

Module 7: Consuming Web Services



GET localhost:8081/united?code=CLE

Body

```
1 {  
2   "airline": "United",  
3   "flightCode": "ER9fje",  
4   "fromAirportCode": "MUA",  
5   "toAirportCode": "CLE",  
6   "departureDate": "2015/07/11",  
7   "emptySeats": 32,  
8   "price": 845,  
9   "planeType": "Boeing 727"  
10 },  
11 {  
12   "airline": "United",  
13   "flightCode": "ER3kfd",  
14   "fromAirportCode": "MUA",  
15   "toAirportCode": "CLE",  
16   "departureDate": "2015/08/11",  
17   "emptySeats": 13,  
18   "price": 245,  
19   "planeType": "Boeing 747"  
20 },  
21 }  
22 ]
```

2

Objectives:

- 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 7-1: 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.

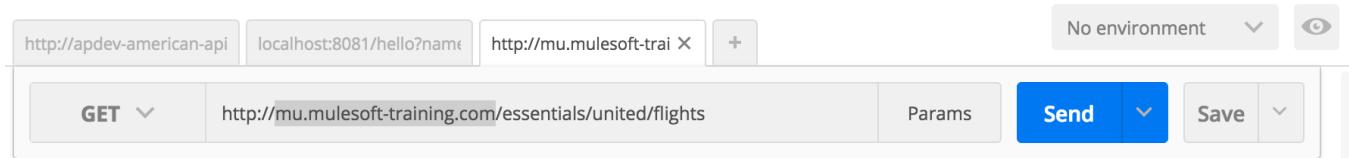
The screenshot shows the Mule Studio interface with a flow named 'getUnitedFlightsFlow'. The flow consists of four components: 'HTTP', 'United REST Request', 'Transform Message', and 'Logger'. The 'HTTP' component has a 'POST' method and a URL of 'localhost:8081/united'. The 'United REST Request' component is highlighted with a red dot. The 'Transform Message' component contains a DataWeave script that transforms the incoming JSON into a specific schema. The 'Logger' component logs the transformed data. The 'Body' tab of the request window shows the JSON response from the service, which includes flight details like airline, price, and departure date. The status bar at the bottom indicates a 200 OK status and a response time of 8799 ms.

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.

The screenshot shows the Postman interface with a tab for 'http://mu.mulesoft-training.com/essentials/united/flights'. A GET request is made to this URL. The response status is 200 OK and the time taken is 787 ms. The 'Body' tab displays the JSON response, which is identical to the one shown in the Mule Studio screenshot, listing flights from United with various details like price and departure date.

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

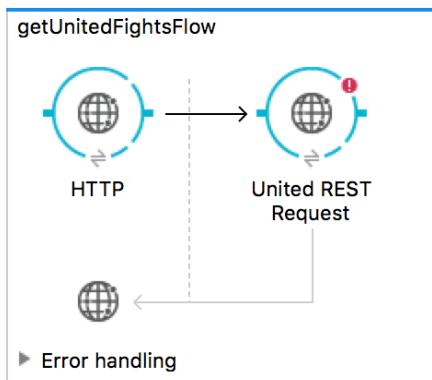
10. In the HTTP properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.
11. Set the path to `/united`.
12. Set the allowed methods to `GET`.

The screenshot shows the Anypoint Studio interface with the 'implementation' flow open. The flow canvas displays a process named 'getUnitedFlightsFlow' containing an 'HTTP' connector. Below the process, there is an 'Error handling' section. At the bottom of the screen, the 'HTTP' properties panel is visible, showing the 'General' tab selected with the path set to '/united' and the allowed methods set to 'GET'. A status message at the top of the properties panel indicates 'There are no errors.'

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.

A screenshot of the Mule Studio interface showing the properties editor. The file being edited is '*flights-DEV.properties'. The content of the file is displayed in a code editor:

```
1 http.port = 8081
2
3 united.host = mu.mulesoft-training.com
```

17. Save the file.

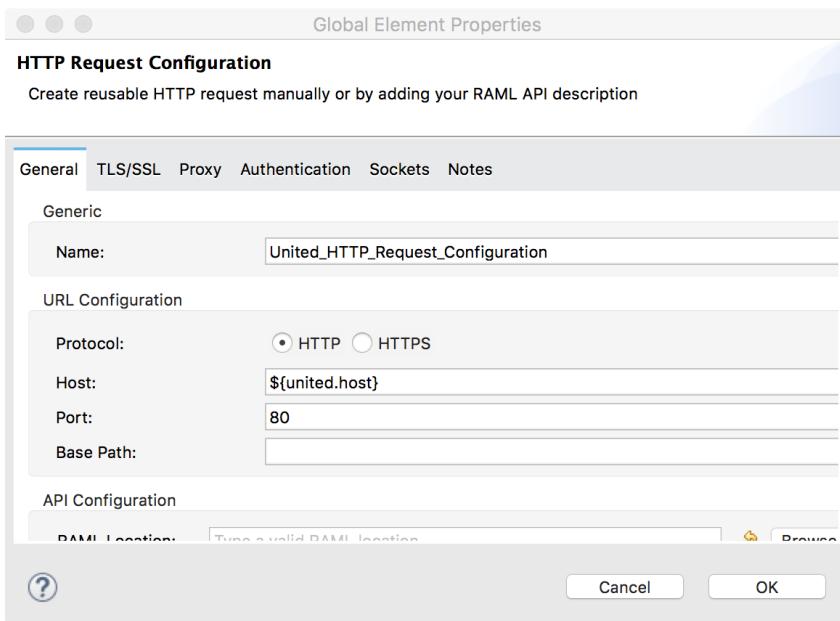
18. Return to global.xml.

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



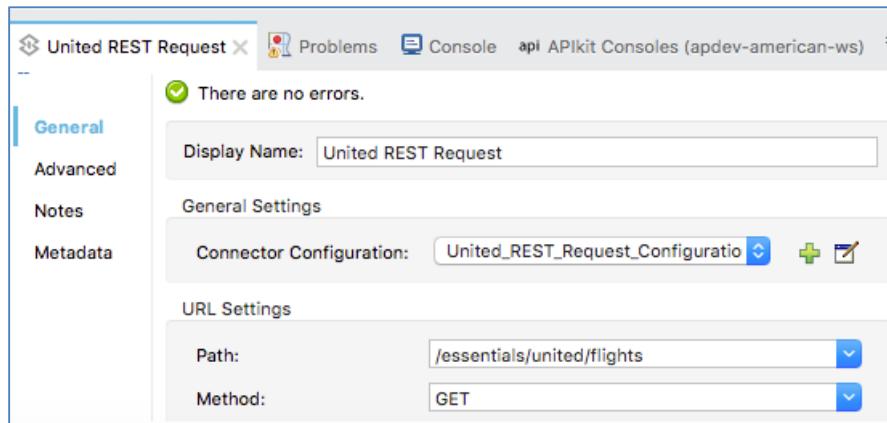
Configure the HTTP Request endpoint

22. Return to implementation.xml.

23. In the United REST Request properties view, set the connector configuration to the existing United_REST_Request_Configuration.

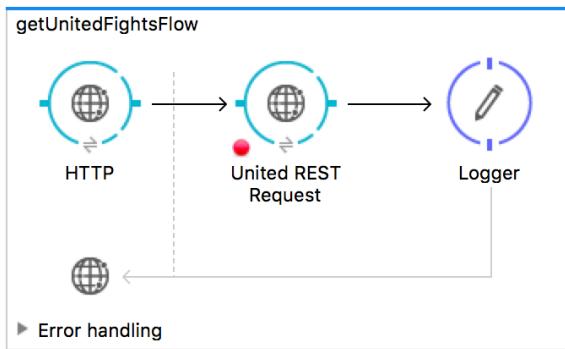
24. Set the path to /essentials/united/flights.

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.

Screenshot of the Mule Debugger interface. The top navigation bar includes tabs for 'Mule Debugger' (selected), 'Console', 'Problems', and 'Mule Properties'. Below the table, there is a status message: 'No messages found. Step through the application or use the "Step" button to start the debugger.'

Name	Value	Type
▶ e DataType	SimpleDataType{type=org.mule.t...}	org.mule.transformer.types.SimpleDataType
▶ a Exception	null	
▶ e Message		org.mule.DefaultMuleMessage
▶ a Message Processor	Transform Message	com.mulesoft.weave.mule.WeaveMessag...
▶ e Payload (mimeType...)	org.glassfish.grizzly.utils.BufferI...	org.glassfish.grizzly.utils.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.

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: localhost:8081/united
- Params: None
- Send and Save buttons
- Body tab selected, showing the JSON response:

```
1 [ { "flights": [ { "airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origin": "MUA", "code": "ER38sd", "emptySeats": 0, "destination": "SFO" }, { "airlineName": "United", "price": 345.99, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origin": "ORD", "code": "ER38sd", "emptySeats": 0, "destination": "SFO" } ] }
```

- Headers tab (3)
- Tests tab
- Status: 200 OK
- Time: 110986 ms

Review the structure for the desired JSON response

35. In Anypoint Studio, open api.raml in src/main/api and review the response example.

36. Open flights-example.json in src/main/resources and review the sample.

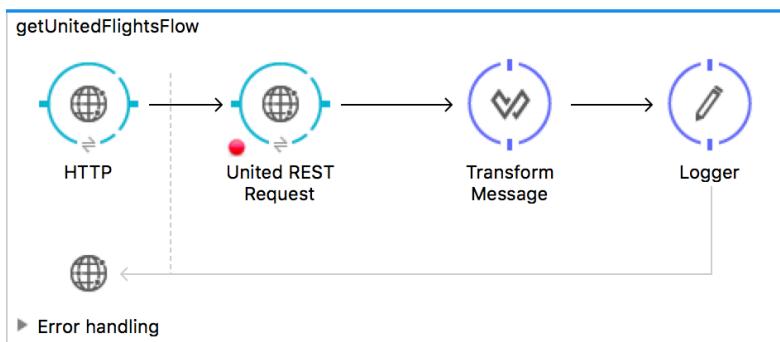
The screenshot shows the contents of the flights-example.json file in Anypoint Studio:

```
[ { "airline": "United", "price": 400, "departureDate": "2015/03/20", "plane": "Boeing 737", "origination": "ORD", "code": "ER38sd", "emptySeats": 0, "destination": "SFO" }, { "airline": "Delta", "price": 345.99, "departureDate": "2015/03/20", "plane": "Boeing 737", "origination": "ORD", "code": "ER38sd", "emptySeats": 0, "destination": "SFO" } ]
```

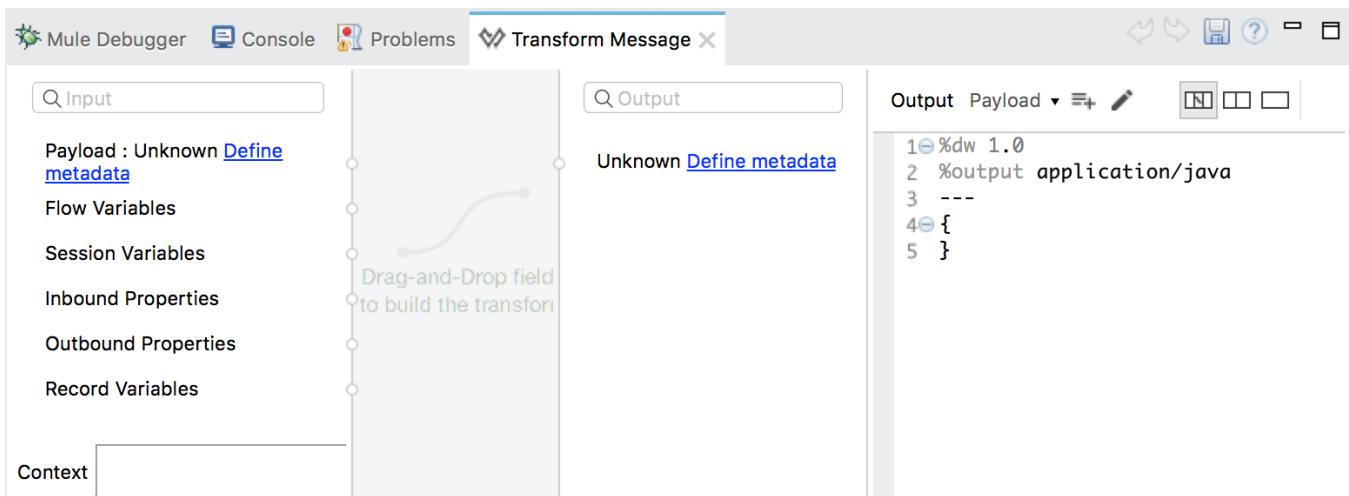
Add input metadata for the transformation

37. Return to implementation.xml.

38. Add a Transform Message component after the United REST Request endpoint.



39. In the input section of the Transform Message properties view, click the Define metadata link for the payload.



40. In the Select metadata type dialog box, click the Add button.

41. In the Create new type button dialog box, set the type id to united_json.

42. Click Create type.

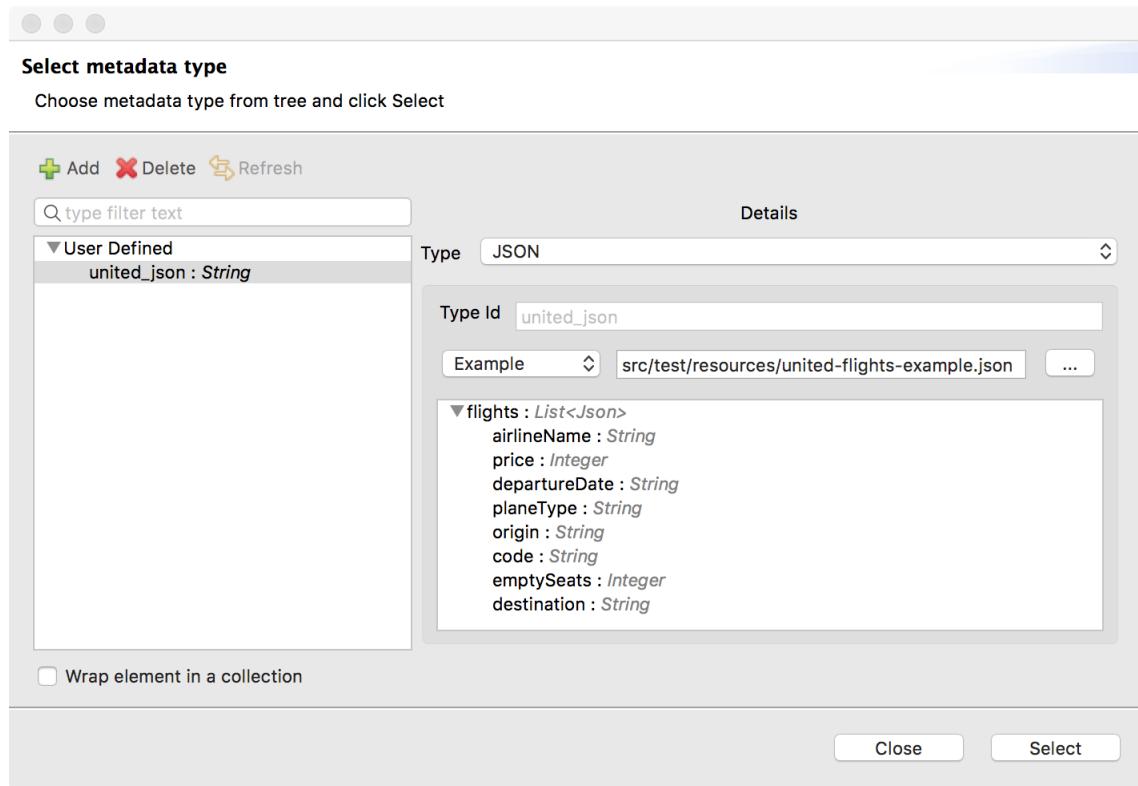
43. In the Select metadata type dialog box, set the type to JSON.

44. Change the Schema selection to Example.

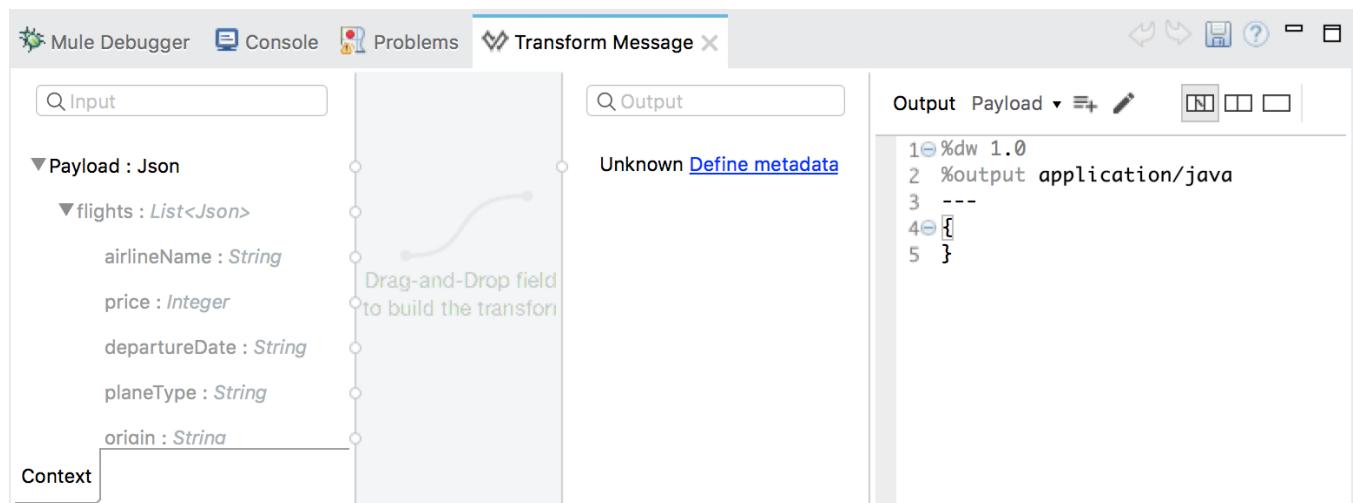
45. Click the browse button.

46. Click the browse button and navigate to the project's src/test/resources folder.

47. Select `united-flights-example.json` and click Open; you should see the example data for the metadata type.



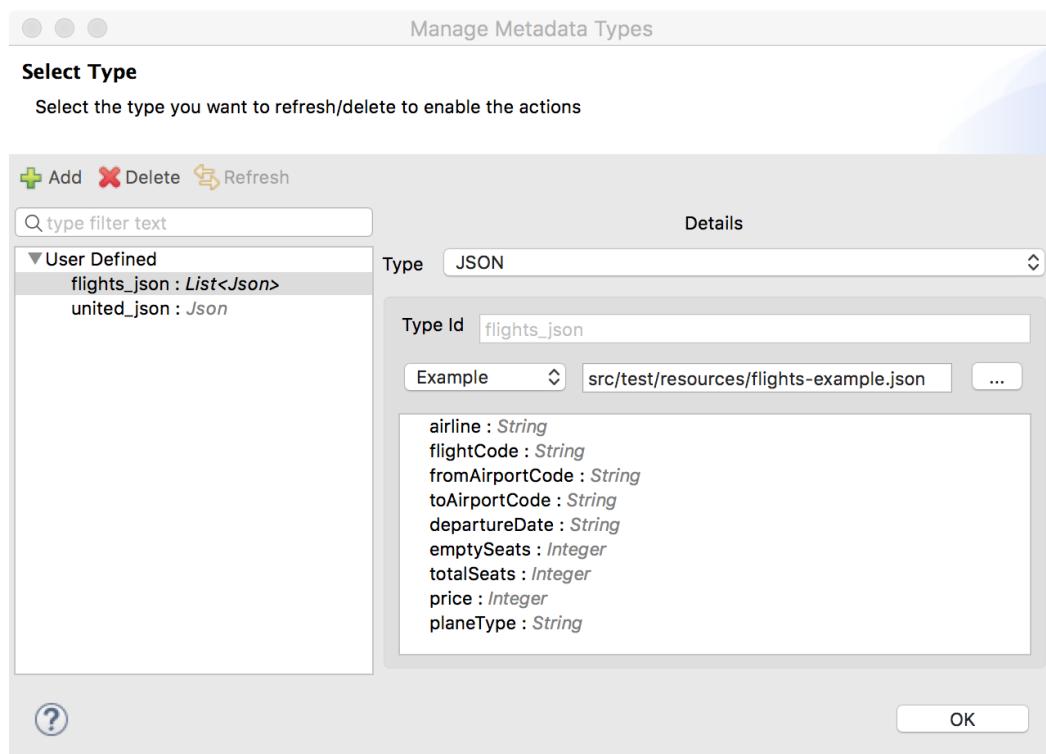
48. Click Select; you should now see output metadata in the input section of the Transform Message properties view.



Add output metadata for the transformation

49. In the output section of the Transform Message properties view, find where the output is set to application/java.
50. Click the Define metadata link.
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 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 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.



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

```
%dw 1.0
%output application/json
---
[]
```

Create the transformation

59. 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.

```
%dw 1.0
%output application/json
---
payload.flights map ((flight , indexOfFlight) -> {
    airline: flight.airlineName,
    flightCode: flight.code,
    fromAirportCode: flight.origin,
    toAirportCode: flight.destination,
    departureDate: flight.departureDate,
    emptySeats: flight.emptySeats,
    price: flight.price,
    planeType: flight.planeType
})
```

Test the application

60. Debug the project.

61. In Postman, make another request to <http://localhost:8081/united>.
62. In the Mule Debugger, step to the Logger; you should see the payload is now a DataWeave ByteArraySeekableStream.

The screenshot shows the Mule Debugger interface with the following details:

- Mule Debugger X**: The active tab.
- Console**, **Problems**, **Mule Properties**: Other tabs.
- Table View** (Name, Value, Type):

Name	Value	Type
» (DataType)	SimpleDataType{typ...	org.mule.transformer.types.SimpleDataType
» (Exception)	null	
» (Message)		org.mule.DefaultMuleMessage
» (Message Processor)	Logger	org.mule.api.processor.LoggerMessageProcessor
» (Payload (mimeType...))	com.mulesoft.weave....	com.mulesoft.weave.reader.ByteArraySeekableStream

63. Step through the application.
64. Return to Postman; you should see the flight data with the different JSON structure.

The screenshot shows the Postman interface with the following details:

- Method**: GET
- URL**: localhost:8081/united
- Params**: None
- Send** and **Save** buttons
- Body** tab selected
- Headers (3)**: None
- Tests**: None
- Status: 200 Time: 8799 ms**
- Pretty**, **Raw**, **Preview**, **JSON** dropdown (set to JSON), **Copy**, **Search** icons
- JSON Response** (Pretty Print):


```

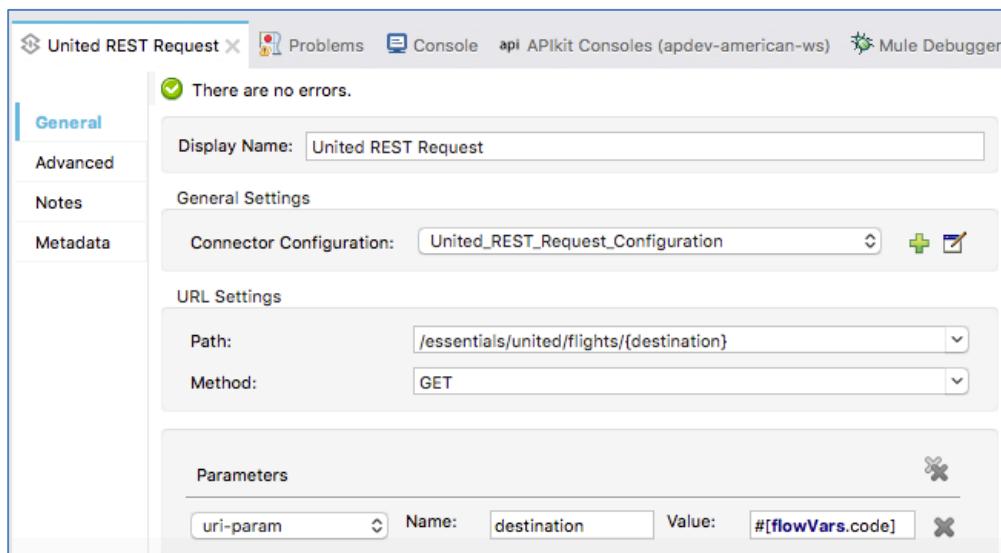
1 [
2   {
3     "airline": "United",
4     "flightCode": "ER38sd",
5     "fromAirportCode": "MUA",
6     "toAirportCode": "SFO",
7     "departureDate": "2015/03/20",
8     "emptySeats": 0,
9     "price": 400,
10    "planeType": "Boeing 737"
11  },
12  {
13    "airline": "United",
14  }
      
```

65. Return to Anypoint Studio and stop the project.

Walkthrough 7-2: 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.

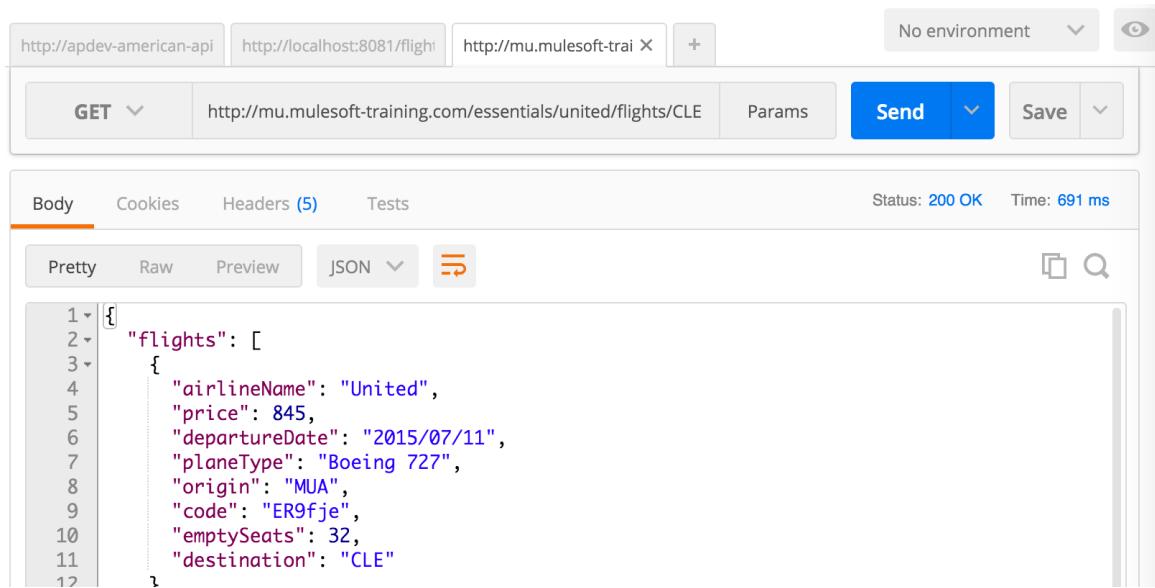


Make a request to the web service specifying a destination

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.

- Send the request; you should see JSON data for only the flights to CLE.



The screenshot shows a REST client interface with the following details:

- Header bar: http://apdev-american-api, http://localhost:8081/flight, http://mu.mulesoft-trai X, No environment
- Method: GET
- URL: http://mu.mulesoft-training.com/essentials/united/flights/CLE
- Buttons: Params, Send, Save
- Body tab selected: Status: 200 OK, Time: 691 ms
- Pretty JSON view:

```
1 [ { "flights": [ { "airlineName": "United", "price": 845, "departureDate": "2015/07/11", "planeType": "Boeing 727", "origin": "MUA", "code": "ER9fje", "emptySeats": 32, "destination": "CLE" } ] }
```
- Raw, Preview, JSON, and copy/paste icons are visible.

- Make additional requests for destinations of LAX, SFO, PDX, or PDF.

Add a URI parameter with a static value

- Return to implementation.xml in Anypoint Studio.
- Double-click the United REST Request endpoint.
- In the Properties view, locate the parameters section; there should be no parameters listed.
- Change the United REST Request path to /essentials/united/flights/{destination}.
- Click the Add Parameter button.
- Select a parameter type of uri-param.
- Set the name to destination.

12. Set the parameter value to SFO.

The screenshot shows the 'United REST Request' configuration in Anypoint Studio. The 'General' tab is active. The 'Connector Configuration' is set to 'United_REST_Request_Configuration'. The 'URL Settings' section shows a 'Path' of '/essentials/united/flights/{destination}' and a 'Method' of 'GET'. In the 'Parameters' section, there is one entry: 'uri-param' with 'Name' 'destination' and 'Value' 'SFO'. There is also an 'Add Parameter' button.

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.

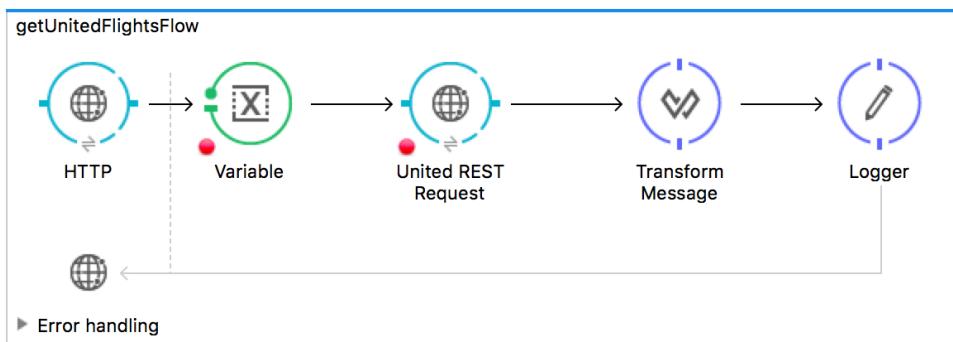
The screenshot shows a POSTMAN request configuration. The method is 'GET'. The URL is 'localhost:8081/united?code=LAX'. In the 'Params' section, there is a single entry: 'code' with 'value' 'LAX'. There is also a 'key' column. A 'Send' button is visible at the top right.

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.

21. Set the operation to Set Variable and the name to code.

22. Set the value to a query parameter called code.

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

General	Display Name: Set airport code variable
Metadata	Operation: <input checked="" type="radio"/> Set Variable <input type="radio"/> Remove Variable
	Name: code
	Value: ##[message.inboundProperties.'http.query.params'.code]

Change the REST request URI parameter to a dynamic value

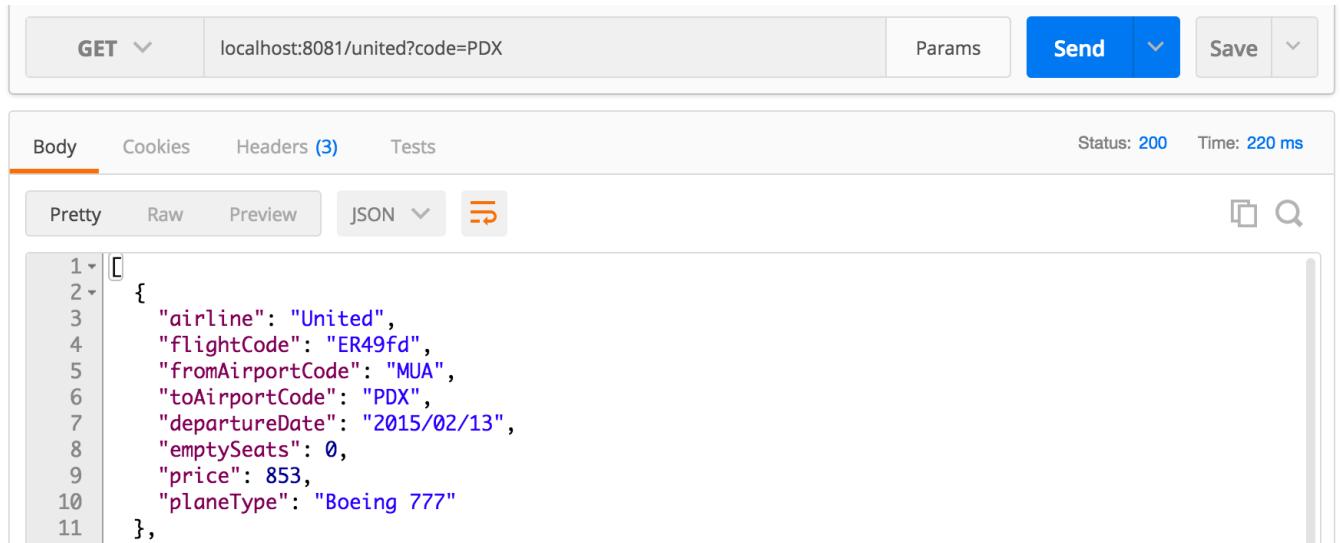
23. In the United REST Request properties view, change the value of the uri-param from SFO to the value of the flow variable containing the airport code.

```
##[flowVars.code]
```

Parameters	X		
uri-param	Name: destination	Value: ##[flowVars.code]	X
Add Parameter			

Test the application

24. Save and redeploy the application.
25. In Postman, send the same request again; this time you should only get flights to LAX.
26. Change the code to PDX and make the request; you should now see flights to PDX.



POST <localhost:8081/united?code=PDX> Params Send Save

Body Cookies Headers (3) Tests Status: 200 Time: 220 ms

Pretty Raw Preview JSON `🔗` `🔍`

```
1 [  
2 {  
3   "airline": "United",  
4   "flightCode": "ER49fd",  
5   "fromAirportCode": "MUA",  
6   "toAirportCode": "PDX",  
7   "departureDate": "2015/02/13",  
8   "emptySeats": 0,  
9   "price": 853,  
10  "planeType": "Boeing 777"  
11 }]
```

27. Remove the code parameter and make the request; you should get a 500 responses and an exception.



POST <localhost:8081/united> Params Send Save

Body Cookies Headers (3) Tests Status: 500 Expression {destination} evaluated to null. (java.lang.NullPointerException). Time: 15 ms

Pretty Raw Preview HTML `🔗` `🔍`

```
i 1 Expression {destination} evaluated to null. (java.lang.NullPointerException).
```

Modify the set airport code flow variable to assign a default value

28. Return to Anypoint Studio.

29. 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.

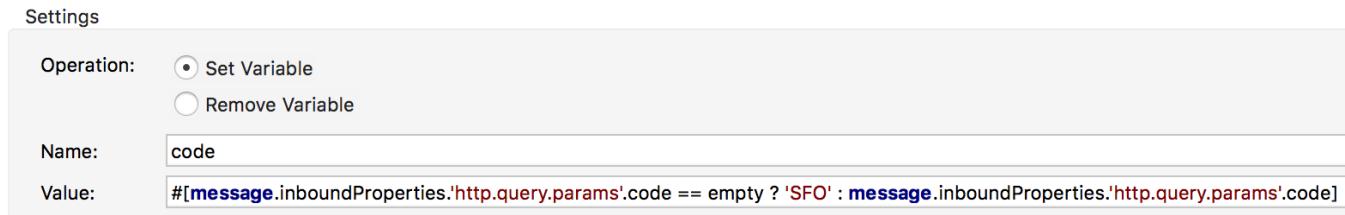
```
#[(message.inboundProperties.'http.query.params'.code == empty) ?  
'SFO' : message.inboundProperties.'http.query.params'.code]
```

Settings

Operation: Set Variable
 Remove Variable

Name: code

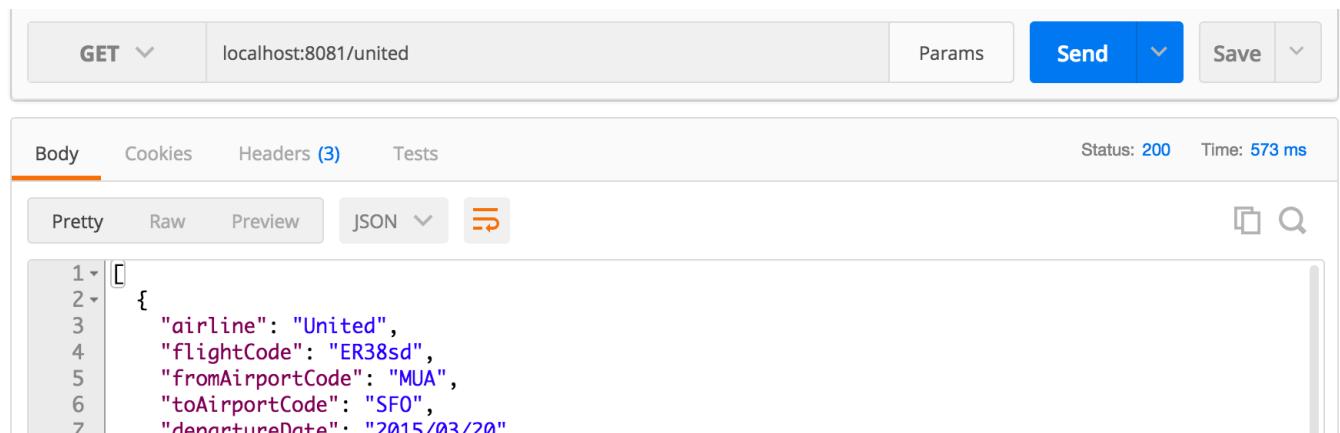
Value: #[message.inboundProperties.'http.query.params'.code == empty ? 'SFO' : message.inboundProperties.'http.query.params'.code]



Test the application

30. Save and redeploy the application.

31. In Postman, send the same request again with no query parameter; this time you should get flights to SFO instead of an exception.



GET localhost:8081/united Params Send Save

Status: 200 Time: 573 ms

Body Cookies Headers (3) Tests

Pretty Raw Preview JSON ↻

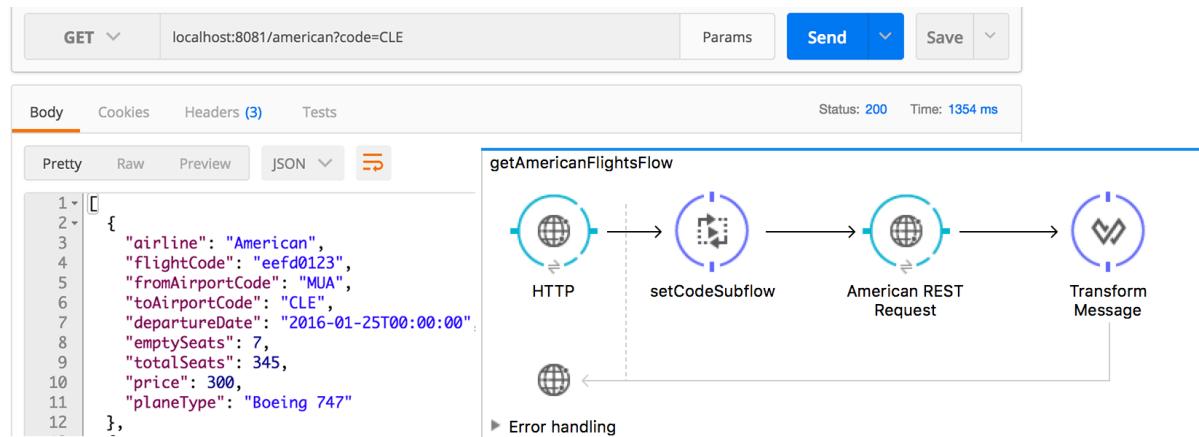
```
[{"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20"}]
```

32. Return to Anypoint Studio and stop the project.

Walkthrough 7-3: Consume a RESTful web service that has a RAML definition

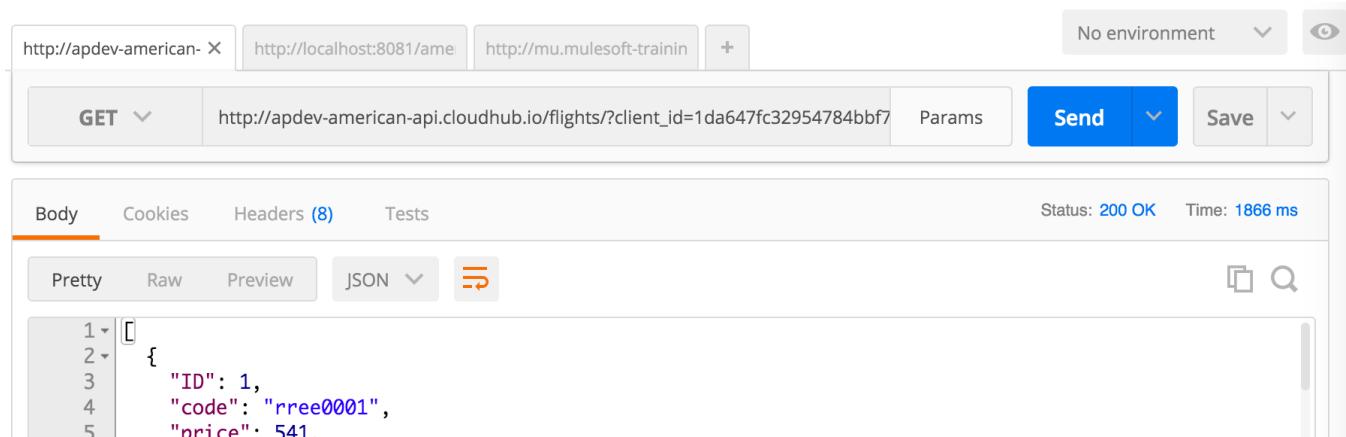
In this walkthrough, you consume the American flights RESTful web service that you built and deployed to the cloud. You will:

- 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.



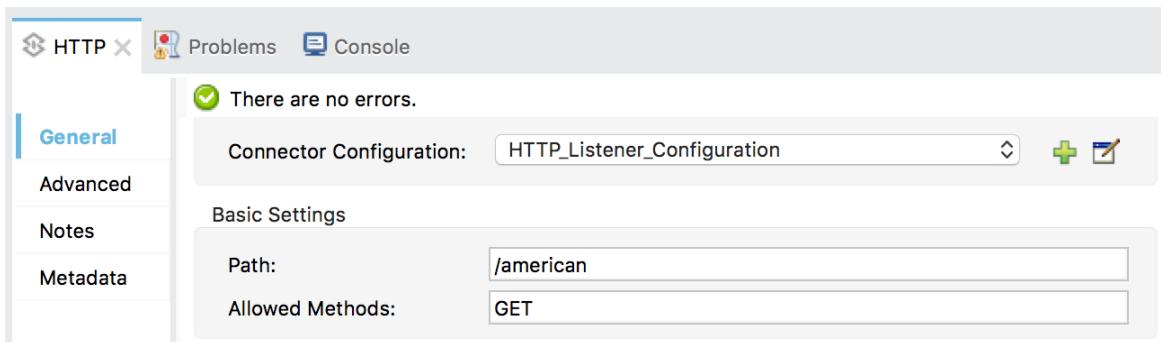
Make a request to the web service

1. In Postman, return to the first tab – the one with the request to your American Flights API <http://apdev-american-api-{lastname#}.cloudbhub.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.



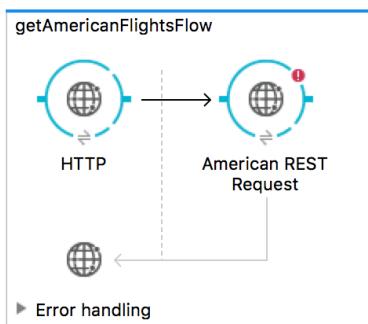
Add a new flow with an HTTP Listener endpoint

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

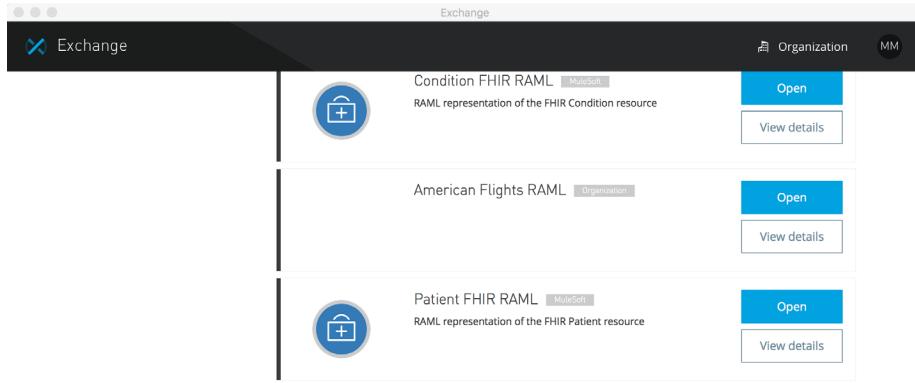
9. Drag out another HTTP connector and drop it in the process section of the new flow.
10. Change the endpoint display name to American REST Request.



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.

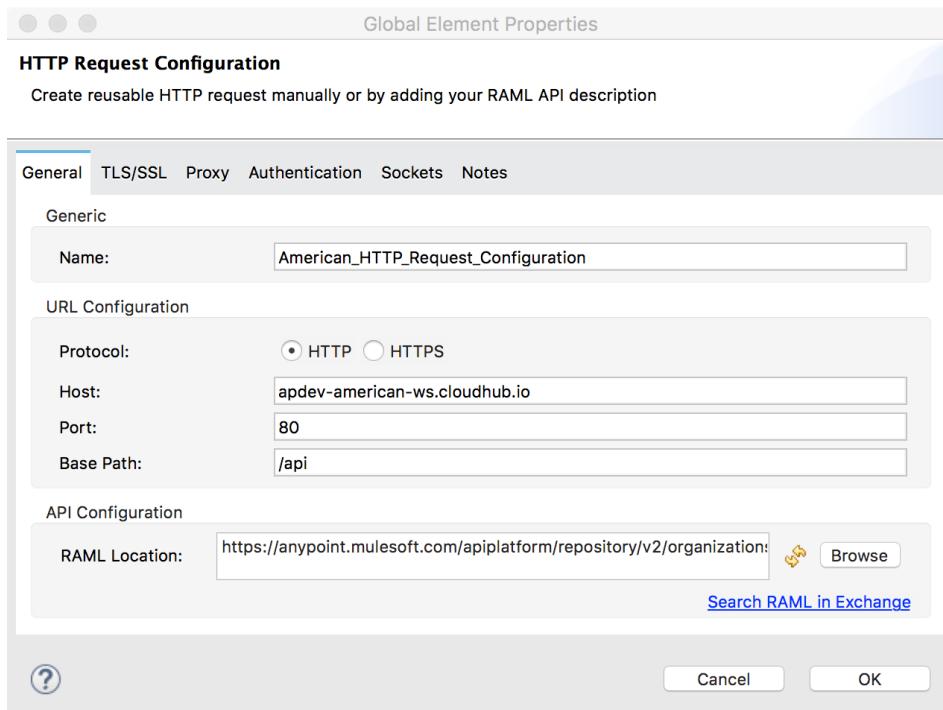
14. In the Global Element Properties dialog box, change the name to American_HTTP_Request_Configuration.
15. Click the Search RAML in Exchange link next to RAML location.
16. In the Exchange window that opens, scroll down and locate your American Flights RAML.



17. Click Open; 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.

Note: If the fields did not populate, click the Reload RAML button next to the RAML location.

Note: If you did not successfully create the RAML in the Exchange or did not successfully deploy the web service and API to the cloud, use the Solution - American Flights RAML URL located in the course snippets.txt file.



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.

Connector Configuration: American_HTTP_Request_Configuration +

URL Settings

Path: /flights/{ID}

Method:

- DELETE
- GET
- PUT

Parameters

25. Set the path to /flights.
26. Look at the method drop-down menu; you should see GET and POST.
27. Set the method to GET.

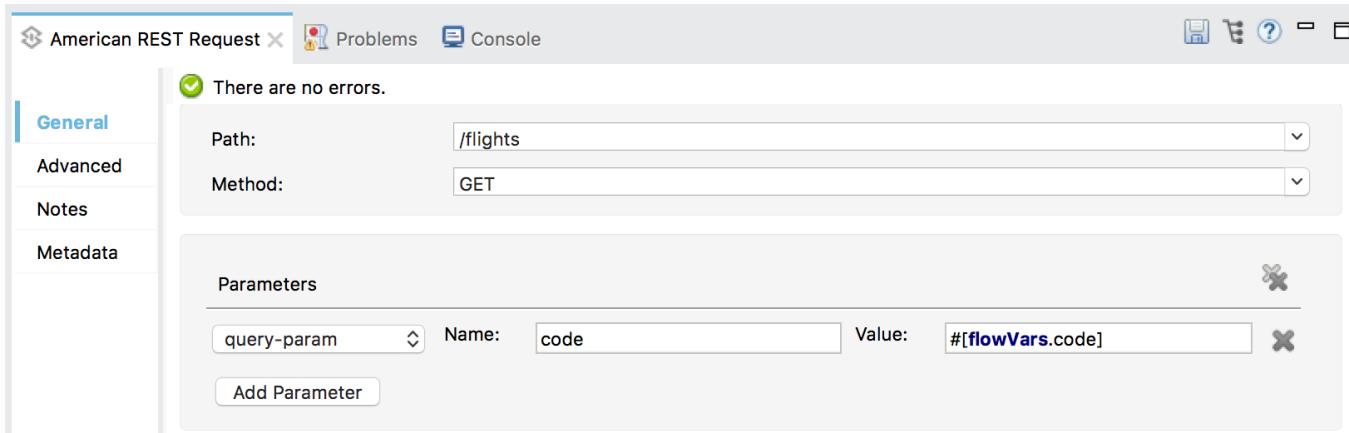
Connector Configuration: American_HTTP_Request_Configuration +

URL Settings

Path: /flights

Method: GET

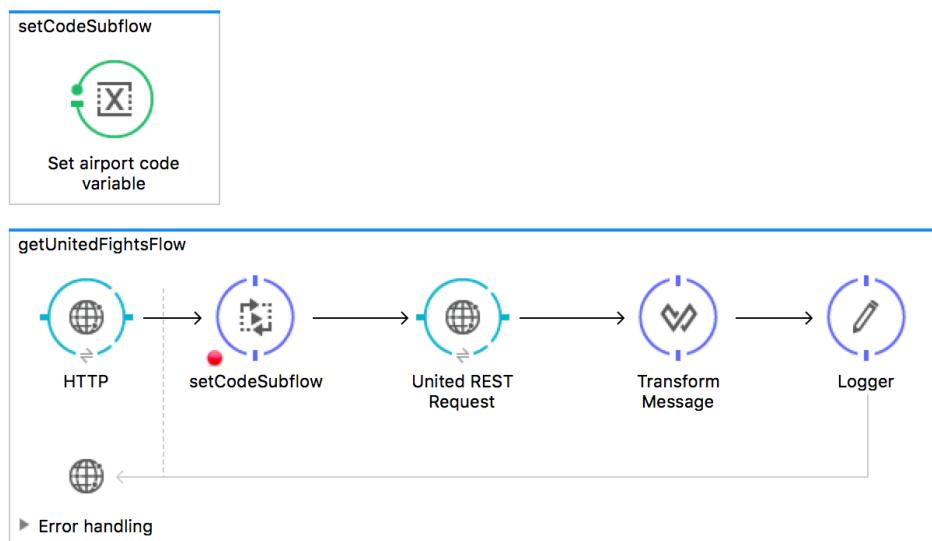
28. Scroll down and check to see if a query parameter called code has been added.
29. If the destination parameter was not automatically created, create it.
30. Set the value to a flow variable called code; you will add this to the flow next.



Extract the Set airport code variable processor into a subflow and use it

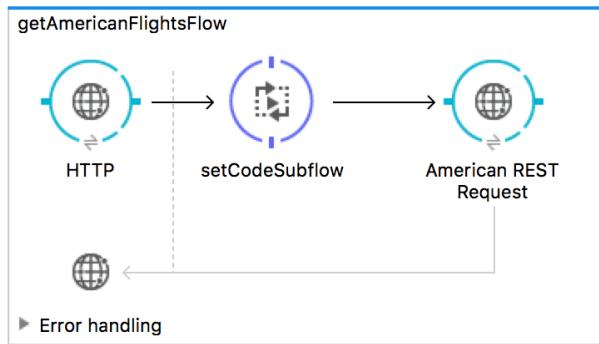
31. Right-click the Set airport code variable processor in getUnitedFlightsFlow and select Extract to > Sub Flow.
32. In the Extract Flow dialog box, set the flow name to setCodeSubflow and click OK.
33. Double-click the new Flow Reference in getUnitedFlightsFlow; the display name should update.
34. 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.

```
Body Cookies Headers (3) Tests Status: 200 OK Time: 1649 ms

Pretty Raw Preview JSON 🔍

1  {
2    "ID": 2,
3    "code": "eefd0123",
4    "price": 300,
5    "departureDate": "2016-01-25T00:00:00",
6    "origin": "MUA",
7    "destination": "CLE",
8    "emptySeats": 7,
9    "plane": {
10      "type": "Boeing 747",
11      "totalSeats": 345
12    }
13  },
14 }
```

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.

```
%dw 1.0
%output application/json
---
payload map ((payload01 , indexOfPayload01) -> {
  flightCode: payload01.code,
  fromAirportCode: payload01.origin,
  toAirportCode: payload01.destination,
  departureDate: payload01.departureDate,
  emptySeats: payload01.emptySeats,
  totalSeats: payload01.plane.totalSeats,
  price: payload01.price,
  planeType: payload01.plane.type
})
```

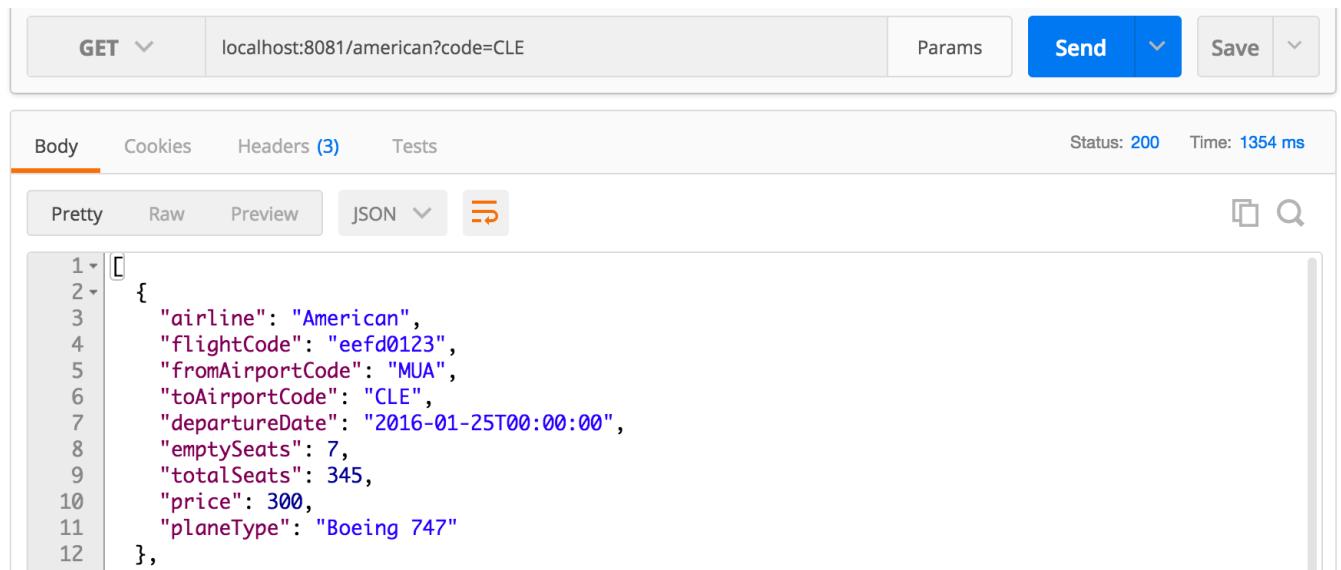
50. Double-click the airline field in the output section.
51. In the generated DataWeave expression, set airline to "American".

```
%dw 1.0
%output application/json
---
payload map ((payload01 , indexOfPayload01) -> {
  airline: "American",
  flightCode: payload01.code,
  fromAirportCode: payload01.origin,
  toAirportCode: payload01.destination
})
```

Test the application

52. Save to redeploy the project.

53. In Postman, make another request to <http://localhost:8081/american>; you should see all the flight data as JSON again but now with a different structure.



The screenshot shows the Postman interface with a GET request to `localhost:8081/american?code=CLE`. The response body is displayed in a JSONpretty format, showing a single flight record:

```
1 [ ]  
2 {  
3   "airline": "American",  
4   "flightCode": "eefd0123",  
5   "fromAirportCode": "MUA",  
6   "toAirportCode": "CLE",  
7   "departureDate": "2016-01-25T00:00:00",  
8   "emptySeats": 7,  
9   "totalSeats": 345,  
10  "price": 300,  
11  "planeType": "Boeing 747"  
12 },
```

The status is 200 and the time taken is 1354 ms.

Walkthrough 7-4: 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.

The screenshot shows a Postman interface. At the top, a GET request is made to `localhost:8081/delta`. The response status is 200 and the time taken is 653 ms. The Body tab displays a JSON response:

```
1 {  
2   "airline": "Delta",  
3   "flightCode": "A1B2C3",  
4   "fromAirportCode": "MUA",  
5   "toAirportCode": "SFO",  
6   "departureDate": "2015/03/20",  
7   "emptySeats": "40",  
8   "price": "400.0",  
9   "planeType": "Boing 737"  
10 }  
11 }
```

To the right of the response, a flow diagram titled "getDeltaFlightsFlow" is shown. It consists of four main components connected sequentially: "HTTP" (represented by a globe icon), "Delta SOAP Request" (represented by an envelope icon with a red dot), "Transform Message" (represented by a diamond icon with a checkmark), and "Logger" (represented by a pen icon). An "Error handling" section is also indicated at the bottom.

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`.

The screenshot shows a Postman interface. A GET request is made to `http://mu.mulesoft-training.com/essentials/delta?wsdl`. The response status is 200 and the time taken is 400 ms. The Body tab displays the WSDL XML:

```
1 <?xml version='1.0' encoding='UTF-8'?>  
2 <wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://soap.training.mulesoft.com/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="TicketServiceService" targetNamespace="http://soap.training.mulesoft.com//">  
3   <wsdl:types>  
4     <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://soap.training.mulesoft.com/" elementFormDefault="unqualified" targetNamespace="http://soap.training.mulesoft.com/" version="1.0">  
5       <xss:element name="findFlight" type="tns:findFlight"/>  
6       <xss:element name="findFlightResponse" type="tns:findFlightResponse"/>  
7       <xss:element name="listAllFlights" type="tns:listAllFlights"/>  
8       <xss:element name="listAllFlightsResponse" type="tns:listAllFlightsResponse"/>
```

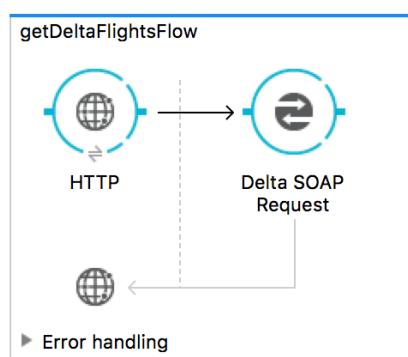
Create a new flow with an HTTP Listener connector endpoint

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.

The screenshot shows the Anypoint Studio interface. At the top, there's a title bar with the flow name 'getDeltaFlightsFlow'. Below it is a process canvas with an 'HTTP' connector icon. To the right is a properties panel for the 'HTTP' connector, showing the path is set to '/delta' and the allowed methods are 'GET'. The properties panel also indicates 'There are no errors'. At the bottom, there are tabs for 'Message Flow', 'Global Elements', and 'Configuration XML'.

Add a Web Service Consumer connector endpoint

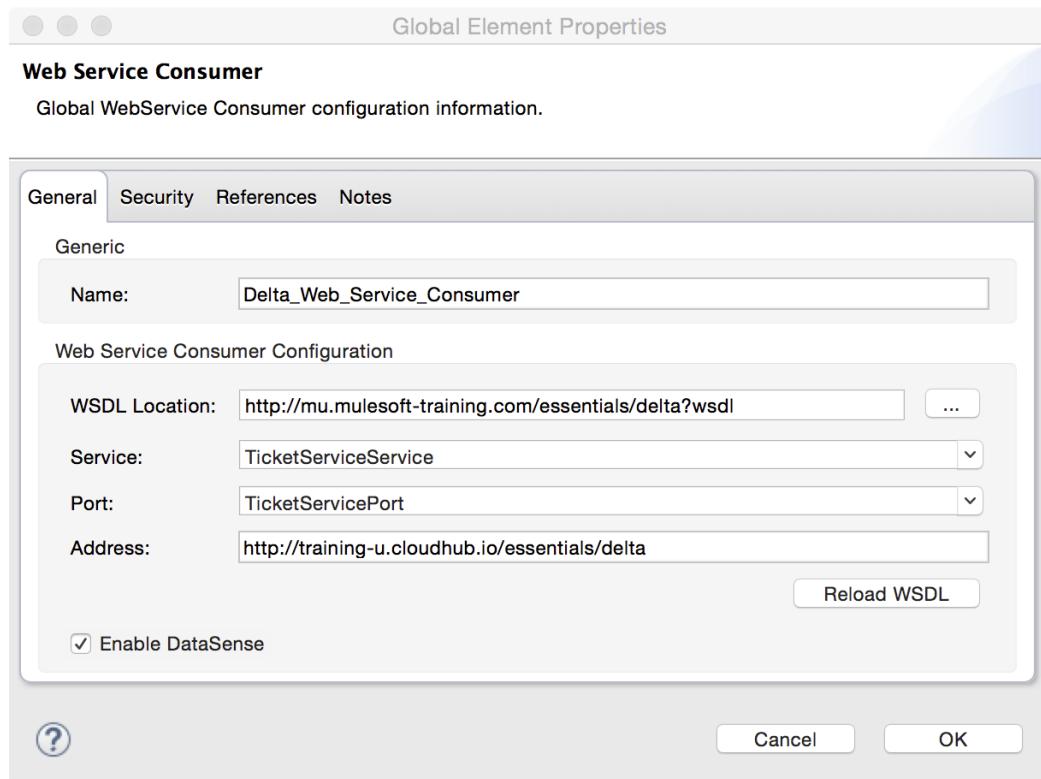
11. Drag out a Web Service Consumer connector and drop it in the process section of the flow.
12. Change its display name to Delta SOAP Request.



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.

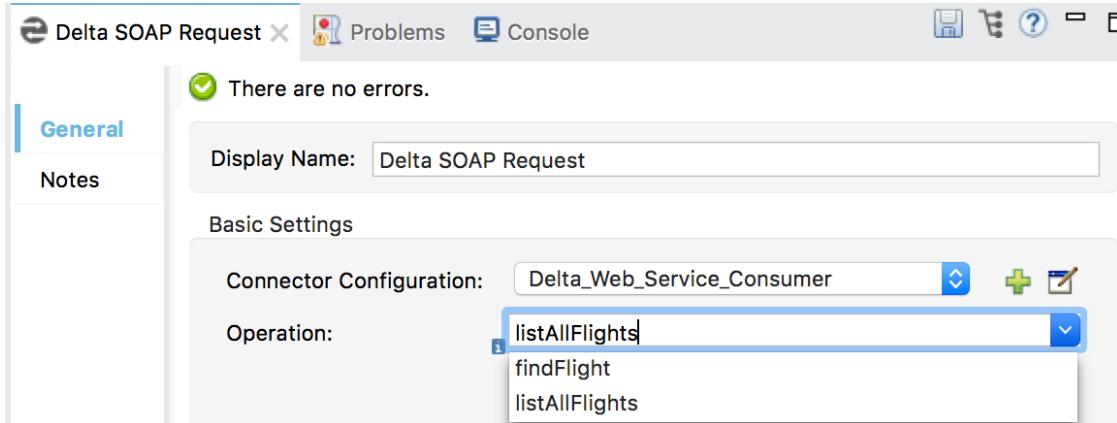


19. Click OK.

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.

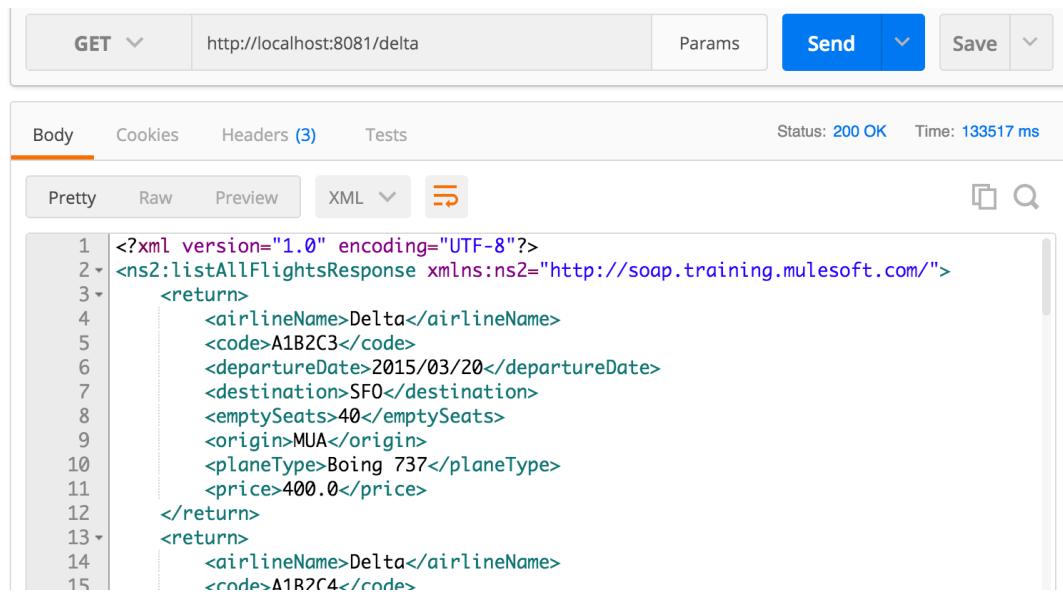
Test the application

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.

Mule Debugger	Console	Problems	Mule Properties
Variables			
Name	Value	Type	
► [E] DataType	SimpleDataType{type=org.ap...	org.mule.transformer.types.SimpleDataType	
@ Exception	null		
► [E] Message		org.mule.DefaultMuleMessage	
@ Message Processor	Logger	org.mule.api.processor.LoggerMessageProcessor	
► [E] Payload (mimeType="text/xml",...	org.mule.module.ws.consume...	org.apache.cxf.staxutils.DepthXMLStreamReader	

30. Step through the application.

31. Return to Postman; you should see the XML flight data returned.



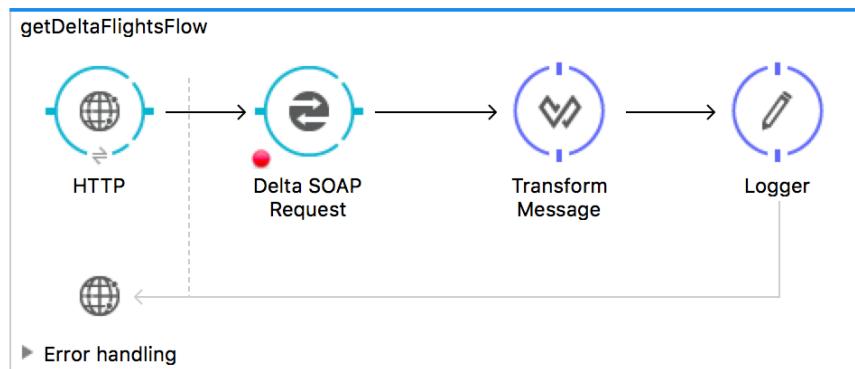
The screenshot shows the Postman interface with a successful HTTP request. The URL is `http://localhost:8081/delta`. The response status is `200 OK` and the time taken is `133517 ms`. The response body is displayed in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
    <return>
        <airlineName>Delta</airlineName>
        <code>A1B2C3</code>
        <departureDate>2015/03/20</departureDate>
        <destination>SFO</destination>
        <emptySeats>40</emptySeats>
        <origin>MUA</origin>
        <planeType>Boing 737</planeType>
        <price>400.0</price>
    </return>
    <return>
        <airlineName>Delta</airlineName>
        <code>A1B2C4</code>
```

Transform the data

32. Return to Anypoint Studio and stop the project.

33. In `getDeltaFlightsFlow`, add a Transform Message component after the Delta SOAP Request endpoint.



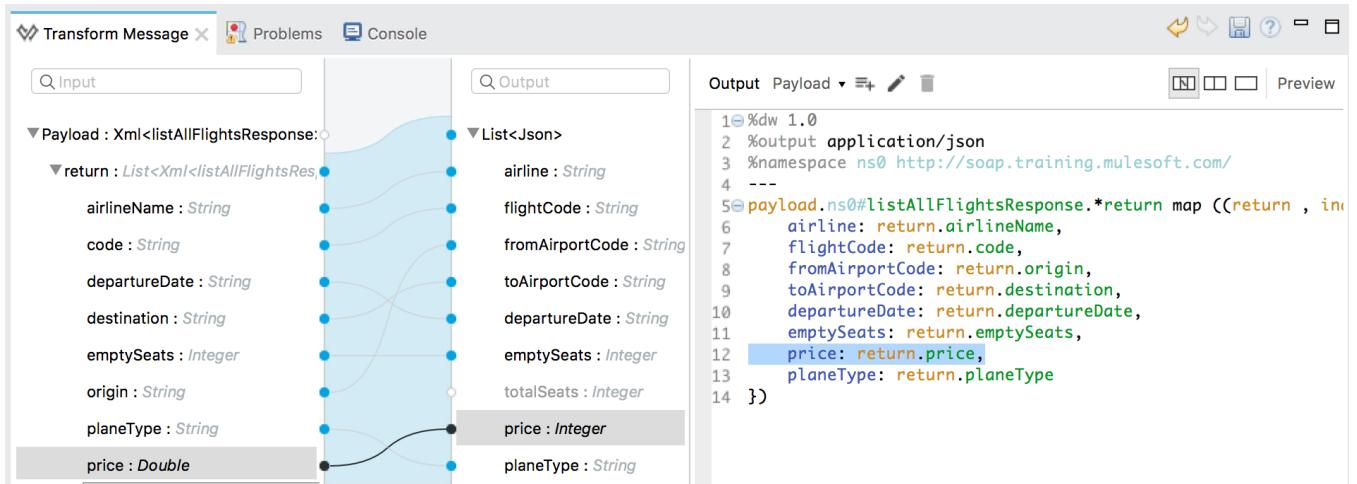
34. Look at the input section of the Transform Message properties view; you should see metadata already defined.

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 `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.

The screenshot shows a Postman request to 'localhost:8081/delta'. The response body is displayed in JSON format, showing a list of flight details. The response status is 200 and the time taken is 1513 ms.

```
[{"airline": "Delta", "flightCode": "A1B2C3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20"}]
```

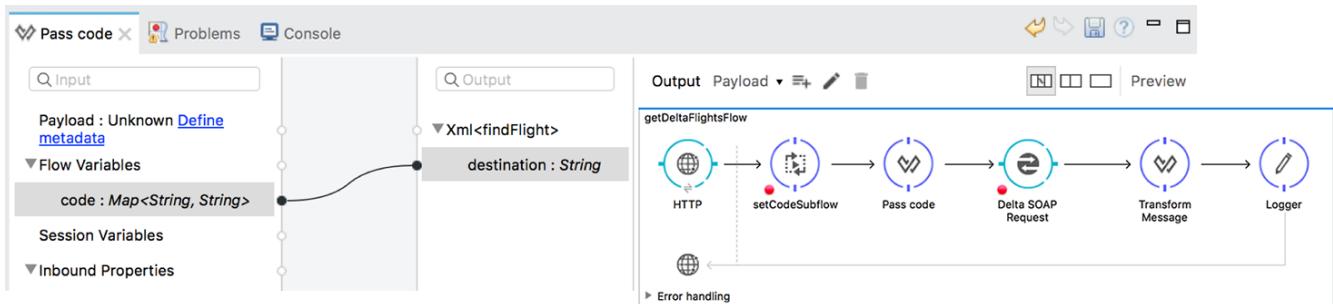
41. Add a query parameter called code and set it equal to LAX.

42. Send the request; you should still get all flights.

Walkthrough 7-5: 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.

- In Postman, send the same request with the query parameter; you should get a 500 response with a message about unknown parameters.

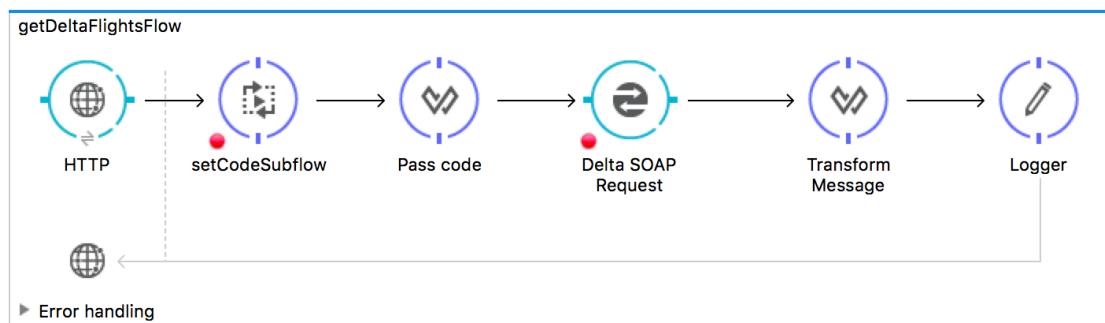
The screenshot shows a Postman interface. At the top, there's a header with 'GET' selected, the URL 'localhost:8081/delta?code=LAX', and a 'Send' button. Below the header, there's a status bar with 'Status: 500 No binding operation info while invoking unknown method with params unknown...' and a time of '734 ms'. The main area is titled 'Body' and contains a single line of text: 'No binding operation info while invoking unknown method with params unknown...'. There are tabs for 'Cookies', 'Headers (3)', 'Tests', 'Pretty', 'Raw', 'Preview', and 'HTML'.

Use the set airport code subflow

- Return to getDeltaFlightsFlow.
- Add a Flow Reference component before the Delta REST Request endpoint.
- In the Flow Reference properties view, set the flow name to setCodeSubflow.

Use DataWeave to pass parameters to the web service

- Add a Transform Message component to the left of the Delta SOAP Request endpoint.
- Change its display name to Pass code.



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

The screenshot shows the 'Pass code' component properties in Anypoint Studio. The 'Input' tab is active, showing 'Payload : Unknown Define metadata'. The 'Output' tab is also visible. In the 'Input' section, there is a 'code : Map<String, String>' variable. In the 'Output' section, there is a 'destination : String' variable. A connection line links the 'code' variable in the input to the 'destination' variable in the output. The code editor on the right shows the DataWeave script:

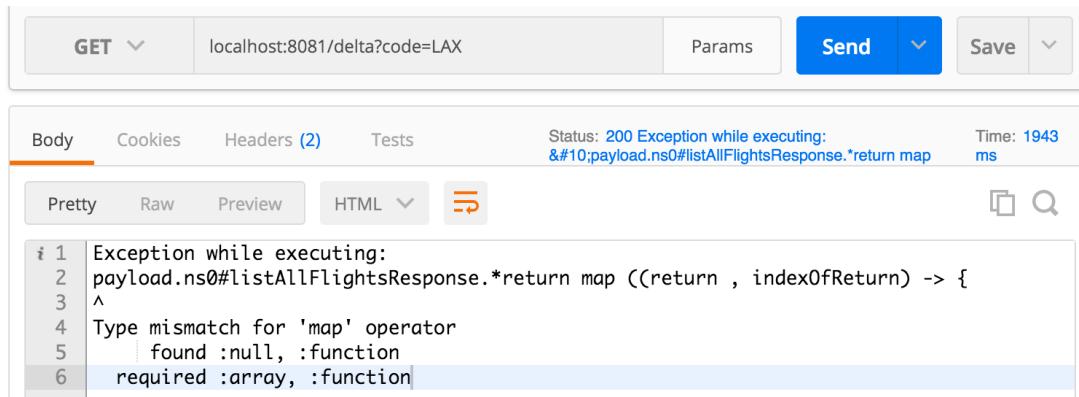
```

1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 --
5 {
6     ns0#findFlight: {
7         destination: flowVars.code as :string
8     }
9 }

```

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.



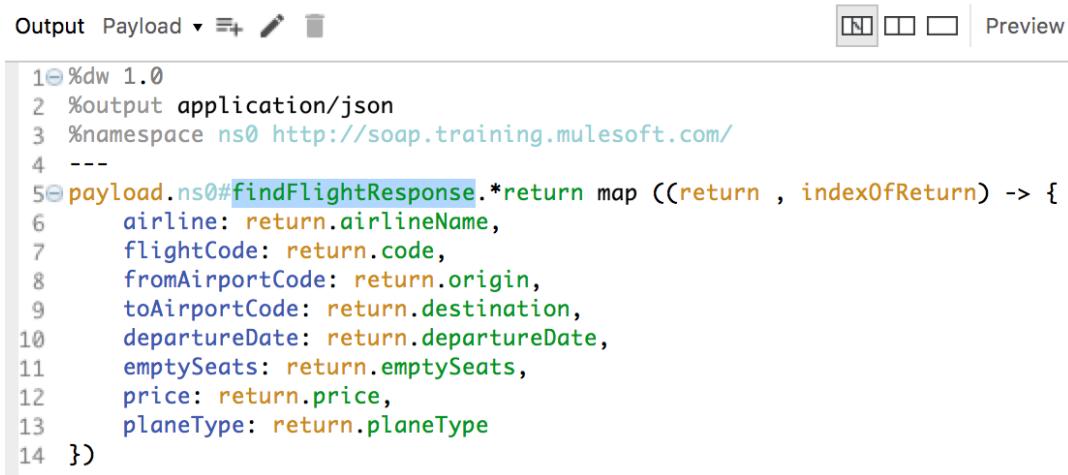
A screenshot of the Postman application interface. At the top, there's a header with 'GET' selected, the URL 'localhost:8081/delta?code=LAX', a 'Params' button, a 'Send' button, and a 'Save' button. Below the header is a status bar showing 'Status: 200 Exception while executing:
payload.ns0#listAllFlightsResponse.*return map' and 'Time: 1943 ms'. The main area is divided into tabs: 'Body' (which is active), 'Cookies', 'Headers (2)', and 'Tests'. Under the 'Body' tab, there are buttons for 'Pretty', 'Raw', 'Preview', and 'HTML'. The 'Pretty' view shows the following error message:

```
i 1 Exception while executing:  
2 payload.ns0#listAllFlightsResponse.*return map ((return , indexOfReturn) -> {  
3 ^  
4 Type mismatch for 'map' operator  
5 | found :null, :function  
6 required :array, :function|
```

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.



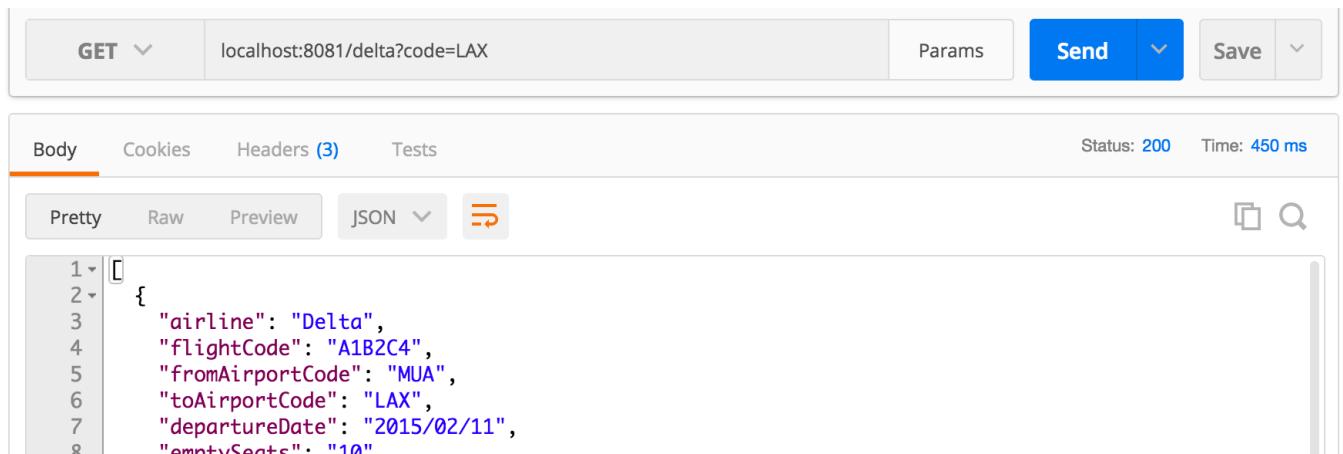
A screenshot of the MuleSoft Anypoint Studio DataWeave editor. The top bar includes 'Output', 'Payload', and a toolbar with icons for copy, paste, and preview. To the right are buttons for 'Text', 'CSV', 'JSON', and 'Preview'. The main code area contains the following DataWeave code:

```
1 %dw 1.0
2 %output application/json
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {
6     airline: return.airlineName,
7     flightCode: return.code,
8     fromAirportCode: return.origin,
9     toAirportCode: return.destination,
10    departureDate: return.departureDate,
11    emptySeats: return.emptySeats,
12    price: return.price,
13    planeType: return.planeType
14 })
```

Test the application

19. Redeploy the application.

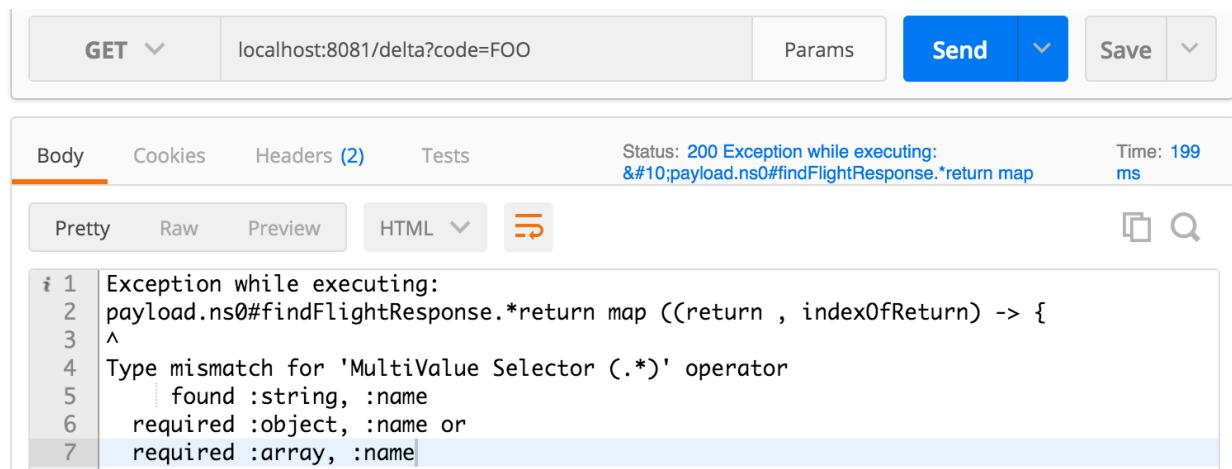
20. In Postman, make the same request; you should now get all the flights to LAX.



The screenshot shows a Postman request configuration and its response. The request method is GET, the URL is `localhost:8081/delta?code=LAX`, and the response status is 200 ms. The response body is a JSON object:

```
1 [  
2 {  
3   "airline": "Delta",  
4   "flightCode": "A1B2C4",  
5   "fromAirportCode": "MUA",  
6   "toAirportCode": "LAX",  
7   "departureDate": "2015/02/11",  
8   "emptySeats": "10"  
9 ]
```

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.



The screenshot shows a Postman request configuration and its response. The request method is GET, the URL is `localhost:8081/delta?code=FOO`, and the response status is 200 ms. The response body contains an exception message:

```
i 1 Exception while executing:  
2 payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {  
3   ^  
4     Type mismatch for 'MultiValue Selector (*.*)' operator  
5       | found :string, :name  
6       required :object, :name or  
7       required :array, :name|
```

25. Return to Anypoint Studio and stop the project.