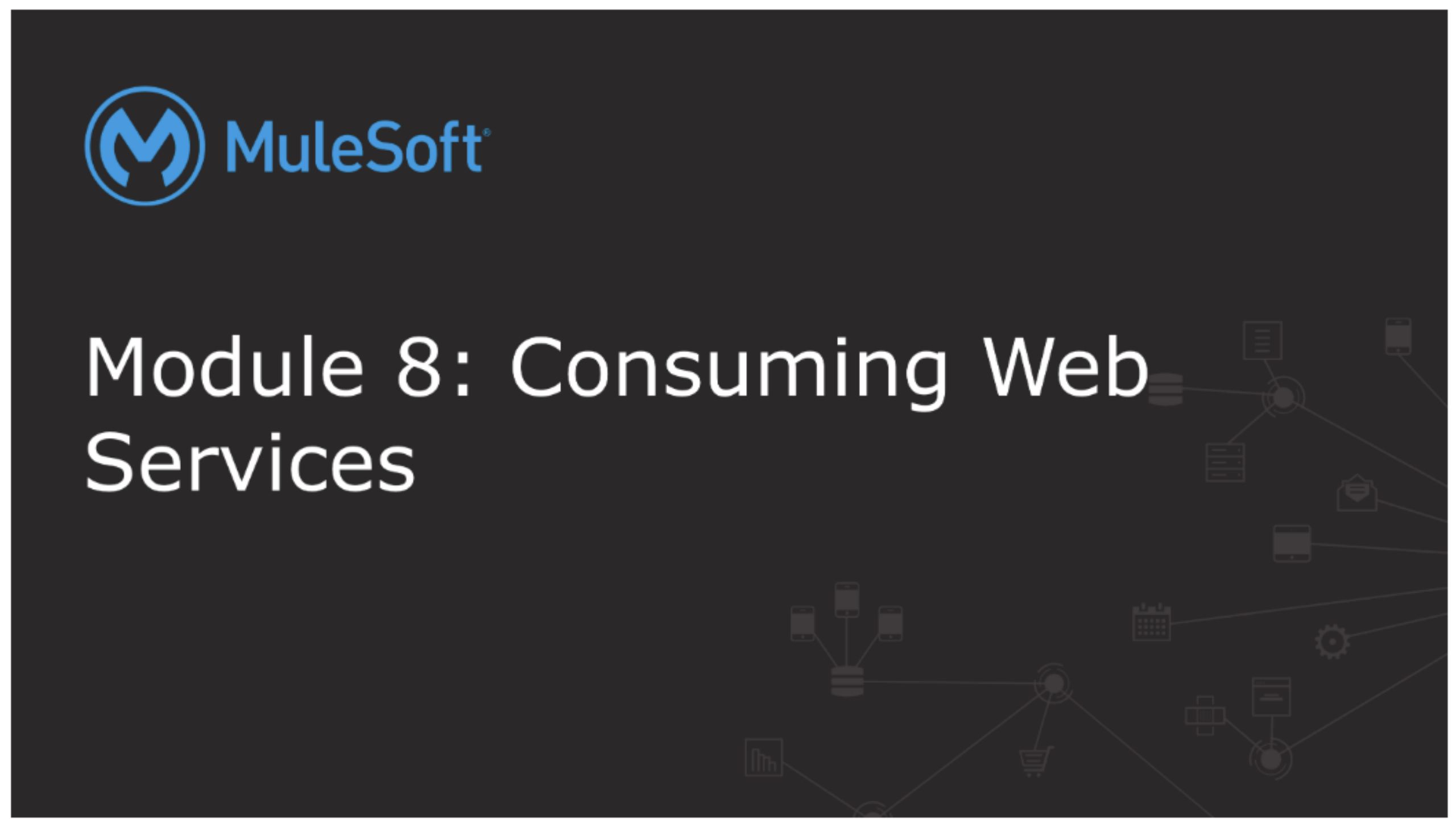

MuleSoft
Module 8: Consuming Web Services

Goal

Call an operation of a connector in Exchange

Call a RESTful web service

Call a SOAP web service

Flight.java

```
1  package com.mulesoft.training;
2
3  import java.util.Comparator;
4
5  public class Flight implements
6
7      String flightCode;
8      String origination;
9      int availableSeats;
10     String departureDate;
11     String airlineName;
12     String destination;
13     double price;
14     String planeType;
```

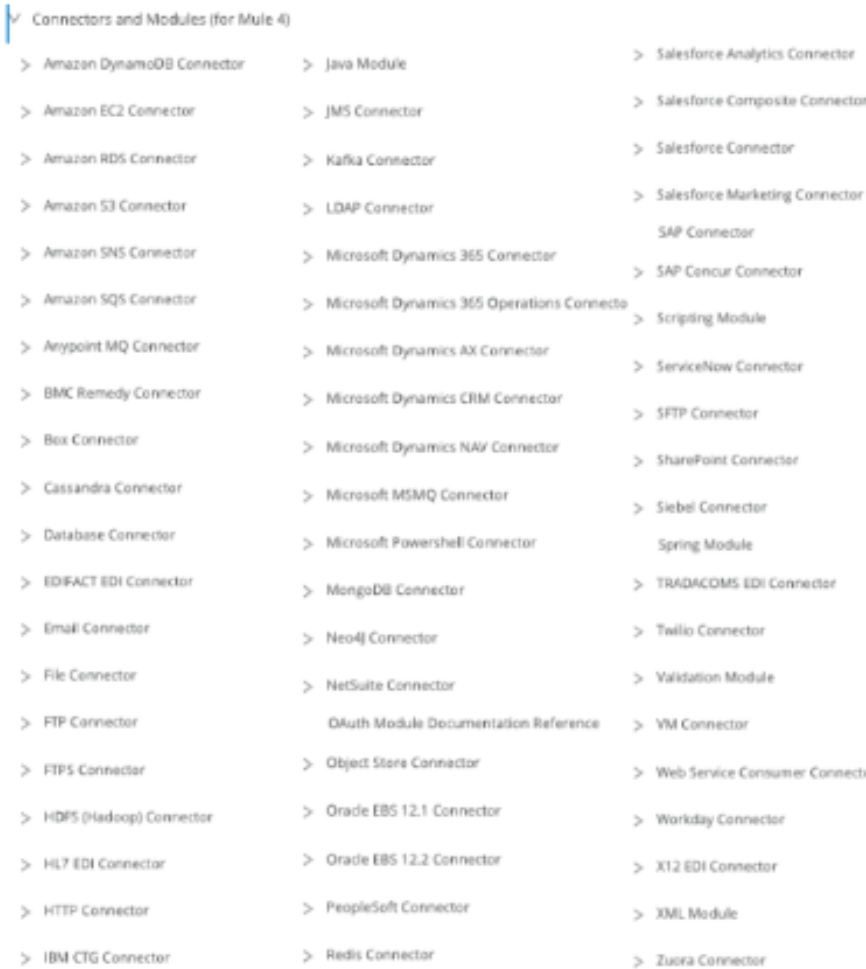
At the end of this module, you should be able to

- Consume web services that have a connector in Anypoint Exchange
- Consume RESTful web services
- Consume SOAP web services
- Pass parameters to SOAP web services using the Transform Message component
- Transform data from multiple services to a canonical format

Consuming web services that have a connector in Exchange

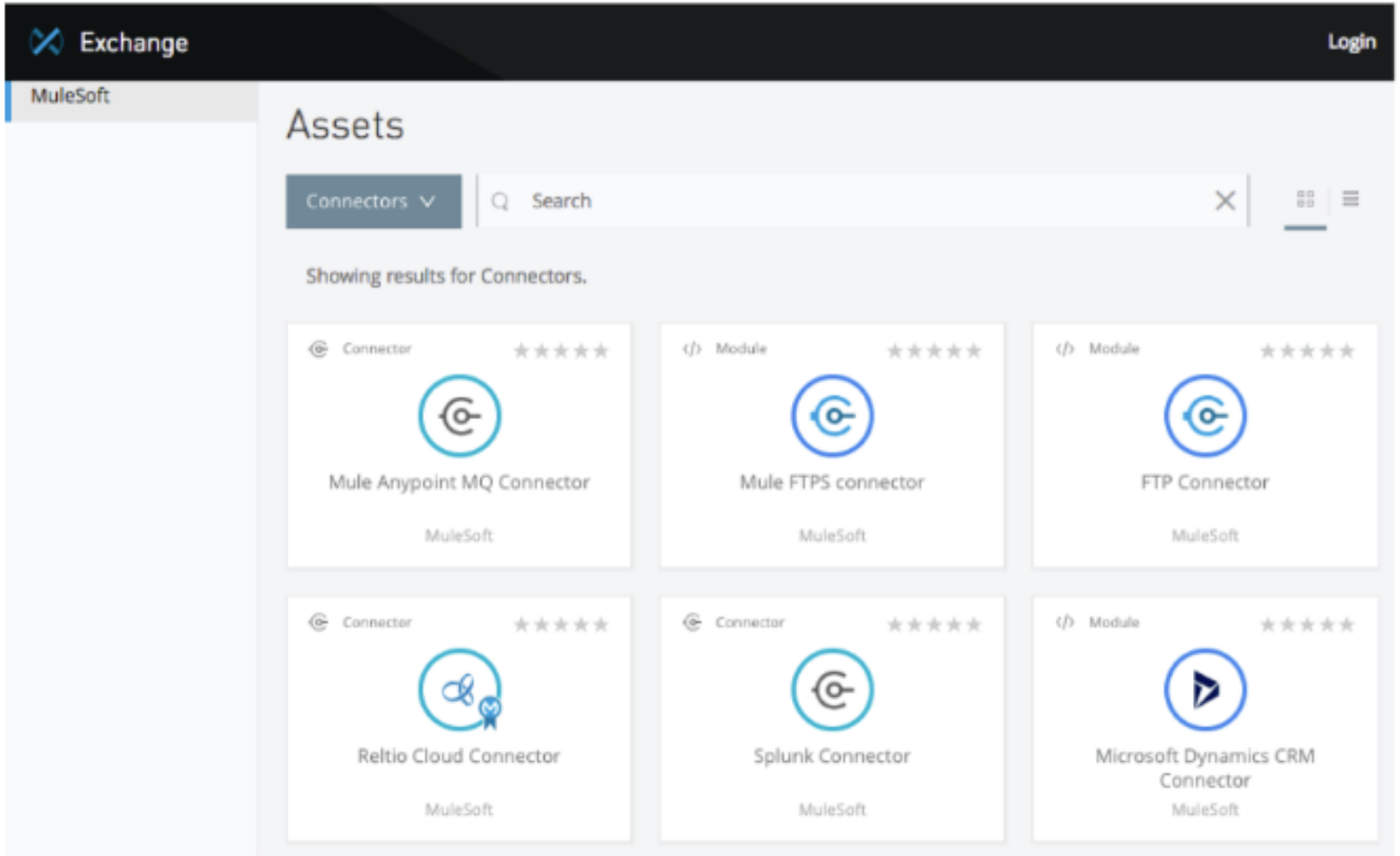
Connectors and modules

- **Modules** are extensions to the Mule runtime that you can use when building a Mule app
  - HTTP, Database, Salesforce, SAP, Slack, Validation, Java, and many more
- **Connectors** are modules that connect to an external server
  - HTTP, Database, Salesforce, SAP, Slack
- For module reference
  - https://docs.mulesoft.com/connectors/



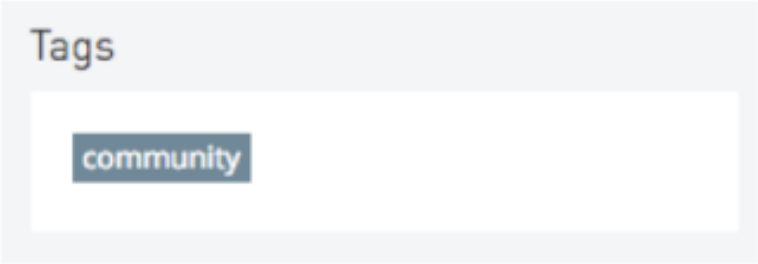
Connectors in Anypoint Exchange

- Many connectors in Exchange package a much easier way to make calls to APIs



Connector types specify creator and support level

- The type of selector is specified in its tags on Exchange



	Premium	Select	MuleSoft Certified	Community
Additional cost	x			
Updated APIs	x	x		
Fully tested	x	x		
MuleSoft Support	Tier 1-3	Tier 1-3	Tier 1 (From developer: T2/T3)	Tier 1
Connector examples	HL7 SAP Siebel	Salesforce Workday	AS/400 Oracle JD Edwards Microsoft Azure Storage	LinkedIn Slack

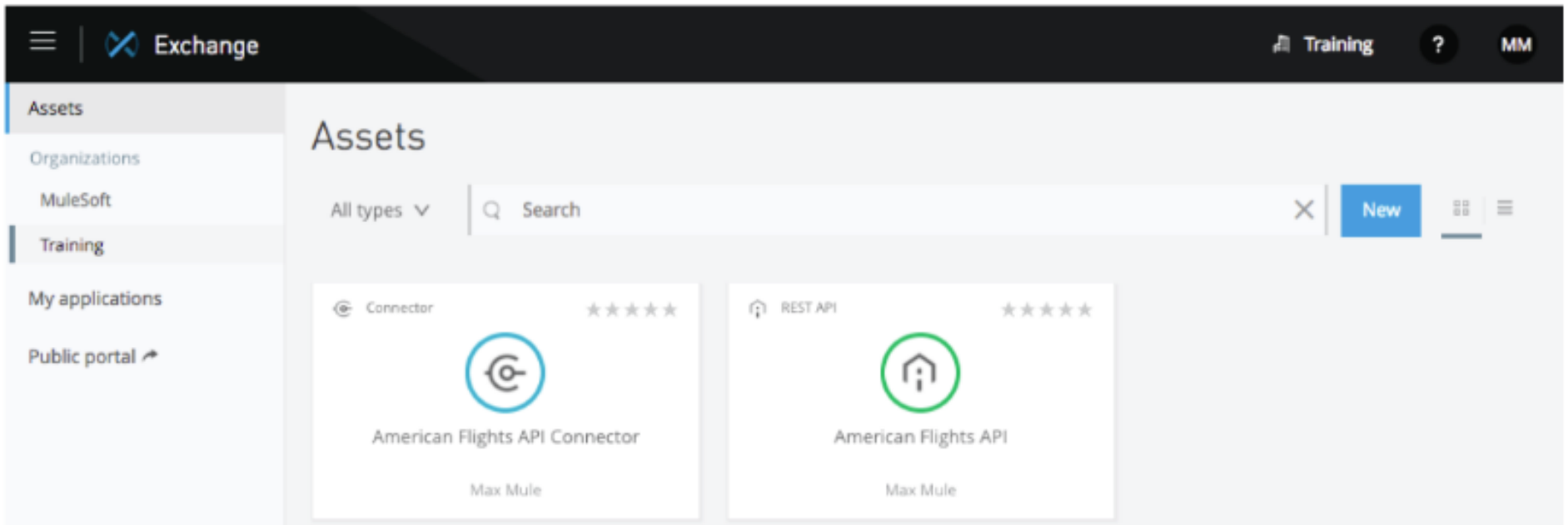
Connector support levels

- Tier 1
  - MuleSoft will isolate the problem and diagnose it
- Tier 2
  - MuleSoft will find a workaround
- Tier 3
  - MuleSoft will fix the code

	Premium	Select	MuleSoft Certified	Community
Not included in Platform license	x			
Tier 2-3 Support	x	x		
Tier 1 Support	x	x	x	x

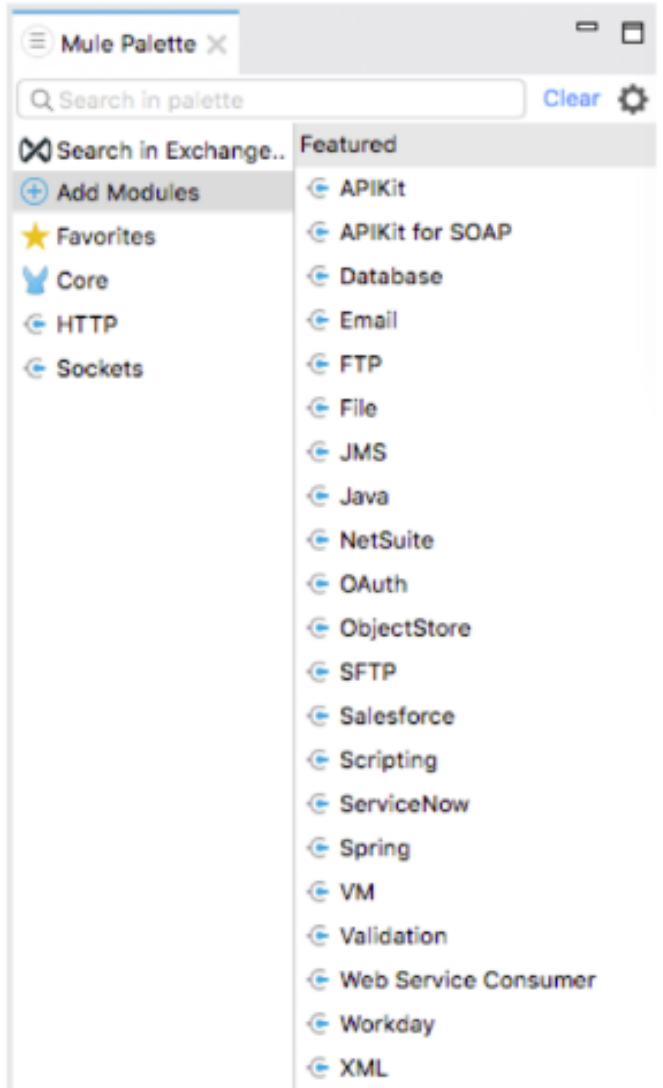
REST Connectors in Anypoint Exchange

- REST Connect converts a RAML 1.0 API specification added to Exchange to a connector
  - You did this in Module 4
- You can use the connector in Anypoint Studio or flow designer



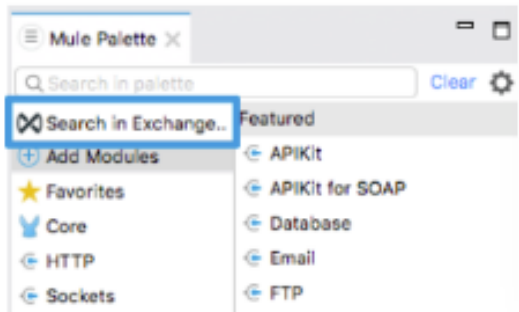
Connectors in Anypoint Studio

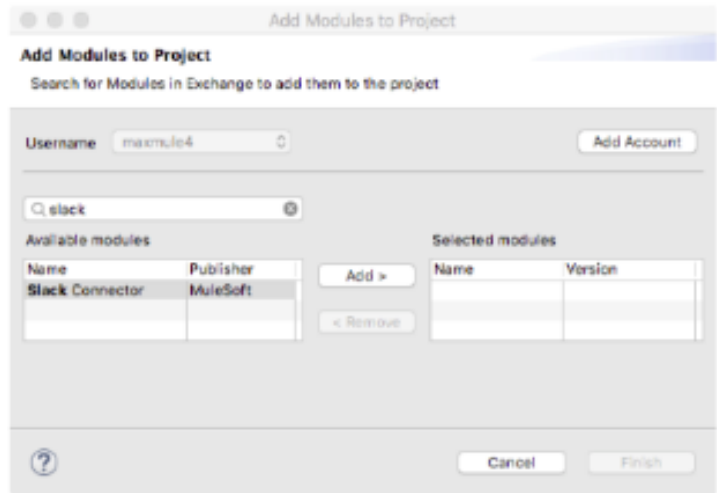
- Some modules are **pre-installed** in Studio
  - HTTP, Database, Salesforce, Validation, Java
- Some modules are **not pre-installed** in Studio
  - SAP, Slack

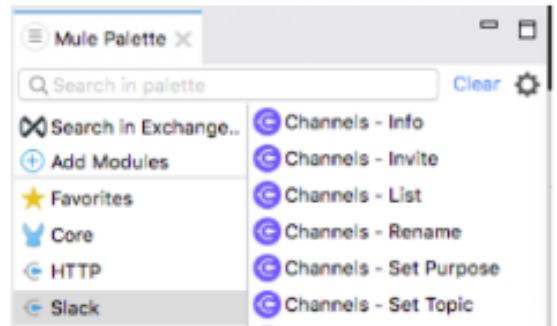


Adding connectors from Exchange

- If connectors are not pre-installed in Anypoint Studio, you can search Exchange and add them to a project

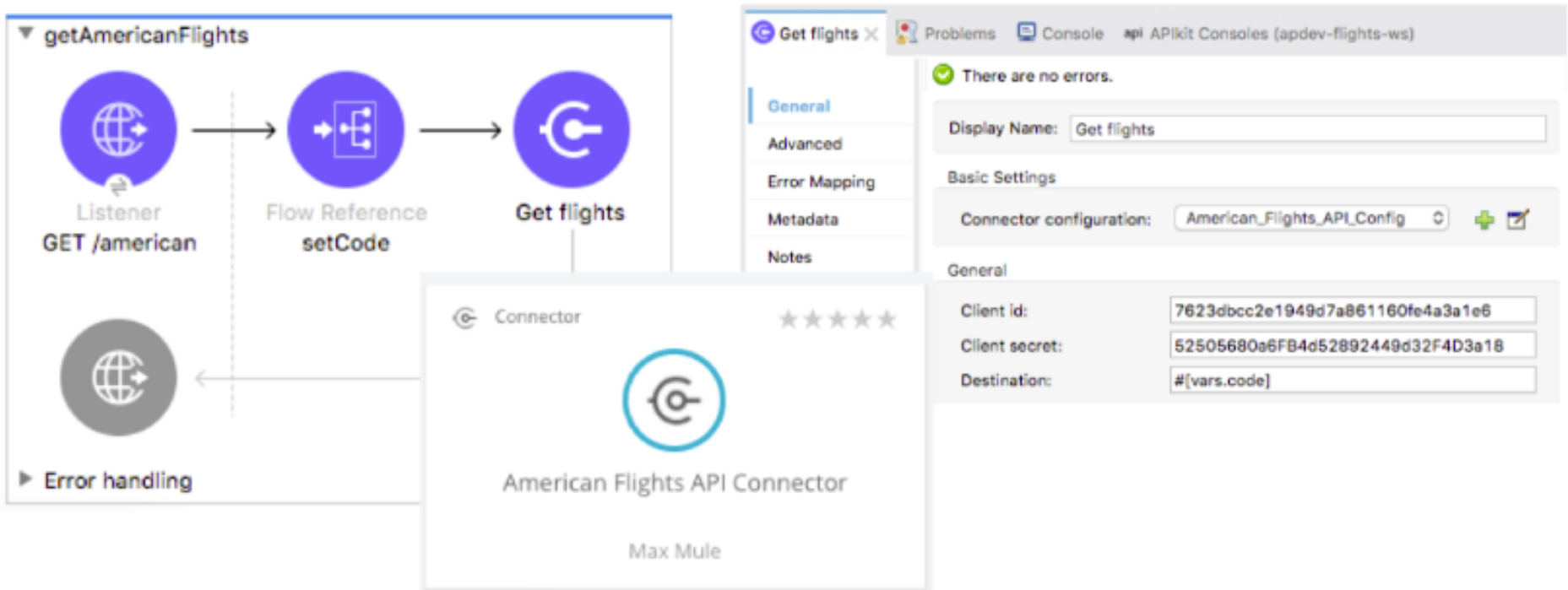






Walkthrough 8-1: Consume a RESTful web service that has a connector in Exchange

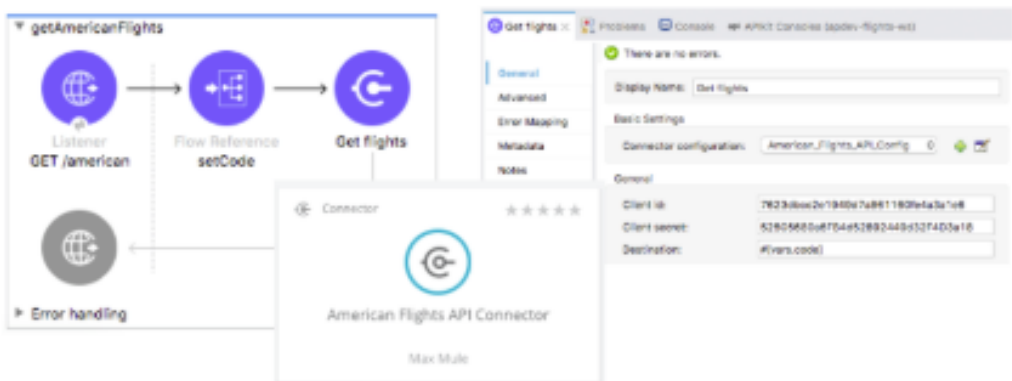
- Create a new flow to call the American RESTful web service
- Add a REST connector from Exchange to an Anypoint Studio project
- Configure and use a REST connector to make a call to a web service
- Dynamically set a query parameter for a web service call



Walkthrough 8-1: Consume a RESTful web service that has a connector in Exchange

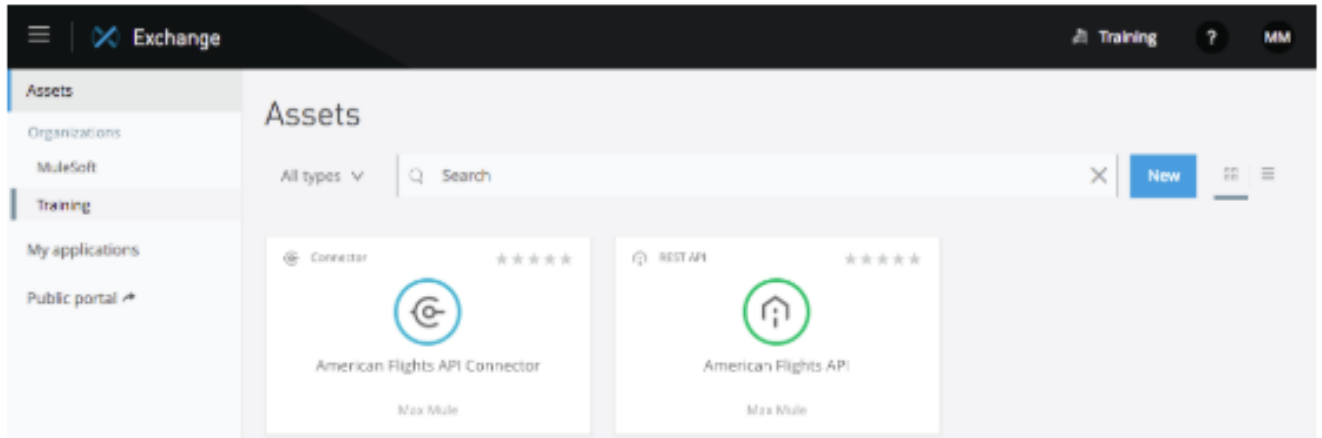
In this walkthrough, you consume the American flights RESTful web service that you built that has an API and a connector in Exchange. You will:

- Create a new flow to call the American RESTful web service.
- Add a REST connector from Exchange to an Anypoint Studio project.
- Configure and use a REST connector to make a call to a web service.
- Dynamically set a query parameter for a web service call.

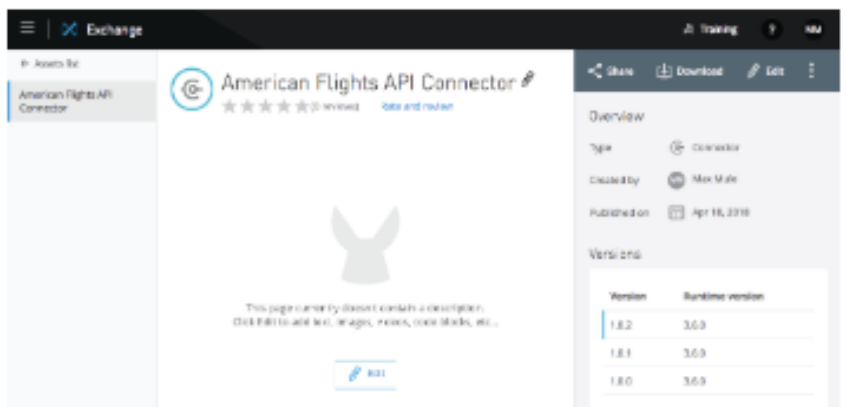


Review the auto-generated REST connector in Exchange

1. Return to Exchange in Anypoint Platform.
2. Locate the American Flights API Connector that was auto-generated for your American flights API.

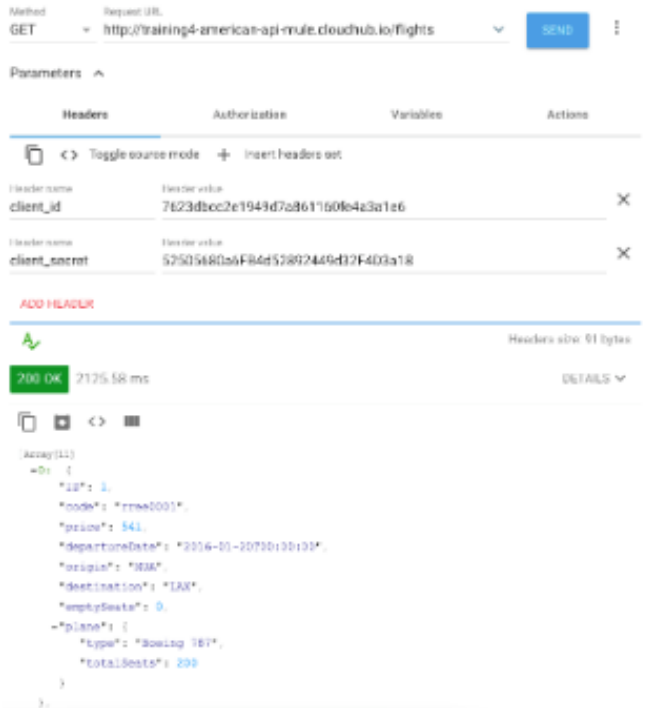


3. Select the connector and review its information.



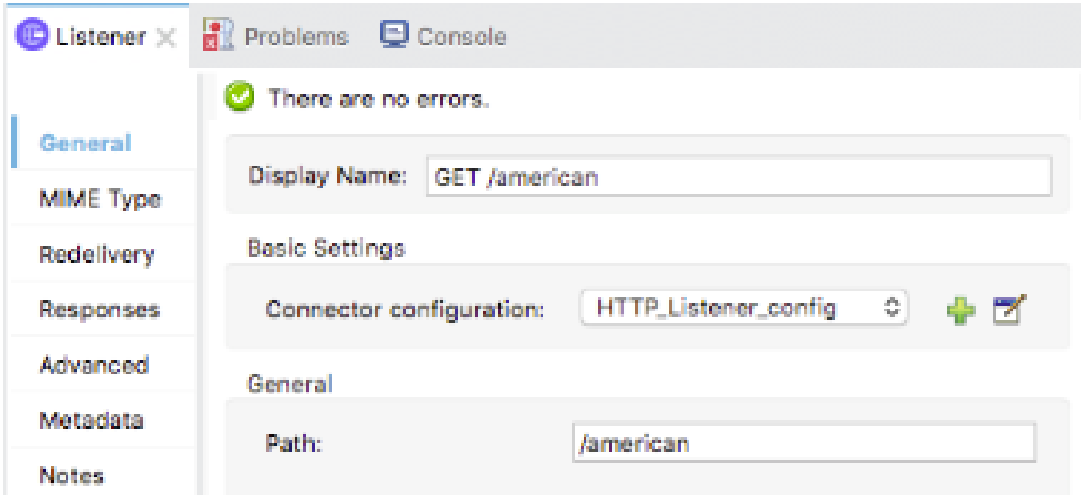
Make a request to the web service

4. Return to Advanced REST Client.
5. Select the first tab – the one with the request to your American Flights API http://training4-american-api-{lastname}.cloudhub.io/flights and that passes a client_id and client_secret as headers.
6. Send the request; you should still see JSON data for the American flights as a response.



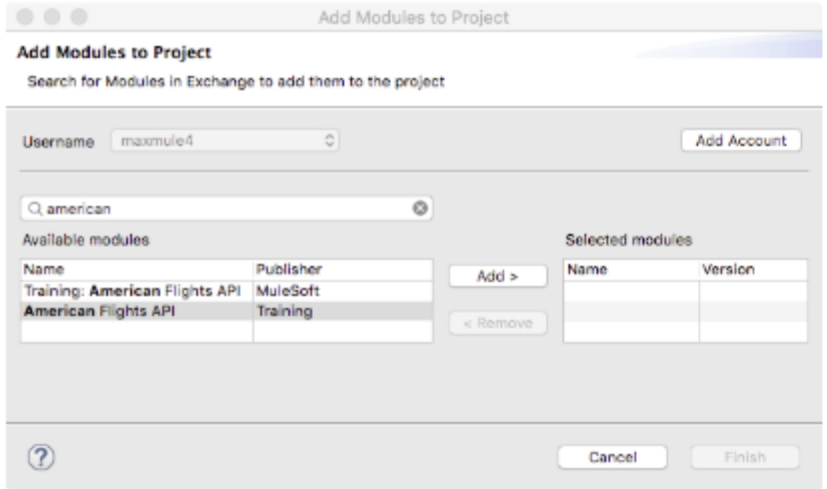
Add a new flow with an HTTP Listener

7. Return to the apdev-flights-ws project in Anypoint Studio.
8. Open implementation.xml
9. Drag out an HTTP Listener and drop it in the canvas.
10. Rename the flow to getAmericanFlights.
11. In the Listener properties view, set the display name to GET /american.
12. Set the connector configuration to the existing HTTP_Listener_config.
13. Set the path to /american.
14. Set the allowed methods to GET.



Add the American Flights API module to Anypoint Studio

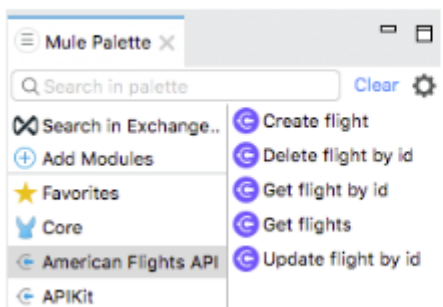
15. In the Mule Palette, select Search in Exchange.
16. In the Add Modules to Project dialog box, enter American in the search field.



17. Select your American Flights API (**not** the Training: American Flights API) and click Add.

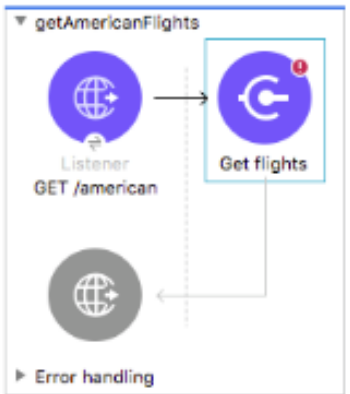
    Note: If you did not successfully create the American Flights API and connector in the first part of the course, you can use the pre-built Training: American Flights API connector. This connector does not require client authentication.

18. Click Finish.
19. Wait for the module to be downloaded.
20. Select the new American Flights API module in the Mule Palette.

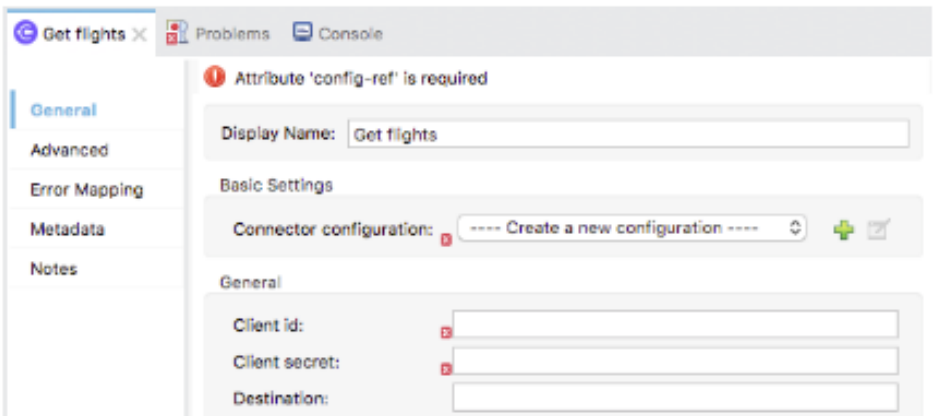


Add a Get all flights operation

21. Select the Get flights operation in the right side of the Mule Palette and drag and drop it in the process section of the flow.

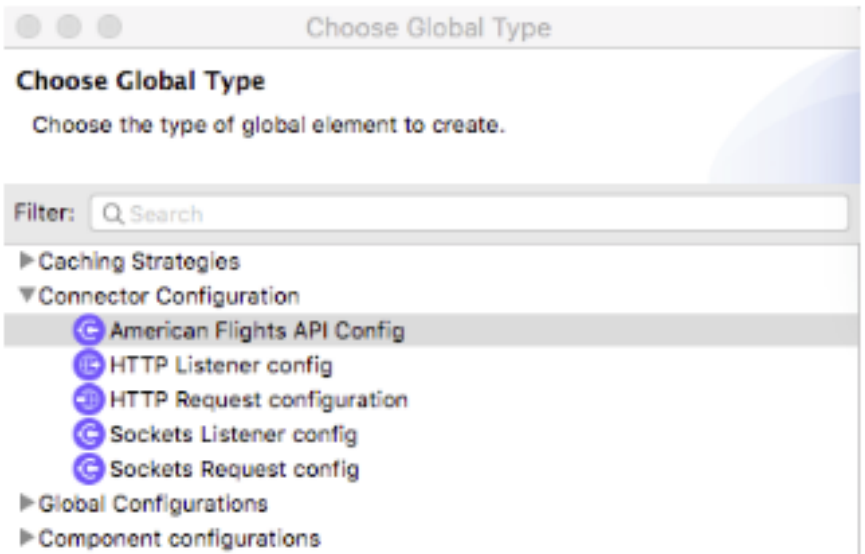


22. Select the Get all flights operation in the canvas and in its properties view, see that you need to need to create a new configuration and enter a client_id and client_secret.

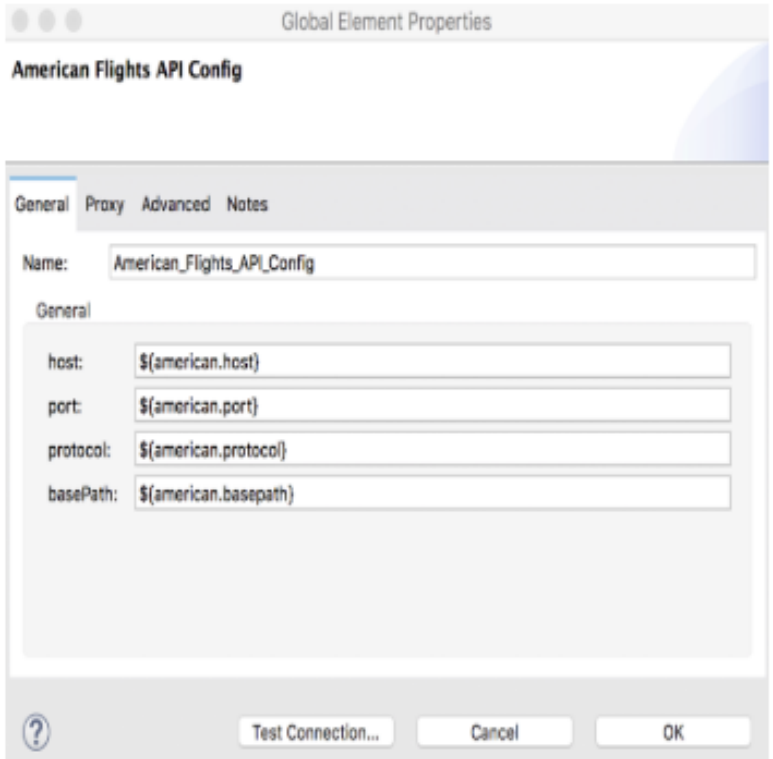


Configure the American Flights API connector

23. Return to global.xml.
24. In the Global Elements view, click Create.
25. In the Choose Global Type dialog box, select Connector Configuration > American Flights API Config and click OK.



26. In the Global Element Properties dialog box, set each field to a corresponding property placeholder (that you will define and set next):
    - host: ${american.host}
    - port: ${american.port}
    - basePath: ${american.basepath}
    - protocol: ${american.protocol}



27. Click OK.
28. Return to the course snippets.txt file and copy the text for the American RESTful web service properties.

29. Return to config.yaml in src/main/resources and paste the code at the end of the file.

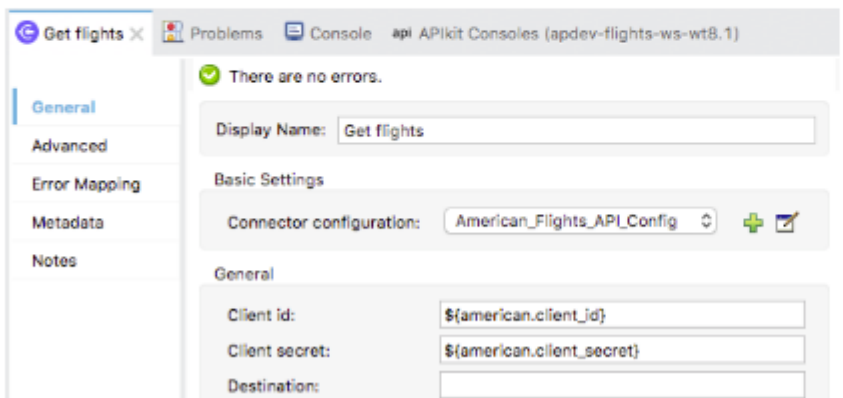
implementation *config.yaml

```yaml
http:
  port: "8081"

american:
  host: "training4-american-api-{lastname}.cloudhub.io"
  port: "80"
  basepath: "/"
  protocol: "HTTP"
  client_id:
  client_secret:
```

30. In the american.host property, replace {lastname} with your application identifier.
31. Return to Advanced REST Client and copy the value for your client_id.
32. Return to config.yaml and paste the value.
33. Repeat for your client_secret.

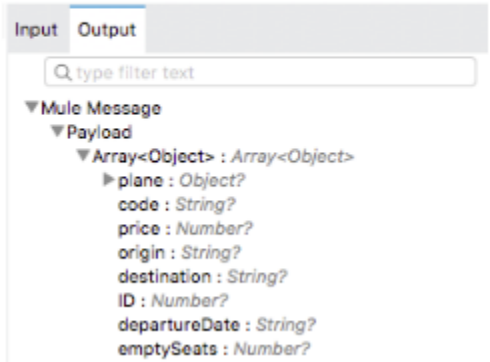
Configure the Get flights operation

34. Return to implementation.xml.
35. In the Get flights properties view, set the Connector configuration to American_Flights_API_Config.
36. Set the client id to the ${american.client_id} property.
37. Leave the destination field blank.



Review metadata associated with the operation

38. Select the output tab in the DataSense Explorer and expand Payload.



Test the application

39. Run the project.
40. In Advanced REST Client, return to the tab with the localhost requests.
41. Change the URL to make a request to http://localhost:8081/american; you should get all the flights.

Review the specification for the API you are building

42. Open mua-flights-api.raml in src/main/resources/api.
43. Review the optional query parameters.

```
/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
```

44. Locate the return type for the get method of the /flights resource.

```
responses:
  200:
    body:
      application/json:
        type: Flight[]
        example: !include examples/FlightsExample.raml
```

45. Open FlightsExample.raml in src/main/resources/api/examples and review its structure.

```
#%RAML 1.0 NamedExample
value:
  -
    airline: United
    flightCode: ER38sd
    fromAirportCode: LAX
    toAirportCode: SFO
    departureDate: May 21, 2016
    emptySeats: 0
    totalSeats: 200
    price: 199
    planeType: Boeing 737
  -
    airline: Delta
    flightCode: ER0945
    fromAirportCode: PDX
    toAirportCode: CLE
    departureDate: June 1, 2016
    emptySeats: 24
    totalSeats: 350
    price: 450
    planeType: Boeing 747
```

Create a variable to set the destination airport code

46. Return to implementation.xml.
47. Drag a Sub Flow scope from the Mule Palette and drop it at the top of the canvas.
48. Change the name of the flow to setCode.
49. Drag a Set Variable transformer from the Mule Palette and drop it in the setCode subflow.

50. In the Set Variable properties view, set the display name and name to code.
51. Set the value to a query parameter called code and give it a default value of SFO if no query parameter is passed to the flow.

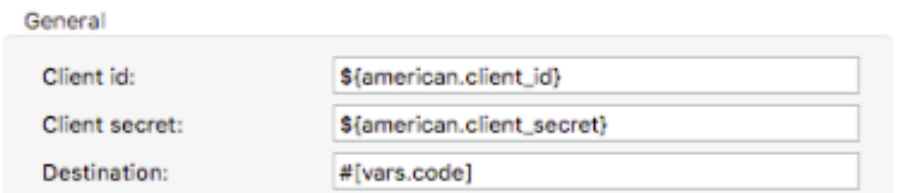
```
#[attributes.queryParams.code default 'SFO']
```

Dynamically set a query parameter for the web service call

52. Drag a Flow Reference component from the Mule Palette and drop it after the GET /american Listener in getAmericanFlights.
53. In the Flow Reference properties view, set the flow name to setCode.

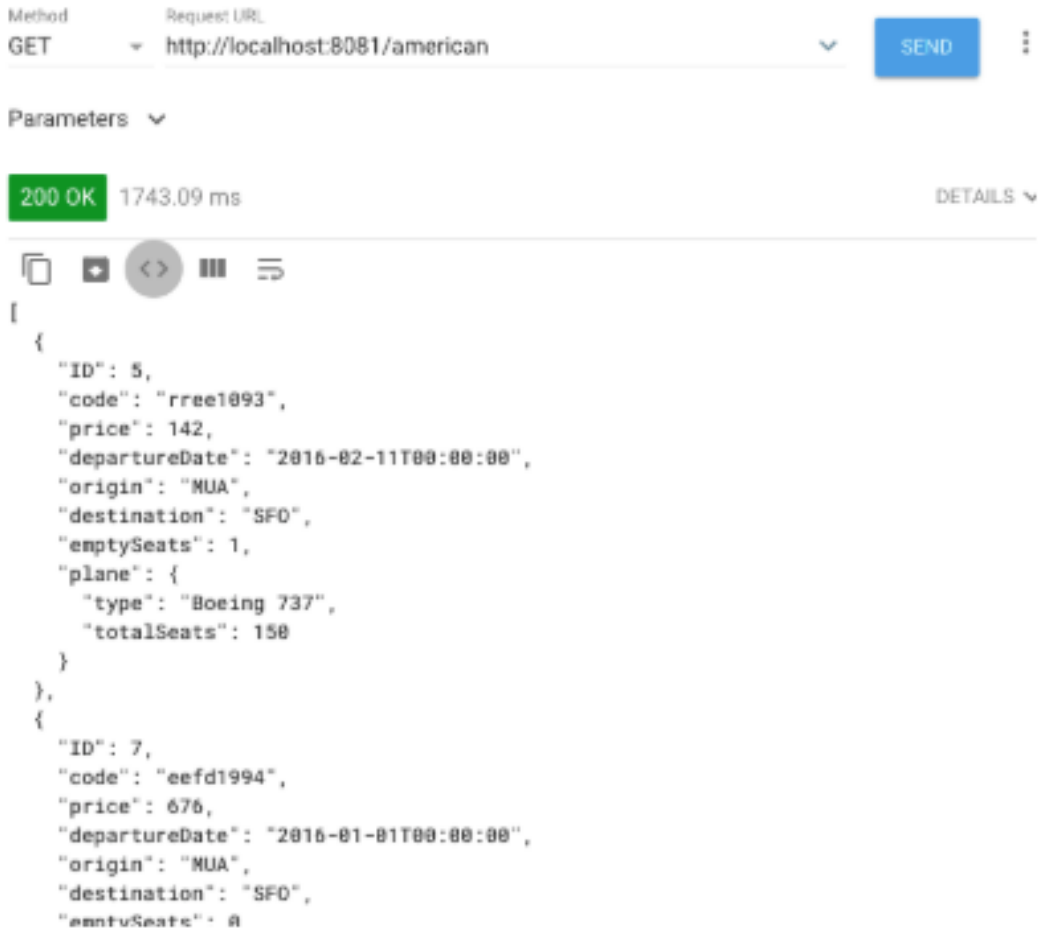
54. In the Get flights properties view, set the value of the destination parameter to the value of the variable containing the airport code.

```
#[vars.code]
```



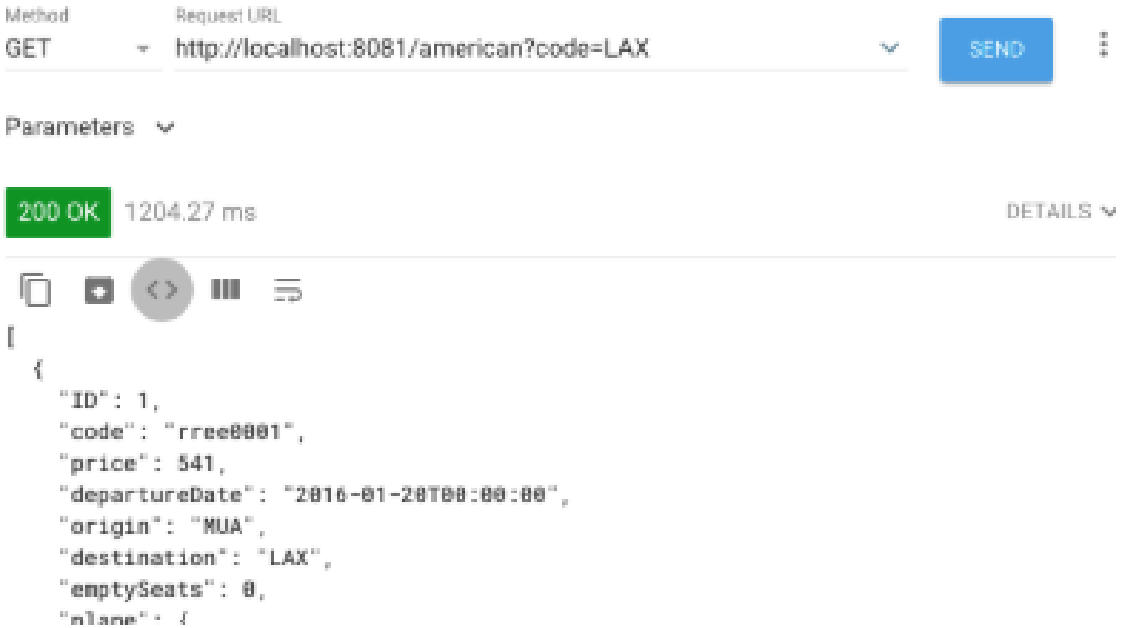
Test the application

55. Save the file to redeploy the application.
56. In Advanced REST Client, send the same request; this time you should only get flights to SFO.




```
[
  {
    "ID": 5,
    "code": "rree1093",
    "price": 142,
    "departureDate": "2016-02-11T00:00:00",
    "origin": "MUA",
    "destination": "SFO",
    "emptySeats": 1,
    "plane": {
      "type": "Boeing 737",
      "totalSeats": 150
    }
  },
  {
    "ID": 7,
    "code": "eefd1994",
    "price": 676,
    "departureDate": "2016-01-01T00:00:00",
    "origin": "MUA",
    "destination": "SFO",
    "emptySeats": 0
```

57. Add query parameter called code equal to LAX and make the request; you should now only see flights to LAX.



58. Examine the data structure of the JSON response.

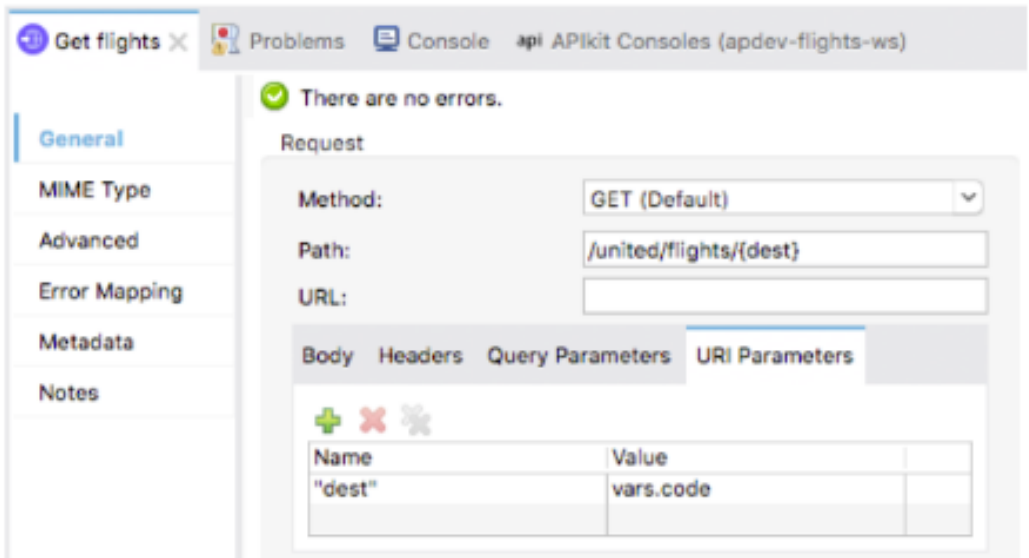
    Note: You will transform the data to the format specified by the mua-flights-api in a later walkthrough.

Consuming RESTful web services

Consuming RESTful web services

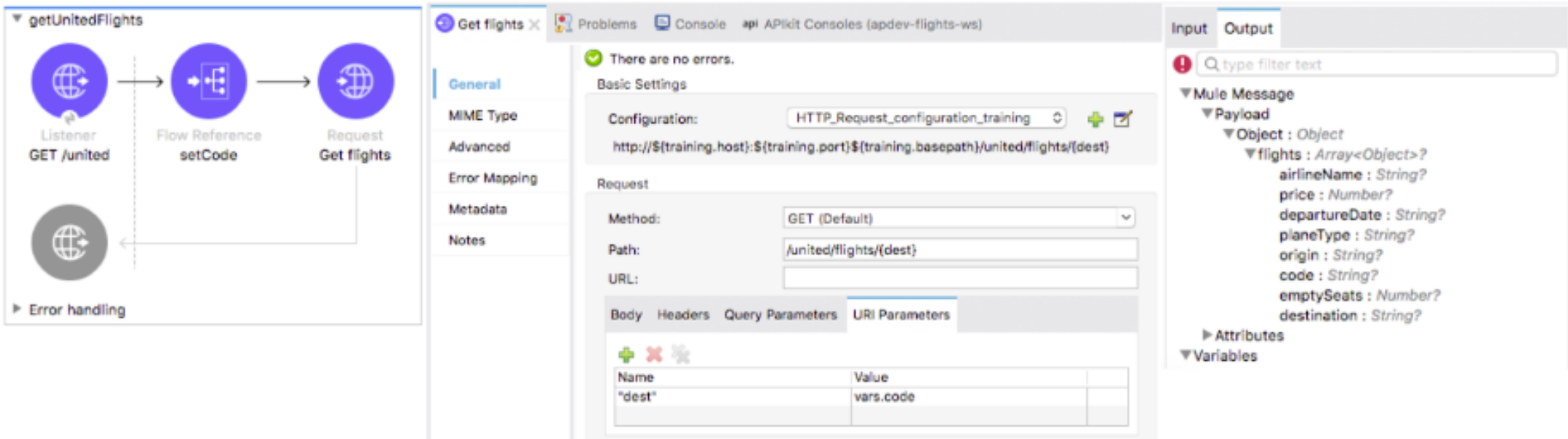
- First check and see if there is an existing Anypoint Connector in Studio or Exchange to connect to the service provider

- If there is not, use the **HTTP** connector and its **Request** operation
  - Configure the operation and/or global element configuration
  - Specify any headers, query parameters, or URI parameters to pass to the call



Walkthrough 8-2: Consume a RESTful web service

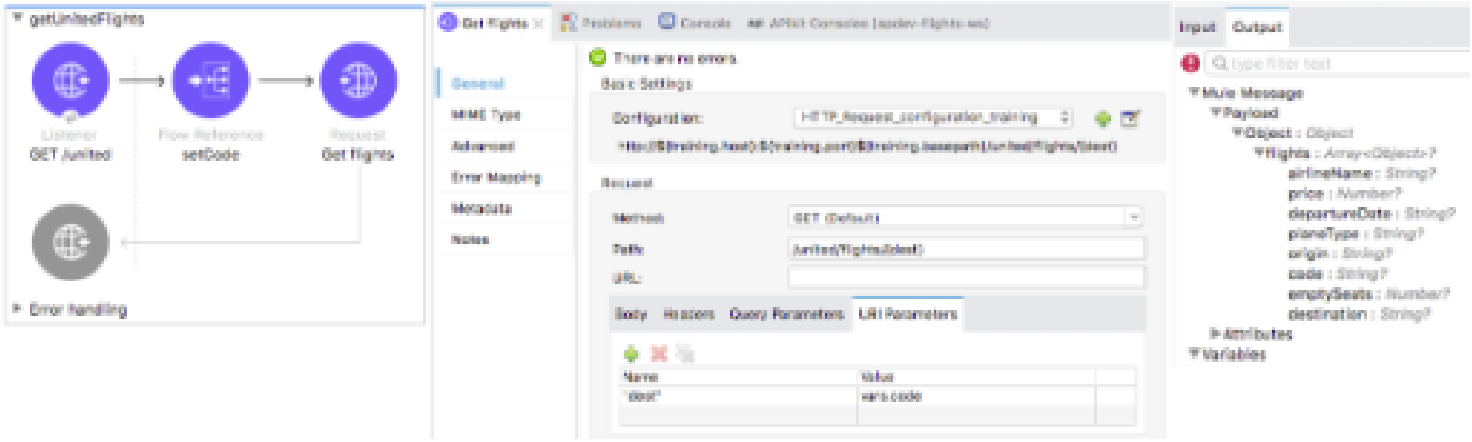
- Create a new flow to call the United RESTful web service
- Use the HTTP Request operation to call a RESTful web service
- Dynamically set a URI parameter for a web service call
- Add metadata for an HTTP Request operation's response



Walkthrough 8-2: Consume a RESTful web service

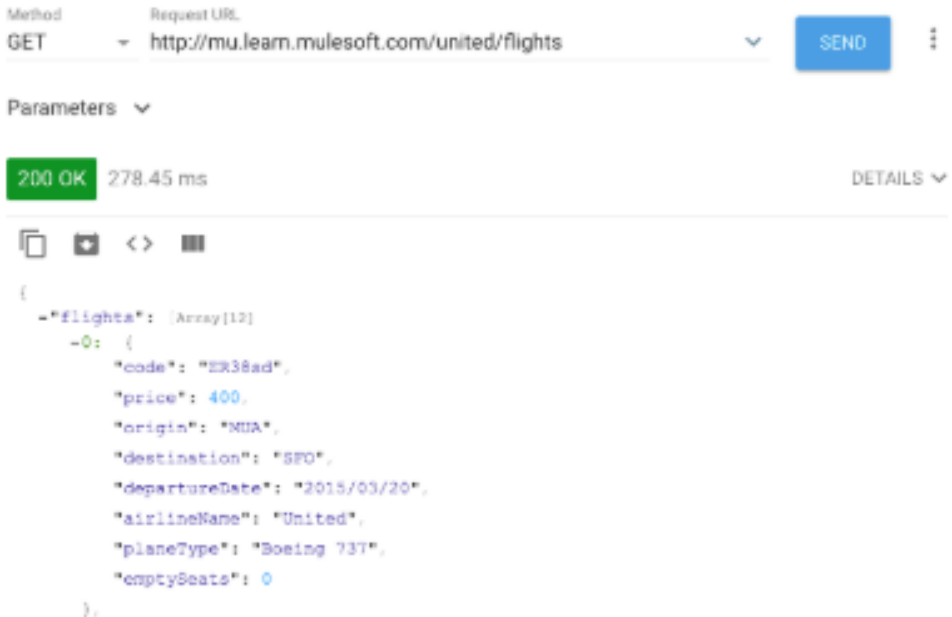
In this walkthrough, you consume the United RESTful web service that does not have an API in Exchange. You will:

- Create a new flow to call the United RESTful web service.
- Use the HTTP Request operation to call a RESTful web service.
- Dynamically set a URI parameter for a web service call.
- Add metadata for an HTTP Request operation's response.



Make a request to the United web service

1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Advanced REST Client, make a new tab and make a GET request to this URL.
4. Review the structure of the JSON response.



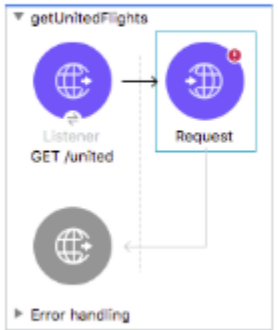
5. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.
6. In the URL field, add the destination CLE as a URI parameter: http://mu.learn.mulesoft.com/united/flights/CLE.
7. Send the request; you should now only see flights to CLE.

Add a new flow with an HTTP Listener operation

8. Return to implementation.xml in Anypoint Studio.
9. Drag out an HTTP Listener from the Mule Palette and drop it at the bottom of the canvas.
10. Change the name of the flow to getUnitedFlights.
11. In the Listener properties view, set the display name to GET /united.
12. Set the connector configuration to the existing HTTP_Listener_config.
13. Set the path to /united.
14. Set the allowed methods to GET.

Add an HTTP Request operation

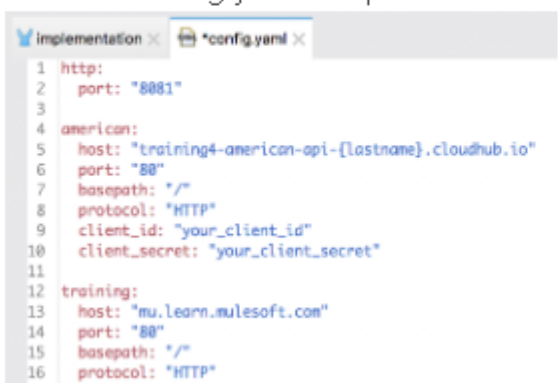
15. Drag out an HTTP Request from the Mule Palette and drop it into the process section of getUnitedFlights.



16. In the Request properties view, set the display name to Get flights.

Create an HTTP Request configuration

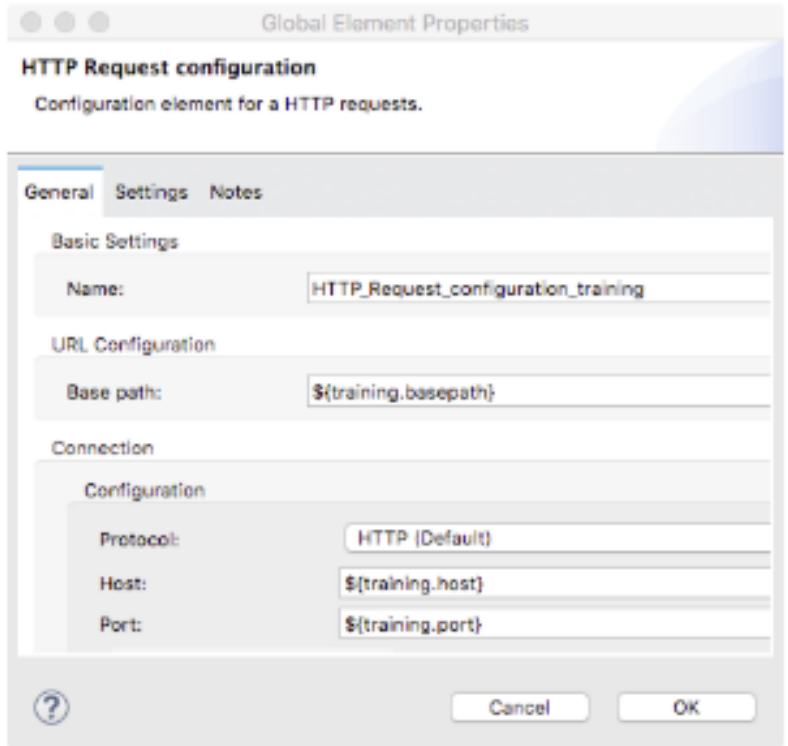
17. Return to the course snippets.txt file and copy the text for the Training web service properties.
18. Return to config.yaml and paste the code at the end of the file.



19. Return to the Global Elements view in global.xml.

20. Create a new HTTP Request configuration with the following values:

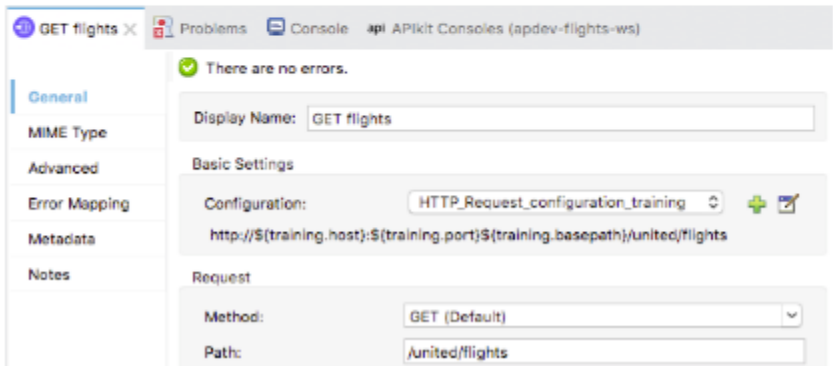
- Name: HTTP_Request_config_training
- Base Path: ${training.basepath}
- Host: ${training.host}
- Port: ${training.port}



21. Click OK.

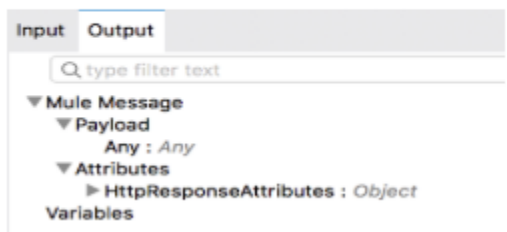
Configure the HTTP Request operation

22. Return to implementation.xml.
23. Navigate to the Get flights Request properties view in getUnitedFlights.
24. Set the configuration to the existing HTTP_Request_configuration_training.
25. Leave the method set to GET.
26. Set the path to /united/flights.



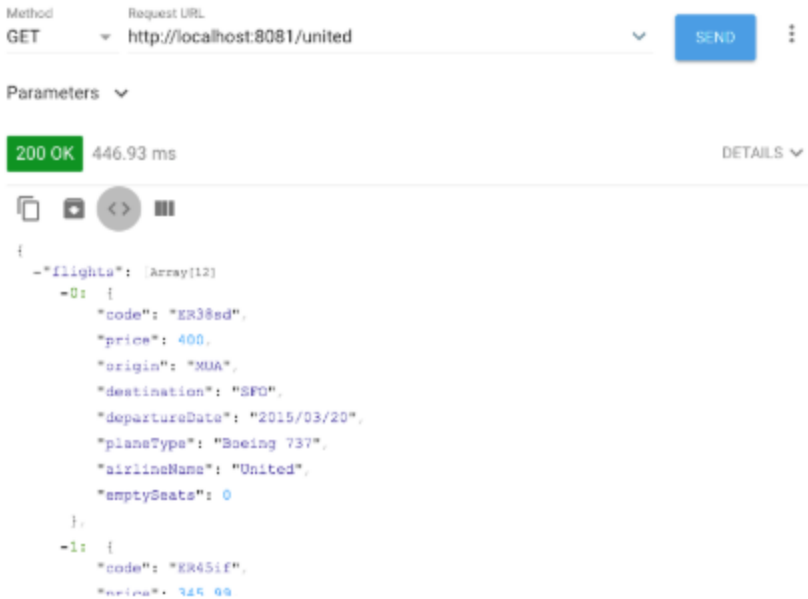
Review metadata associated with the United Get flights operation response

27. Select the Output tab in the DataSense Explorer and expand Payload; you should see no metadata for the payload.



Test the application

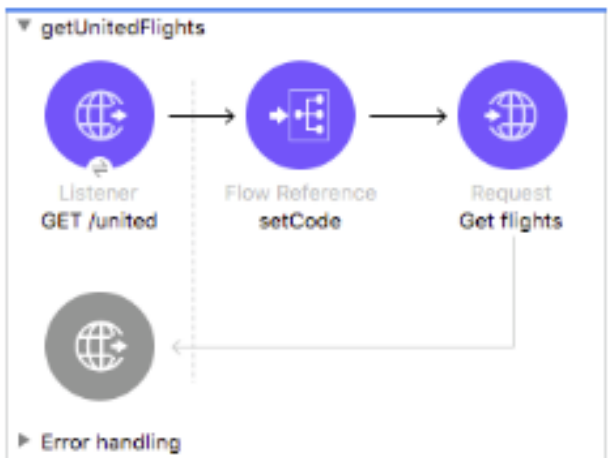
28. Save the files to redeploy the project.
29. In Advanced REST Client, return to the tab with the localhost request.
30. Change the URL to make a request to http://localhost:8081/united; you should get JSON flight data for all destinations returned.



31. Add a query parameter named code and set it to one of the destination airport code values: http://localhost:8081/united?code=CLE; you should still get flights to all destinations.

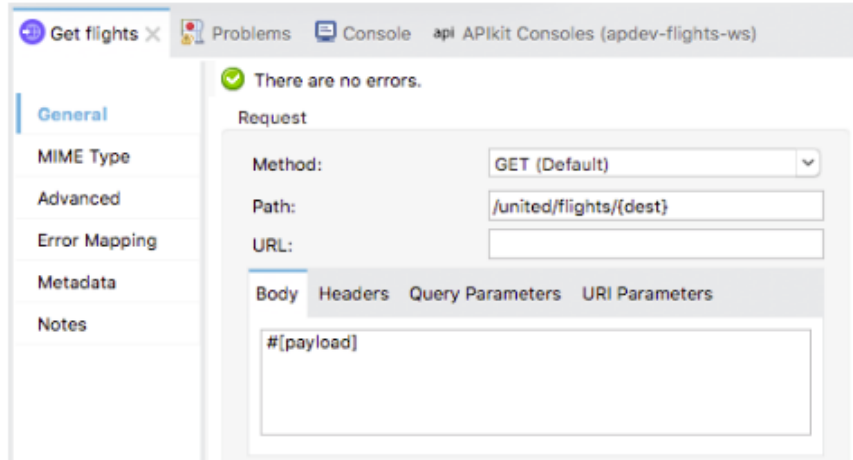
Add a URI parameter to the web service call

32. Return to implementation.xml in Anypoint Studio.
33. Drag a Flow Reference component from the Mule Palette and drop it after the GET /united Listener in getUnitedFlights.
34. In the Flow Reference properties view, set the flow name to setCode.



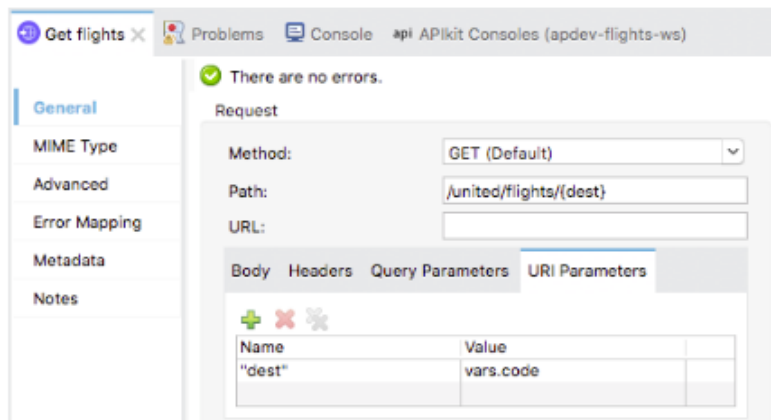
35. Navigate to the Get flights Request properties view in getUnitedFlights.

36. In the Request section, change the path to /united/flights/{dest}.



37. Select the URI Parameters tab.
38. Click the Add button.
39. Set the name to dest.
40. Set the value to the value of the code variable.

    `vars.code`



Test the application

41. Save the file to redeploy the application.
42. In Advanced REST Client, make the same request; you should now only get flights to CLE.

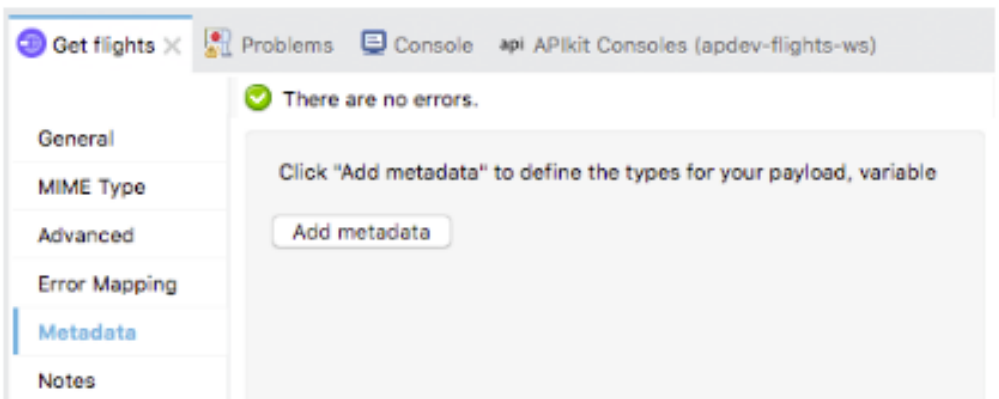
43. Remove the code parameter and make the request; you should now only get results for SFO.

    Note: You will transform the data to the format specified by the mua-flights-api in a later walkthrough.

Add metadata for the United Get flights operation response

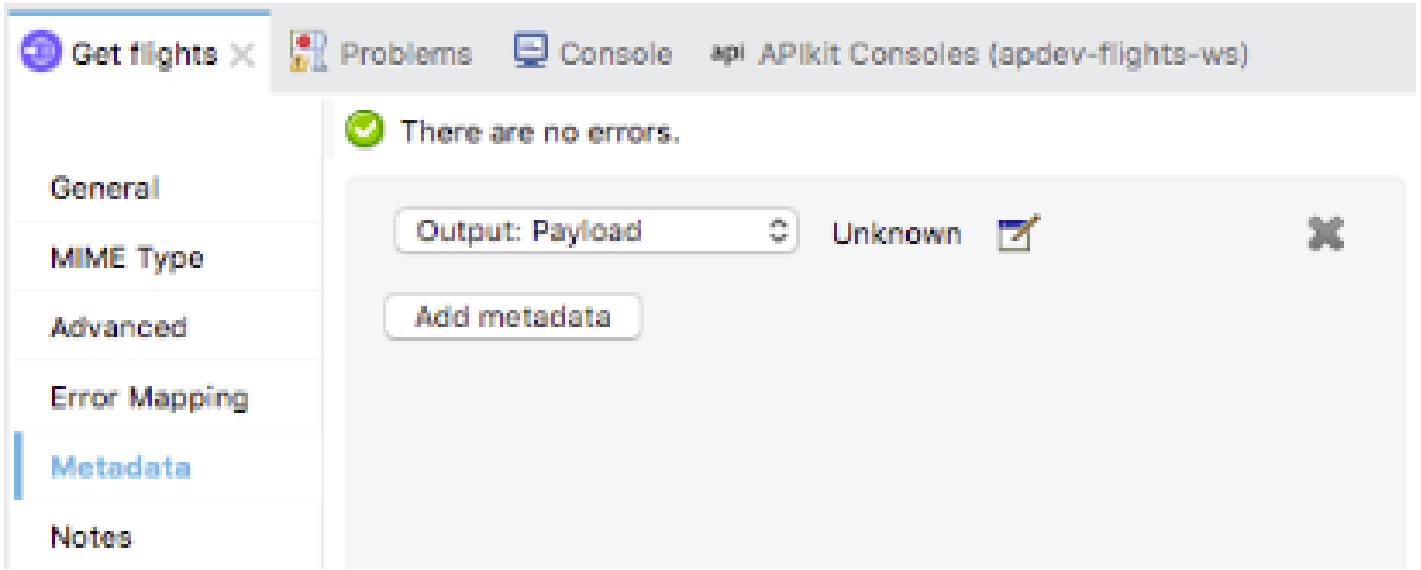
44. Return to Anypoint Studio.
45. Navigate to the properties view for the Get flights operation in getUnitedFlights.
46. Select the Output tab in the DataSense Explorer and expand Payload; you should see not metadata for the payload.

47. In the Get flights properties view, select the Metadata tab.
48. Click the Add metadata button.



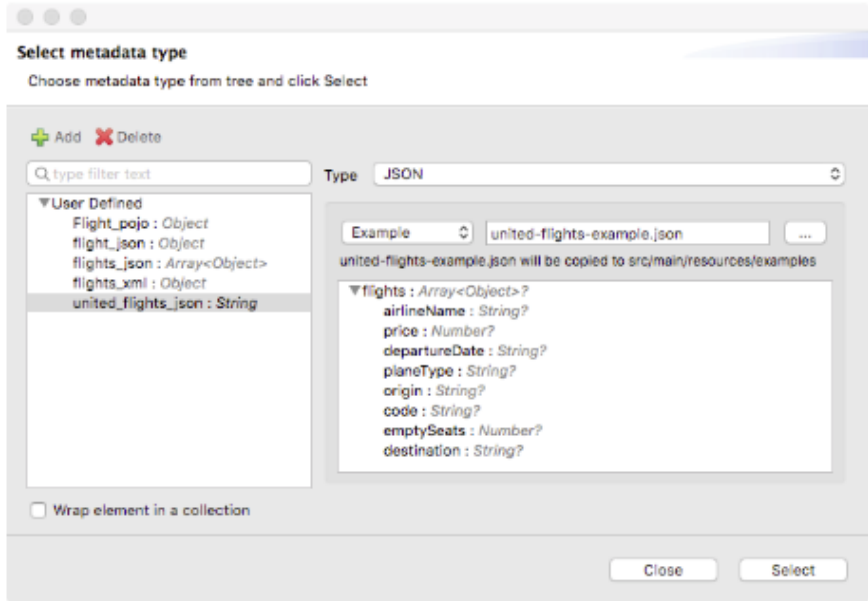
49. Change the drop-down menu to Output: Payload.

50. Click the Edit button.

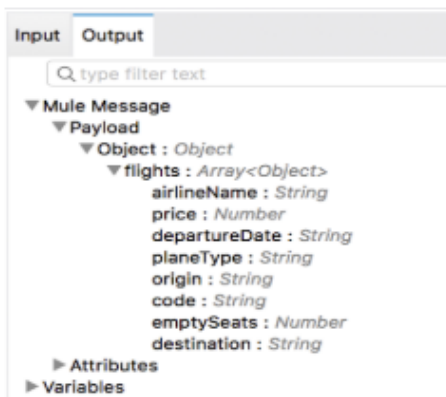


51. In the Select metadata type dialog box, click the Add button.
52. In the Create new type dialog box, set the type id to united_flights_json.
53. Click Create type.
54. In the Select metadata type dialog box, set the type to JSON.
55. Change the Schema selection to Example.
56. Click the browse button and navigate to the projects's src/test/resources folder.

57. Select united_flights-example.json and click Open; you should see the example data for the metadata type.



58. Click Select.
59. In the DataSense Explorer, select the Output tab and expand Payload; you should now see the structure for the payload.

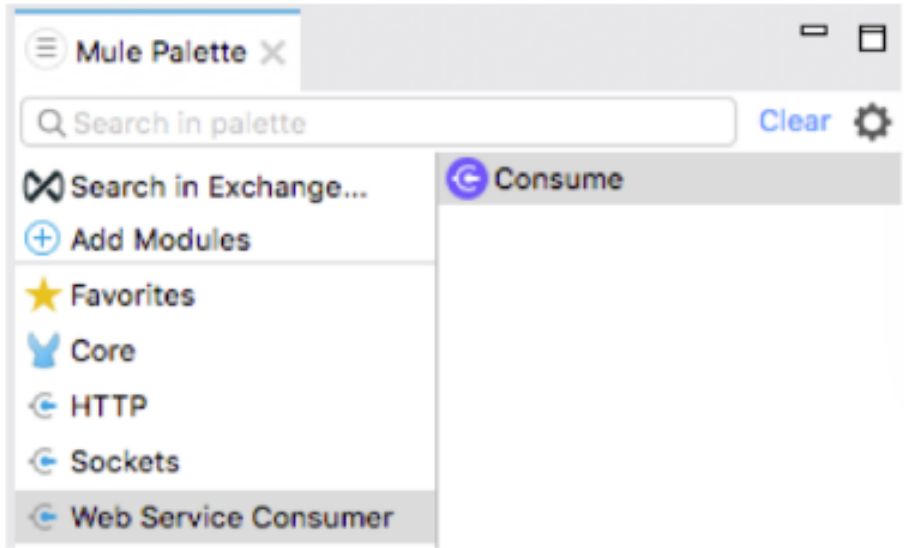


Consuming SOAP web services

Consuming SOAP web services

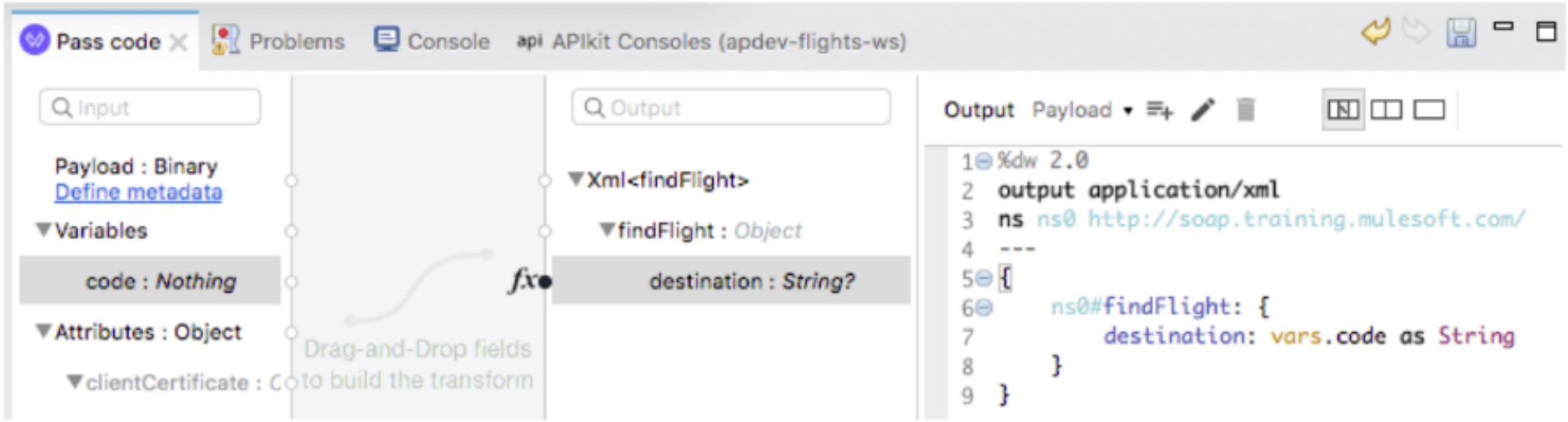
- First check and see if there is an existing Anypoint Connector in Studio or Exchange to connect to the service provider

- If there is not, use the **Web Service Consumer** connector
  - Add the Web Service Consumer module to the project
  - Configure a global element configuration, which includes the location of the WSDL
  - Use the Consume operation
  - Select the SOAP operation to invoke



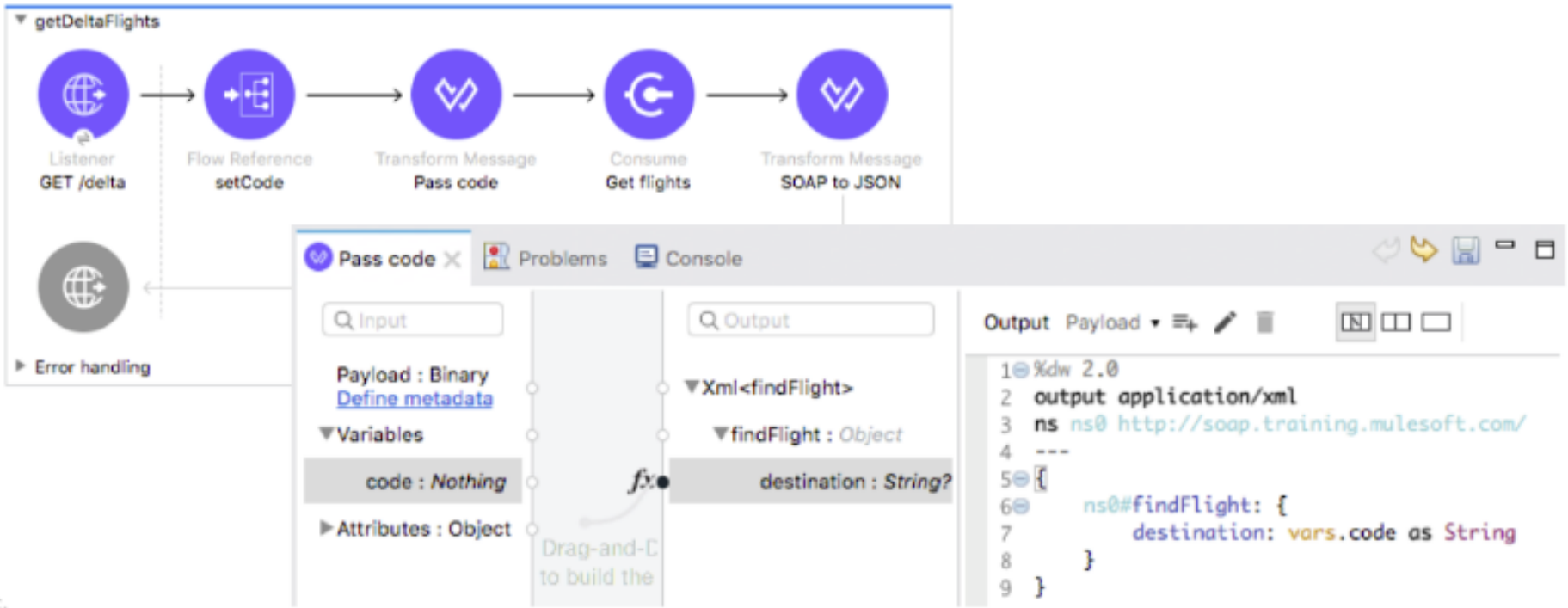
Passing data to a SOAP web service

- Use the **Transform Message** component to pass arguments to a SOAP web service
- When you add it before the Consume operation, DataSense is used to create metadata for the input that includes the arguments



Walkthrough 8-3: Consume a SOAP web service

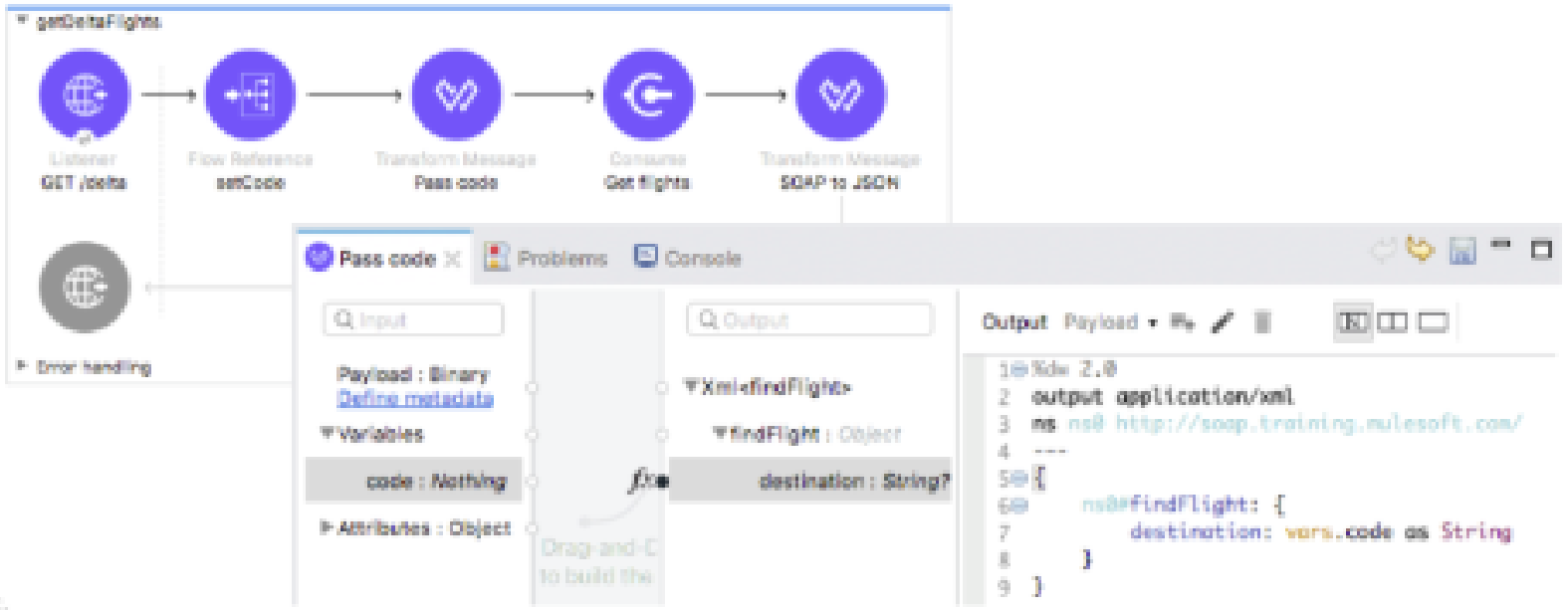
- Create a new flow to call the Delta SOAP web service
- Use the Web Service Consumer connector to call a SOAP web service
- Use the Transform Message component to pass arguments to a SOAP web service



Walkthrough 8-3: Consume a SOAP web service

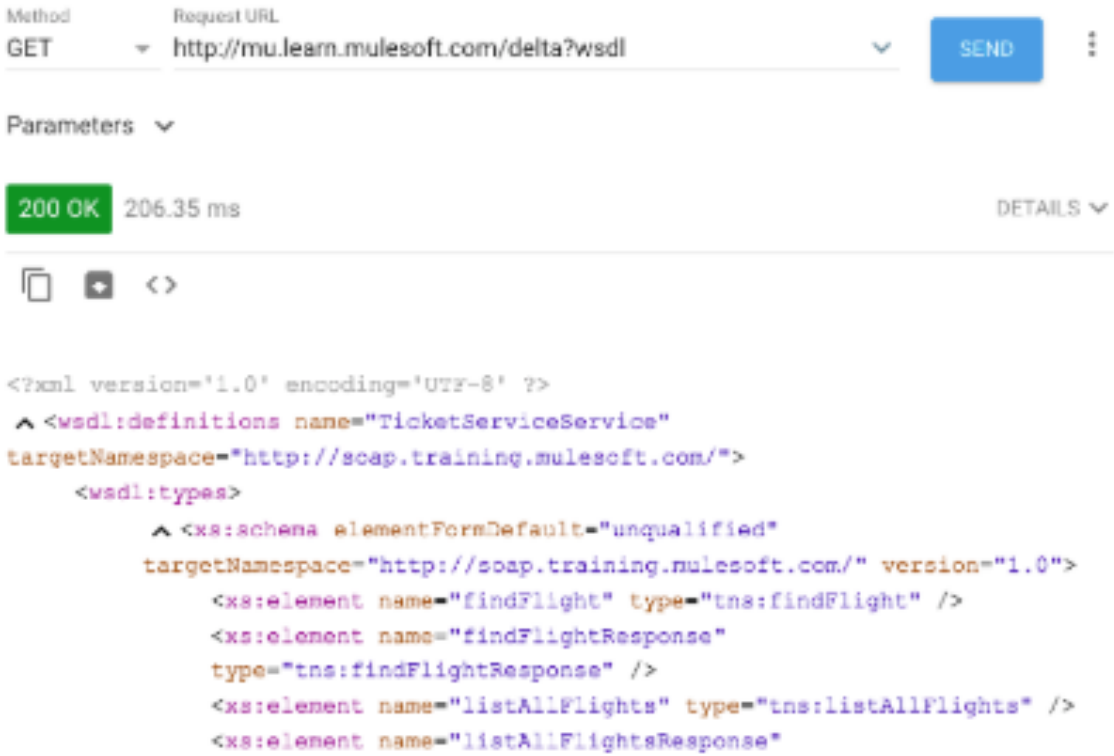
In this walkthrough, you consume a Delta SOAP web service. You will:

- Create a new flow to call the Delta SOAP web service.
- Use a Web Service Consumer connector to consume a SOAP web service.
- Use the Transform Message component to pass arguments to a SOAP web service.



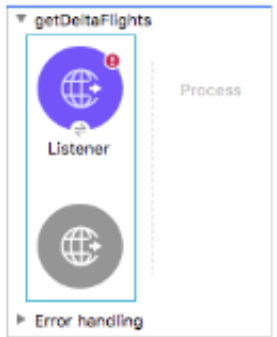
Browse the WSDL

1. Return to the course snippets.txt file and copy the Delta SOAP web service WSDL.
2. In Advanced REST Client, return to the third tab, the one with the learn.mulesoft 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.

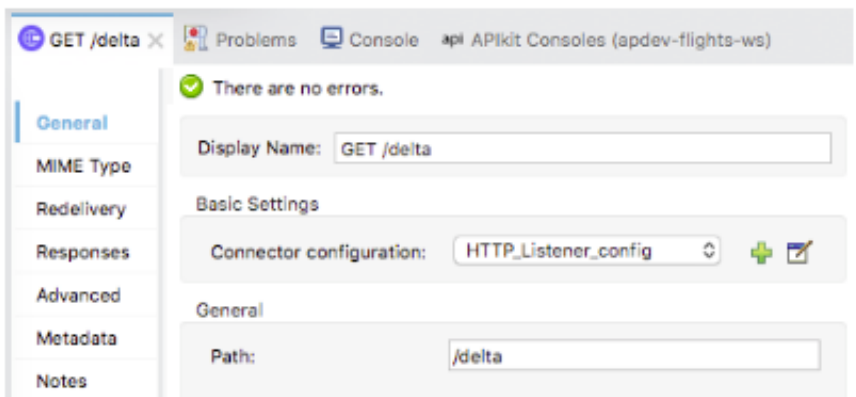


Create a new flow with an HTTP Listener connector endpoint

5. Return to Anypoint Studio.
6. Drag out another HTTP Listener from the Mule Palette and drop it in the canvas after the existing flows.
7. Rename the flow to getDeltaFlights.

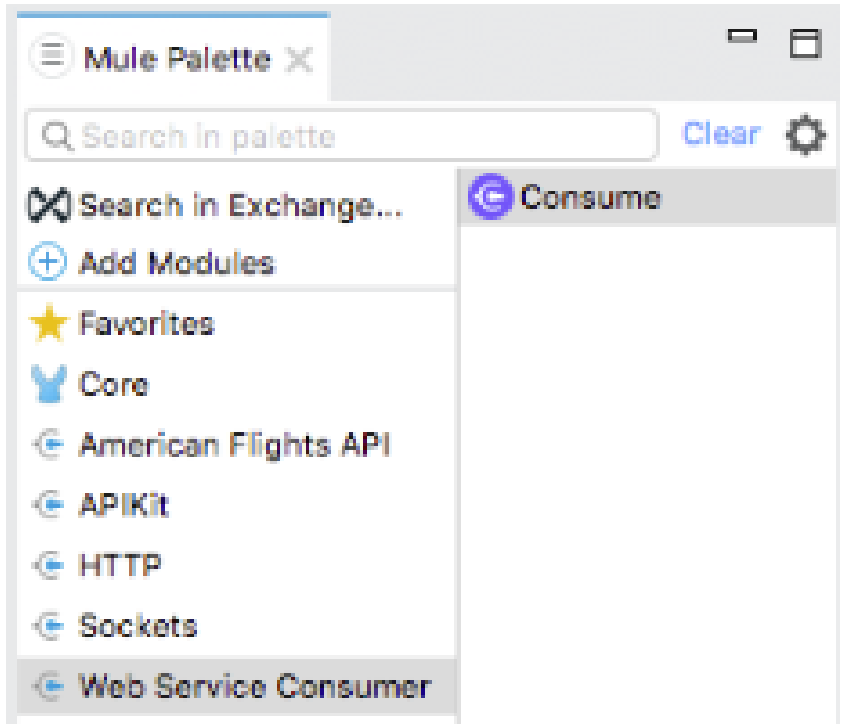


8. In the Listener properties view, set the display name to GET /delta.
9. Set the connector configuration to the existing HTTP_Listener_config.
10. Set the path to /delta and the allowed methods to GET.



Add the Web Service Consumer module to the project

11. In the Mule Palette, select Add Modules.
12. Select the Web Service Consumer connector in the right side of the Mule Palette ar
13. If you get a Select module version dialog box, select the latest version and click Ad



Configure the Web Service Consumer connector

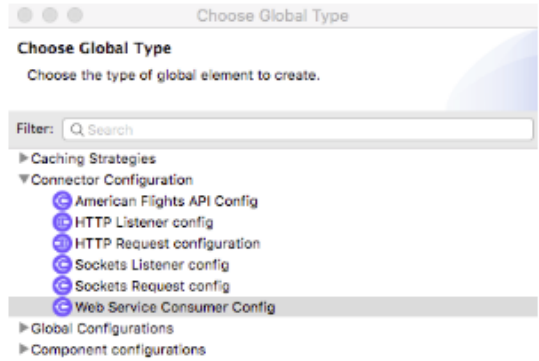
14. Return to the course snippets.txt file and copy the text for the Delta web service properties.
15. Return to config.yaml in src/main/resources and paste the code at the end of the file.

*implementation | *config.yaml | global

```yaml
http:
  port: "8081"

american:
  host: "training4-american-api-{lastname}.cloudhub.io"
  port: "80"
  basepath: "/"
  protocol: "HTTP"
  client_id: "your_client_id"
  client_secret: "your_client_secret"

training:
  host: "mu.learn.mulesoft.com"
  port: "80"
  basepath: "/"
  protocol: "HTTP"

delta:
  wsdl: "http://mu.learn.mulesoft.com/delta?wsdl"
  service: "TicketServiceService"
  port: "TicketServicePort"
```

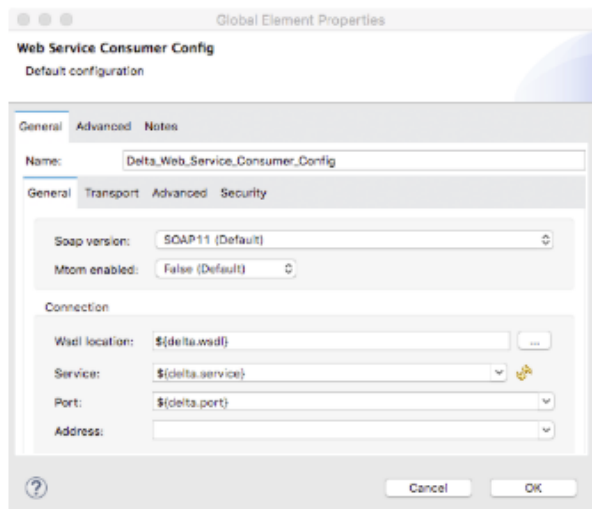
16. Save the file.
17. Return to global.xml.
18. Click Create.

19. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer Config and click OK.



20. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer_Config.
21. Set the

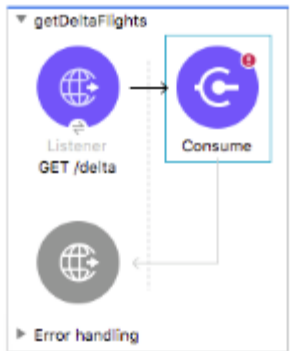
- Wsdl location: ${delta.wsdl}
- Service: ${delta.service}
- Port: ${delta.port}



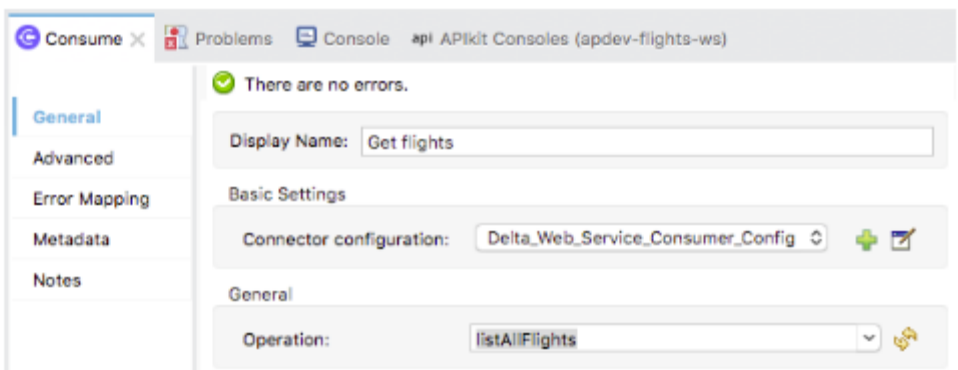
22. Click OK.

Add and configure a Consume operation

23. Return to implementation.xml.
24. Locate the Consume operation for the Web Service Consumer connector in the Mule Palette and drag and drop it in the process section of getDeltaFilghts.

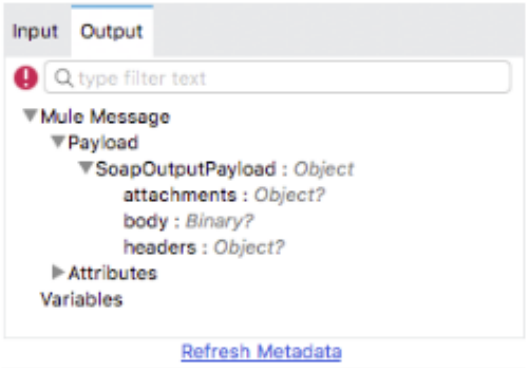


25. In the Consume properties view, change its display name to Get flights.
26. Set the connector configuration to the existing Delta_Web_Service_Consumer_Config.
27. Click the operation drop-down menu button; you should see all of the web service operations listed.
28. Select the listAllFlights operation.

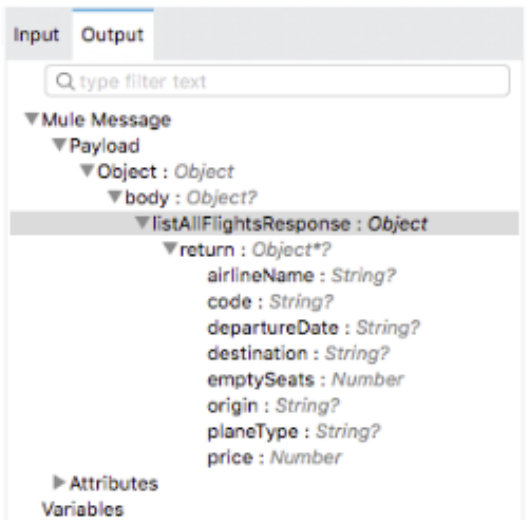


Review metadata associated with the United Get flights operation response

29. Select the Output tab in the DataSense Explorer and expand Payload; you should see payload metadata but with no detailed structure.

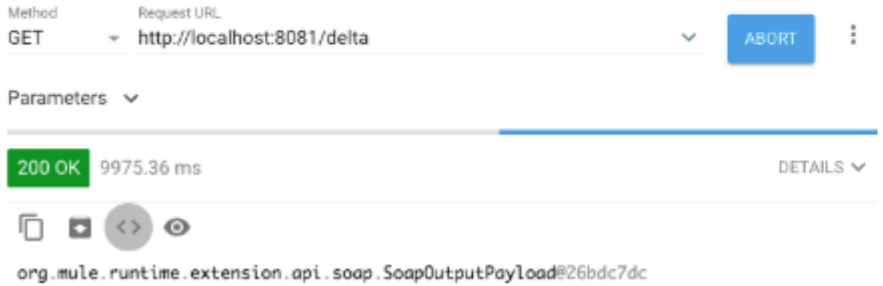


30. Save the file.
31. Select the Output tab in the DataSense Explorer and expand Payload again; you should now see the payload structure.



Test the application

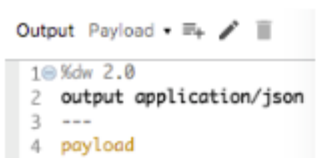
32. Save the files to redeploy the project.
33. In Advanced REST Client, return to the middle tab – the one with the localhost requests.
34. Make a request to http://localhost:8081/delta; you should get a response that is object of type SoapOutputPayload.



Transform the response to JSON

35. Return to Anypoint Studio.
36. Add a Transform Message component to the end of the flow.
37. Set the display name to SOAP to JSON.

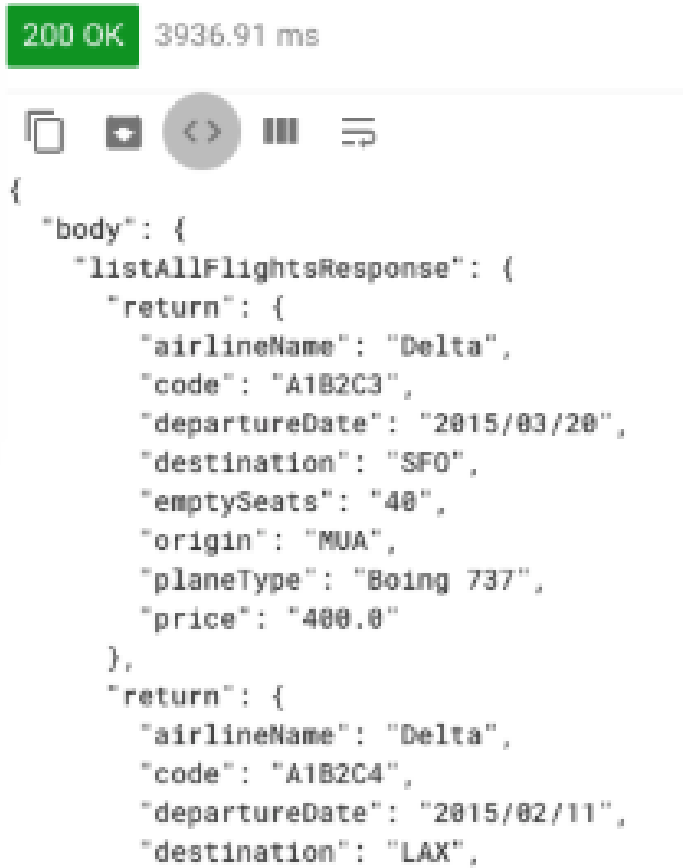
38. In the expression section of the Transform Message properties view, change the output type to json and the output to payload.



Test the application

39. Save the file to redeploy the project.
40. In Advanced REST Client, make another request to http://localhost:8081/delta; you should get JSON returned.

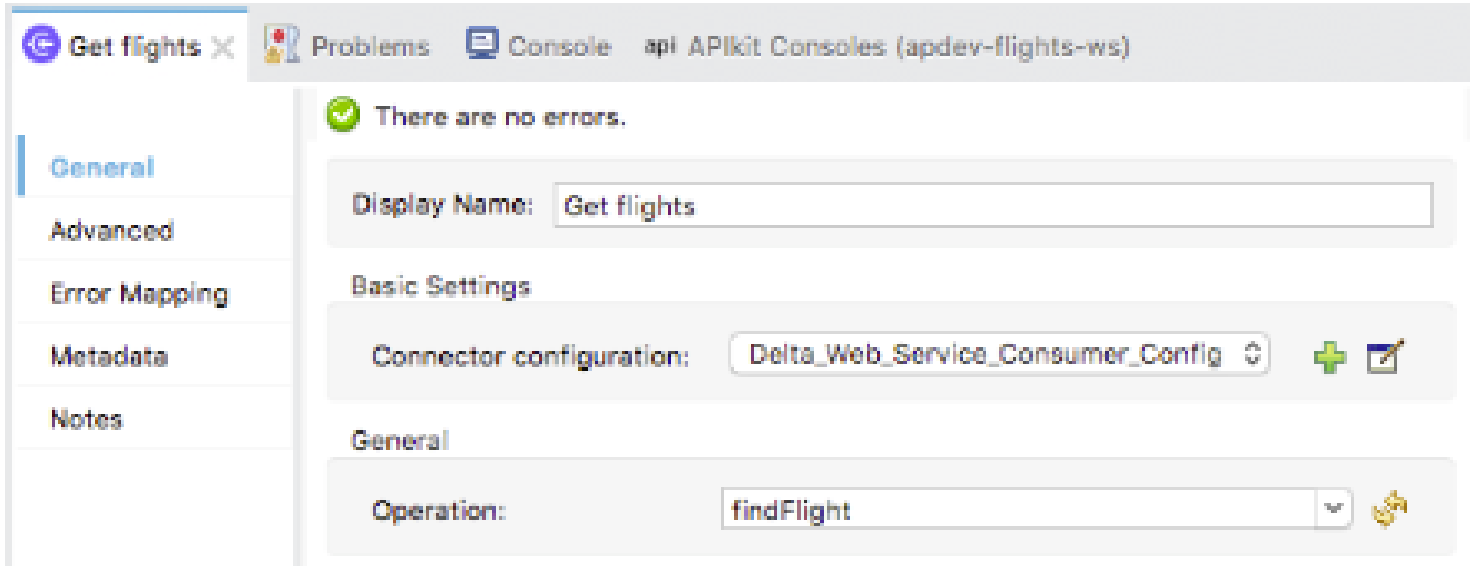
41. Click the Toggle raw response view button; you should see all the flights.



42. Add a query parameter called code and set it equal to LAX.
43. Send the request; you should still get all flights.

Call a different web service operation

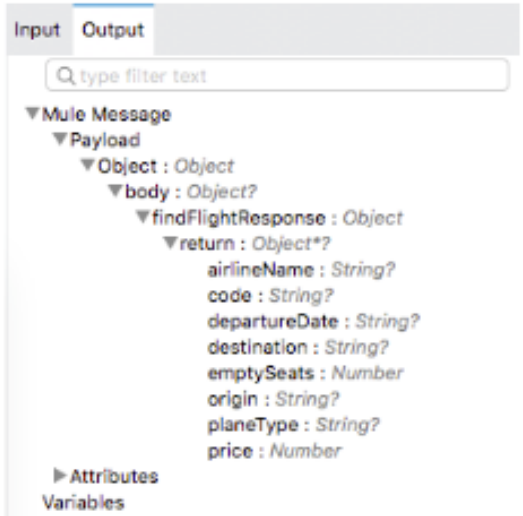
44. Return to getDeltaFlights in Anypoint Studio.
45. In the properties view for Get flights Consume operation, change the operation to findFlight.



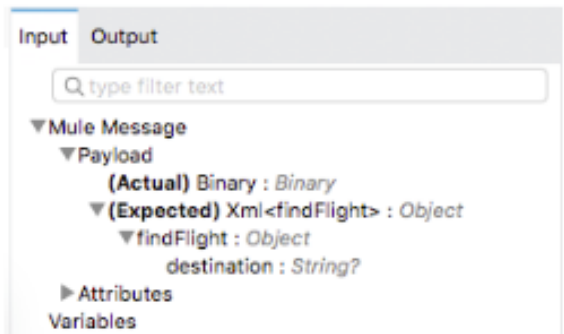
46. Save all files to redeploy the application.

Review metadata associated with the United Get flights operation response

47. Return to the Get flights properties view.
48. Select the Output tab in the DataSense Explorer and expand Payload; you should now see different payload metadata.



49. Select the Input tab and expand Payload; you should see this operation now expects a destination.

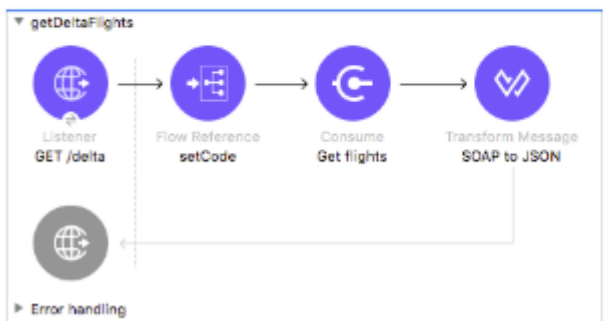


Test the application

50. In Advanced REST Client, send the same request with the query parameter; you should get a 500 Server Error with a message that the operation requires input parameters.

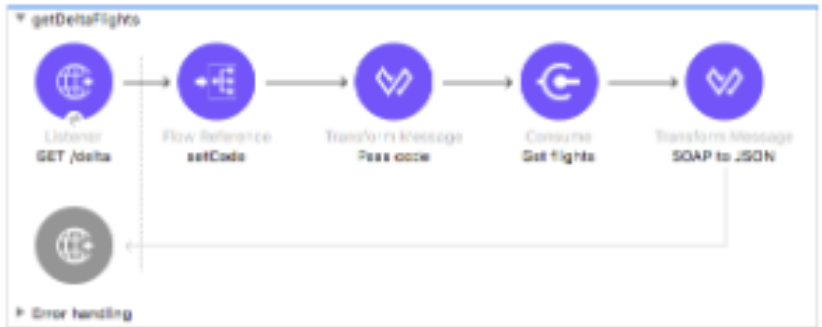
Use the set airport code subflow

51. Return to getDeltaFlights in Anypoint Studio.
52. Add a Flow Reference component after the GET /delta Listener.
53. In the Flow Reference properties view, set the flow name to setCode.

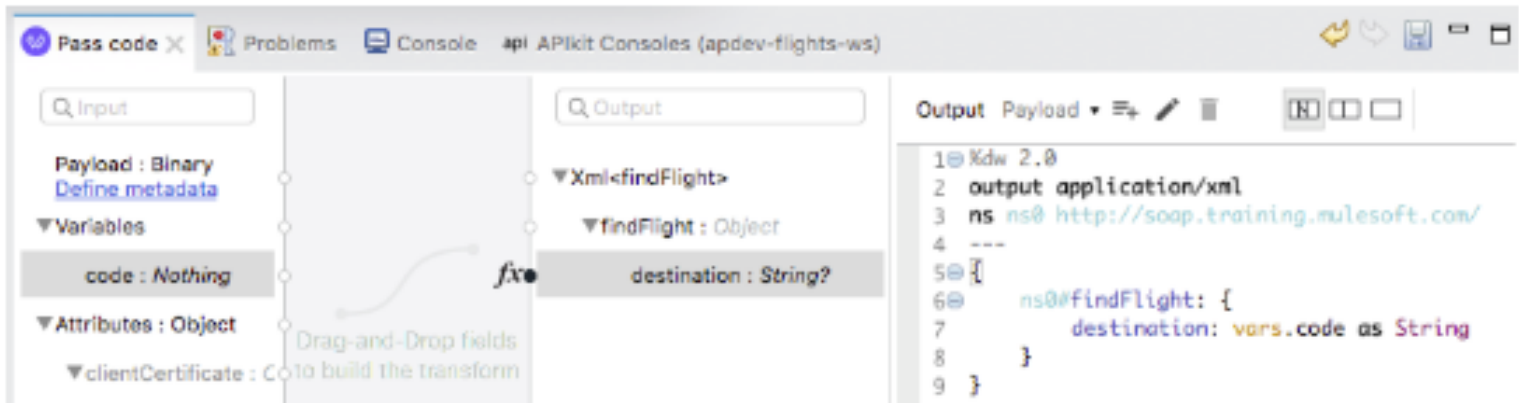


Use the Transform Message component to pass a parameter to the web service

54. Add a Transform Message component after the Flow Reference component.
55. Change its display name to Pass Code.

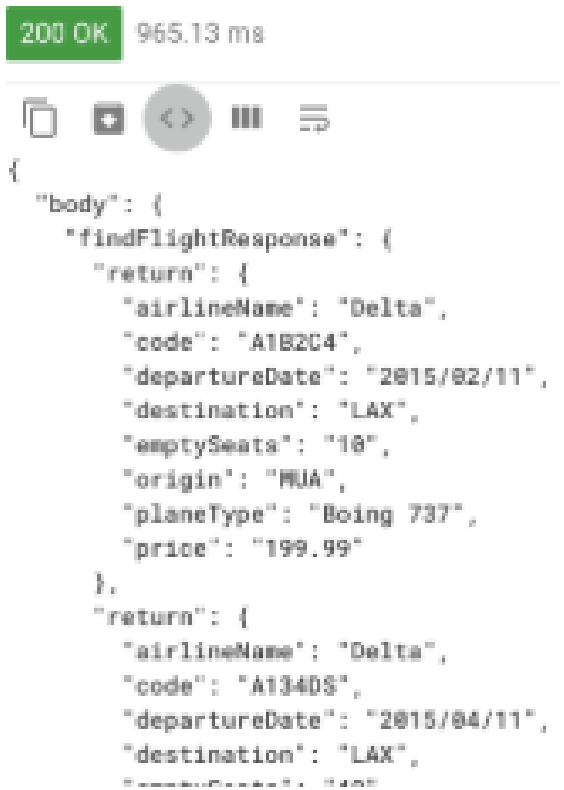


56. In the Pass code properties view, look at the input and output sections.
57. Drag the code variable in the input section to the destination element in the output section.



Test the application

58. Save the file to redeploy the application.
59. In Advanced REST Client, make another request.; you should now only see flights to LAX.



Combining data from multiple services

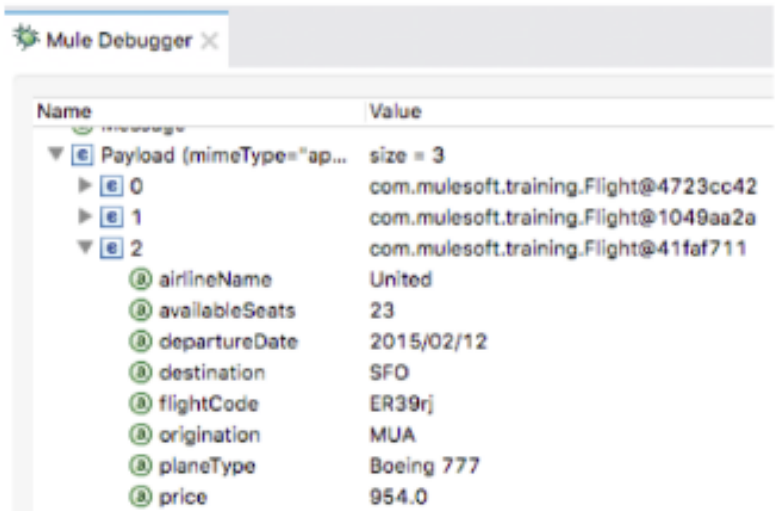
Combining data from multiple services

- Data from different services is pretty much always going to be in different formats

- To combine the data sets, you need to transform each of them to a canonical, or standard format
  - In this module, you will use a Java class as the canonical format
  - In module 11, you will learn to use the DataWeave format as a canonical format

Walkthrough 8-4: Transform data from multiple services to a canonical format

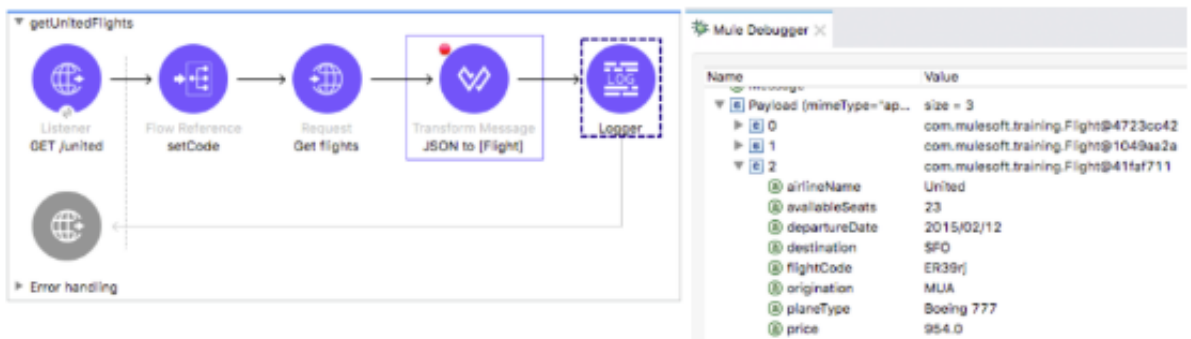
- Define a metadata type for the Flight Java class
- Transform the results from RESTful and SOAP web service calls to a collection of Flight objects



Walkthrough 8-4: Transform data from multiple services to a canonical format

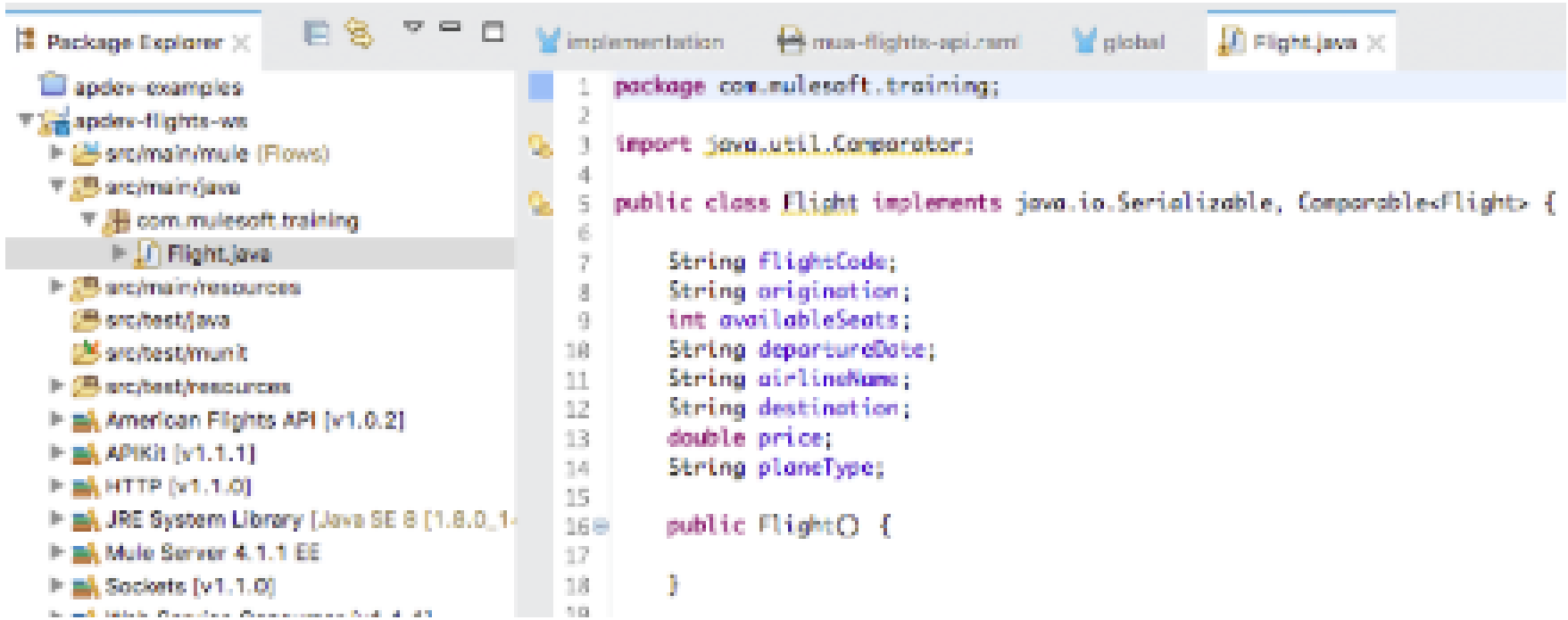
In this walkthrough, you will transform the JSON returned from the American and United web services and the SOAP returned form the Delta web service to the same format. You will:

- Define a metadata type for the Flight Java class.
- Transform the results from RESTful and SOAP web service calls to a collection of Flight objects.



Review Flight.java

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open Flight.java in src/main/java and review the file.

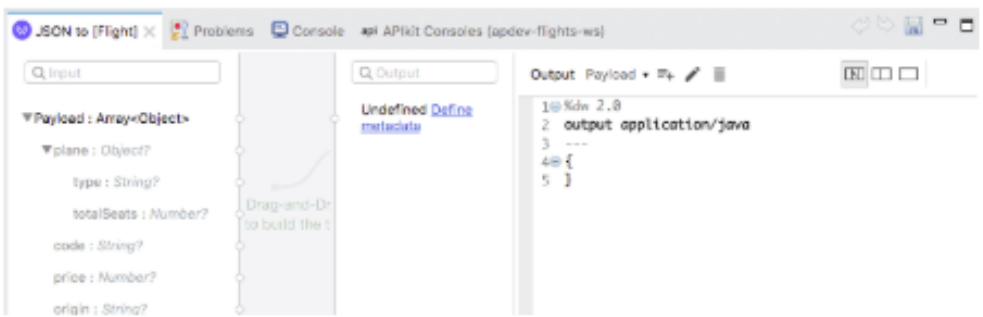


3. Close the file.

Change the American flow to return Java Flight objects instead of JSON

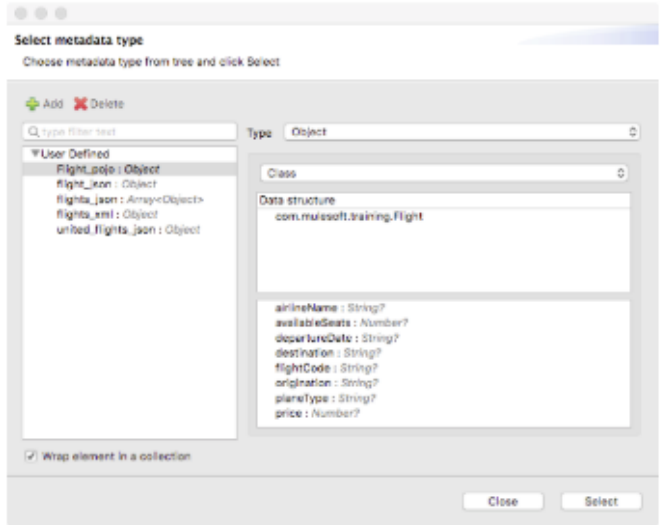
4. Return to getAmericanFlights in implementation.xml.
5. Add a Transform Message component to the end of the flow.
6. Add a Logger at the end of the flow.
7. Change the name of the Transform Message component to JSON to [Flight].

8. In the Transform Message properties view, look at the input section in the Transform Message properties view; you should see metadata already defined.

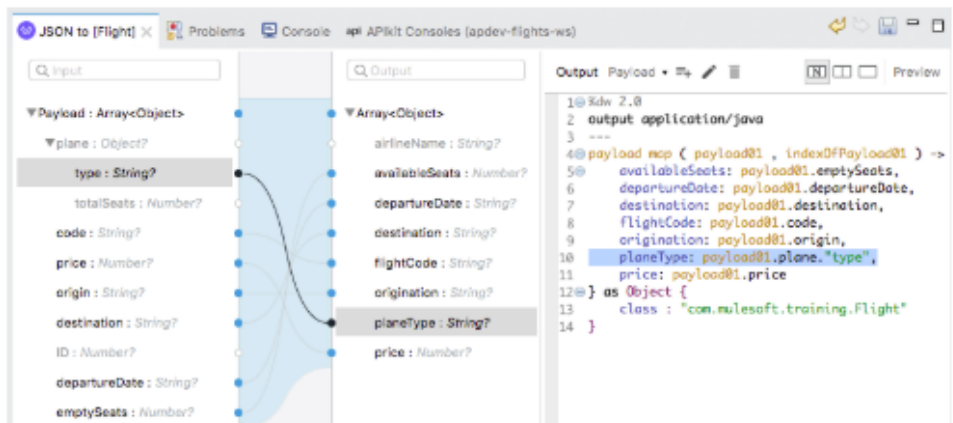


9. In the output section of the Transform Message properties view, click the Define metadata link.
10. In the Select metadata type dialog box, select the user-defined Flight_pojo.

11. Select Wrap element in a collection in the lower-left corner.



12. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
13. Map fields (except ID and totalSeats) by dragging them from the input section and dropping them on the corresponding field in the output section.



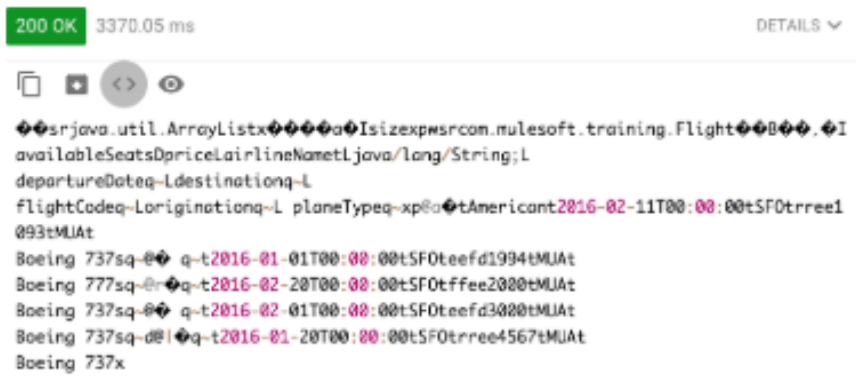
14. Double-click the airlineName field in the output section.

15. In the generated DataWeave expression, change the airlineName value from null to "American".

```
%dw 2.0
output application/java
---
payload map ( payload01 , indexOfPayload01 ) ->
    airlineName: "American",
    availableSeats: payload01.emptySeats,
```

Test the application

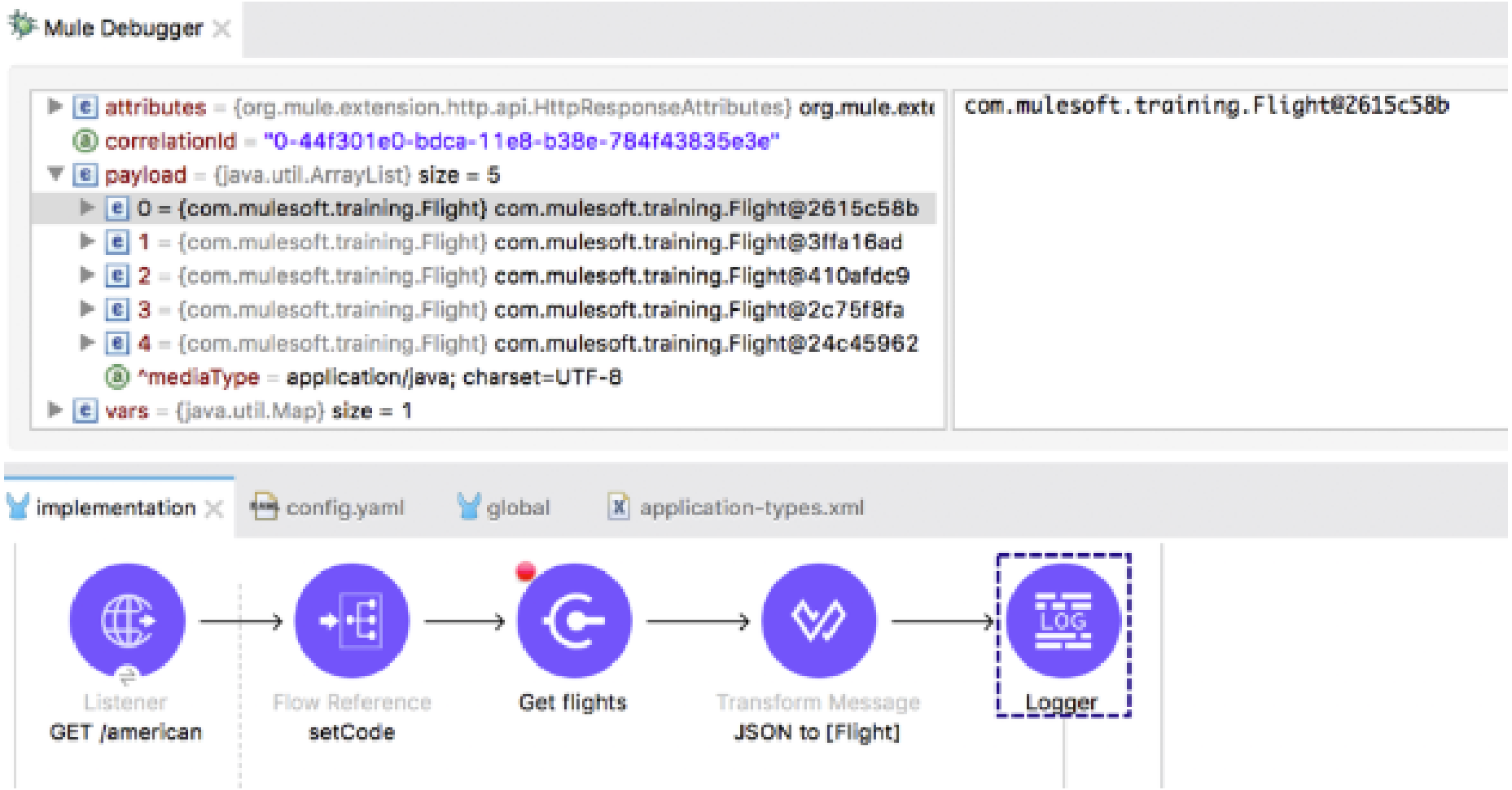
16. Save to redeploy the project.
17. In Advanced REST Client, change the URL and make a request to http://localhost:8081/american; you should see a representation of a collection of Java objects.



Debug the application

18. Return to Anypoint Studio.
19. Stop the project.
20. Add a breakpoint to the Get flights operation.
21. Debug the project.
22. In Advanced REST Client, make another request to http://localhost:8081/american.
23. In the Mule Debugger, step to the Transform message component and examine the payload.

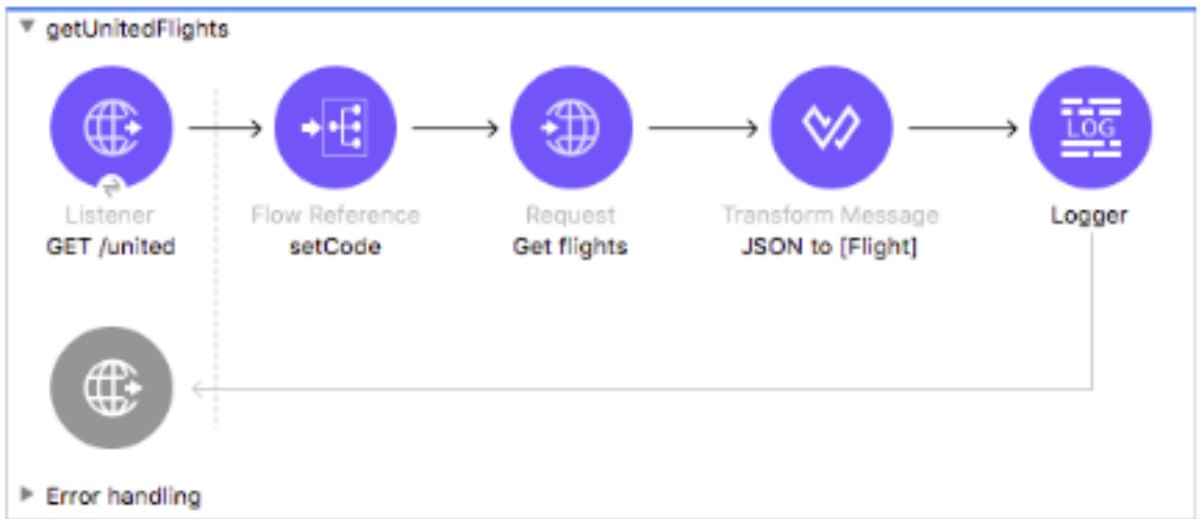
24. Step to the Logger and look at the payload it should be a collection or Flight objects.



25. Step through the rest of the application and switch perspectives.

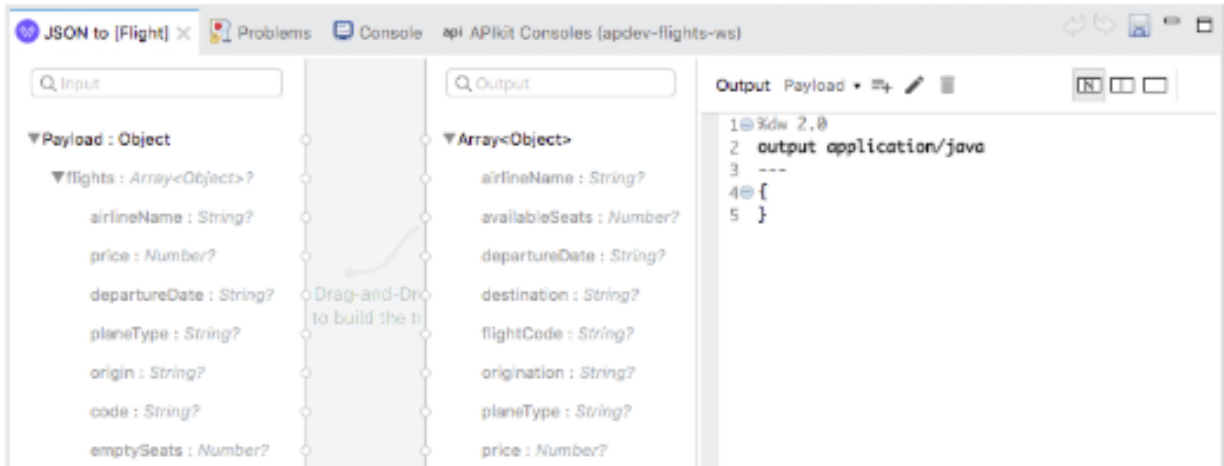
Change the United airline flows to return Java Flight objects instead of JSON

26. Return to geUnitedFlights in implementation.xml.
27. Add a Transform Message component at the end of the flow.
28. Change the name of the Transform Message component to JSON to [Flight].
29. Add a Logger to the end of the flow.

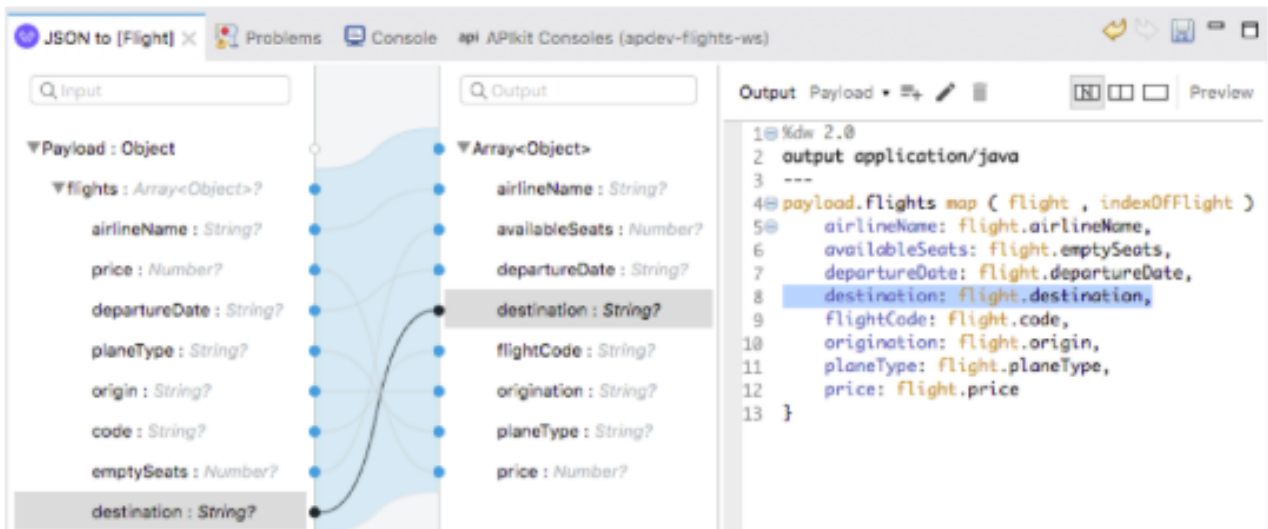


30. In the Transform Message properties view, look at the input section; you should see metadata already defined.
31. In the output section of the Transform Message properties view, click the Define metadata link.
32. In the Select metadata type dialog box, select the user-defined Flight_pojo.
33. Select Wrap element in a collection in the lower-left corner.

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



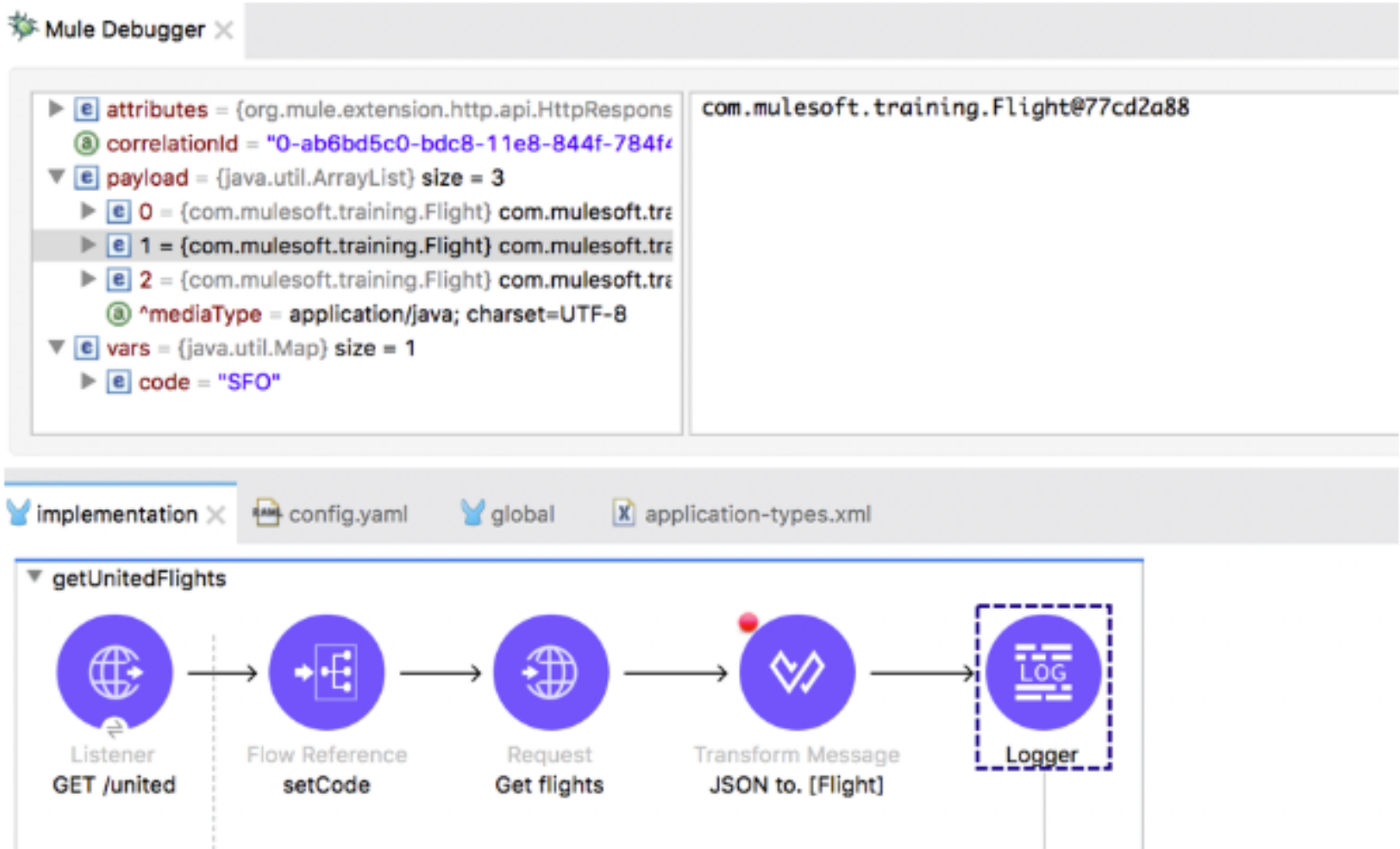
35. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.



Debug the application

36. Add a breakpoint to the Transform Message component.
37. Save the files to redeploy the project.
38. In Advanced REST Client, change the URL to make a request to http://localhost:8081/united.
39. In the Mule Debugger, examine the payload.

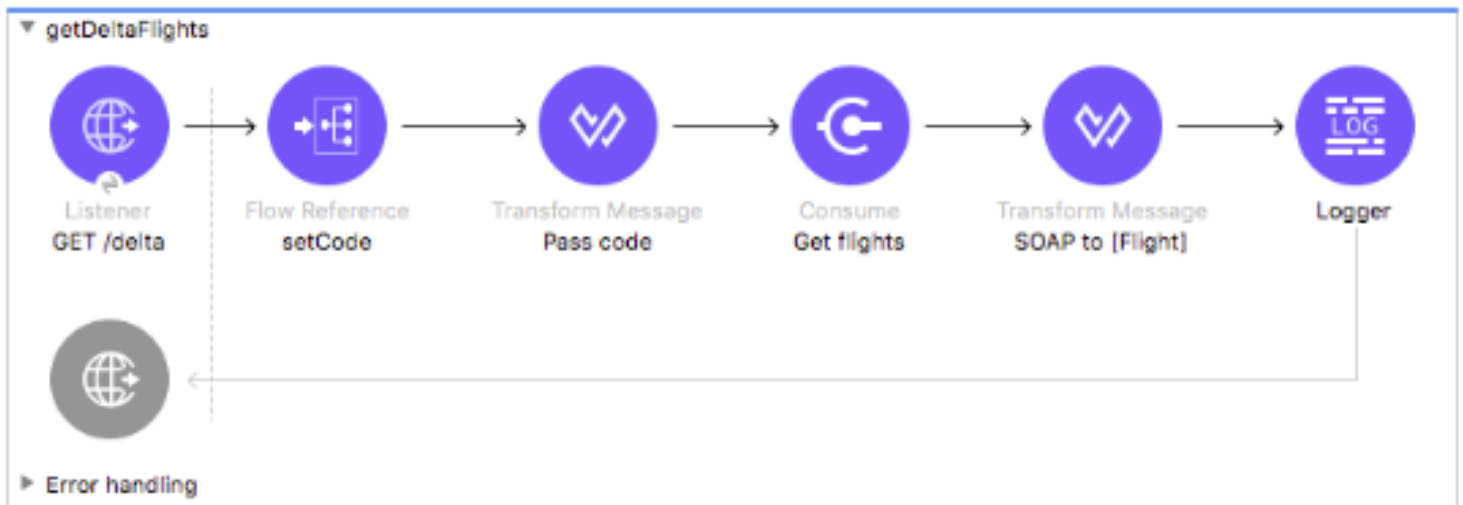
40. Step to the Logger and look at the payload it should be a collection or Flight objects.



41. Step through the rest of the application and switch perspectives.

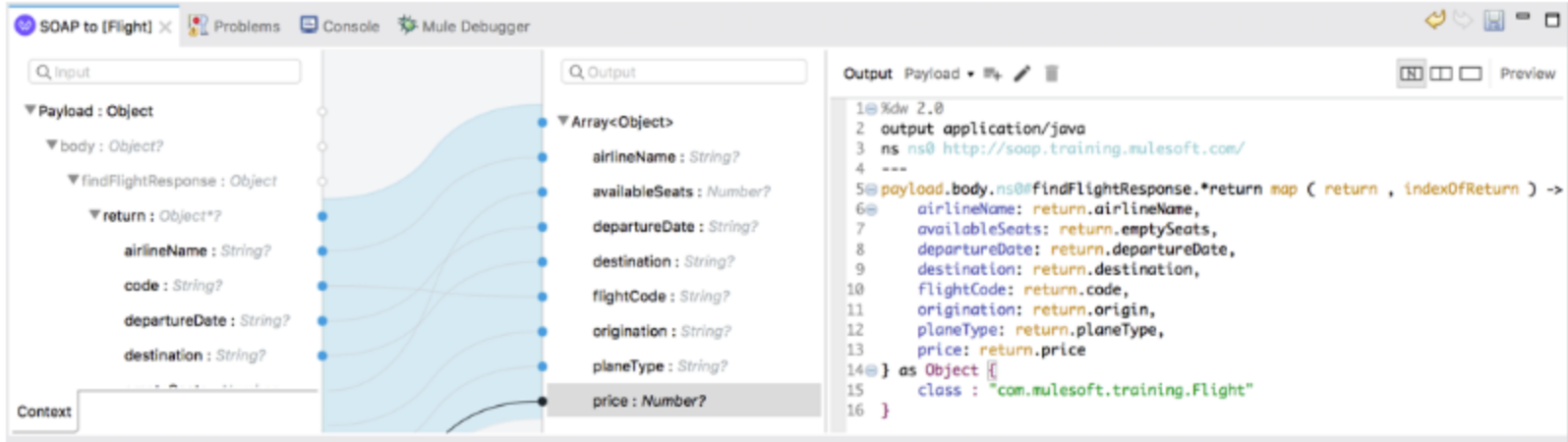
Change the Delta flow to return Java Flight objects

42. Return to geDeltaFlights in implementation.xml.
43. Change the name of the Transform Message component to SOAP to [Flight].
44. Add a Logger to the end of the flow.



45. Look at the input section in the Transform Message properties view; 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 Select metadata type dialog box, select the user-defined Flight_pojo.
48. Select Wrap element in a collection in the lower-left corner.
49. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

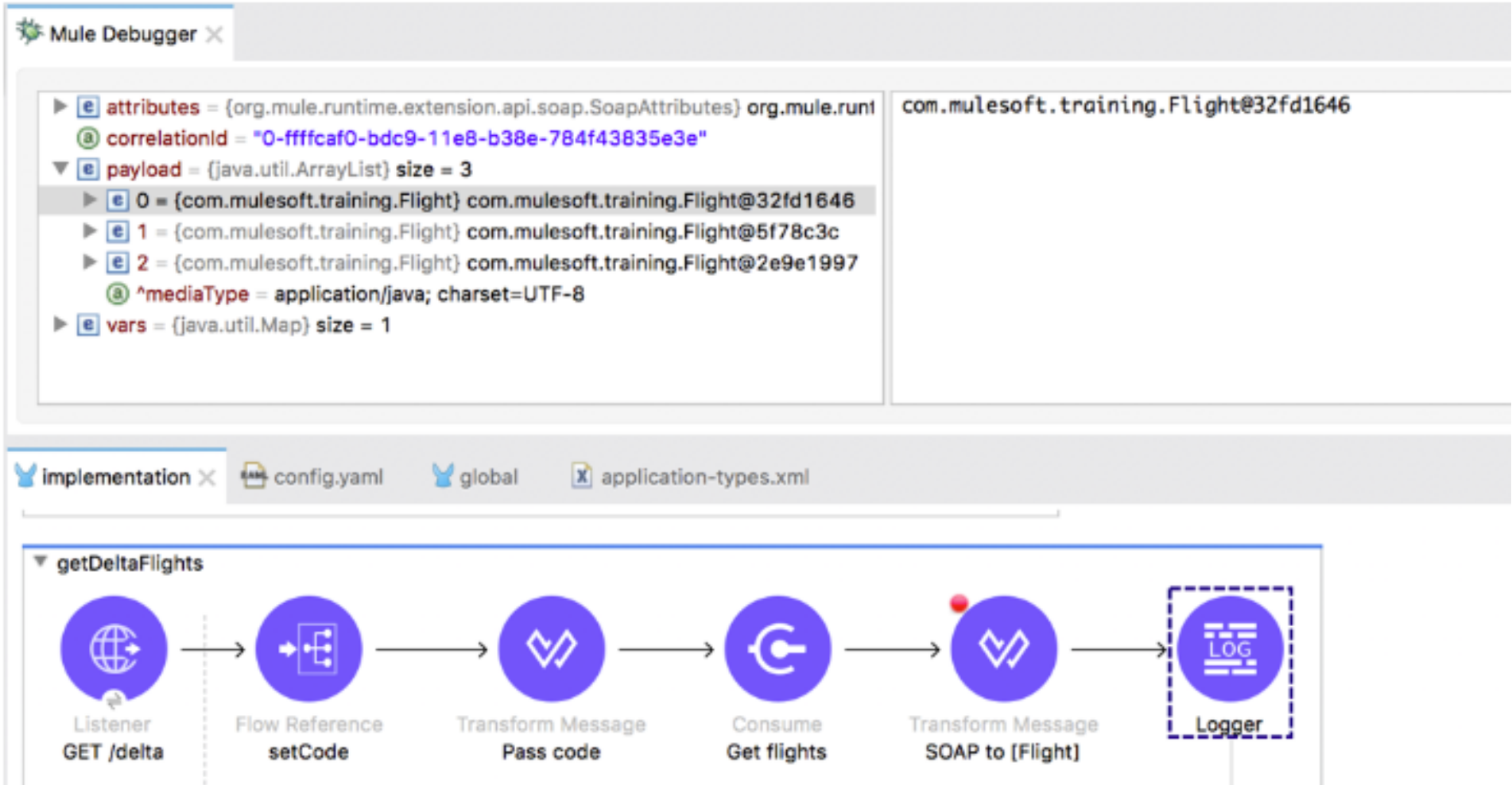
50. Map the fields by dragging them from the input section and dropping them on the corresponding field in the output section.



Debug the application

51. Add a breakpoint to the Transform Message component.
52. Save the file to redeploy the project.
53. In Advanced REST Client, change the URL to make a request to http://localhost:8081/delta.
54. In the Mule Debugger, examine the payload.

55. Step to the Logger and look at the payload it should be a collection or Flight objects.



56. Step through the rest of the application and switch perspectives.

Summary

Summary

- To consume a web service, first look to see if it has a **connector in Anypoint Exchange**
  - Easiest way to consume a web service
- Use the **HTTP Request** operation to consume any REST web service
  - With or without URI parameters and query parameters
  - With or without a RAML definition
- Use the **Web Service Consumer** connector to consume any SOAP web service
- Use the Transform Message component to pass arguments to SOAP web services

DIY Exercise 8-1: Orchestrate web services

Time estimate: 3 hours

Objectives

In this exercise, you retrieve data from several different data sources and combine them using DataWeave. You will:

- Retrieve data from a REST service.
- Retrieve data from a SOAP service.
- Use DataWeave to transform and combine data from two sources into a new data structure.
- Upload data to an external REST service.

Scenario

The Finance department is using two APIs: Accounts and Transactions. Currently, their analysts have to retrieve information from both APIs separately and then combine them using a spreadsheet. The Finance department is requesting a Process API that can combine accounts information with transaction information to reduce the manual work needed to combine the data in a spreadsheet.

Your task is to implement this Process API. First, you need to create a new Mule application that retrieves account information using the Accounts System API and then uses the Transactions System API to retrieve transaction data for each account. Each account can have zero or more transactions. Next, you need to transfer these records to several different Finance directors using an existing Experience API. To do this, you need to join the transaction data for each account with the other account record details to create a new data structure compatible with the existing Experience API. The expected behavior of the new application is specified in the Transfer Accounts API: /files/module08/accounts-transactions-mod08-api.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

Review the Accounts API

Review the Accounts API you created in exercise 3-1 or the solution /files/module03/accounts-mod03-api.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

The Accounts API:

- Has an endpoint http://apdev-accounts-ws.cloudhub.io/api/accounts to request all accounts of a particular type (personal or business).
- Requires a query parameter named type set to personal or business.
- Requires a Requester-ID header.
- Has two other optional query parameters named country and name that can be used to filter the accounts to just a particular account owner's name or to the country of residence of the account holder.
- Returns a JSON array of matching Account objects.

Define a Process API to look up accounts and transfer transactions to the Finance directors reporting system

Define a Transfer Accounts API that accepts GET requests with the required Requester-ID and type query parameter (where type must be business or personal). This API is used to retrieve all accounts from the Accounts API for a particular Requester-ID and account type (business or personal) and then filtering them using the optional country and name query parameters. In the API definition, provide an example to show that the response should be a string indicating the number of messages processed.

Create a new application to retrieve data from the Accounts API

In Anypoint Studio, create a new Mule application that uses the Accounts API to retrieve accounts with the required and optional parameters specified in the Accounts API. The Mule application should accept the type, name, and country query parameters. Set the Requester-ID header to a static value.

Hint: You can use a Validator or a Choice router to test for required values. You can also set default values directly in DataWeave if a required parameter or header is null:

```
#[attributes.queryParams.type default "business"]
```

After checking for required attributes and parameters (and perhaps setting default values), add an HTTP Request to make the GET request to the Accounts API.

Hint: Remember to set the required Requestor-ID header and the required type query parameter.

Retrieve data from the transaction web service

After you have the array of JSON objects returned from the Accounts System API, pass a list of account IDs to the GetTransactionsforCustomers operation of the Transaction SOAP web service: http://apdev-accounts-ws.cloudhub.io/api/transactions?wsdl. The web service requires the request body to contain an XML request with a list of customerID values, where each customerID should correspond to one accountID returned from the Accounts System API:

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

<ns0:GetTransactionsforCustomers xmlns:ns0="http://training.mulesoft.com/">

  <customerID>4402</customerID>

  <customerID>4404</customerID>

  <customerID>4464</customerID>

</ns0:GetTransactionsforCustomers>
```

*Hint: You can use the following DataWeave script to return an array of account IDs from the payload resulting from a call to the Accounts System API. The code default [] returns an array by default if payload.*id evaluates to null.*

```
{customerIDs: payload.*id default []}
```

Make a request to the Mule application and verify matching transactions are returned from the SOAP query.

Here are a few example transactions results:


```
  {
    "amount": "9717.0",
    "customerRef": "4412",
    "flightID": "UNITEDER09452015-06-11",
    "region": "EU",
    "transactionID": "181948488"
  },
  {
    "amount": "1030.0",
    "customerRef": "4985",
    "flightID": "DELTAA1B3D42015-02-12",
    "region": "US",
    "transactionID": "181948625"
  }
```

Combine accounts and transactions data into a simple data structure

After retrieving data from accounts and transactions, combine them together in the simplest way:

{

"accounts": Array of Account Objects

"transactions": Array of all Transaction Objects for all accounts

}

Note: In the next step, you will create a more complex and normalized data structure.

Hint: A Transform Message component can create several different transformations at a time and output the results to variables. To add a new output target for the Transform Message component, click the plus icon to the left of the pencil icon in the Script Editor.

Join accounts and transaction data

Use the following DataWeave code to join the transaction details to each account in a hierarchical object structure. You can also find this code in the text file /files/module08/dwTransform.txt (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

```
%dw 2.0
output application/java


var directorIDs = ["JKLS483S","FJSA48JD","NMA4FJ9K"]


//Combines Accounts and Transactions by the Account ID. Assigns each account to a
//director
fun consolidateAccountsTrans (payload) =
 payload.accounts map ( (account, index) ->
 using(id = account.id as :string)
 (
  account ++
  {
   transactions: payload.transactions filter ($.customerRef == id)
  } ++
  {
```

```
    {
      assignedDirector: directorIDs[index mod sizeOf directorIDs]
    }
  )
)
---
using (data = consolidateAccountsTrans(payload))
(data groupBy $.assignedDirector)
```

Note: This DataWeave code creates the correct schema expected by the Experience API you will POST to in the next step.

Transfer enriched accounts data to corresponding Finance directors

An Experience API has been created that can be used to transfer records to Finance directors. It is available by making a POST request to:
http://apdev-accounts-ws.cloudhub.io/api/accounts_transactions

Note: This current implementation always returns the JSON response {"message": "Success"}. You do not need to set any headers or other parameters, and the body can be anything. This web service is slow and takes more than 10 seconds to return a response.

Return the number of records processed

Modify the Mule application to return a final payload with a message confirming the number of account records processed.

Note: Because the current Experience API does not return the number of processed records, just count the number of records sent in the outbound request to the Experience API.

Complete the transferAccounts API contract

Format the response to the request so that it conforms to the Transfer Accounts API you created.

Verify your solution

Import the solution /files/module08/accounts-transactions-mod08-joining-data-solution.jar deployable archive file into Anypoint Studio (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.