

- At the top of the Anypoint Platform dialog box, set the application name to training4-american-ws-{your-lastname} so it is a unique value.

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

The screenshot shows the 'Deploying Application' dialog box. The application name dropdown contains 'training4-american-ws-mule' with a green checkmark. The deployment target dropdown is set to 'CloudHub'. The application file dropdown contains 'training4-american-ws.jar'.

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

Note: If you don't know what version it is using, look at the Package Explorer and find a library folder with the name of the server being used, like Mule Server 4.1.1 EE.

- Make sure the worker size to 0.1 vCores.

Runtime	Properties	Insight	Logging	Static IPs
Runtime version 4.1.1	Worker size 0.1 vCores	Workers 1		

- Click the Deploy Application button.
- Click the Open in Browser button.

The screenshot shows a deployment progress bar with the message 'Deploying training-american-ws to CloudHub'. Below the progress bar are two buttons: 'Open in Browser' and 'Close Window'.

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

The screenshot shows the Runtime Manager interface. A deployment for 'training4-american-ws-mule' is listed in the table, showing 'CloudHub' as the server, 'Deploying' as the status, and the file 'training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar'.

Name	Server	Status	File
training4-american-ws-mule	CloudHub	Deploying	training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar

Watch the logs and wait for the application to start

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

The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with options like Applications, Servers, Alerts, VPCs, and Load Balancers. The main area has a search bar and a table with columns: Name, Server, Status, and File. A single row is selected: 'training4-american-ws-mu' running on 'CloudHub' with status 'Started'. To the right, a detailed view of the application 'training4-american-ws-1.0.0-SNAPSHOT' is shown. It includes a 'CloudHub' icon, a green 'Started' button, and a 'Choose file' button. Below that, it lists 'Last Updated' (2018-04-18 1:48:21PM), 'App url' (training4-american-ws-mule.cloudhub.io), 'Runtime version' (4.1.1), 'Worker size' (0.1 vCores), and 'Workers' (1). At the bottom, there are buttons for 'Manage Application', 'Logs', and 'Insight', along with a link to 'View Associated Alerts'.

11. Click the Logs button.
12. Watch the logs as the application is deployed.
13. Wait until the application starts (or fails to start).

The screenshot shows the Runtime Manager interface with the 'Logs' option selected in the sidebar. The main area displays the log console for the application 'training4-american-ws-mule'. The logs show the deployment process: 'Starting Bean: listener' followed by deployment logs for 'Worker-0'. The logs indicate the application started successfully at 13:48:21.955. The right side of the screen shows a 'Deployments' panel with a list of recent deployments, including one from today at 13:46. It also shows a 'System Log' and a 'Worker-0' entry.

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

Test the application

14. In the left-side navigation, select Dashboard.
15. Locate the link for the application on its new domain:
training4-american-ws-{lastname}.cloudbhub.io.

The screenshot shows the 'Runtime Manager' interface. At the top, there's a dark header with a menu icon, a search icon, and the text 'Runtime Manager'. Below the header, on the left, is a sidebar with three tabs: 'SANDBOX' (selected), 'Applications' (with a back arrow), and 'Dashboard' (selected). To the right of the sidebar, the main area displays a single application entry: '● training4-american-ws-mule'. Below this entry, it says 'Domain: training4-american-ws-mule.cloudbhub.io'. There are also links for 'Mule messages' and a small 'x' icon.

16. Click the link; a request will be made to that URL in a new browser tab and you should get a message that there is no listener for that endpoint.
17. Modify the path to <http://training4-american-ws-{lastname}.cloudbhub.io/api/flights>; you should see the flights data.

The screenshot shows a browser window with the address bar containing '<http://training4-american-ws-mule.cloudbhub.io/api/flights>'. The page content displays a JSON array of flight data:

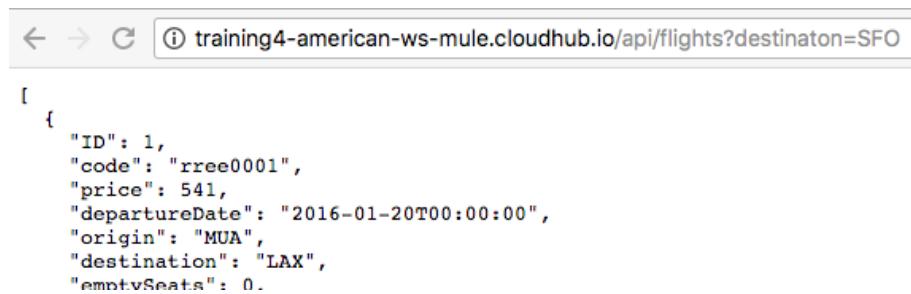
```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2, "code": "eefd0123", "price": 300, "departureDate": "2016-01-25T00:00:00"}]
```

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

18. Add a query parameter called destination to the URL and set it equal to SFO.

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

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



```
[  
  {  
    "ID": 1,  
    "code": "rree0001",  
    "price": 541,  
    "departureDate": "2016-01-20T00:00:00",  
    "origin": "MUA",  
    "destination": "LAX",  
    "emptySeats": 0.  
  }]
```

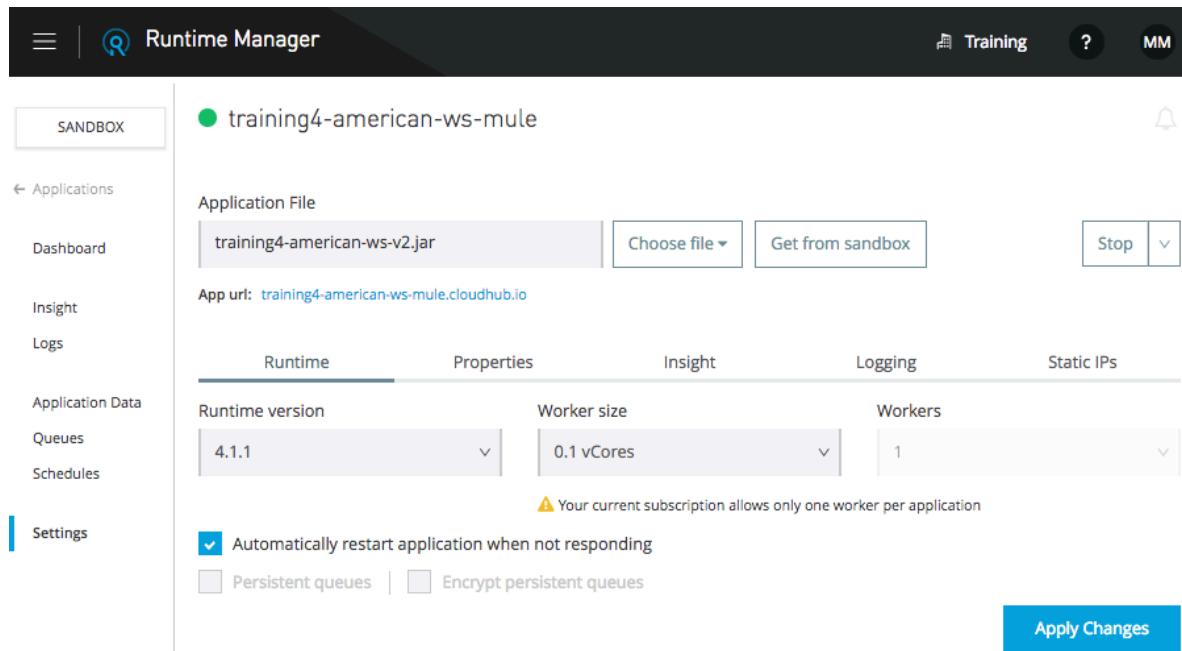
20. Leave this browser tab open.

Update the API implementation deployed to CloudHub

21. Return to the browser tab with Runtime Manager.
22. In the left-side navigation, select Settings.
23. Click the Choose file button and select Upload file.
24. Browse to the jars folder in the course student files.
25. Select training4-american-ws-v2.jar and click Open.

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

26. Click the Apply Changes button.



SANDBOX

Applications

Dashboard

Insight

Logs

Application Data

Queues

Schedules

Settings

training4-american-ws-mule

Application File

training4-american-ws-v2.jar

Choose file ▾

Get from sandbox

Stop

Training

?

MM

Runtime

Properties

Insight

Logging

Static IPs

Runtime version: 4.1.1

Worker size: 0.1 vCores

Workers: 1

⚠ Your current subscription allows only one worker per application

Automatically restart application when not responding

Persistent queues | Encrypt persistent queues

Apply Changes

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

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

28. Close the browser tab with Runtime Manager.

Test the updated application

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



A screenshot of a web browser window. The address bar shows the URL: "training-american-ws-mule.cloudhub.io/api/flights?destination=SFO". The main content area of the browser displays a JSON array of flight records:

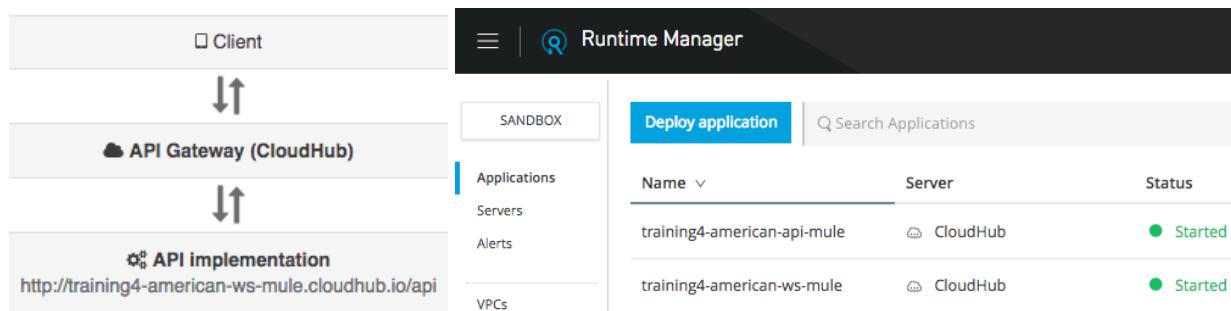
```
[  
  {  
    "ID": 5,  
    "code": "rree1093",  
    "price": 142,  
    "departureDate": "2016-02-11T00:00:00",  
    "origin": "MUA",  
    "destination": "SFO",  
    "emptySeats": 1,  
    "plane": {  
      "type": "Boeing 737",  
      "totalSeats": 150  
    },  
    {  
      "ID": 7, 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

30. Close this browser tab.

Walkthrough 5-2: Create and deploy an API proxy

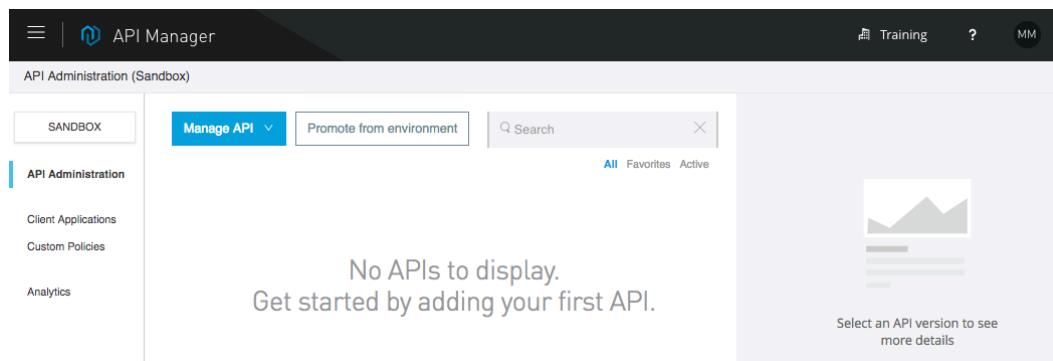
In this walkthrough, you create and deploy an API proxy for your API implementation on CloudHub. You will:

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

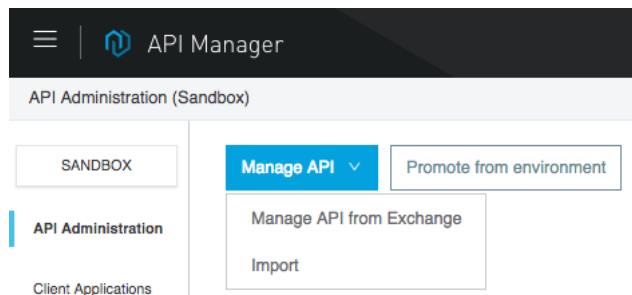


Create and deploy a proxy application

1. Return to Anypoint Platform.
2. In the main menu, select API Manager; you should see no APIs listed.



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



4. For API name, start typing American in the text field and then select your American Flights API in the drop-down menu that appears.
5. Set the rest of the fields to the following values:
 - Asset type: RAML/OAS
 - API version: v1
 - Asset version: 1.0.1
 - Managing type: Endpoint with Proxy
 - Implementation URI: `http://training4-american-ws-{lastname}.cloudbus.io/api`
 - Proxy deployment target: CloudHub

API Administration (Sandbox) | Get from Exchange

SANDBOX

Manage API from Exchange

API Configurations

API name:	American Flights API	<input type="button" value="Q"/>
Asset type:	RAML/OAS	<input type="button" value="▼"/>
API version:	v1	<input type="button" value="▼"/> View API in Exchange
Asset version:	1.0.1	<input type="button" value="▼"/>
Managing type:	<input type="radio"/> Basic Endpoint <input checked="" type="radio"/> Endpoint with Proxy	
Implementation URI:	<input type="text" value="http://training4-american-ws-mule.cloudbus.io/api"/> <input type="button" value="E"/>	
Proxy deployment target:	<input checked="" type="radio"/> CloudHub <input type="radio"/> Hybrid	
Path:	<input type="text" value="/"/>	
<input checked="" type="checkbox"/> Check this box if you are managing this API in Mule 4 or above.		

6. Select the checkbox: Check this box if you are managing this API in Mule 4 or above.

Path:

Check this box if you are managing this API in Mule 4 or above.

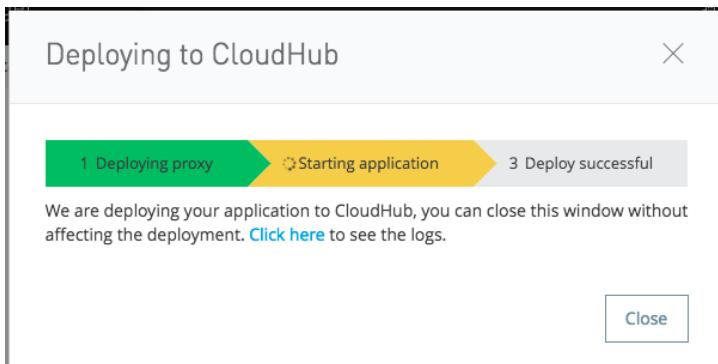
[Advanced options >](#)

7. Click Save.
8. In the Deployment Configuration section, set the following values:
 - Runtime version: 4.x.x
 - Proxy application name: training4-american-api-{lastname}
9. Check Update application if exists.

Deployment Configuration ▾

Runtime version:	4.x.x
Proxy application name: ⓘ	training4-american-api-mule
<input checked="" type="checkbox"/> Update application if exists	
Deploy	

10. Click Deploy.
11. In the Deploying to CloudHub dialog box, click the Click here link to see the logs.

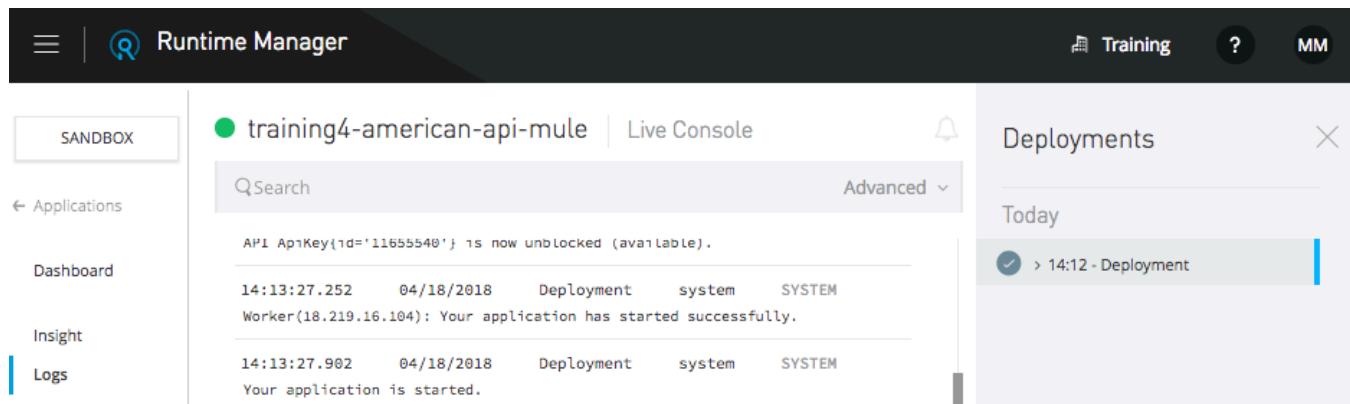


12. In the new browser tab that opens, watch the logs in Runtime Manager.

The Runtime Manager interface shows the deployment of the application "training4-american-api-mule". The left sidebar has "Sandbox" selected. The main area shows the application name and a "Live Console" tab. Below the console, a log entry is displayed: "14:12:08.656 04/18/2018 Deployment system SYSTEM Deploying application to 1 workers.". To the right, a "Deployments" panel shows a single entry: "Today > 14:12 - Deployment".

13. Wait until the proxy application starts.

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



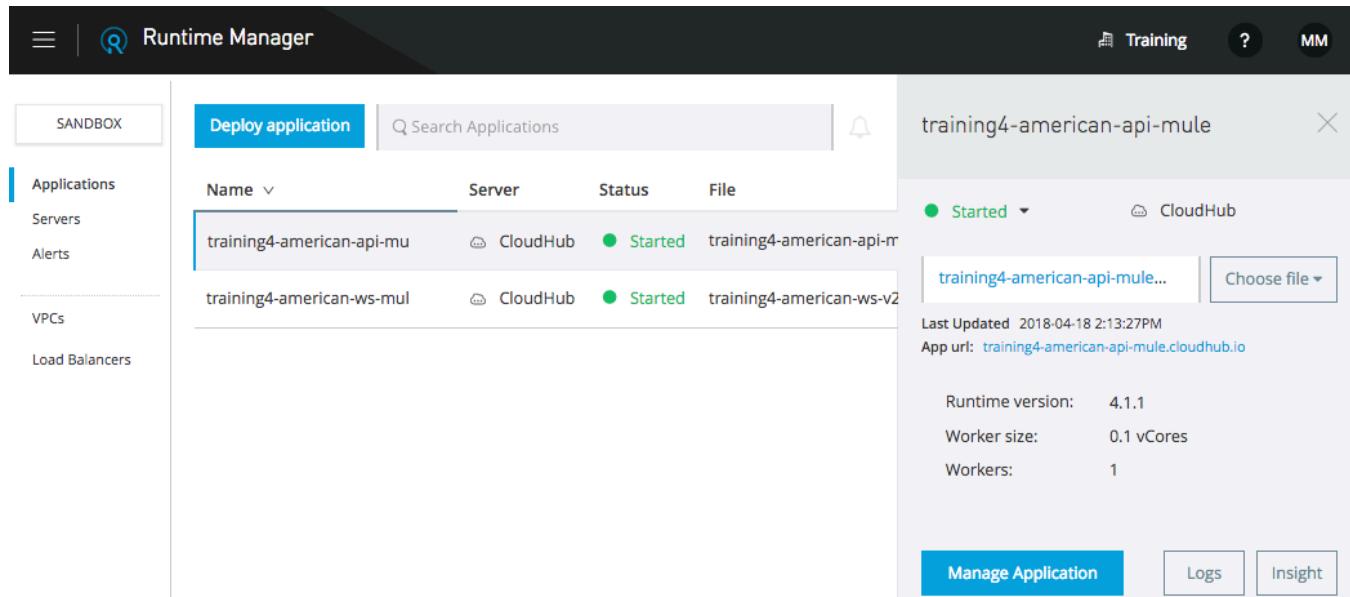
The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'Sandbox' selected, followed by 'Applications', 'Dashboard', 'Insight', and 'Logs'. The main area has a title 'training4-american-api-mule' with a green status dot. Below it is a search bar and a 'Live Console' section containing deployment logs:

```
API ApiKey(id='11655540') is now unblocked (available).
14:13:27.252 04/18/2018 Deployment system SYSTEM
Worker(18.219.16.104): Your application has started successfully.
14:13:27.902 04/18/2018 Deployment system SYSTEM
Your application is started.
```

To the right, a 'Deployments' panel shows a single entry for 'Today' at 14:12 with a deployment checkmark.

14. In the left-side navigation, select Applications; you should see the proxy application.

15. Click the row for the proxy and review its information in the right section of the window.



The screenshot shows the Runtime Manager interface with the 'Applications' tab selected in the sidebar. The main area displays a table of applications:

Name	Server	Status	File
training4-american-api-mu	CloudHub	Started	training4-american-api-m...
training4-american-ws-mul	CloudHub	Started	training4-american-ws-v2

On the right, detailed information for the 'training4-american-api-mule' application is shown:

- Status: Started
- Server: CloudHub
- Last Updated: 2018-04-18 2:13:27PM
- App url: training4-american-api-mule.cloudhub.io
- Runtime version: 4.1.1
- Worker size: 0.1 vCores
- Workers: 1

Buttons for 'Manage Application', 'Logs', and 'Insight' are at the bottom.

16. Close the browser tab.

View API details in API Manager

17. Return to the tab with API Manager and click the Close button in the Deploying to CloudHub dialog box.

18. Review the API proxy information at the top of the page.

The screenshot shows the API Manager interface. The left sidebar has a 'Sandbox' button and navigation links for Alerts, Client Applications, Policies, SLA Tiers, and Settings, with 'Settings' being the active tab. The main content area displays the 'American Flights API v1' settings. It shows the API Status as Active, Asset Version 1.0.1, and Type RAML/OAS. The Implementation URL is <http://training4-american-ws-mule.cloudhub.io/api>. There is a link to 'Add consumer endpoint'. Below this, the API Instance ID is 11655540, and the API ID is 11655540. A 'Label' field with a plus icon and 'Add a label' text is present. Under the 'Proxy' section, the Proxy Application is 'training4-american-api-mule' and the Proxy URL is <http://training4-american-api-mule.cloudhub.io>.

19. In the left-side navigation, click the API Administration link; you should now see your American Flights API listed.

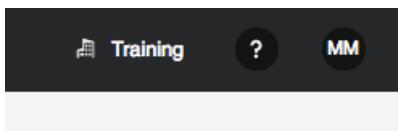
20. Click in the row for the v1 version – but not on the v1 link.

21. Review the API version info that appears on the right side of the window; you should see there are no policies, SLA tiers, or client applications.

The screenshot shows the API Manager API Administration list. The left sidebar includes 'Sandbox', 'API Administration' (which is active), 'Client Applications', 'Custom Policies', and 'Analytics'. The main area lists the 'American Flights API' with one version, 'v1'. The table columns are API Name, Version, Status, Client Applications, and Creation Date. The 'v1' row shows it is Active and was created on 04-18-2018 14:11. To the right, a detailed view for 'American Flights API v1' is shown, featuring a 'Manage CloudHub Proxy' button, 'View API in Exchange', and 'View Analytics Dashboard'. Below these are tabs for 'Applications', 'Policies' (which is active), and 'SLA tiers'. A note states, 'There are no policies configured for this API version.'

22. In the API list, click the v1 link for the API; you should be returned to the Settings page for the API.

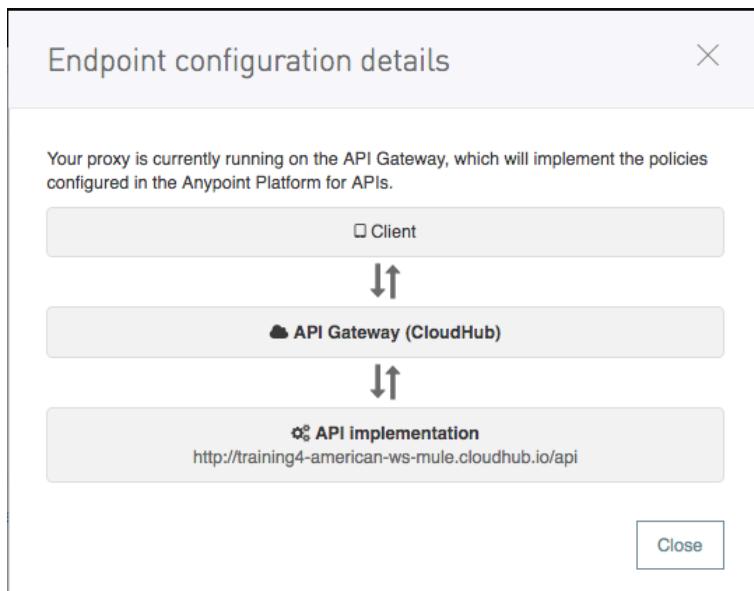
23. Locate and review the links on the right side of the page.



Actions ▾

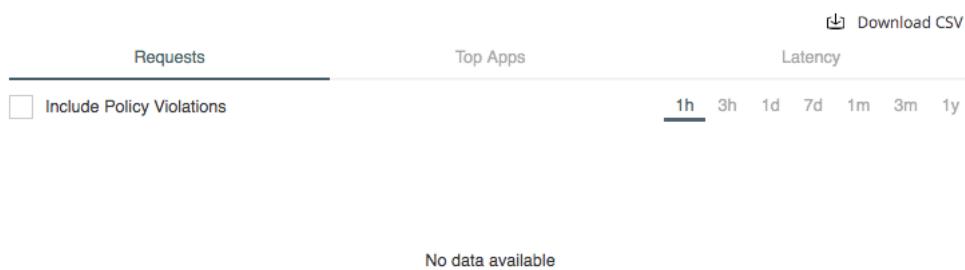
- Manage CloudHub Proxy >
- View API in Exchange >
- View configuration details >
- View Analytics Dashboard >

24. Click the View configuration details link.



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

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



View the new API proxy instance in Exchange

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

The screenshot shows a table titled 'Asset versions for v1'. It has two columns: 'Version' and 'Instances'. There are two rows. The first row is for version 1.0.1, which contains two entries: 'Mocking Service' and 'Sandbox - v1:5831632'. The second row is for version 1.0.0, which is currently empty. Each entry has a three-dot menu icon to its right.

Version	Instances
1.0.1	Mocking Service Sandbox - v1:5831632
1.0.0	

Note: You may need to refresh your browser page.

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

The screenshot shows a dropdown menu for the 'Mocking Service' option. The menu is open and displays two items: 'Mocking Service' and 'Mocking Service'. The first item is highlighted.

- Mocking Service
- Mocking Service

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

The screenshot shows the Exchange interface with the 'API instances' section selected in the left sidebar. The main area displays a table of API instances. There are two rows: one for 'Mocking Service' (URL: https://mocksvc-proxy.anypoint.mulesoft.com/exchange/800b1585-b6be-4c62-82de-02d8c2adff413/american-flights-api/1.0.1) and one for 'v1:5831632' (Environment: Sandbox, URL: https://v1:5831632). A 'Add new instance' button is also visible.

Instances	Environment	URL	Visibility
Mocking Service		https://mocksvc-proxy.anypoint.mulesoft.com/exchange/800b1585-b6be-4c62-82de-02d8c2adff413/american-flights-api/1.0.1	Public
v1:5831632	Sandbox	https://v1:5831632	Private

Set a friendly label for the API instance in API Manager

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

API Instance ⓘ

ID: 5831632

Label: No policy 

Set a consumer endpoint for the proxy in API Manager

36. Locate the proxy URL.

Proxy

Proxy Application: training4-american-api-mule

Proxy URL: training4-american-api-mule.cloudhub.io

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

38. Click the Add consumer endpoint link.

American Flights API v1

API Status:  Active Asset Version: 1.0.1 Type: RAML/OAS

Implementation URL: <http://training4-american-ws-mule.cloudhub.io/api>  Add consumer endpoint

39. Paste the value of the proxy URL.

40. Press Enter/Return.

American Flights API v1

API Status:  Active Asset Version: 1.0.1 Type: RAML/OAS

Implementation URL: <http://training4-american-ws-mule.cloudhub.io/api>

Consumer endpoint: <http://training4-american-api-mule.cloudhub.io/> 

Make requests to the API proxy from Exchange

41. Return to the browser tab with Exchange.

42. Refresh the API instances page for your American Flights API; you should see the new label and the URL.

The screenshot shows the Exchange interface with the 'American Flights API' selected. On the left, the navigation bar includes 'Assets list', 'American Flights API', 'API summary', 'Types', 'Resources', '/flights' (selected), 'GET' (highlighted in green), 'POST', and '/{ID}'. The main area displays the 'American Flights API' details (version v1, Public) and its 'API instances'. A table lists two instances: 'Mocking Service' (Public, URL: https://mocksvc-proxy.anypoint.mulesoft.com/exchange/74922056-9245-48e0-99df-a8141d1d3e9f/american4-flights-api/1.0.1) and 'No policy' (Sandbox, URL: http://training4-american-api-mule.cloudhub.io/). A button '+ Add new instance' is also visible.

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

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

The screenshot shows the 'Asset versions for v1' page. It lists two versions: '1.0.1' and '1.0.0'. Under '1.0.1', there are two instances: 'Mocking Service' (Public) and 'Sandbox - No policy'. Each instance has a three-dot menu icon to its right.

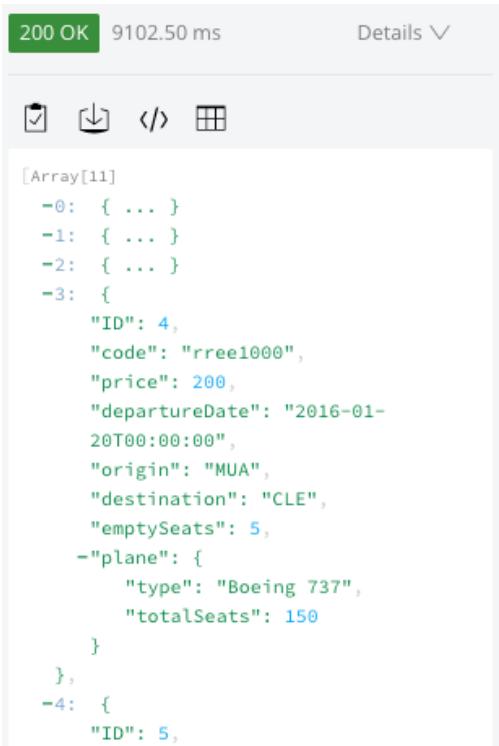
45. Click the GET method for the flights resource.

46. In the API console, click the drop-down arrow next to Mocking Service; you should now see your API proxy instance as a choice.

The screenshot shows a dropdown menu with three options: 'Sandbox - No policy', 'Mocking Service', and 'Sandbox - No policy'. The 'Mocking Service' option is highlighted.

47. Select the Sandbox - No policy instance.

48. Click the Send button; you should now see the real data from the database, which contains multiple flights.



200 OK 9102.50 ms Details ▾

Array[11]

-0: { ... }
-1: { ... }
-2: { ... }
-3: {
 "ID": 4,
 "code": "rree1000",
 "price": 200,
 "departureDate": "2016-01-
20T00:00:00",
 "origin": "MUA",
 "destination": "CLE",
 "emptySeats": 5,
 "plane": {
 "type": "Boeing 737",
 "totalSeats": 150
 },
-4: {
 "ID": 5,

49. Make several more calls to this endpoint.

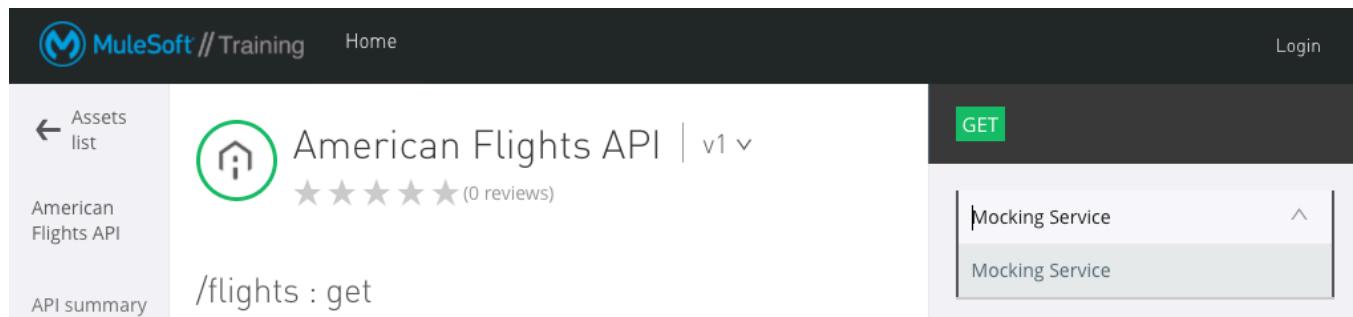
50. Make calls to different methods.

Make requests to the API proxy from the public portal

51. Return to the public portal in the private/incognito window.

52. Click the GET method for the flights resource.

53. In the API reference section, click the drop-down arrow next to Mocking Service; you should NOT see your API proxy instance as a choice.



MuleSoft // Training Home Login

Assets list American Flights API | v1 ▾

American Flights API

API summary /flights : get

GET Mocking Service Mocking Service

Make an API instance visible in the public portal

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

The screenshot shows the 'API instances' section of the American Flights API. There are two entries:

Instances	Environment	URL	Visibility
Mocking Service		https://mocksvc-proxy.anypoint.mulesoft.com/exchange/74922056-9245-48e0-99df-a8141d1d3e9f/american4-flights-api/1.0.1	Public
No policy	Sandbox	http://training4-american-api-mule.cloudhub.io/	Public ▾

[+ Add new instance](#)

57. Return to the public portal in the private/incognito window.
58. Refresh the page.
59. In the API console, change the API instance from Mocking Service to Sandbox - No policy.

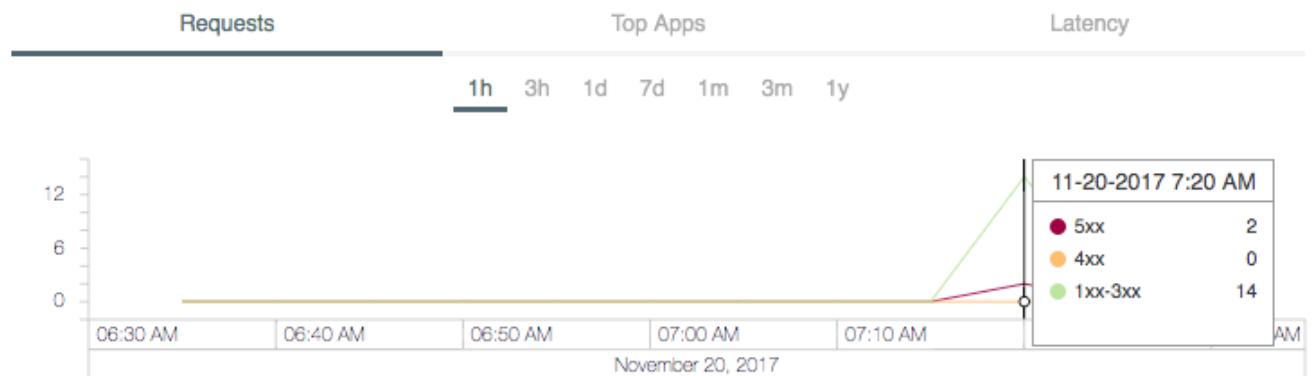
The screenshot shows an API request interface. At the top, there are 'Download' and 'Request access' buttons. Below that, a 'GET' method is selected. The 'Instance' dropdown shows 'Sandbox - No policy'. The URL field contains <http://training4-american-api-mule.cloudhub.io/flights>.

60. Click Send; you should a 200 response and flight data.

Look at the API request data

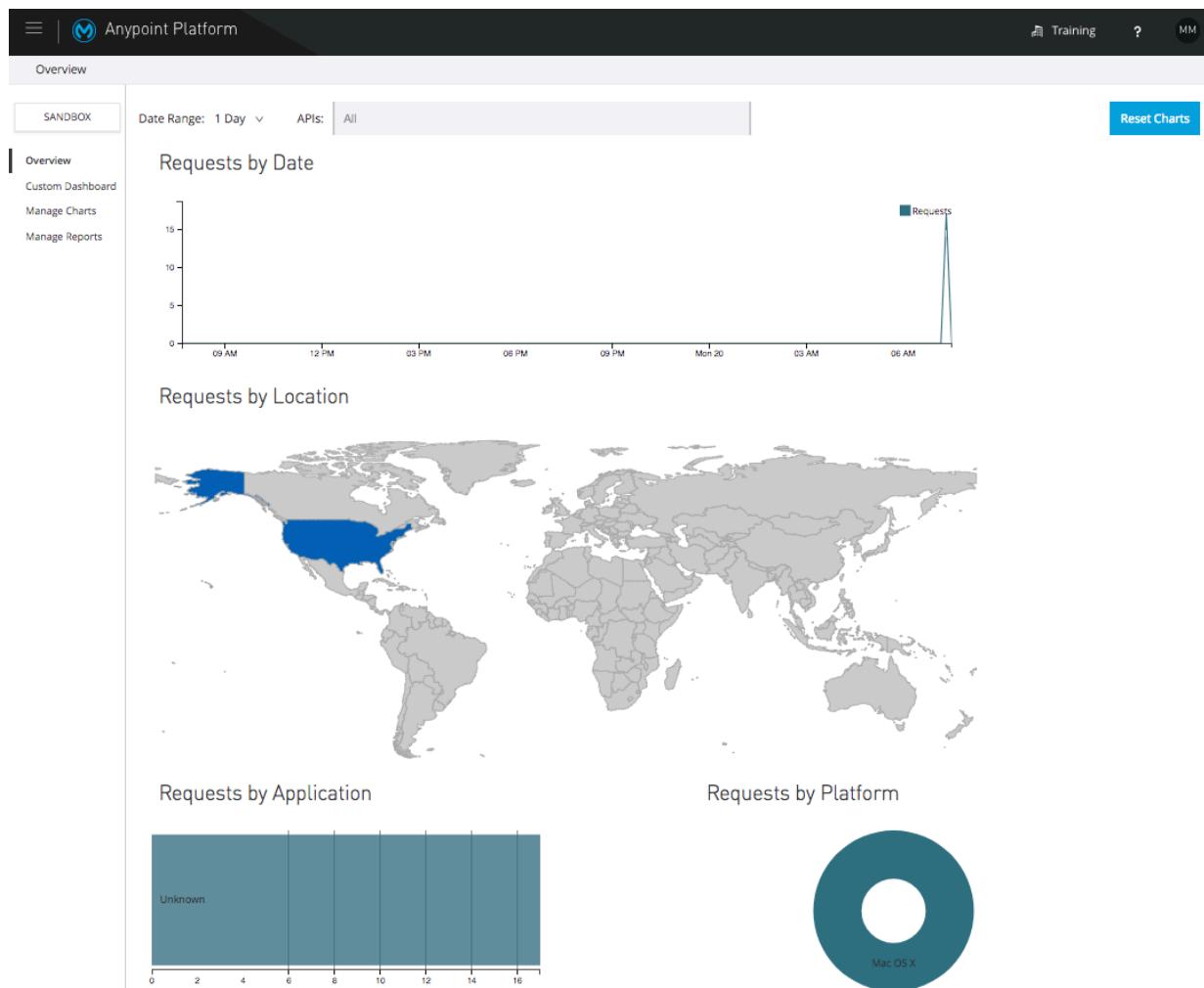
61. Return to the browser tab with API Manager.
62. Refresh the Settings page for your American Flights API.

63. Look at the Request chart again; you should now see data for some API calls.



64. Click the View Analytics Dashboard link located in the upper-right corner.

65. Review the data in the dashboard.



66. Close the browser tab.

Walkthrough 5-3: Restrict API access with policies and SLAs

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

- Add and test a rate limiting policy.
- Add SLA tiers, one with manual approval required.
- Add and test a rate limiting SLA based policy.

Name	Category	Fulfils	Requires
Rate limiting - SLA based	Quality of service	SLA Rate Limiting, Client ID required	RAML snippet

Create a rate limiting policy

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

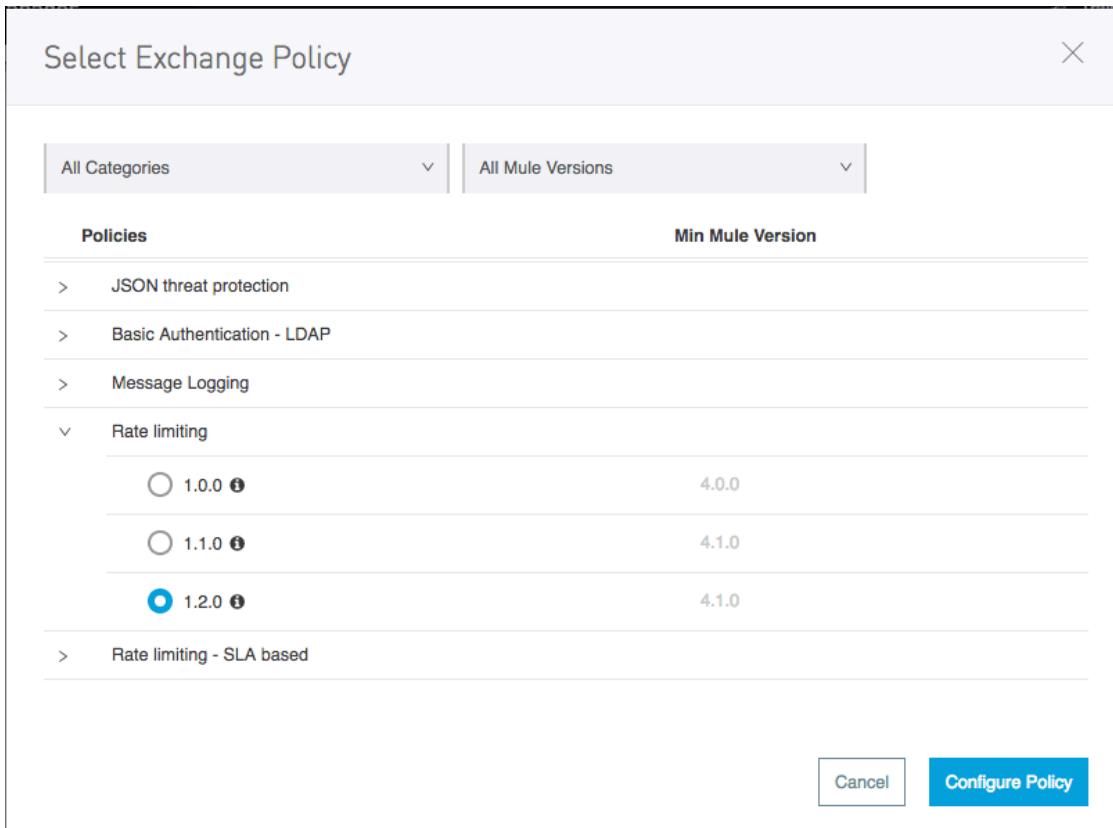
American Flights API v1

API Status: Active Asset Version: 1.0.1 Type: RAML/OAS
Implementation URL: <http://training4-american-ws-mule.cloudhub.io/api>
Consumer endpoint: <http://training4-american-api-mule.cloudhub.io/>

Actions

3. Click the Apply New Policy button.

4. In the Select Exchange Policy dialog box, expand Rate limiting and select the latest version for the Mule runtime version you are using.



5. Click Configure Policy.

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

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

Apply Rate limiting policy

Specifies the maximum value for the number of messages processed per time period, and rejects any messages beyond the maximum. Applies rate limiting to all API calls, regardless of the source.

Identifier

For each identifier value, the set of Limits defined in the policy will be enforced independently. I.e.: # [attributes.queryParams["identifier"]].

Limits

Pairs of maximum quota allowed and time window.

# of Reqs *	Time Period *	Time Unit *	
3	1	Minute	<input type="button" value="Delete"/>

Add Limit

Clusterizable

When using a clustered runtime with this flag enabled, configuration will be shared among all nodes.

Expose Headers

Defines if headers should be exposed in the response to the client. These headers are: x-ratelimit-remaining, x-ratelimit-limit and x-ratelimit-reset.

Method & Resource conditions

- Apply configurations to all API methods & resources
 Apply configurations to specific methods & resources

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

The screenshot shows the MuleSoft API Manager interface. The top navigation bar includes 'API Manager', 'Training', and 'MM'. The left sidebar has a 'Sandbox' button and links for 'API Administration (Sandbox)', 'Alerts', 'Client Applications', 'Policies' (which is selected), 'SLA Tiers', and 'Settings'. The main content area is titled 'American Flights API v1'. It shows the API status as 'Active', asset version '1.0.1', and type 'RAML/OAS'. Implementation URL is 'http://training4-american-ws-mule.cloudhub.io/api' and the consumer endpoint is 'http://training4-american-api-mule.cloudhub.io/'. There is a 'Manage CloudHub Proxy' link. Below this, there is a 'Policies' section with a 'Policy Details' card for 'Rate limiting' (with ID '1') which is categorized under 'Quality of service' and fulfills 'Baseline Rate Limiting'. A blue 'Edit policy order' button is visible. At the bottom, there is a table with columns 'Name', 'Category', 'Fulfils', and 'Requires'.

Name	Category	Fulfils	Requires
> Rate limiting <small>(1)</small>	Quality of service	Baseline Rate Limiting	

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

API Instance ⓘ

ID: 5831632

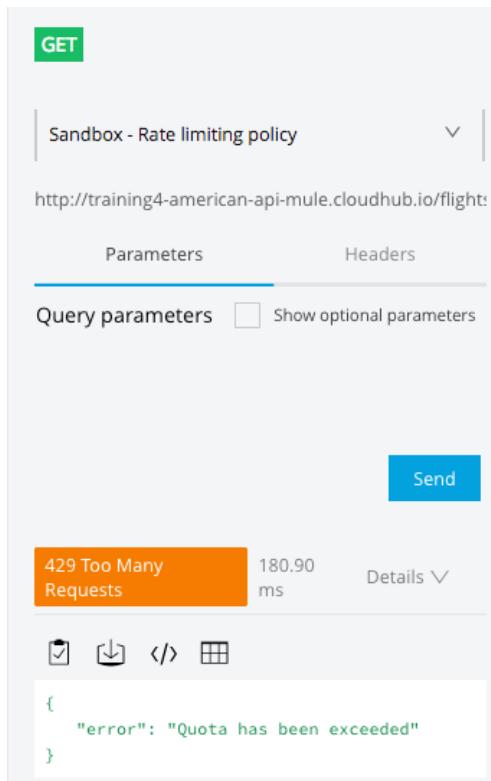
Label: Rate limiting policy 

Test the new rate limiting policy

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

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

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



The screenshot shows the API console interface for a GET request to the 'Sandbox - Rate limiting policy' instance. The URL is `http://training4-american-api-mule.cloudhub.io/flight`. The 'Parameters' tab is selected, and there are no query parameters listed. A 'Send' button is visible at the bottom. Below the request details, a response summary is shown: '429 Too Many Requests' in an orange box, '180.90 ms', and a 'Details' link. The response body is displayed as JSON: `{ "error": "Quota has been exceeded" }`.

Create SLA tiers

14. Return to the browser tab with your American Flights API in API Manager.
15. In the left-side navigation, select SLA Tiers.

16. Click the Add SLA tier button.

The screenshot shows the API Manager interface for the American Flights API (v1). On the left, there's a sidebar with options like API Administration, Alerts, Client Applications, Policies, SLA Tiers (which is selected and highlighted in blue), and Settings. The main content area displays the API details: API Status: Active, Asset Version: 1.0.1, Type: RAML/OAS. It also shows the Implementation URL and Consumer endpoint. At the bottom of this section, there's a message: "There are no SLA tiers for this API version." Below this, there's a prominent blue button labeled "Add SLA tier".

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

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

The dialog box has a header "Add SLA tier" and a close button "X". It contains several input fields:

- Name ***: A text input field containing "Free".
- Description**: A text input field containing "Description".
- Approval ***: A dropdown menu showing "Automatic".
- Limits**: A section with three input fields: "# of Reqs *", "Time Period *", and "Time Unit *".
 - "# of Reqs *": A text input field containing "1".
 - "Time Period *": A text input field containing "1".
 - "Time Unit *": A dropdown menu showing "Minute".A small "visible" checkbox is checked, and a trash can icon is present.

At the bottom, there are "Cancel" and "Add" buttons, with "Add" being highlighted in blue.

18. Click the Add button.

19. Create a second SLA tier with the following values:

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

The screenshot shows a table of SLA tiers. At the top left is a blue button labeled "Add SLA tier". To its right is a search bar with a magnifying glass icon and the placeholder text "Search". On the far right, there is a page navigation area with a "X" button, the text "1 - 2 of 2", and arrows for navigating between pages.

Name	Limits	Applications	Status	Approval		
Free	1	0	Active	Auto	<button>Edit</button>	<button>Delete</button>
Silver	1	0	Active	Manual	<button>Edit</button>	<button>Delete</button>

Change the policy to rate limiting – SLA based

20. In the left-side navigation, select Policies.

21. Expand the Rate limiting policy.

22. Click the Actions button and select Remove.

The screenshot shows the configuration of a "Rate limiting" policy. At the top left is a "Name" field containing "Rate limiting" with a help icon. To its right are "Category" (Quality of service), "Fulfils" (Baseline Rate Limiting), and "Requires" (empty). Below this is a table with columns "Order", "Method", and "Resource URI". The first row shows "All API Methods" and "All API Resources". To the right of this table is a "Actions" dropdown menu with three options: "View Detail", "Actions ▾", "Disable", "Edit", and "Remove".

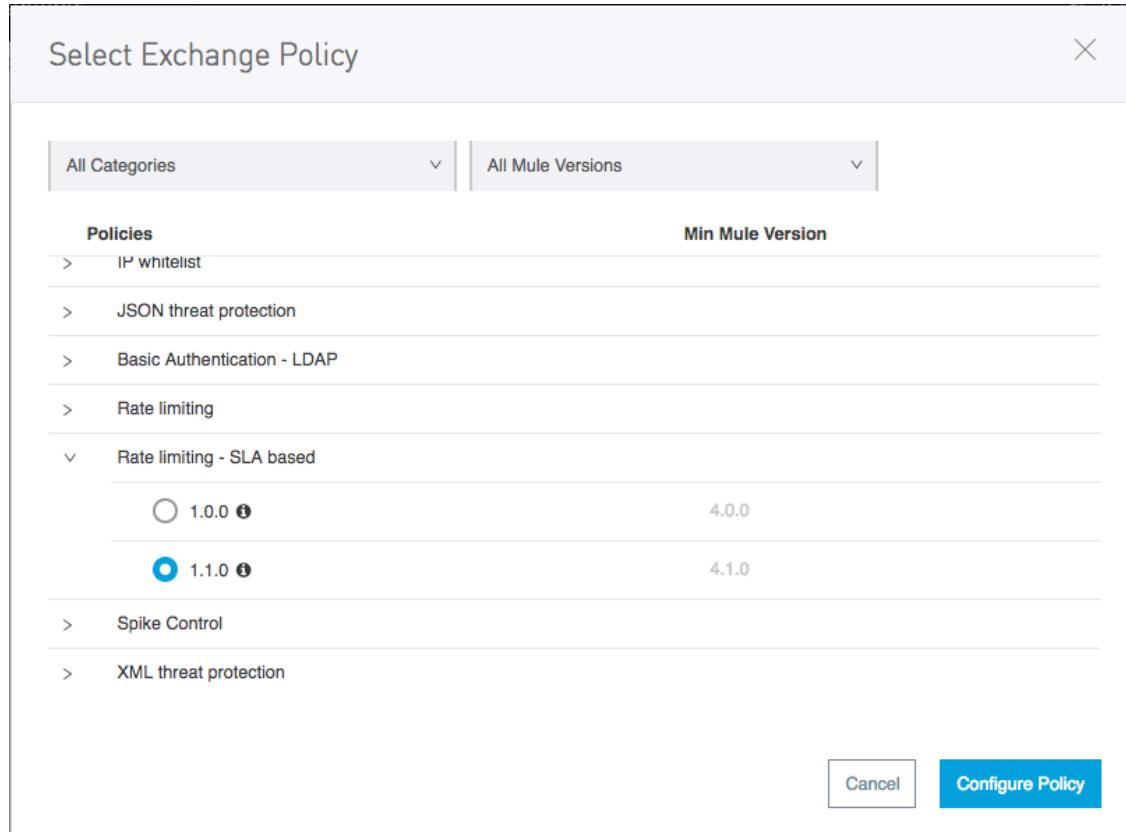
Order	Method	Resource URI	
	All API Methods	All API Resources	<button>View Detail</button> <button>Actions ▾</button>

Disable
Edit
Remove

23. In the Remove policy dialog box, click Remove.

24. Click the Apply New Policy button.

25. In the Select Policy dialog box, expand Rate limiting - SLA based and select the latest version for the Mule runtime version you are using.
26. Click Configure Policy.



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

The screenshot shows the API Manager interface with the following details:

- Sandbox** tab is selected.
- Policies** section is visible on the left.
- Apply Rate limiting - SLA based policy** page is displayed.
- Client ID Expression ***: Mule Expression to extract Client ID from API requests. Value: `#[attributes.headers['client_id']]`.
- Client Secret Expression**: Mule Expression to extract Client Secret from API requests. Value: `#[attributes.headers['client_secret']]`.
- Clusterizable**: A checked checkbox. Description: When using a clustered runtime with this flag enabled, configuration will be shared among all nodes.
- Expose Headers**: A checked checkbox. Description: Defines if headers should be exposed in the response to the client. These headers are: x-ratelimit-remaining, x-ratelimit-limit and x-ratelimit-reset.
- Method & Resource conditions**:
 - Apply configurations to all API methods & resources** (radio button selected)
 - Apply configurations to specific methods & resources** (radio button unselected)
- Buttons**: **Cancel** and **Apply**.

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

The screenshot shows the API Instance settings page with the following details:

- API Instance** (with a help icon)
- ID**: 5831632
- Label**: Rate limiting - SLA based policy (with a edit icon)

Test the rate limiting – SLA based policy in Exchange

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

33. Click Send; you should get a 401 Unauthorized response with a message that there is an invalid client id or secret.

The screenshot shows a REST API testing interface. At the top left is a green "GET" button. Below it is a dropdown menu set to "Sandbox - Rate limiting - SLA based policy". The URL field contains "http://training4-american-api-mule.cloudhub.io/flight:". Below the URL are two tabs: "Parameters" (selected) and "Headers". Under "Parameters", there is a section for "Query parameters" with a checkbox labeled "Show optional parameters". A large blue "Send" button is centered below the parameters. At the bottom of the interface, an orange bar displays the response status "401 Unauthorized" and the execution time "819.80 ms". To the right of the status is a "Details" link with a dropdown arrow. Below this, there are four small icons: a clipboard with a checkmark, a download arrow, a file icon with a slash, and a refresh/reload icon. A code block shows the JSON response:

```
{  "error": "Invalid client id or secret"}
```

Walkthrough 5-4: Request and grant access to a managed API

In this walkthrough, clients request access to an API proxy and administrators grant access. You will:

- Request application access to SLA tiers from private and public API portals.
- Approve application requests to SLA tiers in API Manager.

The screenshot shows the 'Client Applications' section of the API Manager. On the left, there's a sidebar with 'Sandbox' selected. The main table lists two applications: 'Training external app' and 'Training internal app'. The 'Training external app' row shows 'Silver' as the 'Current SLA tier', 'N/A' as the 'Requested SLA tier', and 'Approved' as the 'Status'. There is a 'Revoke' button at the end of this row. The 'Training internal app' row shows 'Free' as the 'Current SLA tier', 'N/A' as the 'Requested SLA tier', and 'Approved' as the 'Status'. There is also a 'Revoke' button at the end of this row. Below the table, there are detailed columns for 'Owners', 'Client ID', 'URL', and 'Redirect URIs'.

Application	Current SLA tier	Requested SLA tier	Status
Training external app	Silver	N/A	Approved
Training internal app	Free	N/A	Approved

Request access to the API as an internal consumer

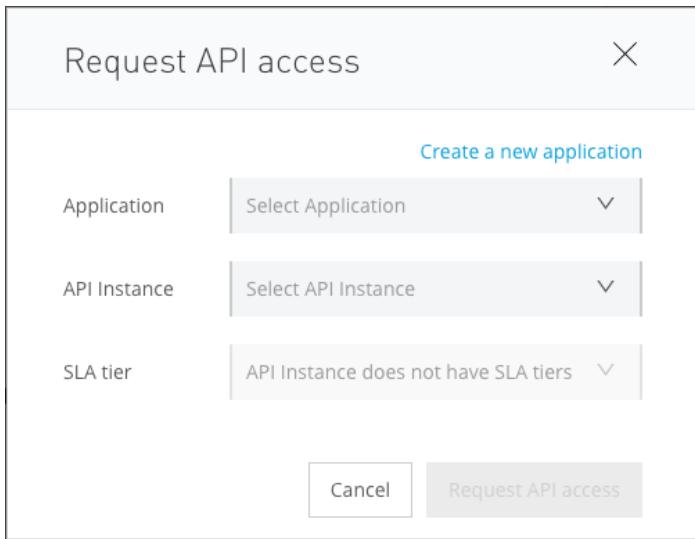
1. Return to the browser tab with Anypoint Exchange.
2. In the left-side navigation, select the name of the API to return to its home page.
3. Click the more options button in the upper-right corner and select Request access.

The screenshot shows the Anypoint Exchange interface for the 'American Flights API'. The top bar includes 'Training', a user icon, and a 'MM' button. The left sidebar has 'Assets list' and 'American Flights API' selected. The main area displays the API's logo, name ('American Flights API'), version ('v1'), a 5-star rating with '(0 reviews)', and 'Rate and review' link. To the right, there are buttons for 'Share', 'Download', 'Edit', and a three-dot menu. Below these is a 'Overview' section with a 'Request access' button.

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

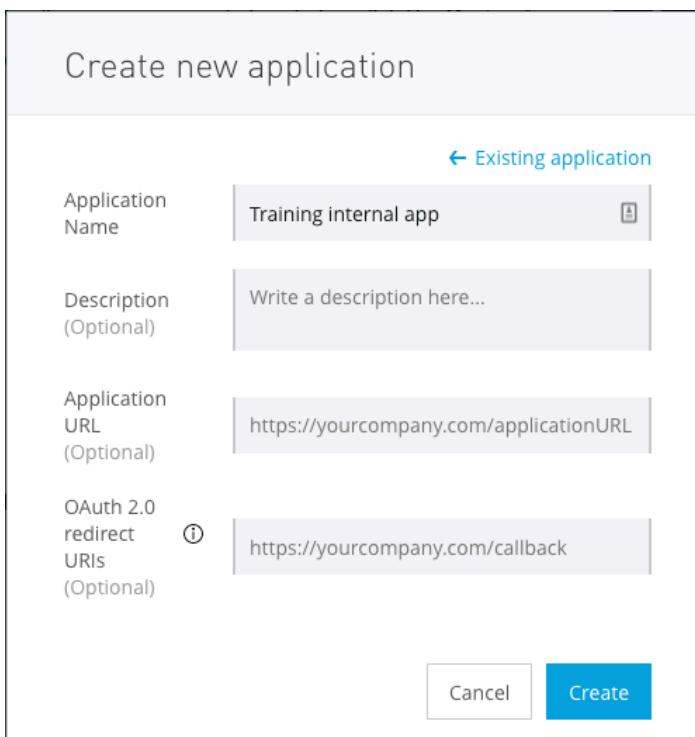
This screenshot shows the same Anypoint Exchange interface as the previous one, but the user has different permissions. The 'Edit' and three-dot menu buttons are replaced by a 'Download' and 'Request access' button. The rest of the interface remains the same, including the sidebar and the 'Overview' section below.

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



The dialog box has a title bar "Request API access" with a close button "X". Below it is a section titled "Create a new application". It contains three dropdown menus: "Application" (set to "Select Application"), "API Instance" (set to "Select API Instance"), and "SLA tier" (set to "API Instance does not have SLA tiers"). At the bottom are two buttons: "Cancel" and "Request API access".

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



The dialog box has a title bar "Create new application" with a back arrow "Existing application". It contains four input fields: "Application Name" (set to "Training internal app"), "Description (Optional)" (set to "Write a description here..."), "Application URL (Optional)" (set to "https://yourcompany.com/applicationURL"), and "OAuth 2.0 redirect URIs (Optional)" (set to "https://yourcompany.com/callback"). At the bottom are two buttons: "Cancel" and a blue "Create" button.

6. In the Request API access dialog box, set the API instance to Sandbox – Rate limiting SLA - based policy.

7. Set the SLA tier to Free.

Request API access

Create a new application

Application	Training internal app	▼
API Instance	Sandbox - Rate limiting - SLA based ...	▼
SLA tier	Free	▼

# of Reqs	Time period	Time Unit
1	1	Minute

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

Request API access ×

✓ API access has been successful!

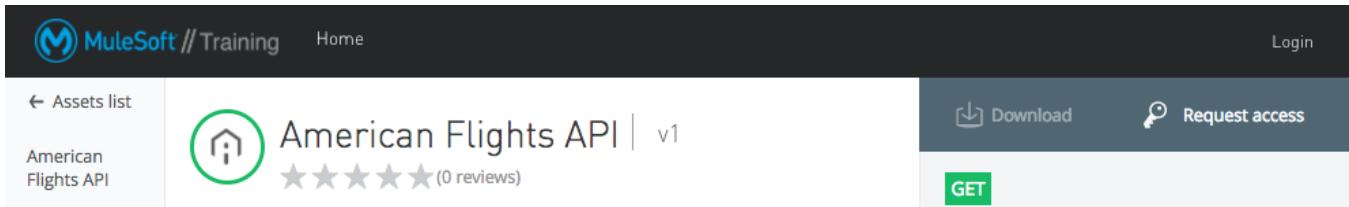
Client ID	e708026bb0cf4c3e8594ca39138b9a00
Client secret	e10A1faF382D47DEA6A90cA8d4FC3891

Application details have been opened in a new tab.

10. Click Close.

Request access to the API as an external consumer

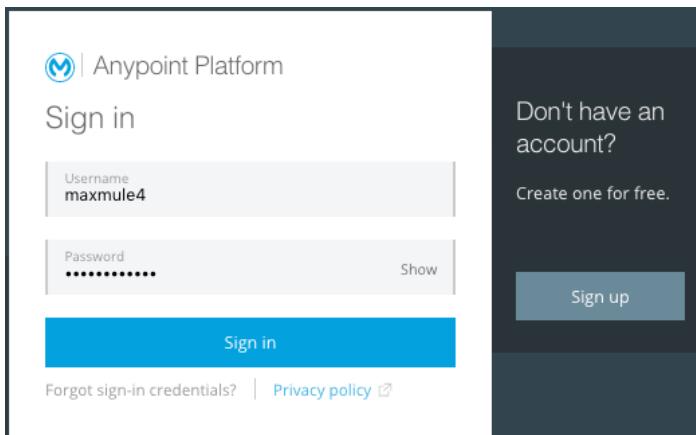
11. Return to the public portal in the private/incognito window.
12. Refresh the page for the American Flights API; you should now see a Request access button.



The screenshot shows the MuleSoft Training portal interface. At the top, there's a navigation bar with the MuleSoft logo, the text "MuleSoft // Training", a "Home" link, and a "Login" link. Below the navigation bar, there's a sidebar on the left with links for "Assets list" and "American Flights API". The main content area displays the "American Flights API | v1" listing, which includes a green circular icon with a house symbol, a star rating of 5 stars, and the text "(0 reviews)". To the right of the listing are two buttons: "Download" and "Request access". The "Request access" button is highlighted with a green border and a white background. Below these buttons is a "GET" button.

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

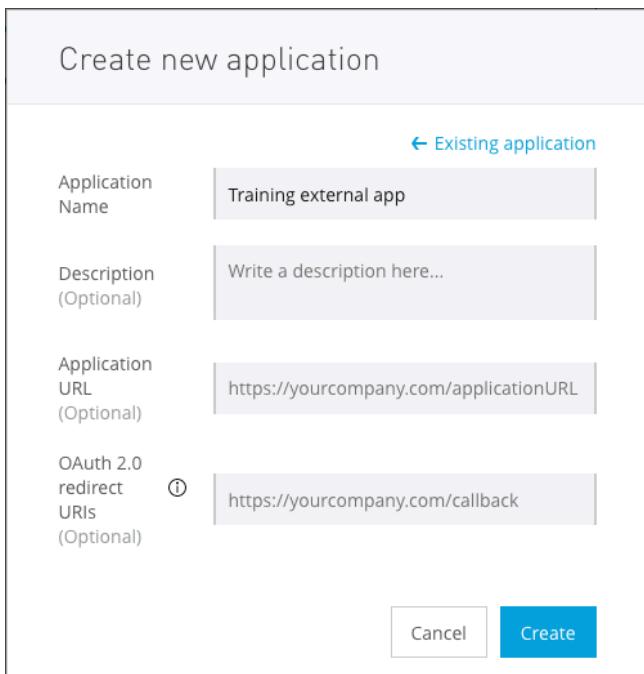
Note: Instead of creating an external user, you will just use your existing account.



The screenshot shows the Anypoint Platform sign-in page. It features a "Sign in" form with fields for "Username" (containing "maxmule4") and "Password" (showing masked input). Below the form is a blue "Sign in" button. To the right of the form, there's a dark sidebar with the text "Don't have an account? Create one for free." and a "Sign up" button. At the bottom of the page, there are links for "Forgot sign-in credentials?" and "Privacy policy".

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

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

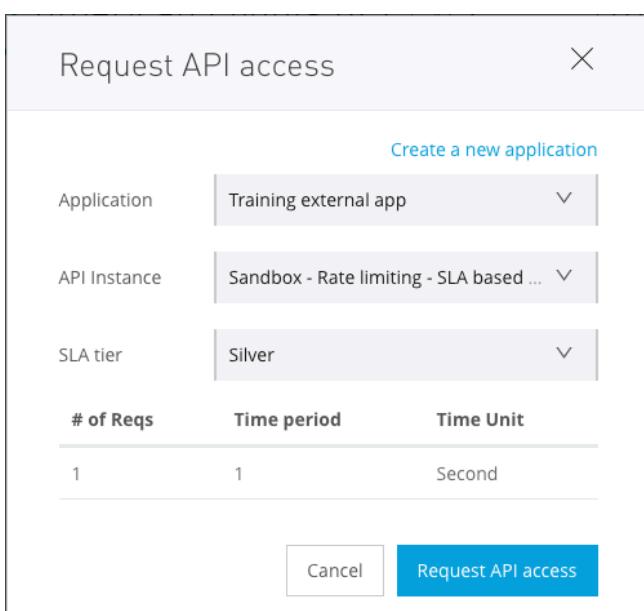


The dialog box has a title 'Create new application' at the top left. It contains four input fields: 'Application Name' with 'Training external app', 'Description (Optional)' with 'Write a description here...', 'Application URL (Optional)' with 'https://yourcompany.com/applicationURL', and 'OAuth 2.0 redirect URIs (Optional)' with 'https://yourcompany.com/callback'. At the bottom right are 'Cancel' and 'Create' buttons, with 'Create' being blue.

18. Click Create.

19. In the Request API access dialog box, set the API instance to Sandbox – Rate limiting SLA-based policy.

20. Set the SLA tier to Silver.



The dialog box has a title 'Request API access' at the top left. It contains three dropdown menus: 'Application' set to 'Training external app', 'API Instance' set to 'Sandbox - Rate limiting - SLA based ...', and 'SLA tier' set to 'Silver'. Below these are three columns: '# of Reqs' (1), 'Time period' (1), and 'Time Unit' (Second). At the bottom right are 'Cancel' and 'Request API access' buttons, with 'Request API access' being blue.

21. Click Request API access.

22. In the Request API access dialog box, click Close.

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

The screenshot shows the 'My applications' page of the MuleSoft // Training portal. At the top, there is a navigation bar with the MuleSoft logo, the text 'MuleSoft // Training', and links for 'Home' and 'My applications'. A search bar with a magnifying glass icon and the placeholder 'Search' is located below the navigation bar. The main content area is titled 'My applications' and contains a table with two rows. The columns are 'Name' and 'Description'. The first row has 'Training internal app' in the Name column and an empty Description column. The second row has 'Training external app' in the Name column and an empty Description column.

Name	Description
Training internal app	
Training external app	

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

The screenshot shows the details page for the 'Training external app'. At the top, there is a navigation bar with the MuleSoft logo, the text 'MuleSoft // Training', and links for 'Home' and 'My applications'. Below the navigation bar, there is a breadcrumb trail with '← My applications' and a back arrow. On the right side, there are three buttons: 'Edit' (highlighted in blue), 'Reset client secret', and 'Delete'. The main content area is titled 'Training external app'. On the left, there is a sidebar with a list: 'Show All (1)' (selected) and 'Sandbox - Rate limiti... v1'. On the right, there is a table with application details: 'Application Description:', 'Application URL:', and 'Redirect URIs:'. To the right of the table, there are fields for 'Client ID: 7623dbcc2e1949d7a861160fe4a3a1e6', 'Client Secret: Show', and 'Grant Types: -'.

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

Grant an application access

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

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

The screenshot shows the API Manager interface with the 'Client Applications' tab selected. A search bar at the top right contains the placeholder 'Search'. Below it, a table lists two applications:

Application	Current SLA tier	Requested SLA tier	Status
> Training external app	N/A	Silver	Pending
> Training internal app	Free	N/A	Approved

Buttons for 'Approve', 'Reject', and 'Delete' are available for the pending application, while a 'Revoke' button is shown for the approved one.

28. Click the Approve button for the application requesting Silver tier access.

29. Expand the Training external app row and review its information.

30. Copy the value of the client_id.

A detailed view of the 'Training external app' row from the previous table. The application is listed as 'Approved' with a 'Silver' current SLA tier. Below the main table, a expanded section provides more details:

Owners	Max Mule	Submitted	2 minutes ago
Client ID	7623dbcc2e1949d7a861160fe4a3a1e6	Approved	a few seconds ago
URL	None	Rejected	-
Redirect URIs	None	Revoked	-

Below this, another row for 'Training internal app' is shown with a 'Free' current SLA tier and an 'Approved' status, accompanied by a 'Revoke' button.

Add authorization headers to test the rate limiting – SLA based policy from an API portal

31. Return to the browser window and tab with the API console in the public portal.
32. Try again to make a call to the Sandbox – Rate limiting – SLA based policy; you should still get a 401 Unauthorized response.
33. Select the Headers tab.
34. Click Add custom header.
35. Set the header name to client_id.

36. Set the value of client_id to the value you copied.

GET

Sandbox - Rate limiting - SLA based policy

http://training4-american-api-mule.cloudhub.io/flights

Parameters Headers

</>

Header name	Value
client_id	7623dbcc2e1949d7

+ Add custom header

Send

37. Return to the browser tab with My applications in the public portal.

38. Copy the value of the client_secret.

39. Return to the browser window and tab with the API console in the public portal.

40. Add another custom header and set the name to client_secret.

41. Set the client_secret header to the value you copied.

Parameters Headers

</>

Header name	Value
client_id	7623dbcc2e1949d7
Header name	Value
client_secret	52505680a6FB4d5

+ Add custom header

42. Click Send; you should now get a 200 response with flight results.

200 OK 1119.00 ms Details

⏪ </> ⌂

```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-
```

Walkthrough 5-5: Add client ID enforcement to an API specification

In this walkthrough, you add client ID enforcement to the API specification. You will:

- Modify an API specification to require client id and client secret headers with requests.
- Update a managed API to use a new version of an API specification.
- Call a governed API with client credentials from API portals.

Note: If you do not complete this exercise for Fundamentals, the REST connector that is created for the API and that you use later in the course will not have client_id authentication.

The screenshot shows a RAML 1.0 specification for the American Flights API. The specification includes a trait named 'client-id-required' which defines 'client_id' and 'client_secret' headers. To the right, a browser-based interface for testing the API is shown. It has a 'GET' method selected, pointing to a 'Mocking Service' at 'https://mocksvc-proxy.anypoint.mulesoft.com/exc'. Under the 'Headers' tab, there are two fields: 'client_id*' and 'client_secret*', both with a checked checkbox. A 'Send' button is at the bottom right.

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

types:
  AmericanFlight: !include https://anypoint.mulesoft.com/api/american-flights/v1/flight.raml

traits:
  client-id-required:
    headers:
      client_id:
        type: string
      client_secret:
        type: string

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

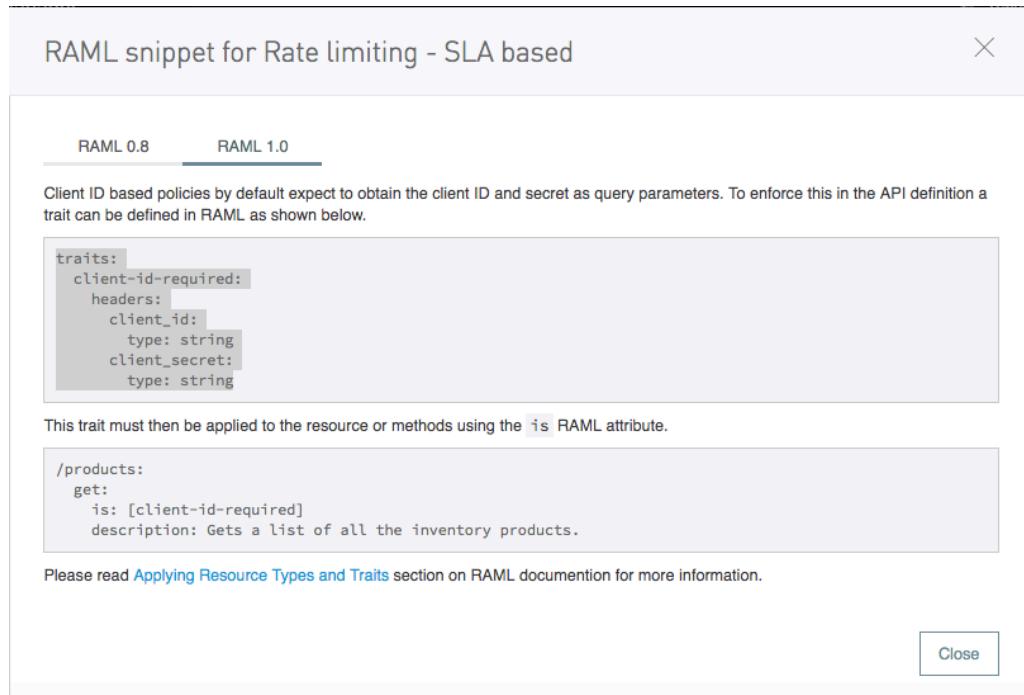
Copy the traits required to add authentication to the API specification

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

The screenshot shows the 'Policies' section of the API Manager. On the left, a sidebar lists 'Sandbox', 'API Administration', 'Alerts', 'Client Applications', 'Policies' (which is selected), 'SLA Tiers', and 'Settings'. The main content area shows a table with one row. The table columns are 'Name', 'Category', 'Fulfils', and 'Requires'. The single row contains 'Rate limiting - SLA based' under 'Name', 'Quality of service' under 'Category', 'SLA Rate Limiting, Client ID required' under 'Fulfils', and a 'RAML snippet' link under 'Requires'. Above the table, there is a 'Consumer endpoint: http://training4-american-api-mule.cloudhub.io/' and buttons for 'Apply New Policy' and 'Edit policy order'.

Name	Category	Fulfils	Requires
Rate limiting - SLA based	Quality of service	SLA Rate Limiting, Client ID required	RAML snippet

- In the RAML snippet for Rate limiting – SLA based dialog box, select RAML 1.0.
- Copy the value for the traits.



- Click Close.

Add authentication headers to the API specification

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

```

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

- Go to a new line after the /flights resource declaration and indent.

10. In the shelf, select is.

```
15 | | | |
16 /flights:
17 |
18 get:
```

Types and Traits Docs

is	description
type	displayName

11. Add empty array brackets.

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

12. Make sure the cursor is inside the brackets and in the shelf, select client-id-required.

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

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

```
17 | | | |
44 /{ID}:
45   is: [client-id-required]
46   get:
```

Test the API in the API console in Design Center

14. In the API console, turn on the mocking service.

15. Select one of the resources and click Try it.

16. Select the Headers tab; you should now see fields to enter client_id and client_secret.

The screenshot shows the Mocking service interface. At the top, there is a back arrow, a shield icon, and the text "Mocking service:" followed by a checkmark. Below this is a "Request URL" input field containing "https://mocksvc.mulesoft.com/moc". Underneath the URL, there are two tabs: "Parameters" and "Headers", with "Headers" being the active tab. In the "Headers" section, there is a checkbox labeled "</>" which is checked. Below it are two input fields: "client_id*" and "client_secret*". To the right of these fields, there is a vertical ellipsis menu with options like "Tl", "hi", "is", "re", "Tl", "hi", "is", and "re". At the bottom of the "Headers" section is a blue "Send" button.

17. Click Send; you should get a 400 Bad Request response with a message that a client_id header is required.

The screenshot shows a 400 Bad Request response. At the top, it says "400 Bad Request" and "499.00 ms". Below this is a large circular icon with a sad face and the text "The requested URL can't be reached. The service might be temporarily down or it may have moved permanently to a new web address. Resource is unavailable". At the bottom, there is a JSON error message: { "error": "headers: client_id: required" }.

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

The screenshot shows the MuleSoft Anypoint Studio Mocking service interface. At the top, there's a header with a back arrow, a shield icon, and the text "Mocking service: —". Below the header, the "Request URL" field contains "https://mocksvc.mulesoft.com/mocks". Under the "Parameters" tab, there are two fields: "client_id*" with value "432" and "client_secret*" with value "765". A checkbox labeled "Show optional parameters" is unchecked. A "Send" button is located below the parameters. The response section shows a green "200 OK" status and a time of "506.59 ms". Below the status, there are icons for copy, download, refresh, and more. The response body is a JSON array: [Array[2], -0: { "ID": 1, "code": "ER38sd", "price": 400 }].

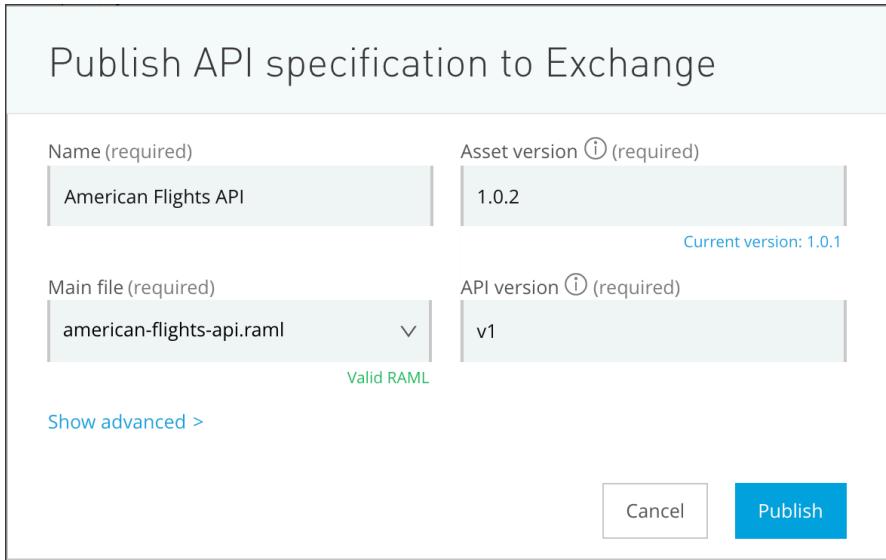
Publish the new version of the API to Exchange

19. Turn off the mocking service.

The screenshot shows the MuleSoft Anypoint Studio Mocking service interface. At the top, there's a header with a back arrow, a shield icon, and the text "Mocking service: X —". Below the header, the "Request URL" field is empty. The "Mocking service" status is now red with an "X" icon.

20. Click the Publish to Exchange button.

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



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

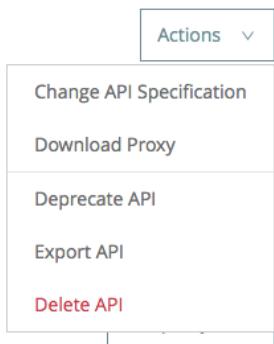
Update the managed API instance to use the new version of the API specification

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

American Flights API v1

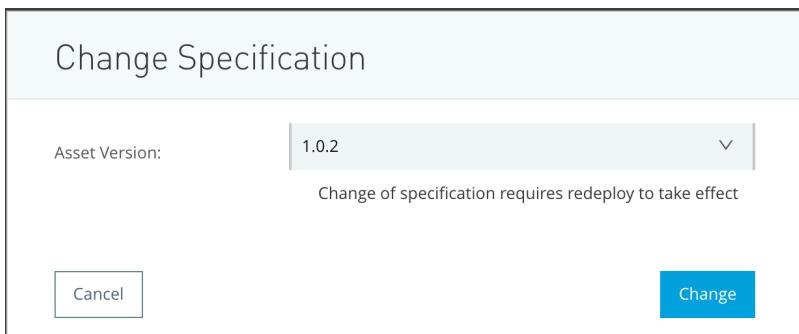
API Status: ● Active Asset Version: 1.0.1 Type: RAML/OAS
Implementation URL: <http://training4-american-ws-mule.cloudhub.io/api>
Consumer endpoint: <http://training4-american-api-mule.cloudhub.io/>

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



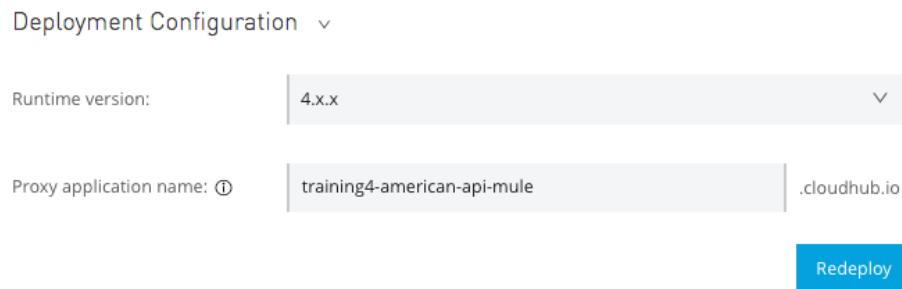
26. In the Change Specification dialog box, select the latest asset version, 1.0.2.

27. Click Change.



Redeploy a new proxy

28. In the left-side navigation, select Settings.
29. Scroll down to the Deployment Configuration settings; the Redeploy button should be disabled.
30. For the runtime version, select 4.x.x again; the Redeploy button should now be enabled.



31. Click Redeploy.
32. In the Deploying to CloudHub dialog box, click the Click here link to see the logs.
33. Watch the logs and wait until the proxy application is redeployed.

The screenshot shows the MuleSoft Runtime Manager interface. On the left, there's a sidebar with 'Sandbox', 'Applications', 'Dashboard', 'Insight', and 'Logs' (which is currently selected). The main area shows a deployment log for the application 'training4-american-api-mule'. The log entries are:

- 17:30:15.404 04/18/2018 worker-0 agw-policy-set-deployment.v1 INFO API ApiKey{id='11655540'} is now unblocked (available).
- 17:31:14.756 04/18/2018 Deployment system SYSTEM Application was updated successfully with zero downtime. The new version of your application has been launched and the old version has been stopped.
- 17:31:14.890 04/18/2018 Deployment system SYSTEM Your application is started.

To the right, there's a 'Deployments' sidebar with a list of deployment logs:

- Today > 17:28 - Deployment
- < 14:12 - Deployment
- System Log
- Worker-0

34. Close the browser tab.
35. Return to the browser tab with API Manager and click Close in the Deploying to CloudHub dialog box.

Test the rate limiting – SLA based policy in the API console in Exchange

36. Return to the browser tab with Exchange.
37. Return to the home page for the API (and refresh if necessary); you should see the new asset version listed.

Version	Instances
1.0.2	Mocking Service Sandbox - Rate limiting - SLA based policy
1.0.1	
1.0.0	

38. Click the GET method for the flights resource and select the Headers tab; you should see required text fields for client_id and client_secret and no longer need to add the headers manually for each request.

Note: You will test and use the authentication with the REST connector later in the Fundamentals course.

GET

Mocking Service

https://mocksvc-proxy.anypoint.mulesoft.com/exc

Parameters Headers

</>

client_id*

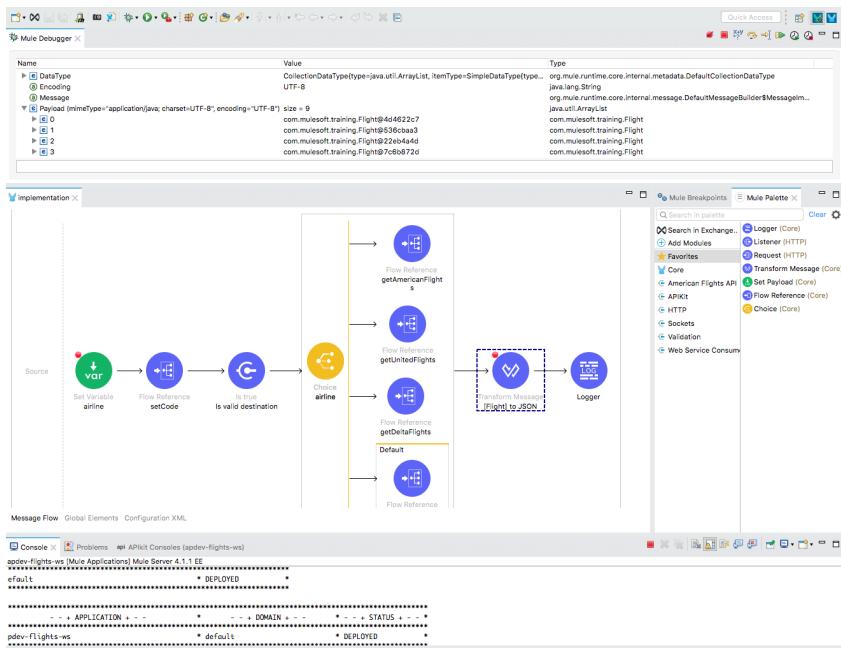
client_secret*

+ Add custom header

Send

39. Close all Anypoint Platform browser windows and tabs.

PART 2: Building Applications with Anypoint Studio



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

- Debug Mule applications.
- Read and write event payloads, attributes, & variables using the DataWeave Expression Language.
- Structure Mule applications using flows, subflows, asynchronous queues, properties files, and configuration files.
- Call RESTful and SOAP web services.
- Route and validate events and handle messaging errors.
- Write DataWeave scripts for transformations.

Module 6: Accessing and Modifying Mule Events

The screenshot shows the Mule Studio interface with the following components:

- Mule Debugger:** A table showing event properties. Key entries include:
 - Name: Attributes, Value: org.mule.extension.http.api.HttpRequestAttributes, Type: org.mule.extension.http.api.HttpRequestAttributes
 - Name: Component Path, Value: fundamentalsFlow/processors/2, Type: org.mule.runtime.dsl.api.component.config.DefaultComponentLocator
 - Name: DataType, Value: SimpleDataType(type=java.lang.String, mimeType="/*"), Type: org.mule.runtime.core.internal.metadata.SimpleDataType
 - Name: Message, Value: org.mule.runtime.core.internal.message.DefaultMessageBuilder\$...
 - Name: Payload (mimeType="/*"), Value: Hello, Type: java.lang.String
 - Name: Variables, Value: size = 1, Type: java.util.HashMap
 - Name: 0, Value: code=SFO, Type: java.util.HashMap\$Node
- DataWeave Editor:** An empty editor window titled "fundamentals".
- Mule Palette:** A panel containing various Mule components like Listener, Set Payload, Set Variable, Request, and Logger.
- Watches:** A table showing watched variables. It has three columns: Name, Value, and Type. The table is currently empty.

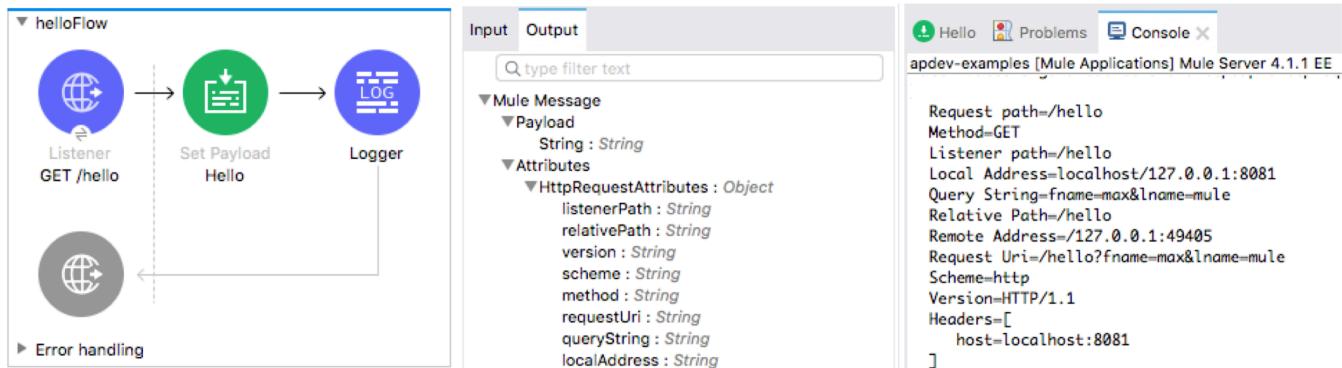
At the end this module, you should be able to:

- Log event data.
- Debug Mule applications.
- Read and write event properties.
- Write expressions with the DataWeave expression language.
- Create variables.

Walkthrough 6-1: View event data

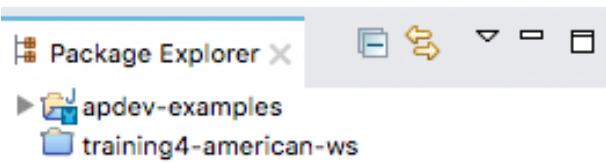
In this walkthrough, you create a new project to use in the next two modules to learn about Mule events and Mule applications. You will:

- Create a new Mule project with an HTTP Listener and set the message payload.
- View event data in the DataSense Explorer.
- Use a Logger to view event data in the Anypoint Studio console.



Create a new Mule project

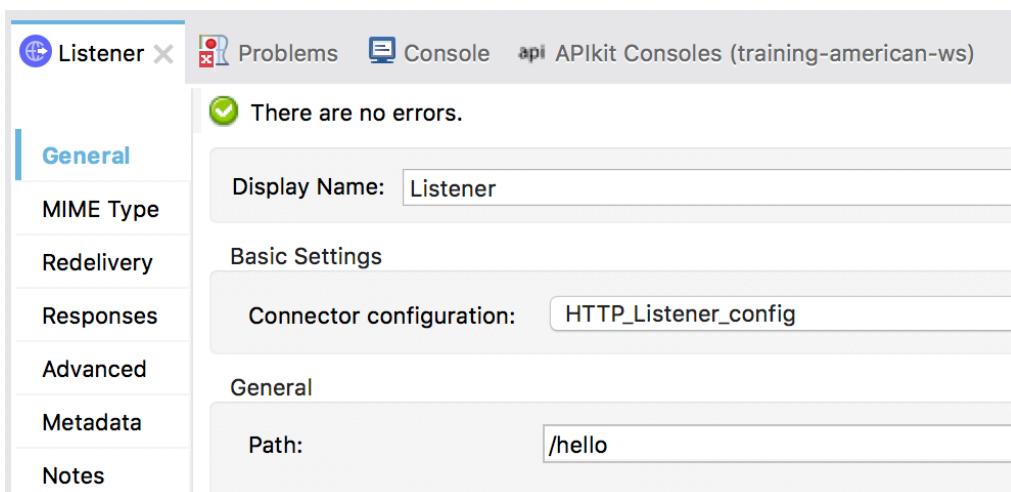
1. Return to Anypoint Studio.
2. Right-click training4-american-ws and select Close Project.
3. Select File > New > Mule Project.
4. Set the project name to apdev-examples and click Finish.



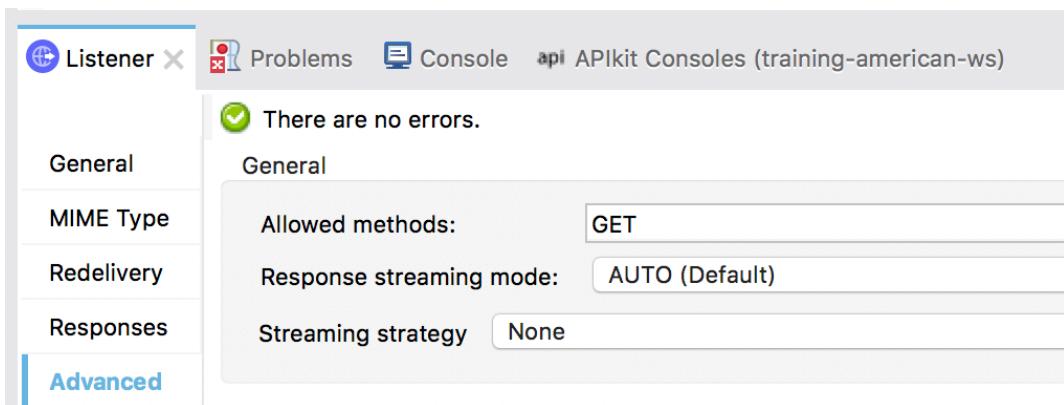
Create an HTTP Listener to receive requests

5. In the Mule Palette, select Favorites.
6. Drag an HTTP Listener from the Mule Palette to the canvas.
7. In the Listener properties view, click the Add button next to Connector configuration.
8. In the Global Element Properties dialog box, set the host to 0.0.0.0 and the port to 8081.
9. Click OK.

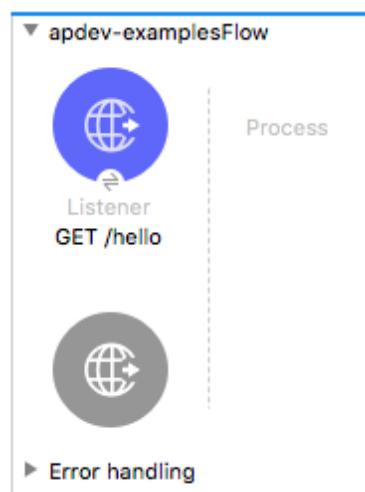
10. In the Listener properties view, set the path to /hello.



11. Click the Advanced tab and set the allowed methods to GET.



12. Click the General tab and set the display name to GET /hello.

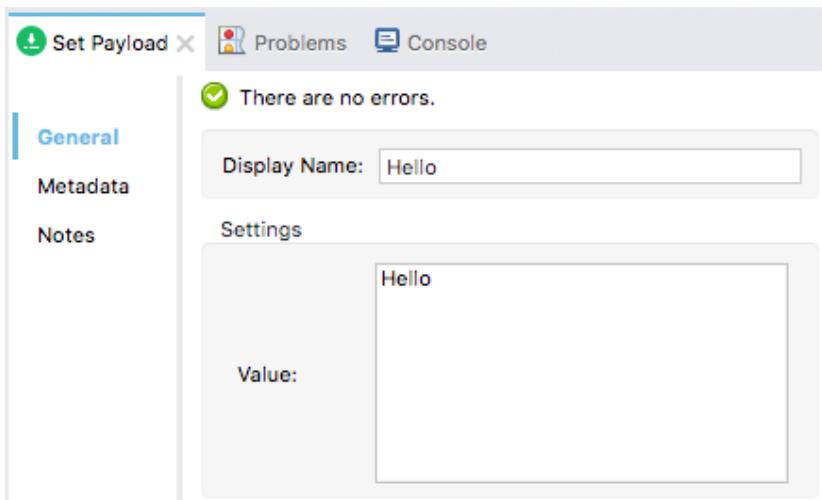


Change the flow name

13. Select the flow.
14. In the apdev-examplesFlow properties view, change the name to helloFlow.

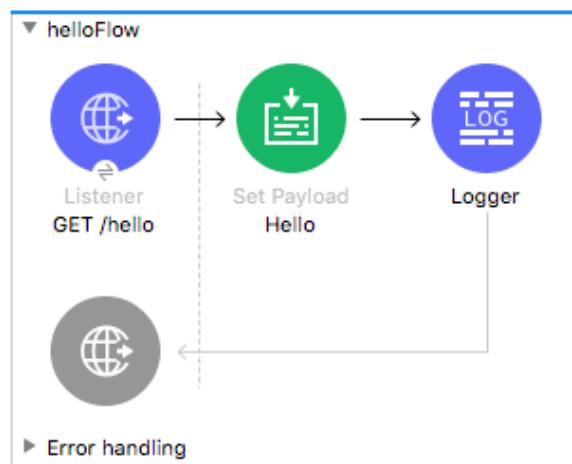
Set the message payload

15. Drag a Set Payload transformer from the Favorites section of the Mule Palette into the process section of the flow.
16. In the Set Payload properties view, set the display name to Hello.
17. Set the value to Hello.



Add a Logger

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



View event structure and metadata in the DataSense Explorer

19. Select the GET /hello HTTP Listener and locate the DataSense Explorer in the right-side of its properties view.
20. Select the Input tab and expand Payload and Attributes.

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. At the top, there are tabs for 'Input' and 'Output'. Below them is a search bar labeled 'Q type filter text'. The main content area displays the structure of a 'Mule Message':

- ▼ Mule Message
 - ▼ Payload
 - Undefined : Unknown
 - ▼ Attributes
 - Undefined : Unknown
 - Variables

21. Select the Output tab and expand Payload and Attributes.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. The structure of a 'Mule Message' is shown, similar to the 'Input' tab, but with more detailed expansion of the 'Attributes' section:

- ▼ Mule Message
 - ▼ Payload
 - Binary : Binary
 - ▼ Attributes
 - ▼ HttpRequestAttributes : Object
 - listenerPath : String
 - relativePath : String
 - version : String
 - scheme : String
 - method : String
 - requestUri : String
 - queryString : String
 - localAddress : String
 - remoteAddress : String
 - clientCertificate : Object?
 - queryParams : Object
 - uriParams : Object
 - requestPath : String
 - headers : Object

22. Select the Set Payload component in helloFlow.

23. In the DataSense Explorer, select the Input tab and expand Payload and Attributes.

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. The structure of a 'Mule Message' is shown, similar to the 'Output' tab, but with more detailed expansion of the 'Attributes' section:

- ▼ Mule Message
 - ▼ Payload
 - Binary : Binary
 - ▼ Attributes
 - ▼ HttpRequestAttributes : Object
 - listenerPath : String
 - relativePath : String
 - version : String

24. Select the Output tab and expand Payload and Attributes.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top contains the placeholder 'type filter text'. Below it, under 'Mule Message', the 'Payload' section is expanded, showing 'String : String'. The 'Attributes' section is also expanded, showing 'HttpRequestAttributes : Object'. There is a 'Variables' section at the bottom.

25. Select the Logger component in helloFlow.

26. In the DataSense Explorer, select the Input tab and expand Payload and Attributes.

27. Select the Output tab and expand the Payload and Attributes.

Run the application and review response data

28. Save the file and run the project.

29. Return to Advanced REST Client and click the button to create a new tab.

Note: You are adding a new tab so that you can keep the request to your American API saved in another tab for later use.

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

The screenshot shows the Advanced REST Client interface. At the top, 'Method' is set to 'GET' and 'Request URL' is 'http://localhost:8081/hello'. Below that, there's a 'Parameters' dropdown. On the right, there's a 'SEND' button and a more options menu. The main area shows a green '200 OK' status box with '190.62 ms' response time. To the right is a 'DETAILS' dropdown. Below the status are download and copy icons. The payload 'Hello' is displayed at the bottom.

View event data in the Anypoint Studio console

31. Return to Anypoint Studio and look at the console.

32. Locate the data displayed by using the Logger.

33. Find where the data type of the payload is specified.

34. Review the event attributes.

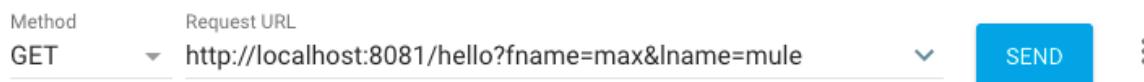


```
*****
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*****
* apdev-examples           * default           * DEPLOYED
*****
```

```
INFO 2018-04-19 09:42:56,942 [[MuleRuntime].cpuLight.07: [apdev-examples].helloFlow.CPU_LITE @43b0c8d4]
org.mule.runtime.core.internal.message.DefaultMessageBuilder$MessageImplementation
{
    payload=java.lang.String
    mediaType=*/
    attributes=org.mule.extension.http.api.HttpRequestAttributes
{
    Request path=/hello
    Method=GET
    Listener path=/hello
    Local Address=localhost/127.0.0.1:8081
    Query String=
    Relative Path=/hello
    Remote Address=/127.0.0.1:49375
    Request Uri=/hello
    Scheme=http
    Version=HTTP/1.1
    Headers=[
        host=localhost:8081
    ]
    Query Parameters={}
    URI Parameters={}
}
    attributesMediaType=*/
    exceptionPayload=<not set>
}
```

Send query parameters with a request

35. Return to Advanced REST Client and add a query parameter with a key of fname and a value of max.
36. Add a second key/value pair of lname and mule.
37. Click Send.



Method Request URL
GET http://localhost:8081/hello?fname=max&lname=mule

38. Return to Anypoint Studio and look at the console.

39. Locate the query parameters in the logged event data.

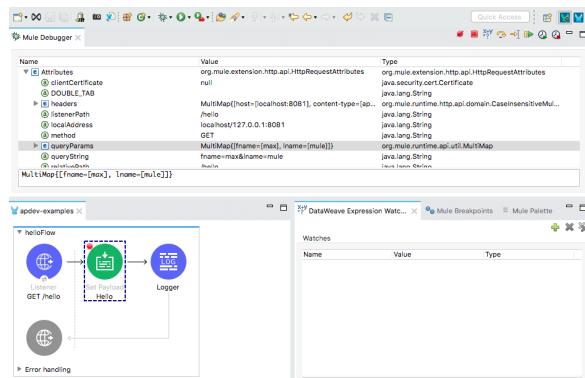
```
{  
    payload=java.lang.String  
    mediaType=/*  
    attributes=org.mule.extension.http.api.Http  
{  
    Request path=/hello  
    Method=GET  
    Listener path=/hello  
    Local Address=localhost/127.0.0.1:8081  
    Query String=fname=max&lname=mule  
    Relative Path=/hello  
    Remote Address=/127.0.0.1:49405  
    Request Uri=/hello?fname=max&lname=mule  
    Scheme=http  
    Version=HTTP/1.1  
    Headers=[  
        host=localhost:8081  
    ]  
    Query Parameters=[  
        fname=max  
        lname=mule  
    ]  
    URI Parameters=[]  
}  
    attributesMediaType=/*  
    exceptionPayload=<not set>  
}
```

40. Stop the project.

Walkthrough 6-2: Debug a Mule application

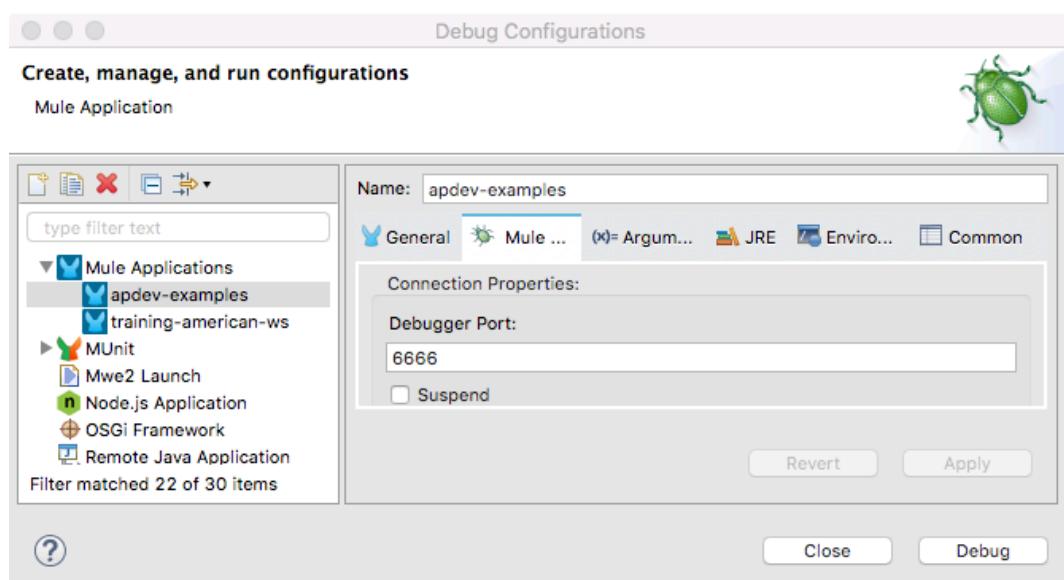
In this walkthrough, you debug and step through the code in a Mule application. You will:

- Locate the port used by the Mule Debugger.
- Add a breakpoint, debug an application, and step through the code.
- Use the Mule Debugger to view event properties.
- Pass query parameters to a request and locate them in the Mule Debugger.
- Increase the request timeout for Advanced REST Client.



Locate the port used by the Mule Debugger

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

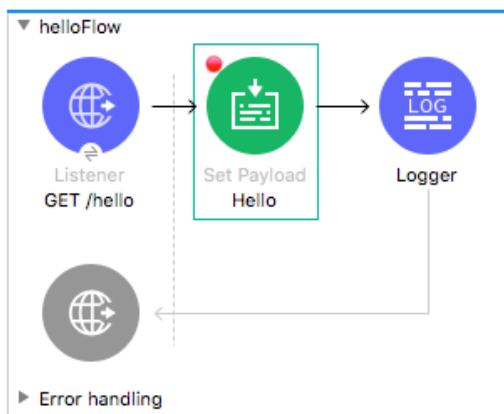


Note: If you know you have another program using port 6666 like McAfee on Windows, change this to a different value. Otherwise, you can test the Debugger first and come back and change the value here later if there is a conflict.

5. Click Close.

Add a breakpoint

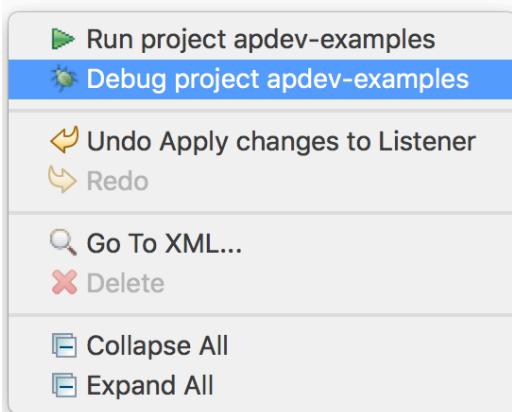
6. Right-click the Set Payload component and select Toggle breakpoint.



Debug the application

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

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



8. If you get a Confirm Perspective Switch dialog box, select Remember my decision and click Yes.
9. In the Debug perspective, close the MUnit view.
10. Look at the console and wait until the application starts.

11. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.

View event data in the Mule Debugger

12. Return to Anypoint Studio and locate the Mule Debugger view.
13. Look at the value of the payload.
14. Expand Attributes and review the values.
15. Locate the queryParams object.

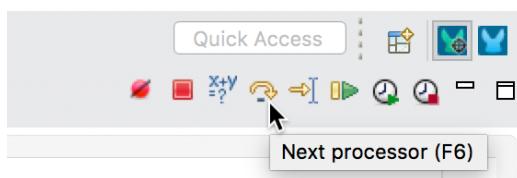
The screenshot shows the Anypoint Studio interface with the following components:

- Mule Debugger View:** Shows a table of event data. One row is expanded to show the `queryParams` object as a `MultiMap` containing `fname=[max]` and `lname=[mule]`.
- Message Flow Editor:** Displays a flow named `helloFlow` with three components: Listener (GET /hello), Set Payload (Hello), and Logger.
- Console View:** Shows deployment information for the application `apdev-examples`:


```
*****
* - + APPLICATION + - - * - + DOMAIN + - - * - + STATUS + - - *
*****  
* apdev-examples * default * DEPLOYED *
```

Step through the application

16. Click the Next processor button.



17. Look at the new value of the payload; it should be Hello.

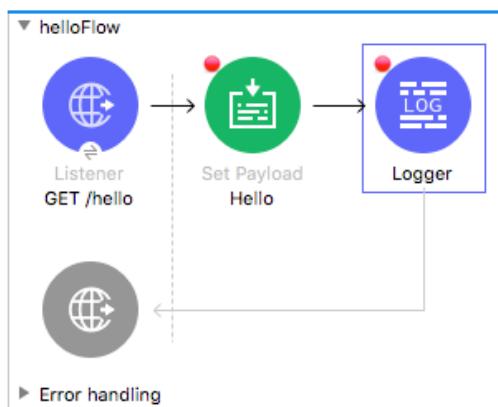
The screenshot shows the Mule Debugger interface with two tabs: "Mule Debugger" and "apdev-examples". The "Mule Debugger" tab displays a table of attributes and their values. One row shows the "Payload (mimeType='*/*')" attribute with the value "Hello". Below the table, the word "Hello" is displayed in a text area. The "apdev-examples" tab shows a flow diagram for "helloFlow". The flow starts with a "Listener GET /hello" component, followed by a "Set Payload" component with the value "Hello", and finally a "Logger" component. A red breakpoint icon is placed on the connection between the Set Payload and Logger components.

18. Expand Attributes and review the values.

19. Click the Next processor button again to finish stepping through the application.

Use Resume to step to the next breakpoint and then the end of the application

20. Add a breakpoint to the Logger.



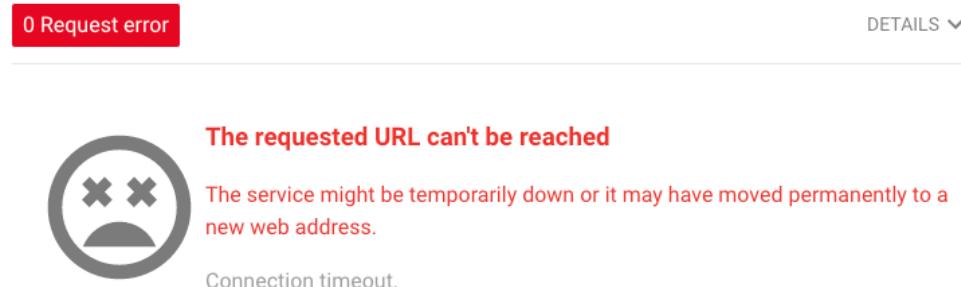
21. In Advanced REST Client, make another request to

<http://localhost:8081/hello?fname=max&lname=mule>.

22. In the Mule Debugger, click the Resume button; you should step to the Logger.
23. Click the Resume button again; you should step through the rest of the application.

Cause the HTTP request to timeout

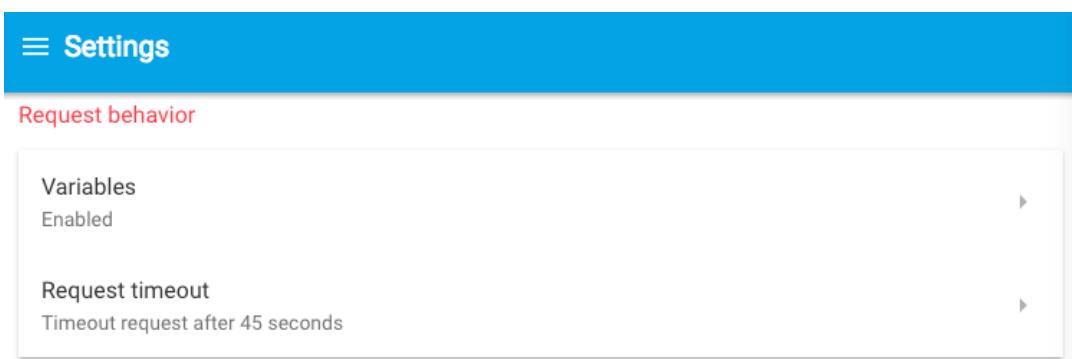
24. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.
25. Do not step through the application and wait (45 seconds) for the request to timeout.



The screenshot shows the Advanced REST Client interface. At the top, there is a red button labeled "0 Request error" and a "DETAILS" dropdown. Below this, a large error icon (a sad face with crossed-out eyes) is displayed next to the text "The requested URL can't be reached". A message follows: "The service might be temporarily down or it may have moved permanently to a new web address." At the bottom, it says "Connection timeout."

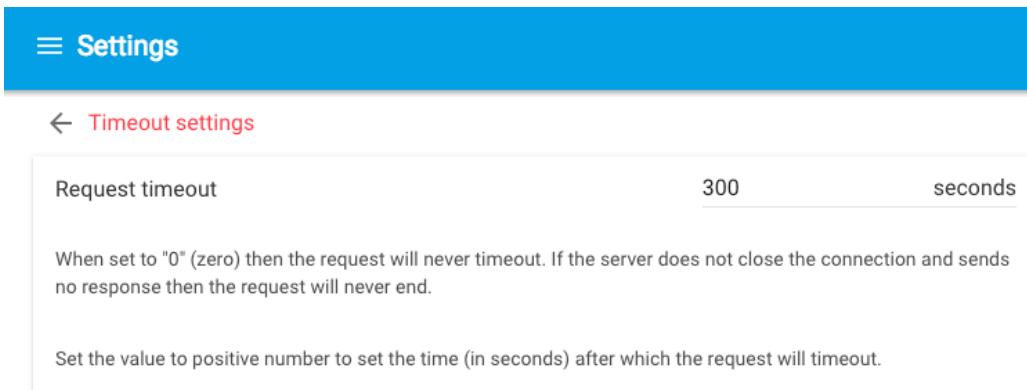
Increase the request timeout for Advanced REST Client

26. In the Advanced REST Client main menu, select Preferences.
27. In the Settings, locate the Request timeout setting and click the arrow next to it.



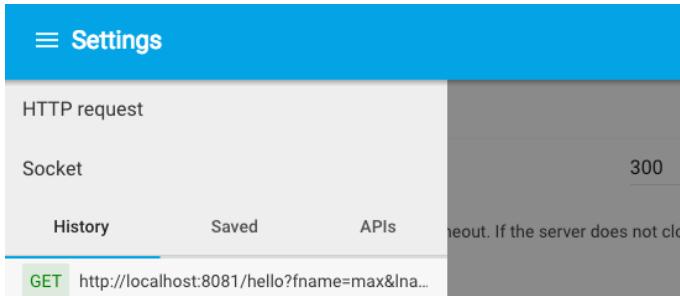
The screenshot shows the "Settings" menu of the Advanced REST Client. Under the "Request behavior" section, there are two items: "Variables" (Enabled) and "Request timeout" (Timeout request after 45 seconds). Each item has a small arrow icon to its right, indicating it can be expanded or collapsed.

28. Change the request timeout to 300 seconds.

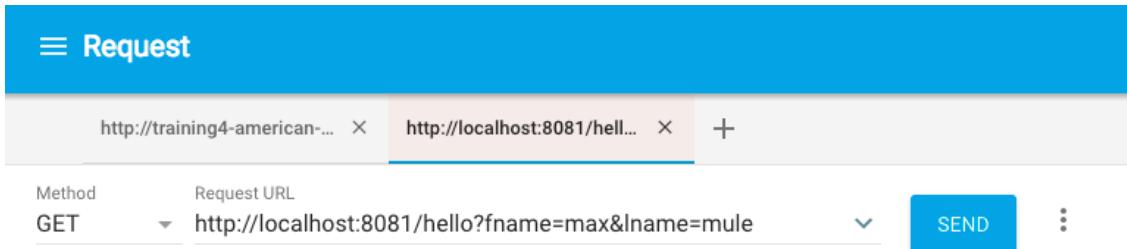


The screenshot shows the "Timeout settings" dialog. It has a "Request timeout" field set to "300 seconds". Below the field, a note states: "When set to \"0\" (zero) then the request will never timeout. If the server does not close the connection and sends no response then the request will never end." At the bottom, a note says: "Set the value to positive number to set the time (in seconds) after which the request will timeout."

29. Click the Toggle Drawer button in the upper-left corner next to Settings and click HTTP Request.

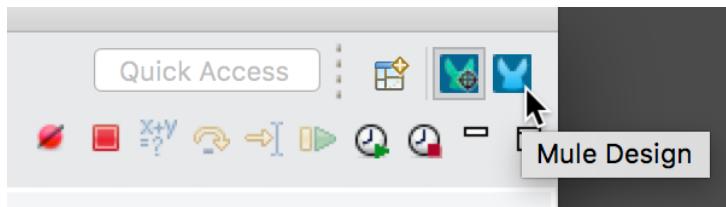


30. Click the Toggle Drawer button again to hide the drawer; you should now see the Request screen again.



Switch perspectives

31. Return to Anypoint Studio.
32. Click the Resume button.
33. Click the Mule Design button in the upper-right corner of Anypoint Studio to switch perspectives.



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

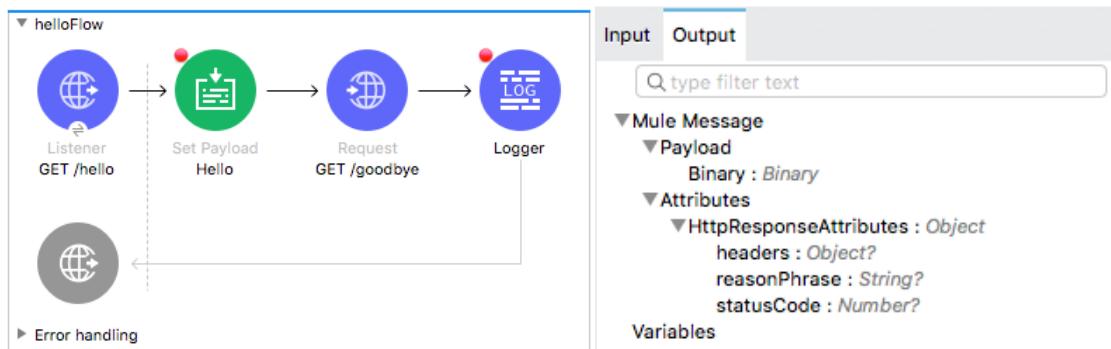
34. Leave the project running.

Walkthrough 6-3: Track event data as it moves in and out of a Mule application

In this walkthrough, you call an external resource, which for simplicity is another HTTP Listener in the same application, so that you can watch event data as it moves in and out of a Mule application. You will:

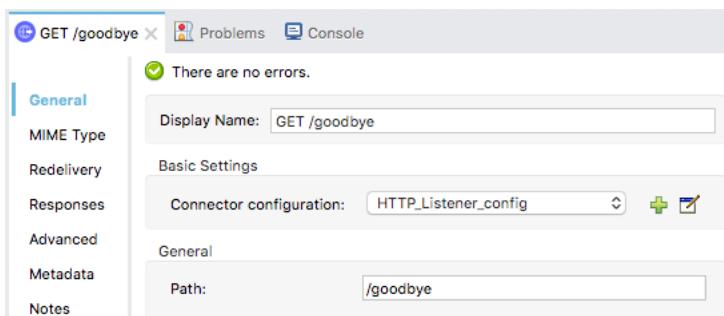
- Create a second flow with an HTTP Listener.
- Make an HTTP request from the first flow to the new HTTP Listener.
- View the event data as it moves through both flows.

*Note: You are making an HTTP request from one flow to another in this exercise **only** so you can watch the value of event data as it moves in and out of a Mule application. You will learn how to pass events between flows within and between Mule applications in the next module.*

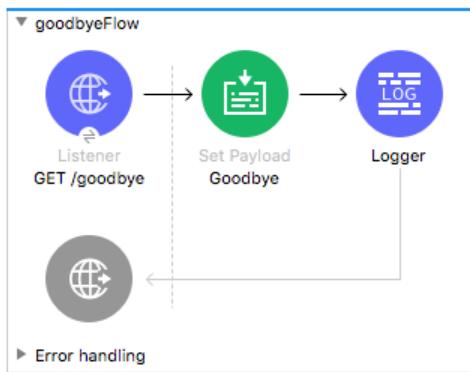


Create a second flow with an HTTP Listener

1. Return to apdev-examples.xml.
2. Drag an HTTP Listener from the Mule Palette and drop it in the canvas beneath the first flow.
3. Change the name of the flow to goodbyeFlow.
4. In the Listener view, set the connector configuration to the existing HTTP_Listener_config.
5. Set the path to /goodbye and the allowed methods to GET.
6. Set the display name to GET /goodbye.

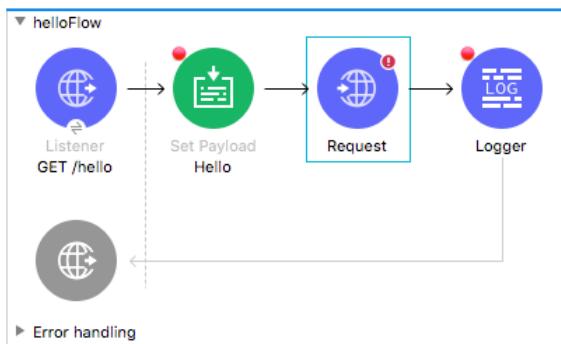


7. Add a Set Payload transformer and a Logger to the flow.
8. In the Set Payload properties view, set the display name and value to Goodbye.

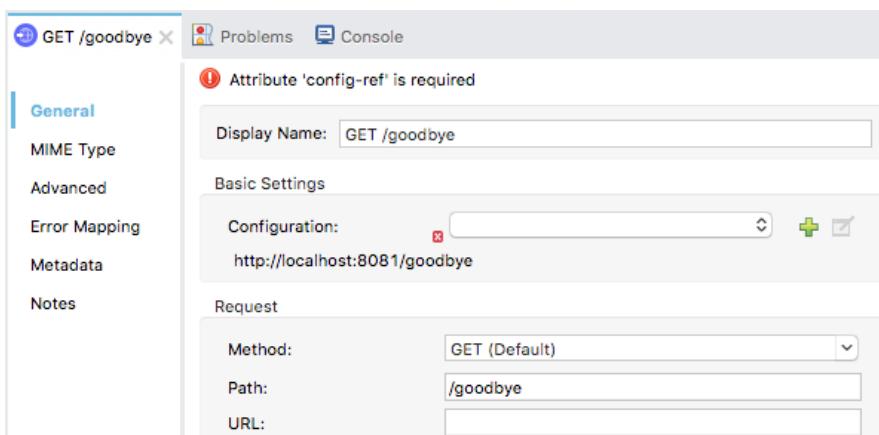


Make an HTTP request

9. From the Mule Palette, drag an HTTP Request to the canvas and drop it before the Logger in helloFlow.

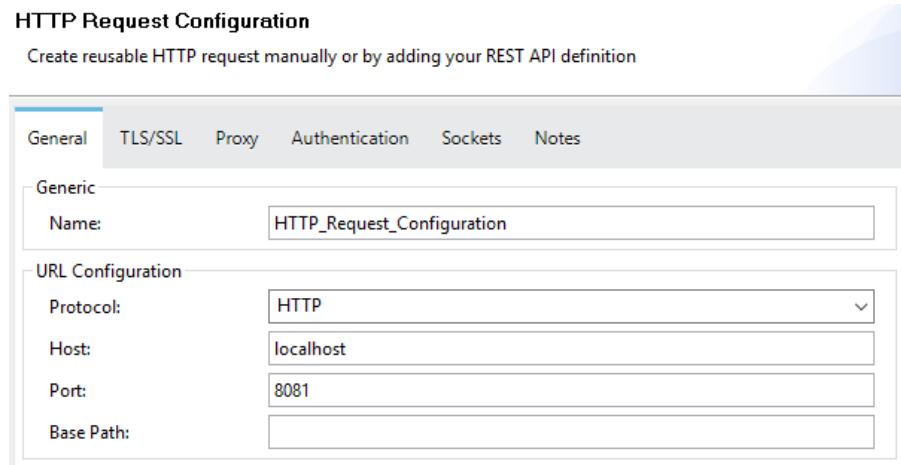


10. In the Request properties view, set the display name to GET /goodbye.
11. Set the path to /goodbye and leave the method set to GET.



12. Click the Add button next to configuration.

13. In the dialog box, set the host to localhost and the port to 8081 and click OK.



View event structure and metadata in the DataSense Explorer

14. In the DataSense Explorer, expand Payload and Attributes in the Input tab.

Input Output

Mule Message

Payload

(Actual) String : String

(Expected) Object : Object

Attributes

HttpRequestAttributes : Object

clientCertificate : Object?

headers : Object?

listenerPath : String?

localAddress : String?

method : String?

queryParams : Object?

queryString : String?

relativePath : String?

remoteAddress : String?

requestPath : String?

requestUri : String?

scheme : String?

uriParams : Object?

version : String?

Variables

15. Select the Output tab and expand Payload and Attributes.

Input Output

Mule Message

Payload

Binary : Binary

Attributes

HttpResponseAttributes : Object

headers : Object?

reasonPhrase : String?

statusCode : Number?

Variables

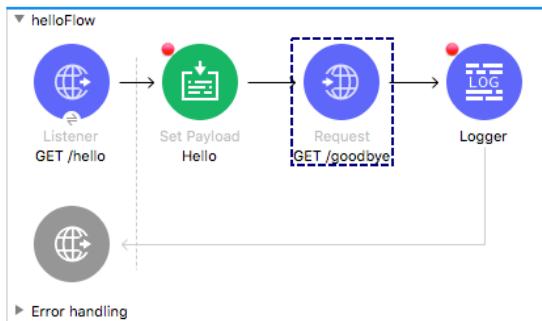
Change timeout for HTTP Request response

16. In the properties view for the GET /goodbye HTTP Request, locate the Response section on the General tab.
17. Set the Timeout to 300000.

Note: This is being set only for debugging purposes so that the HTTP Request does not timeout when you are stepping through the application and examining data in the Mule Debugger.

Debug the application

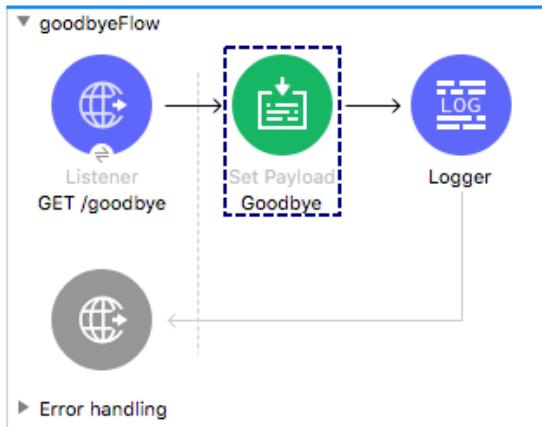
18. Save the file to redeploy the application.
19. In Advanced REST Client, send the same request to <http://localhost:8081/hello?fname=max&lname=mule>.
20. In the Mule Debugger, step to the GET /goodbye request.



21. Look at the values of the payload and the attributes, including the queryParams.

Name	Value
Attributes	org.mule.extension.http.api.HttpRequest/
clientCertificate	null
DOUBLE_TAB	
headers	MultiMap[[host=[localhost:8081]]]
listenerPath	/hello
localAddress	localhost/127.0.0.1:8081
method	GET
queryParams	MultiMap[[fname=[max], lname=[mule]]]
queryString	fname=max&lname=mule
relativePath	/hello
remoteAddress	/127.0.0.1:51469
requestPath	/hello
requestUri	/hello?fname=max&lname=mule
scheme	http
serialVersionUID	7227330842640270811
TAB	
uriParams	MultiMap[[]]
version	HTTP/1.1
Component Path	helloFlow/processors/1
DataType	SimpleDataType{type=java.lang.String, m
Message	
Payload (mimeType="*/*")	Hello

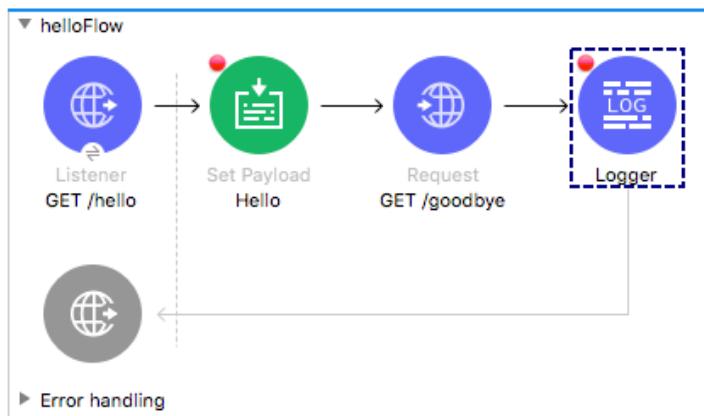
22. Step into goodbyeFlow.



23. Look at the values of the payload and the attributes.

Name	Value
Attributes	
@clientCertificate	org.mule.extension.http.api.HttpReque...
@DOUBLE_TAB	null
headers	MultiMap{[x-correlation-id=[0-440733...}}
@listenerPath	/goodbye
@localAddress	localhost/127.0.0.1:8081
@method	GET
queryParams	MultiMap{[]}
@queryString	
@relativePath	/goodbye
@remoteAddress	/127.0.0.1:51473
@requestPath	/goodbye
@requestUri	/goodbye
@scheme	http
@serialVersionUID	7227330842640270811
@TAB	
uriParams	MultiMap{[]}
@version	HTTP/1.1
Component Path	goodbyeFlow/processors/0
dataType	SimpleDataType{type=org.mule.runtim...
Encoding	UTF-8
Message	
Payload (mimeType="*/*; charset=UTF-8")	

24. Step through the flow until the event returns to the Logger in helloFlow.



25. Look at the values of the payload and the attributes.

Name	Value
Attributes	org.mule.extension.http.api.HttpResponse
DOUBLE_TAB	③
headers	MultiMap{[content-length=[7], date=[Thu, 19 Apr 2018 19:39:33 GMT], content-length=7]}
0	date=Thu, 19 Apr 2018 19:39:33 GMT
1	content-length=7
reasonPhrase	
serialVersionUID	-3131769059554988414
statusCode	200
TAB	③
Component Path	helloFlow/processors/2
dataType	SimpleDataType{type=org.mule.runtime.
Encoding	UTF-8
Message	
Payload (mimeType="appli...")	Goodbye

26. Step through the rest of the application.

Review response data

27. Return to Advanced REST Client and view the return data and the http status code.
28. Click the Details button on the right side to expand this section.
29. Look at the response headers; you should see content-length, content-type, and date headers.

200 OK 86431.06 ms DETAILS ^

GET http://localhost:8081/hello?fname=max&lname=mule

Response headers (3) Request headers (0) Redirects (0) Timings

content-type: application/octet-stream; charset=UTF-8
content-length: 7
date: Thu, 19 Apr 2018 20:20:41 GMT

□ ↻ ↺ ↻

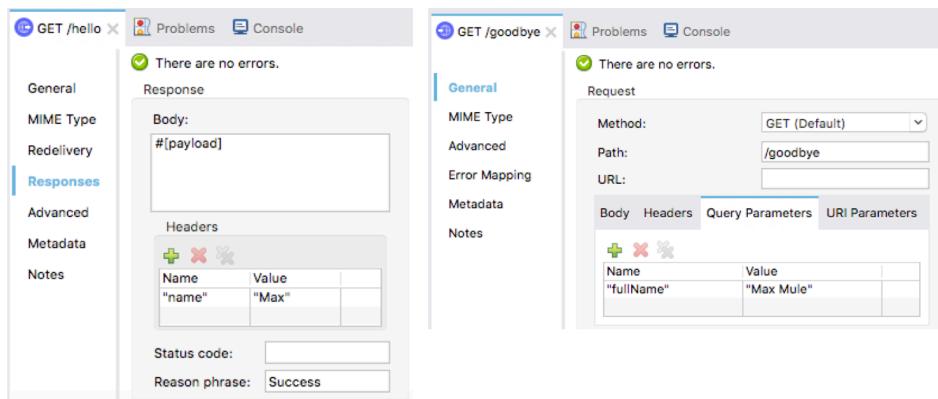
Goodbye

30. Return to Anypoint Studio and switch to the Mule Design perspective.

Walkthrough 6-4: Set request and response data

In this walkthrough, you set response and request data for HTTP Listener and HTTP Request operations. You will:

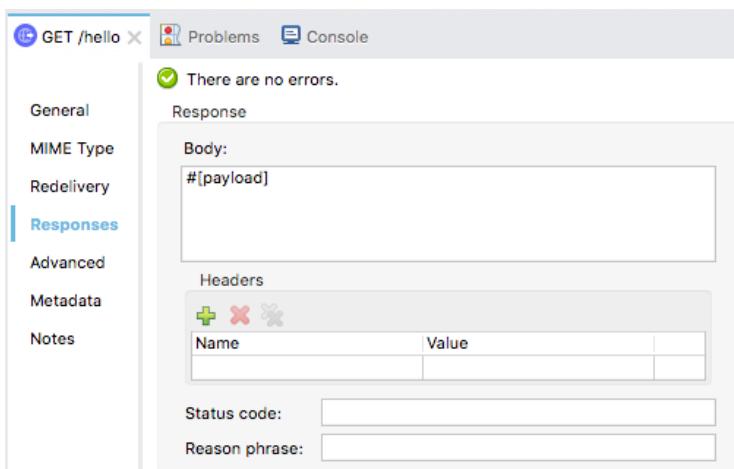
- View the default setting for a response body.
- Set a response header.
- View the default setting for a request body.
- Set a request query parameter.



View default response body

1. Return to apdev-examples.xml.
2. Double-click the GET /hello HTTP Listener in helloFlow.
3. In the GET /hello properties view, select the Responses tab.
4. In the Response section, locate the expression that sets the response body by default to the value of the payload.

Note: The syntax for expressions will be covered in the next walkthrough.



Set a response header

5. In the Headers section for the response, click the Add button.
6. Set the name to "name" and the value to "Max".
7. Locate the status code field and leave it blank so the default value 200 is returned.
8. Set the reason phrase to Success.

The screenshot shows the Mule Studio interface with a GET /hello request selected. The left sidebar has tabs for General, MIME Type, Redelivery, Responses (which is selected), Advanced, and Metadata. The main area shows a table for Headers with one row: Name "name" and Value "Max". Below the table are fields for Status code (empty) and Reason phrase (Success).

Run the application and review response data

9. Save the file to deploy the project.
10. Return to Advanced REST Client and send the same request.
11. In the Mule Debugger, click Resume until you step through the application.
12. In Advanced REST Client, locate your new status code reason phrase.
13. Review the response headers; you should now see the new name header.

The screenshot shows a successful response from the Advanced REST Client. The status bar indicates 200 Success and 1759.59 ms. The request URL is GET http://localhost:8081/hello?fname=max&lname=mule. The Response headers section shows:

Name	Value
name	Max
content-type	application/octet-stream; charset=UTF-8
content-length	7
date	Thu, 19 Apr 2018 20:46:15 GMT

Below the headers, there are icons for copy, download, refresh, and eye. The word "Goodbye" is also visible.

View default request body

14. Return to Anypoint Studio and switch perspectives.
15. Double-click the GET /goodbye HTTP Request in helloFlow.

16. In the GET /goodbye properties view, locate the Request section on the General tab.
17. On the Body tab, locate the expression that sets the request body by default to the value of the payload.

The screenshot shows the Mule Studio interface with the 'GET /goodbye' configuration open. The left sidebar lists tabs: General, MIME Type, Advanced, Error Mapping, Metadata, and Notes. The 'General' tab is selected. The main area shows the 'Request' configuration with the 'Method' set to 'GET (Default)', 'Path' set to '/goodbye', and 'URL' field empty. Below the method and path fields is a tab bar with 'Body', 'Headers', 'Query Parameters', and 'URI Parameters'. The 'Body' tab is active, displaying the expression '#[payload]'.

Set a request query parameter

18. Select the Query Parameters tab and click the Add button.
19. Set the name to "fullName" and the value to "Max Mule".

The screenshot shows the 'Query Parameters' tab selected in the 'Request' configuration. A table is displayed with two columns: 'Name' and 'Value'. There is one row with the values 'fullName' and 'Max Mule' respectively. Above the table are three icons: a green plus sign for adding new entries, a red minus sign for removing existing entries, and a grey cross icon.

Name	Value
"fullName"	"Max Mule"

Debug and verify the query parameter is sent with the request

20. Save the file to redeploy the project.
21. Return to Advanced REST Client and send the same request.
22. Return to the Mule Debugger and step into goodbyeFlow.

23. Expand Attributes and locate the fullName query parameter.

Name	Value
method	GET
queryParams	MultiMap{[fullName=[Max Mule]]}
0	fullName=Max Mule
key	fullName
value	Max Mule
queryString	fullName=Max Mule
relativePath	/goodbye
remoteAddress	/127.0.0.1:52451
requestPath	/goodbye
requestUri	/goodbye?fullName=Max Mule

24. Step through the rest of the application.

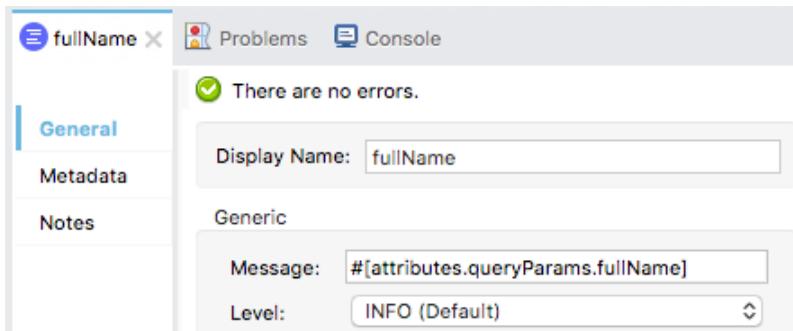
25. Switch to the Mule Design perspective.

26. Stop the project.

Walkthrough 6-5: Get and set event data using DataWeave expressions

In this walkthrough, you get and set event data using DataWeave expressions. You will:

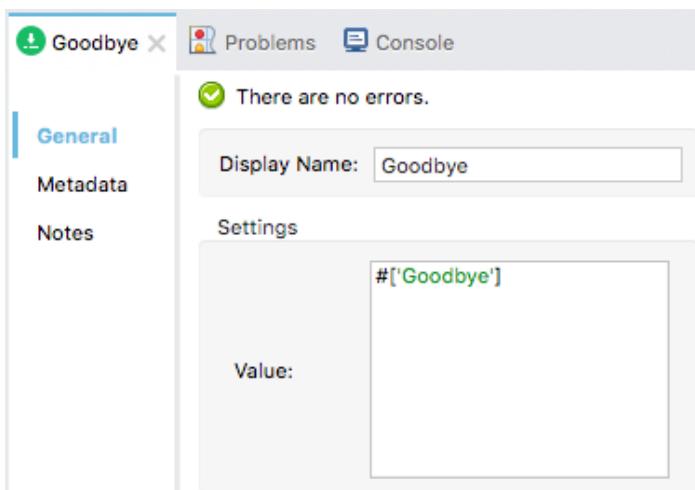
- Use expressions to set the payload and a logged value.
- Use expressions to set a response header and a request query parameter.
- In expressions, reference values for the event payload and attributes.
- Use the DataWeave upper() function and the concatenation, as, and default operators.



Use an expression to set the payload

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

```
#['Goodbye']
```



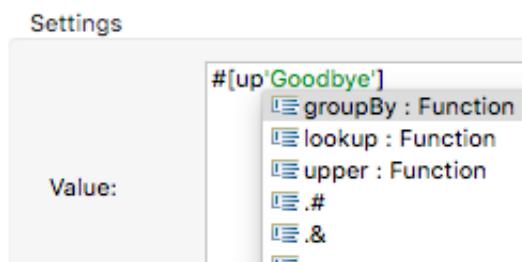
4. Run the project.

5. In Advanced REST Client, send the same request; you should get the same result of Goodbye as before.

A screenshot of the Advanced REST Client interface. At the top, there is a green button labeled "200 Success" and the text "3165.37 ms". Below this is a toolbar with icons for copy, paste, download, refresh, and search. The main content area displays the word "Goodbye".

Use a DataWeave function

6. In Anypoint Studio, return to the Goodbye Set Payload properties view.
7. Inside the brackets and before the string, type the word up and press Ctrl+Spacebar.
8. In the auto-completion menu, select the upper function.



9. Add parentheses around the Goodbye string.

```
# [upper( 'Goodbye' )]
```

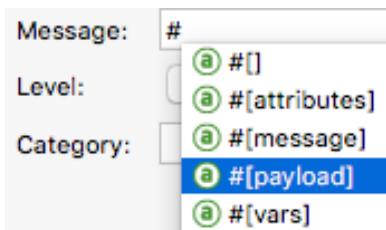
10. Save the file to redeploy the application.
11. In Advanced REST Client, send the same request; the return string should now be in upper case.

A screenshot of the Advanced REST Client interface. At the top, there is a green button labeled "200 Success" and the text "2253.95 ms". Below this is a toolbar with icons for copy, paste, download, refresh, and search. The main content area displays the word "GOODBYE".

Use an expression that references the payload in a Logger

12. Return to Anypoint Studio.
13. Navigate to the properties view for the Logger in helloFlow.
14. Change the display name to payload.

15. Type # in the message field and select #[payload] in the auto-completion menu.



16. Save the file to redeploy the application.

17. In Advanced REST Client, send the same request.

18. Return to the console in Anypoint Studio; you should see GOODBYE displayed for the second Logger instead of the entire event object.

```
INFO 2018-04-19 14:10:48,008 [[MuleRuntime].cpuLight.06: [apdev-examples].helloFlow.CPU_LITE @5dc0c3a3] [event: 0-216c8710-4416-11e8-861e-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: GOODBYE
```

Use a string literal and a DataWeave expression in a property value

19. Return to the properties view for the Logger in helloFlow.

20. Change the value to display the string Message in front of the payload.

Message: #[payload]

21. Save the file to redeploy the application.

22. In Advanced REST Client, send the same request.

23. Return to the console in Anypoint Studio; you should see the new string displayed.

```
INFO 2018-04-19 14:15:43,640 [[MuleRuntime].cpulight.16: [apdev-examples].helloFlow.CPU_LITE @3dd1f9a1] [event: 0-d1a23d00-4416-11e8-b833-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Message: GOODBYE
```

24. Return to the properties view for the Logger in helloFlow.

Use the DataWeave concatenation operator

25. Change the value so the string is part of the evaluated expression and use the concatenation operator.

#['Message: ' ++ payload]

26. Add \n in front of the message to display it on a new line.

#['\nMessage: ' ++ payload]

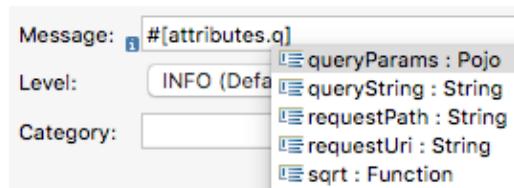
27. Save the file to redeploy the application.

28. In Advanced REST Client, send the same request.
29. Return to the console in Anypoint Studio; you should see the string displayed on a new line.

```
INFO 2018-04-19 14:18:29,925 [[MuleRuntime].cpuLight.09: [apdev-examples].helloFlow.CPU_LITE @1d096383] [event: 0-34ddd
550-4417-11e8-b833-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor:
Message: GOODBYE
```

Use an expression that references an attribute in a Logger

30. Return to the properties view for the Logger in goodbyeFlow.
31. Change the display name to fullName.
32. Type # in the message field and select #[attributes] in the auto-completion menu.
33. At the end of attributes, add a period, type q and select queryParams in the auto-completion menu.



`##[attributes.queryParams]`

34. Click Apply Changes to redeploy the application.
35. In Advanced REST Client, send the same request.
36. Return to the Anypoint Studio console; you should see an object displayed by the first Logger.

```
INFO 2018-04-19 14:25:27,821 [[MuleRuntime].cpuLight.15: [apdev-examples].goodbyeFlow.CPU_LITE @7c601de9] [event: 0-2df
95920-4418-11e8-b833-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {fullName=Max Mule}
```

37. Modify the message for the Logger in goodbyeFlow to display the value of the query parameter.

`##[attributes.queryParams.fullName]`

38. Click Apply Changes to redeploy the application.
39. In Advanced REST Client, send the same request.
40. Return to the console; you should now see the value of the parameter displayed.

```
INFO 2018-04-19 14:26:22,192 [[MuleRuntime].cpuLight.03: [apdev-examples].goodbyeFlow.CPU_LITE @1e7dbbd3] [event: 0-4e6
55dd0-4418-11e8-b833-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Max Mule
```

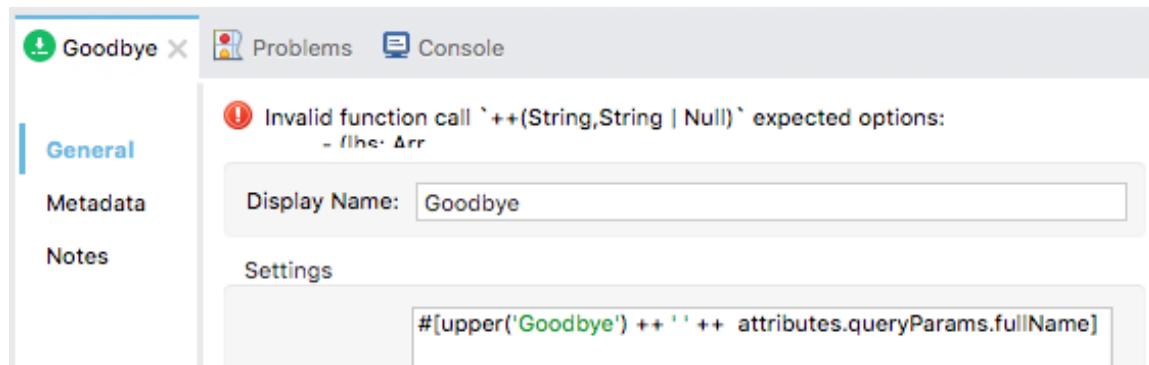
Use an expression that references an attribute when setting the payload

41. Navigate to the properties view for the Goodbye Set Payload in goodbyeFlow.

42. Use the concatenation operator to also display the value of the query parameter separated by a space.

```
##[upper('Goodbye') ++ ' ' ++ attributes.queryParams.fullName]
```

43. Review the error.



Use the as operator to coerce the attribute to a String

44. Use the as operator to also display the value of the query parameter separated by a space.

```
##[upper('Goodbye') ++ ' ' ++ attributes.queryParams.fullName as String]
```

45. Click Apply Changes to redeploy the application.

46. In Advanced REST Client, send the same request; you should now also see the name displayed.

A screenshot of the Advanced REST Client. At the top, it shows a green button labeled '200 Success' and '329.76 ms'. Below this is a toolbar with icons for copy, paste, refresh, and others. The main content area displays the response body: 'GOODBYE Max Mule'.

Use an expression to set a request header

47. Return to the Anypoint Studio and navigate to the properties view for the GET /goodbye HTTP Request in helloFlow.
48. In the Request section, select the Query Parameters tab.

49. Change the value of fullName to the value of the fname query parameter.

Name	Value
"fullname"	attributes.queryParams.fname

50. Click Apply Changes to redeploy the application.

51. In Advanced REST Client, send the same request; you should now see the name of the fname query parameter displayed.

200 Success 312.24 ms



GOODBYE max

Make a request and do not send a query parameter

52. Remove the query parameters and make a request to <http://localhost:8081/hello>; you should get an error.

500 Server Error 164.60 ms

DETAILS ▾



HTTP GET on resource '<http://localhost:8081/goodbye>' failed: internal server error (500).

Set a default value in an expression

53. Return to the Anypoint Studio and navigate to the properties view for the Goodbye Set Payload in goodbyeFlow.

54. Remove as String.

55. Use the default operator to add a default value of Maxine.

```
# [upper('Goodbye') ++ ' ' ++ (attributes.queryParams.fullName default  
'Maxine'))]
```

56. Click Apply Changes to redeploy the application.

57. In Advanced REST Client, send the same request; you should now see the default value Maxine displayed.

200 Success 307.33 ms



GOODBYE Maxine

Use a query parameter in the expression for a response header

58. Return to the Anypoint Studio and navigate to the properties view for the GET /hello HTTP Listener.
59. Select the Responses tab.
60. Change the name header value to the value of the fname query parameter.

attributes.queryParams.fname

Name	Value
"name"	attributes.queryParams.fname

61. Click Apply Changes to redeploy the application.
62. In Advanced REST Client, add a fname and set it equal to max or some other value and send the request.
63. Look at the response headers; you should no longer see a name header.

200 Success 323.58 ms

DETAILS ^

GET http://localhost:8081/hello?fname=max

Response headers 3

Request headers 0

Redirects 0

Timings

content-type: application/java; charset=UTF-8
content-length: 11
date: Thu, 19 Apr 2018 21:46:53 GMT



GOODBYE max

Debug and verify the query parameter is sent with the request

64. Return to Anypoint Studio.
65. Stop and then debug the project.
66. Return to Advanced REST Client and send the same request.
67. Return to the Mule Debugger and look at the attributes and query parameters; you should see the fname query parameter.

```
⑧ method           GET
► ⑨ queryParams    MultiMap{[fname=[max]]}
⑧ queryString     fname=max
⑧ relativePath    /hello
```

68. Step into goodbyeFlow and look at the attributes and query parameters; you should see the fullName query parameter.

```
⑧ method           GET
► ⑨ queryParams    MultiMap{[fullName=[max]]}
⑧ queryString     fullName=max
⑧ relativePath    /goodbye
```

69. Step back to helloFlow and look at the value of the attributes; you should not see any query parameters.

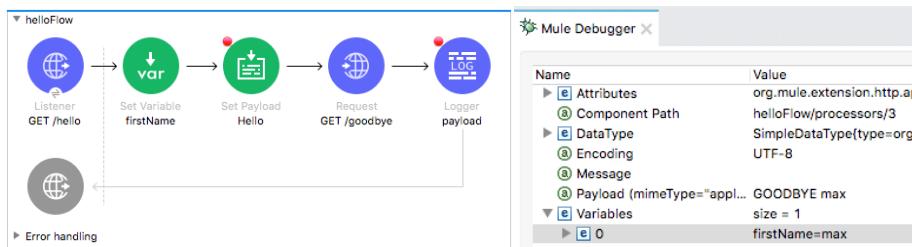
```
▼ ⑨ Attributes      org.mule.extension.http.api.HttpResponse
  ⑧ DOUBLE_TAB
  ► ⑨ headers        MultiMap{[content-type=[application/json]...]}
  ⑧ reasonPhrase
  ⑧ serialVersionUID -3131769059554988414
  ⑧ statusCode       200
  ⑧ TAB
```

70. Step through the rest of the application.
71. Switch to the Mule Design perspective.

Walkthrough 6-6: Set and get variables

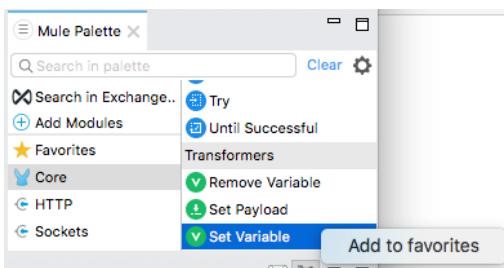
In this walkthrough, you create variables associated with an event. You will:

- Use the Set Variable transformer to create a variable.
- Reference a variable in a DataWeave expression.
- Use a variable to dynamically set a response header.
- Use the Mule Debugger to see the value of a variable.
- Track variables across a transport boundary.



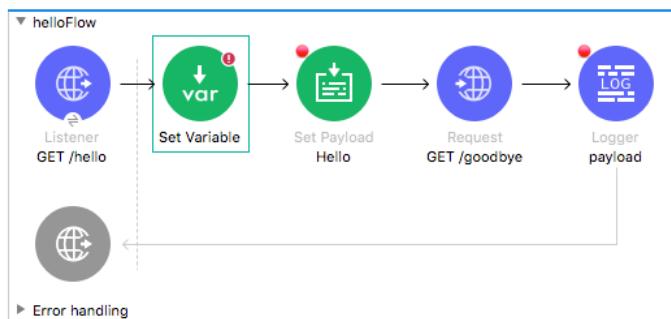
Add the Set Variable transformer to the Favorites section of the Mule Palette

1. Return to apdev-examples.xml.
2. In the Mule Palette, select Core.
3. Locate the Set Variable transformer and right-click it and select Add to favorites.

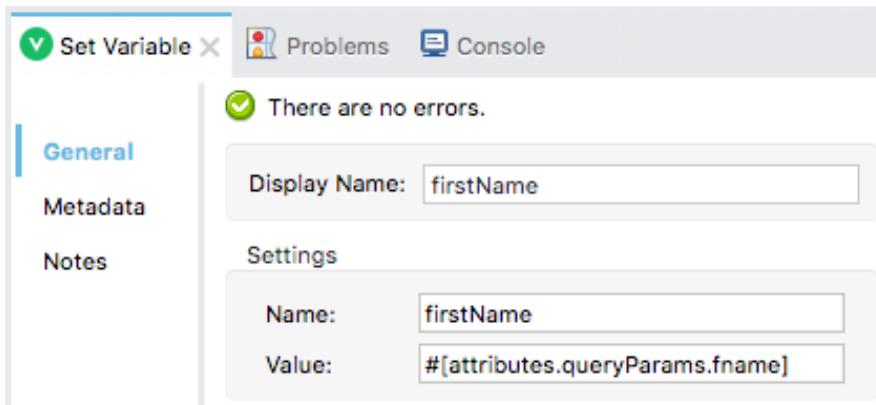


Create a variable

4. Drag the Set Variable transformer from the Favorites section of the Mule Palette and drop it after the GET /hello HTTP Listener in helloFlow.



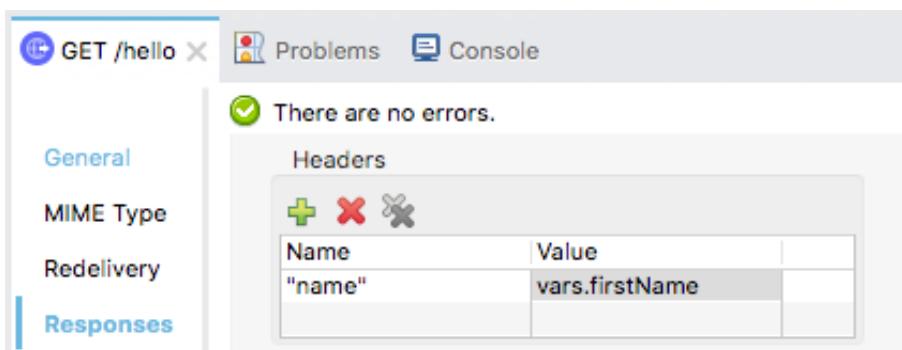
5. In the Set Variable properties view, set the display name and name to firstName.
6. Set the value to your query parameter, fname.



Use an expression that references the variable to set a response header

7. Return to the properties view for the GET /hello HTTP Listener in helloFlow.
8. Select the Responses tab.
9. Change the name header value from attributes.queryParams.fname to the value of the firstName variable.

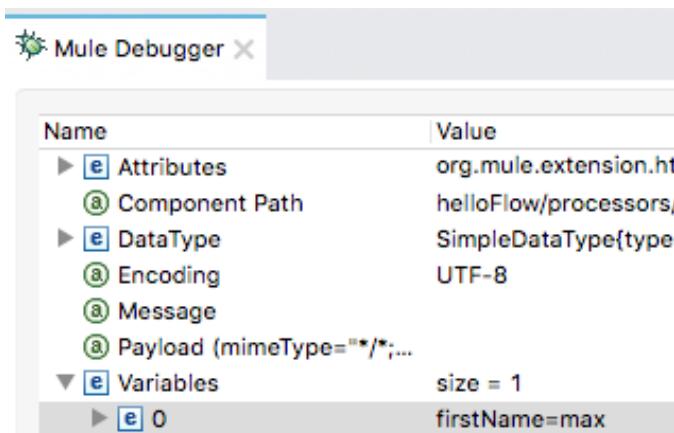
`vars.firstName`



View variables in the Mule Debugger

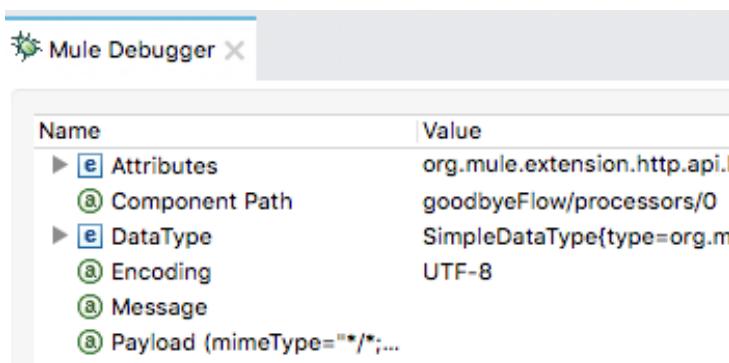
10. Save the file to redeploy the project in debug mode.
11. In Advanced REST Client, send the same request.

12. In the Mule Debugger, locate and expand the new Variables section; you should see your firstName variable.



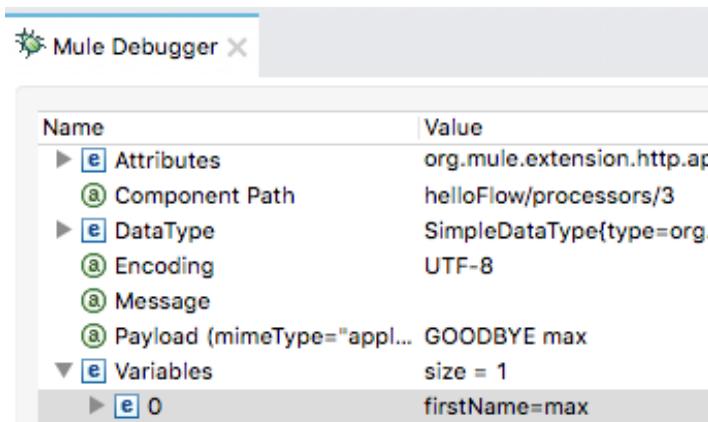
Name	Value
► [e] Attributes	org.mule.extension.http.api...
③ Component Path	helloFlow/processors/0
► [e] DataType	SimpleDataType{type=org.m...
③ Encoding	UTF-8
③ Message	
③ Payload (mimeType="*/*";...	
▼ [e] Variables	size = 1
► [e] 0	firstName=max

13. Step into goodbyeFlow; you should no longer see the Variables section.



Name	Value
► [e] Attributes	org.mule.extension.http.api...
③ Component Path	goodbyeFlow/processors/0
► [e] DataType	SimpleDataType{type=org.m...
③ Encoding	UTF-8
③ Message	
③ Payload (mimeType="*/*";...	

14. Step back to helloFlow; you should see the Variables section again with your firstName variable.



Name	Value
► [e] Attributes	org.mule.extension.http.ap...
③ Component Path	helloFlow/processors/3
► [e] DataType	SimpleDataType{type=org.m...
③ Encoding	UTF-8
③ Message	
③ Payload (mimeType="appl... GOODBYE max	
▼ [e] Variables	size = 1
► [e] 0	firstName=max

15. Step through the rest of the application.

16. Return to Advanced REST Client; you should see the header with the value of the query parameter.

200 Success 30679.11 ms DETAILS ^

GET http://localhost:8081/hello?fname=max

Response headers 4 Request headers 0 Redirects 0 Timings

```
name: max
content-type: application/java; charset=UTF-8
content-length: 11
date: Thu, 19 Apr 2018 22:09:04 GMT
```

Copy ↗ ↘ ↻ ↪

GOODBYE max

17. Change the value of the query parameter and send another request.
18. In the Mule Debugger, click the Resume button until you step through the application.
19. Return to Advanced REST client; you should see the new query parameter value returned in both the body and the header.

Method Request URL
GET ▾ http://localhost:8081/hello?fname=Maxwell ▾ SEND ⋮

Parameters ▾

200 Success 7967.76 ms DETAILS ^

GET http://localhost:8081/hello?fname=Maxwell

Response headers 4 Request headers 0 Redirects 0 Timings

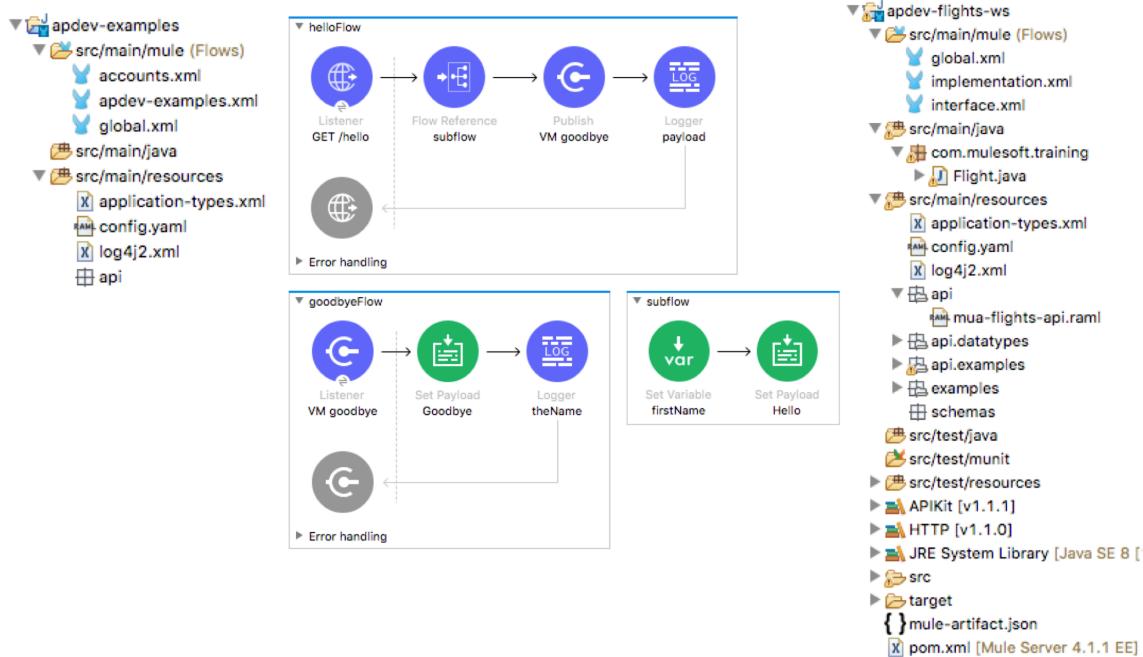
```
name: Maxwell
content-type: application/java; charset=UTF-8
content-length: 15
date: Thu, 19 Apr 2018 22:10:04 GMT
```

Copy ↗ ↘ ↻ ↪

GOODBYE Maxwell

20. Return to Anypoint Studio and switch perspectives.

Module 7: Structuring Mule Applications



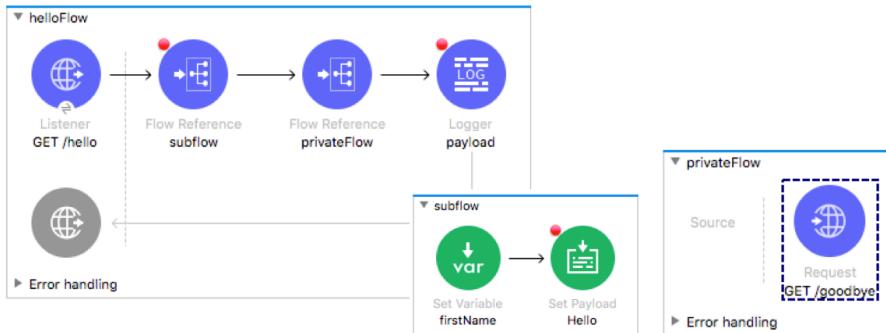
At the end of this module, you should be able to:

- Create applications composed of multiple flows and subflows.
- Pass messages between flows using asynchronous queues.
- Encapsulate global elements in separate configuration files.
- Specify application properties in a separate properties file and use them in the application.
- Describe the purpose of each file and folder in a Mule project.
- Define and manage application metadata.

Walkthrough 7-1: Create and reference subflows and private flows

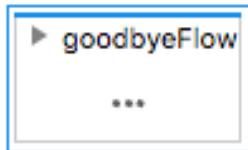
In this walkthrough, you continue to work with apdev-examples.xml. You will:

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



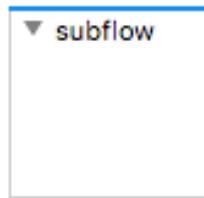
Collapse a flow

1. Return to `apdev-examples.xml` in Anypoint Studio.
2. Click the arrow to the left of the `goodbyeFlow` to collapse it.

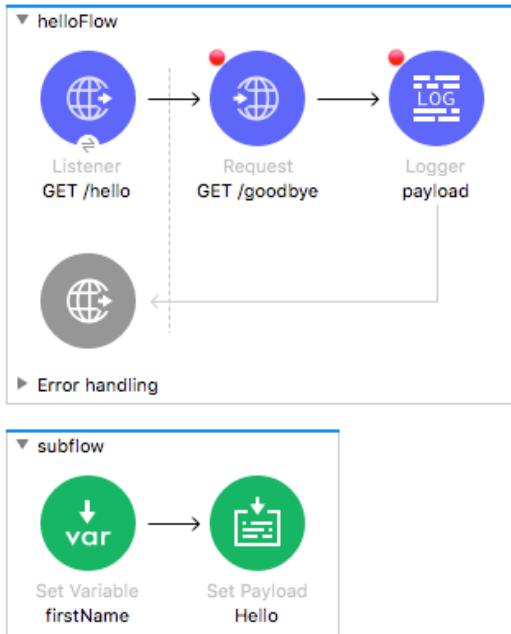


Create a subflow

3. In the Mule Palette, select Core.
4. Drag a Sub Flow scope from the Mule Palette and drop it between the existing flows in the canvas.
5. Change the name of the flow to subflow.

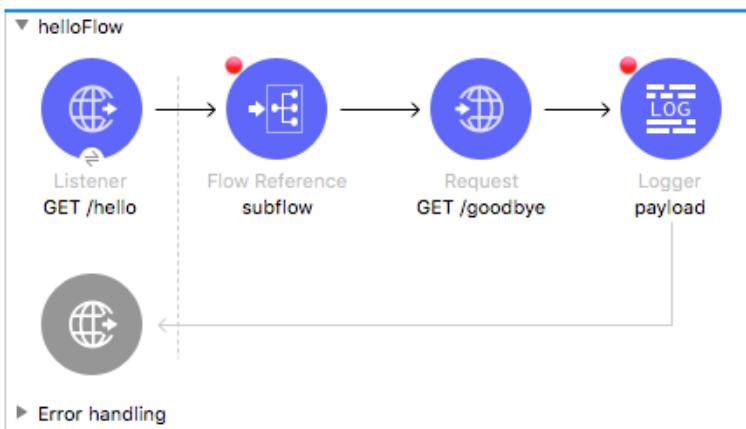


- Select the Set Variable and Set Payload transformers in helloFlow and drag them into the subflow.



Reference a subflow

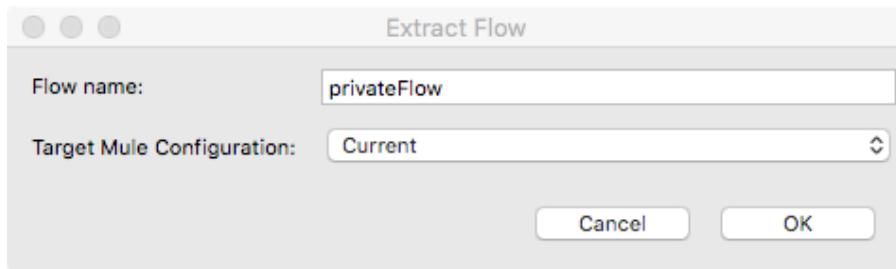
- Drag a Flow Reference component from the Mule Palette and drop it into helloFlow between the GET /hello HTTP Listener and the GET /goodbye HTTP Request.
- In the Flow Reference properties view, set the flow name to subflow.
- Add a breakpoint to the subflow Flow Reference.
- Remove the breakpoint from the GET /goodbye HTTP Request.



Extract processors into a subflow

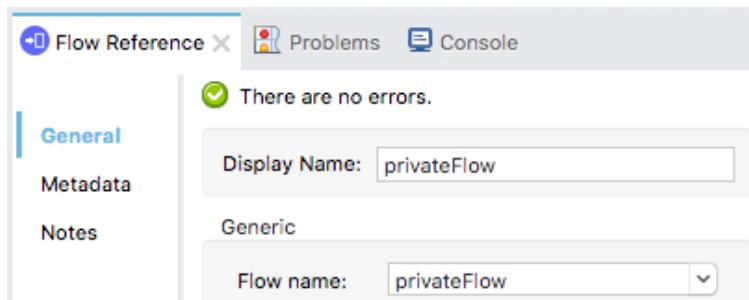
- Right-click the **GET /goodbye** HTTP Request and select Extract to > Flow.

12. In the Extract Flow dialog box, set the flow name to privateFlow.

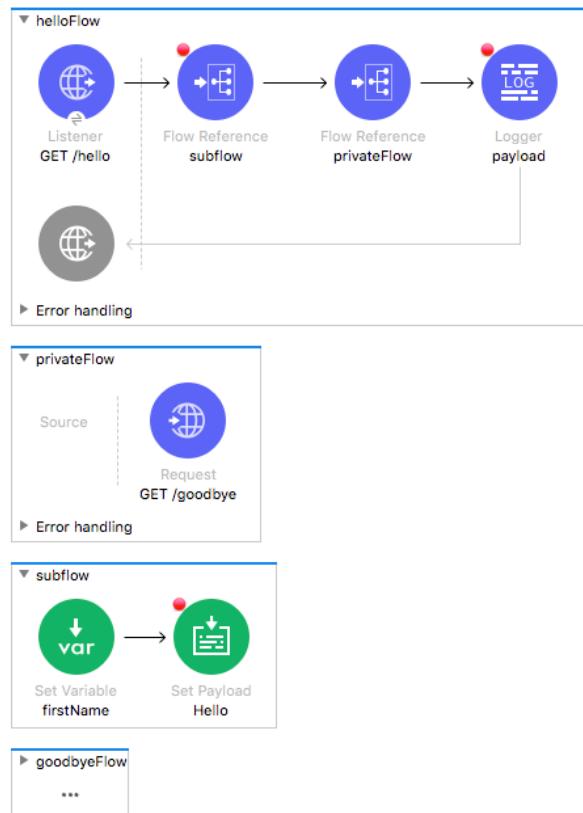


13. Leave the target Mule configuration set to current and click OK.

14. Look at the new Flow Reference properties view; the flow name should already be set to privateFlow – and the display name will be set once you navigate here.

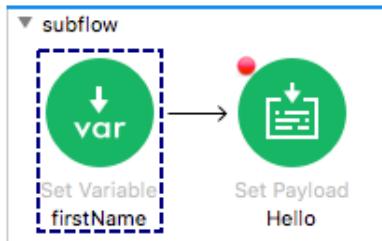


15. Drag privateFlow above subflow in the canvas.



Debug the application

16. Save the file to redeploy the application in debug mode.
17. In Advanced REST Client, send the same request to
<http://localhost:8081/hello?fname=Maxwell>.
18. In the Mule Debugger, step through the application, watching as you step into and out of the flows and subflows.



19. Step through the rest of the application.
20. In Advanced REST Client, send the same request again.
21. In the Mule Debugger, step through the application again, this time watching the values of the attributes, payload, and variables in each of the flows.

The screenshot shows the Mule Debugger interface with the title bar 'Mule Debugger'. Below it is a table with columns 'Name' and 'Value'. The table contains the following data:

Name	Value
Attributes	org.mule.extension.http.ap...
Component Path	privateFlow/processors/0
DataType	SimpleDataType{type=jav...
Message	
Payload (mimeType="*/*")	Hello
Variables	size = 1
0	firstName=Maxwell

Below the table, there is a text input field containing the value 'firstName=Maxwell'.

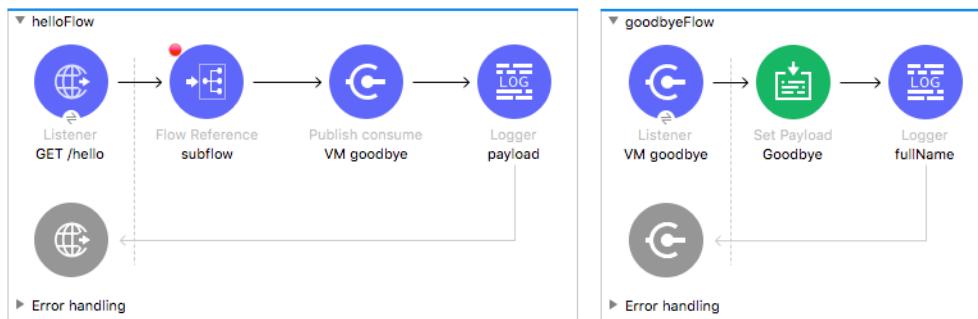
At the bottom of the screen, there is another window titled 'apdev-examples' showing a Mule flow. The flow is named 'privateFlow' and contains a 'Request' component with the URL 'GET /goodbye'.

22. Step through the rest of the application.
23. Switch to the Mule Design perspective.

Walkthrough 7-2: Pass messages between flows using the VM connector

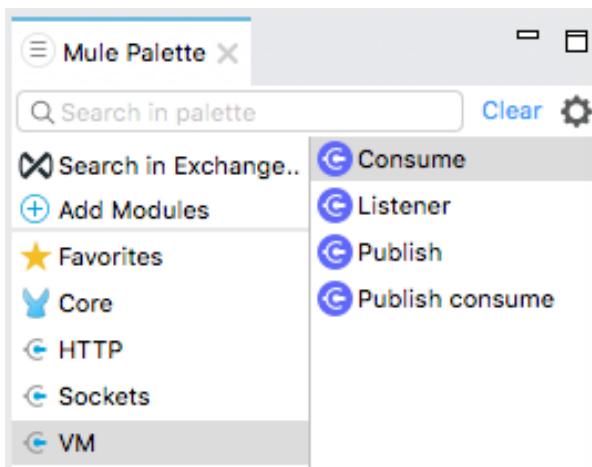
In this walkthrough, you pass messages between flows using asynchronous queues. You will:

- Pass messages between flows using the VM connector.
- Explore variable persistence with VM communication.
- Publish content to a VM queue and then wait for a response.
- Publish content to a VM queue without waiting for a response.

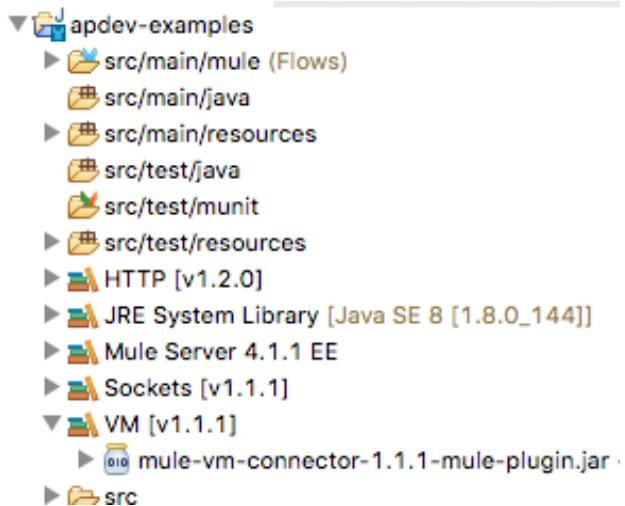


Add the VM module to the project

1. Return to apdev-examples.xml.
2. In the Mule Palette, select Add Modules.
3. Select the VM connector in the right side of the Mule Palette and drag and drop it into the left side.

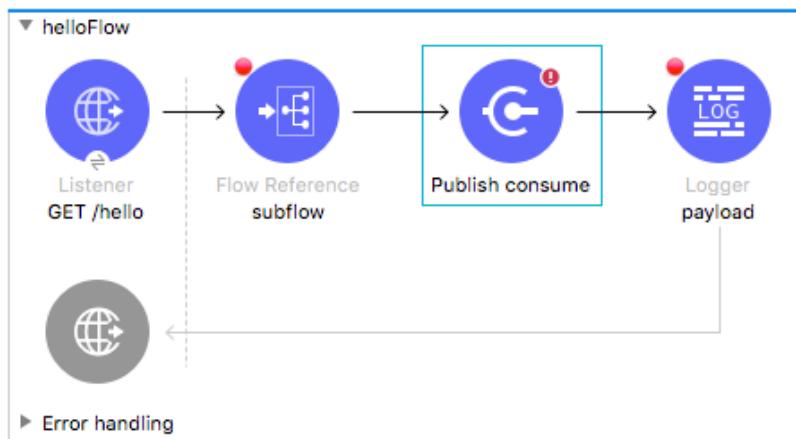


4. In the Project Explorer, locate the JAR file for the VM connector.

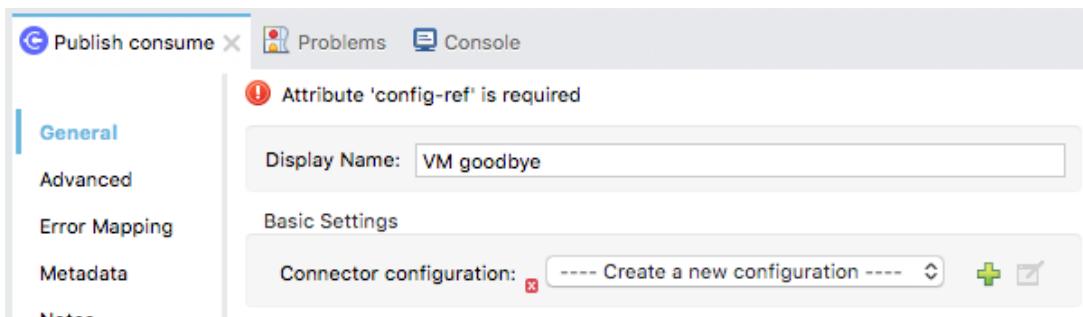


Add a VM Publish Consume operation

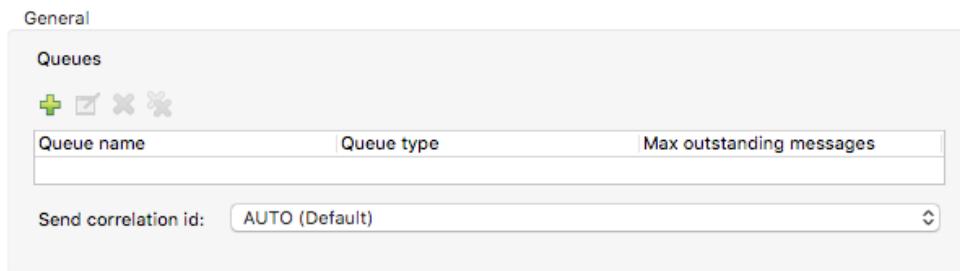
5. In helloFlow, delete the privateFlow Flow Reference.
6. Select VM in the Mule Palette.
7. Select the Publish consume operation and drag and drop it before the Logger in helloFlow.



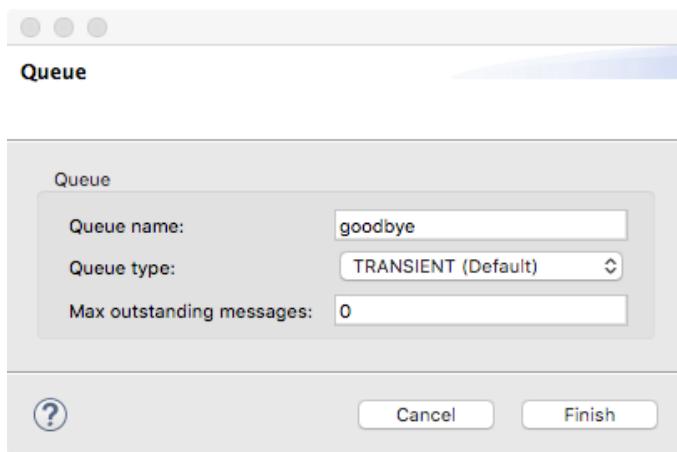
8. In the Publish consume properties view, change the display name to VM goodbye.



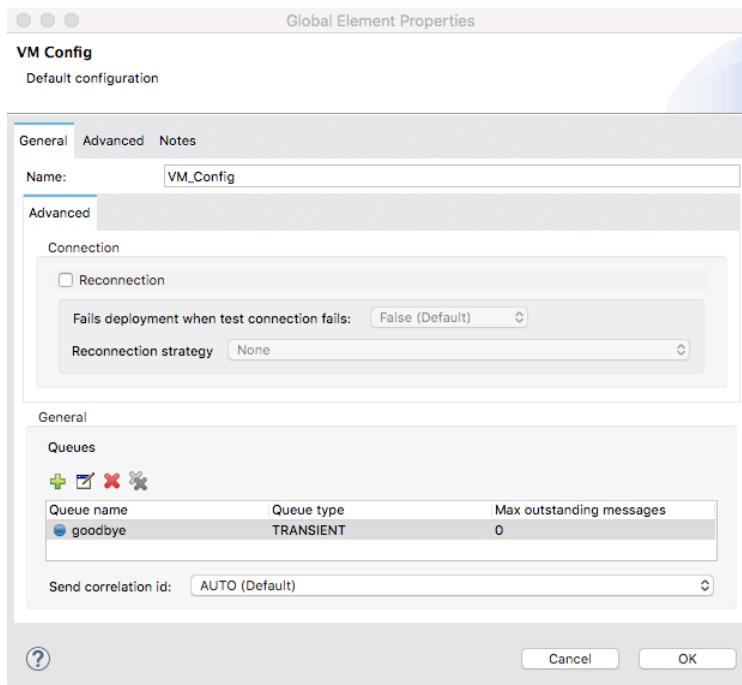
9. Click the Add button next to connector configuration.
10. In the Global Element Properties dialog box, click the Add Queue button.



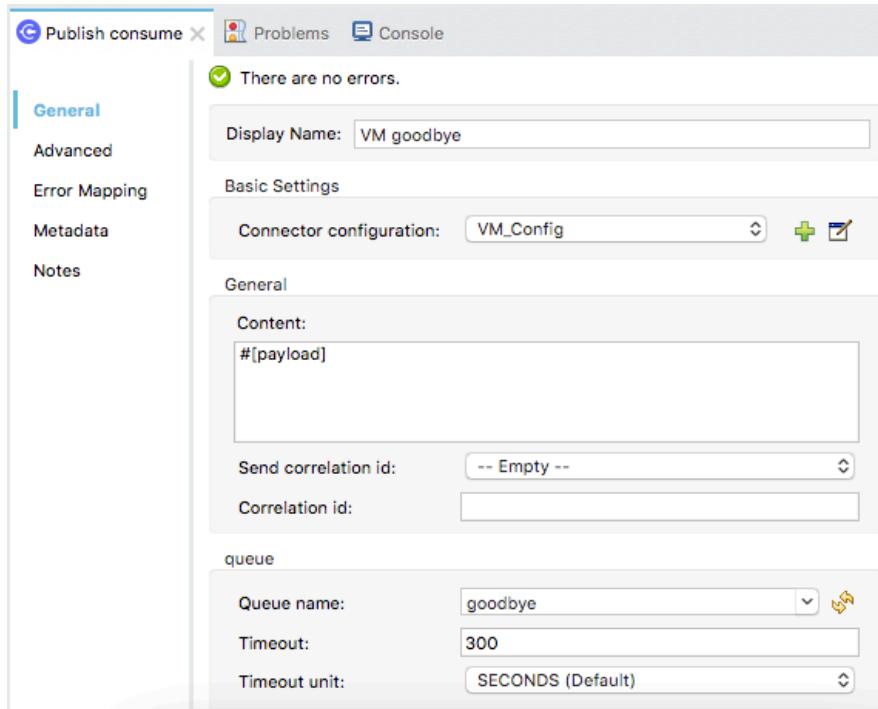
11. In the Queue dialog box, set the queue name to goodbye and click Finish.



12. In the Global Element Properties dialog box, click OK.

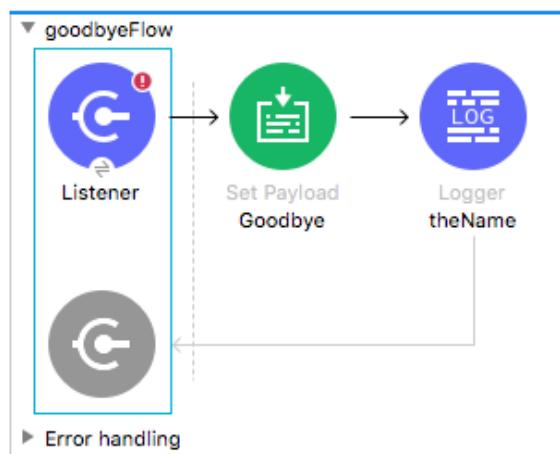


13. In the VM goodbye Publish consume properties view, set the queue name to goodbye.
14. Set the timeout to 300 seconds for debugging purposes.



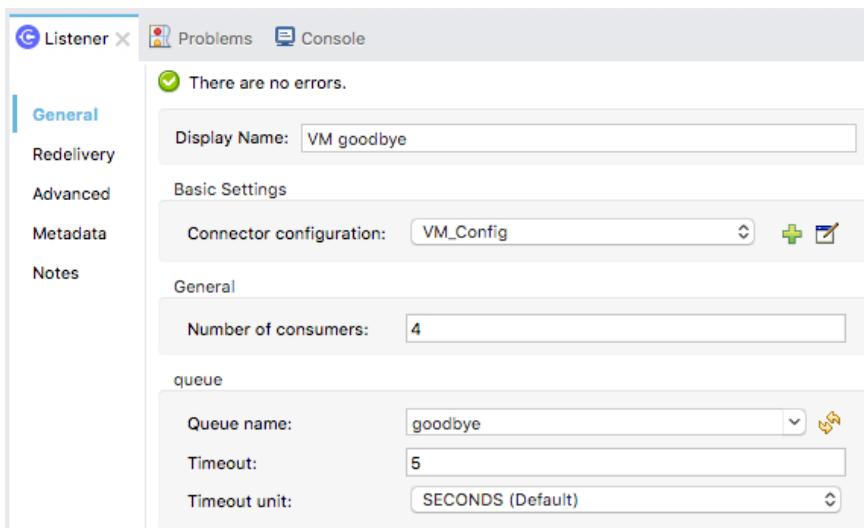
Add a VM Listener

15. Expand goodbyeFlow and delete its HTTP Listener.
16. Locate the Listener operation for the VM connector in the right side of the Mule Palette and drag and drop it in the source section of goodbyeFlow.



17. In the VM Listener properties view, change the display name to VM goodbye.
18. Set the connector configuration to the existing VM_Config

19. Set the queue name to goodbye.



Debug the application

20. Save the file to redeploy the project in debug mode; you should get an error that a dependency is missing.

```
apdev-examples [Mule Applications] Mule Server 4.1.1 EE
+ Failed to deploy artifact 'apdev-examples', see below  +
=====
org.mule.runtime.deployment.model.api.DeploymentException: Failed to deploy artifact [apdev-examples]
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: org.mule.runtime.deployment.model.api.DeploymentInitException: MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.deployment.model.api.DeploymentInitException: MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.core.api.config.ConfigurationException: Error loading: apdev-examples.xml, Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: Error loading: apdev-examples.xml, Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
```

21. Stop the project.

22. Debug the project; the application should successfully deploy.

23. In Advanced REST Client, send the same request.

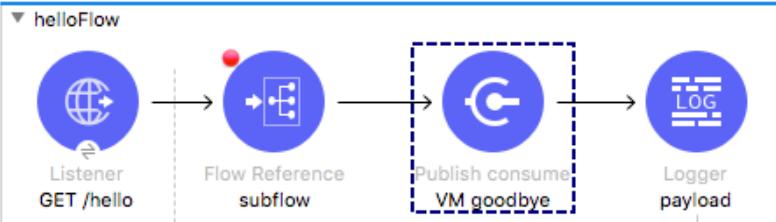
24. In the Mule Debugger, step through the application to the VM Publish consume.

25. Look at the payload, attributes, and variables.

Mule Debugger

Name	Value
► [e] Attributes	org.mule.extension.http.api.HttpRequestAttributes
⑧ Component Path	helloFlow/processors/1
► [e] DataType	SimpleDataType{type=java.lang.String, mimeType="*/*"}
⑧ Message	
⑧ Payload (mimeType="*/*")	Hello
▼ [e] Variables	size = 1
► [e] 0	firstName=Maxwell

apdev-examples



```
graph LR; Listener[Listener GET /hello] --> FlowRef[Flow Reference subflow]; FlowRef --> Publish[Publish consume VM goodbye]; Publish --> Logger[Logger payload]
```

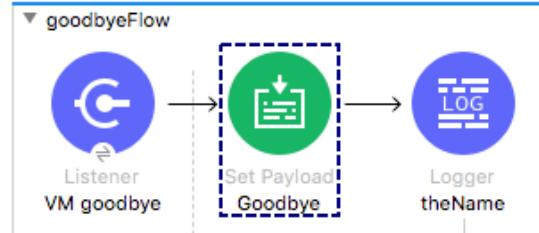
26. Step into goodbyeFlow.

27. Look at the payload, attributes, and variables (or lack thereof).

Mule Debugger

Name	Value
▼ [e] Attributes	org.mule.extensions.vm.api.VMM
⑧ correlationId	0-dad4f120-4424-11e8-9298
⑧ queueName	goodbye
⑧ serialVersionUID	3923721884012142263
► [e] timestamp	2018-04-19T15:57:41.030
⑧ Component Path	goodbyeFlow/processors/0
► [e] DataType	SimpleDataType{type=java.lang.
⑧ Message	
⑧ Payload (mimeType="... Hello	

apdev-examples



```
graph LR; Listener[Listener VM goodbye] --> SetPayload[Set Payload Goodbye]; SetPayload --> Logger[Logger theName]
```

28. Step through the flow until the event returns to helloFlow.

29. Look at the payload, attributes, and variables.

The screenshot shows two perspectives side-by-side. The top perspective is 'Mule Debugger' showing a table of payload attributes and variables. The bottom perspective is 'apdev-examples' showing the 'helloFlow' Mule flow diagram.

Mule Debugger:

Name	Value
Attributes	org.mule.extensions.vm.api.VMMessageAttribute
Component Path	helloFlow/processors/2
(DataType)	SimpleDataType{type=java.lang.String, mimeType=application/json, encoding=UTF-8}
Encoding	UTF-8
Message	
Payload (mimeType="application/json")	GOODBYE Maxine
Variables	size = 1
0	firstName=Maxwell

apdev-examples:

helloFlow:

```
graph LR; Listener[Listener GET /hello] --> Subflow((Flow Reference subflow)); Subflow --> Publish[Publish consume VM goodbye]; Publish --> Logger[Logger payload]
```

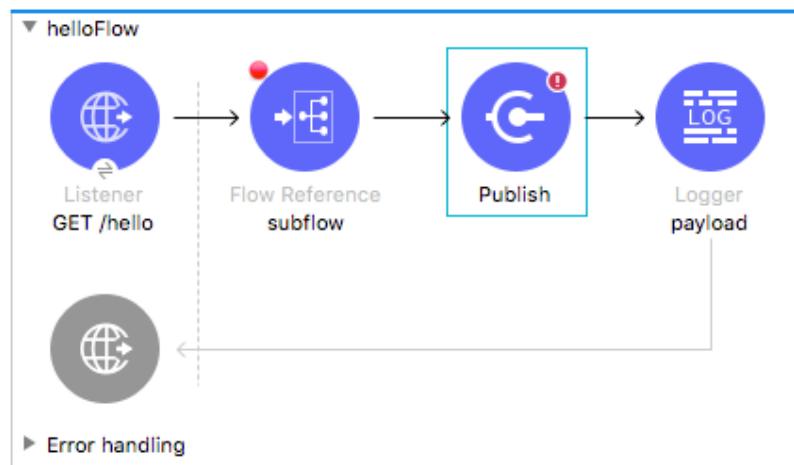
30. Step through the rest of the application.

31. Switch perspectives.

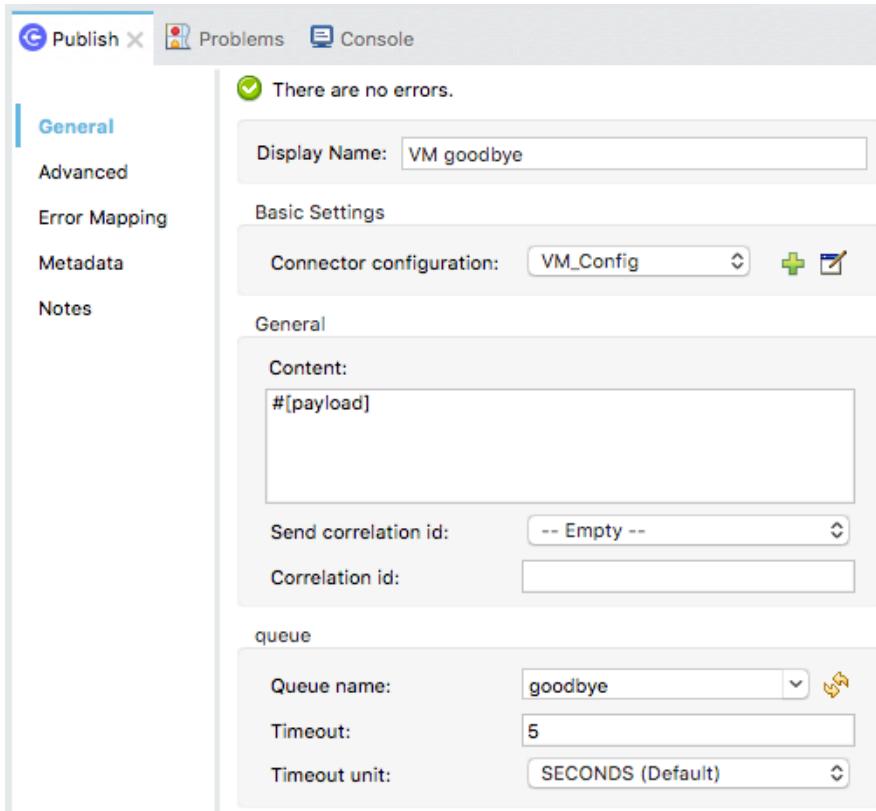
Change the VM Publish Consume operation to Publish

32. Delete the VM Publish consume operation in helloFlow.

33. Drag a VM Publish operation from the Mule Palette and drop it before the Logger.



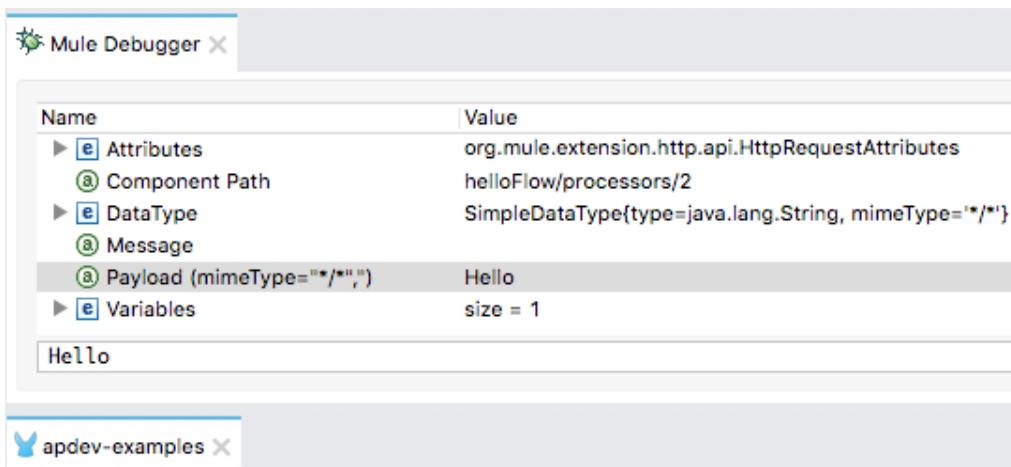
34. In the Publish properties view, set the display name to VM goodbye.
35. Set the connector configuration to the existing VM_Config.
36. Set the queue name to goodbye.



Debug the application

37. Save the file to redeploy the project in debug mode.
38. In Advanced REST Client, send the same request.
39. In the Mule Debugger, step through the application to the VM Publish operation.
40. Step again; you should step to the Logger in helloFlow.

41. Look at the value of the payload.



The screenshot shows the Mule Debugger interface with two tabs: "Mule Debugger" and "apdev-examples". The "Mule Debugger" tab displays a table of variables:

Name	Value
Attributes	org.mule.extension.http.api.HttpRequestAttributes
Component Path	helloFlow/processors/2
DataType	SimpleDataType{type=java.lang.String, mimeType='*/*'}
Message	
Payload (mimeType="*/*")	Hello
Variables	size = 1

A text input field below the table contains the value "Hello".



The screenshot shows the Anypoint Studio interface with a flow diagram titled "apdev-examples". The flow consists of four components connected sequentially:

- Listener (GET /hello)
- Flow Reference (subflow)
- Publish (VM goodbye)
- Logger (payload)

The "Logger (payload)" component is highlighted with a dashed blue border.

42. Step again; you should see execution stopped in goodbyeFlow.

43. Step to the end of the application.

44. Return to Advanced REST Client; you should see a response of Hello – not GOODBYE.

200 Success 160792.30 ms



Hello

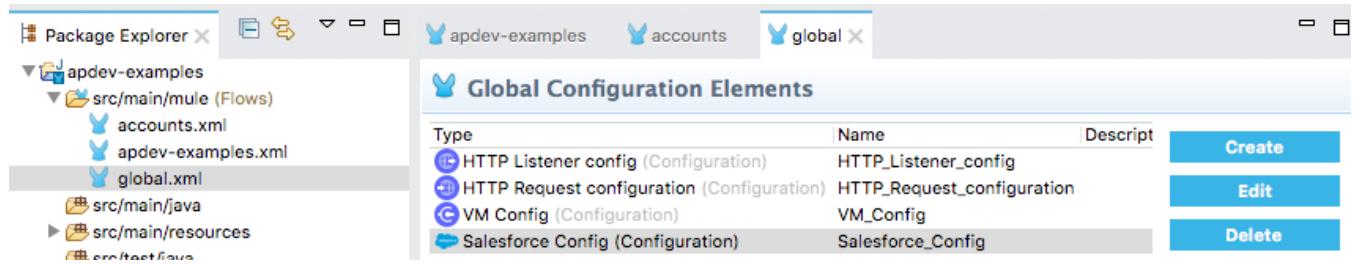
45. Return to Anypoint Studio and switch perspectives.

46. Stop the project.

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

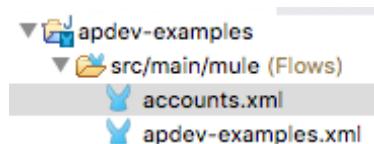
In this walkthrough, you refactor your apdev-examples project. You will:

- Create a new configuration file with an endpoint that uses an existing global element.
- Create a configuration file global.xml for just global elements.
- Move the existing global elements to global.xml.
- Create a new global element in global.xml and configure a new connector to use it.

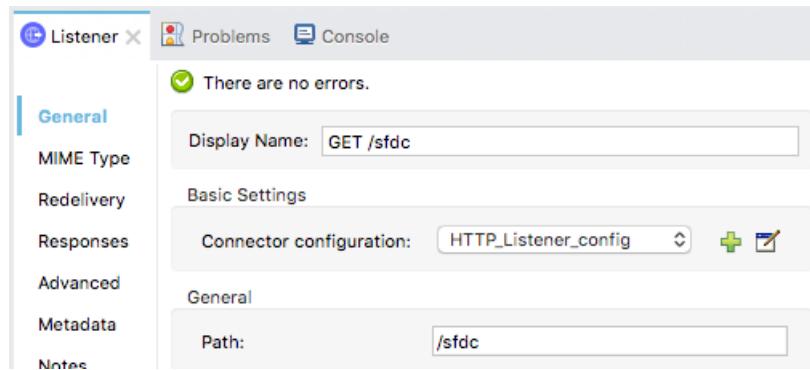


Create a new configuration file

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



4. Drag an HTTP Listener to the canvas from the Mule Palette.
5. In the Listener properties view, set the display name to GET /sfdc.
6. Set the connector configuration to the existing HTTP_Listener_config.
7. Set the path to /sfdc and the allowed methods to GET.



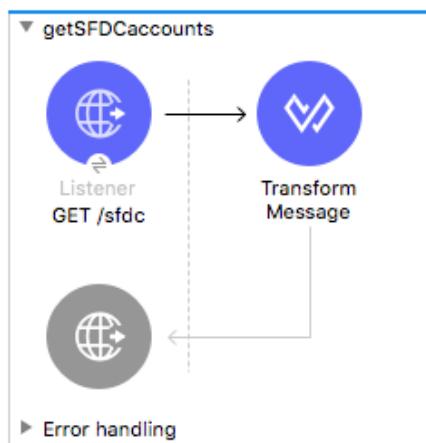
8. Add a Transform Message component to the flow.
9. In the Transform Message properties view, change the output type to application/json and set the expression to payload.

```

1 %dw 2.0
2   output application/json
3   ---
4   payload

```

10. Change the name of the flow to getSFDCCaccounts.



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

The screenshot shows the 'Global Configuration Elements' view in the Mule Studio interface. The top navigation bar has tabs for 'apdev-examples' and 'accounts'. The main area displays a table with columns: Type, Name, Description, Create, Edit, and Delete. The table is currently empty.

Create a global configuration file

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

The screenshot shows the file structure in the Mule Studio interface. The 'apdev-examples' project contains a 'src/main/mule' folder, which in turn contains a 'Flows' folder. Inside 'Flows', there are three files: 'accounts.xml', 'apdev-examples.xml', and 'global.xml', with 'global.xml' being highlighted.

13. In global.xml, switch to the Configuration XML view.

14. Place some empty lines between the start and end mule tags.

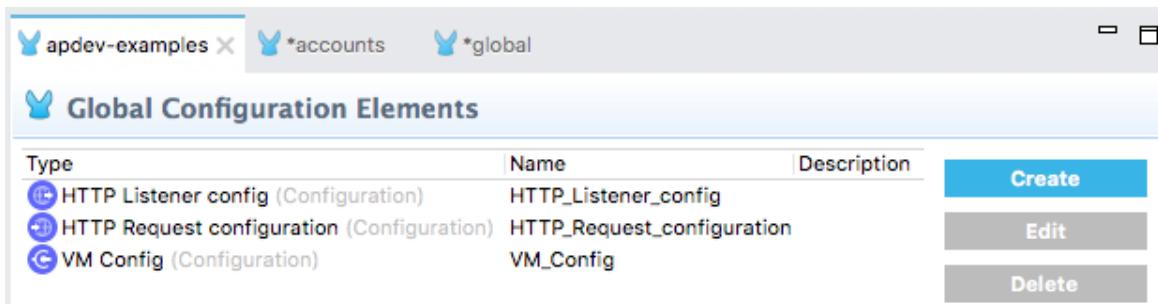


```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core-3.6.xsd">
</mule>
```

Move the existing global elements to the new global configuration file

15. Return to apdev-examples.xml.

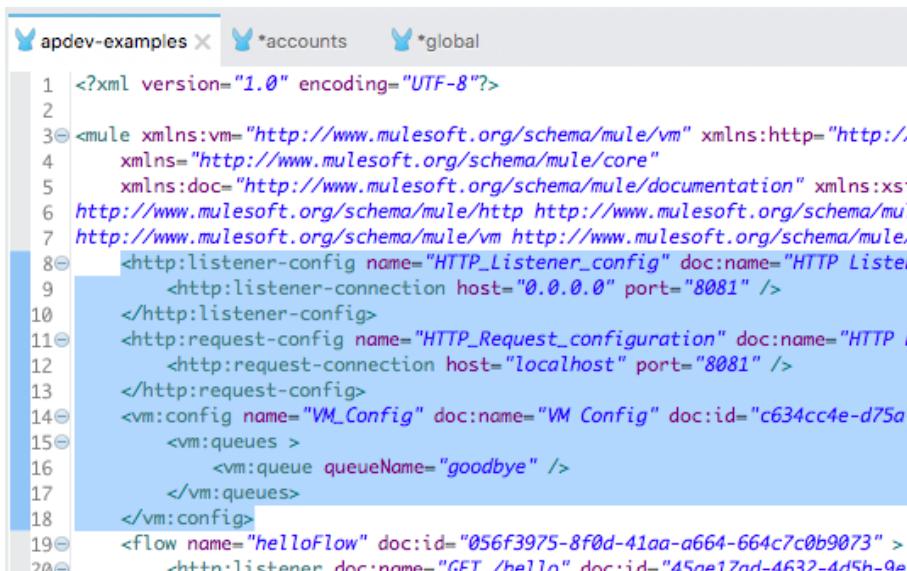
16. Switch to the Global Elements view and see there are three configurations.



Type	Name	Description	Create
HTTP Listener config (Configuration)	HTTP_Listener_config		
HTTP Request configuration (Configuration)	HTTP_Request_configuration		
VM Config (Configuration)	VM_Config		

17. Switch to the Configuration XML view.

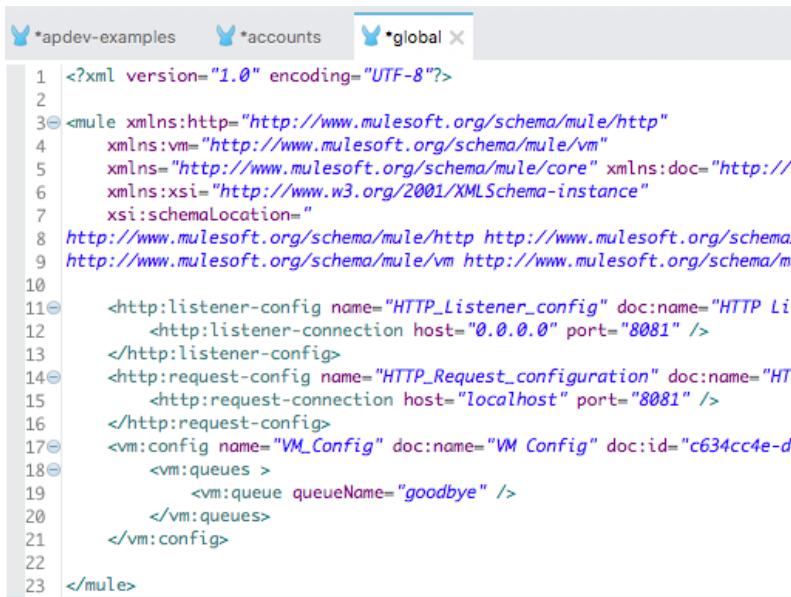
18. Select and cut the three configuration elements defined before the flows.



```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:vm="http://www.mulesoft.org/schema/mule/vm" xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:doc="http://www.mulesoft.org/schema/mule/documentation" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http-3.6.xsd
      http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm-3.6.xsd
      http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core-3.6.xsd">
    <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener config">
        <http:listener-connection host="0.0.0.0" port="8081" />
    </http:listener-config>
    <http:request-config name="HTTP_Request_configuration" doc:name="HTTP Request configuration">
        <http:request-connection host="localhost" port="8081" />
    </http:request-config>
    <vm:config name="VM_Config" doc:name="VM Config" doc:id="c634cc4e-d75a-43d0-8f0d-41aa-a664-664c7c0b9073">
        <vm:queues>
            <vm:queue queueName="goodbye" />
        </vm:queues>
    </vm:config>
    <flow name="helloFlow" doc:id="056f3975-8f0d-41aa-a664-664c7c0b9073">
        <http:listener doc:name="GET /hello" doc:id="45ae17ad-4632-4d5h-0a"/>
```

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

19. Return to the Message Flow view.
20. Return to global.xml.
21. Paste the global elements you cut to the clipboard between the start and end mule tags.

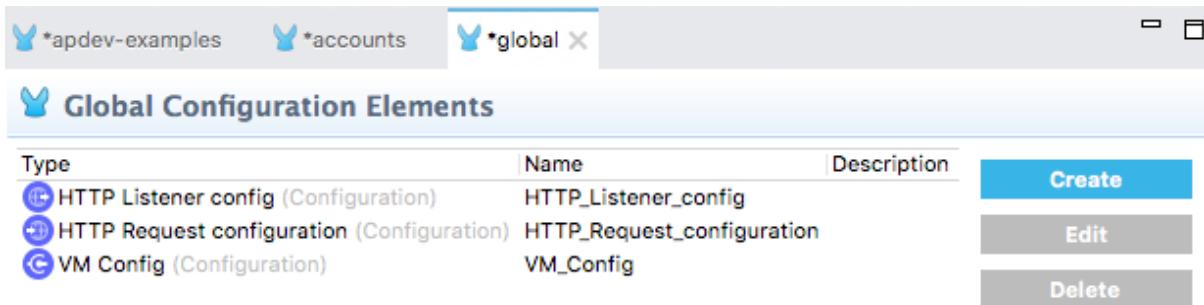


```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3<mule xmlns:http="http://www.mulesoft.org/schema/mule/http"
4      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
5      xmlns="http://www.mulesoft.org/schema/mule/core" doc:name="HTTP Listener Configuration"
6      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7      xsi:schemaLocation="
8         http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http.xsd
9         http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm.xsd">
10
11<http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener Configuration">
12    <http:listener-connection host="0.0.0.0" port="8081" />
13</http:listener-config>
14<http:request-config name="HTTP_Request_configuration" doc:name="HTTP Request Configuration">
15    <http:request-connection host="localhost" port="8081" />
16</http:request-config>
17<vm:config name="VM_Config" doc:name="VM Config" doc:id="c634cc4e-d1d1-43f2-830a-1a2a2a2a2a2a">
18    <vm:queues>
19        <vm:queue queueName="goodbye" />
20    </vm:queues>
21</vm:config>
22
23</mule>

```

22. Switch to the Global Elements view; you should see the three configurations.

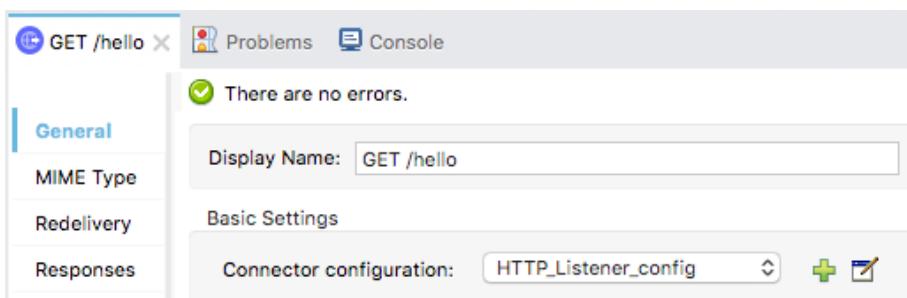


Type	Name	Description
HTTP Listener config (Configuration)	HTTP_Listener_config	
HTTP Request configuration (Configuration)	HTTP_Request_configuration	
VM Config (Configuration)	VM_Config	

Create **Edit** **Delete**

Test the application

23. Return to apdev-examples.xml.
24. Select the GET /hello HTTP Listener in helloFlow; the connector configuration should still be set to HTTP_Listener_config, which is now defined in global.xml.



General

MIME Type

Redelivery

Responses

Display Name: GET /hello

Basic Settings

Connector configuration: HTTP_Listener_config

25. Run the project.

26. In Advanced REST Client, send the same request; you should still get a response of Hello.

200 Success 1583.77 ms



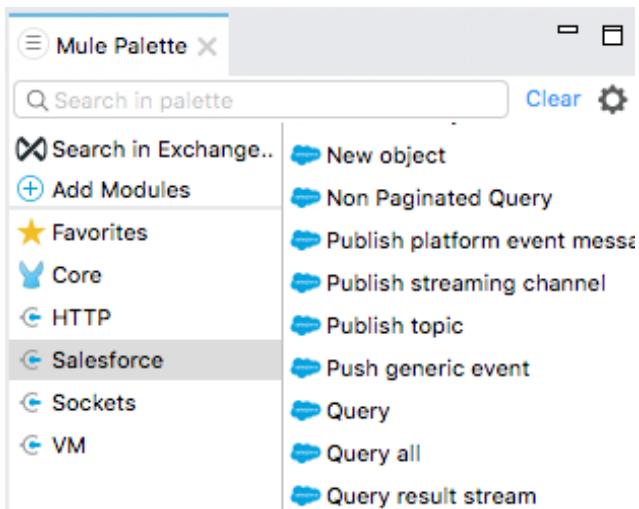
Hello

Create a new global element for the Salesforce component in global.xml

27. Return to apdev-examples.xml.

28. In the Mule Palette, select Add Modules.

29. Select the Salesforce connector in the right side of the Mule Palette and drag and drop it into the left side.

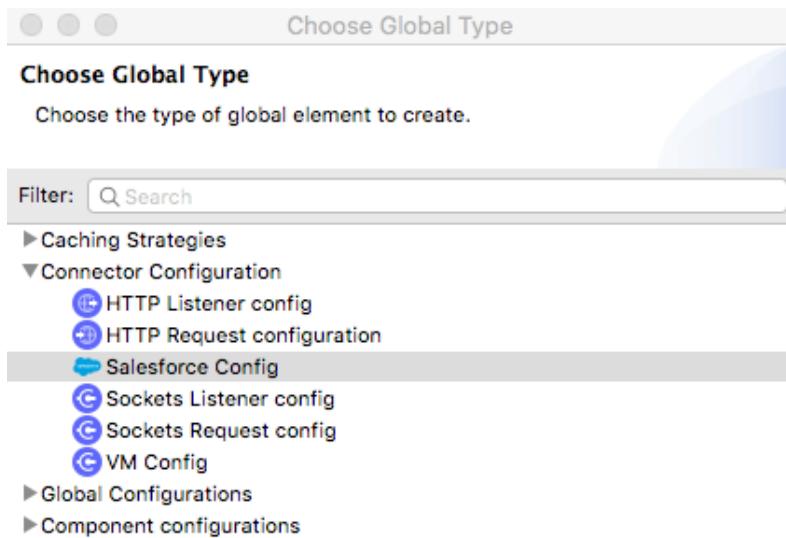


30. Locate the new Salesforce JAR in the project.

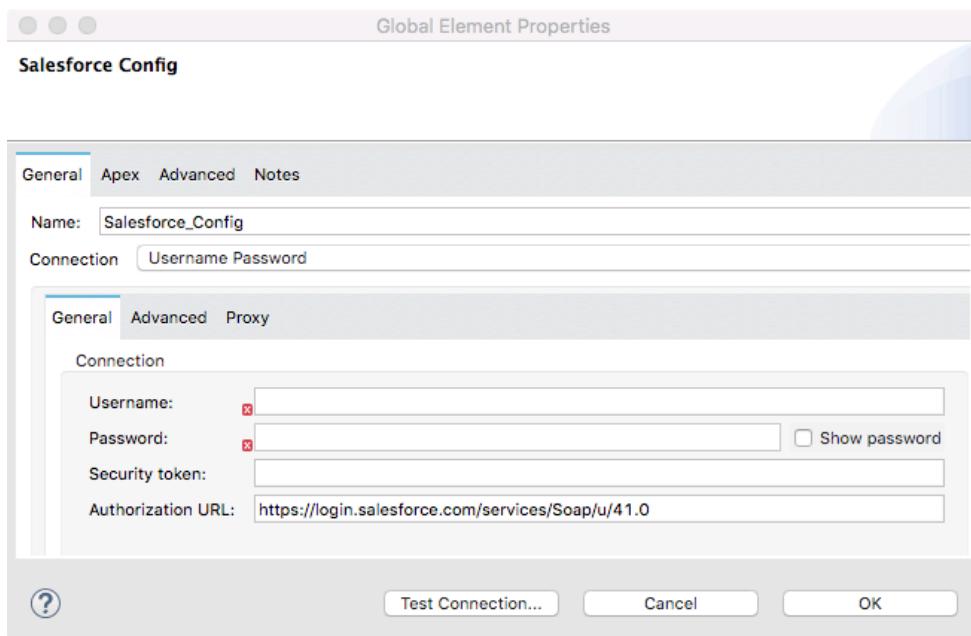
31. Return to global.xml.

32. Click Create.

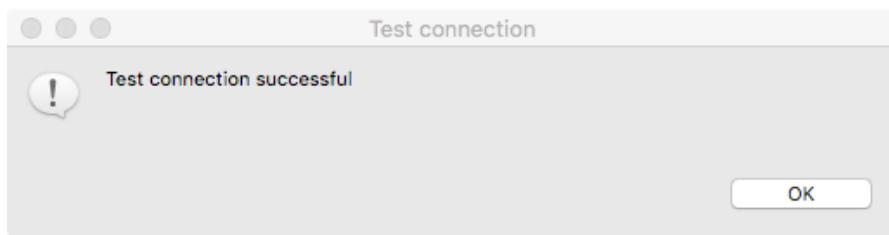
33. In the Choose Global Type dialog box, select Connector Configuration > Salesforce Config and click OK.



34. In the Global Element Properties dialog box enter your Salesforce username, password, and security token.



35. Click Test Connection; your connection should be successful.



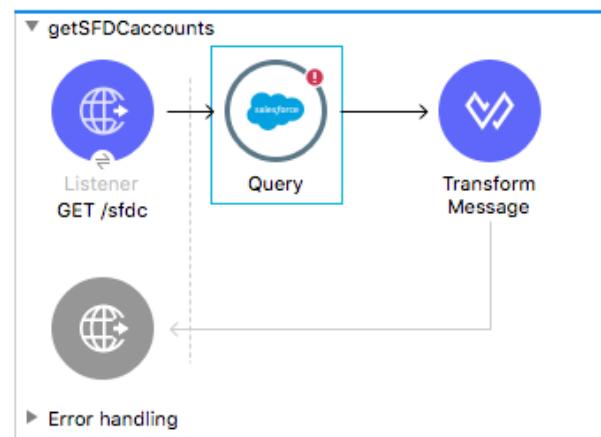
36. In the Test connection dialog box, click OK.
37. In the Global Element Properties dialog box, click OK; you should now see the new configuration in global.

Type	Name	Description	
HTTP Listener config (Configuration)	HTTP_Listener_config		Create
HTTP Request configuration (Configuration)	HTTP_Request_configuration		Edit
VM Config (Configuration)	VM_Config		Delete
Salesforce Config (Configuration)	Salesforce_Config		

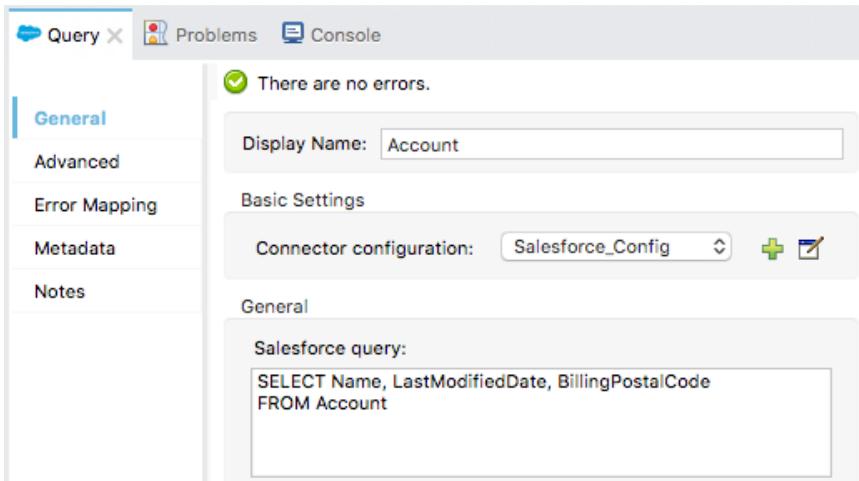
Add a new Salesforce operation that uses the global configuration

38. Return to the Message Flow view in accounts.xml.
39. In the Mule Palette, select Salesforce.

40. Locate the Query operation in the right side of the Mule Palette and drag and drop it before the Logger in getSFDCaccounts.

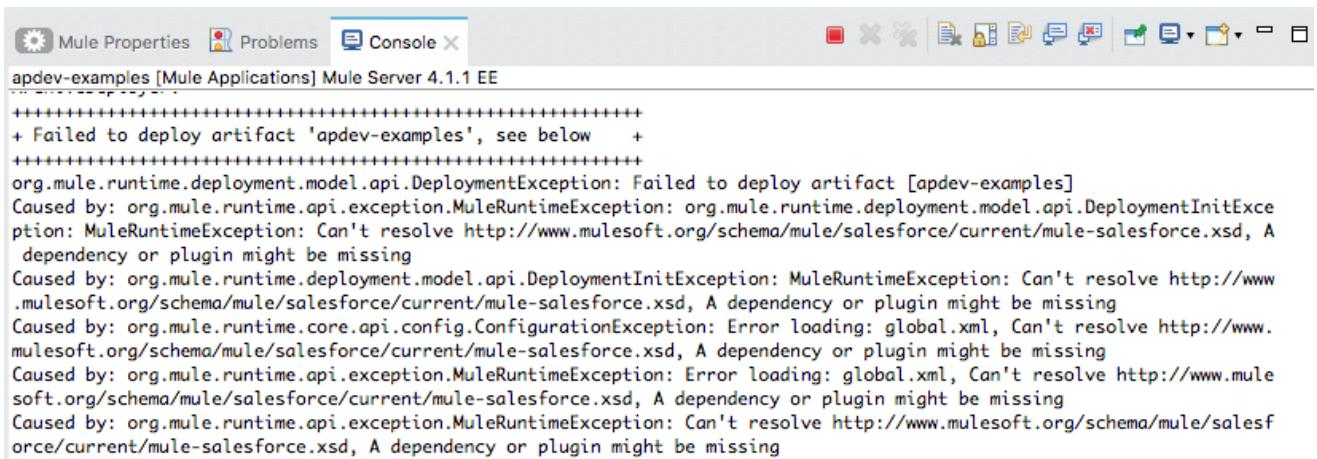


41. In the Query properties view, set the display name to Account.
42. Set the connector configuration to the existing Salesforce_Config.
43. Return to the course snippets.txt file and copy the Salesforce query.
44. Return to Anypoint Studio and paste the query in the Salesforce query section of the Query properties view.



Test the application

45. Save all the files; you should get a missing dependency error in the console.



46. Stop the project.
47. Run the project.

48. In Advanced REST Client, send a request to <http://localhost:8081/sfdc>; you should get a list of the accounts in your Salesforce account.

Method Request URL
GET ▾ <http://localhost:8081/sfdc> ▾

SEND ⋮

Parameters ▾

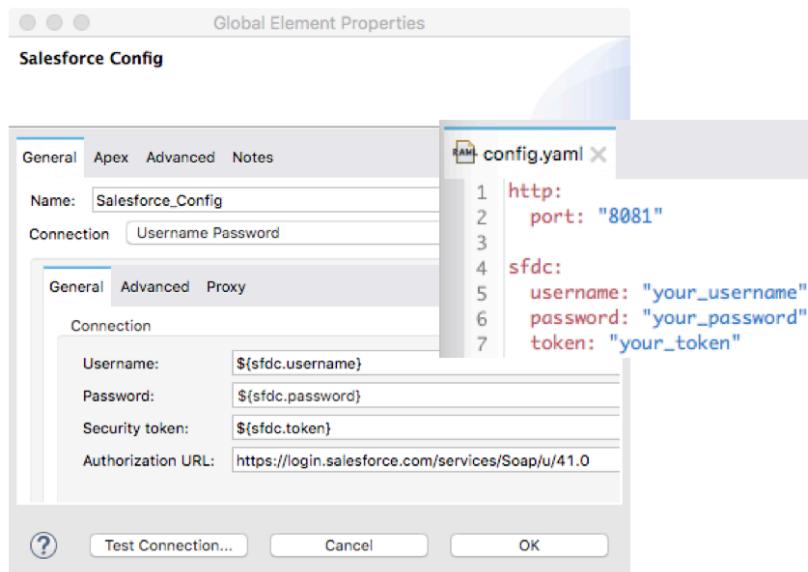
200 OK 1757.84 ms DETAILS ▾

[
{
 "LastModifiedDate": "2015-07-12T21:00:51.000Z",
 "BillingPostalCode": null,
 "Id": null,
 "type": "Account",
 "Name": "GenePoint"
},
{
 "LastModifiedDate": "2015-07-12T21:00:51.000Z",
 "BillingPostalCode": null

Walkthrough 7-4: Use property placeholders in connectors

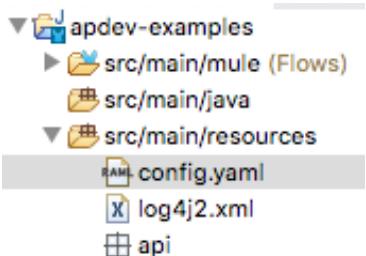
In this walkthrough, you introduce properties into your API implementation. You will:

- Create a YAML properties file for an application.
- Configure an application to use a properties file.
- Define and use HTTP and Salesforce connector properties.



Create a properties file

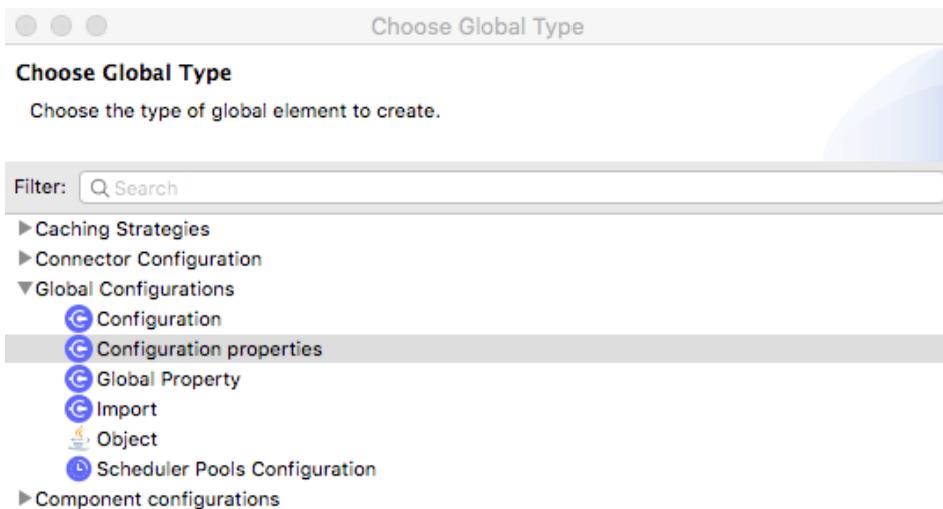
1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > File.
3. Set the file name to config.yaml and click Finish.



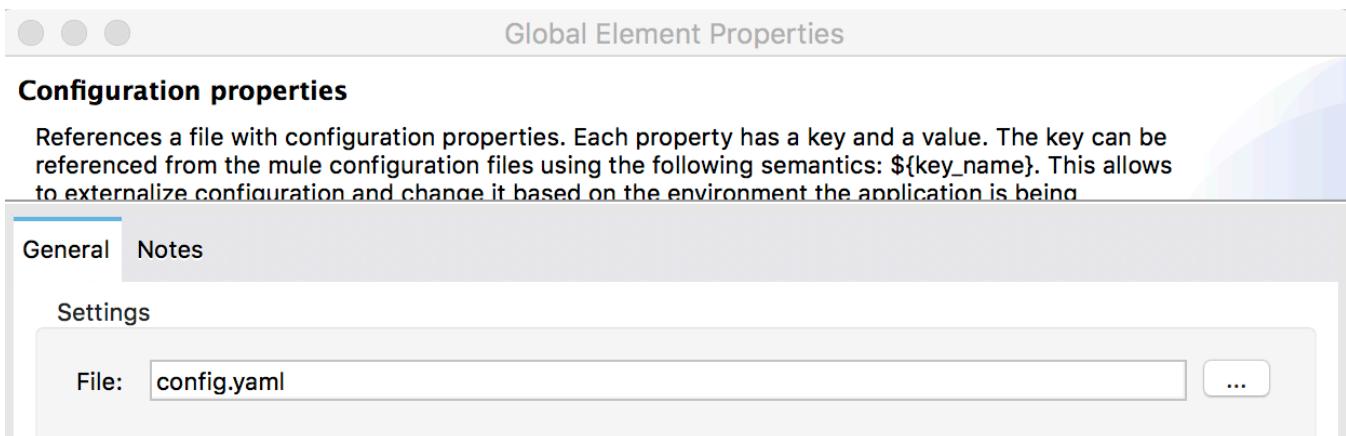
Create a Configuration properties global element

4. Return to global.xml.
5. In the Global Elements view, click Create.

6. In the Choose Global Type dialog box, select Global Configurations > Configuration properties and click OK.



7. In the Global Element Properties dialog box, set the file to config.yaml and click OK.



Parameterize the HTTP Listener port

8. Return to config.yaml.
9. Define a property called http.port and set it to 8081.

The screenshot shows the content of the config.yaml file. The file is titled '*config.yaml' and contains the following YAML code:

```
1 http:  
2   port: "8081"
```

10. Save the file.
11. Return to global.xml.
12. Double-click the HTTP Listener config global element.

13. Change the port from 8081 to the application property, \${http.port}.

HTTP Listener config
Configuration element for a HttpListener.

General Notes

Name: **HTTP_Listener_config**

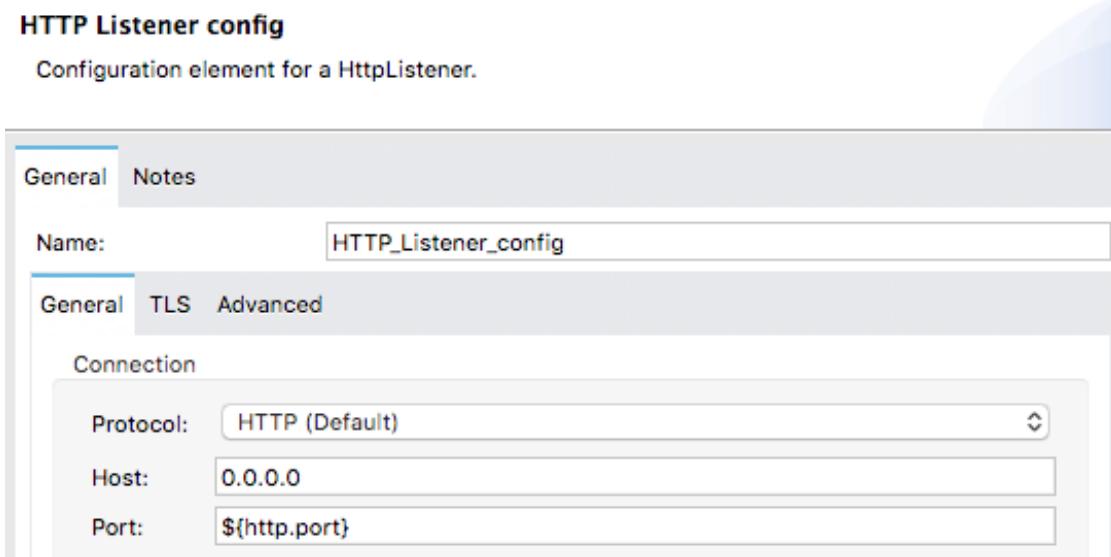
General TLS Advanced

Connection

Protocol: **HTTP (Default)**

Host: **0.0.0.0**

Port: **\${http.port}**



14. Click OK.

Parameterize the Salesforce credentials

15. Double-click the Salesforce Config global element.

16. Copy the token value and click OK.

17. Return to config.yaml .

18. Define a property called sfdc.username and set it to your Salesforce username.

19. Add properties for password and token and set them to your values.

apdev-examples accounts *global *config.yaml X

```
1 http:
2   port: "8081"
3
4 sfdc:
5   username: "your_username"
6   password: "your_password"
7   token: "your_token"
```

20. Save the file.

21. Return to global.xml.

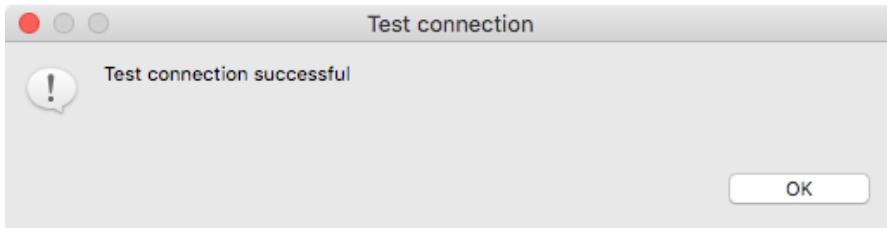
22. Double-click the Salesforce Config global element.

23. Change the values to use the application properties.

Connection

Username:	<code>\$(sfdc.username)</code>
Password:	<code>\$(sfdc.password)</code> <input checked="" type="checkbox"/> Show password
Security token:	<code>\$(sfdc.token)</code>
Authorization URL:	<code>https://login.salesforce.com/services/Soap/u/41.0</code>

24. Click Test Connection and make sure it succeeds.



Note: If your connection fails, click OK and then go back and make sure you do not have any syntax errors in config.yaml and that you saved the file.

25. Click OK.

26. In the Global Element Properties dialog box, click OK.

Test the application

27. Save all files to redeploy the application.

28. In Advanced REST Client, make the same request to <http://localhost:8081/sfdc> and confirm you still get data.

Method Request URL

GET <http://localhost:8081/sfdc>

SEND :
▼

Parameters ▼

200 OK 1757.84 ms DETAILS ▼

▼

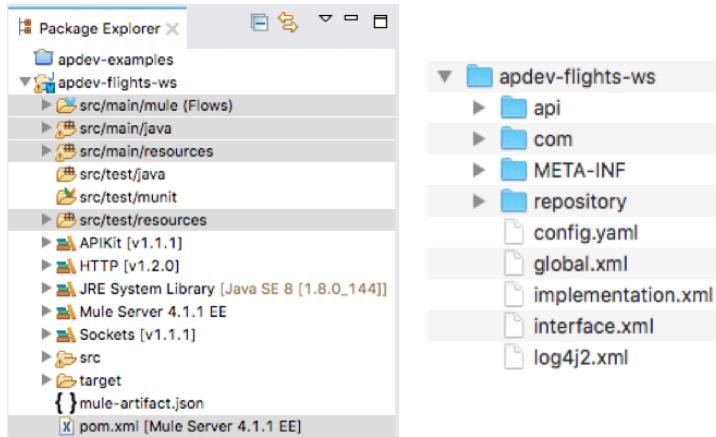
```
[{"LastModifiedDate": "2015-07-12T21:00:51.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "GenePoint"}, {"LastModifiedDate": "2015-07-12T21:00:51.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "GenePoint"}]
```

29. Return to Anypoint Studio and stop the project.

Walkthrough 7-5: Create a well-organized Mule project

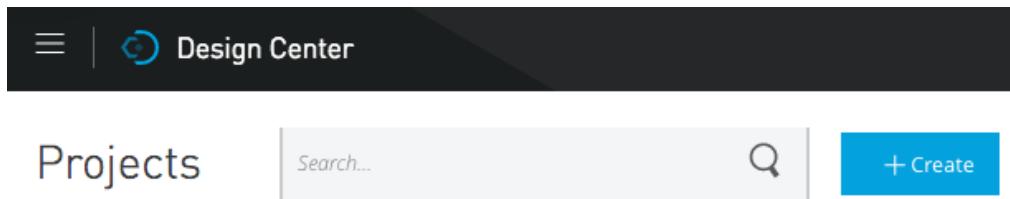
In this walkthrough, you create a new project for the Mule United Airlines (MUA) flights application that you will build during the course and then review and organize its files and folders. You will:

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



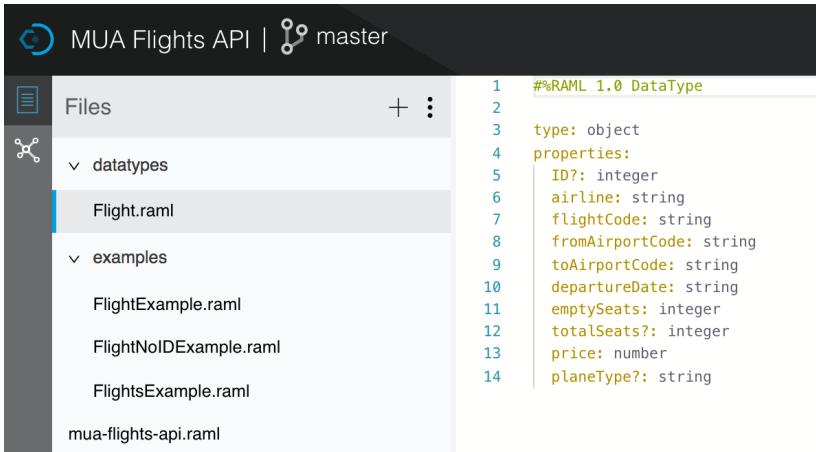
Create a new API in Design Center

1. Return to Anypoint Platform in a web browser.
2. In Design Center, create a new API specification project called MUA Flights API.



3. In API designer, click the options menu in the file browser and select Import.
4. In the Import dialog box, browse to your student files and select the MUA Flights API.zip located in the resources folder.
5. Click Import.
6. In the Replace dialog box, click Replace file.

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

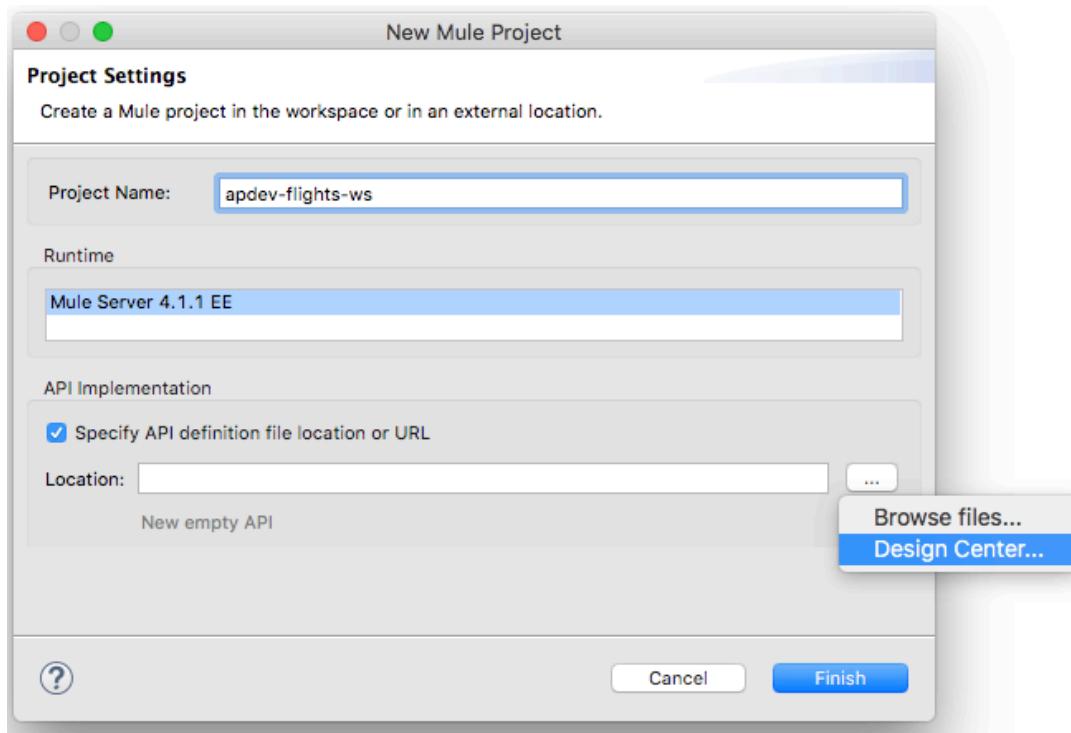


The screenshot shows a file browser interface for the 'MUA Flights API' repository. The 'Flight.raml' file is selected in the list. The right pane displays the RAML code for the Flight data type:

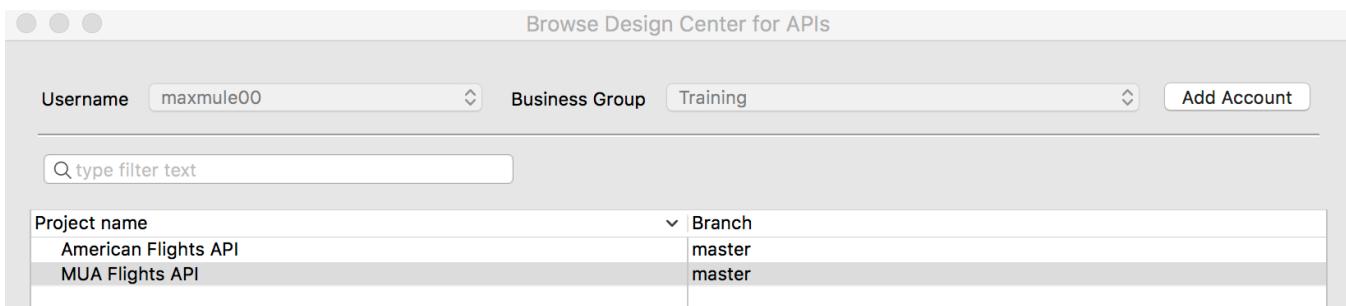
```
1  #%RAML 1.0 DataType
2
3  type: object
4  properties:
5    ID?: integer
6    airline: string
7    flightCode: string
8    fromAirportCode: string
9    toAirportCode: string
10   departureDate: string
11   emptySeats: integer
12   totalSeats?: integer
13   price: number
14   planeType?: string
```

Create a new project to implement this API in Anypoint Studio

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



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

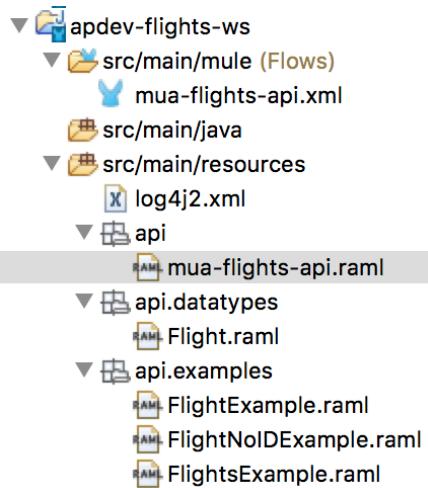


14. In the New Mule Project dialog box, click Finish.

Locate the new RAML files in Anypoint Studio

15. Return to the apdev-flights-ws project in Anypoint Studio.

16. In the Package Explorer, expand the folders in src/main/resources folder; you should see the MUA Flights API files.



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

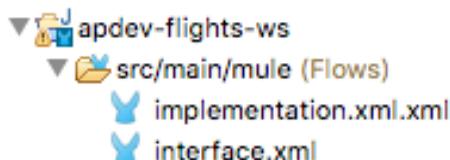
18. Leave the file open.

Review project configuration files

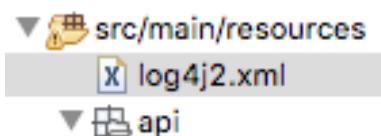
19. Look at the mua-flights-api.xml file that was created; it should have a get:\flights flow and a post:\flights flow.

20. Rename mua-flights-api.xml to interface.xml.

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

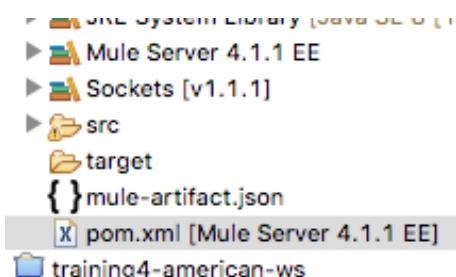


22. Locate log4j2.xml in src/main/resources and open it.



23. Review its contents and then close the file.

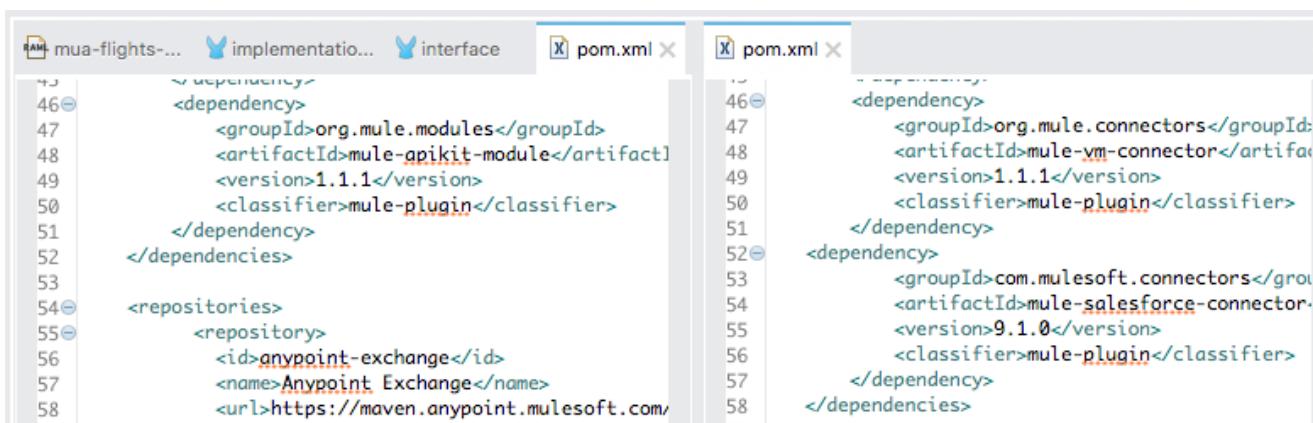
24. Locate pom.xml in the project and open it.



25. Review its contents.

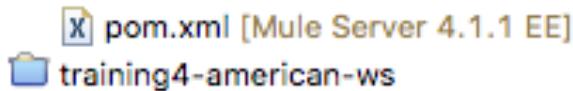
26. Return to the apdev-examples project and open its pom.xml file.

27. Compare the two pom.xml files.



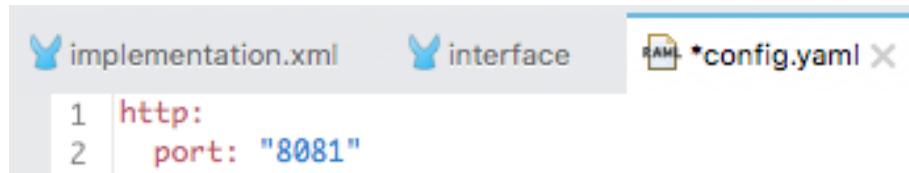
28. Close both pom.xml files.

29. In the Package Explorer, right-click apdev-examples and select Close Project.



Create a properties file for application properties

30. In the apdev-flights-ws project, right-click src/main/resources and select New > File.
31. In the New File dialog box, set the file name to config.yaml and click Finish.
32. In config.yaml, define a property called http.port equal to "8081".

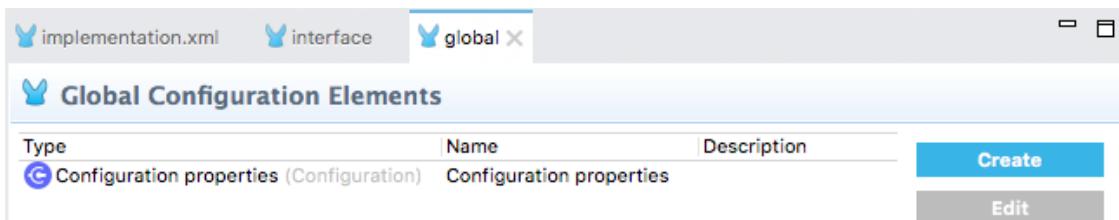


```
implementation.xml interface *config.yaml X
1 http:
2   port: "8081"
```

33. Save and close the file.

Create a global configuration file

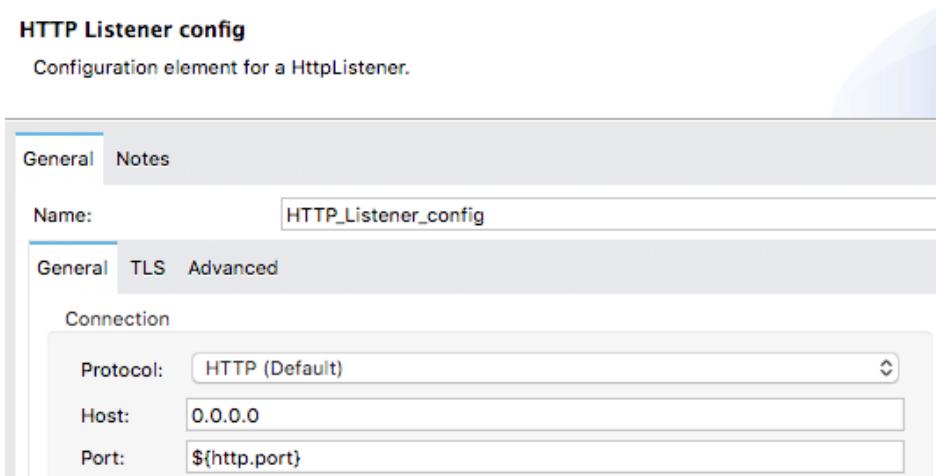
34. In src/main/mule, create a new Mule configuration file called global.
35. In global.xml, switch to the Global Elements view.
36. Create a new Configuration properties element with the file set to config.yaml.



Type	Name	Description
Configuration properties (Configuration)	Configuration properties	

Create **Edit**

37. Create a new HTTP Listener config element with the host set to 0.0.0.0 and the port set to the http.port property.



HTTP Listener config
Configuration element for a HttpListener.

General Notes

Name:

General TLS Advanced

Connection

Protocol:

Host:

Port:

38. Confirm you now have two global elements defined in global.xml.

Type	Name	Description
Configuration properties (Configuration)	Configuration properties	
HTTP Listener config (Configuration)	HTTP_Listener_config	

Create Edit

39. Save and close the file.

40. Go to the Global Elements view in interface.xml.

41. Delete the mua-flights-api-httpListener HTTP Listener Configuration.

Type	Name	Description
Router (Configuration)	mua-flights-api-config	

Create Edit

42. Return to the Message Flow view.

43. Select the HTTP Listener in mua-flights-api-main, and in the Listener properties view set the Connector configuration to HTTP_Listener_config.

Display Name: Listener

Basic Settings

Connector configuration: HTTP_Listener_config

General

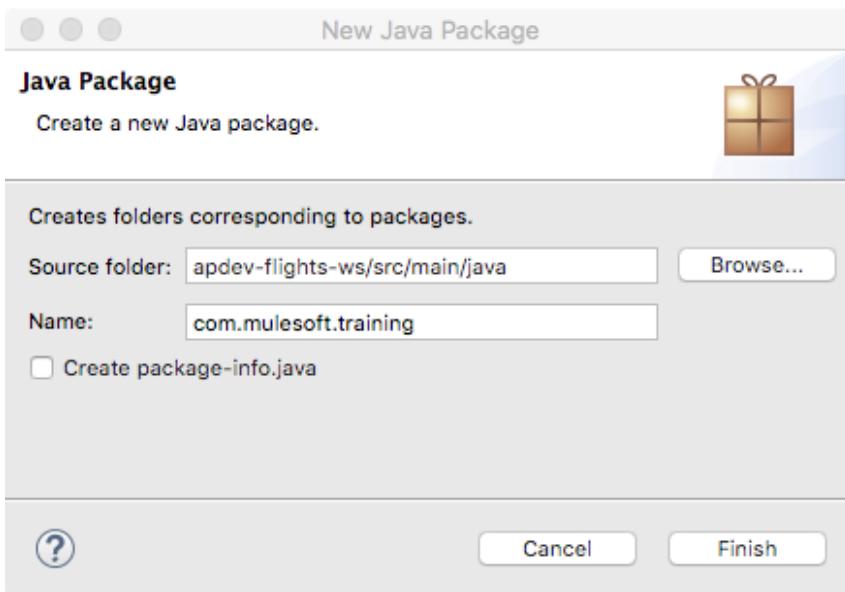
Path: /api/*

44. Select the HTTP Listener in mua-flights-api-console, and in the Listener properties view set the connector configuration to HTTP_Listener_config.

Add Java files to src/main/java

45. In the Package Explorer, right-click the src/main/java folder and select New > Package.

46. In the New Java Package dialog box, set the name to com.mulesoft.training and click Finish.



47. In your computer's file browser, locate the Flight.java file in the resources folder.

48. Drag the Flight.java file into the new com.mulesoft.training package in the src/main/java folder in the Anypoint Studio apdev-flights-ws project.

49. In the File Operation dialog box, select Copy files and click OK.

50. Open the Flight.java file.

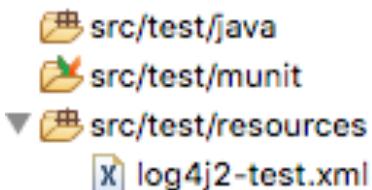
51. Review the code.



Review src/test folders

52. In the project, locate the three src/test folders.

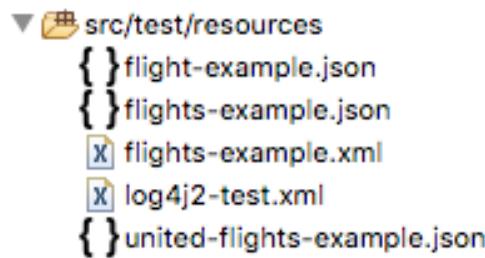
53. Expand the src/test/resources folder.



Add test resources

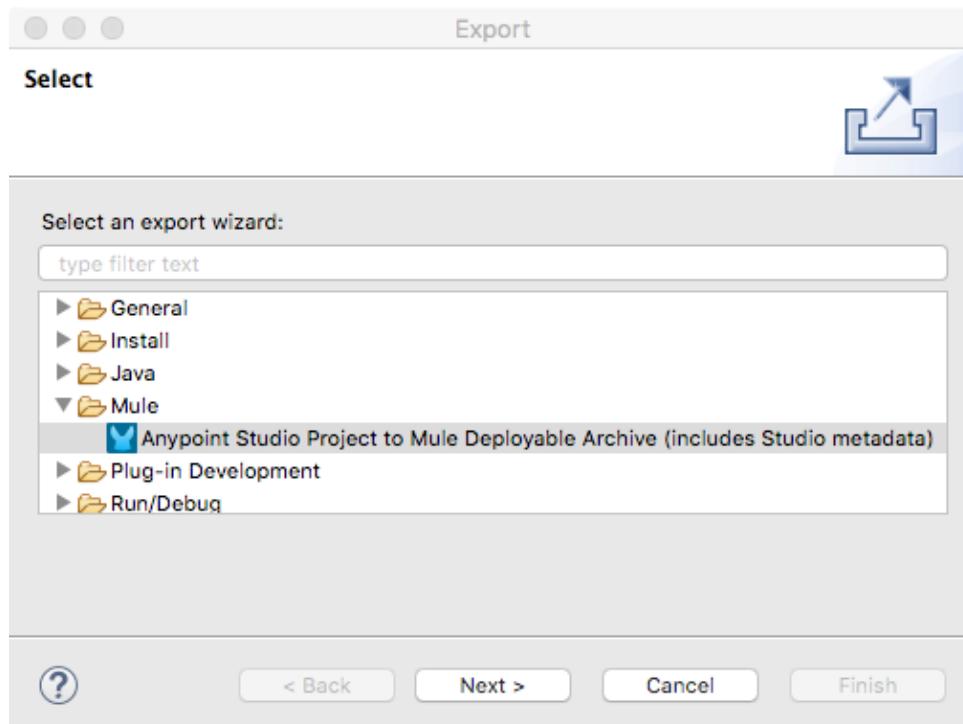
54. In your computer's file browser, expand the examples folder in the student files.

55. Select the three flights and one united-flights files (JSON and XML) and drag them into the src/test/resources folder in the Anypoint Studio apdev-flights-ws project.

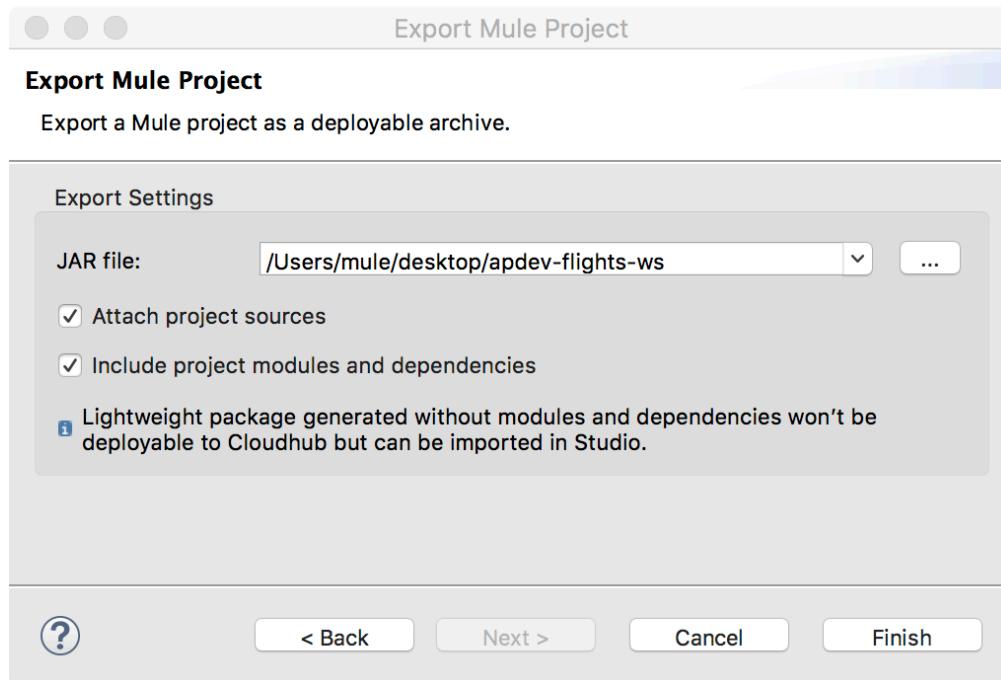


Examine the contents of a deployable archive

56. In Anypoint Studio, right-click the project and select Export.
57. In the Export dialog box, select Mule > Anypoint Studio Project to Mule Deployable Archive and click Next.

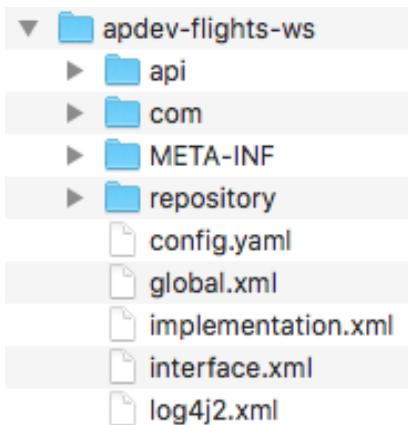


58. In the Export Mule Project dialog box, set the JAR file to a location that you can find and click Finish.



59. In your computer's file browser, locate the JAR file and unzip it.

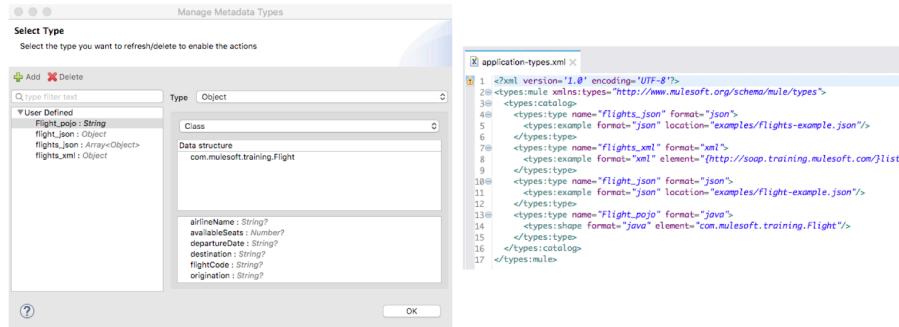
60. Open the resulting apdev-flights-ws folder and examine the contents.



Walkthrough 7-6: Manage metadata for a project

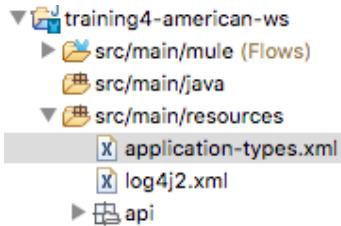
In this walkthrough, you define metadata for the new apdev-flights-ws project. You will:

- Review existing metadata for the training-american-ws project.
- Define new metadata to be used in transformations in the new apdev-flights-ws project.
- Manage metadata.



Locate existing metadata in the training-american-ws project

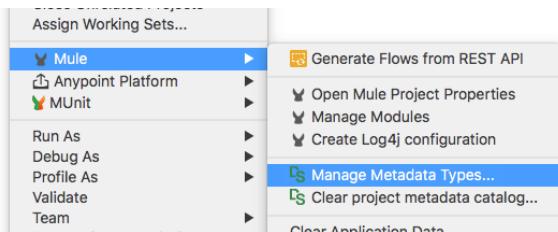
1. Open the training-american-ws project.
2. Locate application-types.xml in src/main/resources.



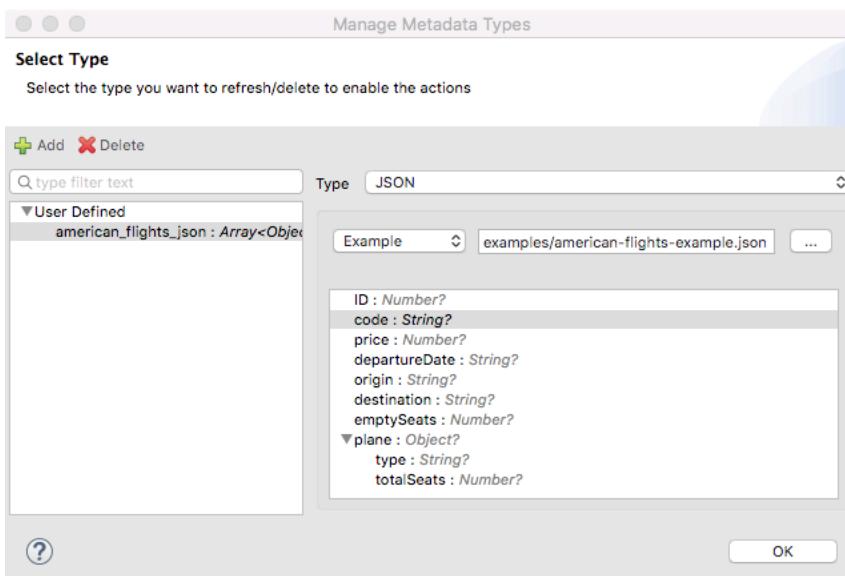
3. Open the file and review the code.

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
<types:catalog>
<types:type name="american_flights_json" format="json">
<types:example format="json" location="examples/american-flights-example.json"/>
</types:type>
</types:catalog>
<types:enrichment select="#89727fef-578a-4d3e-b1ce-533af28359e7">
<types:processor-declaration>
<types:output-event>
<types:message>
<types:payload type="american_flights_json"/>
</types:message>
</types:output-event>
</types:processor-declaration>
</types:enrichment>
</types:mule>
```

- Right-click the training-american-ws project in the Project Explorer and review the options under Mule; you should see two for metadata.



- Select Mule > Manage Metadata types.
- In the Manage Metadata Types dialog box, select the american-flights_json type.



- Click OK.
- Close the project.

Review the API specification and resources for the new apdev-flights-ws project

- Return to the apdev-flights-ws project.
- Return to mua-flights-api.raml and review the specified response and request types.

```

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

post:
  body:
    application/json:
      type: Flight
      example: !include examples/FlightNoIDExample.raml
  
```

11. Locate the files you added to the src/test/resources folder.



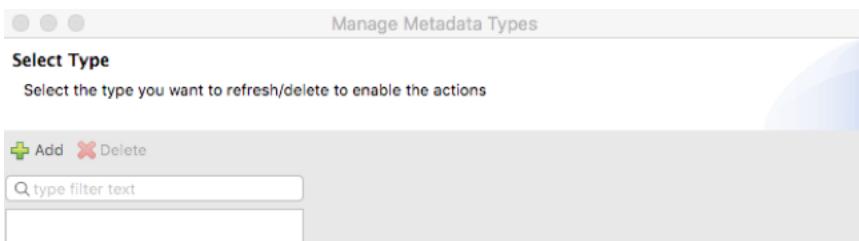
12. Locate the files you added to the src/main/java folder.



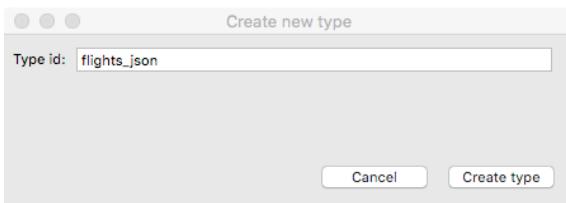
Create flights_json metadata type

13. Right-click the project and select Mule > Manage Metadata Types.

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



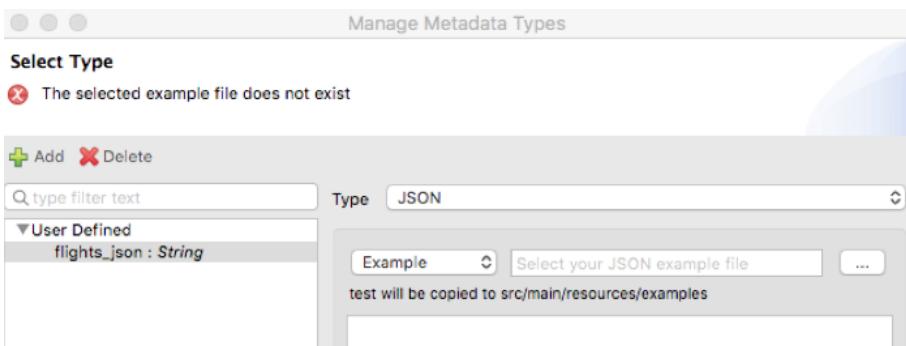
15. In the Create new type dialog box, set the type id to flights_json.



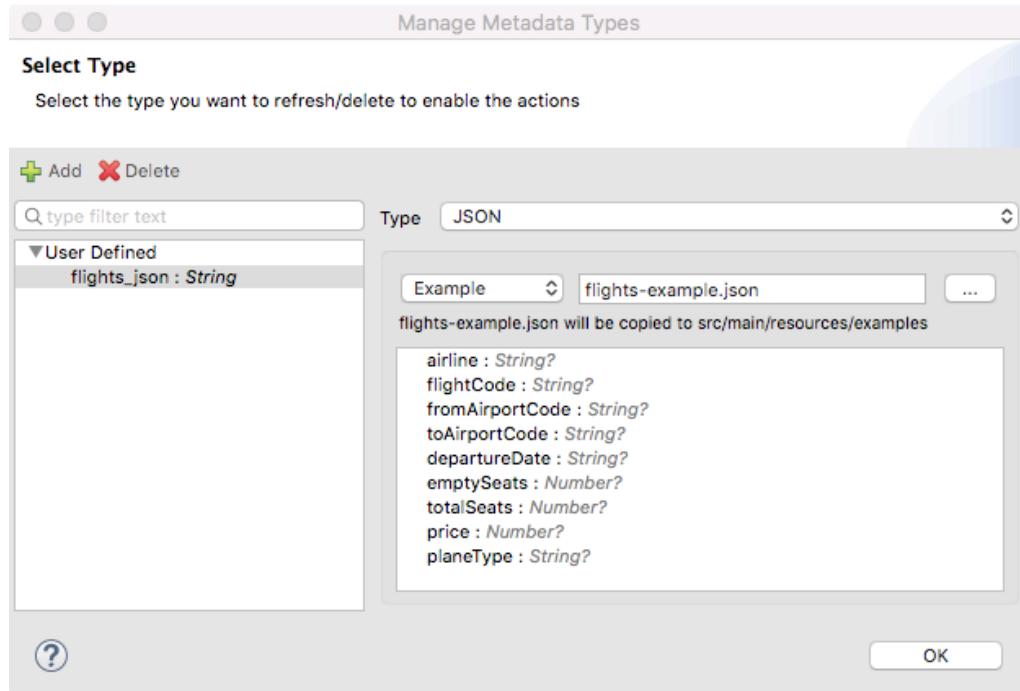
16. Click Create type.

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

18. In the drop-down menu beneath the type, select Example.



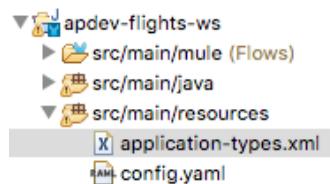
19. Click the browse button and select the flights-example.json file in src/test/resources.



20. In the Select metadata type dialog box, click OK.

Locate the new metadata definition file

21. Locate application-types.xml in src/main/resources.



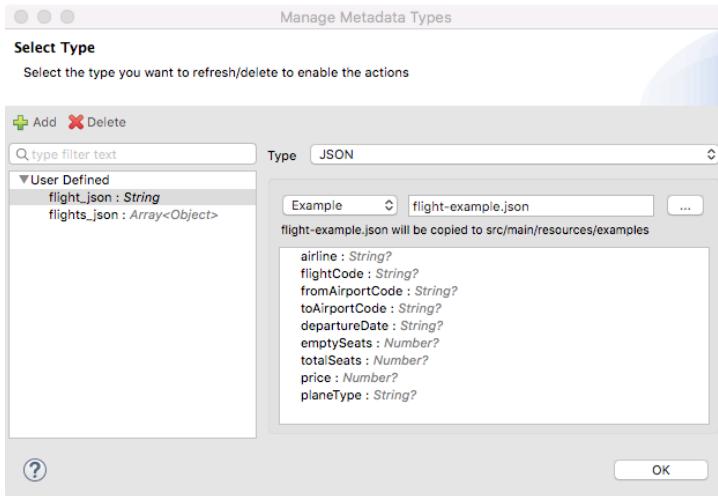
22. Open the file and review the code.

A screenshot of a code editor showing the XML file "application-types.xml". The code is as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
    <types:catalog>
        <types:type name="flights_json" format="json">
            <types:example format="json" location="examples/flights-example.json"/>
        </types:type>
    </types:catalog>
</types:mule>
```

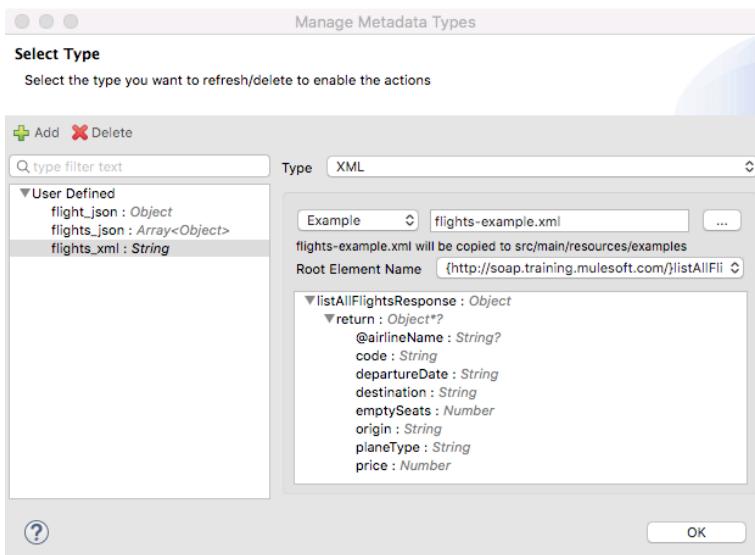
Create flight_json metadata type

23. Right-click the project and select Mule > Manage Metadata Types.
24. In the Select metadata type dialog box, click Add.
25. In the Create new type dialog box, set the type id to flight_json and click Create type.
26. In the Select metadata dialog box, set the type to JSON.
27. Select Example and browse to and select the flight-example.json file in src/test/resources.



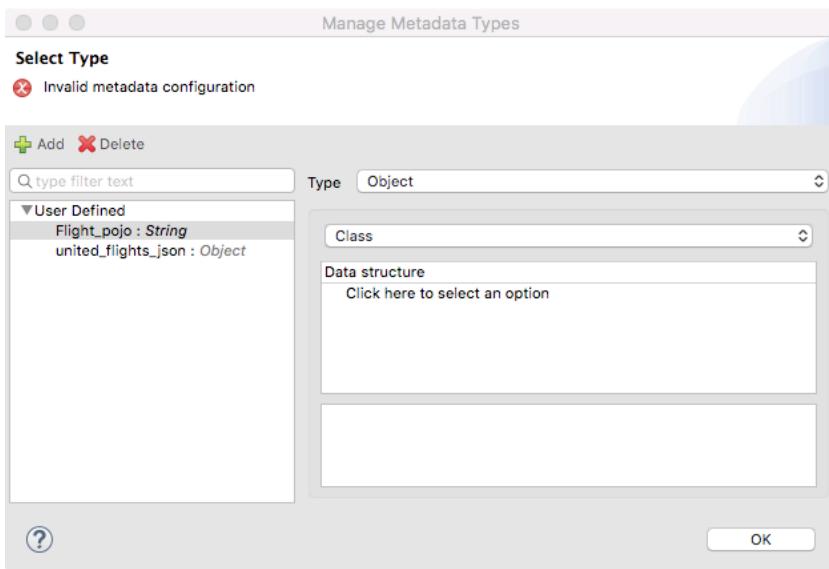
Create flights_xml metadata type

28. In the Select metadata type dialog box, click Add.
29. In the Create new type dialog box, set the type id to flights_xml and click Create type.
30. In the Select metadata dialog box, set the type to XML.
31. Select Example and browse to and select the flights-example.xml file in src/test/resources.

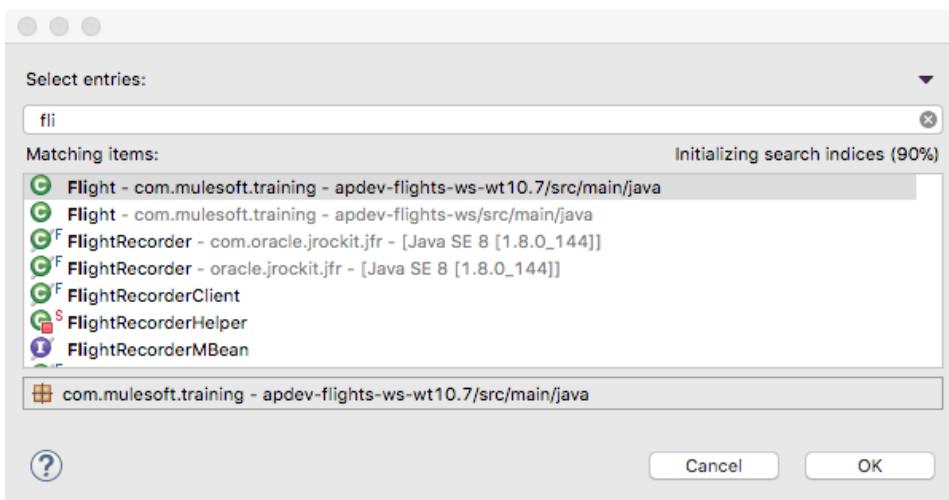


Create Flight metadata type

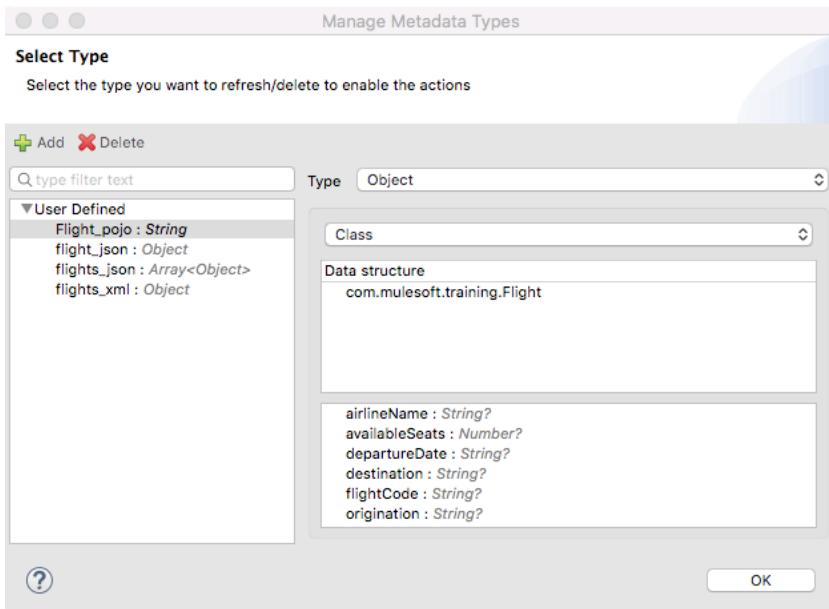
32. In the Select metadata type dialog box, click the Add button.
33. In the Create new type dialog box, set the type id to Flight_pojo and click Create type
34. In the Select metadata type dialog box, set the type to Object.



35. In the Data structure section, click the Click here to select an option link.
36. In the drop-down menu that appears, select Java object.
37. In the dialog box that opens, type fli and then in the list of classes that appears, select Flight – com.mulesoft.training.com.



38. Click OK.



39. In the Manage Metadata Types dialog box, click OK.

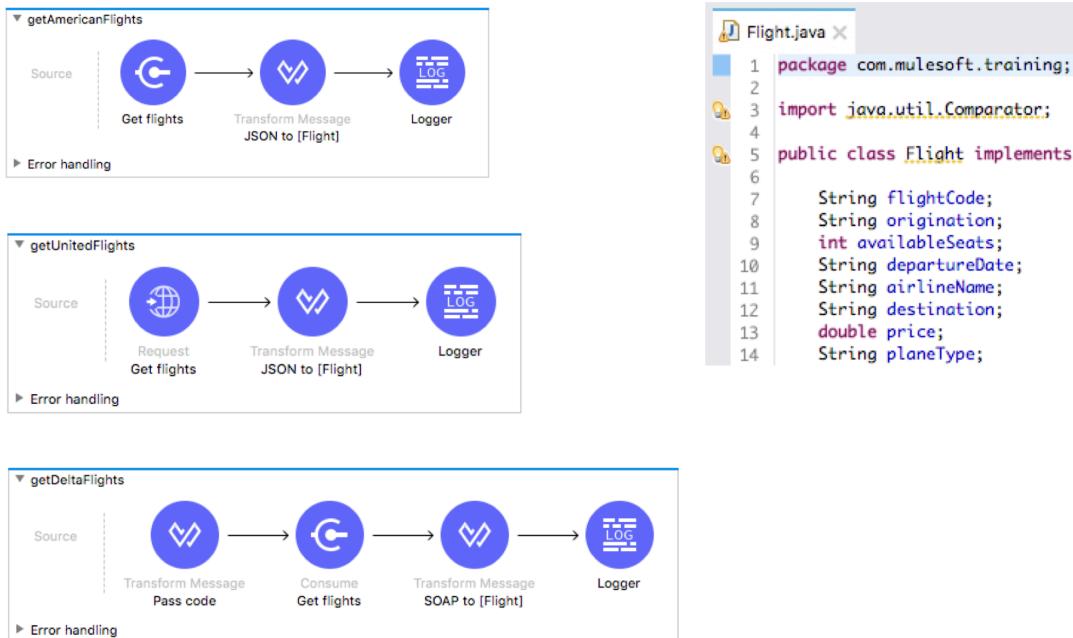
Review the metadata definition file

40. Return to application-types.xml in src/main/resources.

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
    <types:catalog>
        <types:type name="flights_json" format="json">
            <types:example format="json" location="examples/flights-example.json"/>
        </types:type>
        <types:type name="flights_xml" format="xml">
            <types:example format="xml" element="{http://soap.training.mulesoft.com}list">
                <!-- This is a placeholder for the XML example -->
            </types:example>
        </types:type>
        <types:type name="flight_json" format="json">
            <types:example format="json" location="examples/flight-example.json"/>
        </types:type>
        <types:type name="Flight_pojo" format="java">
            <types:shape format="java" element="com.mulesoft.training.Flight"/>
        </types:type>
    </types:catalog>
</types:mule>
```

41. Close the file.

Module 8: Consuming Web Services



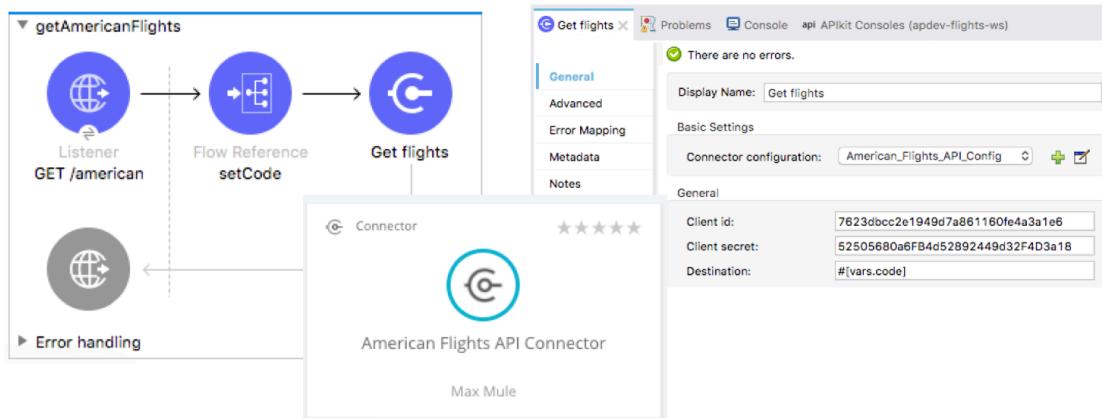
At the end of this module, you should be able to:

- 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.

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.

The Exchange interface shows the following assets:

- American Flights API Connector (Connector, Max Mule)
- American Flights API (REST API, Max Mule)

3. Select the connector and review its information.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a sidebar with a navigation menu and a search bar. The main content area displays the 'American Flights API Connector' page. At the top, there's a header with the connector's name, a star rating of 0 reviews, and a 'Rate and review' button. Below the header is a large placeholder image of a plane. Underneath the image, there's a note: 'This page currently doesn't contain a description. Click Edit to add text, images, videos, code blocks, etc...'. To the right of the note is an 'Edit' button. Further down, there's an 'Overview' section with details: Type: Connector, Created by: Max Mule, Published on: April 18, 2018. Below that is a 'Versions' section with a table:

Version	Runtime version
1.0.2	3.6.0
1.0.1	3.6.0
1.0.0	3.6.0

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-\[lastname\].cloudbu](http://training4-american-api-[lastname].cloudbu).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.

The screenshot shows the Advanced REST Client interface. At the top, there's a header with 'Method: GET', 'Request URL: http://training4-american-api-mule.cloudbu.io/flights', and a 'SEND' button. Below the header is a 'Parameters' section with a '^' icon. Under 'Headers', there are two entries: 'client_id' with value '7623dbcc2e1949d7a861160fe4a3a1e6' and 'client_secret' with value '52505680a6FB4d52892449d32F4D3a18'. There's also an 'ADD HEADER' button. The response section shows a green '200 OK' status with a '2125.58 ms' latency. To the right, there's a 'DETAILS' dropdown. Below the status, there are icons for copy, download, and refresh. The response body is displayed as a JSON array:

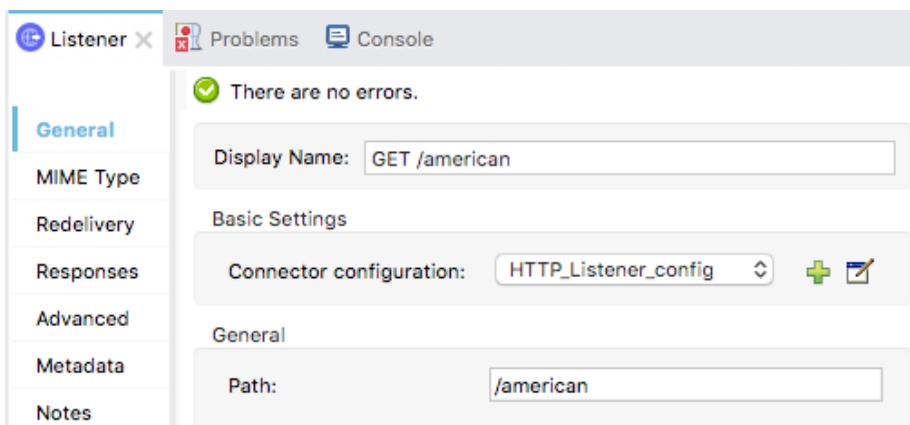
```

Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "-"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
}

```

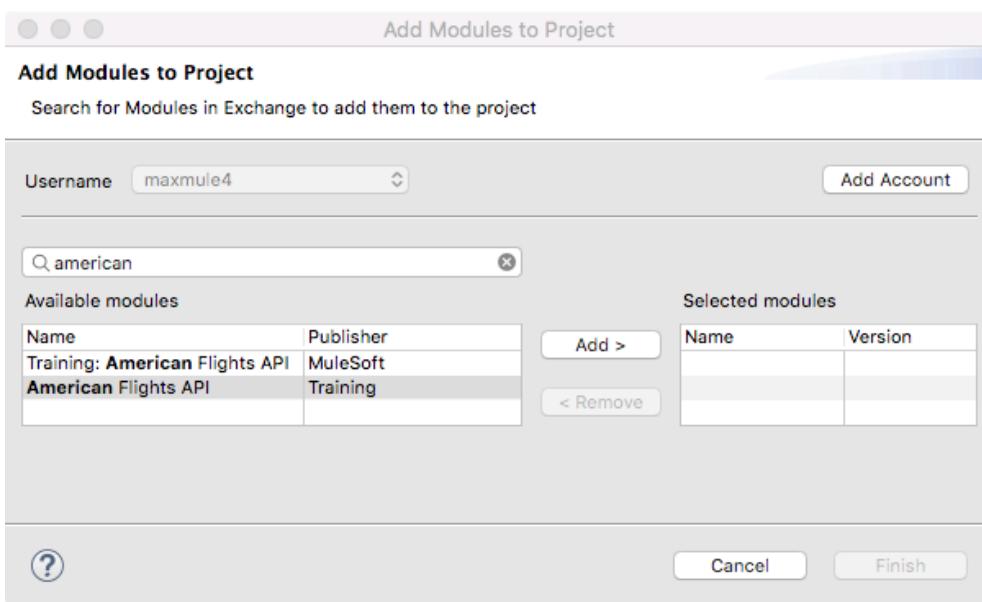
Add a new flow with an HTTP Listener

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.



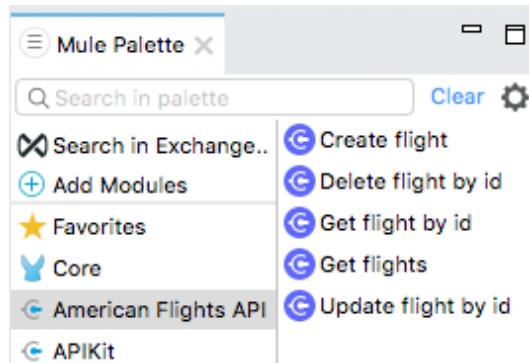
17. Select your American Flights API (**not** the Training: American Flights API) and click Add.

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.

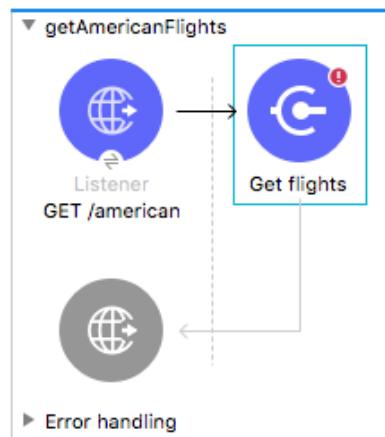
19. Wait for the module to be downloaded.

20. Select the new American Flights API module in the Mule Palette.

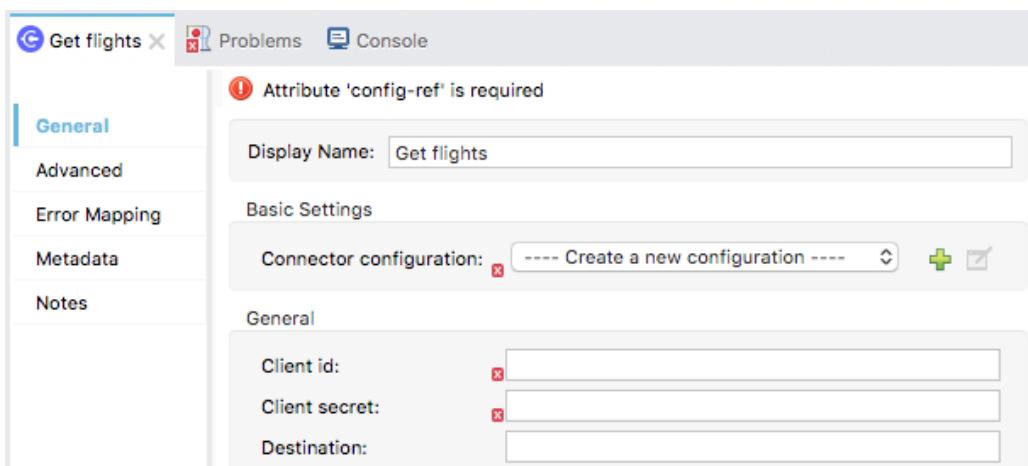


Add a Get all flights operation

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.

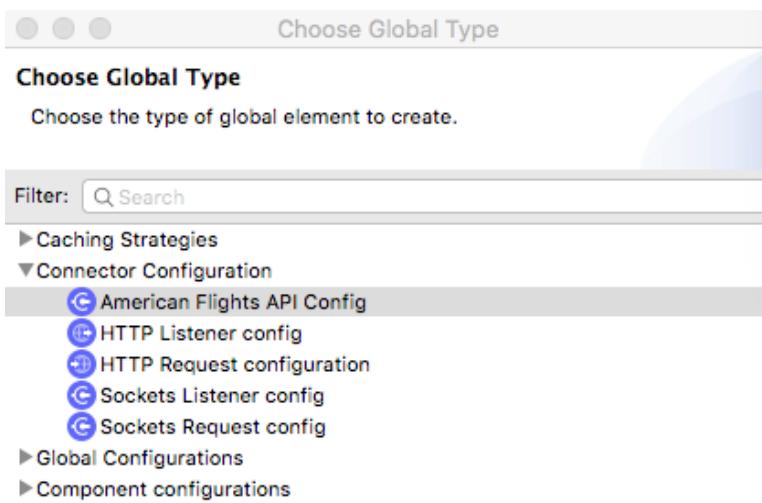


22. Select the Get all flights operation in the canvas and in its properties view, see that you need to need to create a new configuration and enter a client_id and client_secret.



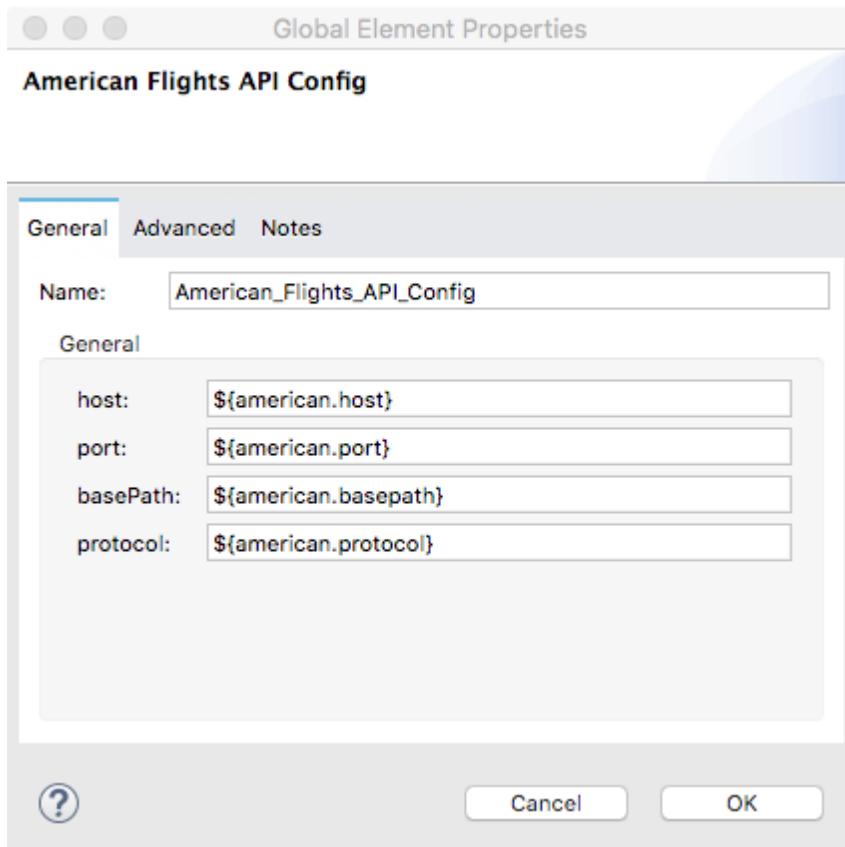
Configure the American Flights API connector

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):

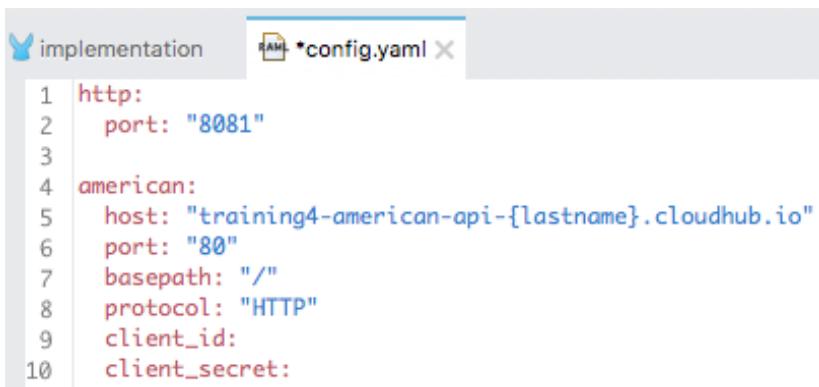
- host: \${american.host}
- port: \${american.port}
- basePath: \${american.basepath}
- protocol: \${american.protocol}



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 X
1 http:
2   port: "8081"
3
4 american:
5   host: "training4-american-api-{lastname}.cloudbhub.io"
6   port: "80"
7   basepath: "/"
8   protocol: "HTTP"
9   client_id:
10  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.

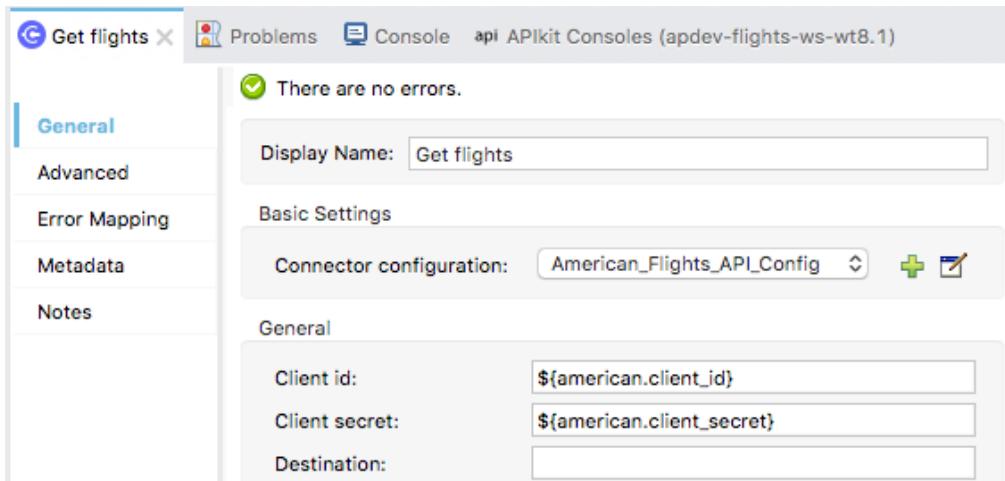
Configure the Get flights operation

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.



The screenshot shows the Mule Studio interface with the 'Get flights' operation selected. The top navigation bar includes 'Problems', 'Console', and 'api APIkit Consoles (apdev-flights-ws-wt8.1)'. The left sidebar has tabs for 'General', 'Advanced', 'Error Mapping', 'Metadata', and 'Notes'. The main panel displays the following configuration:

- General**:
 - Display Name: Get flights
 - Connector configuration: American_Flights_API_Config (with a plus icon and edit icon)
 - General** section:
 - Client id: \${american.client_id}
 - Client secret: \${american.client_secret}
 - Destination: (empty field)

Review metadata associated with the operation

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

```
Input Output
Q type filter text
▼ Mule Message
  ▼ Payload
    ▼ Array<Object> : Array<Object>
      ► plane : Object?
      code : String?
      price : Number?
      origin : String?
      destination : String?
      ID : Number?
      departureDate : String?
      emptySeats : Number?
```

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.

Method Request URL
GET <http://localhost:8081/american> SEND

Parameters

200 OK 2691.91 ms DETAILS

[Array[11]]
-0: { ... }
-1: { ... }
-2: {
 "ID": 3,
 "code": "ffee0192",
 "price": 300,
 "departureDate": "2016-01-20T00:00:00",
 "origin": "MUA",
 "destination": "LAX",
 "emptySeats": 0

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.

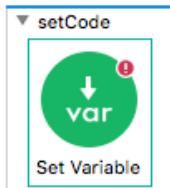
```
responses:  
  200:  
    body:  
      application/json:  
        type: Flight[]  
        example: !include examples/FlightsExample.raml
```

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
```

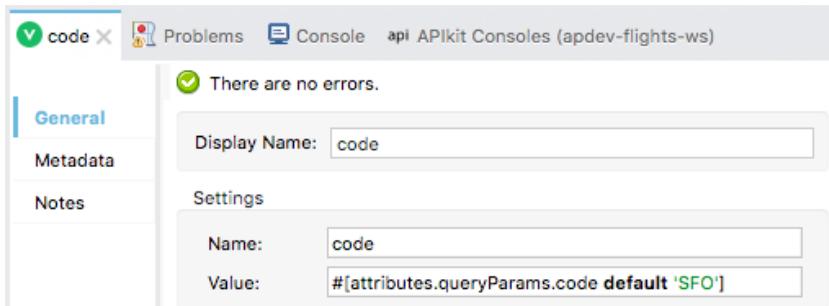
Create a variable to set the destination airport code

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.



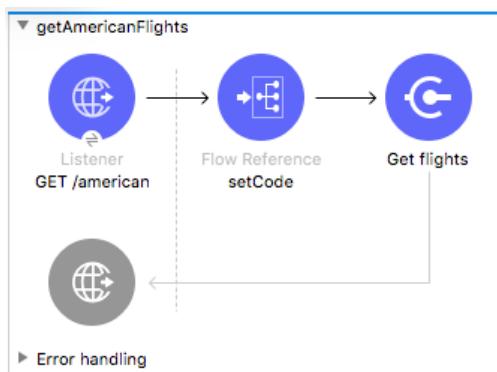
50. In the Set Variable properties view, set the display name and name to code.
51. Set the value to a query parameter called code and give it a default value of SFO if no query parameter is passed to the flow.

```
##[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]
```

General

Client id:	#{american.client_id}
Client secret:	#{american.client_secret}
Destination:	#[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.

Method Request URL
GET http://localhost:8081/american

SEND ::

Parameters ▾

200 OK 1743.09 ms DETAILS ▾

[
{
 "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": 8}

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

The screenshot shows a REST client interface. At the top, it displays the method "GET" and the request URL "http://localhost:8081/american?code=LAX". Below the URL, there are buttons for "SEND" and more options. Underneath, a "Parameters" dropdown is shown. The main area shows the response: a green "200 OK" box indicating success, followed by the response time "1204.27 ms" and a "DETAILS" button. The response body is a JSON array [] containing one object:

```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "nlane": {}}]
```

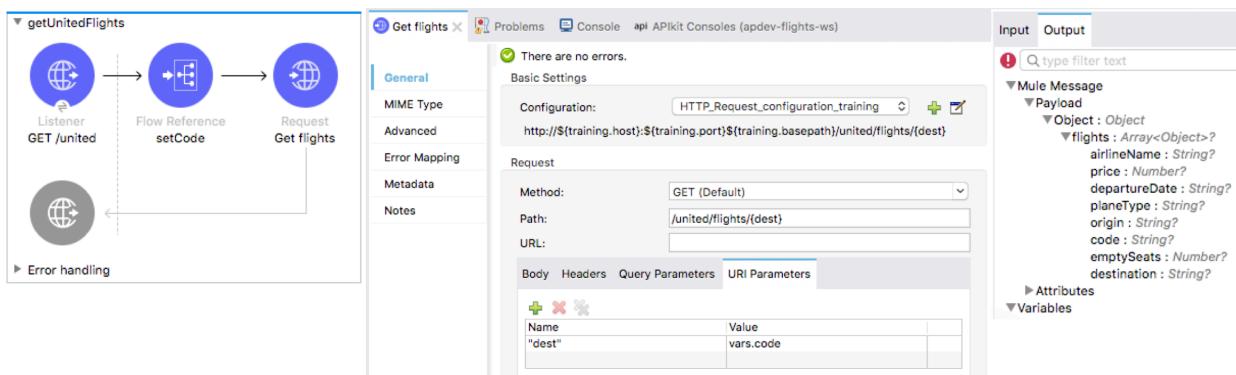
58. Examine the data structure of the JSON response.

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

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

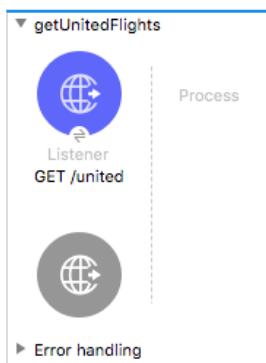
1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Advanced REST Client, make a new tab and make a GET request to this URL.
4. Review the structure of the JSON response.

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

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.

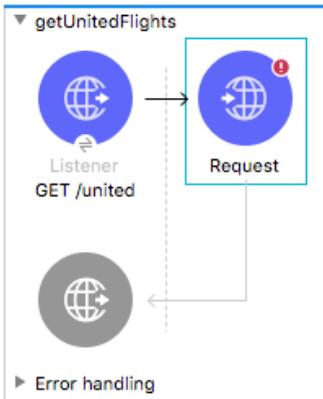
Add a new flow with an HTTP Listener operation

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.

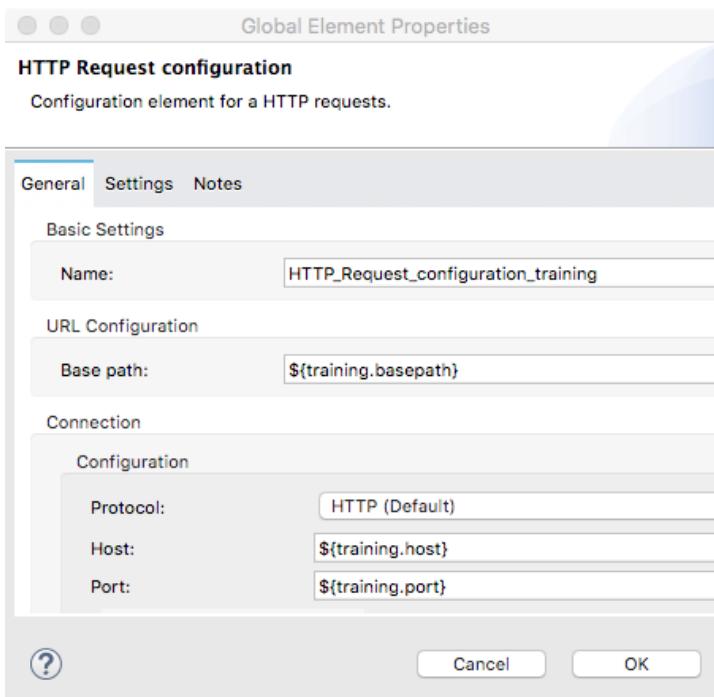
Create an HTTP Request configuration

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.

```
implementation *config.yaml X
1 http:
2   port: "8081"
3
4 american:
5   host: "training4-american-api-{lastname}.cloudbhub.io"
6   port: "80"
7   basepath: "/"
8   protocol: "HTTP"
9   client_id: "your_client_id"
10  client_secret: "your_client_secret"
11
12 training:
13   host: "ilt.learn.mulesoft.com"
14   port: "80"
15   basepath: "/"
16   protocol: "HTTP"
```

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.

The screenshot shows the Mule Studio interface with the 'GET flights' tab selected. The 'General' tab is active. The 'Basic Settings' section contains the following configuration:

- Display Name: GET flights
- Configuration: HTTP_Request_configuration_training
- Path: http://\${training.host}:\${training.port}\${training.basepath}/united/flights

The 'Request' section shows:

- Method: GET (Default)
- Path: /united/flights

A message at the top right says "There are no errors."

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.

The screenshot shows the DataSense Explorer with the 'Output' tab selected. The tree view shows the following structure:

- Mule Message
 - Payload
 - Binary : Binary
 - Attributes
 - Variables

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.

Method: GET Request URL: http://localhost:8081/united

Parameters:

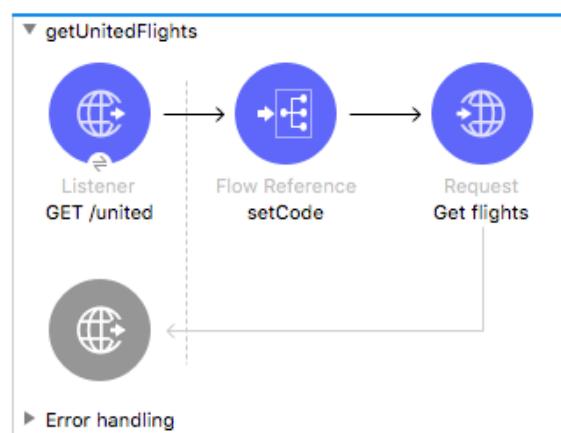
200 OK 446.93 ms DETAILS

```
{ "flights": [ { "code": "ER38sd", "price": 400, "origin": "MUA", "destination": "SFO", "departureDate": "2015/03/20", "planeType": "Boeing 737", "airlineName": "United", "emptySeats": 0 }, { "code": "ER45if", "price": 345.99 } ] }
```

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}.

The screenshot shows the 'Get flights' request configuration in the Mule Studio interface. The 'Request' tab is selected, displaying the following settings:

- Method: GET (Default)
- Path: /united/flights/{dest}
- URL: (empty)
- Body: #[payload]
- Headers, Query Parameters, and URI Parameters tabs are also visible but not selected.

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

The screenshot shows the 'Get flights' request configuration in the Mule Studio interface. The 'URI Parameters' tab is selected, displaying a table with one entry:

Name	Value
"dest"	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.



Method Request URL
GET <http://localhost:8081/united?code=CLE>

Parameters ▾

200 OK 513.70 ms DETAILS ▾

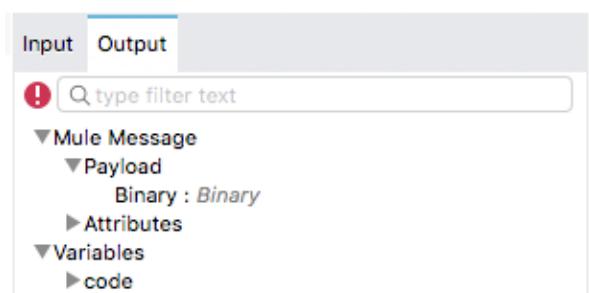
{
 "-flights": [Array[2]]
 "-0": {
 "code": "ER9fje",
 "price": 845,
 "origin": "MUA",
 "destination": "CLE",
 "departureDate": "2015/07/11"

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 no metadata for the payload.



Input Output

! type filter text

▼ Mule Message

 ▼ Payload

 Binary : Binary

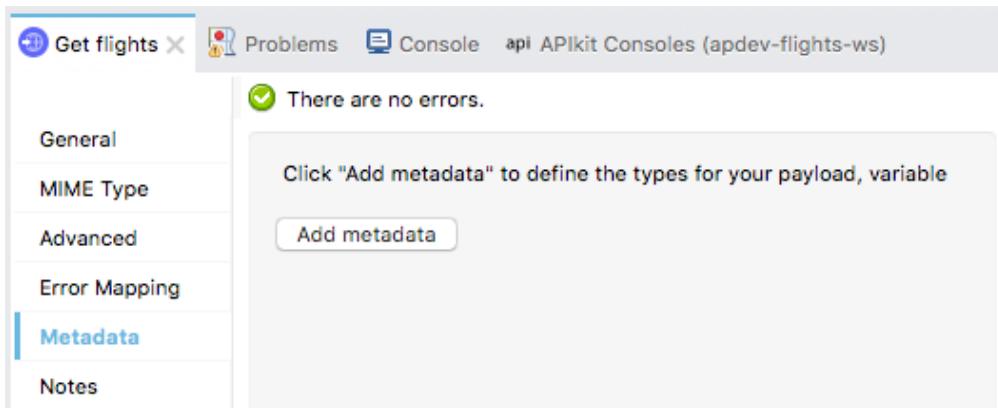
 ► Attributes

 ▼ Variables

 ► code

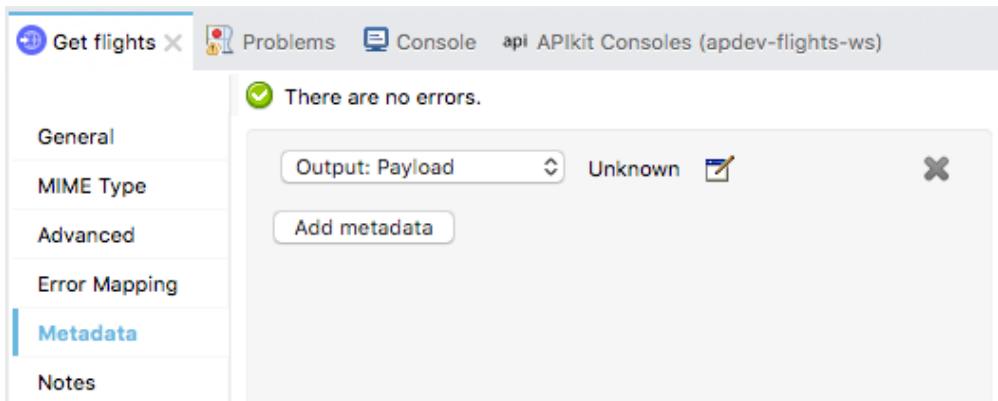
47. In the Get flights properties view, select the Metadata tab.

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.

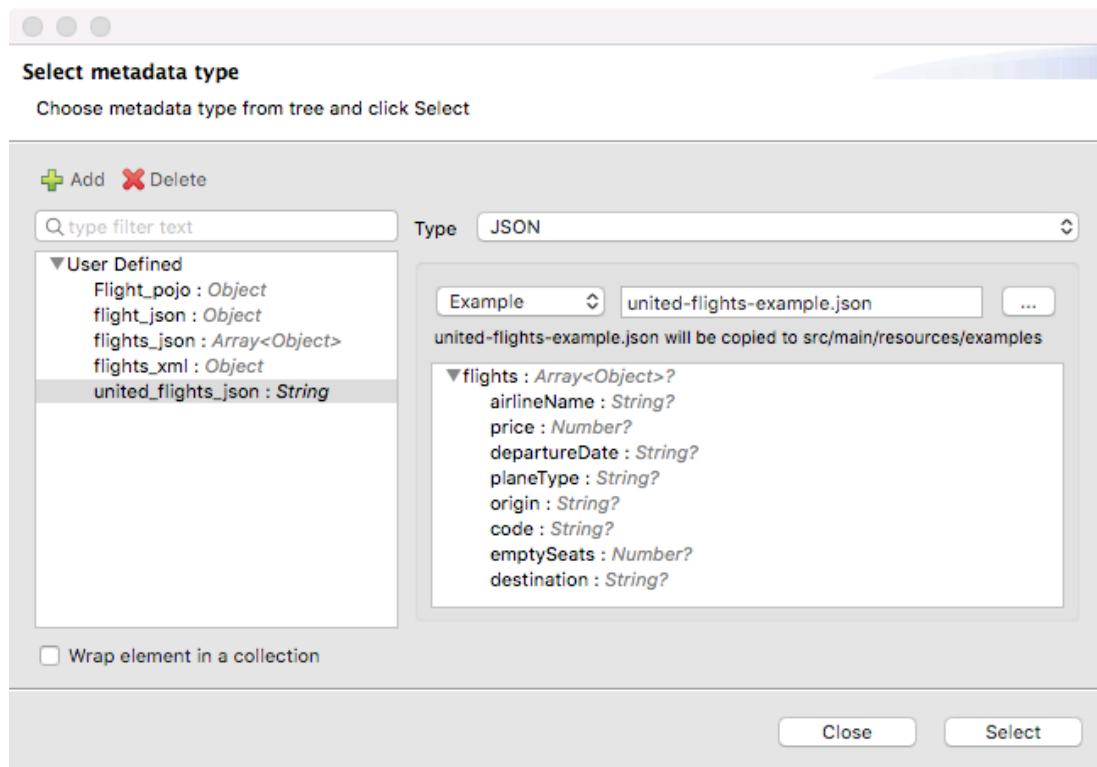
53. Click Create type.

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

55. Change the Schema selection to Example.

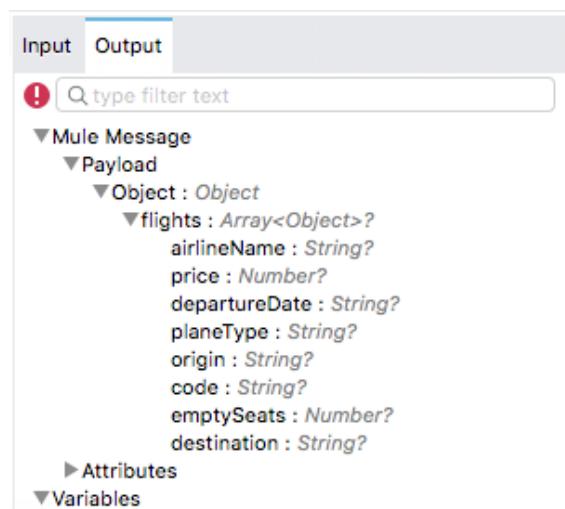
56. Click the browse button and navigate to the project's src/test/resources folder.

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



58. Click Select.

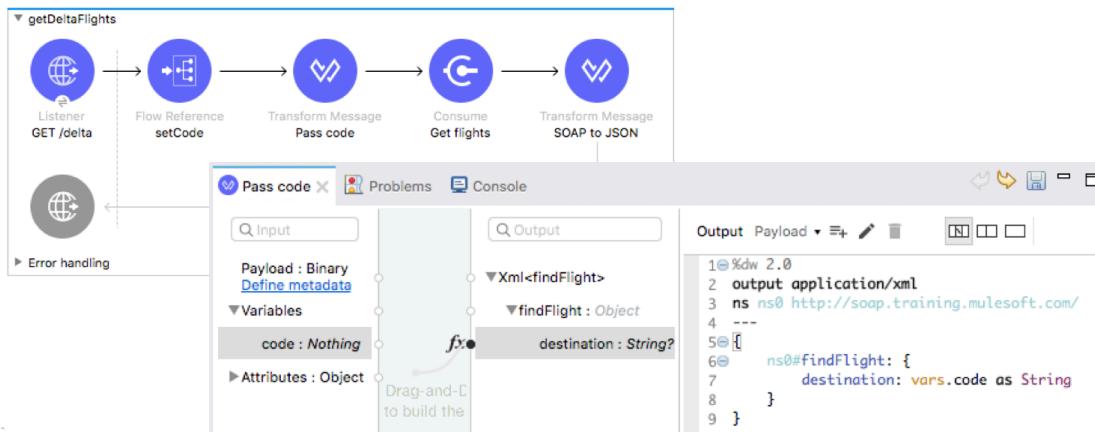
59. In the DataSense Explorer, select the Output tab and expand Payload; you should now see the structure for the payload.



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.



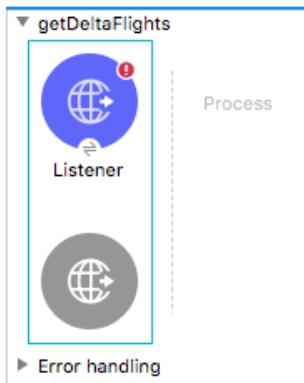
Browse the WSDL

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.

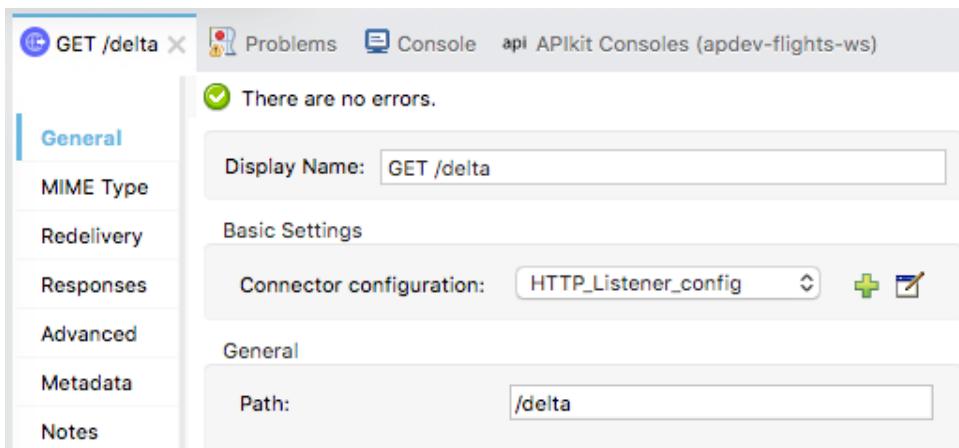
```
<?xml version='1.0' encoding='UTF-8' ?>
<wsdl:definitions name="TicketServiceService"
targetNamespace="http://soap.training.mulesoft.com/">
<wsdl:types>
<xsd:schema elementFormDefault="unqualified"
targetNamespace="http://soap.training.mulesoft.com/" version="1.0">
<xsd:element name="findFlight" type="tns:findFlight" />
<xsd:element name="findFlightResponse" type="tns:findFlightResponse" />
<xsd:element name="listAllFlights" type="tns:listAllFlights" />
<xsd:element name="listAllFlightsResponse" type="tns:listAllFlightsResponse" />
</xsd:schema>
</wsdl:types>
</wsdl:definitions>
```

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.



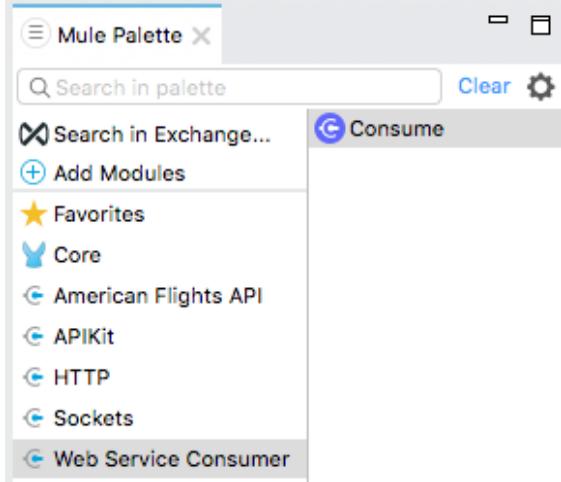
8. In the Listener properties view, set the display name to GET /delta.
9. Set the connector configuration to the existing HTTP_Listener_config.
10. Set the path to /delta and the allowed methods to GET.



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 and drag and drop it into the left side.
13. If you get a Select module version dialog box, select the latest version and click Add.



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.

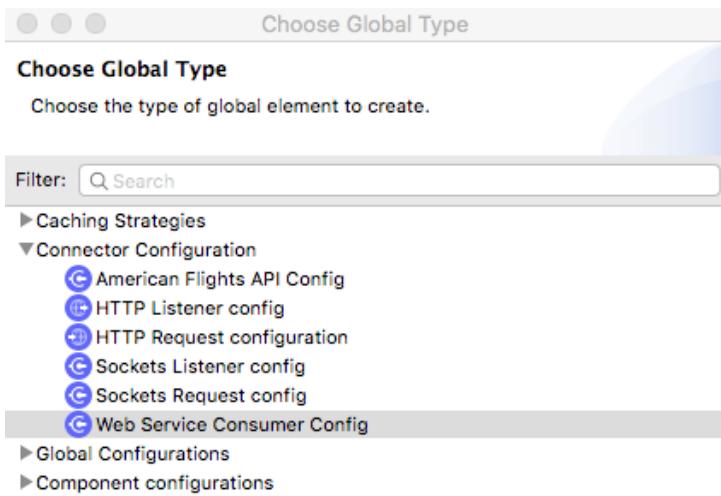
```

implementation *config.yaml global
 1 http:
 2   port: "8081"
 3
 4 american:
 5   host: "training4-american-api-[lastname].cloudbhub.io"
 6   port: "80"
 7   basepath: "/"
 8   protocol: "HTTP"
 9   client_id: "your_client_id"
10   client_secret: "your_client_secret"
11
12 training:
13   host: "mu.learn.mulesoft.com"
14   port: "80"
15   basepath: "/"
16   protocol: "HTTP"
17
18 delta:
19   wsdl: "http://mu.learn.mulesoft.com/delta?wsdl"
20   service: "TicketServiceService"
21   port: "TicketServicePort"

```

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

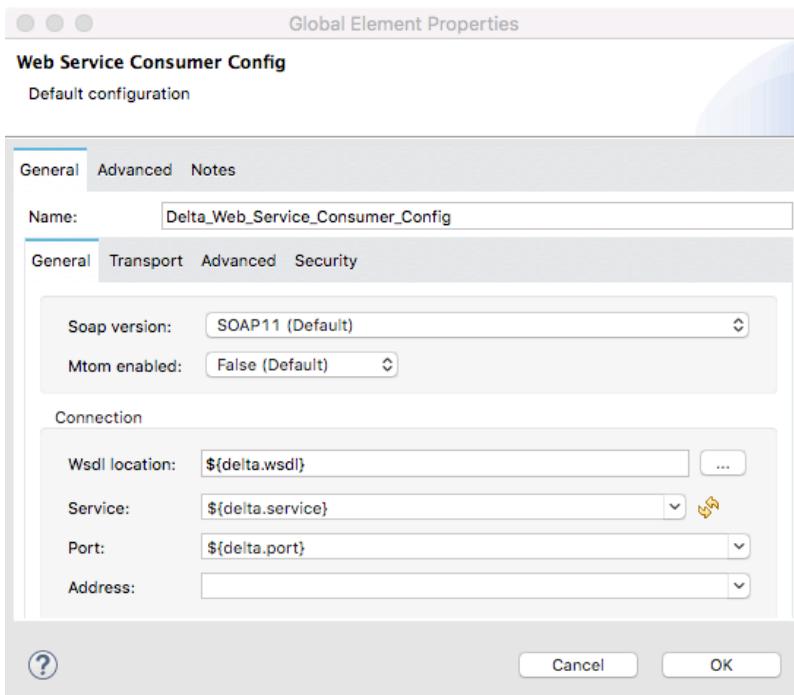
19. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer Config and click OK.



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

21. Set the

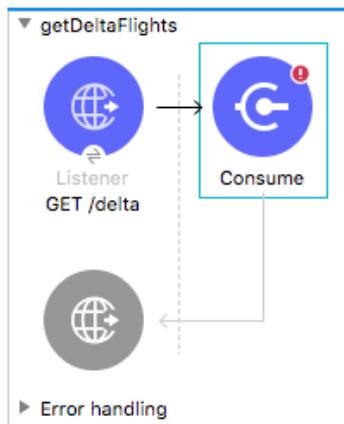
- Wsdl location: \${delta.wsdl}
- Service: \${delta.service}
- Port: \${delta.port}



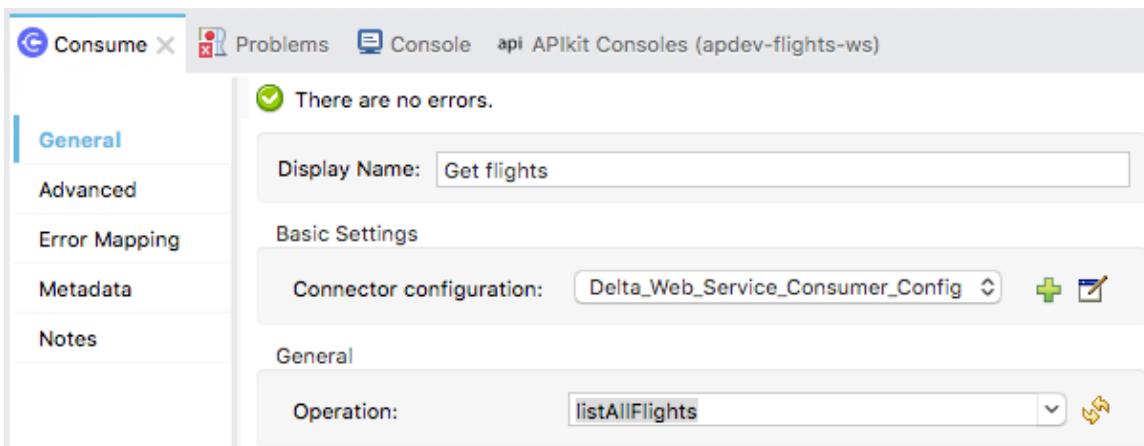
22. Click OK.

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.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top has an exclamation icon and placeholder text 'type filter text'. Below it, the 'Mule Message' section is expanded, showing the 'Payload' section. Under 'Payload', 'SoapOutputPayload : Object' is listed, along with its properties: 'attachments : Object?', 'body : Binary?', and 'headers : Object?'. Other collapsed sections include 'Attributes' and 'Variables'. At the bottom of the panel is a blue 'Refresh Metadata' button.

30. Save the file.

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

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top has placeholder text 'type filter text'. Below it, the 'Mule Message' section is expanded, showing the 'Payload' section. Under 'Payload', 'Object : Object' is listed, then 'body : Object?' is expanded to show 'listAllFlightsResponse : Object'. This expanded section shows the 'return : Object*' field, which contains several properties: 'airlineName : String?', 'code : String?', 'departureDate : String?', 'destination : String?', 'emptySeats : Number', 'origin : String?', 'planeType : String?', and 'price : Number'. Other collapsed sections include 'Attributes' and 'Variables'. The 'listAllFlightsResponse : Object' section is highlighted with a gray background.

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.

Method Request URL
GET <http://localhost:8081/delta> ABORT ::

Parameters ▾

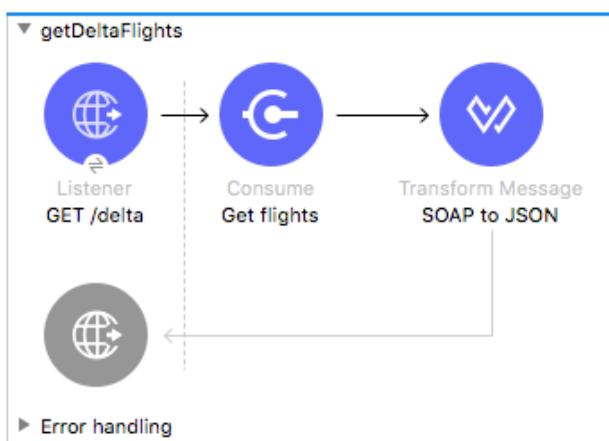
200 OK 9975.36 ms DETAILS ▾

✖️ ↻ ⟲ ⟳ 🔍

org.mule.runtime.extension.api.soap.SapOutputPayload@26bdc7dc

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.

Output Payload ▾

```
1 @ %dw 2.0
2 output application/json
3 ---
4 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.

200 OK 591.36 ms

```
{  
  "body": {  
    "listAllFlightsResponse": {  
      "return": {  
        "airlineName": "Delta",  
        "code": "A1B3D4",  
        "departureDate": "2015/02/12",  
        "destination": "PDX",  
        "emptySeats": "10",  
        "origin": "MUA",  
        "planeType": "Boing 777",  
        "price": "385.0"  
      }  
    }  
  }  
}  
"attachments": {},  
"headers": {}  
}
```

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

200 OK 3936.91 ms

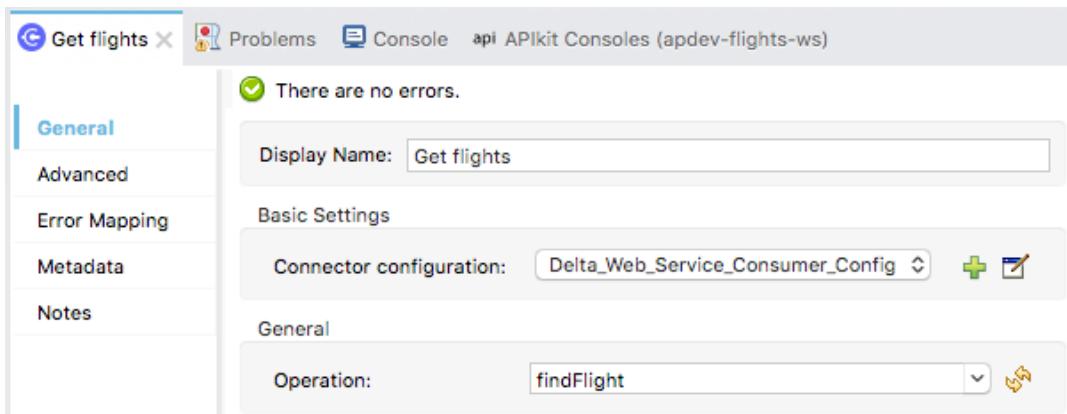
```
{  
  "body": {  
    "listAllFlightsResponse": {  
      "return": {  
        "airlineName": "Delta",  
        "code": "A1B2C3",  
        "departureDate": "2015/03/20",  
        "destination": "SFO",  
        "emptySeats": "40",  
        "origin": "MUA",  
        "planeType": "Boing 737",  
        "price": "400.0"  
      },  
      "return": {  
        "airlineName": "Delta",  
        "code": "A1B2C4",  
        "departureDate": "2015/02/11",  
        "destination": "LAX",  
        "emptySeats": "40",  
        "origin": "MUA",  
        "planeType": "Boing 737",  
        "price": "400.0"  
      }  
    }  
  }  
}
```

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.

A screenshot of the DataSense Explorer. The top navigation bar has tabs for 'Input' and 'Output' (which is selected), and a search bar. Below is a tree view of message components:

- Mule Message
 - Payload
 - Object : Object
 - body : Object?
 - findFlightResponse : Object
 - return : Object?
 - airlineName : String?
 - code : String?
 - departureDate : String?
 - destination : String?
 - emptySeats : Number
 - origin : String?
 - planeType : String?
 - price : Number

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

The screenshot shows the 'Input' tab selected in Anypoint Studio. Under 'Mule Message', the 'Payload' section is expanded. It shows '(Actual) Binary : Binary' and '(Expected) Xml<findFlight> : Object'. Within the expected payload, there is a 'findFlight : Object' entry with a 'destination : String?' key. Other sections like 'Attributes' and 'Variables' are also visible.

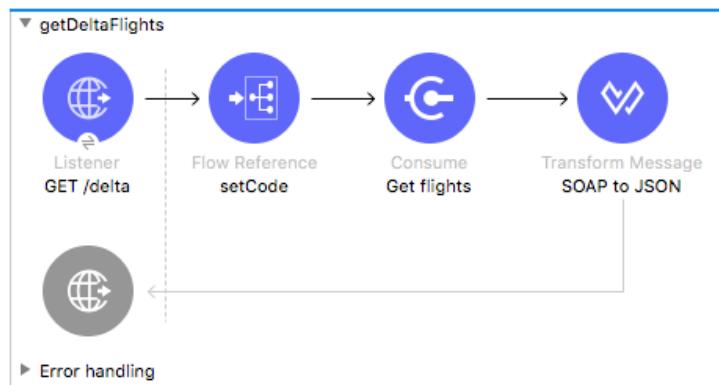
Test the application

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.

The screenshot shows a 500 Server Error response from the Advanced REST Client. The time taken is 395.86 ms. The error message is: "Cannot build default body request for operation [findFlight], the operation requires input parameters". There are icons for copy, download, and refresh.

Use the set airport code subflow

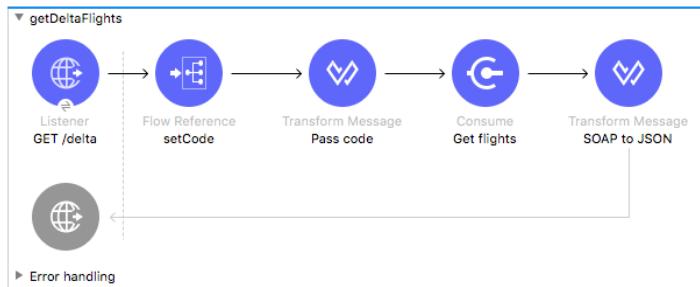
51. Return to getDeltaFlights in Anypoint Studio.
52. Add a Flow Reference component after the GET /delta Listener.
53. In the Flow Reference properties view, set the flow name to setCode.



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.

The screenshot shows the 'Pass code' properties view in the Mule Studio interface. The 'Input' section shows a 'Payload : Binary' node with a 'Define metadata' link. The 'Variables' section contains a 'code : Nothing' variable. The 'Output' section shows an 'Xml<findFlight>' node with a 'destination : String?' variable. A 'fx' icon indicates a transformation between the input and output variables. On the right, the 'Payload' tab of the code editor displays the following XML configuration:

```
1 %dw 2.0
2 output application/xml
3 ns ns0 http://soap.training.mulesoft.com/
4 ---
5 [
6   ns0#findFlight: {
7     destination: vars.code as String
8   }
9 ]
```

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.

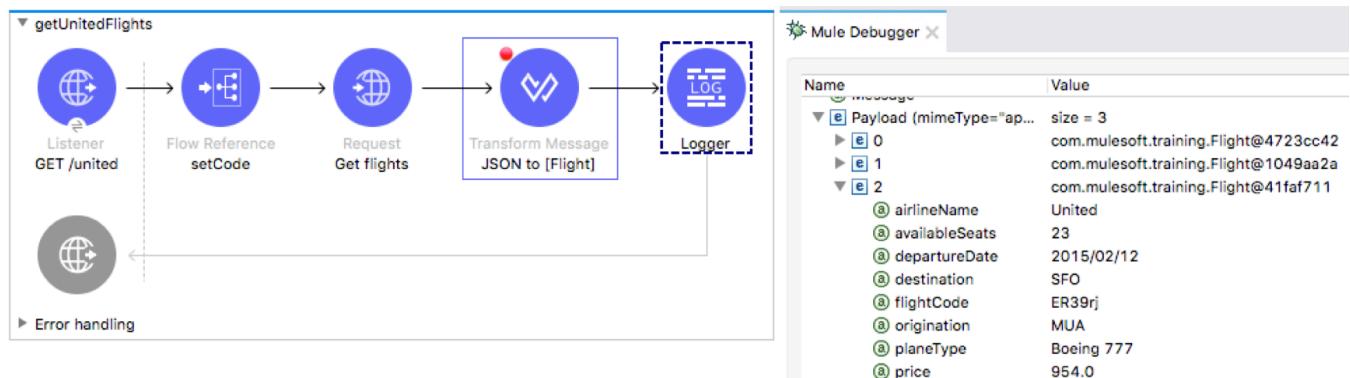
The screenshot shows the response from the Advanced REST Client. The status is '200 OK' and the time is '965.13 ms'. The response body is a JSON object with two 'return' fields, each containing flight details. The second 'return' field has 'destination': 'LAX'.

```
{
  "body": {
    "findFlightResponse": {
      "return": [
        {
          "airlineName": "Delta",
          "code": "A1B2C4",
          "departureDate": "2015/02/11",
          "destination": "LAX",
          "emptySeats": "10",
          "origin": "MUA",
          "planeType": "Boing 737",
          "price": "199.99"
        },
        {
          "airlineName": "Delta",
          "code": "A134DS",
          "departureDate": "2015/04/11",
          "destination": "LAX",
          "emptySeats": "40"
        }
      ]
    }
  }
}
```

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

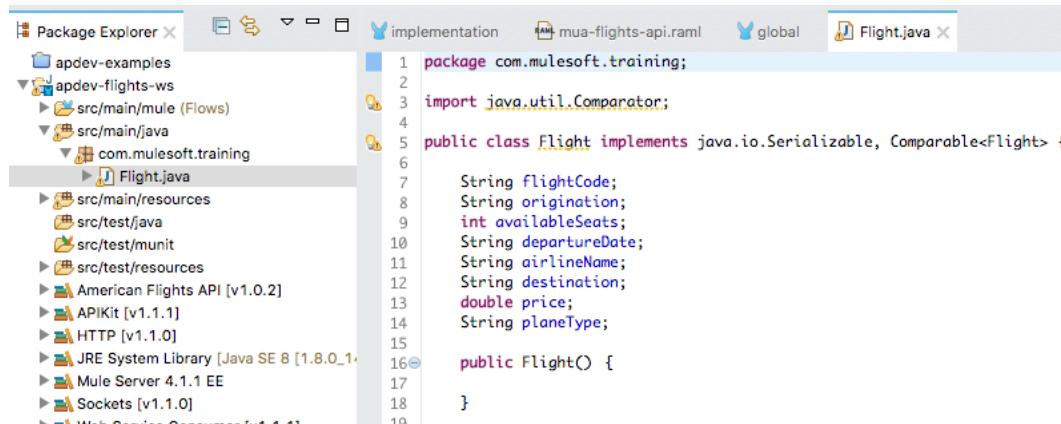
In this walkthrough, you will transform the JSON returned from the American and United web services and the SOAP returned from the Delta web service to the same format. You will:

- 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.

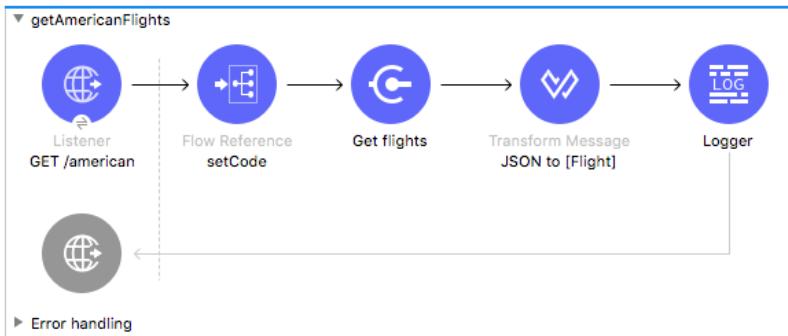


3. Close the file.

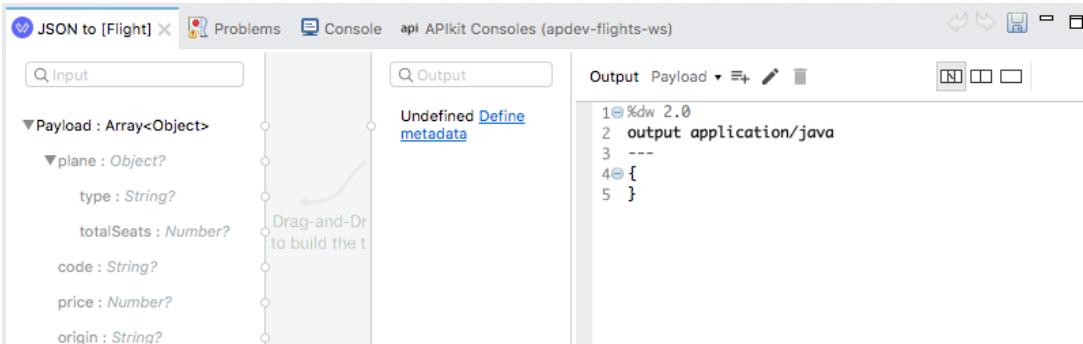
Change the American flow to return Java Flight objects instead of JSON

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].



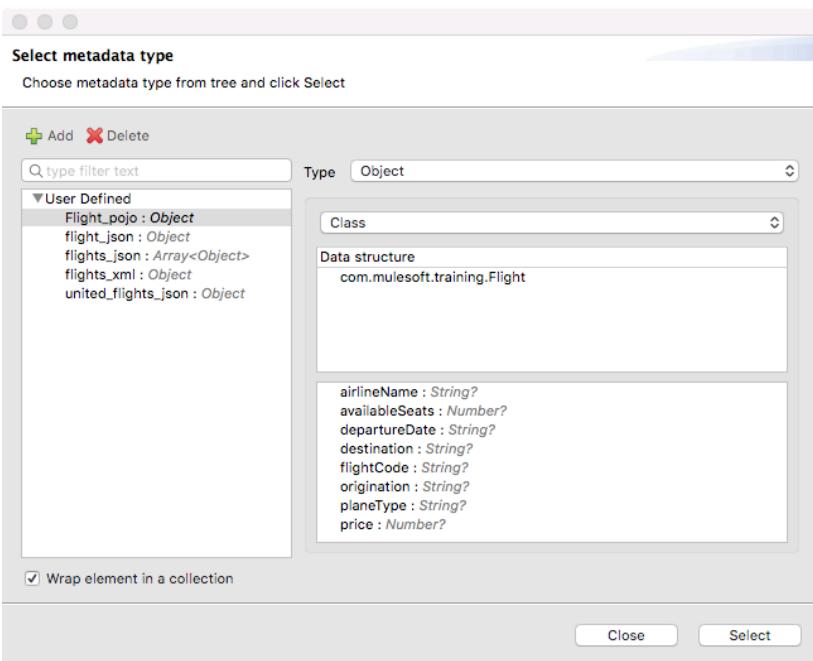
8. In the Transform Message properties view, look at the input section in the Transform Message properties view; you should see metadata already defined.



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.

```

1@ %dw 2.0
2  output application/java
3  ---
4@ payload map ( payload01 , indexOfPayload01 ) ->
5@   airlineName: payload01.airlineName,
6@   availableSeats: payload01.availableSeats,
7@   departureDate: payload01.departureDate,
8@   destination: payload01.destination,
9@   flightCode: payload01.flightCode,
10@  origination: payload01.origination,
11@  planeType: payload01.type,
12@ } as Object {
13@   class : "com.mulesoft.training.Flight"
14@ }

```

14. Double-click the airlineName field in the output section.
15. In the generated DataWeave expression, change the airlineName value from null to "American".

```

1@ %dw 2.0
2  output application/java
3  ---
4@ payload map ( payload01 , indexOfPayload01 ) ->
5@   airlineName: "American",
6@   availableSeats: payload01.availableSeats.

```

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.

200 OK 3370.05 ms DETAILS ▾

```

@srjava.util.ArrayList@Isizexpwsrcom.mulesoft.training.Flight@B@I
availableSeatsDpriceLairlineNameLjava/lang/String;L
departureDateq-Ldestinationq~L
flightCodeq~Loriginationq~L planeTypeq~xp@atAmerican2016-02-11T00:00:00tSF0trree1
093tMUAt
Boeing 737sq~@q~t2016-01-01T00:00:00tSF0teefd1994tMUAt
Boeing 777sq~@q~t2016-02-20T00:00:00tSF0tfee2000tMUAt
Boeing 737sq~@q~t2016-02-01T00:00:00tSF0teefd3000tMUAt
Boeing 737sq~d@q~t2016-01-20T00:00:00tSF0trree4567tMUAt
Boeing 737x

```

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.

Name	Value
datatype	SimpleDatatype{type=org.mule.runtime.core.intern...
Message	
Payload (mimeType="application/json",")	Your payload is not fully displayed as its content is size = 1
Variables	

```
Your payload is not fully displayed as its content is too large to be shown... [  
 {  
 "ID": 5,  
 "code": "rree1093",  
 "price": 142,  
 "departureDate": "2016-02-11T00:00:00",  
 "origin": "MUA",  
 "destination": "SFO",  
 "emptySeats": 1,  
 "plane": {  
 "type": "Boeing 737",  
 "totalSeats": 150  
 }  
 }
```

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

Name	Value
Encoding	UTF-8
Message	
Payload (mimeType="application/java;...")	size = 5
0	com.mulesoft.training.Flight@455541a0
1	com.mulesoft.training.Flight@40f71bf8
2	com.mulesoft.training.Flight@458c44cc
3	com.mulesoft.training.Flight@6567d955
4	com.mulesoft.training.Flight@ad0dc18
airlineName	American
availableSeats	100
departureDate	2016-01-20T00:00:00
destination	SFO

com.mulesoft.training.Flight@455541a0

```
implementation X mua-flights-api.raml global  
getAmericanFlights  
  Listener GET /american --> Flow Reference setCode --> Get flights --> Transform Message JSON to [Flight] --> Logger
```

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

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

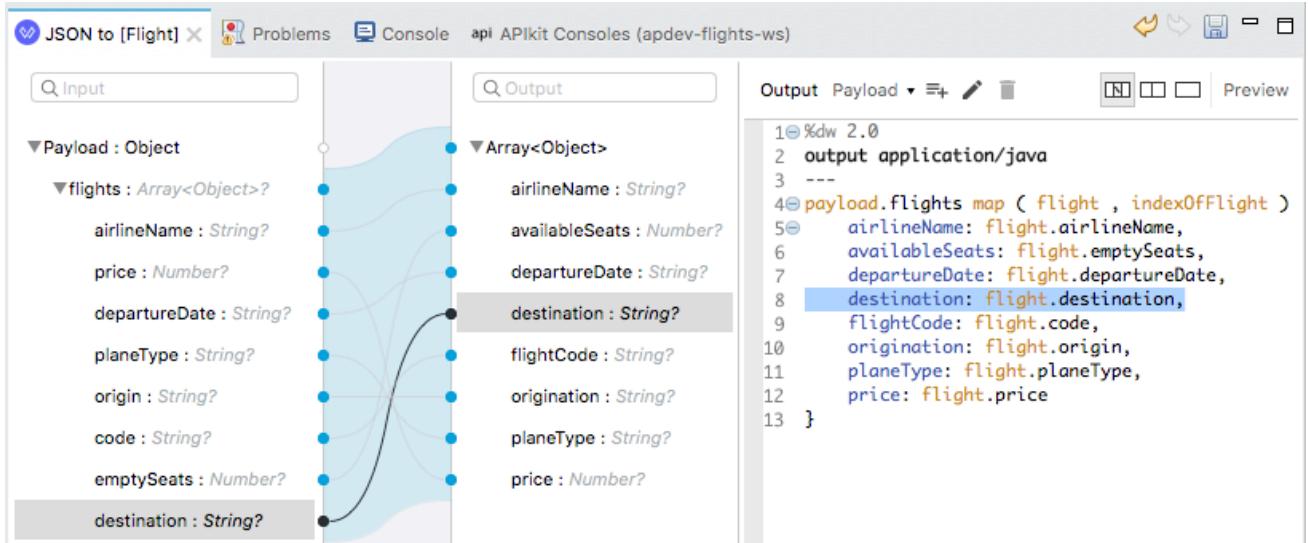
26. Return to geUnitedFlights in implementation.xml.
27. Add a Transform Message component at the end of the flow.
28. Change the name of the Transform Message component to JSON to [Flight].
29. Add a Logger to the end of the flow.



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.

```
%dw 2.0
output application/java
---
{ }
```

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



Note: If you do not see the object type set to com.mulesoft.training.Flight at the end of the DataWeave code, return to the course snippets.txt file and copy the DataWeave code to transform an object to a custom data type, and paste it at the end of the expression.

Note: If you get any DataWeave errors, ignore them.

```

1@ %dw 2.0
2  output application/java
3  ---
4@ payload.flights map ( flight , indexOffFlight ) -> {
5@   airlineName: flight.airlineName,
6@   availableSeats: flight.emptySeats,
7@   departureDate: flight.departureDate,
8@   destination: flight.destination,
9@   flightCode: flight.code,
10@  origination: flight.origin,
11@  planeType: flight.planeType,
12@  price: flight.price
13@ } as Object if
14@   class : "com.mulesoft.training.Flight"
15@ }
16

```

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.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". A table lists various attributes and their values:

Name	Value
Attributes	org.mule.extension.http.api.HttpResponseAttributes
Component Path	getUnitedFlights/processors/2
DataType	SimpleDataType{type=org.mule.runtime.core.internal.streaming.bytes.ManagedC..}
Encoding	UTF-8
Message	
Payload (mimeType="application/json; charset=UTF-8"; encoding=UTF-8)	{"flights": [{"code": "ER38sd", "price": 400, "origin": "MUA", "destination": "SFO", "departureDate": "2015/09/11"}, {"code": "ER39rk", "price": 945, "origin": "MUA", "destination": "SFO", "departureDate": "2015/09/11"}, {"code": "ER39rj", "price": 954, "origin": "MUA", "destination": "SFO", "departureDate": "2015/02/12"}]}
Variables	size = 1

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

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". A table lists the payload as a collection of three flight objects:

Name	Value
message	
Payload (mimeType="application/java; charset=UTF-8"; encoding=UTF-8)	size = 3
0	com.mulesoft.training.Flight@48a6dd7d
1	com.mulesoft.training.Flight@79abe118
2	com.mulesoft.training.Flight@383fdafb
airlineName	United
availableSeats	23
departureDate	2015/02/12
destination	SFO
flightCode	ER39rj
origination	MUA
planeType	Boeing 777
price	954.0
Variables	size = 1

Below the debugger, the "implementation" perspective is shown with the "mua-flights-api.raml" file open. The flow diagram for "getUnitedFlights" is displayed:

```
graph LR; Listener[Listener GET /united] --> FlowRef[Flow Reference]; FlowRef --> Request[Request Get flights]; Request --> Transform[Transform Message JSON to [Flight]]; Transform --> Logger[Logger]
```

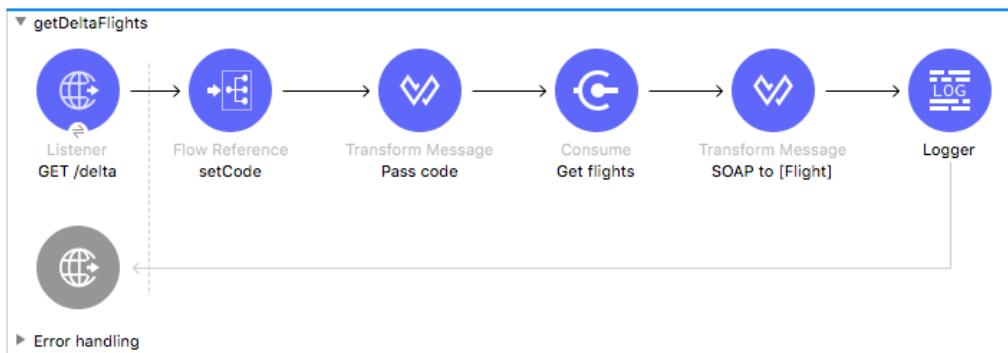
41. Step through the rest of the application and switch perspectives.

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.

Note: If you get a DataWeave error and cannot drag and drop the fields, delete all the existing DataWeave code, return to the course snippets.txt file and copy the DataWeave code for the Delta transformation, and then return to Anypoint Studio and paste it in the expression 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.

The screenshot shows the Mule Debugger interface with a table titled "Name" and "Value". The payload is a SoapOutputPayload object containing attachments (empty), body (TypedValue object), dataType (SimpleDataType), length (-1), serialVersionUID (-2533879516750283994), value (ManagedCursorStreamProvider), headers (empty), and Variables (size = 1).

Name	Value
Payload (mimeType="application/xml")	org.mule.runtime.extension.api.soap.SapOutputPayload@7e25d65
attachments	{}
body	org.mule.runtime.api.metadata.TypedValue@dd7ca55
dataType	SimpleDataType{type=java.io.InputStream, mimeType='application/xml'}
length	-1
serialVersionUID	-2533879516750283994
value	org.mule.runtime.core.internal.streaming.bytes.ManagedCursorStreamProvider
headers	{}
Variables	size = 1

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

The screenshot shows the Mule Debugger interface with a table titled "Name" and "Value". The payload is a collection of three Flight objects (size = 3). The first object is com.mulesoft.training.Flight@5d0b318c, the second is com.mulesoft.training.Flight@6797cda6, and the third is com.mulesoft.training.Flight@42da10ff. Below the table, the implementation perspective shows a flow named "getDeltaFlights" with the following steps: Listener GET /delta, Flow Reference setCode, Transform Message Pass code, Consume Get flights, Transform Message SOAP to [Flight], and a final step labeled "Logger".

Name	Value
Payload (mimeType="application/java; charset=UTF-8", encoding="UTF-8")	size = 3
0	com.mulesoft.training.Flight@5d0b318c
1	com.mulesoft.training.Flight@6797cda6
2	com.mulesoft.training.Flight@42da10ff
Variables	size = 1

implementation X mua-flights-api.raml global

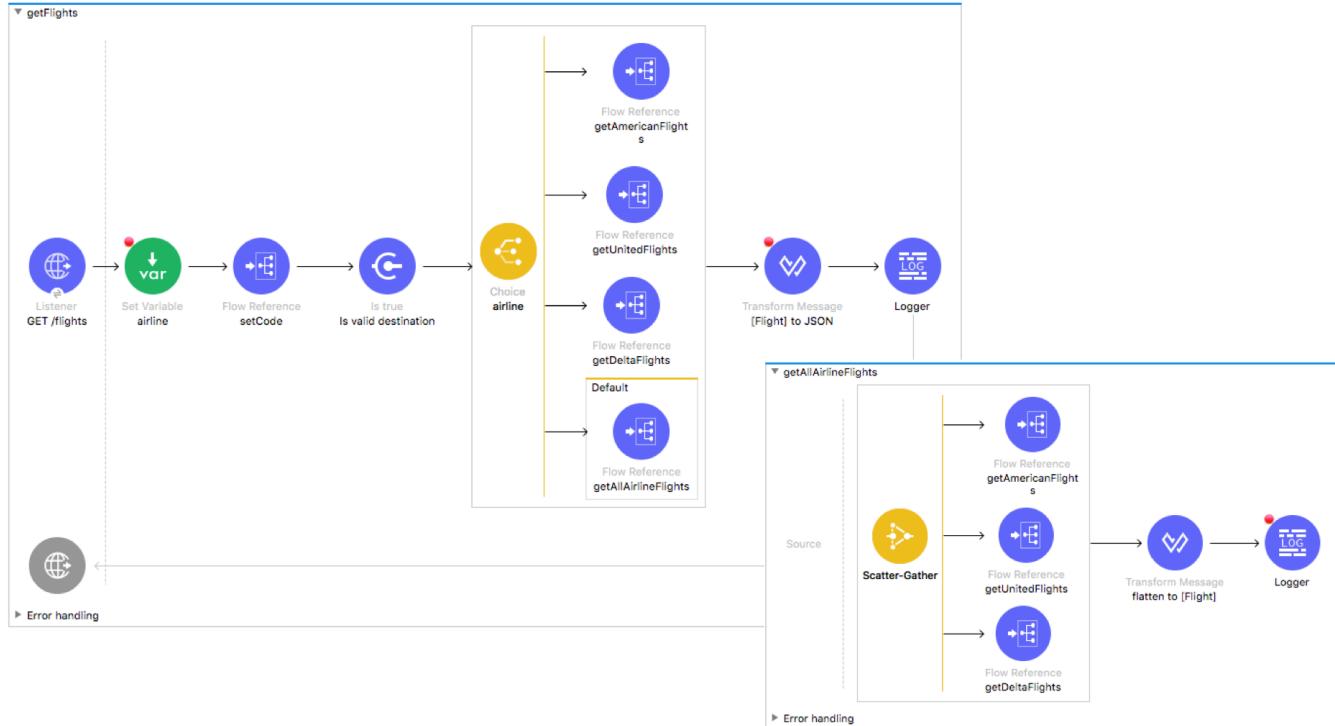
Error handling

getDeltaFlights

- Listener GET /delta
- Flow Reference setCode
- Transform Message Pass code
- Consume Get flights
- Transform Message SOAP to [Flight]
- Logger

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

Module 9: Controlling Event Flow



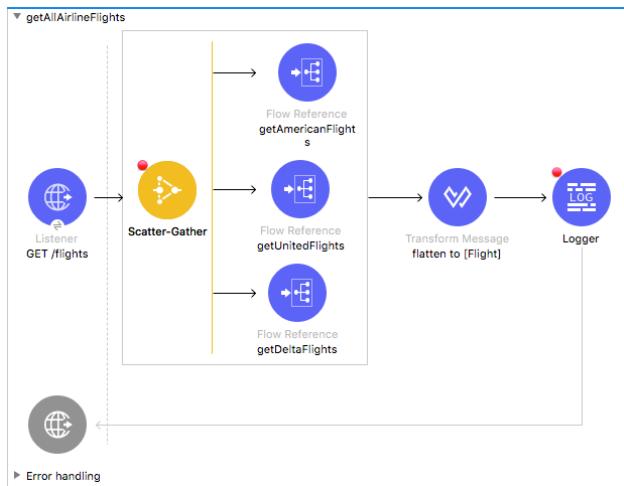
At the end of this module, you should be able to:

- Multicast events.
- Route events based on conditions.
- Validate events.

Walkthrough 9-1: Multicast an event

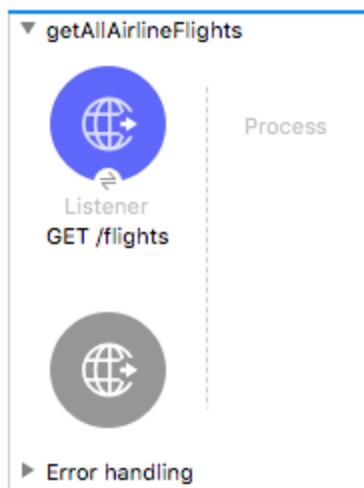
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.



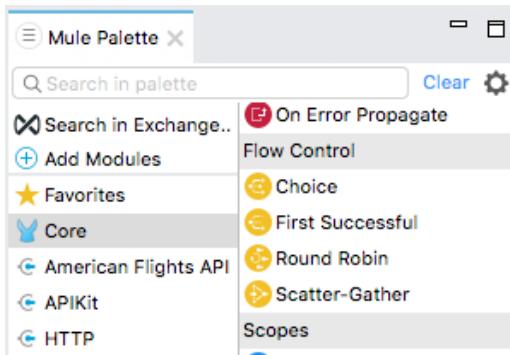
Create a new flow

1. Return to implementation.xml.
2. From the Mule Palette, drag an HTTP Listener and drop it at the top of the canvas.
3. Change the flow name to getAllAirlineFlights.
4. In the Listener properties view, set the display name to GET /flights.
5. Set the connector configuration to the existing HTTP_Listener_config.
6. Set the path to /flights and the allowed methods to GET.



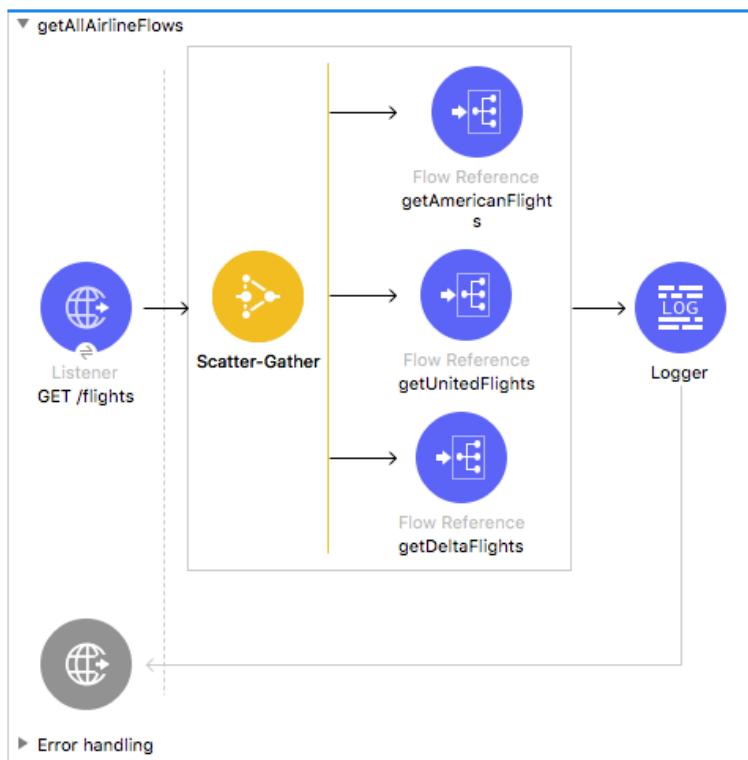
Browse the flow control elements in the Mule Palette

7. In the Core section of the Mule Palette, locate the Flow Control elements.



Add a Scatter-Gather to call all three airline services

8. Drag a Scatter-Gather flow control element from the Mule Palette and drop it in the process section of getAllAirlineFlights.
9. Add three Flow Reference components to the Scatter-Gather router.
10. In the first Flow Reference properties view, set the flow name to getAmericanFlights.
11. Set the flow name of the second Flow Reference to getUnitedFlights.
12. Set the flow name of the third Flow Reference to getDeltaFlights.
13. Add a Logger after the Scatter-Gather.



Review the metadata for the Scatter-Gather output

14. In the Logger properties view, explore the input payload structure in the DataSense Explorer.

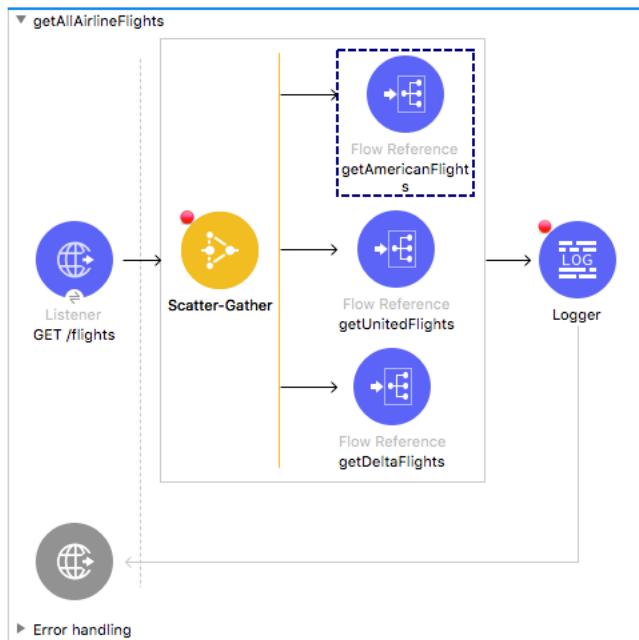
The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. A search bar at the top contains a warning icon and the placeholder 'type filter text'. Below the search bar, the tree view displays the following structure:

- Mule Message**
 - Payload**
 - Object : Object**
 - 0 : Object**
 - 1 : Object**
 - payload : Array<Object>**
 - attributes : Object**
 - 2 : Object**
 - payload : Array<Object>**
 - airline : String?**
 - flightCode : String?**
 - fromAirportCode : String?**
 - toAirportCode : String?**
 - departureDate : String?**
 - emptySeats : Number?**
 - totalSeats : Number?**
 - price : Number?**
 - planeType : String?**
 - attributes : Object**
 - Attributes**
 - Void : Void**
 - Variables**
 - code**
 - Nothing : Nothing**

Debug the application

15. Add a breakpoint to the Scatter-Gather.
16. Add a breakpoint to the Logger.
17. Save the file to redeploy the project in debug mode.
18. In Advanced REST Client, change the URL to make a request to <http://localhost/flights>.

19. In the Mule Debugger, step through the application; you should step through each of the airline flows.



20. Stop at the Logger after the Scatter-Gather and explore the payload.

A screenshot of the Mule Debugger interface. The top window is titled "Mule Debugger" and shows a table of message properties. The "Payload (mimeType="*/*")" row has a value of "size = 3", with three sub-items labeled 0, 1, and 2. The bottom window is titled "implementation" and shows the Mule flow. It includes a "Listener GET /flights", a "Scatter-Gather" component, a "Flow Reference getUnitedFlights", and a "Logger" component. A blue arrow points from the "getUnitedFlights" flow reference to the "Logger" component.

Name	Value
Message	
Payload (mimeType="*/*")	size = 3
▶ e 0	0=
▶ e 1	1=
▶ e 2	2=
size = 3	

21. Drill-down into one of the objects in the payload.

Name	Value
⑧ Message	
▼ ⑨ Payload (mimeType="*/*")	size = 3
► ⑩ 0	0=
► ⑪ 1	1=
▼ ⑫ 2	2=
⑬ key	2
▼ ⑭ value	
⑮ exceptionPayload	null
► ⑯ inboundAttachments	{}
► ⑰ inboundMap	org.mule.runtime.api.util.CaseInsensitiveM
► ⑱ logger	org.apache.logging.slf4j.Log4jLogger@451
⑲ NOT_SET	<not set>
► ⑳ outboundAttachments	{}
► ㉑ outboundMap	org.mule.runtime.api.util.CaseInsensitiveM
⑲ serialVersionUID	1541720810851984845
► ㉓ typedAttributes	org.mule.runtime.api.metadata.TypedValue
▼ ㉔ typedValue	org.mule.runtime.api.metadata.TypedValue
► ㉕ dataType	CollectionDataType(type=java.util.ArrayList
⑲ length	-1
⑲ serialVersionUID	-2533879516750283994
▼ ㉖ value	[com.mulesoft.training.Flight@36875461,
► ㉗ 0	com.mulesoft.training.Flight@36875461
► ㉘ 1	com.mulesoft.training.Flight@1324ac7a
► ㉙ 2	com.mulesoft.training.Flight@224bf009

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

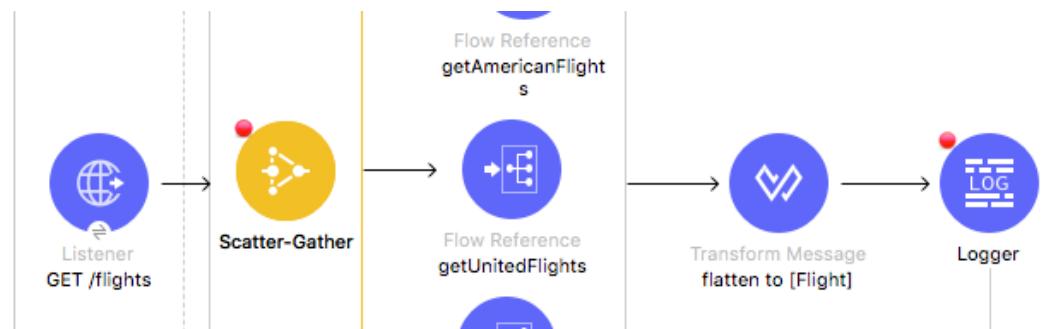
23. Return to Advanced REST Client and review the response; although the result is Java and you see a lot of characters, you should also see the Java for flight data for the three airlines.

Flatten the combined results

24. Return to Anypoint Studio.

25. In getAllAirlineFlights, add a Transform Message component before the Logger.

26. Change the display name to flatten to [Flight].



27. In the input section of the Transform Message properties view, review the payload data structure.

The screenshot shows the 'Transform Message' properties view in Mule Studio. The 'Input' tab is selected. The payload structure is defined as follows:

```
Payload : Object
  ▶ 0 : Object
  ▶ 1 : Object
  ▶ 2 : Object
    ▼ payload : Array<Object>
      airlineName : String?
      availableSeats : Number?
      departureDate : String?
      destination : String?
      flightCode : String?
      origination : String?
```

28. In the expression section, use the DataWeave flatten function to flatten the collection of objects into a single collection.

```
1 %dw 2.0
2   output application/java
3   ---
4   flatten(payload..payload)
```

Debug the application

29. Save the file to redeploy the project.
30. In Advanced REST Client, make the same request to <http://localhost:8081/flights>.

31. In the Mule Debugger, press Resume until you are stopped at the Logger at the end of the Scatter-Gather; you should see the payload is now one ArrayList of Flights.

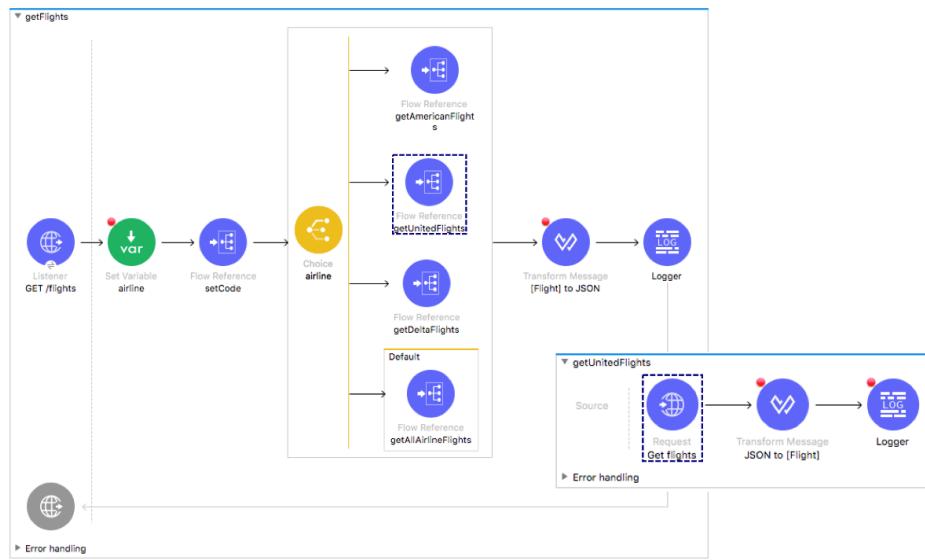
The screenshot shows the Mule Debugger interface. At the top, it says "Mule Debugger". Below that is a table with two columns: "Name" and "Value". The "Name" column contains "Payload (mimeType="application/java; charset=UTF-8", encoding="UTF-8") size = 11" followed by a list of 11 items labeled "e 0" through "e 9". The "Value" column lists 11 Flight objects from com.mulesoft.training. The bottom part of the screenshot shows the Mule flow diagram. It starts with a "Listener GET /flights" component, followed by a "Scatter-Gather" component. A "Flow Reference getUnitedFlights" leads to a "Transform Message flatten to [Flight]" component, which then leads to a "Logger" component. The "Logger" component is highlighted with a dashed blue border.

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

Walkthrough 9-2: Route events based on conditions

In this walkthrough, you create a flow to route events to either the American, United, Delta, or get all airline flows based on the value of an airline query parameter. You will:

- Use a Choice router.
- Use DataWeave expressions to set the router paths.
- Route all flight requests through the router.



Look at possible airline values specified in the API

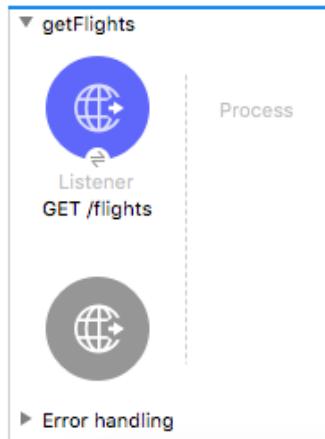
1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Open mua-flights-api.raml in src/main/resources/api.
3. Locate the airline query parameter and its possible values.

```
airline:  
  displayName: Airline  
  required: false  
  enum:  
    - united  
    - delta  
    - american
```

Create a new flow

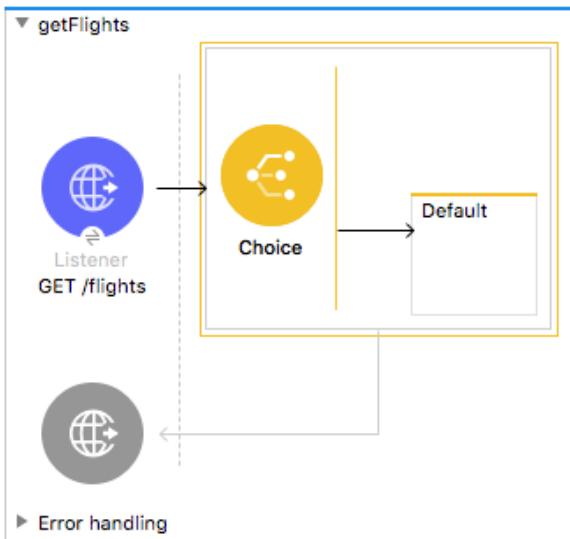
4. Return to implementation.xml.
5. Drag a Flow scope from the Mule Palette and drop it at the top of the canvas above all the other flows.
6. Change the name of the flow to getFlights.

7. Move the GET /flights HTTP Listener from getAllAirlineFlights to the source section of getFlights.

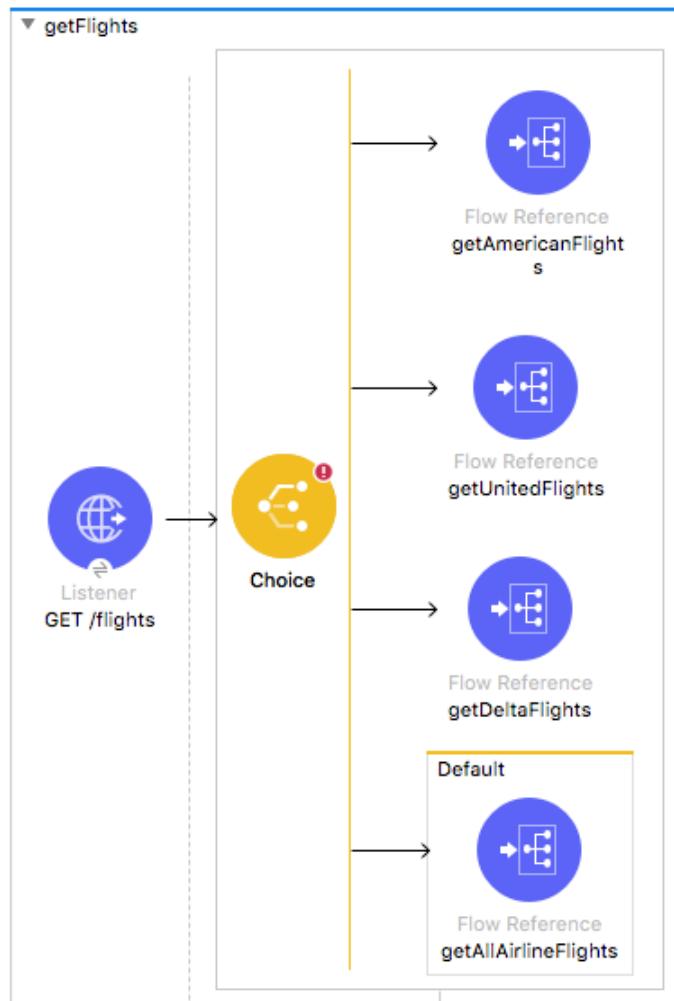


Add a Choice router

8. Drag a Choice flow control element from the Mule Palette and drop it in process section of getFlights.

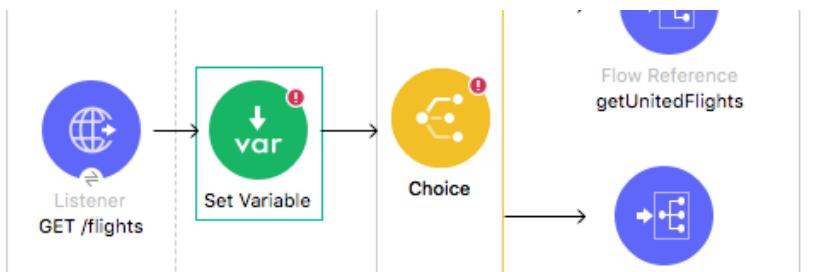


9. Add three Flow Reference components to the Choice router.
10. Add a Flow Reference component to the default branch of the router.
11. In the first Flow Reference properties view, set the flow name to getAmericanFlights.
12. Set the flow names for the other two Flow References to getUnitedFlights and getDeltaFlights.
13. For the Flow Reference in the default branch, set the flow name to getAllAirlineFlights.



Store the airline query parameter in a variable

14. Add a Set Variable transformer before the Choice router.



15. In the Set Variable properties view, set the display name and name to airline.

16. Set the value to a query parameter called airline.

```
##[attributes.queryParams.airline]
```

The screenshot shows the 'Set Variable' properties view in Mule Studio. The 'General' tab is selected. The 'Display Name' field contains 'airline'. The 'Name' field also contains 'airline'. The 'Value' field contains the expression '#[attributes.queryParams.airline]'. A status message at the top right says 'There are no errors.'

Configure the Choice router

17. In the Choice properties view, set the display name to airline.

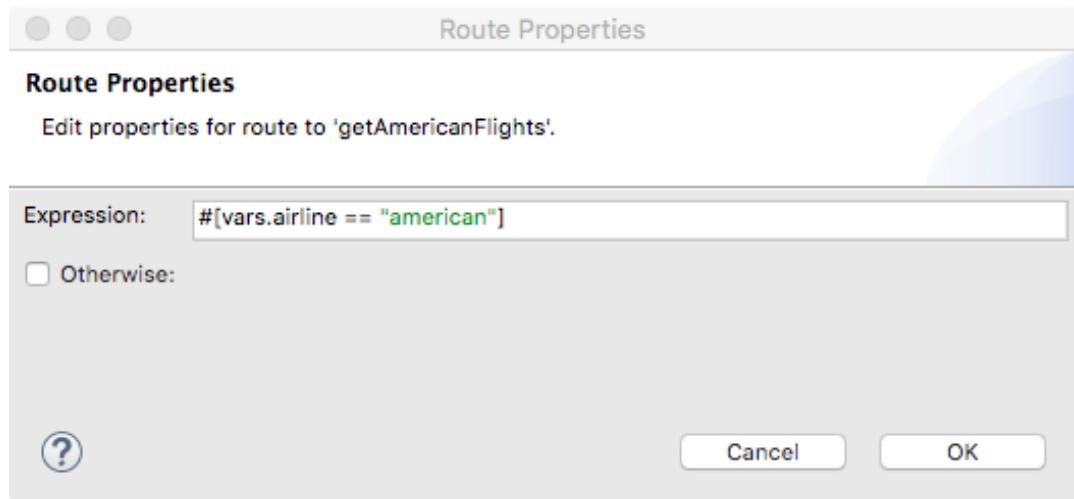
The screenshot shows the 'Choice' properties view in Mule Studio. The 'Choice Properties' tab is selected. The 'Display Name' field contains 'airline'. A warning message at the top right says 'Required expression is not defined in when.' Below the display name, there is a section for 'Business Events' with a note: 'Enabling this option will activate event tracking feature for this element and its children.' There is a checkbox labeled 'Enable default events tracking' which is unchecked. At the bottom, there is a table for defining routes:

When	Route Message to
	→ getAmericanFlights → getUnitedFlights → getDeltaFlights → getAllAirlineFlights
Default	

18. In the Choice properties view, double-click the getAmericanFlights route.

19. In the Route Properties dialog box, add an expression to check if the airline variable is equal to american and click OK.

```
##[vars.airline == "american"]
```



20. Set a similar expression for the United route, routing to it when vars.airline is equal to united.
 21. Set a similar expression for the Delta route, routing to it when vars.airline is equal to delta.

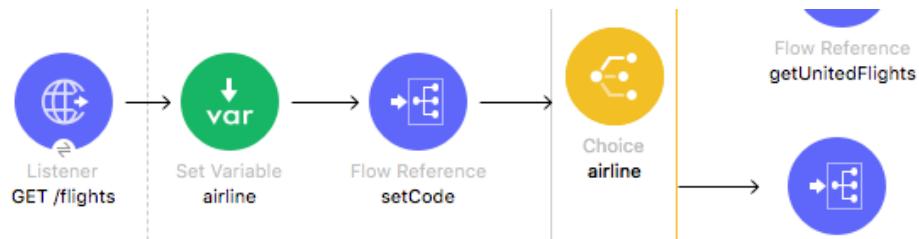
The screenshot shows the Anypoint Studio interface with the 'Choice' tab selected. The title bar includes 'Problems', 'Console', and 'api APIkit Consoles (apdev-flights-ws)'. The main area displays the 'Choice Properties' table:

	When	Route Message to
Metadata	##[vars.airline == "american"]	-> getAmericanFlights
Notes	##[vars.airline == "united"]	-> getUnitedFlights
	##[vars.airline == "delta"]	-> getDeltaFlights
	Default	-> getAllAirlineFlights

A green checkmark icon with the text 'There are no errors.' is displayed above the table.

Route all requests through the router

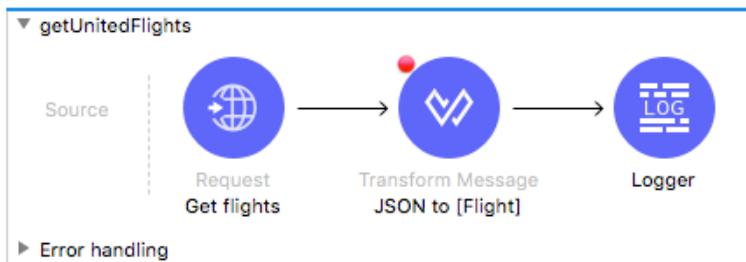
22. Add a Flow Reference to the flow before the Choice router.
 23. Set the flow name to setCode.



24. In getAmericanFlights, delete the HTTP Listener and the setCode Flow Reference.



25. In getUnitedFlights, delete the HTTP Listener and the setCode Flow Reference.



26. In getDeltaFlights, delete the HTTP Listener and the setCode Flow Reference.



Return JSON from the flow

27. Add a Transform Message component after the Choice router.

28. Change its display name to [Flight] to JSON.

29. Add a Logger at the end of the flow.



30. In the Transform Message properties view, set the output type to JSON and the output to payload.

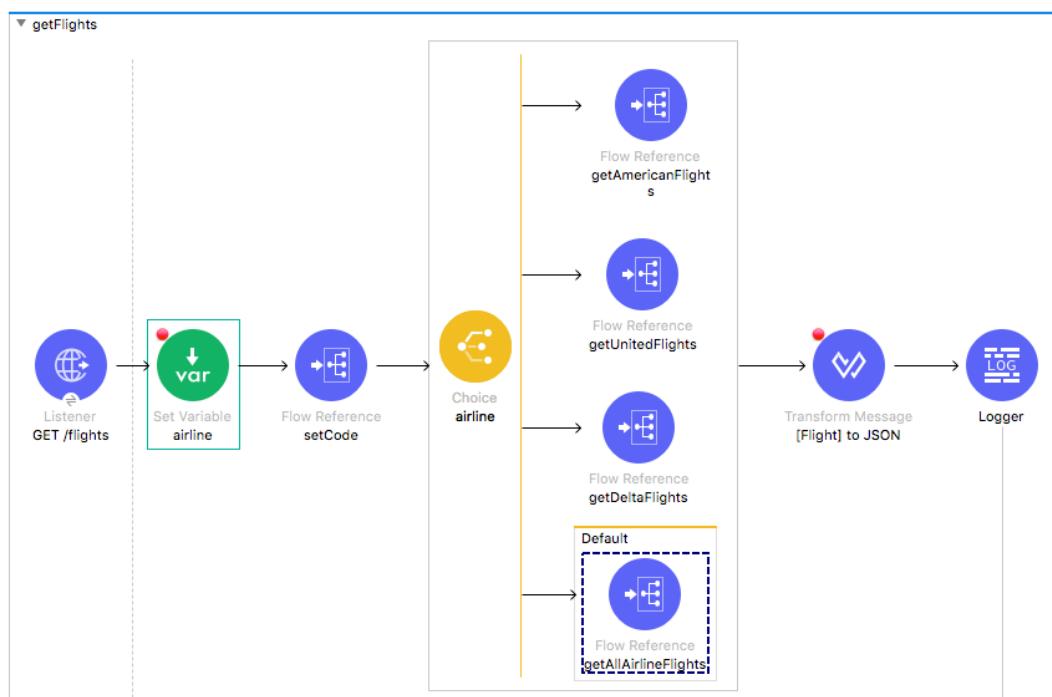
```

1 %dw 2.0
2 output application/json
3 ---
4 payload

```

Debug the application

31. Add a breakpoint to the Set Variable transformer at the beginning of getFlights.
32. Add a breakpoint to the Transform Message component after the Choice router.
33. Save the file to redeploy the project in debug mode.
34. In Advanced REST Client, make a request to <http://localhost:8081/flights>.
35. In the Mule Debugger, step through the application; you should see the Choice router pass the event to the default branch.



36. Resume to the Transform Message component after the Choice router; the payload should be an ArrayList of Flight objects
37. Step to the Logger; the payload should be JSON.
38. Step to the end of the application.

39. Return to Advanced REST Client; you should see American, United, and Delta flights to SFO (the default airport code).

Method Request URL
GET http://localhost:8081/flights

Parameters ▾

200 OK 94251.28 ms DETAILS ▾

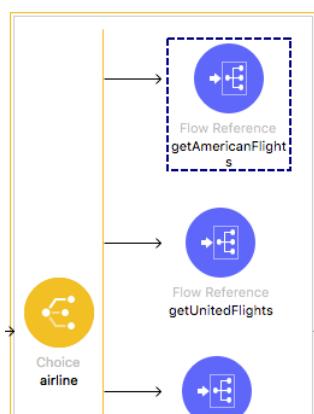
[Array[11]]

```
-0: {  
    "price": 142,  
    "flightCode": "rree1093",  
    "availableSeats": 1,  
    "planeType": "Boeing 737",  
    "departureDate": "2016-02-11T00:00:00",  
    "origination": "MUA",  
    "airlineName": "American",  
    "destination": "SFO"  
},  
-1: { ... }  
-2: { ... }  
-3: { ... }  
-4: { ... }  
-5: {  
    "price": 400,  
    "flightCode": "ER38sd",  
    "availableSeats": 0,  
    "planeType": "Boeing 737",  
    "departureDate": "2015/03/20",  
    "origination": "MUA",  
    "airlineName": "United",  
    "destination": "SFO"
```

40. Add an airline query parameter set to american and send the request:

<http://localhost:8081/flights?airline=american>.

41. In the Mule Debugger, step through the rest of the application; you should see the message passed to getAmericanFlights and then back to getFlights.



42. Return to Advanced REST Client; you should see only American flights to SFO returned.

```
200 OK 19641.09 ms

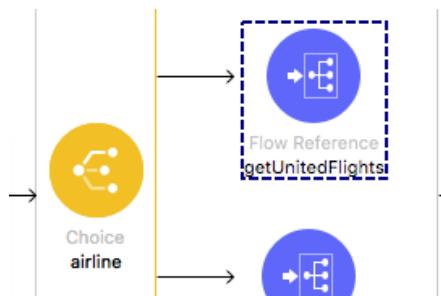
[
{
  "price": 142.0,
  "flightCode": "rree1093",
  "availableSeats": 1,
  "planeType": "Boeing 737",
  "departureDate": "2016-02-11T00:00:00",
  "origination": "MUA",
  "airlineName": "American",
  "destination": "SFO"
},
```

43. Change the airline to delta and add a second query parameter code set to LAX:

<http://localhost:8081/flights?airline=united&code=LAX>.

44. Send the request.

45. In the Mule Debugger, step through the application; the message should be routed to the United branch.



46. Return to Advanced REST Client; you should see only United flights to LAX are returned.

```
200 OK 47214.31 ms

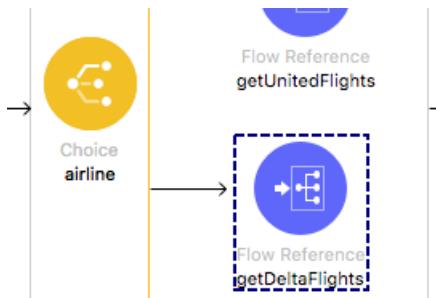
[
{
  "price": 345.99,
  "flightCode": "ER45if",
  "availableSeats": 52,
  "planeType": "Boeing 737",
  "departureDate": "2015/02/11",
  "origination": "MUA",
  "airlineName": "United",
  "destination": "LAX"
},
```

47. Change the airline to delta and the code to PDX:

<http://localhost:8081/flights?airline=united&code=PDX>.

48. Send the request.

49. In the Mule Debugger, step through the application; the message should be routed to the Delta branch.



50. Return to Advanced REST Client; you should see only Delta flights to PDX are returned.

```
200 OK 40936.12 ms

[{"id": 1, "airline": "Delta", "origin": "MUA", "destination": "PDX", "price": 958.0, "flightCode": "A1FGF4", "availableSeats": 80, "planeType": "Boing 777", "departureDate": "2015/02/13"}]
```

A screenshot of the Advanced REST Client interface. At the top, there's a green button labeled '200 OK' and '40936.12 ms'. Below the status bar, there are several icons: a square, a play/pause button, a refresh/circular arrow, a double-headed arrow, and a list icon. The main content area displays a JSON array with one element. The element contains the following fields: 'id': 1, 'airline': 'Delta', 'origin': 'MUA', 'destination': 'PDX', 'price': 958.0, 'flightCode': 'A1FGF4', 'availableSeats': 80, 'planeType': 'Boing 777', and 'departureDate': '2015/02/13'.

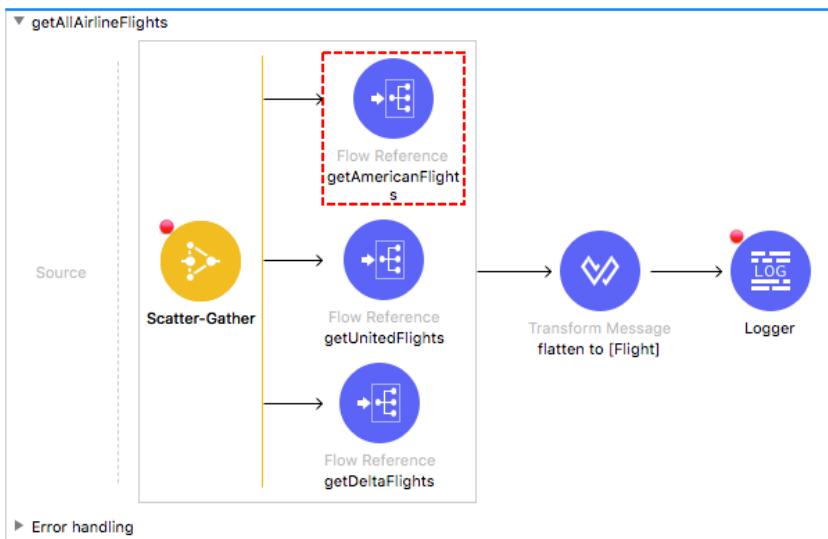
*Note: This JSON structure does not match that specified in the API: mua-flights-api.raml in src/main/resources/api. The data **should**, of course, be transformed so the response from the API matches this specification – but we are going to hold off doing that right now so that the scenario stays simpler for the error handling module (there is one less error to handle at the beginning).*

Test the application with a destination that has no flights

51. Remove the airline parameter and set the code to FOO:

<http://localhost:8081/flights?code=FOO>.

52. In the Mule Debugger, step through the application; you should see multiple errors.



53. Return to Advanced REST Client; you should get a 500 Server Response and an exception.

Method Request URL
GET <http://localhost:8081/flights?code=FOO> **SEND** **⋮**

Parameters **⋮**

500 Server Error 43387.29 ms DETAILS **⋮**

Exception(s) were found for route(s):

0: org.mule.extension.http.api.request.validator.ResponseValidatorTypedException
n: HTTP GET on resource '<http://training4-american-api-mule.cloudhub.io:80/flights>'
failed: bad request (400).
1: org.mule.runtime.core.api.expression.ExpressionRuntimeException: "You called
the function 'Value Selection' with these arguments."