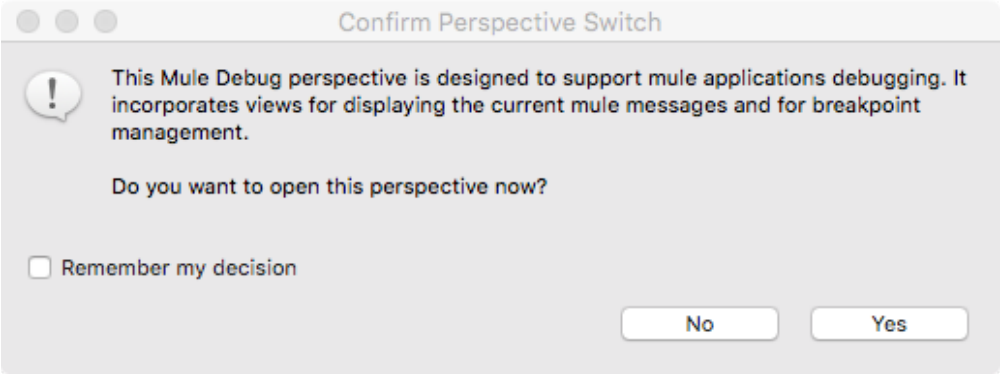


7. Right-click in the canvas and select Debug project apdev-examples.

   Note: You can also select Run > Debug or click the Debug button in the main menu bar.



8. If you get an Accept incoming network connections dialog box, click Allow.
9. If you get a Confirm Perspective Switch dialog box, select Remember my decision and click Yes.



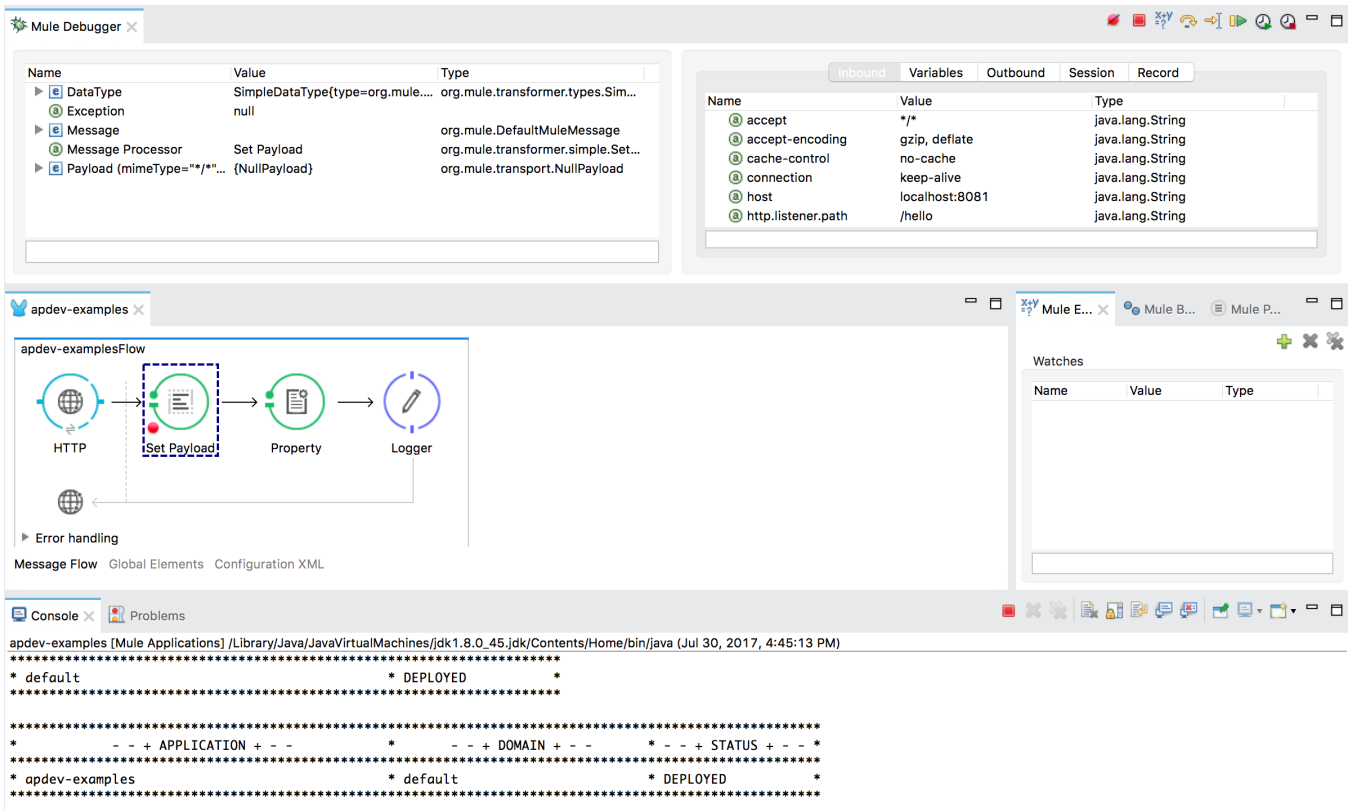
10. In the Debug perspective, close the MUnit view.

    Note: You will not use MUnit in this course. MUnit is covered in the Anypoint Platform Development: Advanced course.

11. Look at the console and wait until the application starts.
12. In Postman, make another request to http://localhost:8081/hello.

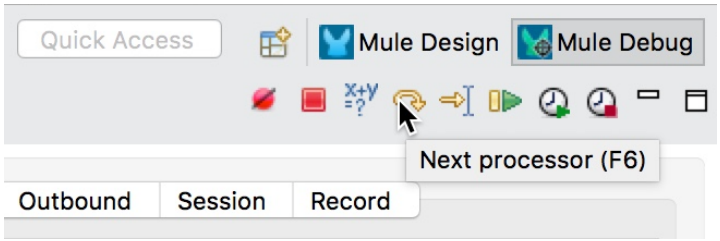
View message properties and variables in the Mule Debugger

13. Return to Anypoint Studio and locate the Mule Debugger view.
14. Drill-down and explore the variables on the left side of the Mule Debugger.
15. Look at the values of the inbound properties on the right side of the Mule Debugger.
16. Select each of the other tabs: Variables, Outbound, Session, Record; you should not see values for any other properties or variables.

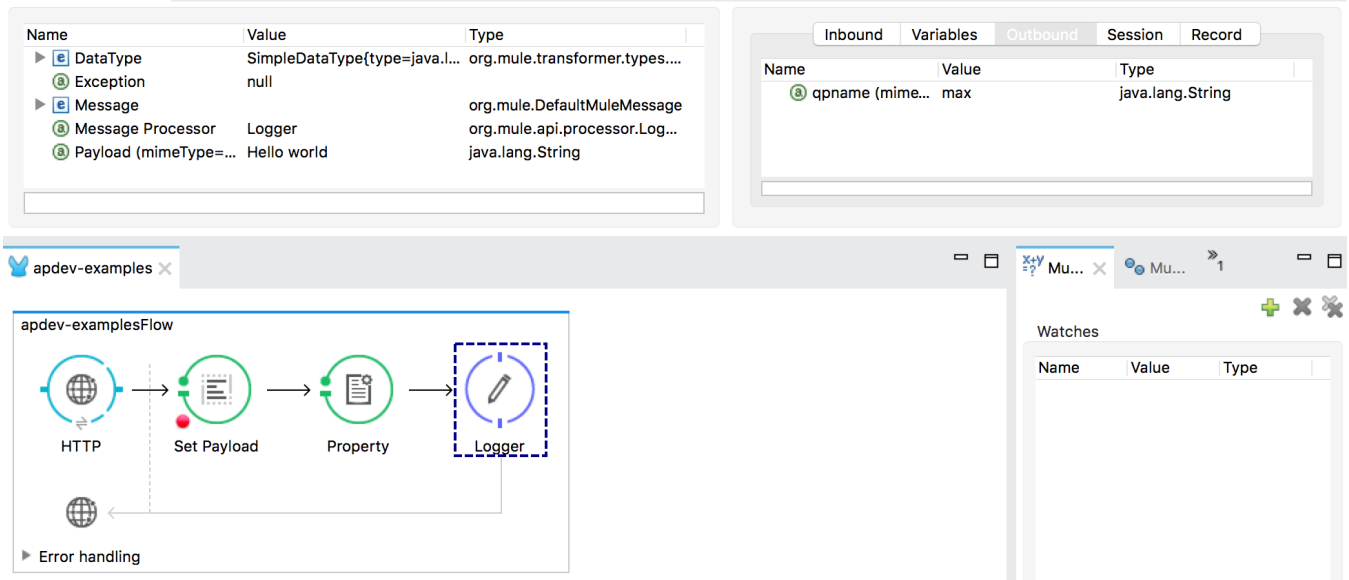


Step through the application

17. Click the Next processor button.



18. Look at the new value of the payload; it should be Hello world.
19. Look at the inbound and outbound properties; you should see they have not changed.
20. Click the Next processor button; you should now see the outbound property qpname.

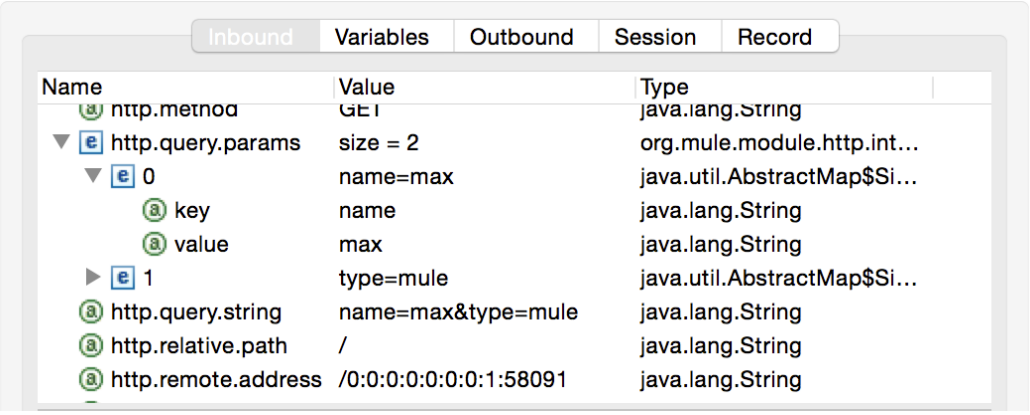


21. Step through the rest of the application.

Send query parameters with a request

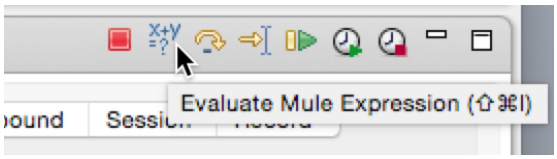
22. Return to Postman and add a parameter with a key of name and a value of max.
23. Add a second key / value pair of type and mule.
24. Click Send.

25. Return to the Mule Debugger view and locate your query parameters.



Use the Mule Expression evaluator

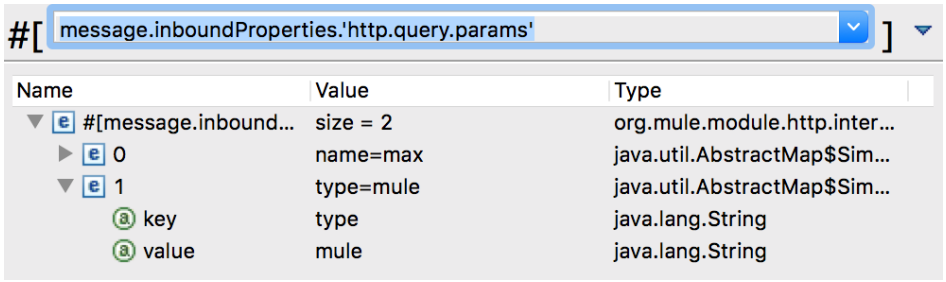
26. Click the Evaluate Mule Expression button.



27. Enter the following expression and press the Enter key.

```
#[message.inboundProperties.'http.query.params']
```

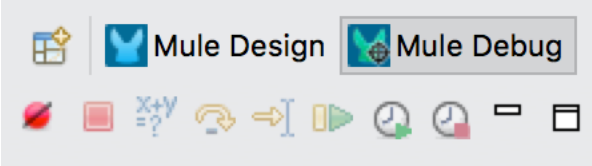
28. Expand the results; you should see the query parameters.



29. Click anywhere outside the expression window to close it.
30. Click the Resume button to execute the rest of the flow.
31. Stop the project.

Switch perspectives

32. Click the Mule Design tab in the upper-right corner of Anypoint Studio to switch perspectives.

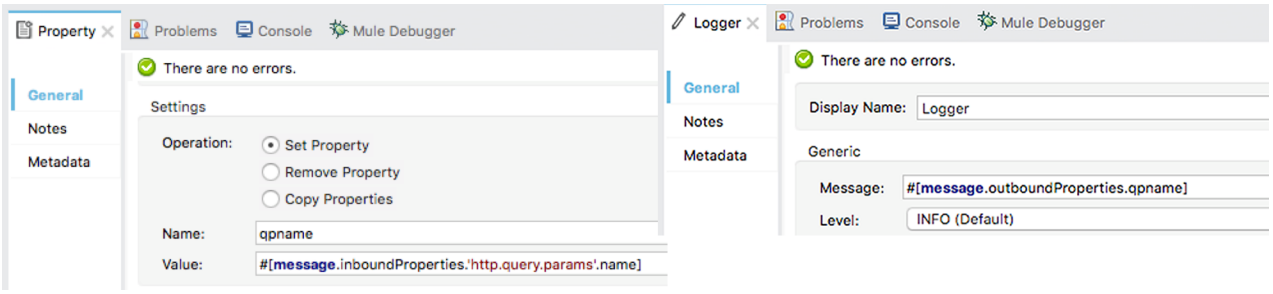


Note: In Eclipse, a perspective is a specific arrangement of views in specific locations. You can rearrange the perspective by dragging and dropping tabs and views to different locations. Use the Window menu in the main menu bar to save and reset perspectives.

Walkthrough 6-3: Read and write message properties using MEL expressions

In this walkthrough, you manipulate message properties. You will:

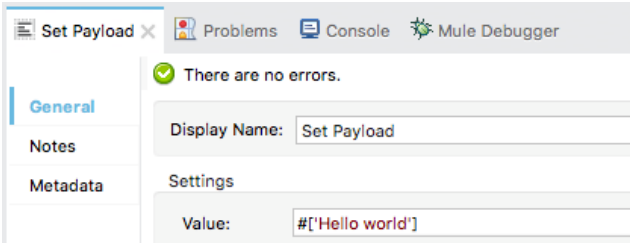
- Use an expression to set the payload.
- Use an expression to display specific info to the console.
- Use an expression to set an outbound property.
- Use an expression to read an outbound property.



Use an expression to set the payload

1. Return to apdev-examples.xml.
2. Navigate to the Properties view for the Set Payload transformer.
3. Change the value to an expression.

```
#['Hello world']
```

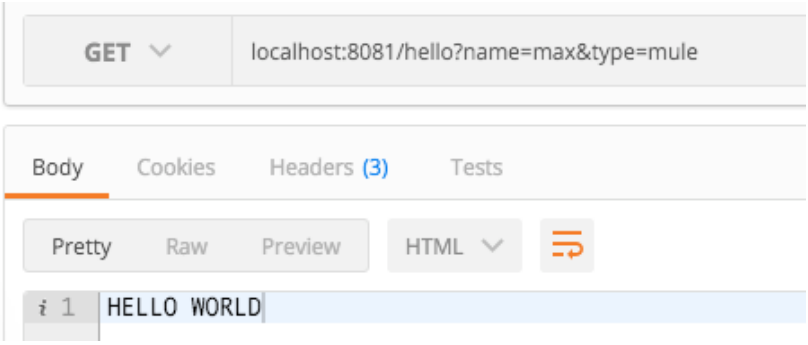


4. Run the project.
5. In Postman, make a request to http://localhost:8081/hello with the query parameters; the application should work as before.
6. In Anypoint Studio, return to the Set Payload expression and use autocomplete to use the toUpperCase() method to return the value in upper case.

```
#['Hello world'.toUpperCase()]
```

Note: Press Ctrl+Space to trigger autocomplete if it does not appear.

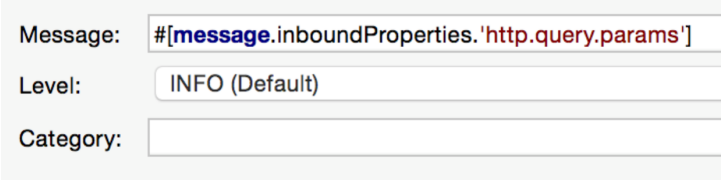
7. Click the Apply Changes button in the upper-right corner of the Properties view to redeploy the application.
8. In Postman, send the same request; the return string should now be in upper case.



Use an expression to display specific info to the console

9. In Anypoint Studio, navigate to the Properties view for the Logger component.
10. Set the message to display the http.query.params property of the message inbound properties.

```
#[message.inboundProperties.'http.query.params']
```



11. Click Apply Changes to redeploy the application.
12. In Postman, send the same request.
13. Return to the Anypoint Studio console; you should see a ParameterMap object listed.

```
**********************************************************************************************************
* apdev-examples                          * default                       * DEPLOYED           *
**********************************************************************************************************

INFO  2016-05-15 18:48:06,358 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.pro
cessor.LoggerMessageProcessor: ParameterMap{[name=[max], type=[mule]]}
```

14. Modify the Logger to display one of the query parameters.

```
#[message.inboundProperties.'http.query.params'.name]
```

15. Click Apply Changes to redeploy the application.
16. In Postman, send the same request.

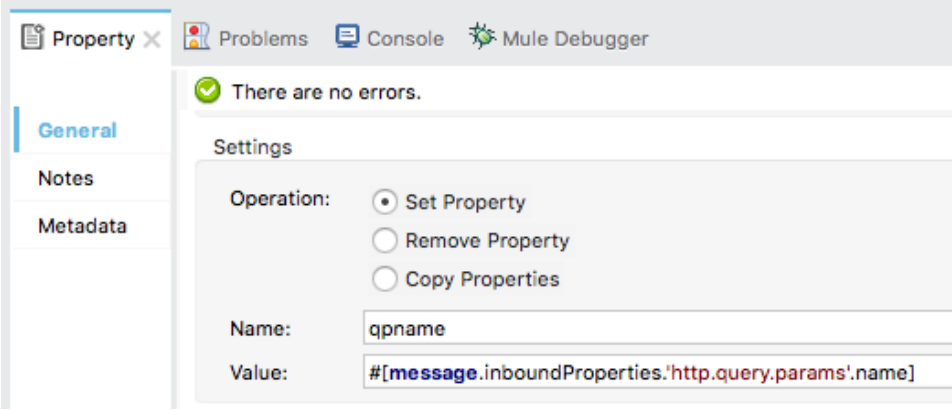
17. Return to the console; you should now see the value of the parameter displayed.

```
..........................................................
INFO  2016-05-15 18:48:41,451 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.pro
cessor.LoggerMessageProcessor: max
```

Use an expression to set an outbound property

18. In the Properties view for the Property transformer, change the value to an expression that evaluates the name query parameter.

    `#[message.inboundProperties.'http.query.params'.name]`



19. Modify the Logger to display the value of this outbound property.

    `#[message.outboundProperties.qpname]`

20. Click Apply Changes to redeploy the application.
21. In Postman, send the same request.
22. Return to the console; you should see the value of your variable displayed.

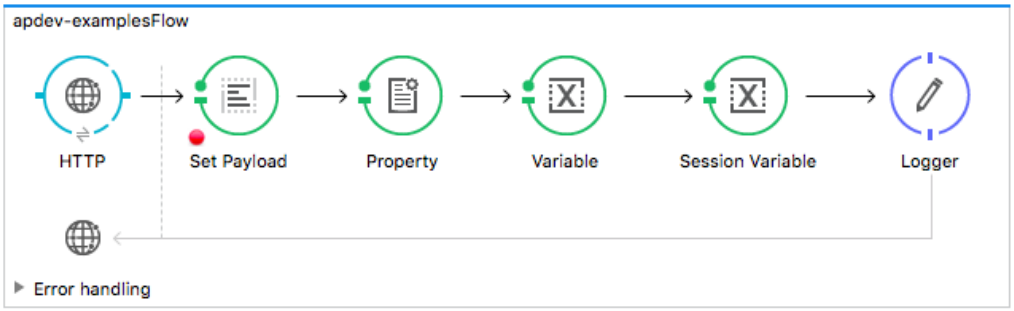
```
..........................................................
INFO  2016-05-15 18:50:12,742 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.pro
cessor.LoggerMessageProcessor: max
```

23. Stop the project.

Walkthrough 6-4: Read and write variables

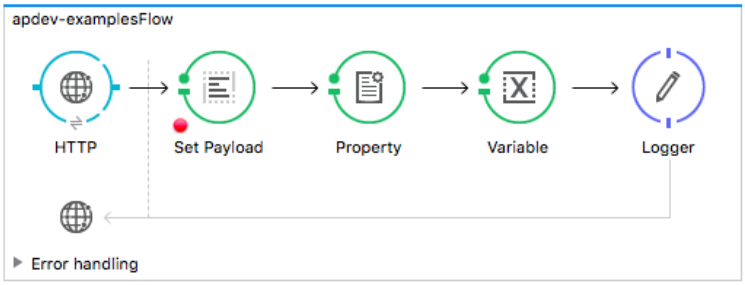
In this walkthrough, you create flow and session variables. You will:

- Use the Variable transformer to create a flow variable.
- Use the Session transformer to create a session variable.
- Use the Mule Debugger to see their values.



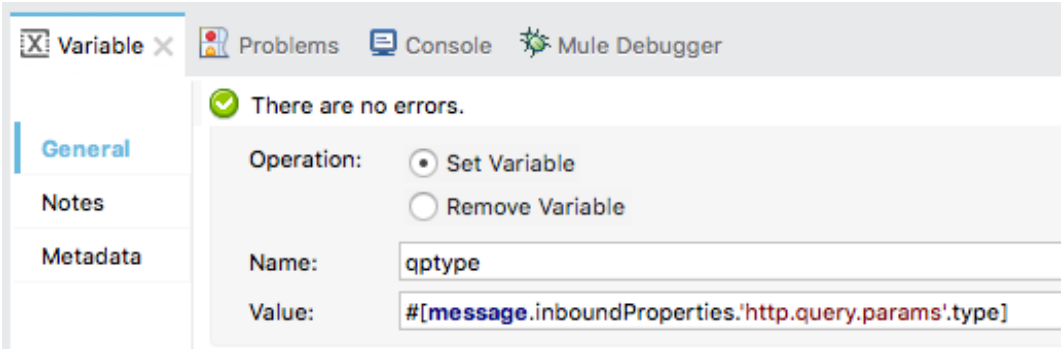
Create a flow variable

1. Return to apdev-examples.xml.
2. Add a Variable transformer between the Property and Logger processors.



3. In the Variable properties view, select Set Variable.
4. Set the name to qptype and the value to your second query parameter, type.

```
#[message.inboundProperties.'http.query.params'.type]
```

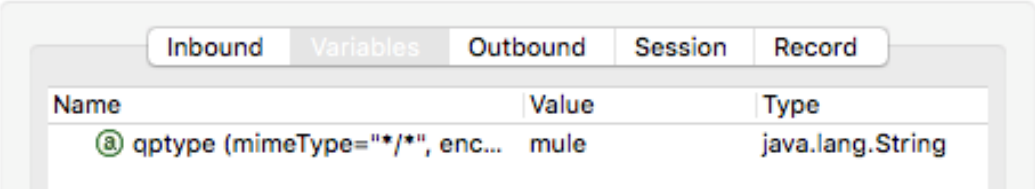


5. Modify the Logger to also display the value of this new variable.

```
Name: #[message.outboundProperties.qpname] Type: #[flowVars.qptype]
```

Debug the application

6. Debug the project.
7. In Postman, send the same request with the parameters.
8. In the Mule Debugger, select the Variables tab.
9. Step to the Logger; you should see your new flow variable.

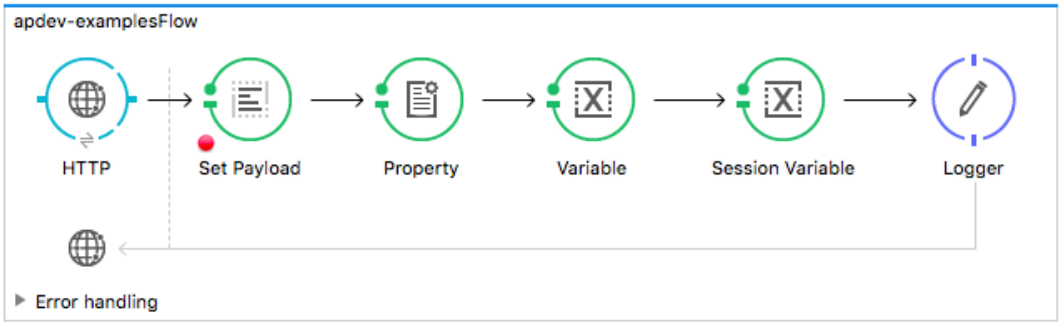


10. Click the Resume button.
11. Look at the console; you should see the value of your outbound property and the value of your new flow variable displayed.

```
INFO  2016-05-15 18:57:51,141 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.pro
cessor.LoggerMessageProcessor: Name: max Type: mule
```

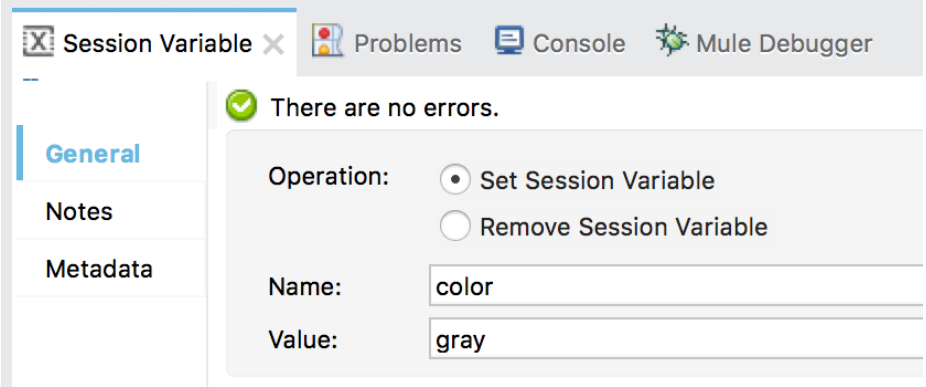
Create a session variable

12. Switch to the Mule Design perspective.
13. Add a Session Variable transformer between the Variable and Logger processors.



14. In the Session Variable Properties view, select Set Session Variable.

15. Set the name to color and give it any value, static or dynamic.

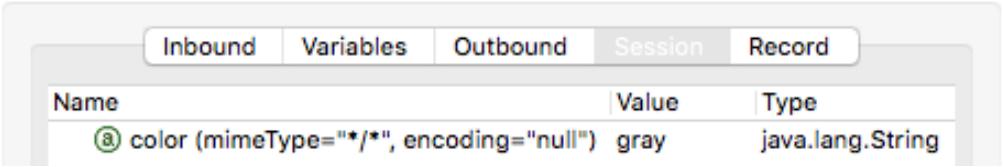


16. Modify the Logger to also display the session variable.

```
Name: #[message.outboundProperties.qpname] Type: #[flowVars.qptype]
Color: #[sessionVars.color]
```

Debug the application

17. Click Apply Changes to redeploy the application in debug mode.
18. In Postman, send the same request.
19. In the Mule Debugger, select the Session tab.
20. Step to the Logger; you should see your new session variable.



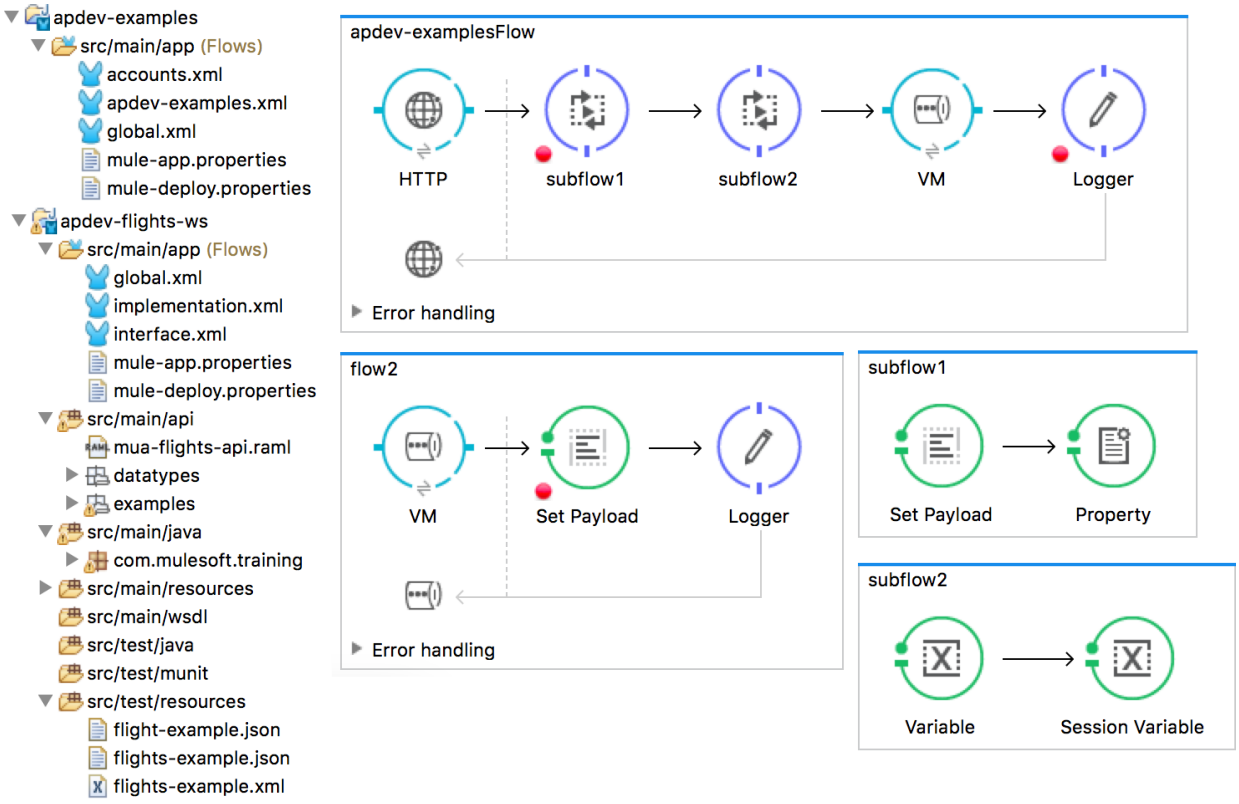
21. Click the Resume button.
22. Look at the console; you should see the value of your session variable displayed.

```
INFO  2016-05-15 19:00:03,872 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.pro
cessor.LoggerMessageProcessor: Name: max Type: mule Color: gray
```

23. Stop the project and switch perspectives.

Note: You will explore the persistence of these variables in the next module.

Module 7: Structuring Mule Applications



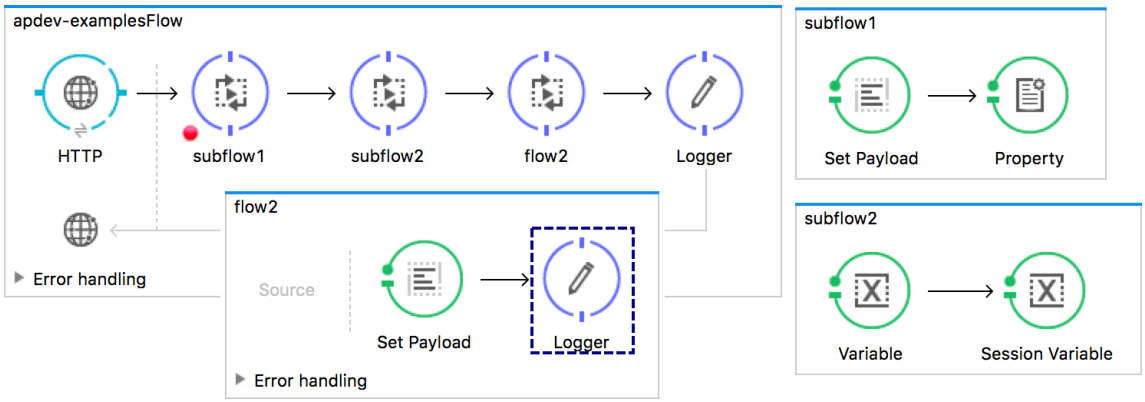
At the end of this module, you should be able to:

- Create and reference flows and subflows.
- Pass messages between flows using the Java Virtual Machine (VM) transport.
- Investigate variable persistence through subflows and flows and across transport barriers.
- Encapsulate global elements in separate configuration files.
- Explore the files and folder structure of Mule projects.

Walkthrough 7-1: Create and reference flows and subflows

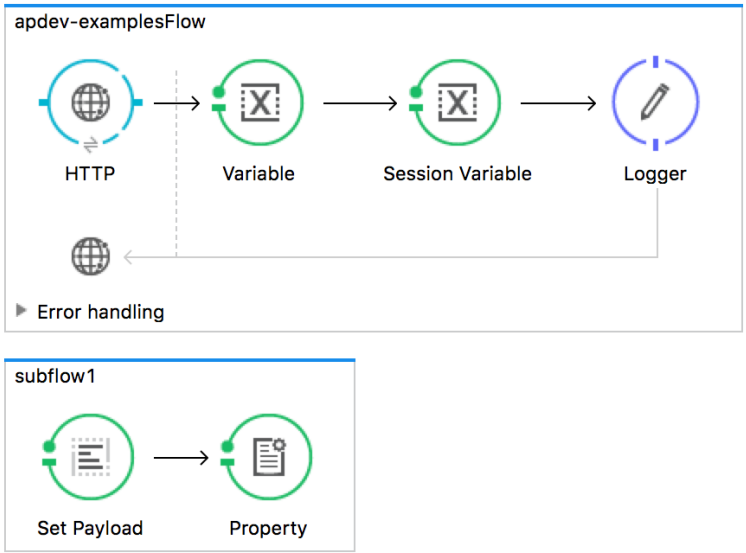
In this walkthrough, you continue to work with apdev-examples.xml. You will:

- Extract processors into separate subflows and flows.
- Use the Flow Reference component to reference other flows.
- Explore variable persistence through flows and subflows.



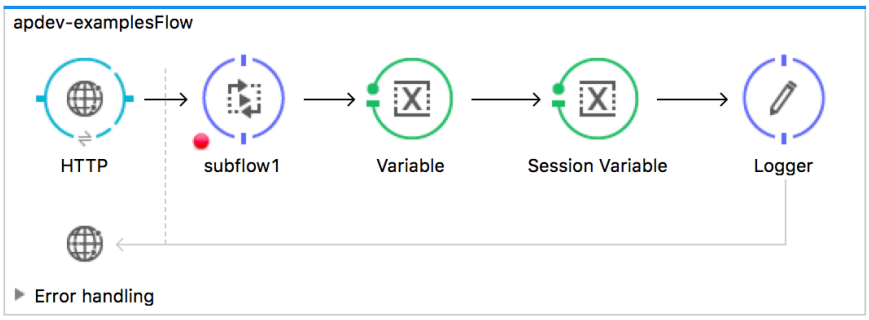
Create a subflow

1. Return to apdev-examples.xml in Anypoint Studio.
2. Drag a Sub Flow scope from the Mule Palette and drop it beneath the existing flow in the canvas.
3. Select the Set Payload and Property transformers in apdev-examplesFlow and drag them into the subflow.
4. Change the name of the subflow to subflow1.



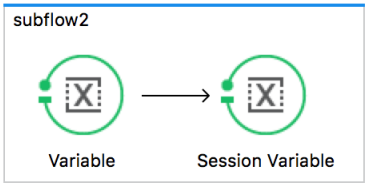
Reference a subflow

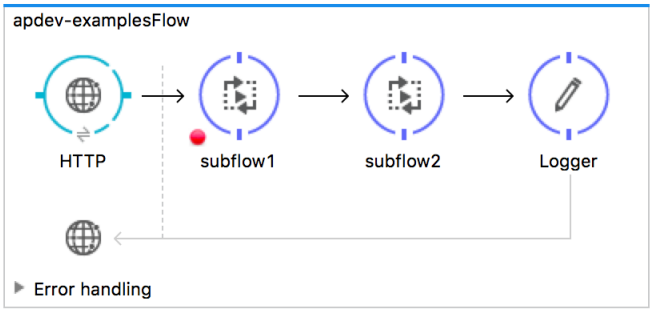
5. Drag a Flow Reference component from the Mule Palette and drop it into the apdev-examplesFlow between the HTTP Listener endpoint and the Variable transformer.
6. In the Flow Reference properties view, set the flow name to subflow1.
7. Add a breakpoint to the subflow1 Flow Reference.

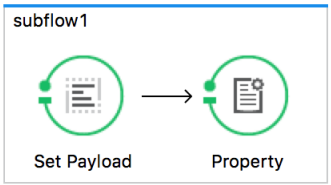


Extract processors into a subflow

8. Right-click the Variable and Session Variable transformers and select Extract to > Sub Flow.
9. In the Extract Flow dialog box, set the flow name to subflow2.
10. Leave the target Mule configuration set to current and click OK.
11. Look at the new Flow Reference Properties view; the flow name should already be set to subflow2.



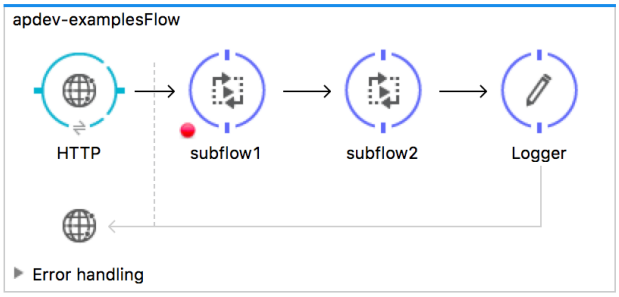


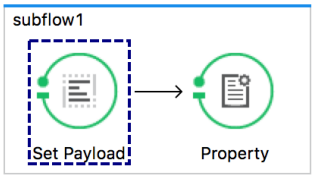


12. Drag subflow2 below subflow1 in the canvas.

Debug the application

13. Debug the project.
14. In Postman, send the same request to http://localhost:8081/hello?name=max&type=mule.
15. In the Mule Debugger, step through the application, watching messages move into and out of the subflows.

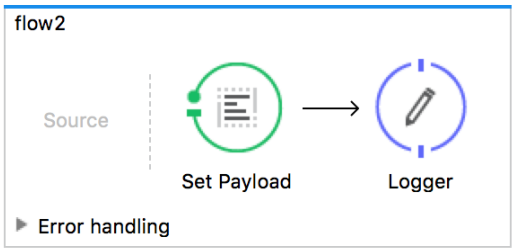




16. In Postman, send the same request again.
17. In the Mule Debugger, step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables in each of the flows.

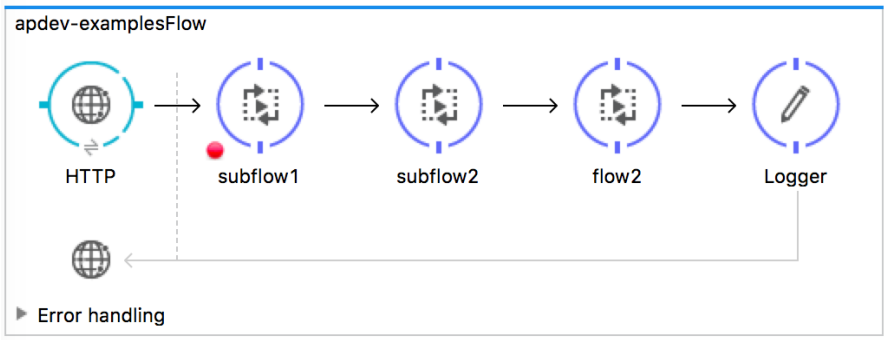
Create a second flow

18. Switch perspectives.
19. Drag a Flow scope element to the canvas and drop it beneath the existing flows.
20. Change the flow name to flow2.
21. Add a Set Payload transformer to the process section of the flow.
22. In the Set Payload properties view, set the value to Goodbye.
23. Add a Logger after the Set Payload transformer.



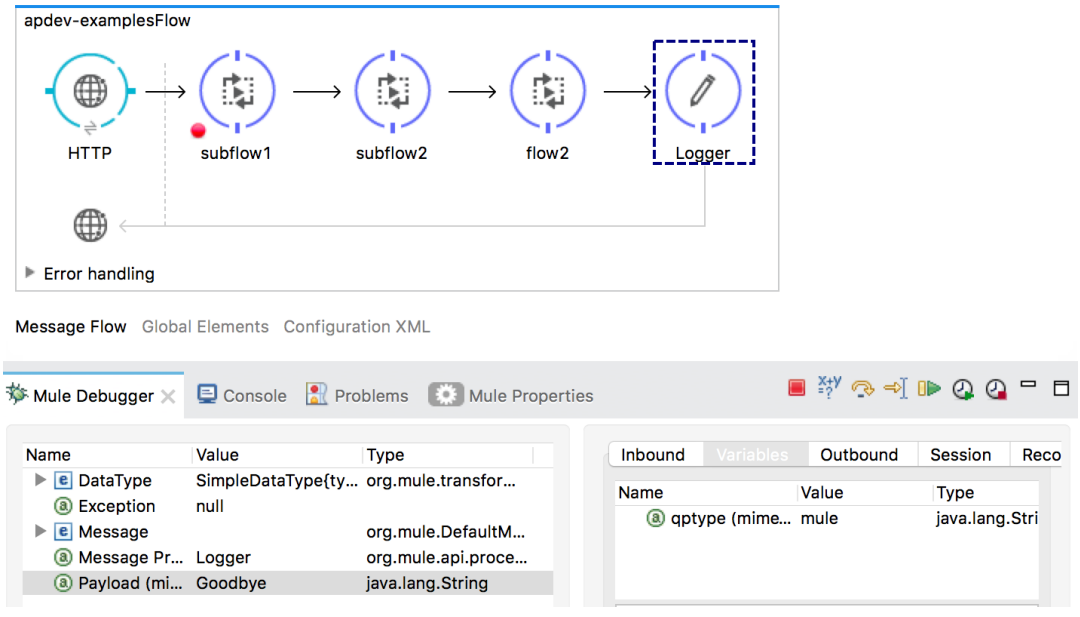
Reference a flow

24. In apdev-examplesFlow, add a Flow Reference component between the subflow2 Flow Reference and the Logger.
25. In the Flow Reference Properties view, set the flow name to flow2.



Debug the application

26. Save the file to redeploy the application in debug mode.
27. In Postman, send the same request.
28. In the Mule Debugger, step through the application, watching the flow move into and out of the second flow; at the end, you should see the payload is the value set in the second flow.

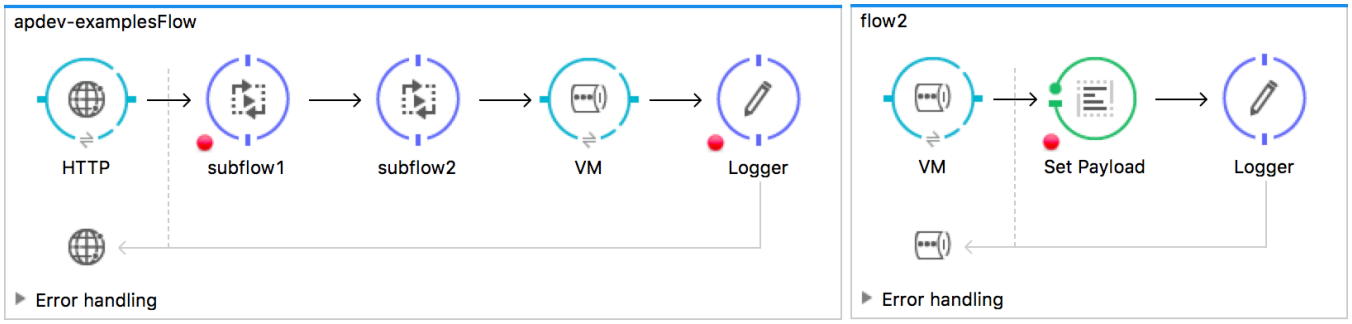


29. In Postman, send the same request again.
30. In the Mule Debugger, step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables in each of the flows.
31. Stop the project and switch perspectives.

Walkthrough 7-2: Pass messages between flows using the Java Virtual Machine (VM) transport

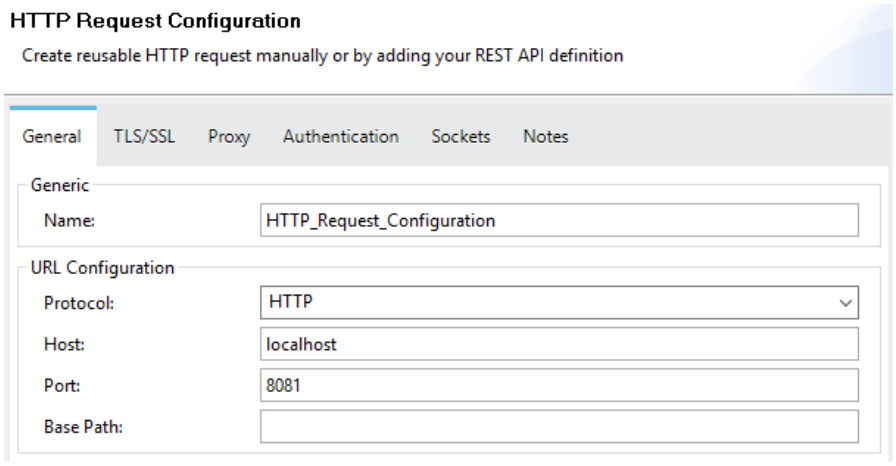
In this walkthrough, you continue to work with the apdev-examplesFlow. You will:

- Pass messages through an HTTP transport barrier.
- Pass messages between flows using the VM transport.
- Explore variable persistence across these transport barriers.



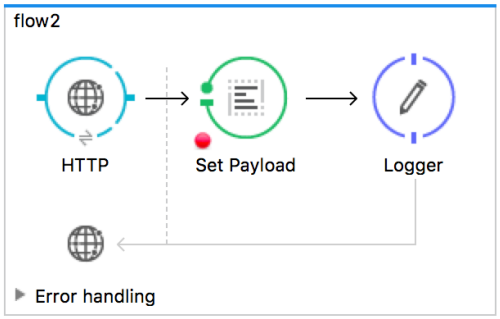
Create an HTTP transport barrier

1. Return to apdev-examples.xml.
2. Switch to the Global Elements view.
3. Click Create.
4. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.
5. In the Global Elements dialog box, set the host to localhost, the port to 8081, and click OK.

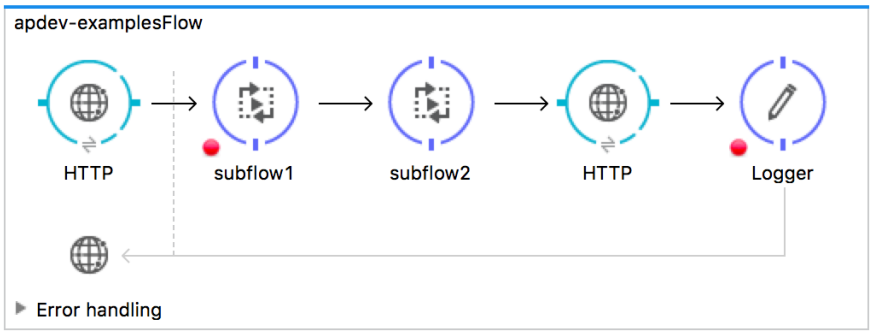


6. Return to the Message Flow view.
7. Drag an HTTP connector into the Source section of flow2.

8. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
9. Set the path to /flow2 and the allowed methods to GET.
10. Add a breakpoint to the Set Payload transformer in flow2.



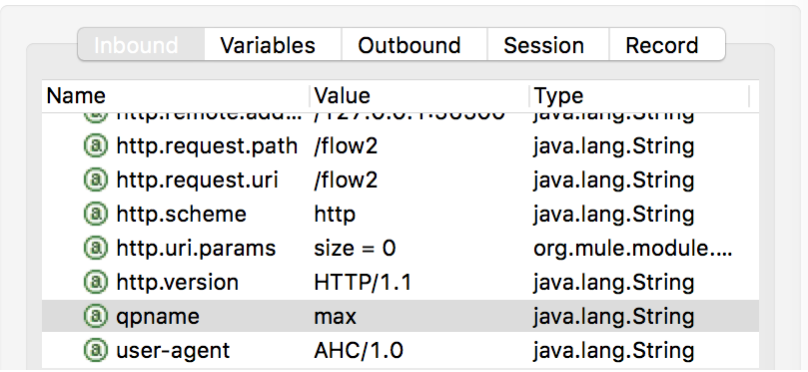
11. Add an HTTP Request endpoint after the flow2 reference in apdev-examplesFlow.
12. Delete the flow2 reference.
13. In the HTTP properties view, set the connector configuration to HTTP_Request_Configuration.
14. Set the path to /flow2 and the method to GET.
15. Add a breakpoint to the Logger.



Debug the application

16. Debug the application.
17. In Postman, send the same request.
18. In the Mule Debugger, step into flow2.

19. Locate the qpname property in the inbound properties.



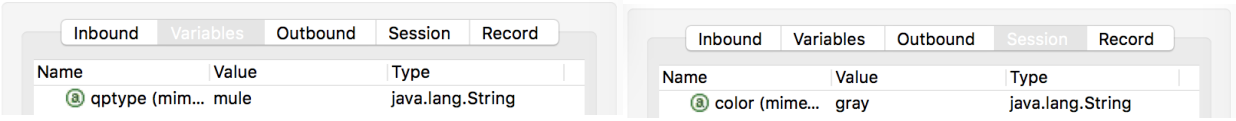
20. Look at the outbound properties, the flow variables, and the session variables.

    Note: Session variables are persisted across some but not all transport barriers. As you see here, they are not propagated across the HTTP transport barrier.

21. Step through the flow until the message returns to apdev-examplesFlow.

    Note: If the application times out, debug it again and this time use the Resume button to quickly step to the breakpoint in flow2 and then step through until the message returns to apdev-examplesFlow.

22. Look at the inbound and outbound properties and the flow and session variables.

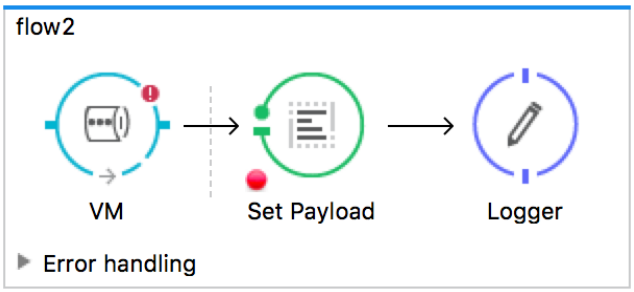


23. Click Resume.
24. Stop the project and switch perspectives.

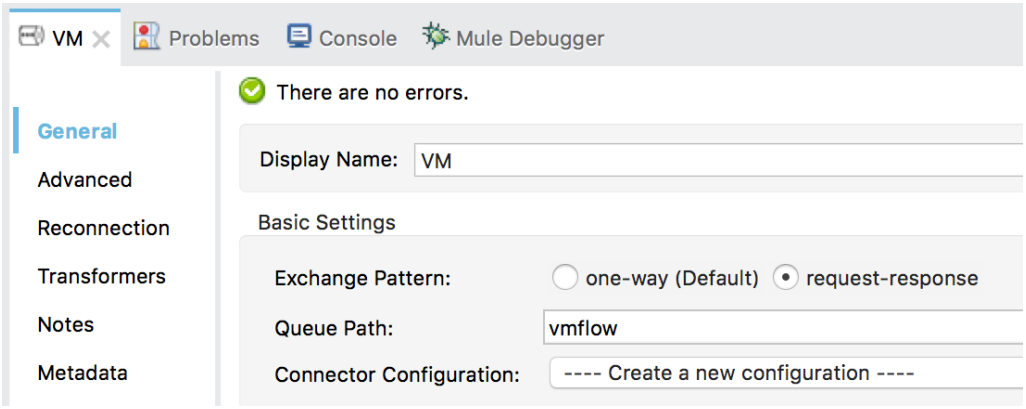
Create a VM transport barrier

25. In flow2, delete the HTTP Listener endpoint.

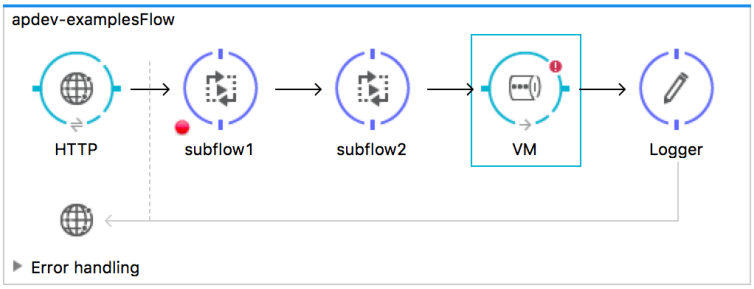
26. Drag a VM connector from the Mule Palette into the source section of flow2.



27. In the VM properties view, set the exchange pattern to request-response.
28. Set the queue path to vmflow.



29. In apdev-examplesFlow, add a VM connector endpoint after the HTTP Request endpoint.
30. Delete the HTTP Request endpoint.

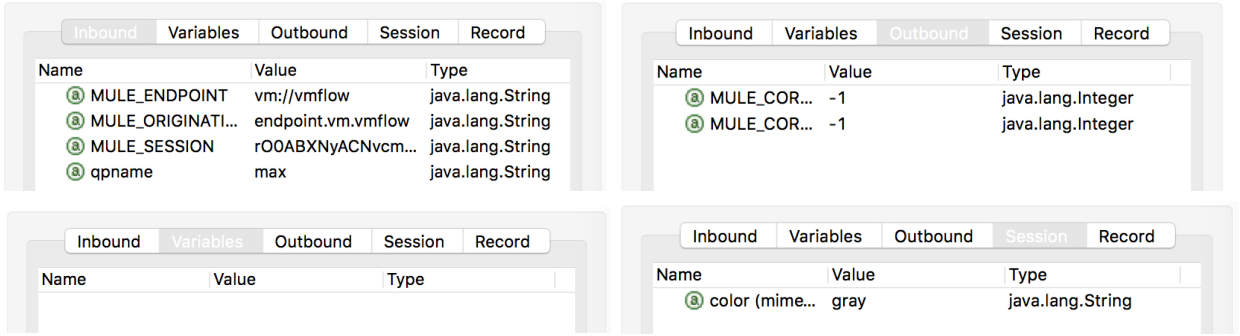


31. In the VM properties view, set the exchange pattern to request-response.
32. Set the queue path to vmflow.

Debug the application

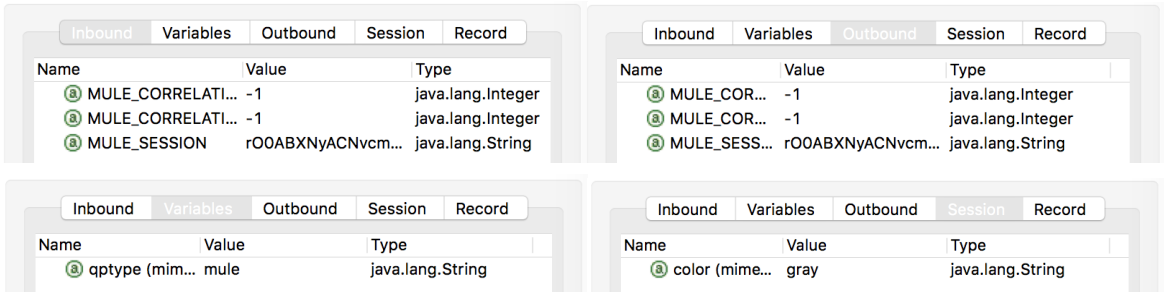
33. Debug the project.
34. In Postman, send the same request.
35. In the Mule Debugger, step through the application into flow2.

36. Look at the inbound and outbound properties and the flow and session variables.



Note: Session variables are persisted across some but not all transport barriers. As you see here, they are propagated across the VM transport barrier.

37. Step through the flow until the message returns to apdev-examplesFlow.
38. Look at the inbound and outbound properties and the flow and session variables.

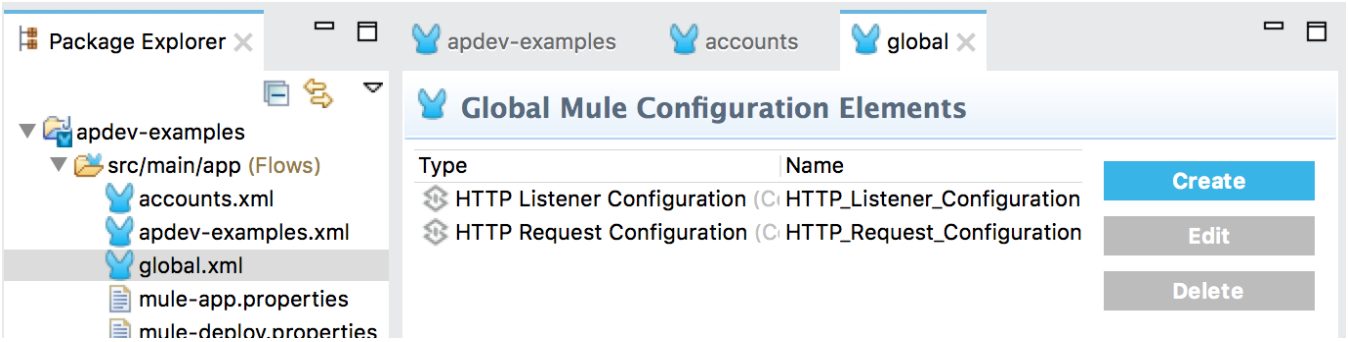


39. Step through the rest of the application.
40. Stop the project and switch perspectives.

Walkthrough 7-3: Encapsulate global elements in a separate configuration file

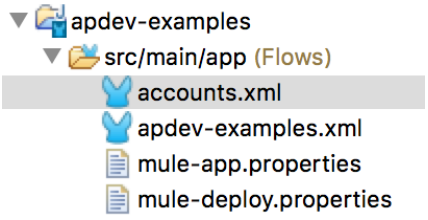
In this walkthrough, you refactor your apdev-examples project. You will:

- Create a new configuration file with an endpoint that uses an existing global element.
- Create a configuration file global.xml for just global elements.
- Move the existing global elements to global.xml.



Create a new configuration file

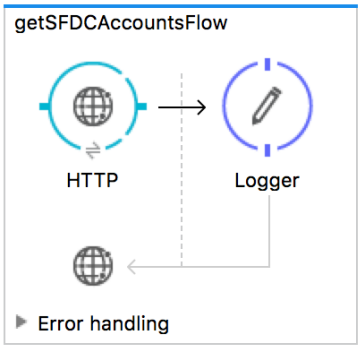
1. Return to the apdev-examples project.
2. In the Package Explorer, right-click the project and select New > Mule Configuration File.
3. In the New Mule Configuration File dialog box, set the name to accounts.xml and click Finish.



4. Drag out an HTTP Listener to the canvas.
5. In the HTTP properties view, set the connector configuration to the existing configuration.
6. Set the path to /sfdc and the allowed methods to GET.
7. Add a Logger component to the flow.

8. Change the name of the existing flow to getSFDCAccountsFlow.

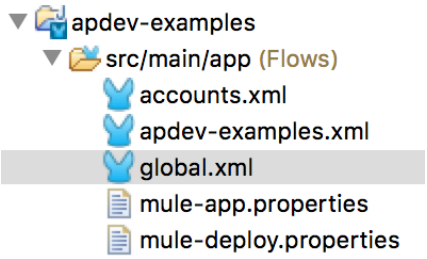
   Note: You will use this flow in a later module to retrieve data from Salesforce.



9. Switch to the Global Elements view; you should not see any global elements.

Create a global configuration file

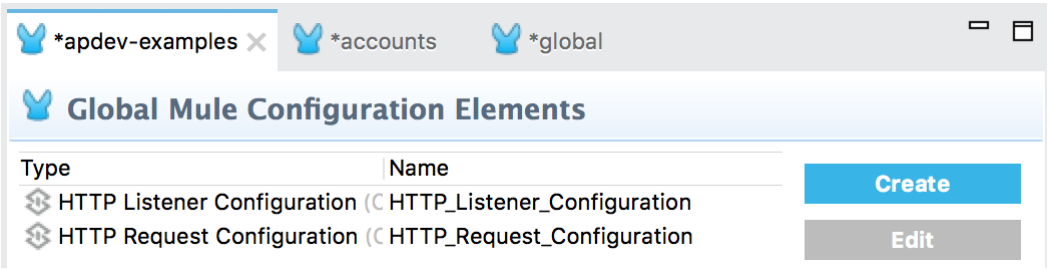
10. Create a new Mule configuration file called global.xml.



11. In global.xml, switch to the Configuration XML view.
12. Place some empty lines between the start and end mule tags.

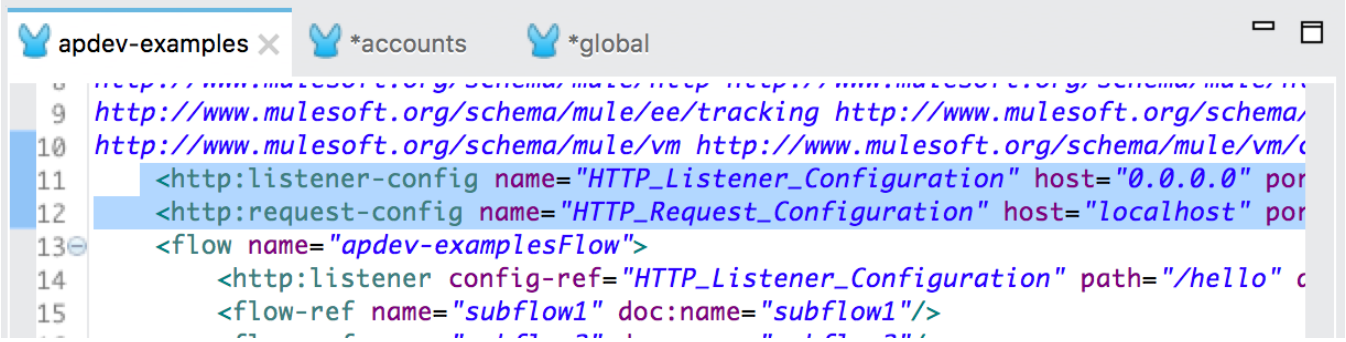
Remove the existing global elements

13. Return to apdev-examples.xml.
14. Switch to the Global Elements view and see there are two configurations.



15. Switch to the Configuration XML view.

16. Select and cut the two configuration elements defined before the flows.



Note: If you delete the global elements from the Global Elements view instead, the config-ref values are also removed from the connector endpoints and you need to re-add them.

17. Return to the Message Flow view.

Move the global elements to a new configuration file

18. Return to global.xml.
19. Paste the global elements you cut to the clipboard between the start and end mule tags.

20. Switch to the Global Elements view; you should see the two configurations.

Test the application

21. Save all the files; any problems should disappear.
22. Return to apdev-examples.xml.
23. Double-click the HTTP Listener connector in apdev-examplesFlow; the connector configuration should still be set to HTTP_Listener_Configuration, which is now defined in global.xml.
24. Run the project.
25. In Postman, send the same request; you should still get a response of Goodbye.

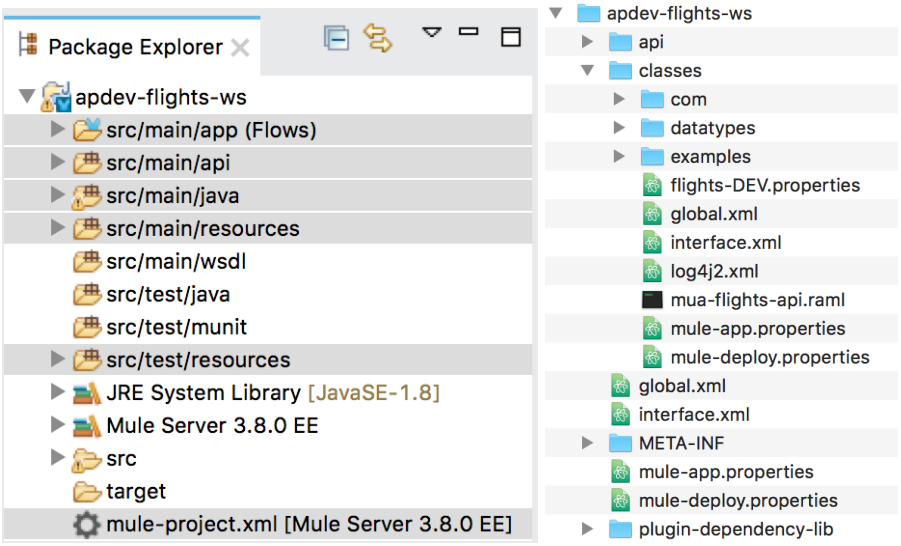
Close the project

26. Return to Anypoint Studio.
27. Stop the project.
28. In the Package Explorer, right-click apdev-examples and select Close Project.

Walkthrough 7-4: Create a well-organized Mule project

In this walkthrough, you create a new project for the Mule United Airlines (MUA) flights application that you will build during the course and then review and organize its files and folders. You will:

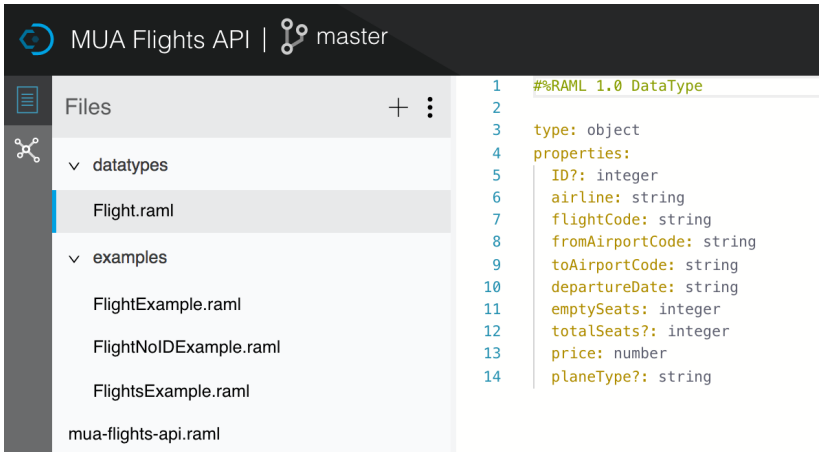
- Create a project based on a new API in Anypoint Platform Design Center.
- Review the project's configuration and properties files.
- Create an application properties file and a global configuration file for the project.
- Add Java files and test resource files to the project.
- Create and examine the contents of a deployable archive for the project.



Create a new API in Design Center

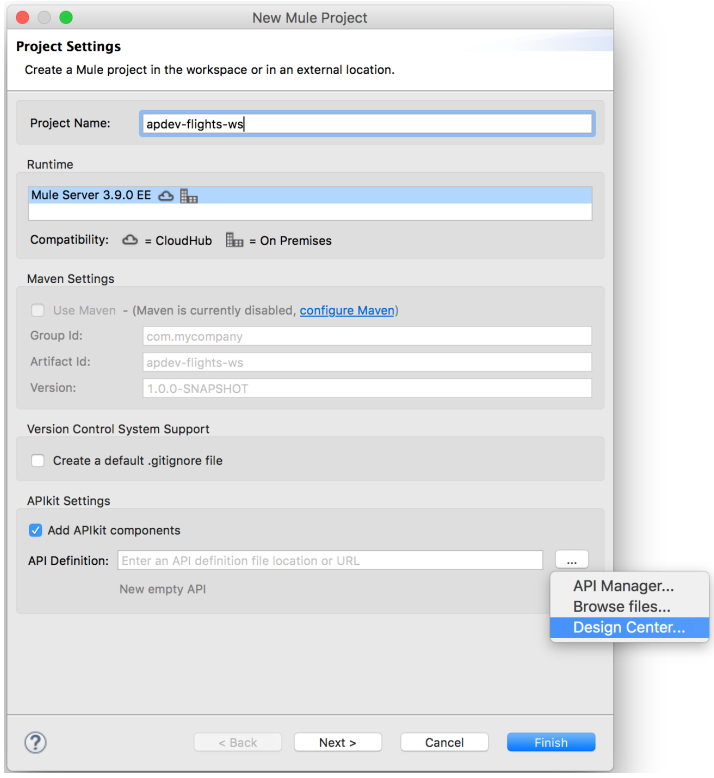
1. Return to Anypoint Platform in a web browser.
2. In Design Center, create a new API Specification project called MUA Flights API.
3. In API designer, click the options menu in the file browser and select Import.
4. In the Import dialog box, browse to your student files and select the MUA-Flights-API.zip located in the resources folder.
5. Click Import.
6. In the Replace dialog box, click Replace file.

7. Explore the API; be sure to look at what resources are defined and the structure of the Flight data type.

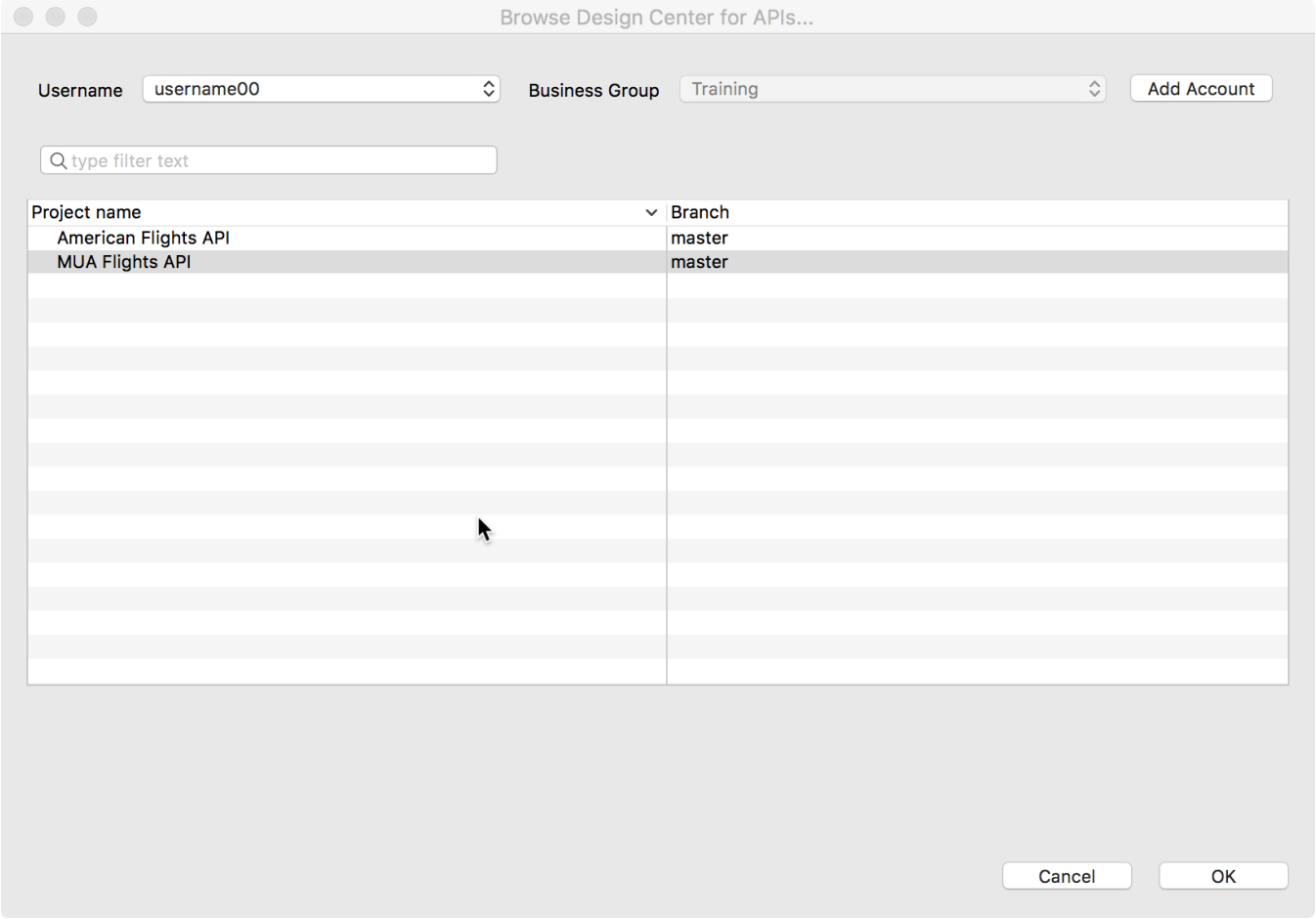


Create a new project to implement this API in Anypoint Studio

8. Return to Anypoint Studio.
9. Right-click in the Package Explorer and select New > Mule Project.
10. In the New Mule Project dialog box, set the project name to apdev-flights-ws.
11. Check Add APIkit components.
12. Click the browse button next to API Definition and select Design Center.



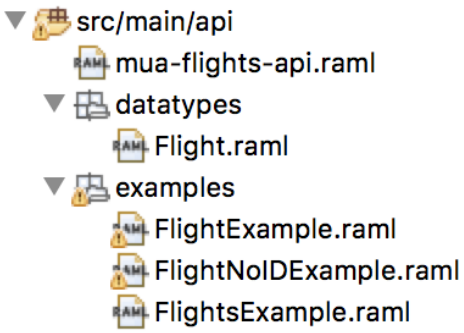
13. In the Browse API Design Center for APIs dialog box, select MUA Flights API and click OK.



14. In the New Mule Project dialog box, click Finish.

Locate the new RAML files in Anypoint Studio

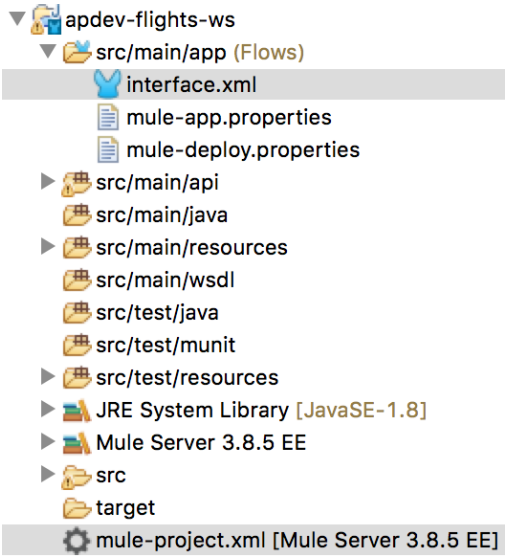
15. Return to the apdev-flights-ws project in Anypoint Studio.
16. In the Package Explorer, expand the src/main/api folder; you should see the MUA Flights API files.



17. Open mua-flights-api.raml and review the file.

Review project configuration files

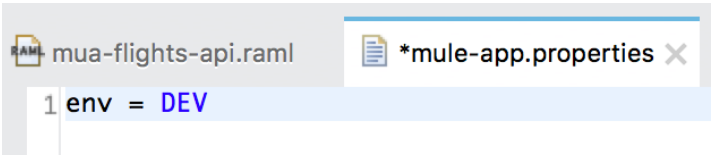
18. Look at the mua-flights-api.xml file that was created; it should have a get:/flights flow.
19. Rename mua-flights-api.xml to interface.xml.
20. Locate mule-project.xml in the project and open it.
21. Review its contents and then close the file.



22. Create a new Mule configuration file called implementation.xml.

Review project properties files

23. Locate mule-app.properties in the project and open it.
24. Add an environment variable called env equal to DEV.

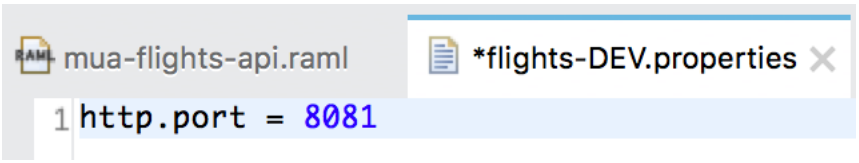


25. Save the file and close it.
26. Locate and open mule-deploy.properties.
27. Review its contents and then close the file.

Create a properties file for application parameters

28. Right-click src/main/resources and select New > File.

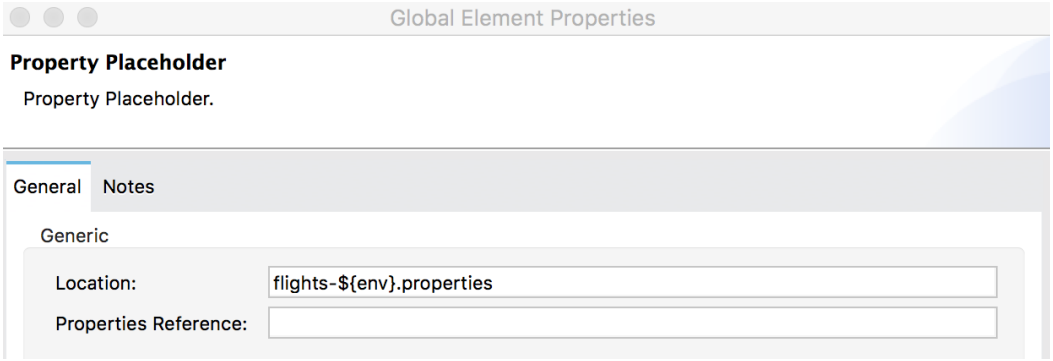
29. In the New File dialog box, set the file name to flights-DEV.properties and click Finish.
30. Add a property called http.port equal to 8081.



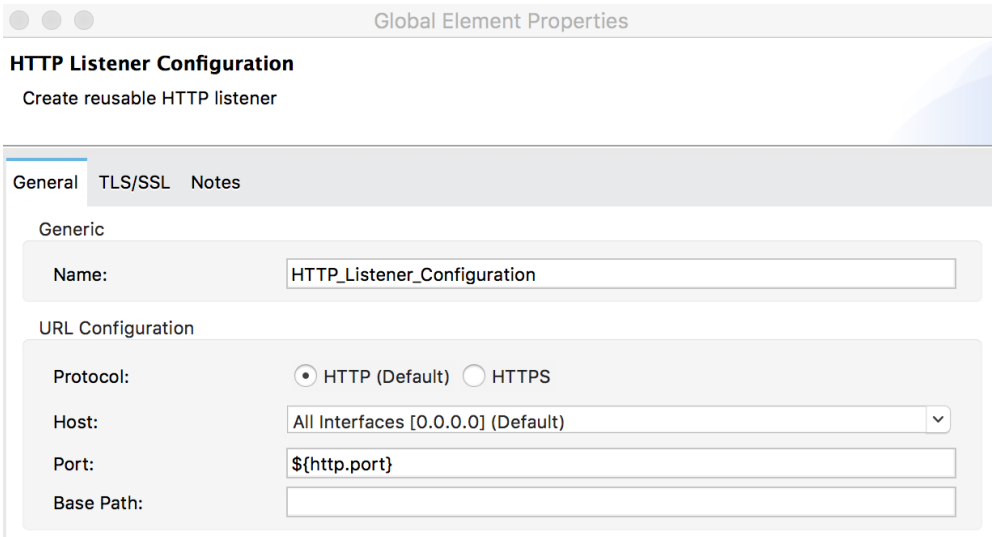
31. Save and close the file.

Create a global configuration file

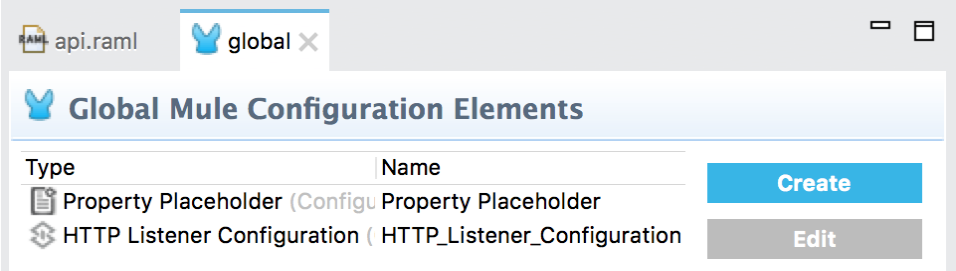
32. In src/main/app, create a new Mule configuration file called global.xml.
33. Switch to the Global Elements view.
34. Create a new Property Placeholder configuration with a location set to flights-${env}.properties.



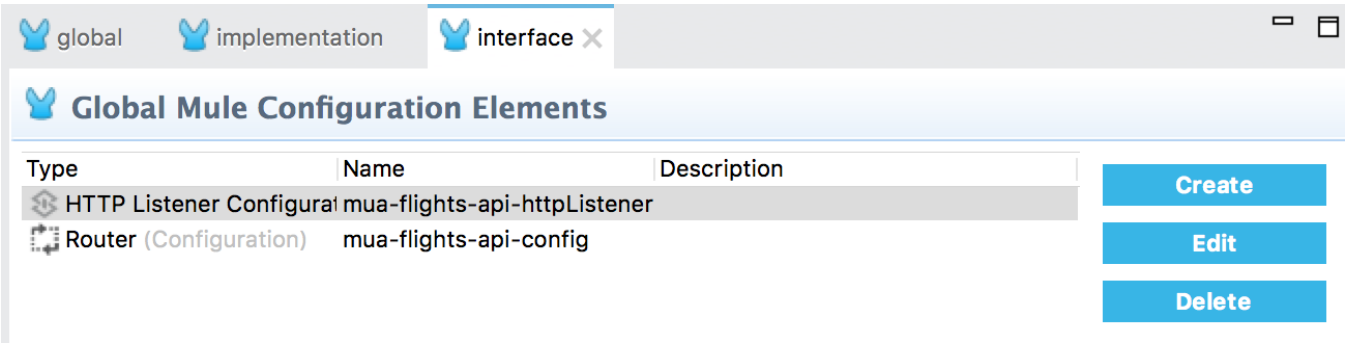
35. Click OK.
36. Create a new HTTP Listener Configuration with a port set to ${http.port}.



37. Confirm you now have two global elements defined in the global.xml file.



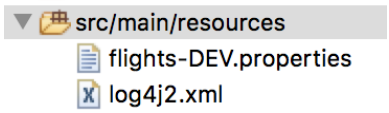
38. Save and close the file.
39. Go to the Global Elements view in interface.xml.
40. Delete the mua-flights-api-httpListener HTTP Listener Configuration.



41. Return to the Message Flow view.
42. In the Properties view for the HTTP Listener in mua-flights-api-main, set the connector configuration to HTTP_Listener_Configuration.
43. In the Properties view for the HTTP Listener in mua-flights-api-console, set the connector configuration to HTTP_Listener_Configuration.

Review src/main/resources

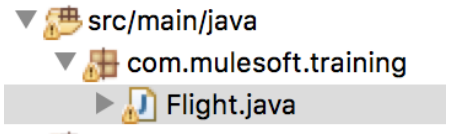
44. In src/main/resources, open log4j2.xml.
45. Review and then close the file.



Add Java files to src/main/Java

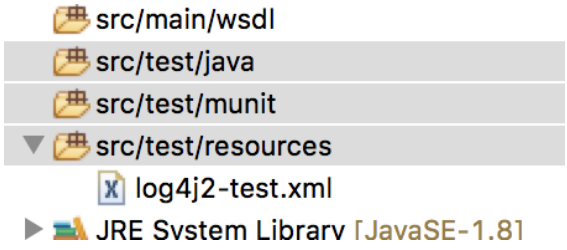
46. In your computer's file browser, locate the com folder in the java folder of the student files.
47. Drag the com folder into the src/main/java folder in the Anypoint Studio apdev-flights-ws project.
48. Expand the new com directory.

49. Open the Flight.java file.
50. Review the code and then close the file.



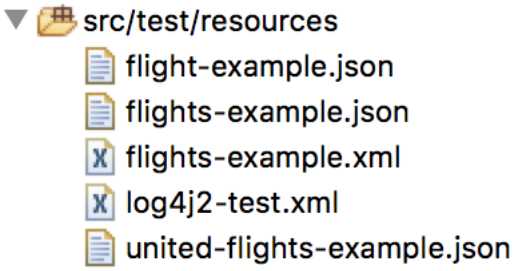
Review src/test folders

51. In the project, locate the three src/test folders.
52. Expand the src/test/resources folder.



Add test resources

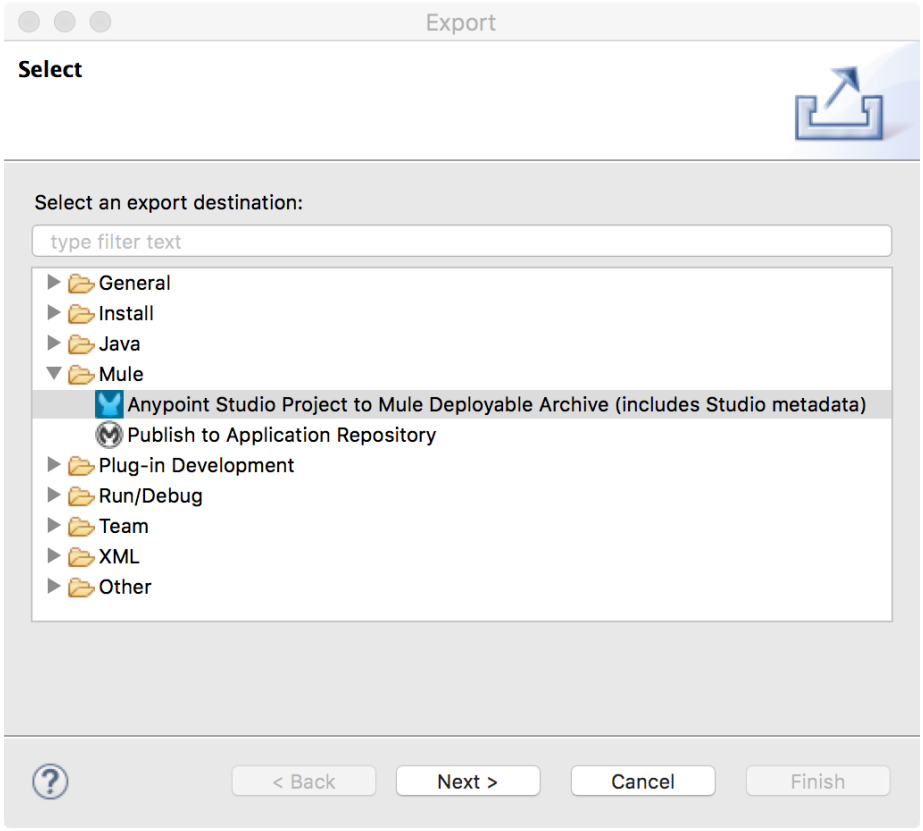
53. In your computer's file browser, expand the examples folder in the student files.
54. Select the four flights and one united-flights files (JSON AND XML) and drag them into the src/test/resources folder in the Anypoint Studio apdev-flights-ws project.



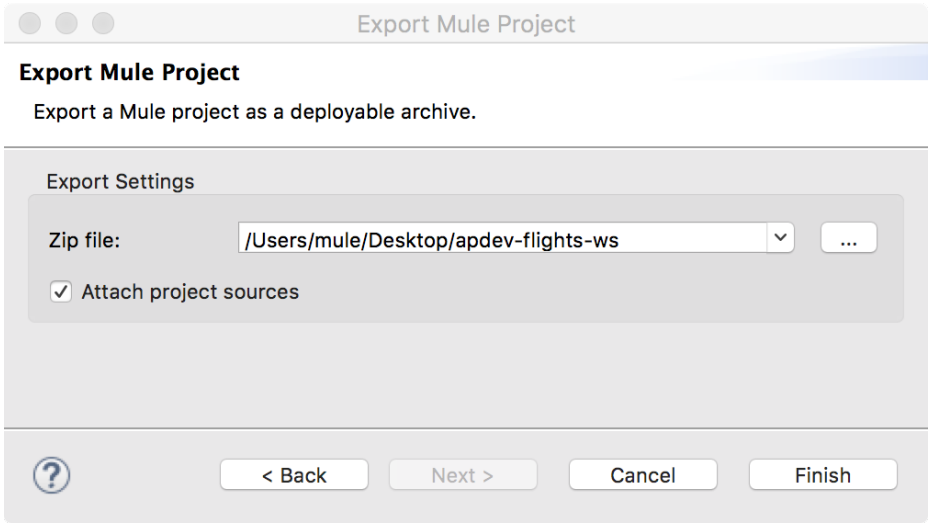
Examine the contents of a deployable archive

55. In Anypoint Studio, right-click the project and select Export.

56. In the Export dialog box, select Mule > Anypoint Studio to Mule Deployable Archive and click Next.

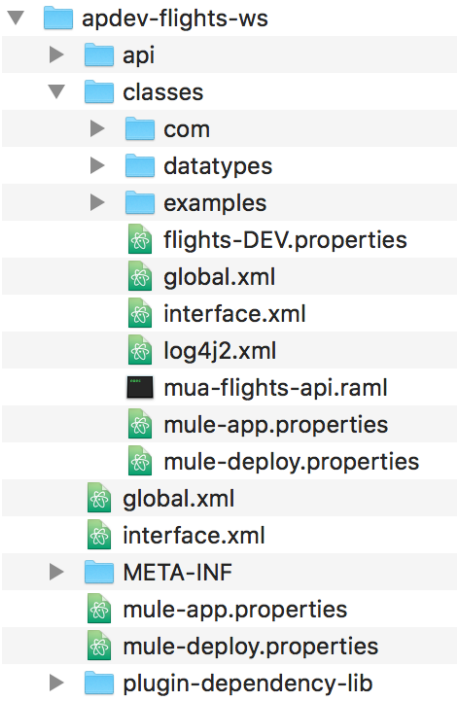


57. Set the Zip file to a location that you can find and click Finish.

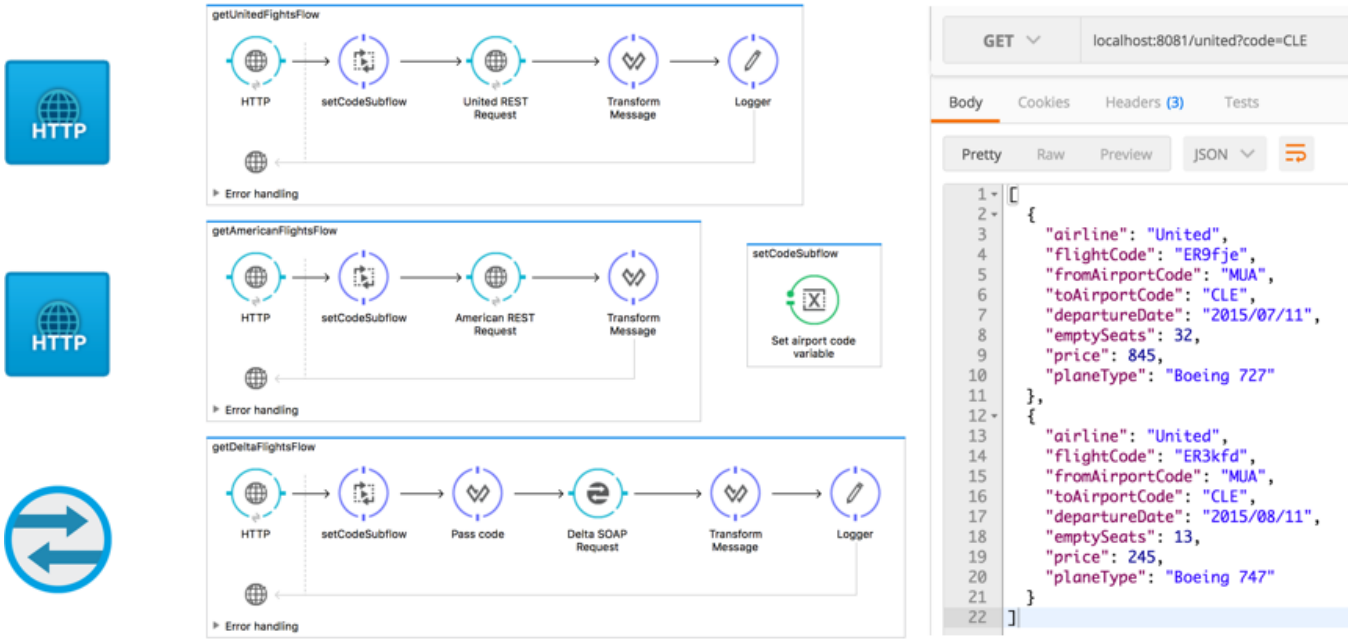


58. In your computer's file browser, locate the ZIP file and expand it.
59. Open the resulting apdev-flights-ws folder.

60. Expand the classes folder and examine the contents.



Module 8: Consuming Web Services



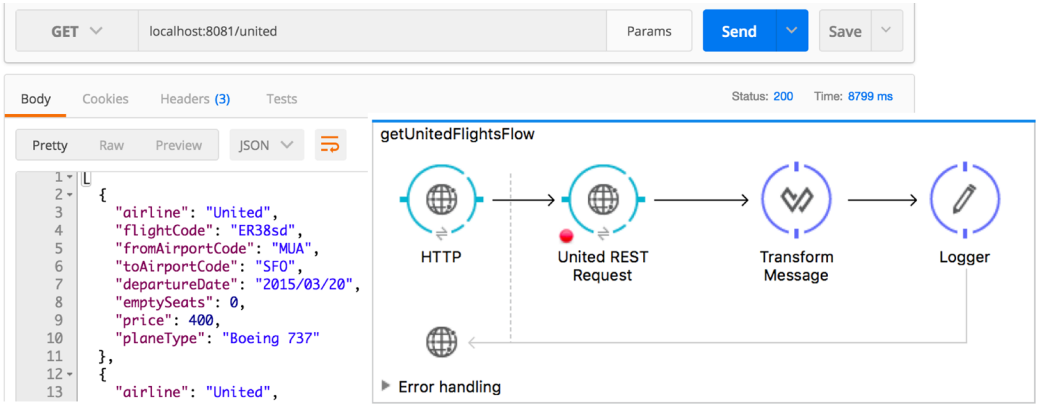
At the end of this module, you should be able to:

- Consume RESTful web services with and without parameters.
- Consume RESTful web services that have RAML definitions.
- Consume SOAP web services.
- Use DataWeave to pass parameters to SOAP web services.

Walkthrough 8-1: Consume a RESTful web service

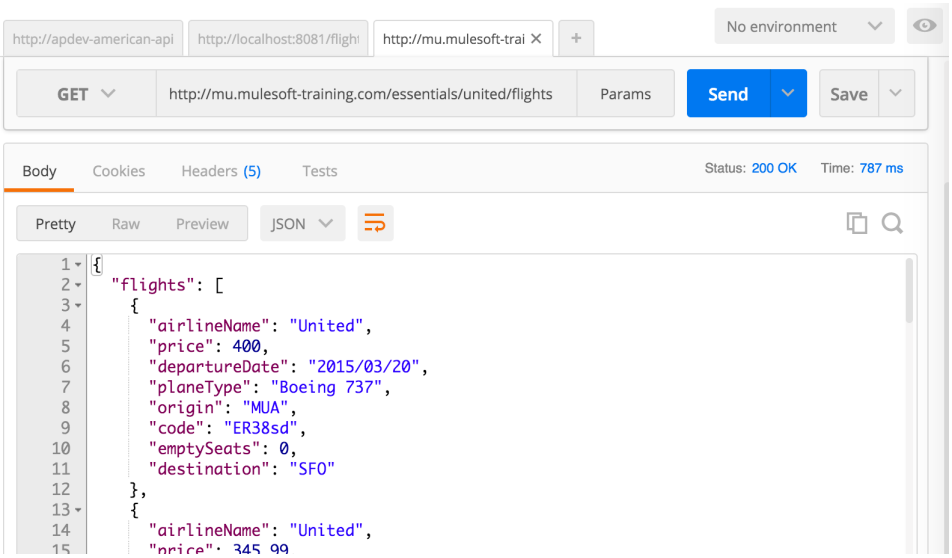
In this walkthrough, you consume a RESTful web service that returns a list of all United flights as JSON. You will:

- Create a new flow to call a RESTful web service.
- Use an HTTP Request endpoint to consume a RESTful web service.
- Use DataWeave to transform the JSON response into JSON specified by an API.

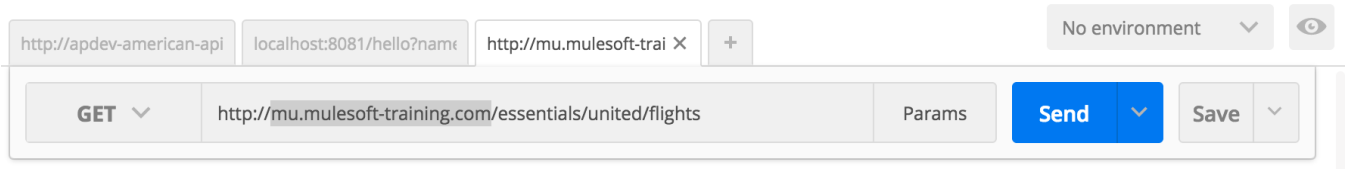


Make a request to the web service

1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Postman, make a new tab.
4. Make a GET request to this URL; you should see JSON data for the United flights as a response.



5. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.
6. Copy the host name from the URL; this should be something like mu.mulesoft-training.com.

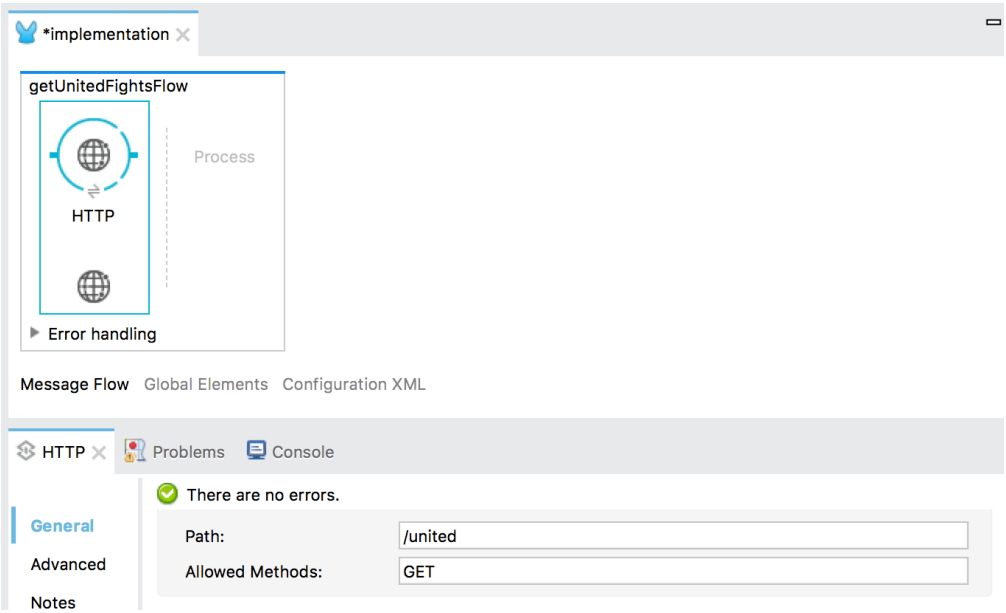


Add a new flow with an HTTP Listener endpoint

7. Return to implementation.xml in Anypoint Studio.
8. Drag out an HTTP connector and drop it in the canvas.
9. Double-click the name of the flow in the canvas and give it a new name of getUnitedFlightsFlow.

Configure the HTTP Listener endpoint

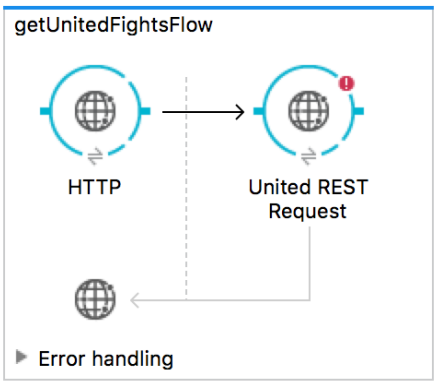
10. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
11. Set the path to /united.
12. Set the allowed methods to GET.



Add an HTTP Request endpoint

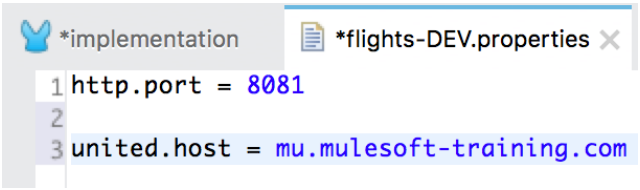
13. Drag out another HTTP connector and drop it into the process section of the flow.

14. Change the HTTP Request endpoint display name to United REST Request.



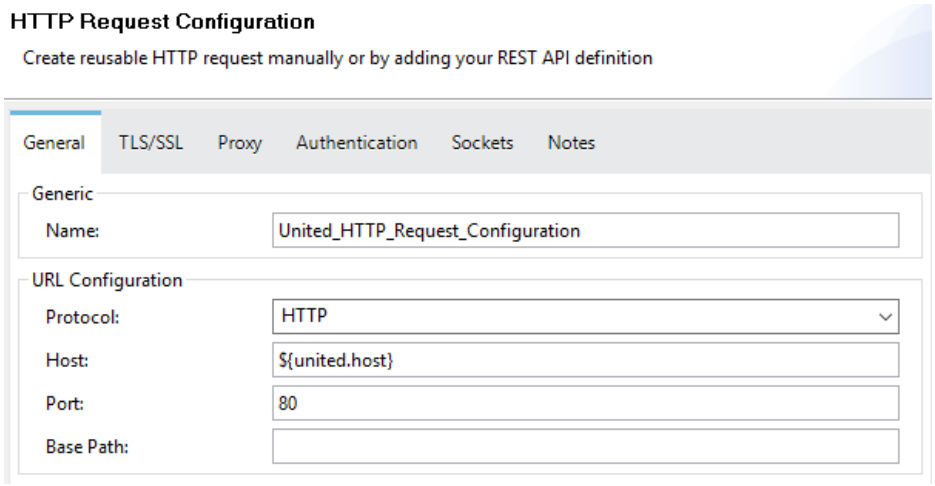
Configure the HTTP Request connector

15. Return to flights-DEV.properties in src/main/resources.
16. Create a property called united.host and set it equal to the value you copied from the URL.



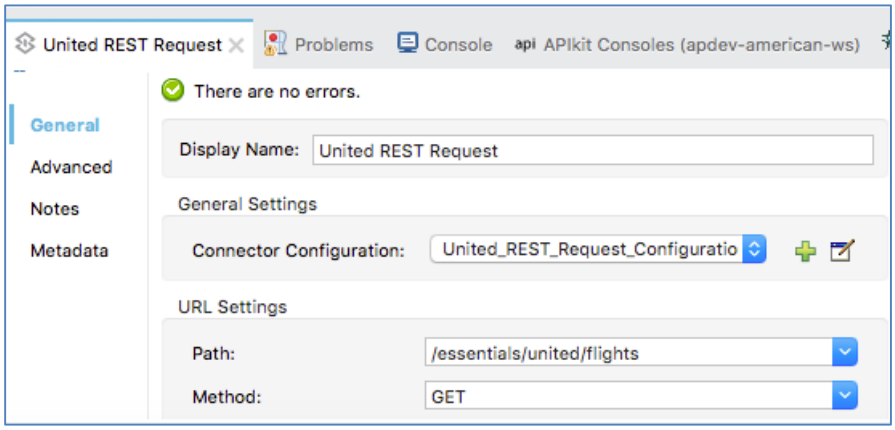
17. Save the file.
18. Return to global.xml.
19. In the Global Elements view, click Create.
20. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

21. In the Global Element Properties dialog box, set the following values and click OK.
    - Name: United_REST_Request_Configuration
    - Host: ${united.host}
    - Port: *80*
    - Base Path: *Leave blank*



Configure the HTTP Request endpoint

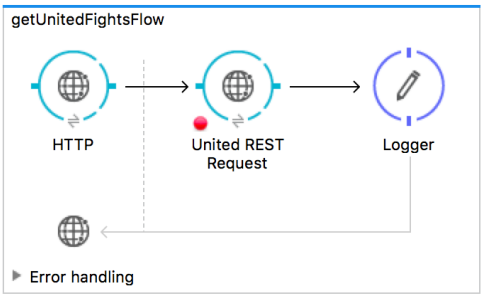
22. Return to implementation.xml.
23. In the United REST Request properties view, set the connector configuration to the existing United_REST_Request_Configuration.
24. Set the path to /essentials/united/flights.
25. Set the method to GET.



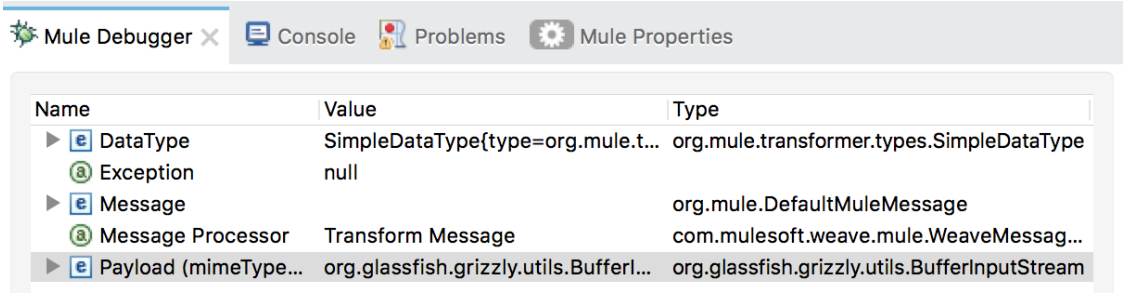
Test the application

26. Make sure the United REST Request endpoint has a breakpoint.

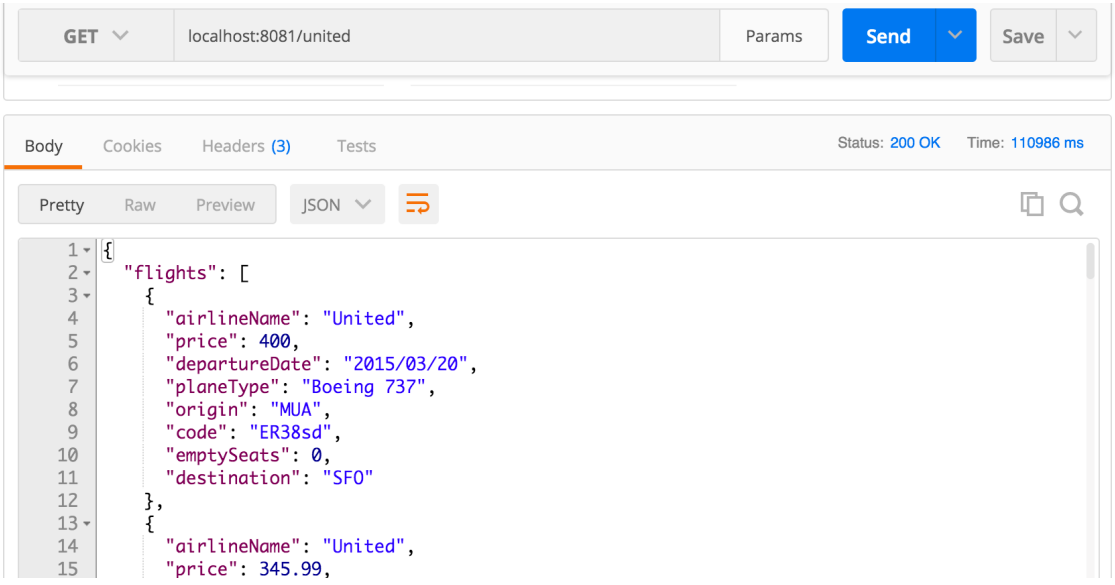
27. Add a Logger after the United REST Request endpoint.



28. Debug the project.
29. In Postman, return to the tab with the localhost request.
30. Make a request to http://localhost:8081/united.
31. In the Mule Debugger, step to the Logger and look at the payload; it should be of type BufferInputStream.

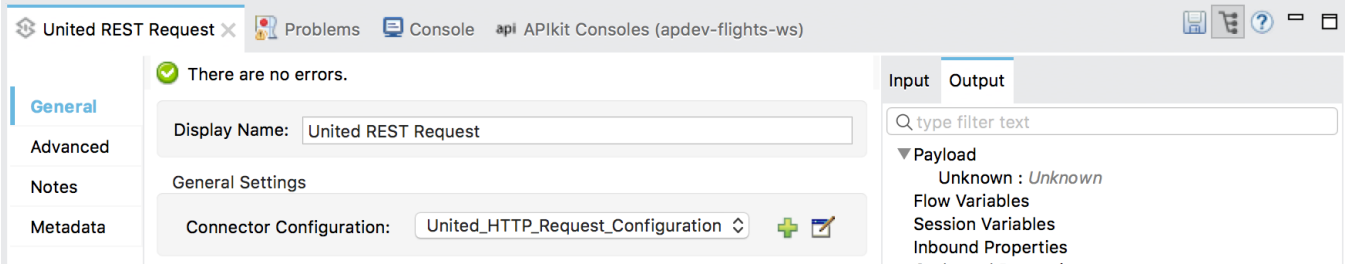


32. Step through the application.
33. Return to Postman; you should see the JSON flight data returned.
34. Examine the data structure of the JSON response.

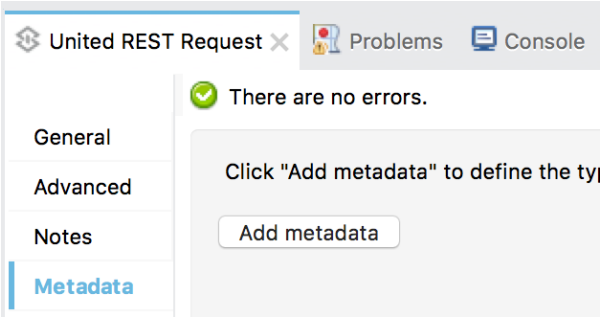


Add metadata for the United REST Request response

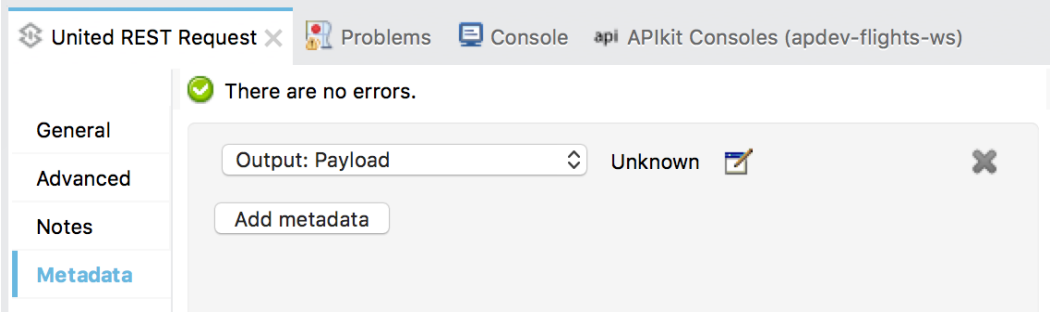
35. Return to Anypoint Studio and switch perspectives.
36. In the Properties view for the United REST Request, click the Output tab; you should see the payload is of type unknown.



37. In the left-side navigation, click the Metadata link.
38. Click the Add metadata button.

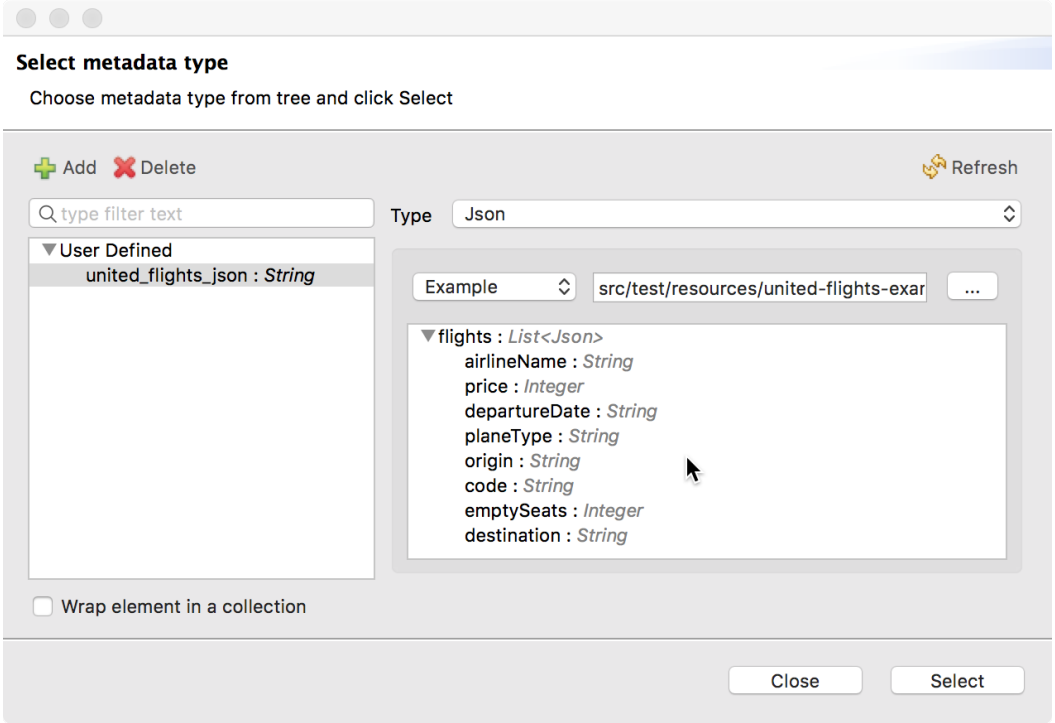


39. Change the drop-down menu to Output: Payload.
40. Click the Edit button.

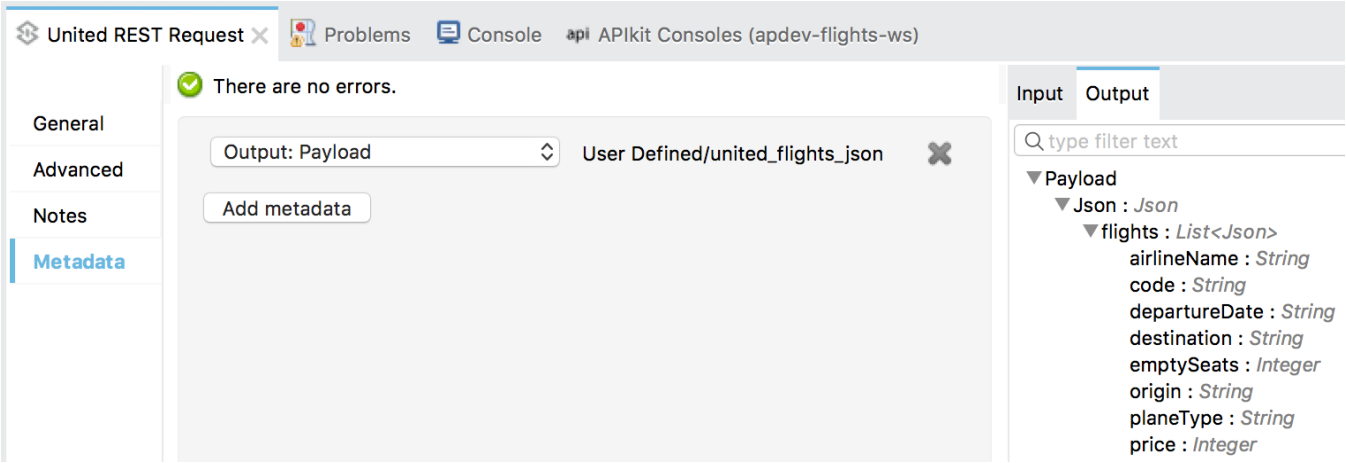


41. In the Select metadata type dialog box, click the Add button.
42. In the Create new type dialog box, set the type id to united_flights_json.
43. Click Create type.
44. In the Select metadata type dialog box, set the type to JSON.
45. Change the Schema selection to Example.
46. Click the browse button and navigate to the projects's src/test/resources folder.

47. Select united_flights-example.json and click Open; you should see the example data for the metadata type.

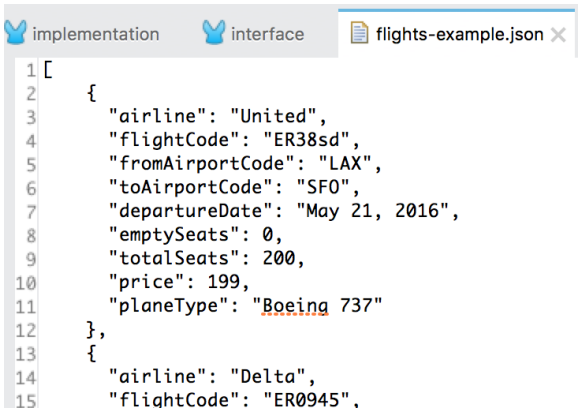


48. Click Select.
49. In the Properties view for the United REST Request, click the Output tab; you should see the payload is of type Json.
50. Expand the Json object and its flights object; you should see the properties of the return objects.



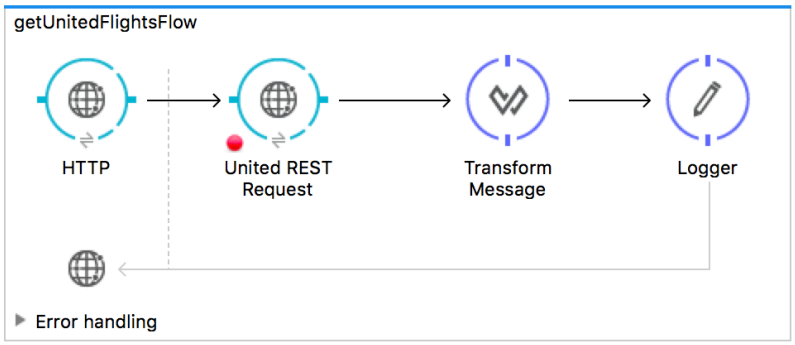
Review the structure for the desired JSON response

51. Open flights-example.json in src/test/resources and review the sample.

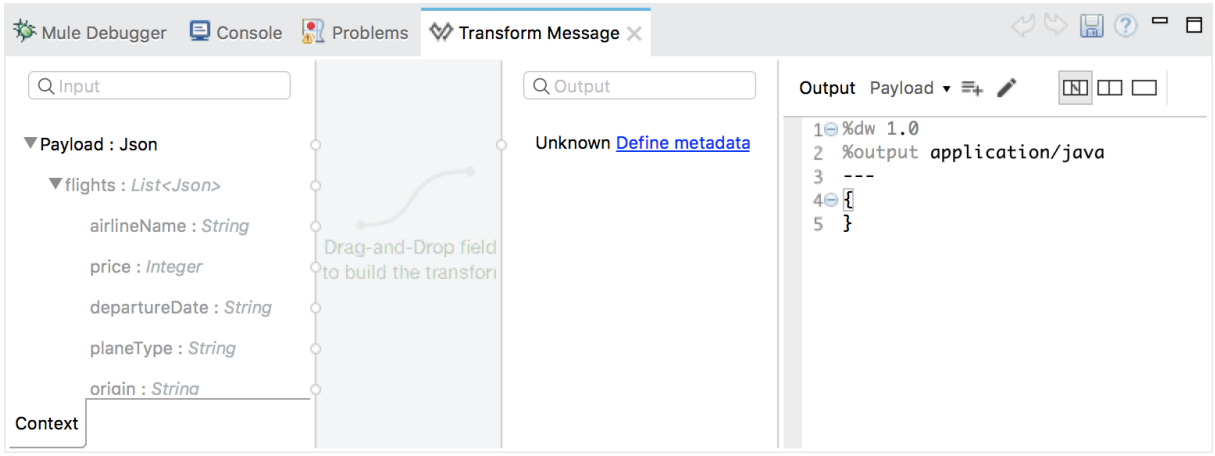


Add a Transform Message component

52. Return to implementation.xml.
53. Add a Transform Message component after the United REST Request endpoint.



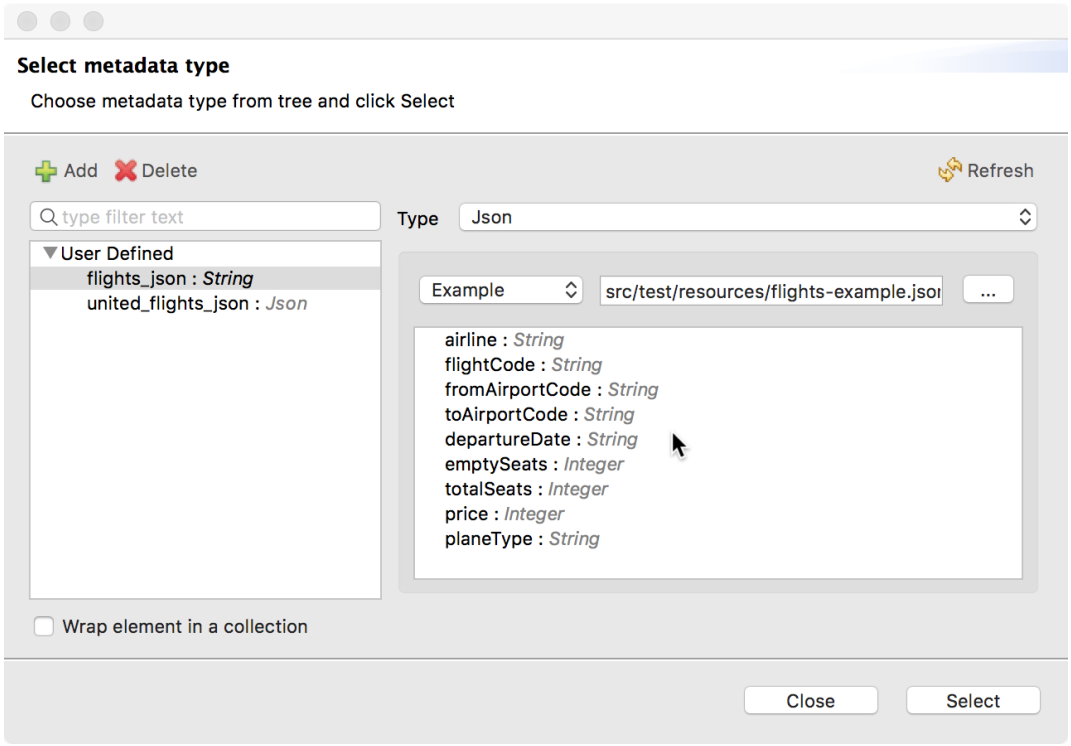
54. Look at the input section; you should see metadata for the output from the United REST Request.



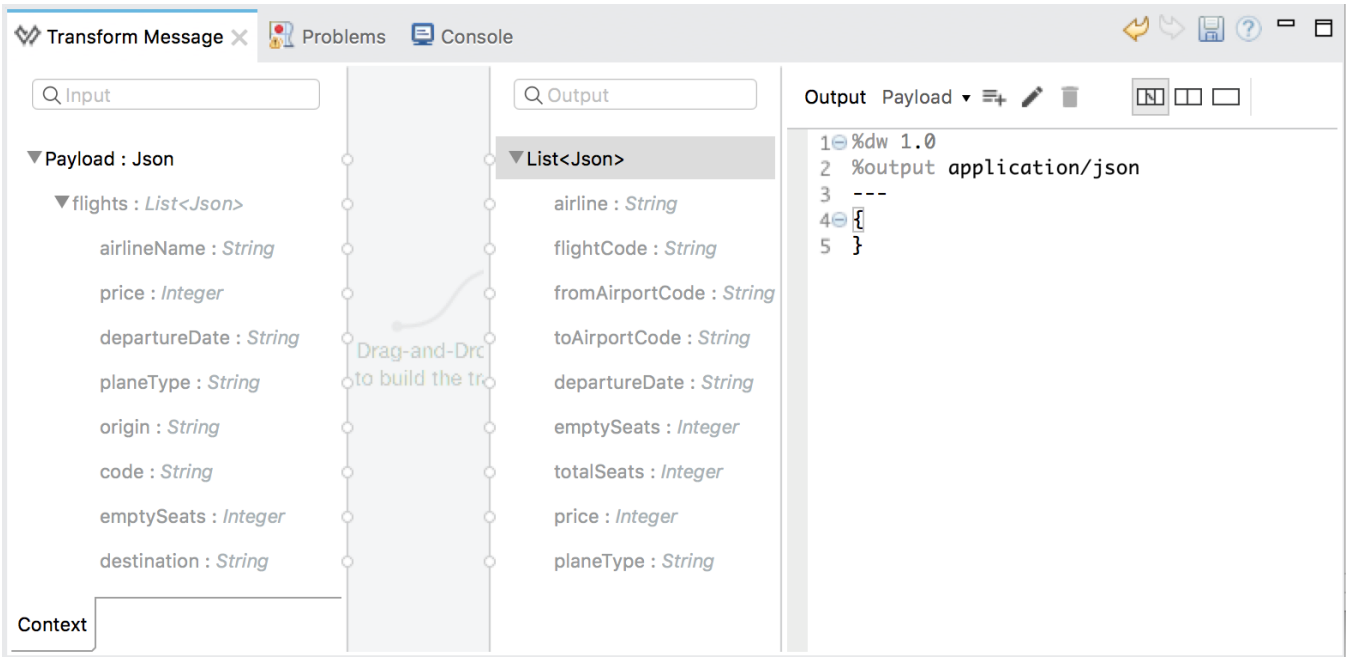
Add output metadata for the transformation

55. In the output section of the Transform Message properties view, click the Define metadata link.
56. In the Select metadata type dialog box, click the Add button.
57. In the Create new type dialog box, set the type id to flights_json.
58. Click Create type.
59. In the Select metadata type dialog box, set the type to JSON.
60. Change the Schema selection to Example.
61. Click the browse button and navigate to the projects's src/test/resources folder.
62. Select flights-example.json and click Open; you should see the example data for the metadata type.

    Note: Be sure to select the JSON file with flights plural, not singular.



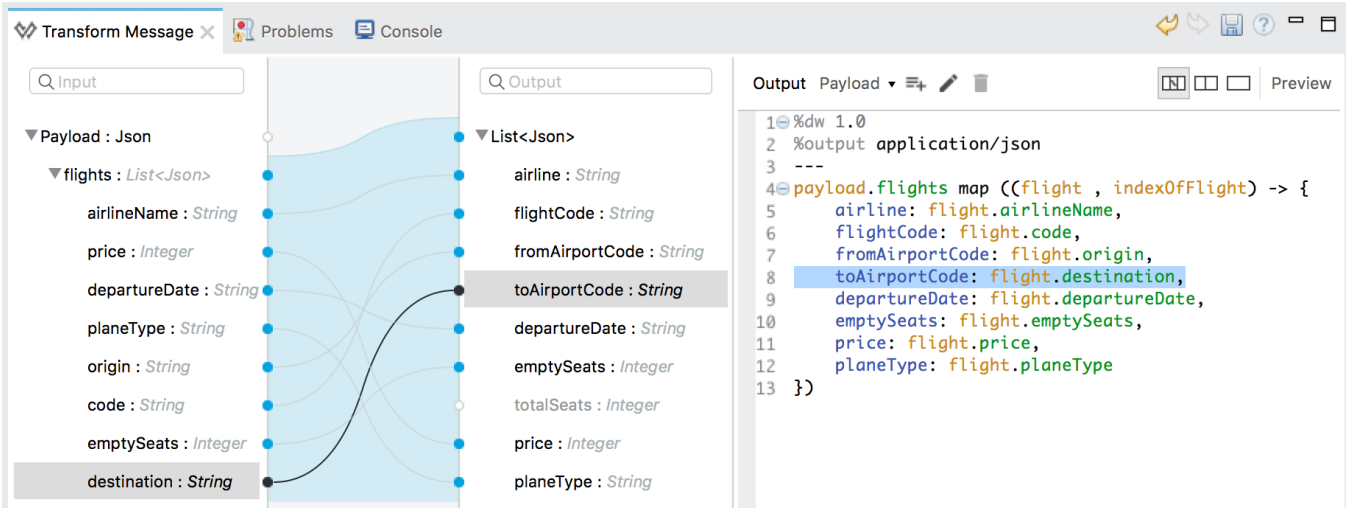
63. Click Select; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

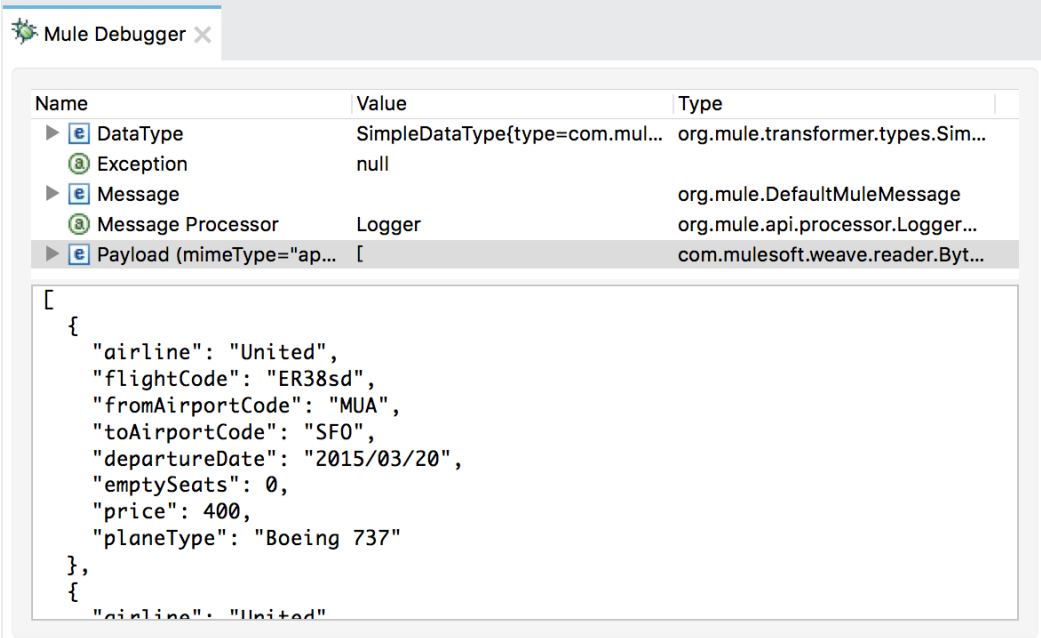
64. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.

Note: There is no input field to map to totalSeats.

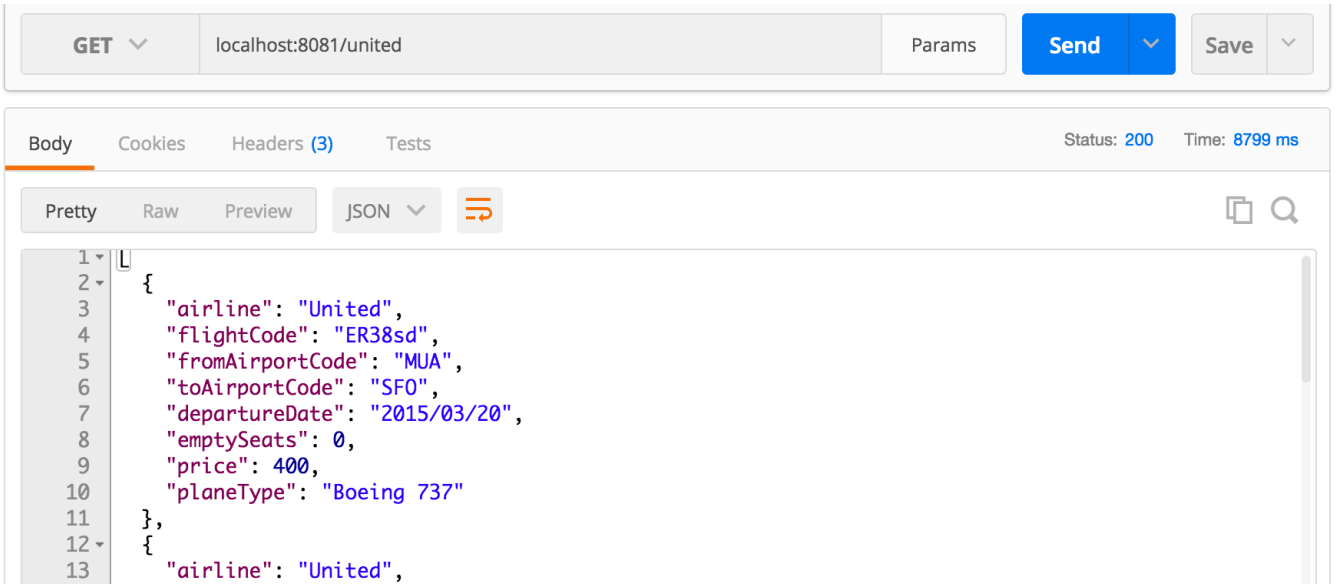


Test the application

65. Debug the project.
66. In Postman, make another request to http://localhost:8081/united.
67. In the Mule Debugger, step to the Logger; you should see the payload is now a DataWeave ByteArraySeekableStream.



68. Step through the application.
69. Return to Postman; you should see the flight data with the different JSON structure.

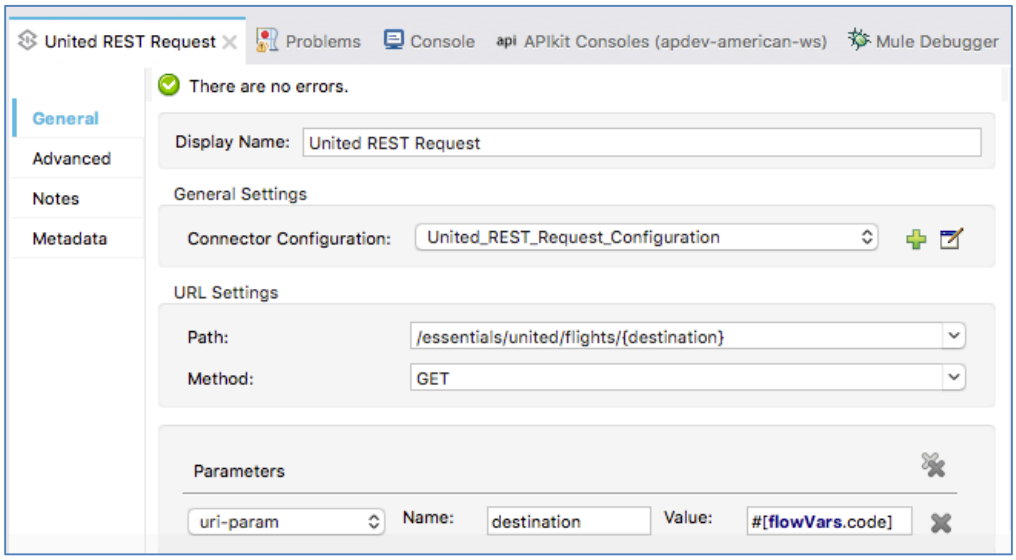


70. Return to Anypoint Studio, stop the project, and switch to the Mule Design perspective.

Walkthrough 8-2: Pass arguments to a RESTful web service

In this walkthrough, you retrieve United flights for a specific destination by setting the destination as a URI parameter. You will:

- Modify the HTTP Request endpoint to use a URI parameter for the destination.
- Set the destination to a static value.
- Create a flow variable to store the value of a query parameter with an airport code value.
- Set the destination to the dynamic value of this flow variable.

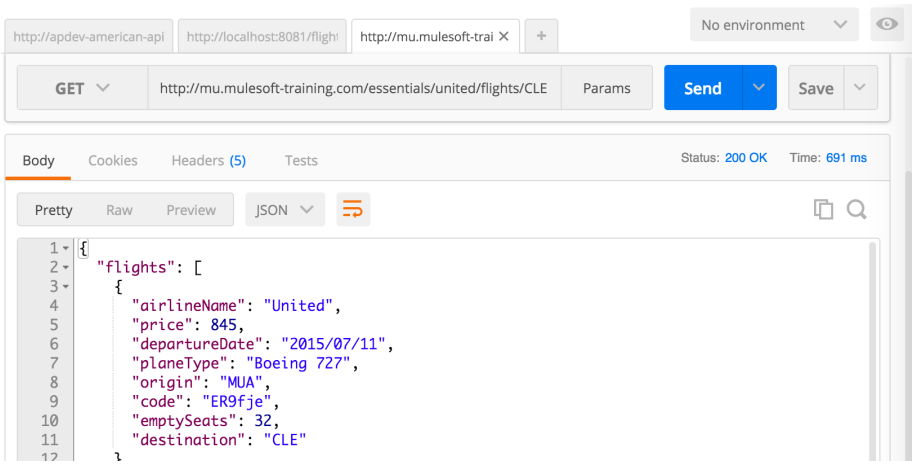


Make a request to the web service specifying a destination

1. Return to Postman and click the third tab, the one with the request to the United web service.
2. In the URL field, add the destination CLE as a URI parameter: http://mu.mulesoft-training.com/essentials/united/flights/CLE.

   Note: The API you are building for flights.raml has a query parameter called code for the destination; this existing United web service uses a URI parameter for the destination.

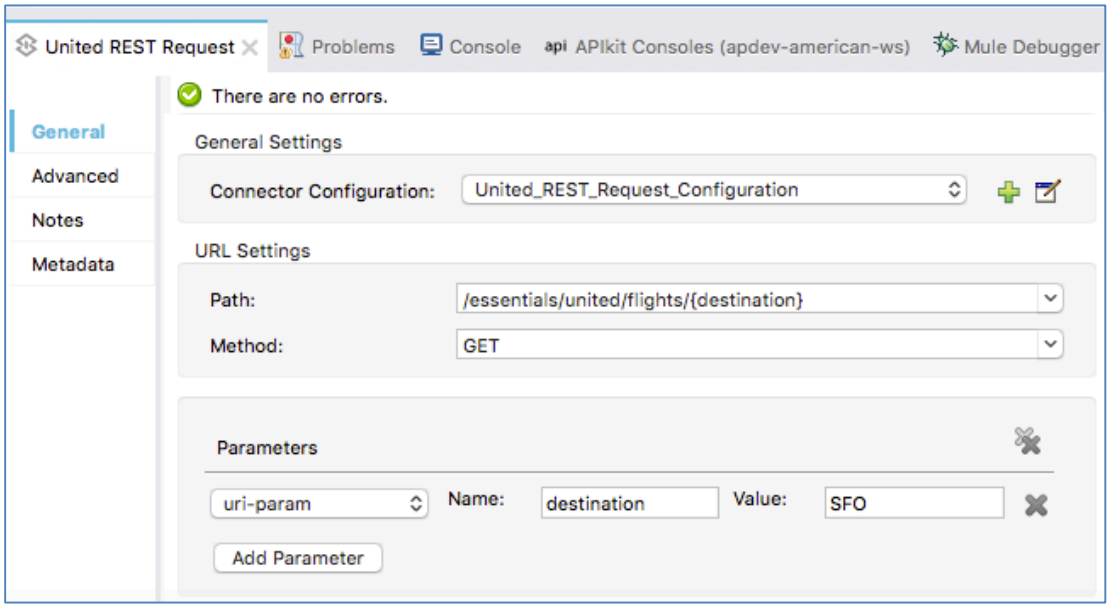
3. Send the request; you should see JSON data for only the flights to CLE.



4. Make additional requests for destinations of LAX, SFO, PDX, or PDF.

Add a URI parameter with a static value

5. Return to implementation.xml in Anypoint Studio.
6. Double-click the United REST Request endpoint.
7. In the Properties view, locate the parameters section; there should be no parameters listed.
8. Change the United REST Request path to /essentials/united/flights/{destination}.
9. Click the Add Parameter button.
10. Select a parameter type of uri-param.
11. Set the name to destination.
12. Set the parameter value to SFO.



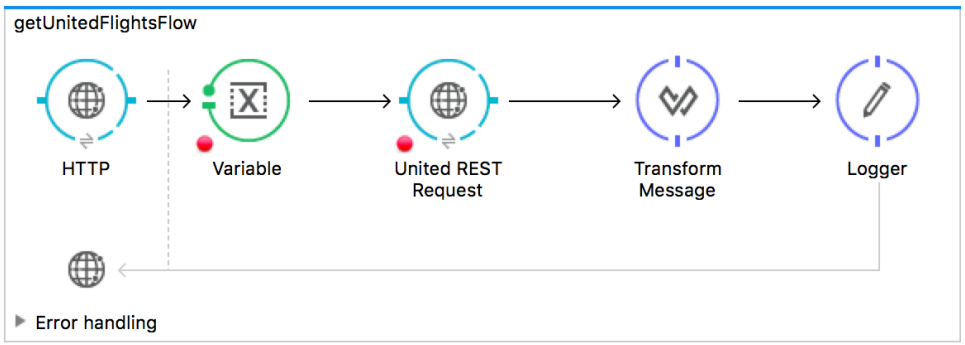
Test the application

13. Run the project.
14. In Postman, return to the tab with the localhost requests.
15. Make a request to http://localhost:8081/united/; you should only get the flights to SFO.
16. Add a query parameter called code and set it to LAX.

17. Send the request; of course, you should still get only flights to SFO.

Create a variable to set the destination airport code

18. Return to Anypoint Studio.
19. Add a Variable transformer before the United REST Request endpoint.

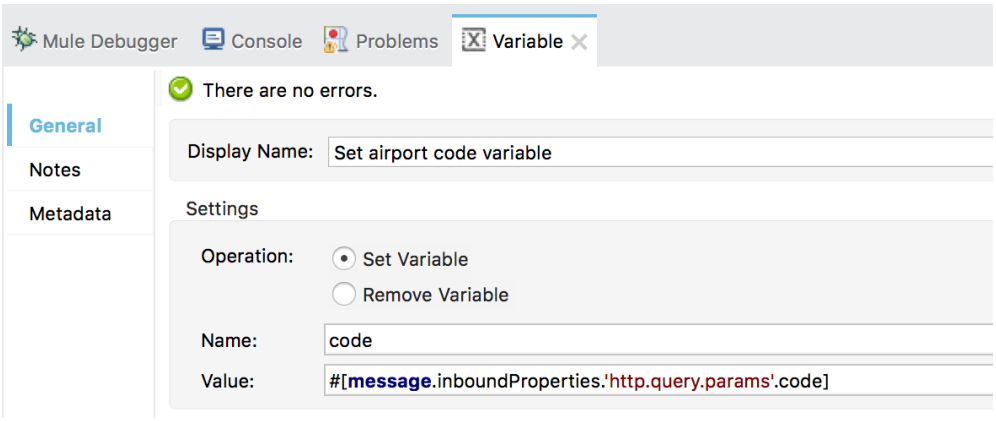


20. In the Variable Properties view, change the display name to Set airport code variable.

Set the operation to Set Variable and the name to code.

21. Set the value to a query parameter called code.

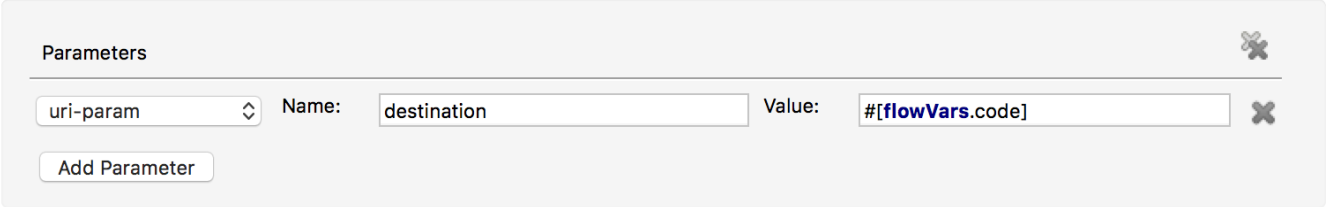
```
#[message.inboundProperties.'http.query.params'.code]
```



Change the REST request URI parameter to a dynamic value

22. In the United REST Request properties view, change the value of the uri-param from SFO to the value of the flow variable containing the airport code.

```
#[flowVars.code]
```

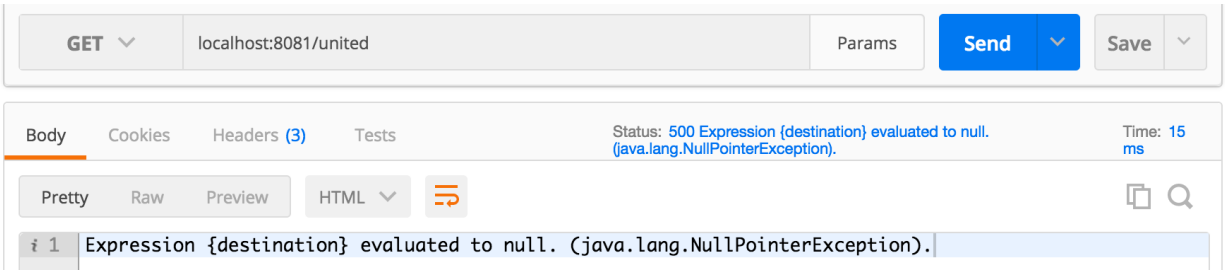


Test the application

23. Save and redeploy the application.
24. In Postman, send the same request again; this time you should only get flights to LAX.

25. Change the code to PDX and make the request; you should now see flights to PDX.

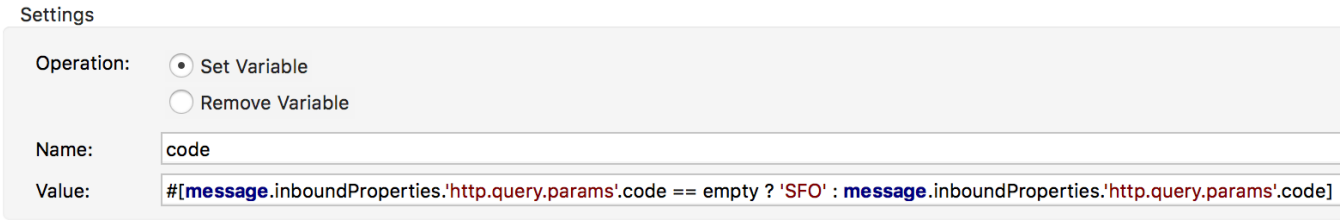
26. Remove the code parameter and make the request; you should get a 500 responses and an exception.



Modify the set airport code flow variable to assign a default value

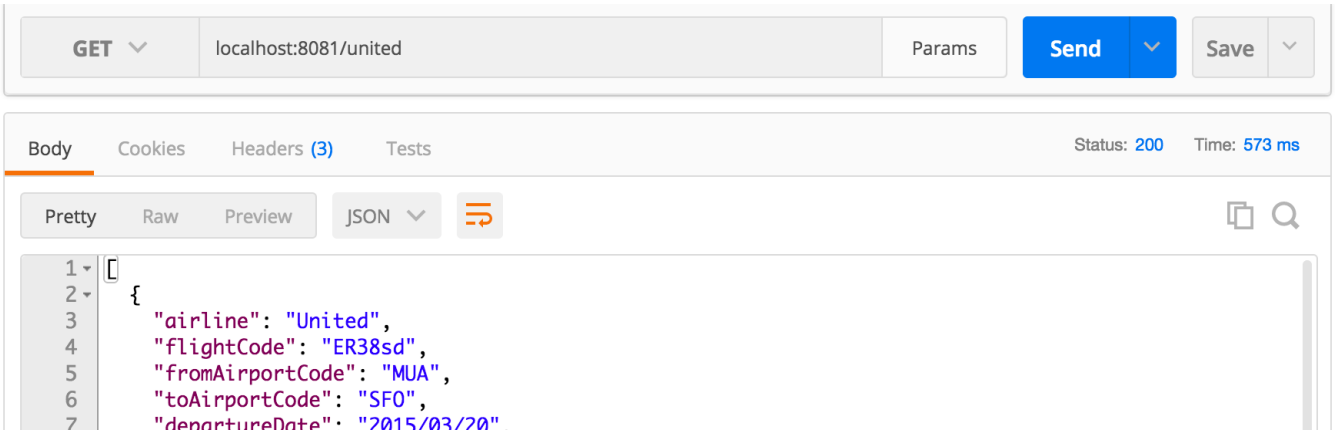
27. Return to Anypoint Studio.
28. Modify the Set variable transformer to use a ternary expression to assign a default value of SFO if no query parameter is passed to the flow.

```
#[(message.inboundProperties.'http.query.params'.code == empty) ?
'SFO' : message.inboundProperties.'http.query.params'.code]
```



Test the application

29. Save and redeploy the application.
30. In Postman, send the same request again with no query parameter; this time you should get flights to SFO instead of an exception.

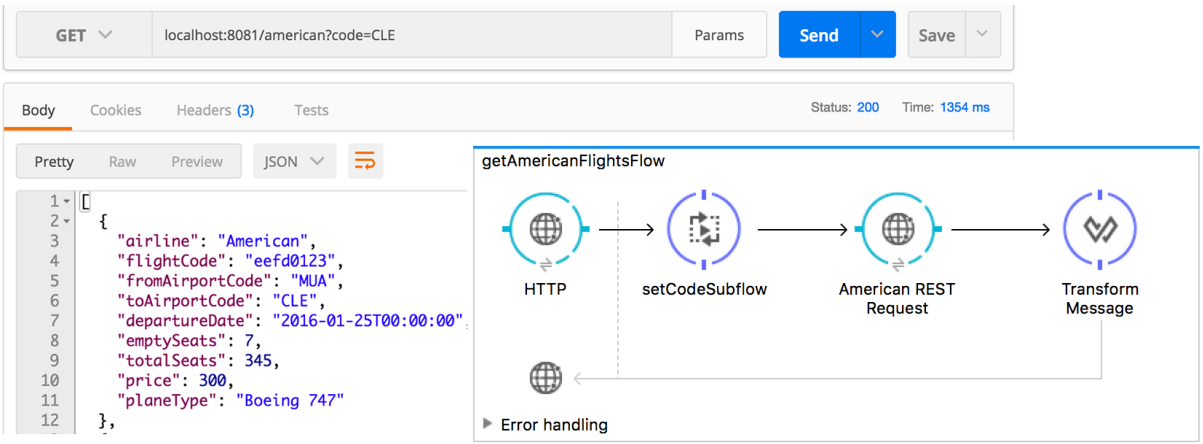


31. Return to Anypoint Studio and stop the project.

Walkthrough 8-3: Consume a RESTful web service that has a RAML definition

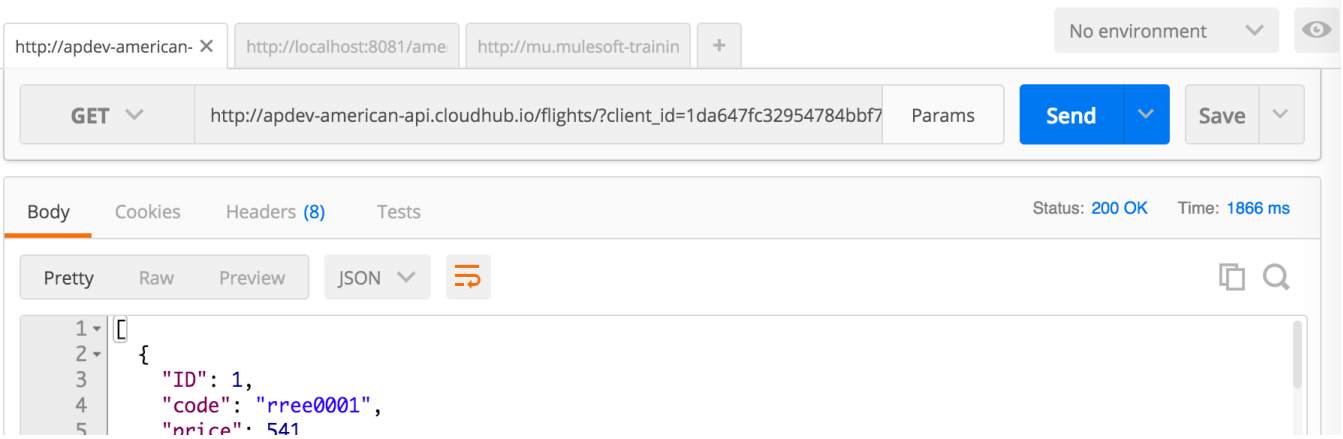
In this walkthrough, you consume the American flights RESTful web service that you built and deployed to the cloud. You will:

- Create a new flow to call a RESTful web service that has a RAML definition.
- Select the web service resource from the list provided by Anypoint Studio from the RAML file.
- Use DataWeave to transform the JSON response into JSON specified by an API.



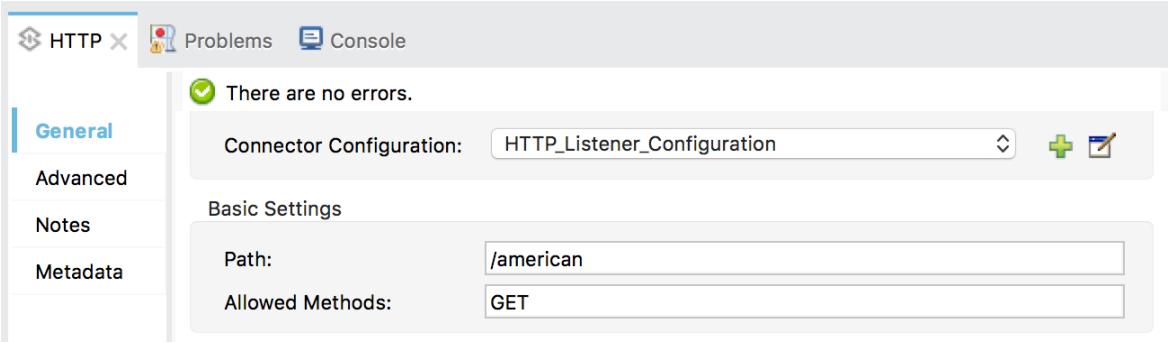
Make a request to the web service

1. In Postman, return to the first tab – the one with the request to your American Flights API http://training-american-api-{lastname}.cloudhub.io/flights and that passes a client_id and client_secret.
2. Send the request; you should still see JSON data for the American flights as a response.



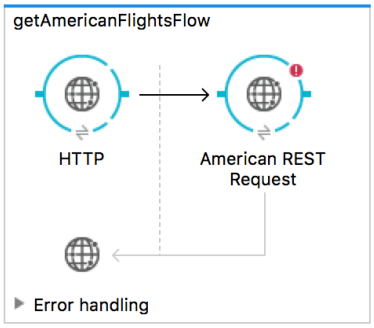
Add a new flow with an HTTP Listener endpoint

3. Return to implementation.xml.
4. Drag out another HTTP connector and drop it in the canvas.
5. Rename the flow to getAmericanFlightsFlow.
6. In the Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
7. Set the path to /american.
8. Set the allowed methods to GET.



Add an HTTP Request endpoint for a web service with a RAML definition

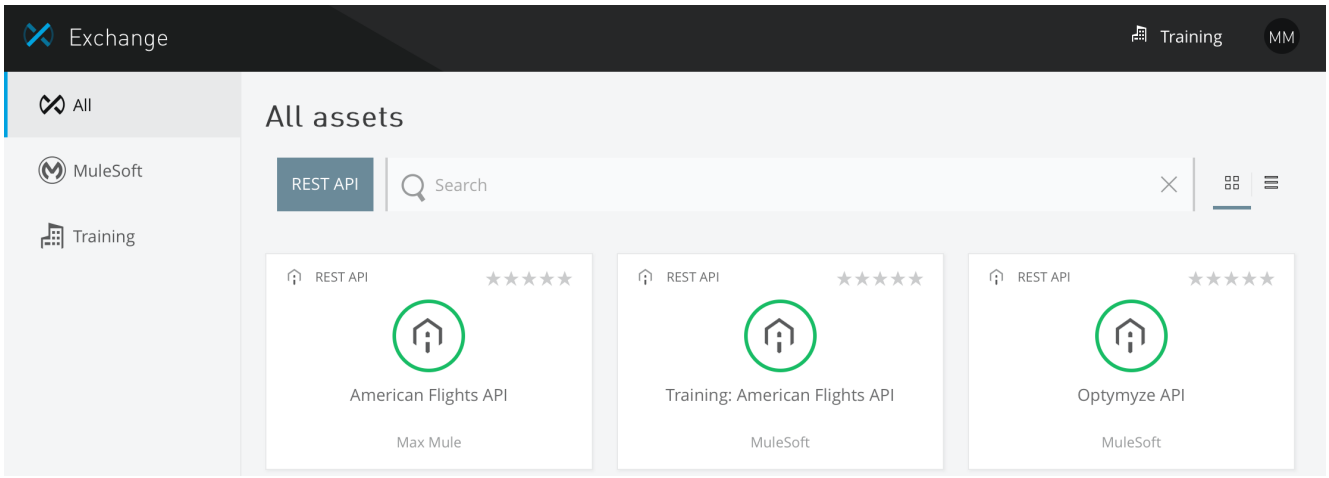
9. Drag out another HTTP connector and drop it in the process section of the new flow.
10. Change the endpoint display name to American REST Request.



Configure the HTTP Request connector

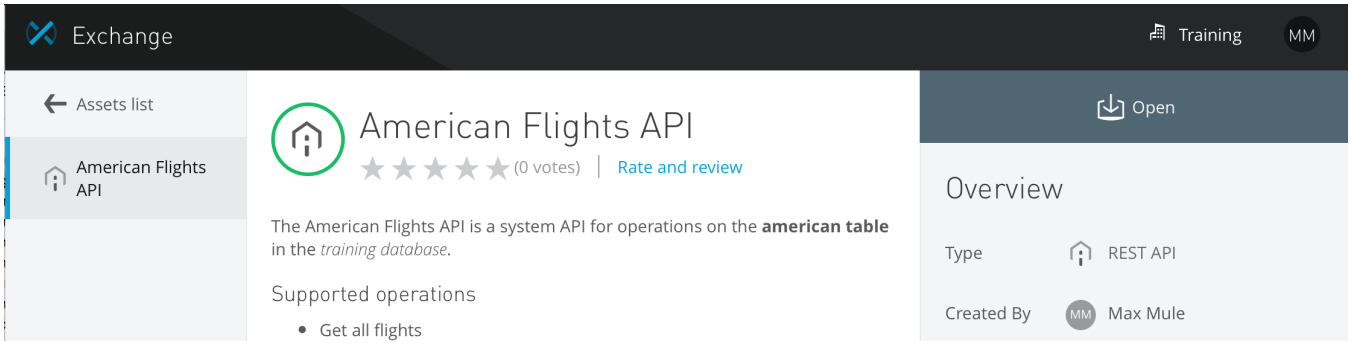
11. Return to global.xml.
12. In the Global Elements view, click Create.
13. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

14. In the Global Element Properties dialog box, change the name to American_HTTP_Request_Configuration.
15. Click the Search in Exchange link next to REST API Location.
16. In the Exchange window that opens, locate your American Flights RAML and click it.

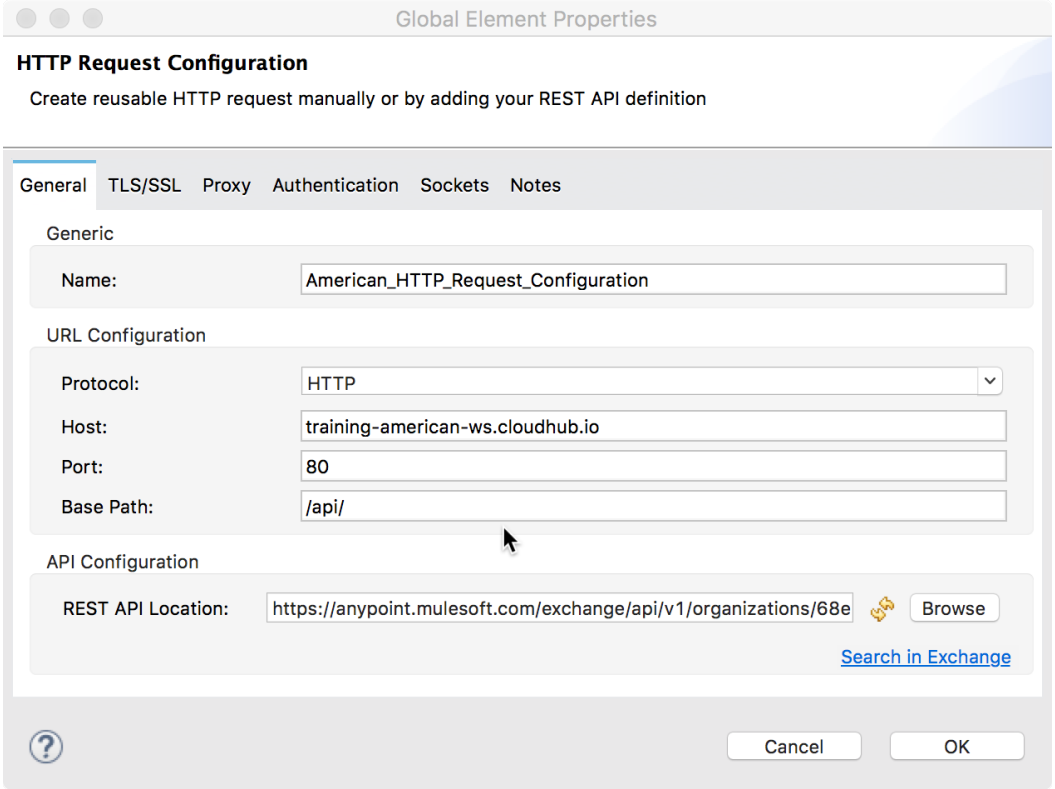


Note: If you do not have a functional API implementation for the American Flights API, you can select the Training: American Flights API instead.

17. Click the Open button; the Exchange window should close.



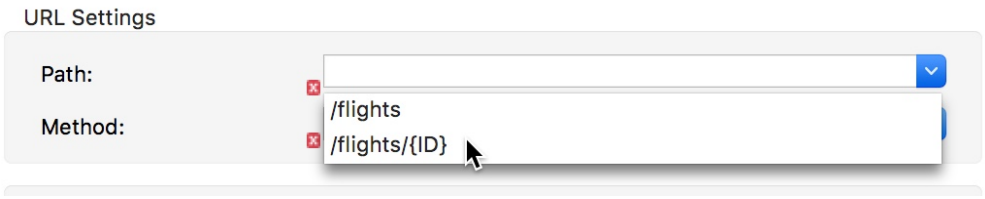
18. Back in the Global Element Properties dialog box, wait for the RAML to be parsed and the host, port, and base path fields to be populated and then click OK.



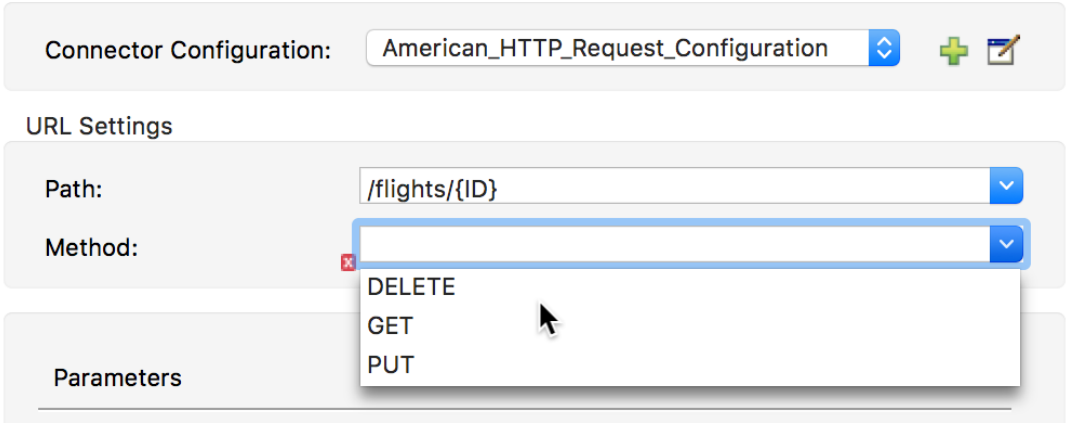
19. Click OK.

Configure the HTTP Request endpoint

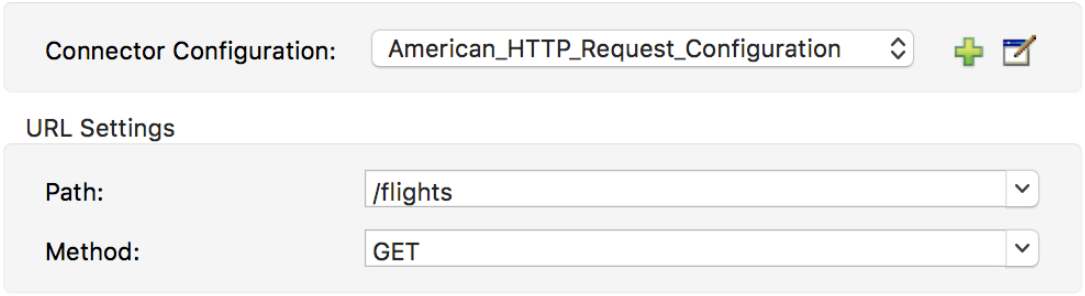
20. Return to implementation.xml.
21. In the American REST Request properties view, set the connector configuration to the existing American_HTTP_Request_Configuration.
22. Click the expand button for the path field; you should see all the available resources for the RESTful web service defined by the RAML file listed – in this case, there are two.
23. Select /flights/{ID}.



24. Look at the method drop-down menu; you should see DELETE, GET, and PUT.

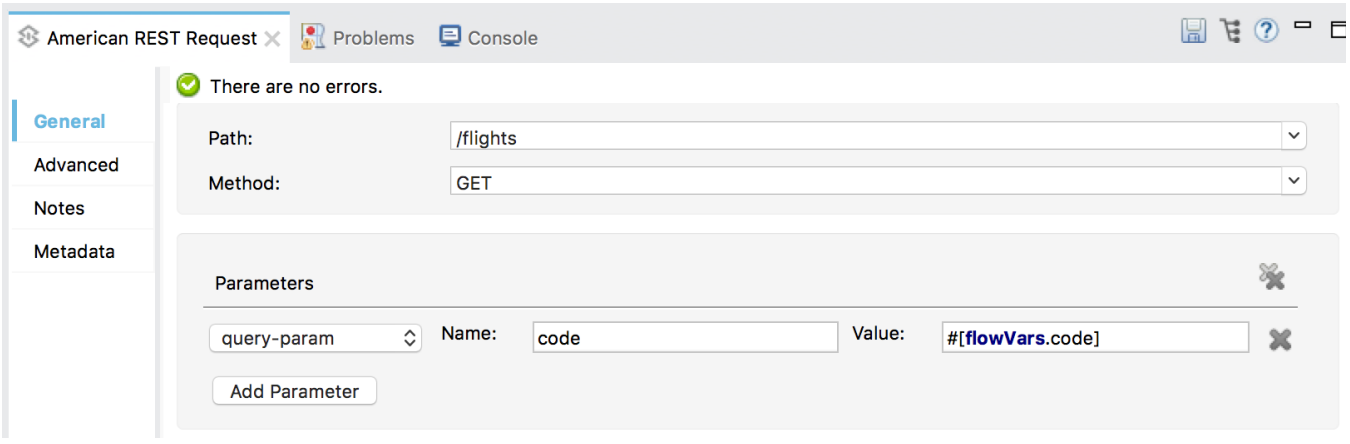


25. Change the path to /flights.
26. Look at the method drop-down menu; you should see GET and POST.
27. Set the method to GET.



28. Scroll down and check to see if a query parameter called code has been added.
29. If the destination parameter was not automatically created, create it.
30. Set the value to a flow variable called code; you will add this to the flow next.

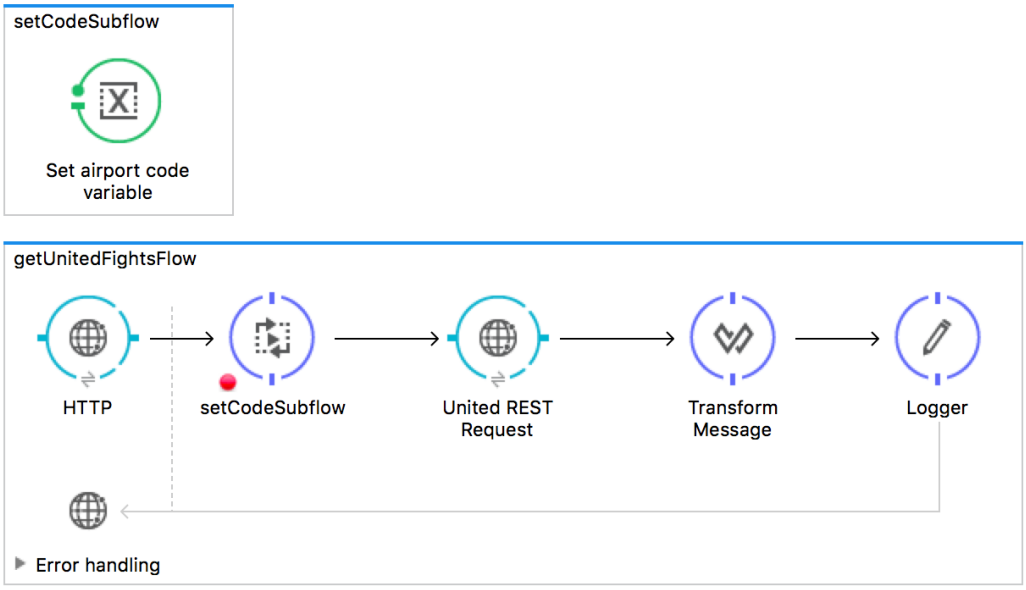
```
#[flowVars.code]
```



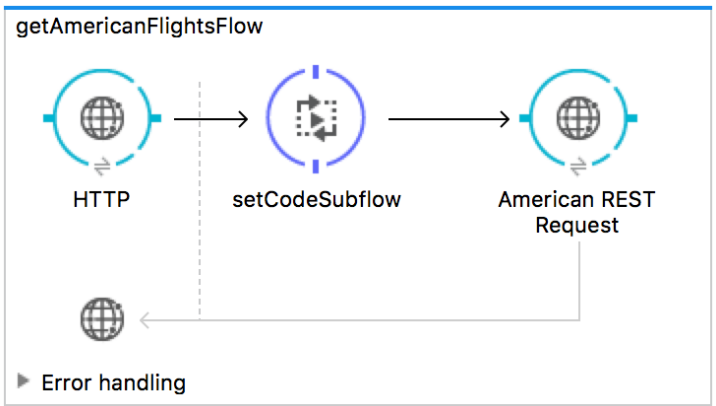
Extract the Set airport code variable processor into a subflow and use it

31. Right-click the Set airport code variable processor in getUnitedFlightsFlow and select Extract to > Sub Flow.
32. In the Extract Flow dialog box, set the flow name to setCodeSubflow and click OK.
33. Double-click the new Flow Reference in getUnitedFlightsFlow; the display name should update.
34. Locate the new subflow.

    Note: If you do not see the subflow in the canvas, switch to the Configuration XML view and then back to the Message Flow view.

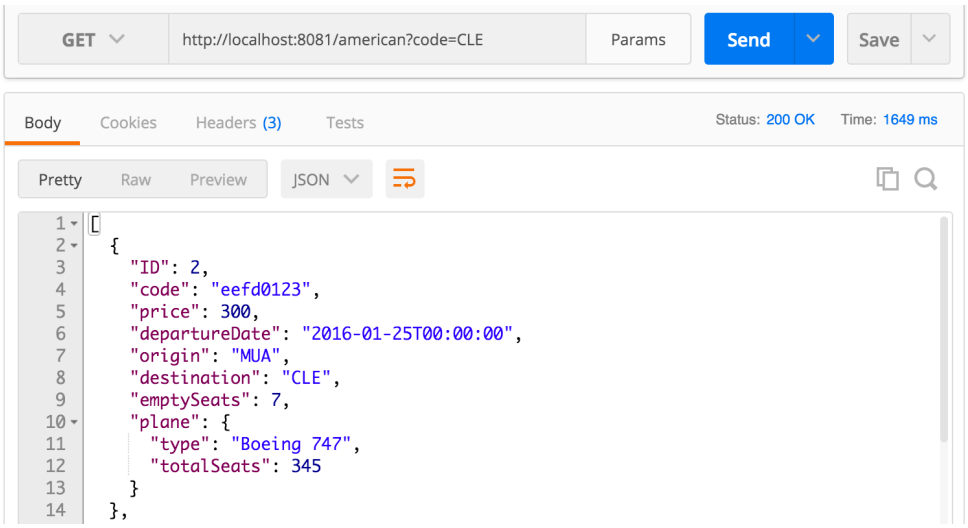


35. Drag a Flow Reference component from the Mule Palette before the American REST Request in getAmericanFlightsFlow.
36. In the Flow Reference properties view, set the flow name to setCodeSubflow.



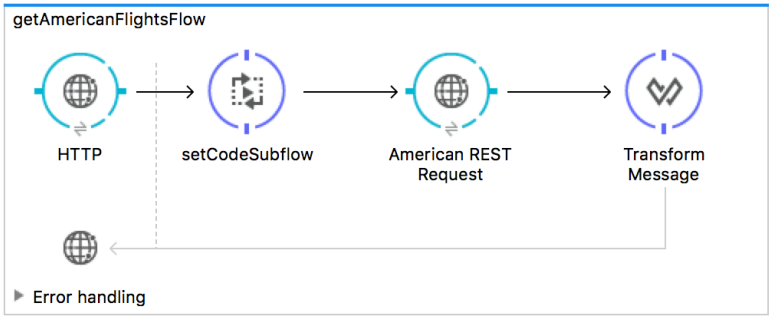
Test the application

37. Save all the files to redeploy the application.
38. In Postman, return to the tab with the local requests.
39. Make a request to http://localhost:8081/american; you should get the American flights to SFO.
40. Add a query parameter called code with a value of CLE.
41. Send the request; you should get just the flights to CLE.



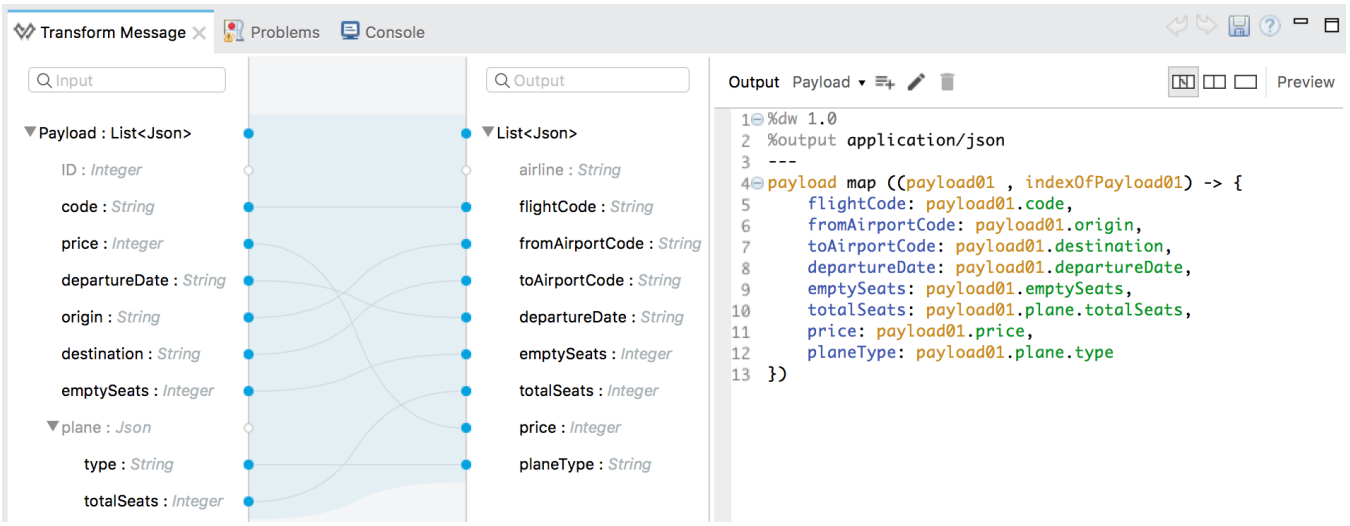
Transform the data

42. Return to getAmericanFlightsFlow.
43. Add a Transform Message component after the American REST Request endpoint.

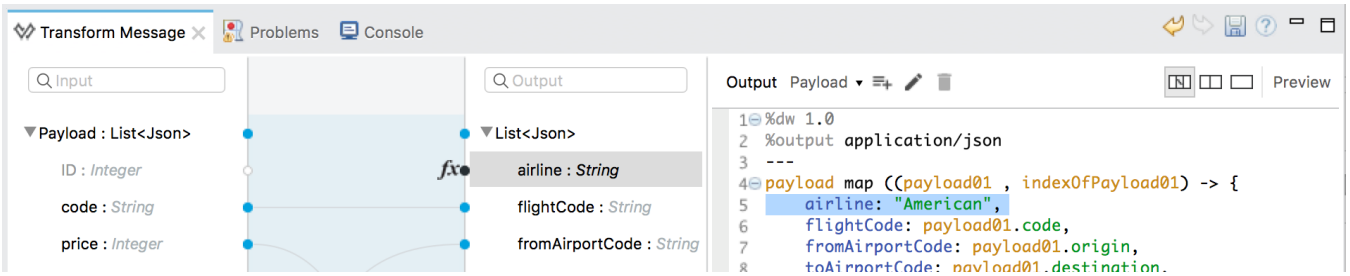


44. Double-click the Transform Message component.
45. Look at the input section; you should see metadata already defined.
46. In the output section of the Transform Message properties view, click the Define metadata link.
47. In the Define metadata type dialog box, select flights_json.
48. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

49. Map fields (except ID and airline) by dragging them from the input section and dropping them on the corresponding field in the output section.

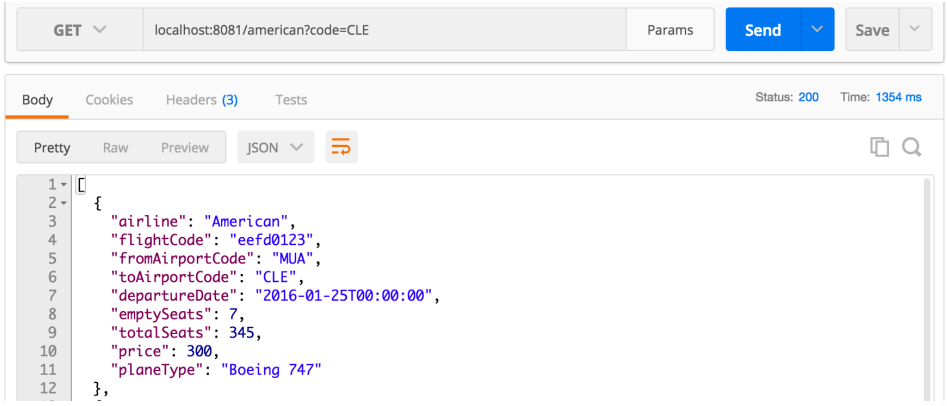


50. Double-click the airline field in the output section.
51. In the generated DataWeave expression, change the airline value from null to "American".



Test the application

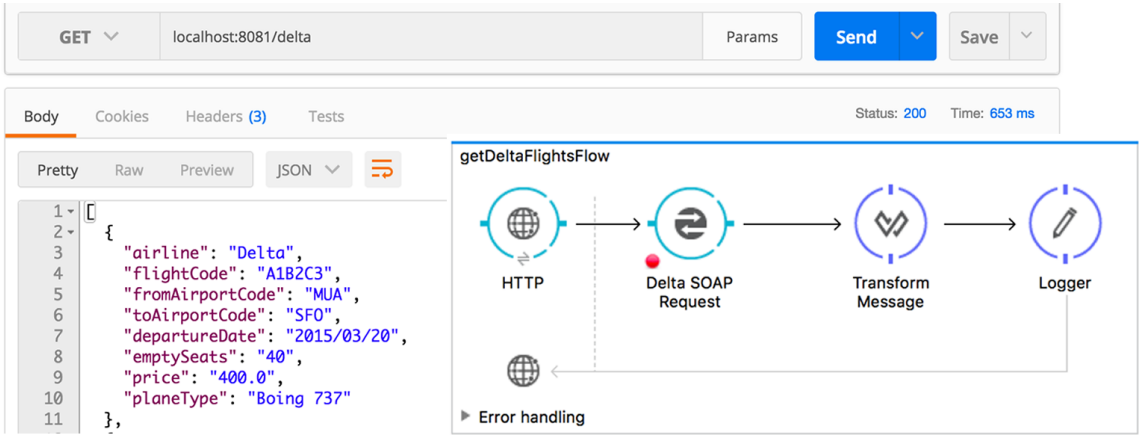
52. Save to redeploy the project.
53. In Postman, make a request to http://localhost:8081/american; you should see all the flight data as JSON again but now with a different structure.



Walkthrough 8-4: Consume a SOAP web service

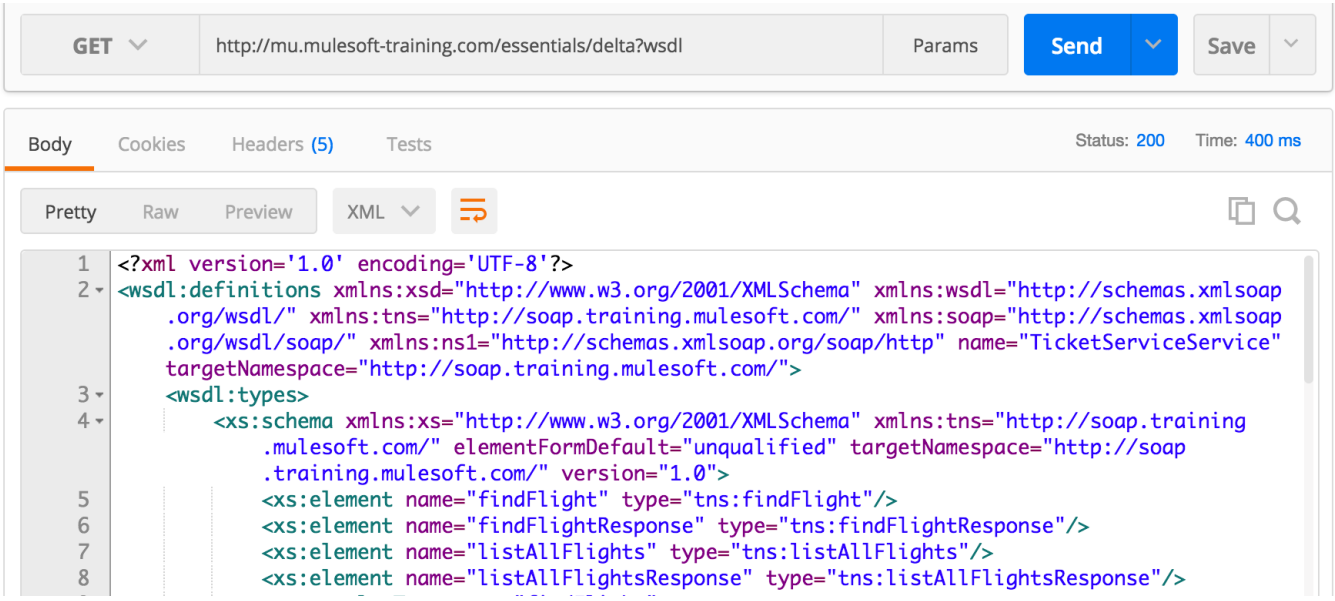
In this walkthrough, you consume a SOAP web service that returns a list of all Delta flights as XML. You will:

- Create a new flow to call a SOAP web service.
- Use a Web Service Consumer endpoint to consume a SOAP web service for Delta flight data.
- Use DataWeave to transform the XML response into JSON specified by the MUA Flights API.



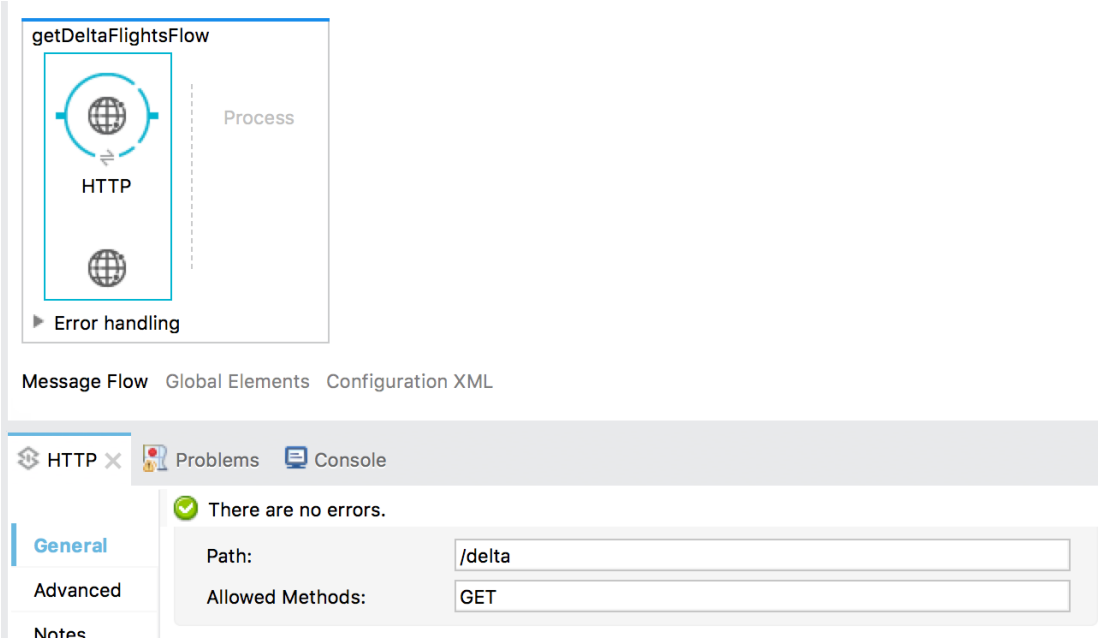
Browse the WSDL

1. Return to the course snippets.txt file and copy the WSDL URL for the Delta SOAP web service.
2. In Postman, return to the third tab, the one with the mulesoft-training request.
3. Paste the URL and send the request; you should see the web service WSDL returned.
4. Browse the WSDL; you should find references to operations listAllFlights and findFlight.



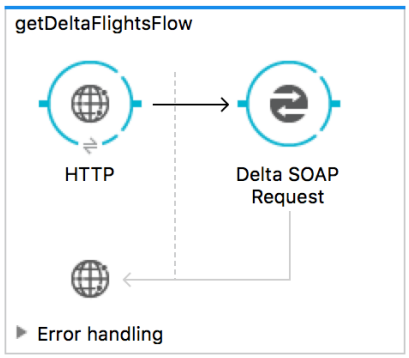
Create a new flow with an HTTP Listener connector endpoint

5. Return to Anypoint Studio.
6. Drag out another HTTP connector and drop it in the canvas after the existing flows.
7. Rename the flow to getDeltaFlightsFlow.
8. In the Properties view for the endpoint, set the connector configuration to the existing HTTP_Listener_Configuration.
9. Set the path to /delta.
10. Set the allowed methods to GET.



Add a Web Service Consumer connector endpoint

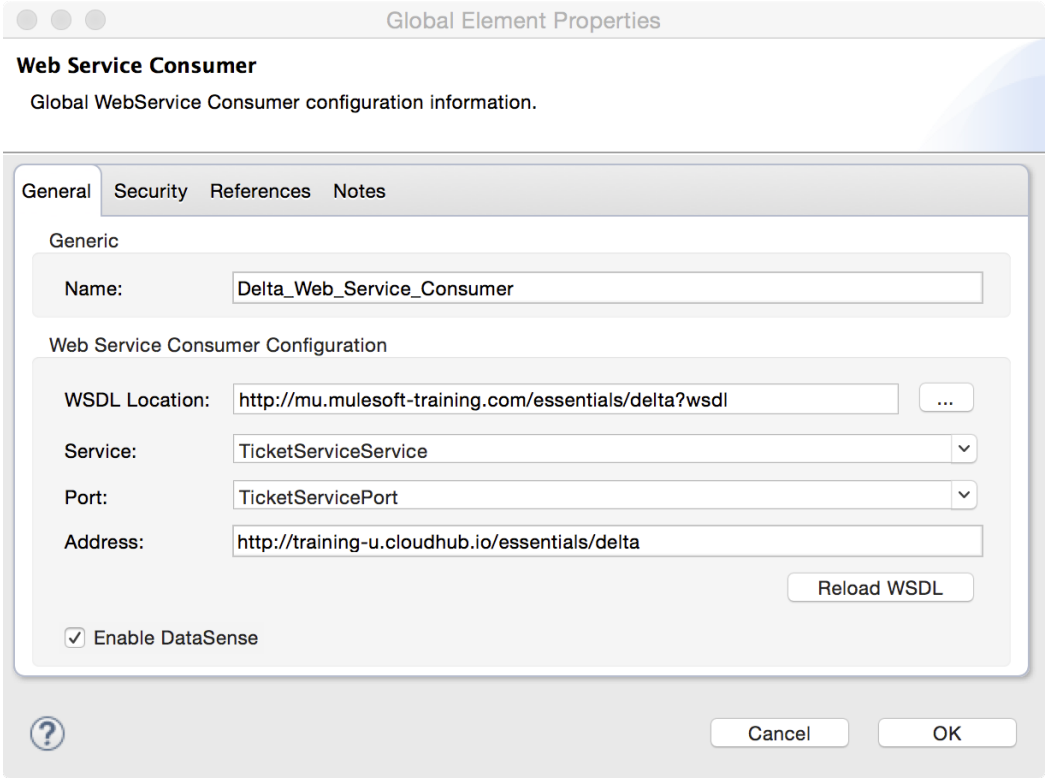
11. Drag out a Web Service Consumer connector and drop it in the process section of the flow.
12. Change its display name to Delta SOAP Request.



Configure the Web Service Consumer connector

13. Return to global.xml.
14. Click Create.
15. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer and click OK.
16. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer.
17. Set the WSDL location to the value you copied from the course snippets.txt file.
18. Wait for the service, port, and address fields to populate.

    Note: If the fields do not populate, click the Reload WSDL button. If the fields still do not auto-populate, select TicketServiceService from the service drop-down menu and select TicketServicePort from the port drop-down menu.

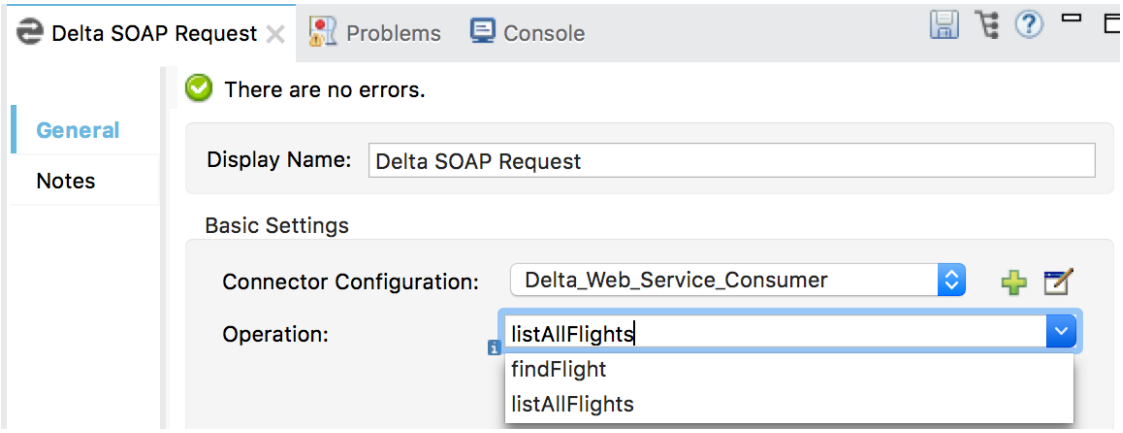


19. Click OK.

Configure the Web Service Consumer endpoint

20. Return to implementation.xml.

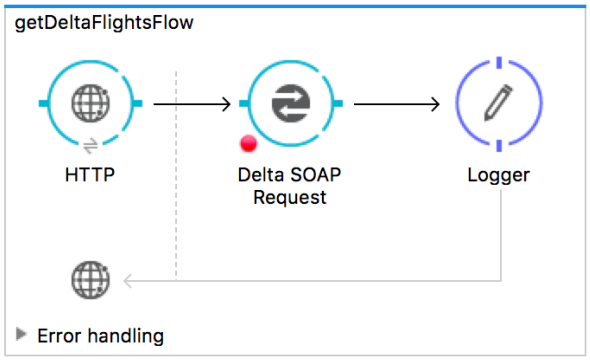
21. In the Delta SOAP Request properties view, set the connector configuration to the existing Delta_Web_Service_Consumer.
22. Click the operation drop-down menu button; you should see all of the web service operations listed.



23. Select the listAllFlights operation.

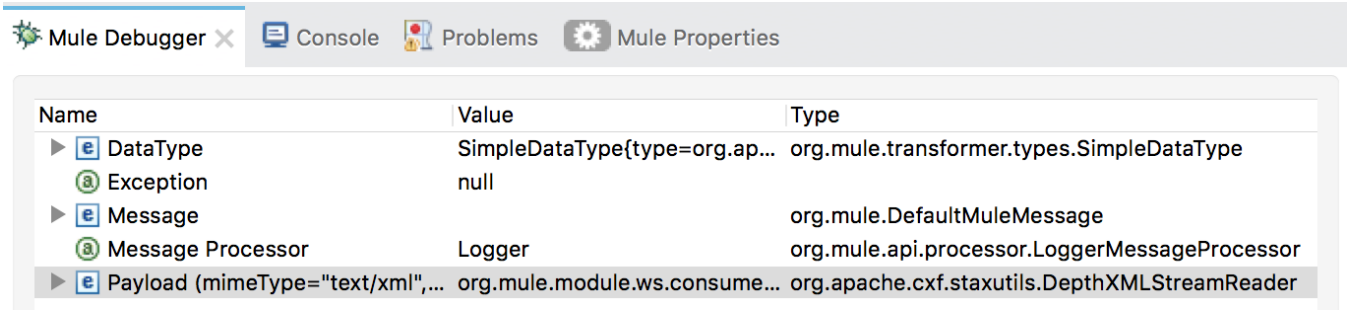
Test the application

24. Add a breakpoint to the Delta SOAP Request endpoint.
25. Add a Logger to the end of the flow.

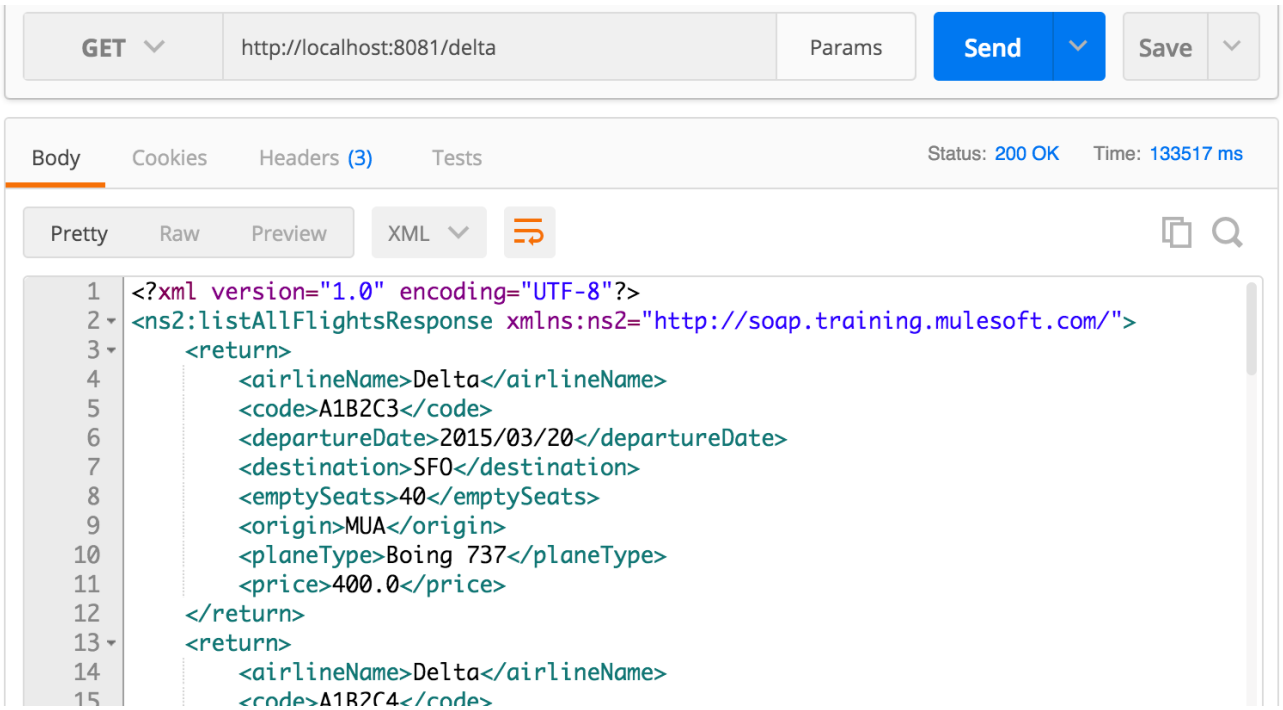


26. Debug the project.
27. In Postman, return to the middle tab – the one with the localhost requests.
28. Make a request to http://localhost:8081/delta.

29. In the Mule Debugger, step to the Logger and look at the payload; it should be of type DepthXMLStreamReader.



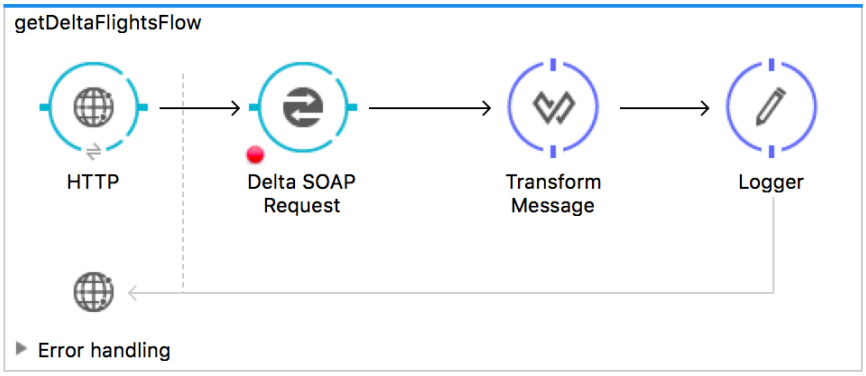
30. Step through the application.
31. Return to Postman; you should see the XML flight data returned.



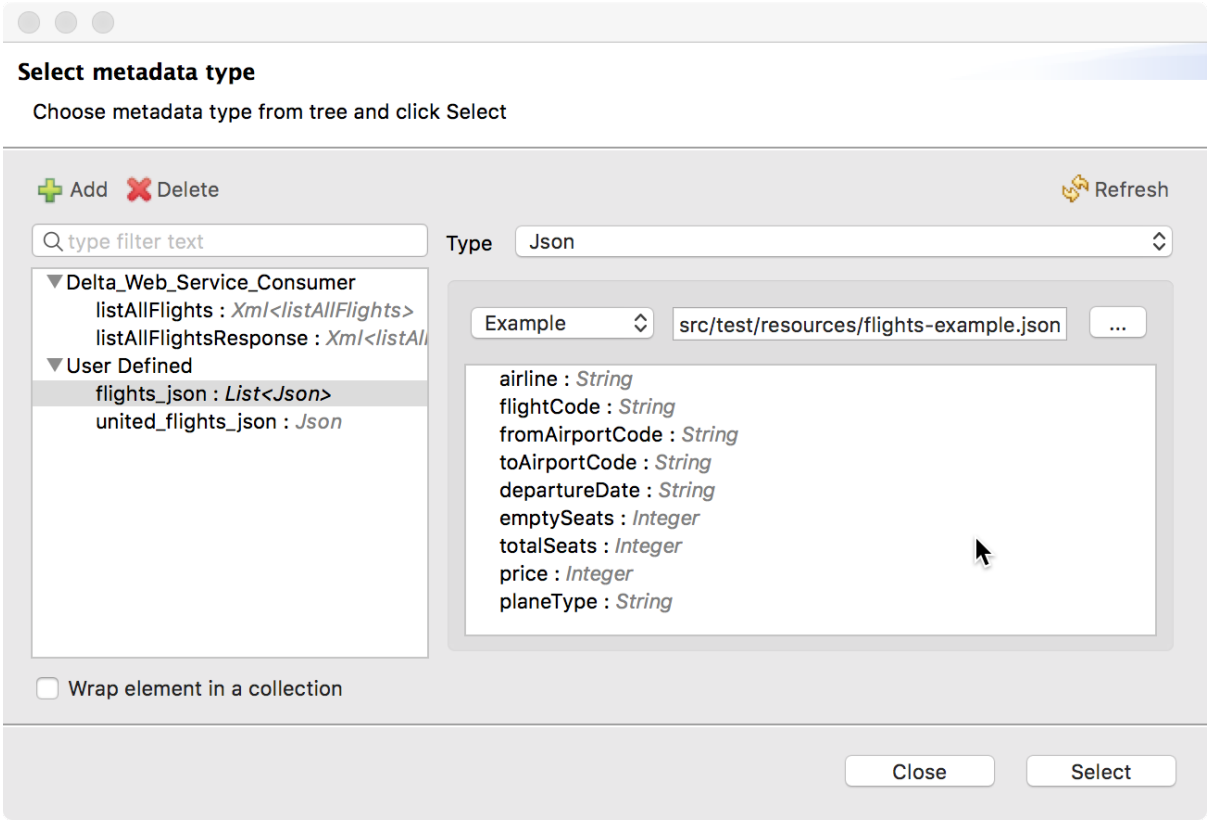
Transform the data

32. Return to Anypoint Studio, stop the project, and switch to the Mule Design perspective.

33. In getDeltaFlightsFlow, add a Transform Message component after the Delta SOAP Request endpoint.

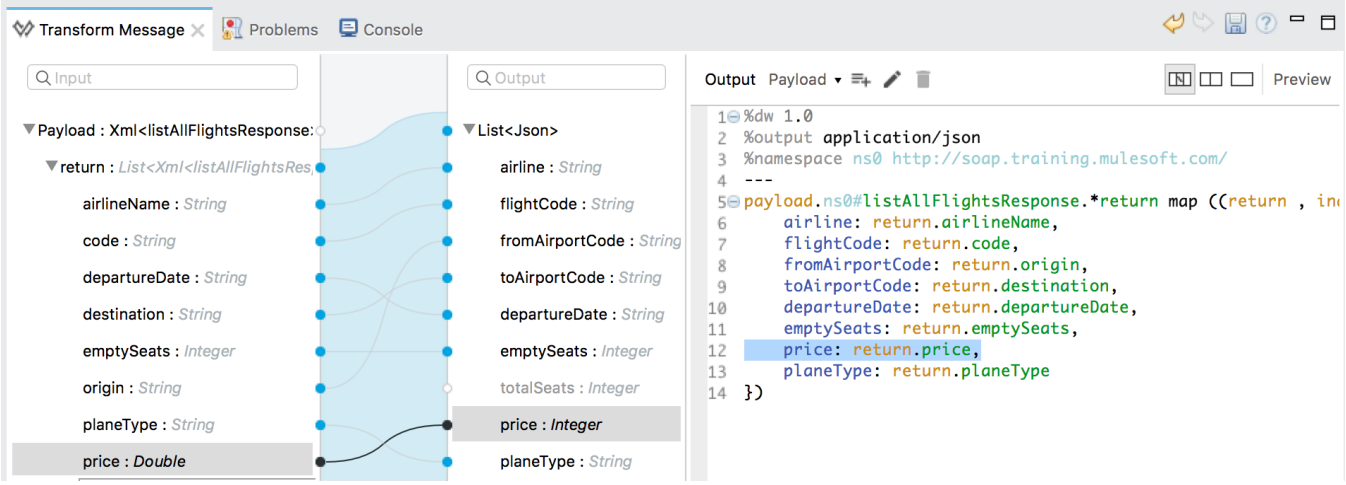


34. Look at the input section of the Transform Message properties view; you should see metadata already defined.
35. In the output section of the Transform Message properties view, click the Define metadata link.
36. In the Define metadata type dialog box, select User Defined > flights_json.



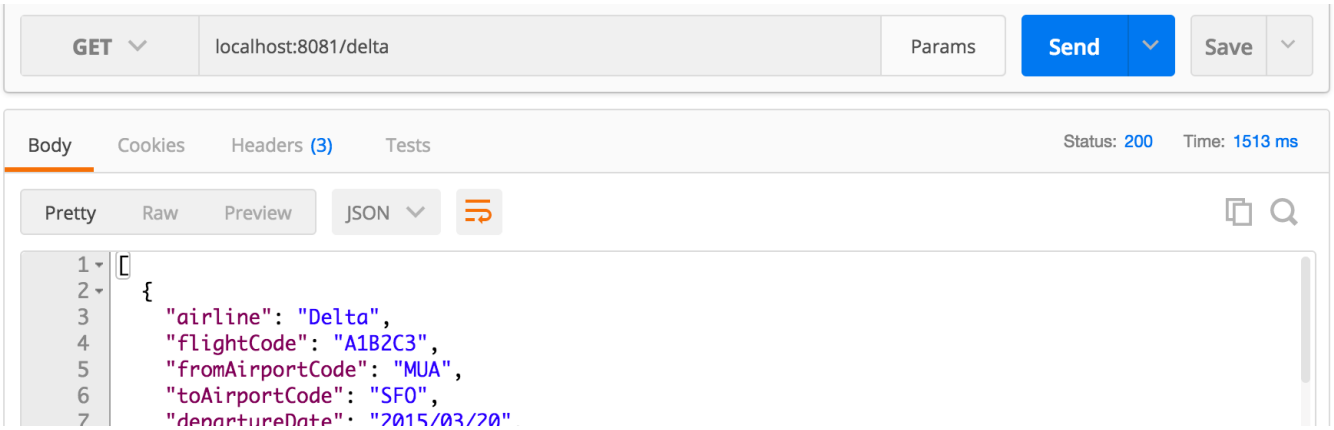
37. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

38. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.



Test the application

39. Run the project.
40. In Postman, make another request to http://localhost:8081/delta; you should see all the flight data but now as JSON.

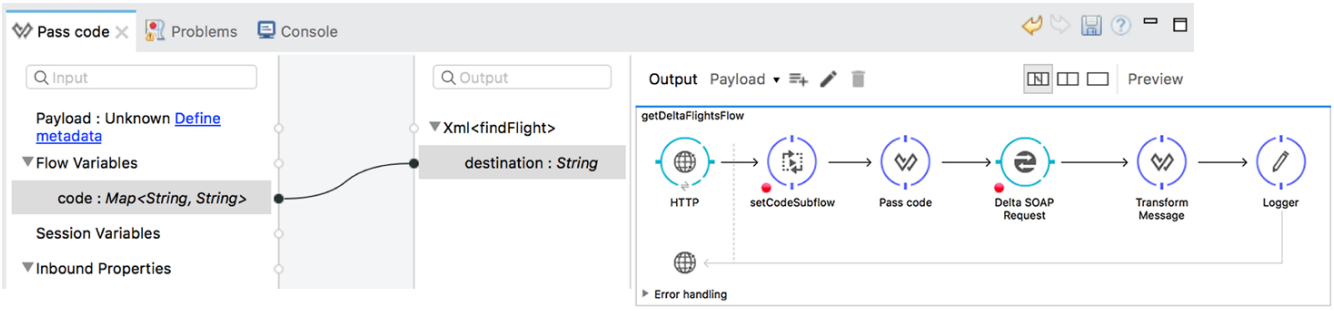


41. Add a query parameter called code and set it equal to LAX.
42. Send the request; you should still get all flights.

Walkthrough 8-5: Pass arguments to a SOAP web service using DataWeave

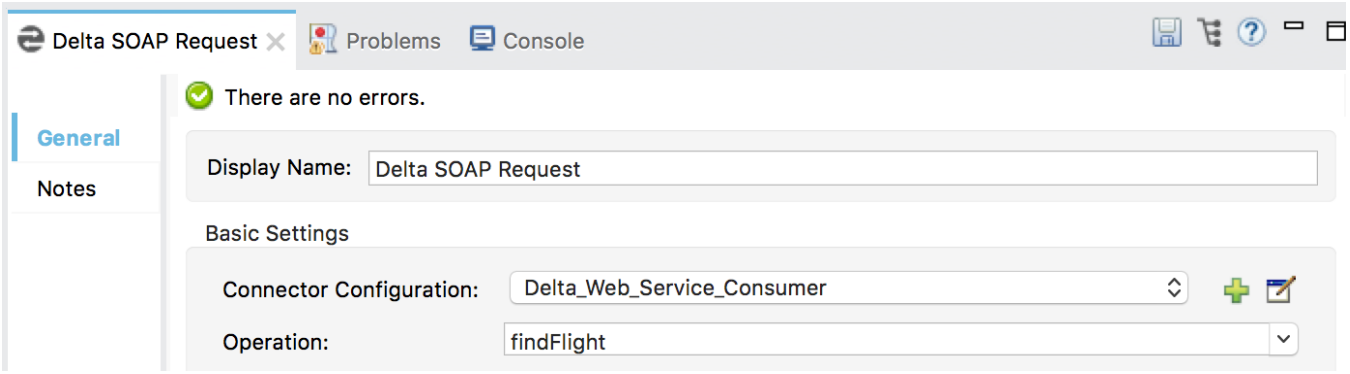
In this walkthrough, you modify the Delta flow to return the flights for a specific destination instead of all the flights. You will:

- Change the web service operation invoked to one that requires a destination as an input argument.
- Set a flow variable to the desired destination.
- Use DataWeave to pass the flow variable to the web service operation.



Call a different web service operation

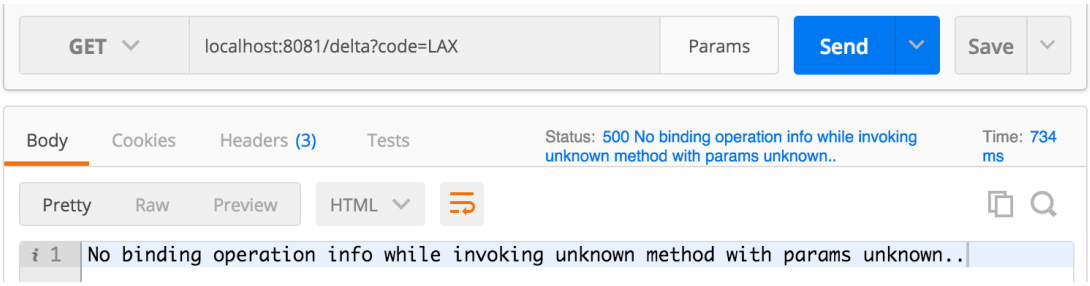
1. Return to getDeltaFlightsFlow.
2. In the Properties view for the Delta SOAP Request endpoint, change the operation to findFlight.



Test the application

3. Apply the changes to redeploy the application.

4. In Postman, send the same request with the query parameter; you should get a 500 response with a message about unknown parameters.

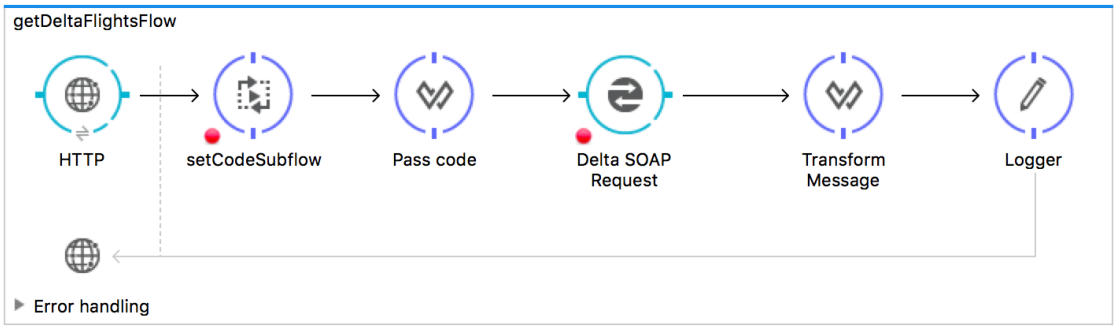


Use the set airport code subflow

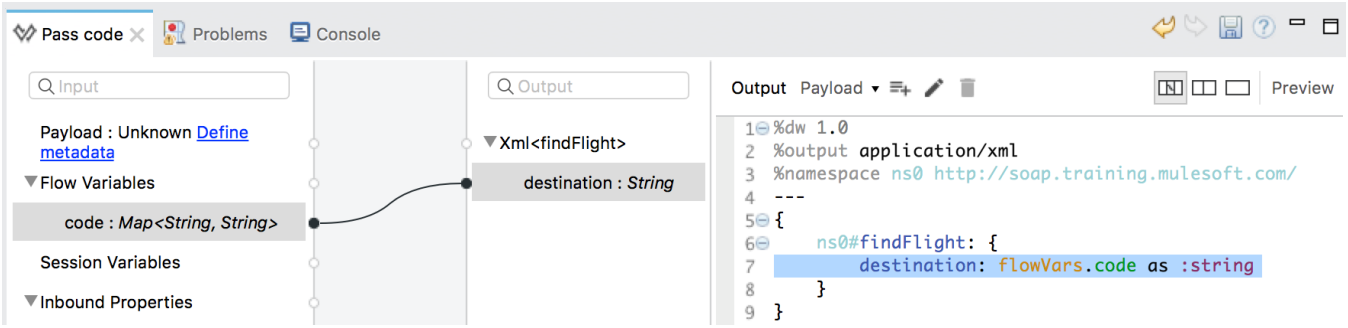
5. Return to getDeltaFlightsFlow.
6. Add a Flow Reference component before the Delta REST Request endpoint.
7. In the Flow Reference properties view, set the flow name to setCodeSubflow.

Use DataWeave to pass parameters to the web service

8. Add a Transform Message component to the left of the Delta SOAP Request endpoint.
9. Change its display name to Pass code.

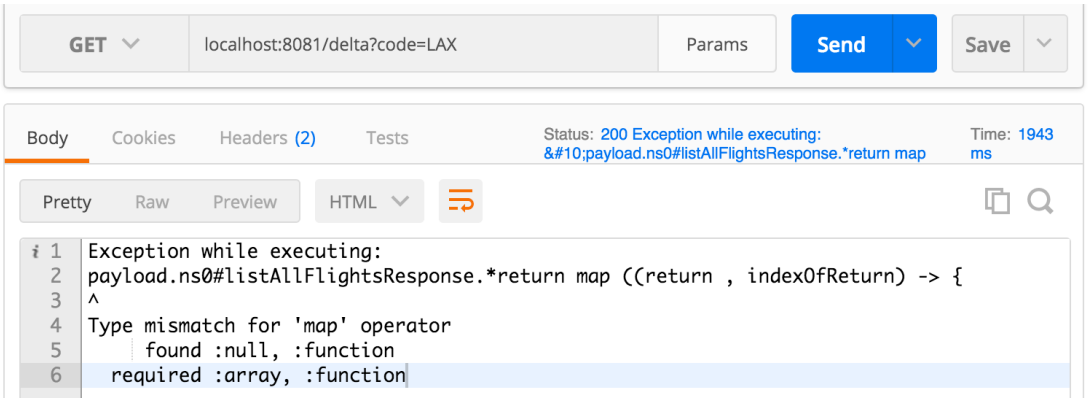


10. In the Pass code properties view, look at the input and output sections.
11. Drag the code flow variable in the input section to the destination element in the output section.



Test the application

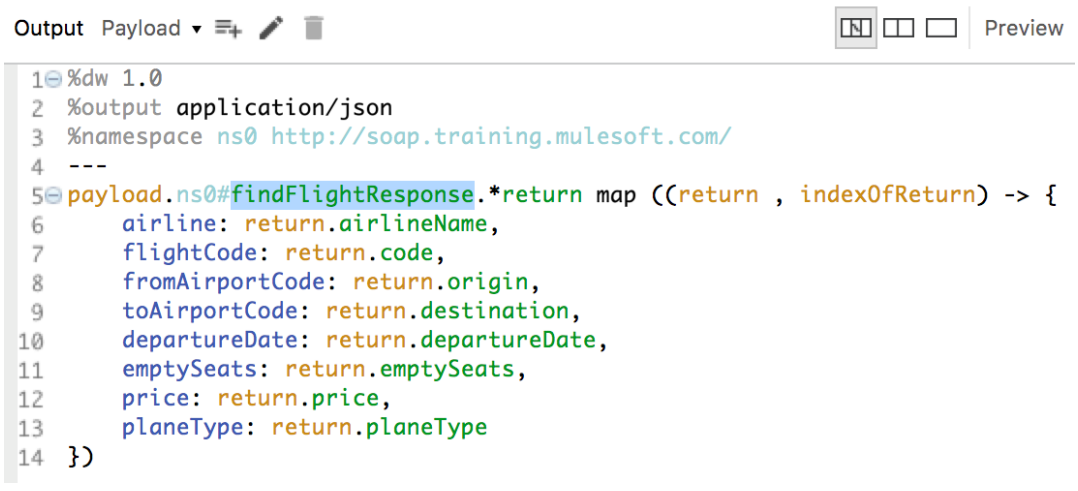
12. Redeploy the application.
13. In Postman, make the same request; you should get a 200 status code with an exception message.



14. Examine the exception message and figure out what is wrong with the transformation.

Modify the DataWeave expression

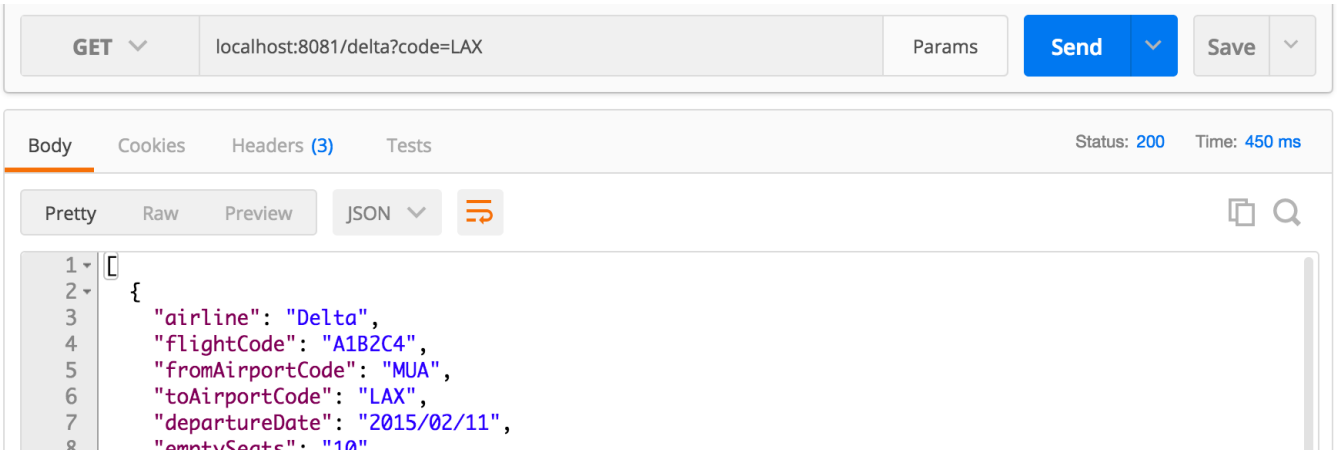
15. Return to getDeltaFlightsFlow.
16. Go to the Properties view for the Transform Message component after Delta SOAP Request.
17. Look at the transformation expression.
18. Change ns0#listAllFlightsResponse in the transformation code to ns0#findFlightResponse.



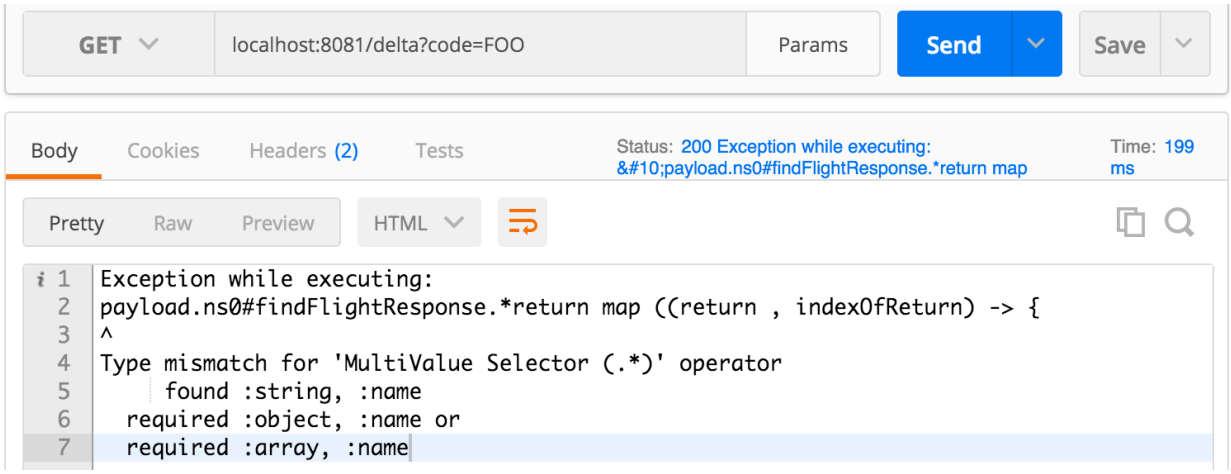
Test the application

19. Redeploy the application.

20. In Postman, make the same request; you should now get all the flights to LAX.

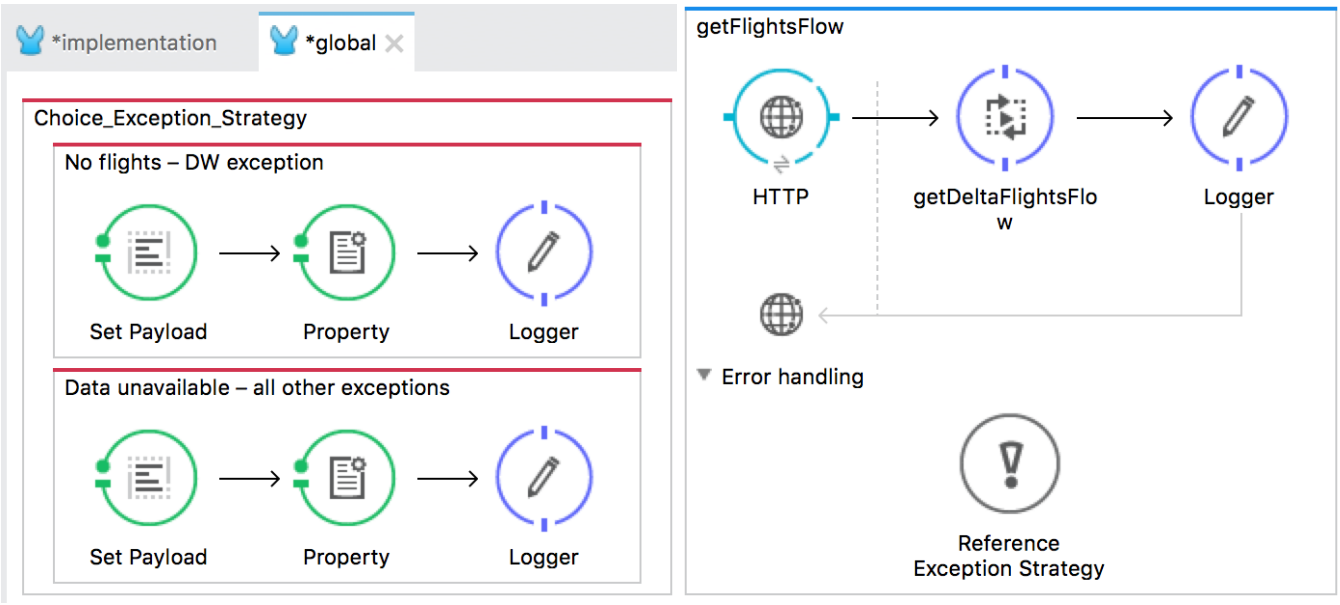


21. Change the code query parameter to have a value of CLE.
22. Send the request; you should now get only flights to CLE.
23. Change the code query parameter to have a value of FOO.
24. Send the request; you should get a 200 status code and a message with an exception.



25. Return to Anypoint Studio and stop the project.

Module 9: Handling Errors



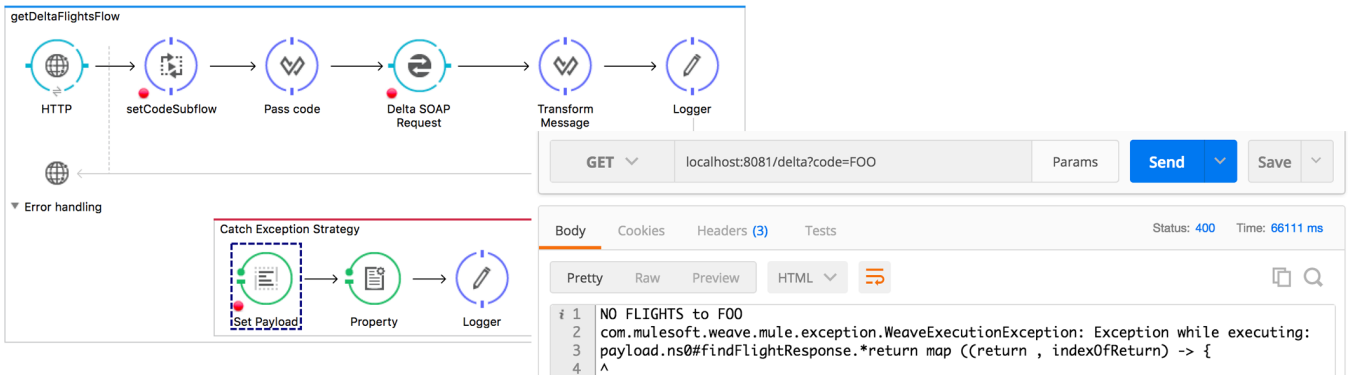
At the end of this module, you should be able to:

- Describe the different types of exception strategies.
- Handle messaging exceptions in flows.
- Create and use global exception handlers.
- Specify a global default exception strategy.

Walkthrough 9-1: Handle a messaging exception

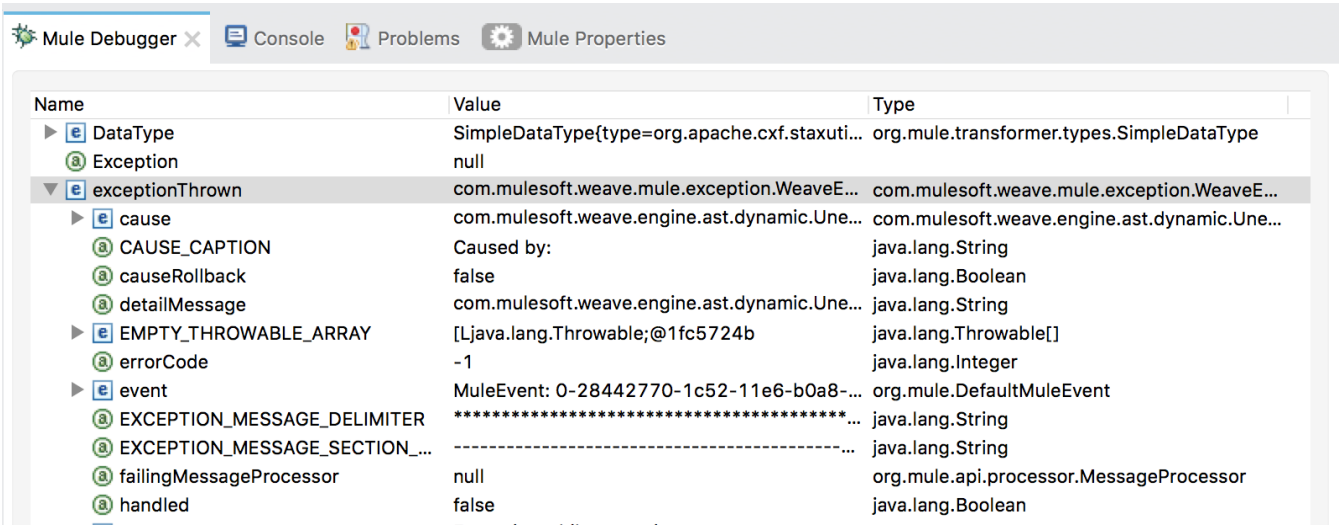
In this walkthrough, you handle an exception thrown by the Delta flow when a destination with no flights is used. You will:

- Add a catch exception strategy to a flow.
- Catch an exception and set the payload to send an error message back.
- Reference an exception object inside an exception handler.
- Set an HTTP status code inside an exception handler.



Debug the application for a request with a non-existent destination

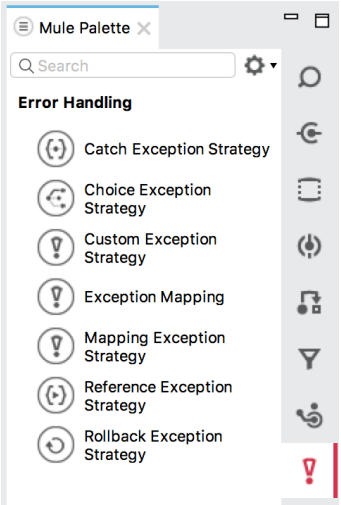
1. Return to getDeltaFlightsFlow.
2. Make sure there is a breakpoint on the flow reference component.
3. Debug the project.
4. In Postman, make another request to http://localhost:8081/delta?code=FOO.
5. In the Mule Debugger, step through the application and when you get the exception, drill-down into the exceptionThrown object.



6. Click the Resume button.

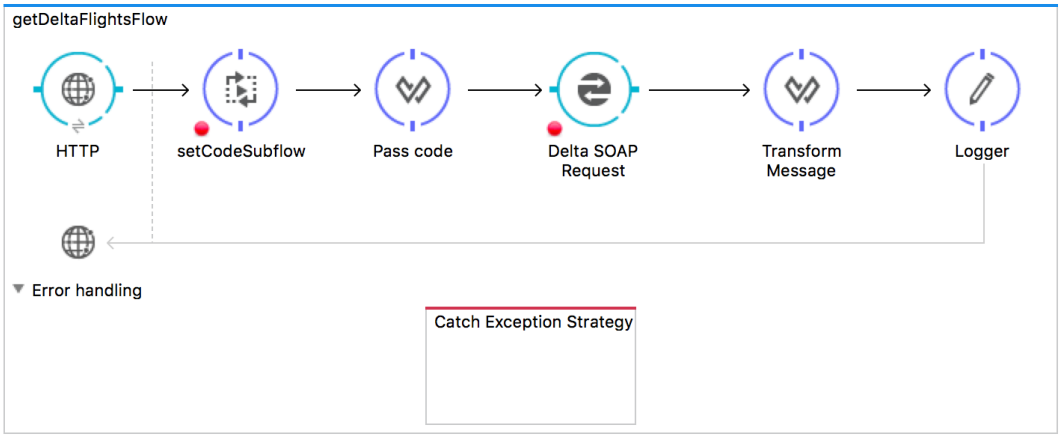
Browse the error handling elements in the Mule Palette

7. Switch perspectives and in the Mule Palette, select the Error Handling tab.
8. View the available error handling processors.



Add a catch exception strategy

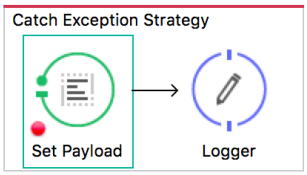
9. In getDeltaFlightsFlow, click the arrow to expand the Error handling section.
10. Drag a Catch Exception Strategy from the Mule Palette into the error handling section of the flow.



11. Add a Set Payload transformer to the catch exception strategy.
12. In the Set Payload properties view, set the value the following MEL expression:

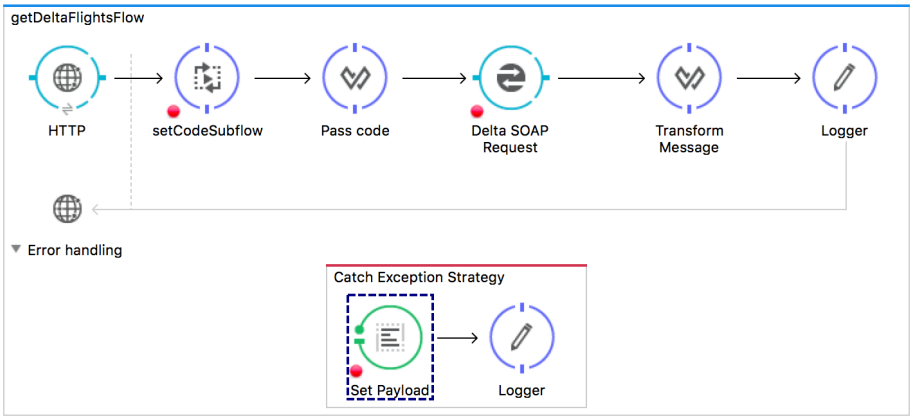
```
NO FLIGHTS to #[flowVars.code +  '\n' + exception]
```

13. Make sure there is a breakpoint on the transformer inside the catch exception.
14. Add a Logger after the transformer.

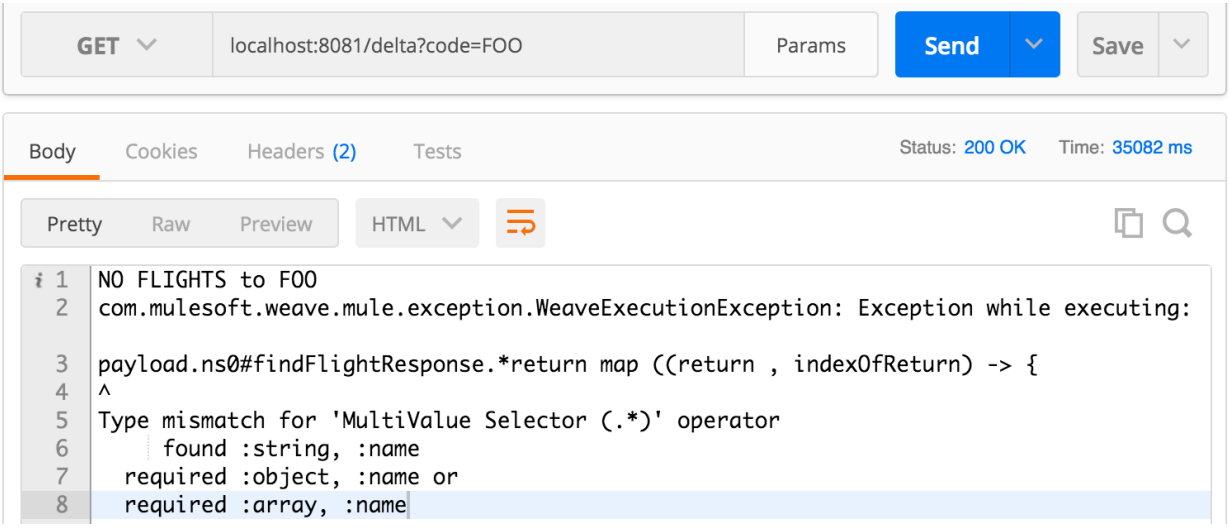


Test the application

15. Redeploy the application in debug mode.
16. In Postman, make another request to http://localhost:8081/delta?code=FOO.
17. Step through the application; you should see the exception thrown in getDeltaFlightsFlow and handled by the exception handler.

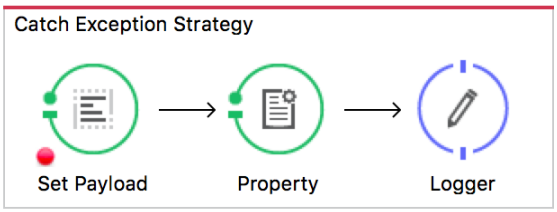


18. Step to the end of the application.
19. In Postman, you should get a 200 response with your custom message.

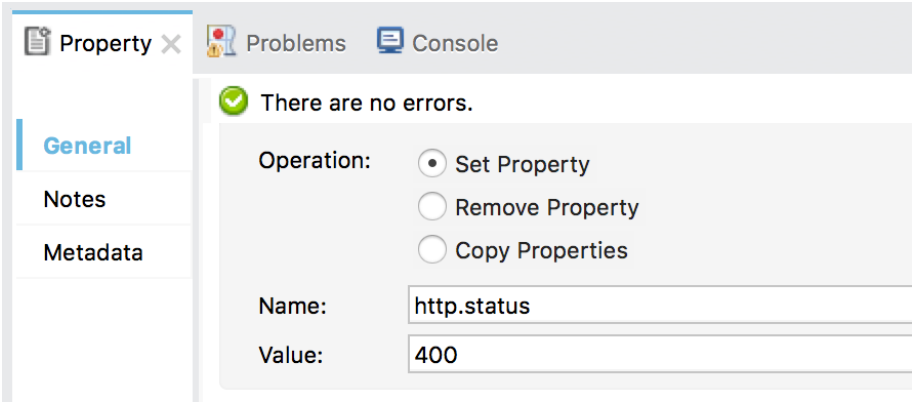


Set the http status code

20. Return to getDeltaFlightsFlow in the Mule Design perspective.
21. In the catch exception strategy, add a Property transformer after the Set Payload transformer.

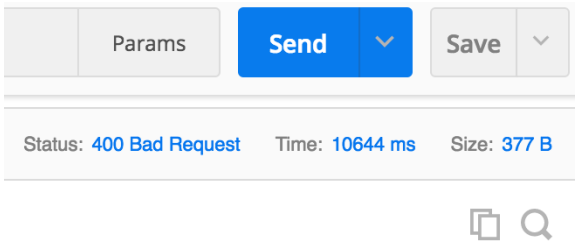


22. In the Property properties view, select Set Property.
23. Set the name to http.status and the value to 400.



Test the application

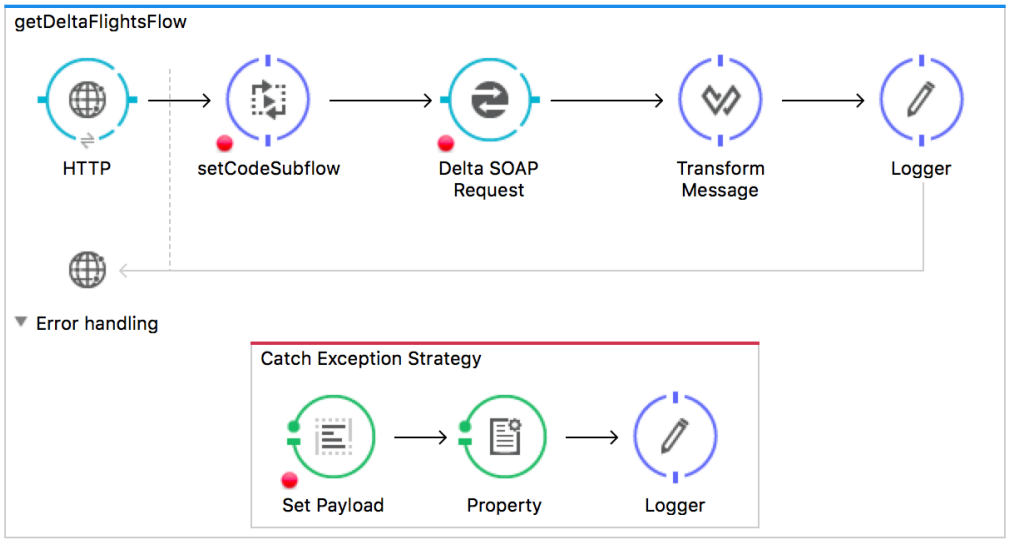
24. Redeploy the application.
25. In Postman, make another request to http://localhost:8081/delta?code=FOO.
26. In the Mule Debugger, step through the application.
27. Return to Postman; you should now get a 400 status code.



Create a different type of error

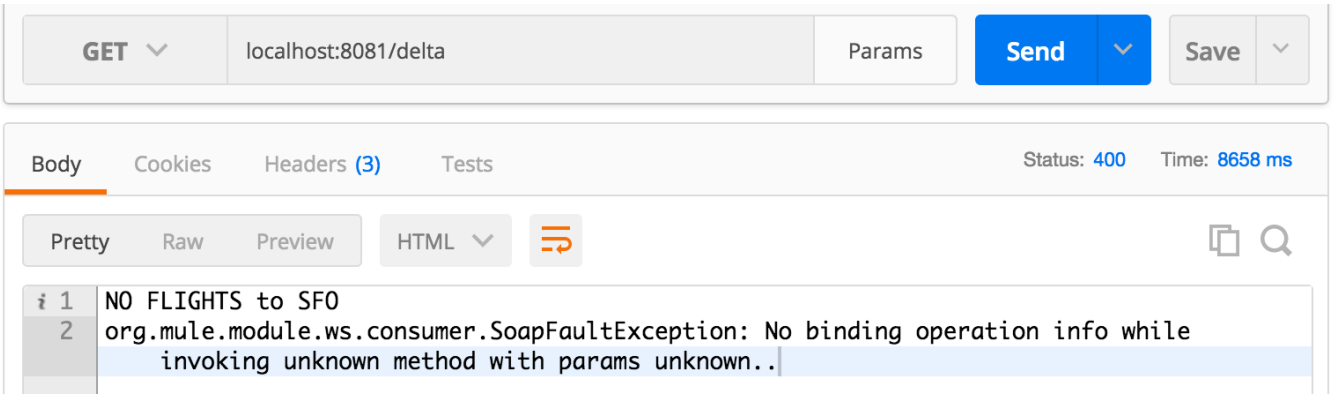
28. Return to getDeltaFlightsFlow.

29. Delete the Pass code processor.



Test the application

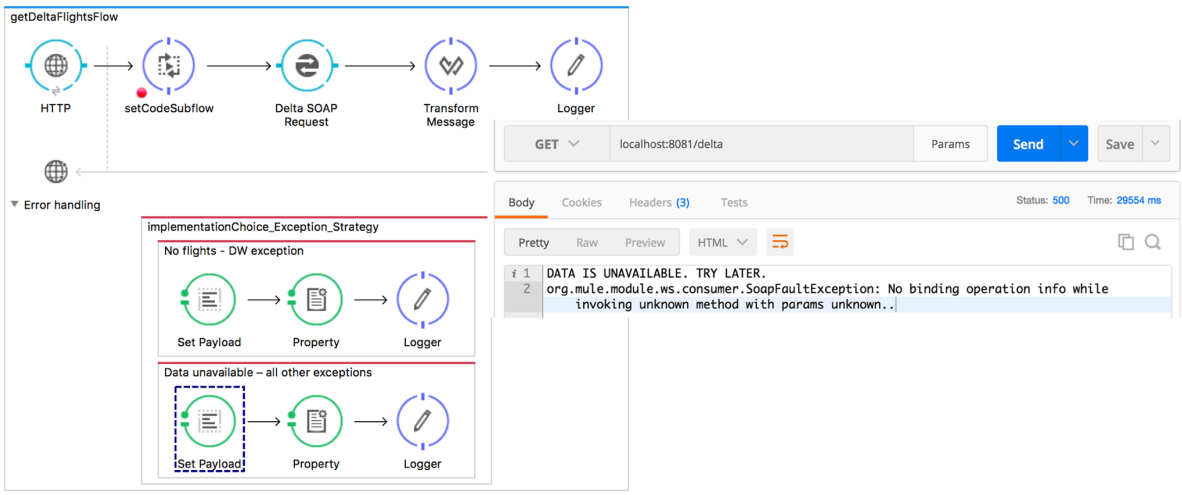
30. Redeploy the application.
31. In Postman, delete the query parameter and send the request.
32. In the Mule Debugger, step through the application until you get the exception and move into the exception handler; you should see the exception is handled by the same exception handler.
33. Drill-down into the Exception object in the debugger.
34. Click Resume.
35. In Postman, you should get the same message, but this time saying there are no flights to SFO even though that was not the problem.



Walkthrough 9-2: Handle different types of messaging exceptions

In this walkthrough, you handle multiple types of exceptions thrown by the Delta flow. You will:

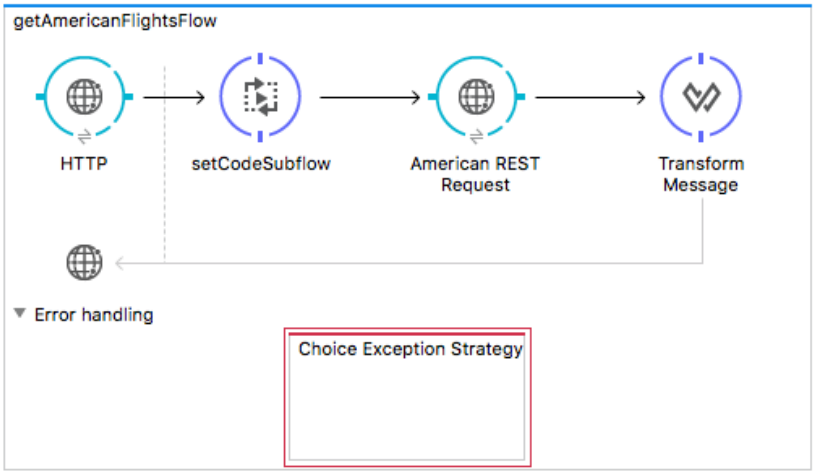
- Add and configure a choice exception strategy.
- Get exceptions handled by both of the catch exception strategies in the choice strategy.
- Create a new flow that calls a flow that has an exception so the exception can bubble up and be handled by the calling flow.



Add a choice exception strategy

1. Return to implementation.xml in the Mule Design perspective.
2. Drag a Choice Exception Strategy from the Mule Palette and drop it into the error handling section of getAmericanFlightsFlow.

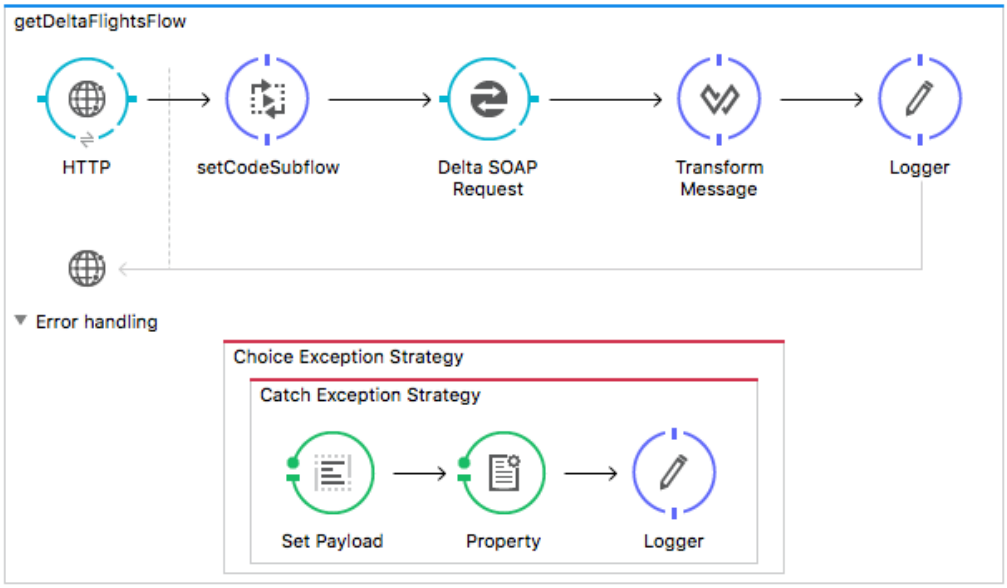
   Note: You cannot it directly to getDeltaFlightsFlow because that flow already contains an exception strategy, so you are adding it to getAmericanFlightsFlow temporarily.

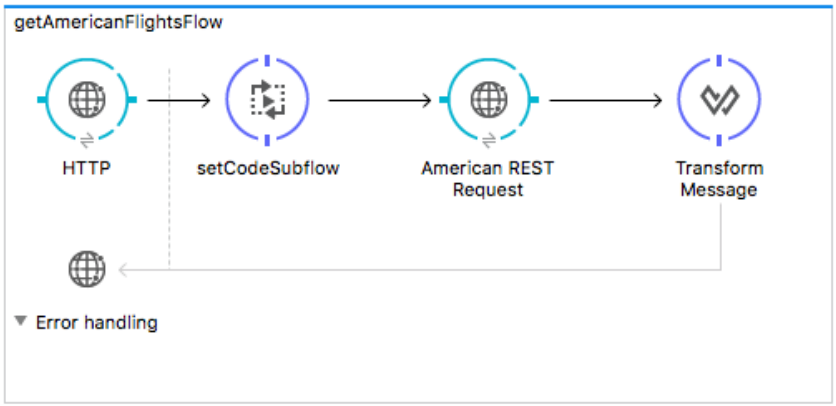


3. Drag the catch exception strategy from getDeltaFlightsFlow and drop it in the choice exception strategy in getAmericanFlightsFlow.
4. Drag getDeltaFlightsFlow and drop it above getAmericanFlightsFlow.

   Note: If you must scroll to see all the flows on the canvas, it is difficult to drag-and-drop elements near the bottom of the canvas.

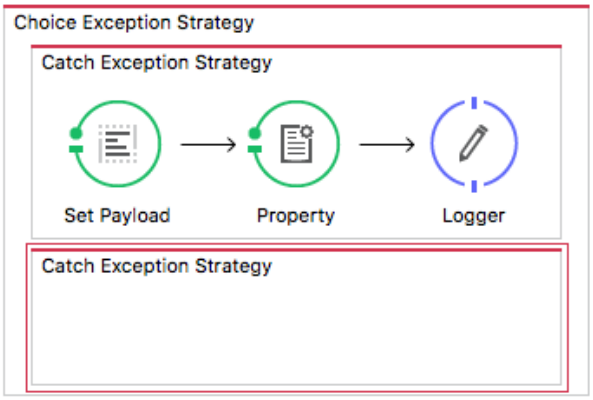
5. Drag the choice exception strategy from getAmericanFlightsFlow to getDeltaFlightsFlow.



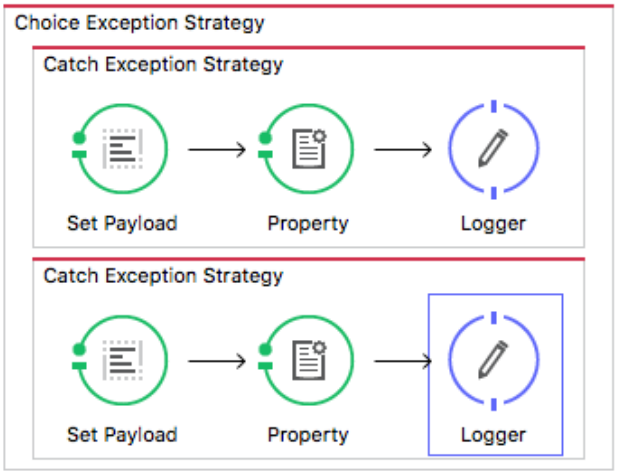


6. Drag a second Catch Exception Strategy from the Mule Palette and drop it in the choice exception strategy.

   Note: If you are having difficulty adding it, try dropping it on the catch exception strategy already there.



7. Add a Set Payload transformer, a Property transformer, and a Logger to the new catch exception strategy.



8. In the Set Payload properties view, set the value the following MEL expression:

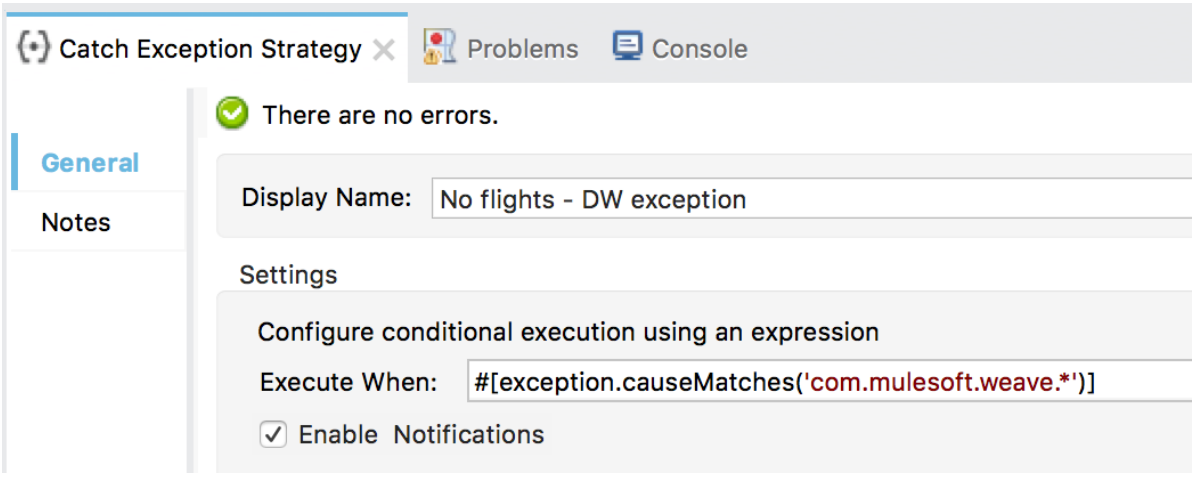
   ```
   DATA IS UNAVAILABLE. TRY LATER. #['\n' + exception]
   ```

9. In the Property properties view, select Set Property.
10. Set the name to http.status and the value to 500.

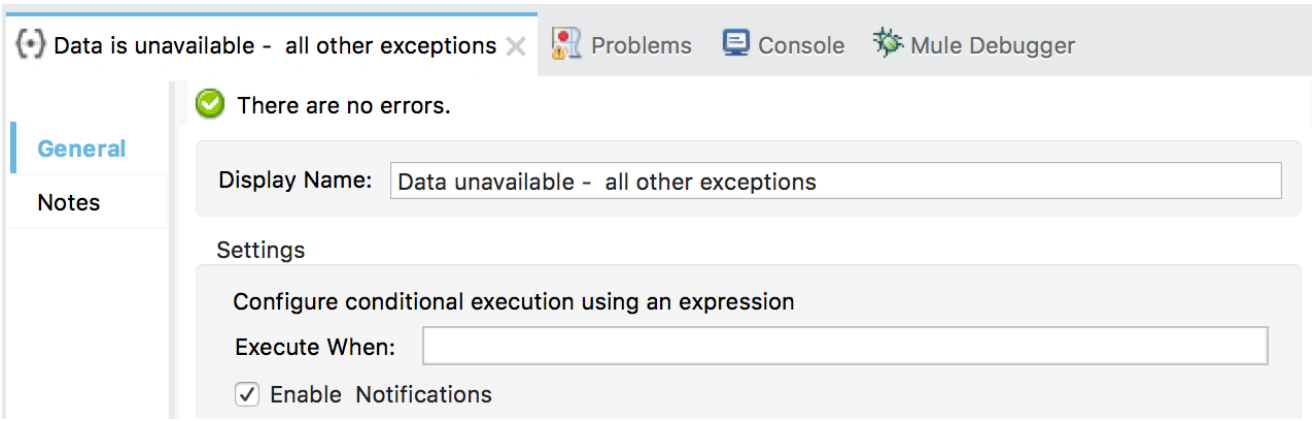
Configure the choice exception strategy

11. In the Properties view for the first catch exception strategy, set the display name to No flights – DW exception.
12. Set the execute when value to an expression for when a com.mulesoft.weave.* exception is thrown; use the causeMatches() method.

```
#[exception.causeMatches('com.mulesoft.weave.*')]
```



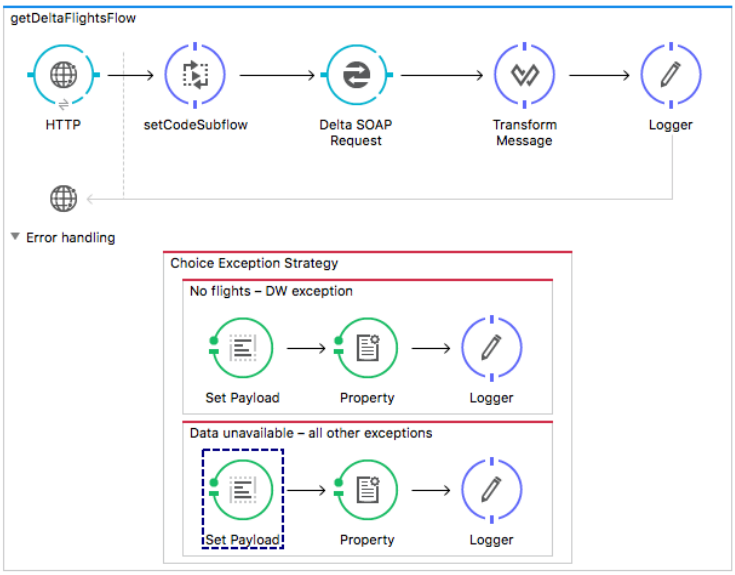
13. In the Properties view for the second catch exception strategy, set the display name to Data unavailable – all other exceptions.
14. Leave the execute when value blank.



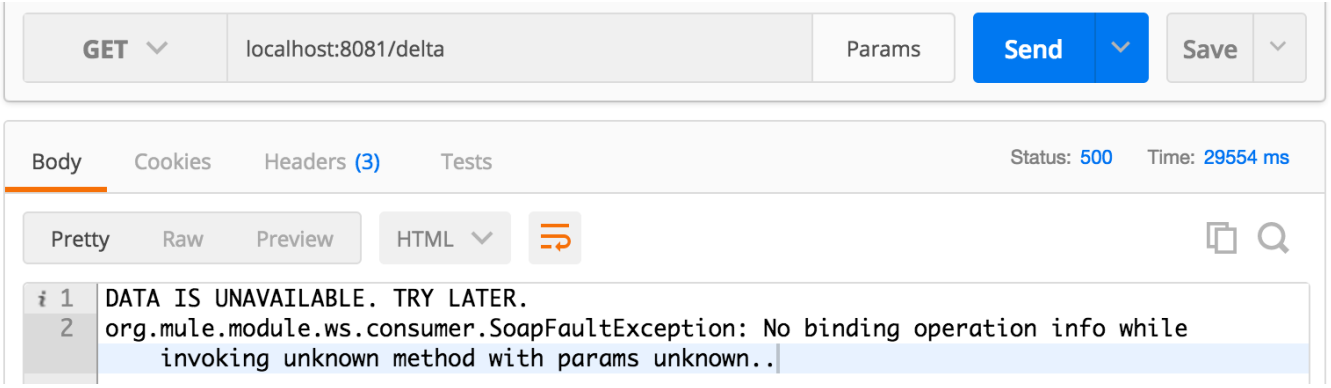
Test the application

15. Debug the application.
16. In Postman, make another request to http://localhost:8081/delta.

17. In the Mule Debugger, step through the application; the exception should be handled by the second catch block.

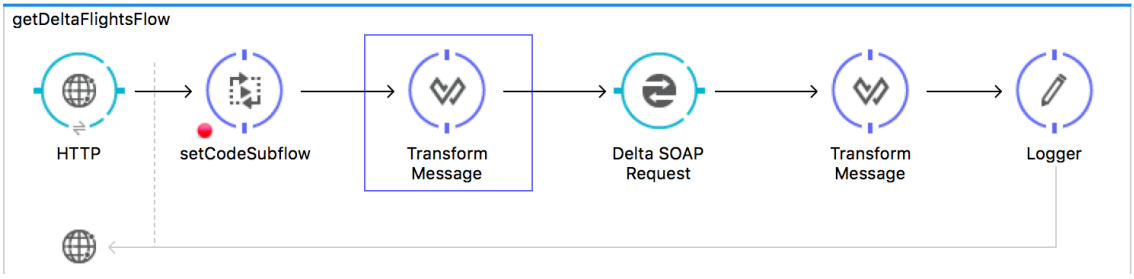


18. Step to the end of the application.
19. Return to Postman; you should see the second error message.



Fix the request error

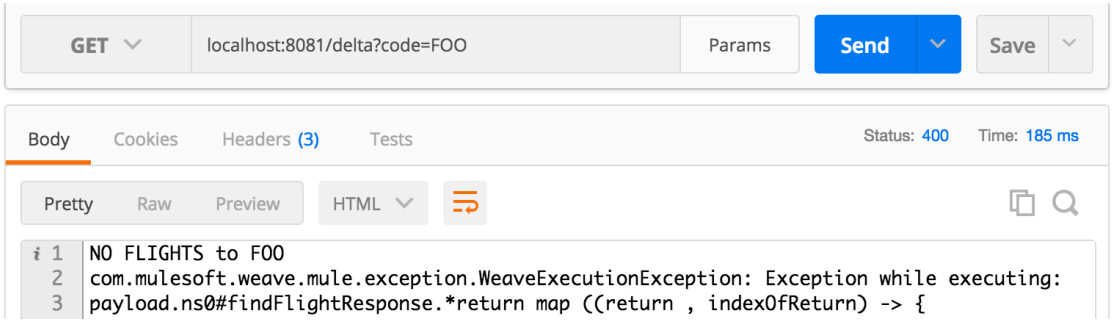
20. Return to getDeltaFlightsFlow and switch perspectives.
21. Add a Transform Message component before the Delta SOAP Request processor.



22. In the Transform Message properties view, drag the code flow variable in the input section to the destination element in the output section.

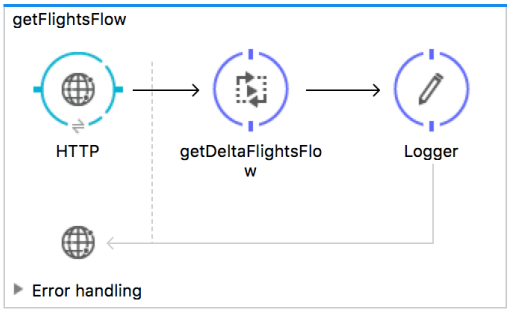
Test the application

23. Run the project.
24. In Postman, make the same request to http://localhost:8081/delta; you should get the SFO results again.
25. Make a request to http://localhost:8081/delta?code=FOO; the exception is handled by the first exception strategy and you should get the NO FLIGHTS TO FOO message.



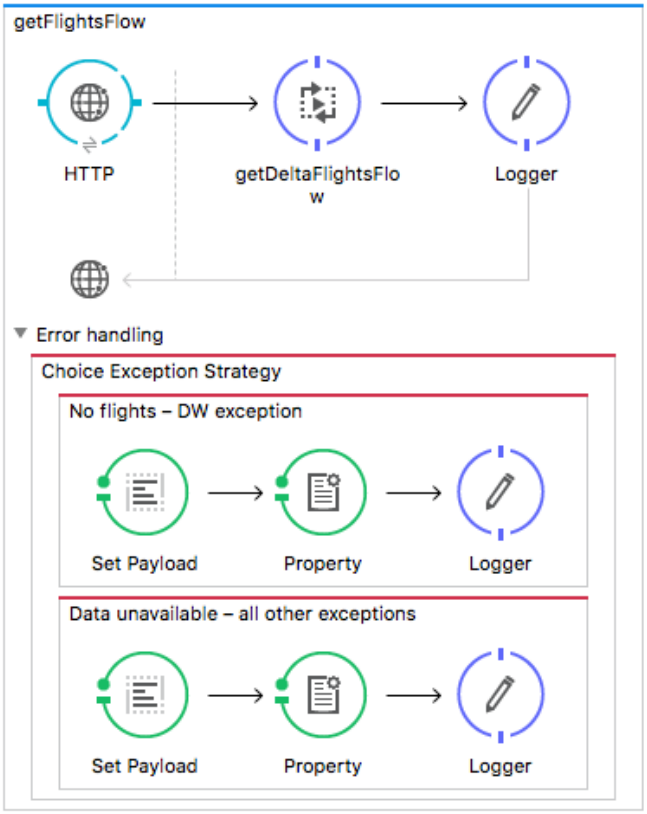
Call the Delta flow from another flow

26. Return to Anypoint Studio.
27. From the Mule Palette, drag an HTTP connector and drop it at the top of the canvas above all the other flows.
28. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
29. Set the path to /flights and the allowed methods to GET.
30. Change the flow name to getFlightsFlow.
31. Add a Flow Reference component to the flow.
32. In the Flow Reference properties view, set the flow to getDeltaFlightsFlow.
33. Add a Logger to the end of the flow.



Move a catch exception strategy to the calling flow

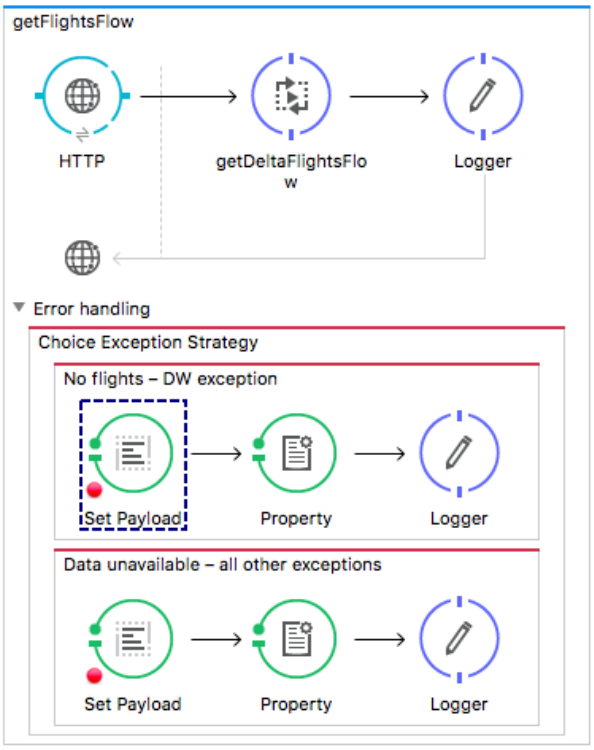
34. Expand the error handling section of getFlightsFlow.
35. Move the choice exception strategy from getDeltaFlightsFlow to getFlightsFlow.



36. Test the application Make sure there is a breakpoint on each of the Set Payload transformers in the catch exception strategies.
37. Debug the project.
38. In Postman, make a request to http://localhost:8081/flights?code=FOO.

    Note: Call the new /flights endpoint; not the /delta one.

39. In the Mule Debugger, step through the application; you should see the exception thrown by the Delta flow is caught by the calling flow.

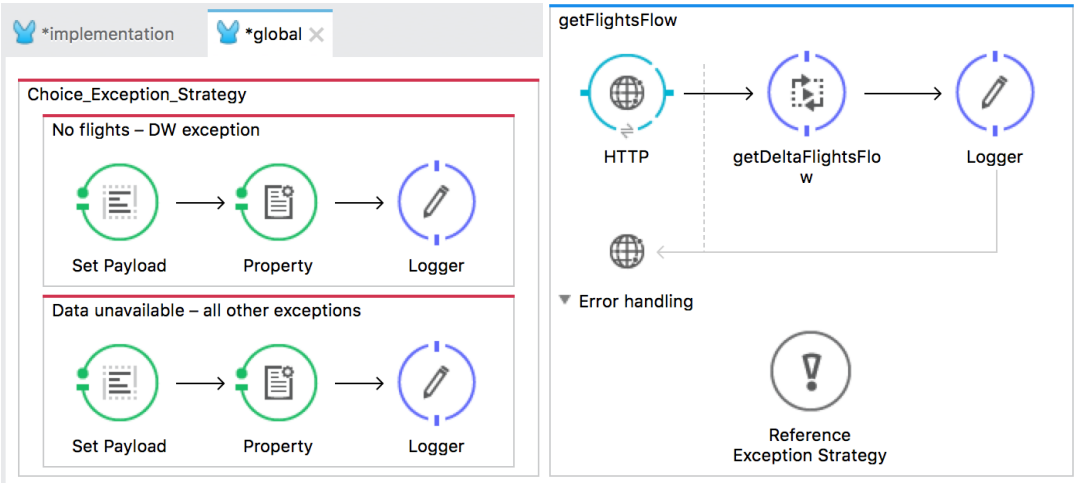


40. Step through the rest of the application and then stop the project and switch perspectives.

Walkthrough 9-3: Create and use global exception strategies

In this walkthrough, you create a global exception handler for the project. You will:

- Create a global exception handler.
- Reference and use the global exception handler in flows.



Create a global exception handler

1. Return to implementation.xml.
2. Switch to the Configuration XML view.
3. Locate the choice exception strategy and select it and cut it.

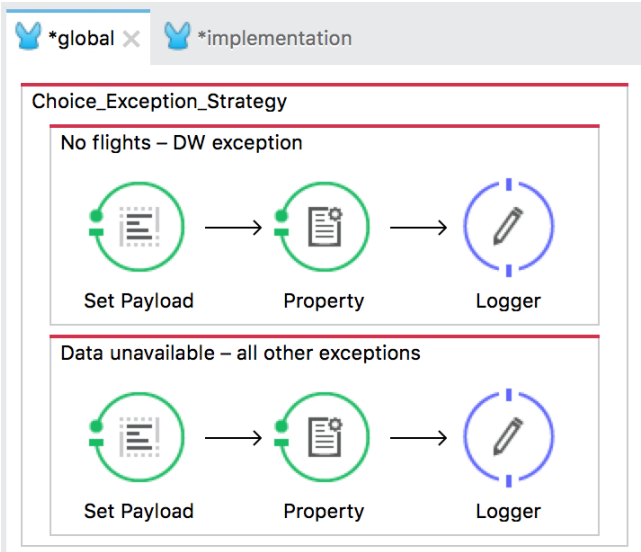
```
global
implementation
12      <flow name="getFlightsFlow">
13          <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedMetho
14          <flow-ref name="getDeltaFlightsFlow" doc:name="getDeltaFlightsFlow"/>
15          <logger level="INFO" doc:name="Logger"/>
16          <choice-exception-strategy doc:name="Choice Exception Strategy">
17              <catch-exception-strategy when="#[exception.causeMatches('com.mulesoft.weave.*')
18                  <set-payload value="NO FLIGHTS to #[flowVars.code +  '\n' + exception]" doc:
19                  <set-property propertyName="http.status" value="400" doc:name="Property"/>
20                  <logger level="INFO" doc:name="Logger"/>
21              </catch-exception-strategy>
22              <catch-exception-strategy doc:name="Data unavailable &#8211; all other exception
23                  <set-payload value="DATA IS UNAVAILABLE. TRY LATER. #['\n' + exception]
24  " doc:name="Set Payload"/>
25                  <set-property propertyName="http.status" value="500" doc:name="Property"/>
26                  <logger level="INFO" doc:name="Logger"/>
27              </catch-exception-strategy>
28          </choice-exception-strategy>
29      </flow>
30      <sub-flow name="setCodeSubflow">
```

4. Go to the Configuration XML view in global.xml.
5. Place the cursor on a new line inside the start and end mule tags after the other tags.
6. Paste the choice exception strategy.

7. In the choice-exception-strategy tag, change doc:name to name and replace the spaces in the name with underscores; the problem should go away

```
*global  *implementation
    <ws:consumer-config name="Delta_Web_Service_Consumer" wsdlLocation="http://mu.mulesoft-trair

        <choice-exception-strategy name="Choice_Exception_Strategy">
            <catch-exception-strategy when="#[exception.causeMatches('com.mulesoft.weave.*')]" c
                <set-payload value="NO FLIGHTS to #[flowVars.code +  '\n' + exception]" doc:name
                <set-property propertyName="http.status" value="400" doc:name="Property"/>
                <logger level="INFO" doc:name="Logger"/>
            </catch-exception-strategy>
            <catch-exception-strategy doc:name="Data unavailable &#8211; all other exceptions">
                <set-payload value="DATA IS UNAVAILABLE. TRY LATER. #['\n' + exception]
" doc:name="Set Payload"/>
                <set-property propertyName="http.status" value="500" doc:name="Property"/>
                <logger level="INFO" doc:name="Logger"/>
            </catch-exception-strategy>
        </choice-exception-strategy>

</mule>
```

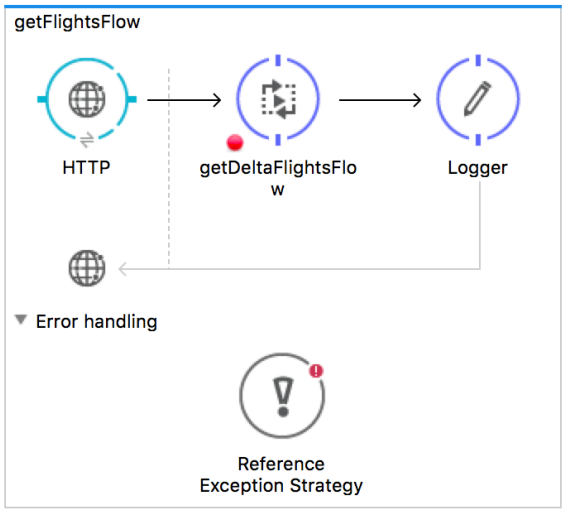
8. Switch to the Message Flow view; you should see the choice exception strategy.



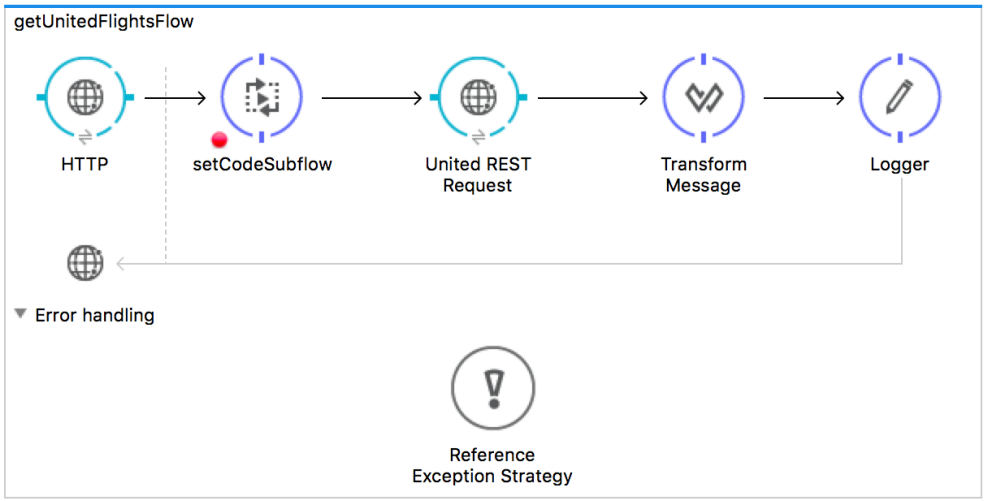
Use the global exception handler

9. Return to implementation.xml and switch to the Message Flow view.
10. Locate getFlightsFlow and expand its error handling section.

11. Drag a Reference Exception Strategy from the Mule Palette and drop it in the error handling section.



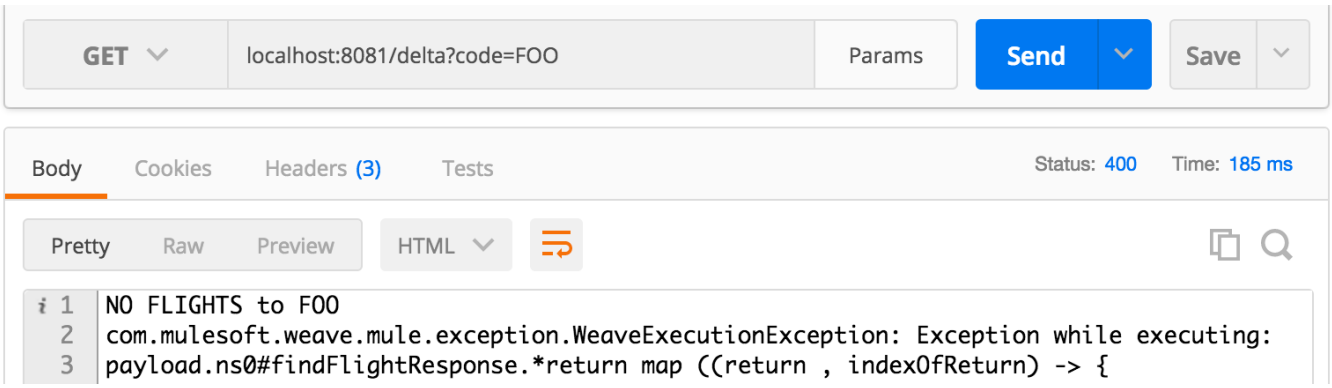
12. In the Reference Exception Strategy properties view, set the global exception strategy to the existing Choice_Exception_Strategy.
13. Expand the error handling section of getUnitedFlightsFlow.
14. Drag a Reference Exception Strategy from the Mule Palette and drop it in the error handling section.
15. In the Properties view, set the global exception strategy to Choice_Exception_Strategy.



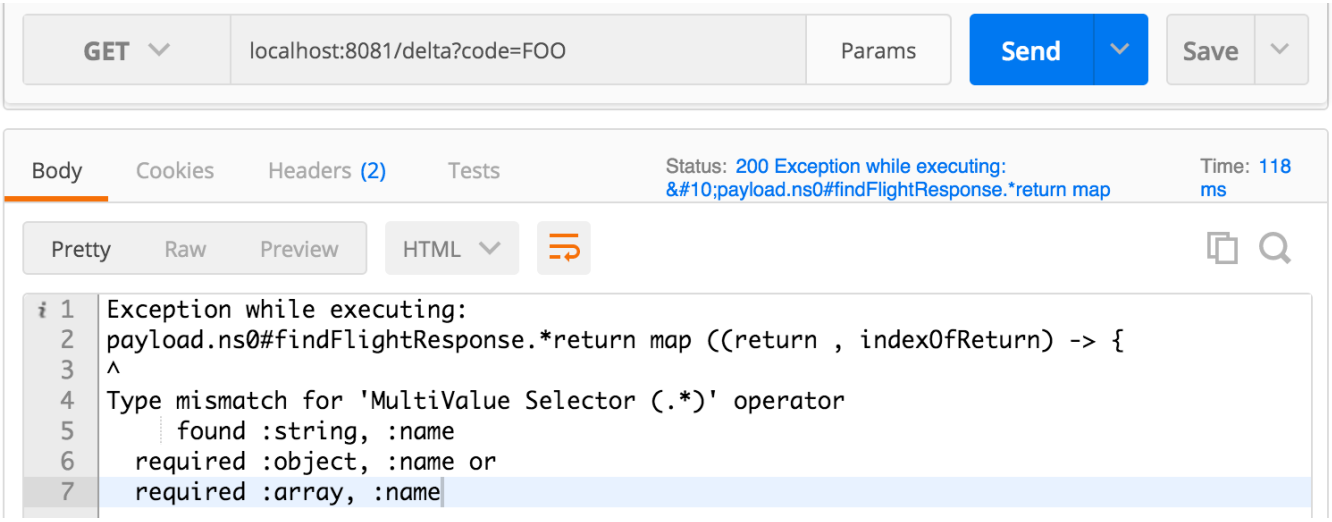
Note: You could add a reference exception strategy to the rest of the flows, but instead, you will create a global default exception strategy in the next walkthrough.

Test the application

16. Run the project.
17. In Postman, make the same request to http://localhost:8081/flights?code=FOO; you should get the same NO FLIGHTS to FOO message.



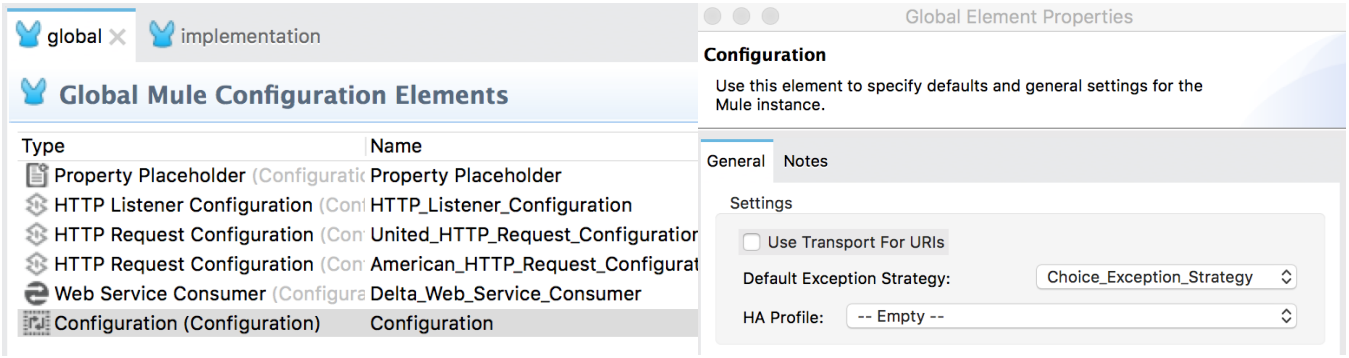
18. Make a request to http://localhost:8081/united?code=FOO; you should see the same message.
19. Make a request to http://localhost:8081/delta?code=FOO; you should NOT see the same message because the exception is not being handled.



Walkthrough 9-4: Specify a global default exception strategy

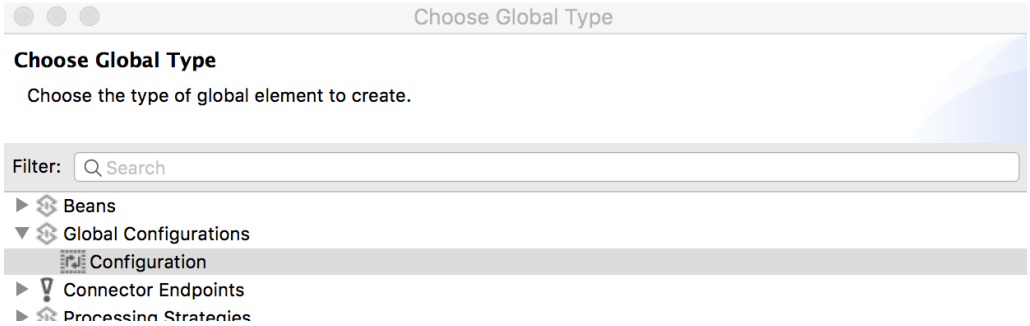
In this walkthrough, you change the default exception handling for the application. You will:

- Create a global configuration element in the global.xml file.
- Specify a default exception strategy in the global configuration element.
- Remove the existing exception handling strategies.
- Use the default exception handling strategy.

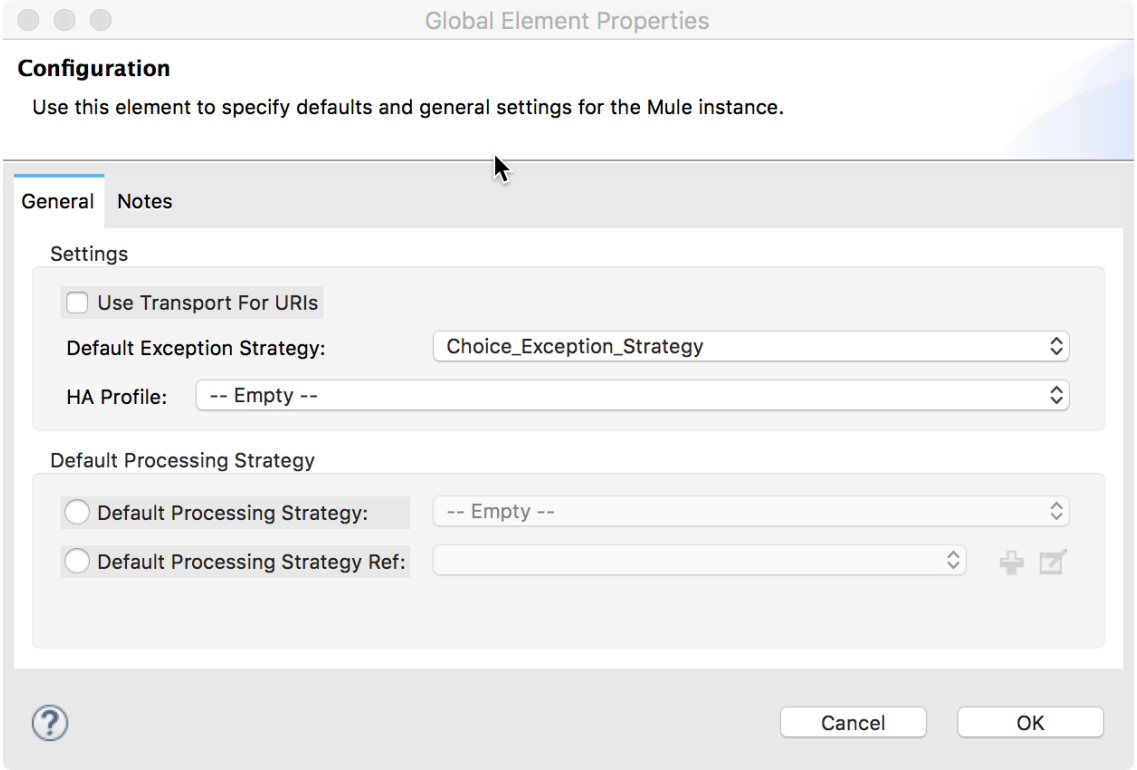


Specify a global default exception strategy

1. Return to global.xml.
2. Switch to the Global Elements view.
3. Click the Create button.
4. In the Choose Global Type dialog box, select Global Configurations > Configuration and click OK.

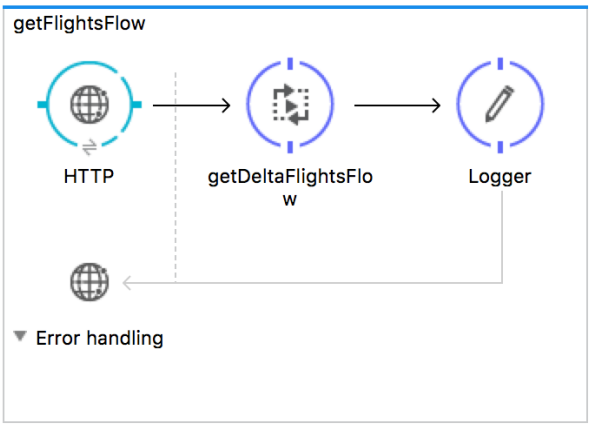


5. In the Global Element Properties dialog box, set the default exception strategy to Choice_Exception_Strategy and click OK.

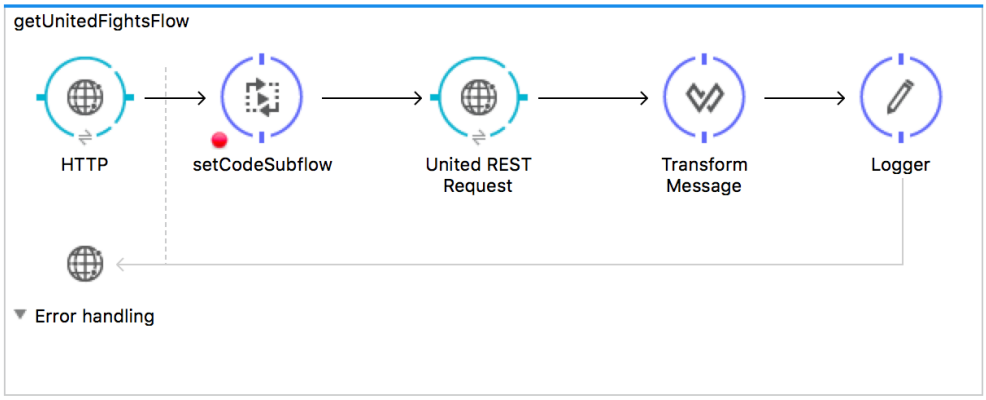


Remove the existing exception strategy references

6. Return to implementation.xml.
7. Delete the reference exception strategy in getFlightsFlow.

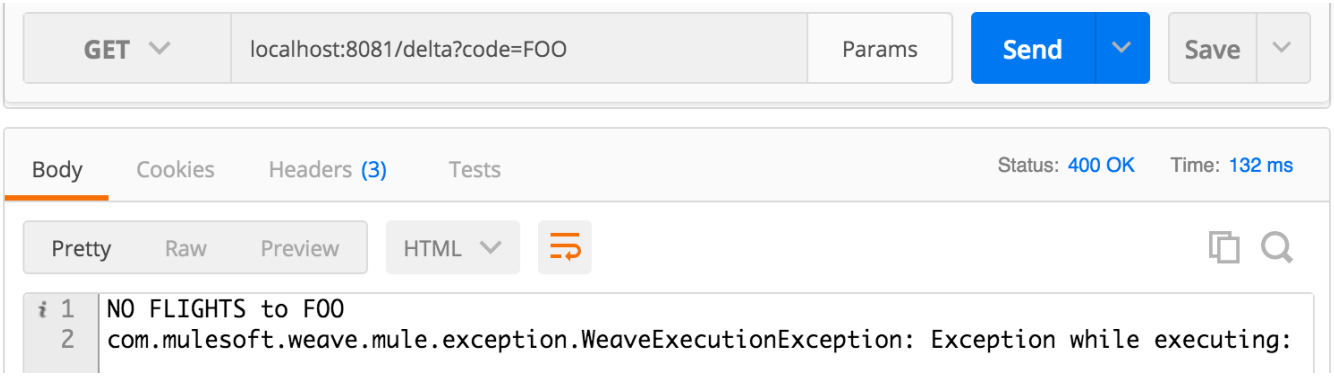


8. Delete the reference exception strategy in getUnitedFlightsFlow.



Test the application

9. Save all the files and run the project.
10. In Postman, make a request to http://localhost:8081/flights?code=FOO; you should still see the no flights error message displayed.
11. Make a request to http://localhost:8081/delta?code=FOO; you should now see the no flights error message displayed.

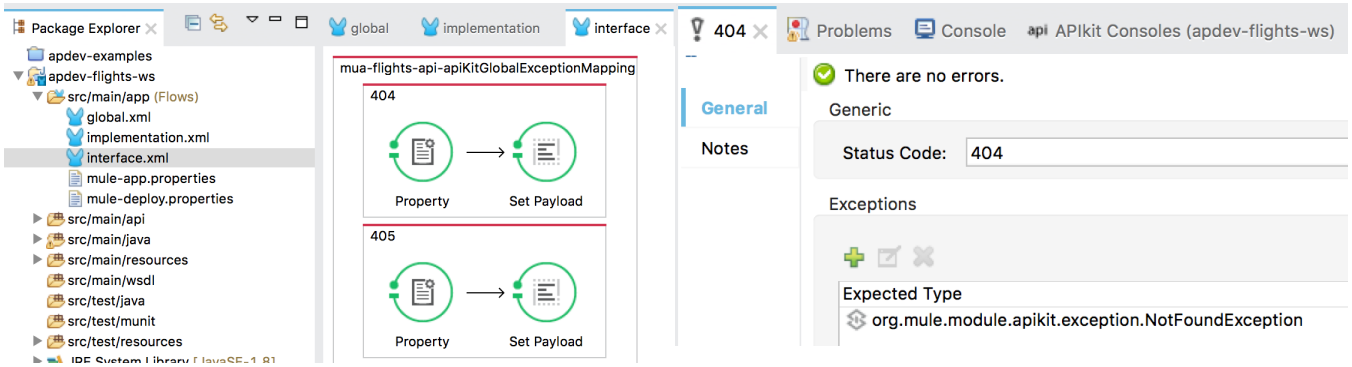


12. Return to Anypoint Studio and stop the project.

Walkthrough 9-5: Review a mapping exception strategy

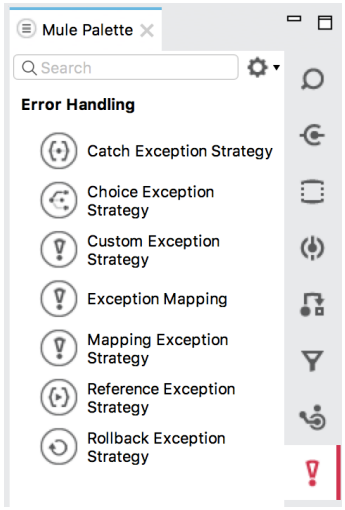
In this walkthrough, you review the mapping exception strategy that was created automatically by APIkit for the interface. You will:

- Locate the mapping exception strategy.
- Review the exception mappings.



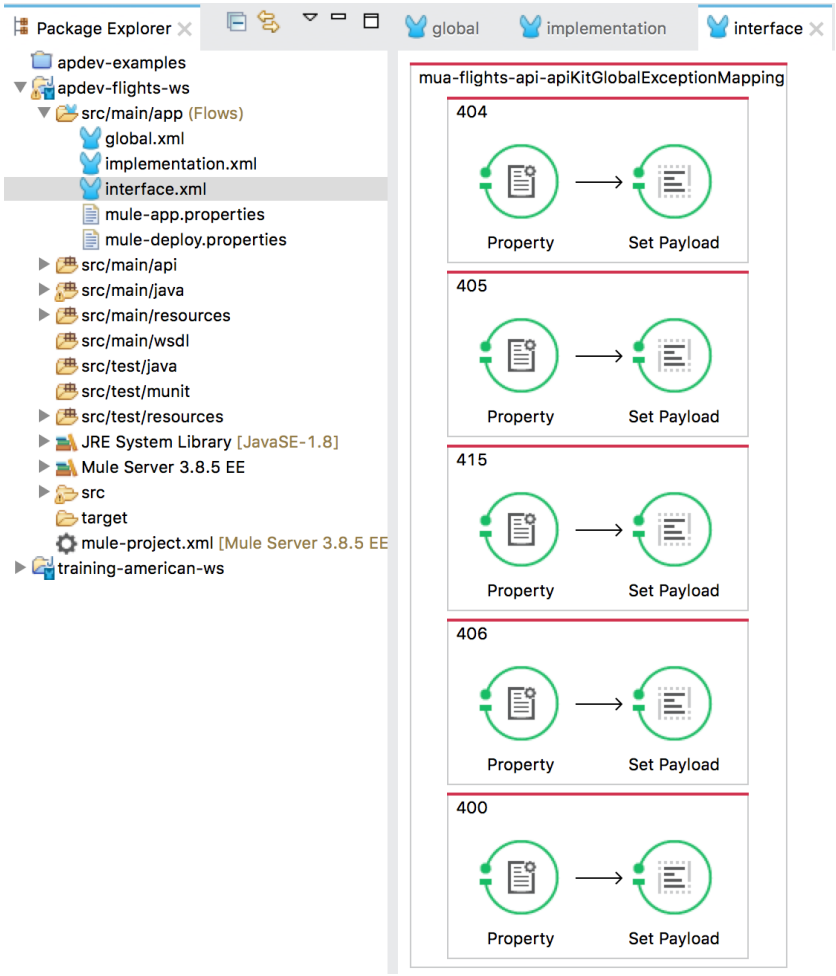
Browse the error handling elements in the Mule Palette

1. In the apdev-flights-ws project, open interface.xml in src/main/app.
2. In the Mule Palette, select the Error Handling tab.
3. Locate the Mapping Exception Strategy.

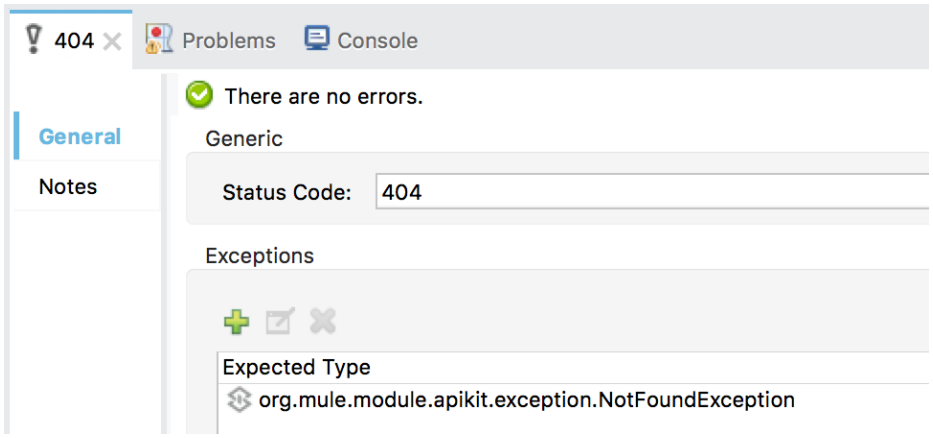


Review a mapping exception strategy

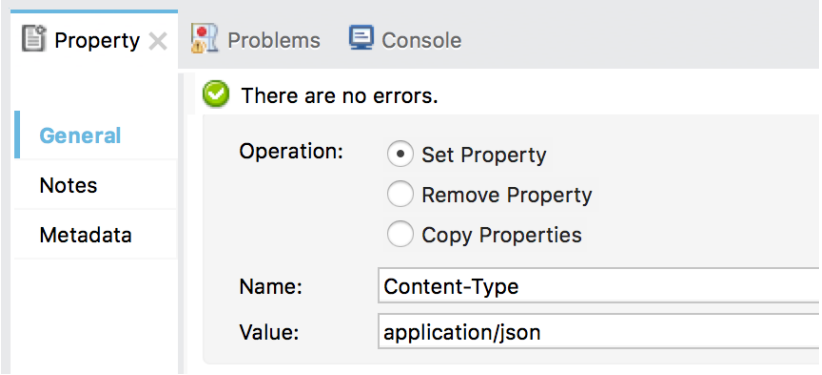
4. In interface.xml, locate the mua-flights-apiKitGlobalExceptionMapping Mapping exception strategy.



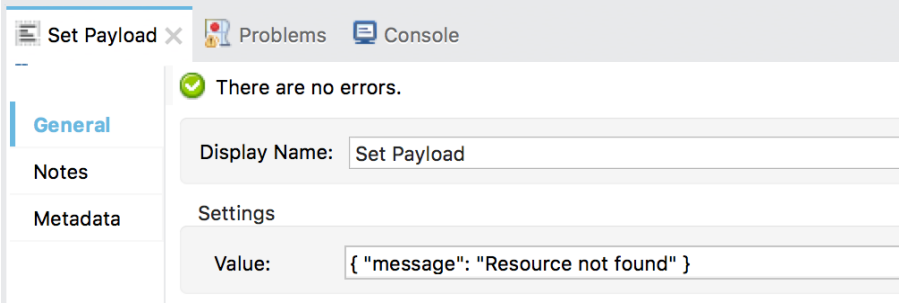
5. Double-click the 404 exception mapping and look at the settings in the 404 properties view.



6. Double-click the Property transformer in the 404 exception mapping and look at the settings in the Set Payload properties view.

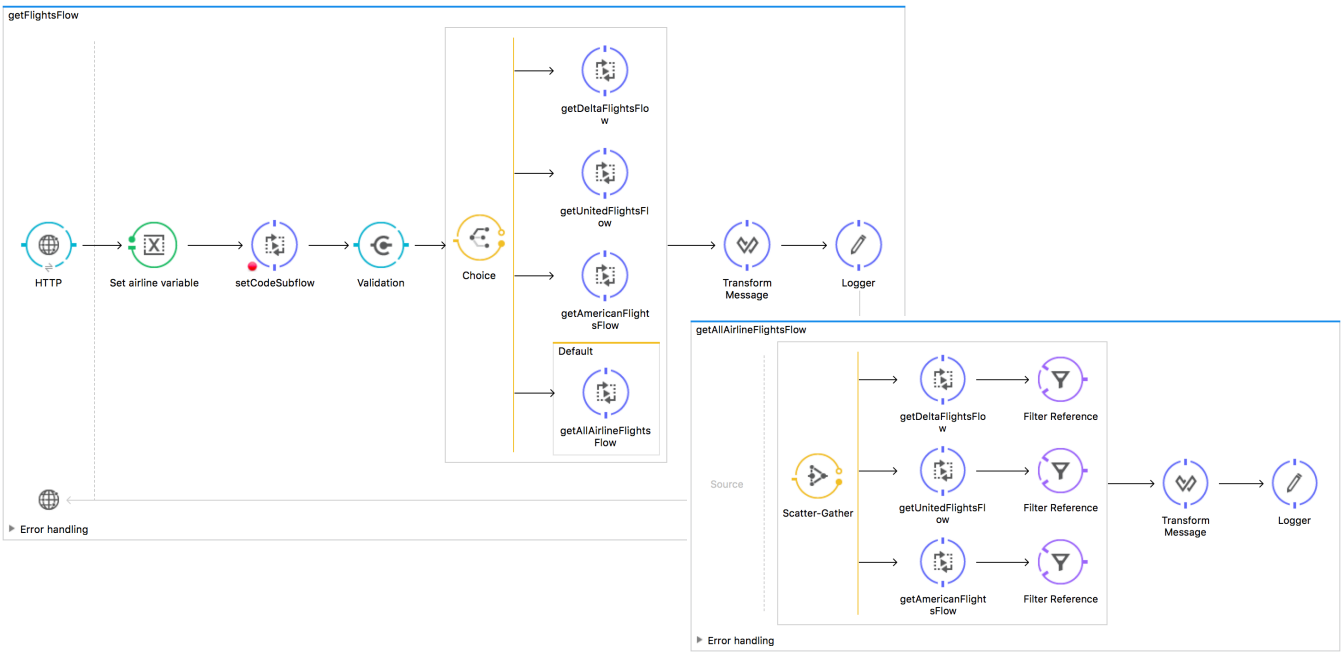


7. Double-click the Set Payload transformer in the 404 exception mapping and look at the settings in the Set Payload properties view.



8. Look at the values for the Set Payload transformers in the other exception mappings.

Module 10: Controlling Message Flow



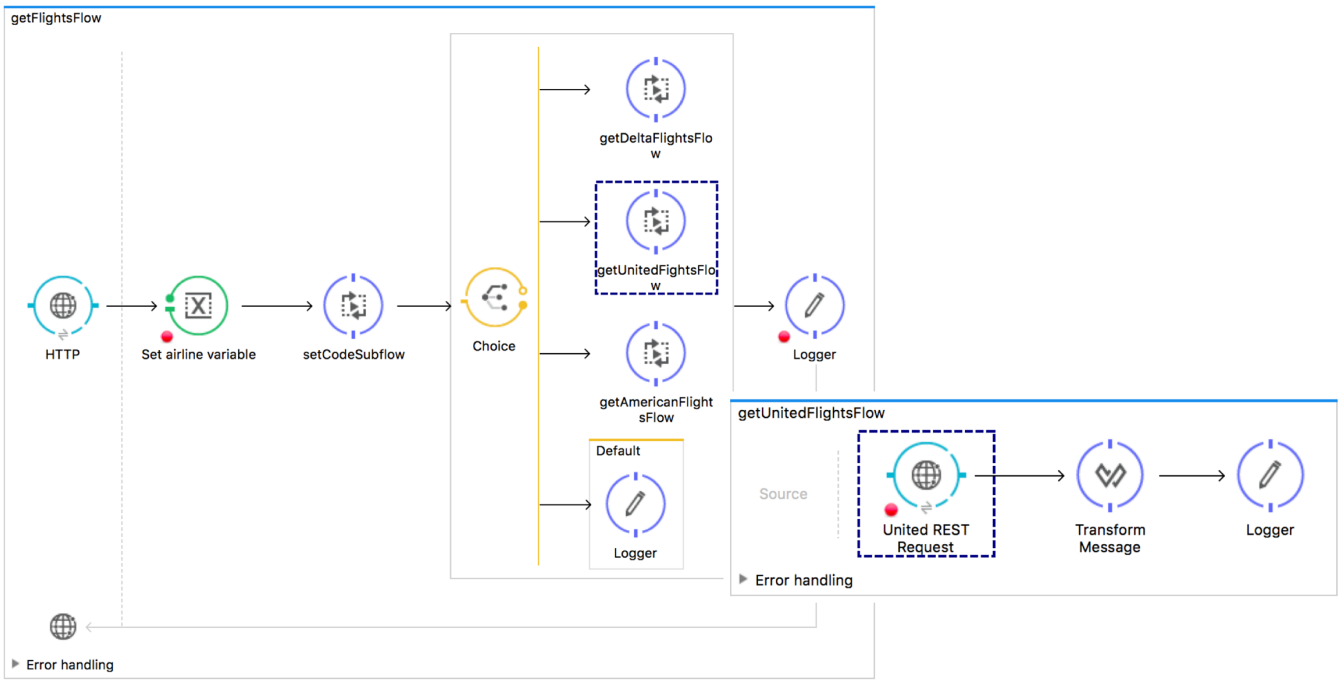
At the end of this module, you should be able to:

- Route messages based on conditions.
- Multicast messages.
- Filter messages.
- Validate messages.

Walkthrough 10-1: Route messages based on conditions

In this walkthrough, you modify getFlightsFlow to route messages to either the United, Delta, or American flows based on the value of an airline query parameter. You will:

- Use a Choice router.
- Set the router paths.



Look at possible airline values specified in the API

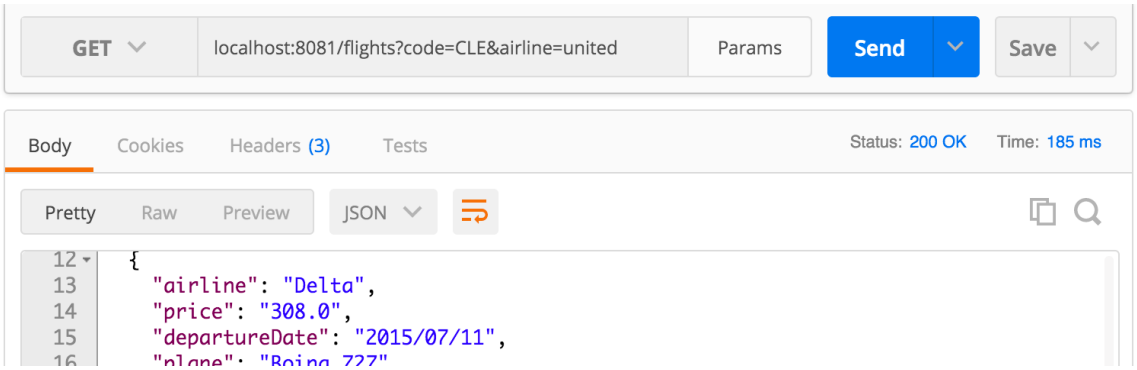
1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Open mua-flights-api.raml.
3. Locate the airline query parameter and its possible values.

```
/flights:
  get:
    displayName: Get flights
    queryParameters:
      code:
        displayName: Destination airport code
        required: false
        enum: [SFO, LAX, PDX, CLE, PDF]
      airline:
        displayName: Airline
        required: false
        enum: [united, delta, american]
```

4. Run the project.

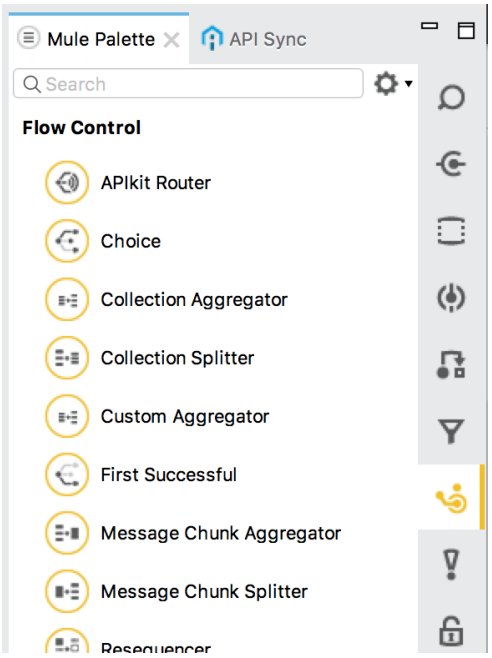
Test the application

5. In Postman, make a request to http://localhost:8081/flights?code=CLE; you should see only Delta flights to CLE.
6. Add a query parameter called airline and set it equal to united.
7. Send the request again; you should still only see the Delta flights.



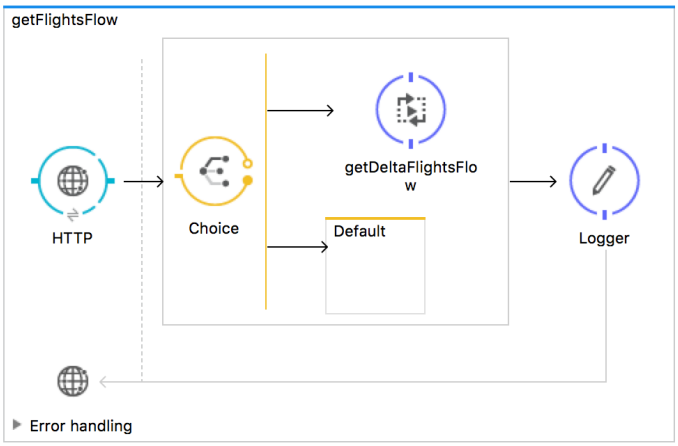
Browse the flow control elements in the Mule Palette

8. Return to implementation.xml in Anypoint Studio.
9. In the Mule Palette, select the Flow Control tab.
10. View the available flow control processors.

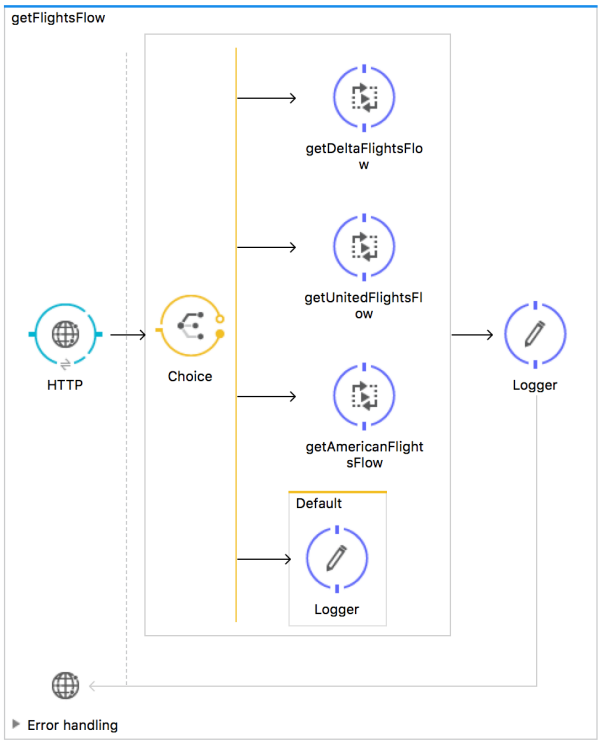


Add a Choice router

11. Drag a Choice flow control element from the Mule Palette and drop it in getFlightsFlow before getDeltaFlightsFlow.
12. Drag the getDeltaFlightsFlow flow reference into the router.



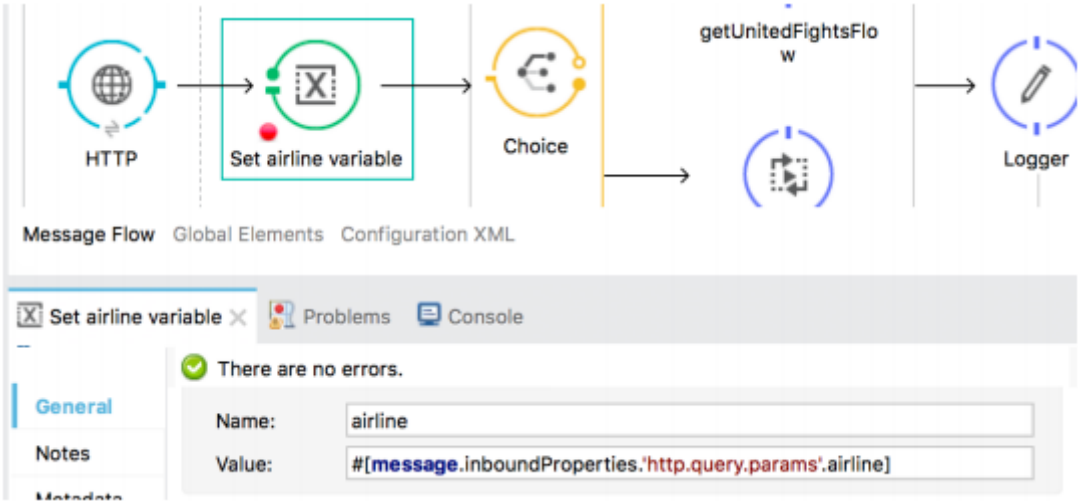
13. Add two additional Flow Reference components to the Choice router.
14. In the Properties view for the first flow reference, set the flow name to getUnitedFlightsFlow.
15. In the Properties view for the second flow reference, set the flow name to getAmericanFlightsFlow.
16. Drag a Logger component from the Mule Palette and drop it in the default branch.



Store the airline query parameter in a flow variable

17. Add a Variable transformer before the Choice router.
18. Change its display name to Set airline variable.
19. Set the flow variable name to airline and set it equal to a query parameter called airline.

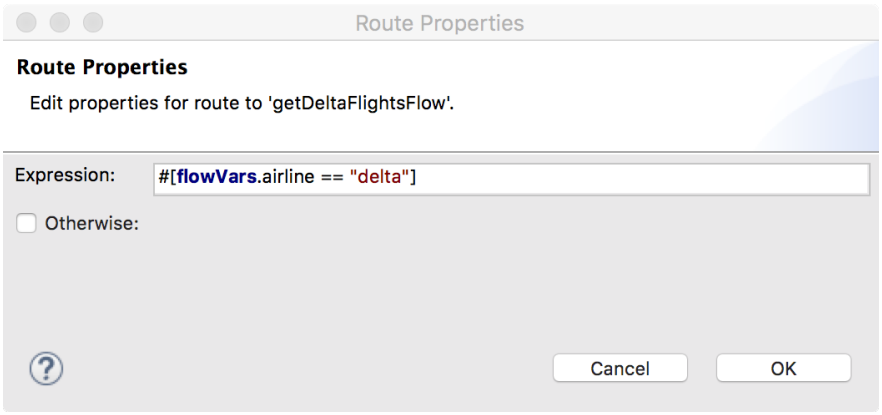
```
#[message.inboundProperties.'http.query.params'.airline]
```



Configure the Choice router

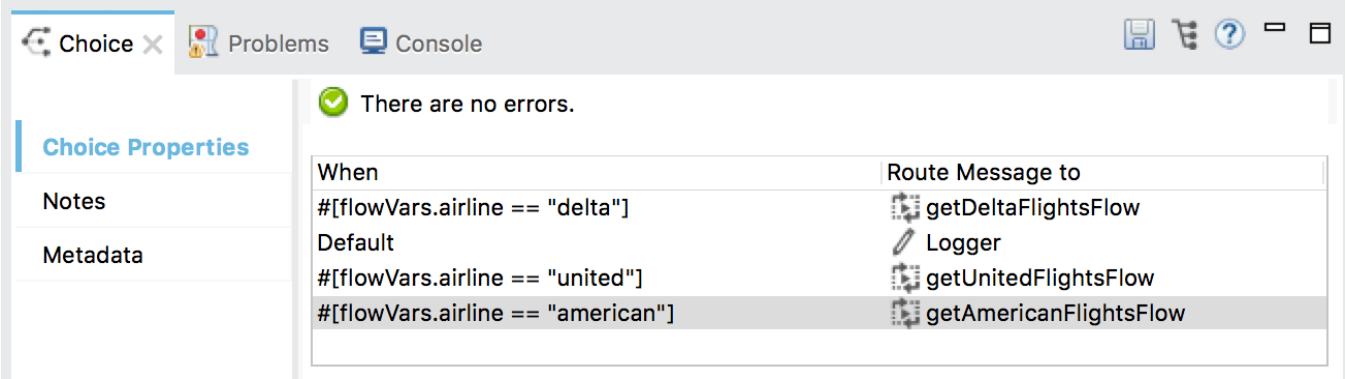
20. In the Choice properties view, double-click the getDeltaFlightsFlow route.
21. In the Route Properties dialog box, set the expression to true if the airline flow variable is equal to delta and click OK.

```
#[flowVars.airline == "delta"]
```



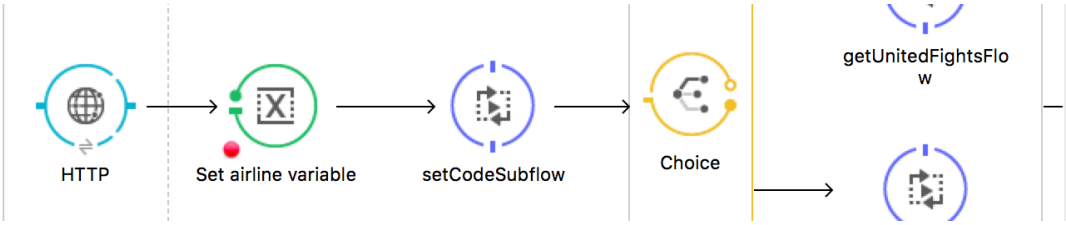
22. Set a similar expression for the United route, routing to it when flowVars.airline is equal to united.

23. Set a similar expression for the American route, routing to it when flowVars.airline is equal to american.

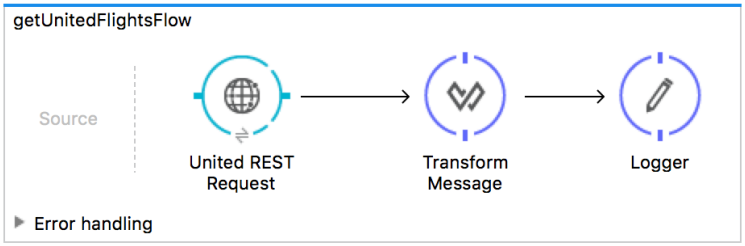


Route all requests through the router

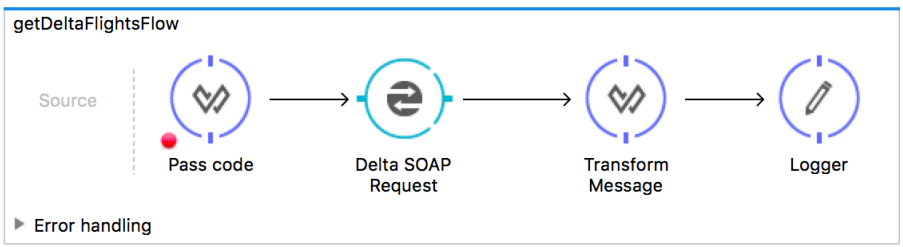
24. From getUnitedFlightsFlow, drag the setCodeSubflow flow reference into getFlightsFlow before the choice router.



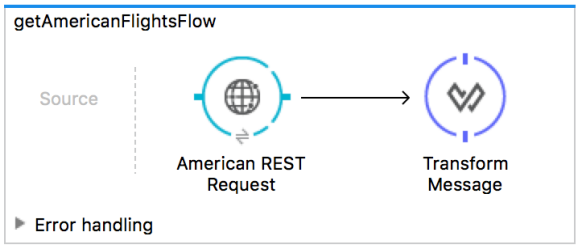
25. In getUnitedFlightsFlow, delete the HTTP Listener endpoint.



26. In getDeltaFlightsFlow, delete the HTTP Listener endpoint and the setCodeSubflow flow reference.

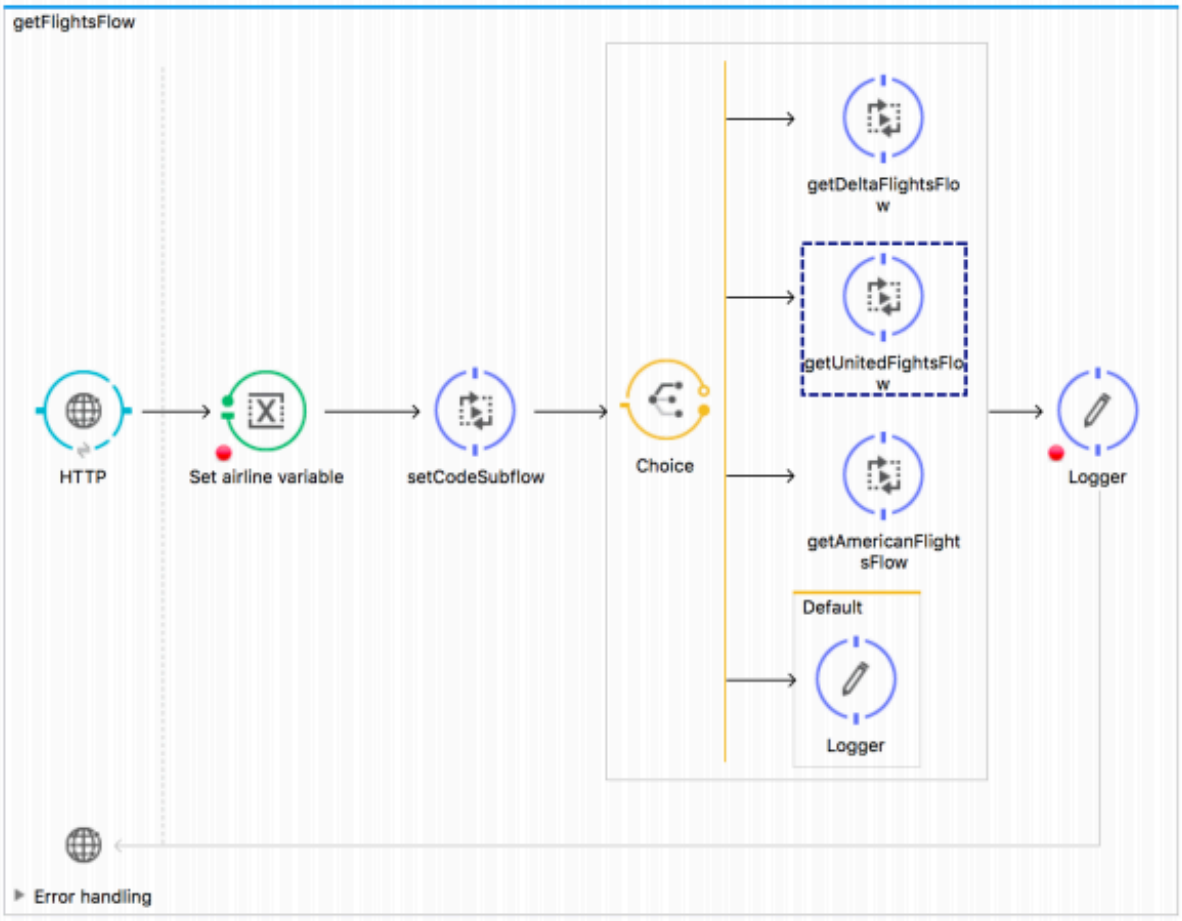


27. In getAmericanFlightsFlow, delete the HTTP Listener endpoint and the setCodeSubflow flow reference.

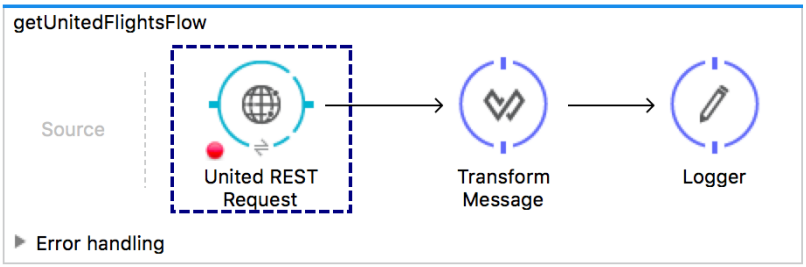


Test the application

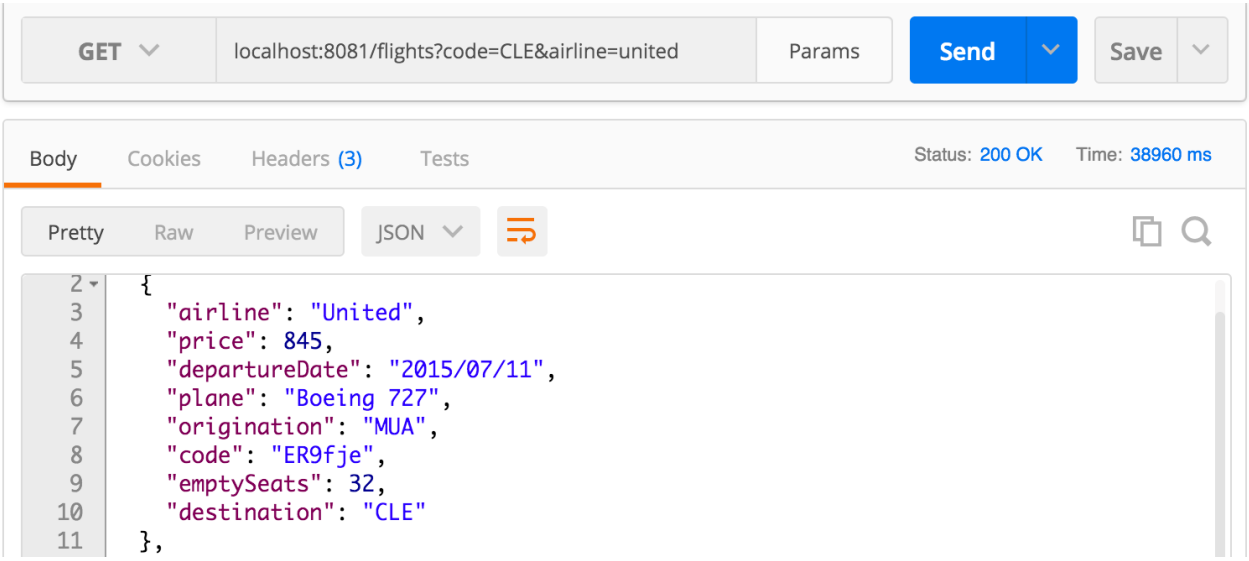
28. In getFlightsFlow, make sure there is a breakpoint on one of the processors before the Choice router.
29. Debug the project.
30. In Postman, make the same request to http://localhost:8081/flights?code=CLE&airline=united.
31. In the Mule Debugger, step through the application; you should see the Choice router pass the message to the United branch.



32. Step through the rest of the application; you should see the message passed to getUnitedFlightsFlow and then back to getFlightsFlow.

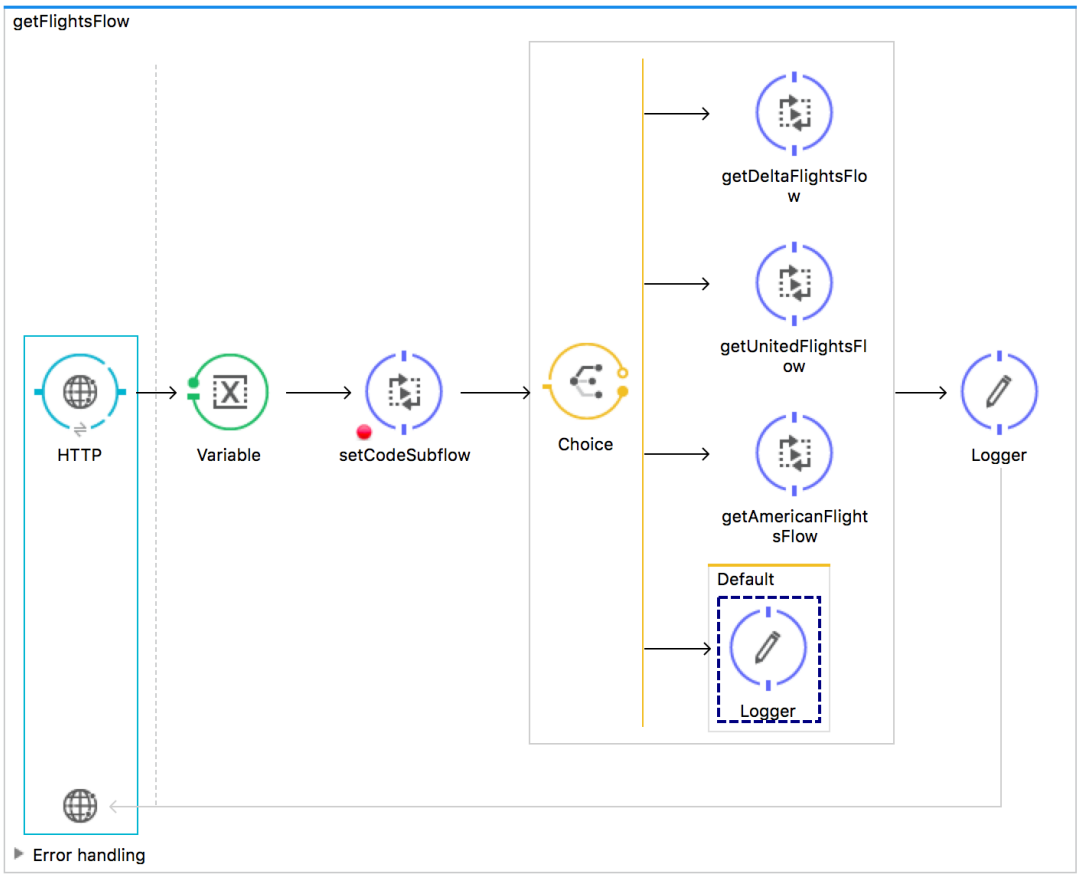


33. Return to Postman; you should see only United flights to CLE returned.

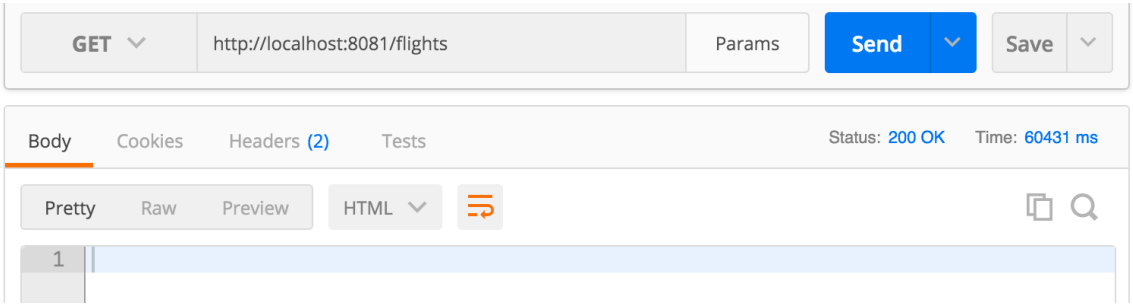


34. Change the airline to delta and the code to LAX.
35. Send the request.
36. Step through the application; the message should be routed to the Delta branch.
37. Return to Postman; you should see only Delta flights to LAX are returned.

38. In Postman, remove both parameters and make a request to http://localhost:8081/flights.
39. In the Mule Debugger, step through the application; you should see the Choice router pass the message to the default branch.



40. Click the Resume button.
41. Return to Postman; no flights are returned.



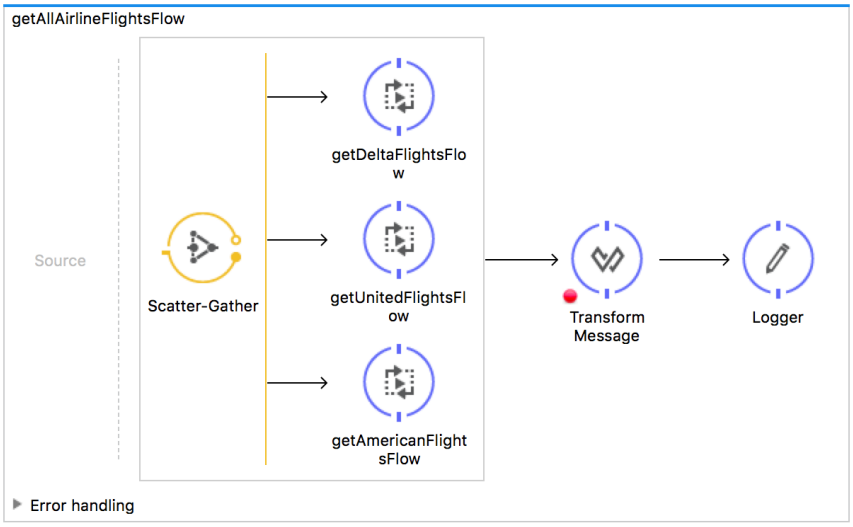
Note: If the next walkthrough, you will change this behavior so that if no airline is specified, flights for all airlines will be returned.

42. Return to Anypoint Studio, stop the project, and switch to the Mule Design perspective.

Walkthrough 10-2: Multicast a message

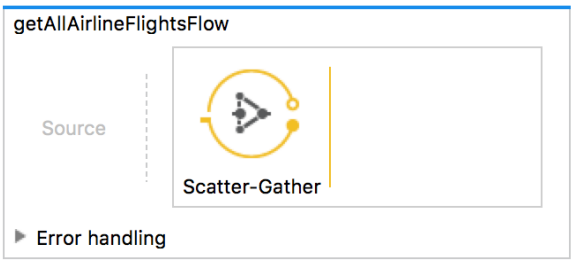
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.
- Use DataWeave to sort the flights by price and return them as JSON.
- (Optional) Modify the airline flows to use DataWeave to each return a Java collection of Flight objects.



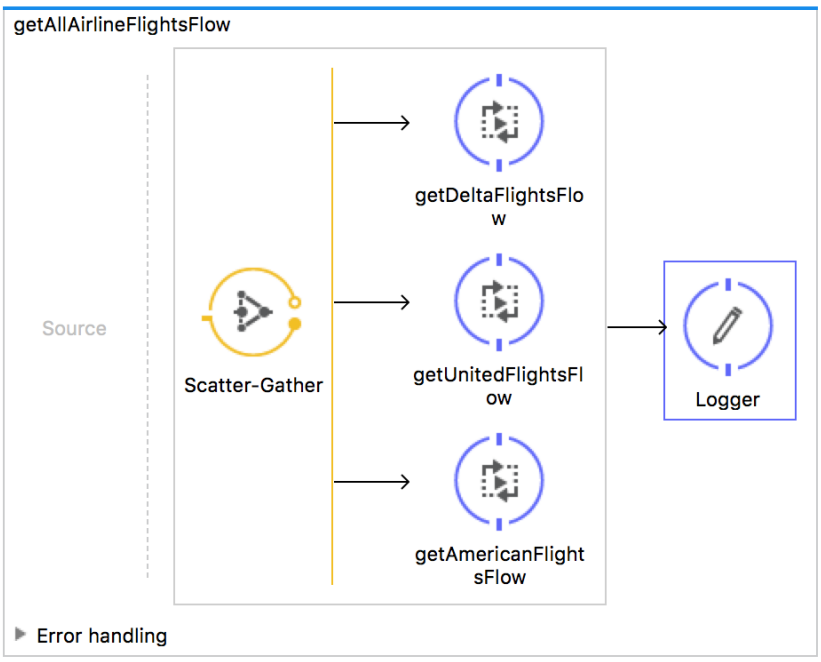
Create a flow to use a router to call all three airline services

1. Return to implementation.xml.
2. Drag a Scatter-Gather flow control element from the Mule Palette and drop it at the bottom of the canvas.
3. Change the name of the flow to getAllAirlineFlightsFlow.



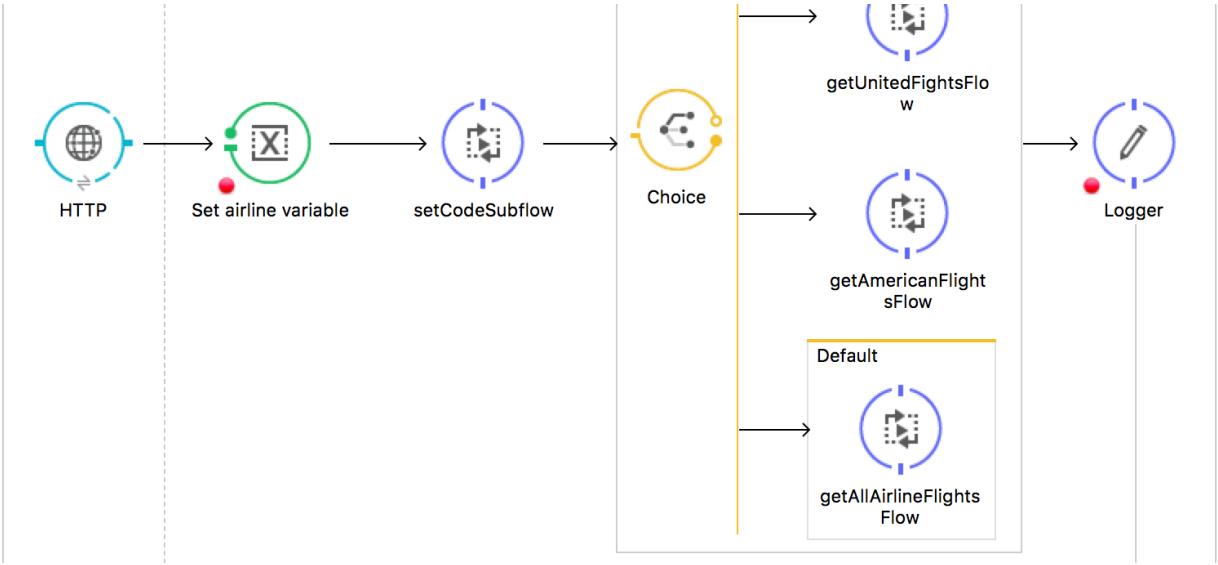
4. Drag three Flow Reference components into the Scatter-Gather router.
5. Using the Properties view, set the flow name of the first Flow Reference to getDeltaFlightsFlow.

6. Set the flow name of the second Flow Reference to getUnitedFlightsFlow.
7. Set the flow name of the third Flow Reference to getAmericanFlightsFlow.
8. Add a Logger after the router.



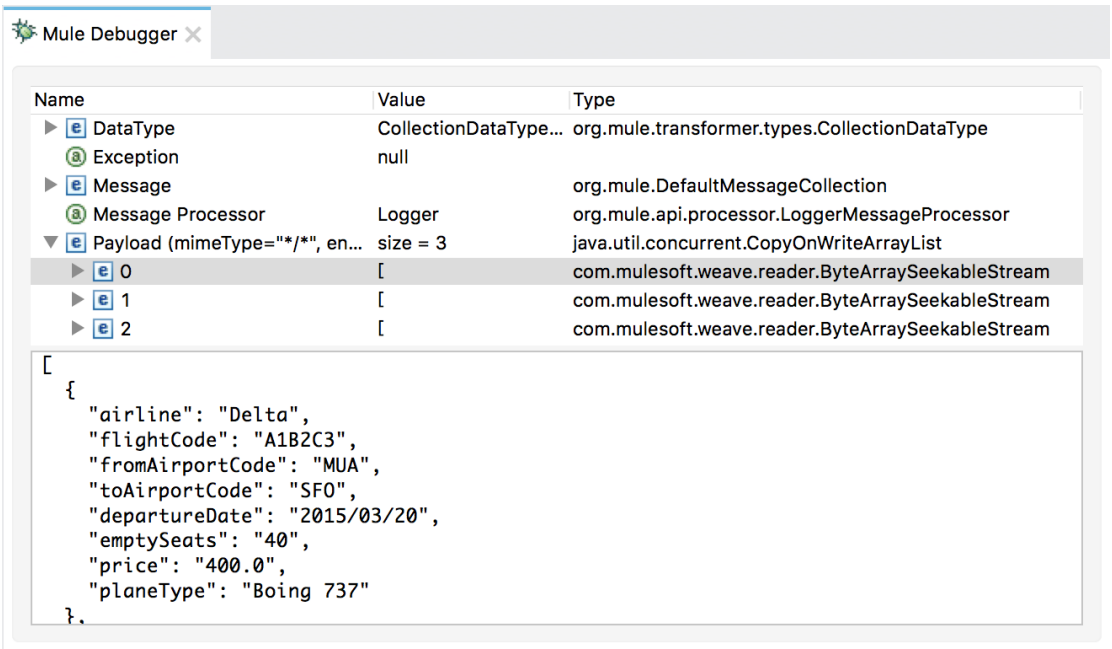
Call this flow if no airline is specified

9. Return to getFlightsFlow at the top of the canvas.
10. Delete the Logger in the default branch.
11. Add a Flow Reference to the default branch and set its flow name to getAllAirlineFlightsFlow.



Test the application

12. Debug the project.
13. In Postman, make the same request to http://localhost/flights.
14. In the Mule Debugger, step through the application to the default branch of the Choice router, then into the Scatter-Gather, and then into each of the airline flows.
15. Stop at the Logger back in getFlightsFlow and look at the payload.



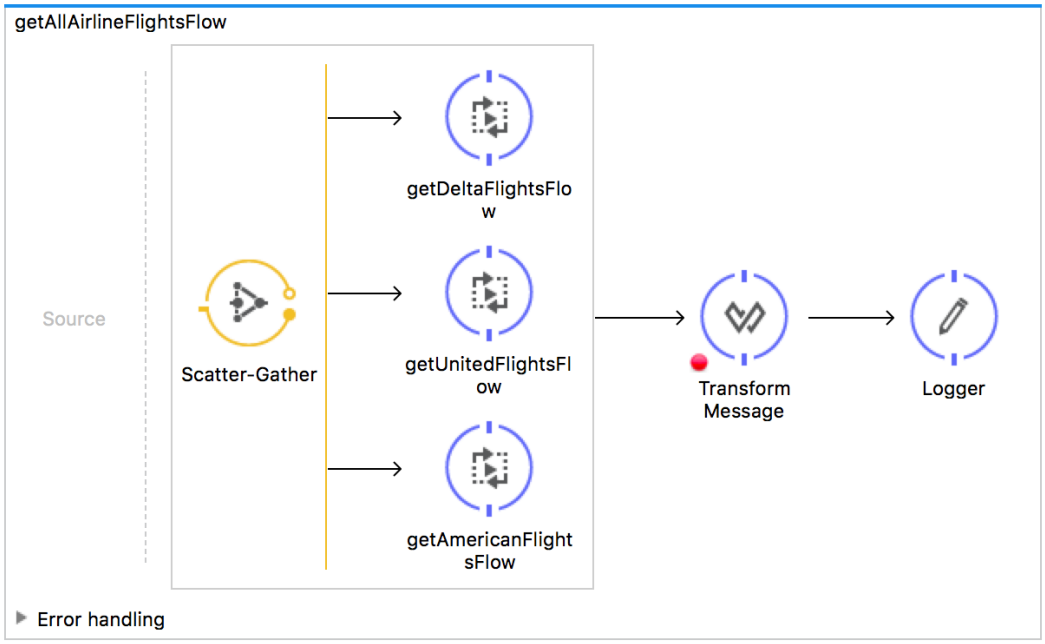
Note: You get three different DataWeave stream objects. You want to combine the three collections and then sort them by price and return them as JSON. Because the airline flight flows were written first and already return JSON, you can just flatten the collections into one and then order by price. More typically, however, you would have each of the airline flows return data of a common canonical format – in this case, a collection of Flight Java objects – and then flatten, order, and transform that data to the JSON specified by the API.

16. Click the Resume button.
17. Stop the project.

Flatten the combined results

18. Return to getAllAirlineFlightsFlow in the Mule Design pespective.

19. Add a Transform Message component before the Logger.



20. In the Properties view, set the DataWeave expression to flatten the payload.

```
flatten payload
```

```
1 %dw 1.0
2 %output application/java
3 ---
4 flatten payload
```

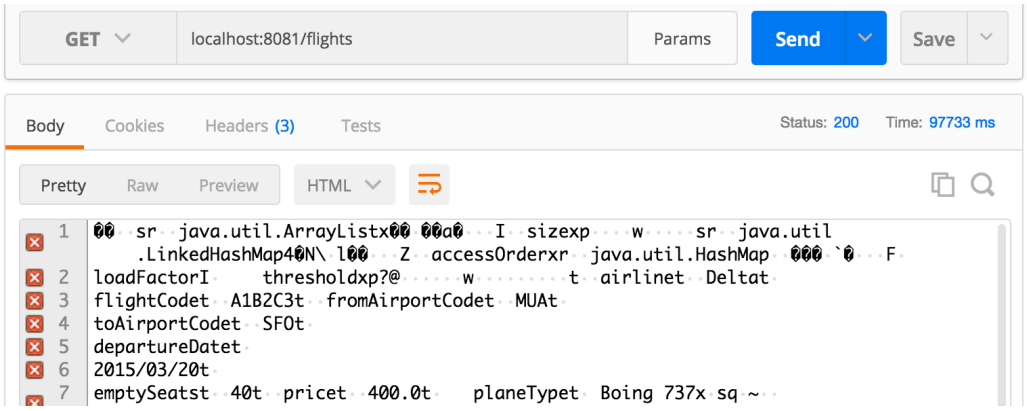
Note: You will learn about DataWeave operators in the later module Writing DataWeave Transformations.

Test the application

21. Save the file to redeploy the application.
22. In Postman, make the same request to http://localhost:8081/flights.

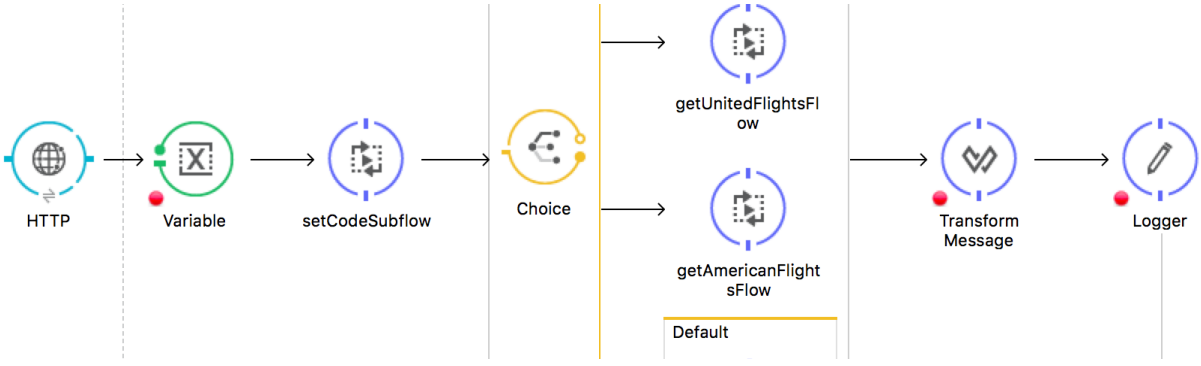
23. In the Mule Debugger, step through the application to the Logger after the Choice router; you should see the payload is now one ArrayList of HashMaps.

24. Step through the rest of the application and stop the project.
25. In Postman, you should get a representation of Java objects returned.

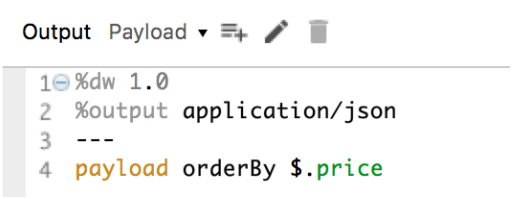


Return the data as JSON and sort by price

26. Return to getFlightsFlow in the Mule Design perspective.
27. Add a Transform Message component after the Choice router.



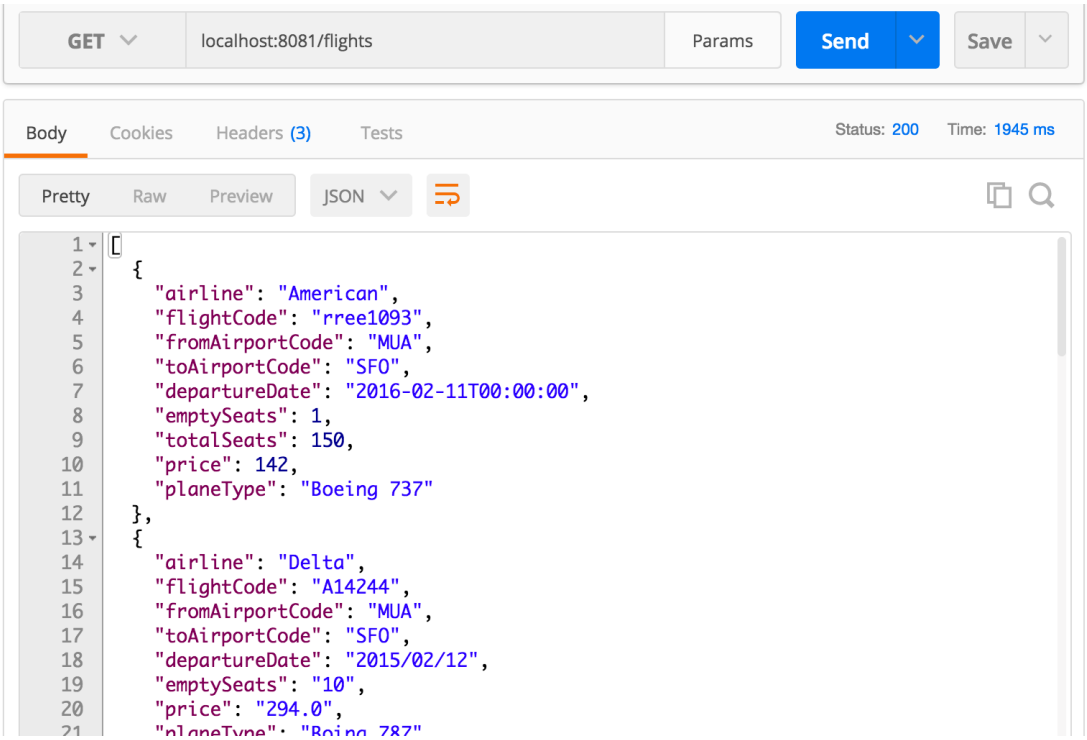
28. In the Transform Message properties view, change the output to application/json.
29. Change the transformation expression to order the payload by price.



Note: You will learn about DataWeave operators in the next module, Writing DataWeave Transformations.

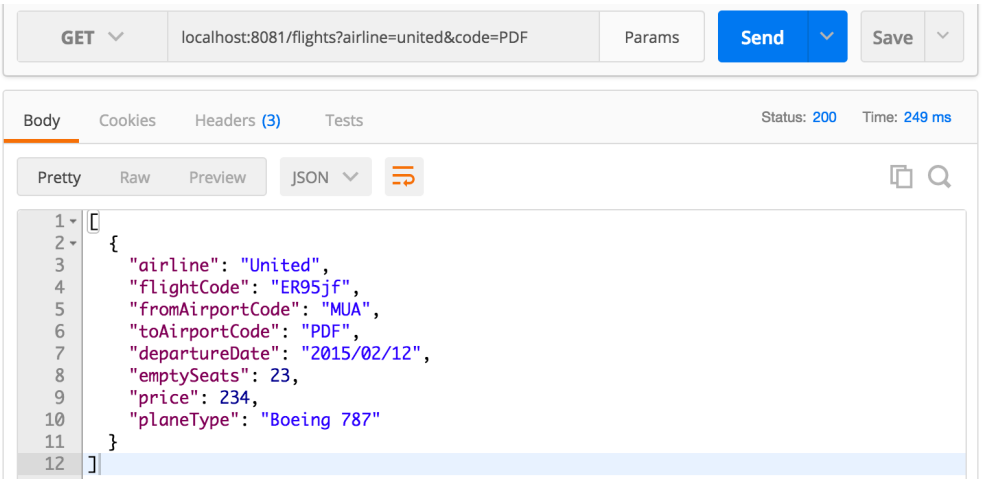
Test the application

30. Run the project.
31. In Postman, send the same request to http://localhost:8081/flights; you should get all the flights to SFO returned and they should be sorted by price.

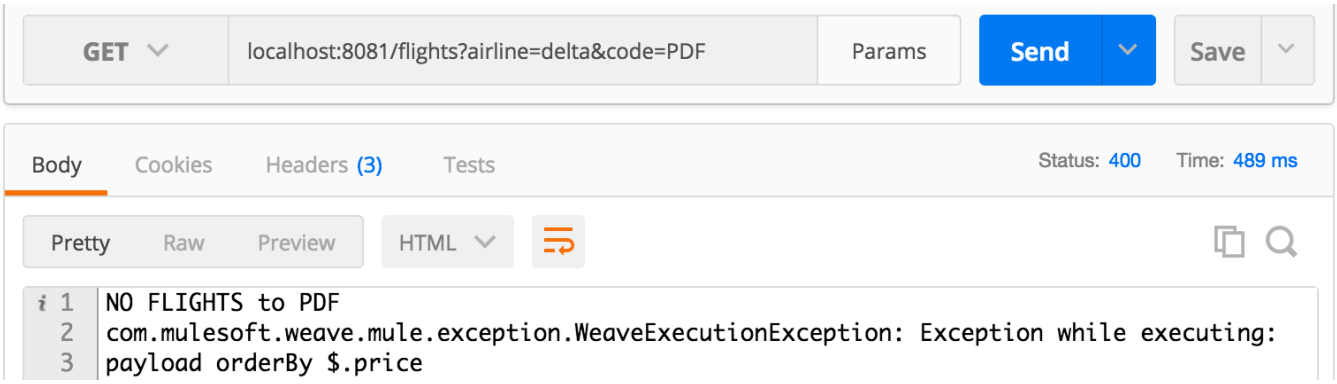


32. Add an airline parameter equal to united and send the request: http://localhost:8081/flights?airline=united; you should get united flights to SFO sorted by price.

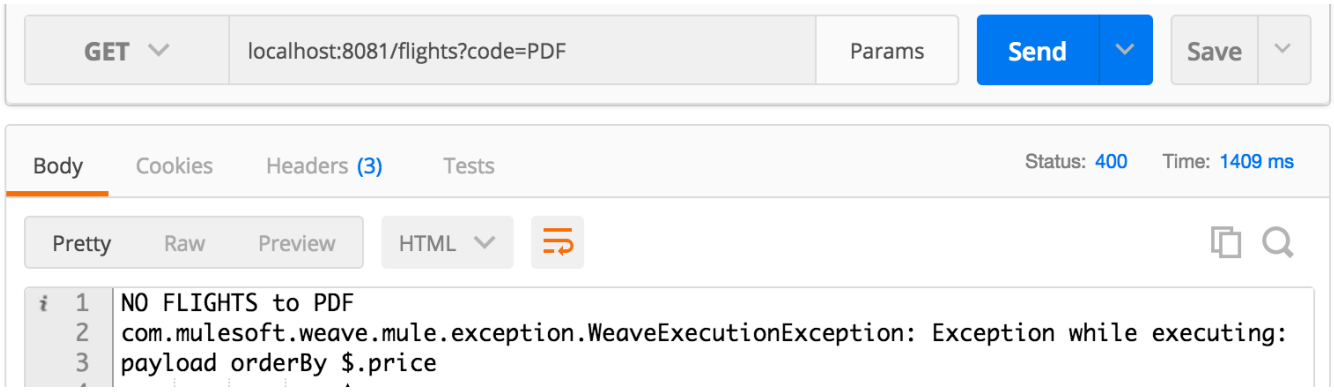
33. Add a code parameter equal to PDF and send the request: http://localhost:8081/flights?airline=united&code=PDF; you should get one united flight to PDF.



34. Change the airline to delta and send the request: http://localhost:8081/flights?airline=delta&code=PDF; you should the message that there are not flights to PDF – which is correct.



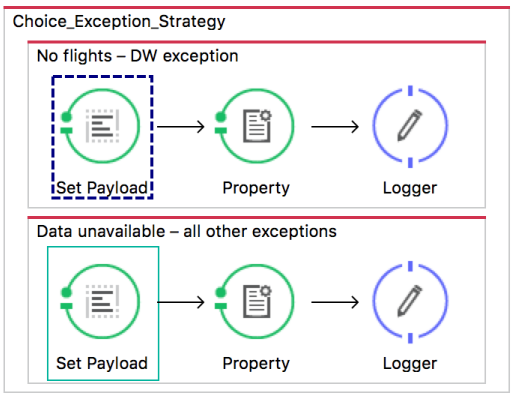
35. Remove the airline parameter and send the request: http://localhost:8081/flights?code=PDF; you should get the one United flight to PDF, but instead you get the message that there are no flights.



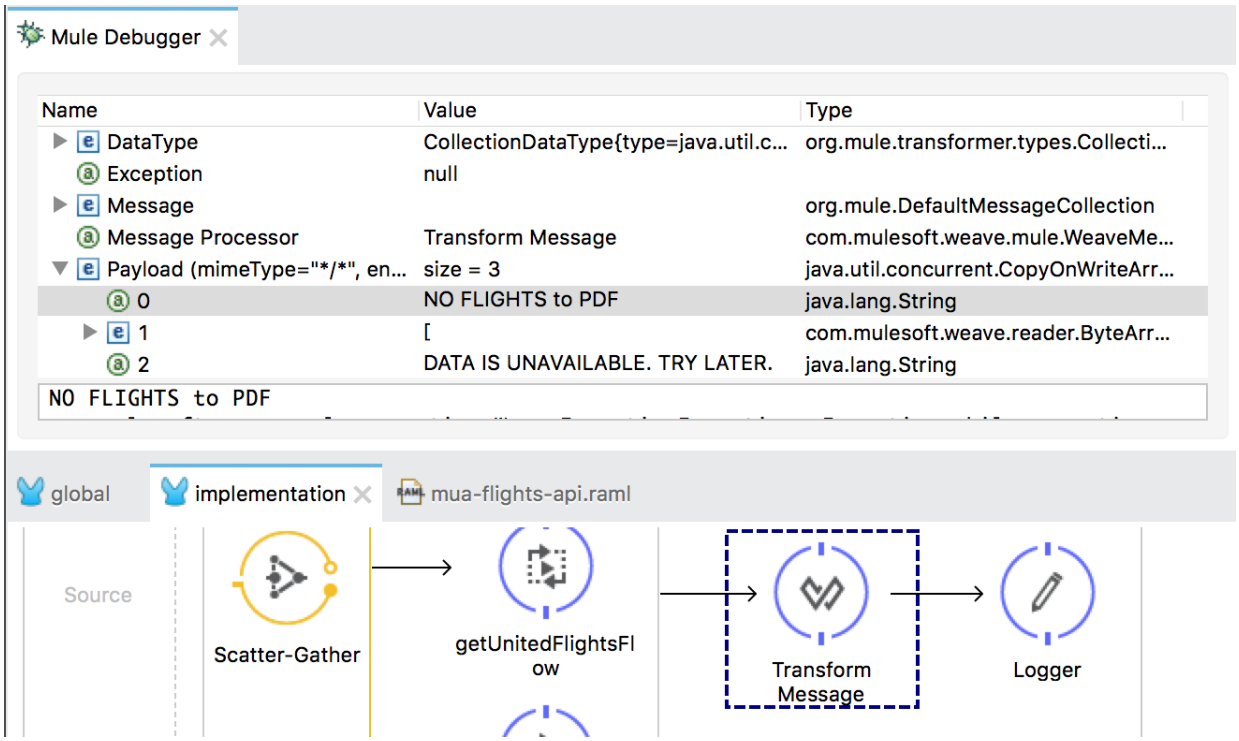
36. Return to Anypoint Studio and stop the project.

Debug the application

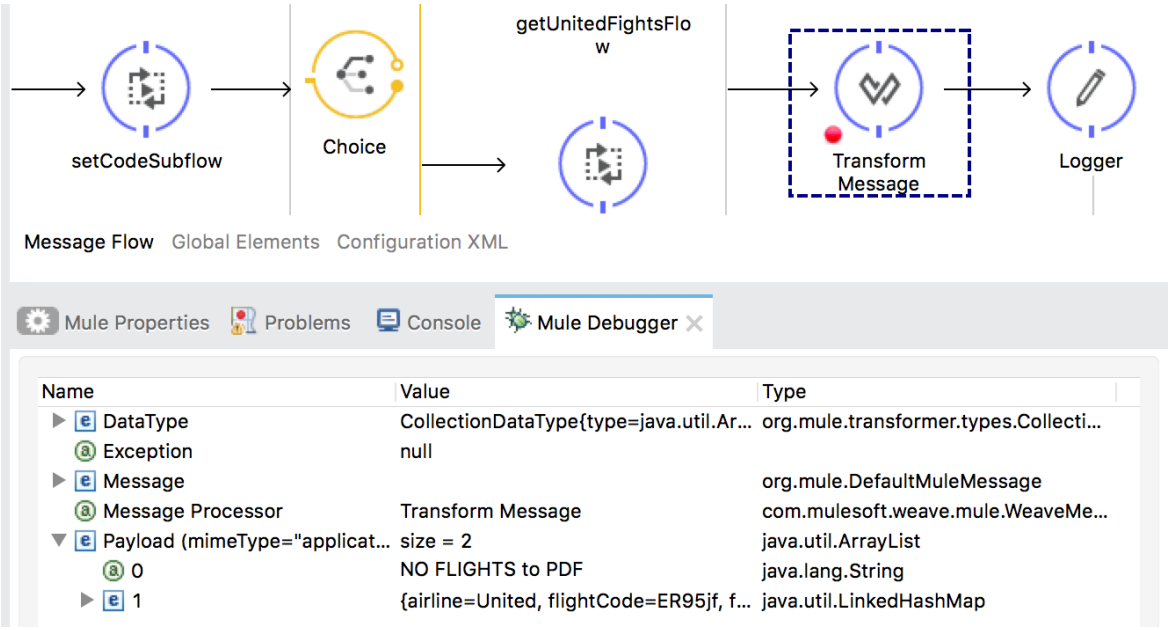
37. Debug the project.
38. In Postman, send the same request to http://localhost:8081/flights?code=PDF.
39. In the Mule Debugger, step through the application until an exception is thrown – and handled by the default global exception handler.



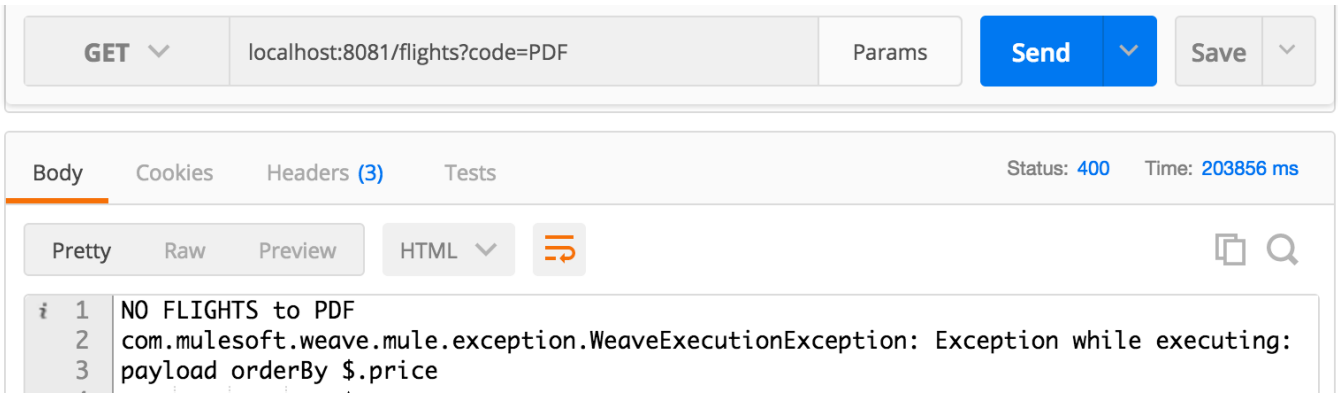
40. Continue to step through the application until you reach the Transform Message component in getAllAirlineFlightsFlow; you should see different types of objects returned.



41. Step to the Transform Message component in getFlightsFlow; you should see the payload still contains the error message in addition to the valid flight results to PDF.



42. Step to the end of the application; you should not get the United results to PDF.



Note: You could write a custom aggregation strategy for the Scatter-Gather router with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages in the next walkthrough.

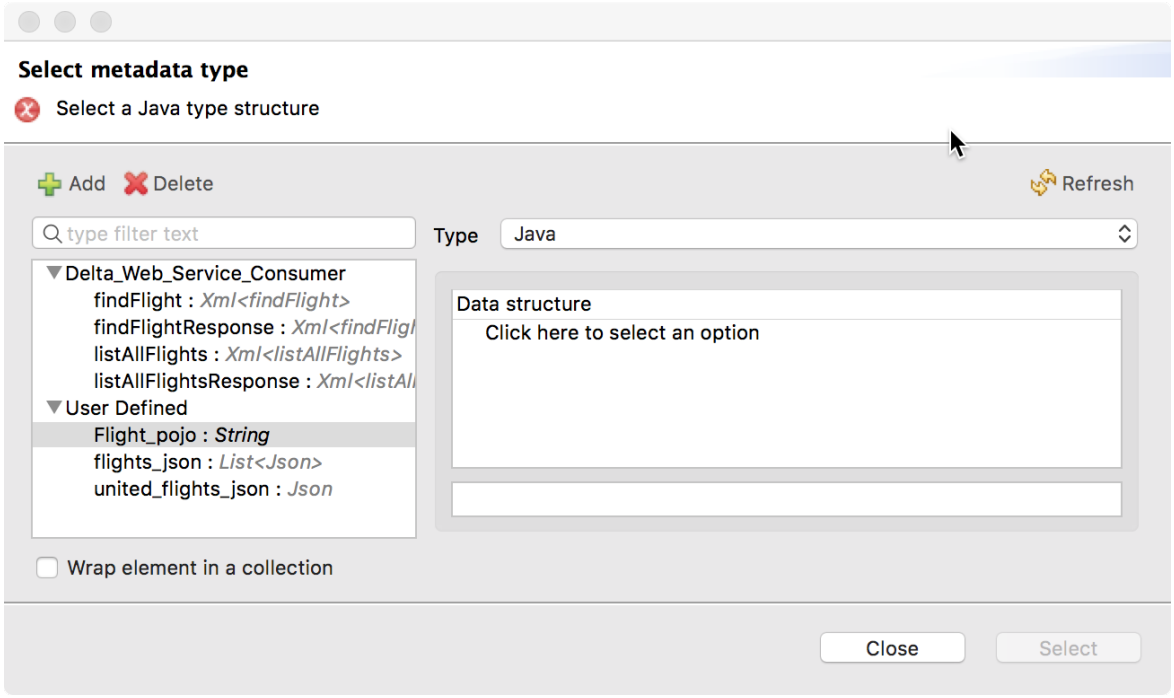
43. Return to Anypoint Studio, stop the project, and switch to the Mule Design perspective.

(Optional) Modify the airline flows to return Java Flight objects instead of JSON

Note: The rest of the walkthrough is optional. It contains steps to modify each of the airline flows to return data of a common canonical format – in this case, a collection of Flight Java objects.

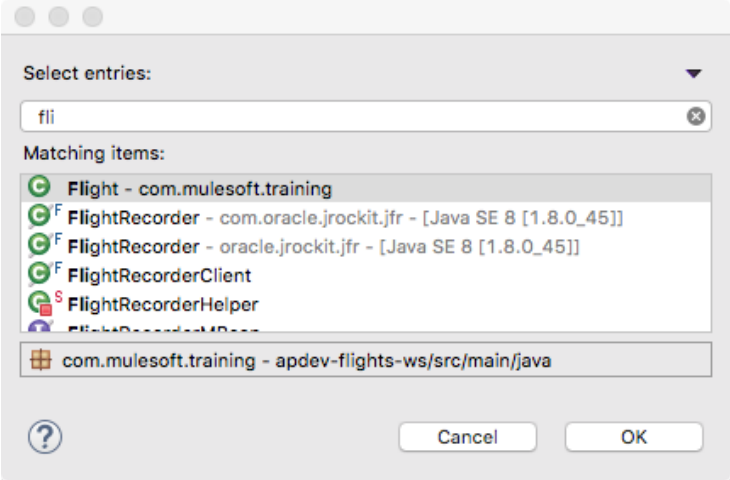
Change the United airline flows to return Java Flight objects instead of JSON

44. Return to getUnitedFlightsFlow in implementation.xml.
45. In the Transform Message properties view, right-click List <Json> in the output section and select Clear Metadata.
46. Click the Define metadata link.
47. In the Select metadata type dialog box, click the Add button.
48. In the Create new type dialog box, set the name to Flight_pojo and click Create type.
49. In the Select metadata type dialog box, change the type to Java.
50. In the Data structure section, click the Click here to select an option link.

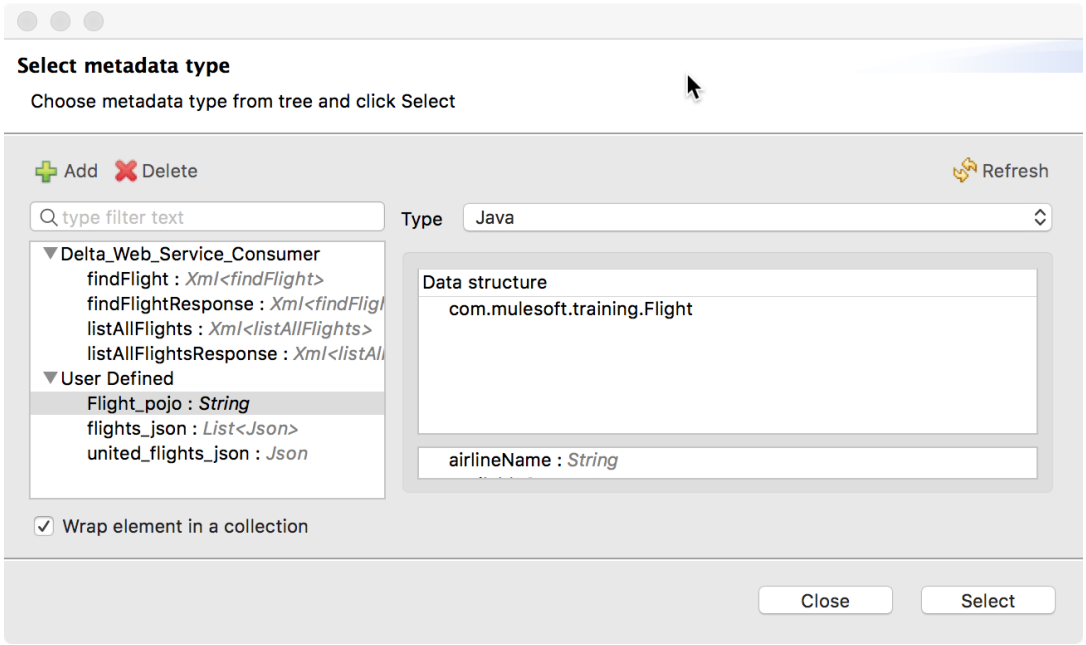


51. In the drop-down menu that appears, select Java object.

52. In the dialog box that opens, type fli and then in the list of classes that appears, select Flight – com.mulesoft.training.com.

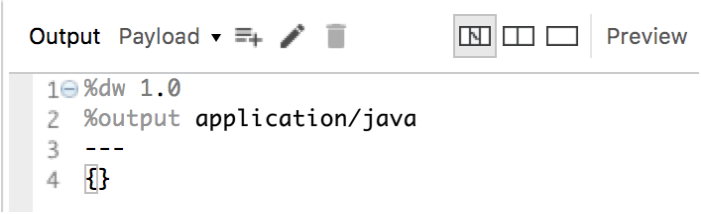


53. Click OK.
54. In the Select metadata type dialog box, select Wrap element in a collection in the lower-left corner.

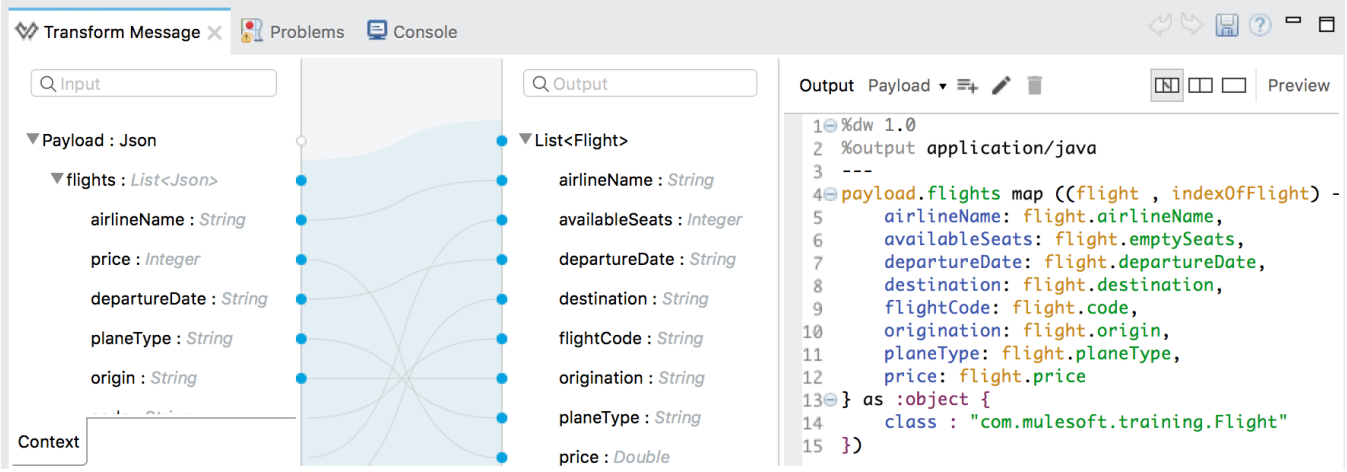


55. Click Select.

56. In the Transform Message properties view, replace the current transformation expression with an empty object.



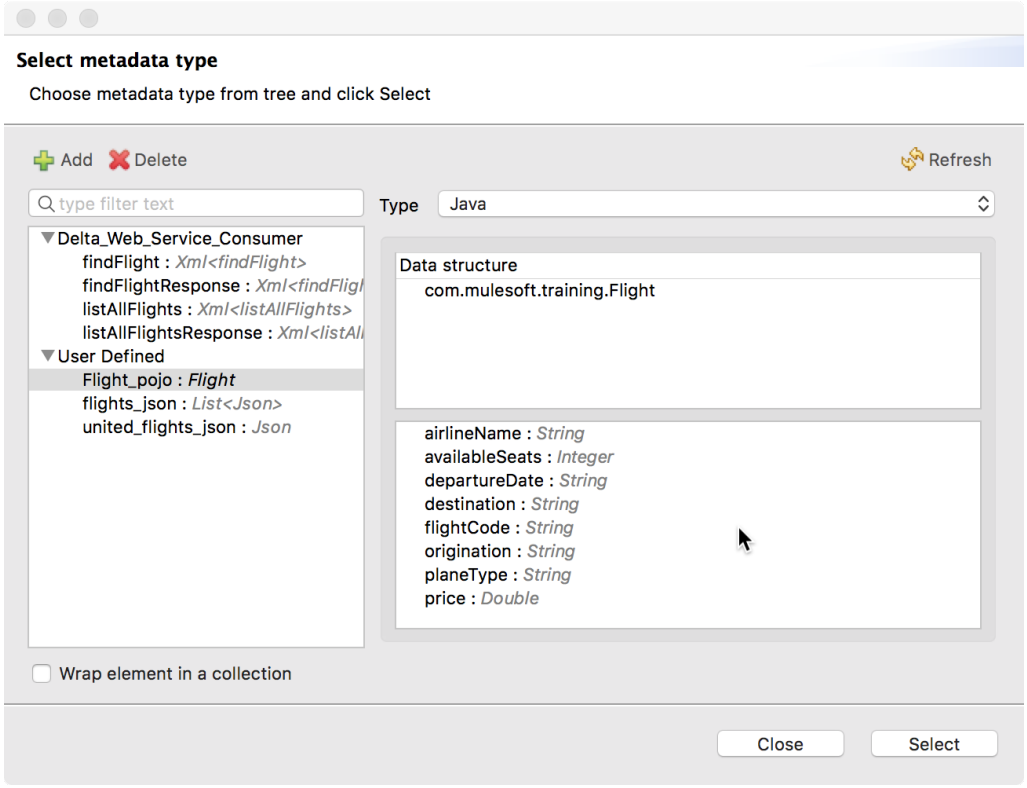
57. Use the graphical editor to map the fields appropriately.



Change the American flow to return Java Flight objects instead of JSON

58. Return to getAmericanFlightsFlow.
59. In the Transform Message properties view, right-click List <Json> in the output section and select Clear Metadata.
60. Click the Define metadata link.
61. In the Select metadata type dialog box, select Flight_pojo.

62. Select the Wrap element in a collection checkbox in the lower-left corner.

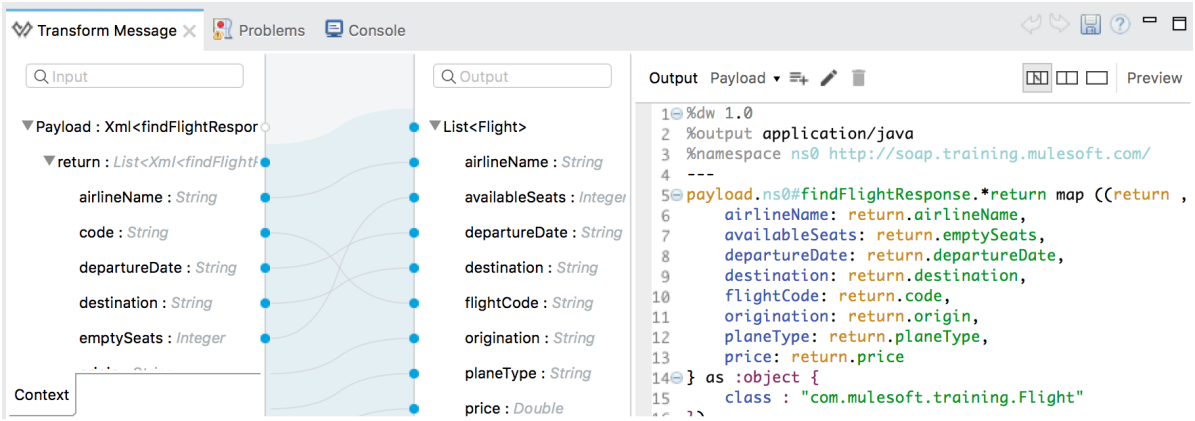


63. Click Select.
64. In the Transform Message properties view, replace the current transformation expression with an empty object.
65. Use the graphical editor to map the fields appropriately.
66. In the output section of the graphical editor, double-click airlineName; the field should be added to the DataWeave expression.
67. Set its value to "American".

```
%dw 1.0
%output application/java
---
payload map ((payload01 , indexOfPayload01) -> {
    airlineName: "American",
    availableSeats: payload01.emptySeats,
    departureDate: payload01.departureDate,
    destination: payload01.destination,
    flightCode: payload01.code,
    origination: payload01.origin,
    planeType: payload01.plane.type,
    price: payload01.price
} as :object {
    class : "com.mulesoft.training.Flight"
})
```

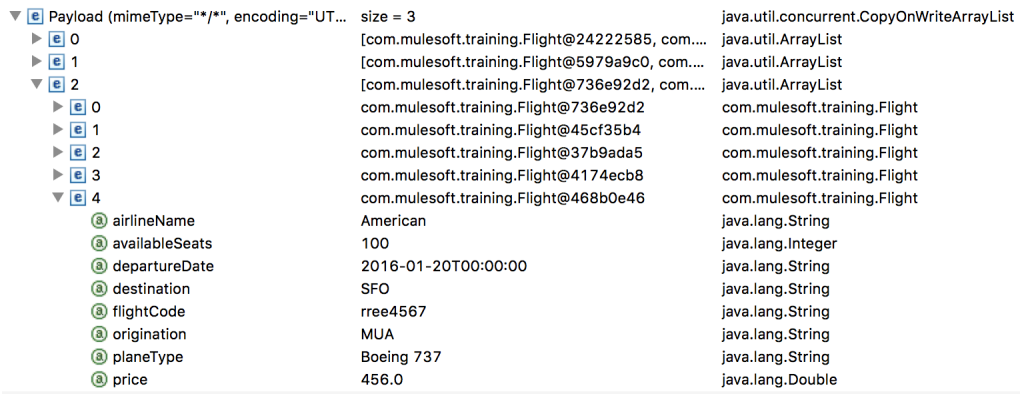
Change the Delta flow to return Java Flight objects instead of JSON

68. Return to getDeltaFlightsFlow.
69. In the Transform Message Properties view for the component after the Delta SOAP Request, right-click List <Json> in the output section and select Clear Metadata.
70. Click the Define metadata link.
71. In the Select metadata type dialog box, select Flight_pojo.
72. Select the Wrap element in a collection checkbox in the lower-left corner.
73. Click Select.
74. In the Transform Message properties view, replace the current transformation expression with an empty object.
75. Use the graphical editor to map the fields appropriately.

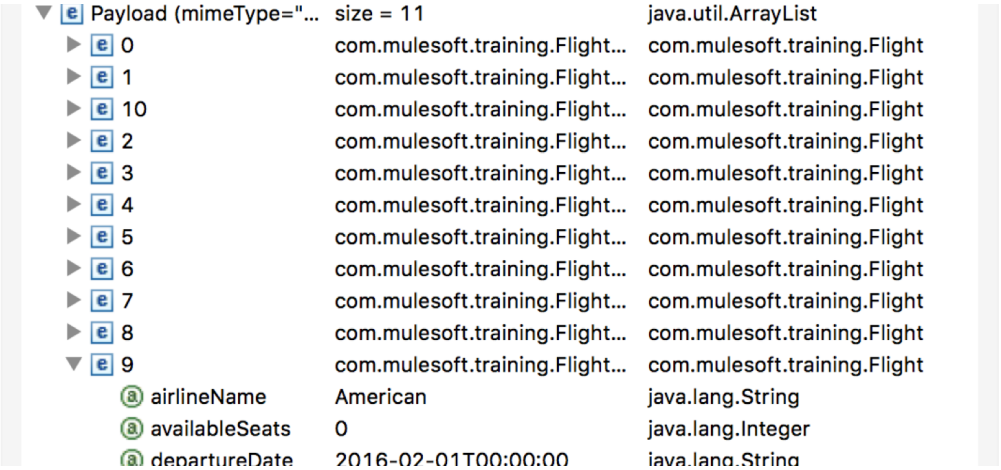


Test the application

76. Debug the project.
77. In Postman, remove the parameters and send a request to http://localhost:8081/flights.
78. In the Mule Debugger, step through the application to the Transform Message component in getAllAirlineFilghtsFlow; the payload should be a collection of three ArrayLists of Flight objects.



79. Step to the Logger; the payload should now be one ArrayList of Flight objects.

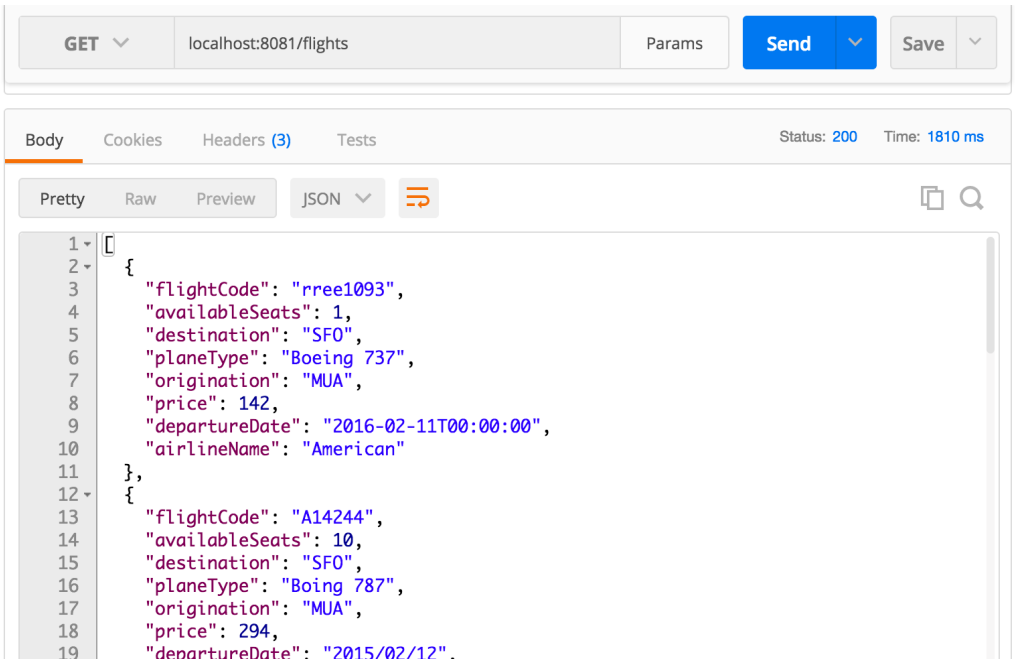


80. Step through to the end of the application.

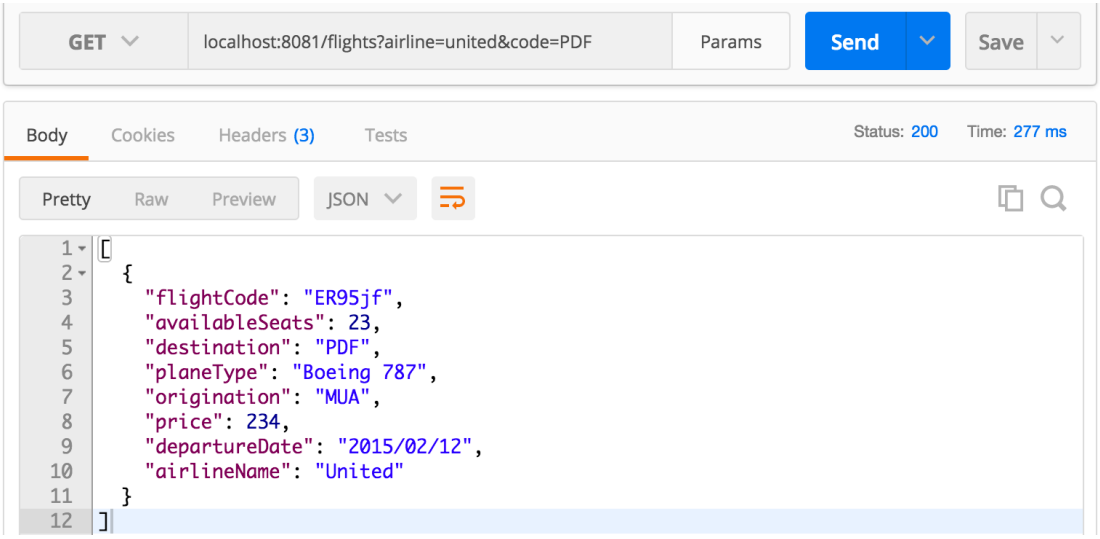
Test the application

Note: You completed the rest of the walkthrough steps already before modifying the airline flows to return collections of Flight objects. You are repeating them as a lead in to the next walkthrough.

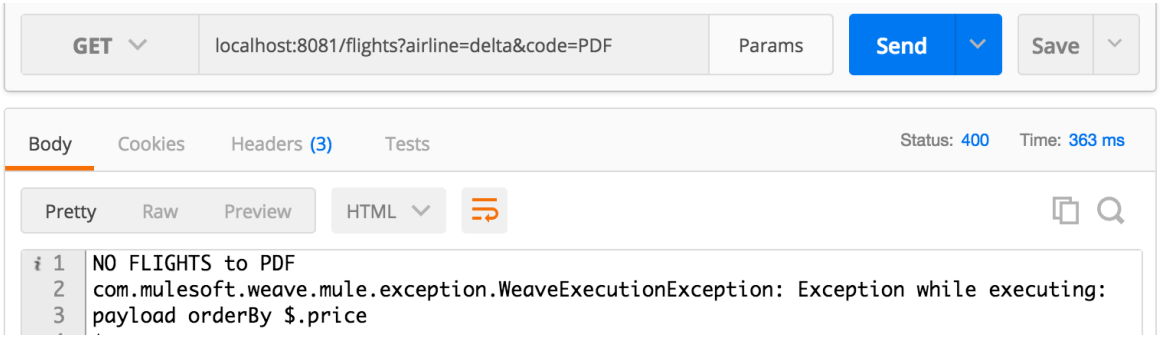
81. Run the project.
82. In Postman, send the same request to http://localhost:8081/flights; you should get all the flights to SFO returned and they should be sorted by price.



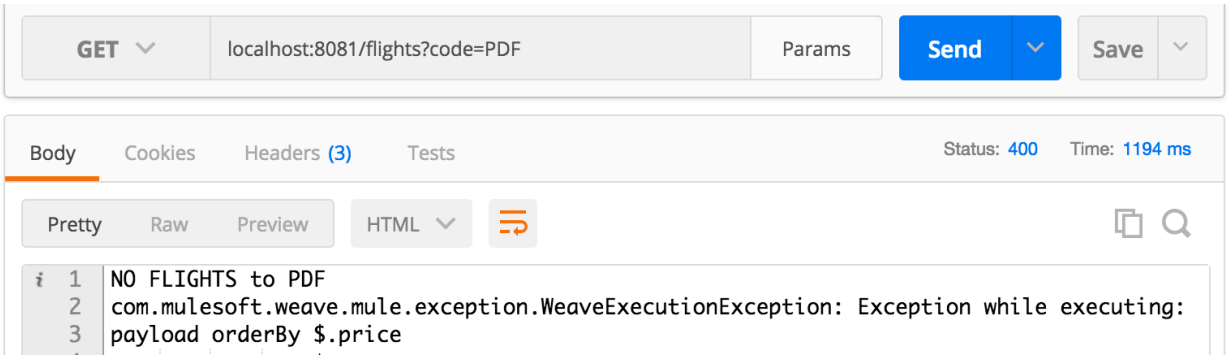
83. Add an airline parameter equal to united and send the request: http://localhost:8081/flights?airline=united; you should get united flights to SFO sorted by price.
84. Add a code parameter equal to PDF and send the request: http://localhost:8081/flights?airline=united&code=PDF; you should get one united flight to PDF.



85. Change the airline to delta and send the request: http://localhost:8081/flights?airline=delta&code=PDF; you should the message that there are not flights to PDF – which is correct.



86. Remove the airline parameter and send the request: http://localhost:8081/flights?code=PDF; you should get the one United flight to PDF, but instead you get the message that there are no flights.

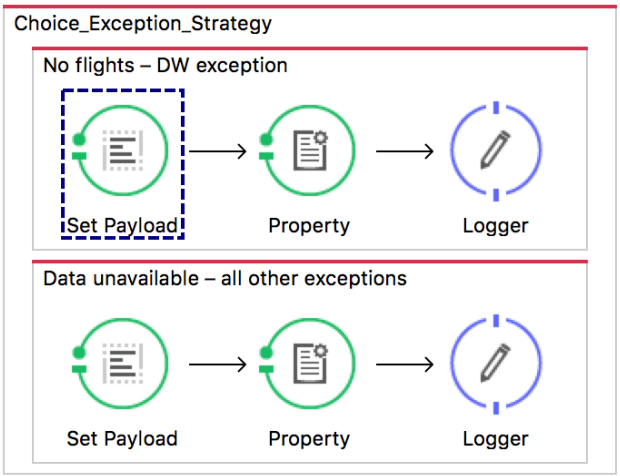


87. Return to Anypoint Studio and stop the project.

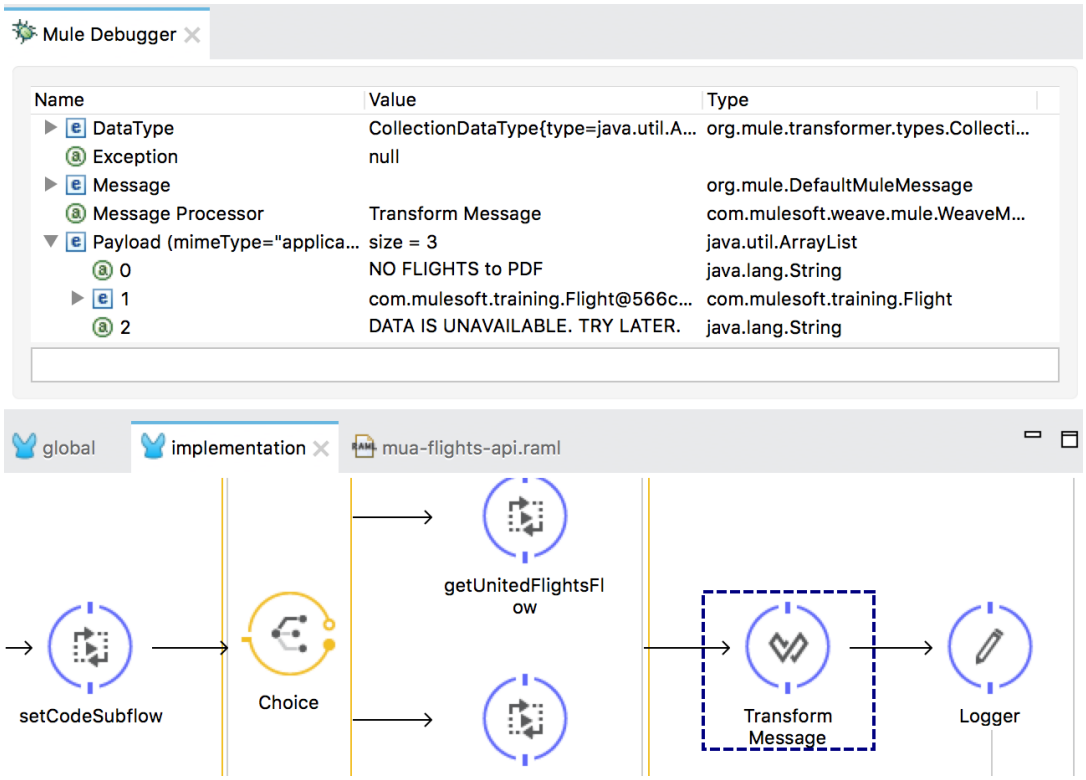
Debug the application

88. Debug the project.
89. In Postman, send the same request to http://localhost:8081/flights?code=PDF.
90. In the Mule Debugger, step through the Delta flow throws an exception that is handled by the default global exception handler.

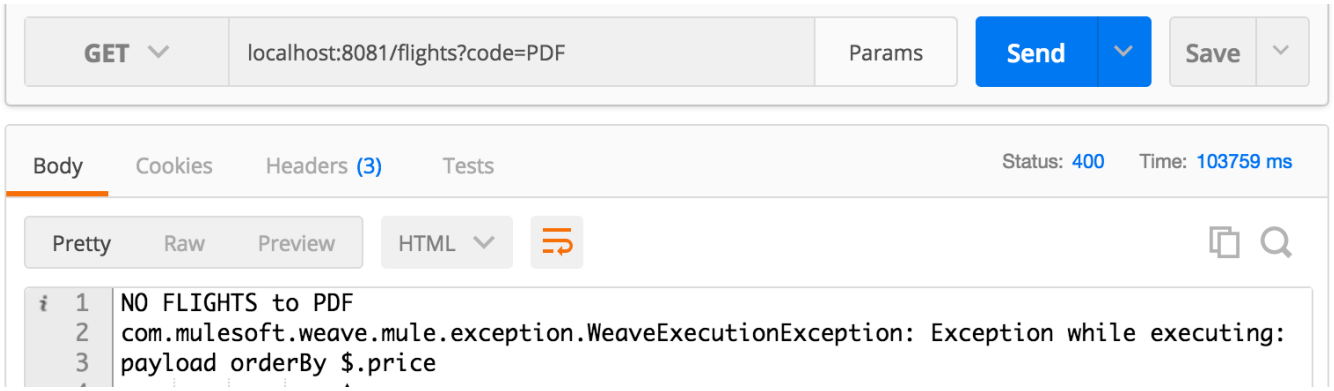
    Note: The American flow also throws an exception that is handled by this choice exception strategy.



91. Continue to step through the application until you reach the Transform Message component in getFlightsFlow; you should see the payload still contains the error messages in addition to the valid flight results to PDF.



92. Step to the end of the application; you should not get the United results to PDF.



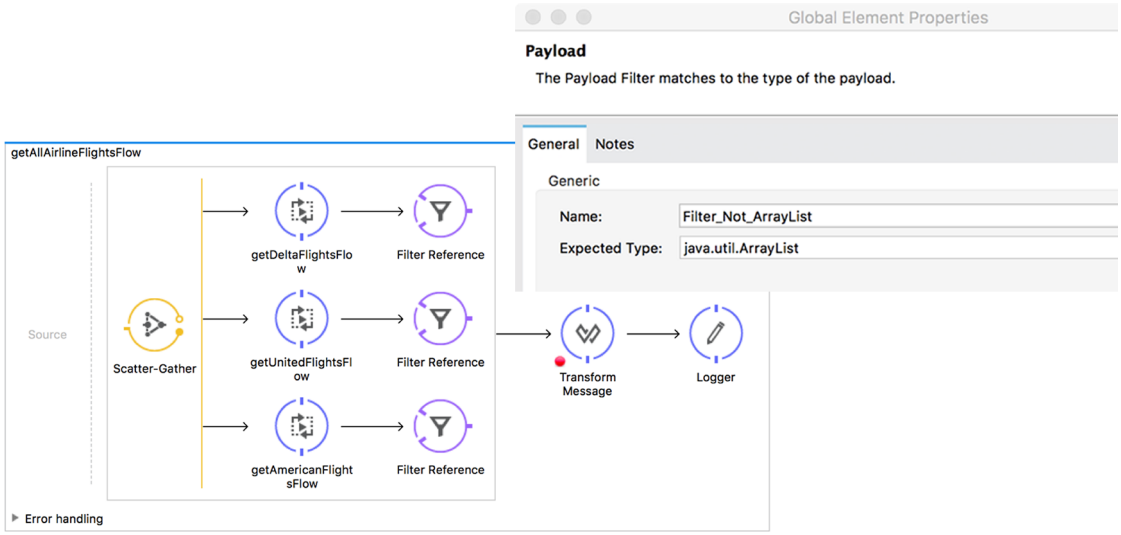
Note: You could write a custom aggregation strategy for the Scatter-Gather router with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages in the next walkthrough.

93. Return to Anypoint Studio, stop the project, and switch perspectives.

Walkthrough 10-3: Filter messages

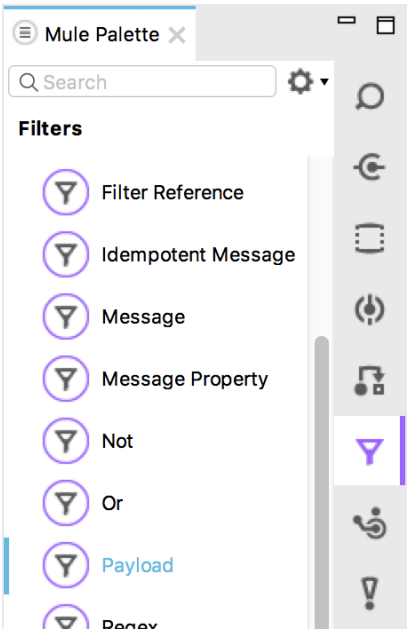
In this walkthrough, you filter the results in the multicast to ensure they are ArrayLists and not exception strings. You will:

- Use the Payload filter.
- Create and use a global filter.



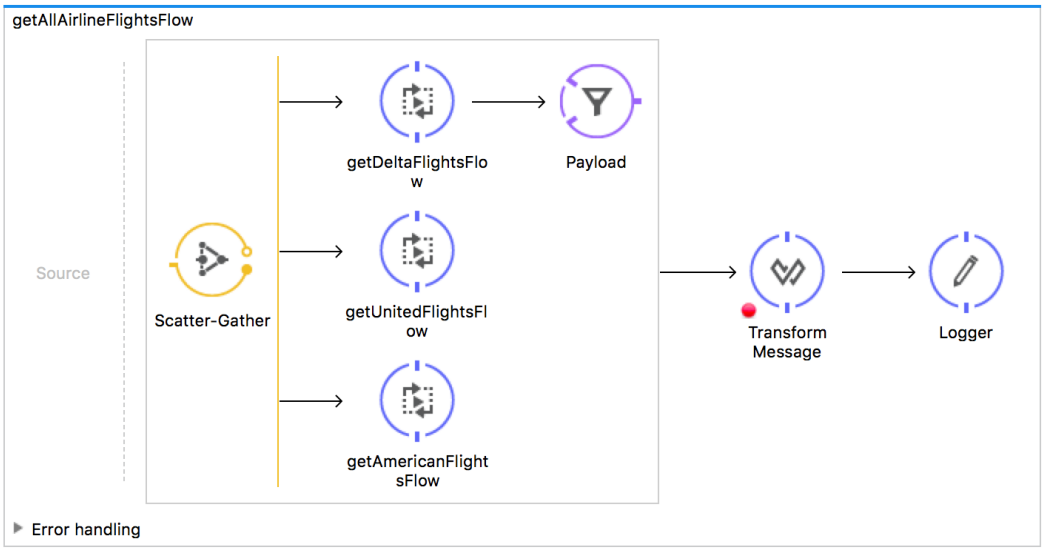
Browse the filter elements in the Mule Palette

1. In the Mule Palette, select the Filters tab.
2. View the available filters.

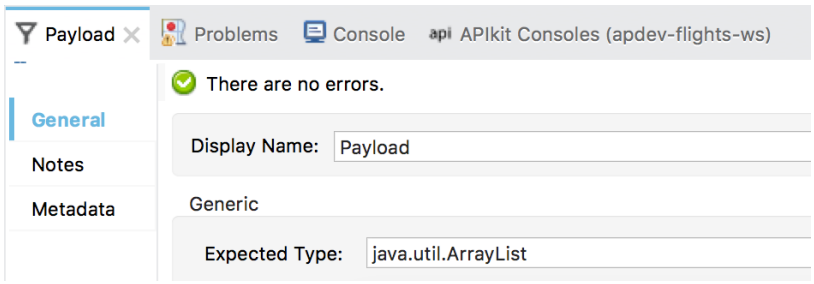


Add a filter

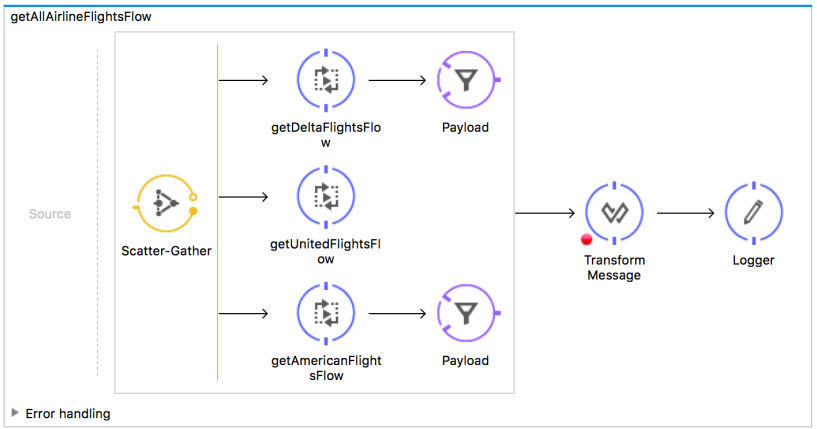
3. Return to getAllAirlineFlightsFlow.
4. Drag out a Payload filter from the Mule Palette and drop it after the getDeltaFlightsFlow reference in the Scatter-Gather.



5. In the Payload properties view, set the expected type to java.util.ArrayList.

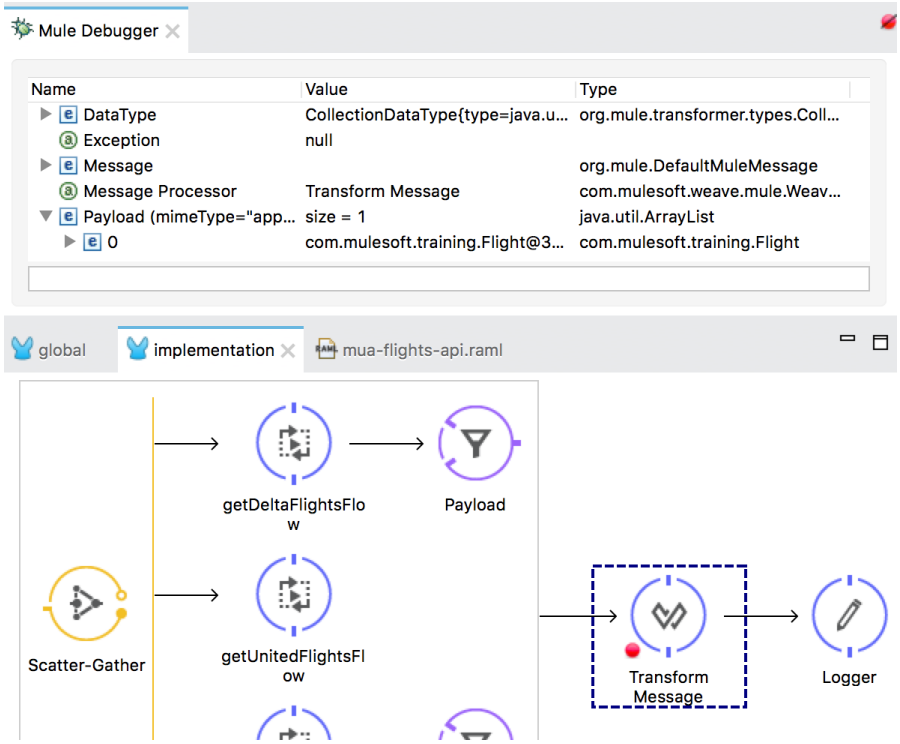


6. Add a Payload filter after the getAmericanFlightsFlow reference and set its expected type to java.util.ArrayList.

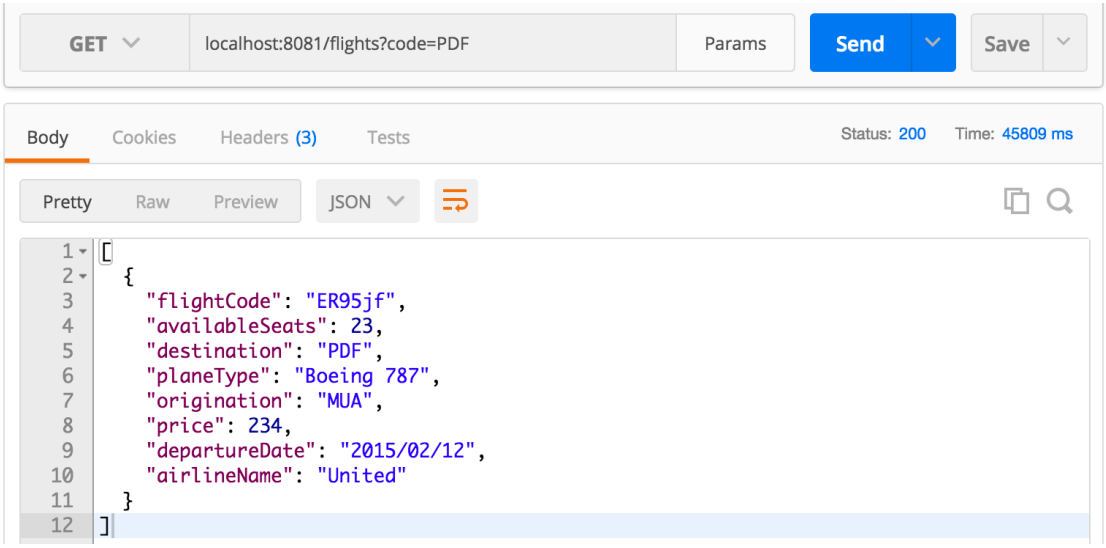


Test the application

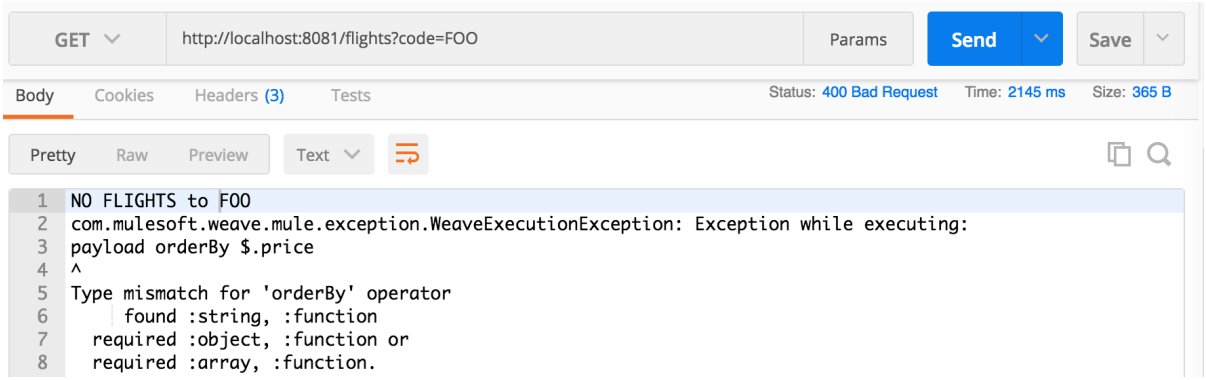
7. Debug the project.
8. In Postman, make the same request to PDF and all airlines.
9. In the Mule Debugger, step to the Transform Message component after the Scatter-Gather; this time you should see the payload does not contain the error strings.



10. Step to the end of the application.
11. In Postman, view the response; you should see the United results to PDF.

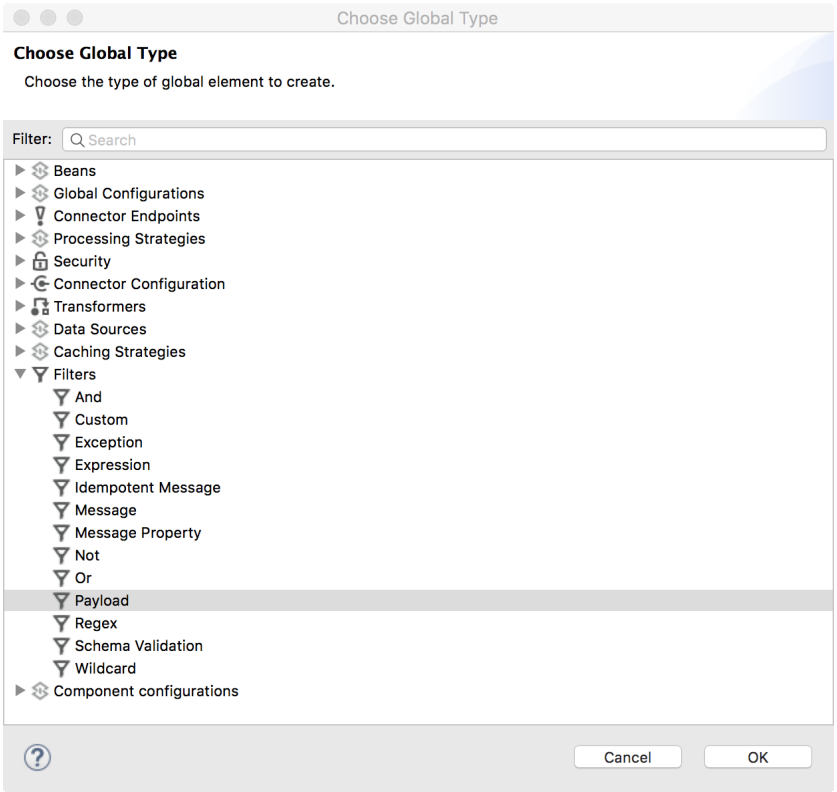


12. Change the code parameter to FOO and send the request.
13. Click Resume until you step through to the end of the application.
14. In Postman, view the response; you should see an error message.

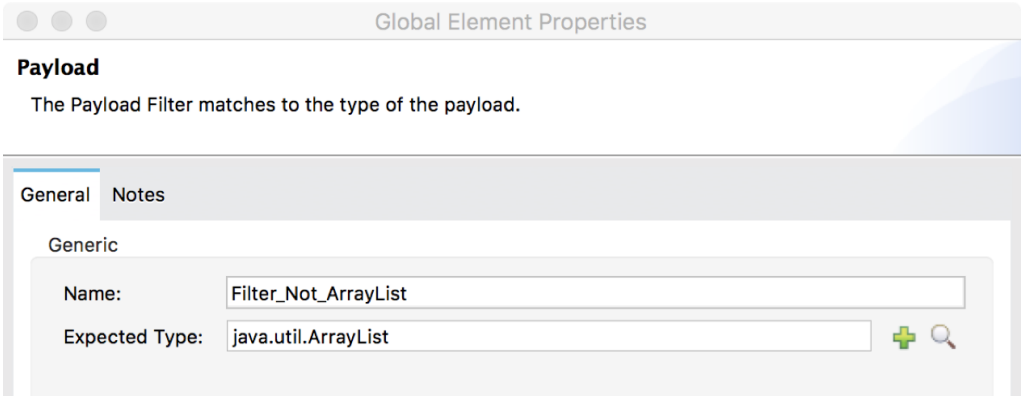


Create a global filter

15. Return to getAllAirlineFlightsFlow in the Mule Design perspective.
16. Delete the Payload filters.
17. Return to global.xml.
18. Switch to the Global Elements view and click Create.
19. In the Choose Global Type dialog box, select Filters > Payload and click OK.

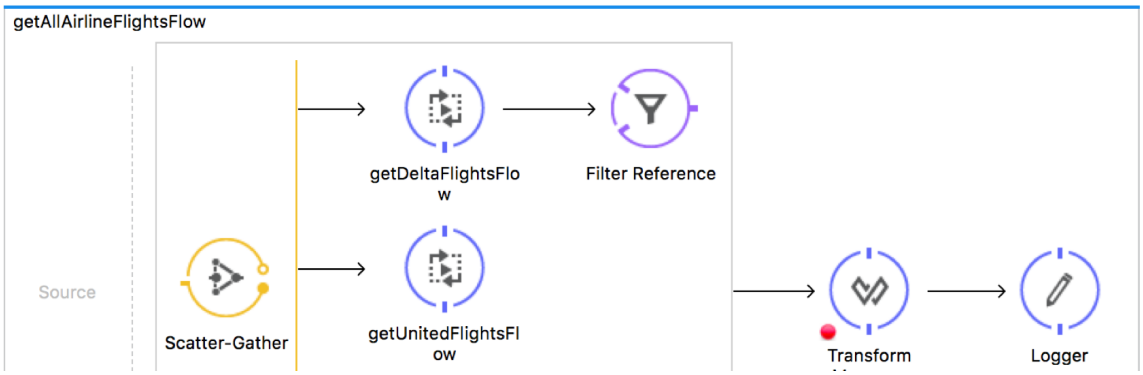


20. In the Global Elements dialog box, set the name to Filter_Not_ArrayList.
21. Set the expected type to java.util.ArrayList and click OK.

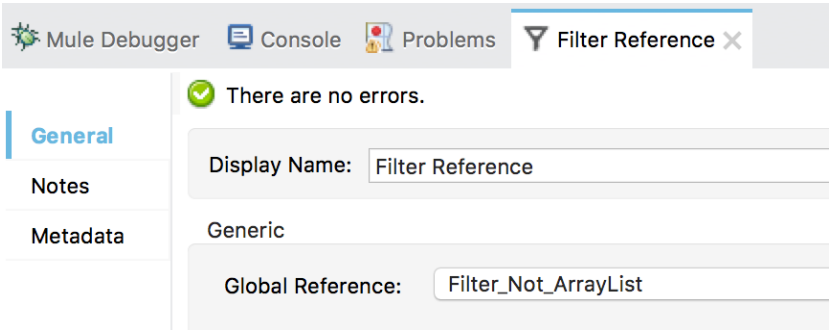


Use the global filter

22. Return to getAllAirlineFlightsFlow in implementation.xml.
23. Drag a Filter Reference from the Mule Palette and drop it after getDeltaFlightsFlow in the Scatter-Gather.

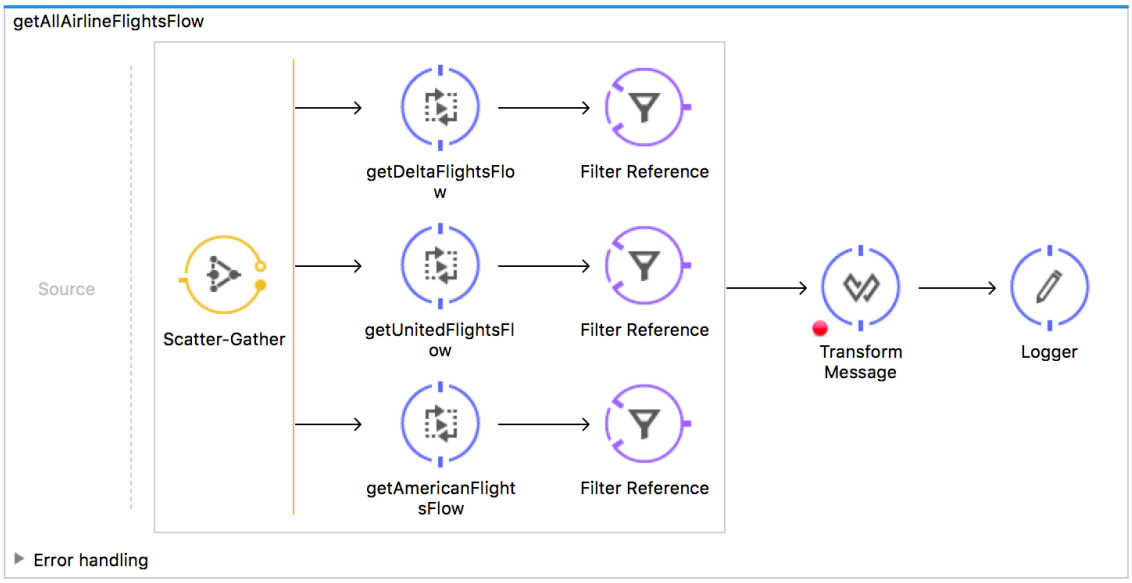


24. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.



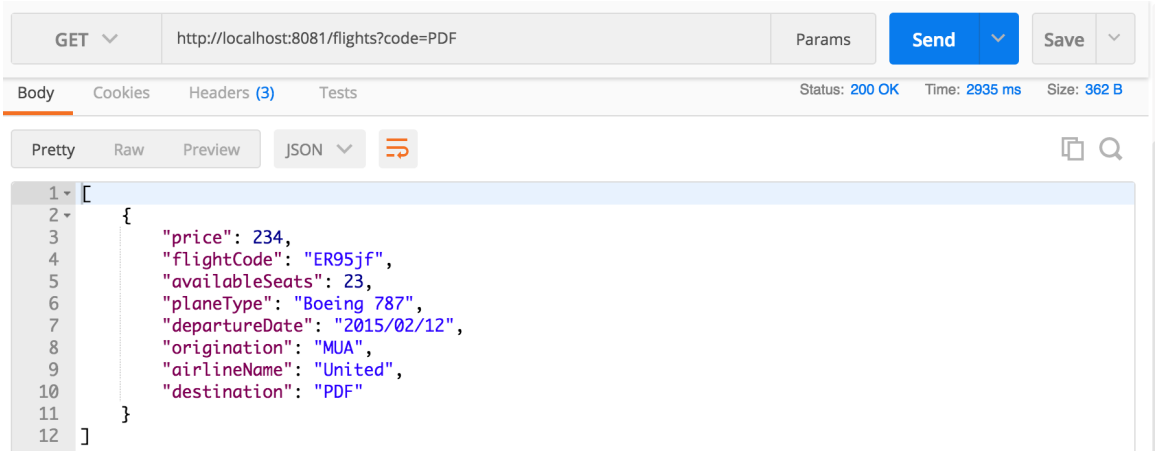
25. Add a Filter Reference after getUnitedFlightsFlow.
26. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.
27. Add a Filter Reference after getAmericanFlightsFlow.

28. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.

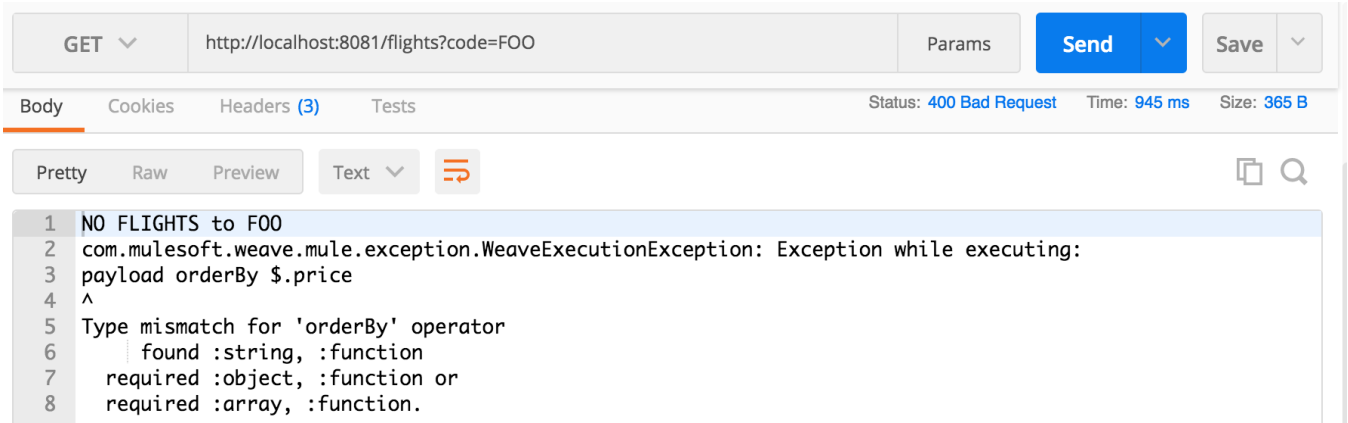


Test the application

29. Debug the project.
30. In Postman, make the same request to http://localhost:8081/flights?code=PDF; you should still get the one United flight.



31. In Postman, make a request to http://localhost:8081/flights?code=FOO; you should correctly get the message that there are no flights to FOO.

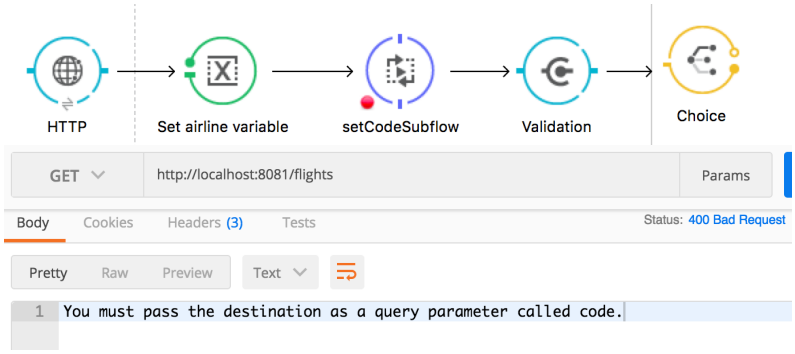


32. Return to Anypoint Studio and stop the project.

Walkthrough 10-4: Validate messages

In this walkthrough, you modify the application to require a destination code be sent to it as a query parameter and then use a validator to make sure a value is sent and to throw an exception if it is not. You will:

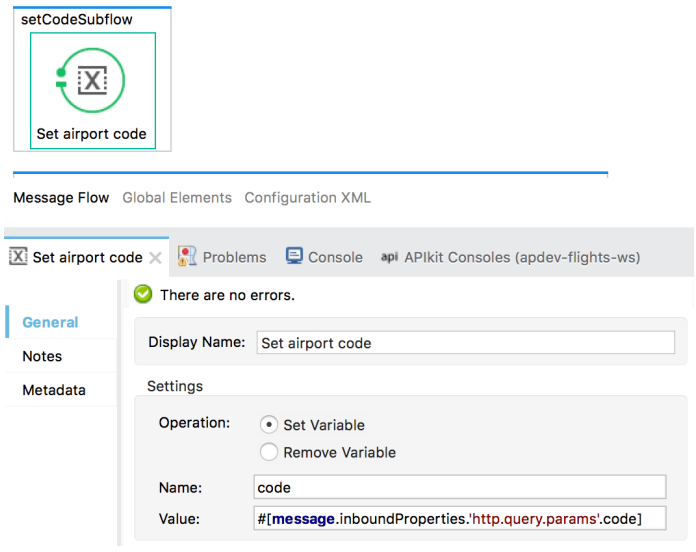
- Require a destination code to be passed to the application as a query parameter.
- Use the Validation component to throw an exception if the query parameter is not set.
- Catch the exception in the global exception strategy and return an appropriate message.



Require a destination code to be specified

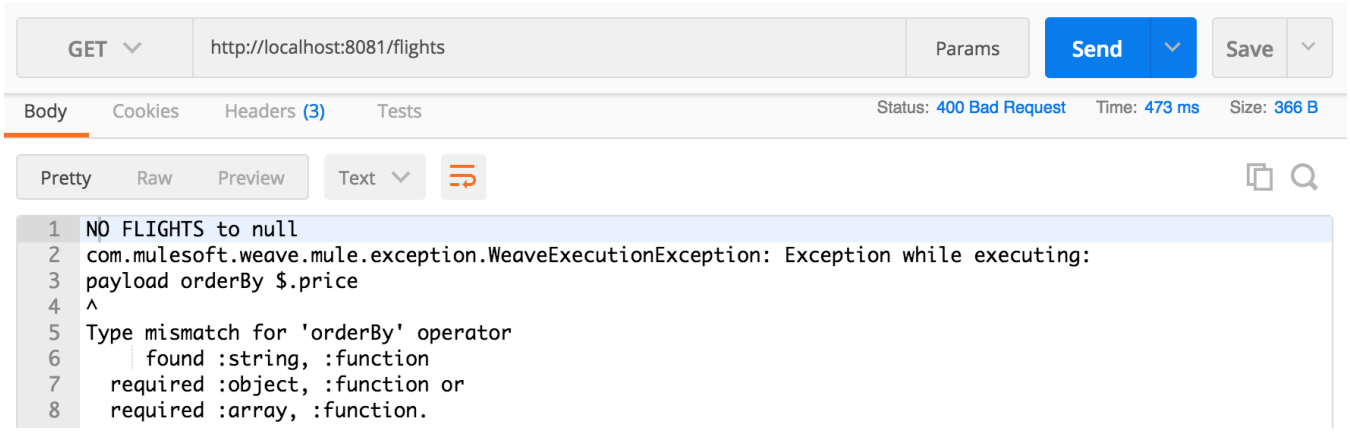
1. Return to getFlightsFlow and locate the setCodeSubflow flow reference.
2. Locate the setCodeSubflow subflow and review the value set for the code flow variable.
3. Delete the ternary expression and instead just set the flow variable to the value of the code query parameter.

```
#[message.inboundProperties.'http.query.params'.code]
```



Test the application

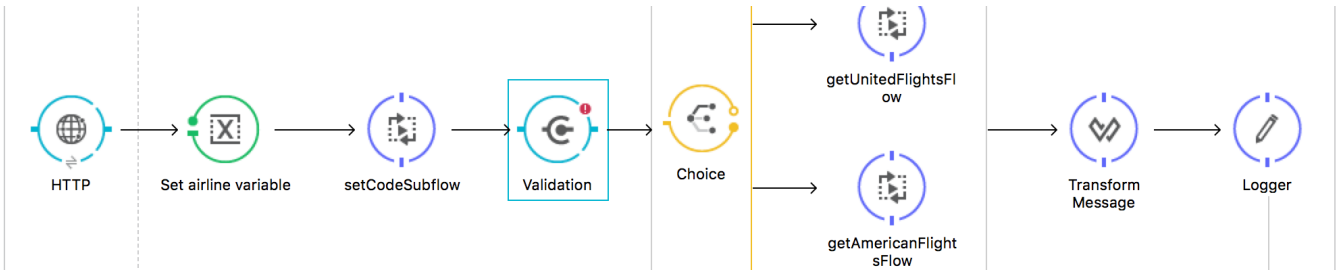
4. Run the project.
5. In Postman, make a request to http://localhost:8081/flights; you should get an incorrect message that there are no flights to null.



6. Return to Anypoint Studio and stop the project.

Add a validator

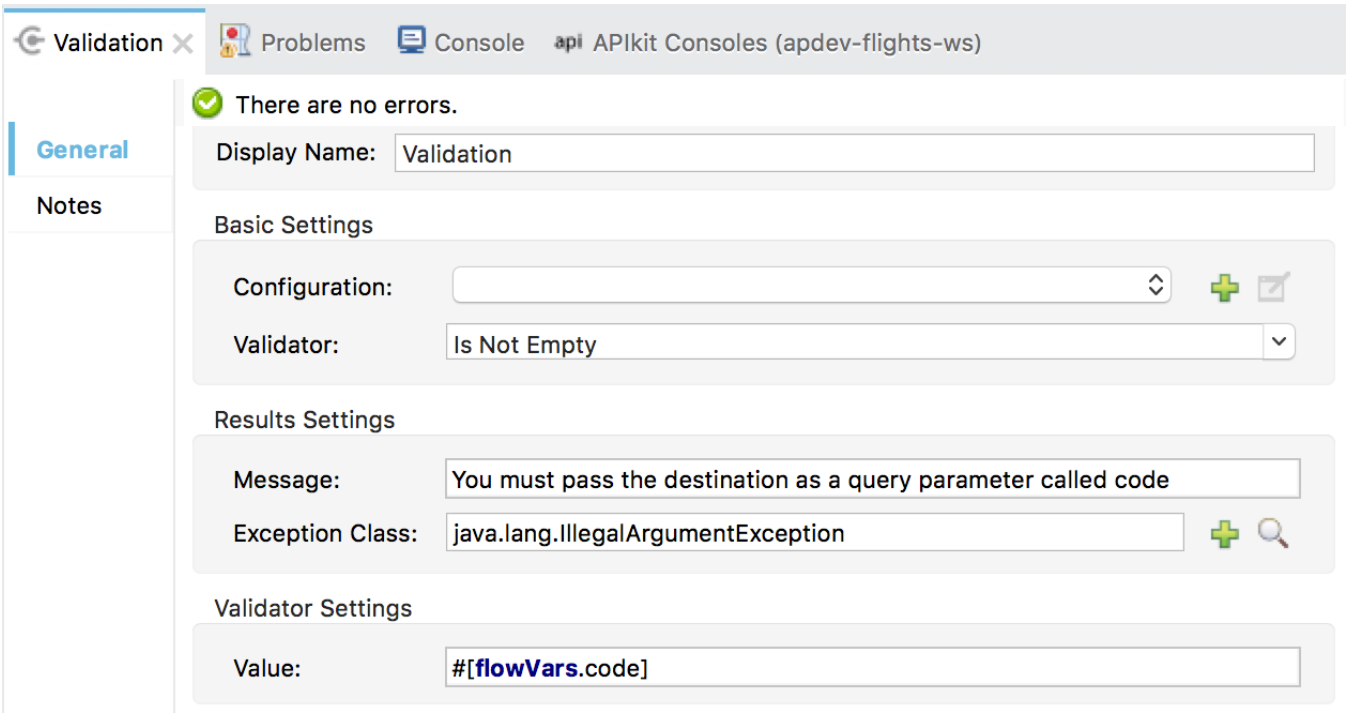
7. Return to getFlightsFlow.
8. Drag out a Validation component from the Mule Palette and drop it after the setCodeSubflow flow reference.



Configure the validator

9. In the Validation properties view, set the validator to Is Not Empty.
10. Set the value to #[flowVars.code].
11. Set the message to You must pass the destination as a query parameter called code.
12. Set the Exception class to java.lang.IllegalArgumentException.

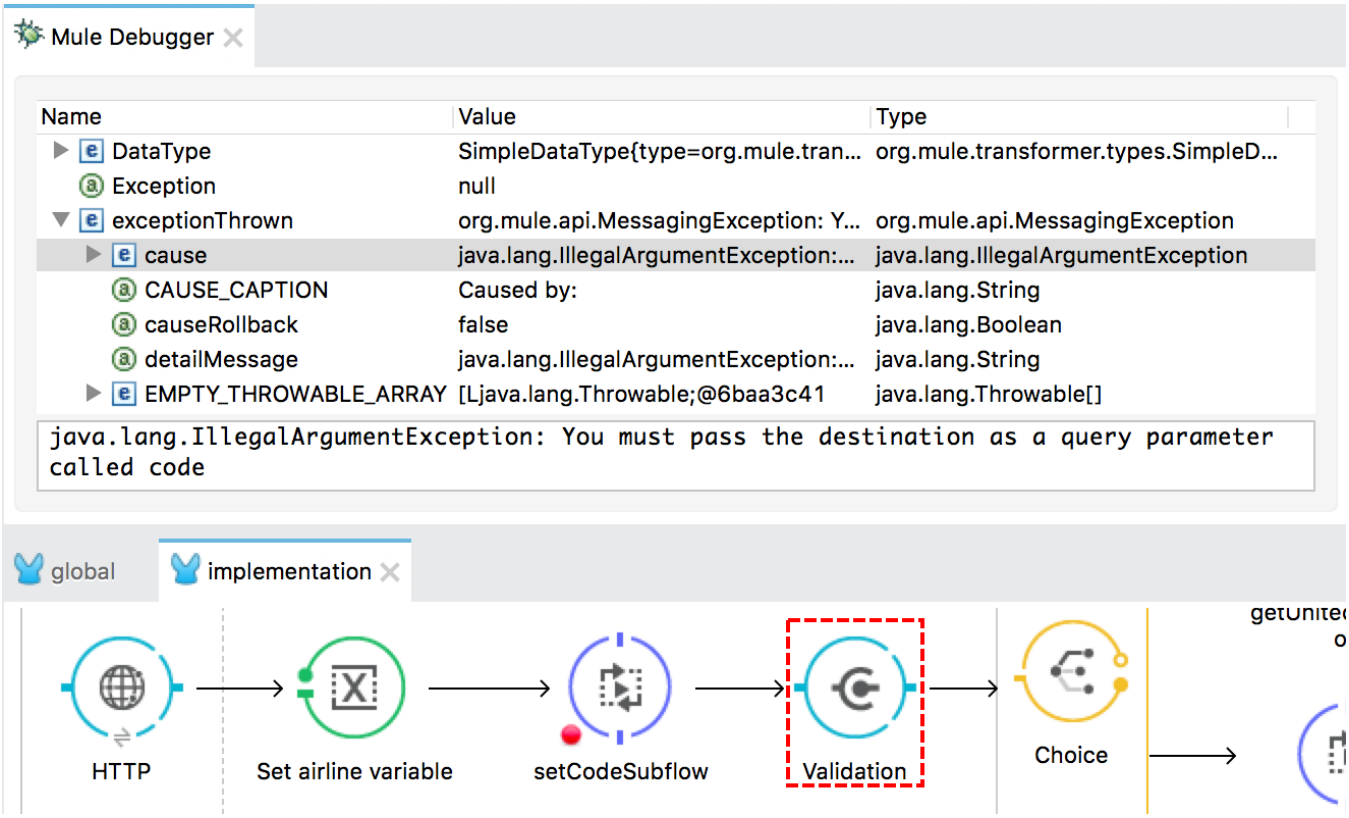
    Note: If you do not set an exception type, it will default to be of type org.mule.extension.validation.api.ValidationException.



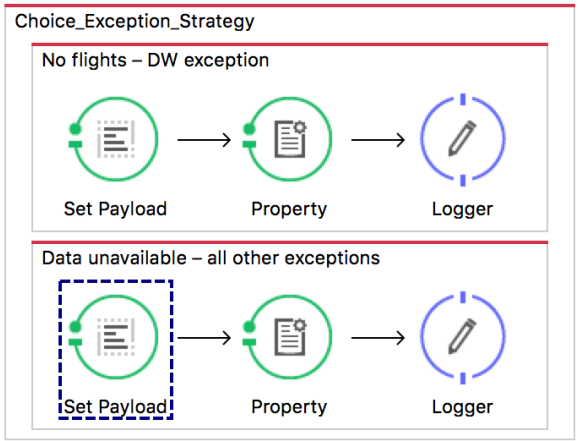
Test the application

13. Place a breakpoint on the setCodeSubflow flow reference.
14. Debug the project.
15. In Postman, make the same request to http://localhost:8081/flights.

16. In the Mule Debugger, step to the Validation component and look at the exception it throws.



17. Step again; you should move into the Data unavailable catch exception strategy in global.xml.

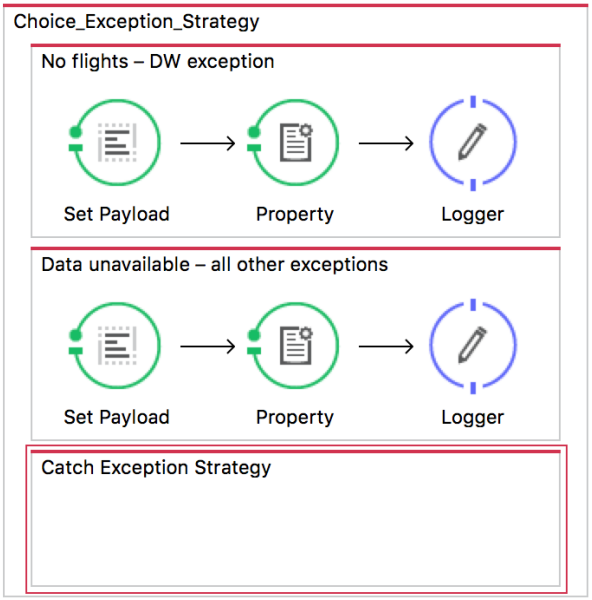


18. Step to the end of the application.

Add a catch exception strategy

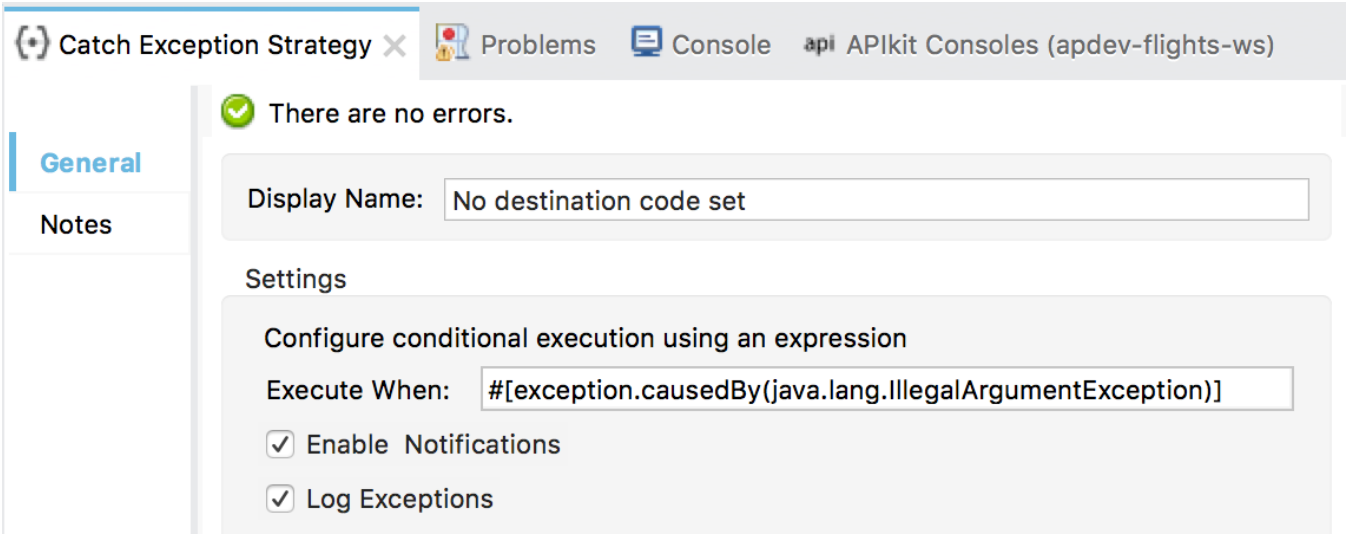
19. Return to global.xml.

20. Drag a Catch Exception Strategy form the Mule Palette and drop it in the choice exception strategy.

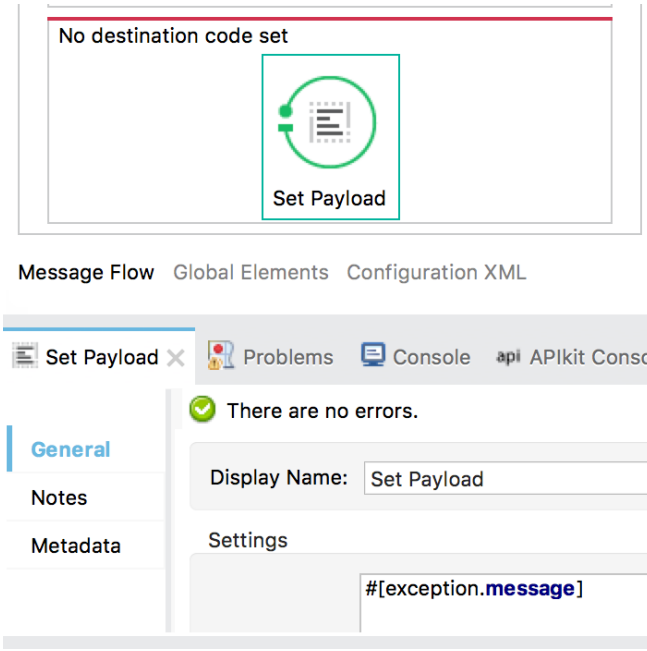


21. In the Properties view for this catch strategy, set the display name to No destination code set.
22. Specify to execute this catch strategy when the exception is of type java.util.IllegalArgumentException.

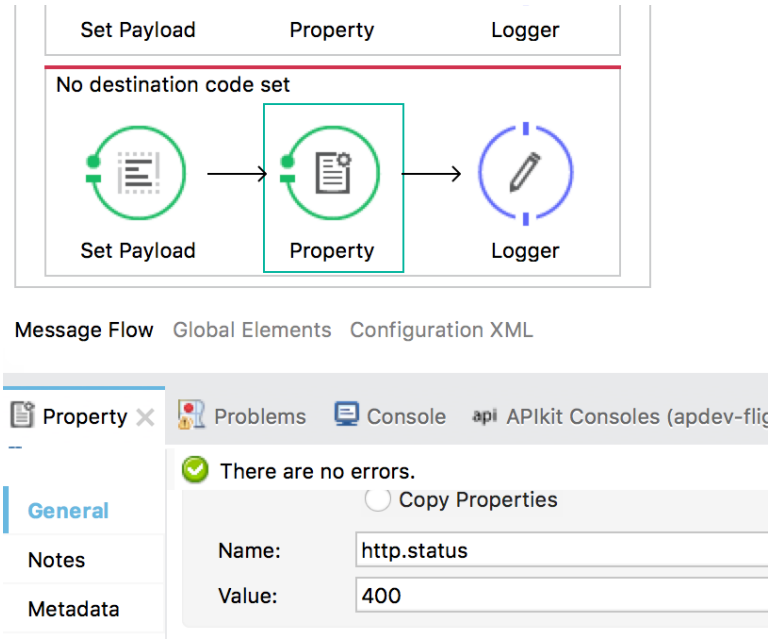
```
#[exception.causedBy(java.lang.IllegalArgumentException)]
```



23. Add a Set Payload transformer to the catch strategy and set the value to the message property of the exception.



24. Add a Property transformer and set a property with the name http.status and a value of 400.
25. Add a Logger component.

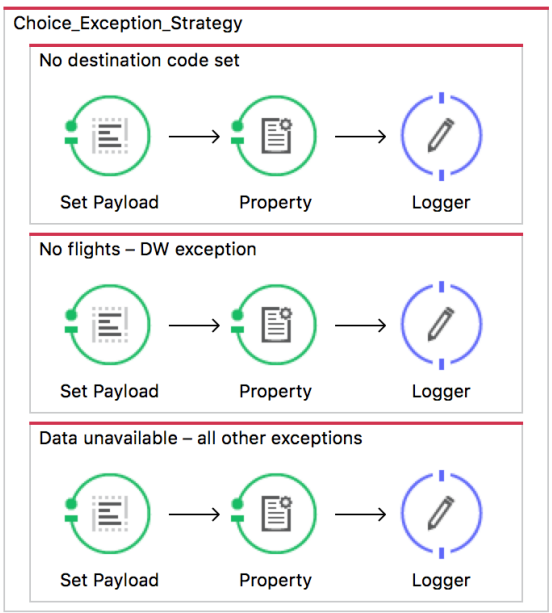


26. Switch to the Configuration XML view.

27. Move this catch exception strategy so it is the first strategy in the choice exception strategy.

```
*global   implementation

22
23      <choice-exception-strategy name="Choice_Exception_Strategy">
24          <catch-exception-strategy when="#[exception.causedBy(java.lang.IllegalArgumentEx
25              <set-payload value="#[exception.message]" doc:name="Set Payload"/>
26              <set-property propertyName="http.status" value="400" doc:name="Property"/>
27              <logger level="INFO" doc:name="Logger"/>
28          </catch-exception-strategy>
29          <catch-exception-strategy when="#[exception.causeMatches('com.mulesoft.weave.*')
30              <set-payload value="NO FLIGHTS to #[flowVars.code +  '\n' + exception]" doc:
31              <set-property propertyName="http.status" value="400" doc:name="Property"/>
32              <logger level="INFO" doc:name="Logger"/>
33          </catch-exception-strategy>
34      <catch-exception-strategy doc:name="Data unavailable &#8211; all other exceptions">
35          <set-payload value="DATA IS UNAVAILABLE. TRY LATER. #['\n' + exception] " doc:na
36          <set-property propertyName="http.status" value="500" doc:name="Property"/>
37          <logger level="INFO" doc:name="Logger"/>
38      </catch-exception-strategy>
39
40      </choice-exception-strategy>
```

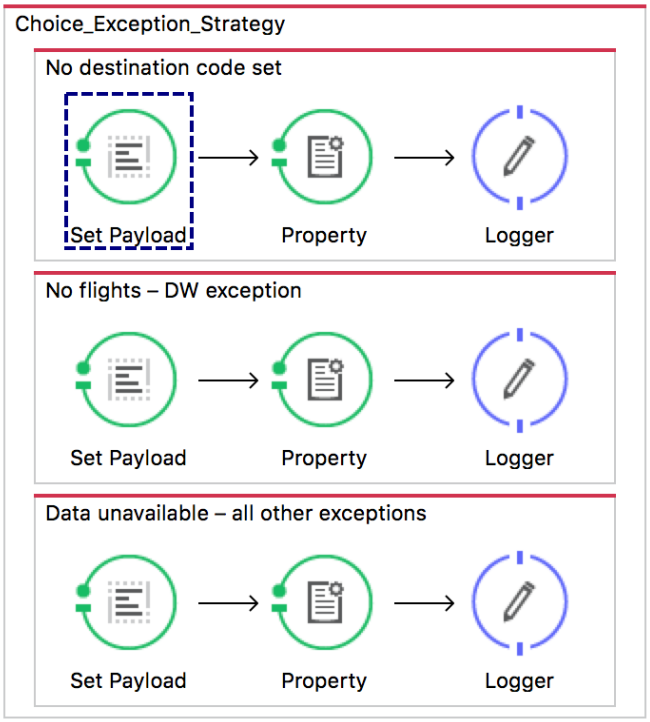
28. Return to the Message Flow view; the default catch exception strategy should now be at the bottom.



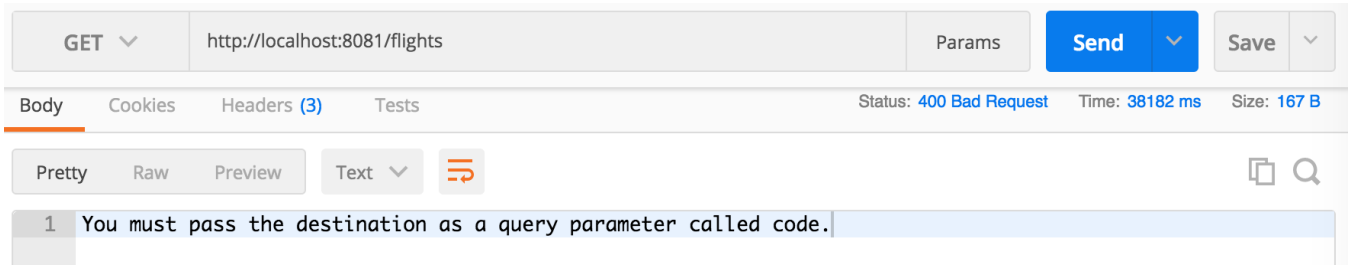
Test the application

29. Debug the project.
30. In Postman, make the same request to http://localhost:8081/flights.

31. In the Mule Debugger, step past the Validation component into the exception strategy; this time, you should move into the No destination code set strategy.

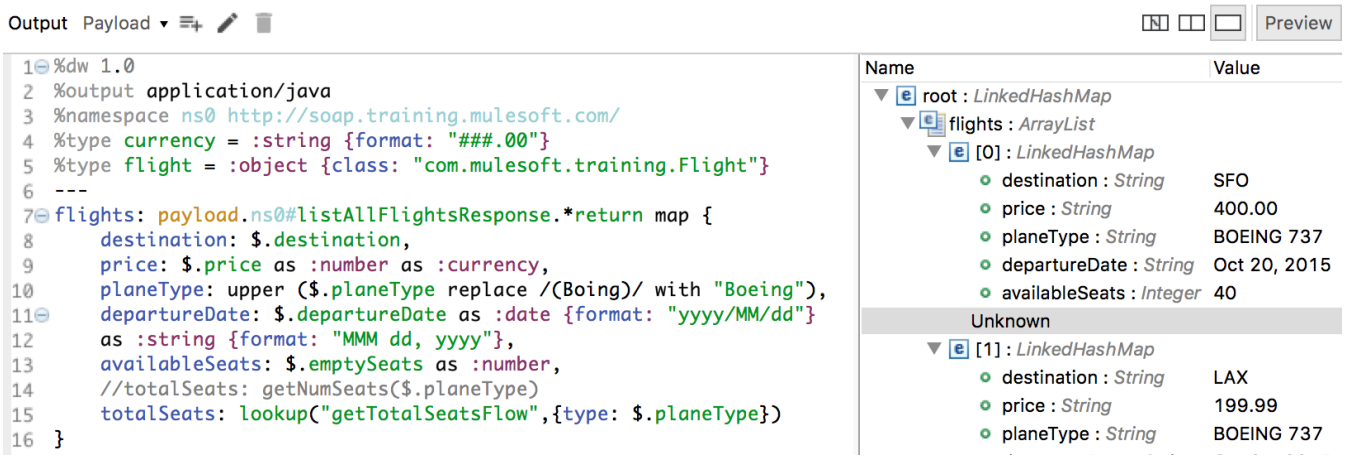


32. Step to the end of the application.
33. In Postman, you should see the You must pass the destination as a code query parameter message.



34. Return to Anypoint Studio, stop the project, and switch perspectives.
35. Close the project.

Module 11: Writing DataWeave Transformations



At the end of this module, you should be able to:

- Write DataWeave expressions for basic XML, JSON, and Java transformations.
- Store DataWeave transformations in external files.
- Write DataWeave transformations for complex data structures with repeated elements.
- Coerce and format strings, numbers, and dates.
- Use DataWeave operators.
- Define and use custom data types.
- Call MEL functions and Mule flows from DataWeave transformations.

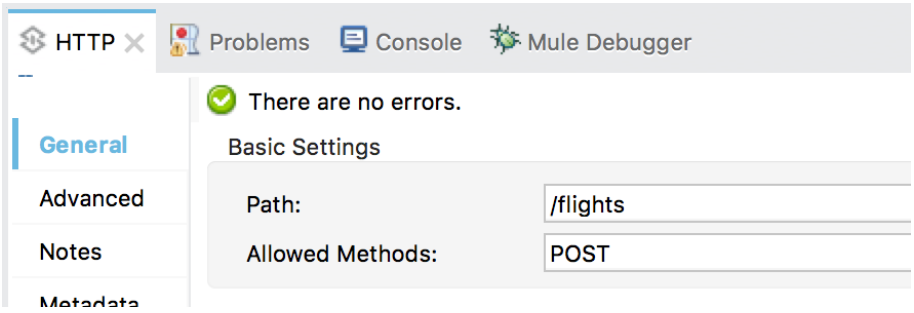
Walkthrough 11-1: Write your first DataWeave transformation

In this walkthrough, you work with JSON data for a flight posted to a flow. You will:

- Create a new flow that receives POST requests.
- Write a DataWeave expression to transform the data to Java, XML, or JSON.
- Add sample data and use live preview.
- Save the transformation in an external file.

Create a new flow

1. Return to implementation.xml.
2. Drag an HTTP connector from the Mule Palette to the bottom of the canvas.
3. Change the name of the new flow to postFlightFlow.
4. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /flights and the allowed methods to POST.

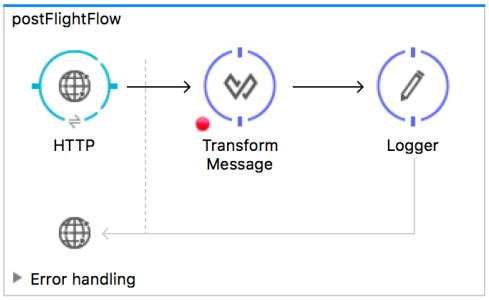


6. Add a Logger to the flow.

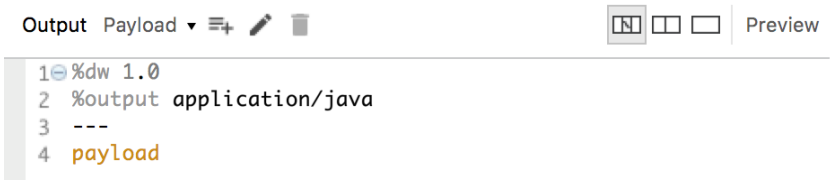
Add a DataWeave Transform Message component

7. Add a DataWeave Transform Message component before the Logger.

8. Add a breakpoint to it.

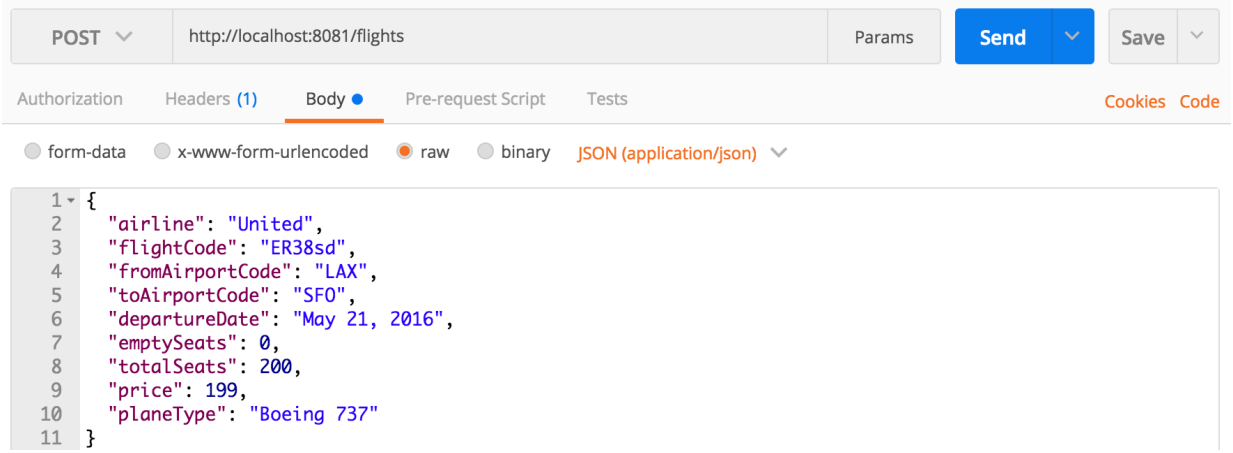


9. In the Transform Message properties view, locate the drop-down menu at the top of the view that sets the output type; it should be set to Payload.
10. Beneath it, locate the directive that sets the output to application/java.
11. Delete the existing DataWeave expression (the curly braces) and set it to payload.

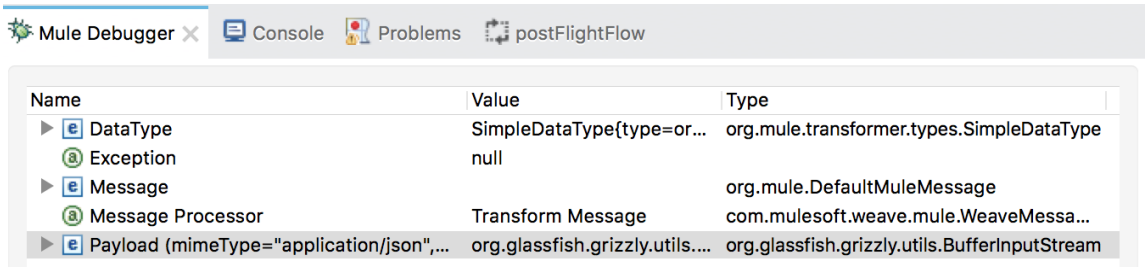


Post data to the flow

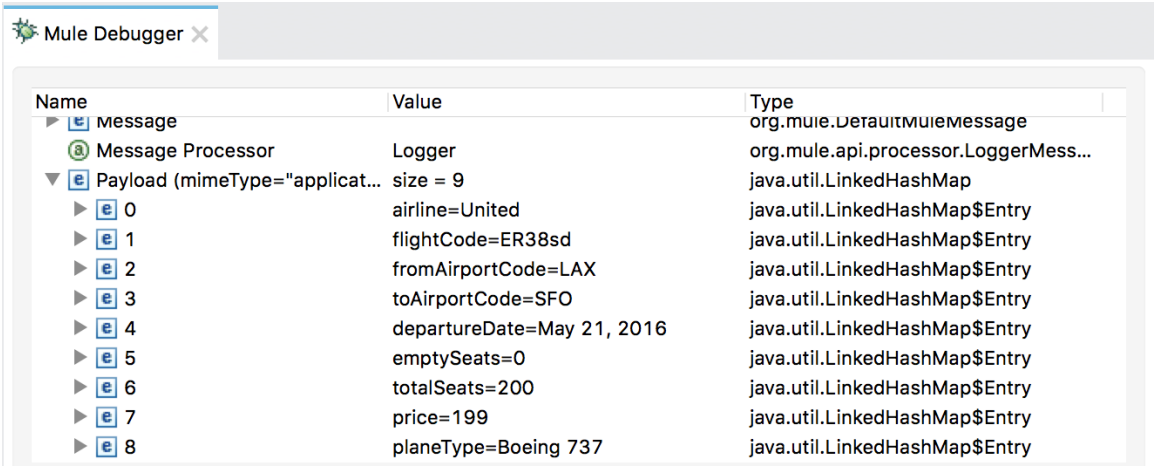
12. Open flight-example.json in src/test/resources.
13. Copy the code.
14. Debug the project.
15. In Postman, change the method to POST and remove any query parameters.
16. Add a request header called Content-Type and set it equal to application/json.
17. For the request body, select raw and paste the value you coped from flight-example.json.
18. Make the request to http://localhost:8081/flights.



19. In the Mule Debugger, you should see the payload has a mime-type of application/json and is of type BufferInputStream.



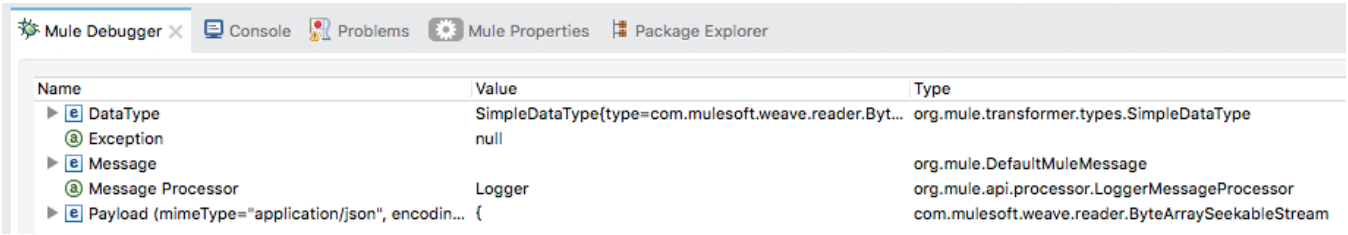
20. Look at the inbound properties; you should see the content-type is set to application/json.
21. Step to the Logger; you should see the payload now has a mime-type of application/java and is a LinkedHashMap.



22. Click the Resume button.

Change output data type to JSON

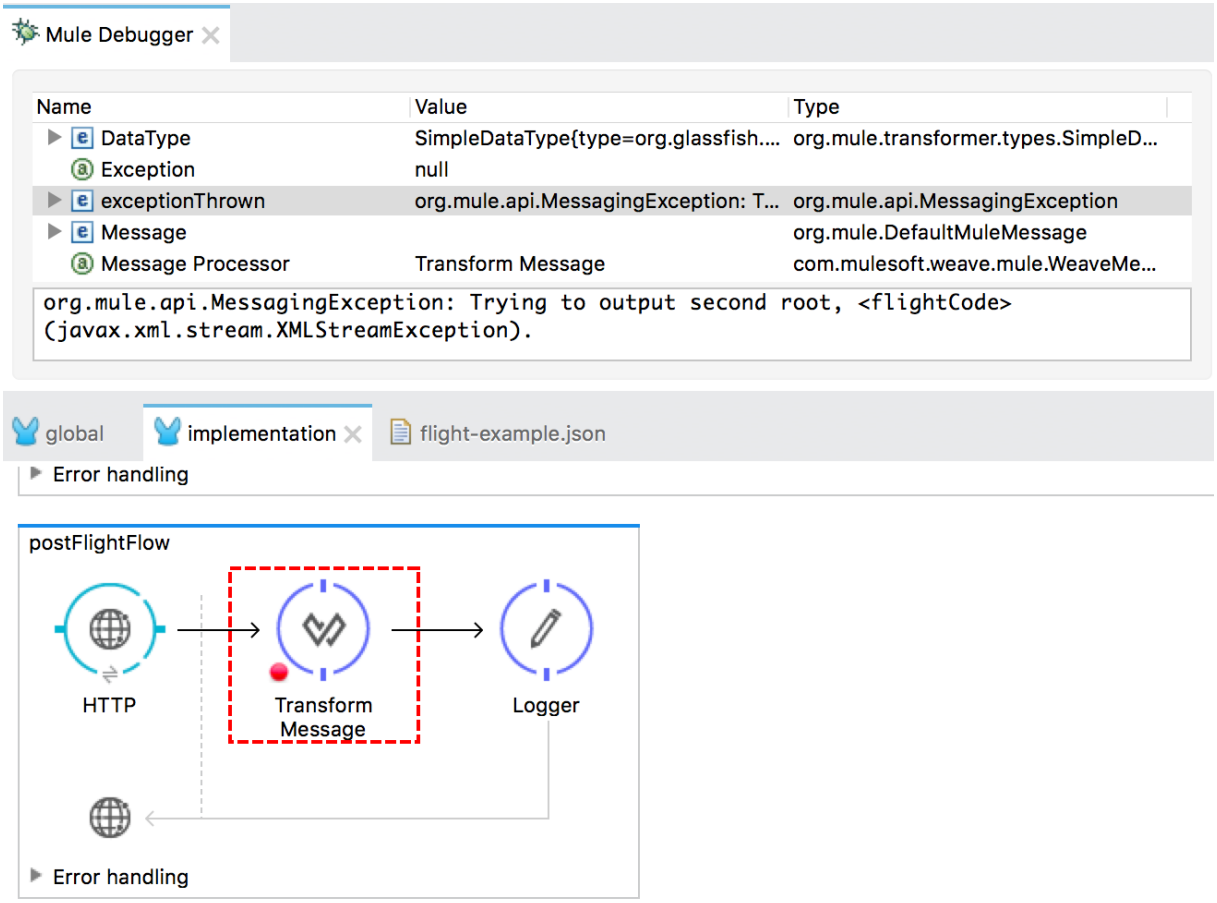
23. In the Transform Message properties view, change the output directive to application/json.
24. Debug the project.
25. In Postman, post the same request to http://localhost:8081/flights.
26. In the Mule Debugger, step to the Logger; you should see the payload has a mime-type of application/json and is a Weave ByteArraySeekableStream.



27. Step to the end of the application.

Change output data type to XML

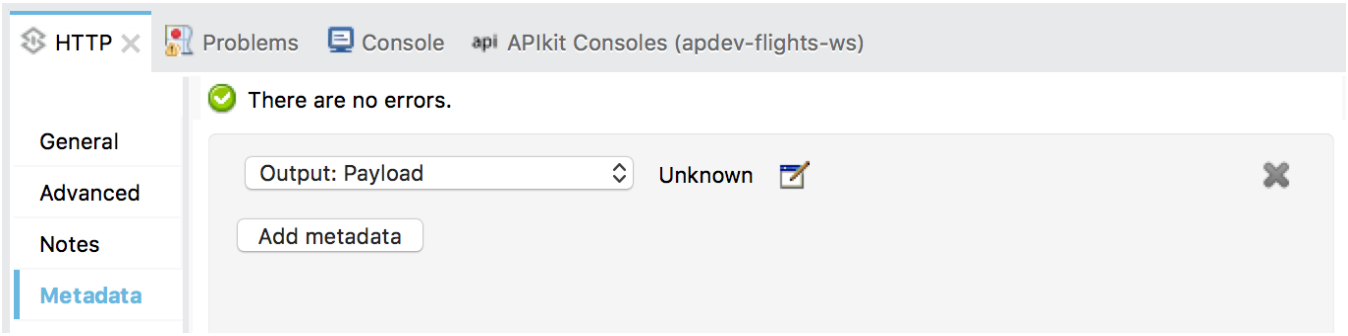
28. In the Transform Message properties view, change the output directive to application/xml.
29. Debug the project.
30. In Postman, post the same request to http://localhost:8081/flights.
31. In the Mule Debugger, click Next processor; you should get an exception.
32. Look at the exceptionThrown; you should see there is a DataWeave error when trying to output the second root.



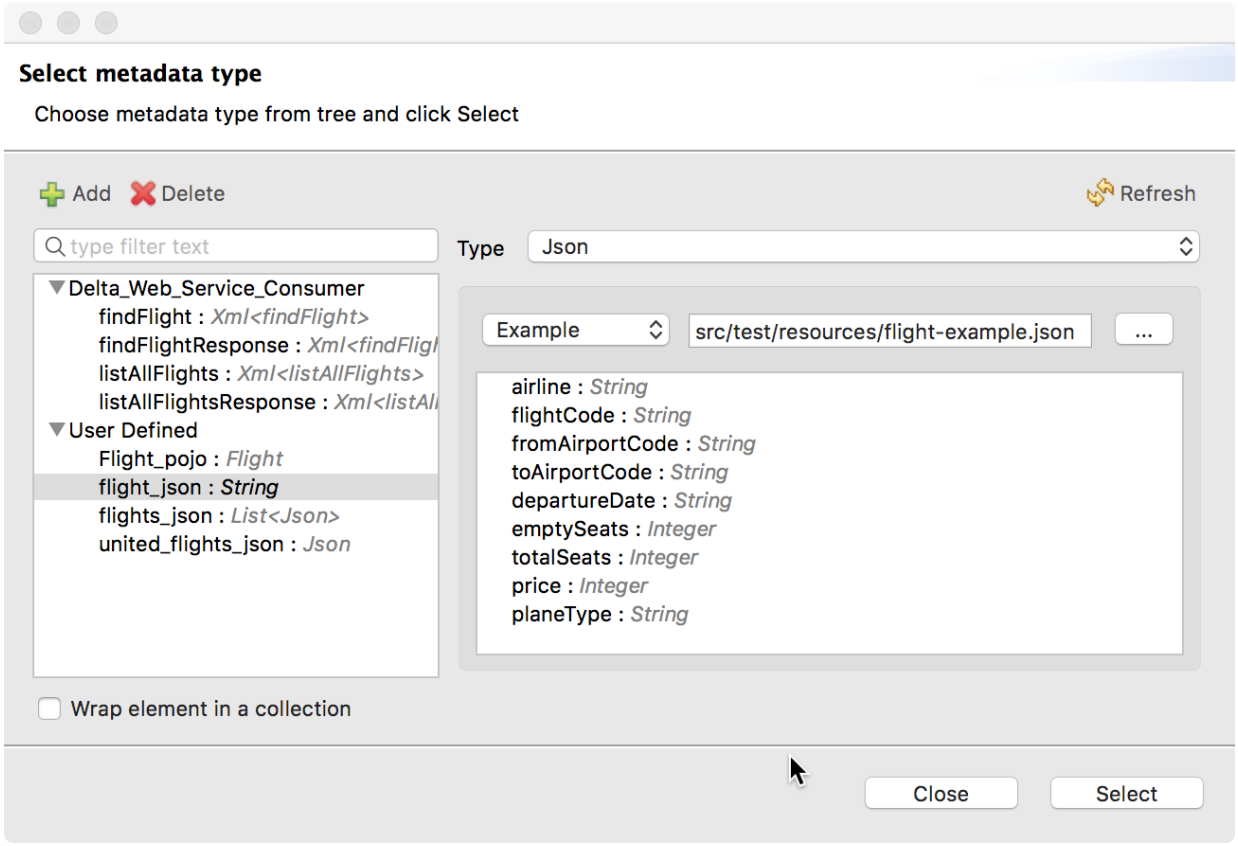
33. Click Resume.
34. Stop the project and switch to the Mule Design perspective.

Add HTTP Listener output metadata

35. In the HTTP Listener Properties view, select Metadata and click the Add metadata button.
36. Select Output:Payload and click the Edit button.



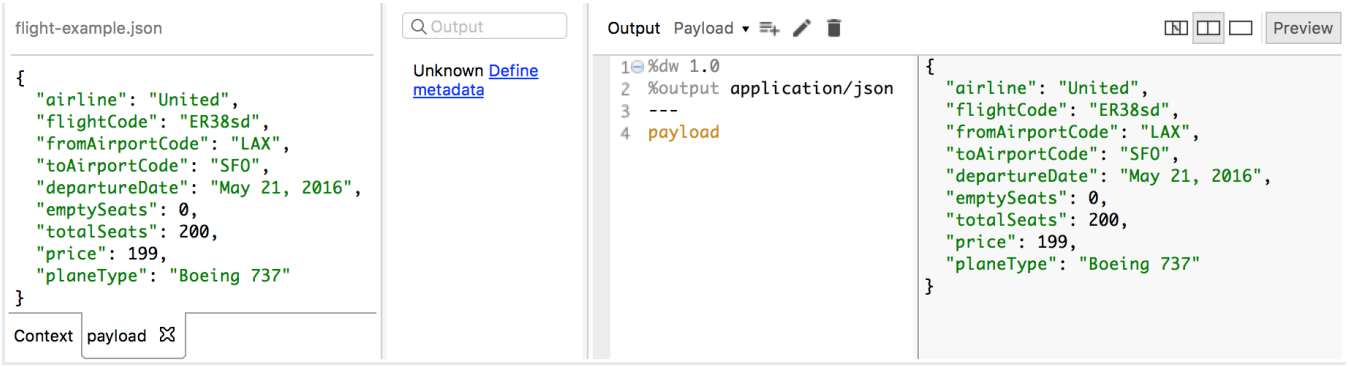
37. In the Select metadata type dialog box, click Add.
38. In the Create new type dialog box, set the type id to flight_json and click Create type.
39. In the Select metadata dialog box, set the type to JSON.
40. Select Example and browse to and select the flight-example.json file in src/test/resources.
41. In the Select metadata type dialog box, click Select.



42. In the Transform Message Properties view, you should now see metadata for the input.

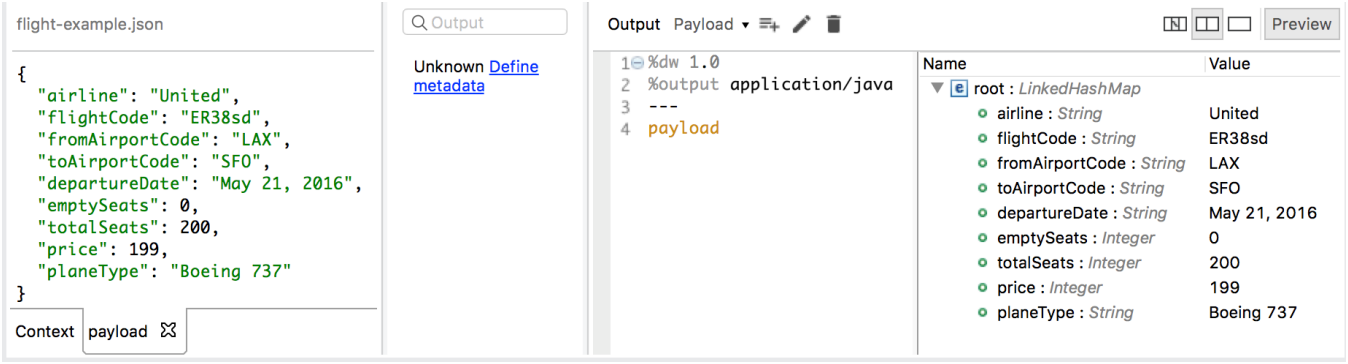
Preview sample data and sample output

43. In the transform section, change the output type from application/xml back to application/json.
44. Click the Preview button.
45. Click the Create required sample data to execute preview link.
46. Look at the input section; you should see a new tab called payload and it should contain the sample data.
47. Click the Show Source With Trees button to the left of the Preview button.
48. Look at the preview section; you should see the sample output there of type JSON.



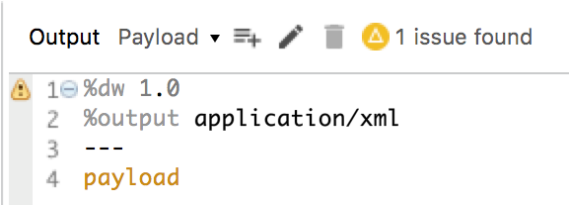
Change the output type

49. In the transform section, change the output type from application/json to application/java.
50. Look at the preview section; you should see the sample output there is now a LinkedHashMap object populated with the sample data.

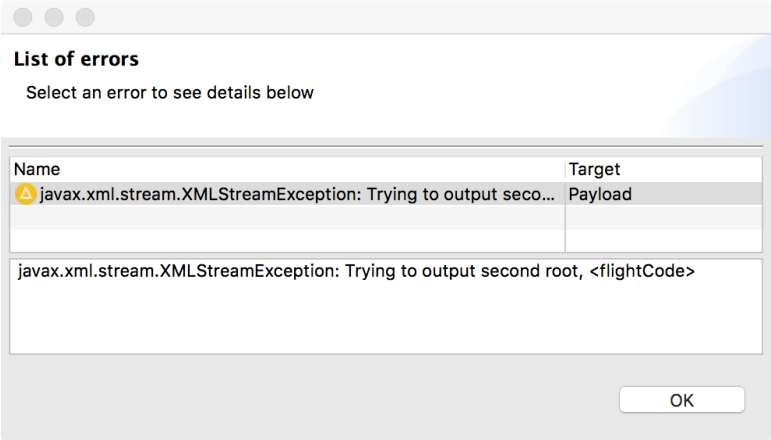


51. In the transform section, change the output type from application/java to application/xml.

52. Locate the warning icons indicating that there is a problem.



53. Mouse over the icon located to the left of the code; you should see a message that there was an exception trying to output the second root <flightCode>.
54. Click the icon above the code; a List of errors dialog box should open.



55. In the List of errors dialog box, click OK.
56. Change the output from application/xml to application/java for now.

    Note: You will learn how to successfully transform to XML in the next walkthrough.

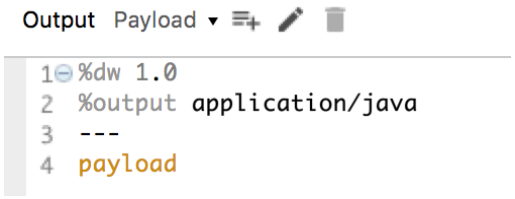
Save the transformation in an external file

57. Switch the canvas to the Configuration XML view.
58. Locate and review the code for the DataWeave transformation.

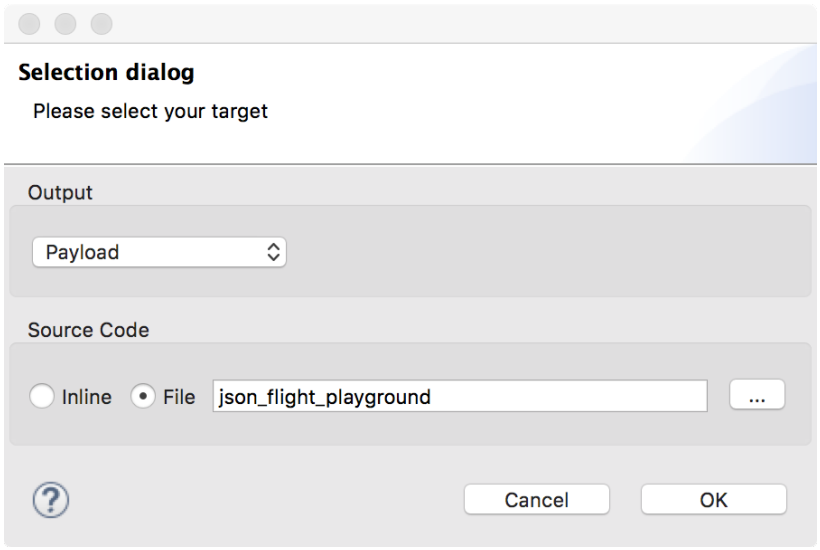
```
148     </flow>
149     <flow name="postFlightFlow">
150         <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedM
151         <dw:transform-message metadata:id="5ded8a12-3d01-4c39-98f1-395a9ac25b31" doc:nam
152             <dw:input-payload doc:sample="flight-example.json"/>
153             <dw:set-payload><![CDATA[%dw 1.0
154 %output application/java
155 ---
156 payload]]></dw:set-payload>
157         </dw:transform-message>
158         <logger level="INFO" doc:name="Logger"/>
159     </flow>
```

59. Switch back to the Message Flow view.

60. In the Transform Message Properties view, click the Edit current target button (the pencil) above the code.



61. In the Selection dialog dialog box, change the source code selection from inline to file.
62. Set the file name to json_flight_playground.



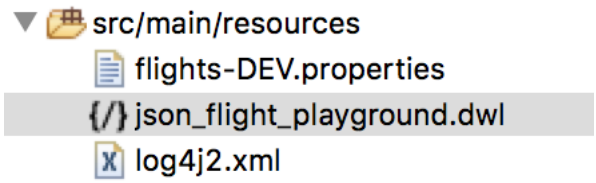
63. Click OK.

Locate and review the code

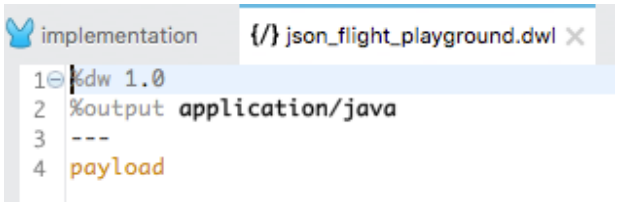
64. Switch to Configuration XML view.
65. Scroll down and look at the new code for the transformation in postFlightFlow.

```
</flow>
<flow name="postFlightFlow">
    <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowe
    <dw:transform-message metadata:id="5ded8a12-3d01-4c39-98f1-395a9ac25b31" doc:n
        <dw:input-payload doc:sample="flight-example.json"/>
        <dw:set-payload resource="classpath:json_flight_playground.dwl"/>
    </dw:transform-message>
    <logger level="INFO" doc:name="Logger"/>
</flow>
```

66. In the Package Explorer, expand src/main/resources.



67. Open json_flight_playground.dwl.
68. Review and then close the file.



69. Return to Message Flow view in implementation.xml.
70. Save the file.

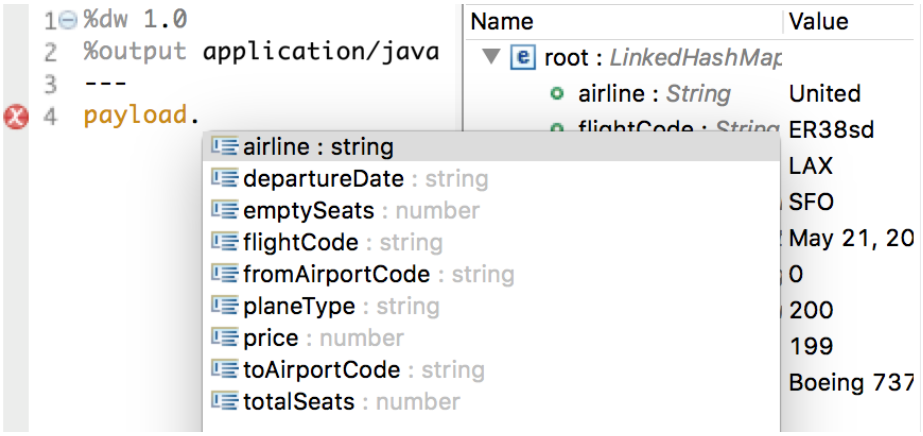
Walkthrough 11-2: Transform basic Java, JSON, and XML data structures

In this walkthrough, you continue to work with the JSON flight data posted to the flow. You will:

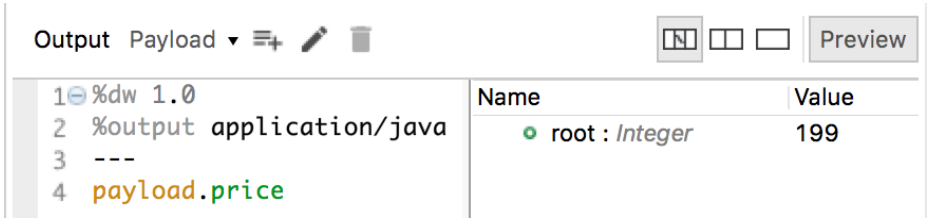
- Write expressions to transform the JSON payload to various Java structures.
- Create a second transformation to store a transformation output in a flow variable.
- Write expressions to transform the JSON payload to various XML structures.

Write expressions to transform JSON to various Java structures

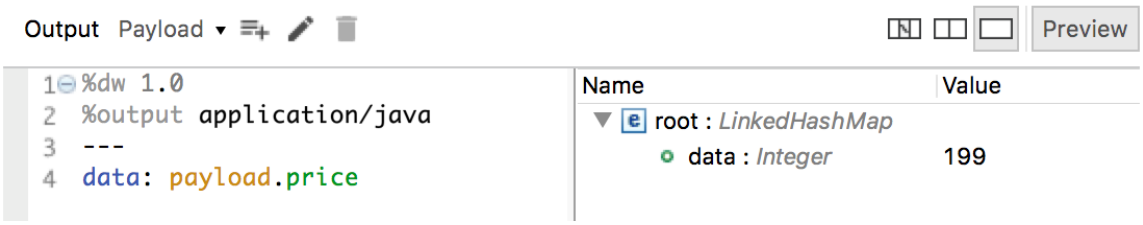
1. Return to the Transform Message Properties view in postFlightFlow.
2. Type a period after payload and select price from the pop-up menu.



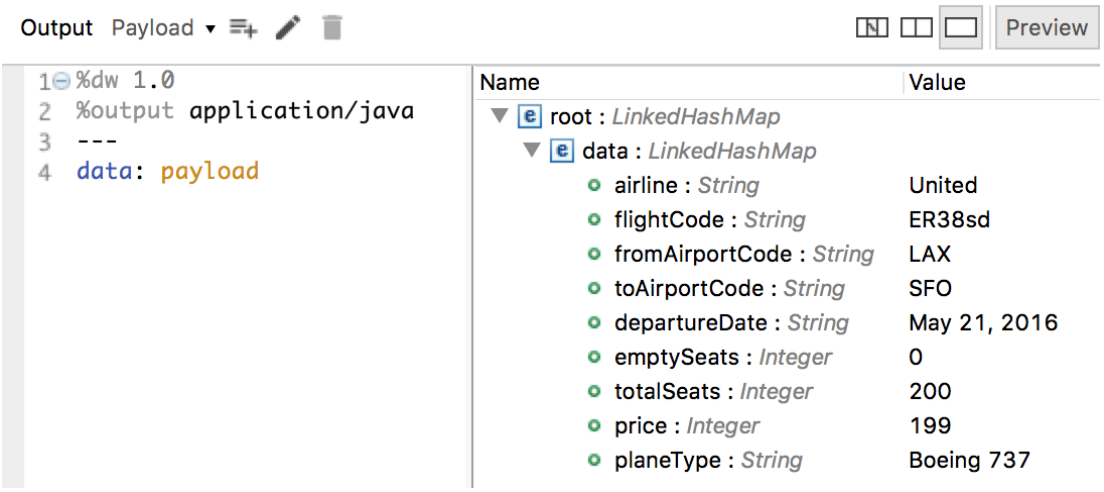
3. Look at the preview; you should see the output is an integer.



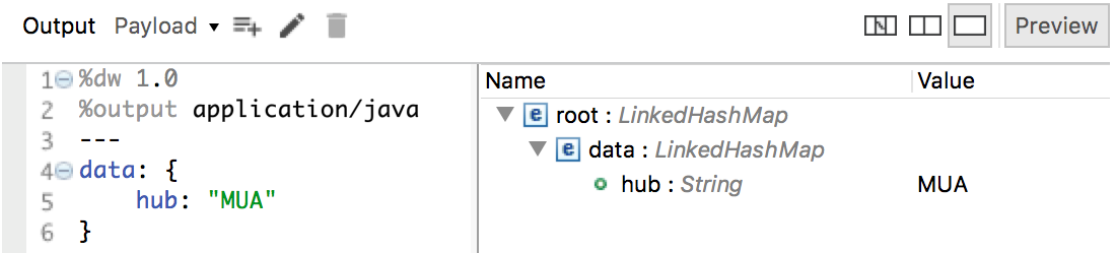
4. Change the DataWeave expression to data: payload.price.



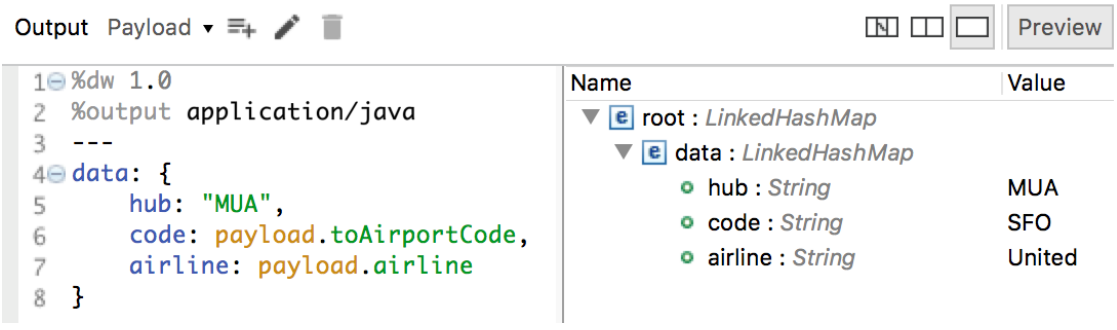
5. Change the DataWeave expression to data: payload.



6. Change the DataWeave expression to data: {}.
7. Add a field called hub and set it to “MUA”.

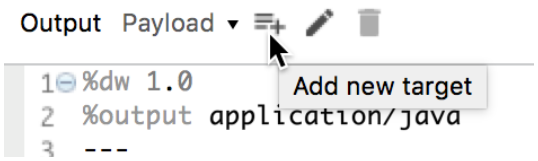


8. Add a field called code and set it to the toAirportCode property of the payload.
9. Add a field called airline and set it to the airline property of the payload.

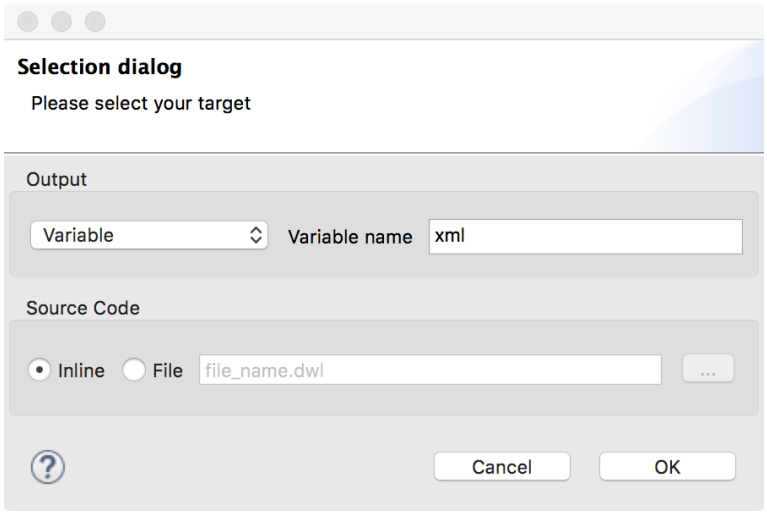


Create a second transformation with the same component

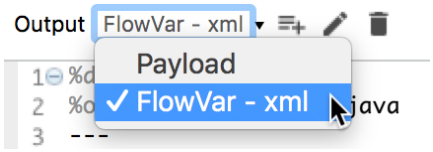
10. Click the Add new target button.



11. In the Selection dialog dialog box, leave the output set to variable.
12. Set the variable name to xml.



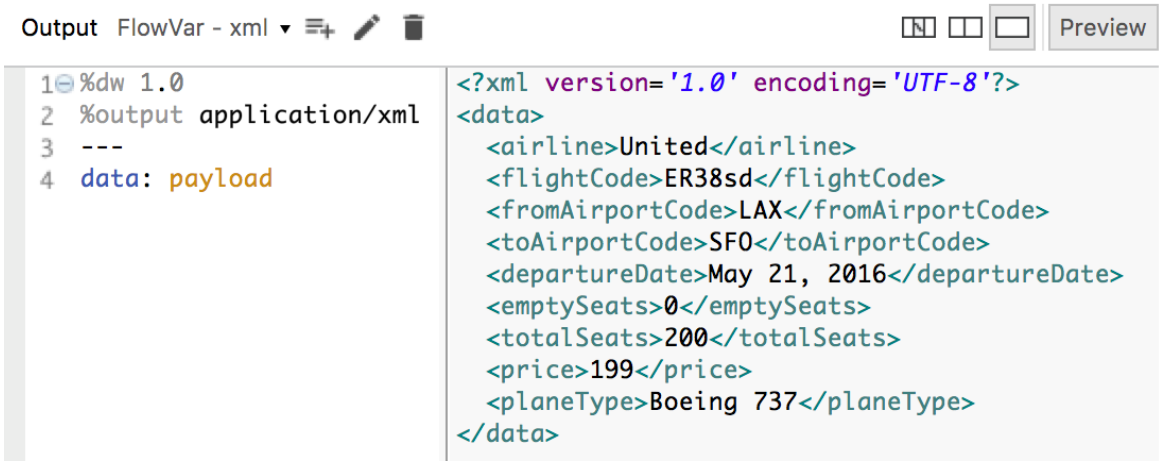
13. Click OK.
14. In the Transform Message properties view, click the drop-down menu button for the output; you should see and can switch between the two transformations.



Write an expression to output data as XML

15. For the FlowVar – xml transformation, click the Preview button.
16. Set the DataWeave expression to payload.
17. Change the output type to application/xml; you should get an issue displayed.
18. Mouse over the warning icon and read the message; it should be the same one you saw in the last walkthrough.
19. Change the DataWeave expression to data: payload.

20. Look at the preview; you should now see XML.



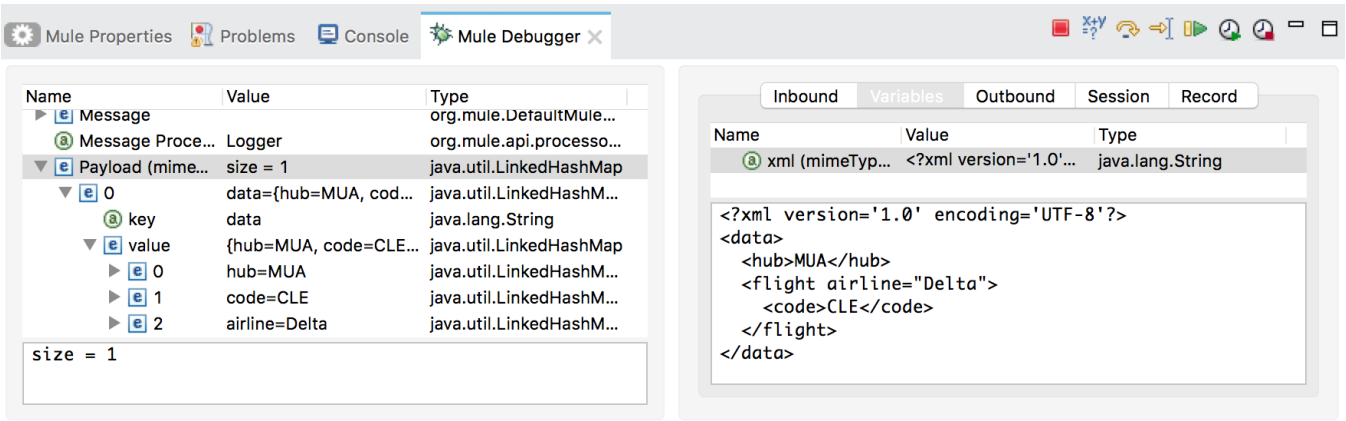
21. In the output section menu bar, click the drop-down arrow next to FlowVar – xml and select Payload.
22. Copy the DataWeave expression.
23. Switch back to FlowVar – xml and replace the DataWeave expression with the one you just copied.

24. Modify the expression so the code and airline properties are child elements of a new element called flight.

25. Modify the expression so the airline is an attribute of the flight element.

Debug the application

26. Debug the project.
27. In Postman, post the same request to http://localhost:8081/flights.
28. In the Mule Debugger, step to the Logger; the payload should be a Java object and there should be an xml flow variable.

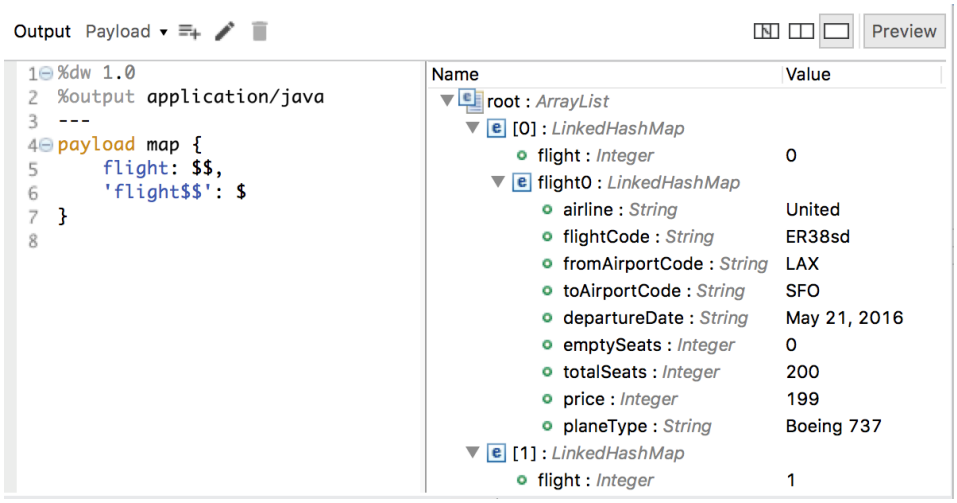


29. Stop the project and switch perspectives.

Walkthrough 11-3: Transform complex data structures with arrays

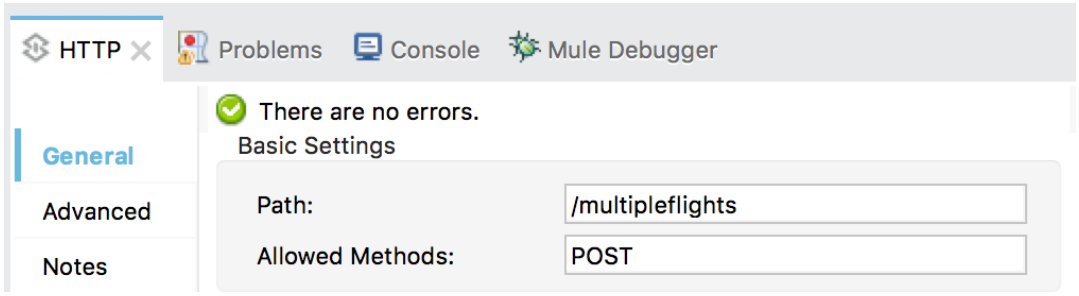
In this walkthrough, you work with JSON data for multiple flights posted to a flow. You will:

- Create a new flow that receives POST requests of a JSON array of objects.
- Transform a JSON array of objects to Java.



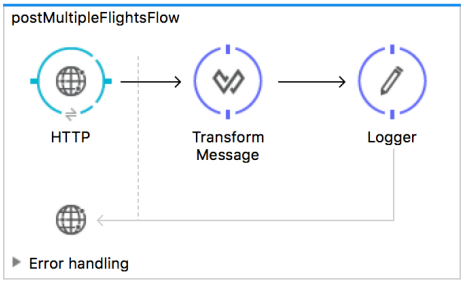
Create a new flow

1. Return to implementation.xml.
2. Drag out an HTTP connector to the bottom of the canvas.
3. Change the flow name to postMultipleFlightsFlow
4. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /multipleflights and set the allowed methods to POST.



Transform the payload to Java

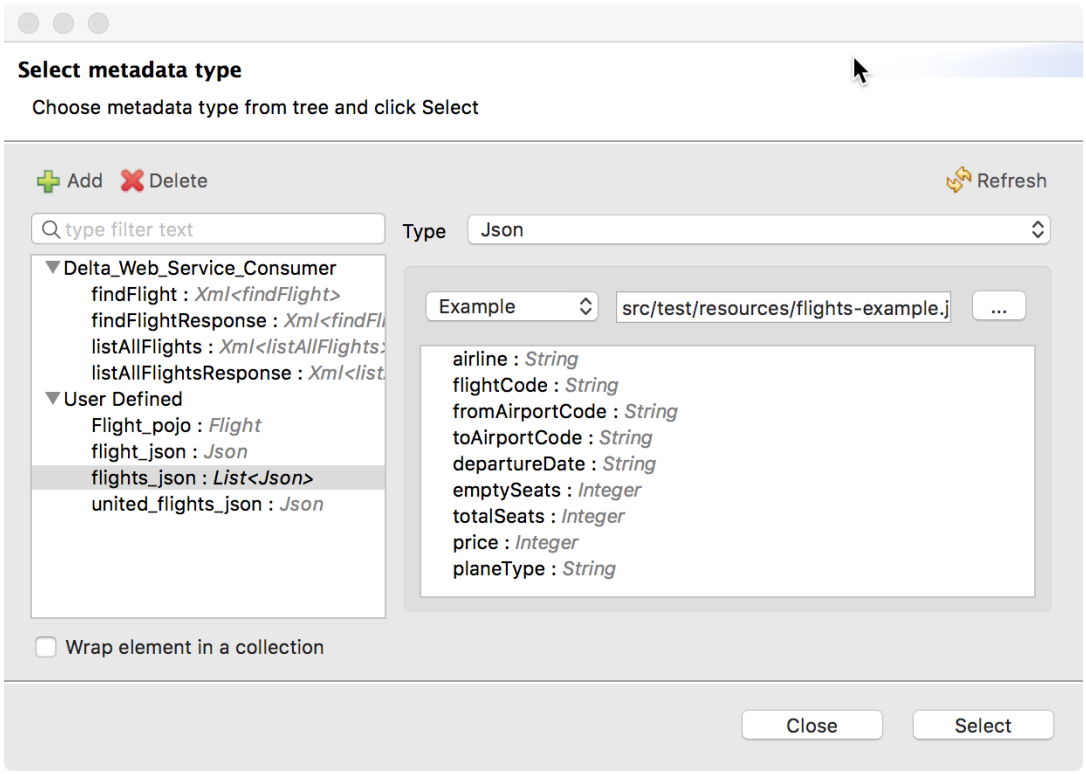
6. Add a Transform Message component and a Logger to the flow.



7. In the Transform Message properties view, set the DataWeave expression to payload.
8. Click the Preview button.

Add HTTP Listener output metadata

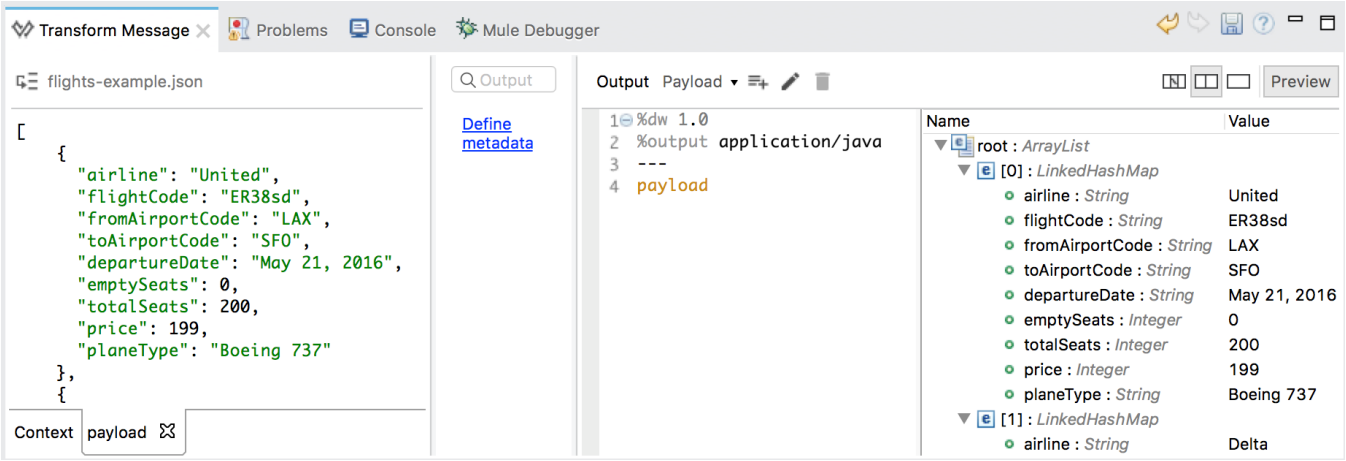
9. In the HTTP Listener Properties view, select Metadata and click the Add metadata button.
10. Select Output:Payload and click the Edit button.
11. In the Select metadata type dialog box, select flights_json and click Select.
12. In the Select metadata type dialog box, click Select.



13. In the Transform Message Properties view, you should now see metadata for the input.

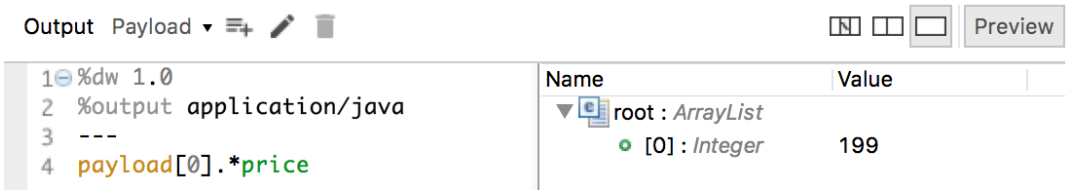
Preview sample data and sample output

14. In the preview section, click the Create required sample data to execute preview link.
15. Look at the preview section; the sample output should be an ArrayList of LinkedHashMaps.



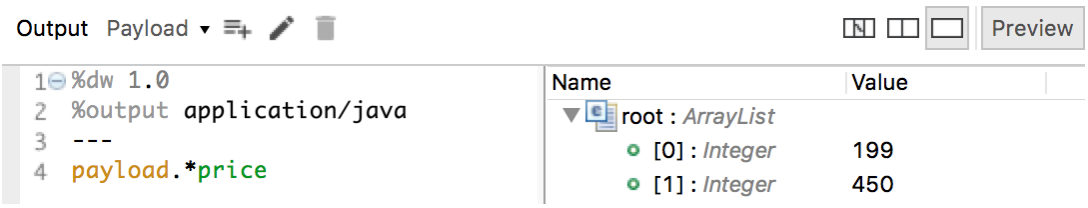
Return values for the first object in the collection

16. Change the DataWeave expression to payload[0]; in the preview section, you should see one LinkedHashMap object.
17. Change the DataWeave expression to payload[0].price; in the preview section, you should get 199, the first price value.
18. Change the DataWeave expression to payload[0].*price; in the preview section, you should see an ArrayList with one integer value of 199 – the first price value.



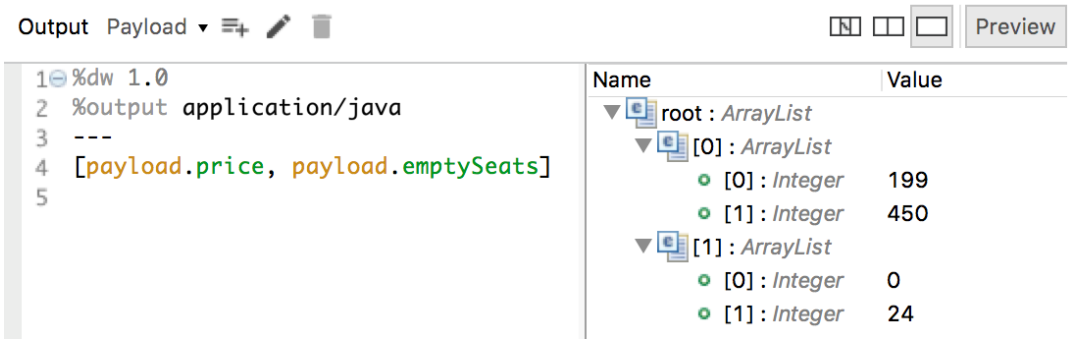
Return collections of values

19. Change the DataWeave expression to return a collection of all the prices; in the preview section, you should see two prices in an ArrayList.



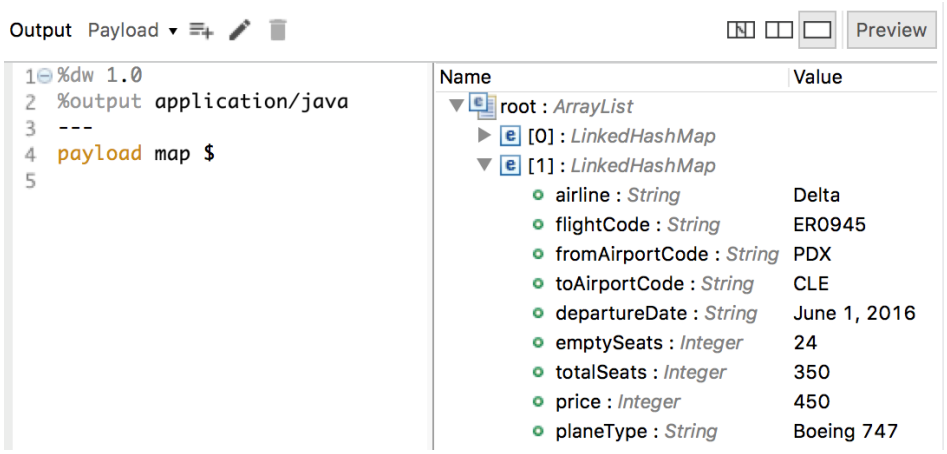
20. Change the DataWeave expression to payload.price; you should get the same result.
21. Change the DataWeave expression to return a collection of prices and available seats; in the preview section, you should see an ArrayList with two ArrayLists.

```
[payload.price, payload.emptySeats]
```

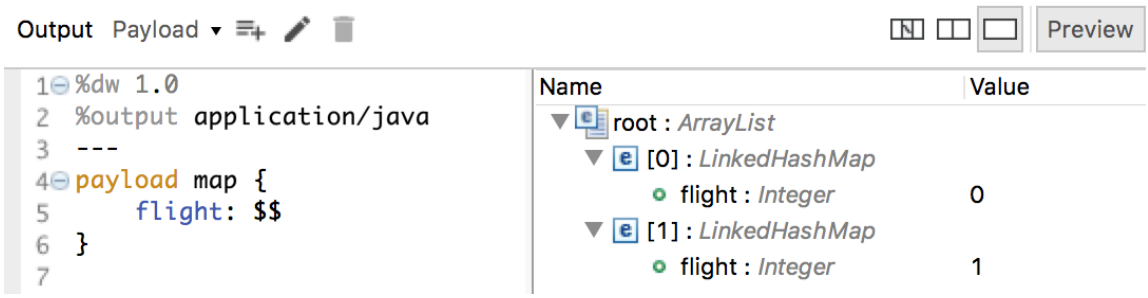


Use the map operator to return object collections with different data structures

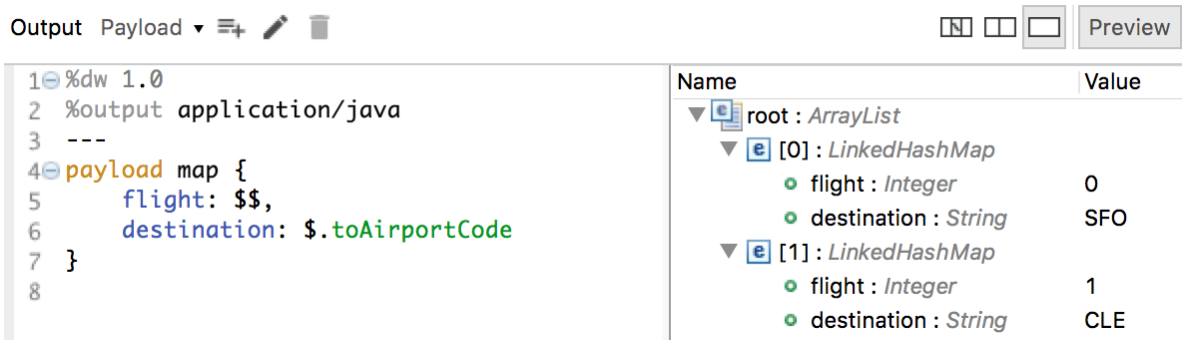
22. Change the DataWeave expression to payload map $; in the preview section, you should see two ArrayList of two LinkedHashMaps again.



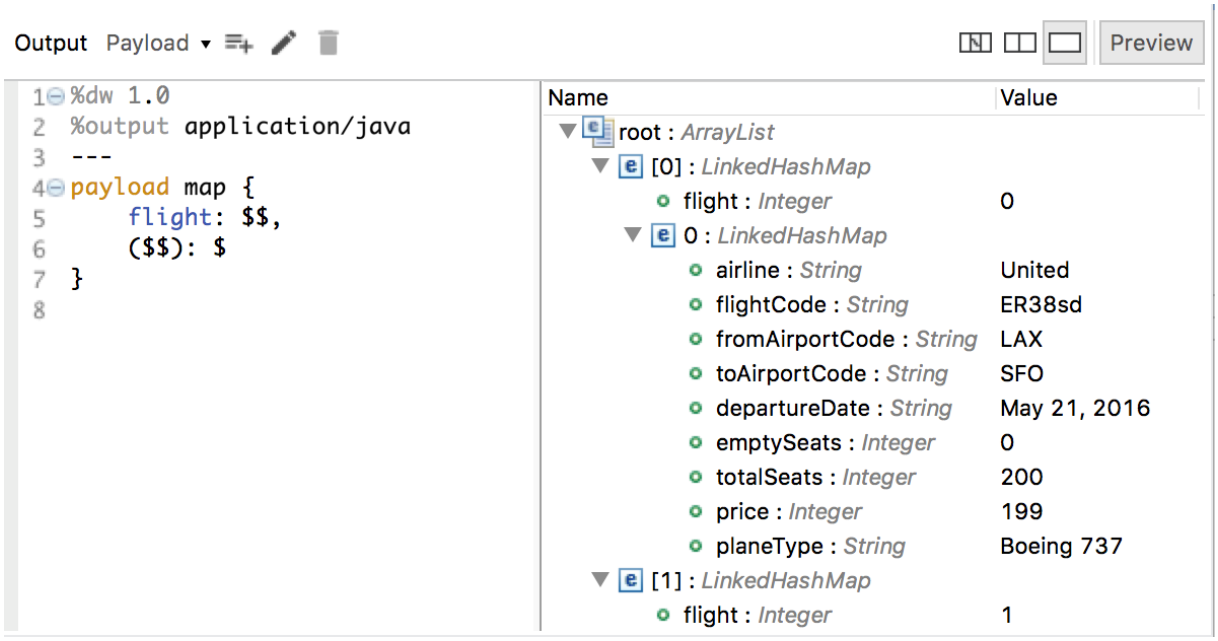
23. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.



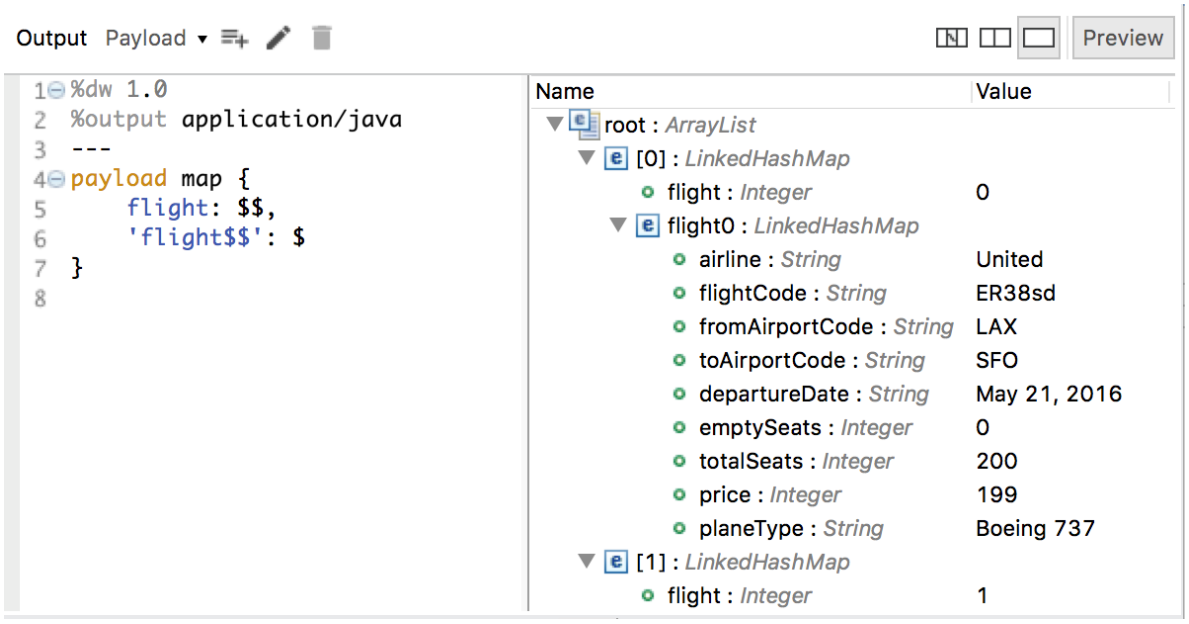
24. Change the DataWeave expression to create a property called destination for each object that is equal to the value of the toAirportCode field in the payload collection.



25. Change the DataWeave expression to dynamically add each of the properties present on the payload objects.



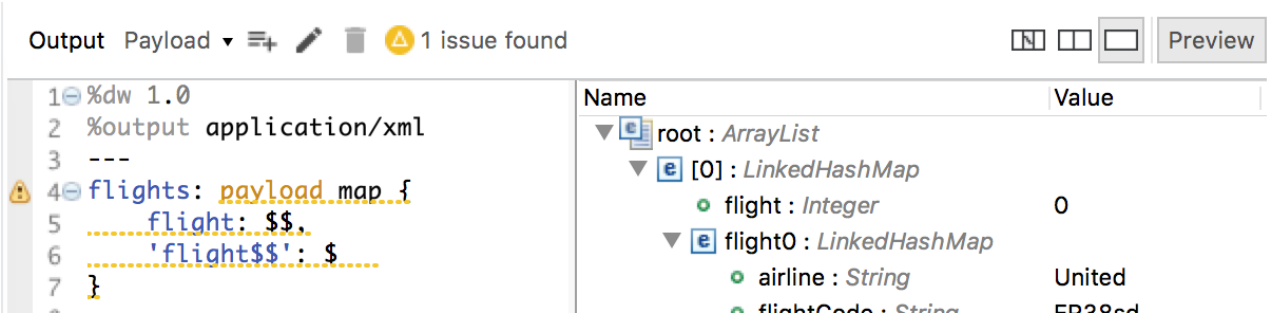
26. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.



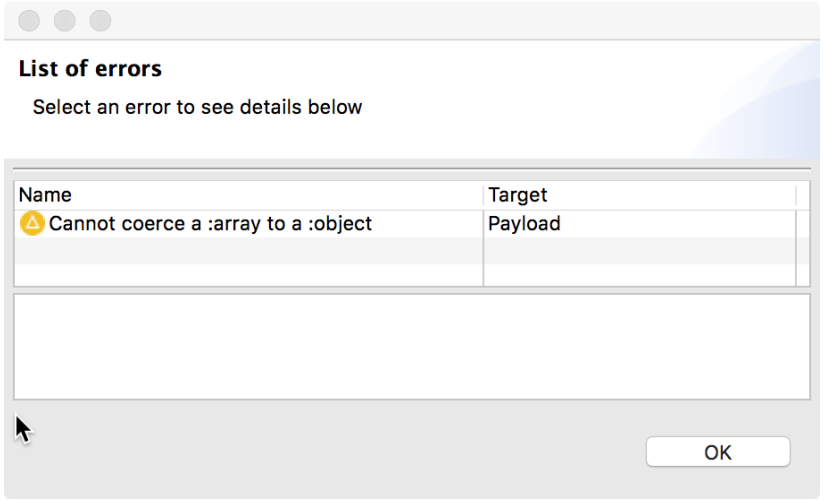
Note: If you want to test the application, change the output type to application/json and then in Postman, change the request URL to http://localhost:8081/multipleflights and replace the request body with JSON from the flights-example.json file, .

Change the expression to output XML

27. Change the DataWeave expression output type from application/java to application/xml; you should get an issue displayed.
28. Change the DataWeave expression to create a root node called flights; you should still get an issue.



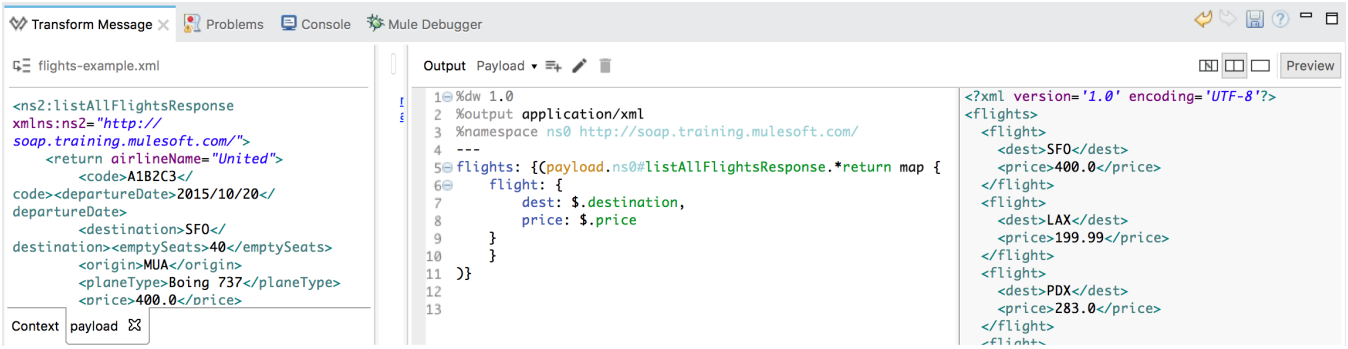
29. Look at the error message.



Walkthrough 11-4: Transform to and from XML with repeated elements

In this walkthrough, you continue to work with the JSON data for multiple flights posted to the flow. You will:

- Transform a JSON array of objects to XML.
- Transform XML with repeated elements to Java.

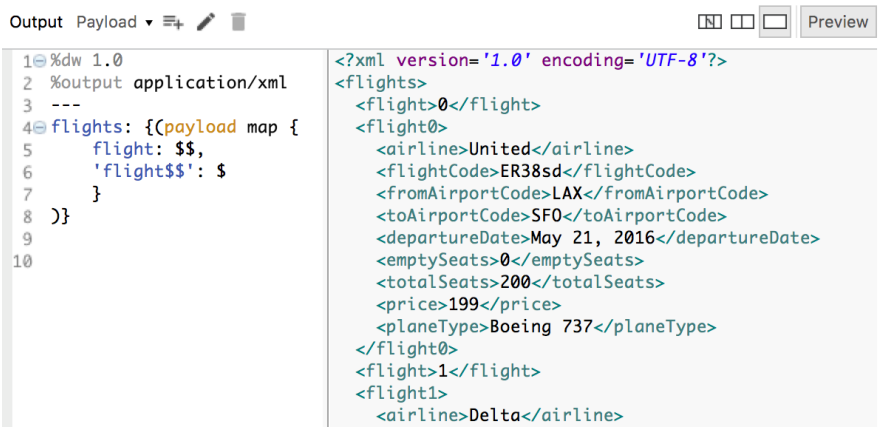


Transform the JSON array of objects to XML

1. Return to the Transform Message Properties view in postMultipleFlightsFlow.
2. Change the expression to map each item in the input array to an XML element.

```
flights: {(payload map {
      flight: $$,
      'flight$$':$
    }
)}
```

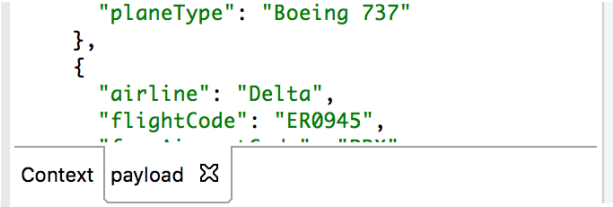
3. Look at the preview; the JSON should be transformed to XML successfully.



4. Remove the flight property.
5. Modify the expression so the flights object has a single property called flight.

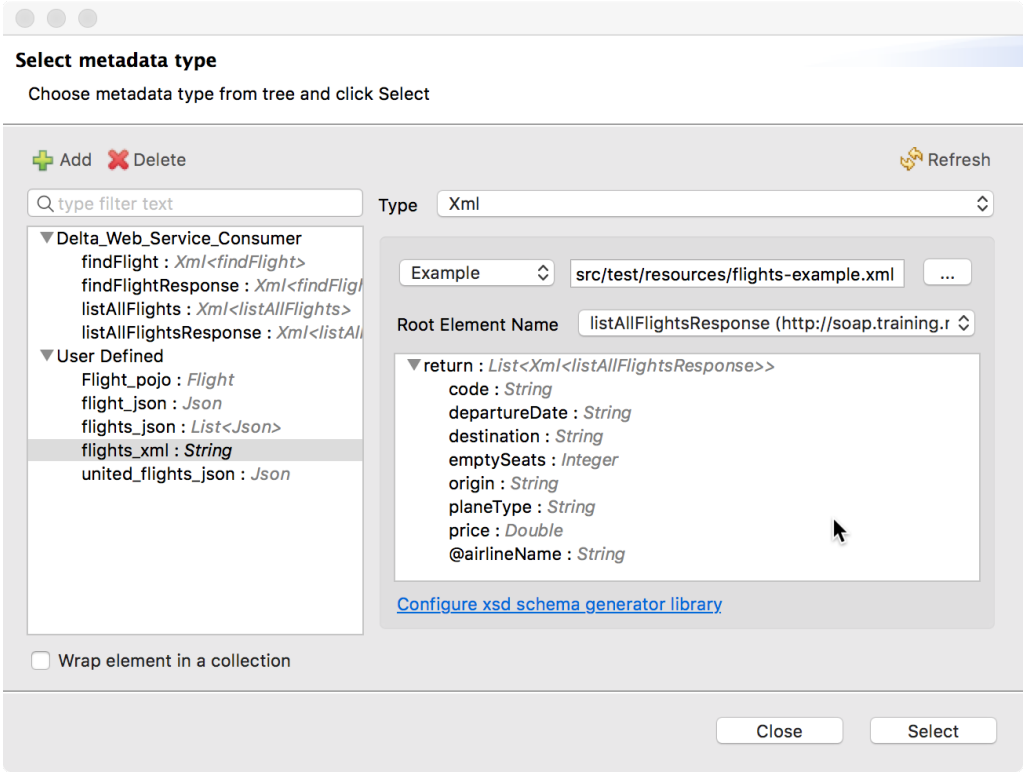
Change the HTTP Listener output metadata to XML

6. In the input section, click the x on the payload tab.



7. In the Close Sample Data dialog box, click Close tab and Keep file.
8. In the input section, right-click Payload: List<Json> and select Clear Metadata.
9. In the HTTP Listener Properties view, select Metadata and click the Edit button.
10. In the Select metadata type dialog box, click Add.
11. In the Create new type dialog box, set the type id to flights_xml and click Create type.
12. In the Select metadata dialog box, set the type to XML.

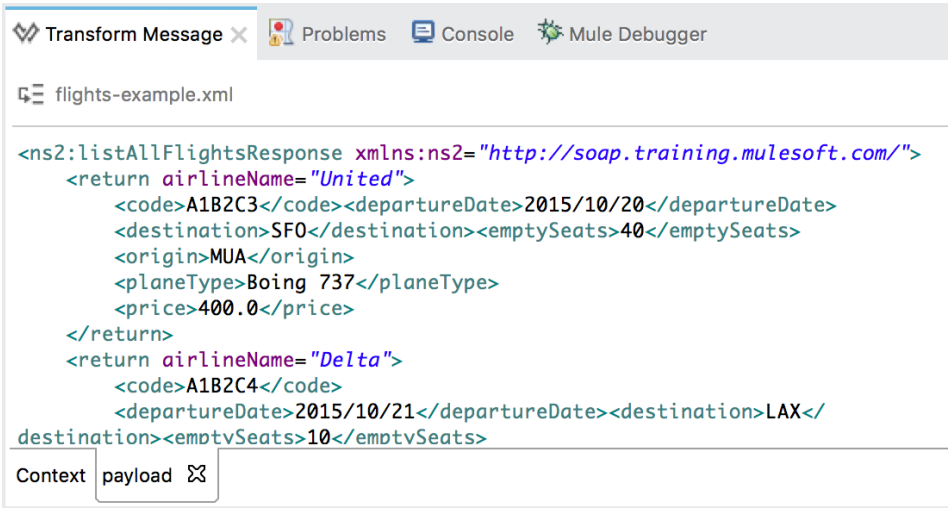
13. Select Example and browse to and select the flights-example.xml file in src/test/resources.



14. In the Select metadata type dialog box, click Select.
15. In the Transform Message Properties view, you should now see metadata for the input.

Preview sample data and sample output

16. In the preview section, click the Create required sample data to execute preview link.
17. Look at the XML sample payload in the input section.



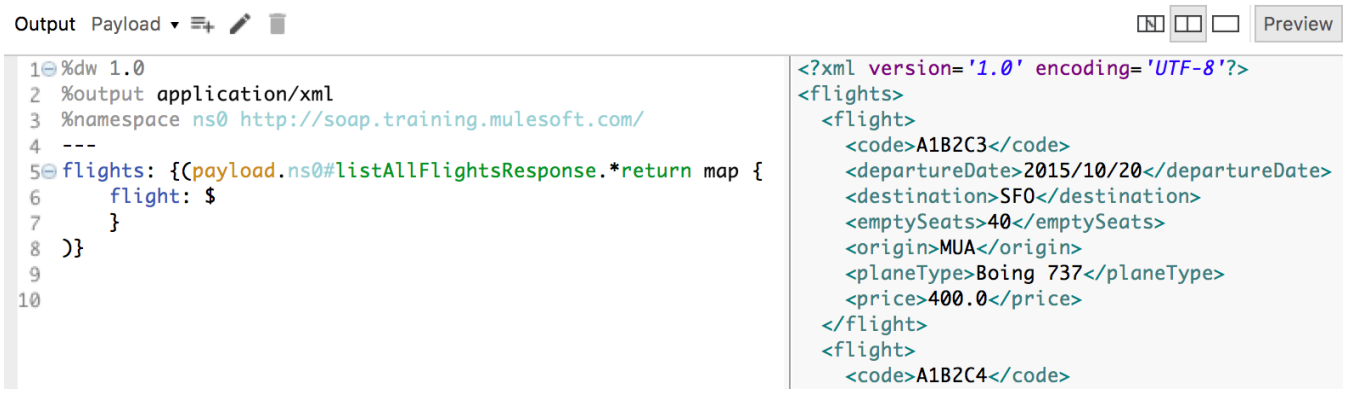
18. Review the transformation expression.
19. Look at the preview section; you should see all the flights are children of the flight element.

Change the return structure of the XML

20. Change the DataWeave expression to loop over the listAllFlightsResponse element of the payload; you should see the flights are now correctly transformed.

21. Change the DataWeave expression to loop over the payload.listAllFlightsResponse.return element.

22. Change the DataWeave expression use * to loop over the XML repeated return elements.



23. Change the DataWeave expression to return flight elements with dest and price elements that map to the corresponding destination and price input values.

Note: If you want to test the application with Postman, be sure to change the content-type header to application/XML and replace the request body with XML from the flights-example.xml file.

Change the transformation to return Java

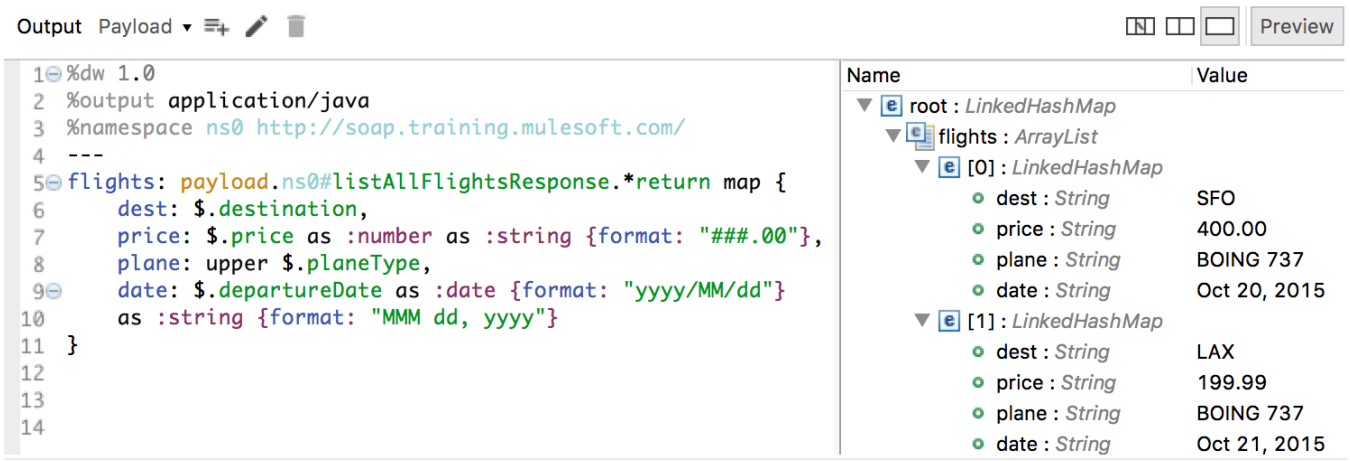
24. Change the output type from application/xml to application/java.
25. Look at the preview; you should see flights is a LinkedHasMap with one flight property.

26. In the DataWeave expression, remove the {(}) around the map expression.
27. Change the DataWeave expression to remove the flight property.
28. Look at the preview; you should see flights is an ArryaList with five LinkedHashMaps.

Walkthrough 11-5: Coerce and format strings, numbers, and dates

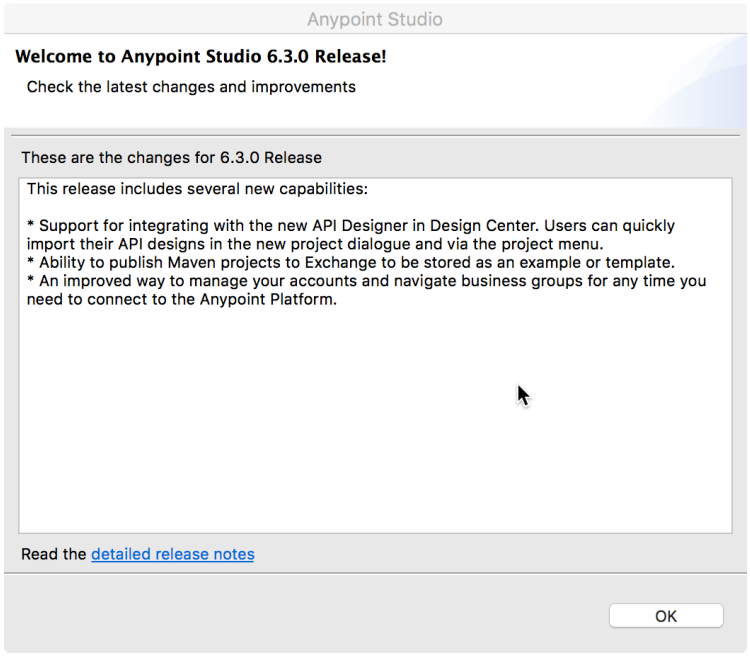
In this walkthrough, you continue to work with the XML flights data posted to the flow. You will:

- Explore the DataWeave documentation.
- Coerce data types.
- Format strings, numbers, and dates.

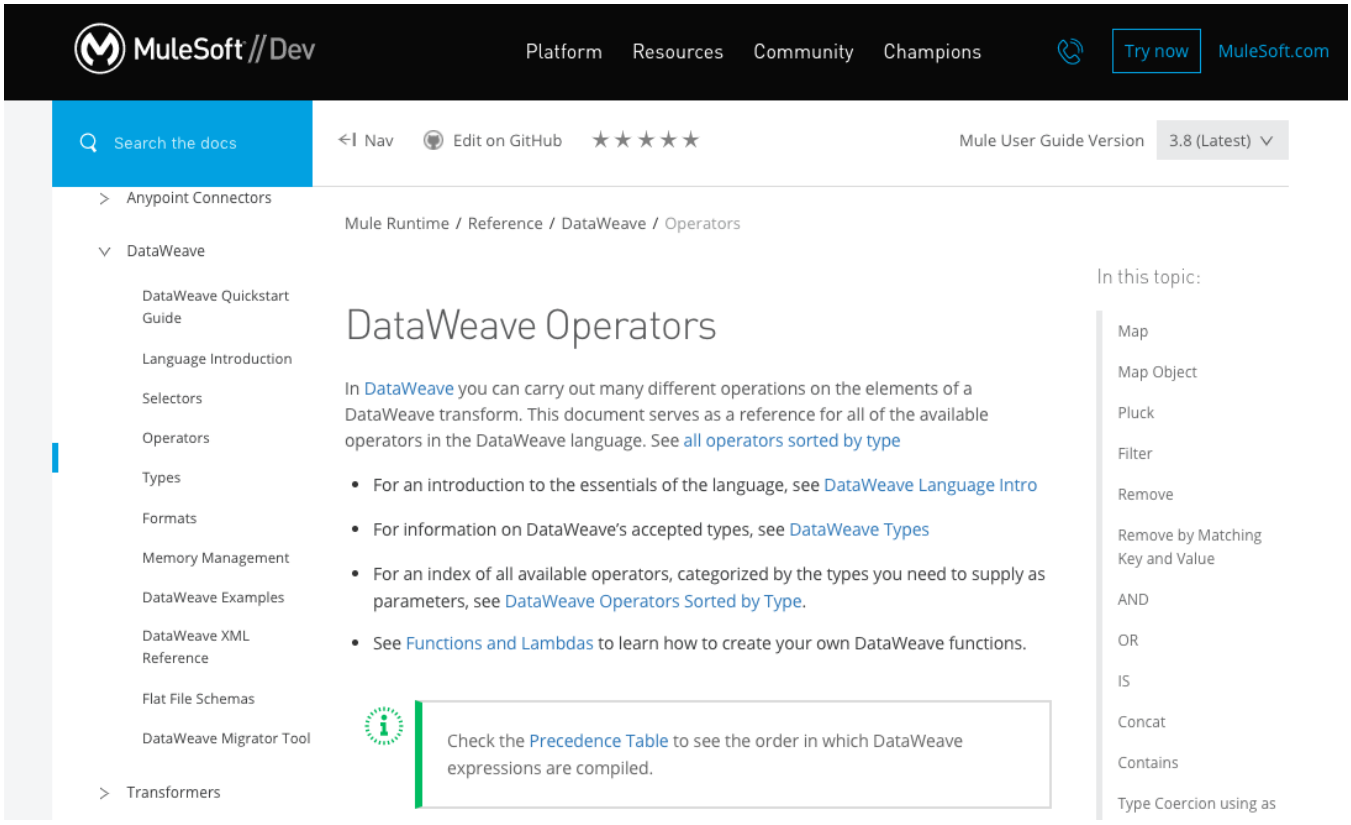


Browse DataWeave reference documentation

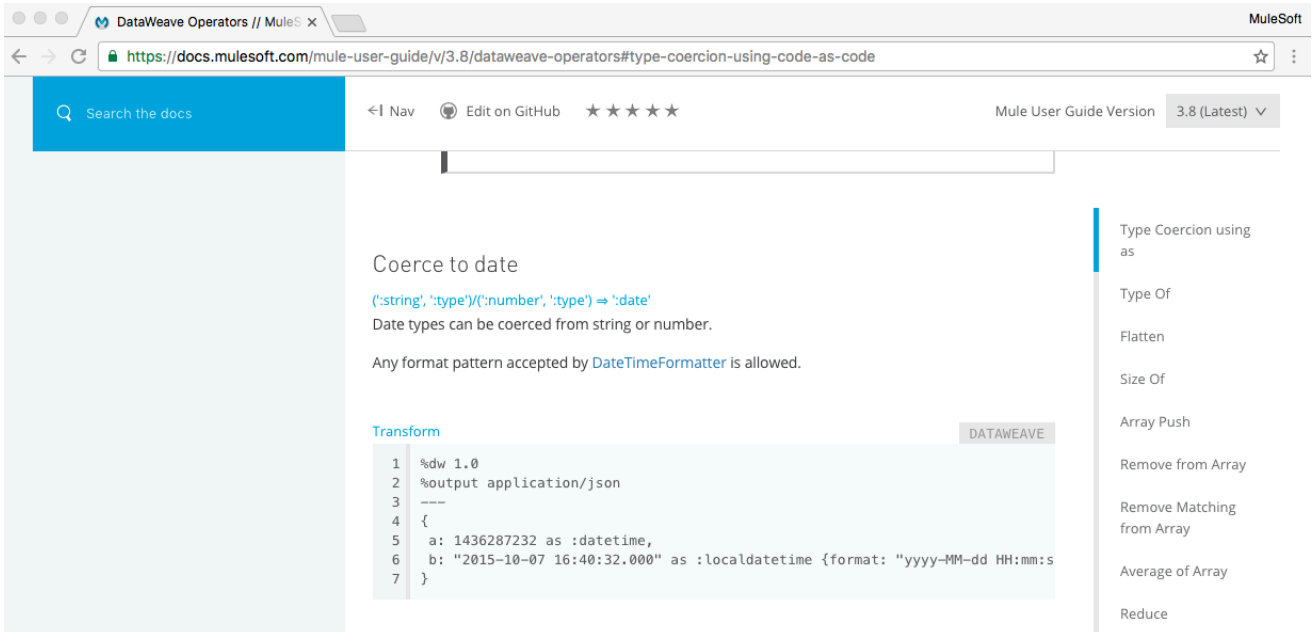
1. In the main menu bar in Anypoint Studio, select Help > What's new in Anypoint Studio?
2. In the Anypoint Studio dialog box, click the detailed release notes link; a new web browser window should open.



3. In the left-side navigation, navigate to Mule Runtime > Reference > DataWeave > Operators.

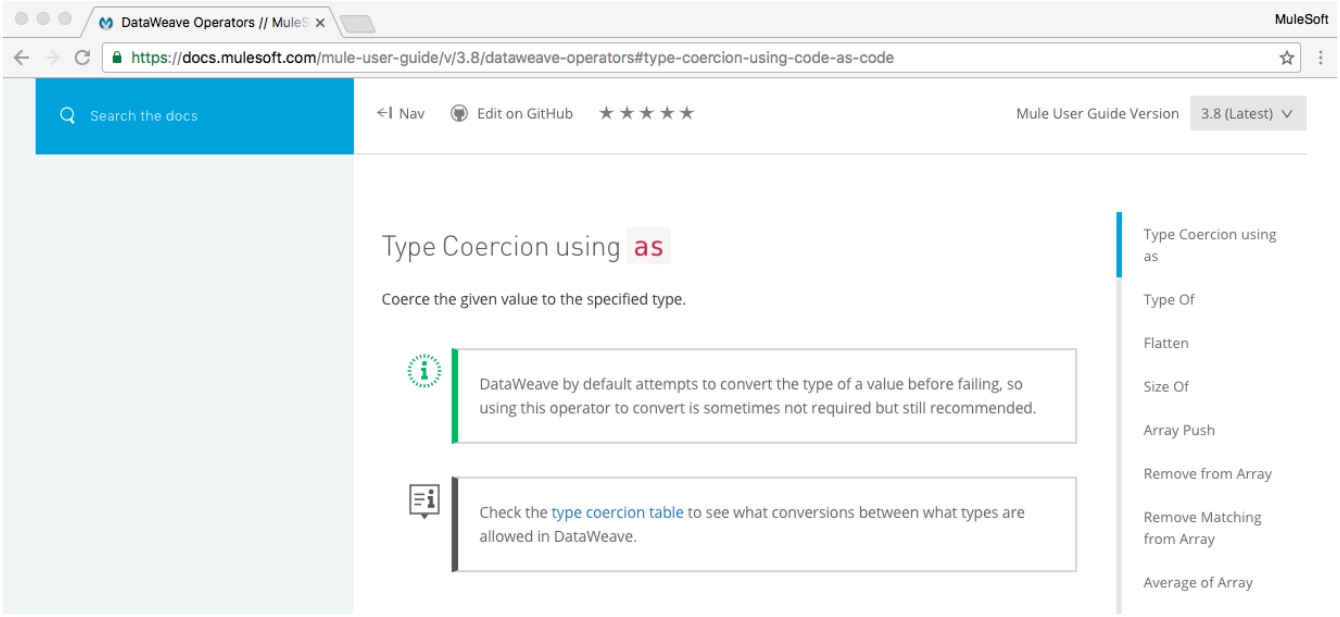


4. In the right-side navigation, click the Type Coercion using as link.
5. Scroll the page and browse the examples and documentation.

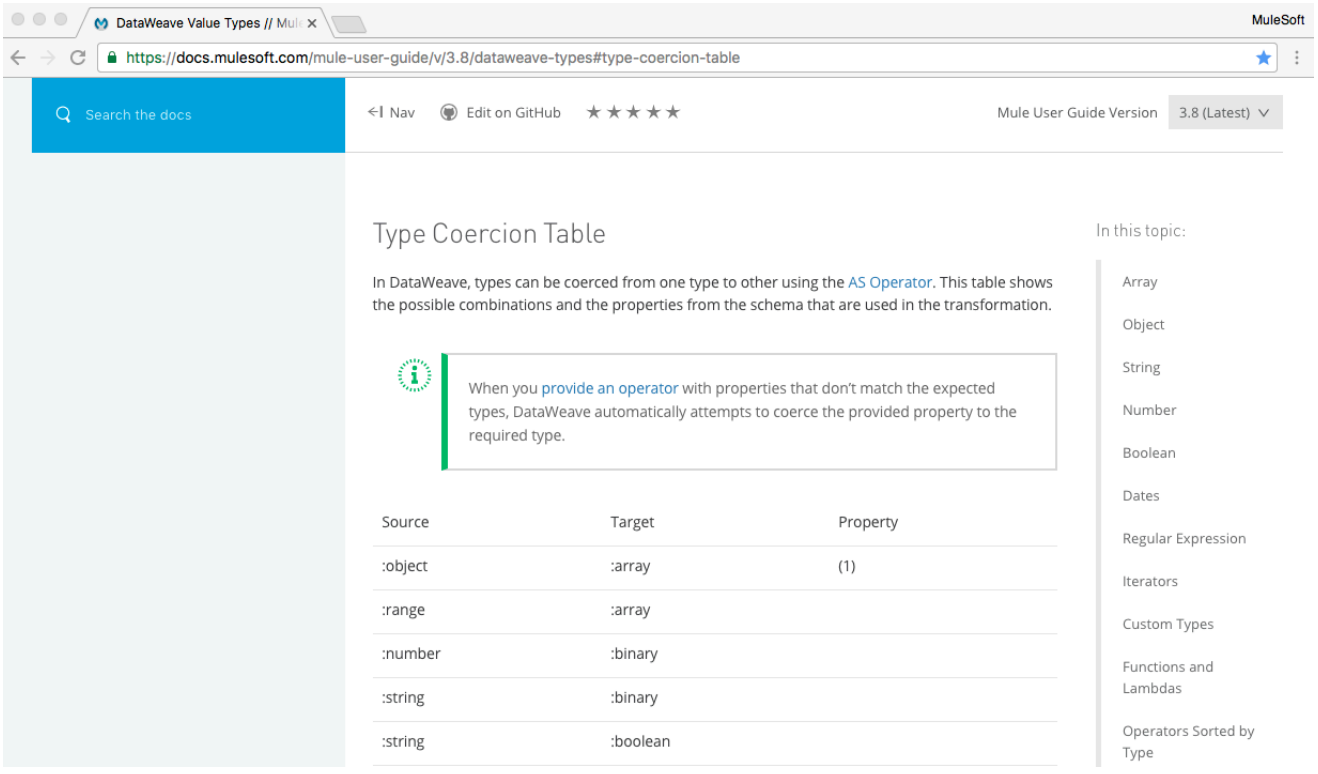


6. In the right-side navigation, click the Type Coercion using as link again.

7. In the page, locate and click the type coercion table link.



8. Browse the type coercion table.



Format a string

9. Return to Anypoint Studio.
10. In the Anypoint Studio dialog box, click OK.
11. Navigate to the Transform Message Properties view for the transformation in postMultipleFlightsFlow.
12. Change the DataWeave expression to return objects that also have a property called plane equal to the value of the input planeType values.
13. Use the upper operator to return the value in uppercase.

```
plane: upper $.planeType
```

14. Look at the preview; you should see the uppercase plane fields.

Coerce a string to a number

15. In the preview, look at the data type of the prices; you should see that they are strings.
16. Change the DataWeave expression to use the as operator to return the prices as numbers.

```
price: $.price as :number,
```

17. Look at the preview; you should see the prices are now either Integer or Double objects.

Coerce a string to a specific type of number object

18. Change the DataWeave expression to use the class metadata key to coerce the prices to java.lang.Double objects.

```
price: $.price as :number {class:"java.lang.Double"},
```

19. Look at the preview; you should see the prices are now all Double objects.

Format a number

20. Use the format schema property in the DataWeave expression to format the prices to zero decimal places.
21. Remove the coercion to a java.lang.Double.

```
price: $.price as :number as :string {format: "###"},
```

22. Look at the preview; the prices should now be formatted.

23. Change the DataWeave expression to format the prices to two decimal places.

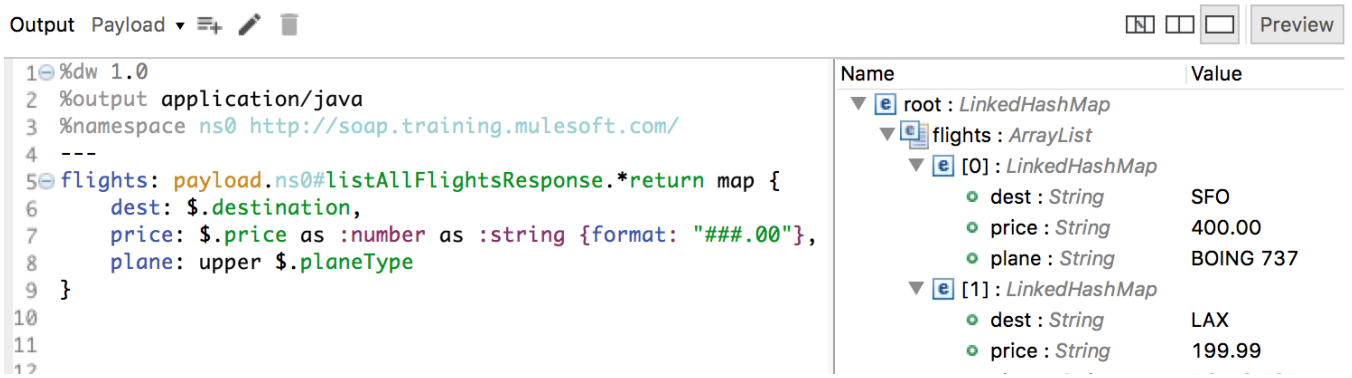
```
price: $.price as :number as :string {format: "###.##"},
```

24. Look at the preview; the prices should be formatted differently.

25. Change the DataWeave expression to format the prices to two minimal decimal places.

```
price: $.price as :number as :string {format: "###.00"},
```

26. Look at the preview; all the prices should be formatted to two decimal places.



Note: If you are not a Java programmer, you may want to look at the Java documentation for the DecimalFormatter class to review documentation on patterns.
https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html

Coerce a string to a date

27. Add a date field to the return object.

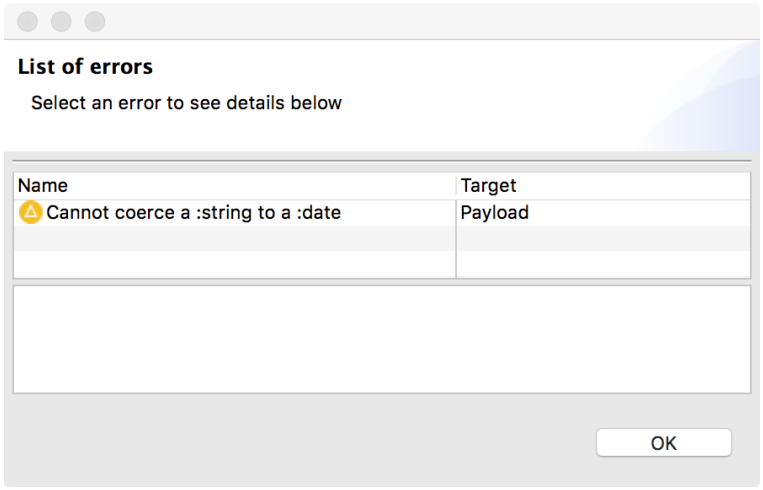
```
date: $.departureDate
```

28. Look at the preview; you should see the date property is a String.

29. Change the DataWeave expression to use the as operator to convert departureDate to a date object; you should get an exception.

```
date: $.departureDate as :date
```

30. Look at the exception.

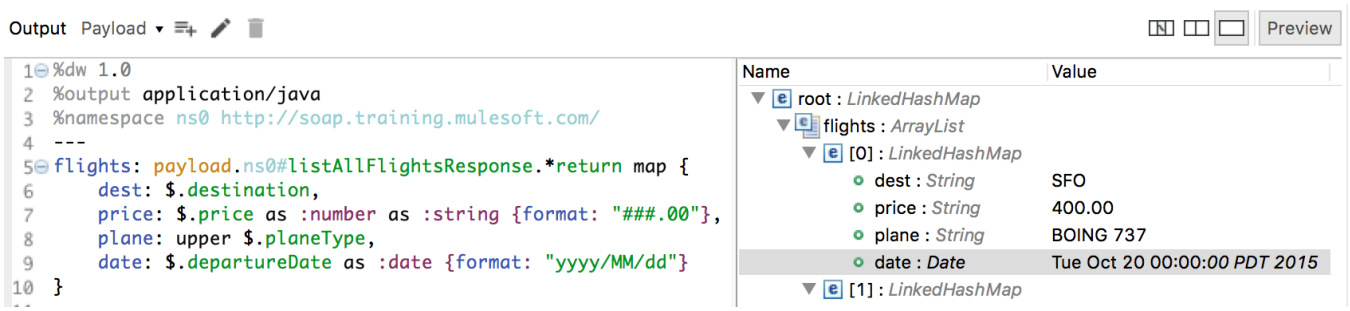


31. Look at the format of the dates in the preview.
32. Change the DataWeave expression to use the format schema property to specify the pattern of the input date strings.

```
date: $.departureDate as :date {format: "yyyy/MM/dd"}
```

Note: If you are not a Java programmer, you may want to look at the Java documentation for the DateTimeFormatter class to review documentation on pattern letters.
https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

33. Look at the preview; you should see date is now a Date object.



Format a date

34. Use the as operator to convert the dates to strings, which can then be formatted.

```
date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
```

35. Look at the preview section; the date is again a String – but with a different format.

36. Use the format schema property with any pattern letters and characters to format the date strings.

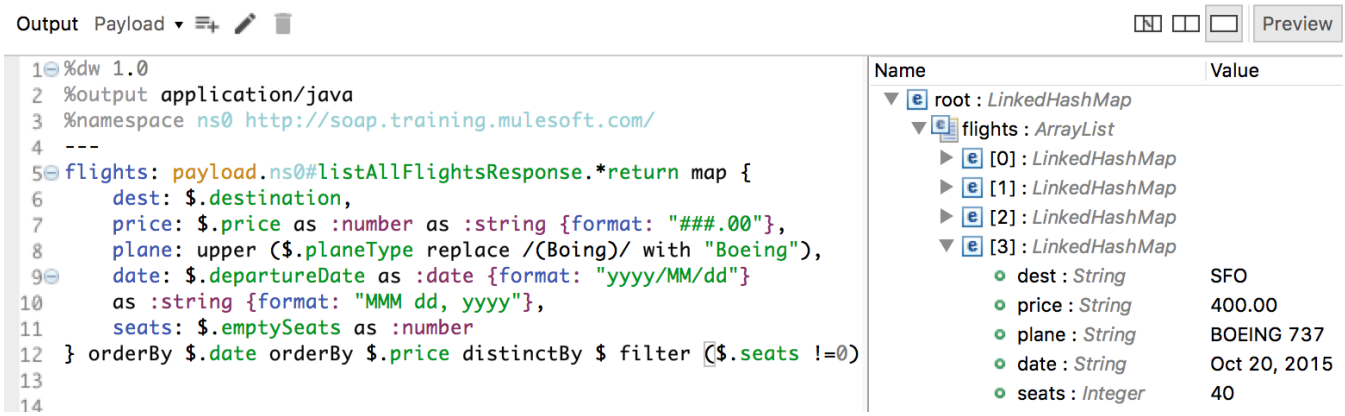
```
date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
{format: "MMM dd, yyyy"}
```

37. Look at the preview; the dates should now be formatted according to the pattern.

Walkthrough 11-6: Use DataWeave operators

In this walkthrough, you continue to work with the flights JSON posted to the flow. You will:

- Replace data values using pattern matching.
- Order data, remove duplicate data, and filter data.



Replace data values

1. Return to the Transform Message Properties view for the transformation in postMultipleFlightsFlow.
2. Use the replace operator in the DataWeave expression to replace the string Boing with Boeing.

```
plane: upper $.planeType replace /(Boing)/ with "Boeing",
```

3. Look at the preview section; Boeing should not be spelled correctly.
4. In the DataWeave expression, place parentheses around the replace operator expression.

```
plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
```

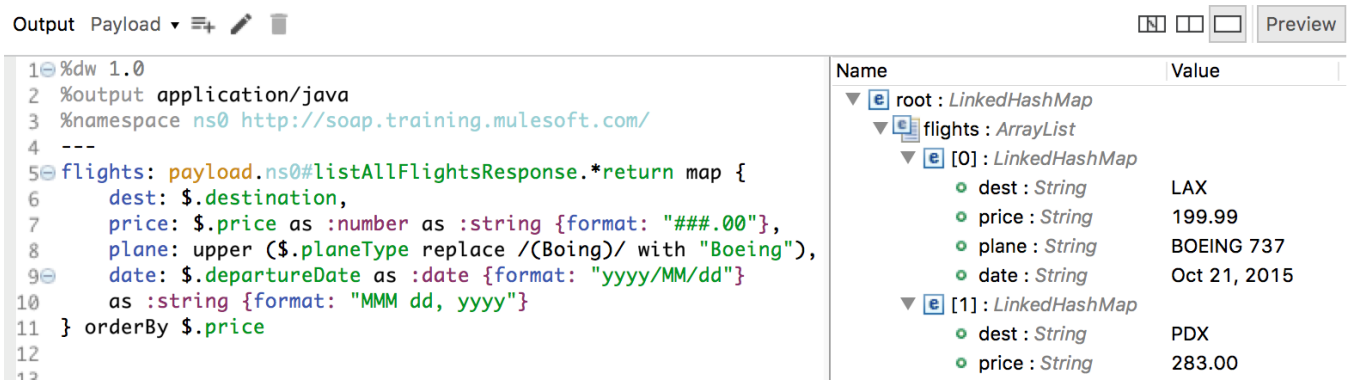
5. Look at the preview section; Boeing should now be spelled correctly.

Order data

6. In the preview section, look at the flight prices; the flights should not be ordered by price.
7. Change the DataWeave expression to use the orderBy operator to order the objects by price.

```
payload.listAllFlightsResponse.*return map {
        …
} orderBy $.price
```

8. Look at the preview; the flights should now be ordered by price.



9. Look at the PDX flights; they should be ordered by date in addition to price.
10. Change the DataWeave expression to first sort by date and then by price.

```
payload.listAllFlightsResponse.*return map {
        …
} orderBy $.date orderBy $.price
```

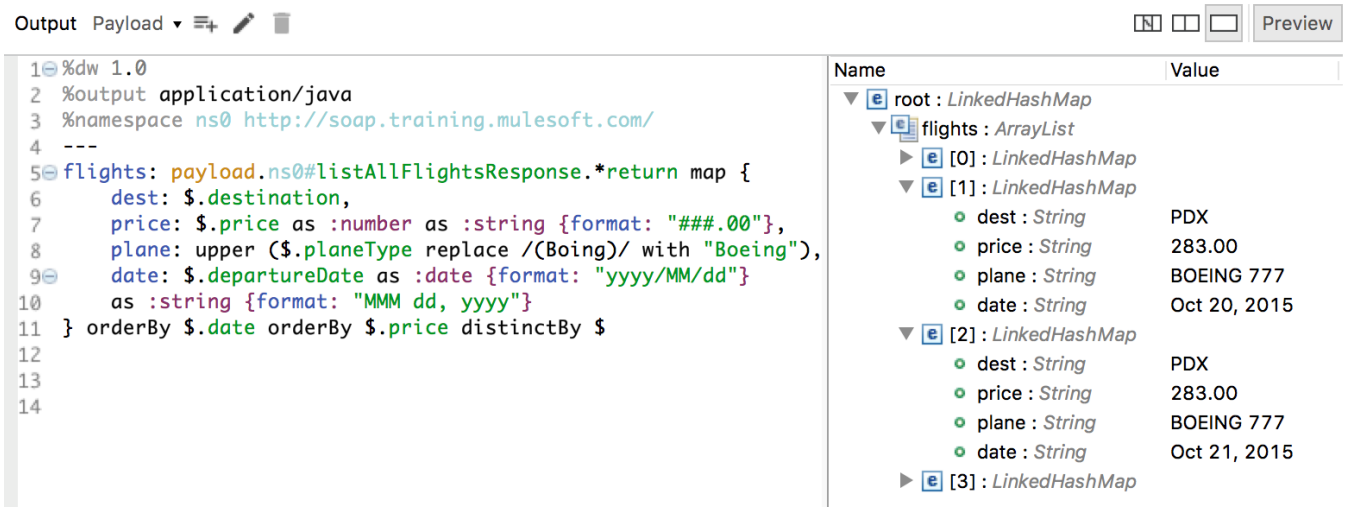
11. Look at the preview; the flights should now be ordered by price and flights of the same price should be sorted by date.

Remove duplicate data

12. Use the distinctBy operator in the DataWeave expression to remove any duplicate objects.

```
payload.listAllFlightsResponse.*return map {
      …
} orderBy $.date orderBy $.price distinctBy $
```

13. Look at the preview; you should now get only four flights instead of five.



14. Add a seats field that is equal to the emptySeats field and coerce it to a number.

```
seats: $.emptySeats as :number
```

15. Look at the preview section; you should get five flights again.

Filter data

16. In the preview section, look at the values of the seats properties; you should see some are equal to zero.
17. Add a filter to the DataWeave expression that removes any objects that have availableSeats equal to 0.

```
payload.listAllFlightsResponse.*return map {
        …
} orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)
```

18. Look at the preview; you should no longer get the flight that had no available seats.

Walkthrough 11-7: Define and use custom data types

In this walkthrough, you continue to work with the flight JSON posted to the flow. You will:

- Define and use custom data types.
- Transform objects to POJOs.

Define a custom data type

1. Return to the Transform Message Properties view for the transformation in postMultipleFlightsFlow.
2. Select and cut the string formatting expression for the prices.
3. In the header section of the transform section, define a custom data type called currency.
4. Set it equal to the value you copied.
5. Remove the as operator.

```
%type currency = :string {format: "###.00"}
```

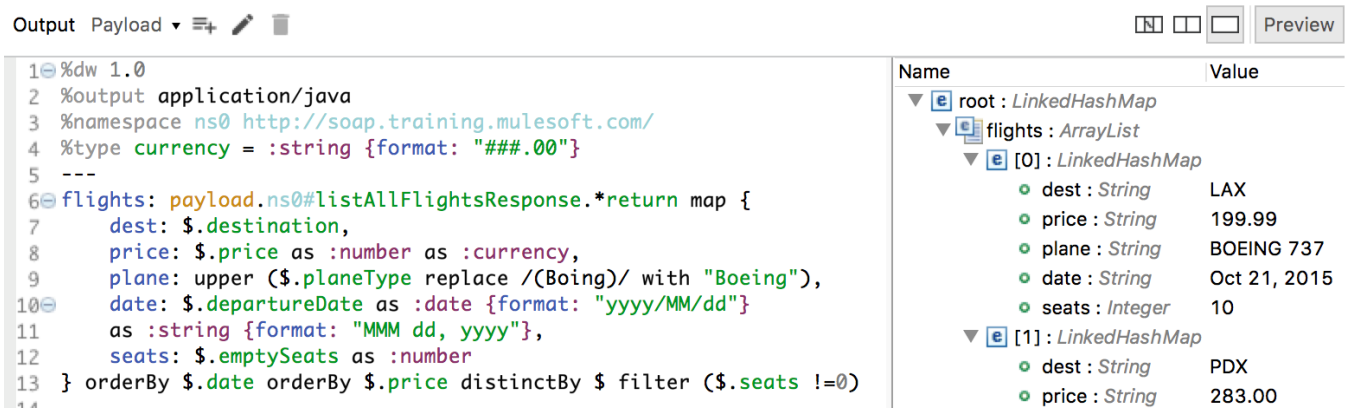
```
%dw 1.0
%output application/java
%namespace ns0 http://soap.training.mulesoft.com/
%type currency = :string {format: "###.00"}
---
```

Use a custom data type

6. In the DataWeave expression, set the price to be of type currency.

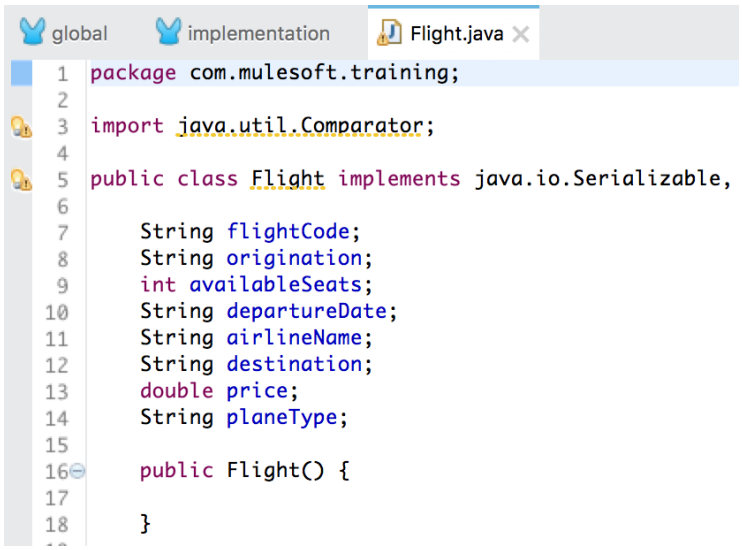
```
price: $.price as :number as :currency,
```

7. Look at the preview; the prices should still be formatted as strings to two decimal places.



Transform objects to POJOs

8. Open the Flight.java class in the project's src/main/java folder and look at the names of the properties.



9. Return to postMultipleFlightsFlow in implementation.xml.
10. In the header section of the transform section, define a custom data type called flight that is of type com.mulesoft.traning.Flight.

```
%type flight = :object {class: "com.mulesoft.training.Flight"}
```

```
%dw 1.0
%output application/java
%namespace ns0 http://soap.training.mulesoft.com/
%type currency = :string {format: "###.00"}
%type flight = :object {class: "com.mulesoft.training.Flight"}
---
```

11. In the DataWeave expression, set the map objects to be of type flight.

```
payload.listAllFlightsResponse.*return map {
      …
} as :flight orderBy $.date orderBy $.price distinctBy $ filter
($.seats !=0)
```

12. Change the name of the dest key to destination.

```
destination: $.destination,
```

13. Change the other keys to match the names of the Flight class properties:

- plane to planeType
- date to departureDate
- seats to availableSeats

14. Change the operators to use the new property names.

```
 payload.listAllFlightsResponse.*return map {
      …
} as :flight orderBy $.departureDate orderBy $.price distinctBy $
filter ($.availableSeats !=0)
```

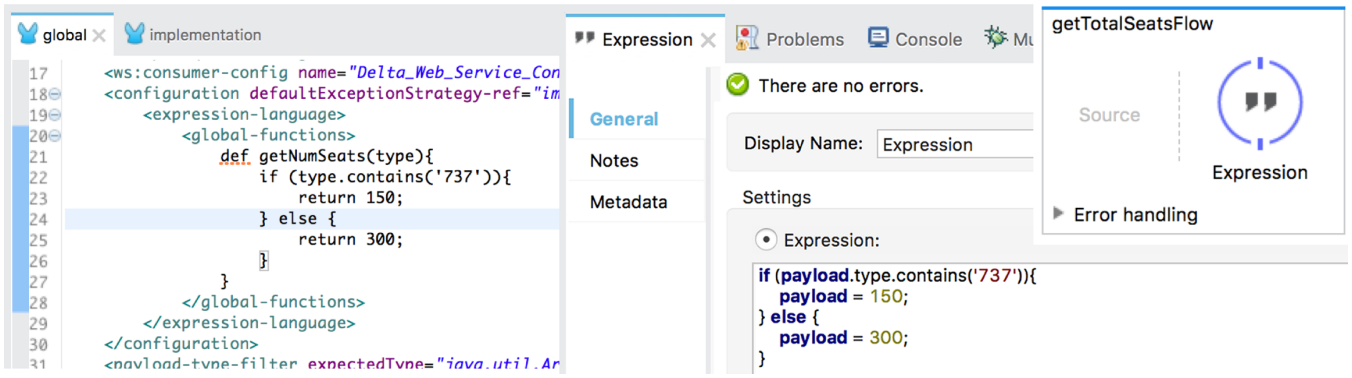
15. Look at the preview; you should now get an ArrayList of Flight objects.

Walkthrough 11-8: Call MEL functions and other flows

In this walkthrough, you continue to work with the DataWeave transformation in getMultipleFlightsFlow. You will:

- Define a global MEL function in global.xml.
- Call a global MEL function in a DataWeave expression.
- Call a flow in a DataWeave expression.

```
//totalSeats: getNumSeats($.planeType)
totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
```



Define a global MEL function in global.xml

1. Return to global.xml and switch to the Configuration XML view.
2. Locate the configuration element.
3. On a new line between the configuration tags, type <e and the press Ctrl+Space for autocompletion; an expression-language tag set should be added.

```
17    <ws:consumer-config name="Delta_Web_Service_Consumer" wsd
18    <configuration defaultExceptionStrategy-ref="Choice_Excep
19        <http:config useTransportForUris="false"/>
20        <expression-language></expression-language>
21    </configuration>
22    <payload-type-filter expectedType="java.util.ArrayList" n
```

4. Place the expression-language tags on different lines and place the cursor between them.
5. Type <g, press Ctrl+Space, and select global-functions; a global-functions tag set should be added.

6. Place the global-functions tags on different lines and place the cursor between them.

```
17      <ws:consumer-config name="Delta_Web_Service_Consumer" ws
18      <configuration defaultExceptionStrategy-ref="Choice_Exce
19          <http:config useTransportForUris="false"/>
20          <expression-language>
21              <global-functions>
22
23              </global-functions>
24          </expression-language>
25      </configuration>
26      <payload-type-filter expectedType="java.util.ArrayList"
```

7. Use the def keyword to define a function called getNumSeats with an argument called type.

```
def getNumSeats(type){
}
```

```
20          <expression-language>
21              <global-functions>
22                  def getNumSeats(type){
23
24                  }
25              </global-functions>
26          </expression-language>
```

8. Inside the function, add an if/else block that checks to see if type contains the string 737.

```
if (type.contains('737')){
}
```

9. If the expression is true, return 150 and if false, return 300.

```
if (type.contains('737')){
  return 150;
} else {
  return 300;
}
```

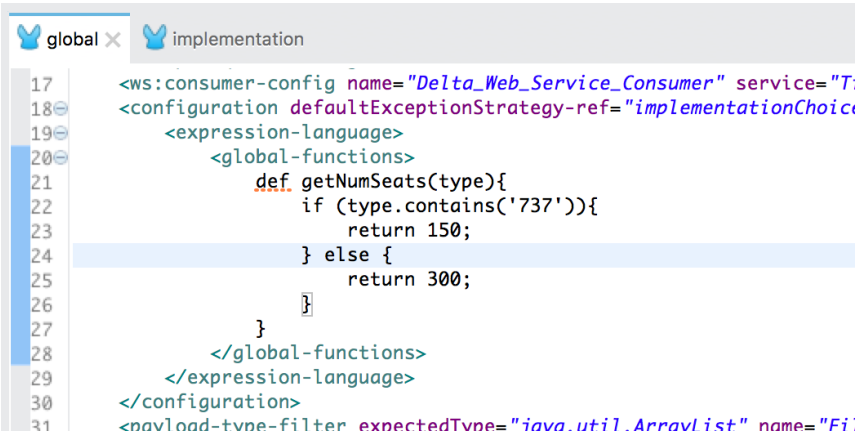
10. Save the file.
11. Run the project; the application should fail to deploy.

12. Locate the exception in the console.

```
ERROR 2016-06-07 19:53:03,383 [main] org.mule.module.launcher.application.DefaultMuleApplication: null
org.xml.sax.SAXParseException: cvc-complex-type.2.4.a: Invalid content was found starting with element
'expression-language'. One of '{"http://www.mulesoft.org/schema/mule/core":abstract-configuration-exten
sion}' is expected.
        at org.apache.xerces.util.ErrorHandlerWrapper.createSAXParseException(Unknown Source) ~[?:?]
        at org.apache.xerces.util.ErrorHandlerWrapper.error(Unknown Source) ~[?:?]
```

13. Return to global.xml and delete the http:config tag inside the configuration tag set.

```
<http:config useTransportForUris="false"/>
```



14. Run the project; the application should deploy.
15. Stop the project.

Call a global MEL function from DataWeave

16. Return to postMultipleFlightsFlow in implementation.xml.
17. In the Transform Message properties view, add a field called totalSeats inside the transformation function.
18. Set totalSeats equal to the return value from the getNumSeats() function.
19. Pass to getNumSeats(), the planeType.

```
totalSeats: getNumSeats($.planeType)
```

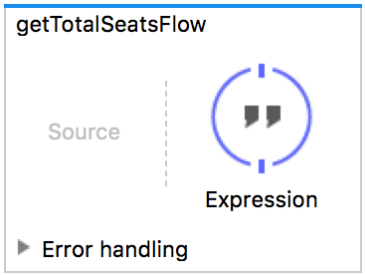
20. Look at the issue you get; the Flight class does not have a totalSeats property assigned.
21. Remove as :flight from the DataWeave expression.

    Note: You can also go add a new property to the Flight class.

22. Look at the preview; you should see the Flight objects now have a totalSeats property equal to 150 or 300.

Create a lookup flow

23. Return to implementation.xml.
24. Drag an Expression component (not a filter or a transformer!) from the Mule Palette and drop it at the bottom of the canvas to create a new flow.
25. Change the name of the flow to getTotalSeatsFlow.

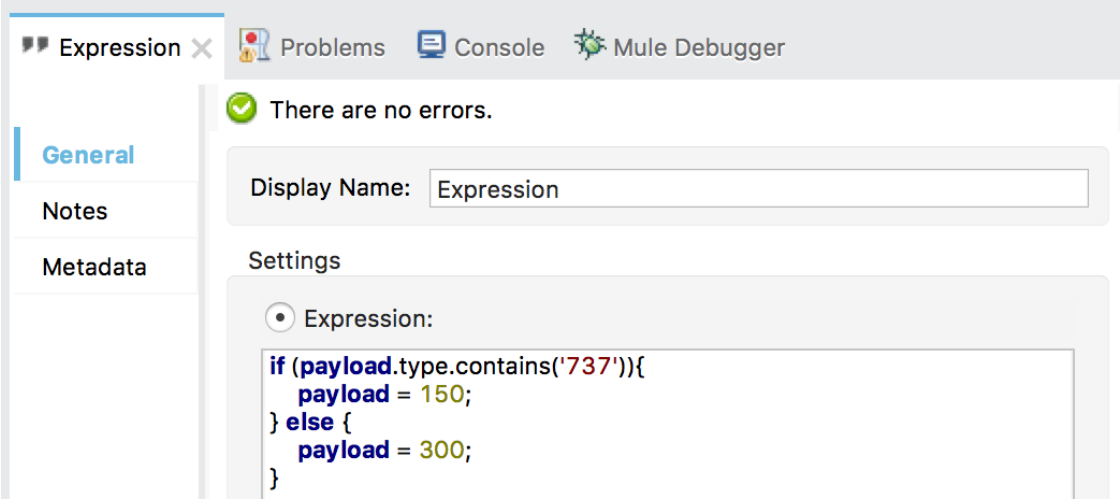


26. Return to global.xml and copy the if/else block.
27. Return to implementation.xml.
28. In the Expression Properties view, paste the if/else block you copied into the expression text area.
29. In the expression, change type to payload.type.

```
if (payload.type.contains('737')){
```

30. Change the two return statements to set the payload with the values instead.

```
if (payload.type.contains('737')){
      payload = 150;
} else {
      payload = 300;
}
```



Call a flow from a DataWeave expression

31. Return to the Transform Message Properties view of the component in postMultipleFlightsFlow.
32. Comment out the existing totalSeats assignment by placing // in front of it.

```
//totalSeats: getNumSeats($.planeType)
```

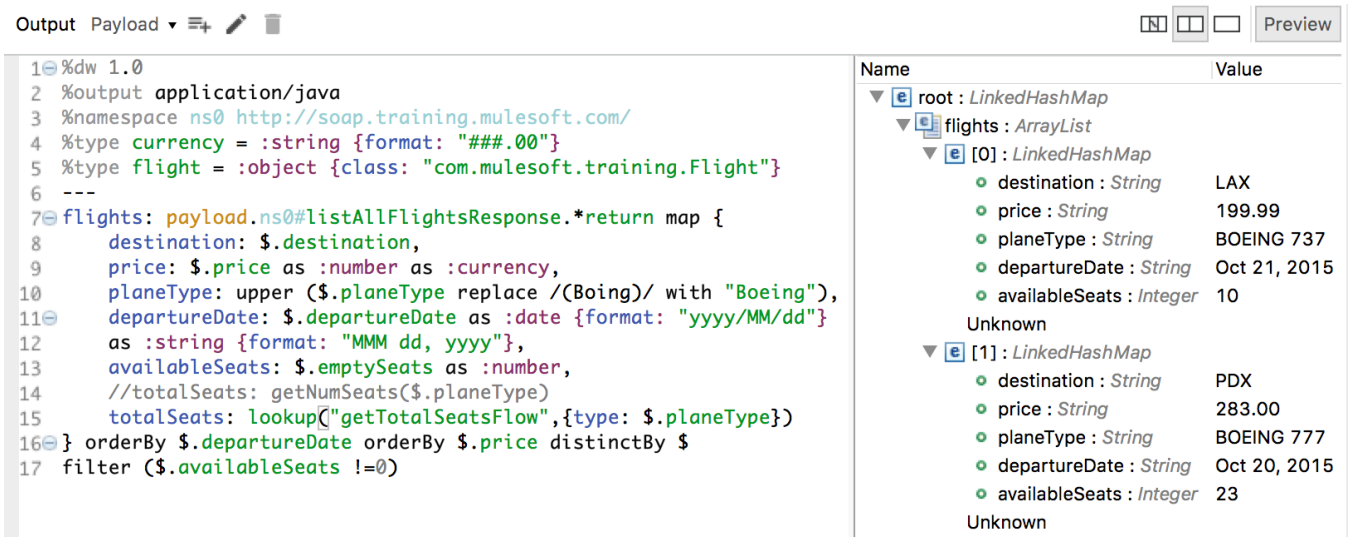
33. Add another property called totalSeats inside the transformation function.
34. Set totalSeats equal to the return value from the DataWeave lookup() function.

```
totalSeats: lookup()
```

35. Pass to lookup() an argument that is equal to the name of the flow to call: "getTotalSeatsFlow".
36. Pass an object to lookup() as the second argument.
37. Give the object a field called type (to match what the flow is expecting) and set it equal to the value of the planeType field.

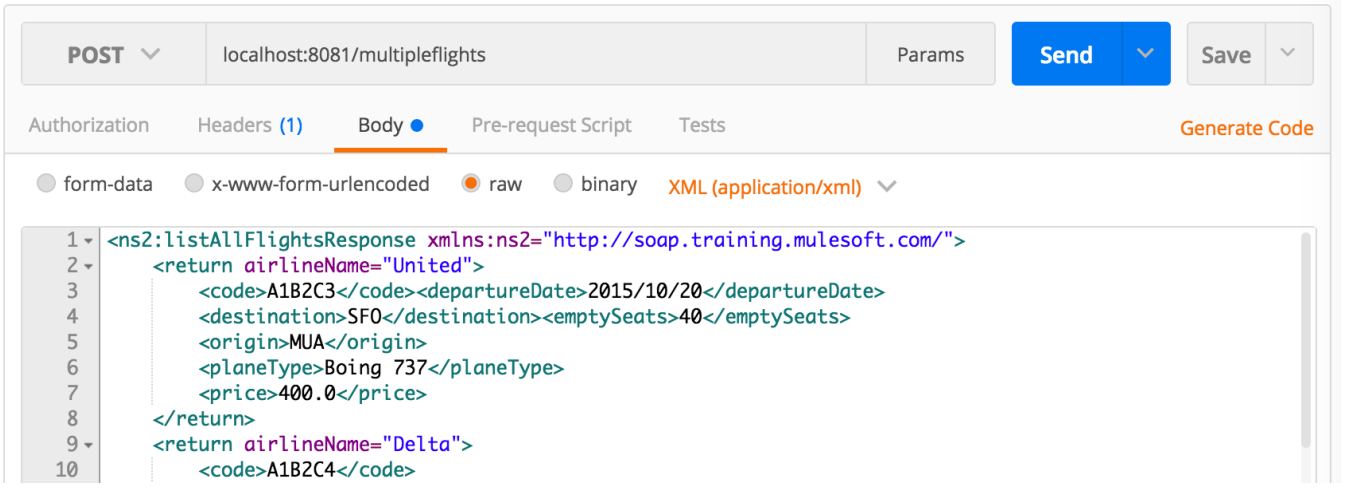
```
totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
```

38. Look at the preview; the value for totalSeats can only be assigned by calling the flow at runtime.

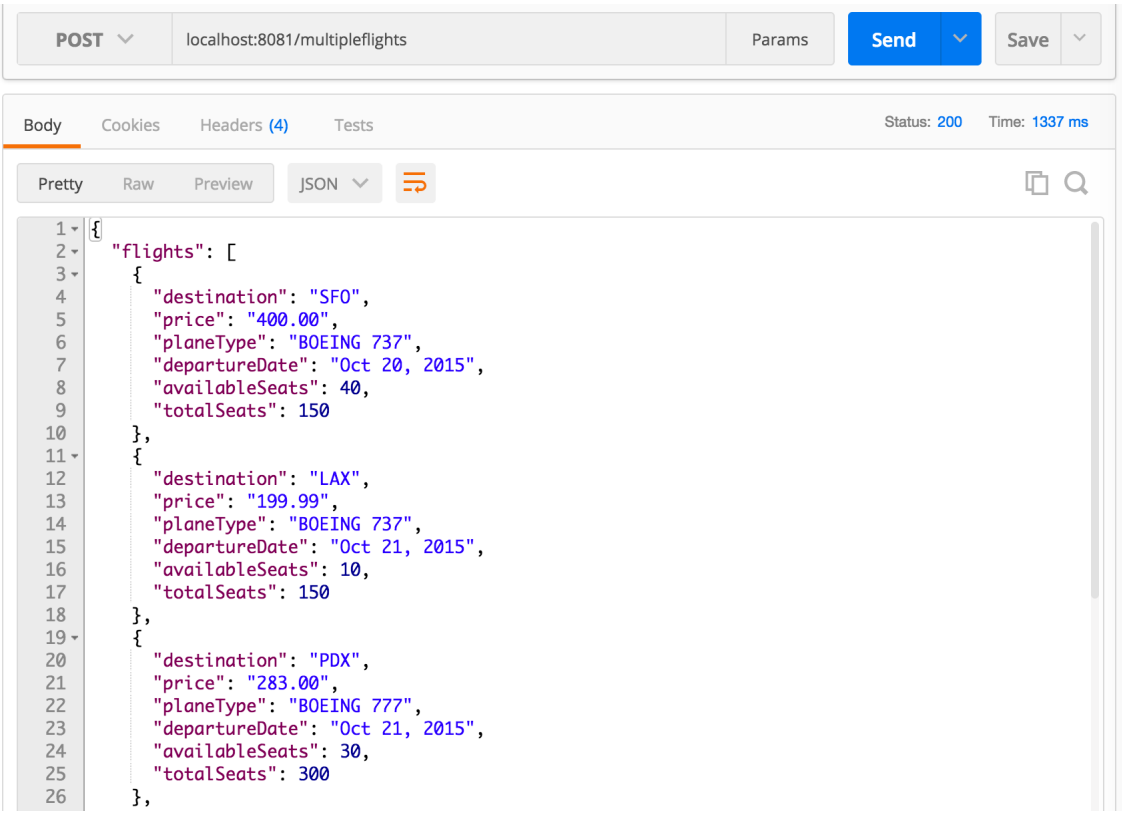


Test the application

39. Change the output type to application/json; you should see totalSeats has a value of null.
40. Run the project.
41. In Postman, make sure the method is set to POST and the request URL is set to localhost:8081/multipleflights.
42. Set a Content-Type header to application/xml.
43. Set the request body to the value contained in the flights-example.xml file in the src/test/resources folder.



44. Send the request; you should get JSON flight data returned and each flight should have a totalSeats property equal to 150 or 300.



45. Stop the project.
46. Close the project.

Module 12: Connecting to Additional Resources

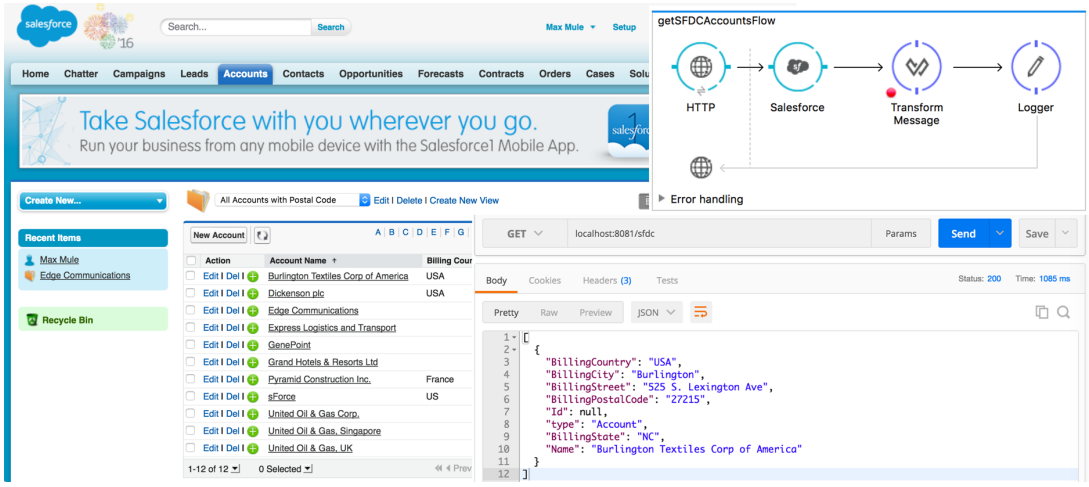
At the end of this module, you should be able to:

- Connect to SaaS applications.
- Connect to files.
- Poll resources.
- Connect to JMS queues.
- Discover and install connectors not bundled with Anypoint Studio.

Walkthrough 12-1: Connect to a SaaS application (Salesforce)

In this walkthrough, you retrieve account records from Salesforce. You will:

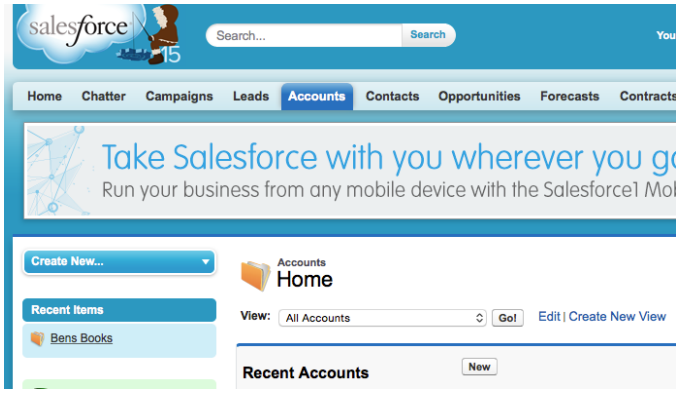
- Browse Salesforce data on http://salesforce.com.
- Add and configure a Salesforce connector.
- Add a Salesforce endpoint to retrieve accounts for a specific postal code.
- Use the Query Builder to write a query.



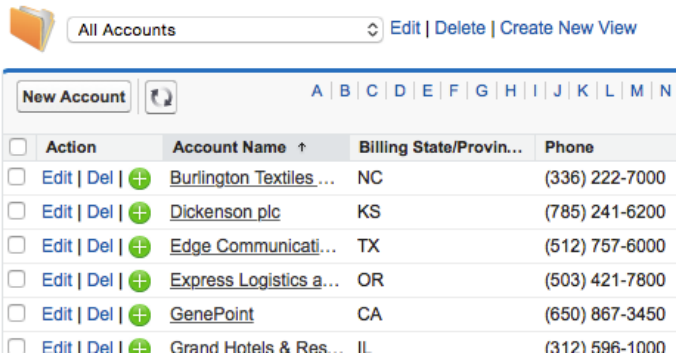
Note: To complete this walkthrough, you need a Salesforce Developer account. Instructions for creating a Salesforce developer account and getting an API access token are included in the first walkthrough at the beginning of this student manual.

Look at existing Salesforce account data

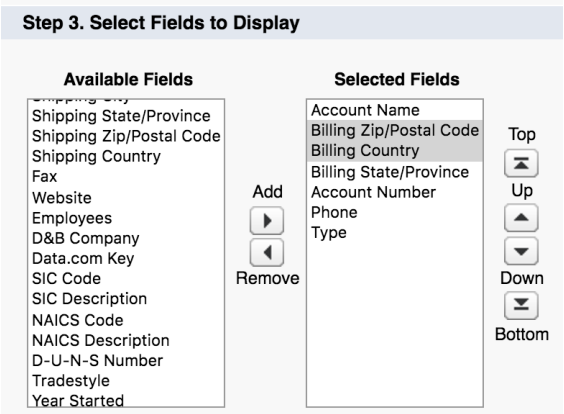
1. In a web browser, navigate to http://login.salesforce.com/ and log in with your Salesforce Developer account.
2. Click the Accounts link in the main menu bar.
3. In the view drop-down menu, select All Accounts and click the Go button.



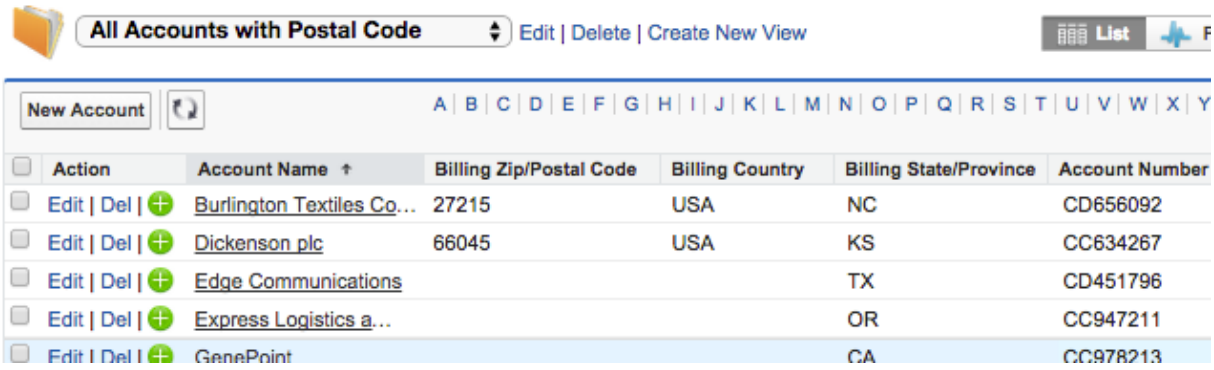
4. Look at the existing account data; a Salesforce Developer account is populated with some sample data.



5. Notice that countries and postal codes are not displayed by default.
6. Click the Create New View link next to the drop-down menu displaying All Accounts.
7. Set the view name to All Accounts with Postal Code.
8. Locate the Select Fields to Display section.
9. Select Billing Zip/Postal Code as the available field and click the Add button.
10. Add the Billing Country field.
11. Use the Up and Down buttons to order the fields as you prefer.

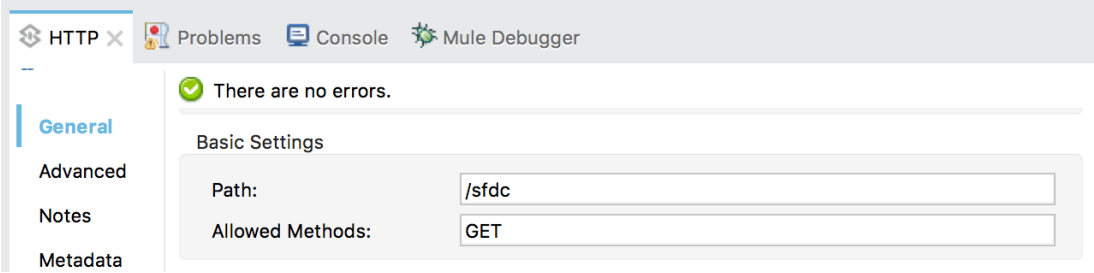


12. Click the Save button; you should now see all the accounts with postal codes and countries.



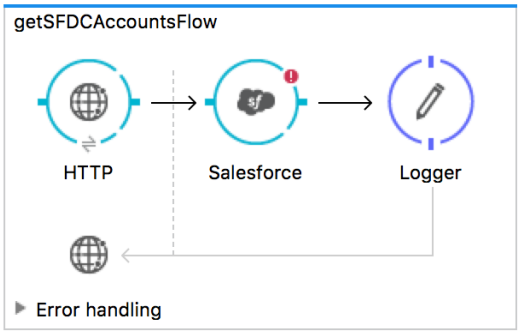
Review the starting flow in the examples project

13. Return to Anypoint Studio.
14. Open the apdev-examples project.
15. Open accounts.xml and review the HTTP Listener endpoint.

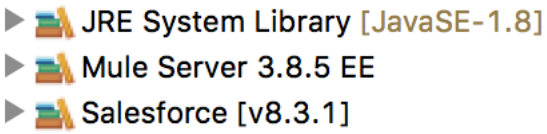


Add a Salesforce endpoint

16. Drag out a Salesforce connector and drop it before the Logger.

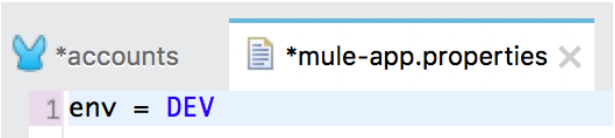


17. In the Package Explorer, locate the new Salesforce library that was added.



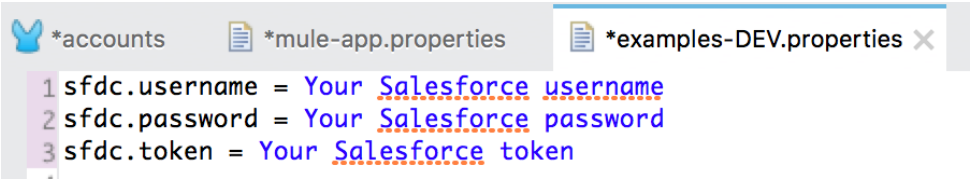
Configure the Salesforce connector

18. Open mule-app.properties in src/main/app.
19. Set an environment variable called env equal to DEV.



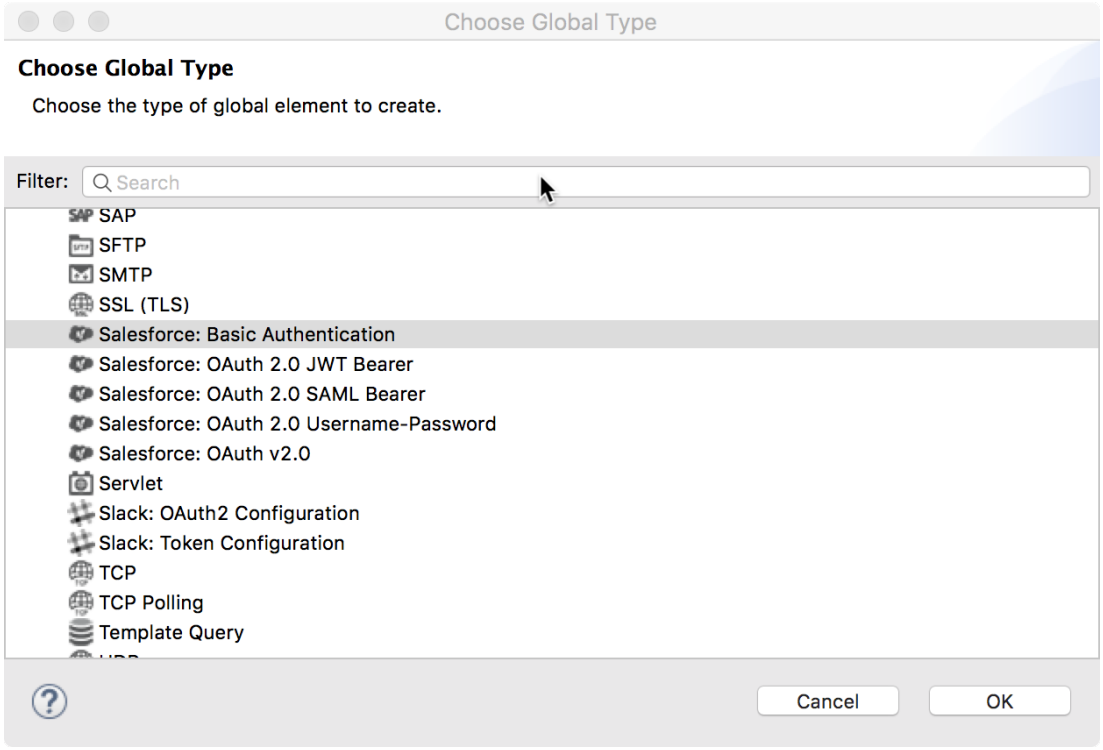
20. In the Package Explorer, right-click src/main/resources and select New > File.

21. In the new File dialog box, set the file name to examples-DEV.properties and click Finish.
22. In examples-DEV.properties, create properties for your Salesforce username, password, and security token.

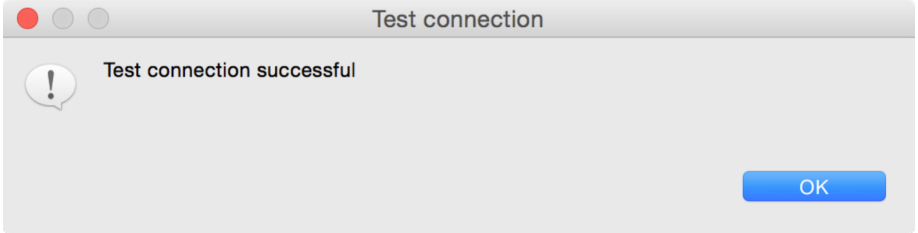


Note: Instructions for creating a Salesforce Developer account and getting a security token are included in the first walkthrough at the beginning of this student manual.

23. Save all the files.
24. Return to the Global Elements view of global.xml.
25. Click Create.
26. In the Choose Global Type dialog box, select Component configurations > Property Placeholder and click OK.
27. Set the location to examples-${env}.properties and click OK.
28. Click Create.
29. In the Choose Global Type dialog box, select Connector Configuration > Salesforce: Basic Authentication and click OK.



30. In the Global Element Properties dialog box, set the following values and click OK.
    - Username: ${sfdc.username}
    - Password: ${sfdc.password}
    - Security Token: ${sfdc.token}
31. Click the Validate configuration button; you will get a Test Connection dialog box letting you know if the connection succeeds or fails.



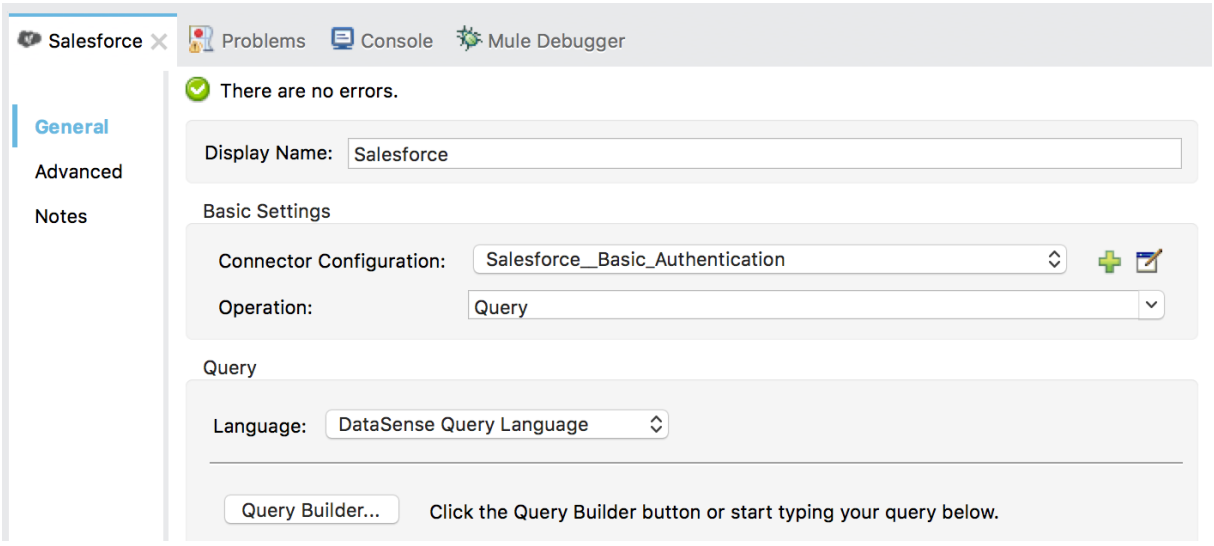
32. Click OK to close the Test connection dialog box.
33. If your test connection failed, fix it; do not proceed until it is working.

    Note: If it failed, check to see if you have any extra whitespace in your entries.

34. Click OK to close the Global Elements Properties dialog box.

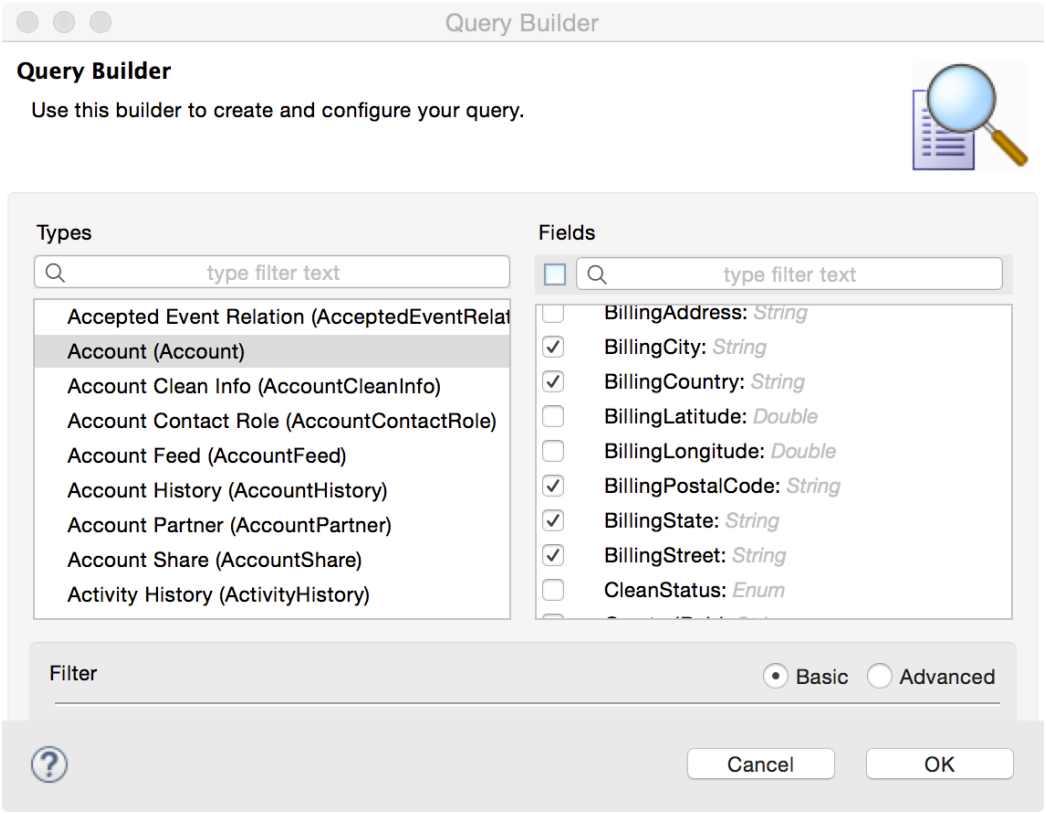
Configure the Salesforce endpoint

35. Return to accounts.xml.
36. In the Salesforce properties view, set the connector configuration to the existing Salesforce__Basic_Authentication.
37. Set the operation to Query.

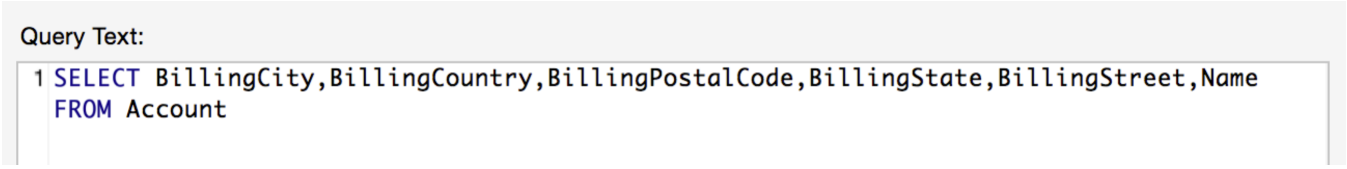


Write the query

38. Click the Query Builder button.
39. In the Query Builder dialog box, select a type of Account (Account).
40. Select fields BillingCity, BillingCountry, BillingPostalCode, BillingState, BillingStreet, and Name.



41. Click OK.
42. In the Salesforce properties view, examine the generated query.



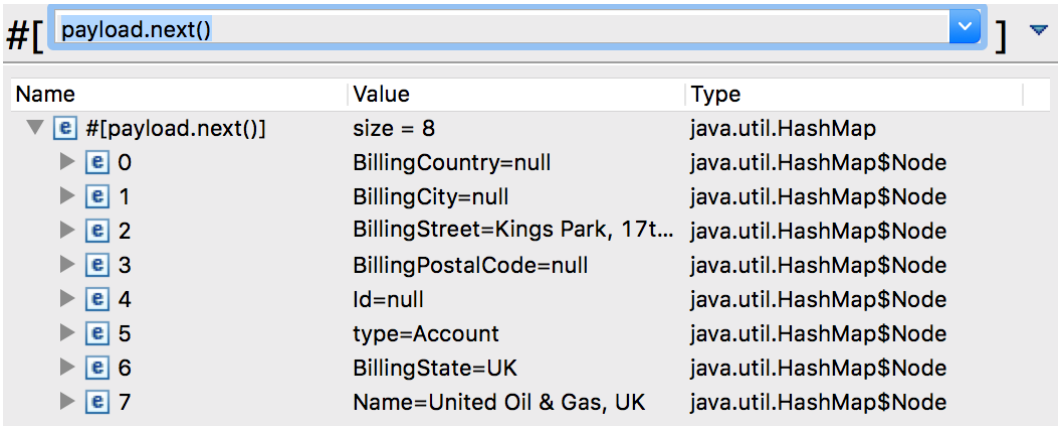
Debug the application

43. Add a breakpoint to the Logger.
44. Debug the project.
45. In Postman, make a request to http://localhost:8081/sfdc.
46. In the Mule Debugger, drill-down into the payload variable.
47. Click the Evaluate Mule Expression button.

48. Enter the following expression and press the Enter key.

```
#[payload.next()]
```

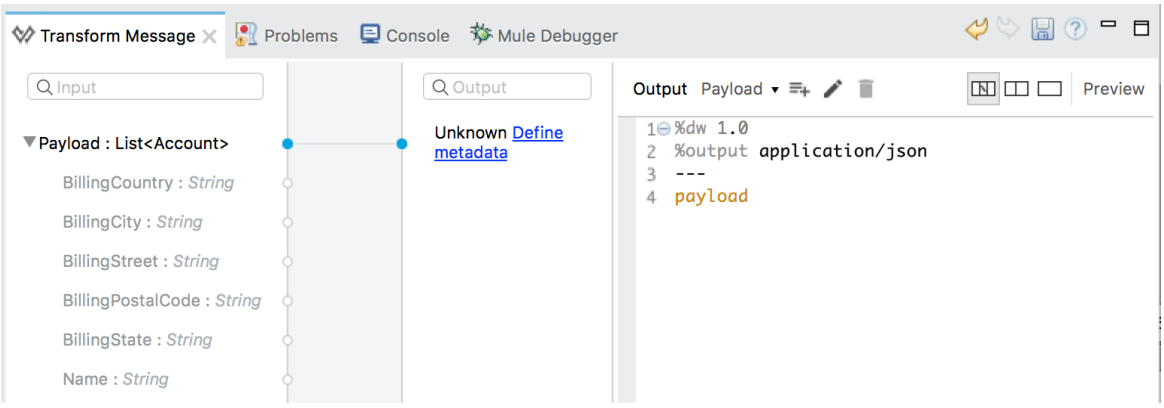
49. Expand the results; you should see the account data for the first account.



50. Click anywhere outside the expression window to close it.
51. Stop the project and switch perspectives.

Display the return data

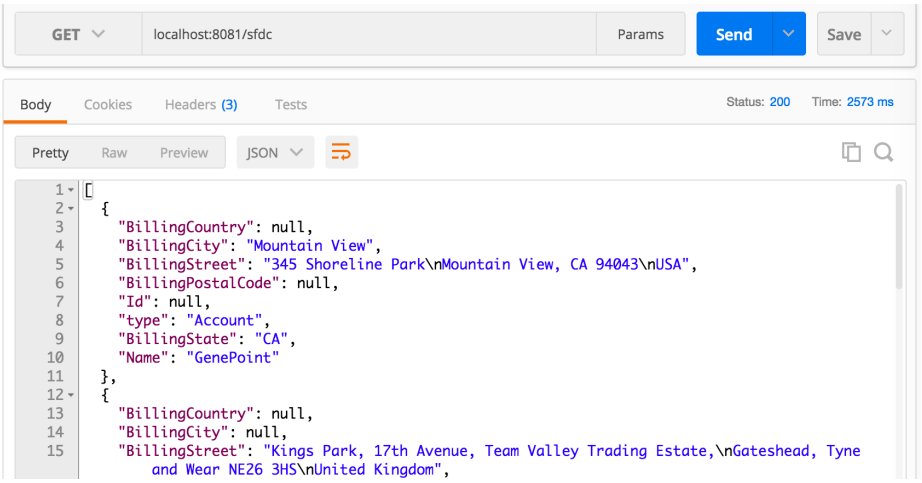
52. Add a Transform Message component before the Logger component.
53. In the Transform Message properties view, look at the input metadata.
54. In the output expression section, change the output type to application/json.
55. Set the DataWeave expression to payload.



Test the application

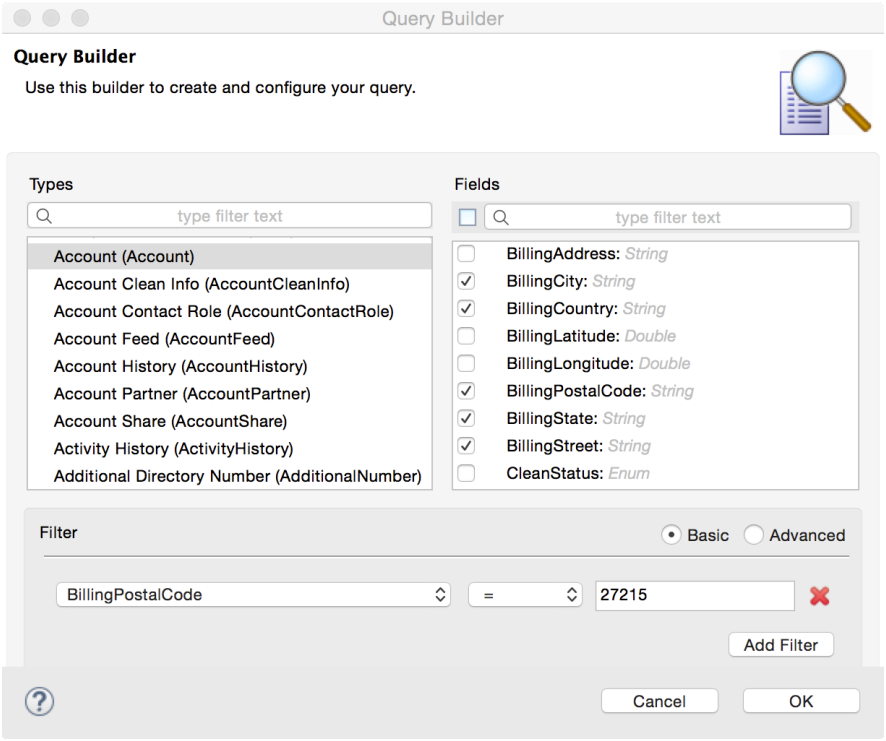
56. Run the project.

57. In Postman, make another request to http://localhost:8081/sfdc; you should see the Salesforce accounts displayed.



Modify the query

58. Return to accounts.xml.
59. In the Salesforce Properties view, click the Query Builder button.
60. In the Query Builder dialog box, click the Add Filter button.
61. Create a filter for BillingPostalCode equal to one of the postal codes (like 27215) in your Salesforce account data.



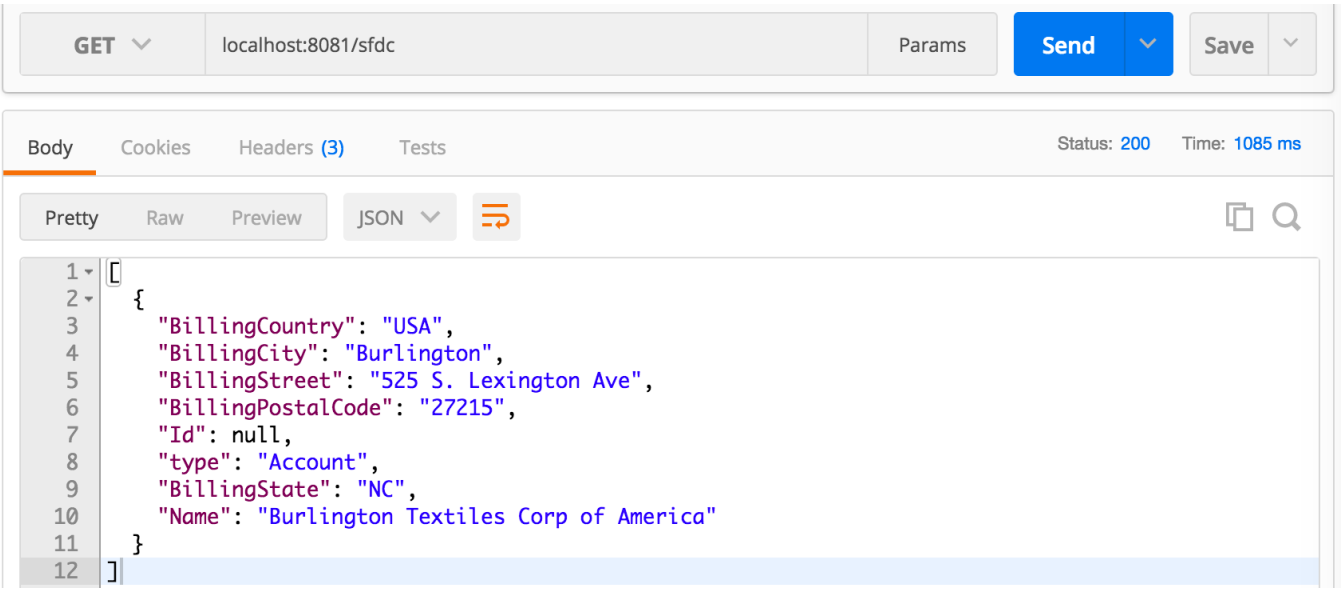
62. Click OK.
63. Examine the generated query.

Query Text:

```
SELECT BillingCity,BillingCountry,BillingPostalCode,BillingState,BillingStreet,Name
FROM Account
WHERE BillingPostalCode = '27215'
```

Test the application

64. Save the file to redeploy the application.
65. In Postman, make another request to http://localhost:8081/sfdc; you should see only the accounts with the specified postal code displayed.



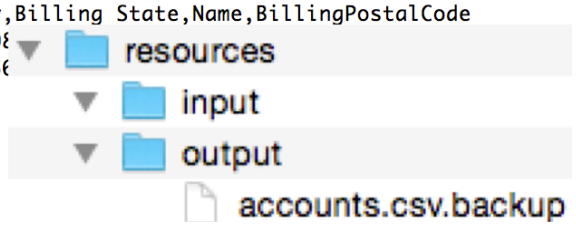
66. Return to Anypoint Studio and stop the project.

Walkthrough 12-2: Connect to a file (CSV)

In this walkthrough, you load data from a local CSV file. You will:

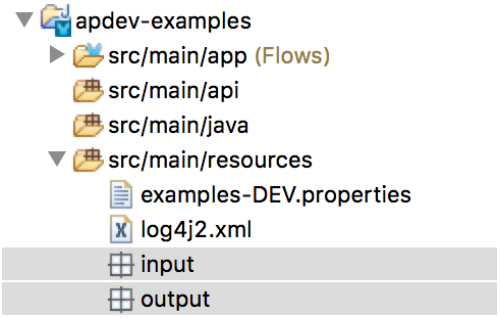
- Add and configure a File endpoint to watch an input directory, read the contents of any added files, and then rename and move the files.
- Use DataWeave to convert a CSV file to a string.
- Add a CSV file to the input directory and watch it renamed and moved.
- Restrict the type of file read.
- Add metadata to a file endpoint.

```
INFO  2015-08-19 13:44:16,688 [[apessentials].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Logger
MessageProcessor: Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode
111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008
400 South St,San Francisco,USA,CA,Iguana Park Industries,91156
777 North St,San Francisco,USA,CA,Cat Park Industries,91156
```



Create new directories

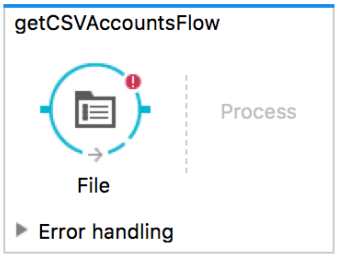
1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > Folder.
3. Set the folder name to input and click Finish.
4. Create a second folder called output.



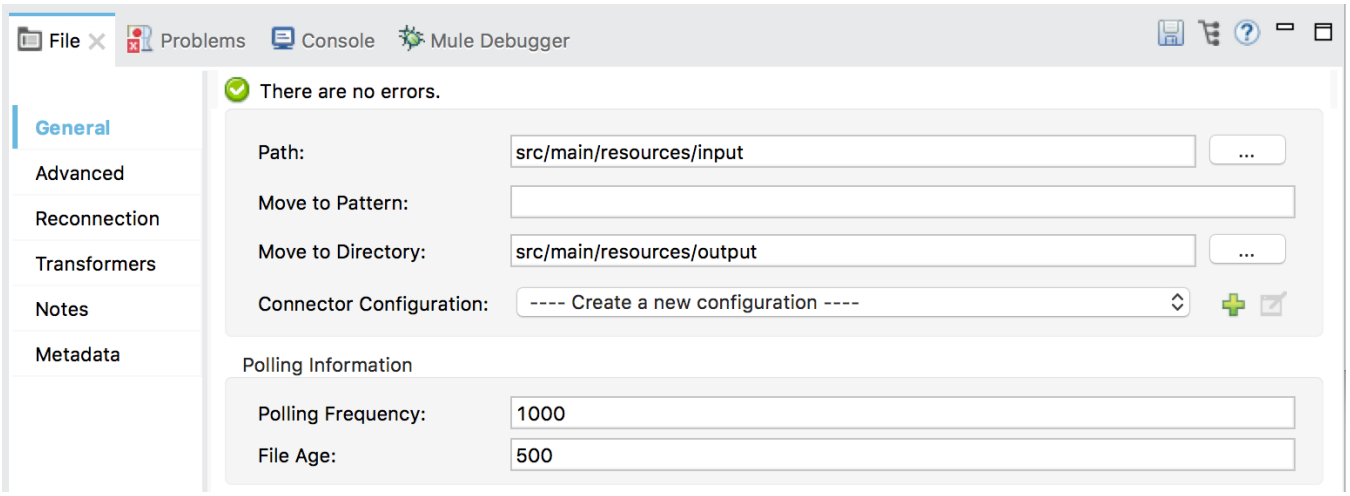
Add and configure a File endpoint

5. Return to accounts.xml.
6. Drag a File connector from the Mule Palette and drop it in the canvas to create a new flow.

7. Rename the flow to getCSVAccountsFlow.



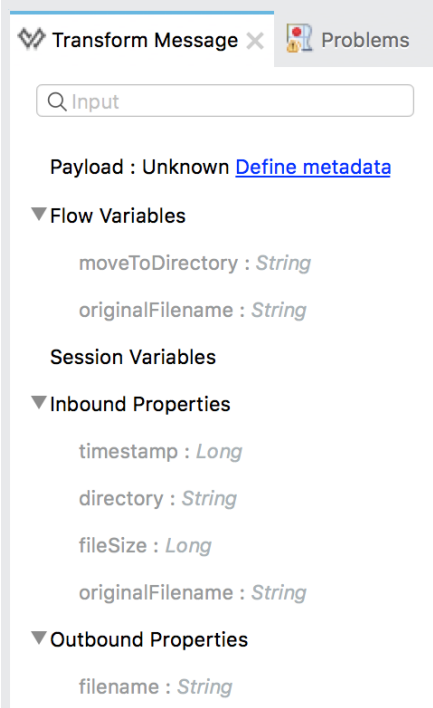
8. In the File properties view, set the path to src/main/resources/input.
9. Set the move to directory to src/main/resources/output.
10. Look at the default polling information; the endpoint checks for incoming messages every 1000 milliseconds and sets a minimum of 500 milliseconds a file must wait before it is processed.



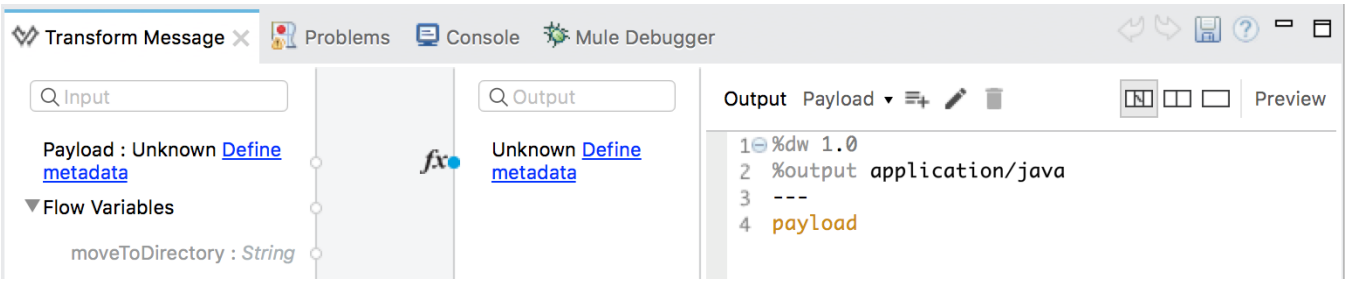
Display the file contents

11. Add a Transform Message component to the process section of the flow.

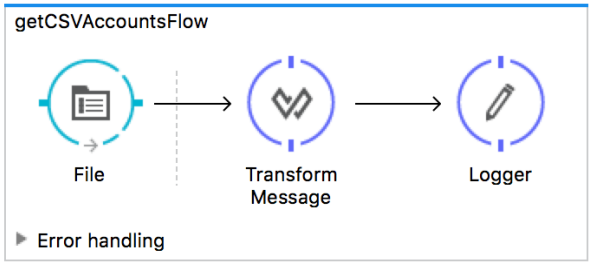
12. In the Transform Message properties view, look at the metadata for the input; there should be no metadata for the payload but there should be metadata for variables and properties.



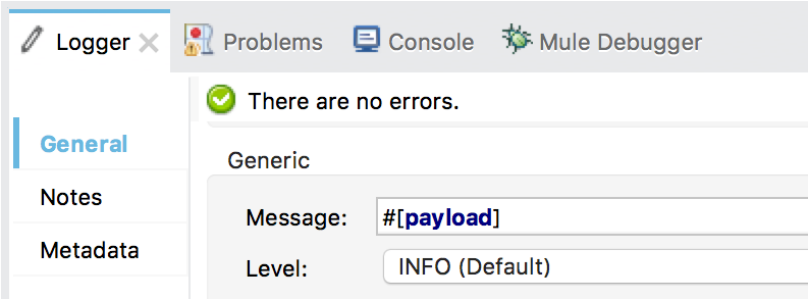
13. Set the transformation expression to payload.



14. Add a Logger after the component.



15. In the Logger properties view, set the message to display the payload.

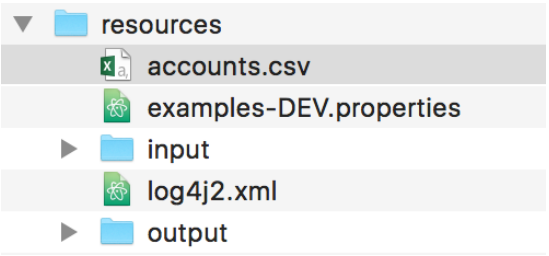


Run the application

16. Run the project.

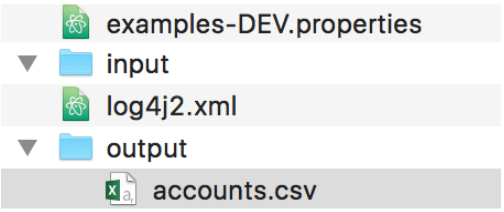
Add a CSV file to the input directory

17. Right-click src/main/resources in the Package Explorer and select Show In > System Explorer.
18. Open the resources folder.
19. In another file browser window, navigate to the student files for the course.
20. Locate the resources/accounts.csv file.
21. Copy this file and then paste it in the src/main/resources folder of the apdev-examples project.



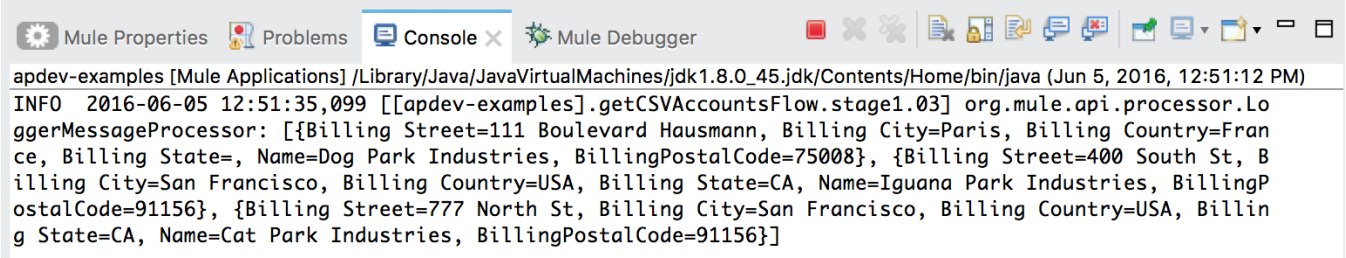
Test the application

22. Drag the accounts.csv file into the input folder; you should see it almost immediately moved to the output folder.



23. Drag it back into the input folder; it will be read again and then moved to the output folder.
24. Leave this folder open.

25. Return to Anypoint Studio and look at the console; you should see the file contents displayed.



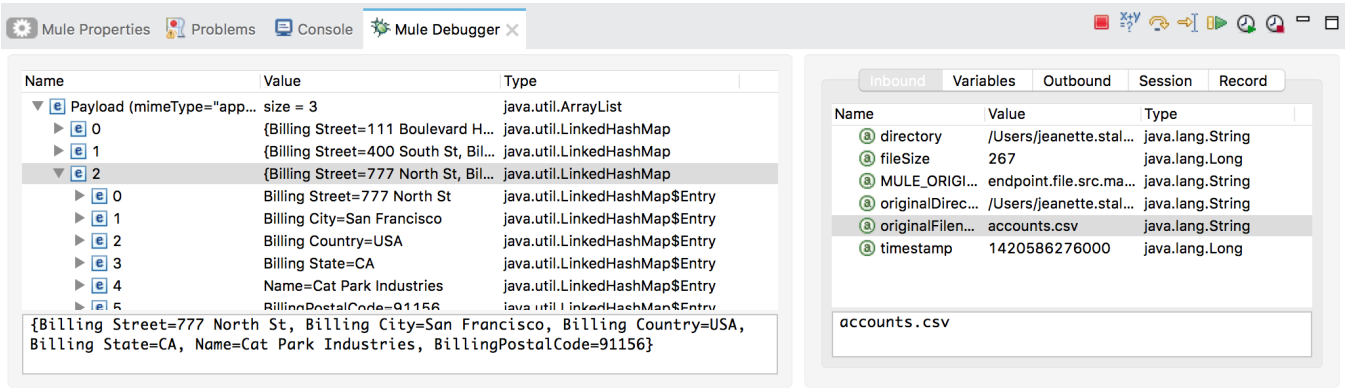
26. Stop the project.

Debug the application

27. In getCSVAccountsFlow, add a breakpoint to the Logger.
28. Debug the project.
29. Move the accounts.csv file from the output folder back to the input folder.

    Note: You can do this in your computer's file browser or in Anypoint Studio. If you use Anypoint Studio, you will need to right-click the project and select Refresh to see the file.

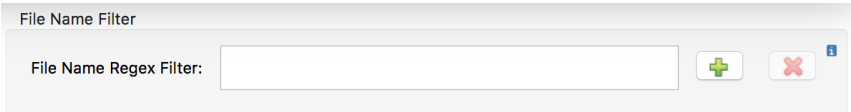
30. In the Mule Debugger, drill-down and look at the payload.
31. Look at the inbound message properties and locate the original filename.



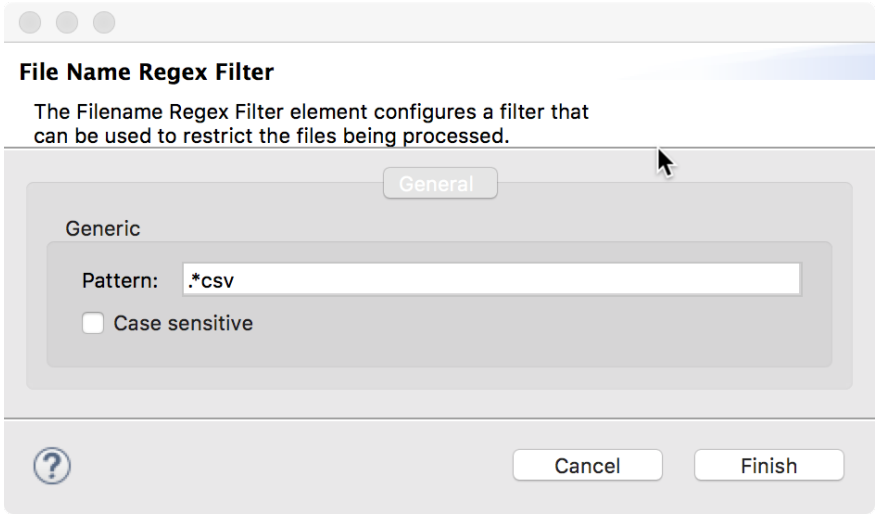
32. Click the Resume button.
33. Stop the project and switch perspectives.

Restrict the type of file read

34. In the File Properties view, click the Add button next to file name regex filter.



35. In the File Name Regex Filter dialog box, set the pattern to .*csv (not *.csv) and uncheck case sensitive.



36. Click Finish.

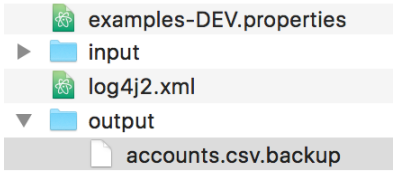
Rename the file

37. Set the move to pattern to append .backup to the original filename.

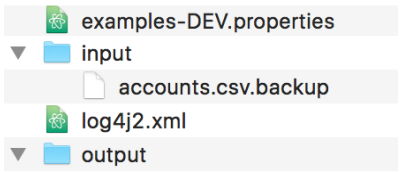
```
#[message.inboundProperties.originalFilename].backup
```

Test the application

38. Run the project.
39. Move the accounts.csv file from the output folder back to the input folder; you should see it appear in the output folder with its new name.



40. Move the accounts.csv.backup file from the output folder back to the input folder; it should not be processed and moved to the output folder.



41. Return to Anypoint Studio and stop the project.

Modify the File endpoint to not rename the files

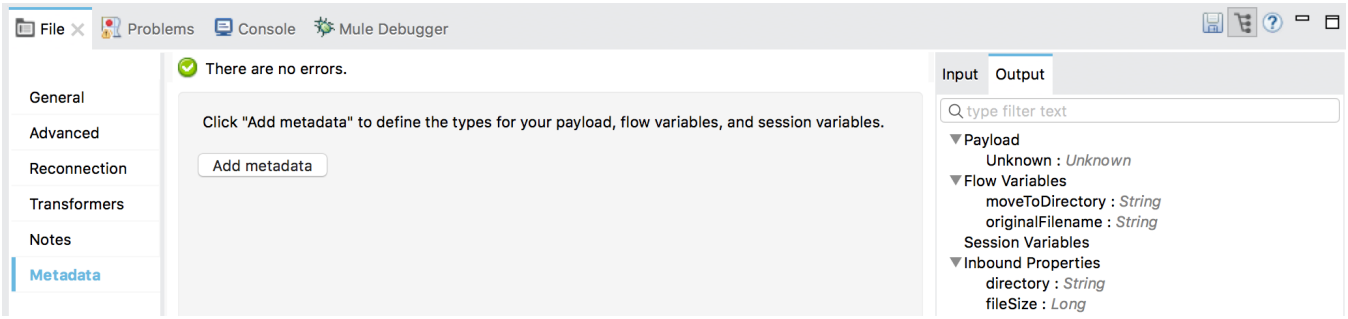
Note: In the last part of the walkthrough, you add metadata for the File endpoint. Although not needed here, this metadata will be used when the data is synchronized to Salesforce in the next module.

42. In the Package Explorer, right-click the src/main/resources/input folder and select Refresh.
43. Expand the src/main/resources/input folder.
44. Right-click accounts.csv.backup and select Refactor > Rename.
45. In the Rename Resource dialog box, set the new name to accounts.csv and click OK.
46. In the File Properties view, delete the move to pattern.

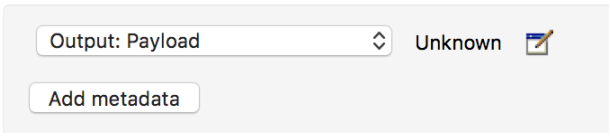
    Note: This will make it easier to test the application because you won't have to keep renaming the file as you move it back to the input directory.

Add file endpoint metadata

47. In the File Properties view, click the output tab in the metadata explorer and review the metadata.
48. Click Metadata in the left-side navigation.
49. Click the Add metadata button.

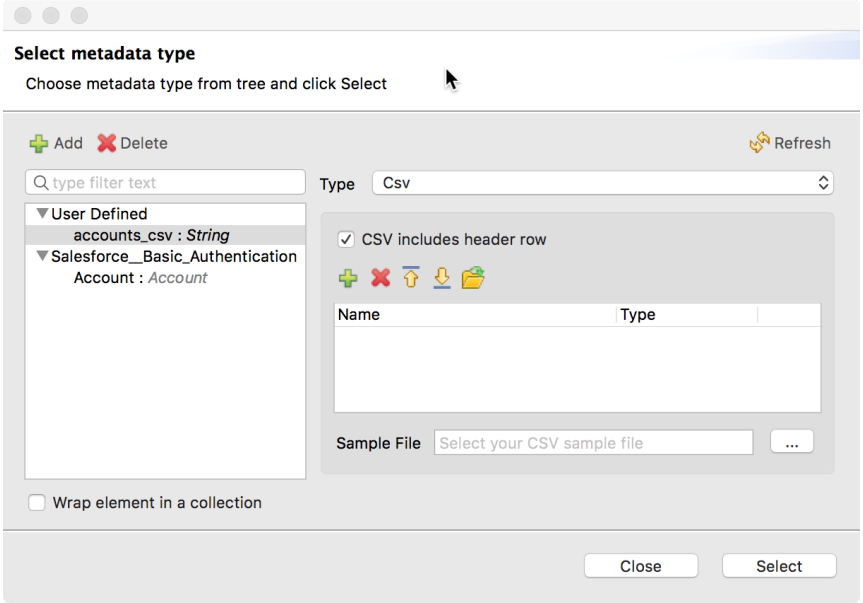


50. In the drop-down menu that appears, make sure Output: Payload is selected.
51. Click the Edit button next to the drop-down menu.

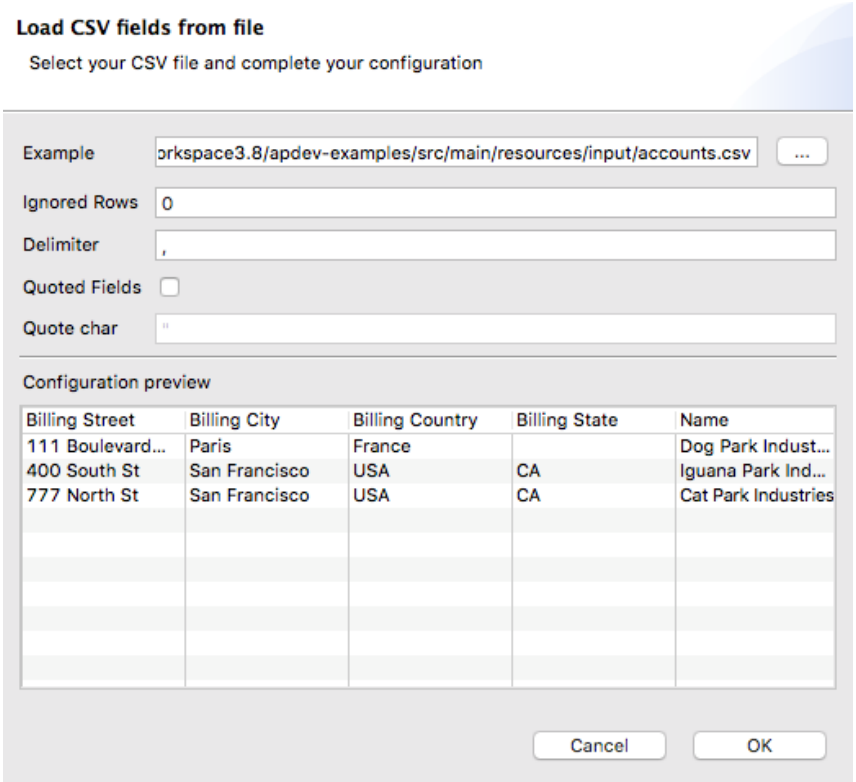


52. In the Select metadata type dialog box, click the Add button.
53. In the Create new type dialog box, set the name to accounts_csv and click Create type.

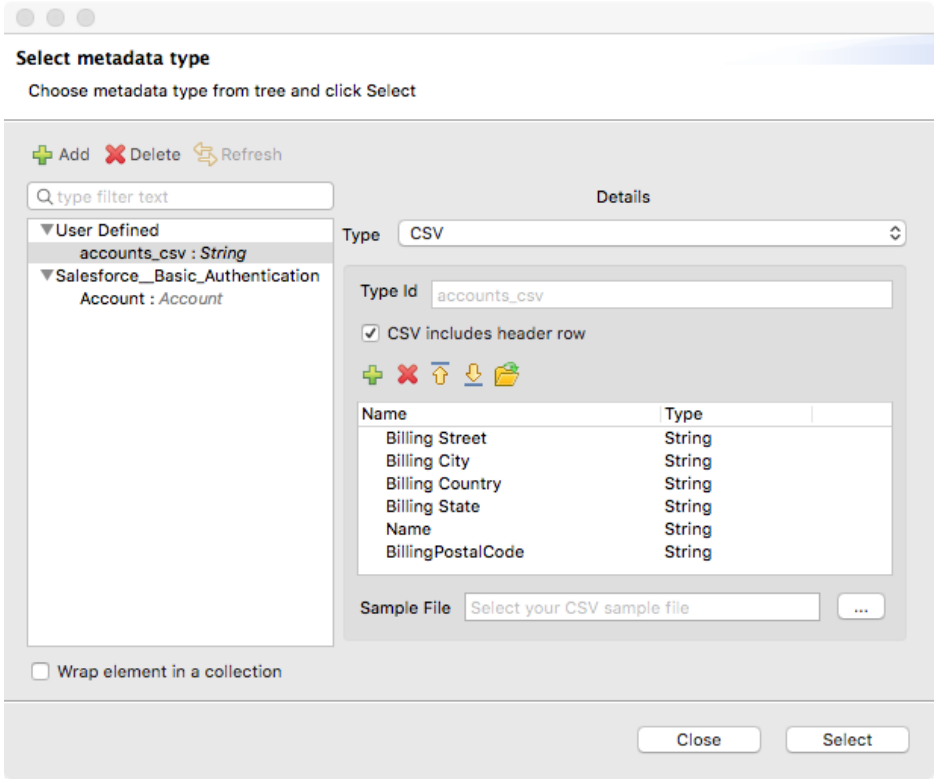
54. In the Select metadata dialog box, change the type to CSV.
55. Make sure CSV includes header row is checked.



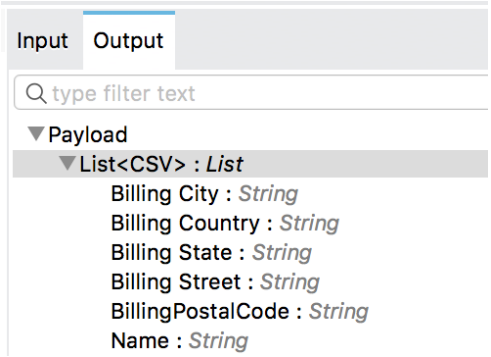
56. Click the Load from example button (the folder icon).
57. In the Load CSV fields from file dialog box, click the Browse button next to Example.
58. Browse to the project's src/main/resources/input folder, select accounts.csv and click Open.
59. In the Load CSV fields from file dialog box, click OK.



60. In the Select metadata type dialog box, click Select.



61. In the File Properties view, click the Output tab of the metadata explorer again; the payload should now have associated metadata and be shown to be a List <CSV>.

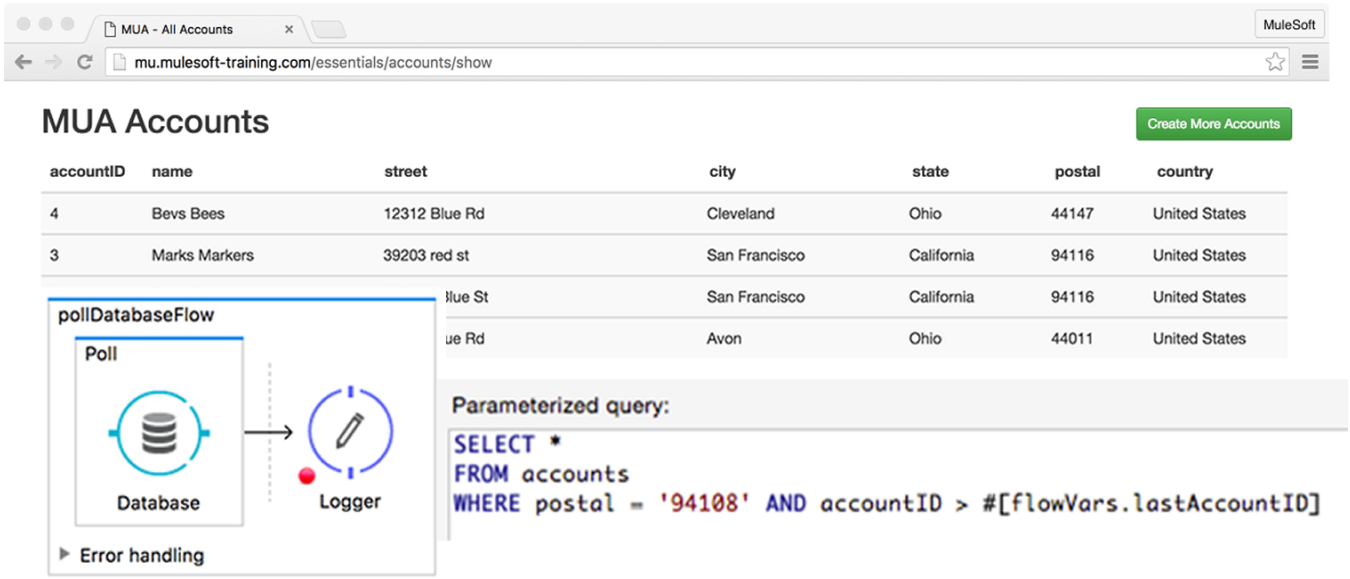


62. In the Transform Message Properties view, look at the payload metadata in the input section; the payload should now have associated metadata.
63. Save the file.

Walkthrough 12-3: Poll a resource

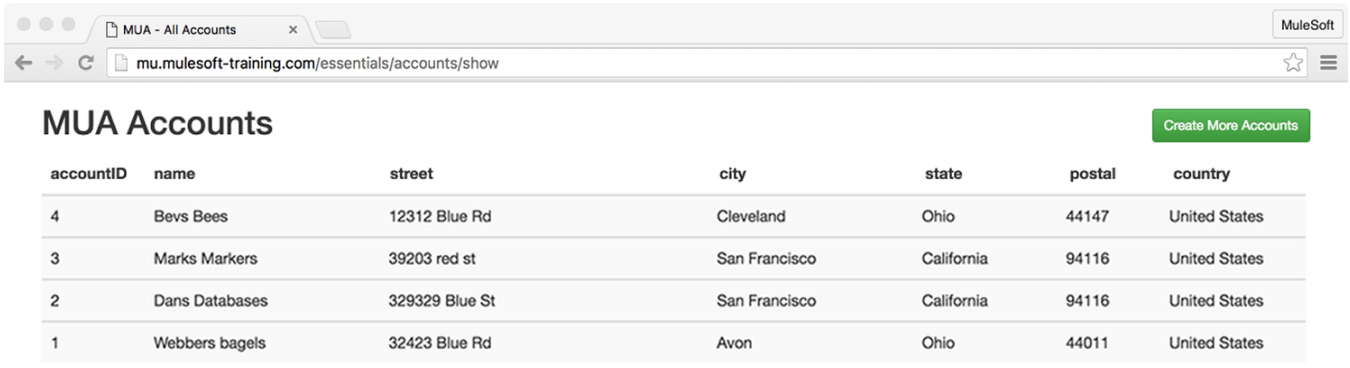
In this walkthrough, you poll a database. You will:

- Use a form to add accounts for a specific postal code to an accounts table in a database.
- Create a flow with a Poll scope as the message source.
- Poll a database every 5 seconds for records with a specific postal code.
- Use a poll watermark to track the ID of the latest record retrieved.
- Use the watermark to only retrieve new records with that postal code from the database.



Get familiar with the data in the database

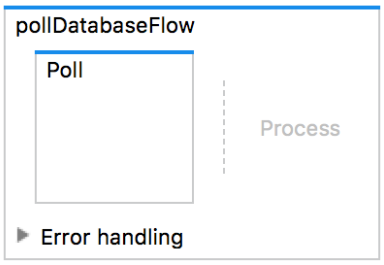
1. Return to the course snippets.txt file and copy the Account list URL for the MySQL database.
2. In a web browser, navigate to the URL you copied.
3. Look at the existing data and the name of the columns (which match the names of the database fields) and the postal values.



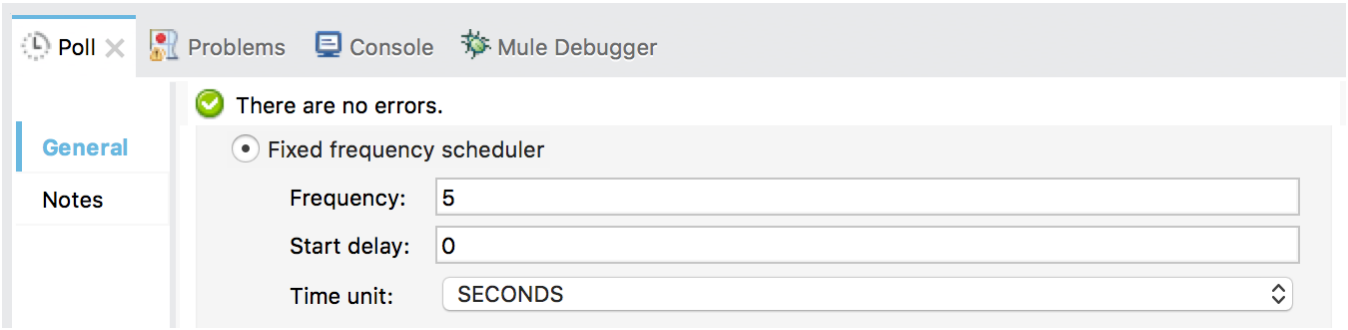
4. Click the Create More Accounts button.
5. Add a new record with a specific postal code; you will retrieve this record in this walkthrough and insert it into your Salesforce account in the next module.

Create a flow that polls a database

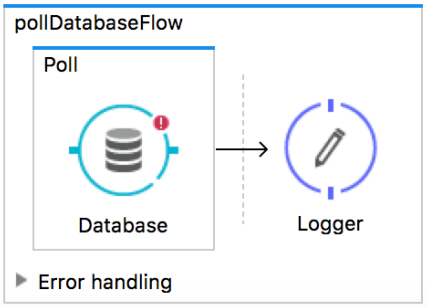
6. Return to accounts.xml in Anypoint Studio.
7. Drag a Poll scope from the Mule Palette and drop it at the top of the canvas.
8. Change the name of the flow to pollDatabaseFlow.



9. In the Poll Properties view, select the fixed frequency scheduler.
10. Set the frequency to 5 and the time unit to seconds.



11. Drag a Database connector from the Mule Palette to the poll scope.
12. Add a Logger to the process section of the flow.

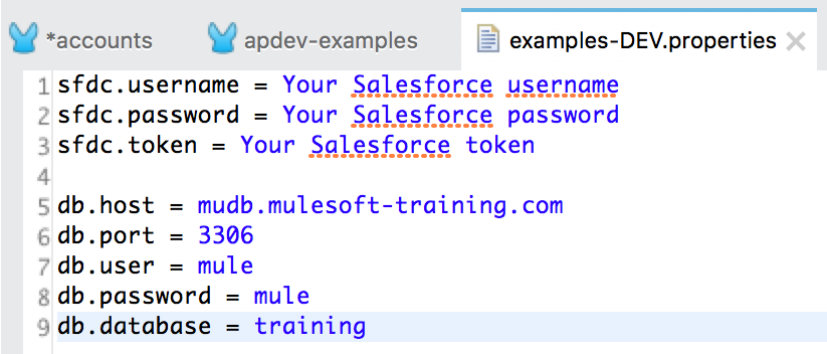


13. In the Logger properties view, set the message to display the payload.

    Note: Instead of polling the database directly, you could use the poll scope to make periodic requests to an API that governs access to the database.

Configure the Database connector

14. Return to the course snippets.txt file and copy the database parameters (the five starting with db).
15. Return to examples-DEV.properties and paste the values.



16. Save the file.
17. Return to the Global Elements view of global.xml.
18. Click Create.
19. In the Choose Global Type dialog box, select Connector Configuration > MySQL Configuration and click OK.
20. In the Global Element Properties dialog box, set the following values and click OK.

    - Host: ${db.host}
    - Port: ${db.port}
    - User: ${db.user}
    - Password: ${db.password}
    - Database: ${db.database}

21. Under Required dependencies, click the Add File button next to MySQL Driver.
22. Navigate to the student files folder, select the MySQL JAR file located in the resources folder, and click Open.
23. Back in the Global Element Properties dialog box, click the Test Connection button; you should get a successful test dialog box.

    Note: Make sure the connection succeeds before proceeding.

24. Click OK to close the dialog box.
25. Click OK to close the Global Element Properties dialog box.

Configure the Database endpoint

26. Return to accounts.xml.
27. In the Database properties view, set the connector configuration to the existing MySQL_Configuration.
28. Set the operation to Select.
29. Add a query to select the data for the postal code of the record you added to the accounts table.

```
SELECT *
FROM accounts
WHERE postal = '94108'
```

Test the application

30. Run the project.
31. Watch the console; you should see records displayed every 5 seconds.

```
INFO  2016-05-14 10:52:18,088 [[apdev-examples].pollDatabaseFlow.stage1.02] org.mule.api.processor.Logg
erMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF
, postal=94108}]
INFO  2016-05-14 10:52:27,845 [[apdev-examples].pollDatabaseFlow.stage1.02] org.mule.api.processor.Logg
erMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF
, postal=94108}]
INFO  2016-05-14 10:52:37,855 [[apdev-examples].pollDatabaseFlow.stage1.02] org.mule.api.processor.Logg
erMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF
, postal=94108}]
```

Note: Right now, all records with matching postal code are retrieved – over and over again. Next, you will modify this so only new records with the matching postal code are retrieved.

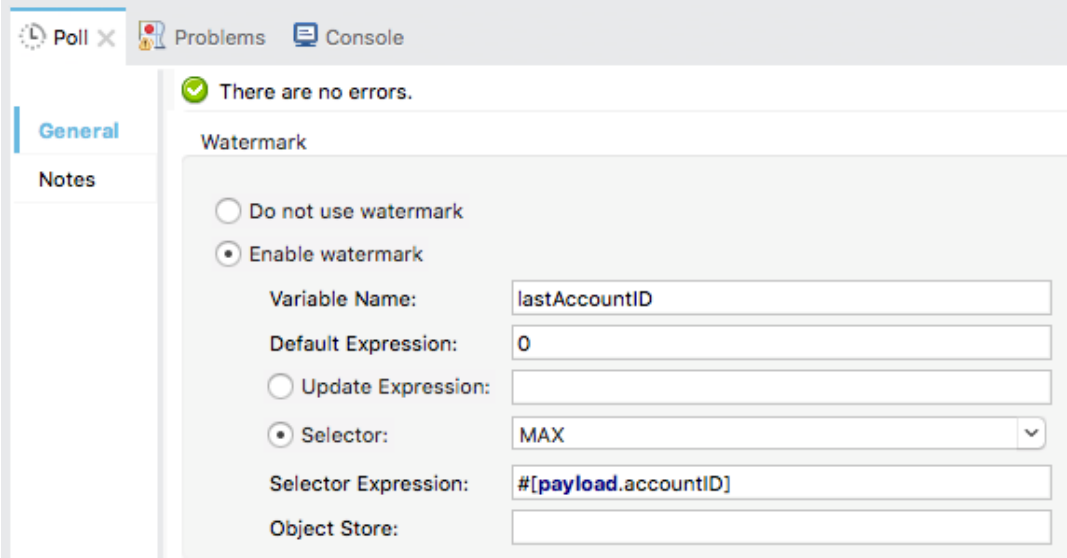
32. Stop the project.

Use a watermark to keep track of the last record retrieved

33. In the Poll properties view, select Enable watermark.

34. Set the watermark to store the max accountID returned by setting the following values.

    - Variable name: lastAccountID
    - Default expression: 0
    - Selector: MAX
    - Selector Expression: #[payload.accountID]



Run the application

35. Run the project; the application should deploy but then throw an exception.
36. Locate the exception in the console; you should see a message that watermarking requires synchronous polling.

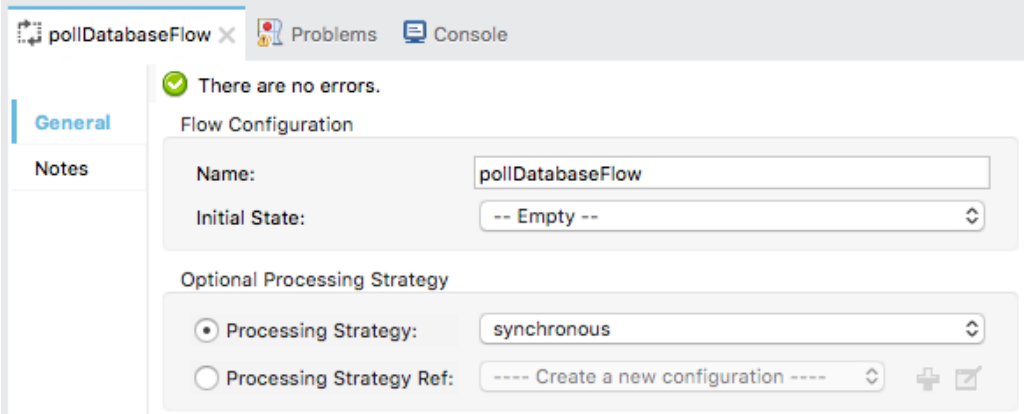
```
ERROR 2016-05-14 10:55:23,871 [pool-32-thread-1] org.mule.exception.DefaultSystemExceptionStrategy:
********************************************************************************
Message               : Watermarking requires synchronous polling
--------------------------------------------------------------------------------
Root Exception stack trace:
org.mule.api.config.ConfigurationException: Watermarking requires synchronous polling
        at org.mule.transport.polling.watermark.WatermarkPollingInterceptor.prepareRouting(WatermarkPollingInterceptor.java:53)
```

37. Stop the project.

Make the flow synchronous

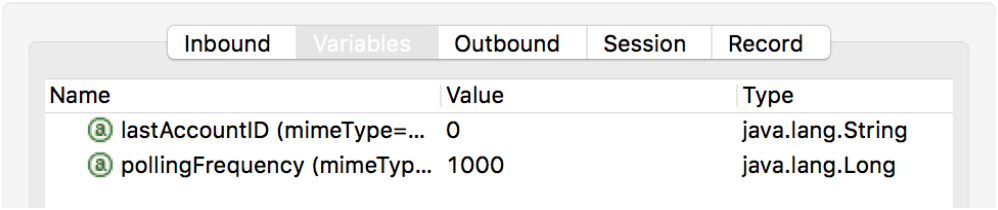
38. In the Properties view for pollDatabaseFlow, select processing strategy.

39. In the processing strategy drop-down menu, select synchronous.

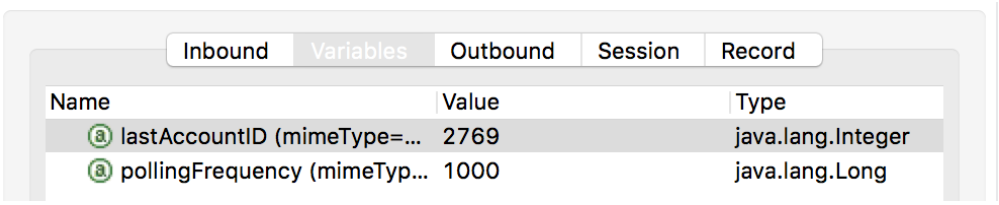


Debug the application and examine the watermark value

40. Place a breakpoint on the Logger.
41. Debug the project.
42. Wait until the breakpoint is hit.
43. Locate your watermark variable in the Variables section of the Mule Debugger view; initially, you should see a default value of zero.



44. Click the Resume button.
45. Look at the value of the watermark variable again; it should now be equal to the max accountID for training accounts records with the postal code you are using.



46. Resume the application multiple times; the same records should still be selected over and over again.
47. Stop the project and switch perspectives.

Modify the database query to use the watermark

48. In the Database properties view, modify the query so it only returns records for your postal code and with accountID values greater than the watermark lastAccountID value.

    ```
    WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
    ```

    Parameterized query:

    ```
    SELECT *
    FROM accounts
    WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
    ```

Test the application

49. Run the project.
50. Look at the console; you should see that no records are retrieved at all this time.

    ```
    INFO  2016-05-14 11:06:06,312 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
    INFO  2016-05-14 11:06:06,312 [pool-31-thread-1] org.mule.transport.polling.watermark.Watermark: Waterm
    ark value will not be updated since poll processor returned no results
    INFO  2016-05-14 11:06:16,039 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
    INFO  2016-05-14 11:06:16,039 [pool-31-thread-1] org.mule.transport.polling.watermark.Watermark: Waterm
    ark value will not be updated since poll processor returned no results
    ```

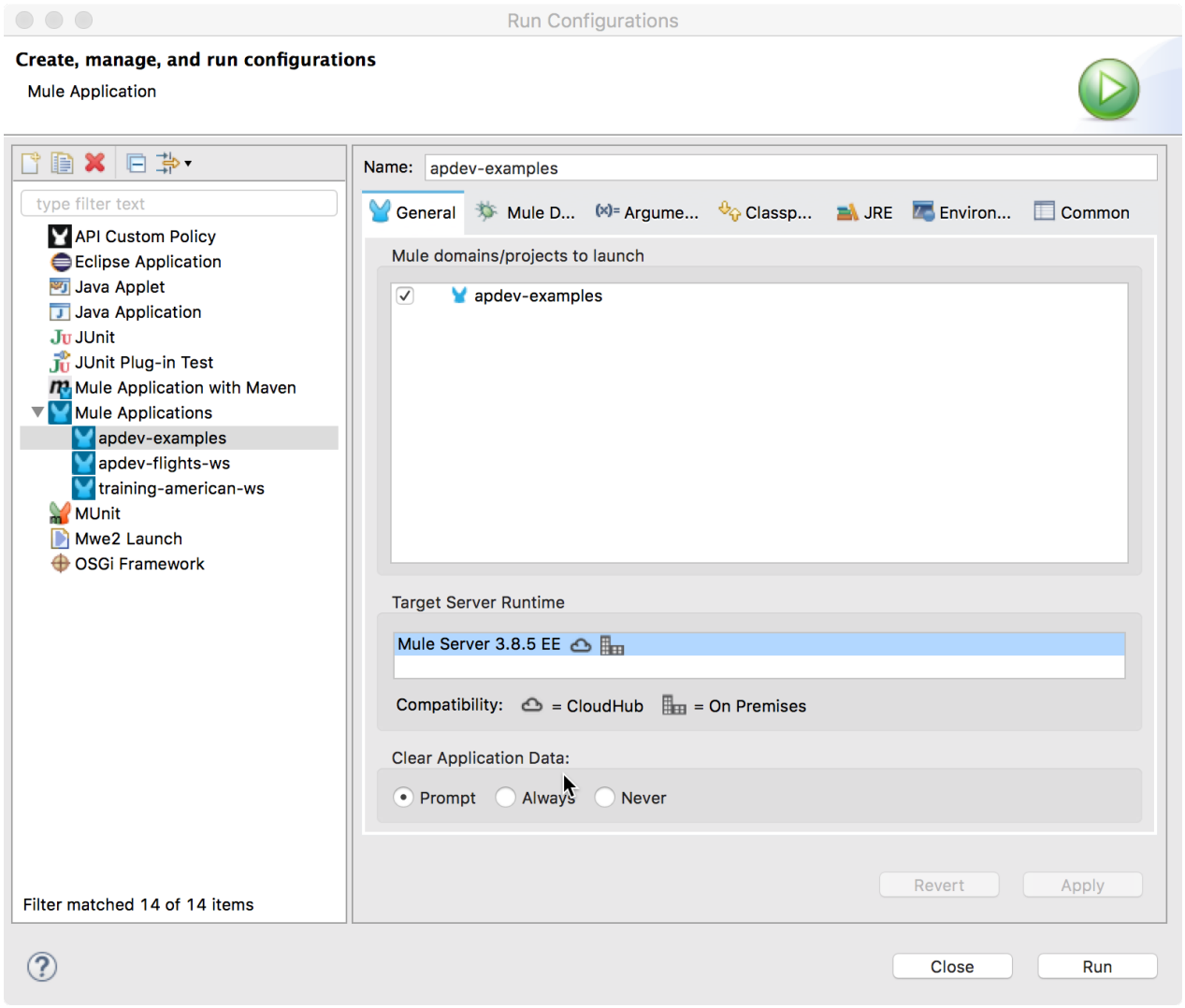
    Note: By default, the watermark is stored in a persistent object store so its value is retained between different executions of the application.

51. Stop the project.

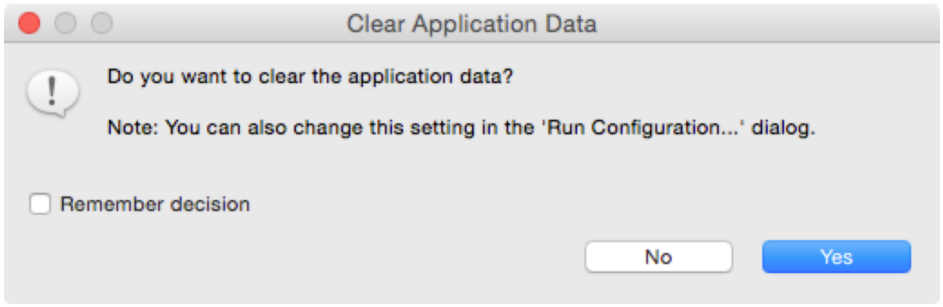
Clear application data

52. Select Run > Run Configurations.

53. Make sure your apdev-examples project is selected and then on the General tab, scroll down and change Clear Application Data from Never to Prompt.



54. Click Run; you should be prompted to clear the application data.
55. In the Clear Application Data dialog box, click Yes.



Test the application

56. Look at the console; you should see the latest matching records retrieved from the database again – but this time, only once.
57. Watch the console and see that all subsequent polling events retrieve no records.

```
INFO  2016-05-14 11:07:52,600 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: [{count
ry=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF, postal=94108}]
INFO  2016-05-14 11:08:02,318 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
INFO  2016-05-14 11:08:02,318 [pool-31-thread-1] org.mule.transport.polling.watermark.Watermark: Waterm
ark value will not be updated since poll processor returned no results
```

Add a new account with the same postal code

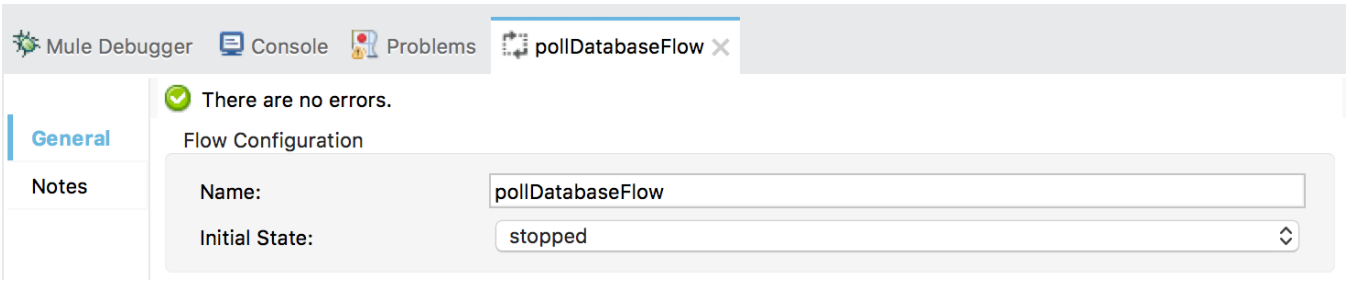
58. Return to the web browser displaying the account data.
59. Add another record with the same postal code.
60. Return to the console in Anypoint Studio.
61. Watch the console until you see your new record displayed on the next polling event.

```
INFO  2016-06-05 14:13:10,810 [pool-27-thread-1] org.mule.transport.polling.watermark.Watermark: Waterm
ark value will not be updated since poll processor returned no results
INFO  2016-06-05 14:13:15,810 [pool-27-thread-1] org.mule.api.processor.LoggerMessageProcessor: [{count
ry=United States, accountID=2770, street=77 Geary Street, state=CA, name=Molly Mule, city=San Francisco
, postal=94108}]
INFO  2016-06-05 14:13:20,821 [pool-27-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
INFO  2016-06-05 14:13:20,821 [pool-27-thread-1] org.mule.transport.polling.watermark.Watermark: Waterm
ark value will not be updated since poll processor returned no results
```

62. Stop the project.

Stop the flow

63. In the Properties view for pollDatabaseFlow, set initial state to stopped.

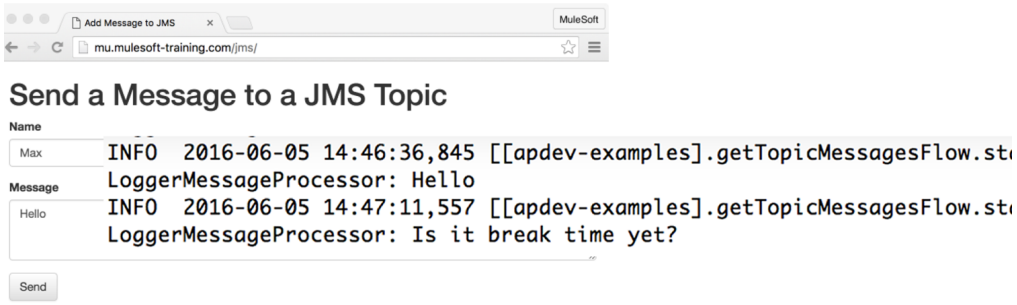


64. Save the file.

Walkthrough 12-4: Connect to a JMS queue (ActiveMQ)

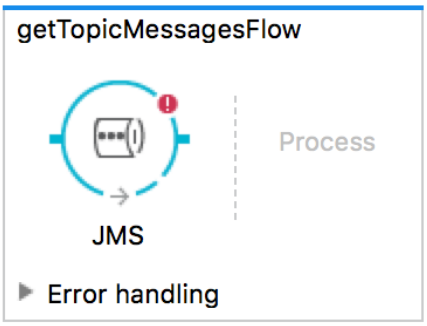
In this walkthrough, you read and write messages from a JMS topic. You will:

- Create a flow accessible at http://localhost:8081/jms.
- Add and configure an ActiveMQ connector.
- Use a JMS endpoint to retrieve messages from a JMS topic.
- Add messages to the topic using a web form.
- Use a JMS endpoint to send messages to a JMS topic.



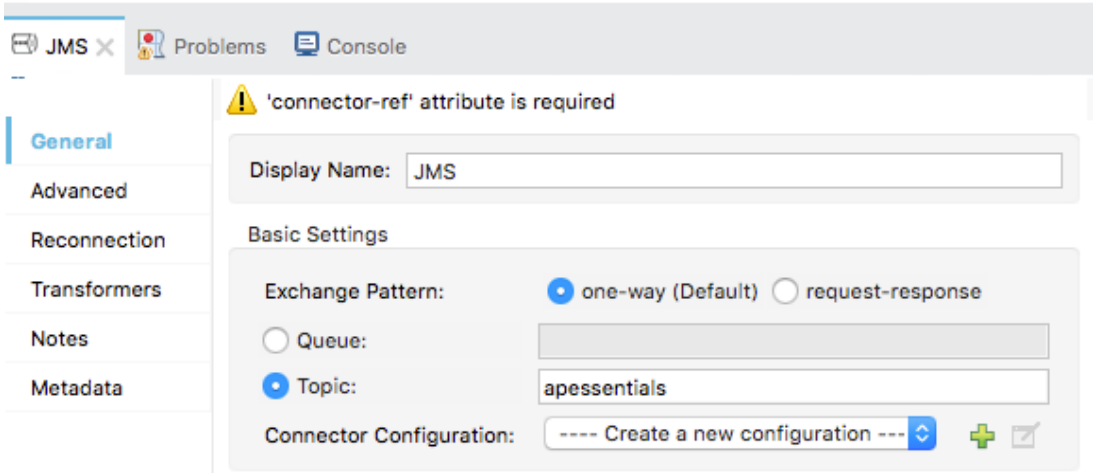
Create a JMS inbound endpoint

1. Return to the apdev-examples project.
2. Create a new Mule configuration file called jms.xml.
3. Drag out a JMS connector from the Mule Palette and drop it in the canvas to create a new flow.
4. Give the flow a new name of getTopicMessagesFlow.



5. In the JMS properties view, leave the exchange pattern set to one-way.

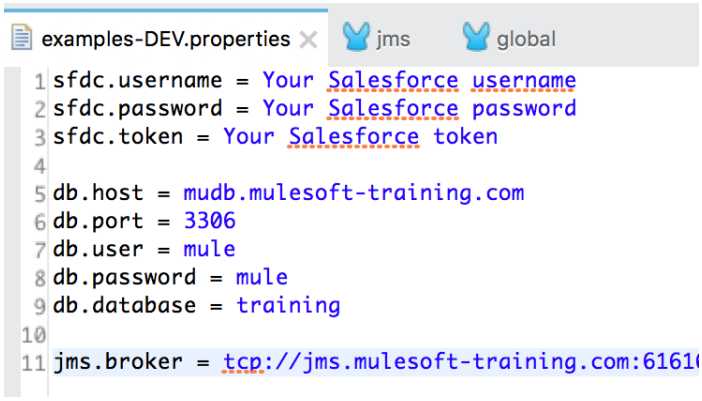
6. Select topic and set it to apessentials.



Configure the JMS connector

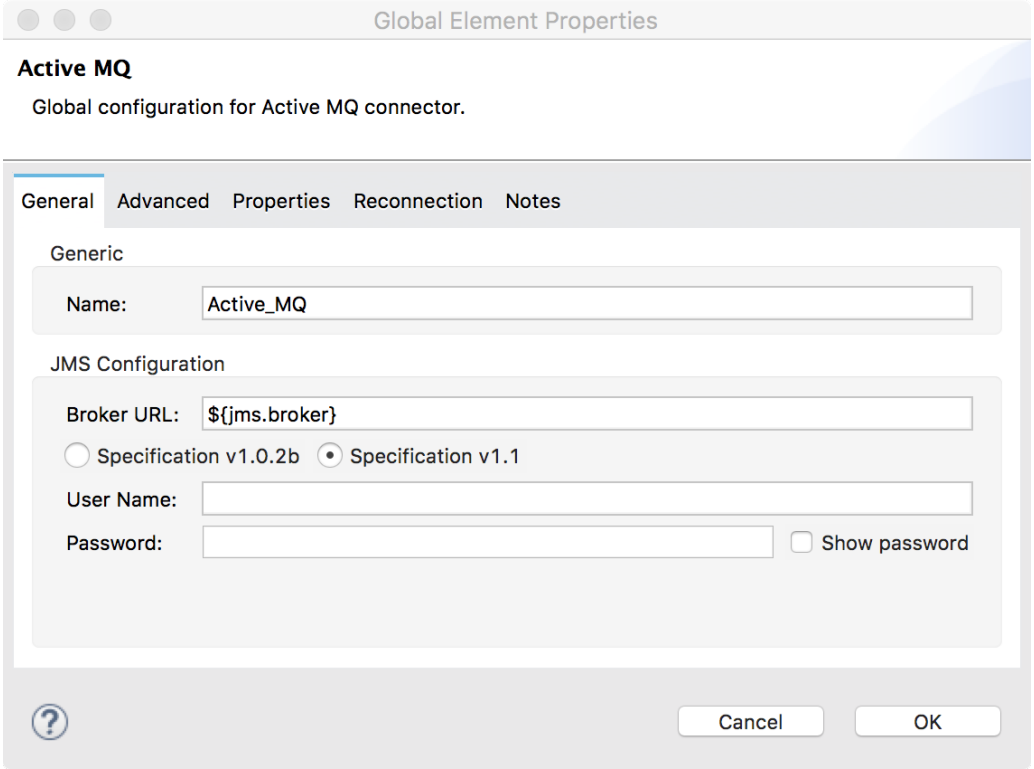
7. Return to the course snippets.txt file and copy the value for the ActiveMQ Broker URL.
8. Return to Anypoint Studio.
9. Return to examples-DEV.properties.
10. Create a new property called jms.broker and set it equal to the value you copied from the course snippets.txt file.

    Note: If you do not have access to port 61616, set jms.broker equal to vm://localhost instead.



11. Save the file.
12. Return to the Global Elements view in global.xml.
13. Click Create.
14. In the Choose Global Type dialog box, select Connector Configuration > JMS > Active MQ and click OK.

15. In the Global Element Properties dialog box, change the broker URL to the property ${jms.broker}.
16. Set the Specification to v1.1.

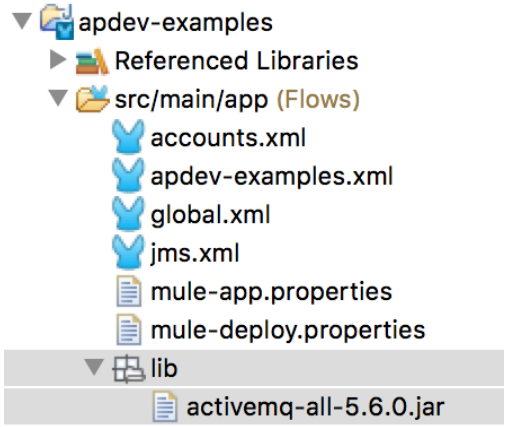


17. Click OK.
18. Return to jms.xml.
19. In the JMS properties view, set the connector configuration to the existing Active_MQ configuration.

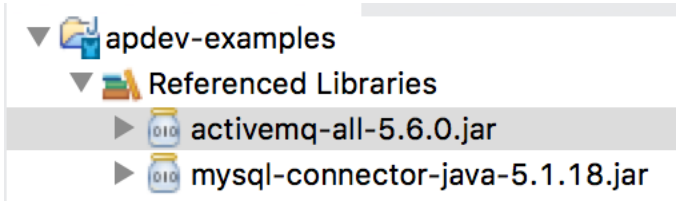
Add the ActiveMQ library

20. In the Package Explorer, right-click apdev-examples and select New > Folder.
21. In the New Folder dialog box, set the folder name to lib and click Finish.
22. In your computer's file browser, locate the activemq-all.jar file located in the jars folder in the student files.

23. Copy and paste or drag the JAR file into the lib folder.



24. Right-click the JAR file in the Package Explorer and select Build Path > Add to Build Path; you should now see it under Referenced Libraries.

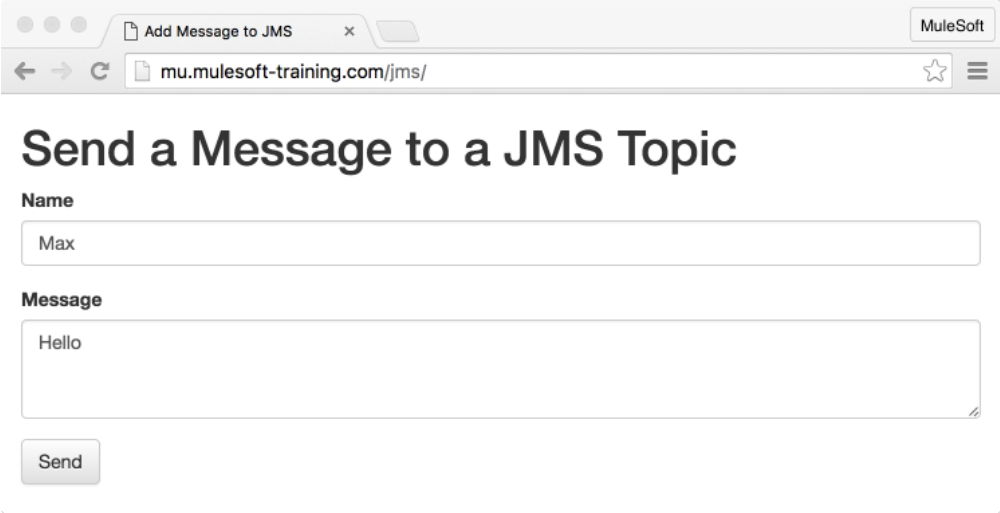


Note: Adding activemq-all.jar can create conflicts with other dependencies in projects, so it is recommended that you only add only the JAR files you need in relation to what you need ActiveMQ for. For more details, see the documentation at https://docs.mulesoft.com/mule-user-guide/v/3.8/activemq-integration.

Test the application and receive messages

25. Return to jms.xml.
26. Add a Logger to the flow and set its message to #[payload].
27. Run the project.
28. In the Clear Application Data dialog box, select Remember decision and click No.
29. Return to the course snippets.txt file and copy the value for the JMS Form URL.
30. In a web browser, navigate to the URL you copied.

31. In the form, enter your name and a message and click Send.



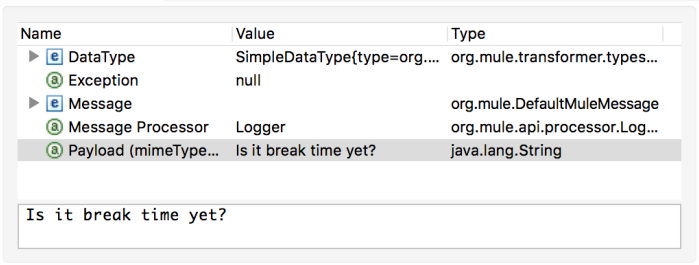
32. In the popup window with the response, click OK.
33. Return to the console in Anypoint Studio; you should see your message along with those from your classmates – but you don't see the names of the people who sent the messages.

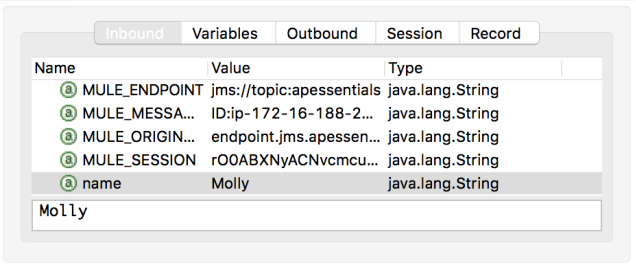
```
INFO  2016-06-05 14:46:36,845 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.
LoggerMessageProcessor: Hello
INFO  2016-06-05 14:47:11,557 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.
LoggerMessageProcessor: Is it break time yet?
```

34. Stop the project.

Debug the application

35. Add a breakpoint to the Logger.
36. Debug the project.
37. Make another request to the JMS form URL and submit another message.
38. In the Mule Debugger view, look at the payload and inbound message properties.





39. Step through the rest of the application.
40. Stop the project and switch perspectives.

Display names with the messages

41. In the Logger properties view, change the message to use an expression to display the name and message.

    ```
    #[message.inboundProperties.name + ": " + payload]
    ```

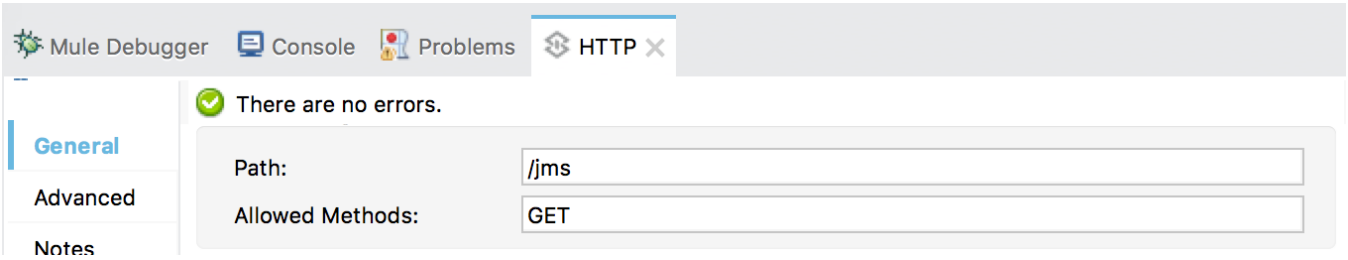
Test the application

42. Run the project.
43. Return to the message form and submit a new message.
44. Look at the console; you should now see names along with messages.

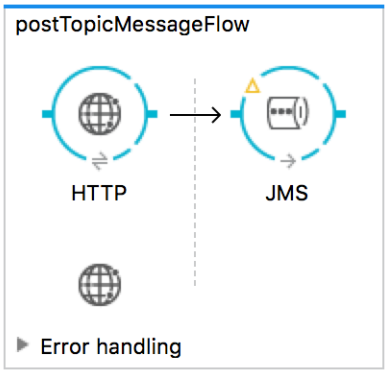
    ```
    INFO  2016-06-05 14:50:26,999 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.
    LoggerMessageProcessor: Max: Hello
    INFO  2016-06-05 14:50:32,902 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.
    LoggerMessageProcessor: Molly: Is it break time yet?
    ```

Create a JMS outbound-endpoint

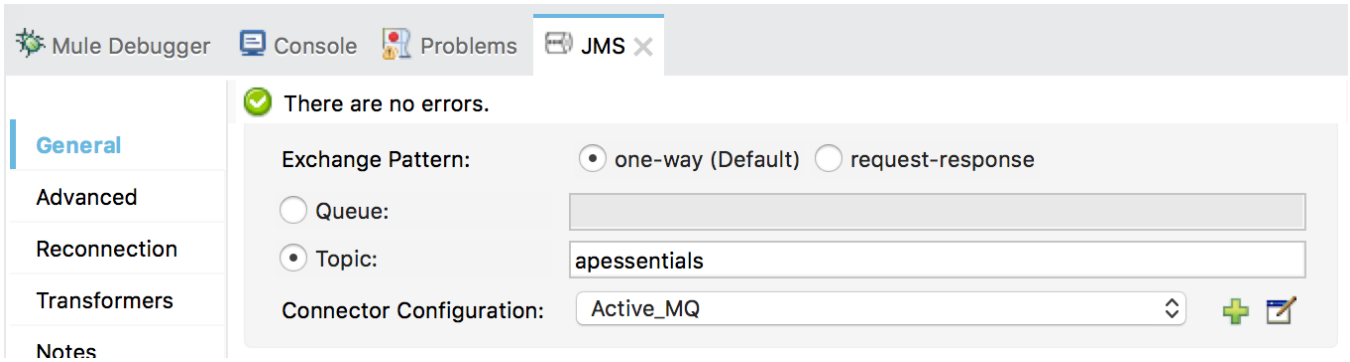
45. Drag out an HTTP connector from the Mule Palette and drop it in the canvas.
46. Give the flow a new name of postTopicMessageFlow.
47. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
48. Set the path to /jms and the allowed methods to GET.



49. Drag out a JMS connector from the Mule Palette and drop it into the process section of the flow.



50. In the JMS Properties view, select topic and set it to apessentials.
51. Set the connector configuration to the existing Active_MQ.

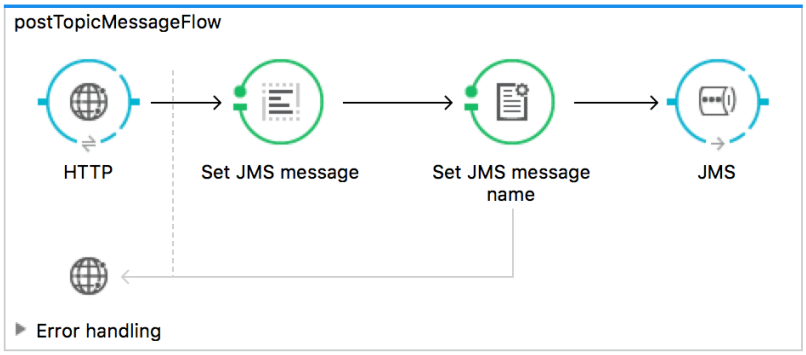


Set a message

52. Add a Set Payload transformer between the HTTP and JMS connector endpoints.
53. In the Set Payload Properties view, change the display name to Set JMS message.
54. Set the value to a message query parameter.

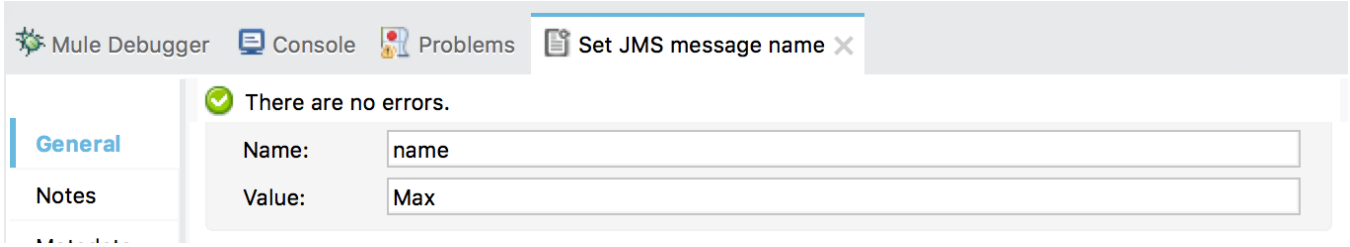
```
#[message.inboundProperties.'http.query.params'.message]
```

55. Add a Property transformer after the Set Payload transformer.
56. In the Properties view, change the display name to Set JMS message name.



57. Select Set Property and set the name to name and the value to your name.

Note: You can set this to a query parameter instead if you prefer.



Test the application and post messages

58. Save the file to redeploy the project.
59. Make a request to http://localhost:8081/jms?message=Hello.
60. Look at the console; you should see your name and message displayed – along with those of your classmates.

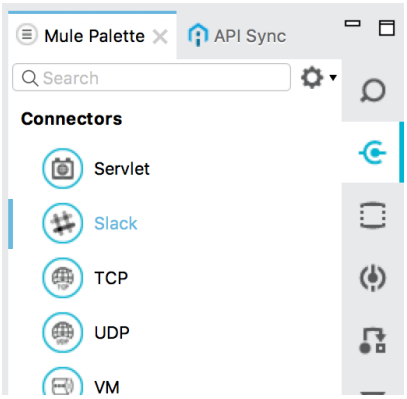
```
INFO  2016-06-05 14:58:05,084 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.
LoggerMessageProcessor: Max: Hello
```

61. Stop the project.

Walkthrough 12-5: Find and install not-in-the-box connectors

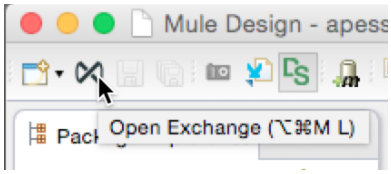
In this walkthrough, you learn how to add a new connector to Anypoint Studio. You will:

- Browse Anypoint Exchange from Anypoint Studio.
- Install a connector from Exchange into Anypoint Studio.
- Locate the new connector in Anypoint Studio.
- Manage installed software.

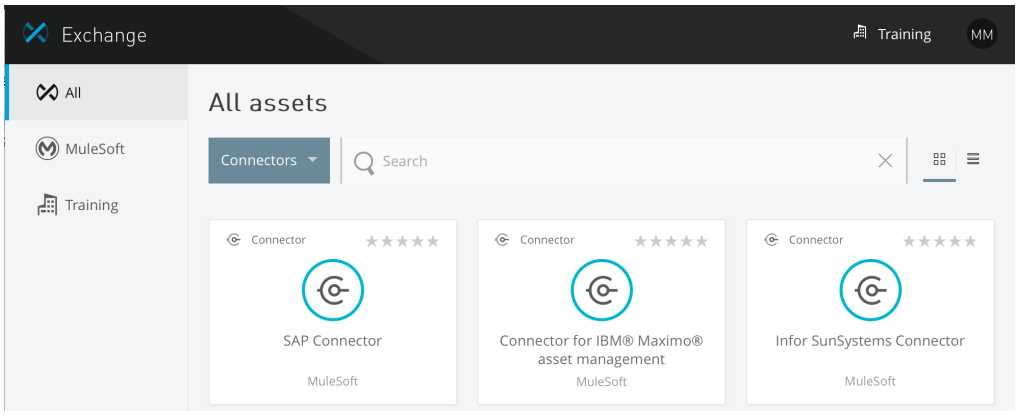


Browse connectors in Anypoint Exchange from Anypoint Studio

1. In Anypoint Studio, click the Open Exchange button; the Anypoint Exchange should open in a new window.



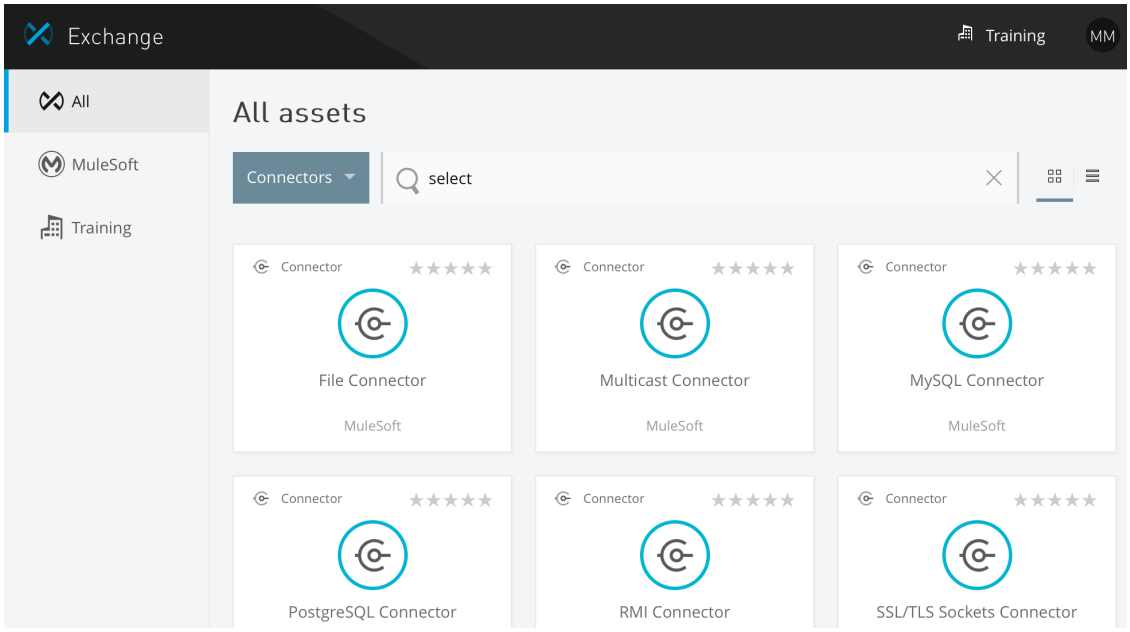
2. In Exchange, select Connectors in the types drop-down menu.



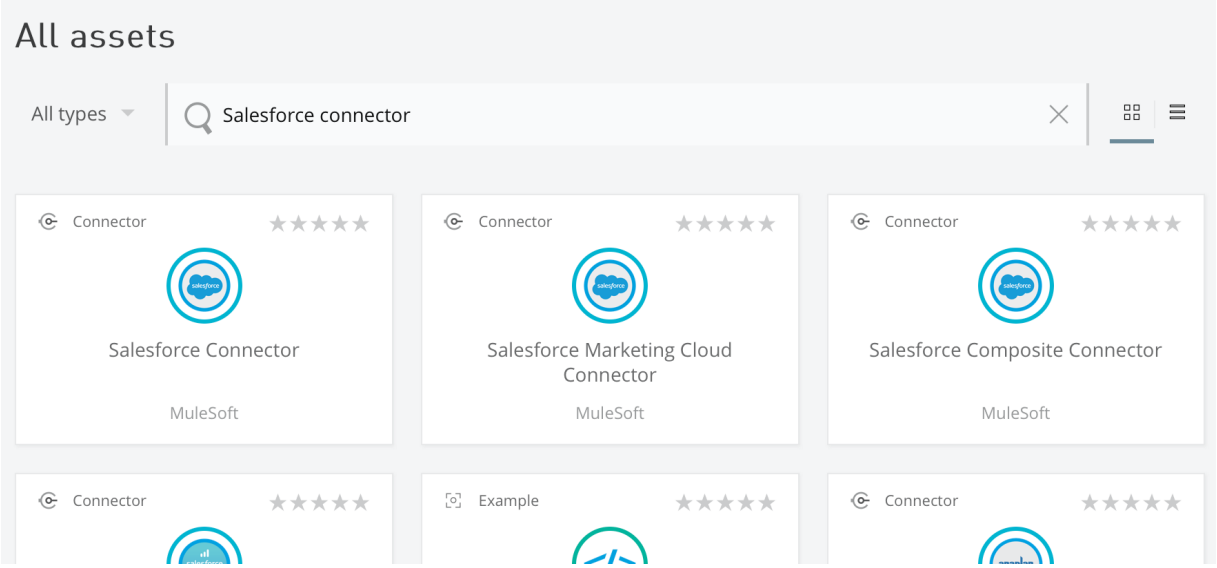
3. Browse the connectors.

Review the Salesforce connector details

4. Search for just the select connectors.

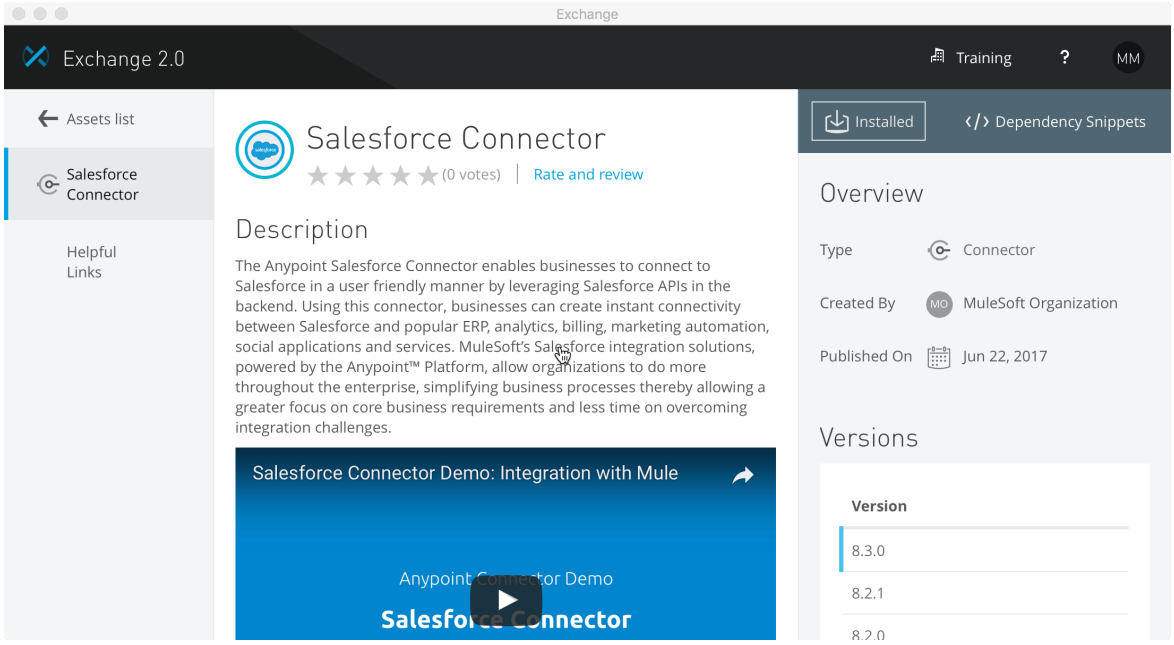


5. Browse the connectors.
6. Search for the Salesforce connector.

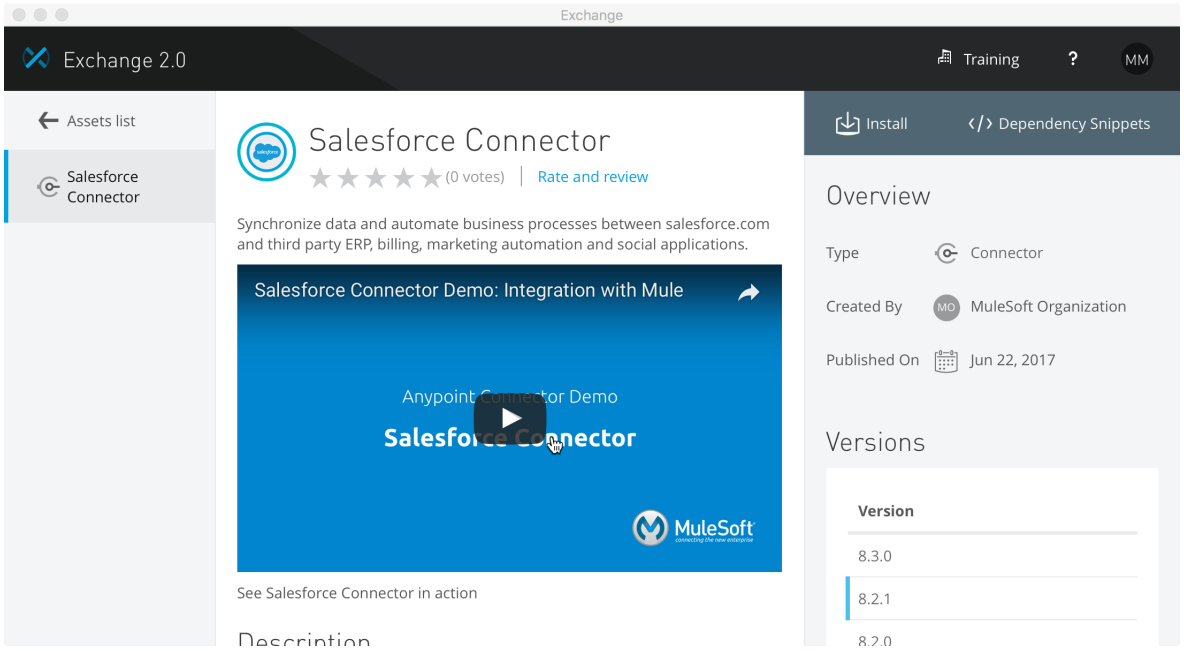


7. Select the Salesforce Connector.
8. In the page that opens, browse the connector's documentation.

9. Review the versions and see if the latest version is installed or not; you should see Installed or Install above the versions at the top of the window.



10. Select an older version in the versions list; you should now see a clickable Install button.

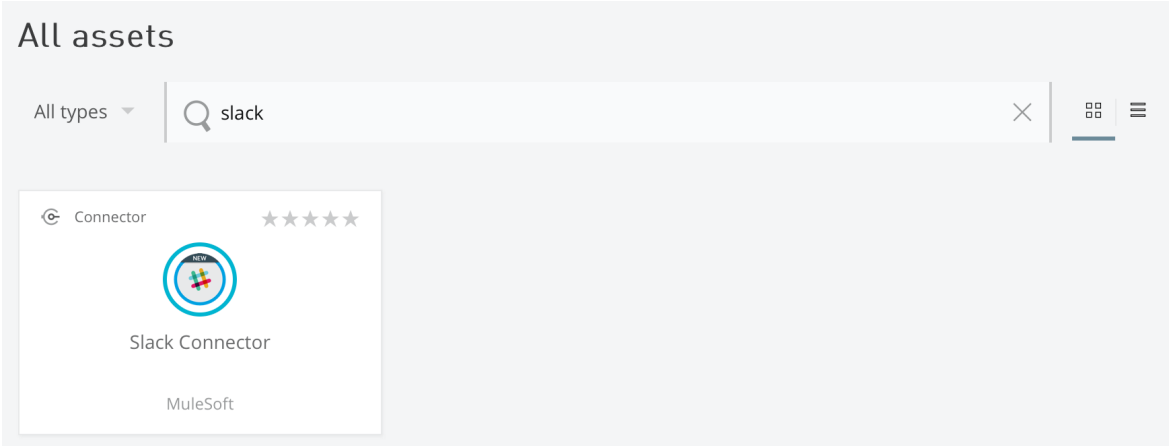


11. To return to the assets list, click the Assets list link on the left side of the page.

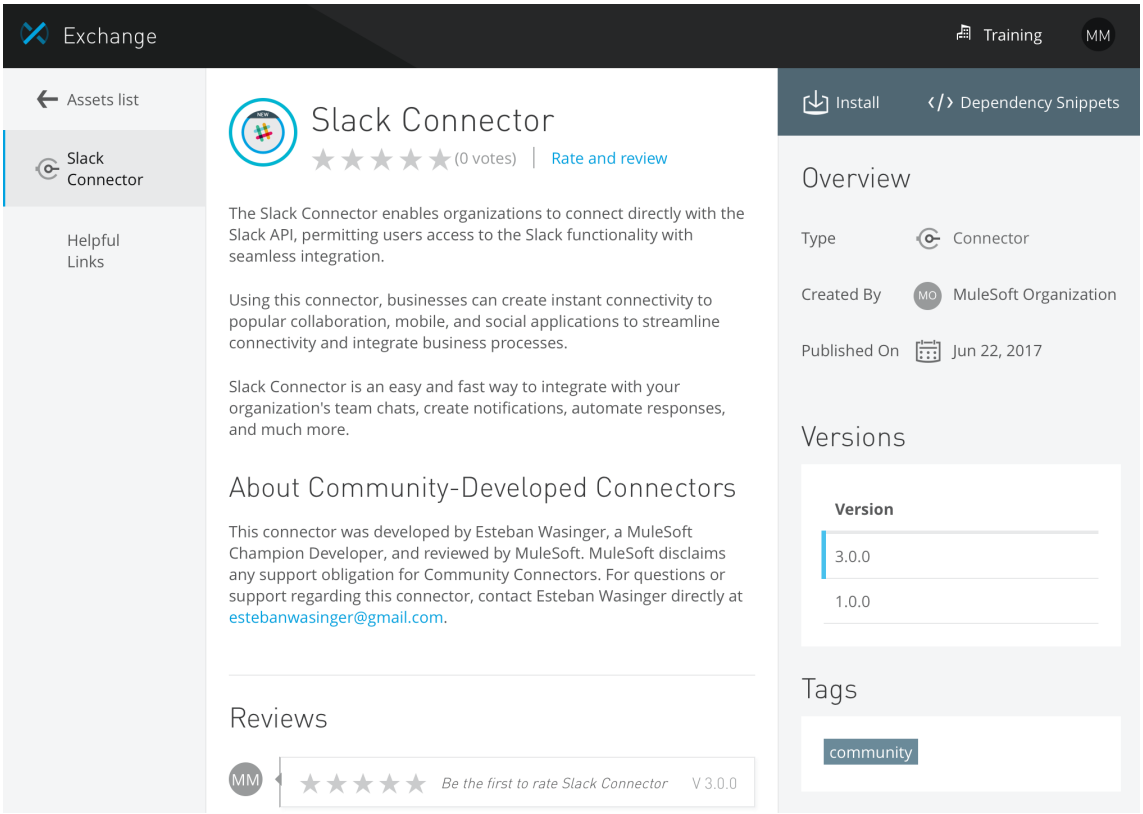
Review the Slack connector

12. Browse the connectors again.

13. Search for and select the Slack Connector (or any other connector you're interested in).



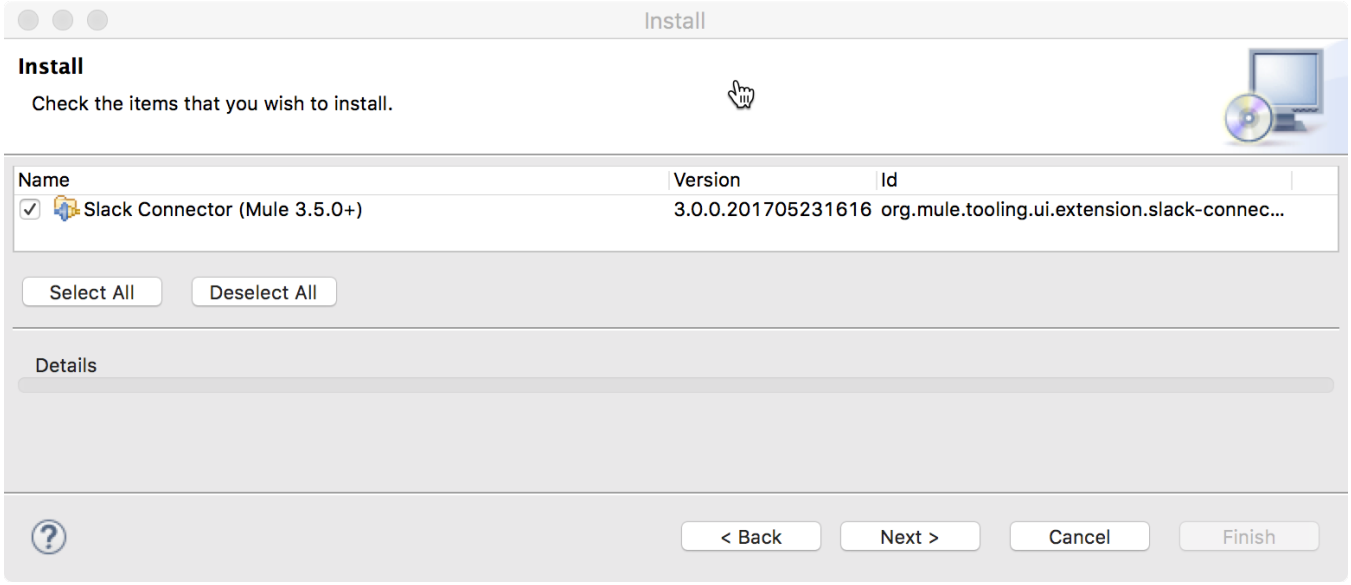
14. Select it and review its documentation; you should see this is a community connector and not supported by MuleSoft.



Install the connector

15. Click the Install button.

16. Wait until an Install dialog box appears in Anypoint Studio.

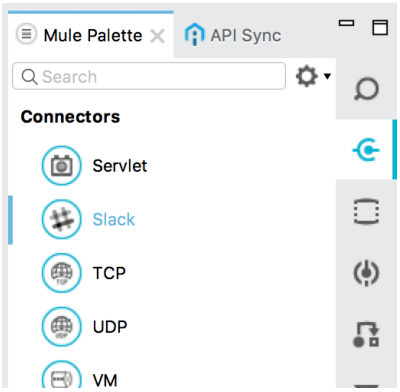


Note: You can also install connector's directly from Anypoint Studio. From the main menu bar, select Help > Install New Software. In the Install dialog box, click the Work with drop-down button and select Anypoint Connectors Update Site. Drill-down into Community and select the Slack Connector (or some other connector).

17. Click Next.
18. On the Install Details page, click Next.
19. On the Review Licenses page, select the I accept the terms of the license agreement radio button.
20. Click Finish; the software will be installed and then you should get a message to restart Anypoint Studio.
21. In the Software Updates dialog box, click Yes to restart Anypoint Studio.

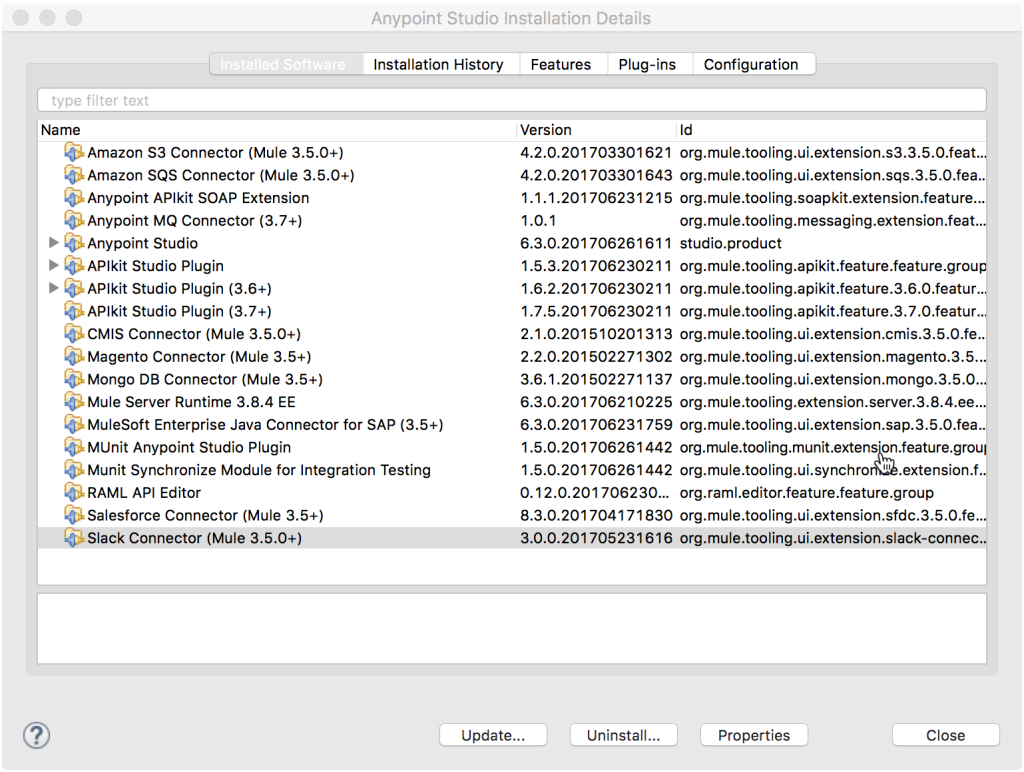
Locate the new connector in Anypoint Studio

22. After Anypoint Studio restarts, locate the new connector in the Connectors section of the Mule Palette.



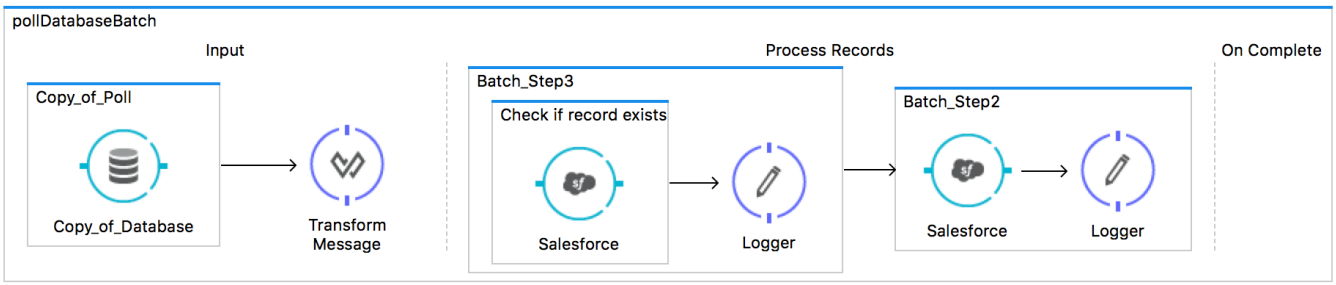
Manage installed software

23. In the main menu, select Help > Installation Details.
24. Select the Slack Connector; you should see Update, Uninstall, and Properties buttons for the connector.



25. Click Close.

Module 13: Processing Records



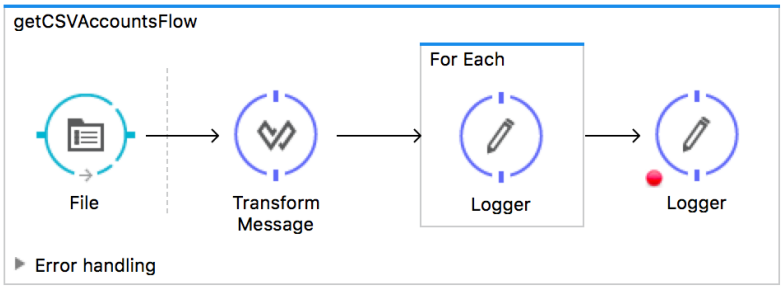
At the end of this module, you should be able to:

- Use the For Each scope to process items in a collection individually.
- Use the batch job element (EE) to process individual records.
- Trigger a batch job using a poll.
- Use a batch job to synchronize data from a legacy database to a SaaS application.

Walkthrough 13-1: Process items in a collection individually

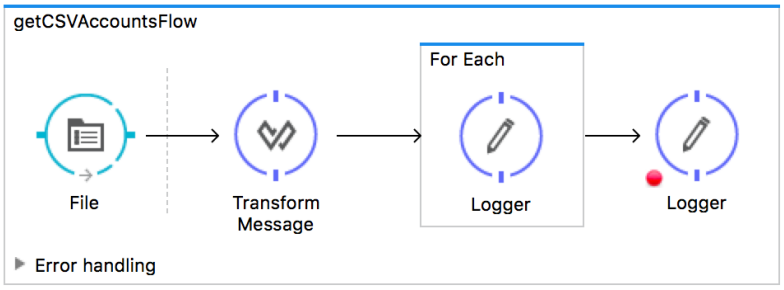
In this walkthrough, you split a collection and process each item in it. You will:

- Use the For Each scope element to process each item in a collection individually.
- Look at the thread used to process each record.



Add a for each scope element

1. Return to accounts.xml.
2. Review getCSVAccountsFlow.
3. Drag a For Each scope element from the Mule Palette and drop it after the Transform Message component.
4. Add a Logger to the For Each scope.



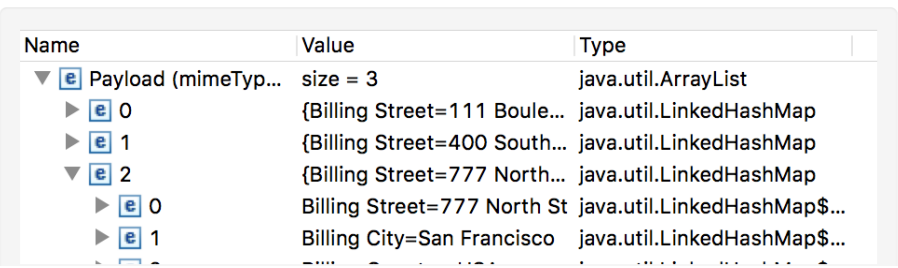
Process each element

5. In the Logger Properties view, set the message to #[payload].

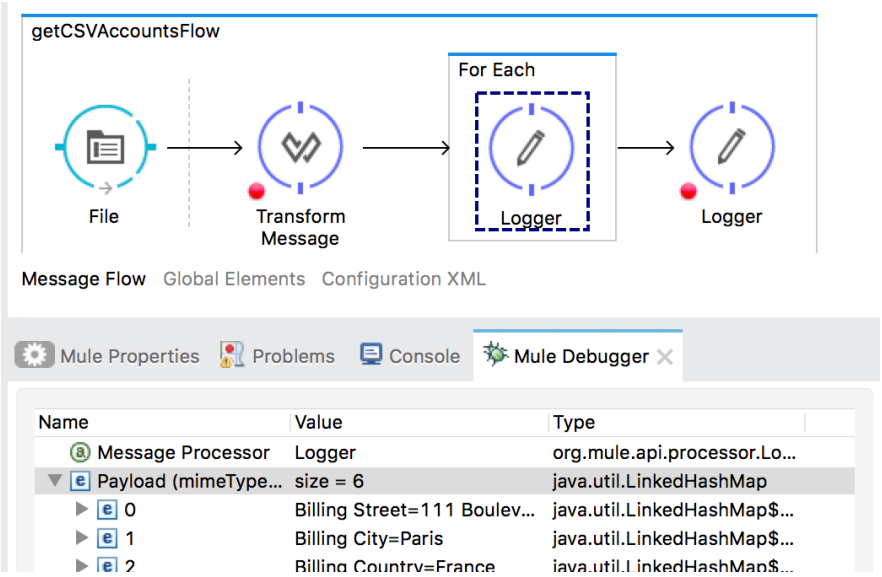
Debug the application

6. Add a breakpoint to the Transform Message component.
7. Debug the project.
8. Drag accounts.csv from the src/main/resources/output folder to the src/main/resources/input folder; application execution should stop at the Transform Message component.

9. In the Mule Debugger view, look at the payload value.
10. Step to the For Each scope; the payload should be an ArrayList of LinkedHashMaps.



11. Step into the For Each scope; the payload should be a LinkedHashMap.



12. Look at the console; you should see the data for the first record.
13. Step through the application; you should see each record displayed.

```
ggerMessageProcessor: {Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=Franc
e, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}
INFO  2016-06-05 15:55:30,638 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
ggerMessageProcessor: {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Bi
lling State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}
INFO  2016-06-05 15:55:30,639 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
ggerMessageProcessor: {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Bi
lling State=CA, Name=Cat Park Industries, BillingPostalCode=91156}
INFO  2016-06-05 15:55:33,708 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
```

14. Step through to the Logger after the For Each scope; the payload should be an ArrayList again.
15. Step to the end of the application.
16. Stop the project and switch perspectives.

Look at the processing threads

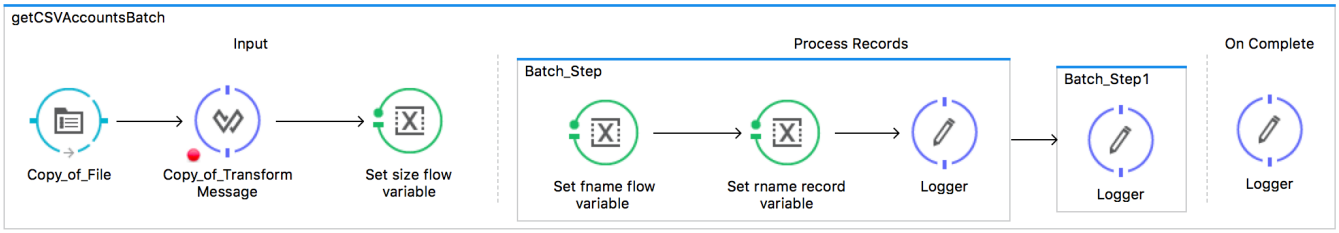
17. In the console, locate the thread number used to process each record in the collection; the same thread should be used for each: apdev-examples.getCSVAccountsFlow.stage1.**02.**

```
INFO  2016-06-05 15:55:30,637 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
ggerMessageProcessor: {Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=Franc
e, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}
INFO  2016-06-05 15:55:30,638 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
ggerMessageProcessor: {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Bi
lling State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}
INFO  2016-06-05 15:55:30,639 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
ggerMessageProcessor: {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Bi
lling State=CA, Name=Cat Park Industries, BillingPostalCode=91156}
INFO  2016-06-05 15:55:33,708 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Lo
ggerMessageProcessor: [{Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=Fran
ce, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}, {Billing Street=400 South St, B
illing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Iguana Park Industries, BillingP
ostalCode=91156}, {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billin
g State=CA, Name=Cat Park Industries, BillingPostalCode=91156}]
```

Walkthrough 13-2: Create a batch job for records in a file

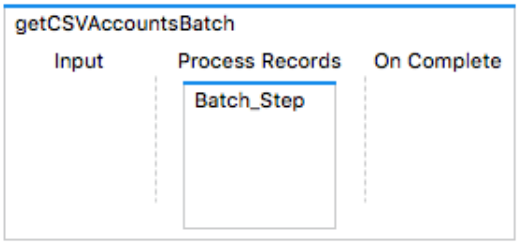
In this walkthrough, you create a batch job to process records in a CSV file. You will:

- Create a batch job.
- Explore flow and record variable persistence across batch steps and phases.
- In the input phase, check for CSV files every second and convert them to a collection of objects.
- In the process records phase, create two batch steps for setting and tracking variables.
- In the on complete phase, look at the number of records processed and failed.
- Look at the thread used to process each record in each step.



Create a batch job

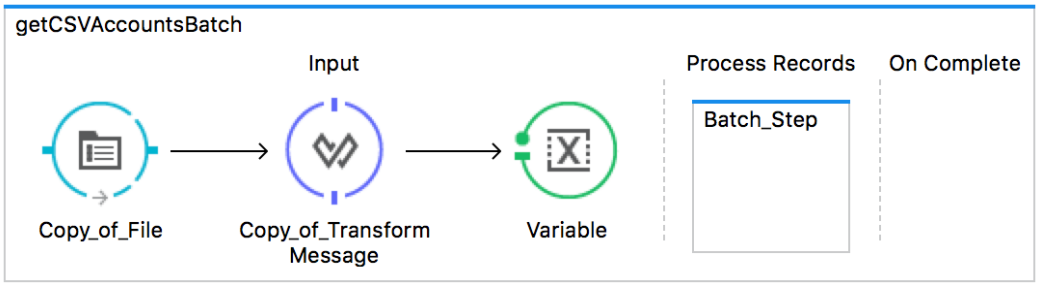
1. Return to accounts.xml.
2. Drag a Batch scope element from the Mule Palette and drop it above getCSVAccountsFlow.
3. Change the batch job name to getCSVAccountsBatch.



Add elements to the input phase

4. Select the File and Transform Message elements in getCSVAccountsFlow and select Edit > Copy.
5. Click the input section of the batch job and select Edit > Paste.

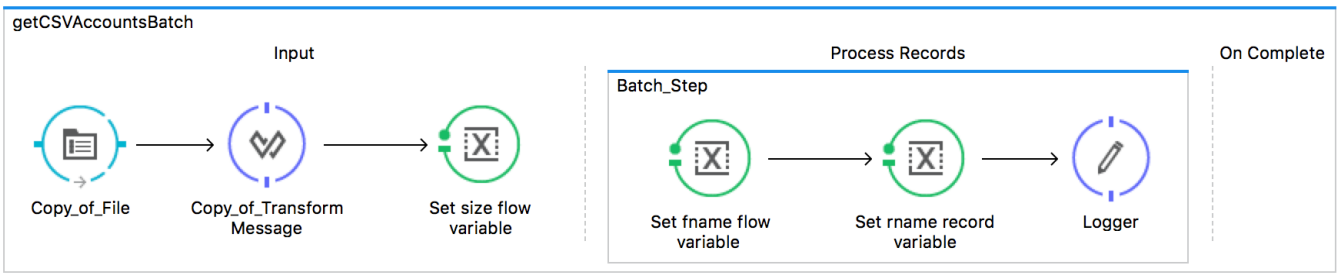
6. Add a Variable transformer at the end of input phase.



7. In the Variable properties view, change the display name to Set size flow variable and select Set Variable.
8. Set the name to size and the value to #[payload.size()].

Add processors to a batch step in the process records phase

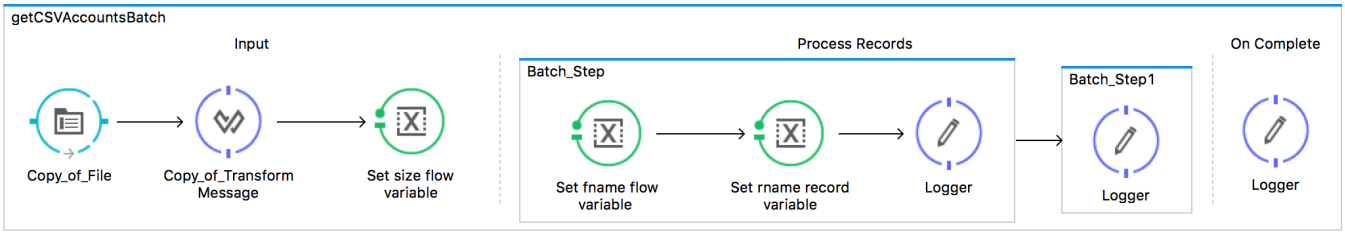
9. Add a Variable transformer to the batch step in the process records phase.
10. In the Variable properties view, change the display name to Set fname flow variable and select Set Variable.
11. Set the name to fname and the value to #[payload.Name].
12. Add a Record Variable transformer to the batch step.
13. Change the display name to Set rname record variable.
14. In the Record Variable properties view, select Set Record Variable.
15. Set the name to rname and the value to #[payload.Name].
16. Add a Logger component to the batch step.
17. Set its message to #[recordVars.rname].



Create a second batch step

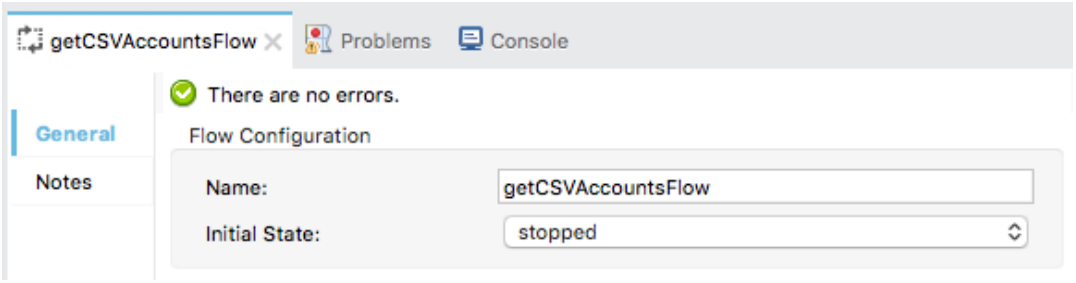
18. Drag a Batch Step scope from the Mule Palette and drop it in the process records phase after the first batch step.

19. Add a Logger to the second batch step.
20. Set its message to #[recordVars.rname].
21. Add a Logger to the on complete phase.



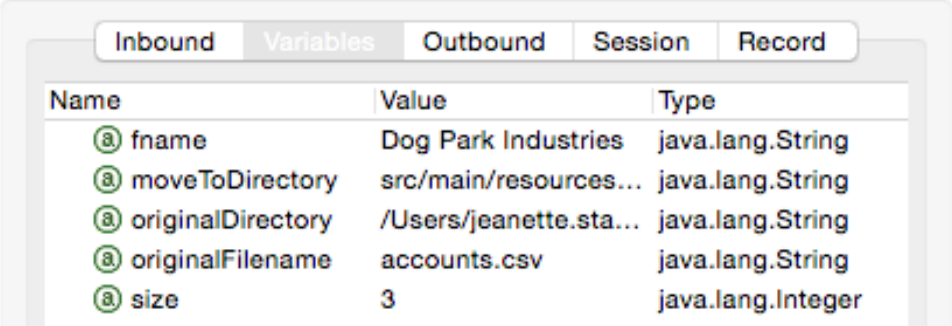
Stop the other flow watching the same file location from being executed

22. Double-click getCSVAccountsFlow.
23. In the getCSVAccountsFlow properties view, set the initial state to stopped.

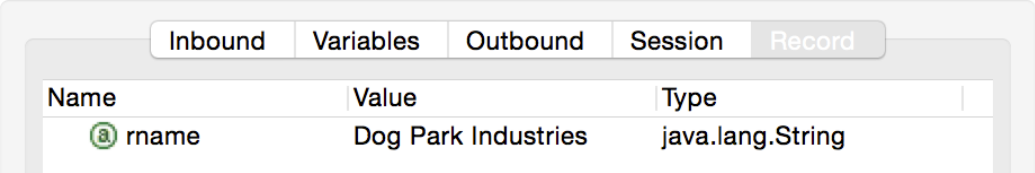


Debug the application

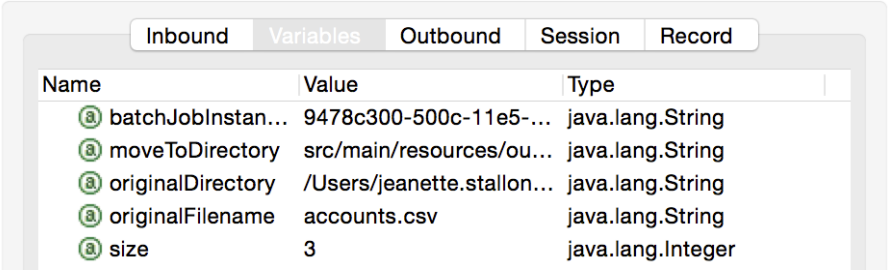
24. In getCSVAccountsBatch, add a breakpoint to the Transform Message component.
25. Debug the project.
26. After the application starts, drag the accounts.csv file from the project's src/main/resources/output folder to the src/main/resources/input folder.
27. In the Mule Debugger view, watch the payload value.
28. Step into the process records phase.
29. Click the Variables tab; you should see the size flow variable.



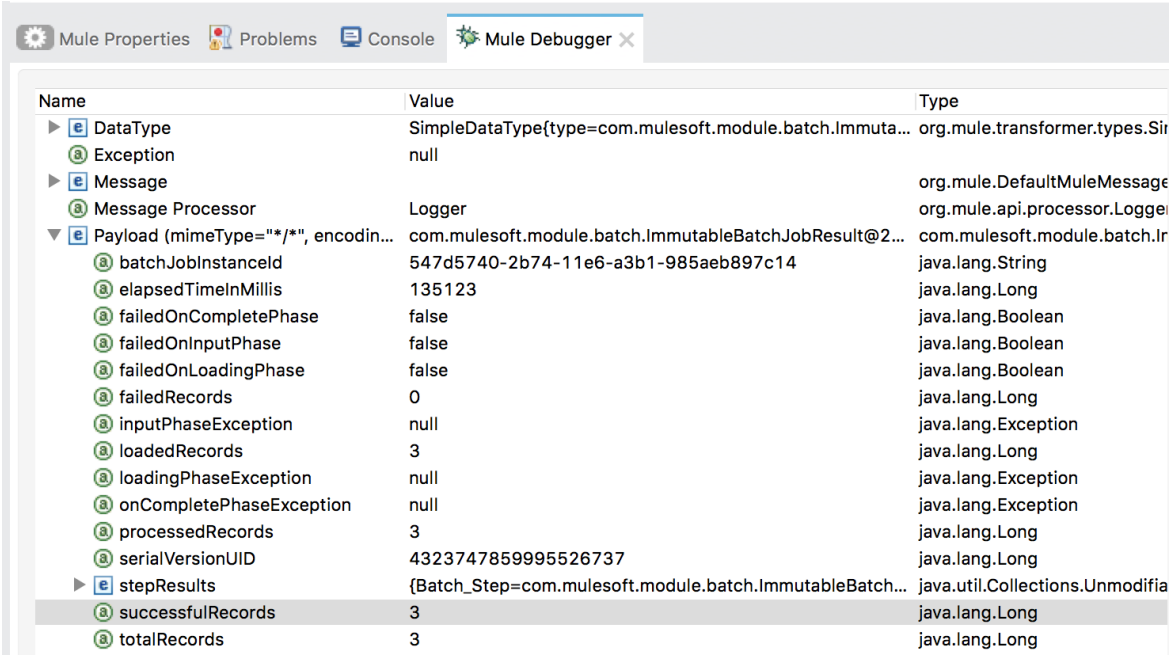
30. Step to the Logger in the first batch step.
31. Click the Record tab; you should see the rname flow variable.



32. Step through the rest of the records in the first batch step and watch the payload, the fname flow variable, and the rname record variable.
33. Step into the second batch step and look at the payload, the flow variables, and the record variables; you should see the rname record variable and size flow variable but not the fname flow variable.



34. Step through the rest of the records in the second batch step.
35. Step into the on complete phase; you should see the payload is an ImmutableBatchJobResult.
36. Locate the values for totalRecords, successfulRecords, and FailedRecords.



37. Step to the end of the application.
38. Stop the project and switch perspectives.

Look at the processing threads

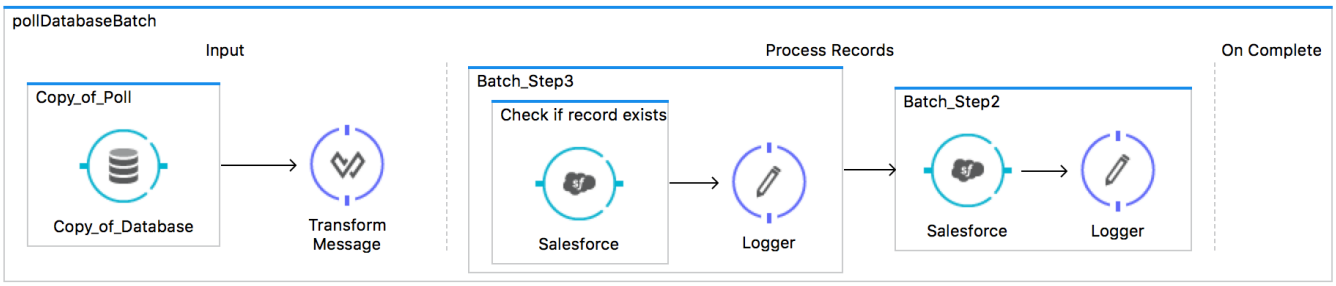
39. In the console, locate the thread number used to process each record in the collection in each step of the batch process.

```
INFO  2016-06-05 16:32:46,099 [[apdev-examples].connector.file.mule.default.receiver.01] com.mulesoft.m
odule.batch.engine.queue.BatchQueueLoader: Finished loading phase for instance ce37a4e0-2b75-11e6-beca-
985aeb897c14 of job getCSVAccountsBatch. 3 records were loaded
INFO  2016-06-05 16:32:46,100 [[apdev-examples].connector.file.mule.default.receiver.01] com.mulesoft.m
odule.batch.engine.DefaultBatchEngine: Started execution of instance 'ce37a4e0-2b75-11e6-beca-985aeb897
c14' of job 'getCSVAccountsBatch'
INFO  2016-06-05 16:32:46,150 [batch-job-getCSVAccountsBatch-work-manager.01] org.mule.api.processor.Lo
ggerMessageProcessor: Dog Park Industries
INFO  2016-06-05 16:32:46,157 [batch-job-getCSVAccountsBatch-work-manager.01] org.mule.api.processor.Lo
ggerMessageProcessor: Iguana Park Industries
INFO  2016-06-05 16:32:46,157 [batch-job-getCSVAccountsBatch-work-manager.01] org.mule.api.processor.Lo
ggerMessageProcessor: Cat Park Industries
INFO  2016-06-05 16:32:46,171 [batch-job-getCSVAccountsBatch-work-manager.01] com.mulesoft.module.batch
.DefaultBatchStep: Step Batch_Step finished processing all records for instance ce37a4e0-2b75-11e6-beca
-985aeb897c14 of job getCSVAccountsBatch
INFO  2016-06-05 16:32:46,173 [batch-job-getCSVAccountsBatch-work-manager.02] org.mule.api.processor.Lo
ggerMessageProcessor: Dog Park Industries
INFO  2016-06-05 16:32:46,173 [batch-job-getCSVAccountsBatch-work-manager.02] org.mule.api.processor.Lo
ggerMessageProcessor: Iguana Park Industries
INFO  2016-06-05 16:32:46,174 [batch-job-getCSVAccountsBatch-work-manager.02] org.mule.api.processor.Lo
ggerMessageProcessor: Cat Park Industries
INFO  2016-06-05 16:32:46,183 [batch-job-getCSVAccountsBatch-work-manager.02] com.mulesoft.module.batch
.engine.DefaultBatchEngine: Starting execution of onComplete phase for instance ce37a4e0-2b75-11e6-beca
-985aeb897c14 of job getCSVAccountsBatch
```

Walkthrough 13-3: Create a batch job to synchronize records from a database to Salesforce

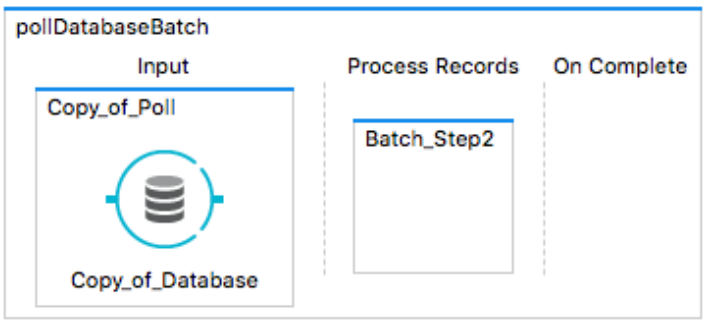
In this walkthrough, you create a batch job to retrieve records from a legacy database and insert them into Salesforce if they do not already exist. You will:

- Create a batch job that polls a database for records with a specific postal code.
- Use a Message Enricher scope to check if a record already exists in Salesforce (an account with the same Name) and stores the result in a record variable and retains the original payload.
- Add a second batch step with a filter that only allows new records (records that don't already exist in Salesforce) to be added to Salesforce.
- Use a batch commit scope to commit records in batches.



Create a batch job that polls a database

1. Return to accounts.xml.
2. Drag a Batch scope from the Mule Palette and drop it above pollDatabaseFlow.
3. Change the flow name to pollDatabaseBatch.
4. Copy the Poll scope in pollDatabaseFlow and paste it into the input phase of pollDatabaseBatch.



Log each record to the console in the process records phase

5. Add a Logger component to the batch step in the process records phase.

6. Set its message to display each record – the message payload in that phase – on a new line.

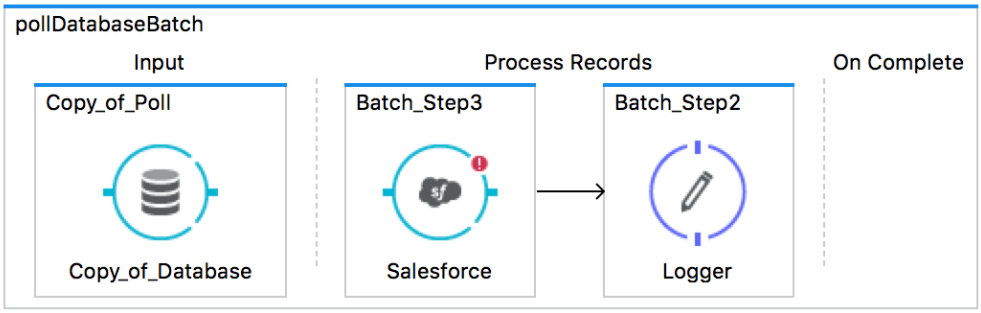
```
#['\n\nRECORD: ' + payload]
```

Test the application

7. Select Run > Run Configurations.
8. In the Run Configurations dialog box, set clear application data to prompt.
9. Click Run.
10. In the Clear Application Data dialog box, click Yes.
11. Watch the console; you should see the same behavior as for pollDatabaseFlow: records with the specific postal code are displayed once and then every 5 seconds no new records are returned.

Add logic to check if a record already exists in Salesforce

12. Drag a Batch Step scope from the Mule Palette and drop it at the beginning of the process records phase before the existing batch step.
13. Add a Salesforce endpoint to the new batch step.

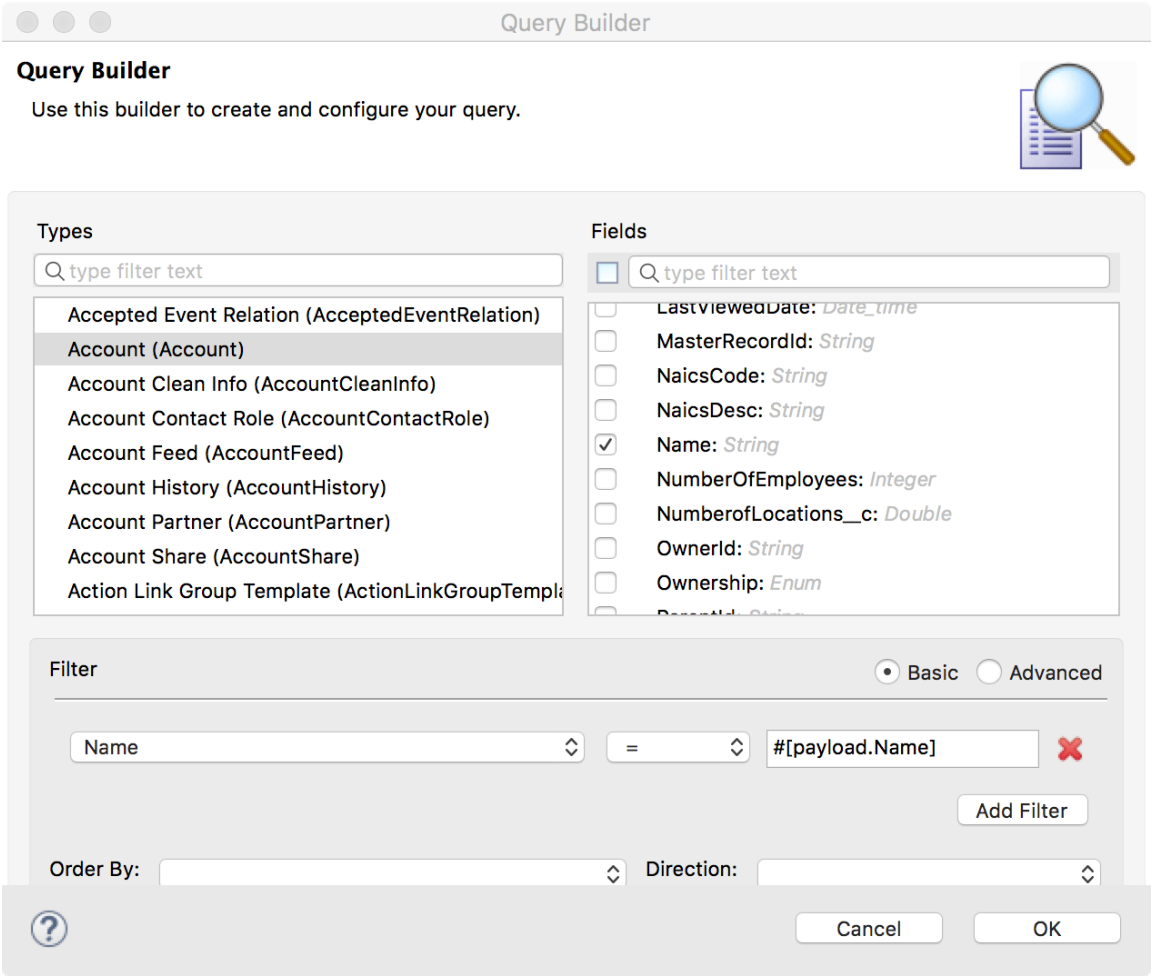


14. Configure the Salesforce endpoint to use the existing Salesforce connector configuration.
15. Set the operation to Query.
16. Click the Query Builder button.
17. Set the type to Account (Account).
18. Select a field of Name; it does not matter what you select, you just want to see if any records are returned.
19. Click the Add Filter button.

20. Create a filter to check if the Name field in the record being processed already exists in the database.

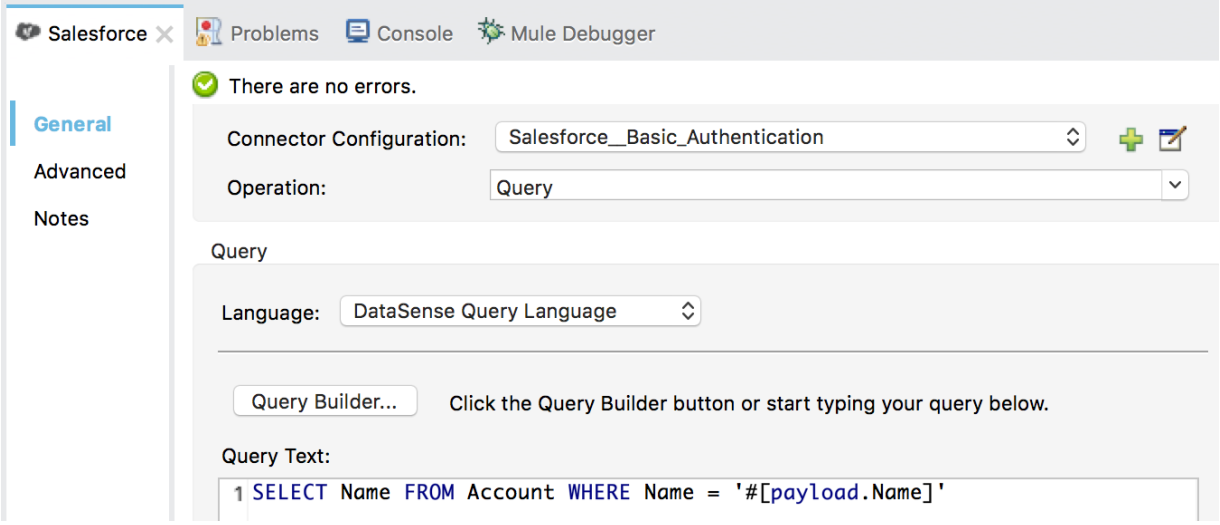
```
Name = #[payload.Name]
```

Note: Right now, you can use payload.name or payload.Name because the payload is a caseSensitiveHashMap. Later this walkthrough, however, you will transform the Map to an Account object with a Name property and will have to refer to this as payload.Name.



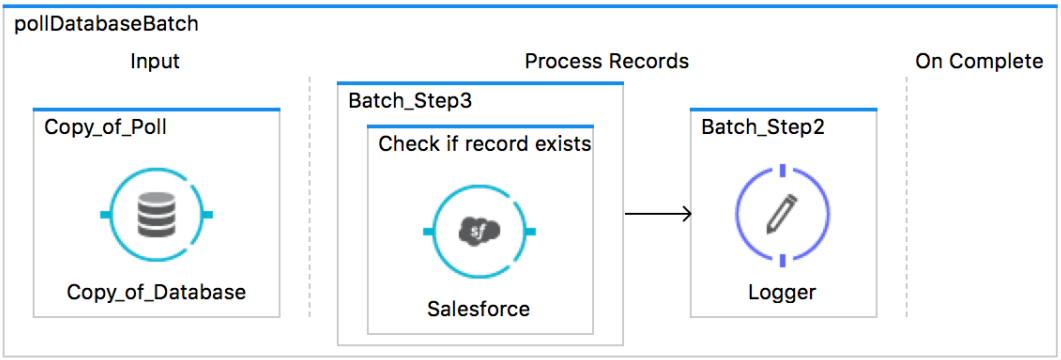
21. Click OK.

22. Review the generated query.



Place the logic in a message enricher

23. Add a Message Enricher scope element to the first batch step.
24. Move the Salesforce endpoint so it is inside the message enricher.
25. Change the Message Enricher display name to Check if record exists.

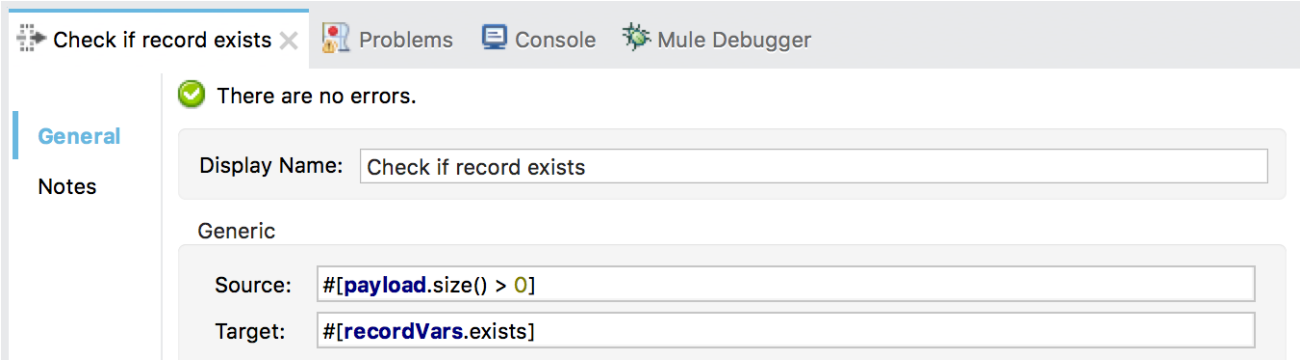


26. In the Message Enricher Properties view, set the source to an expression that checks to see if any records were returned, i.e. if there is a payload.

```
#[payload.size() > 0]
```

27. Set the target to assign the result of the source expression to a record variable called exists.

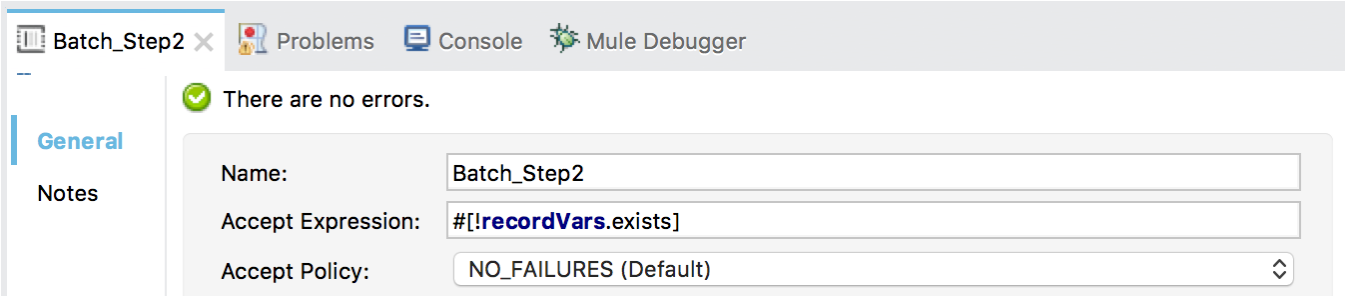
```
#[recordVars.exists]
```



Set a filter for the insertion step

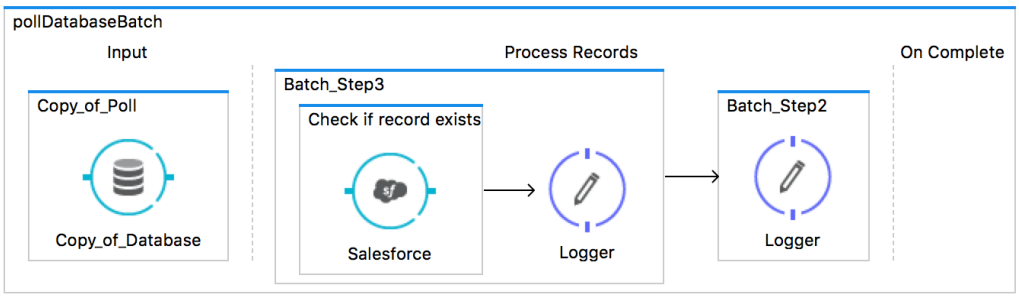
28. Navigate to the Properties view for the second batch step.
29. Set the accept expression so that only records that have a record variable exists set to false are processed.

```
#[!recordVars.exists]
```

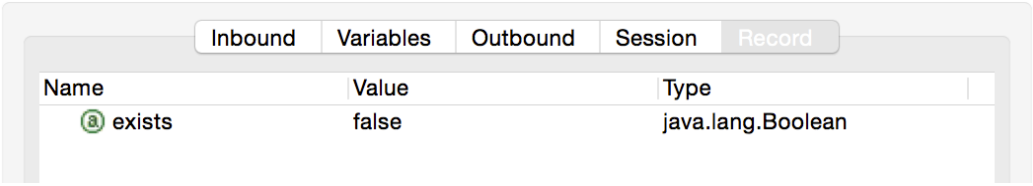


Test the application

30. Add a breakpoint to the Message Enricher.
31. Add a Logger after the Message Enricher scope in the first batch step.



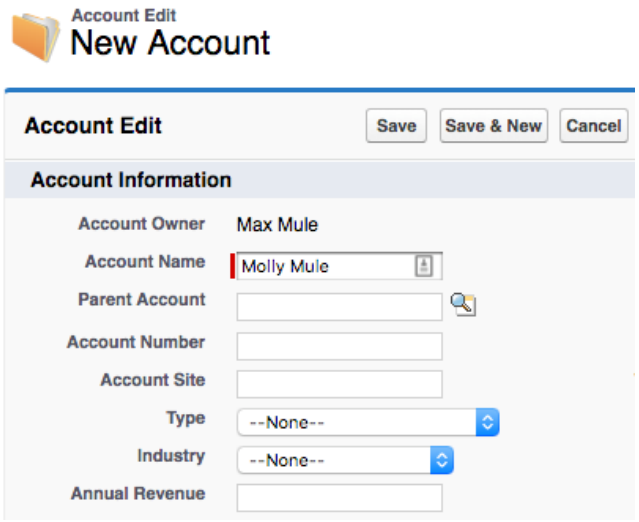
32. Debug the project and clear the application data.
33. Step through the application and watch the record variables; you should see exists set to false for records with names that don't exist in Salesforce and true for those that do.



34. Stop the project.

Add an account to Salesforce manually with a name matching one of the database records

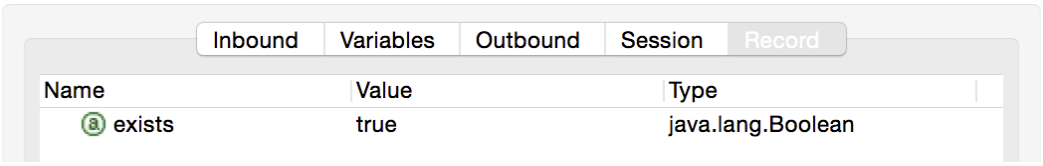
35. In a web browser, navigate to http://login.salesforce.com/ and log in with your Salesforce Developer account.
36. Click the Accounts link in the main menu bar.
37. In the view drop-down menu, select All Accounts with Postal Code and click the Go button.
38. Click the New Account button.
39. Enter an account name (one that matches one of the accounts you added with your postal code to the MySQL database) and click Save.



Test the application

40. Return to Anypoint Studio.
41. Debug the project and clear the application data.

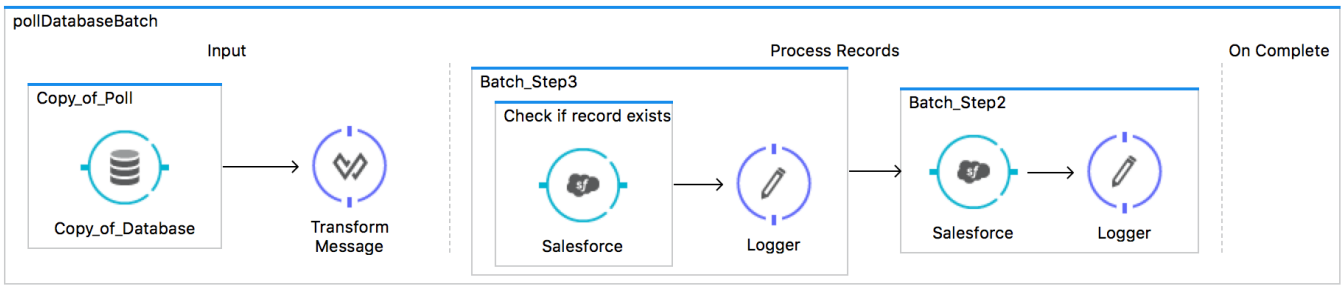
42. Step through the application and watch the record variables; you should now get one record that gets its exists record variable set to true.



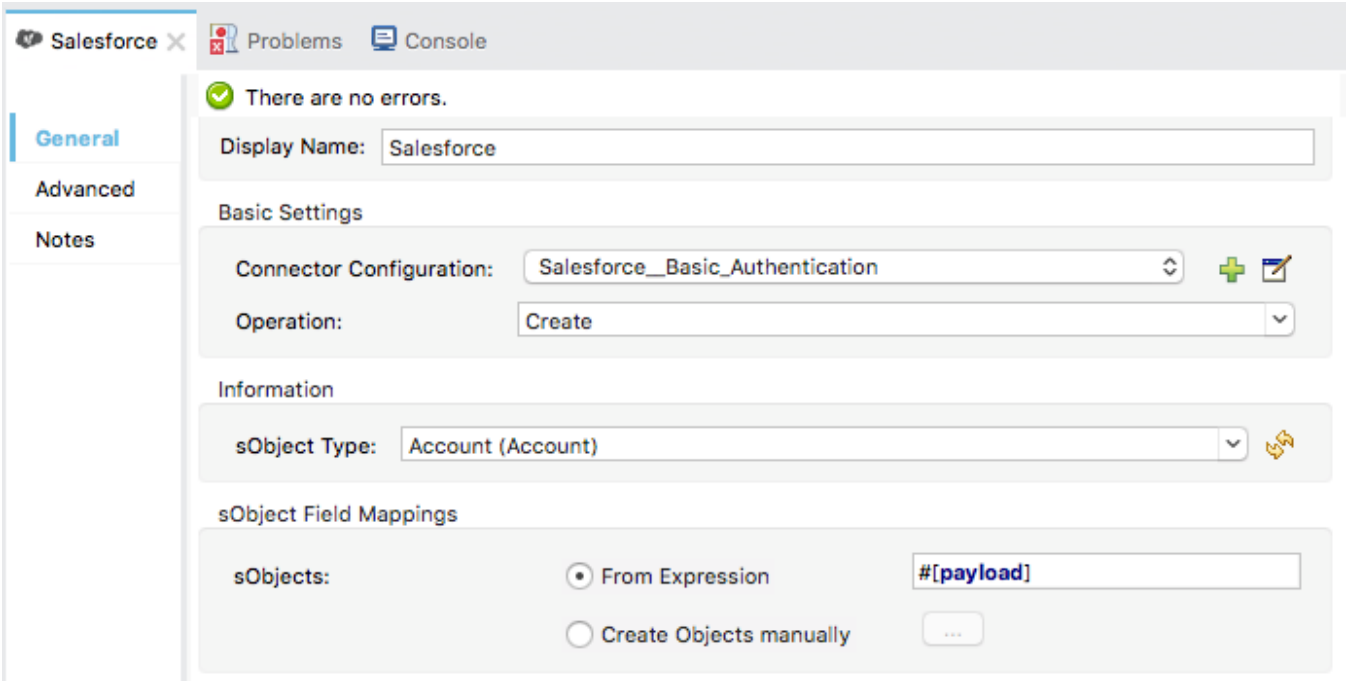
43. Stop the project and switch perspectives.

Add an endpoint to add new account records to Salesforce

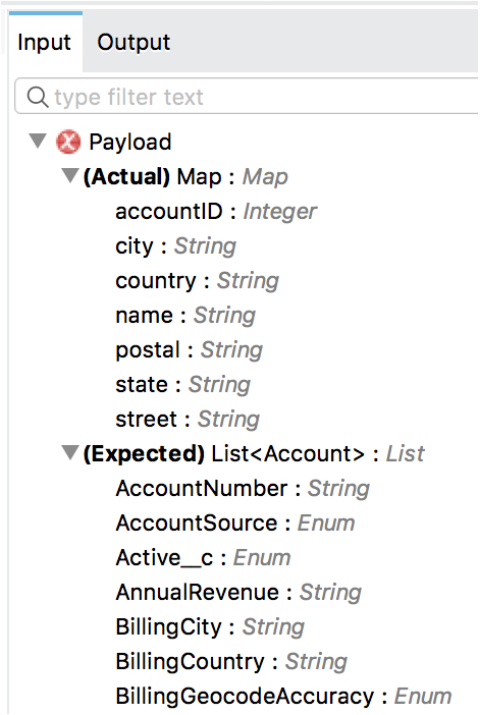
44. Add a Salesforce endpoint to the second batch step in the processing phase.



45. Configure the Salesforce endpoint to use the existing Salesforce connector configuration.
46. Set the operation to Create.
47. Set the sObject type to Account (Account).
48. Leave the sObject field mappings to from expression #[payload].



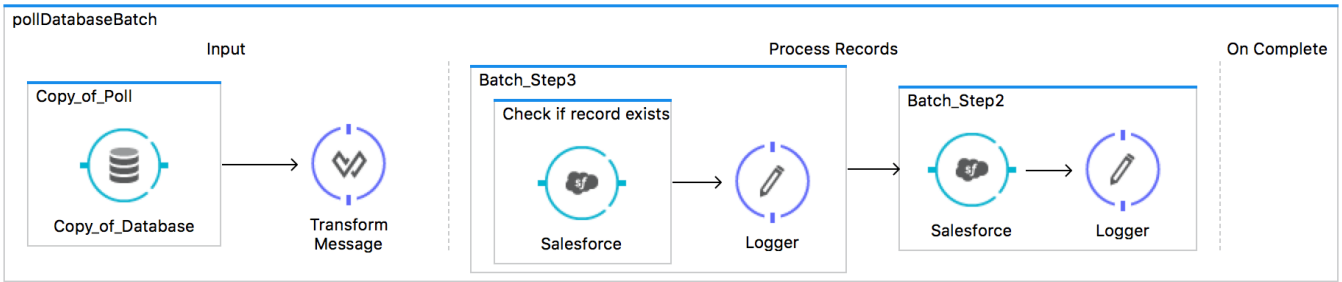
49. Look at the input tab of the metadata explorer; you should see a problem letting you know that payload is a Map in this step but the outbound Salesforce endpoint is expecting a List of Account objects.
50. Drill-down in both values and look at the fields.



Transform the objects into the Account objects that the endpoint expects

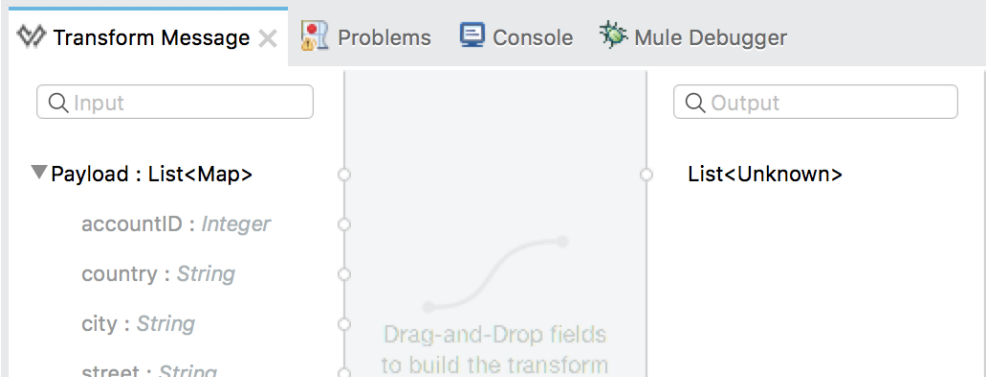
51. Add a Transform Message component after the poll scope in the input phase.

    Note: You want to transform all the records at once in the input phase – not one by one in the processing phase.



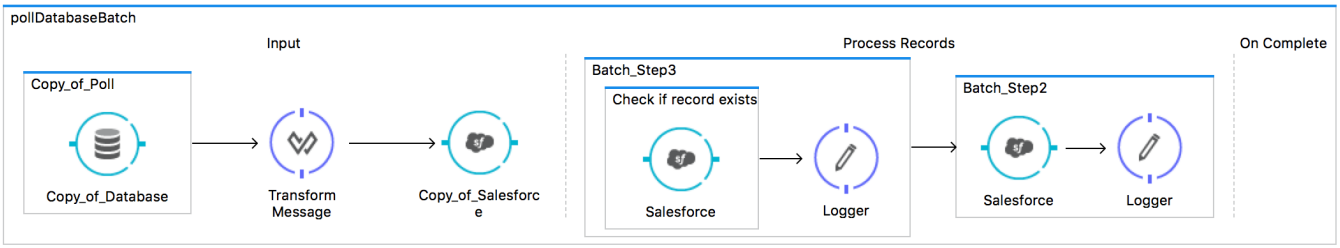
52. In the Transform Message Properties view, look at the input section and see the input has associated metadata.

53. Look at the output section; it has no metadata and no way to add any.

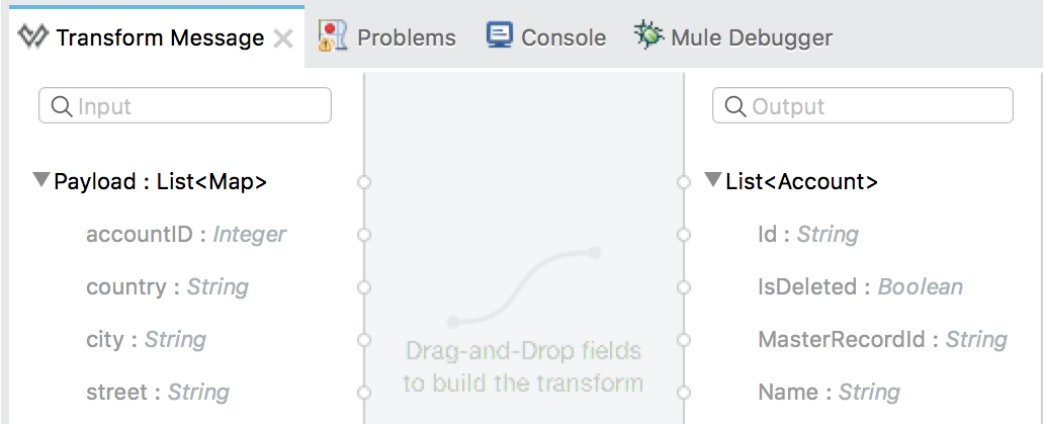


54. Copy the Salesforce endpoint in the second batch step and paste it after the Transform Message component in the batch input phase.

    Note: You are temporarily adding this Salesforce endpoint after the database endpoint so DataSense will work to create the mappings. You could instead add the DataWeave transformation to the process records phase, but this would add additional overhead.

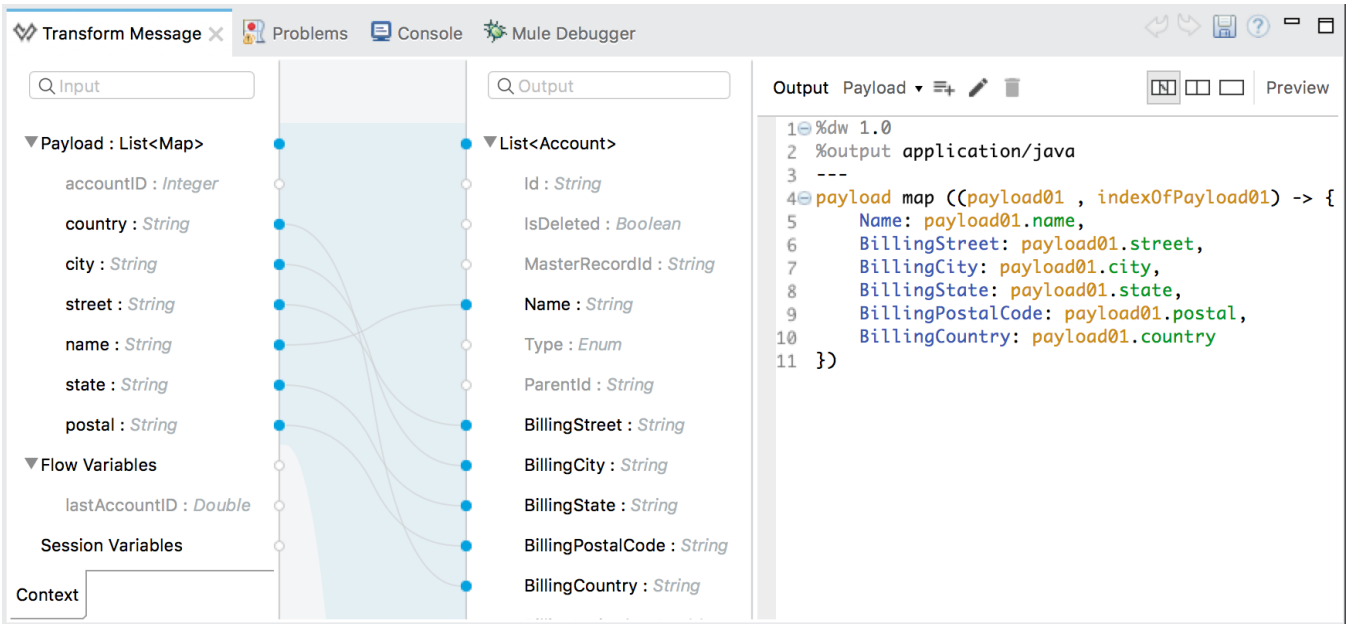


55. Look at the output section of the Transform Message Properties view again; you should now see metadata for a List of Account objects.

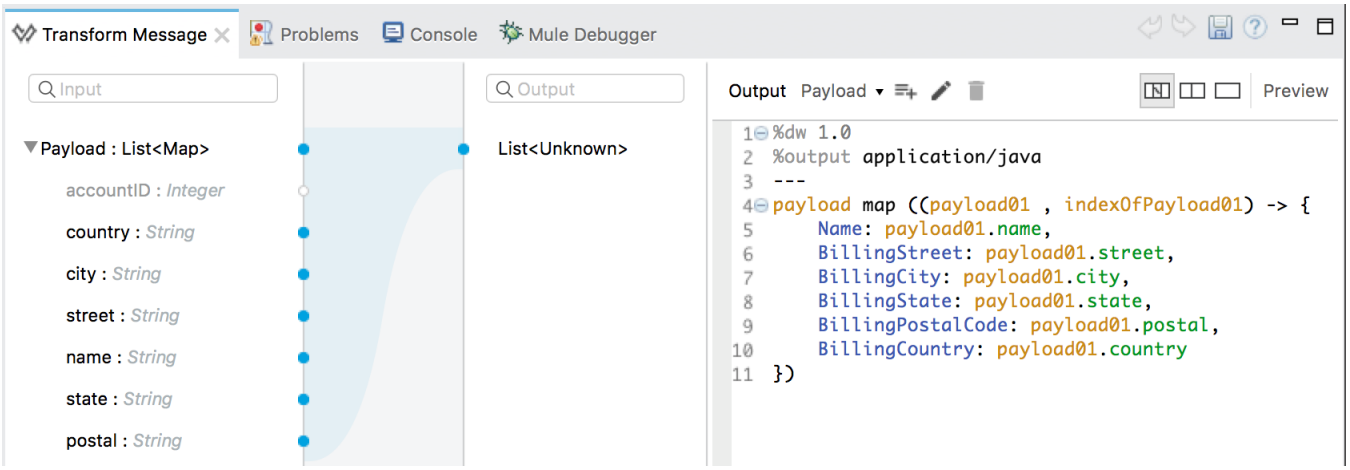


56. Drag properties from the input section to the output section to make the following mappings:

- country to BillingCountry
- city to BillingCity
- street to BillingStreet
- name to Name
- state to BillingState
- postal to BillingPostalCode

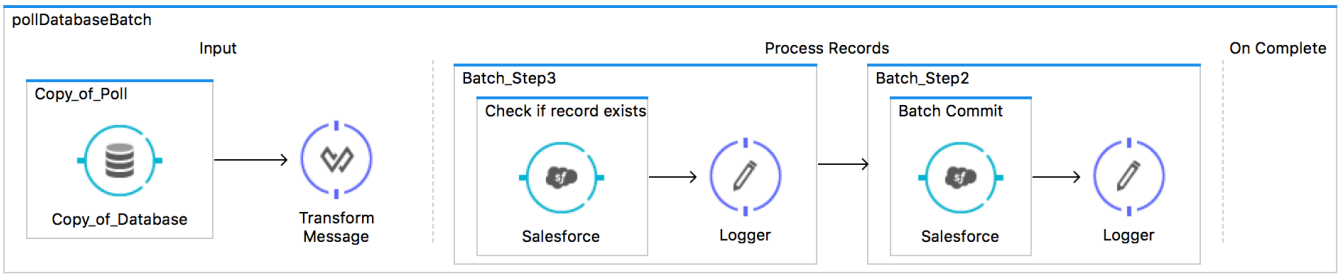


57. Delete the Salesforce endpoint in the input phase.
58. Look at the mapping in the Transform Message Properties view; the graphical mapping is gone but the DataWeave expression is still there.

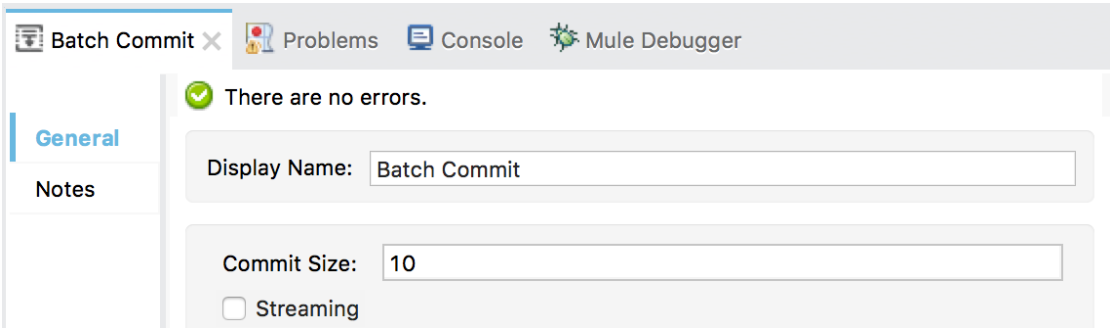


Create the List that the endpoint is expecting

59. Return to the Properties view for the Salesforce endpoint in the second batch step.
60. Look at the input tab of the metadata explorer; you should see that the outbound Salesforce endpoint is expecting a List of Account objects.
61. Drag a Batch Commit scope from the Mule Palette and drop it in the second batch step.
62. Add the Salesforce endpoint to it.



63. In the Batch Commit properties view, set the commit size to 10.

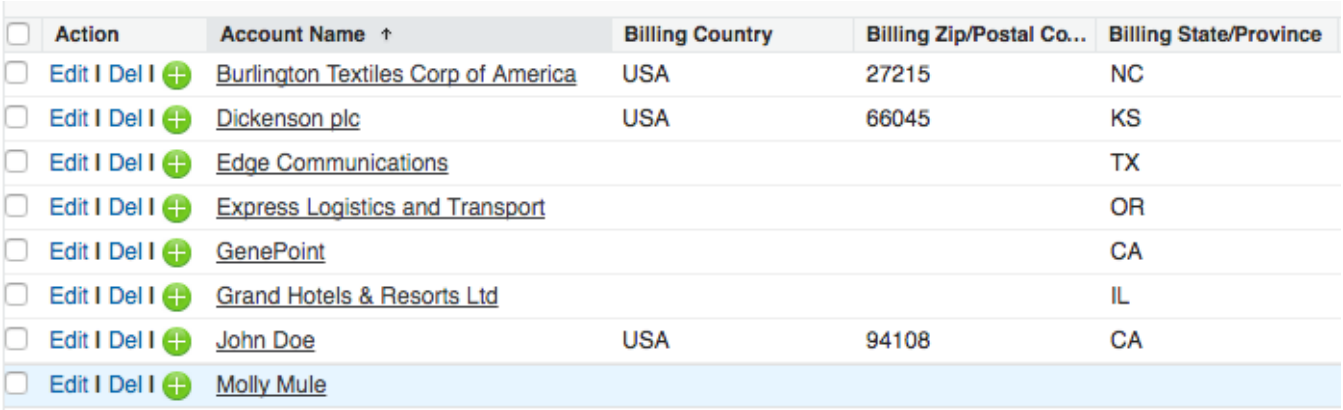


Test the application

64. Run the project and clear the application data.
65. Check the console and make sure you see your new records processed.

66. Return to Salesforce in a web browser and look at the accounts; you should see the records from the legacy MySQL database are now in Salesforce.

    Note: You could also check for the records by making a request to http://localhost:8081/sfdc.



Action	Account Name ↑	Billing Country	Billing Zip/Postal Co...	Billing State/Province
Edit | Del |	Burlington Textiles Corp of America	USA	27215	NC
Edit | Del |	Dickenson plc	USA	66045	KS
Edit | Del |	Edge Communications			TX
Edit | Del |	Express Logistics and Transport			OR
Edit | Del |	GenePoint			CA
Edit | Del |	Grand Hotels & Resorts Ltd			IL
Edit | Del |	John Doe	USA	94108	CA
Edit | Del |	Molly Mule			

67. Run the application again but do not clear the application data; no records should be processed.
68. Return to salesforce.com and locate your new record(s); they should have been inserted only once.
69. Return to Anypoint Studio and stop the project.