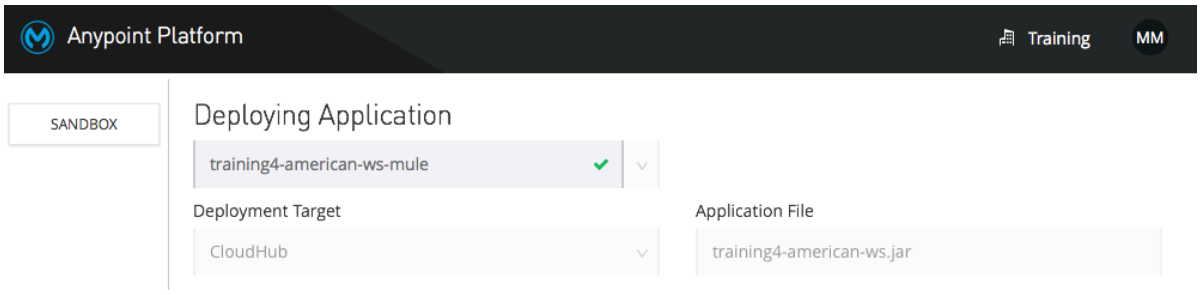
4. At the top of the Anypoint Platform dialog box, set the application name to training4-american-ws-*{your-lastname}* so it is a unique value.

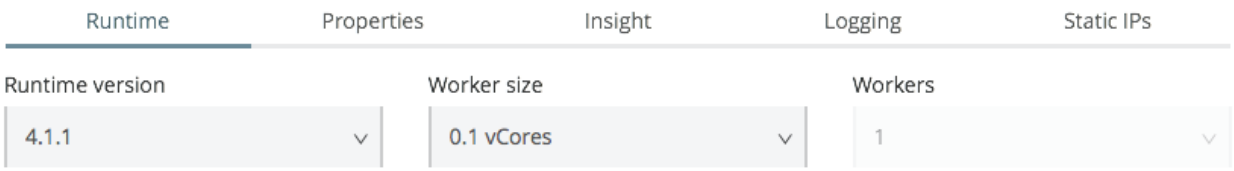
   Note: This name will be part of the URL used to access the application on CloudHub. It must be unique across all applications on CloudHub. The availability of the domain is instantly checked and you will get a green check mark if it is available.



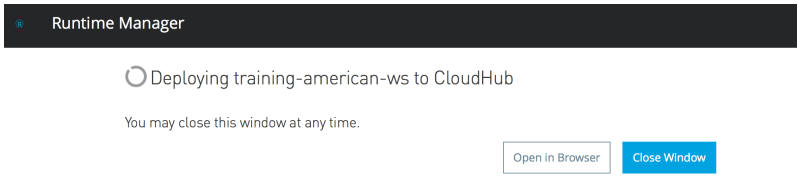
5. Make sure the runtime version is set to the version your project is using.

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

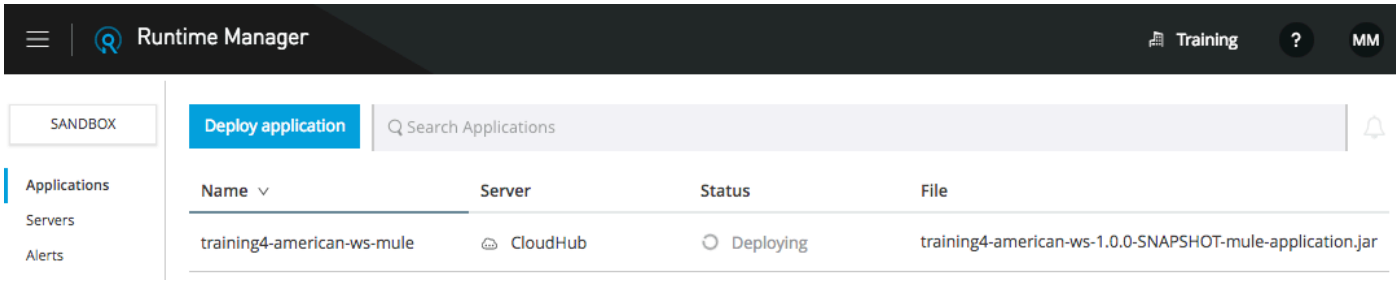
6. Make sure the worker size to 0.1 vCores.



7. Click the Deploy Application button.
8. Click the Open in Browser button.

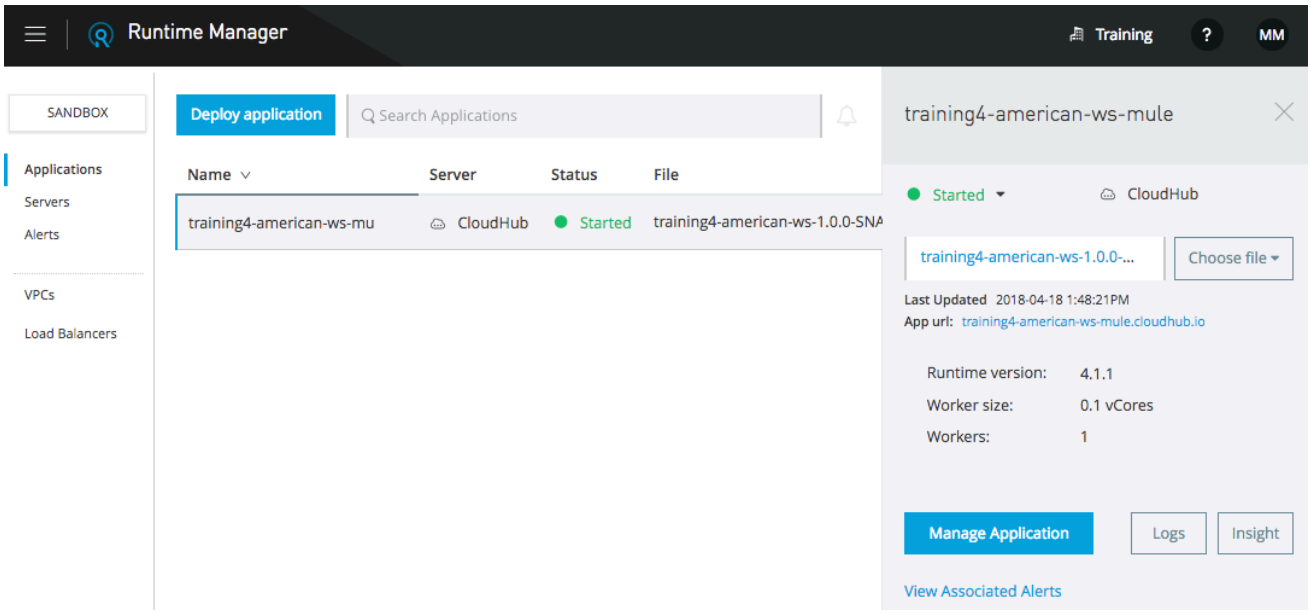


9. In the Anypoint Platform browser window that opens, locate the status of your deployment in the Runtime Manager.

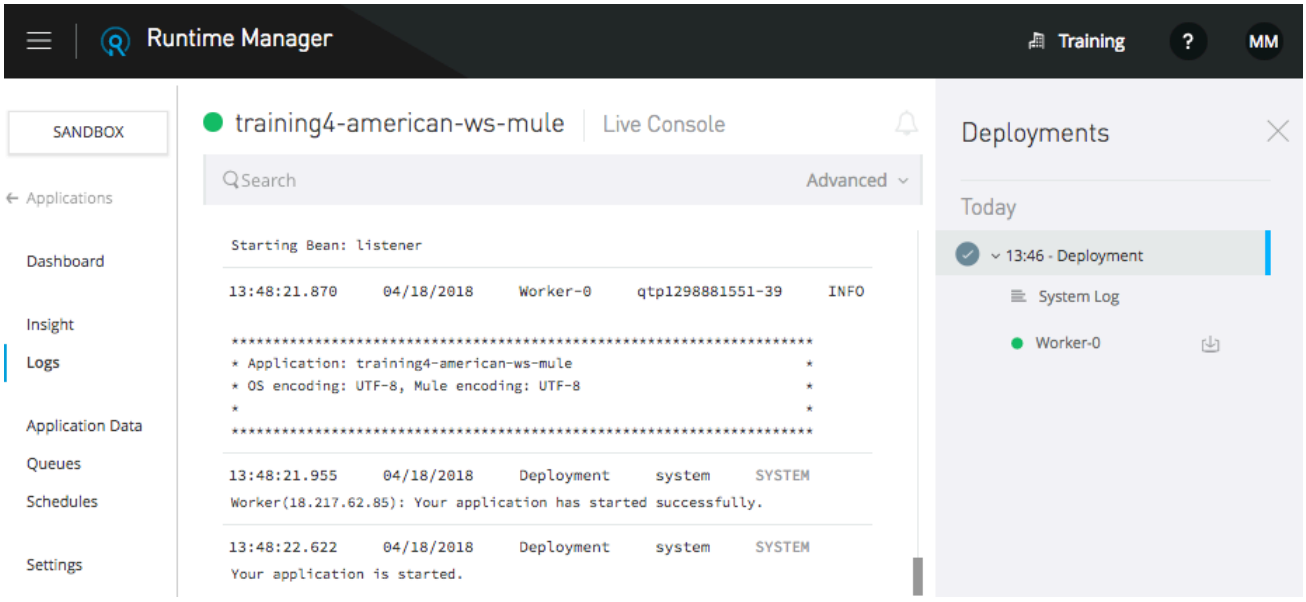


Watch the logs and wait for the application to start

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.



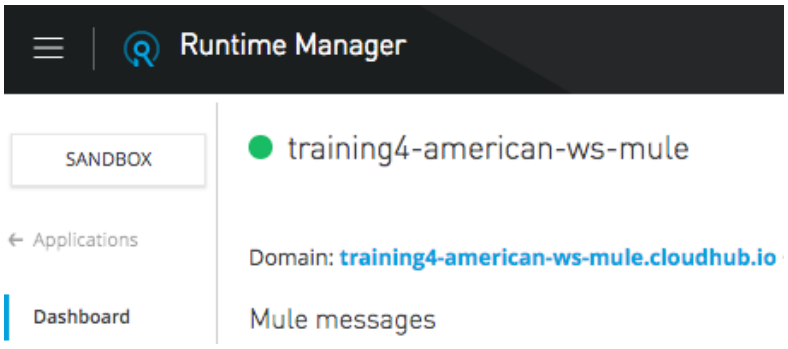
11. Click the Logs button.
12. Watch the logs as the application is deployed.
13. Wait until the application starts (or fails to start).



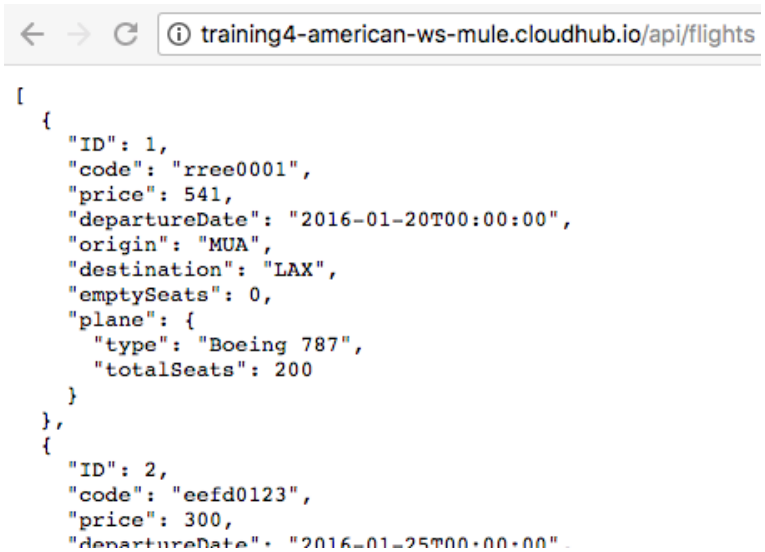
Note: If your application did not successfully deploy, read the logs to help figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them, and then redeploy.

Test the application

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



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*}.cloudhub.io/api/flights; you should see the flights data.

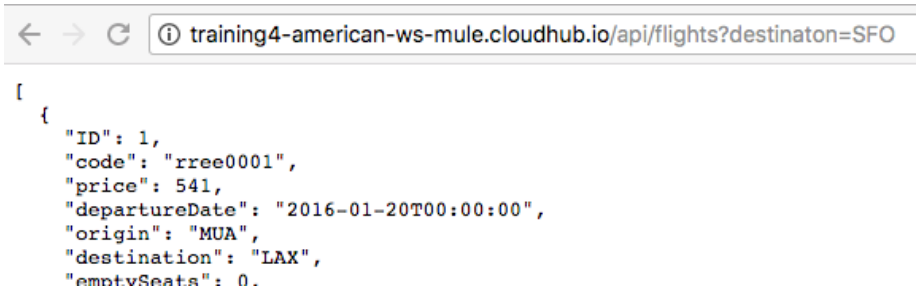


Note: If you are using the local Derby database, your application will not return results when deployed to CloudHub. You will update the application with a version using the MySQL database in the next section, so it works.

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.



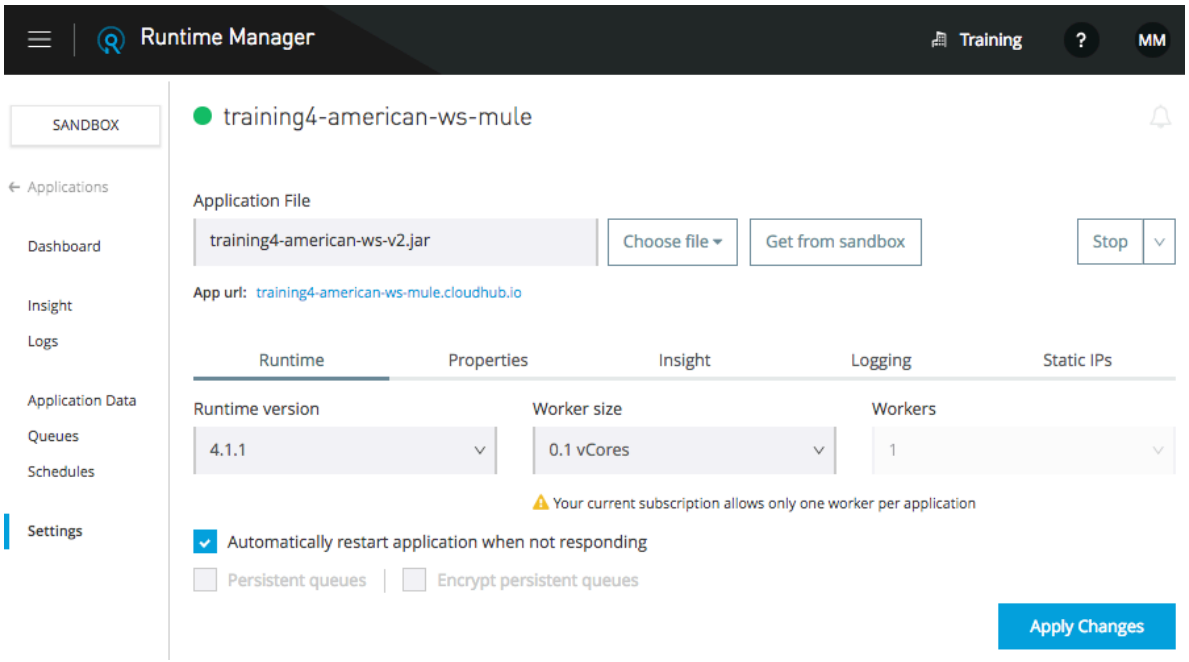
20. Leave this browser tab open.

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.



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

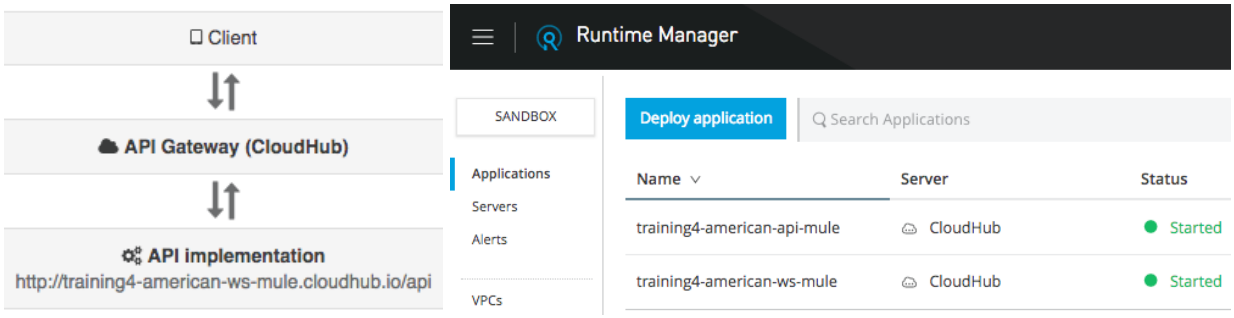
29. Return to the browser tab with a request to the API implementation on CloudHub with a destination of SFO and refresh it; you should now get only flights to SFO.

30. Close this browser tab.

Walkthrough 5-2: Create and deploy an API proxy

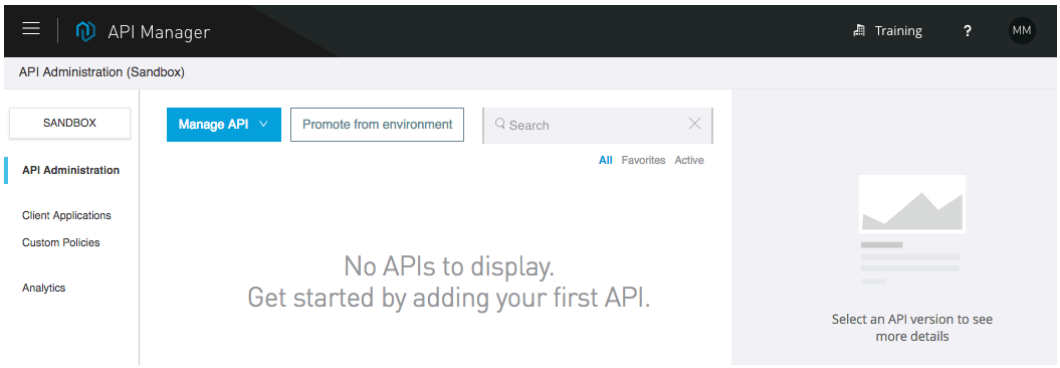
In this walkthrough, you create and deploy an API proxy for your API implementation on CloudHub. You will:

- 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.

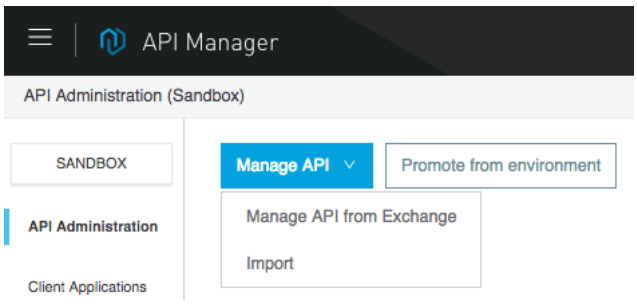


Create and deploy a proxy application

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

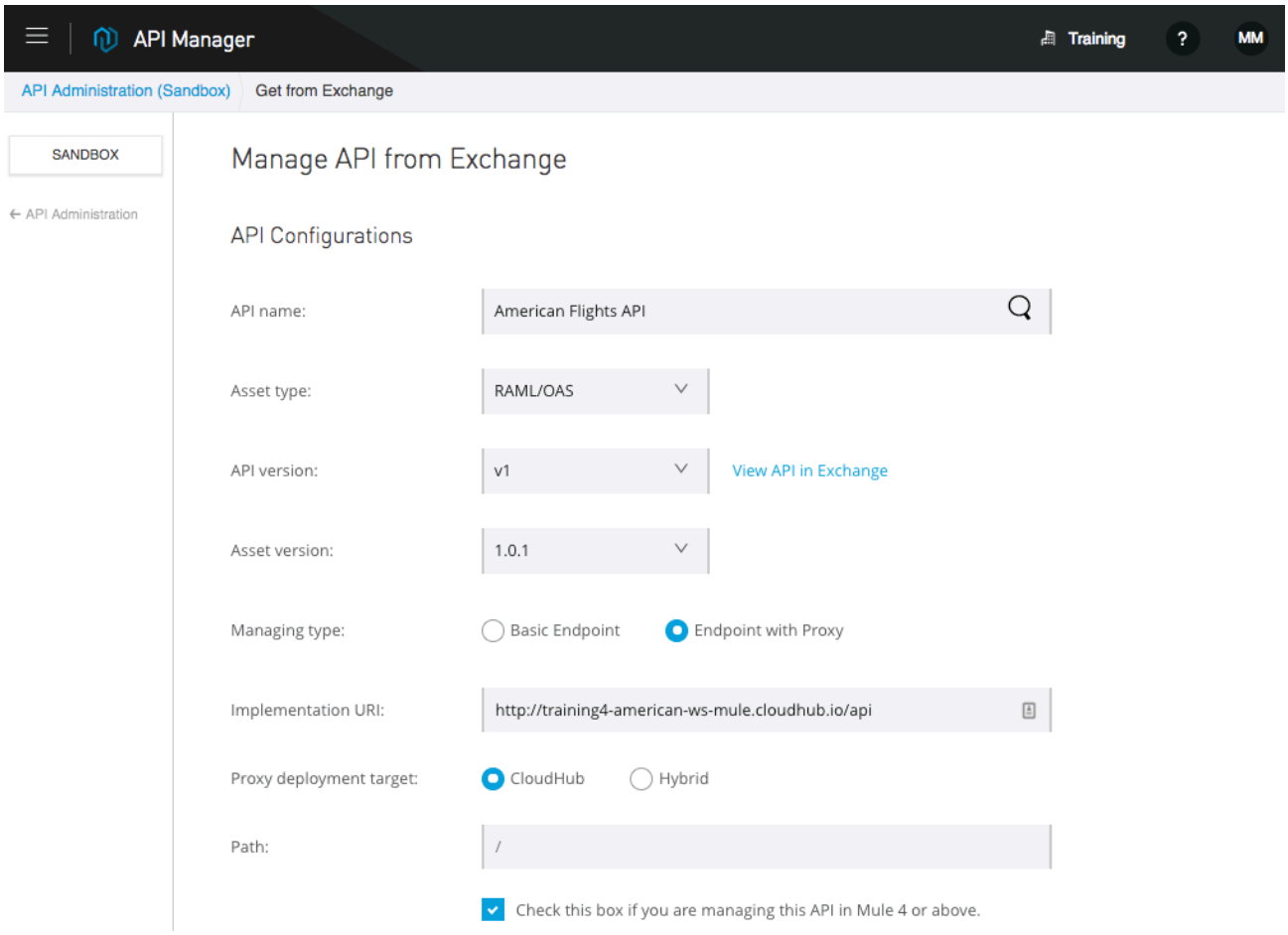


3. Click the Manage API button and select Manage API from Exchange.

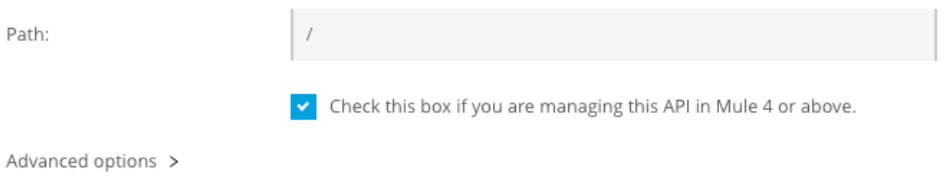


4. For API name, start typing American in the text field and then select your American Flights API in the drop-down menu that appears.
5. Set the rest of the fields to the following values:

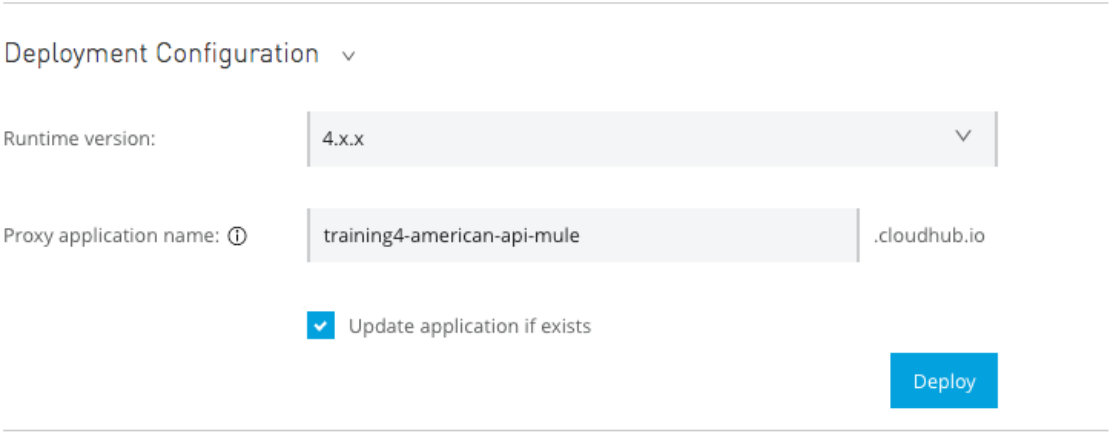
   - Asset type: RAML/OAS
   - API version: v1
   - Asset version: 1.0.1
   - Managing type: Endpoint with Proxy
   - Implementation URI: http://training4-american-ws-{*lastname*}.cloudhub.io/api
   - Proxy deployment target: CloudHub



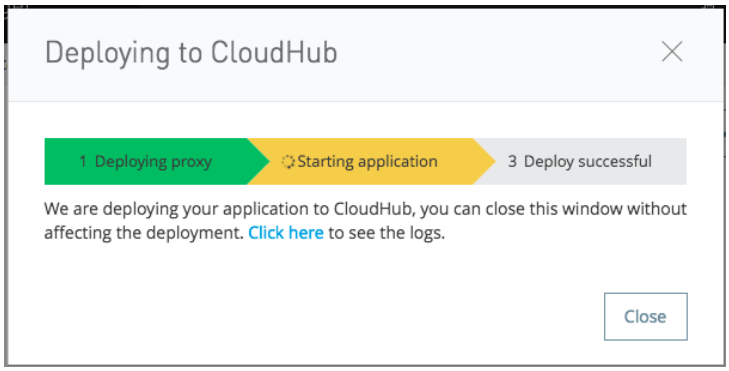
6. Select the checkbox: Check this box if you are managing this API in Mule 4 or above.



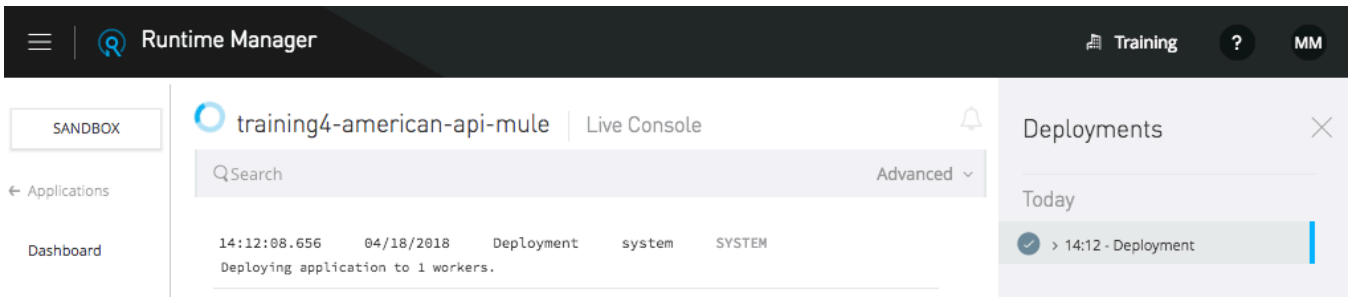
7. Click Save.
8. In the Deployment Configuration section, set the following values:
   - Runtime version: 4.x.x
   - Proxy application name: training4-american-api-{*lastname*}
9. Check Update application if exists.



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

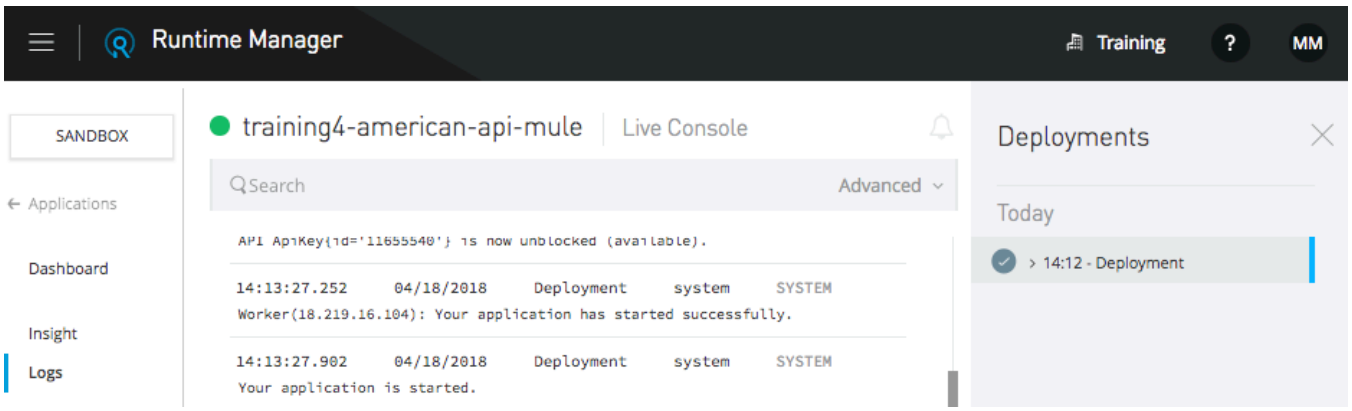


12. In the new browser tab that opens, watch the logs in Runtime Manager.

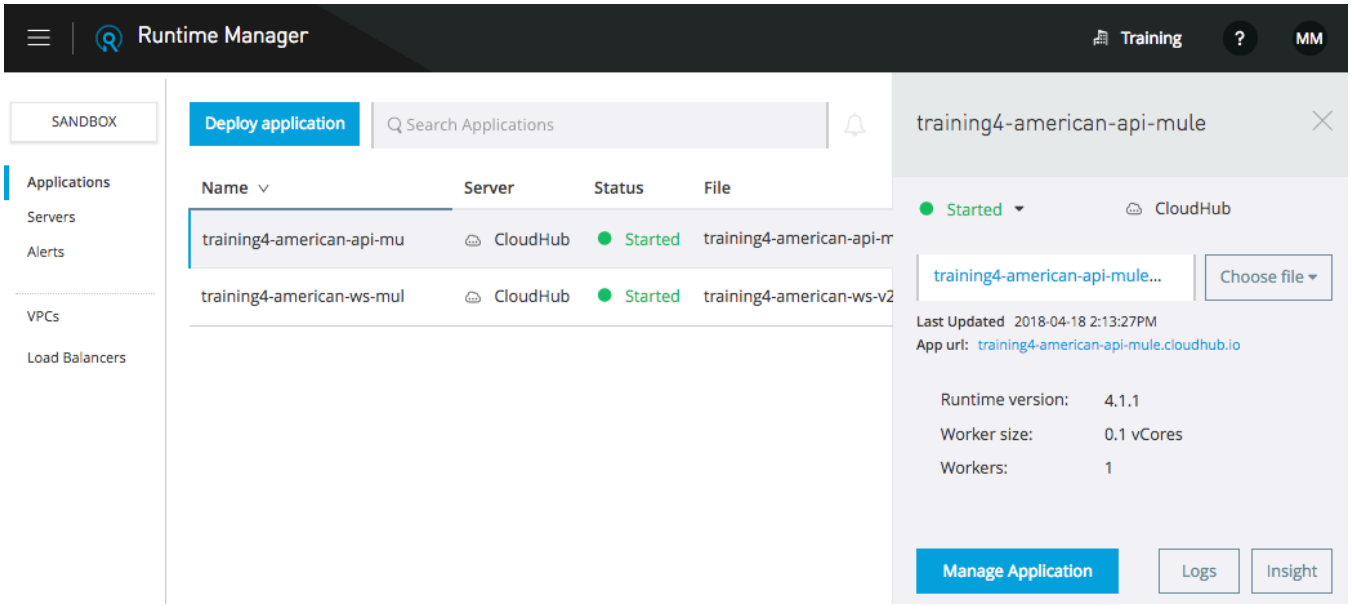


13. Wait until the proxy application starts.

    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.



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.

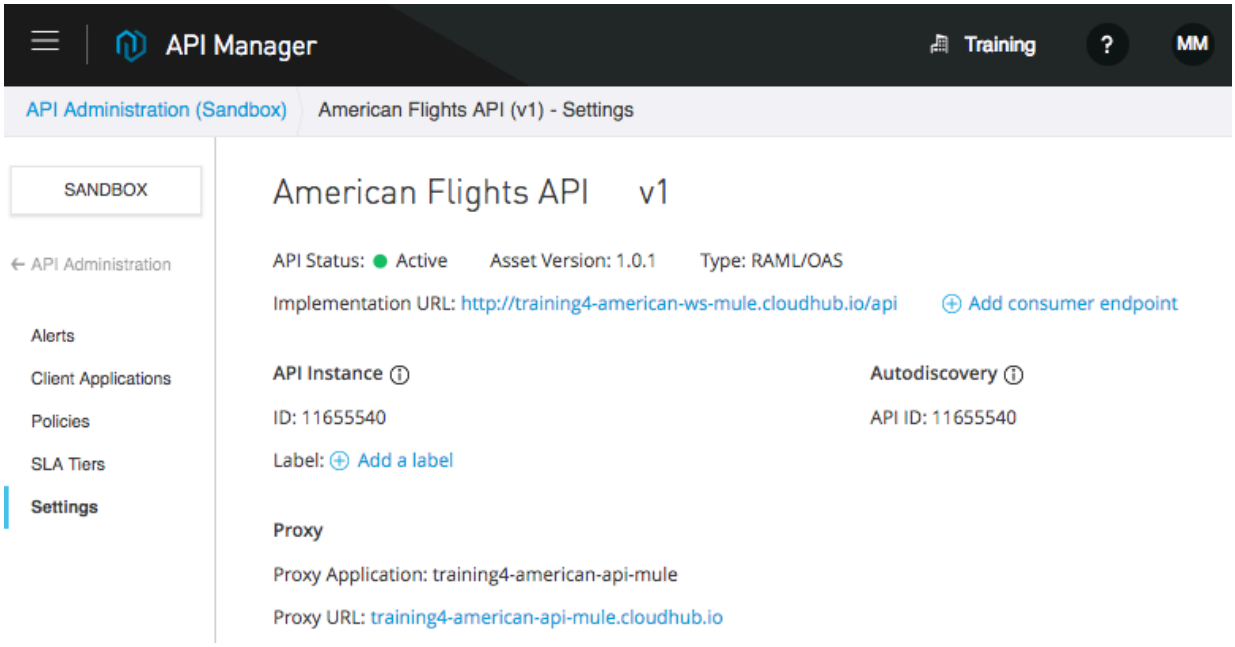


16. Close the browser tab.

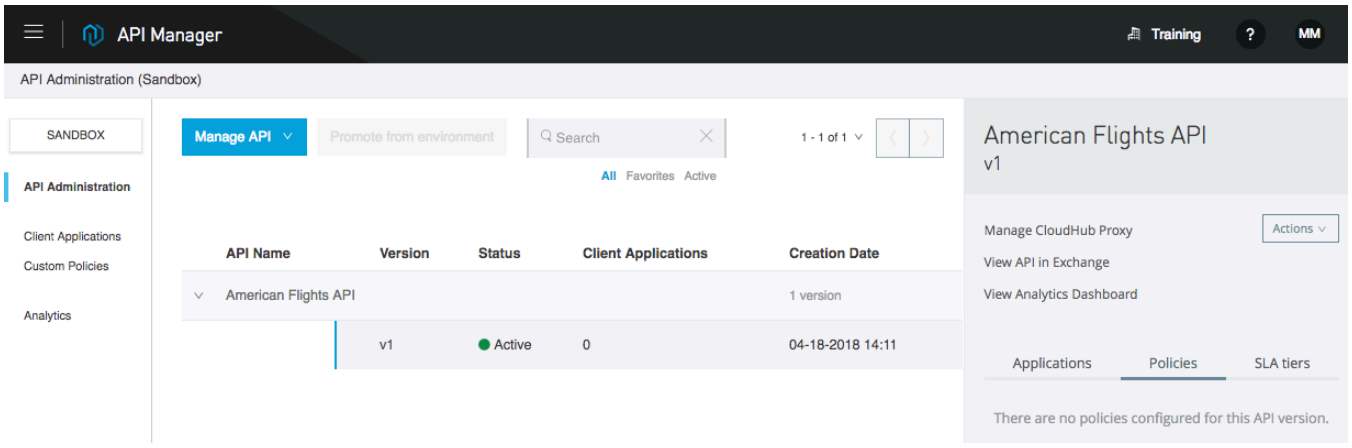
View API details in API Manager

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.

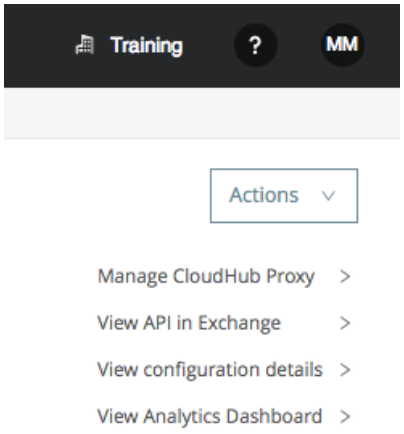


19. In the left-side navigation, click the API Administration link; you should now see your American Flights API listed.
20. Click in the row for the v1 version – but not on the v1 link.
21. Review the API version info that appears on the right side of the window; you should see there are no policies,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

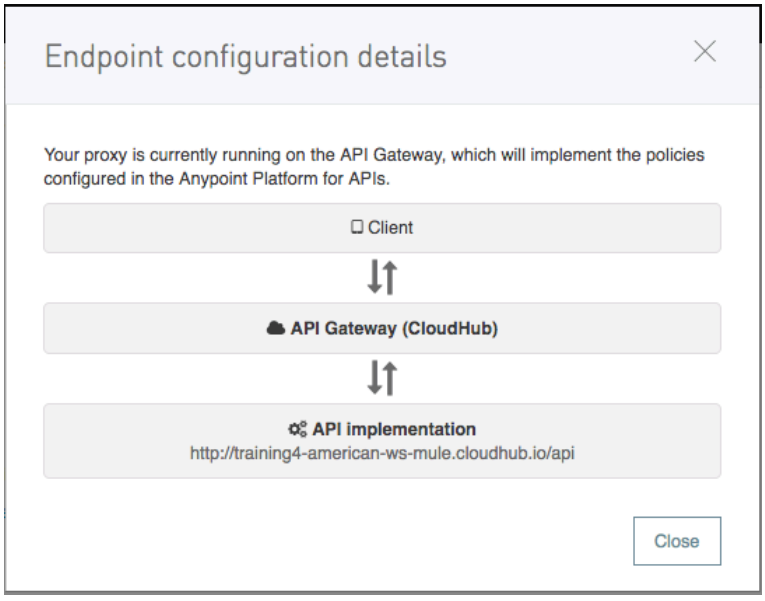


22. In the API list, click the v1 link for the API; you should be returned to the Settings page for the API.

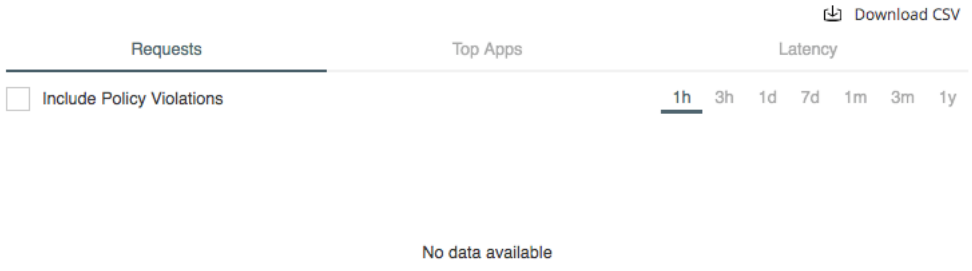
23. Locate and review the links on the right side of the page.



24. Click the View configuration details link.

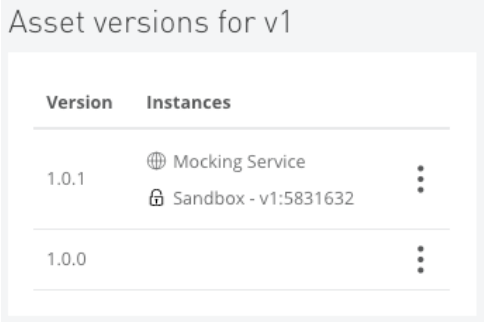


25. In the Endpoint configuration details dialog box, click Close.
26. On the Settings page, look at the requests graph; you should not see any API requests yet.



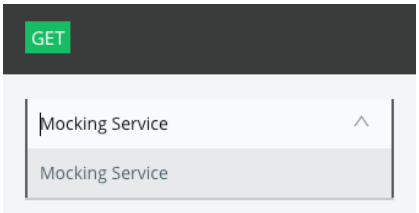
View the new API proxy instance in Exchange

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.

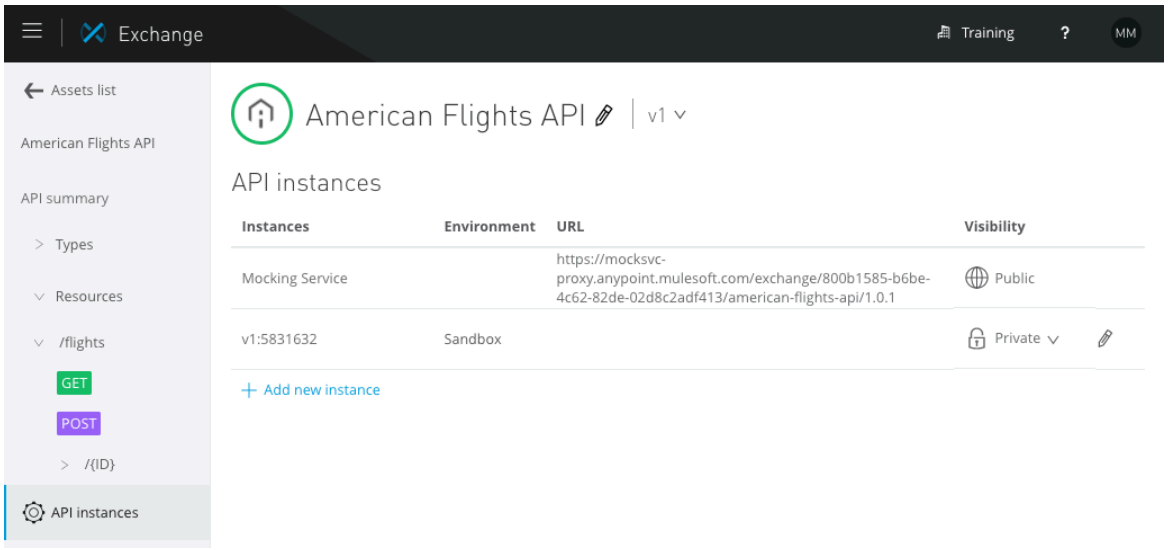


Note: You may need to refresh your browser page.

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.

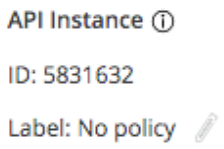


32. In the left-side navigation, select API instances; you should see that the new proxy instance does not have a URL.



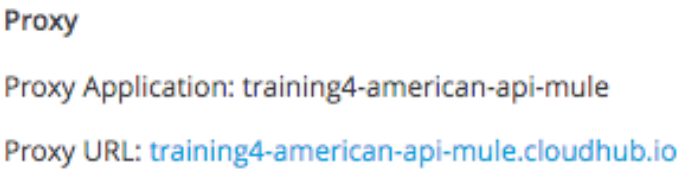
Set a friendly label for the API instance in API Manager

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.



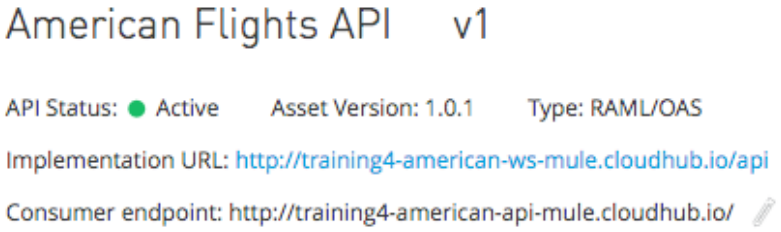
Set a consumer endpoint for the proxy in API Manager

36. Locate the proxy URL.



37. Right-click it and copy the link address.
38. Click the Add consumer endpoint link.

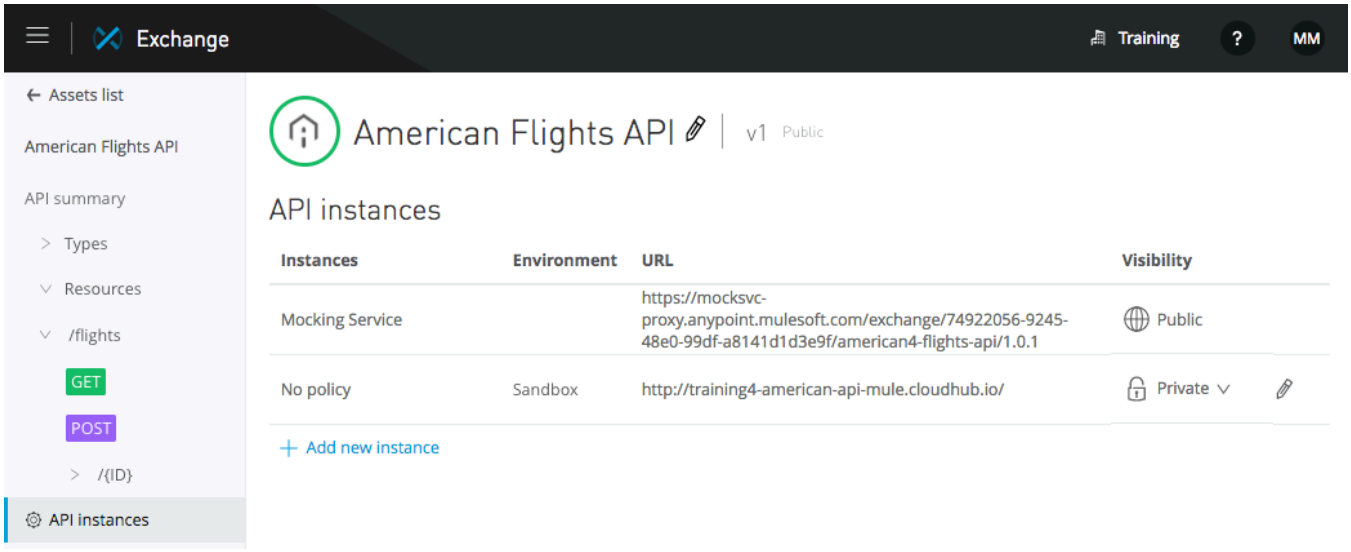
39. Paste the value of the proxy URL.
40. Press Enter/Return.



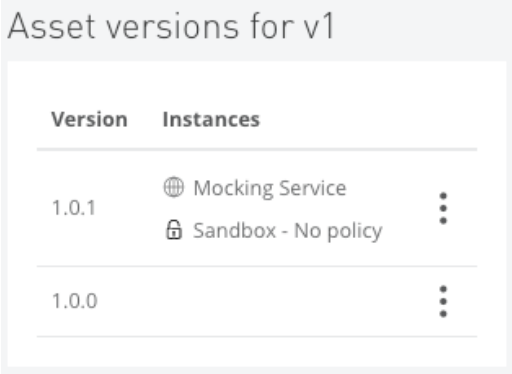
Make requests to the API proxy from Exchange

41. Return to the browser tab with Exchange.

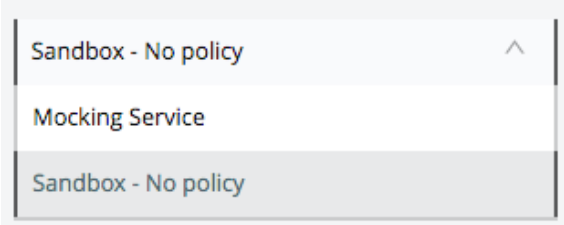
42. Refresh the API instances page for your American Flights API; you should see the new label and the URL.



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.

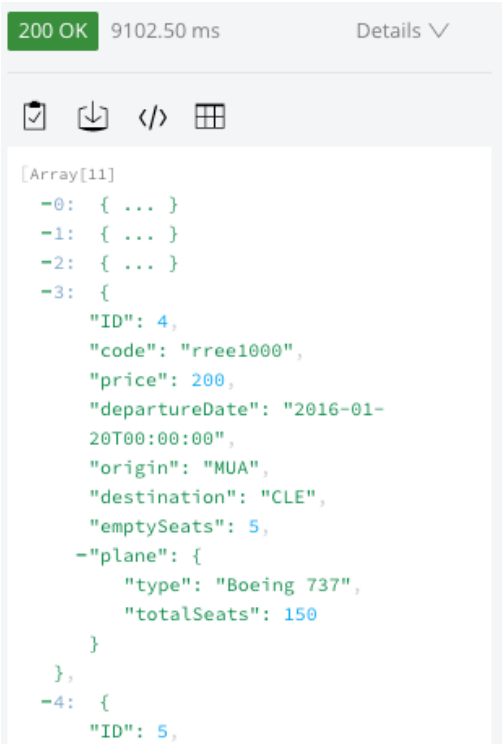


45. Click the GET method for the flights resource.
46. In the API console, click the drop-down arrow next to Mocking Service; you should now see your API proxy instance as a choice.



47. Select the Sandbox - No policy instance.

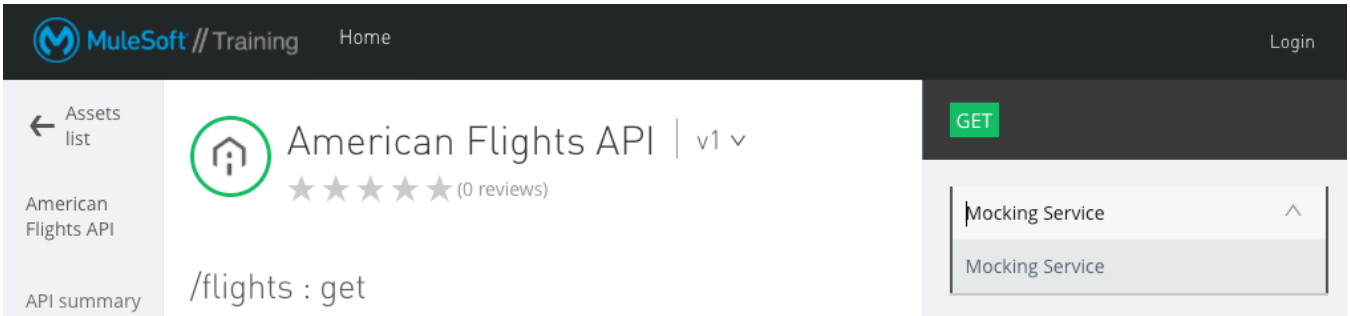
48. Click the Send button; you should now see the real data from the database, which contains multiple flights.



49. Make several more calls to this endpoint.
50. Make calls to different methods.

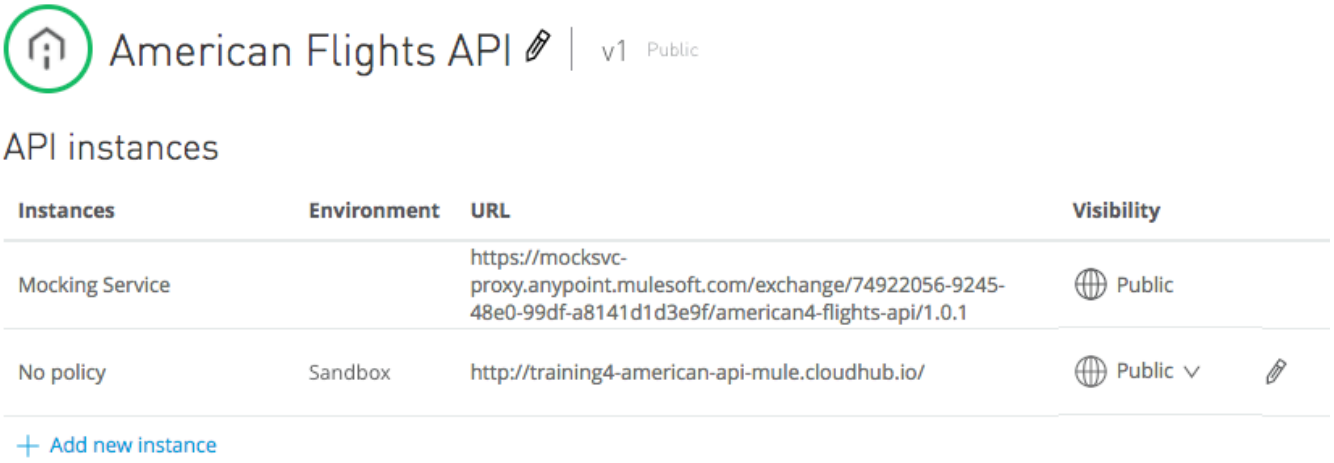
Make requests to the API proxy from the public portal

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.

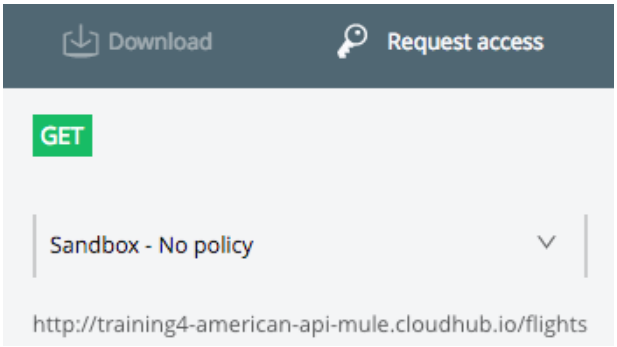


Make an API instance visible in the public portal

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.



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.

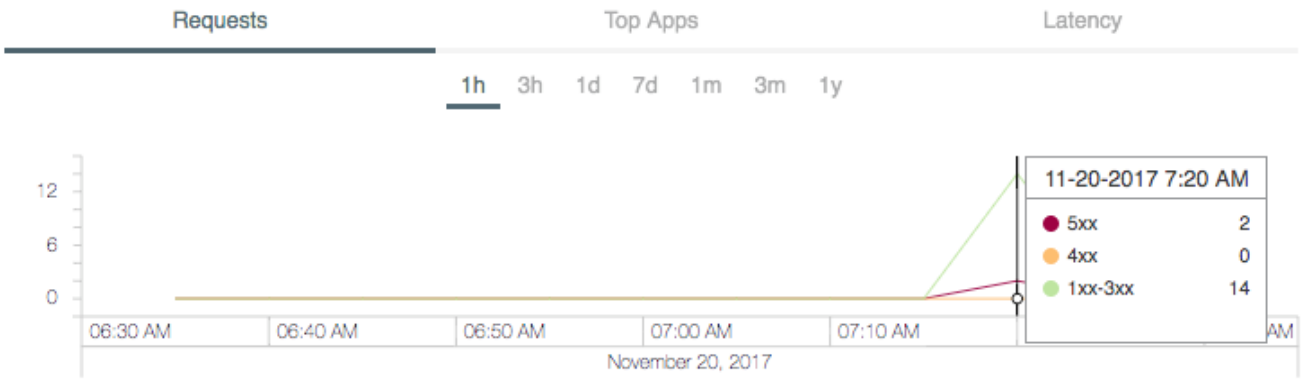


60. Click Send; you should a 200 response and flight data.

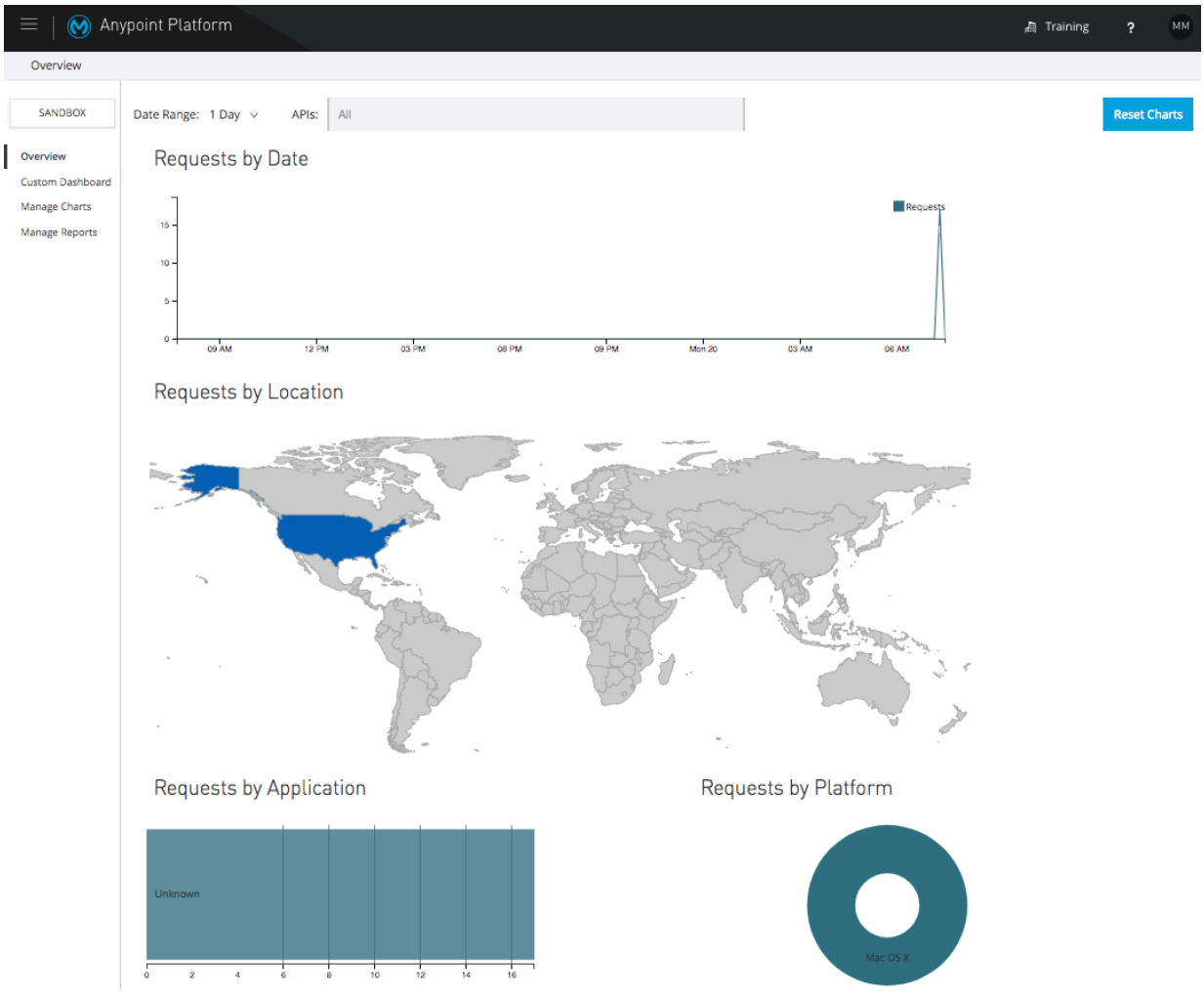
Look at the API request data

61. Return to the browser tab with API Manager.
62. Refresh the Settings page for you 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.

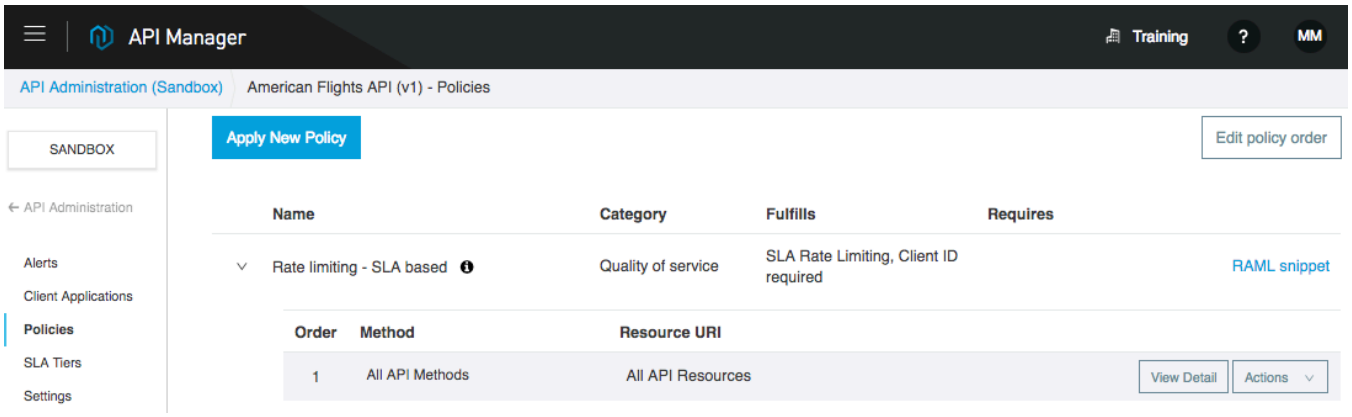


66. Close the browser tab.

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

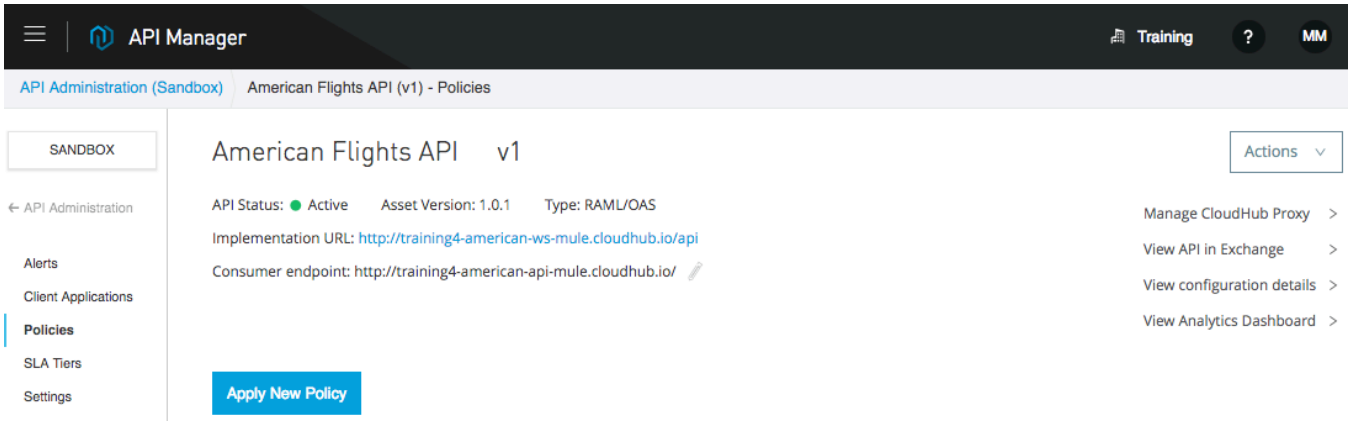
In this walkthrough, you govern access to the API proxy. You will:

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



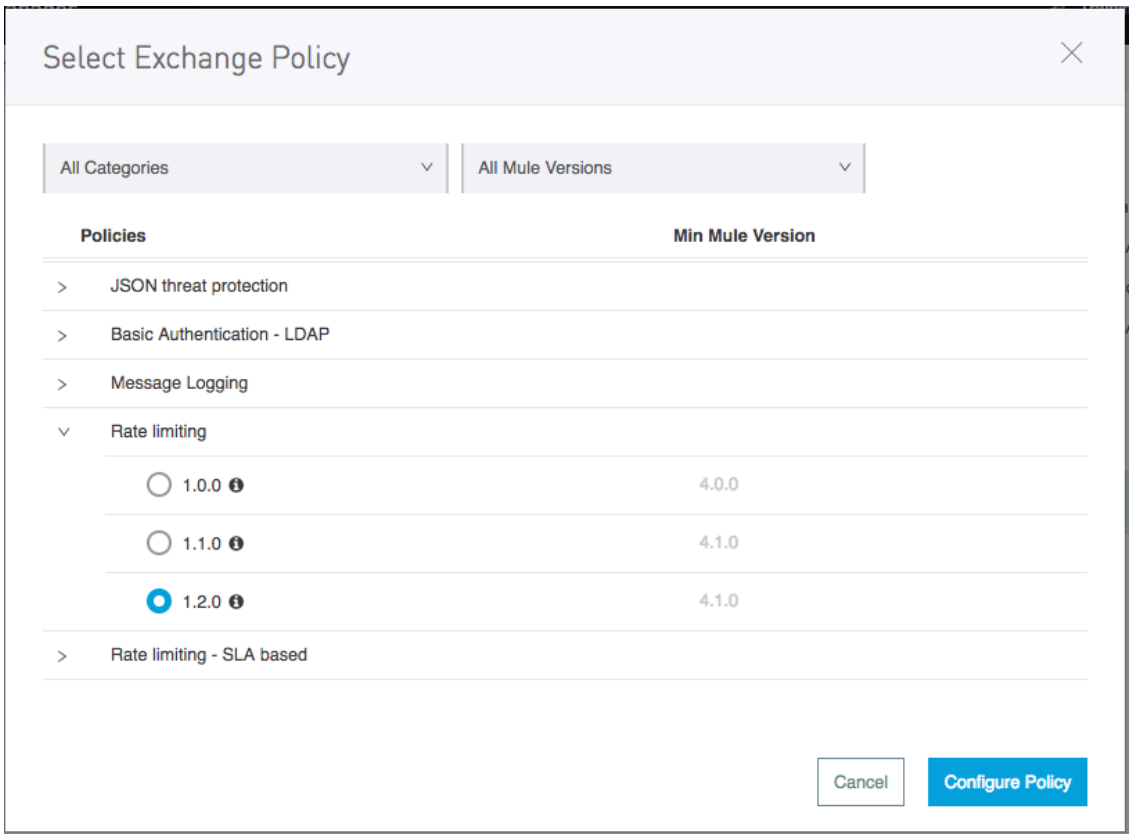
Create a rate limiting policy

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



3. Click the Apply New Policy button.

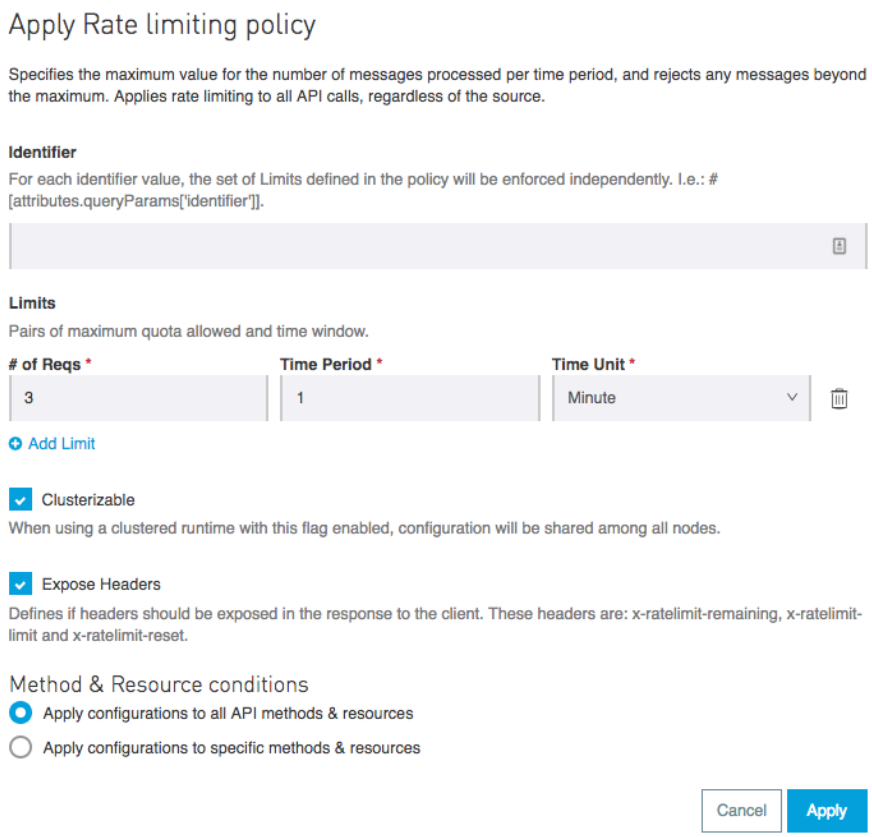
4. In the Select Exchange Policy dialog box, expand Rate limiting and select the latest version for the Mule runtime version you are using.



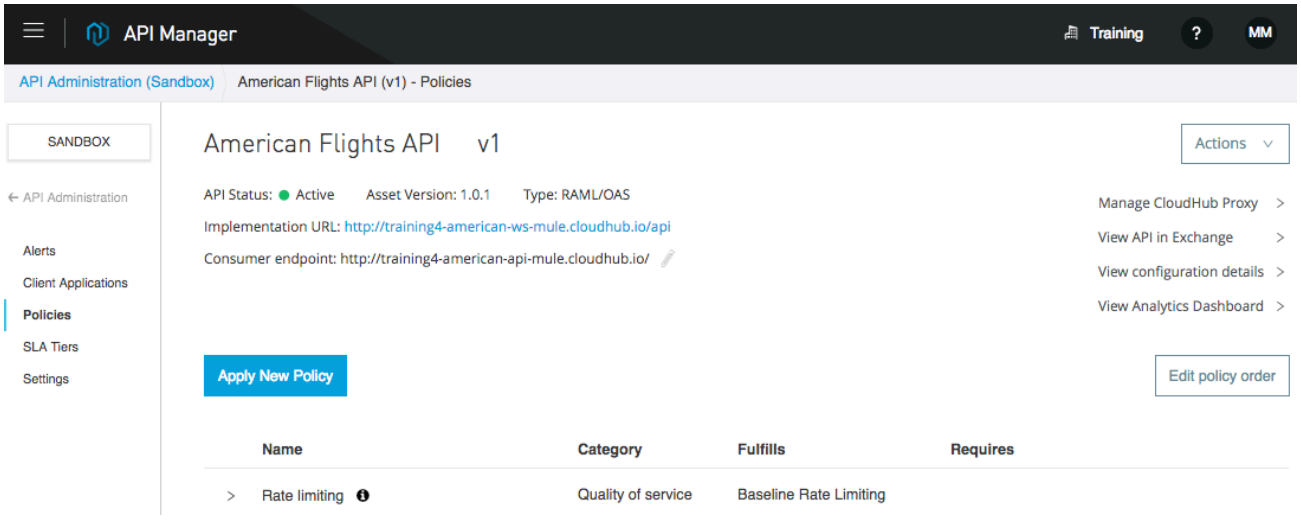
5. Click Configure Policy.

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

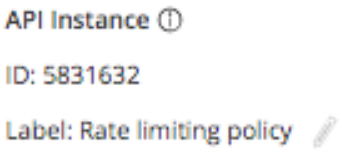
   - # of Reqs: 3
   - Time Period: 1
   - Time Unit: Minute
   - Method & Resource conditions: Apply configurations to all API methods & resources



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



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

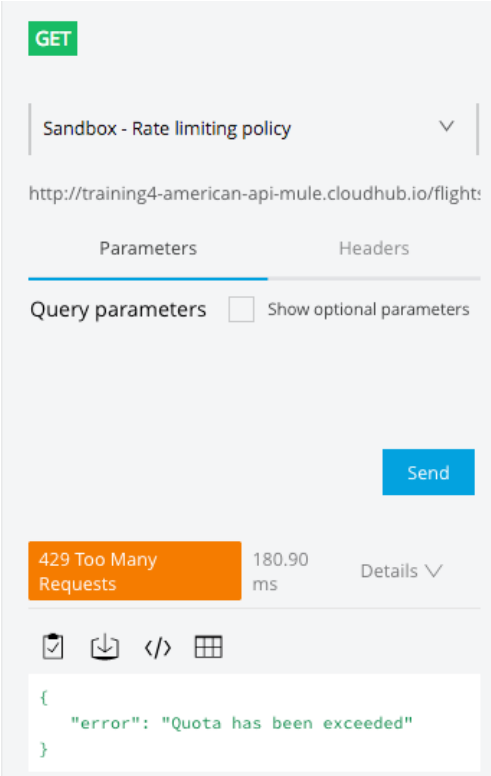


Test the new rate limiting policy

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

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

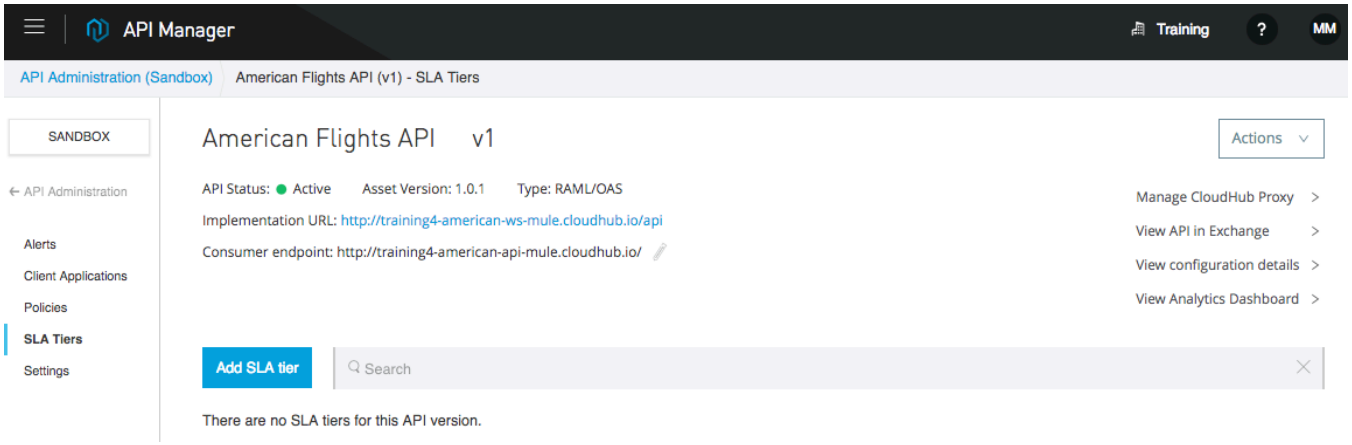
13. Press Send until you get a 429 Too Many Requests response.



Create SLA tiers

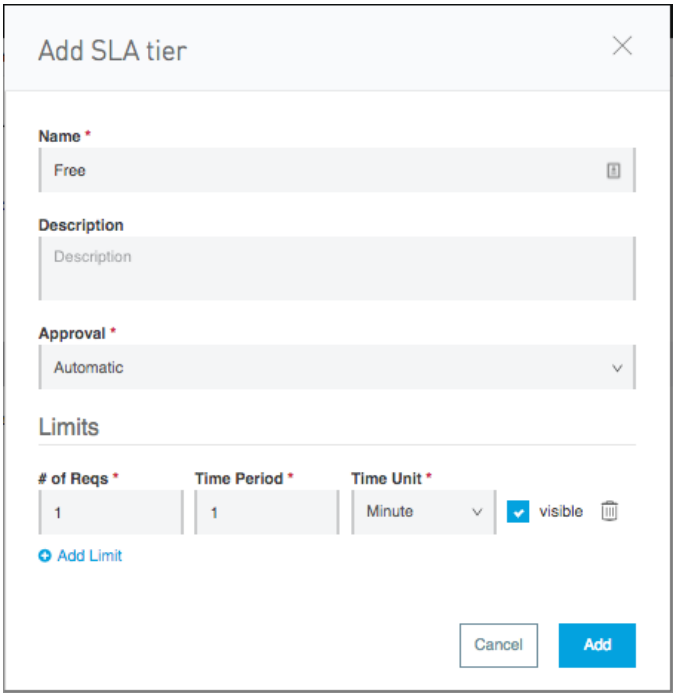
14. Return to the browser tab with your American Flights API in API Manager.
15. In the left-side navigation, select SLA Tiers.

16. Click the Add SLA tier button.



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

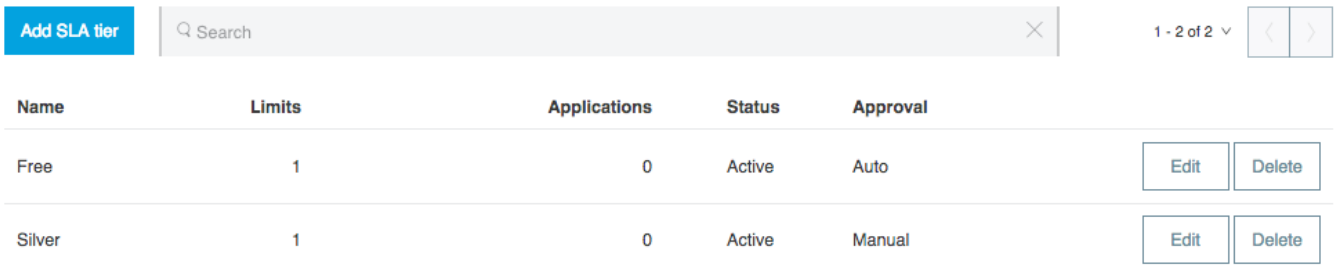
- Name: Free
- Approval: Automatic
- # of Reqs: 1
- Time Period: 1
- Time Unit: Minute



18. Click the Add button.

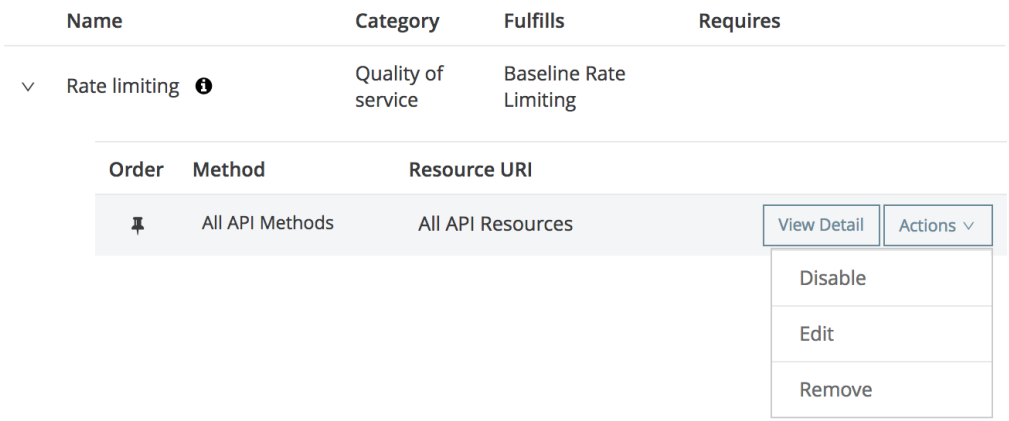
19. Create a second SLA tier with the following values:

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



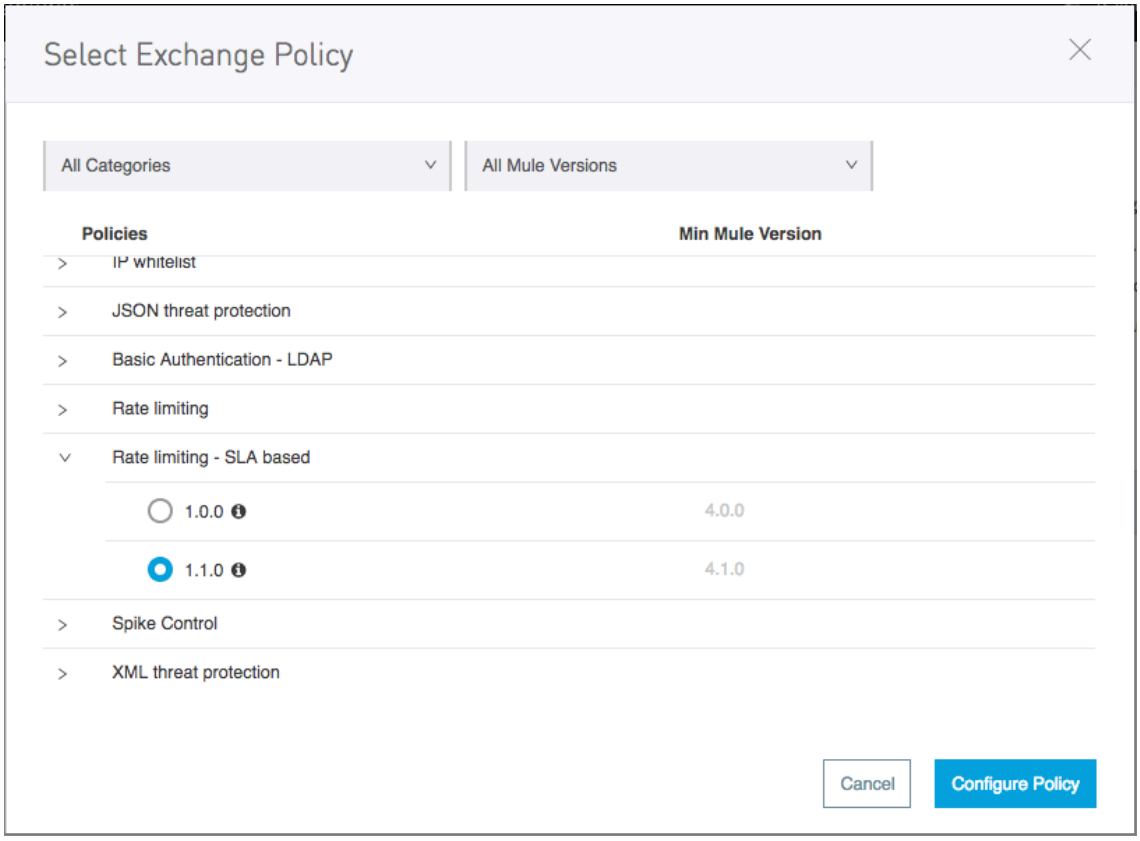
Change the policy to rate limiting – SLA based

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

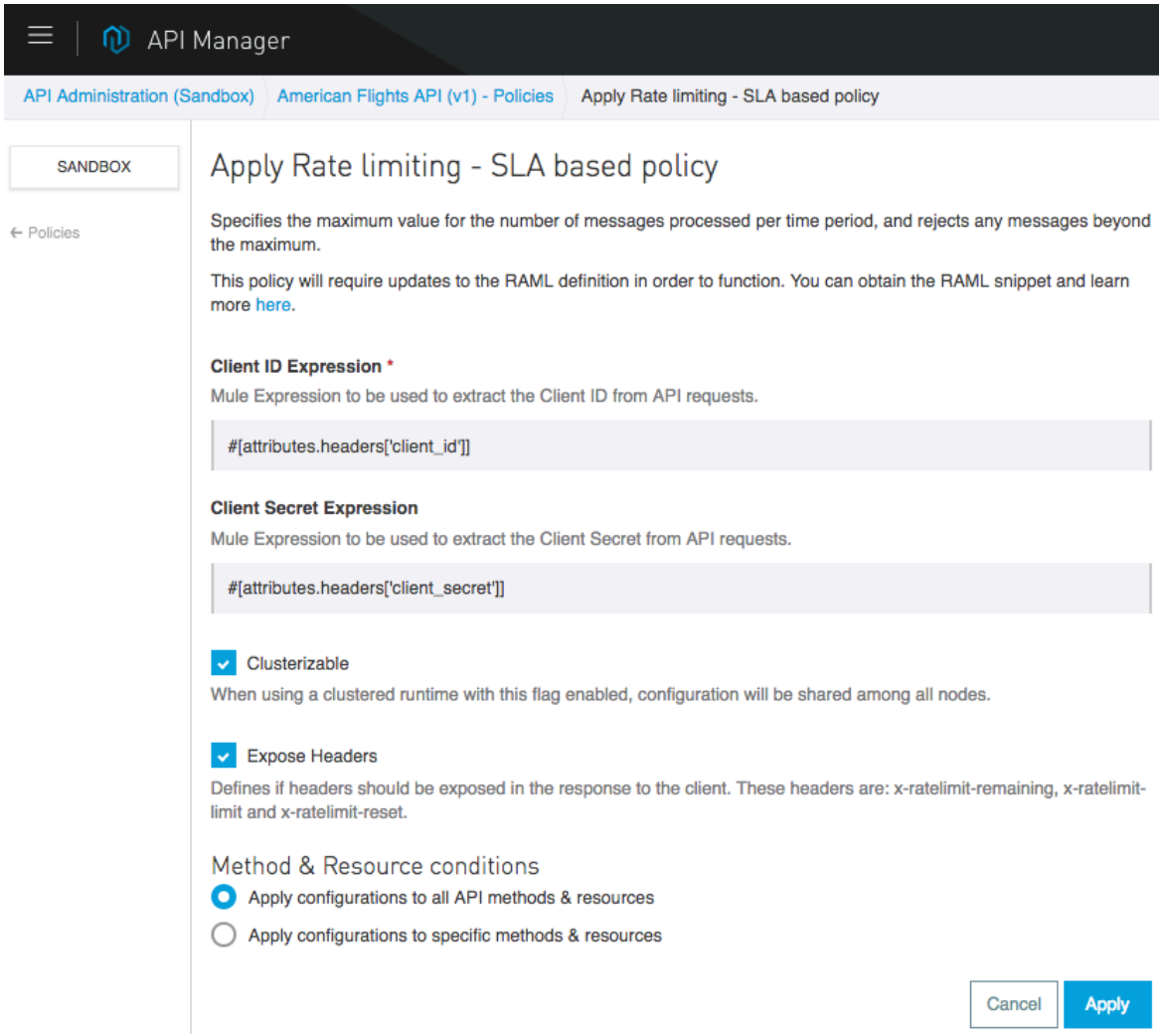


23. In the Remove policy dialog box, click Remove.
24. Click the Apply New Policy button.

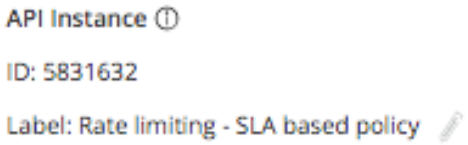
25. In the Select Policy dialog box, expand Rate limiting - SLA based and select the latest version for the Mule runtime version you are using.
26. Click Configure Policy.



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.



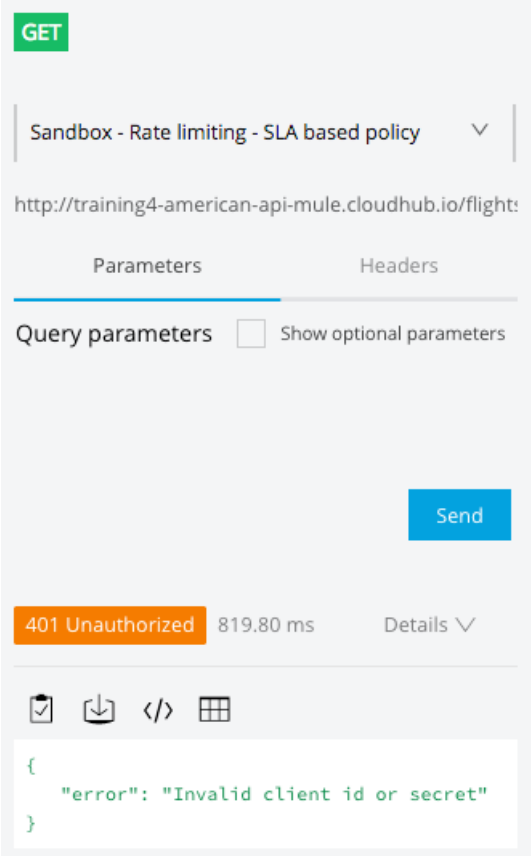
28. Click Apply.
29. In the left-side navigation, select Settings.
30. Change the API instance label to Rate limiting – SLA based policy.



Test the rate limiting – SLA based policy in Exchange

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

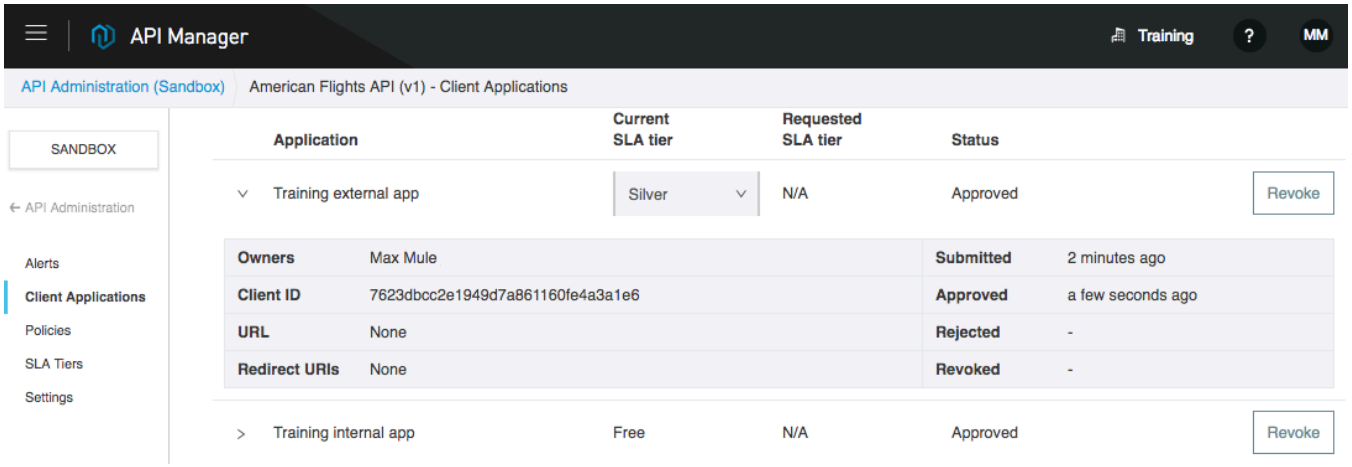
33. Click Send; you should get a 401 Unauthorized response with a message that there is an invalid client id or secret.



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

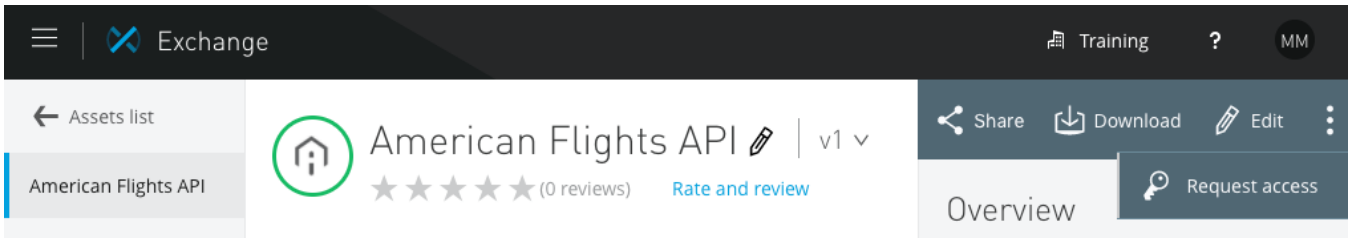
In this walkthrough, clients request access to an API proxy and administrators grant access. You will:

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

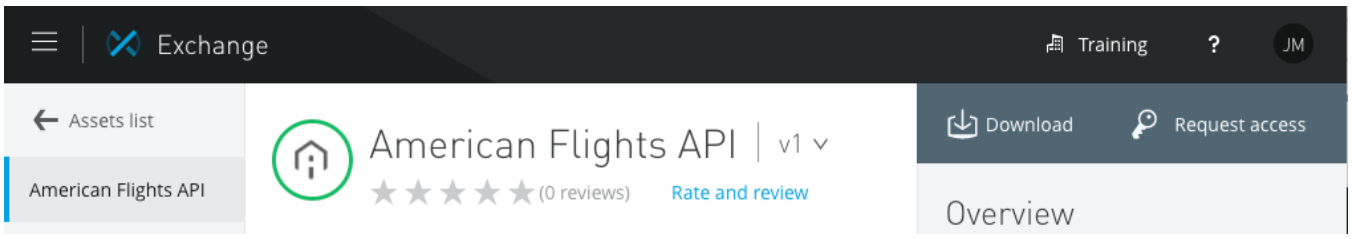


Request access to the API as an internal consumer

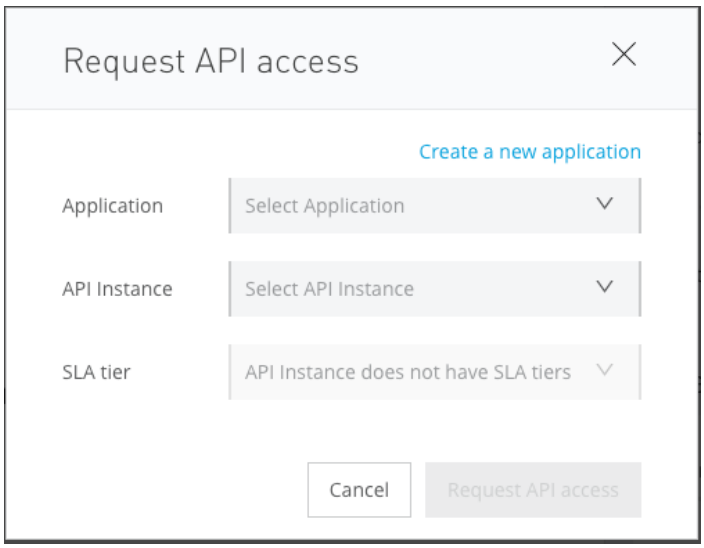
1. Return to the browser tab with Anypoint Exchange.
2. In the left-side navigation, select the name of the API to return to its home page.
3. Click the more options button in the upper-right corner and select Request access.



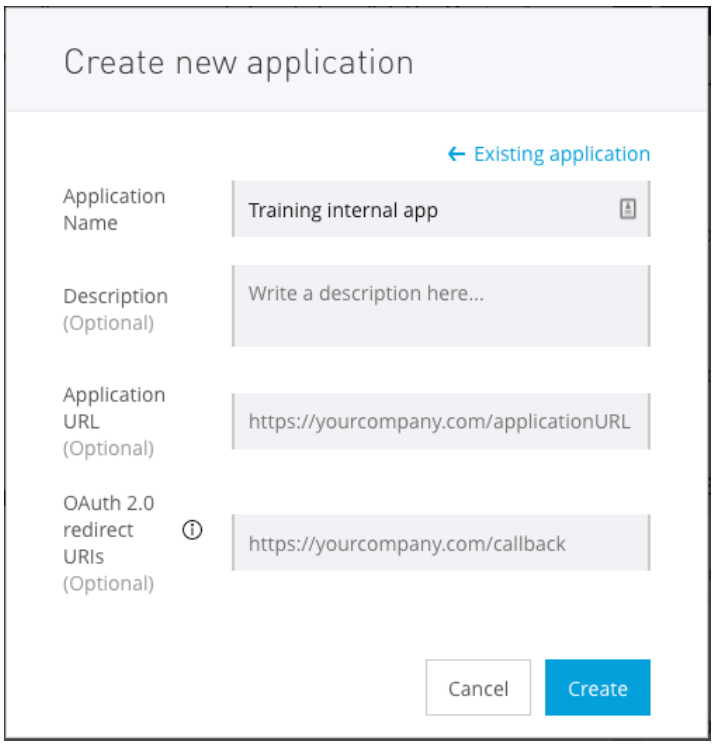
Note: Other internal users that you shared the API with that do not have Edit permissions will see a different menu.



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

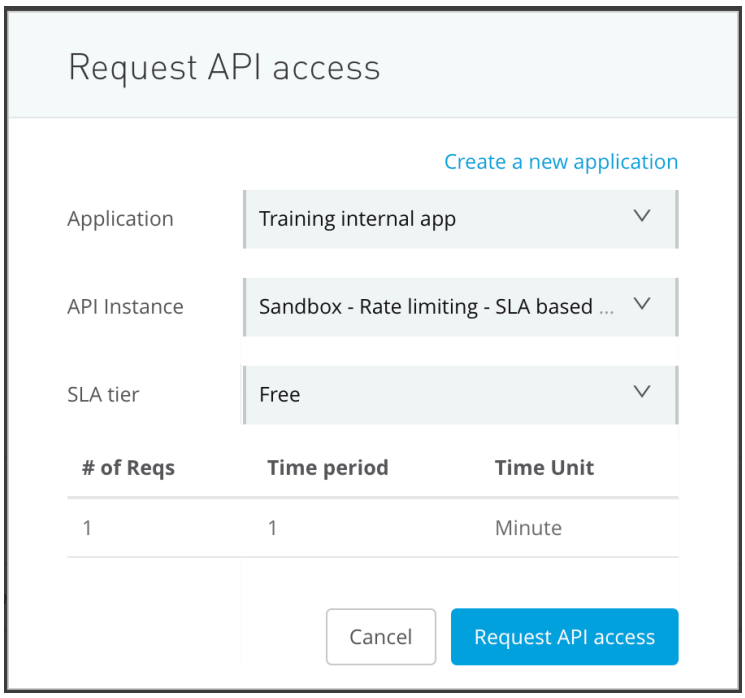


5. In the Create new application dialog box, set the name to Training internal app and click Create.

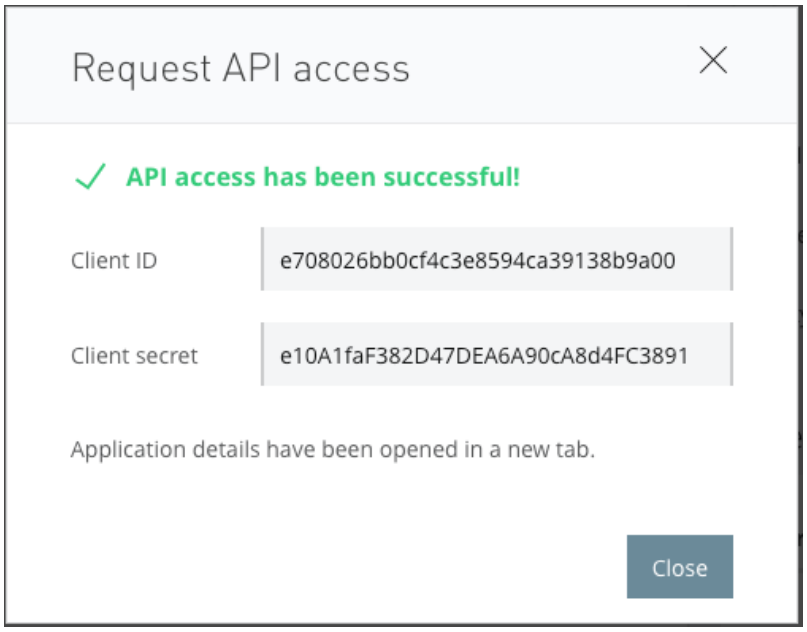


6. In the Request API access dialog box, set the API instance to Sandbox – Rate limiting SLA - based policy.

7. Set the SLA tier to Free.



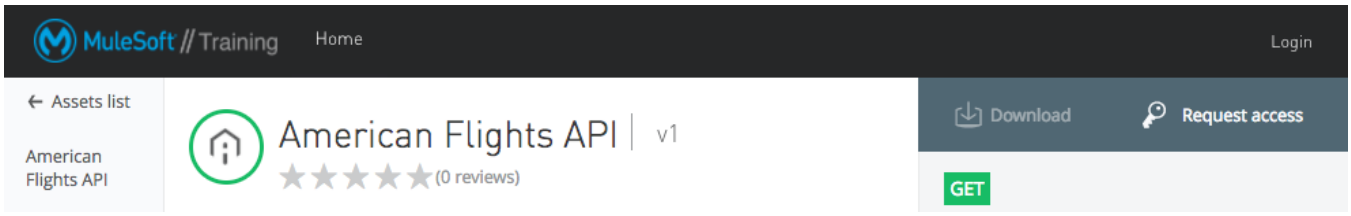
8. Click Request API access.
9. In the Request API access dialog box, view the assigned values for the client ID and client secret.



10. Click Close.

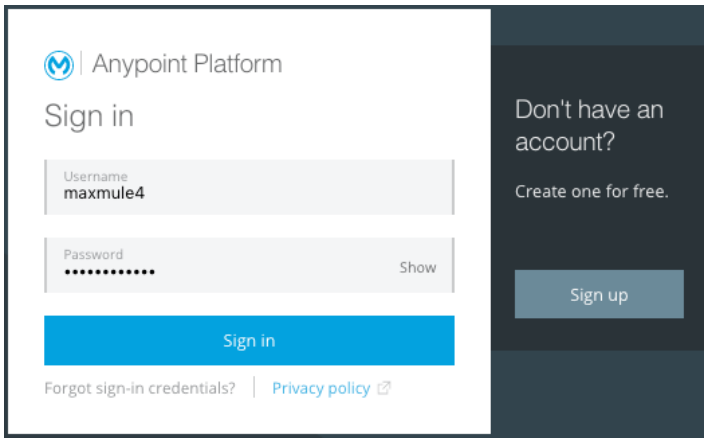
Request access to the API as an external consumer

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.



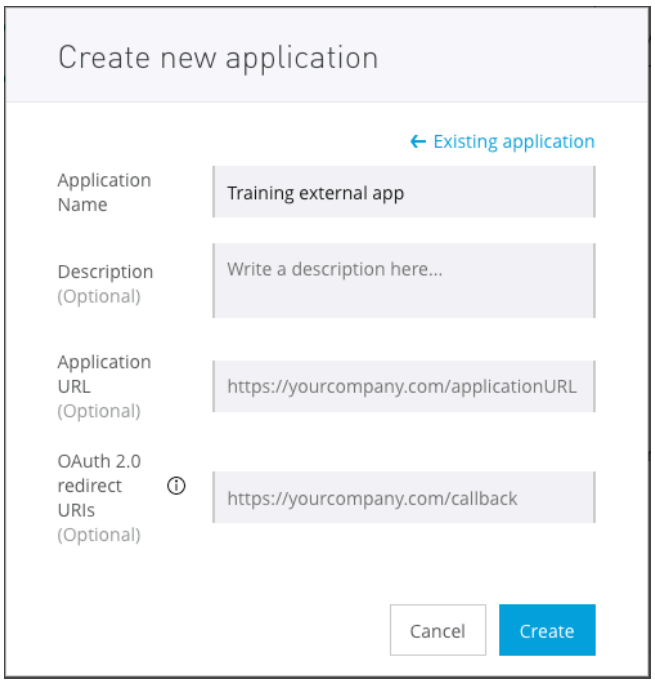
13. Click the Request access button; you should get a page to sign in or create an Anypoint Platform account.
14. Enter your existing credentials and click Sign in.

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

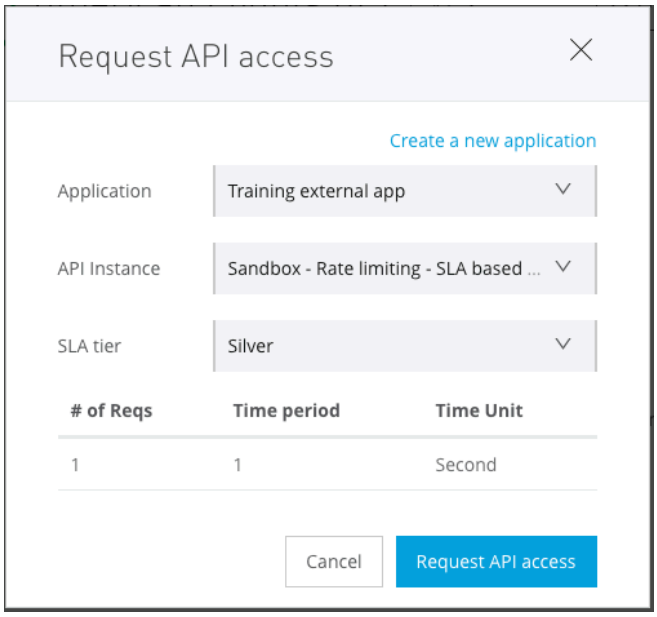


15. Back in the public portal, click the Request access button again.
16. In the Request API access dialog box, click the Create a new application button

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

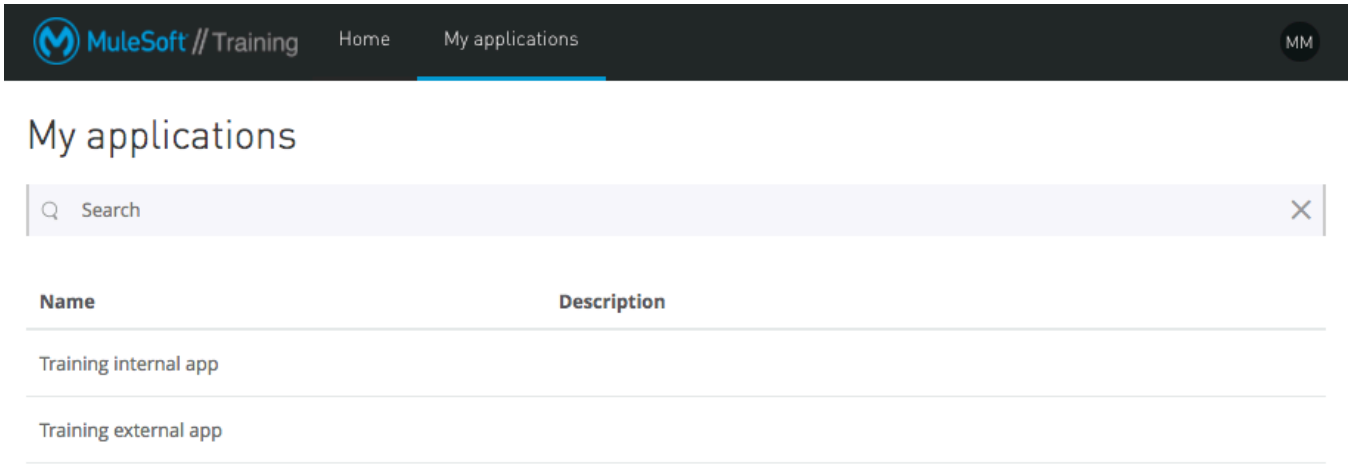


18. Click Create.
19. In the Request API access dialog box, set the API instance to Sandbox – Rate limiting SLA-based policy.
20. Set the SLA tier to Silver.

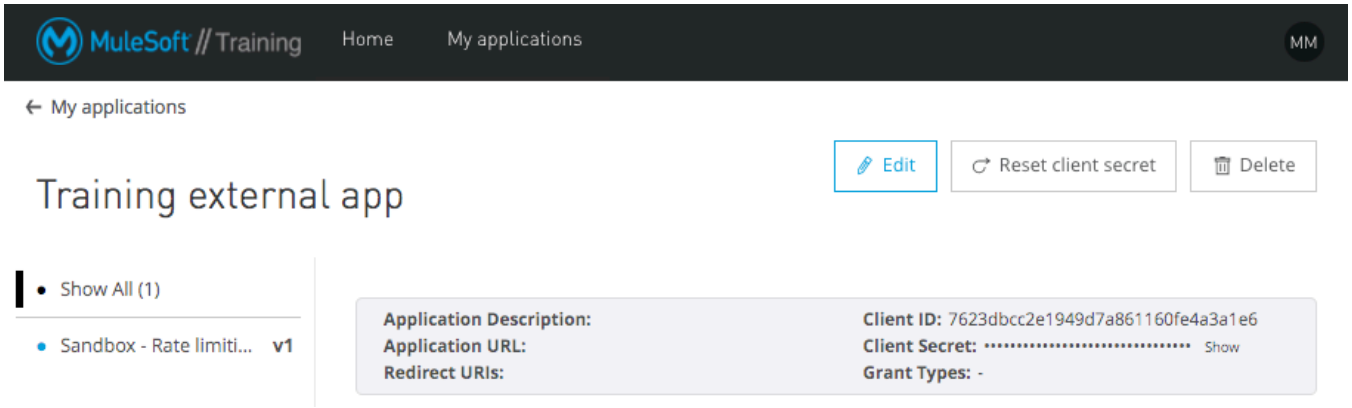


21. Click Request API access.
22. In the Request API access dialog box, click Close.

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.



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.

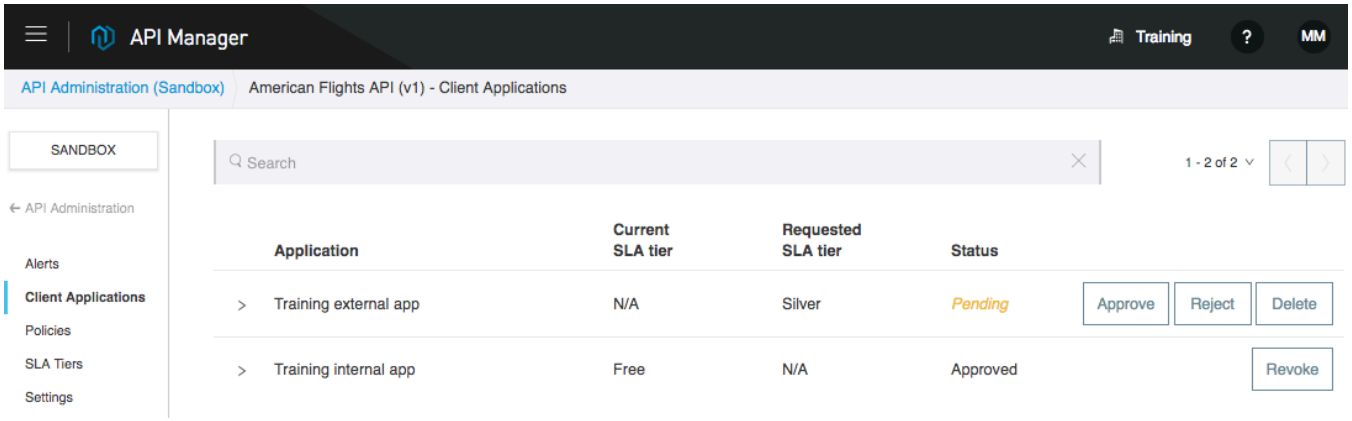


25. Leave this page open in a browser so you can return to it and copy these values.

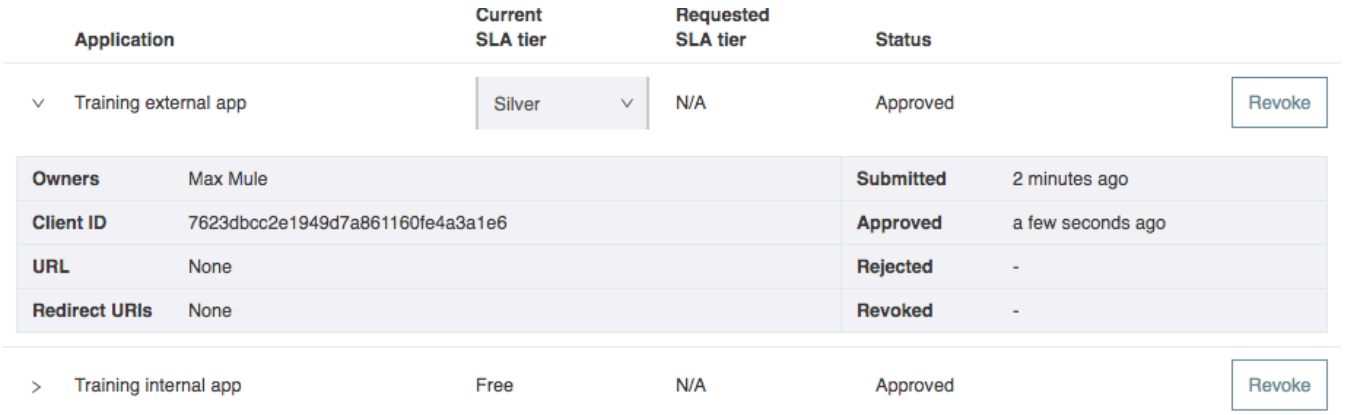
Grant an application access

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.



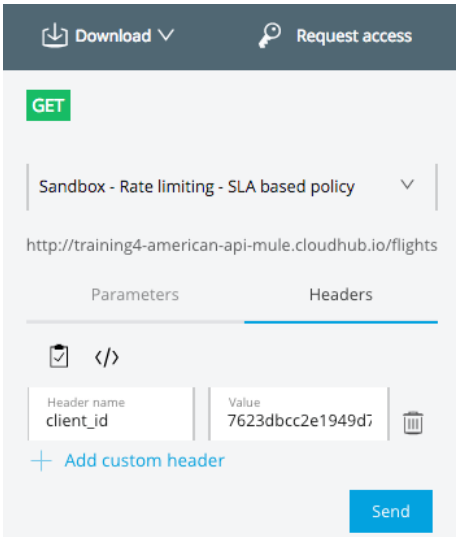
28. Click the Approve button for the application requesting Silver tier access.
29. Expand the Training external app row and review its information.
30. Copy the value of the client_id.



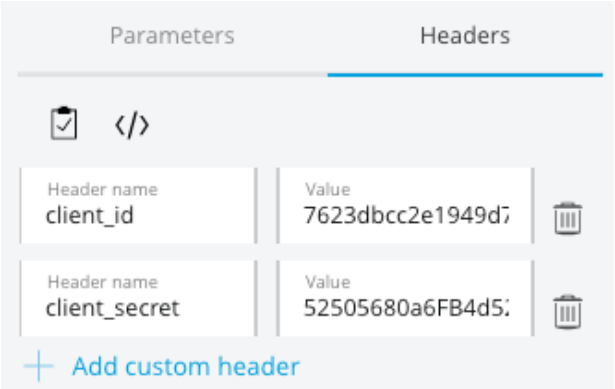
Add authorization headers to test the rate limiting – SLA based policy from an API portal

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.

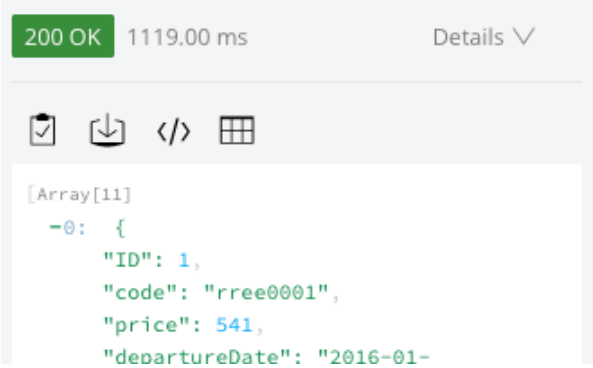
36. Set the value of client_id to the value you copied.



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.



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

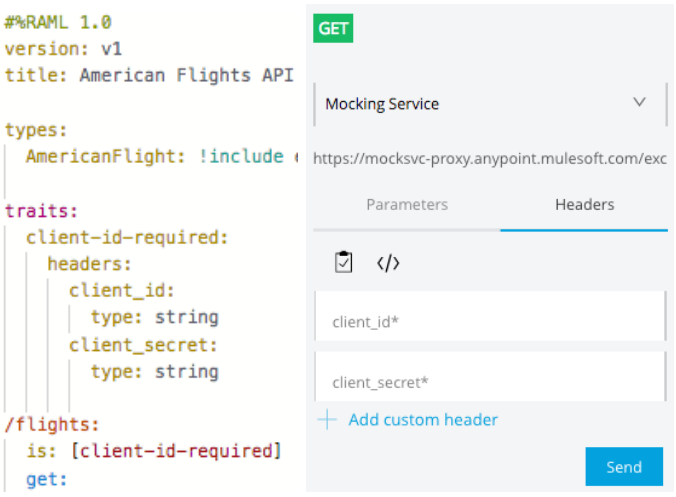


Walkthrough 5-5: Add client ID enforcement to an API specification

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

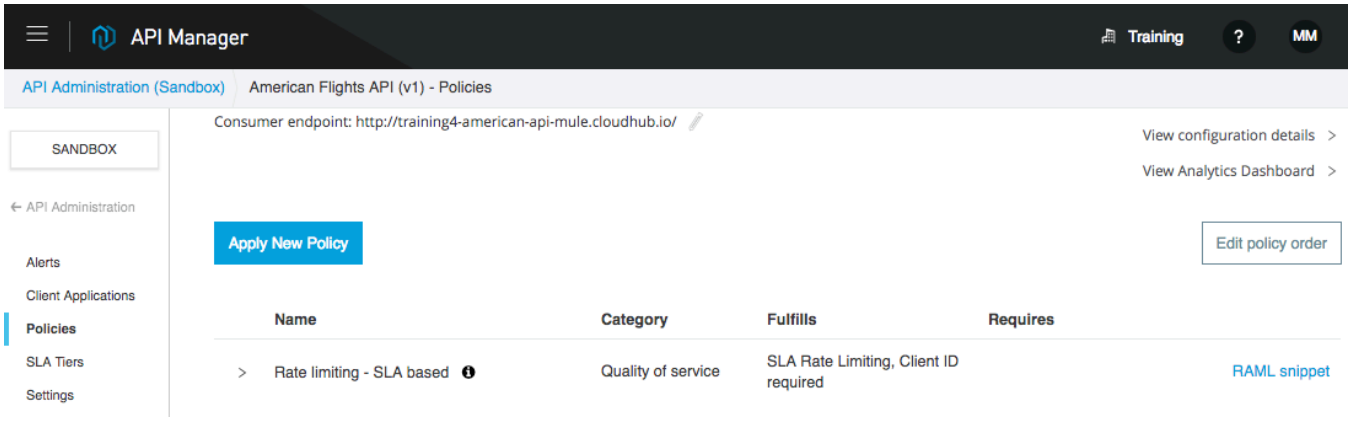
- Modify an API specification to require client id and client secret headers with requests.
- Update a managed API to use a new version of an API specification.
- Call a governed API with client credentials from API portals.

  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.

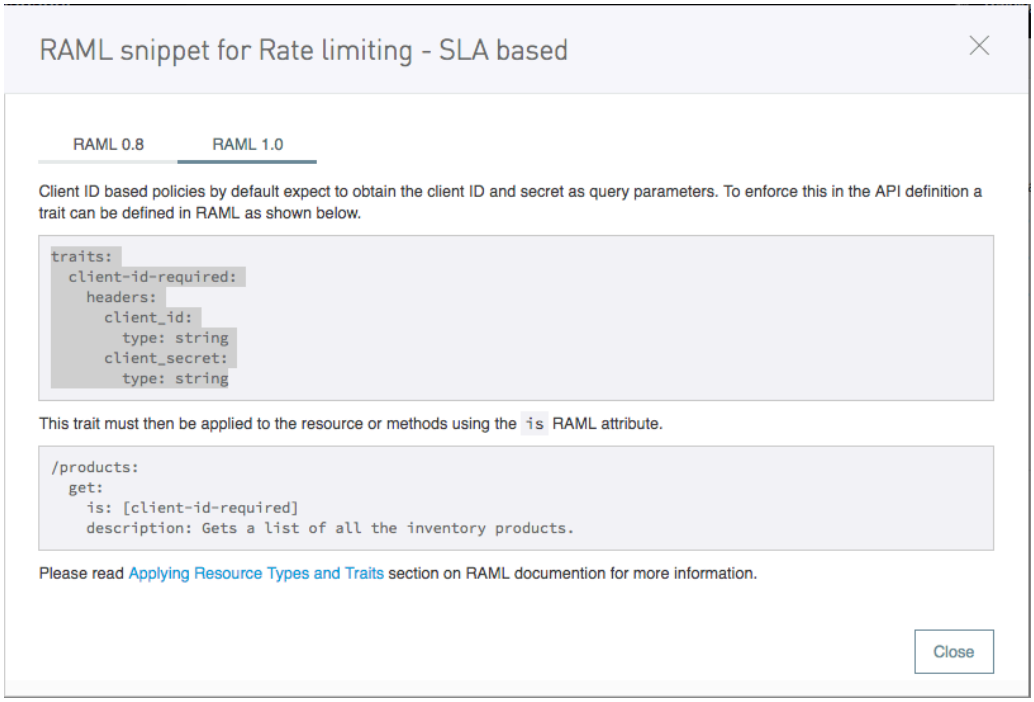


Copy the traits required to add authentication to the API specification

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.



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



6. Click Close.

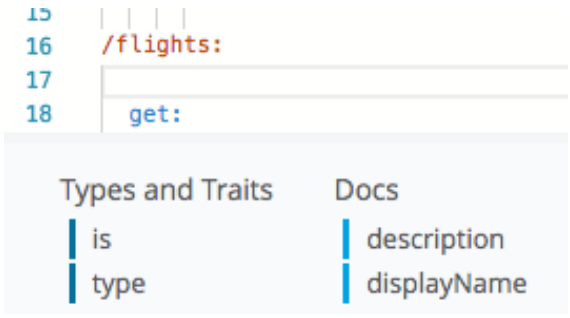
Add authentication headers to the API specification

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

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

types:
  AmericanFlight: !include exchange_modules/

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

/flights:
```

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

10. In the shelf, select is.



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:
```

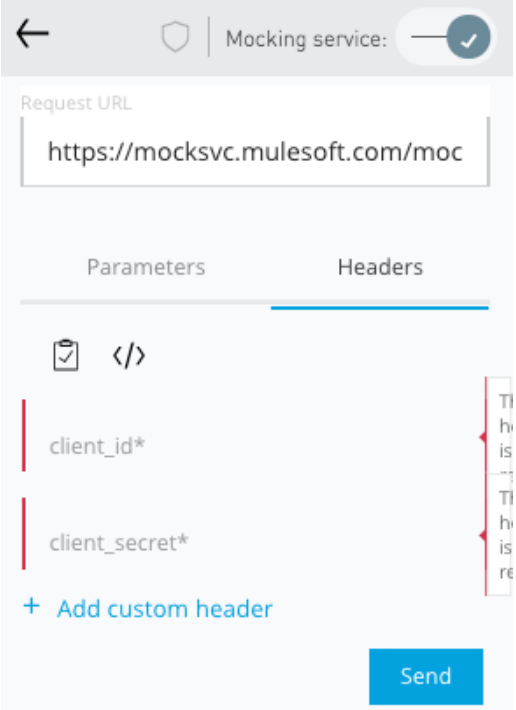
13. Repeat this process so the trait is applied to all methods of the {ID} resource as well.

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

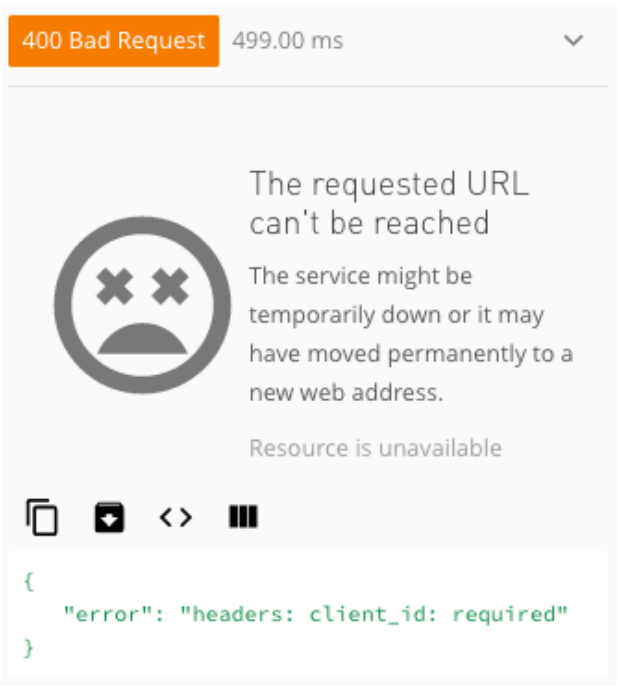
Test the API in the API console in Design Center

14. In the API console, turn on the mocking service.
15. Select one of the resources and click Try it.

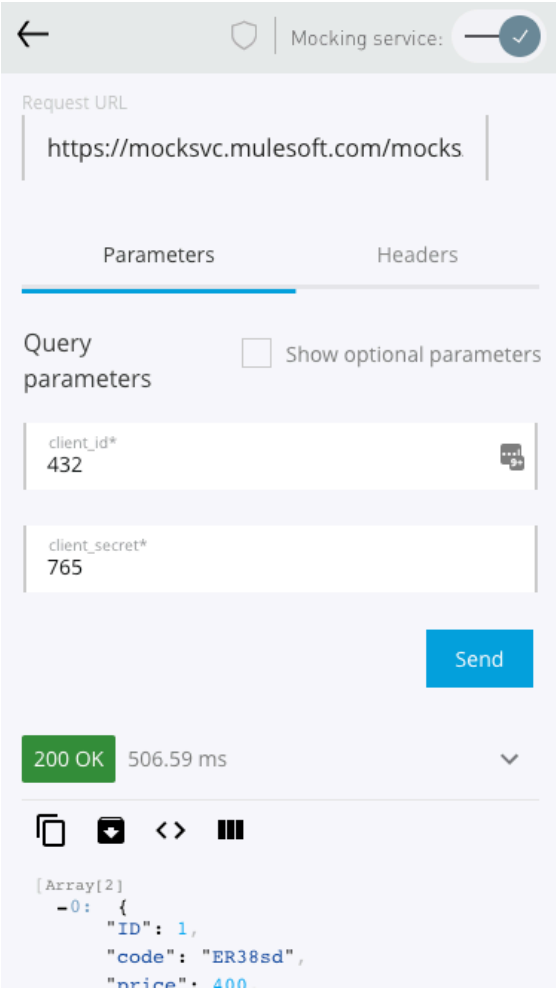
16. Select the Headers tab; you should now see fields to enter client_id and client_secret.



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

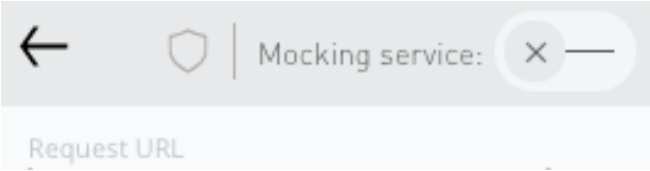


18. Enter *any* values for the client_id and client_secret and click Send; you should get a 200 response with the example results.



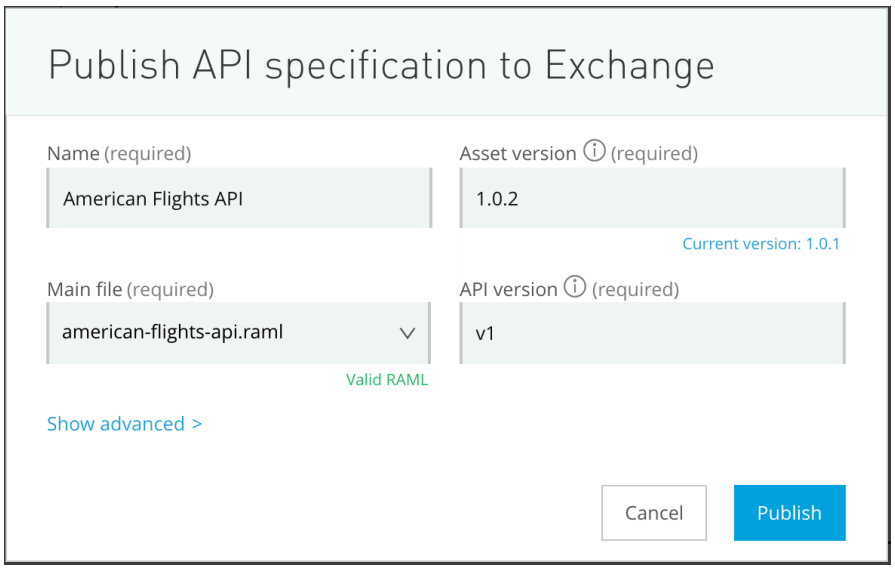
Publish the new version of the API to Exchange

19. Turn off the mocking service.



20. Click the Publish to Exchange button.

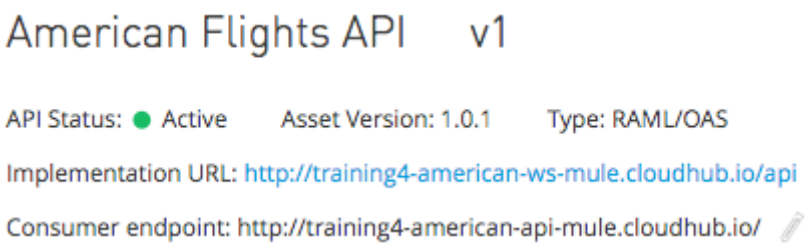
21. In the Publish API specification to Exchange dialog box, note the asset version and click Publish.



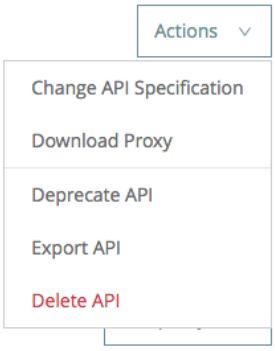
22. After the API is published, click Done in the Publish API specification to Exchange dialog box.

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.

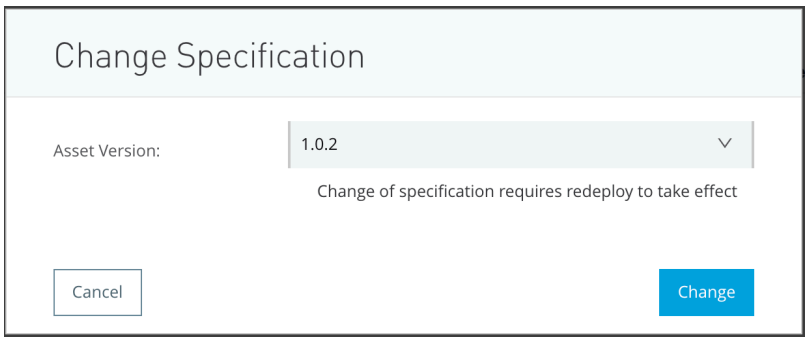


25. Click the Actions button in the upper-right corner and select Change API Specification.



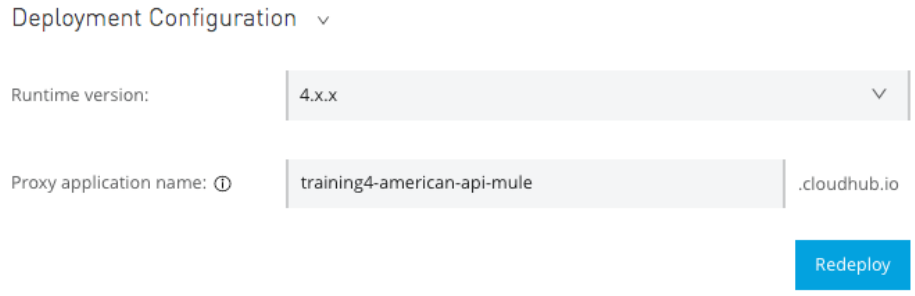
26. In the Change Specification dialog box, select the latest asset version, 1.0.2.

27. Click Change.

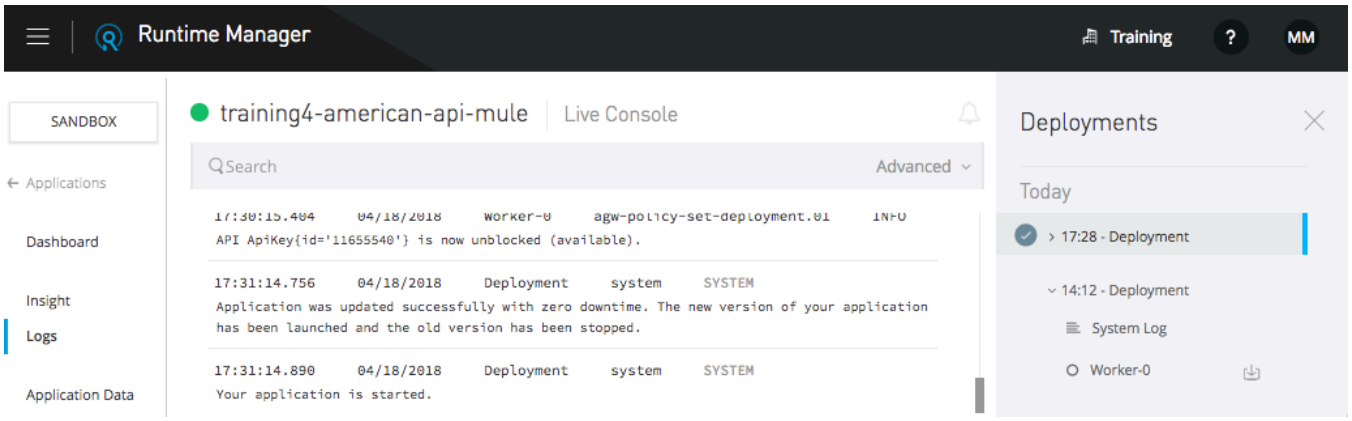


Redeploy a new proxy

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.



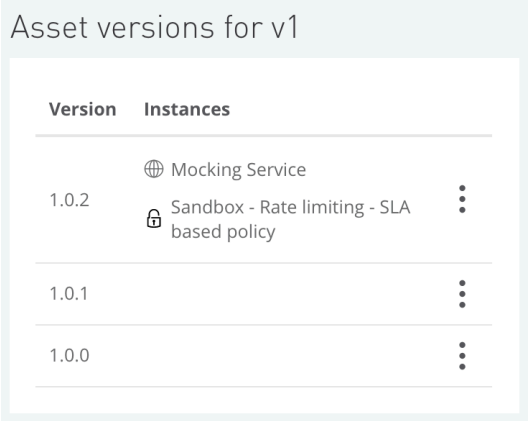
31. Click Redeploy.
32. In the Deploying to CloudHub dialog box, click the Click here link to see the logs.
33. Watch the logs and wait until the proxy application is redeployed.



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

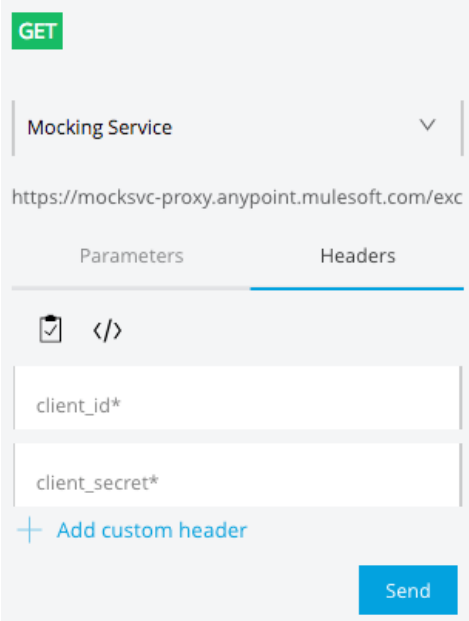
Test the rate limiting – SLA based policy in the API console in Exchange

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.



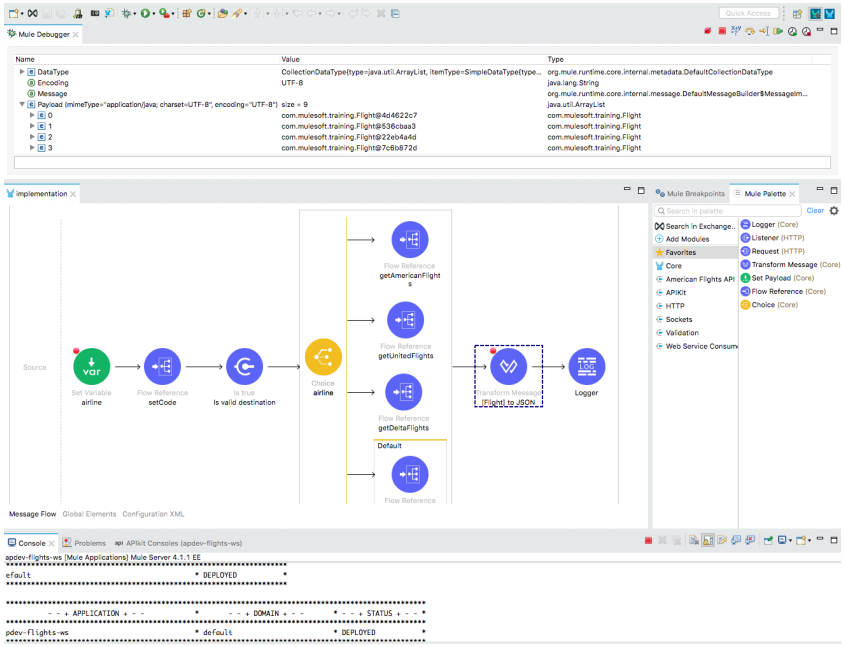
38. Click the GET method for the flights resource and select the Headers tab; you should see required text fields for client_id and client_secret and no longer need to add the headers manually for each request.

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



39. Close all Anypoint Platform browser windows and tabs.

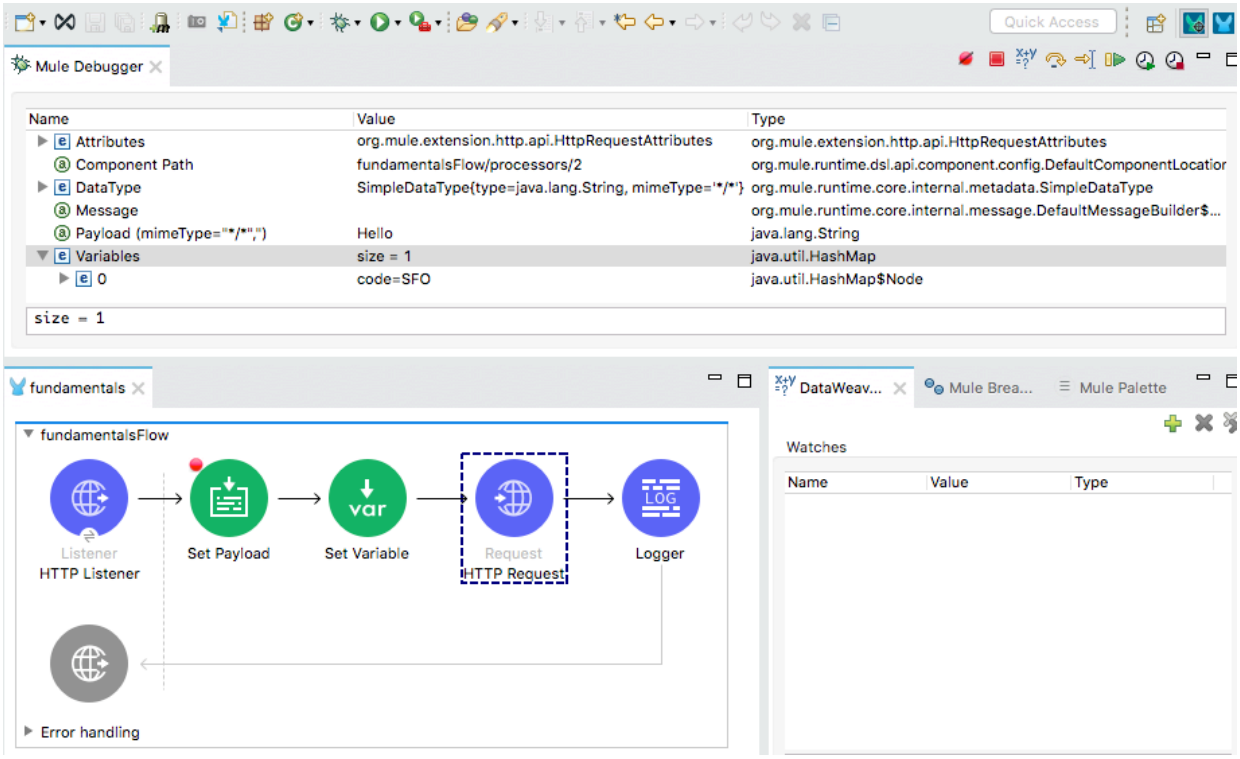
PART 2: Building Applications with Anypoint Studio



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



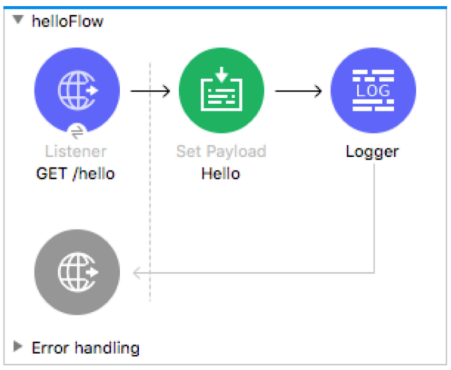
At the end this module, you should be able to:

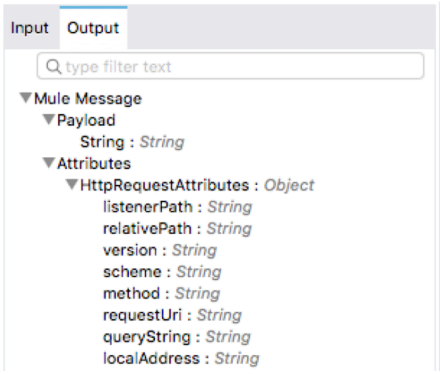
- Log event data.
- Debug Mule applications.
- Read and write event properties.
- Write expressions with the DataWeave expression language.
- Create variables.

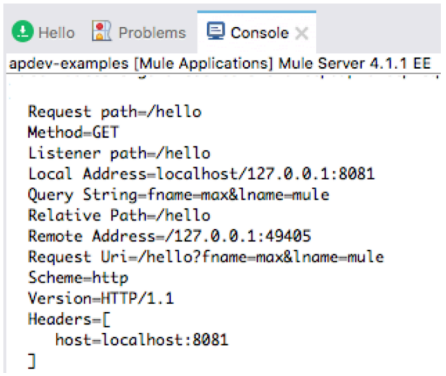
Walkthrough 6-1: View event data

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.

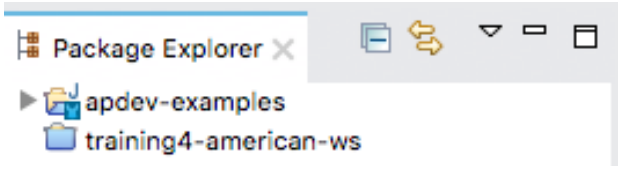






Create a new Mule project

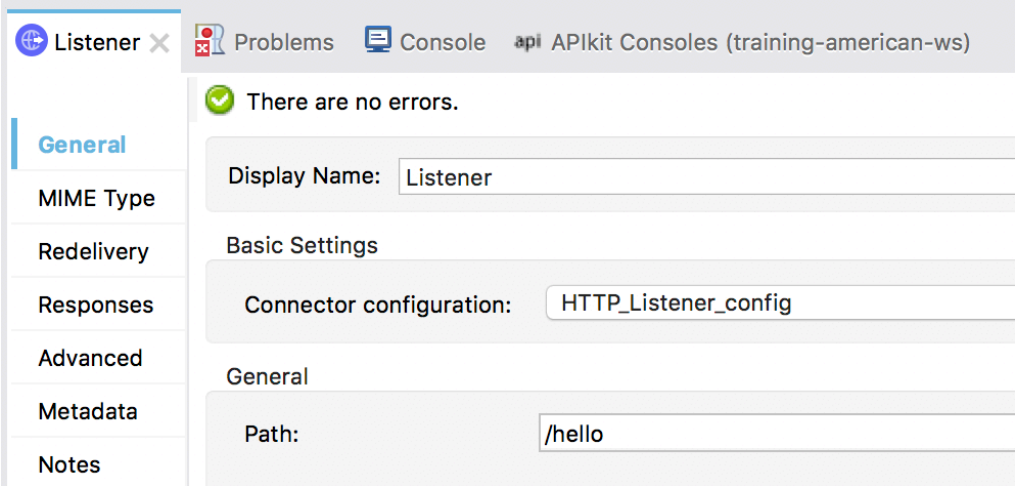
1. Return to Anypoint Studio.
2. Right-click training4-american-ws and select Close Project.
3. Select File > New > Mule Project.
4. Set the project name to apdev-examples and click Finish.



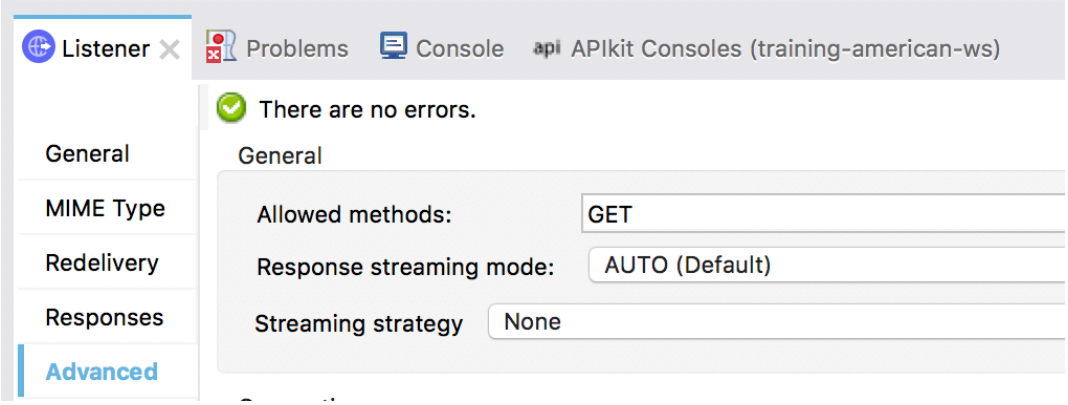
Create an HTTP Listener to receive requests

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.

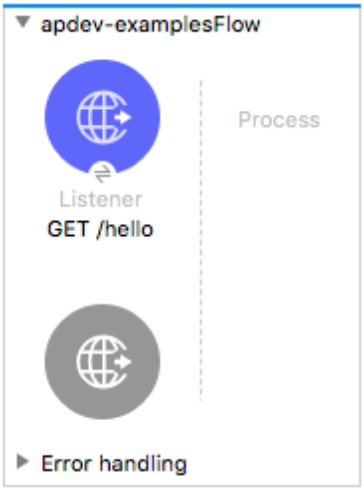
10. In the Listener properties view, set the path to /hello.



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.

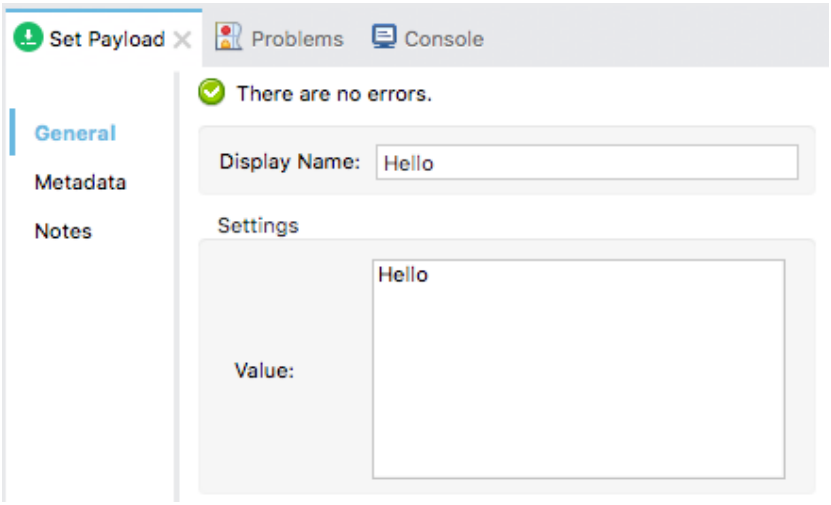


Change the flow name

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

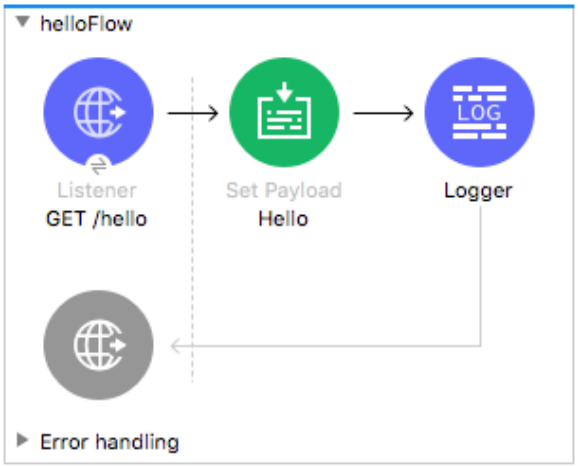
Set the message payload

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.



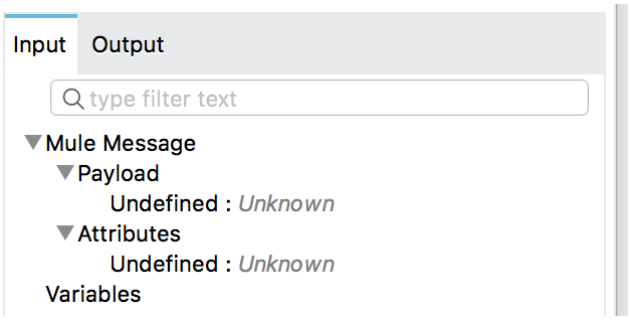
Add a Logger

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

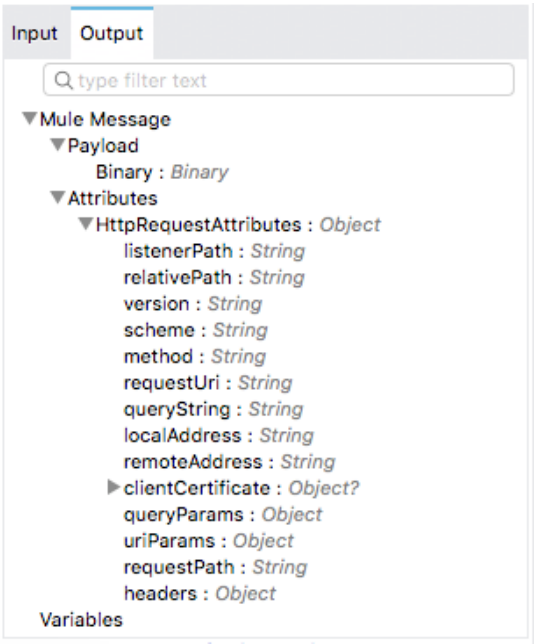


View event structure and metadata in the DataSense Explorer

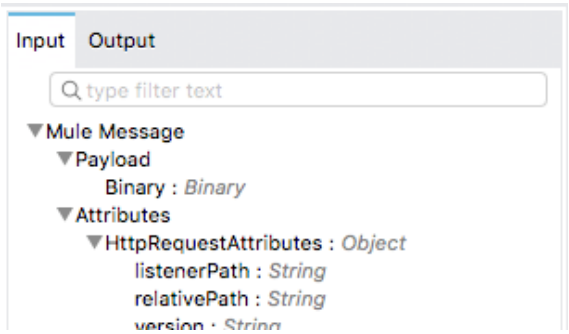
19. Select the GET /hello HTTP Listener and locate the DataSense Explorer in the right-side of its properties view.
20. Select the Input tab and expand Payload and Attributes.



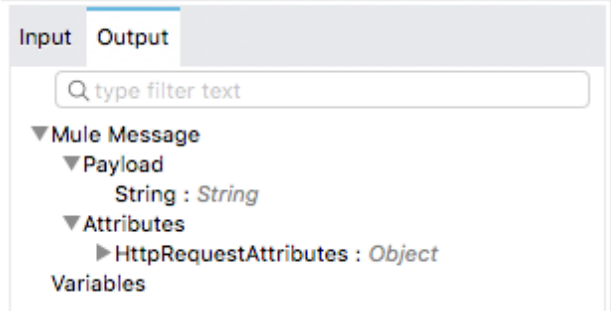
21. Select the Output tab and expand Payload and Attributes.



22. Select the Set Payload component in helloFlow.
23. In the DataSense Explorer, select the Input tab and expand Payload and Attributes.



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



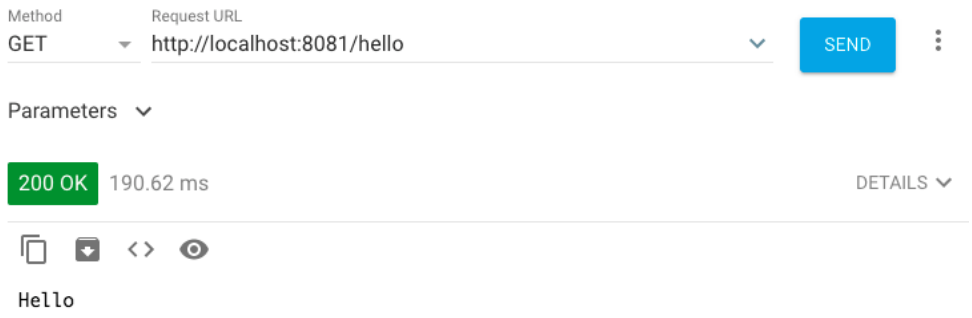
25. Select the Logger component in helloFlow.
26. In the DataSense Explorer, select the Input tab and expand Payload and Attributes.
27. Select the Output tab and expand the Payload and Attributes.

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.



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.

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.

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

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

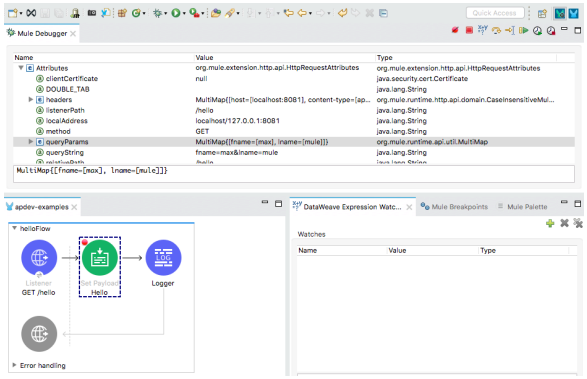
```
{
  payload=java.lang.String
  mediaType=*/*
  attributes=org.mule.extension.http.api.Ht
{
   Request path=/hello
   Method=GET
   Listener path=/hello
   Local Address=localhost/127.0.0.1:8081
   Query String=fname=max&lname=mule
   Relative Path=/hello
   Remote Address=/127.0.0.1:49405
   Request 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

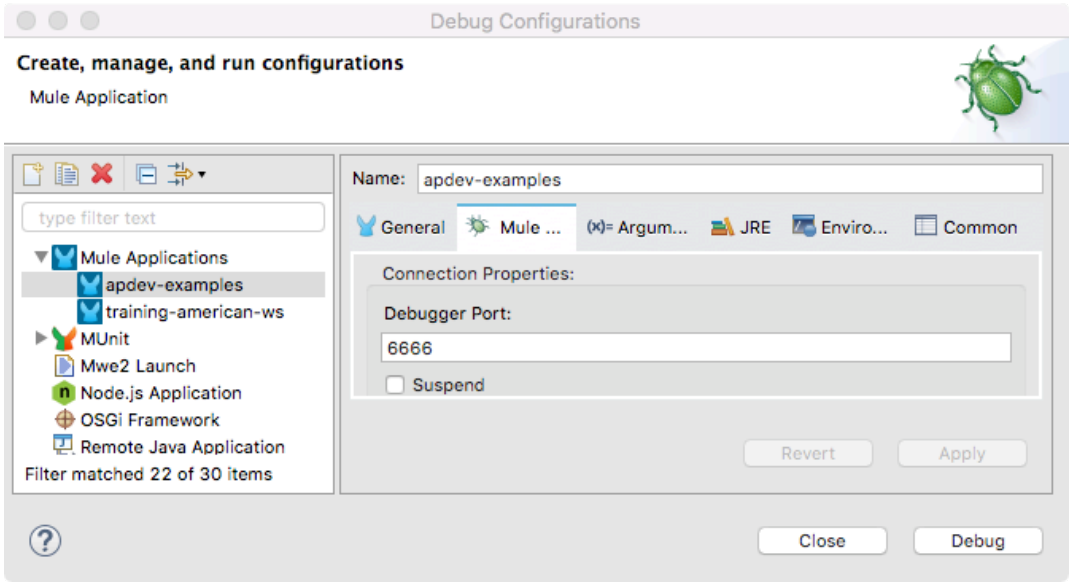
In this walkthrough, you debug and step through the code in a Mule application. You will:

- 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.

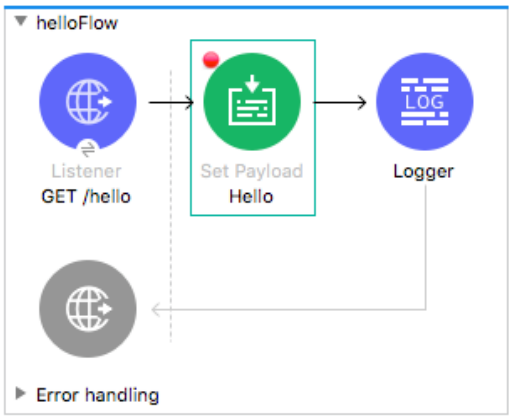


Note: If you know you have another program using port 6666 like McAfee on Windows, change this to a different value. Otherwise, you can test the Debugger first and come back and change the value here later if there is a conflict.

5. Click Close.

Add a breakpoint

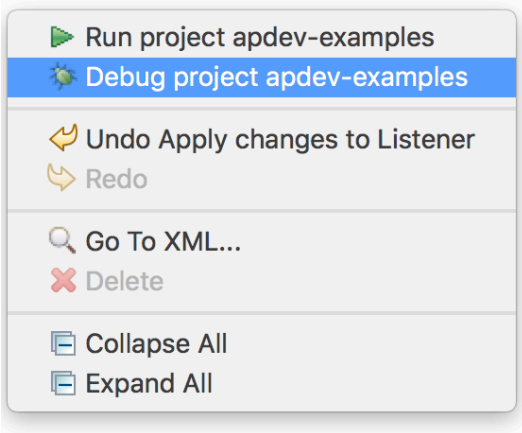
6. Right-click the Set Payload component and select Toggle breakpoint.



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.

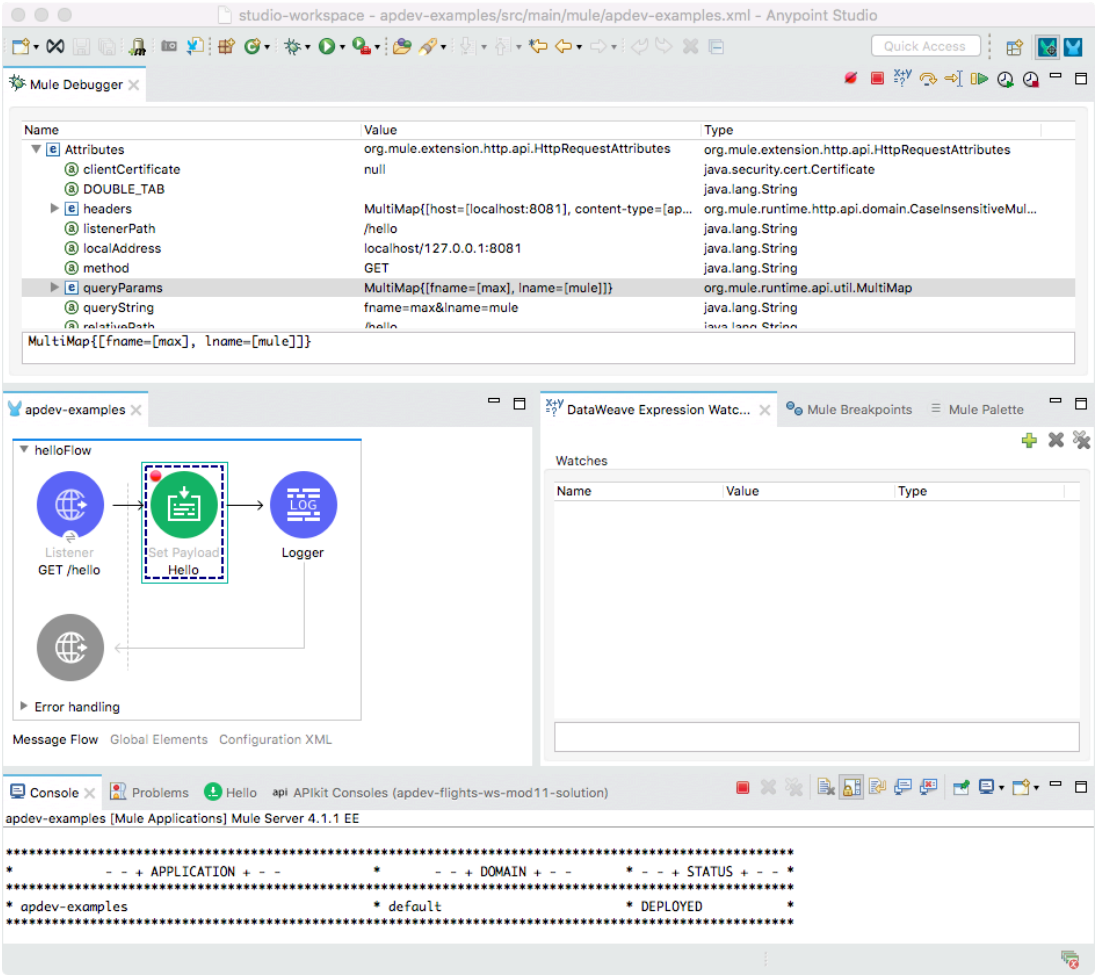


8. If you get a Confirm Perspective Switch dialog box, select Remember my decision and click Yes.
9. In the Debug perspective, close the MUnit view.
10. Look at the console and wait until the application starts.

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

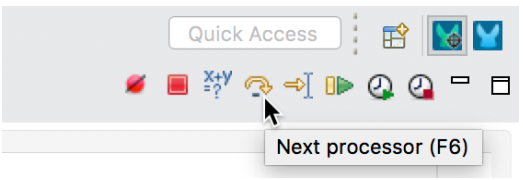
View event data in the Mule Debugger

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.

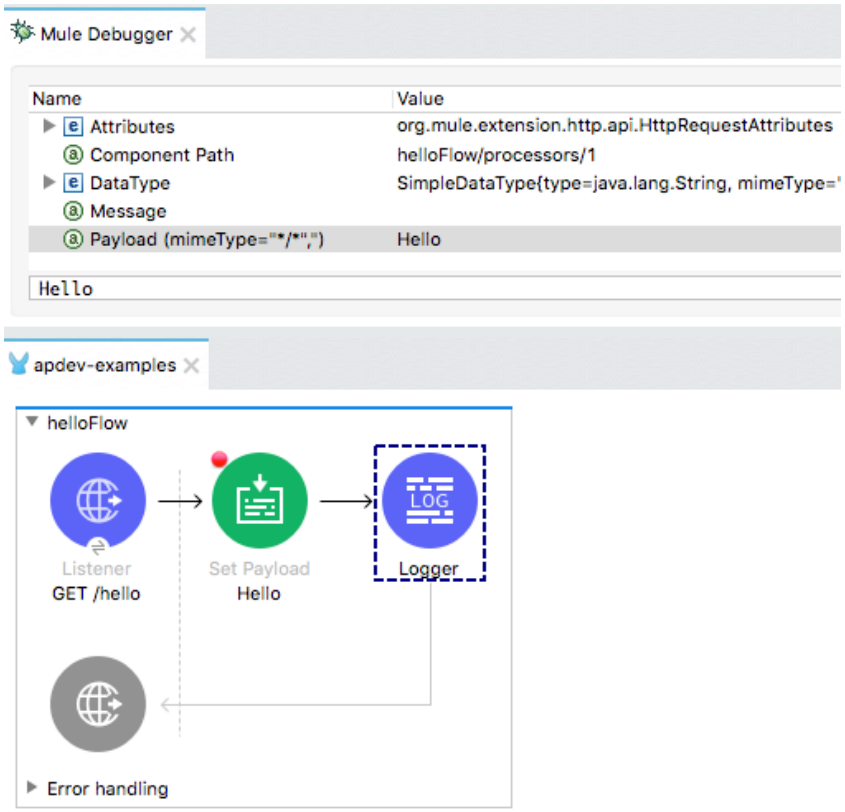


Step through the application

16. Click the Next processor button.



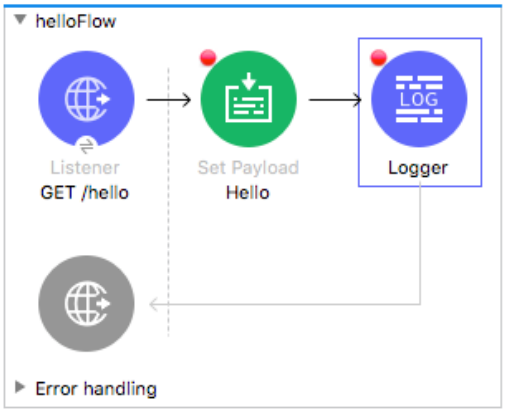
17. Look at the new value of the payload; it should be Hello.



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.

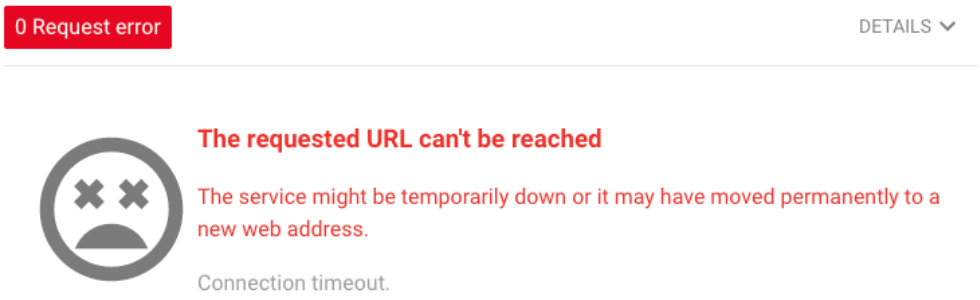


21. In Advanced REST Client, make another request to http://localhost:8081/hello?fname=max&lname=mule.

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.

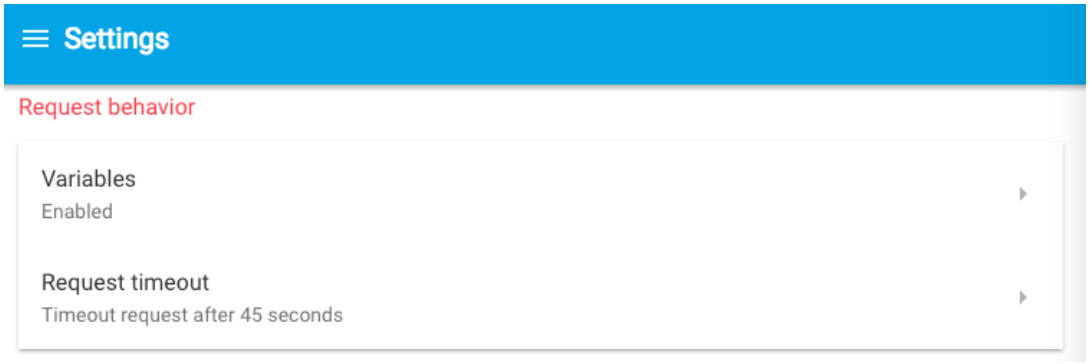
Cause the HTTP request to timeout

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.

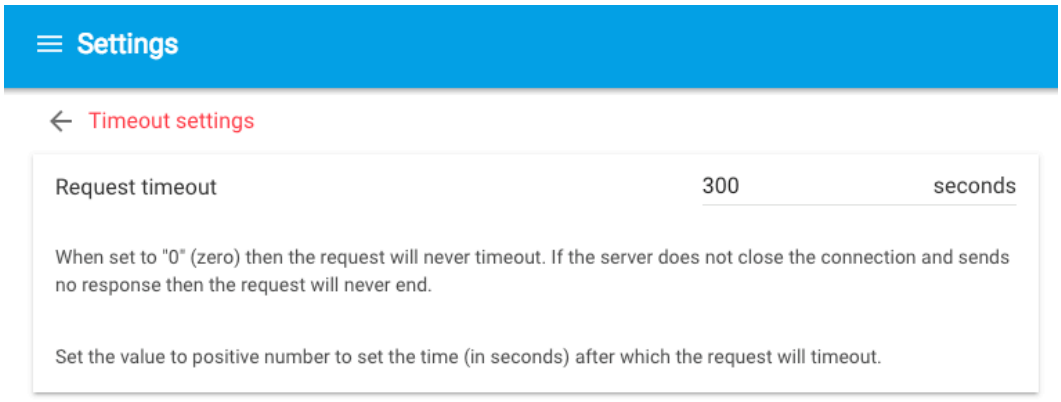


Increase the request timeout for Advanced REST Client

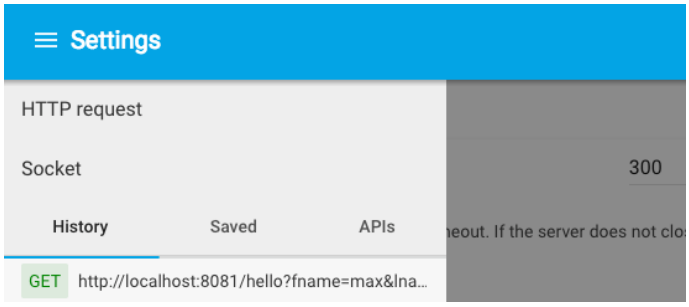
26. In the Advanced REST Client main menu, select Preferences.
27. In the Settings, locate the Request timeout setting and click the arrow next to it.



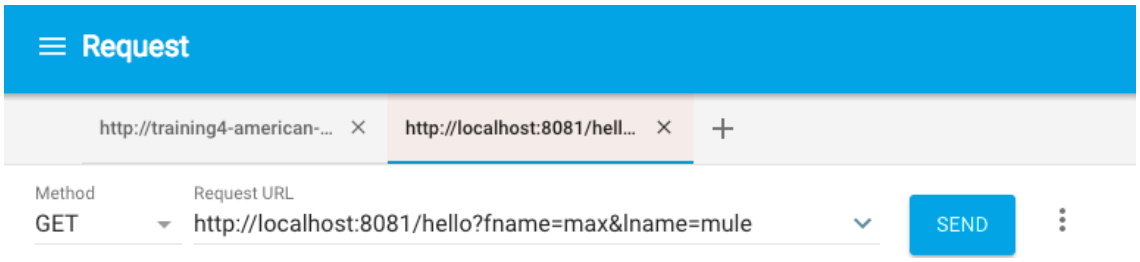
28. Change the request timeout to 300 seconds.



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

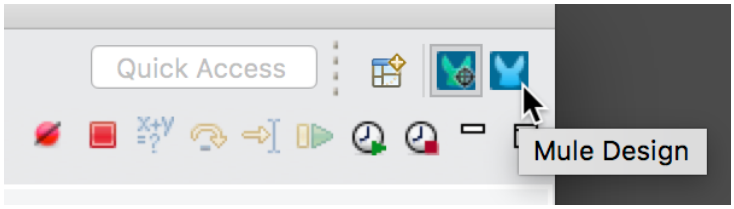


30. Click the Toggle Drawer button again to hide the drawer; you should now see the Request screen again.



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.

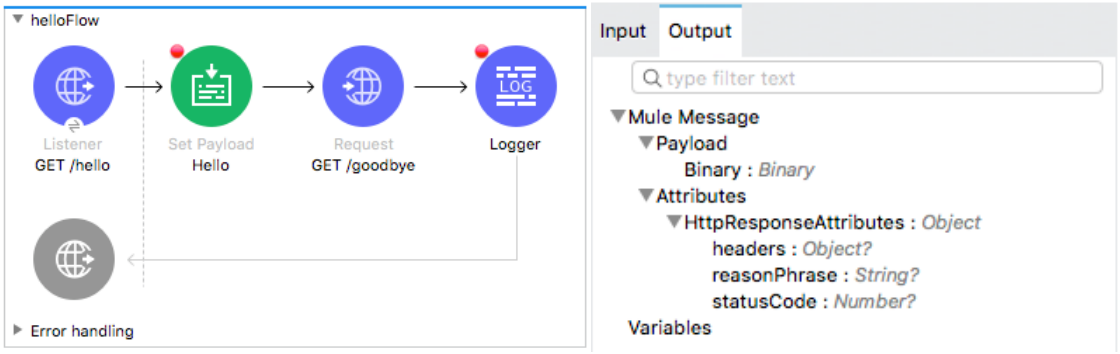
34. Leave the project running.

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 so you can watch event data as it moves in an out of a Mule application. You will:

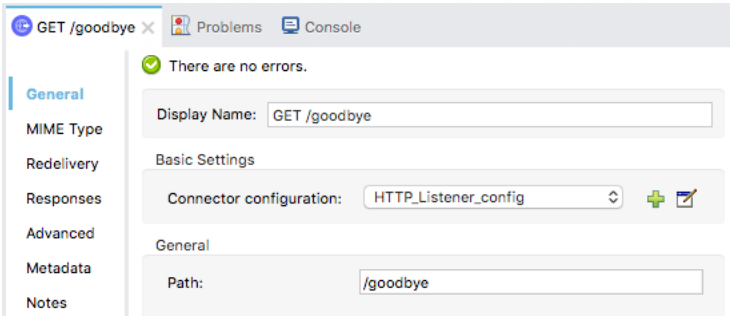
- Create a second flow with an HTTP Listener.
- Make an HTTP request from the first flow to the new HTTP Listener.
- View the event data as it moves through both flows.

  *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.*

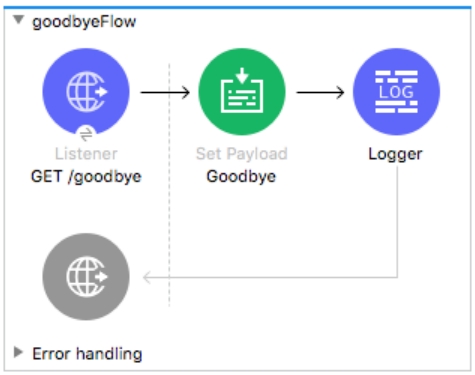


Create a second flow with an HTTP Listener

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.

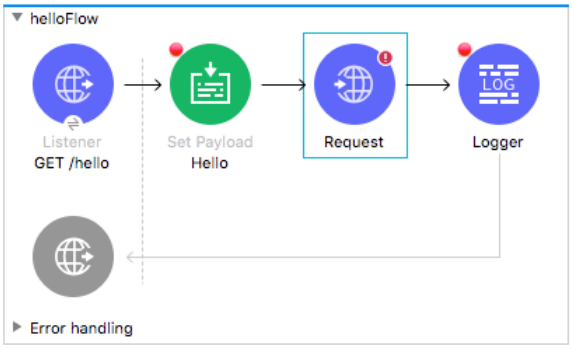


7. Add a Set Payload transformer and a Logger to the flow.
8. In the Set Payload properties view, set the display name and value to Goodbye.

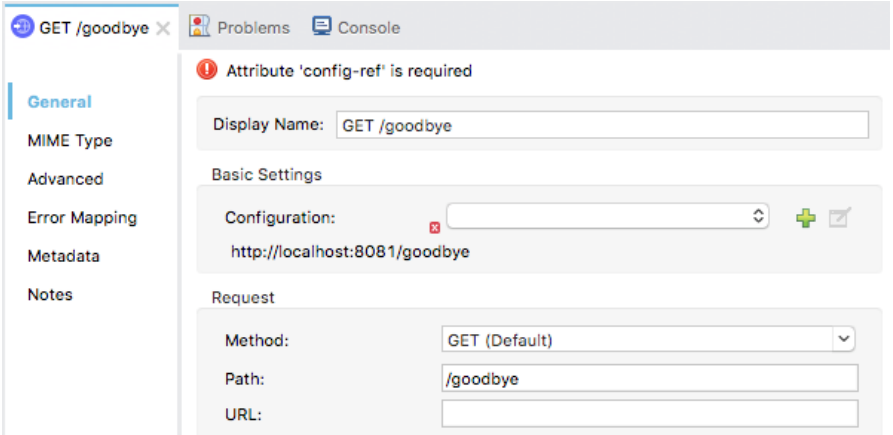


Make an HTTP request

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

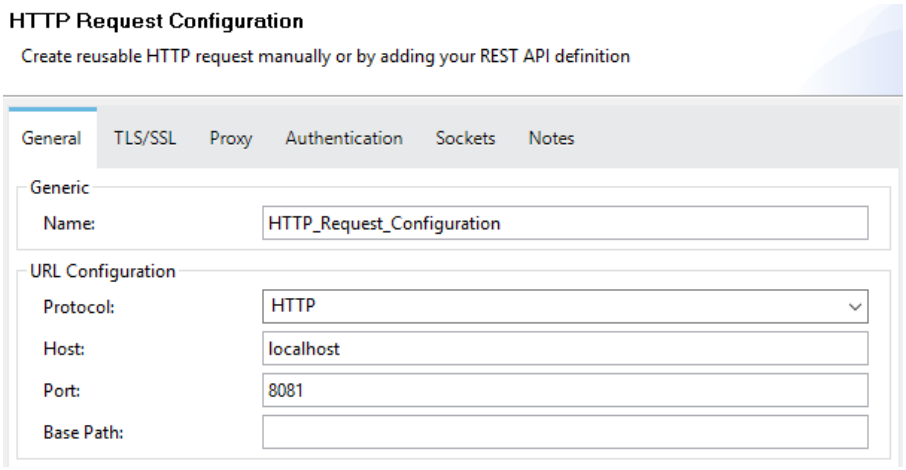


10. In the Request properties view, set the display name to GET /goodbye.
11. Set the path to /goodbye and leave the method set to GET.



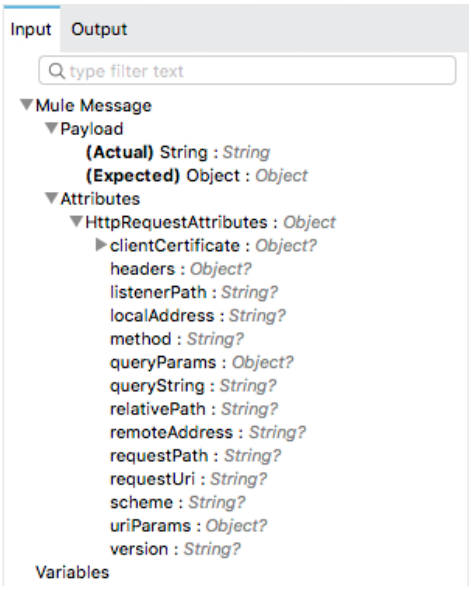
12. Click the Add button next to configuration.

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

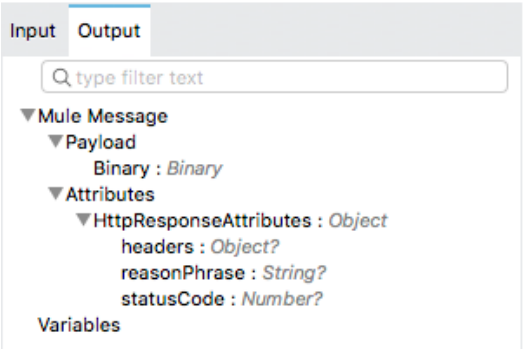


View event structure and metadata in the DataSense Explorer

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



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



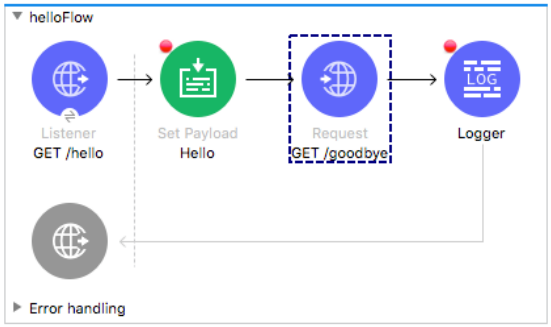
Change timeout for HTTP Request response

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.

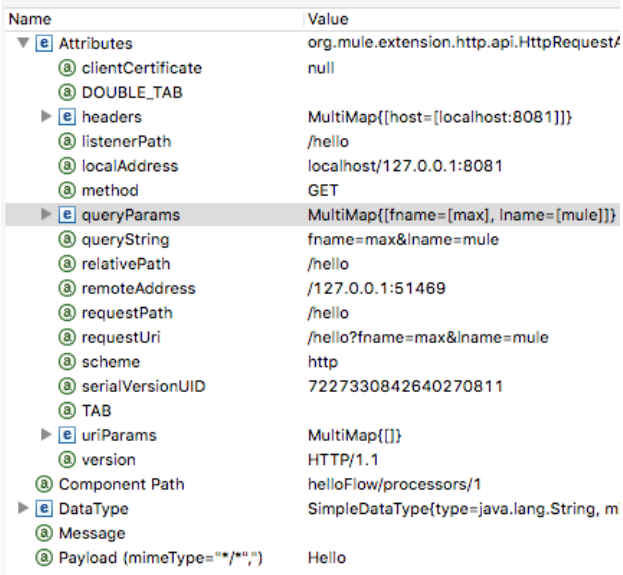
    Note: This is being set only for debugging purposes so that the HTTP Request does not timeout when you are stepping through the application and examining data in the Mule Debugger.

Debug the application

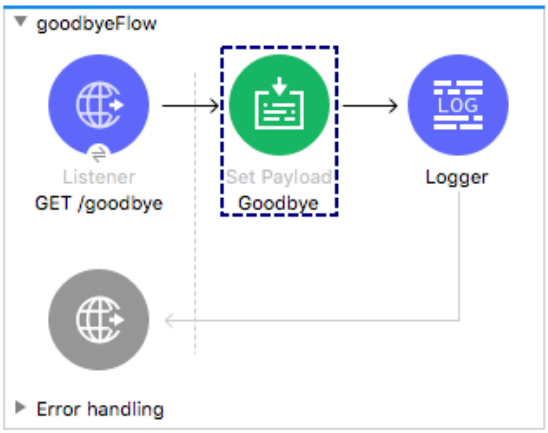
18. Save the file to redeploy the application.
19. In Advanced REST Client, send the same request to http://localhost:8081/hello?fname=max&lname=mule.
20. In the Mule Debugger, step to the GET /goodbye request.



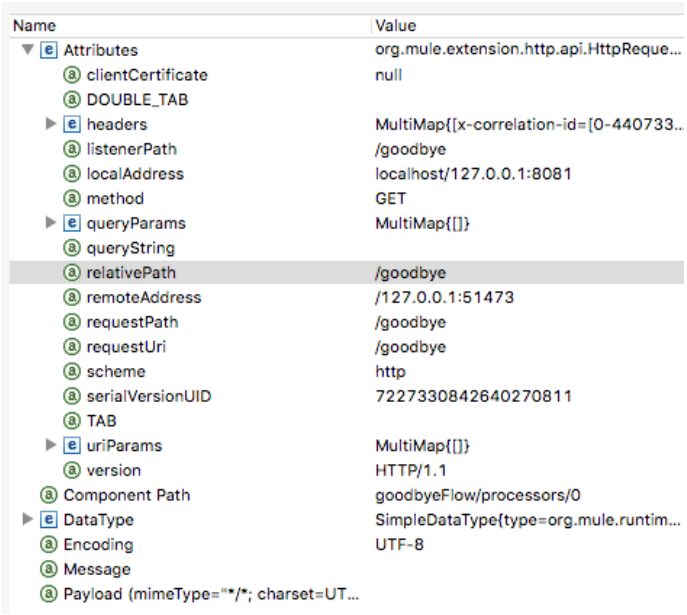
21. Look at the values of the payload and the attributes, including the queryParams.



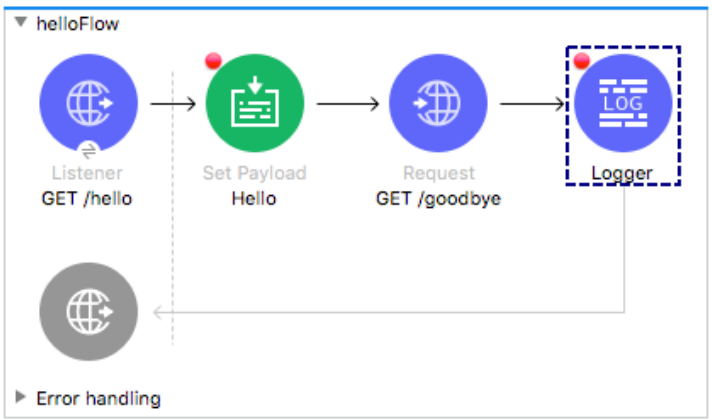
22. Step into goodbyeFlow.



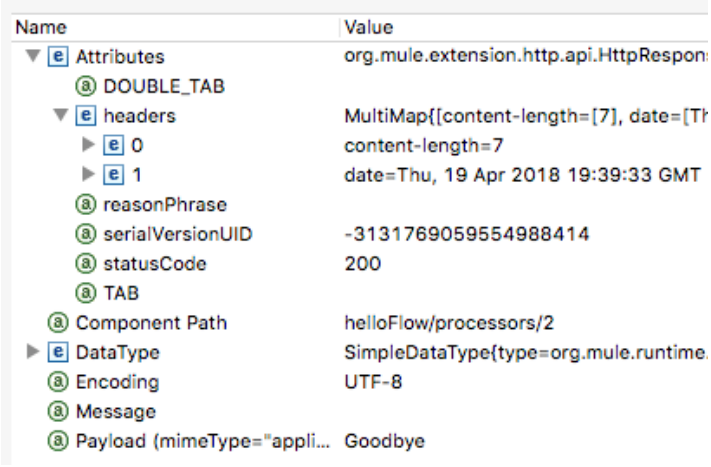
23. Look at the values of the payload and the attributes.



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



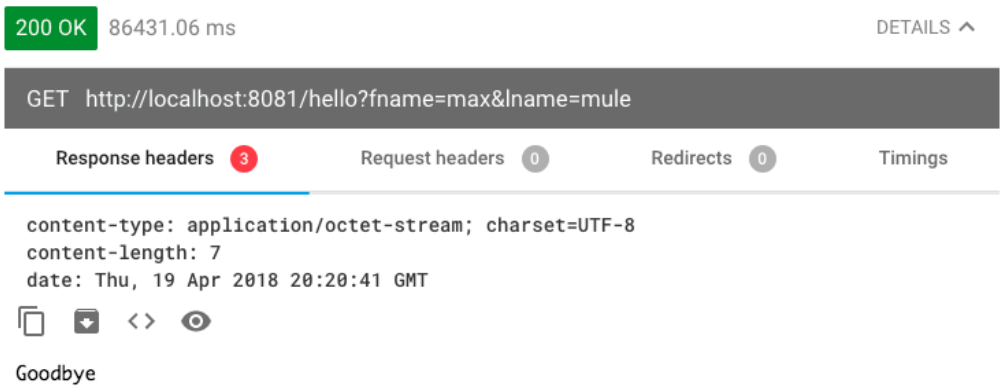
25. Look at the values of the payload and the attributes.



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.

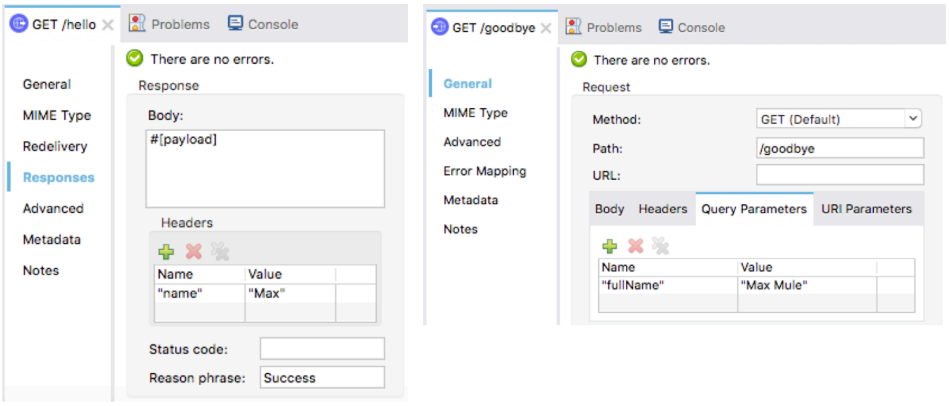


30. Return to Anypoint Studio and switch to the Mule Design perspective.

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:

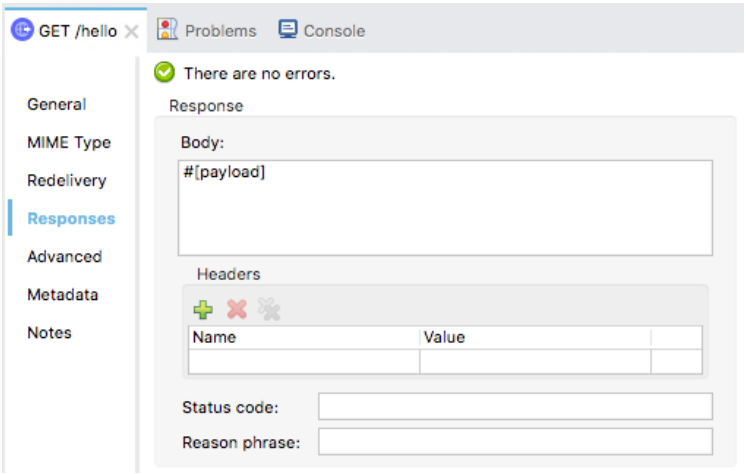
- View the default setting for a response body.
- Set a response header.
- View the default setting for a request body.
- Set a request query parameter.



View default response body

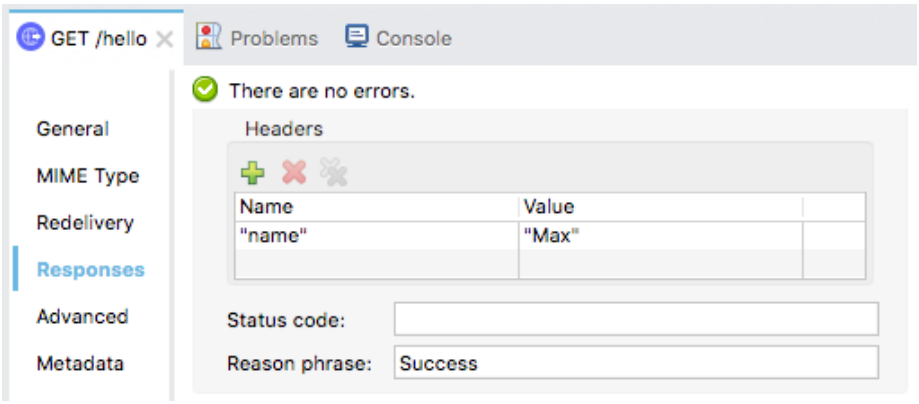
1. Return to apdev-examples.xml.
2. Double-click the GET /hello HTTP Listener in helloFlow.
3. In the GET /hello properties view, select the Responses tab.
4. In the Response section, locate the expression that sets the response body by default to the value of the payload.

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



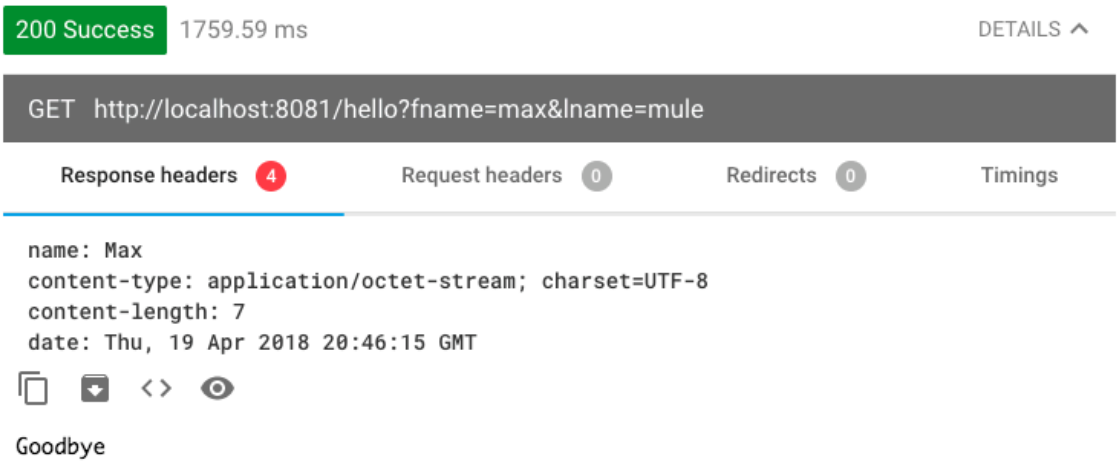
Set a response header

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.



Run the application and review response data

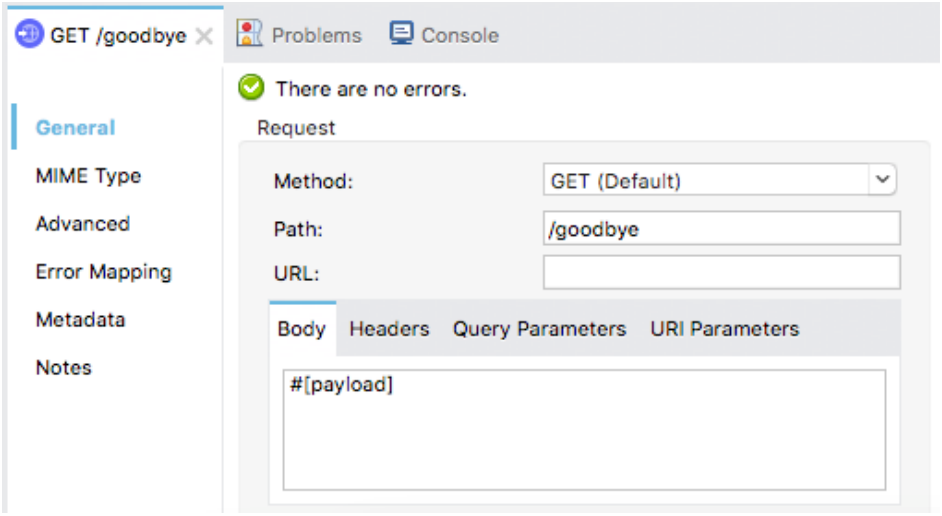
9. Save the file to deploy the project.
10. Return to Advanced REST Client and send the same request.
11. In the Mule Debugger, click Resume until you step through the application.
12. In Advanced REST Client, locate your new status code reason phrase.
13. Review the response headers; you should now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



View default request body

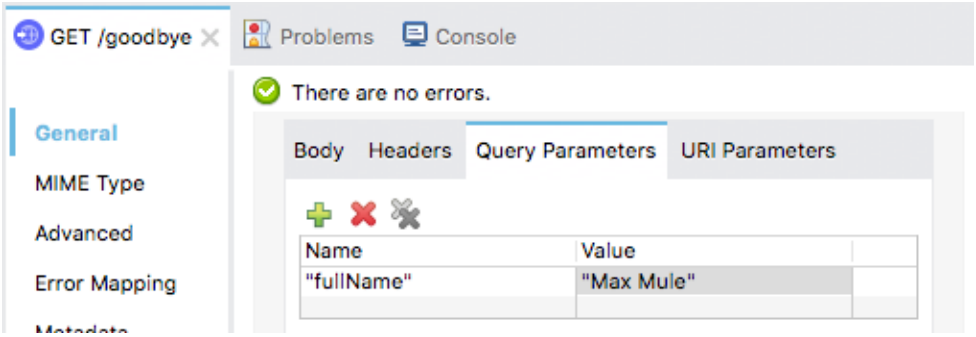
14. Return to Anypoint Studio and switch perspectives.
15. Double-click the GET /goodbye HTTP Request in helloFlow.

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.



Set a request query parameter

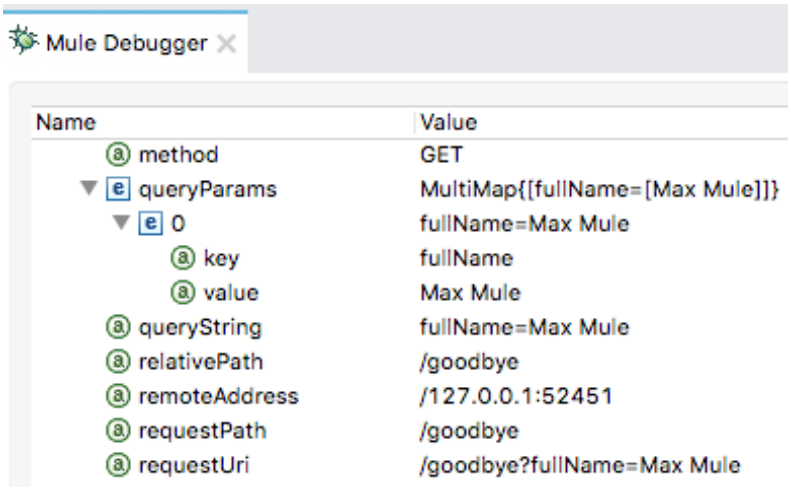
18. Select the Query Parameters tab and click the Add button.
19. Set the name to "fullName" and the value to "Max Mule".



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.

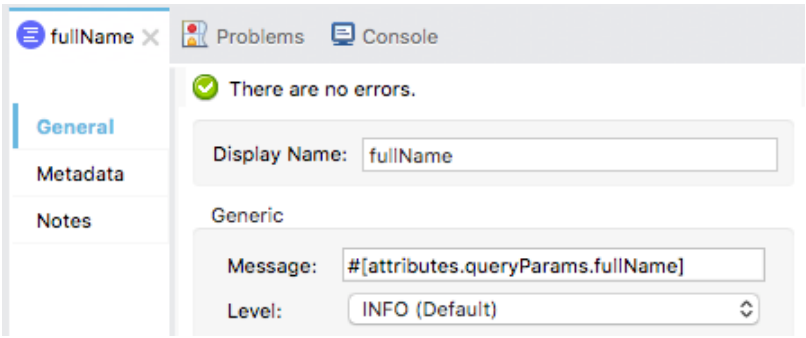


24. Step through the rest of the application.
25. Switch to the Mule Design perspective.
26. Stop the project.

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:

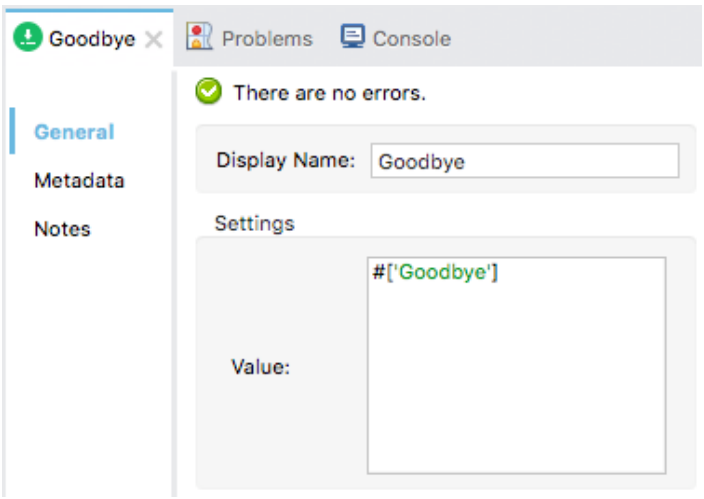
- Use expressions to set the payload and a logged value.
- Use expressions to set a response header and a request query parameter.
- In expressions, reference values for the event payload and attributes.
- Use the DataWeave upper() function and the concatenation, as, and default operators.



Use an expression to set the payload

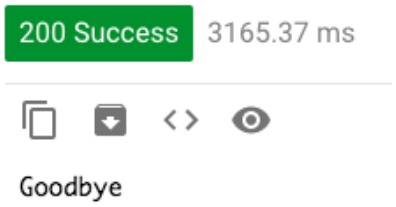
1. Return to apdev-examples.xml.
2. Navigate to the properties view for the Goodbye Set Payload transformer in goodbyeFlow.
3. Change the value to an expression.

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



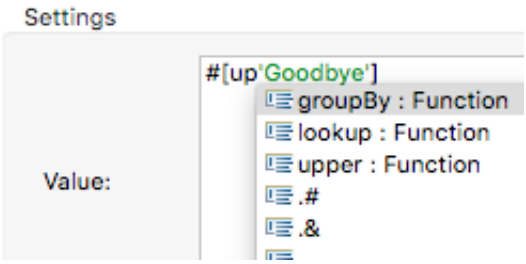
4. Run the project.

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



Use a DataWeave function

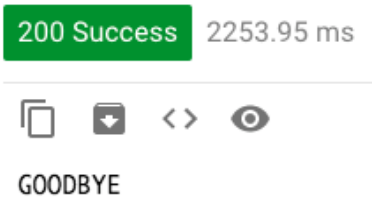
6. In Anypoint Studio, return to the Goodbye Set Payload properties view.
7. Inside the brackets and before the string, type the word up and press Ctrl+Spacebar.
8. In the auto-completion menu, select the upper function.



9. Add parentheses around the Goodbye string.

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

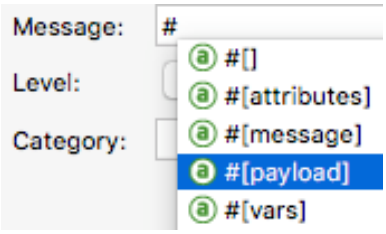
10. Save the file to redeploy the application.
11. In Advanced REST Client, send the same request; the return string should now be in upper case.



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: 0-216c8
710-4416-11e8-861e-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-d1a23
d00-4416-11e8-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]
```

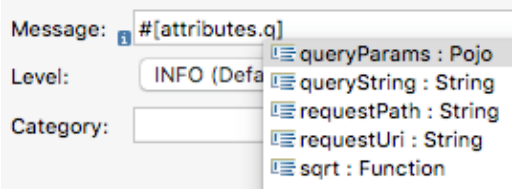
27. Save the file to redeploy the application.

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]
```

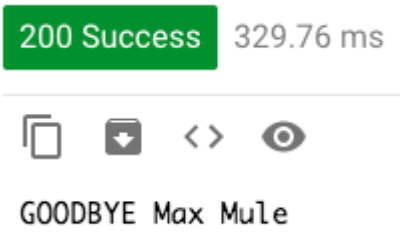
43. Review the error.

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]
```

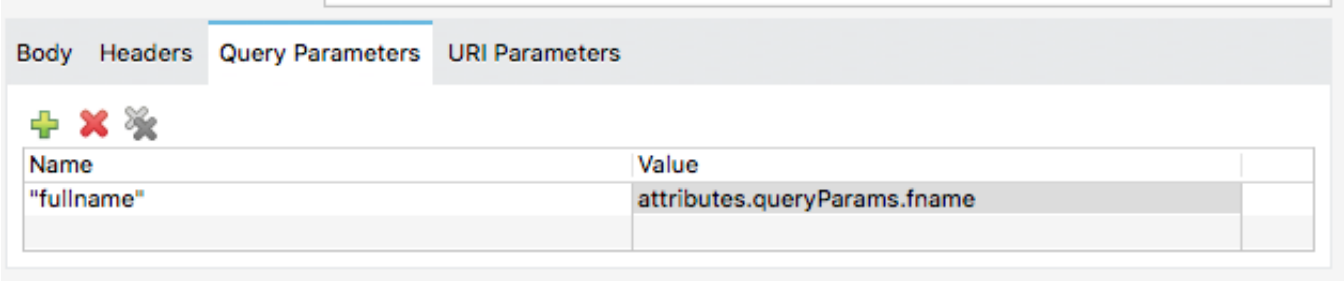
45. Click Apply Changes to redeploy the application.
46. In Advanced REST Client, send the same request; you should now also see the name displayed.



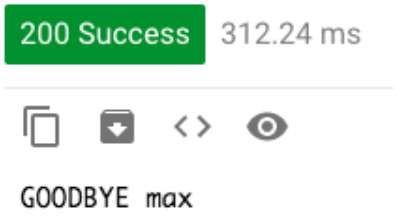
Use an expression to set a request header

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.

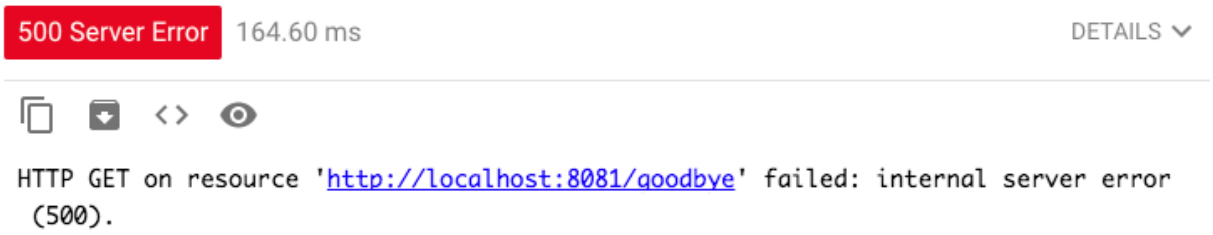


50. Click Apply Changes to redeploy the application.
51. In Advanced REST Client, send the same request; you should now see the name of the fname query parameter displayed.



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.



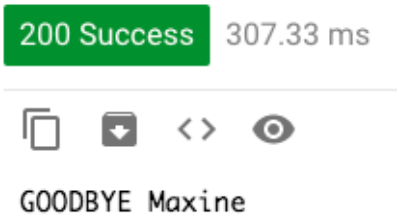
Set a default value in an expression

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.

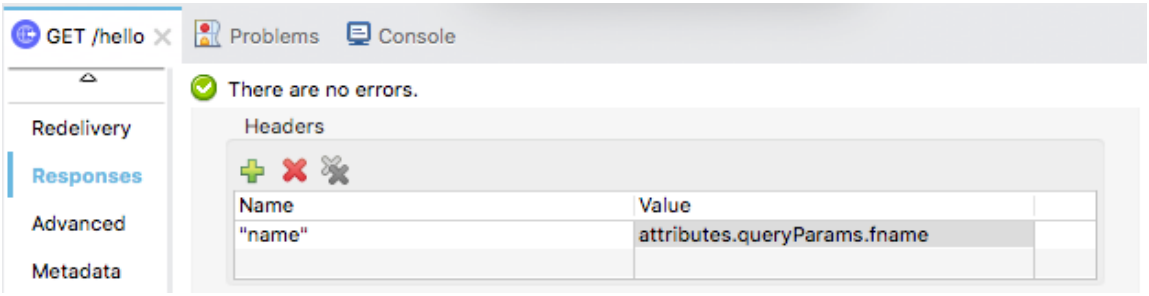
57. In Advanced REST Client, send the same request; you should now see the default value Maxine displayed.



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.

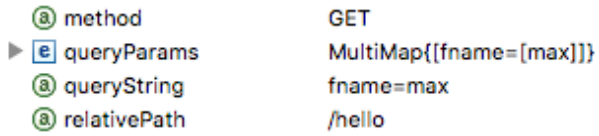
```
attributes.queryParams.fname
```



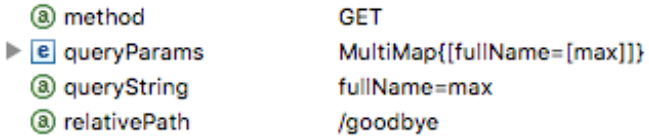
61. Click Apply Changes to redeploy the application.
62. In Advanced REST Client, add a fname and set it equal to max or some other value and send the request.
63. Look at the response headers; you should no longer see a name header.

Debug and verify the query parameter is sent with the request

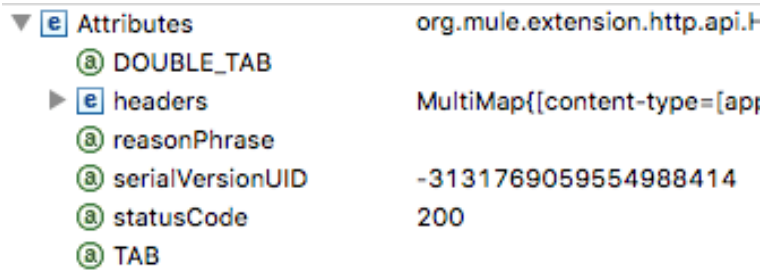
64. Return to Anypoint Studio.
65. Stop and then debug the project.
66. Return to Advanced REST Client and send the same request.
67. Return to the Mule Debugger and look at the attributes and query parameters; you should see the fname query parameter.



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



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

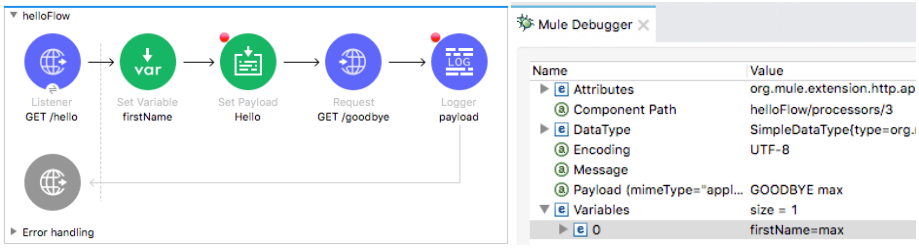


70. Step through the rest of the application.
71. Switch to the Mule Design perspective.

Walkthrough 6-6: Set and get variables

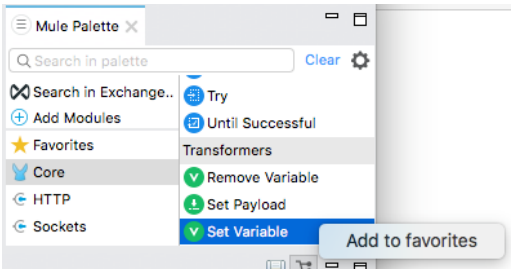
In this walkthrough, you create variables associated with an event. You will:

- 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.



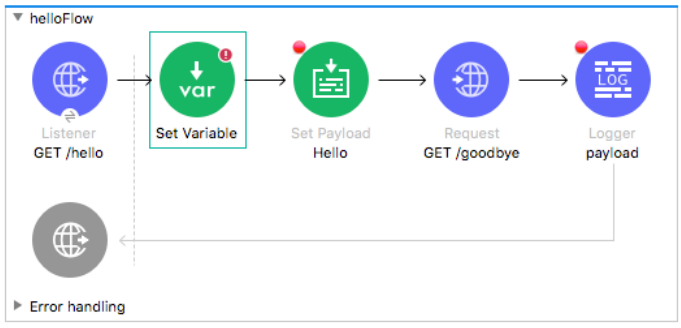
Add the Set Variable transformer to the Favorites section of the Mule Palette

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.

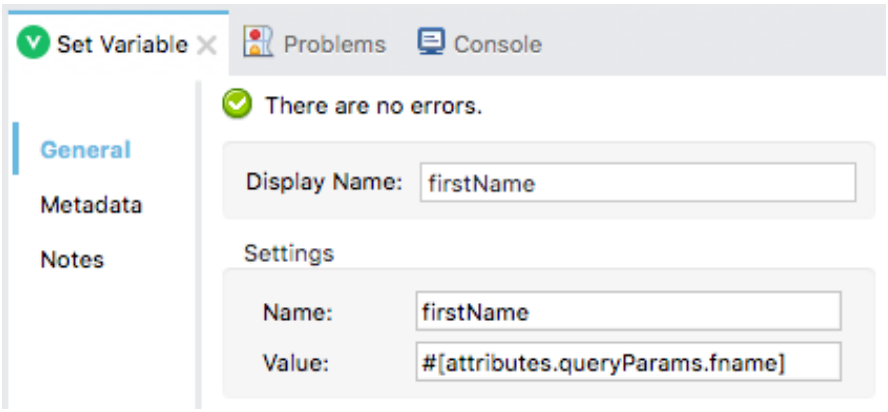


Create a variable

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.



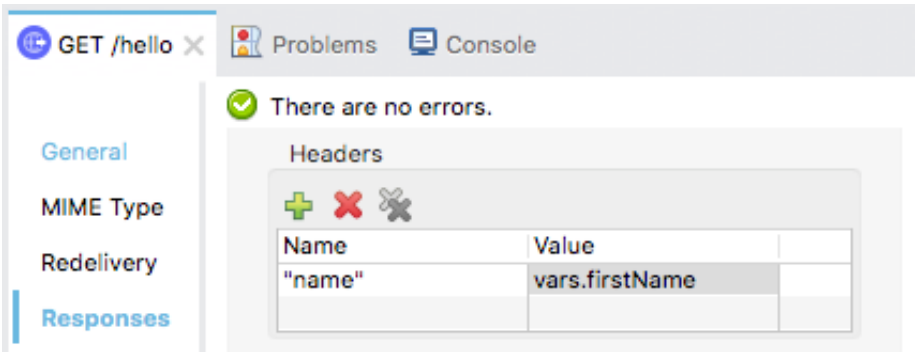
5. In the Set Variable properties view, set the display name and name to firstName.
6. Set the value to your query parameter, fname.



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.

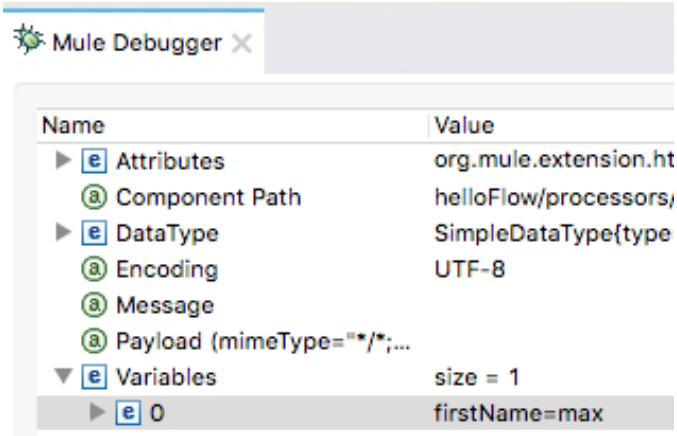
```
vars.firstName
```



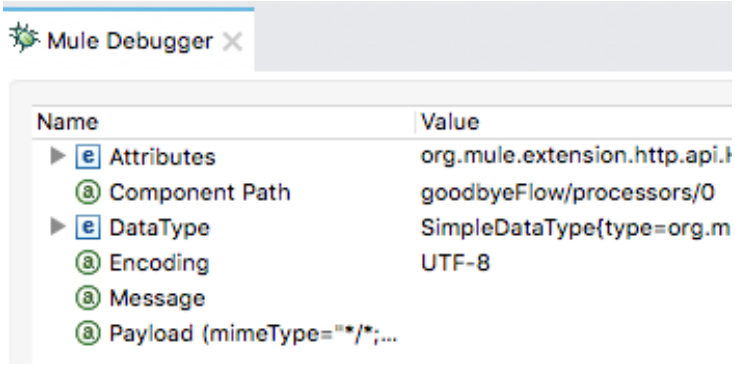
View variables in the Mule Debugger

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

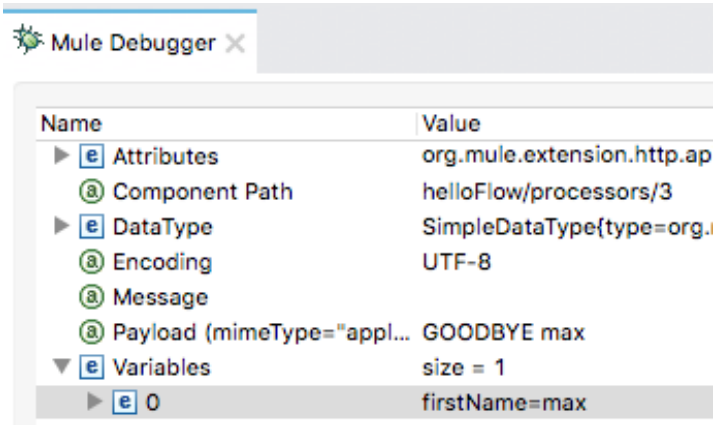
12. In the Mule Debugger, locate and expand the new Variables section; you should see your firstName variable.



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

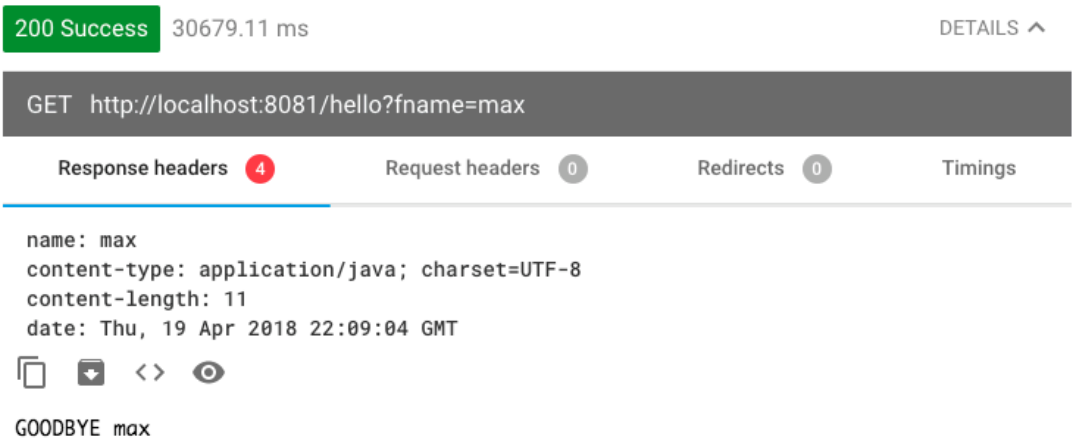


14. Step back to helloFlow; you should see the Variables section again with your firstName variable.

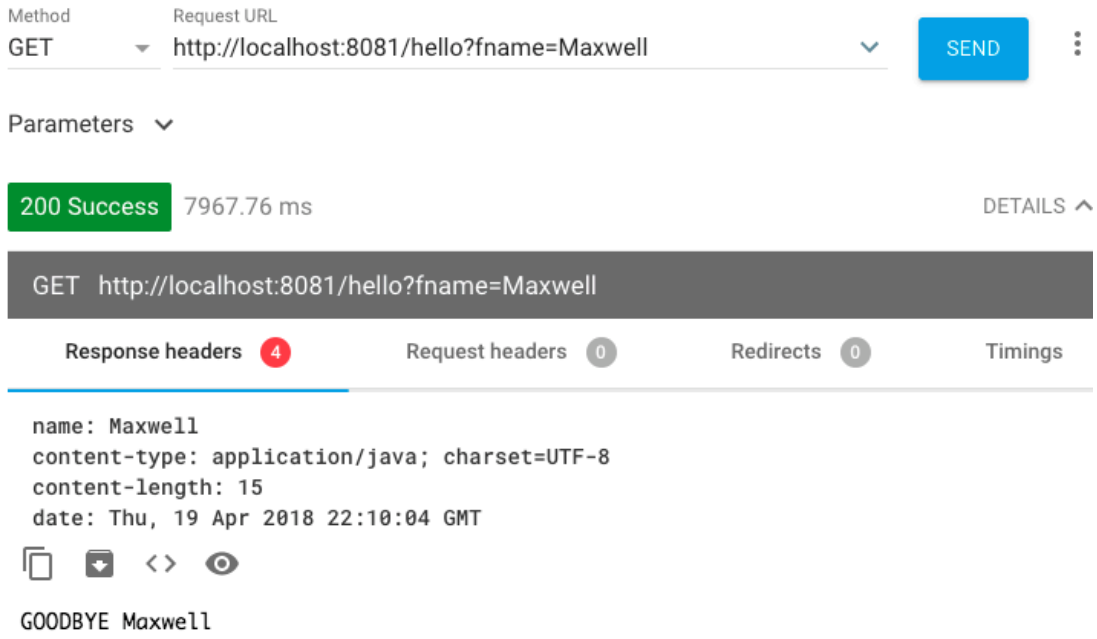


15. Step through the rest of the application.

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

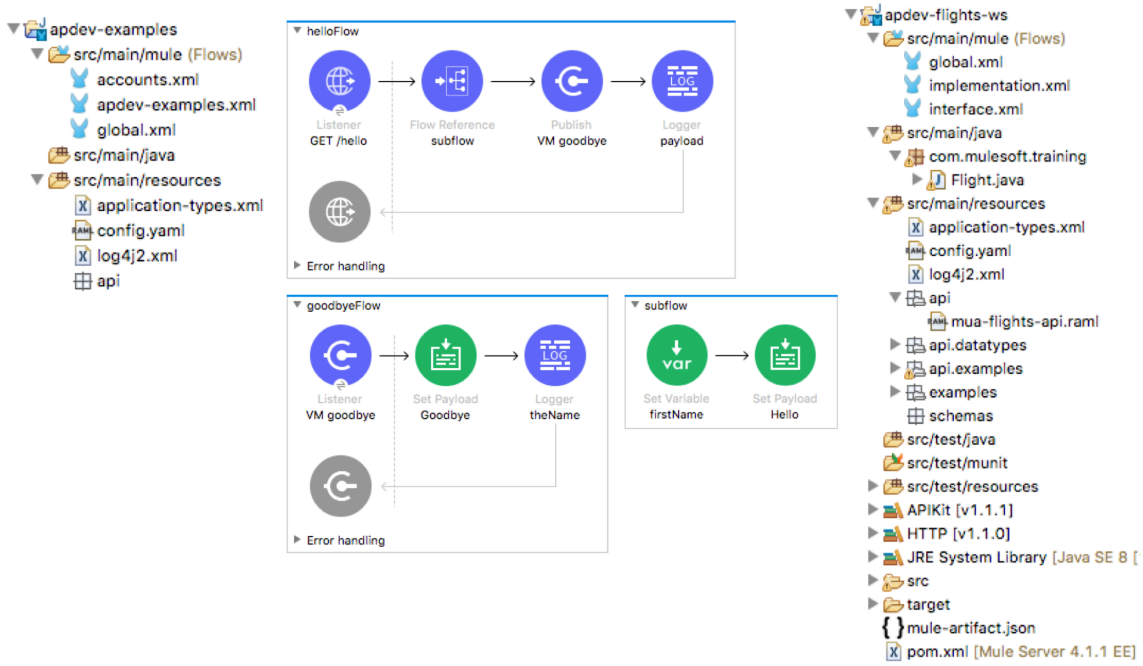


17. Change the value of the query parameter and send another request.
18. In the Mule Debugger, click the Resume button until you step through the application.
19. Return to Advanced REST client; you should see the new query parameter value returned in both the body and the header.



20. Return to Anypoint Studio and switch perspectives.

Module 7: Structuring Mule Applications



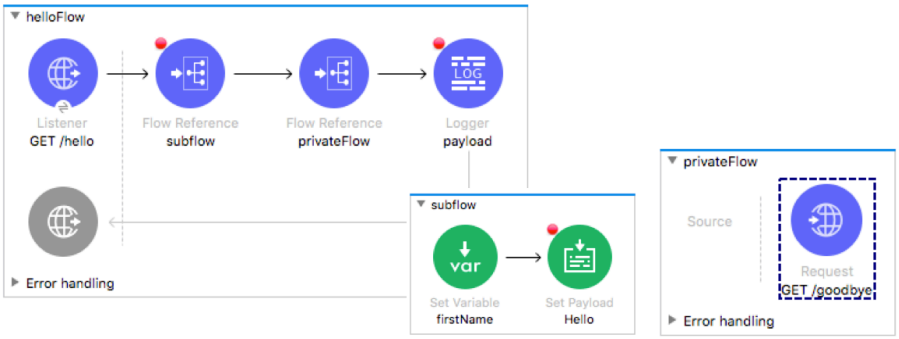
At the end of this module, you should be able to:

- 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

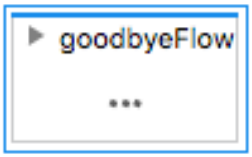
In this walkthrough, you continue to work with apdev-examples.xml. You will:

- 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

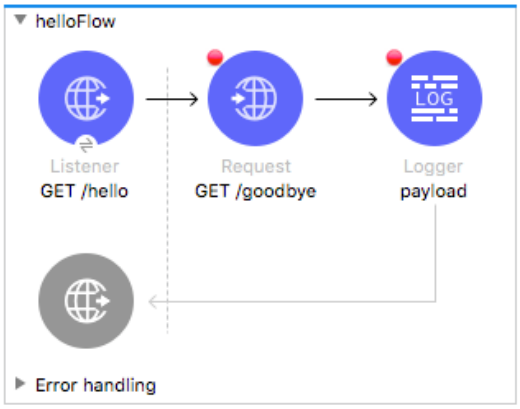
1. Return to apdev-examples.xml in Anypoint Studio.
2. Click the arrow to the left of the goodbyeFlow to collapse it.

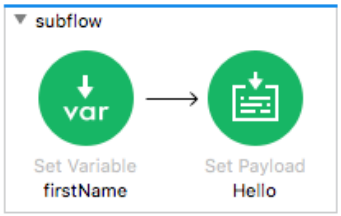


Create a subflow

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.

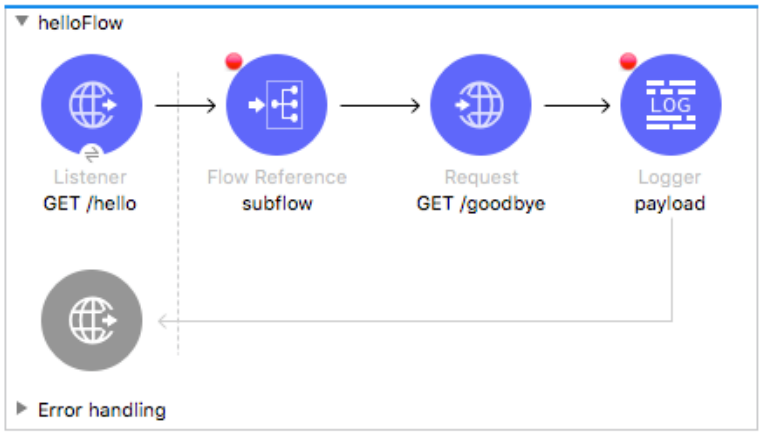
6. Select the Set Variable and Set Payload transformers in helloFlow and drag them into the subflow.





Reference a subflow

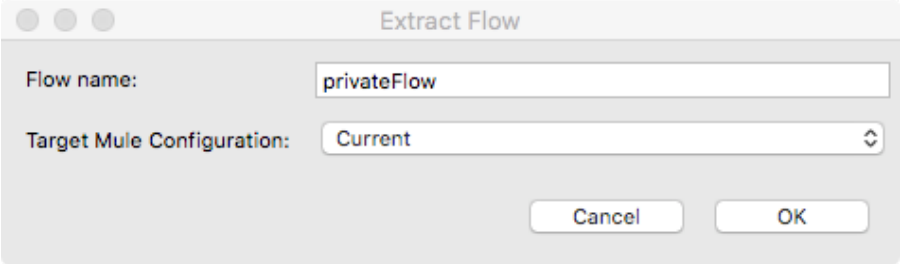
7. Drag a Flow Reference component from the Mule Palette and drop it into helloFlow between the GET /hello HTTP Listener and the GET /goodbye HTTP Request.
8. In the Flow Reference properties view, set the flow name to subflow.
9. Add a breakpoint to the subflow Flow Reference.
10. Remove the breakpoint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



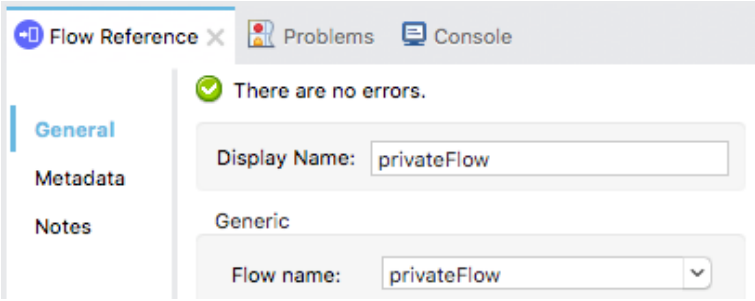
Extract processors into a subflow

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

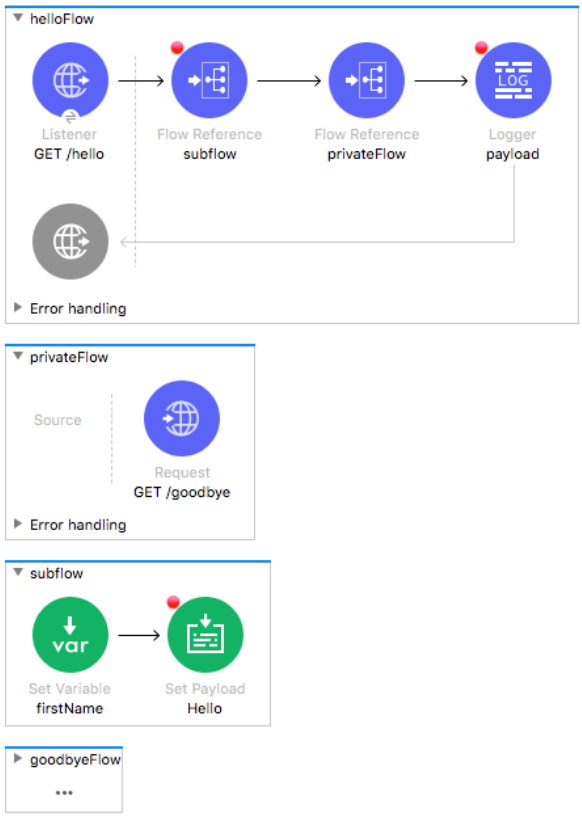
12. In the Extract Flow dialog box, set the flow name to privateFlow.



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.

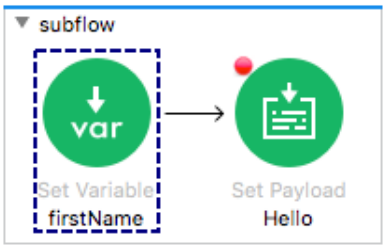


15. Drag privateFlow above subflow in the canvas.

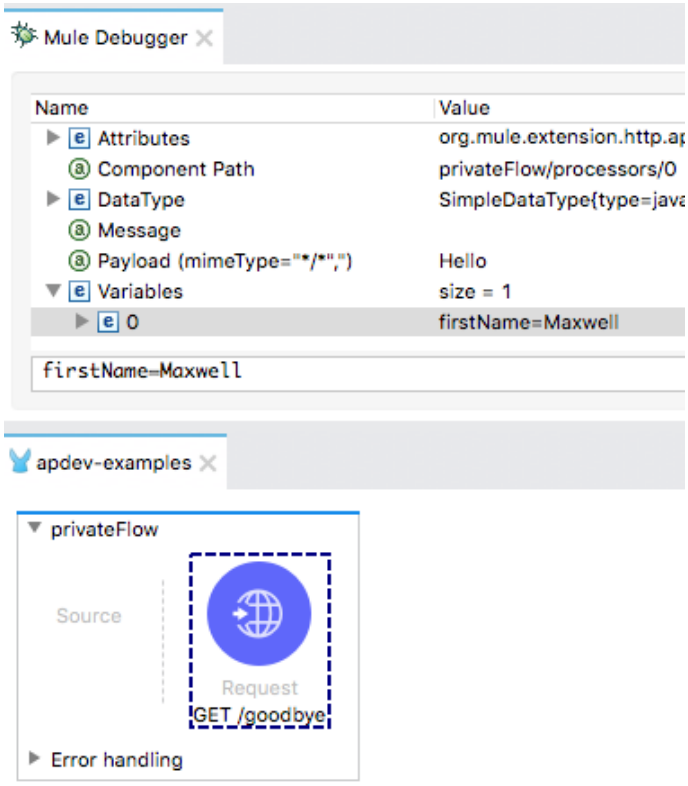


Debug the application

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.

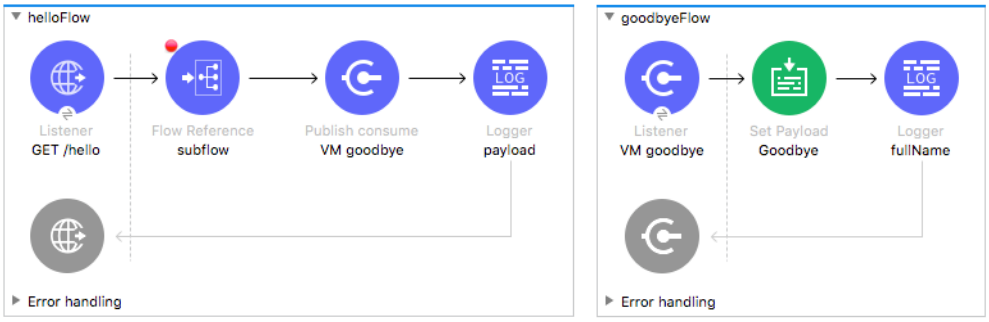


22. Step through the rest of the application.
23. Switch to the Mule Design perspective.

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

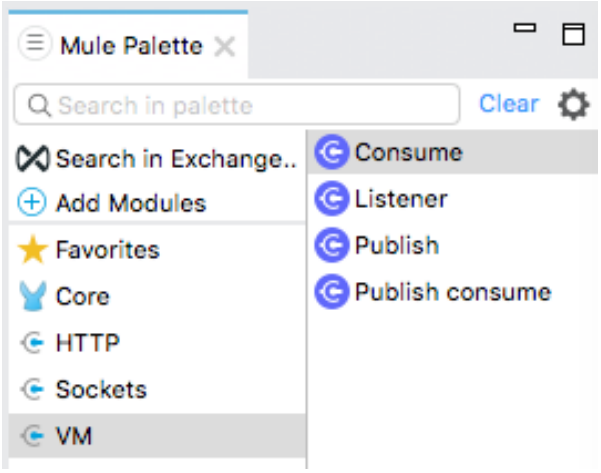
In this walkthrough, you pass messages between flows using asynchronous queues. You will:

- 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.

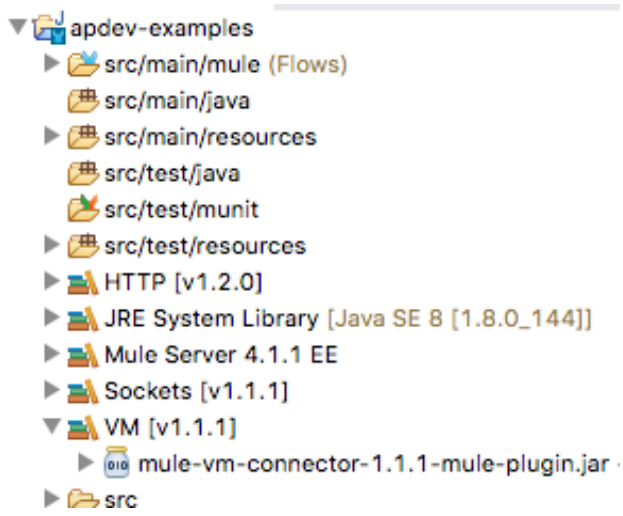


Add the VM module to the project

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.

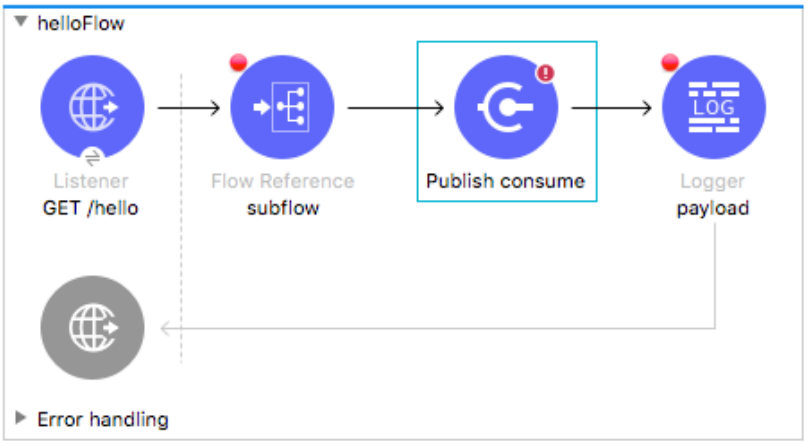


4. In the Project Explorer, locate the JAR file for the VM connector.

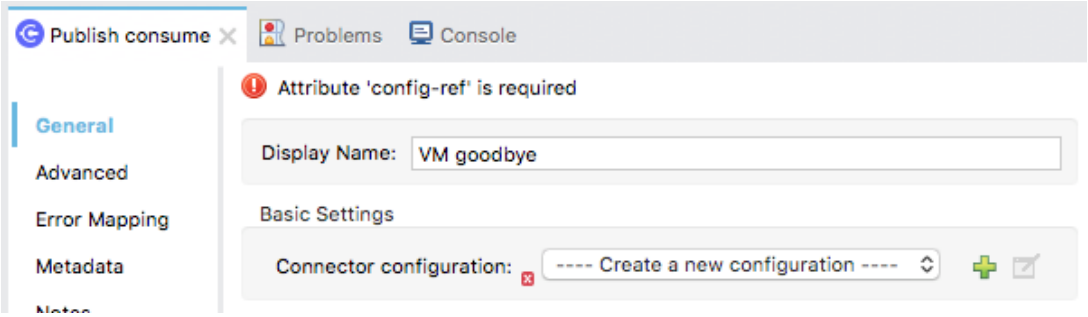


Add a VM Publish Consume operation

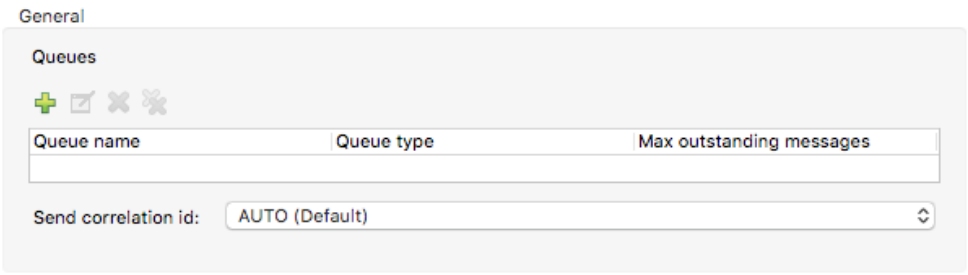
5. In helloFlow, delete the privateFlow Flow Reference.
6. Select VM in the Mule Palette.
7. Select the Publish consume operation and drag and drop it before the Logger in helloFlow.



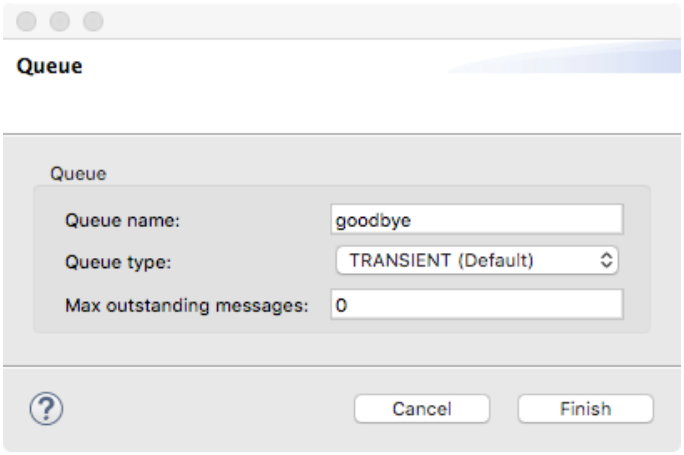
8. In the Publish consume properties view, change the display name to VM goodbye.



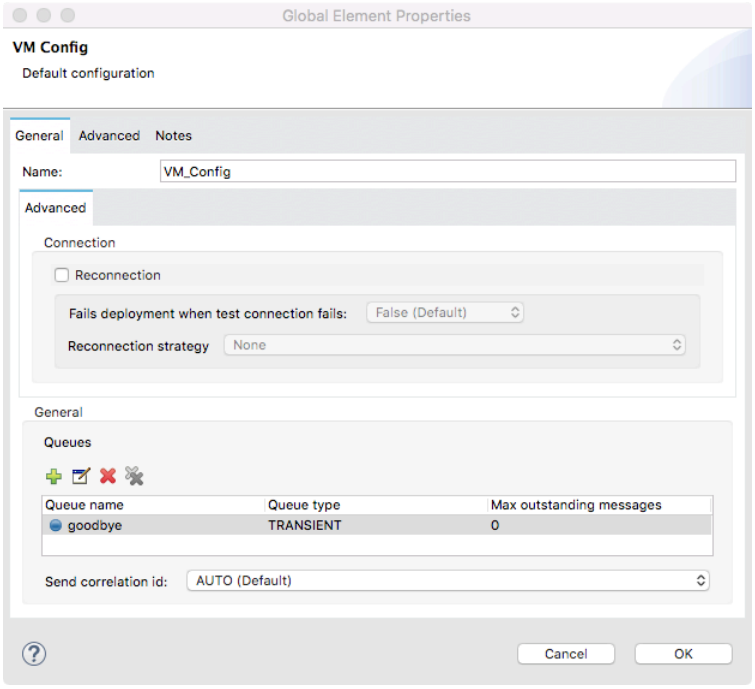
9. Click the Add button next to connector configuration.
10. In the Global Element Properties dialog box, click the Add Queue button.



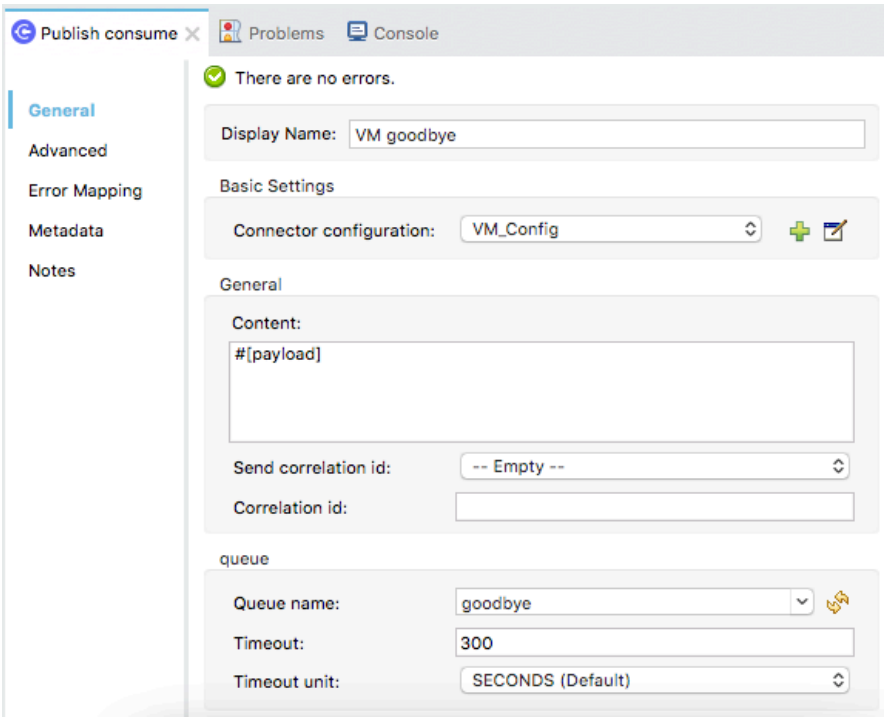
11. In the Queue dialog box, set the queue name to goodbye and click Finish.



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

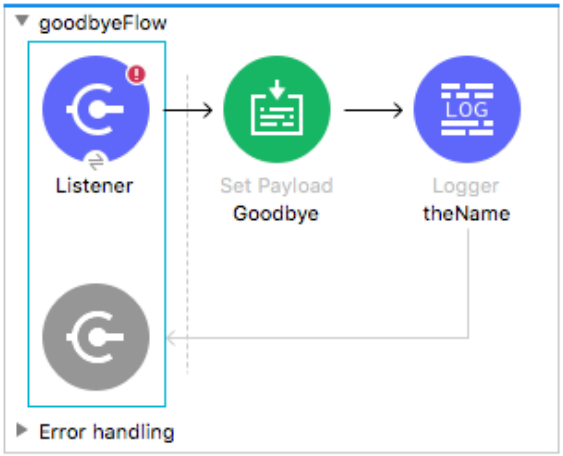


13. In the VM goodbye Publish consume properties view, set the queue name to goodbye.
14. Set the timeout to 300 seconds for debugging purposes.



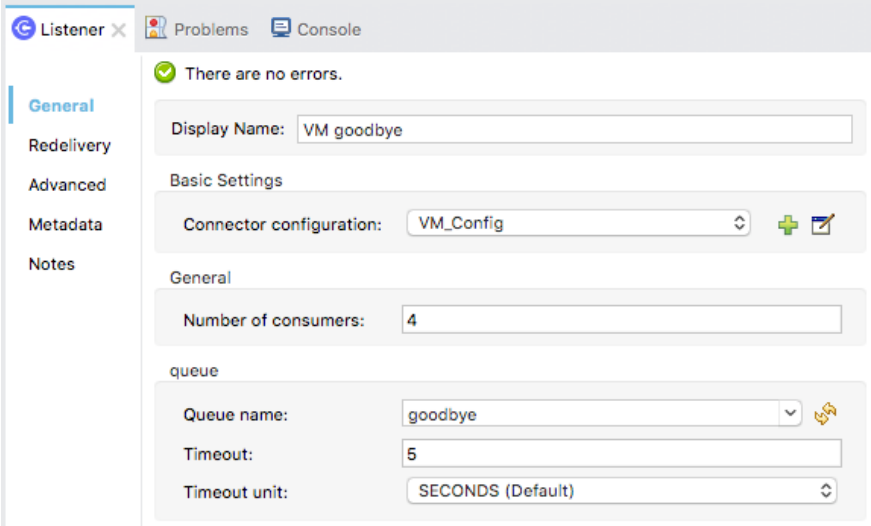
Add a VM Listener

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.



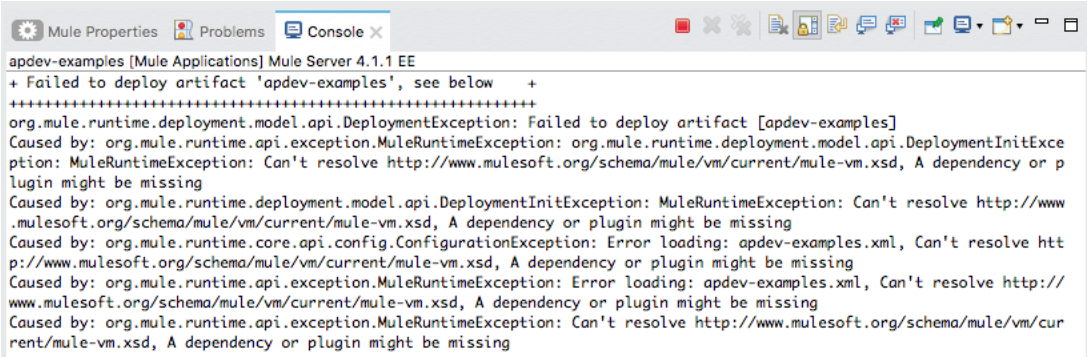
17. In the VM Listener properties view, change the display name to VM goodbye.
18. Set the connector configuration to the existing VM_Config

19. Set the queue name to goodbye.



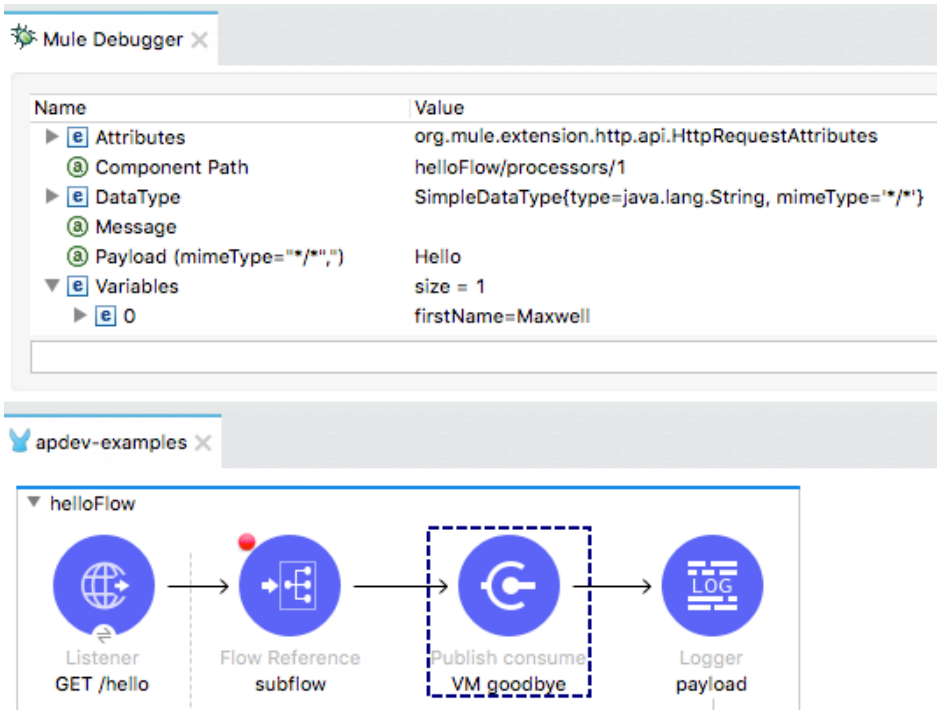
Debug the application

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

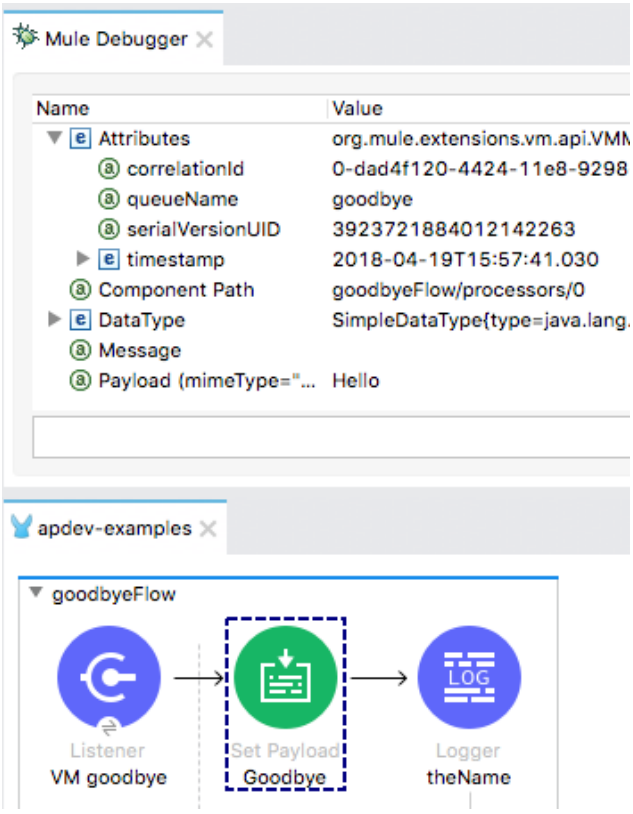


21. Stop the project.
22. Debug the project; the application should successfully deploy.
23. In Advanced REST Client, send the same request.
24. In the Mule Debugger, step through the application to the VM Publish consume.

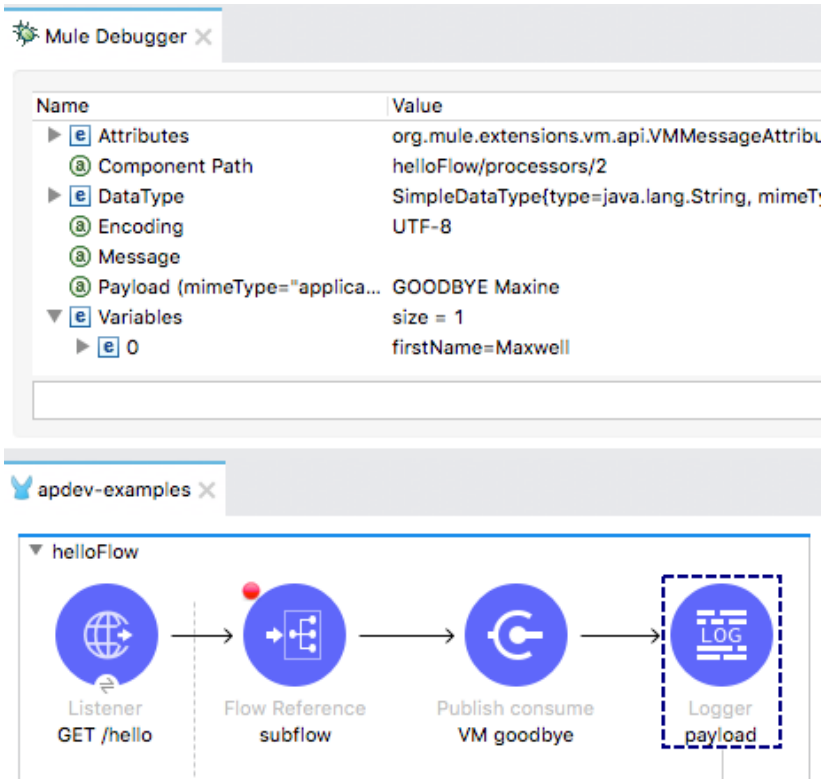
25. Look at the payload, attributes, and variables.



26. Step into goodbyeFlow.
27. Look at the payload, attributes, and variables (or lack thereof).



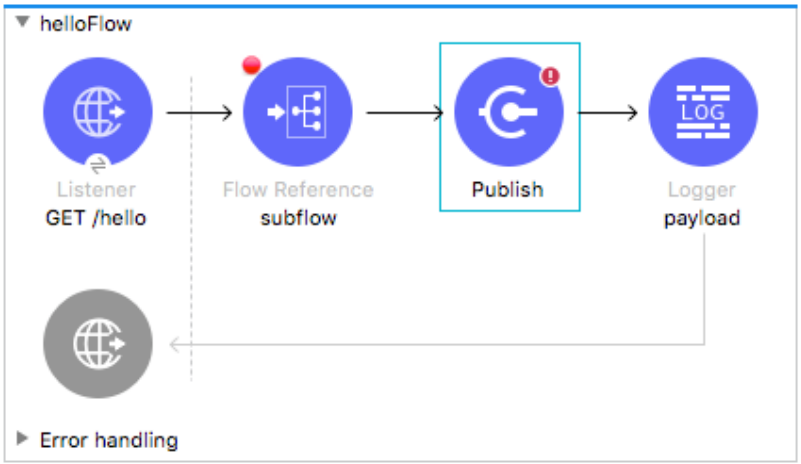
28. Step through the flow until the event returns to helloFlow.
29. Look at the payload, attributes, and variables.



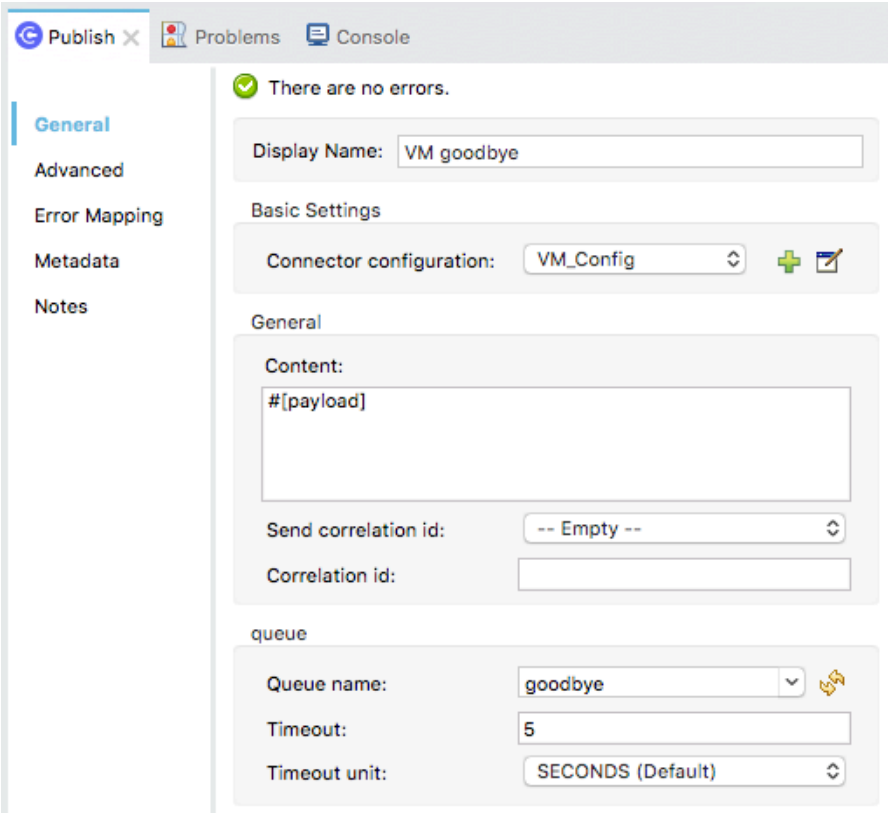
30. Step through the rest of the application.
31. Switch perspectives.

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.



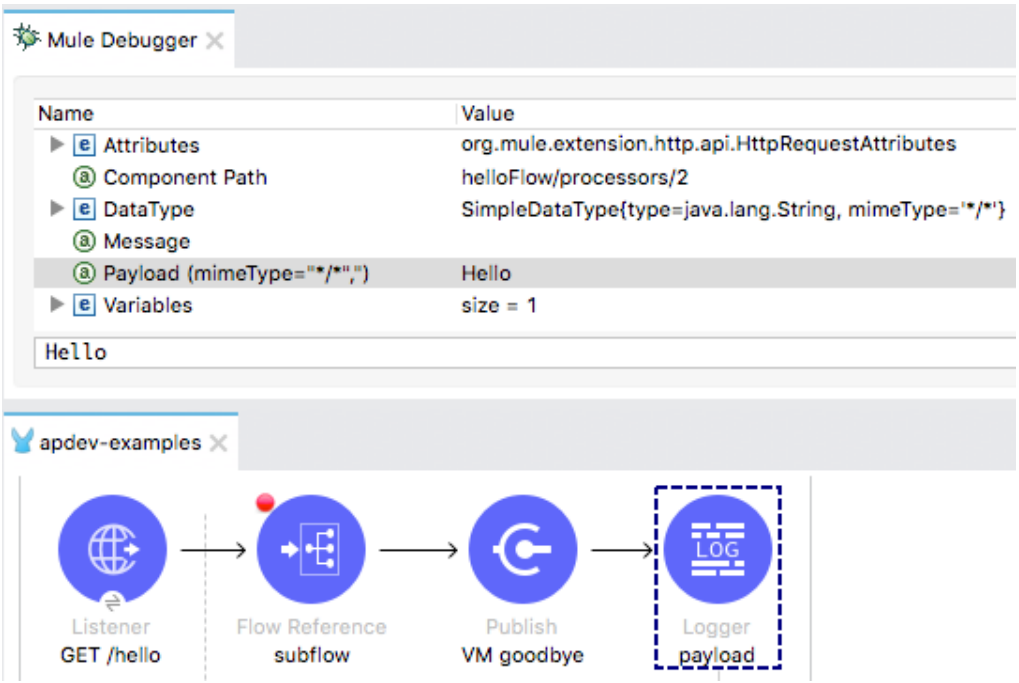
34. In the Publish properties view, set the display name to VM goodbye.
35. Set the connector configuration to the existing VM_Config.
36. Set the queue name to goodbye.



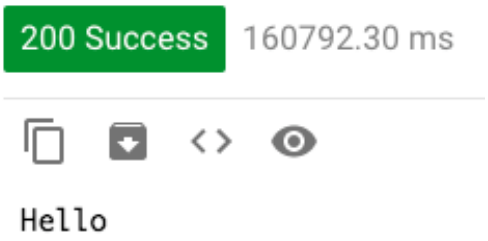
Debug the application

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.



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.

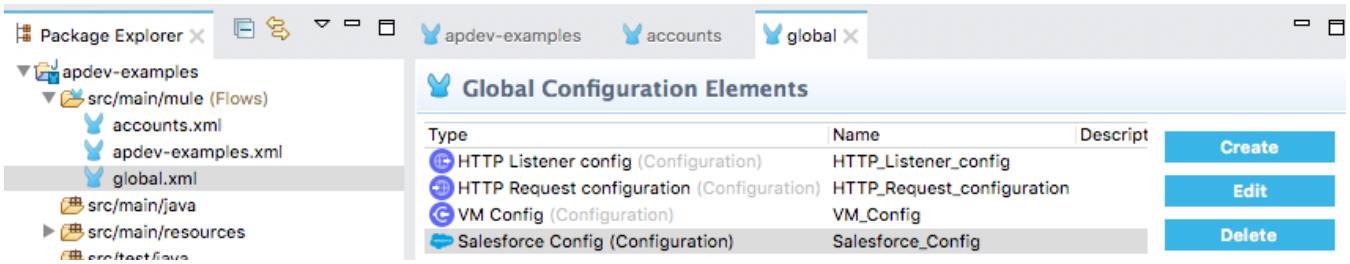


45. Return to Anypoint Studio and switch perspectives.
46. Stop the project.

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

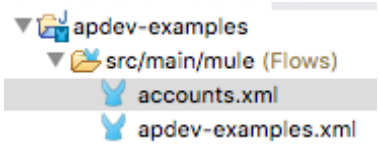
In this walkthrough, you refactor your apdev-examples project. You will:

- 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.

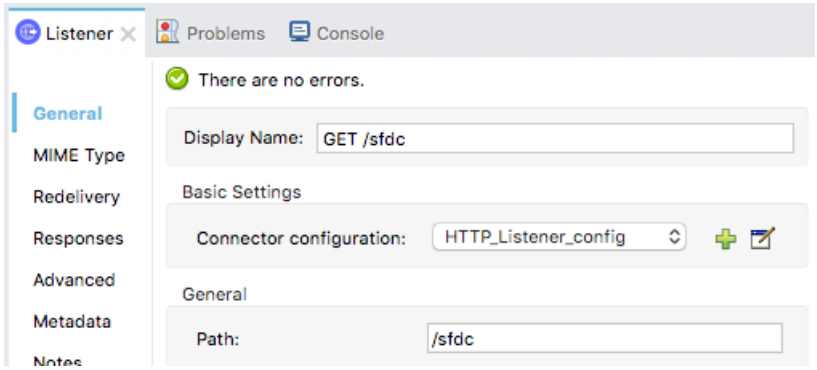


Create a new configuration file

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.



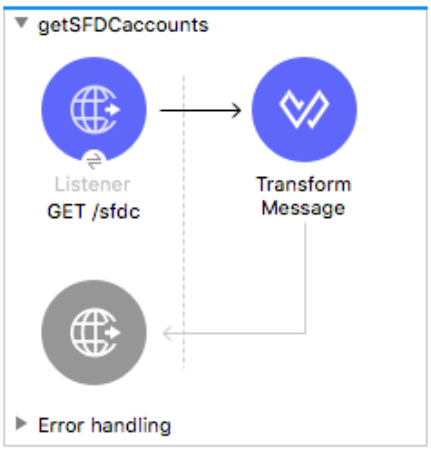
4. Drag an HTTP Listener to the canvas from the Mule Palette.
5. In the Listener properties view, set the display name to GET /sfdc.
6. Set the connector configuration to the existing HTTP_Listener_config.
7. Set the path to /sfdc and the allowed methods to GET.



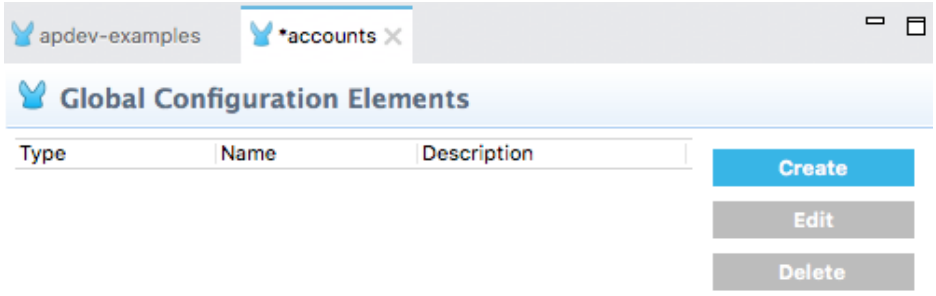
8. Add a Transform Message component to the flow.
9. In the Transform Message properties view, change the output type to application json and set the expression to payload.

```
%dw 2.0
output application/json
---
payload
```

10. Change the name of the flow to getSFDCaccounts.

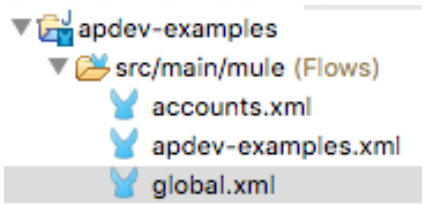


11. Switch to the Global Elements view; you should not see any global elements.



Create a global configuration file

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

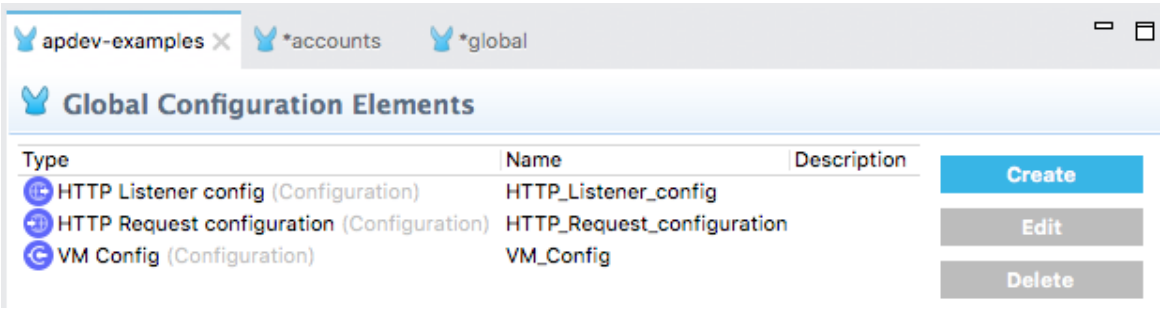


13. In global.xml, switch to the Configuration XML view.

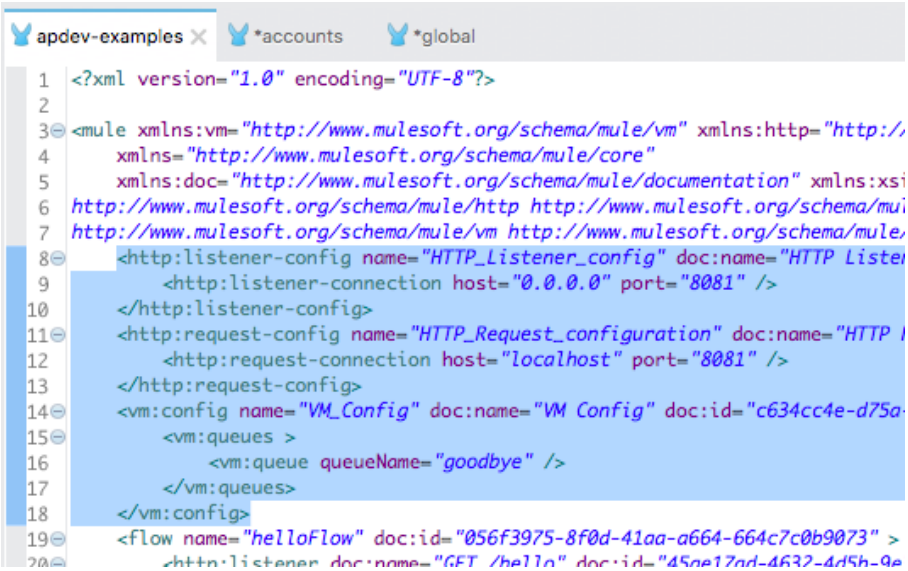
14. Place some empty lines between the start and end mule tags.

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.

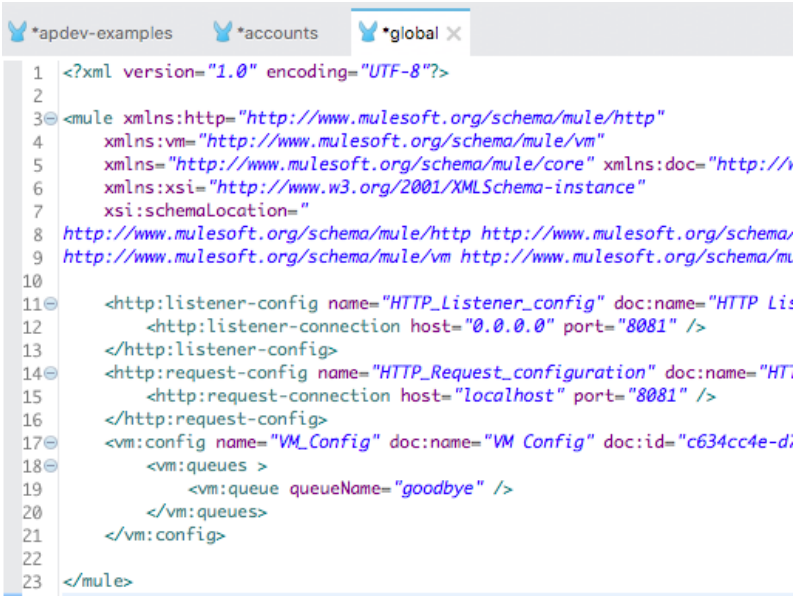


17. Switch to the Configuration XML view.
18. Select and cut the three configuration elements defined before the flows.

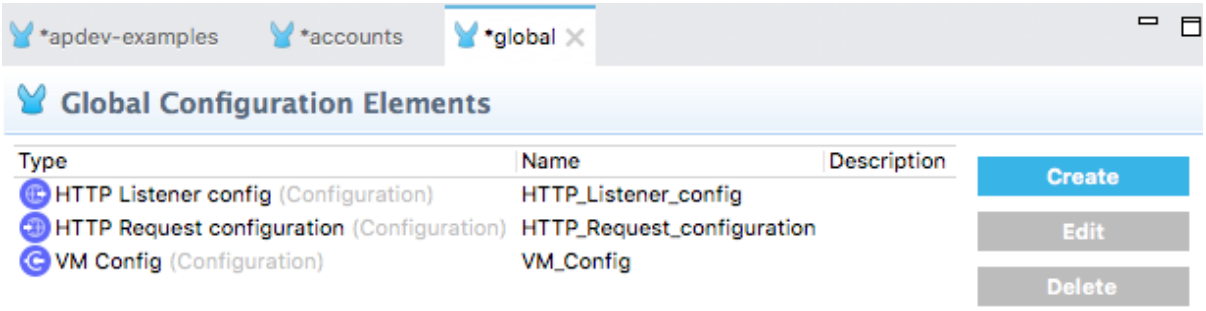


Note: If you delete the global elements from the Global Elements view instead, the config-ref values are also removed from the connector operations and you need to re-add them.

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.

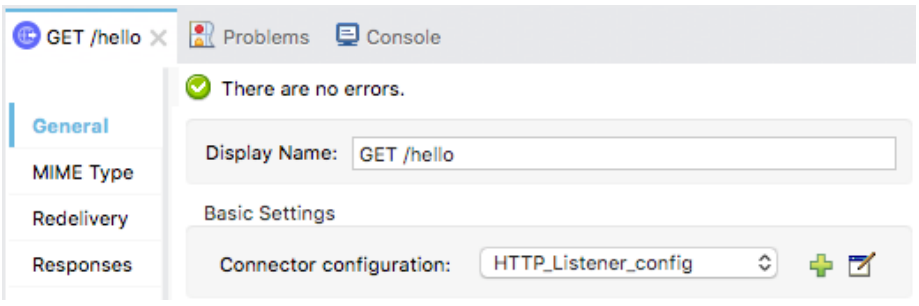


22. Switch to the Global Elements view; you should see the three configurations.

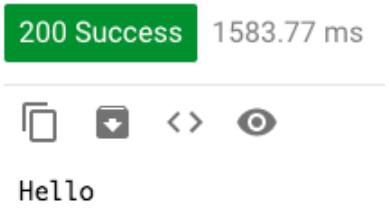


Test the application

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.

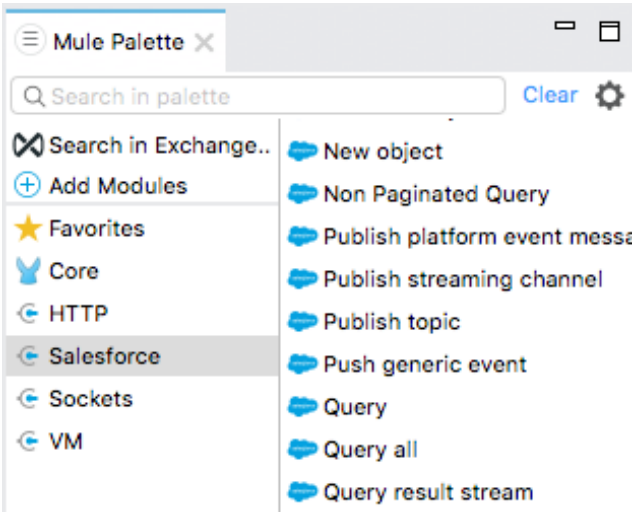


25. Run the project.
26. In Advanced REST Client, send the same request; you should still get a response of Hello.



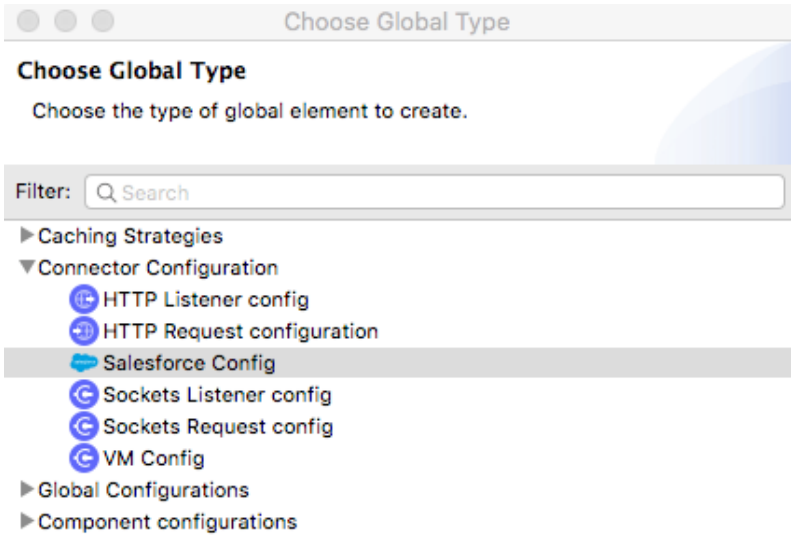
Create a new global element for the Salesforce component in global.xml

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.

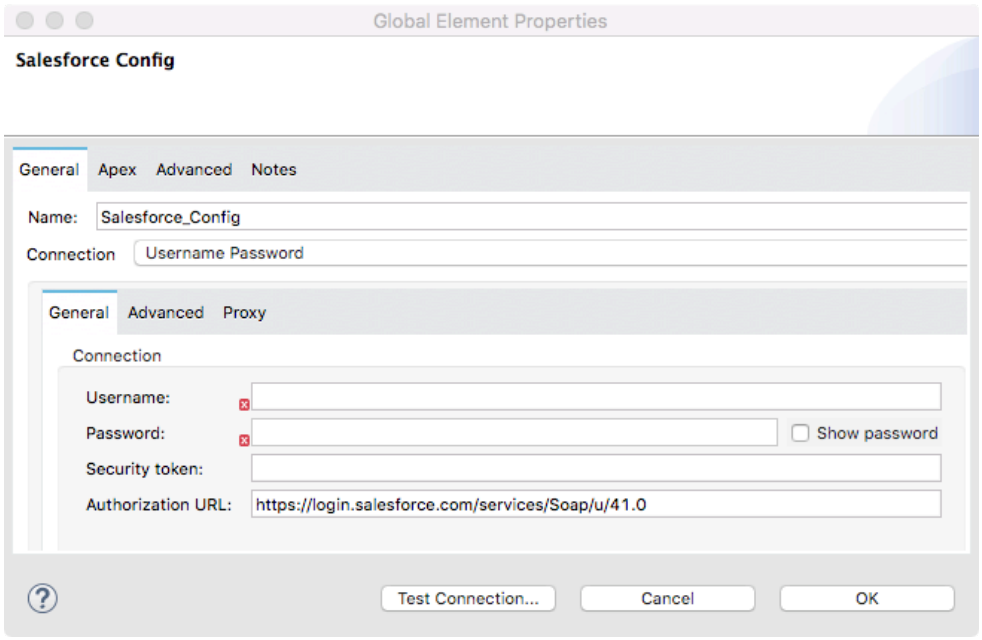


30. Locate the new Salesforce JAR in the project.
31. Return to global.xml.
32. Click Create.

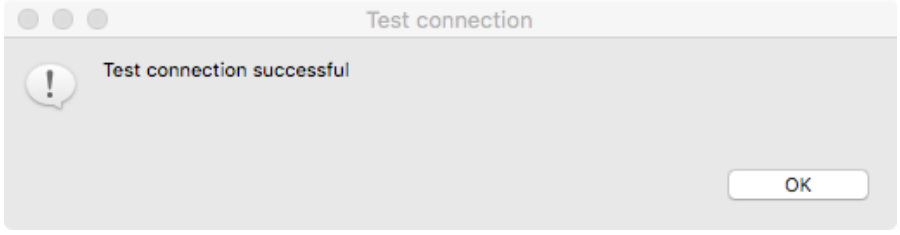
33. In the Choose Global Type dialog box, select Connector Configuration > Salesforce Config and click OK.



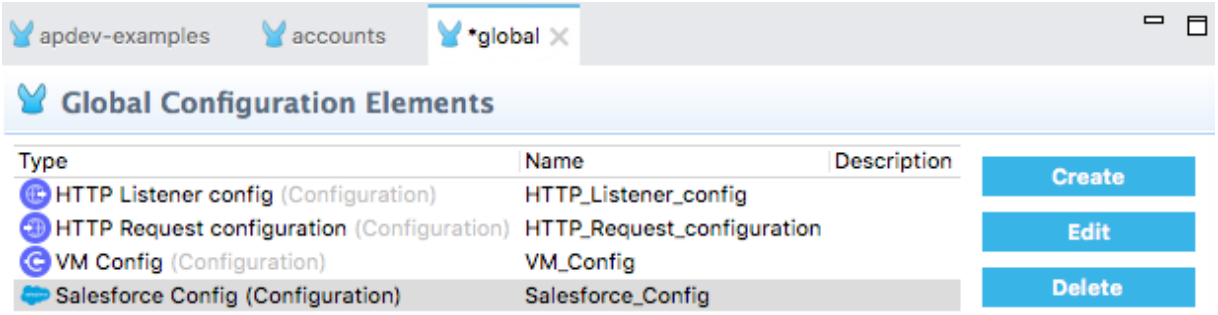
34. In the Global Element Properties dialog box enter your Salesforce username, password, and security token.



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

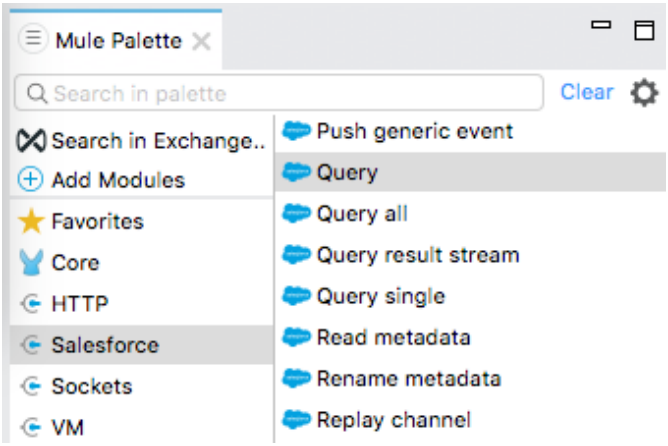


36. In the Test connection dialog box, click OK.
37. In the Global Element Properties dialog box, click OK; you should now see the new configuration in global.

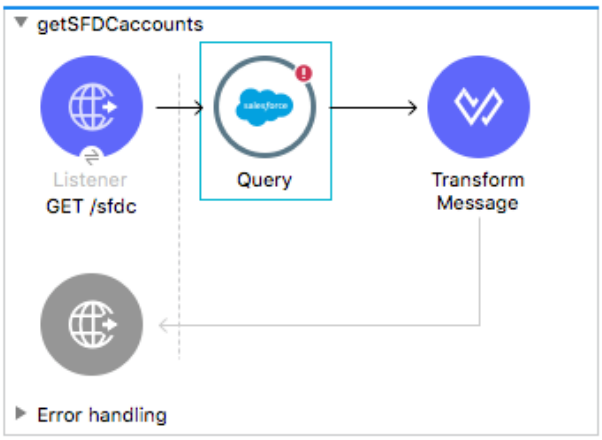


Add a new Salesforce operation that uses the global configuration

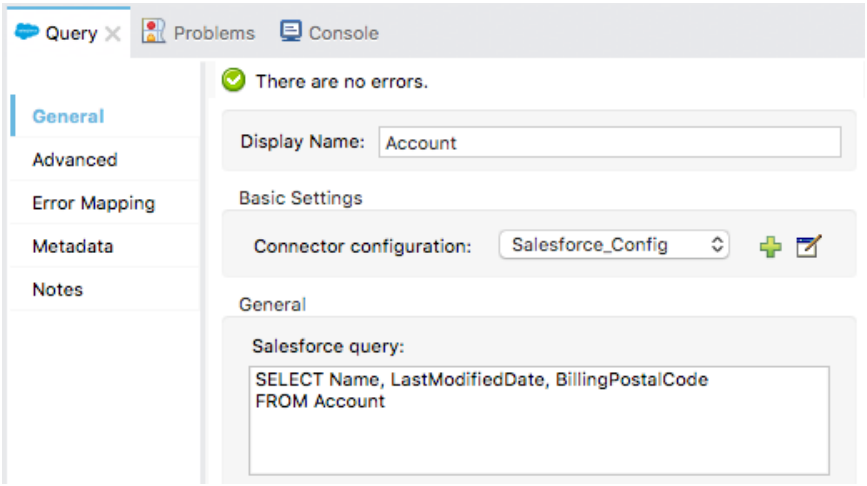
38. Return to the Message Flow view in accounts.xml.
39. In the Mule Palette, select Salesforce.



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

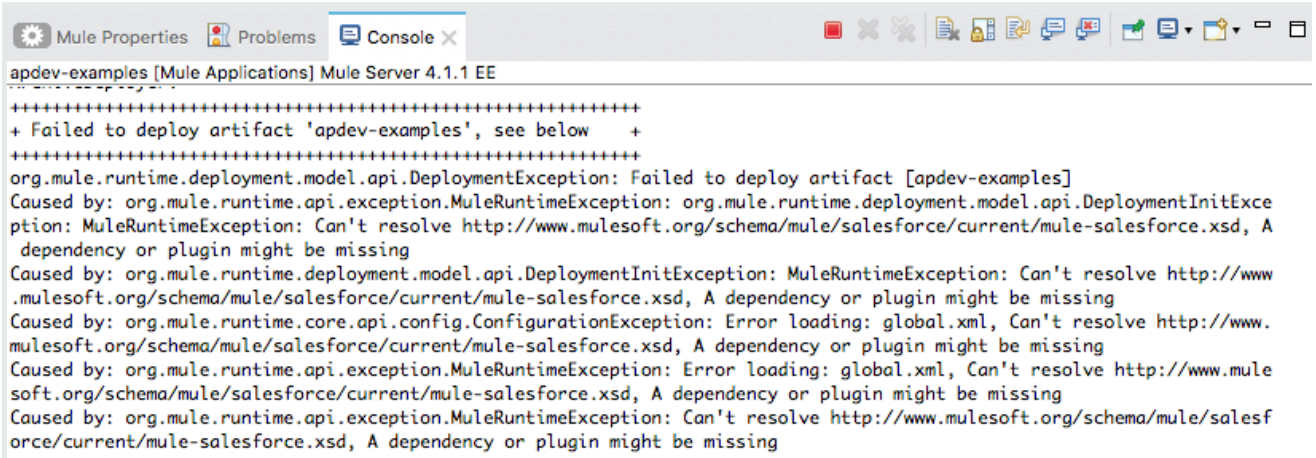


41. In the Query properties view, set the display name to Account.
42. Set the connector configuration to the existing Salesforce_Config.
43. Return to the course snippets.txt file and copy the Salesforce query.
44. Return to Anypoint Studio and paste the query in the Salesforce query section of the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



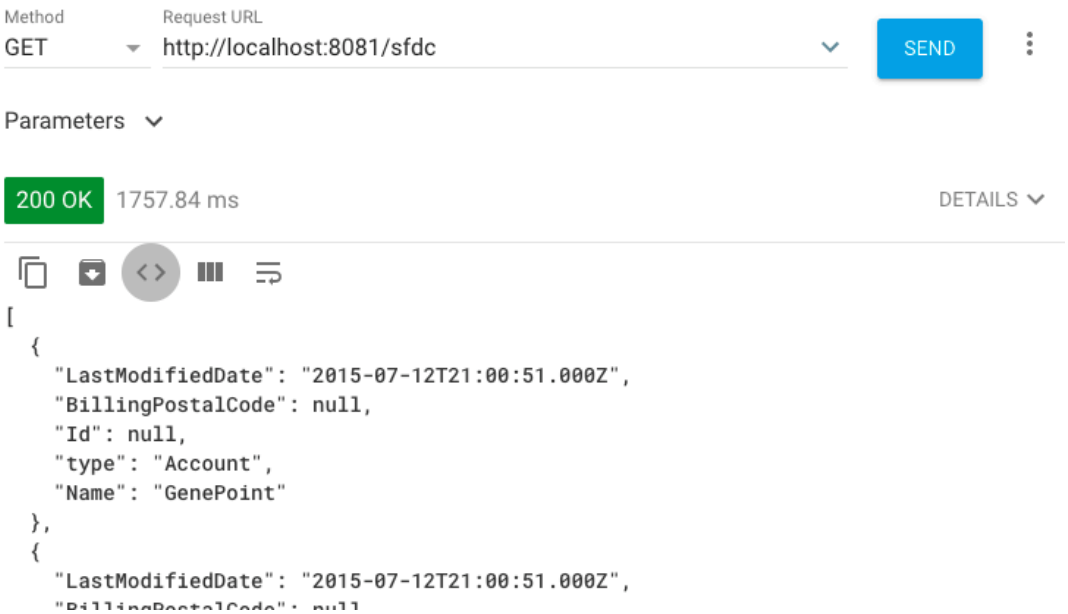
Test the application

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



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

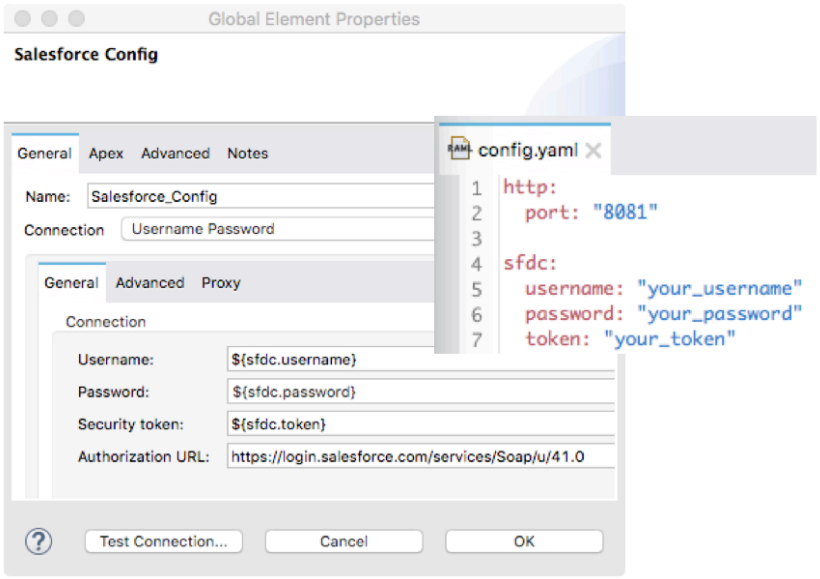
48. In Advanced REST Client, send a request to http://localhost:8081/sfdc; you should get a list of the accounts in your Salesforce account.



Walkthrough 7-4: Use property placeholders in connectors

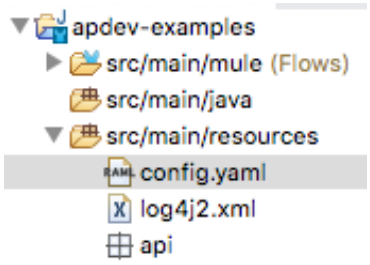
In this walkthrough, you introduce properties into your API implementation. You will:

- 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

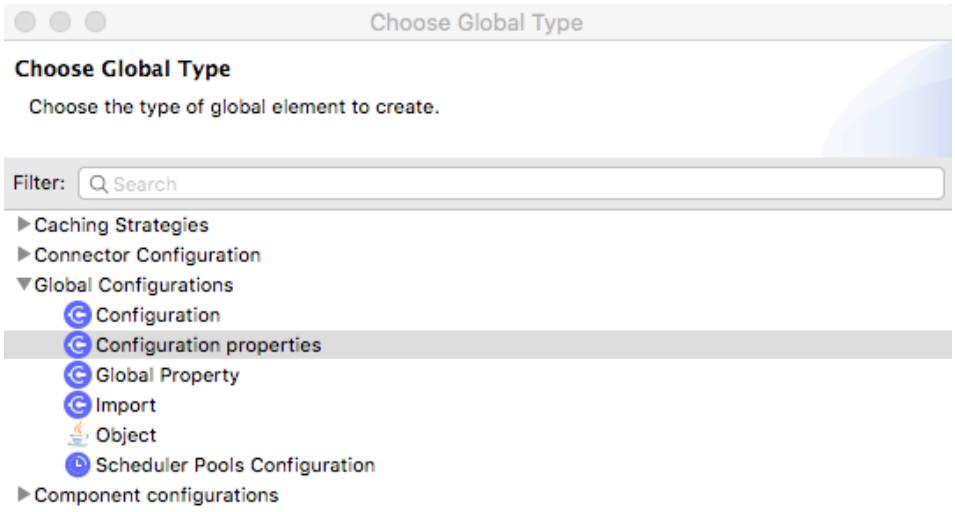
1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > File.
3. Set the file name to config.yaml and click Finish.



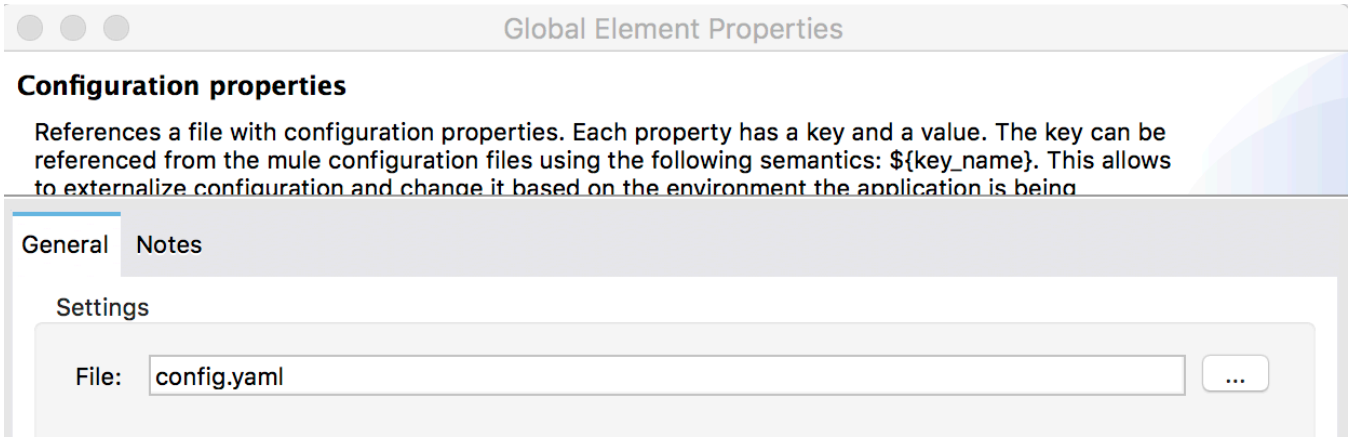
Create a Configuration properties global element

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.

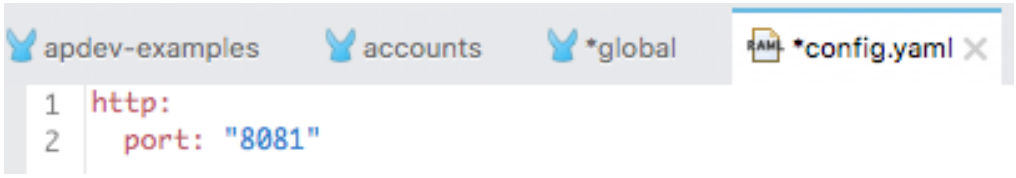


7. In the Global Element Properties dialog box, set the file to config.yaml and click OK.



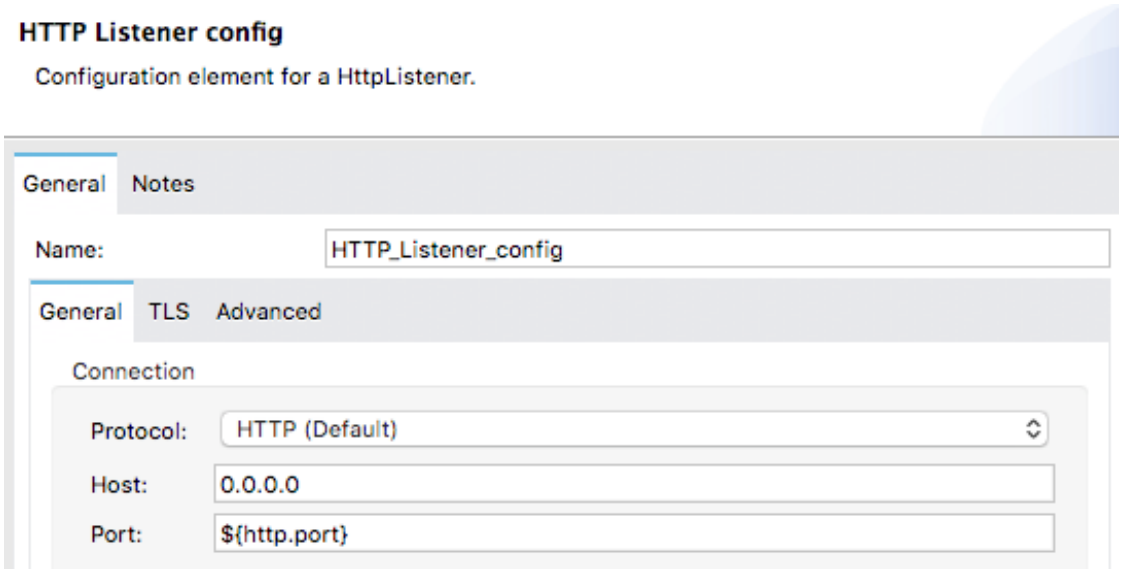
Parameterize the HTTP Listener port

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



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

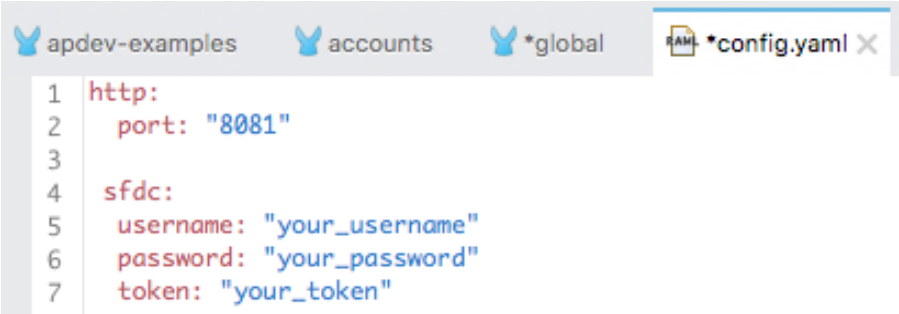
13. Change the port from 8081 to the application property, ${http.port}.



14. Click OK.

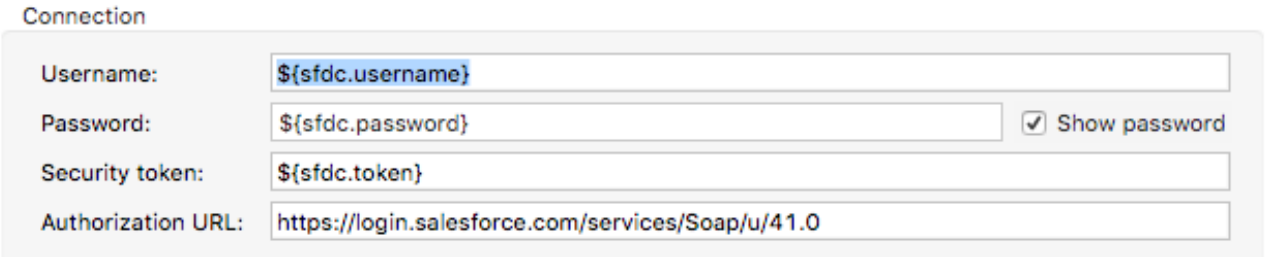
Parameterize the Salesforce credentials

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.

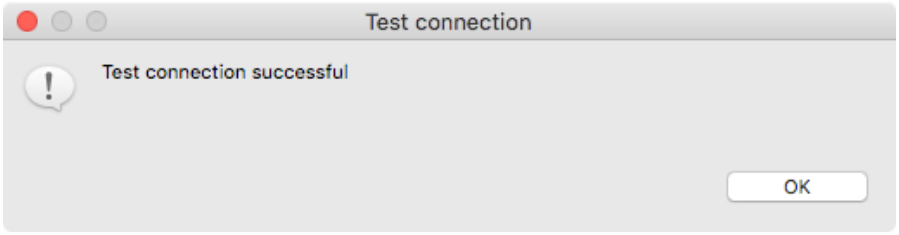


20. Save the file.
21. Return to global.xml.
22. Double-click the Salesforce Config global element.

23. Change the values to use the application properties.



24. Click Test Connection and make sure it succeeds.

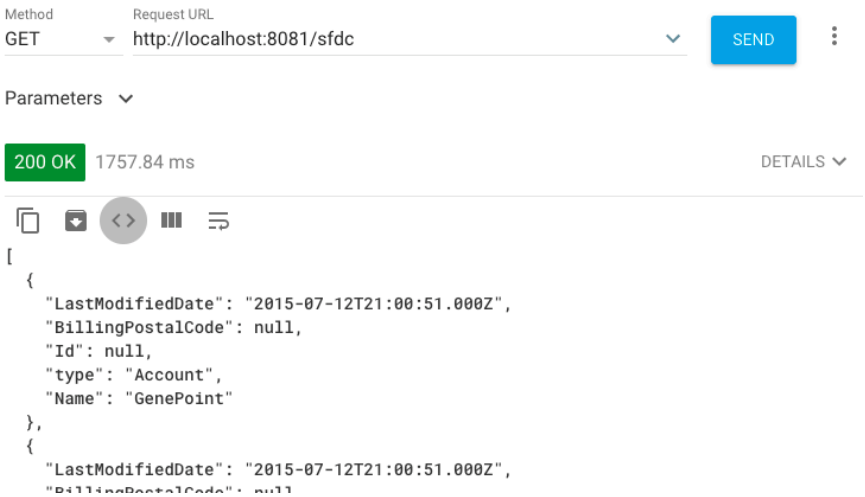


Note: If your connection fails, click OK and then go back and make sure you do not have any syntax errors in config.yaml and that you saved the file.

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.

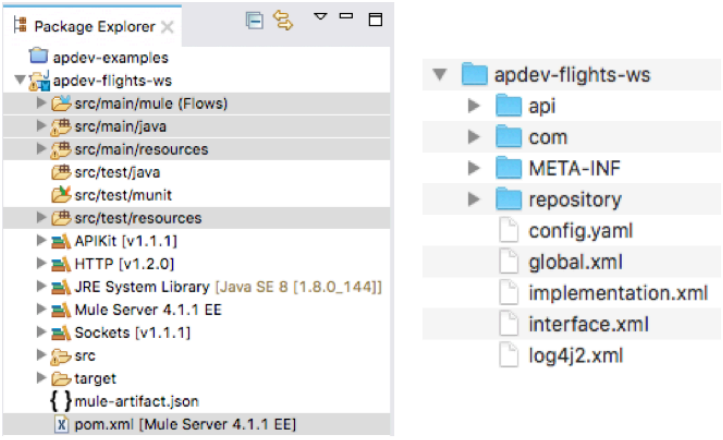


29. Return to Anypoint Studio and stop the project.

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

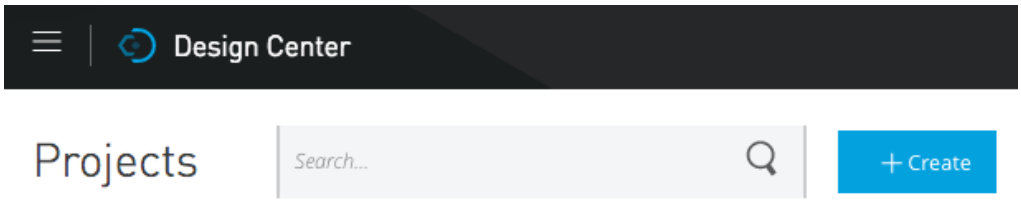
In this walkthrough, you create a new project for the Mule United Airlines (MUA) flights application that you will build during the course and then review and organize its files and folders. You will:

- 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.



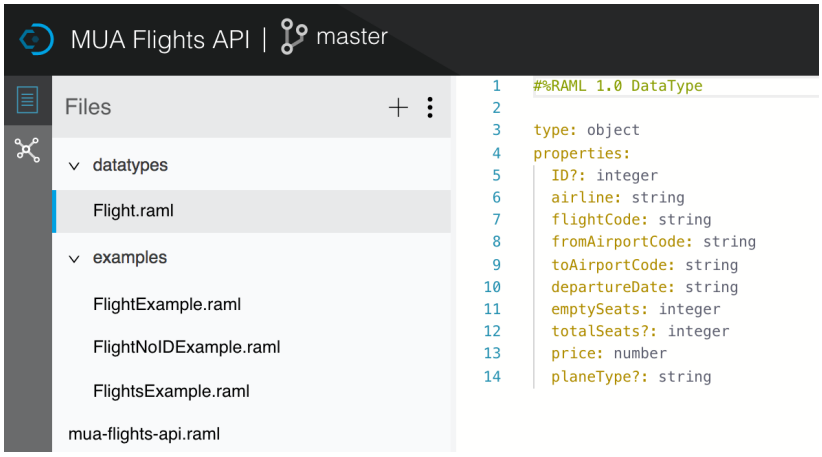
Create a new API in Design Center

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



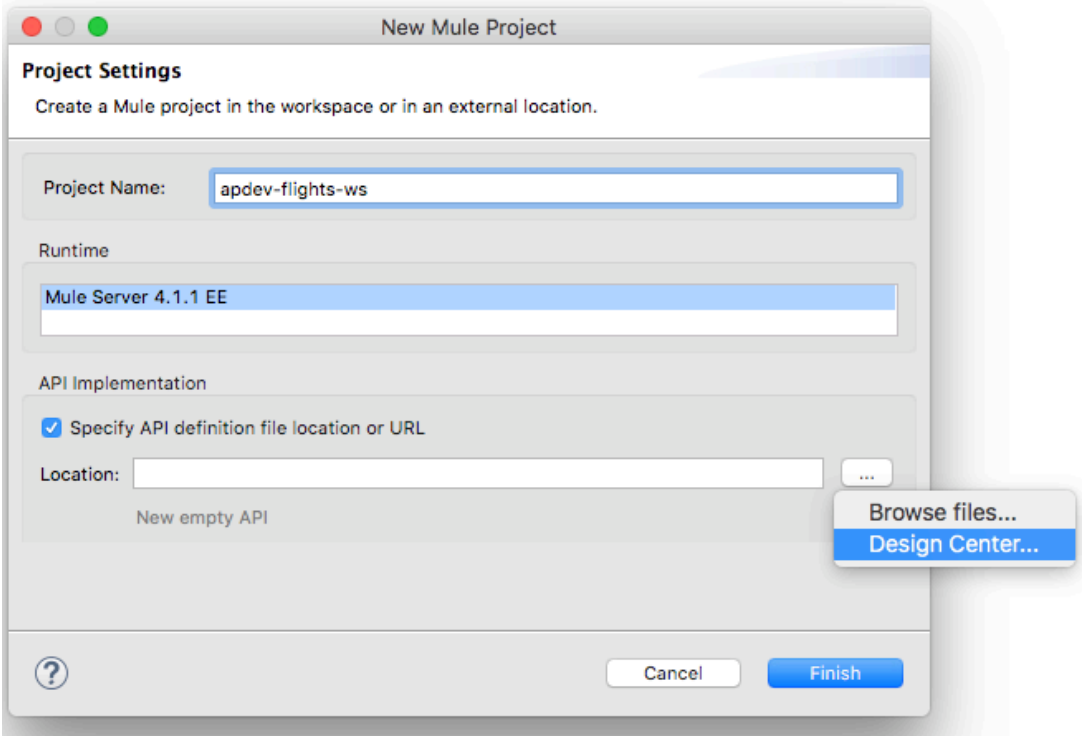
3. In API designer, click the options menu in the file browser and select Import.
4. In the Import dialog box, browse to your student files and select the MUA Flights API.zip located in the resources folder.
5. Click Import.
6. In the Replace dialog box, click Replace file.

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.

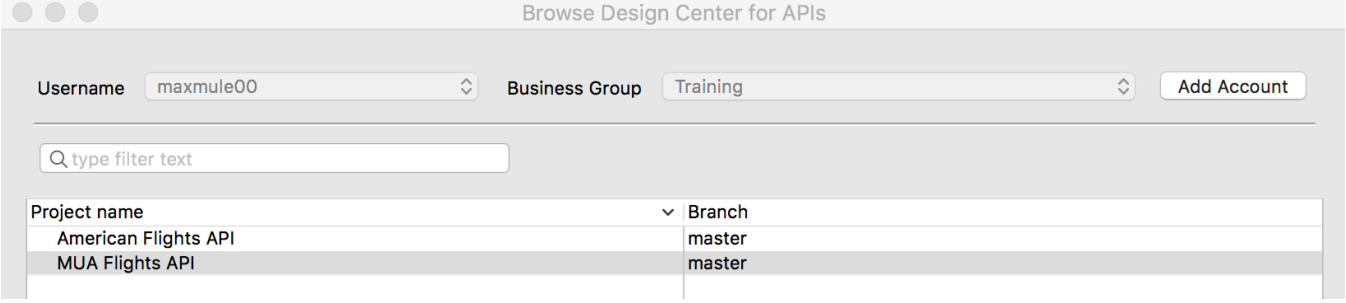


Create a new project to implement this API in Anypoint Studio

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.



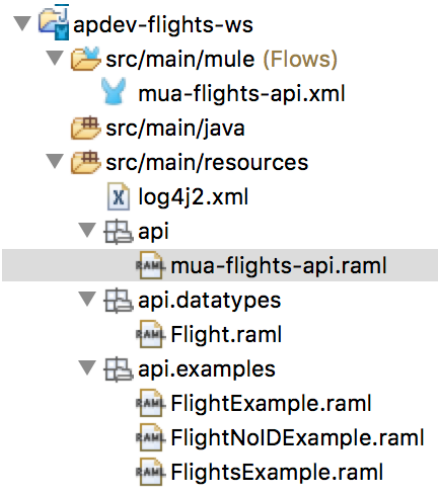
13. In the Browse API Design Center for APIs dialog box, select MUA Flights API and click OK.



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.

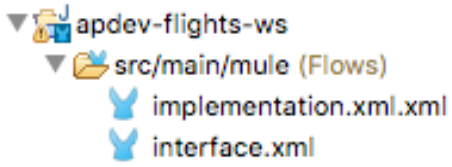


17. Open mua-flights-api.raml and review the file.
18. Leave the file open.

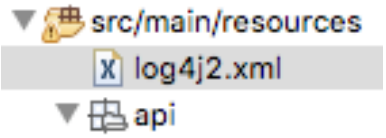
Review project configuration files

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.

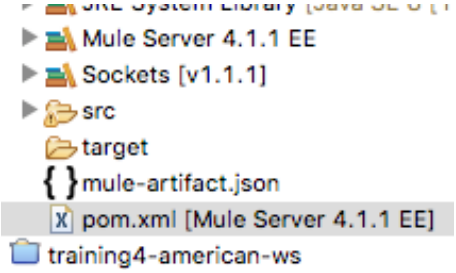
21. Create a new Mule configuration file called implementation.xml.



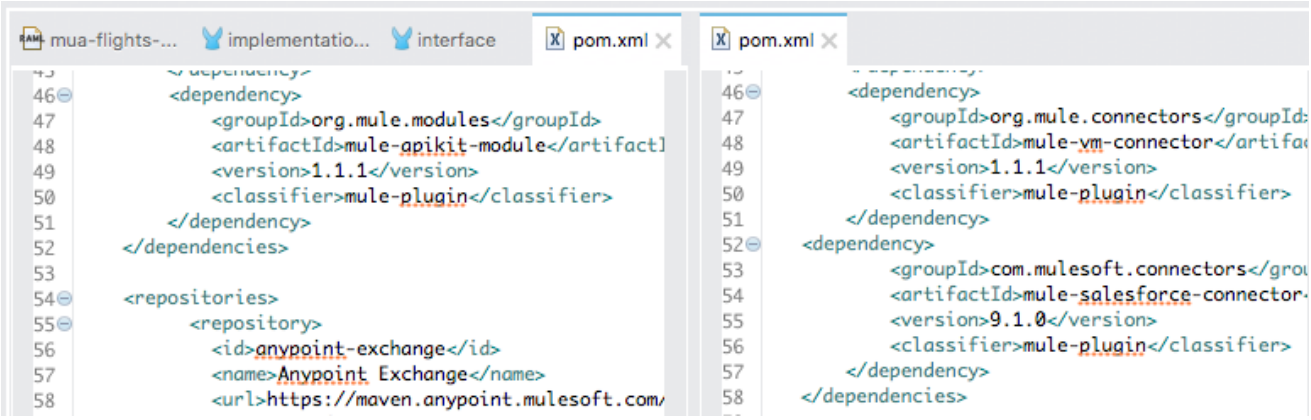
22. Locate log4j2.xml in src/main/resources and open it.



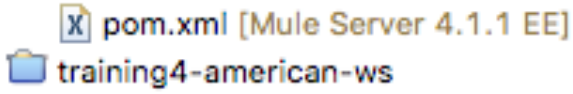
23. Review its contents and then close the file.
24. Locate pom.xml in the project and open it.



25. Review its contents.
26. Return to the apdev-examples project and open its pom.xml file.
27. Compare the two pom.xml files.

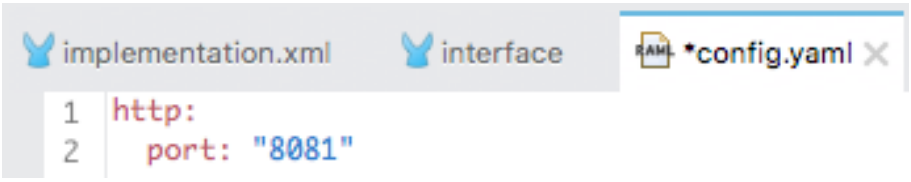


28. Close both pom.xml files.
29. In the Package Explorer, right-click apdev-examples and select Close Project.



Create a properties file for application properties

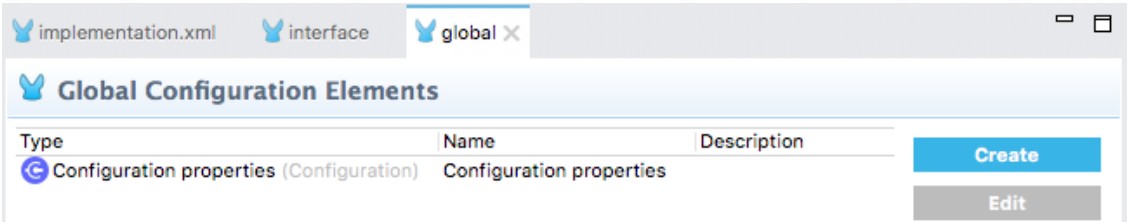
30. In the apdev-flights-ws project, right-click src/main/resources and select New > File.
31. In the New File dialog box, set the file name to config.yaml and click Finish.
32. In config.yaml, define a property called http.port equal to “8081”.



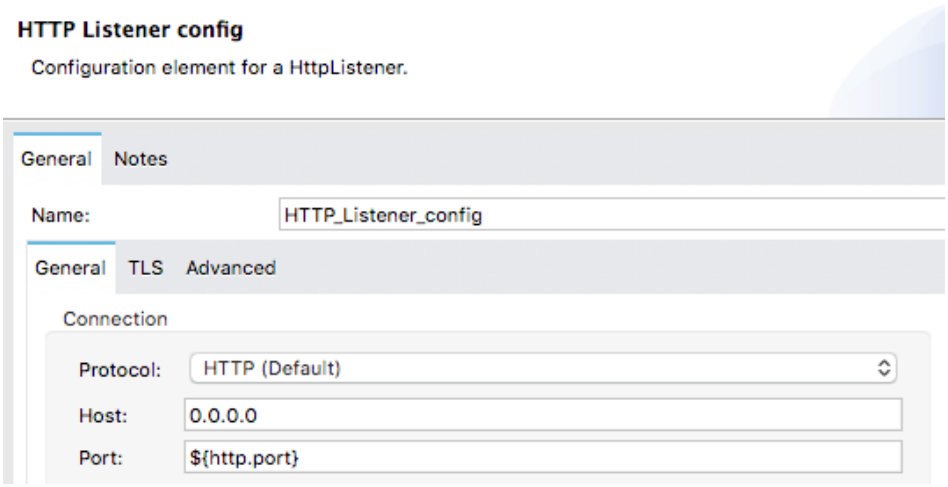
33. Save and close the file.

Create a global configuration file

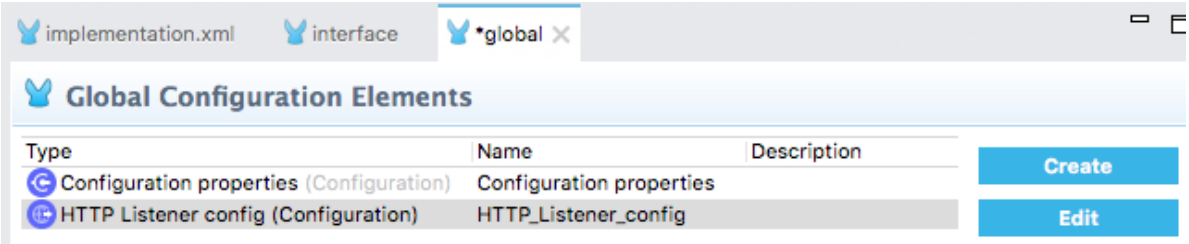
34. In src/main/mule, create a new Mule configuration file called global.
35. In global.xml, switch to the Global Elements view.
36. Create a new Configuration properties element with the file set to config.yaml.



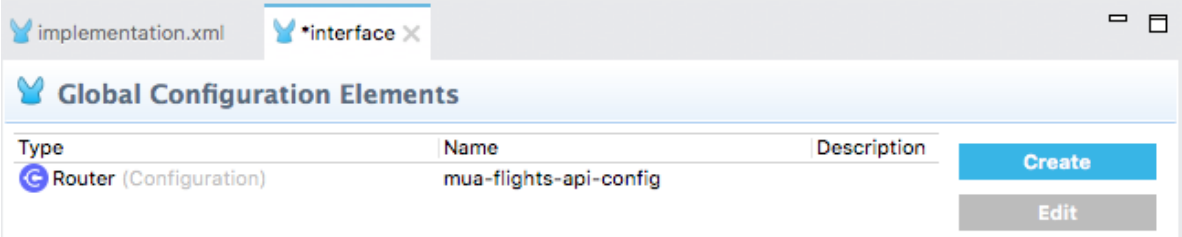
37. Create a new HTTP Listener config element with the host set to 0.0.0.0 and the port set to the http.port property.



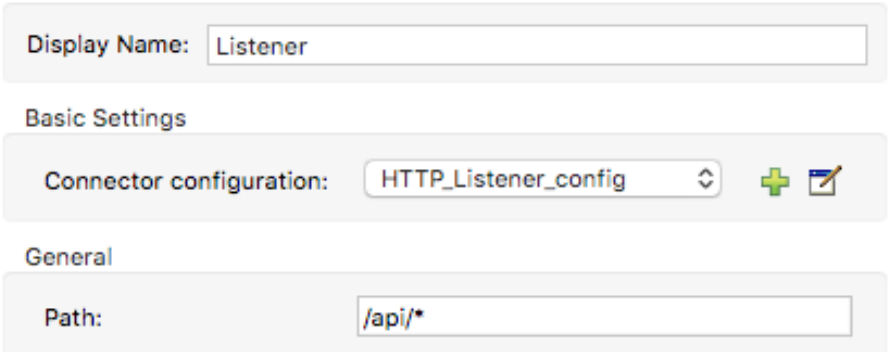
38. Confirm you now have two global elements defined in global.xml.



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.



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.

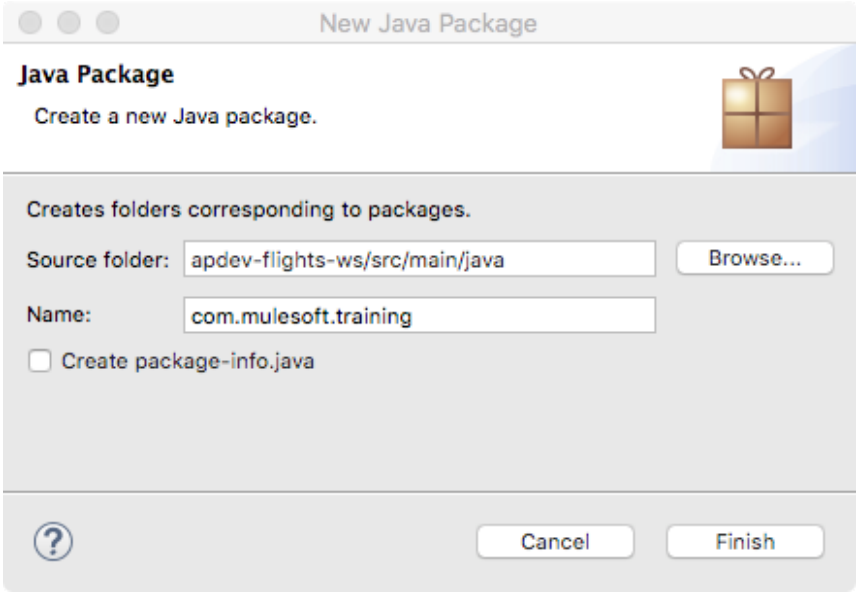


44. Select the HTTP Listener in mua-flights-api-console, and in the Listener properties view set the connector configuration to HTTP_Listener_config.

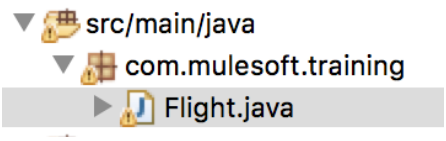
Add Java files to src/main/java

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.

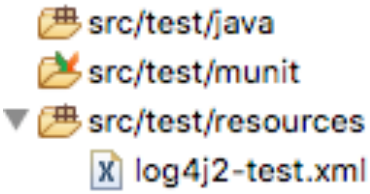


47. In your computer's file browser, locate the Flight.java file in the resources folder.
48. Drag the Flight.java file into the new com.mulesoft.training package in the src/main/java folder in the Anypoint Studio apdev-flights-ws project.
49. In the File Operation dialog box, select Copy files and click OK.
50. Open the Flight.java file.
51. Review the code.



Review src/test folders

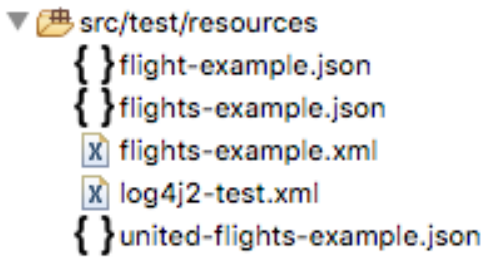
52. In the project, locate the three src/test folders.
53. Expand the src/test/resources folder.



Add test resources

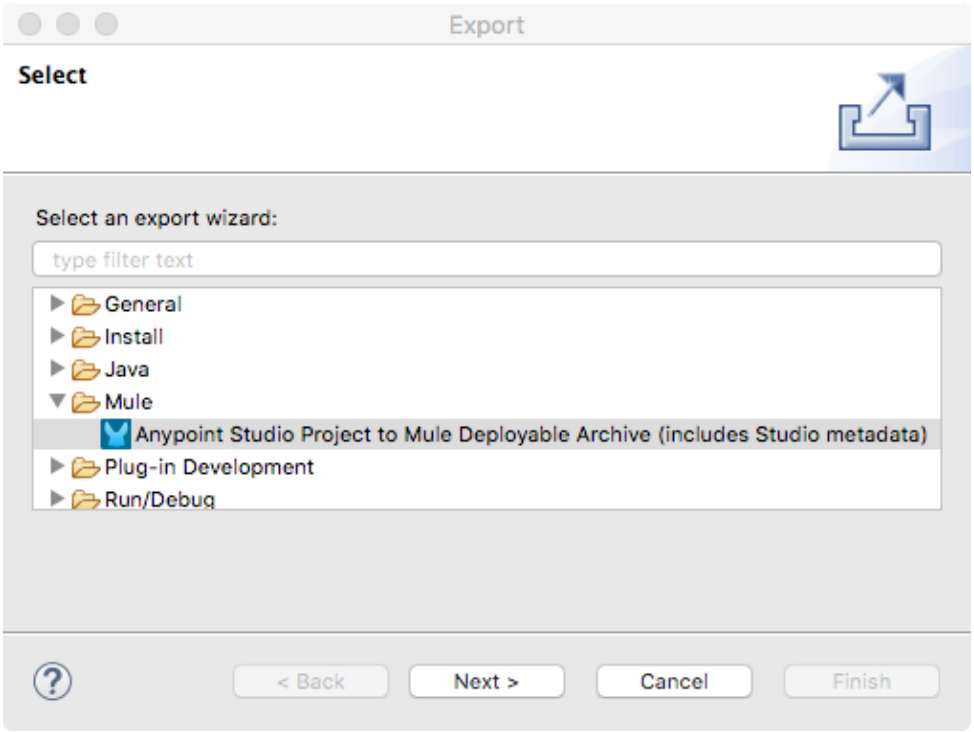
54. In your computer's file browser, expand the examples folder in the student files.

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.

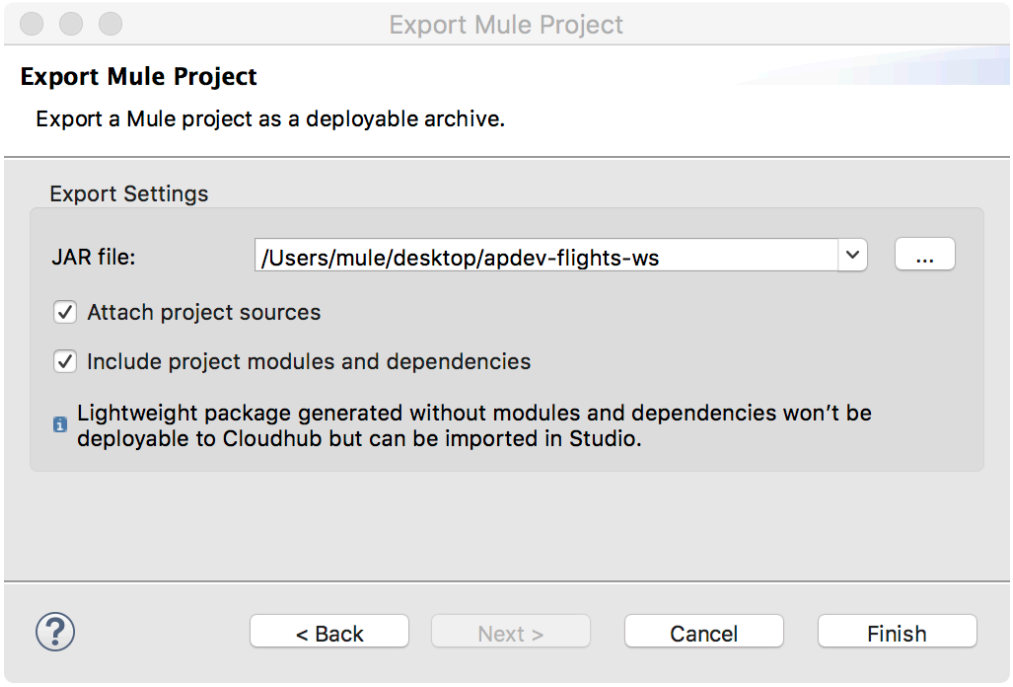


Examine the contents of a deployable archive

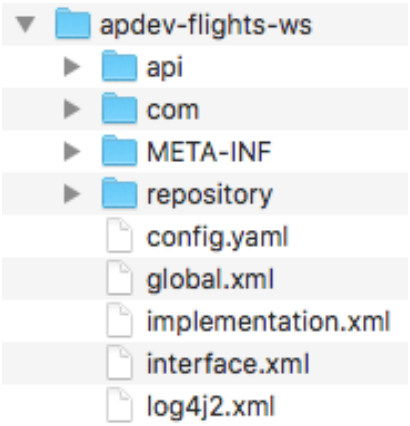
56. In Anypoint Studio, right-click the project and select Export.
57. In the Export dialog box, select Mule > Anypoint Studio Project to Mule Deployable Archive and click Next.



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



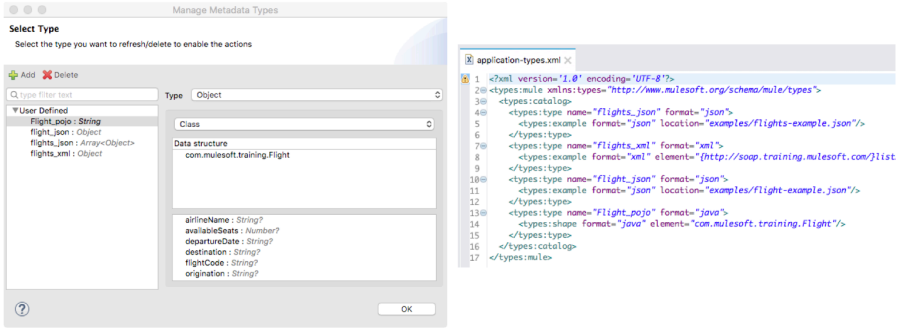
59. In your computer's file browser, locate the JAR file and unzip it.
60. Open the resulting apdev-flights-ws folder and examine the contents.



Walkthrough 7-6: Manage metadata for a project

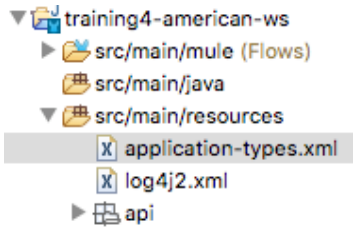
In this walkthrough, you define metadata for the new apdev-flights-ws project. You will:

- 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.

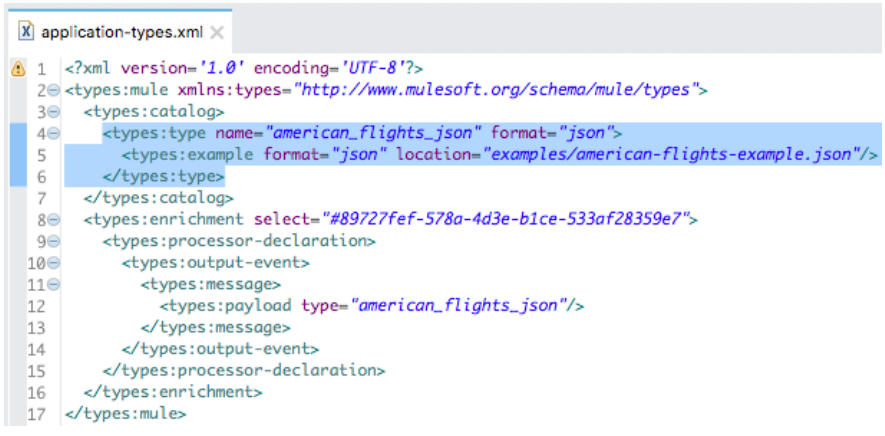


Locate existing metadata in the training-american-ws project

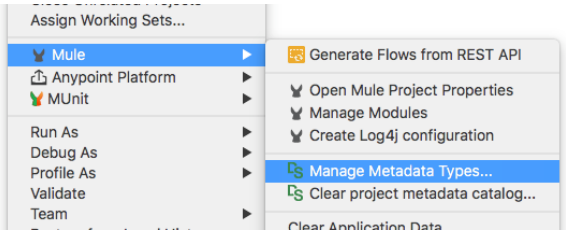
1. Open the training-american-ws project.
2. Locate application-types.xml in src/main/resources.



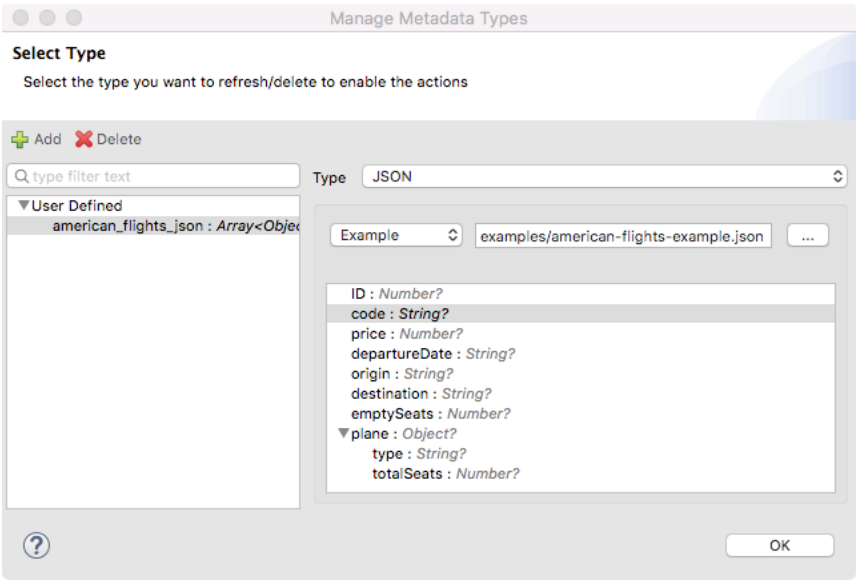
3. Open the file and review the code.



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



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



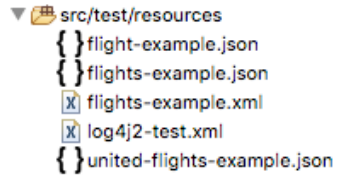
7. Click OK.
8. Close the project.

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

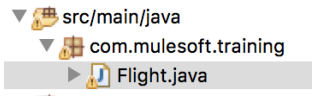
9. Return to the apdev-flights-ws project.
10. Return to mua-flights-api.raml and review the specified response and request types.

```
  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.



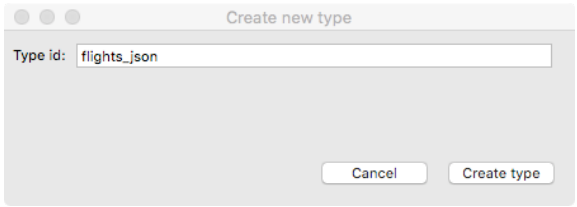
12. Locate the files you added to the src/main/java folder.



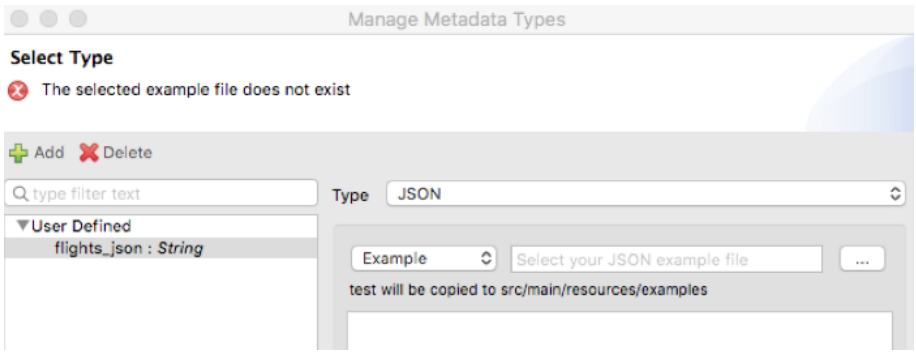
Create flights_json metadata type

13. Right-click the project and select Mule > Manage Metadata Types.
14. In the Select metadata type dialog box, click the Add button.

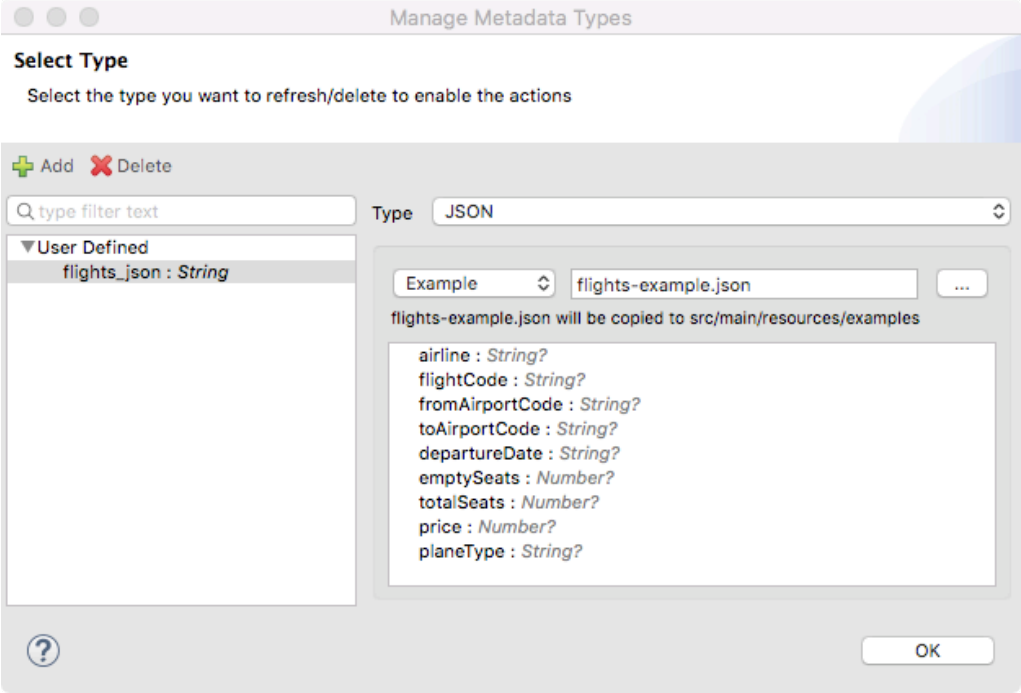
15. In the Create new type dialog box, set the type id to flights_json.



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.



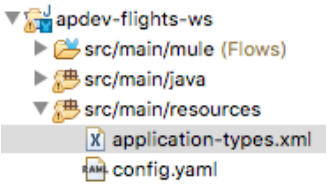
19. Click the browse button and select the flights-example.json file in src/test/resources.



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

Locate the new metadata definition file

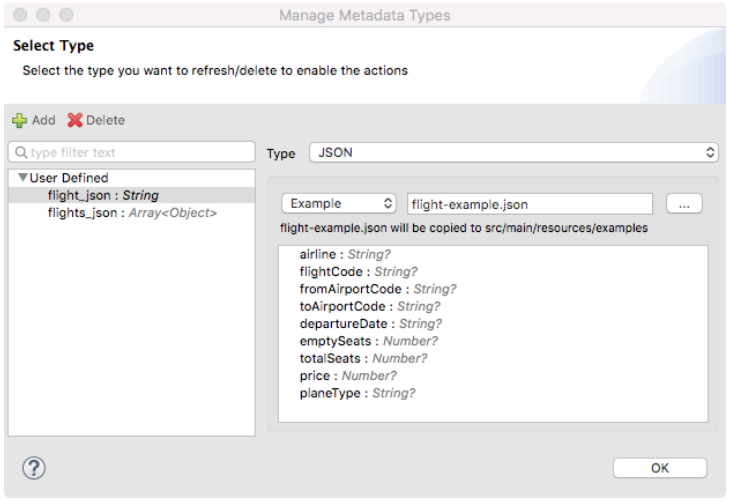
21. Locate application-types.xml in src/main/resources.



22. Open the file and review the code.

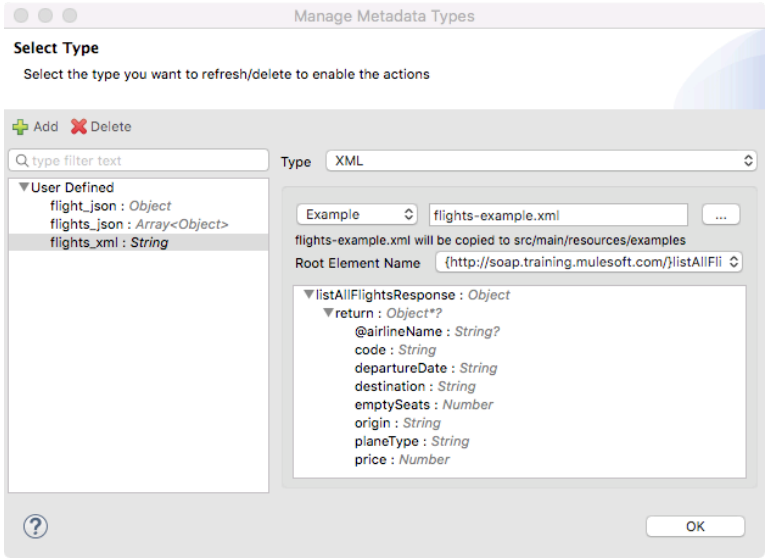
Create flight_json metadata type

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.



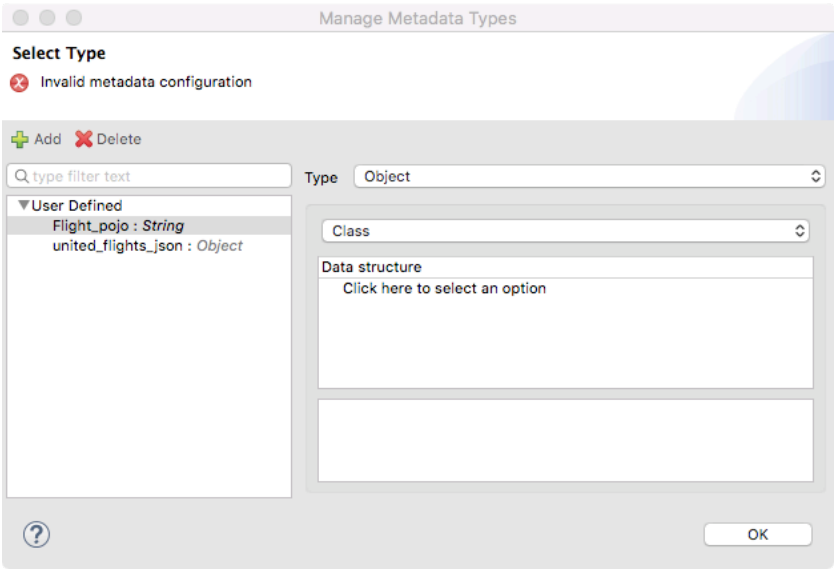
Create flights_xml metadata type

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.

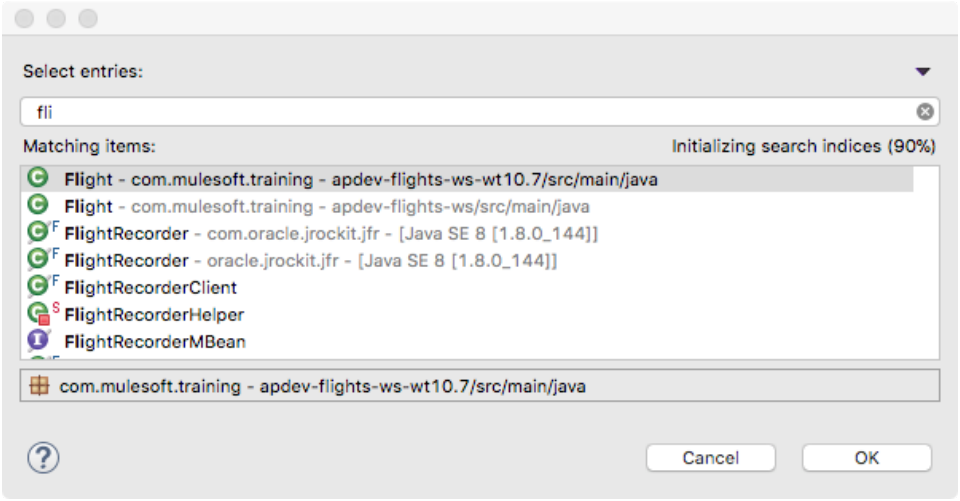


Create Flight metadata type

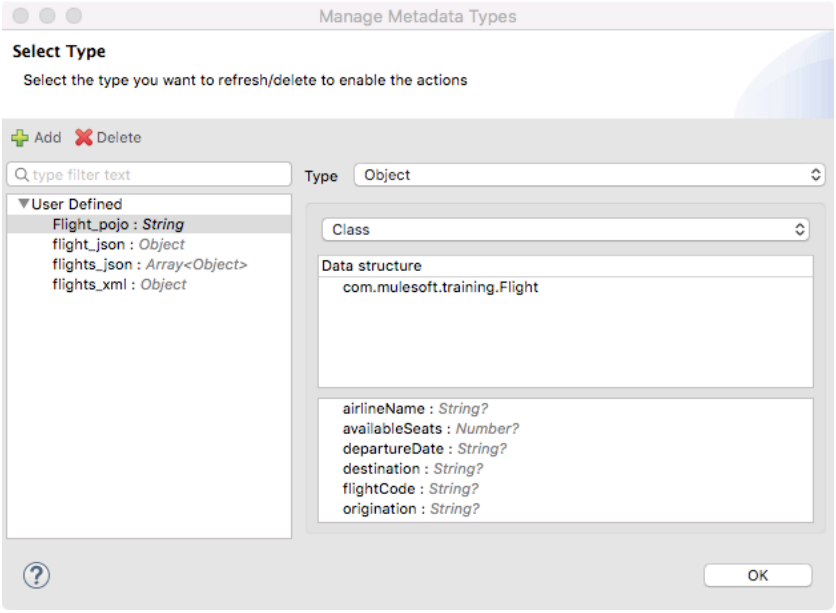
32. In the Select metadata type dialog box, click the Add button.
33. In the Create new type dialog box, set the type id to Flight_pojo and click Create type
34. In the Select metadata type dialog box, set the type to Object.



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.



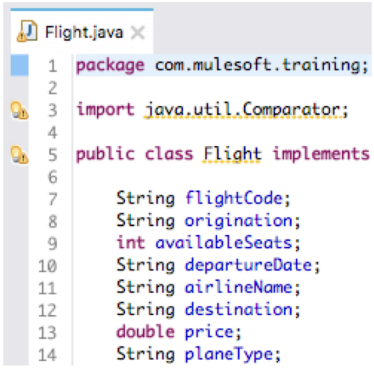
39. In the Manage Metadata Types dialog box, click OK.

Review the metadata definition file

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

41. Close the file.

Module 8: Consuming Web Services



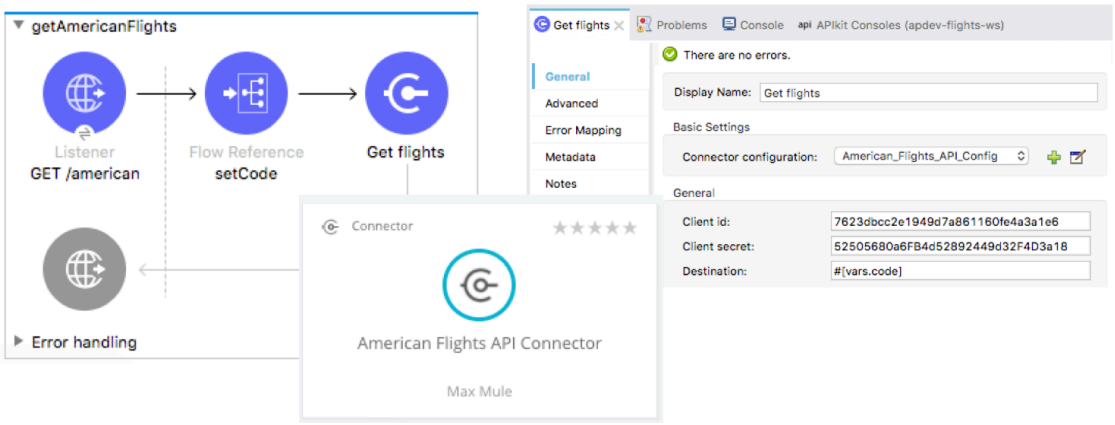
At the end of this module, you should be able to:

- 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

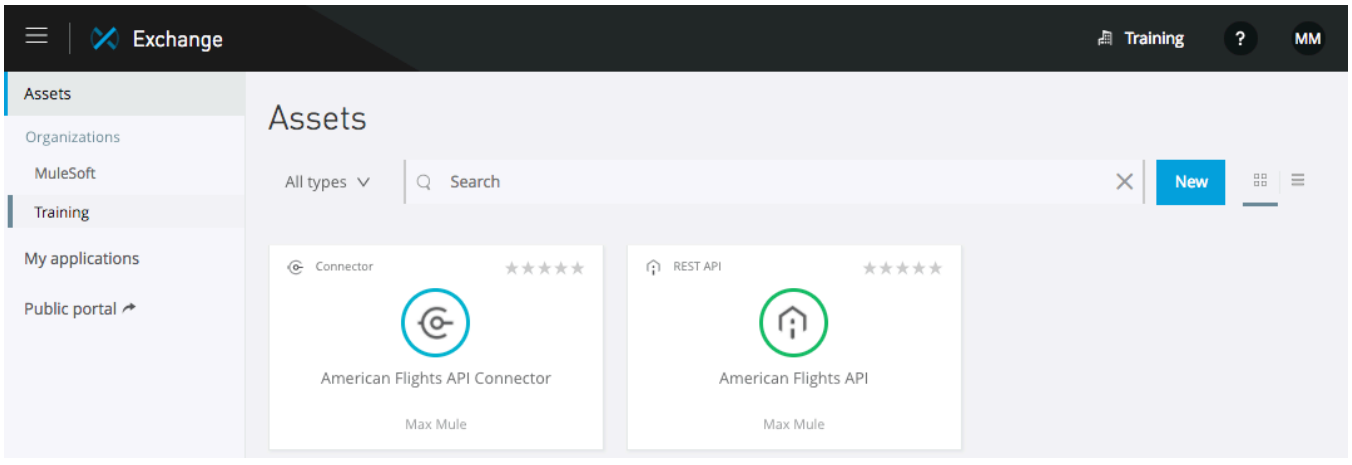
In this walkthrough, you consume the American flights RESTful web service that you built that has an API and a connector in Exchange. You will:

- 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.

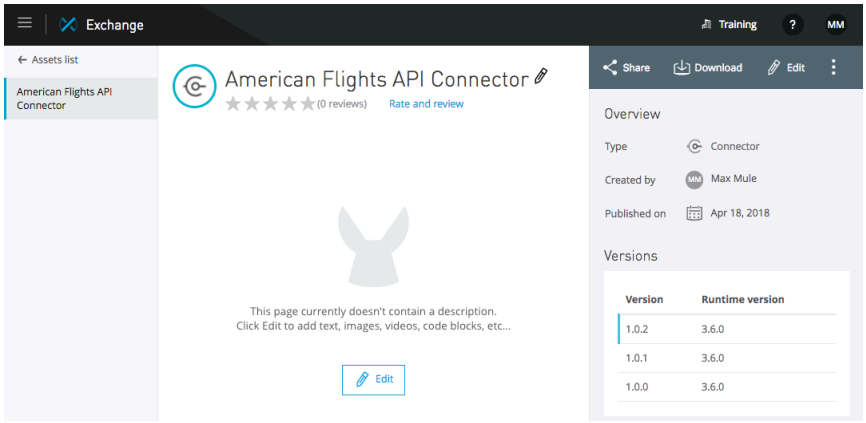


Review the auto-generated REST connector in Exchange

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

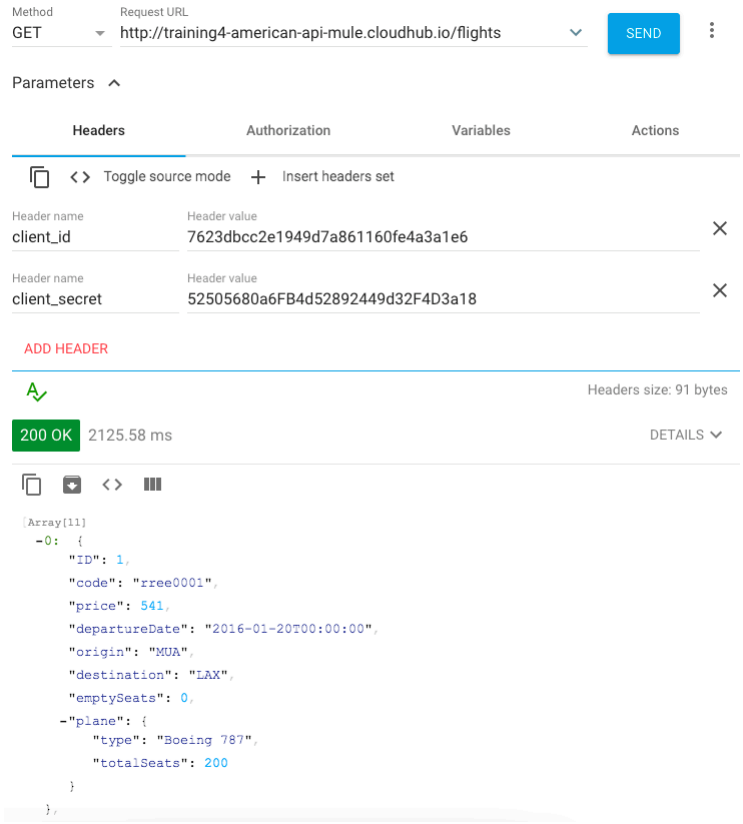


3. Select the connector and review its information.



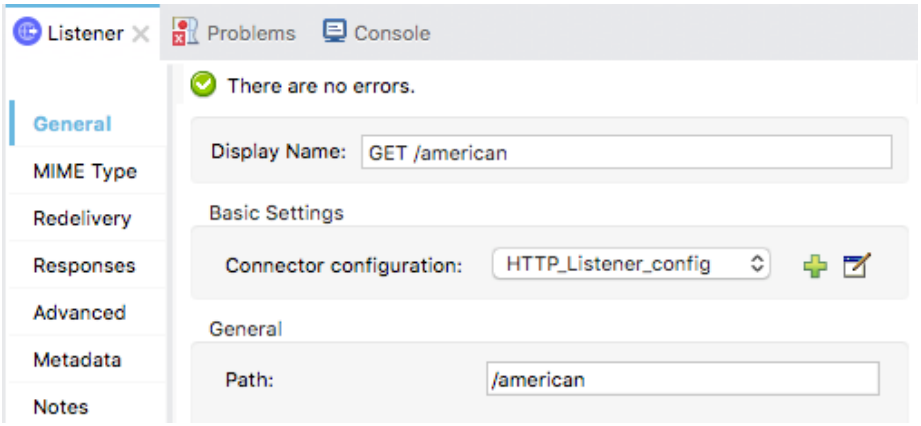
Make a request to the web service

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



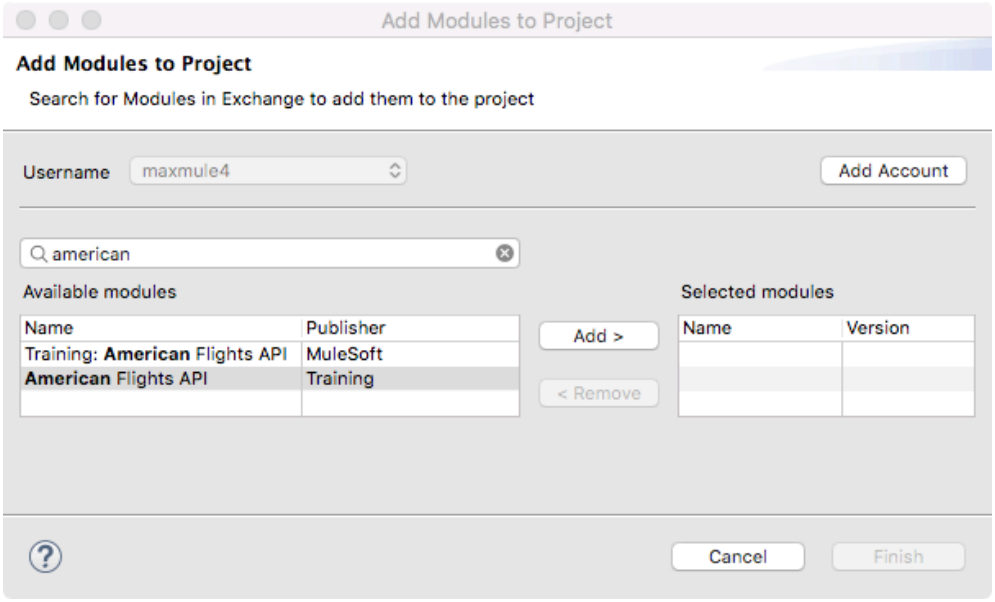
Add a new flow with an HTTP Listener

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



Add the American Flights API module to Anypoint Studio

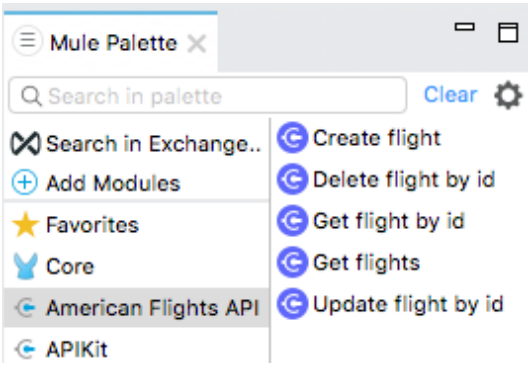
15. In the Mule Palette, select Search in Exchange.
16. In the Add Modules to Project dialog box, enter American in the search field.



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

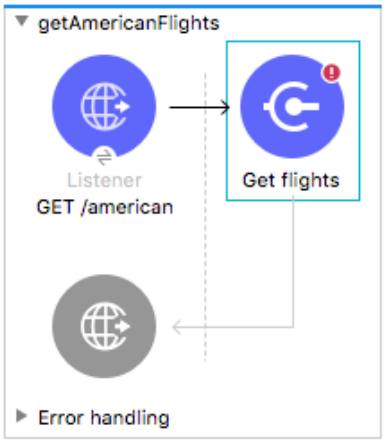
    Note: If you did not successfully create the American Flights API and connector in the first part of the course, you can use the pre-built Training: American Flights API connector. This connector does not require client authentication.

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

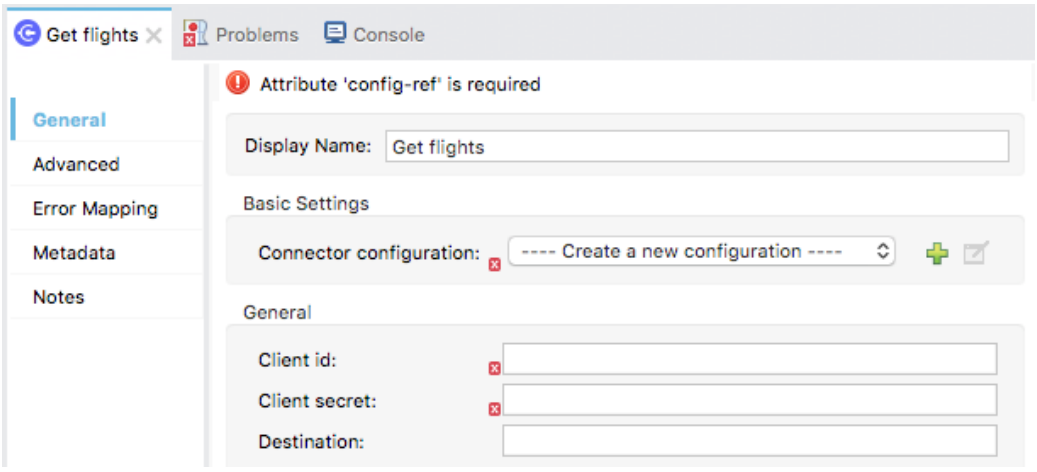


Add a Get all flights operation

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

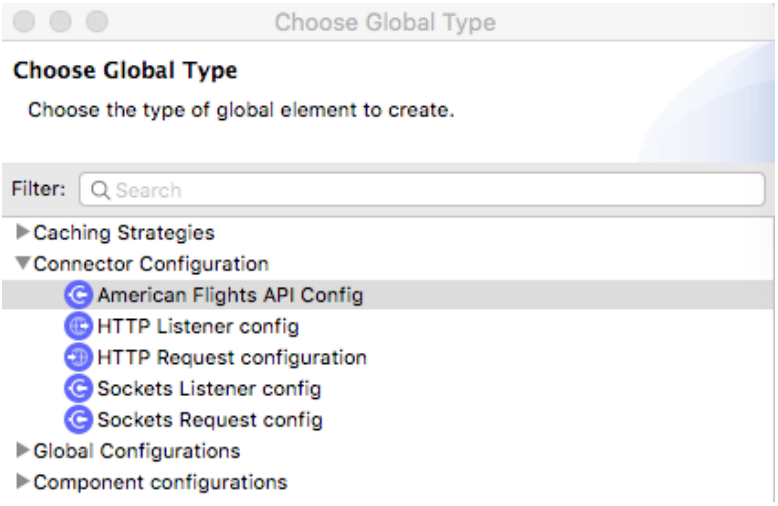


22. Select the Get all flights operation in the canvas and in its properties view, see that you need to need to create a new configuration and enter a client_id and client_secret.

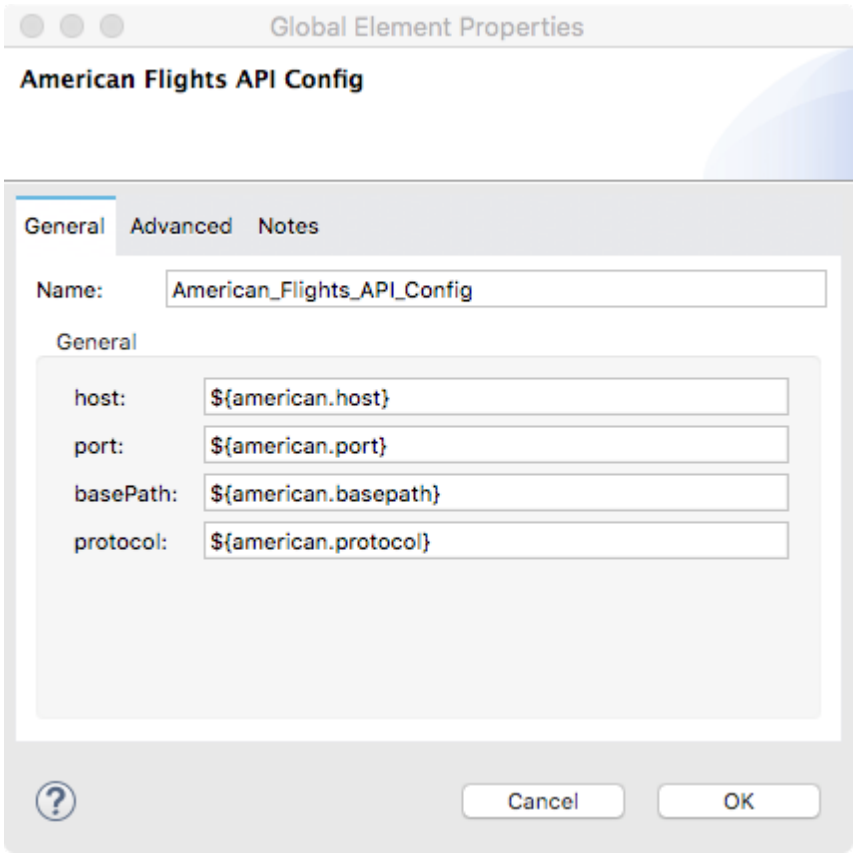


Configure the American Flights API connector

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

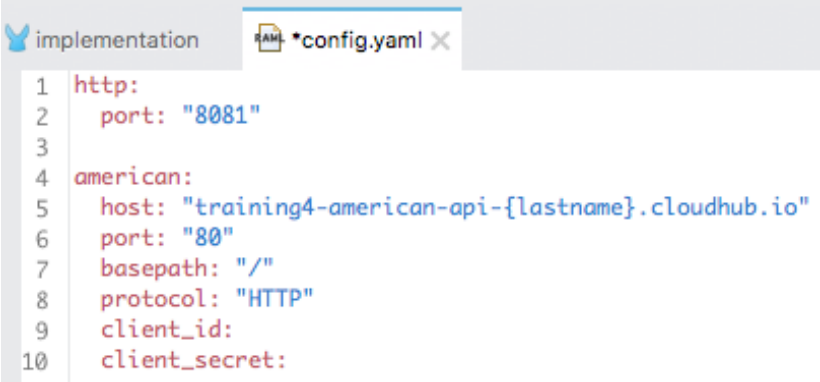


26. In the Global Element Properties dialog box, set each field to a corresponding property placeholder (that you will define and set next):
    - host: ${american.host}
    - port: ${american.port}
    - basePath: ${american.basepath}
    - protocol: ${american.protocol}



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

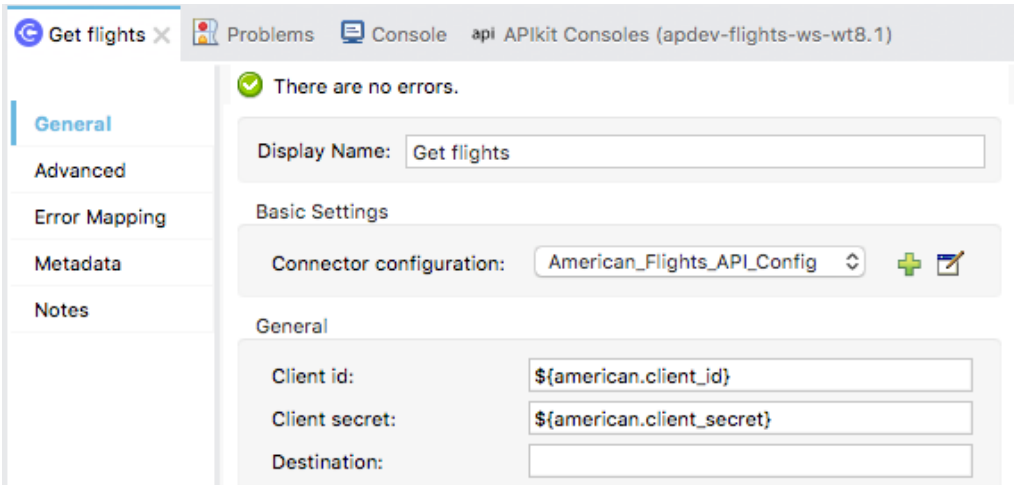
29. Return to config.yaml in src/main/resources and paste the code at the end of the file.



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

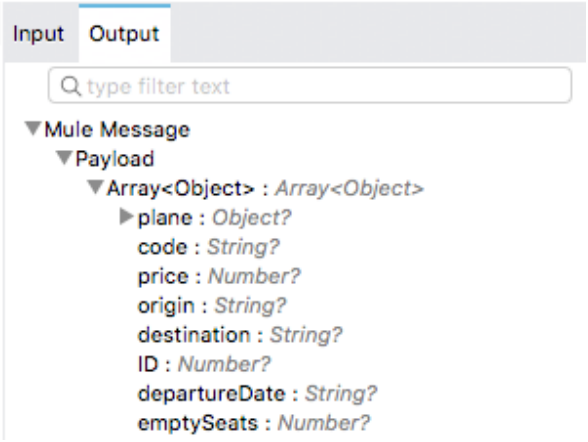
Configure the Get flights operation

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



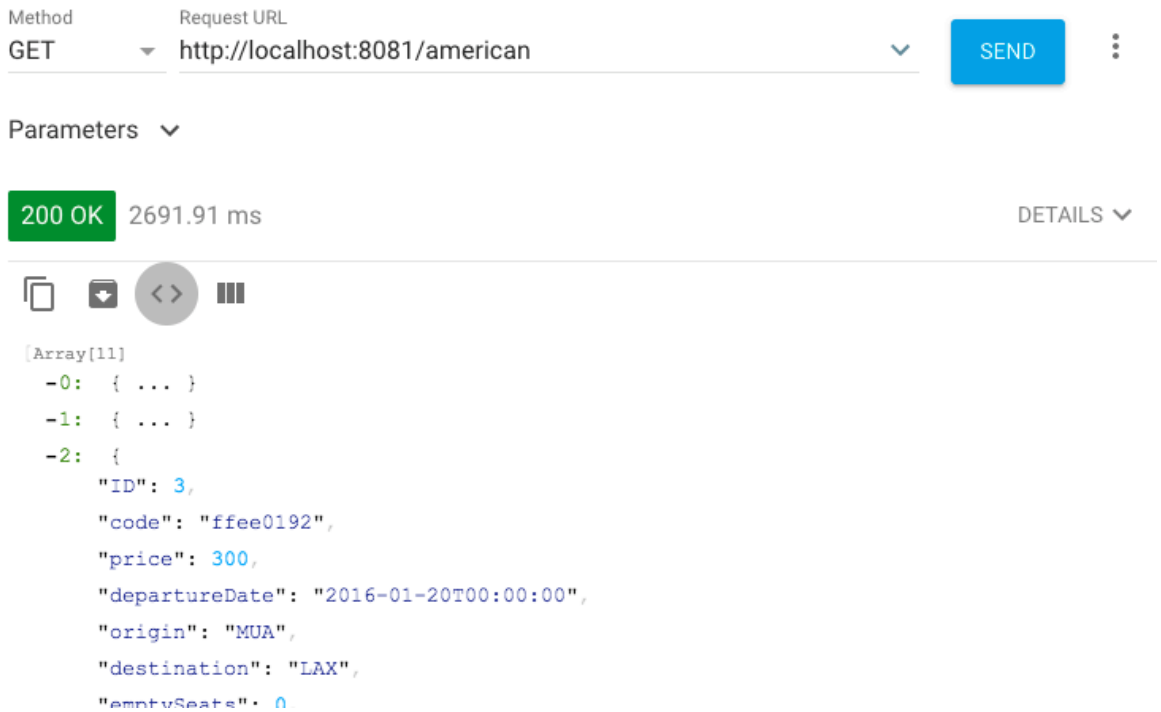
Review metadata associated with the operation

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



Test the application

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



Review the specification for the API you are building

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

```
/flights:
  get:
    displayName: Get flights
    queryParameters:
      code:
        displayName: Destination airport code
        required: false
        enum:
          - SFO
          - LAX
          - PDX
          - CLE
          - PDF
      airline:
        displayName: Airline
        required: false
        enum:
          - united
          - delta
          - american
```

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

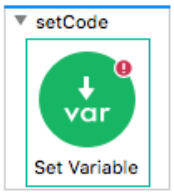
```
responses:
  200:
    body:
      application/json:
        type: Flight[]
        example: !include examples/FlightsExample.raml
```

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

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

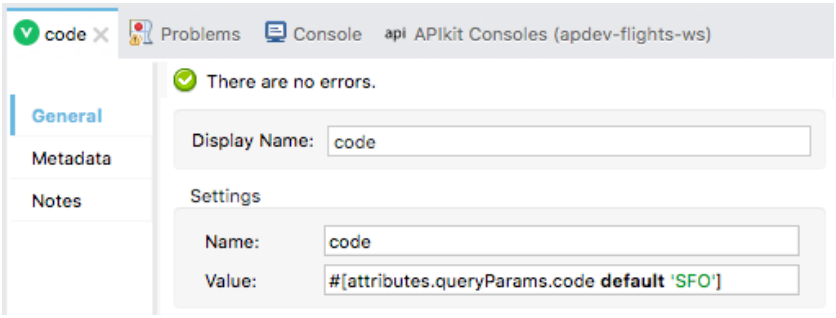
Create a variable to set the destination airport code

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



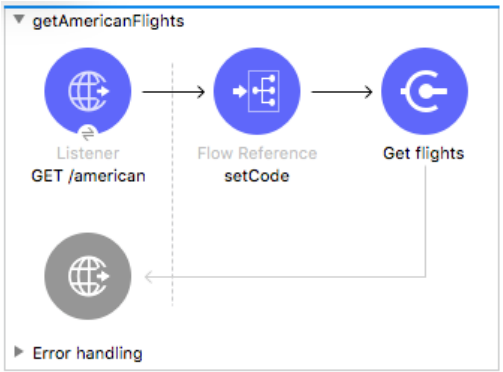
50. In the Set Variable properties view, set the display name and name to code.
51. Set the value to a query parameter called code and give it a default value of SFO if no query parameter is passed to the flow.

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



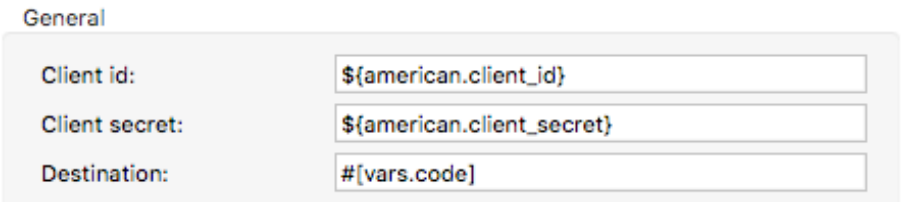
Dynamically set a query parameter for the web service call

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



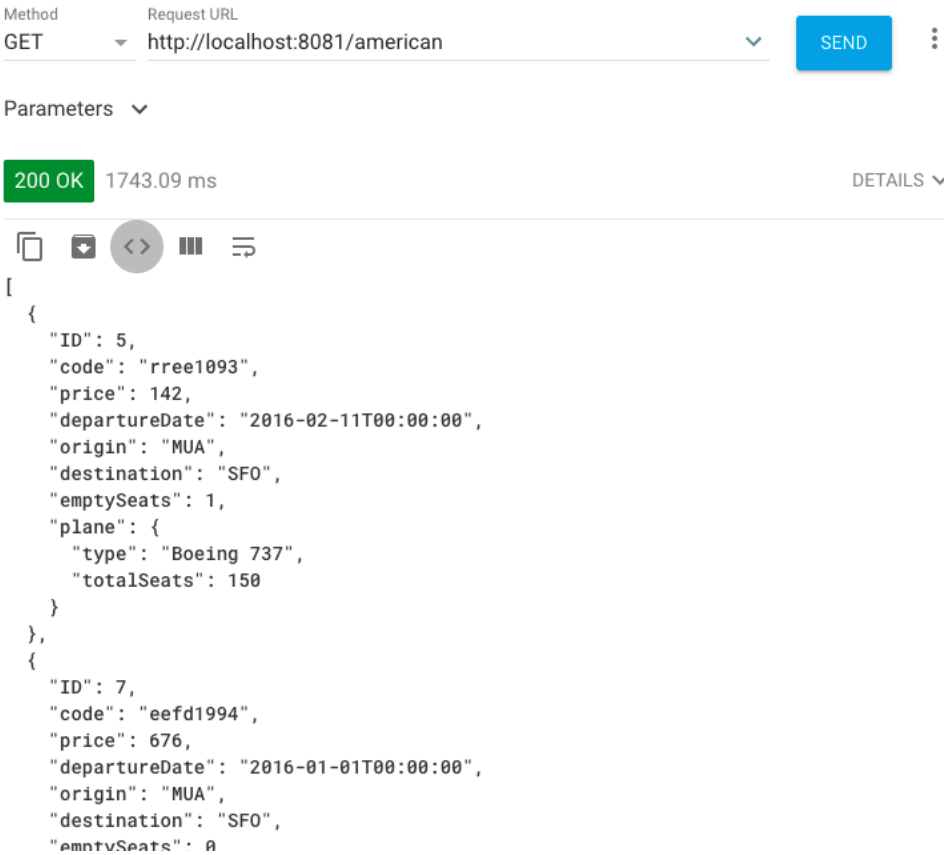
54. In the Get flights properties view, set the value of the destination parameter to the value of the variable containing the airport code.

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

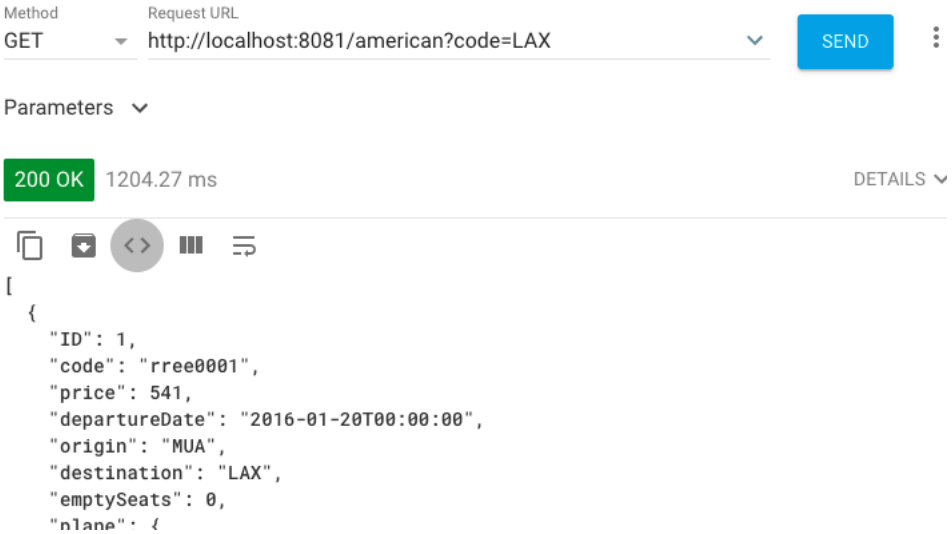


Test the application

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



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



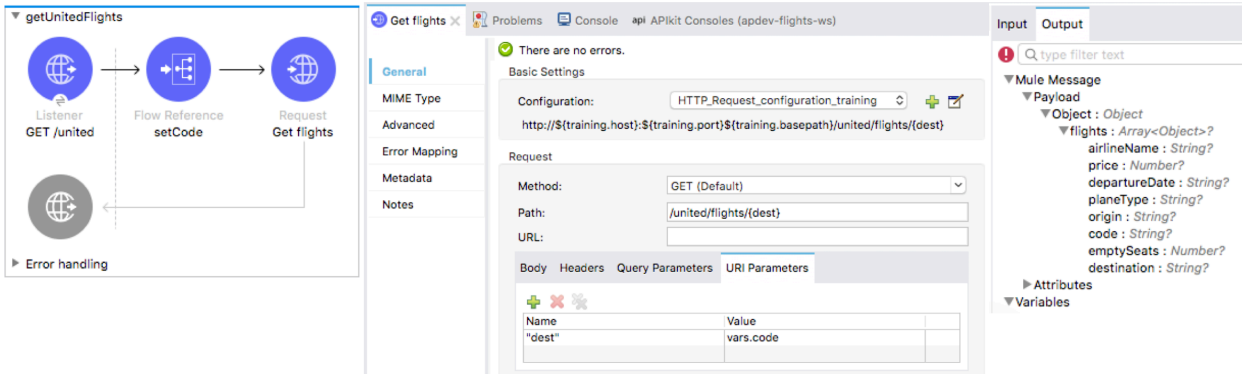
58. Examine the data structure of the JSON response.

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

Walkthrough 8-2: Consume a RESTful web service

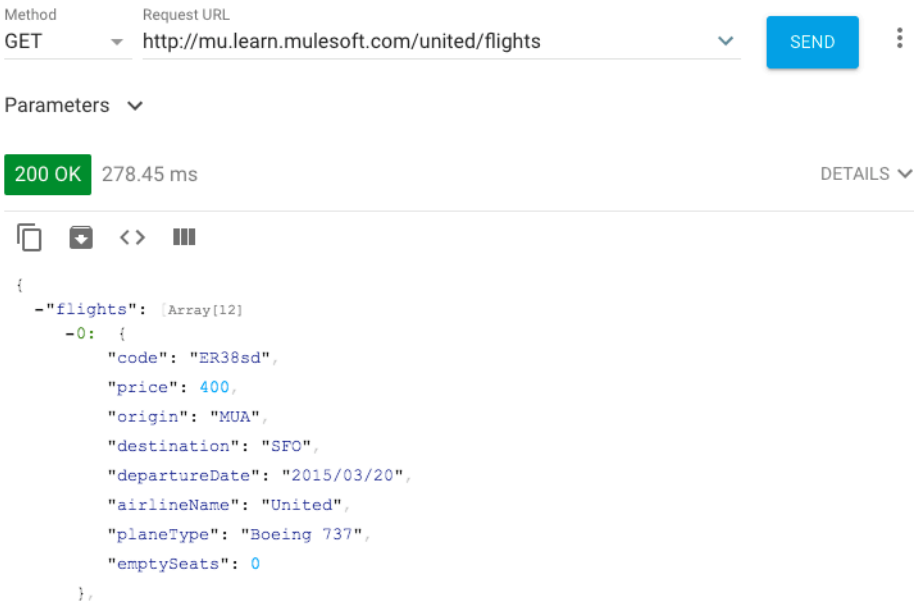
In this walkthrough, you consume the United RESTful web service that does not have an API in Exchange. You will:

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



Make a request to the United web service

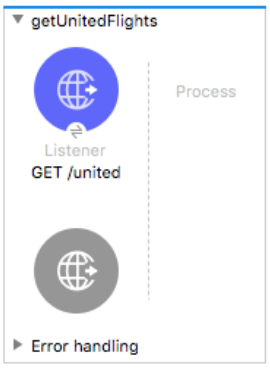
1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Advanced REST Client, make a new tab and make a GET request to this URL.
4. Review the structure of the JSON response.



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

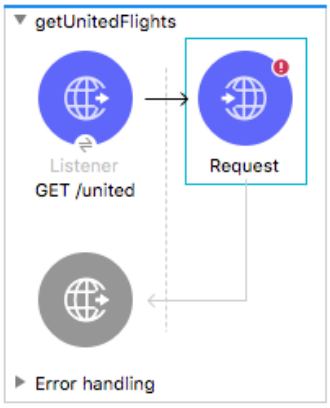
Add a new flow with an HTTP Listener operation

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



Add an HTTP Request operation

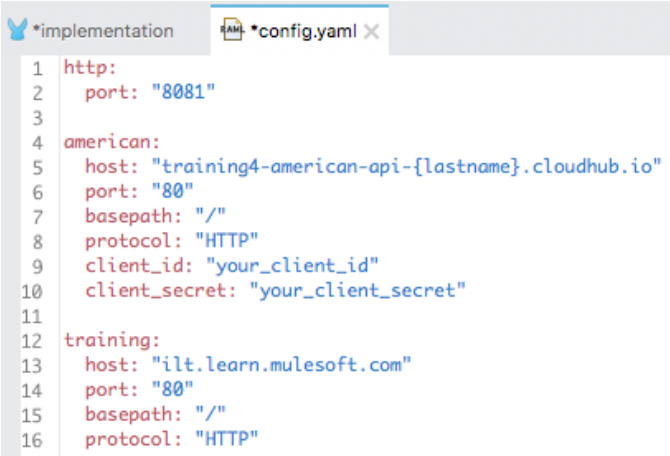
15. Drag out an HTTP Request from the Mule Palette and drop it into the process section of getUnitedFlights.



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

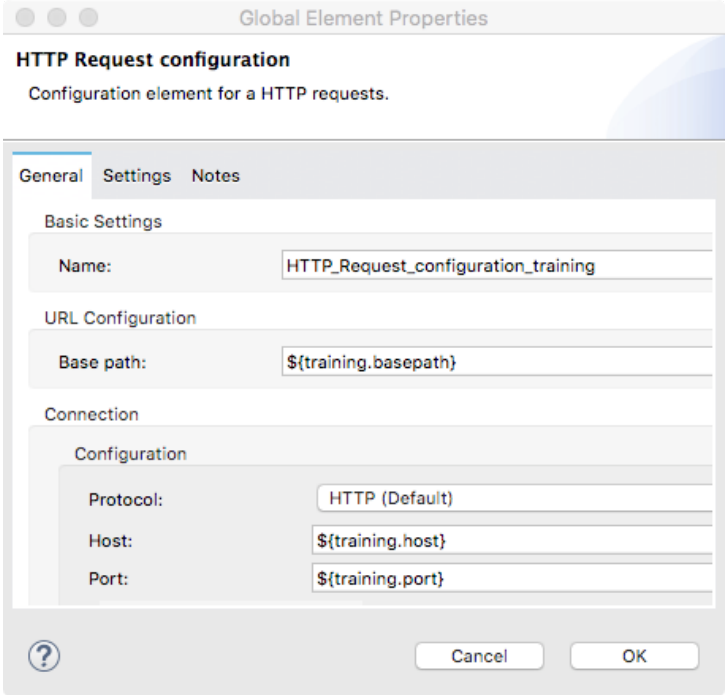
Create an HTTP Request configuration

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



19. Return to the Global Elements view in global.xml.
20. Create a new HTTP Request configuration with the following values:

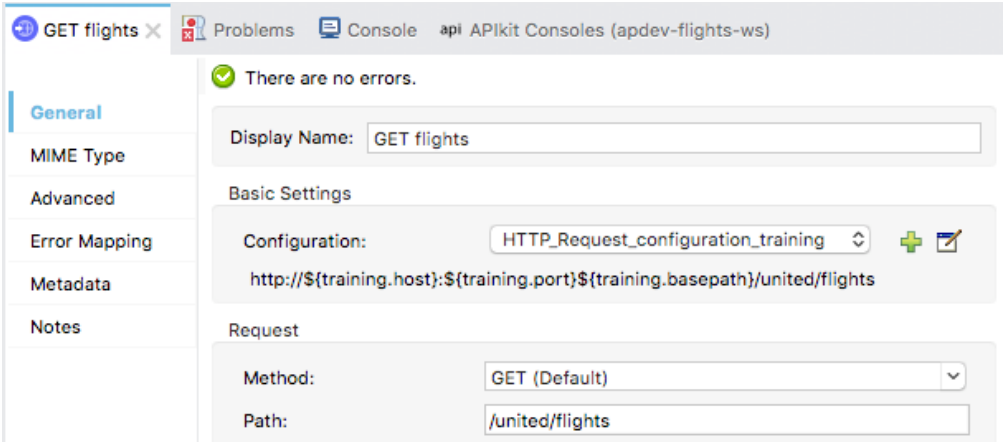
- Name: HTTP_Request_config_training
- Base Path: ${training.basepath}
- Host: ${training.host}
- Port: ${training.port}



21. Click OK.

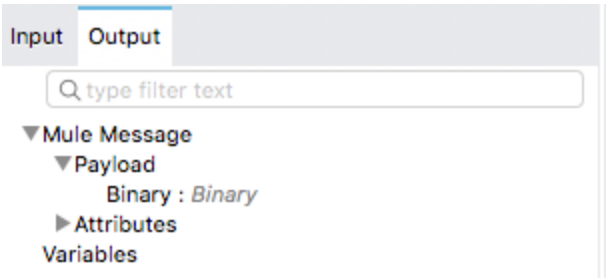
Configure the HTTP Request operation

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



Review metadata associated with the United Get flights operation response

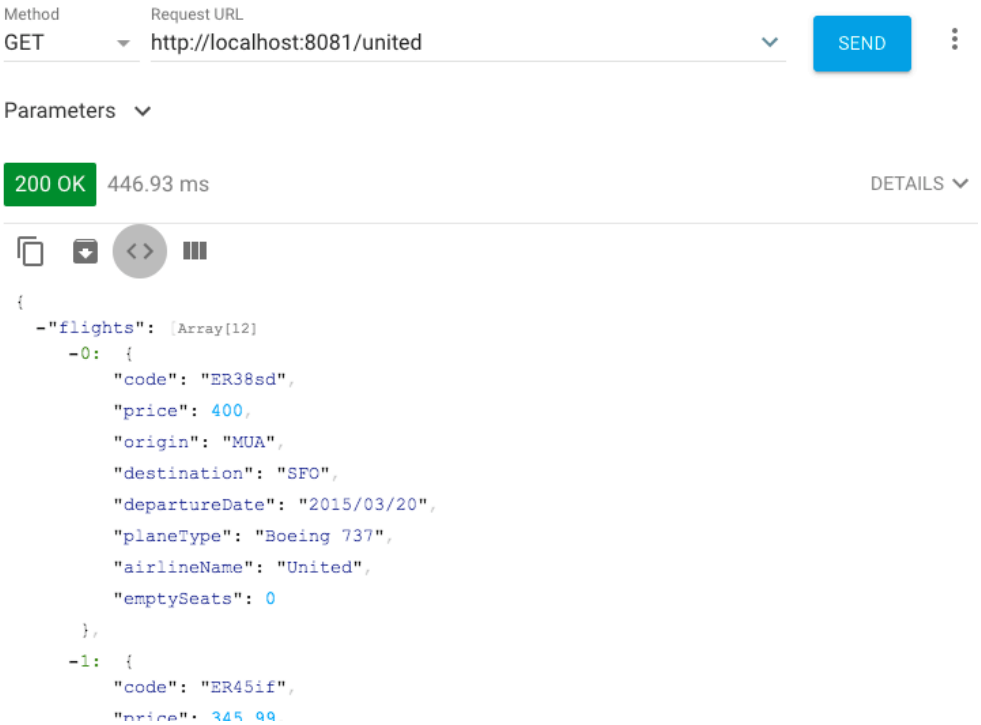
27. Select the Output tab in the DataSense Explorer and expand Payload; you should see no metadata for the payload.



Test the application

28. Save the files to redeploy the project.
29. In Advanced REST Client, return to the tab with the localhost request.

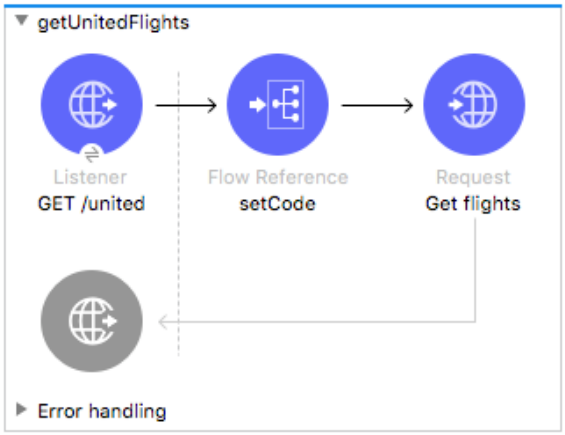
30. Change the URL to make a request to http://localhost:8081/united; you should get JSON flight data for all destinations returned.



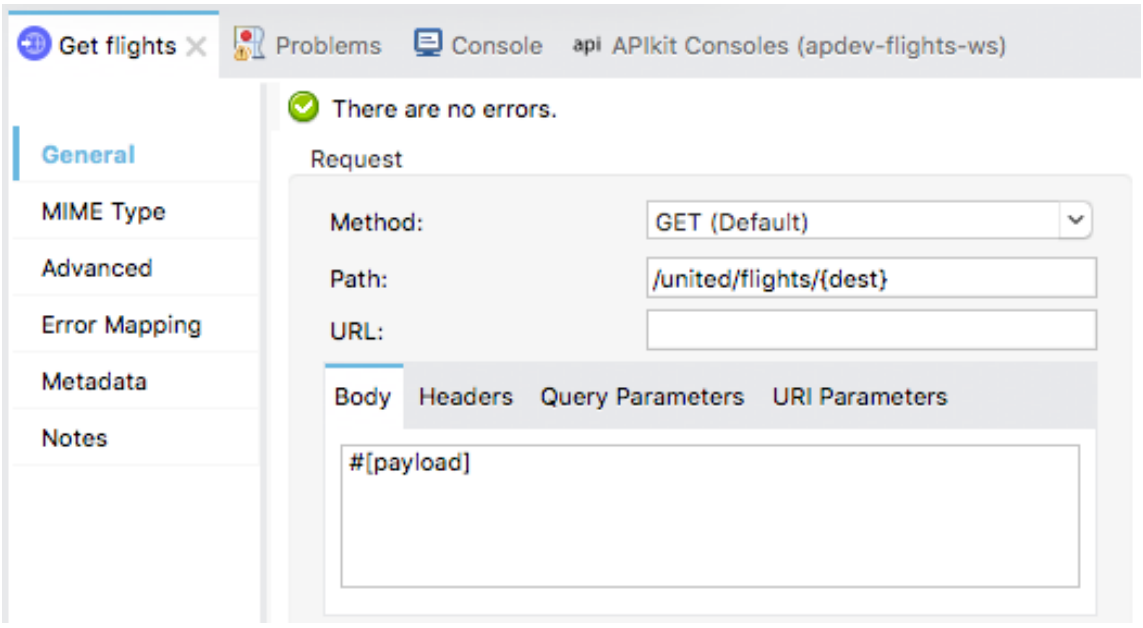
31. Add a query parameter named code and set it to one of the destination airport code values: http://localhost:8081/united?code=CLE; you should still get flights to all destinations.

Add a URI parameter to the web service call

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

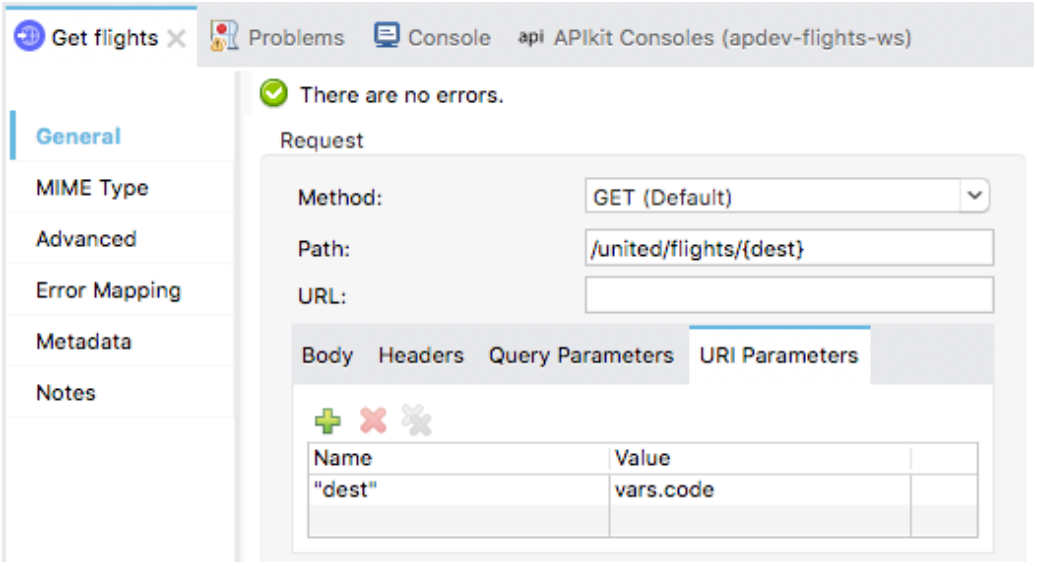


35. Navigate to the Get flights Request properties view in getUnitedFlights.
36. In the Request section, change the path to /united/flights/{dest}.



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

```
vars.code
```



Test the application

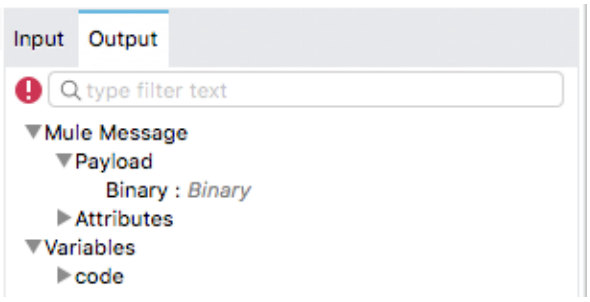
41. Save the file to redeploy the application.
42. In Advanced REST Client, make the same request; you should now only get flights to CLE.

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

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

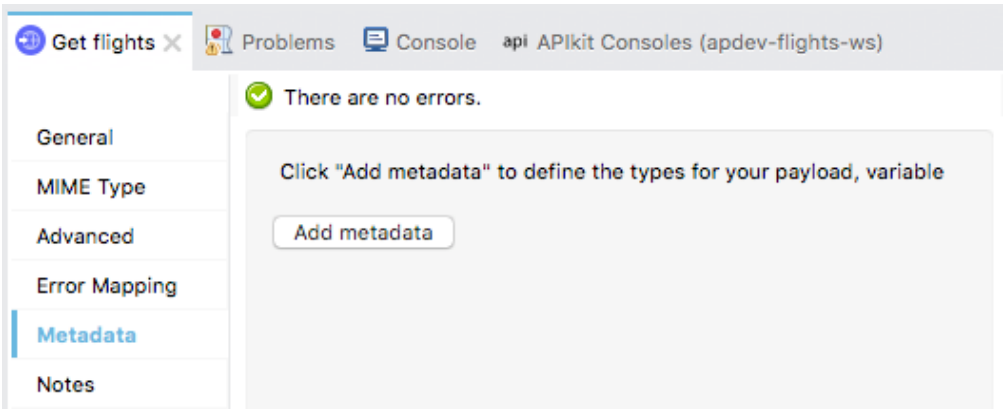
Add metadata for the United Get flights operation response

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

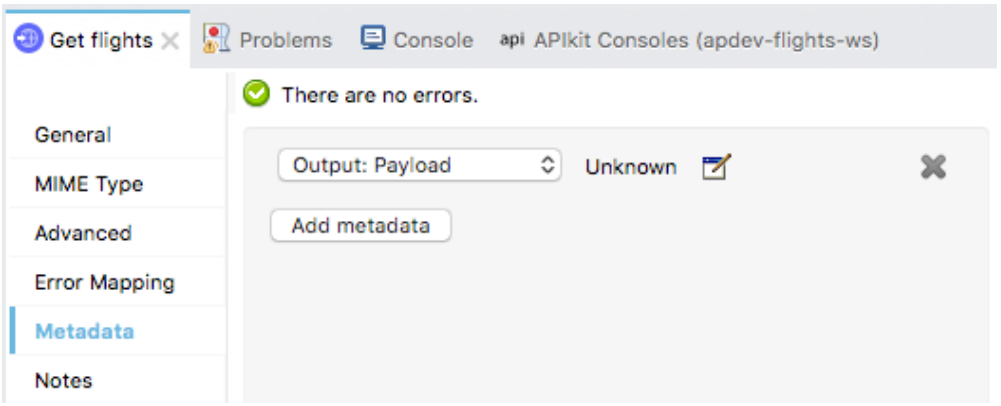


47. In the Get flights properties view, select the Metadata tab.

48. Click the Add metadata button.

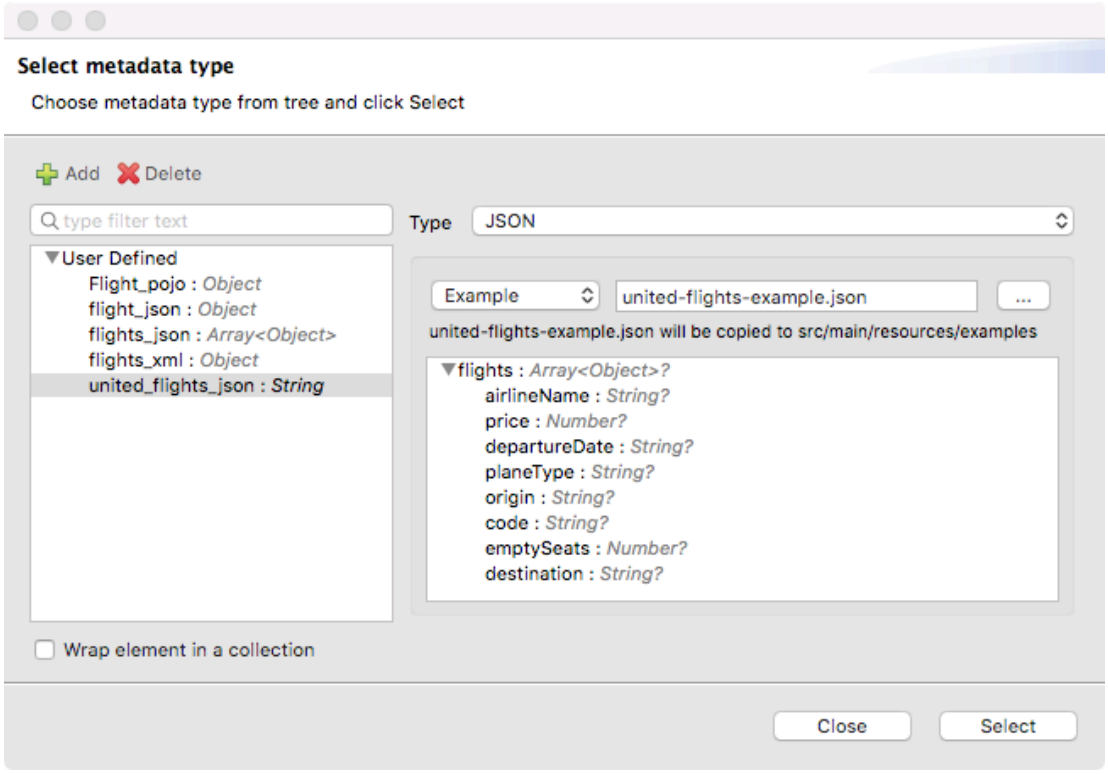


49. Change the drop-down menu to Output: Payload.
50. Click the Edit button.

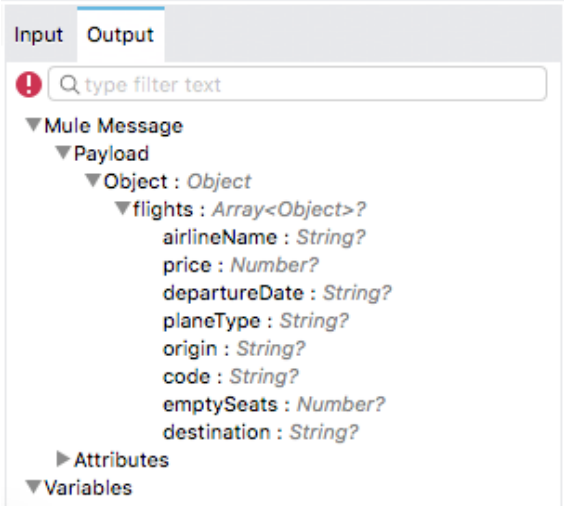


51. In the Select metadata type dialog box, click the Add button.
52. In the Create new type dialog box, set the type id to united_flights_json.
53. Click Create type.
54. In the Select metadata type dialog box, set the type to JSON.
55. Change the Schema selection to Example.
56. Click the browse button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

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



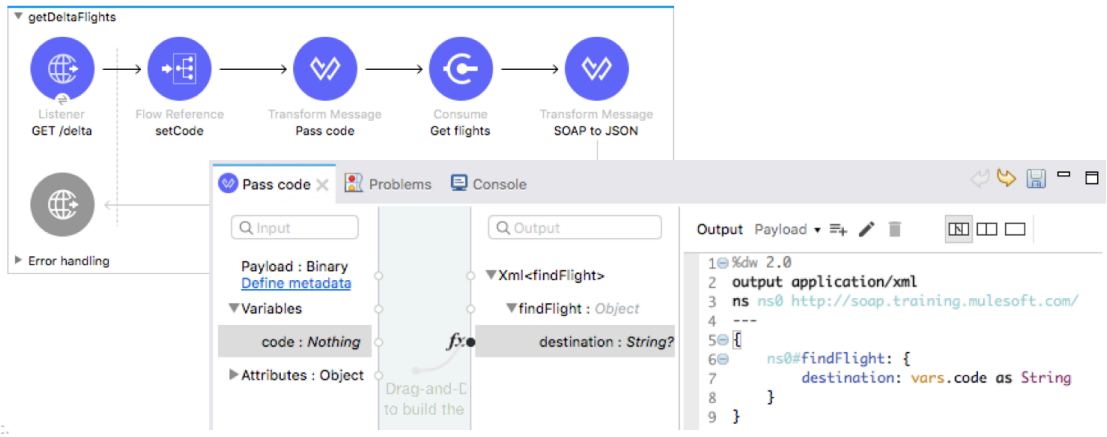
58. Click Select.
59. In the DataSense Explorer, select the Output tab and expand Payload; you should now see the structure for the payload.



Walkthrough 8-3: Consume a SOAP web service

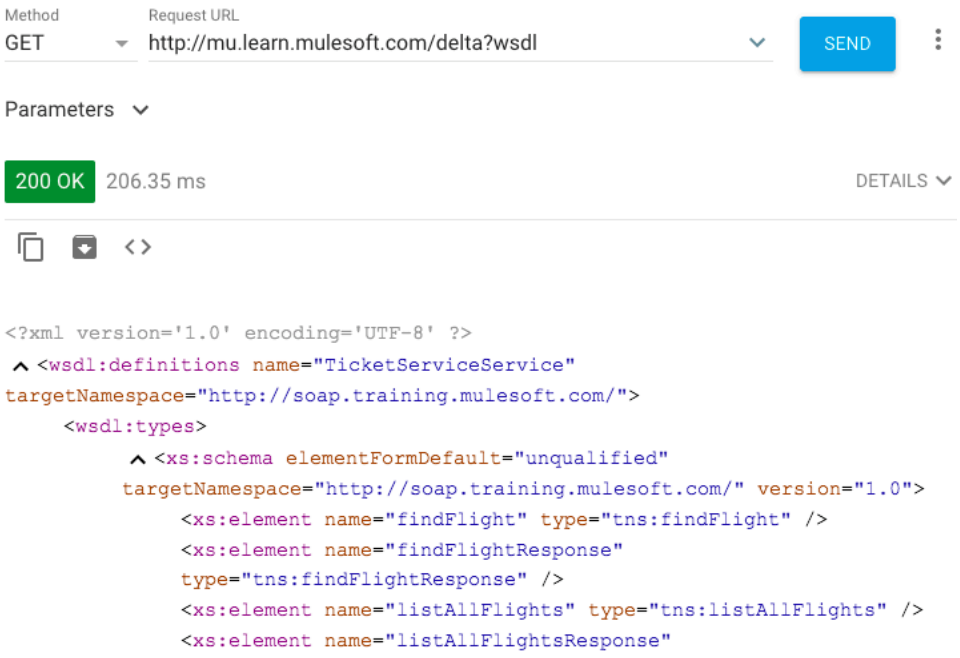
In this walkthrough, you consume a Delta SOAP web service. You will:

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



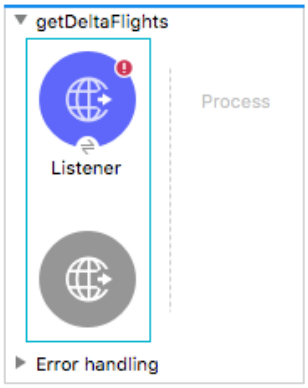
Browse the WSDL

1. Return to the course snippets.txt file and copy the Delta SOAP web service WSDL.
2. In Advanced REST Client, return to the third tab, the one with the learn.mulesoft request.
3. Paste the URL and send the request; you should see the web service WSDL returned.
4. Browse the WSDL; you should find references to operations listAllFlights and findFlight.

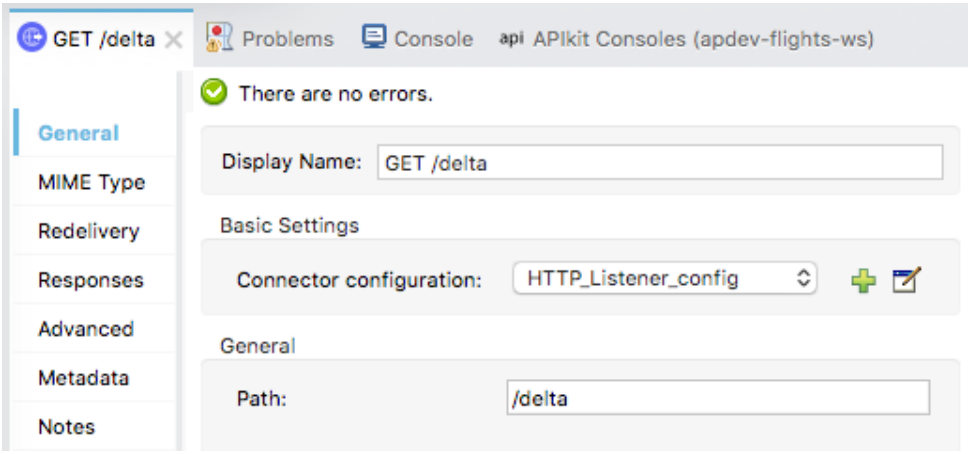


Create a new flow with an HTTP Listener connector endpoint

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



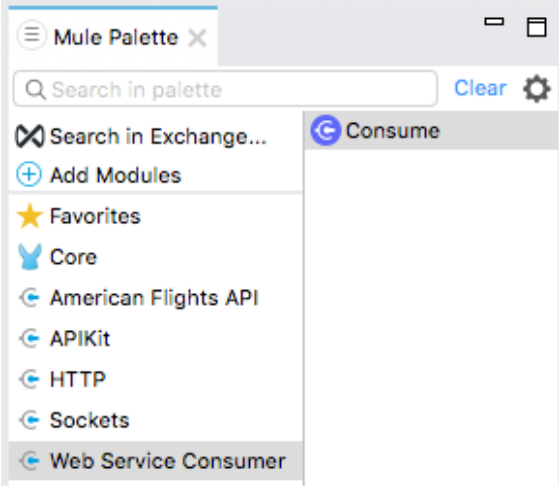
8. In the Listener properties view, set the display name to GET /delta.
9. Set the connector configuration to the existing HTTP_Listener_config.
10. Set the path to /delta and the allowed methods to GET.



Add the Web Service Consumer module to the project

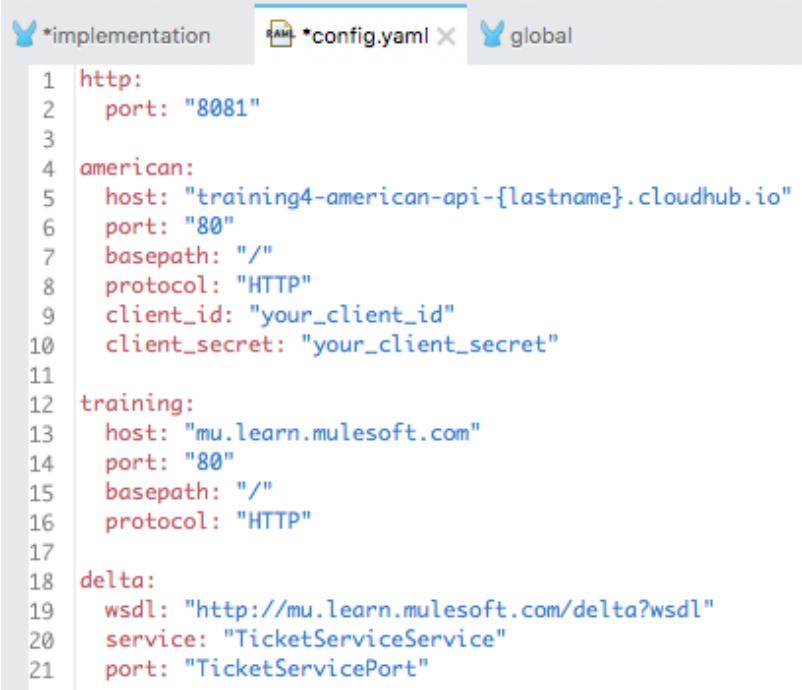
11. In the Mule Palette, select Add Modules.

12. Select the Web Service Consumer connector in the right side of the Mule Palette 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.



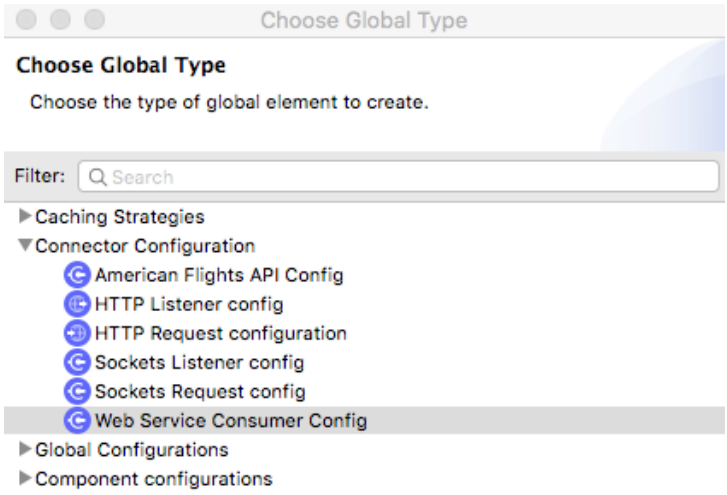
Configure the Web Service Consumer connector

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.



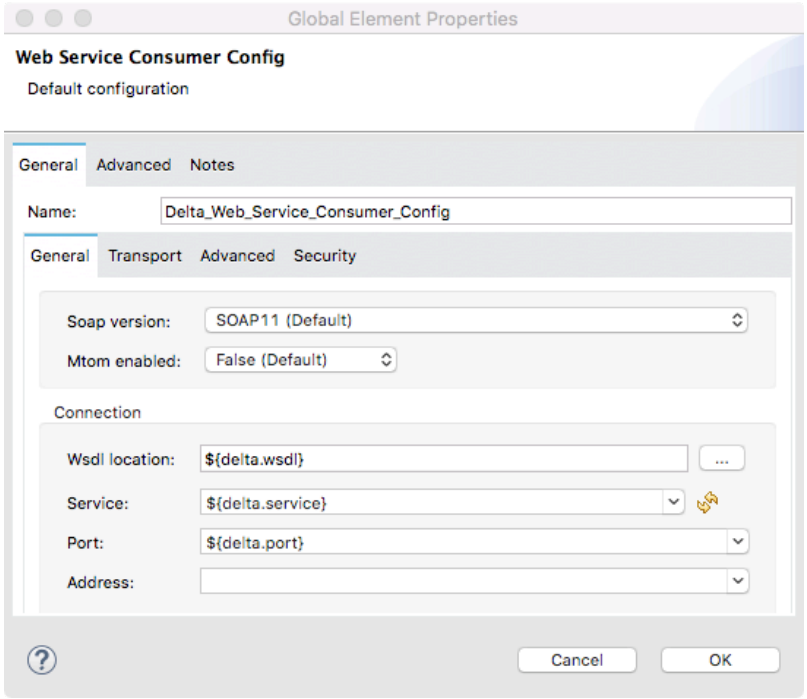
16. Save the file.
17. Return to global.xml.
18. Click Create.

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



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

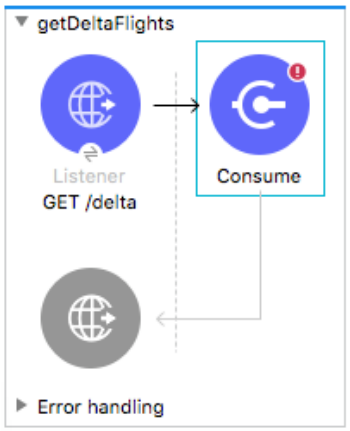
- Wsdl location: ${delta.wsdl}
- Service: ${delta.service}
- Port: ${delta.port}



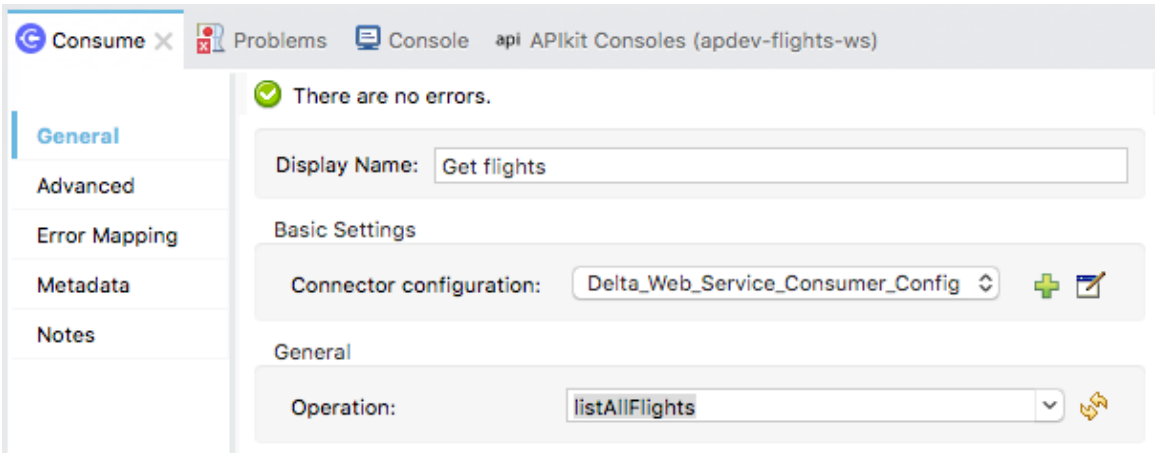
22. Click OK.

Add and configure a Consume operation

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

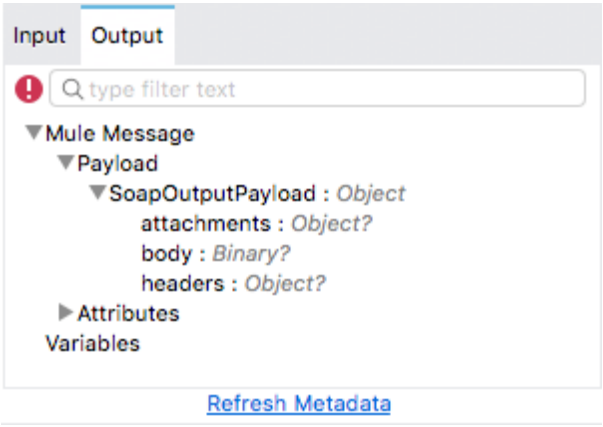


25. In the Consume properties view, change its display name to Get flights.
26. Set the connector configuration to the existing Delta_Web_Service_Consumer_Config.
27. Click the operation drop-down menu button; you should see all of the web service operations listed.
28. Select the listAllFlights operation.

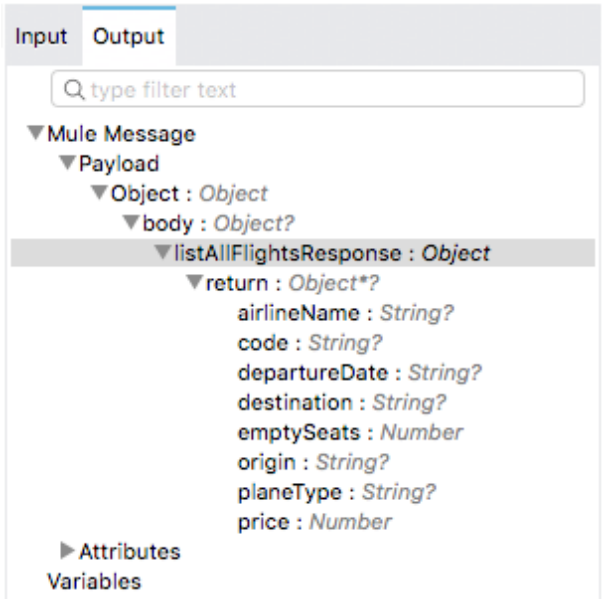


Review metadata associated with the United Get flights operation response

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



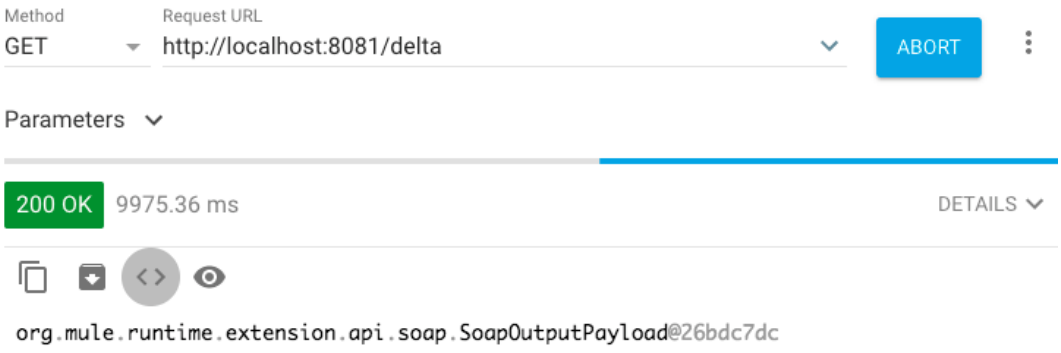
30. Save the file.
31. Select the Output tab in the DataSense Explorer and expand Payload again; you should now see the payload structure.



Test the application

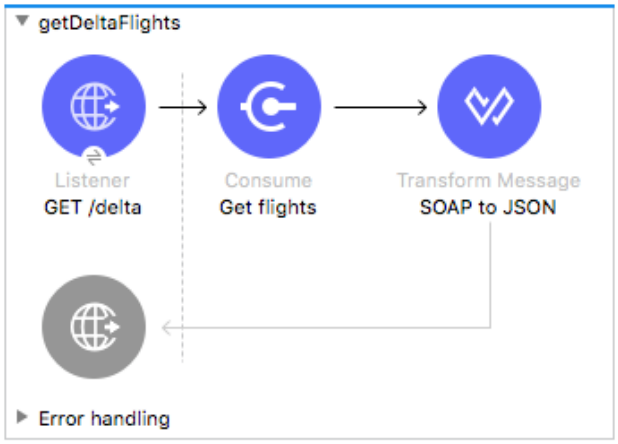
32. Save the files to redeploy the project.
33. In Advanced REST Client, return to the middle tab – the one with the localhost requests.

34. Make a request to http://localhost:8081/delta; you should get a response that is object of type SoapOutputPayload.

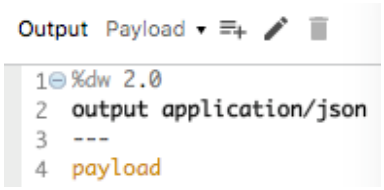


Transform the response to JSON

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



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

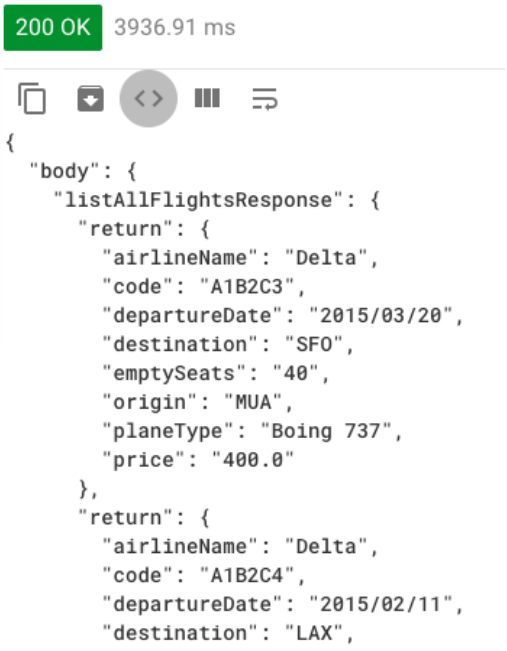


Test the application

39. Save the file to redeploy the project.

40. In Advanced REST Client, make another request to http://localhost:8081/delta; you should get JSON returned.

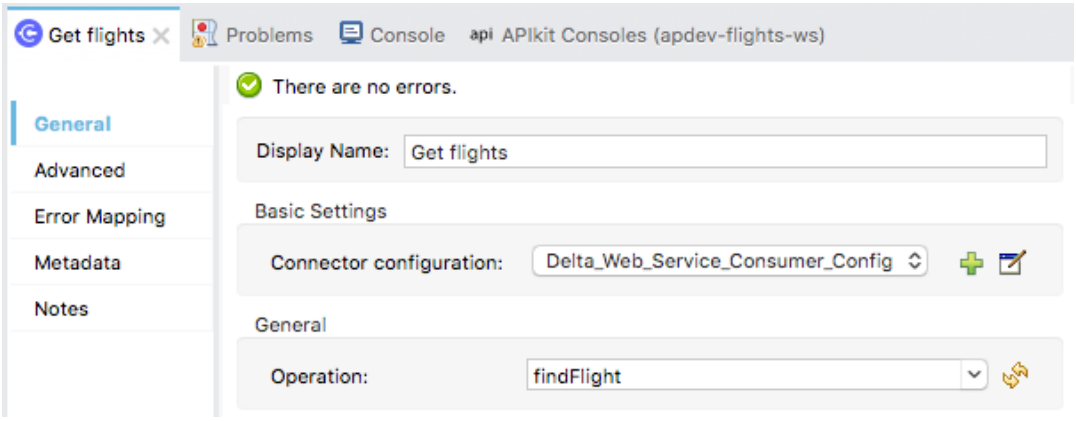
41. Click the Toggle raw response view button; you should see all the flights.



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

Call a different web service operation

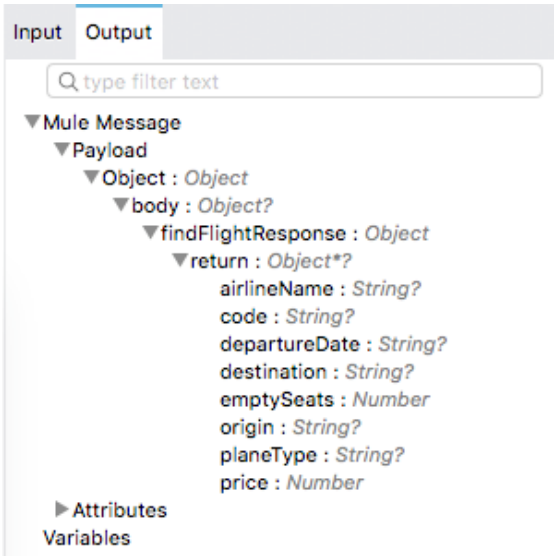
44. Return to getDeltaFlights in Anypoint Studio.
45. In the properties view for Get flights Consume operation, change the operation to findFlight.



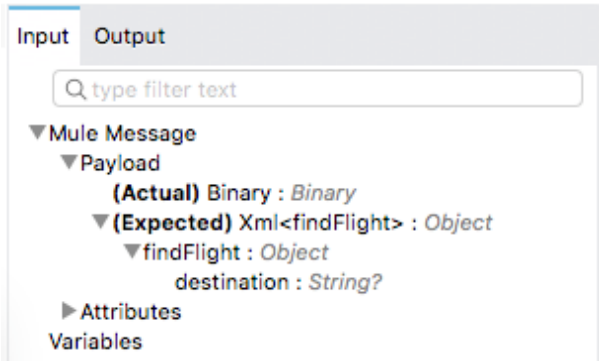
46. Save all files to redeploy the application.

Review metadata associated with the United Get flights operation response

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

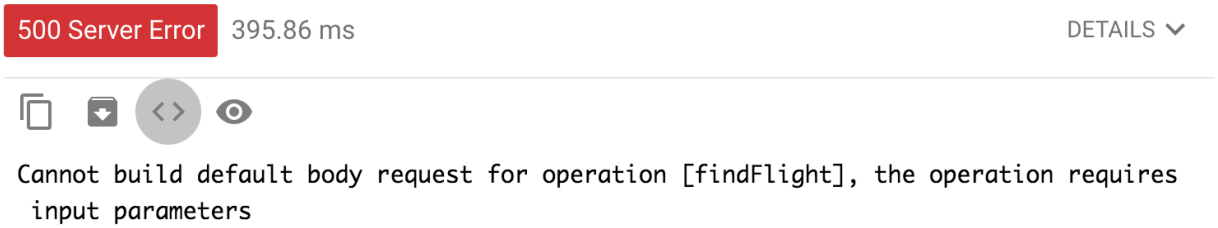


49. Select the Input tab and expand Payload; you should see this operation now expects a destination.



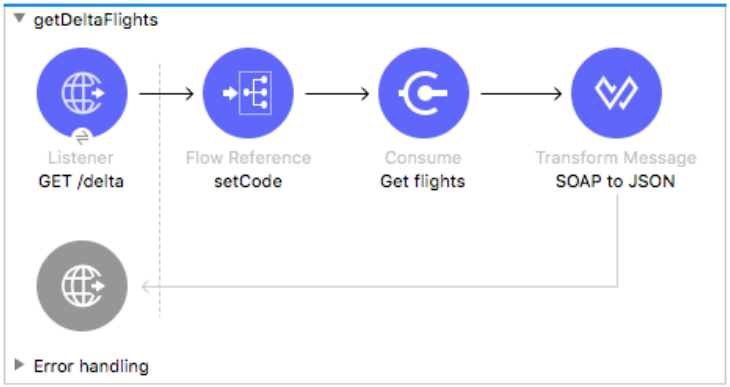
Test the application

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



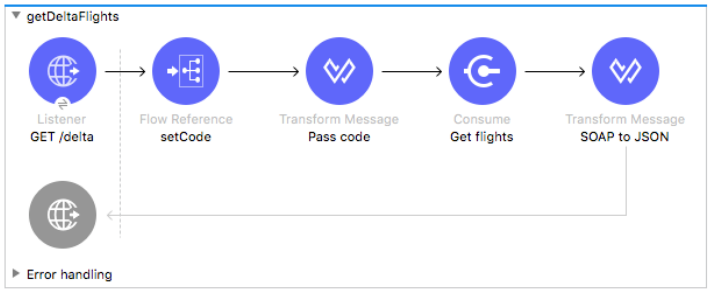
Use the set airport code subflow

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

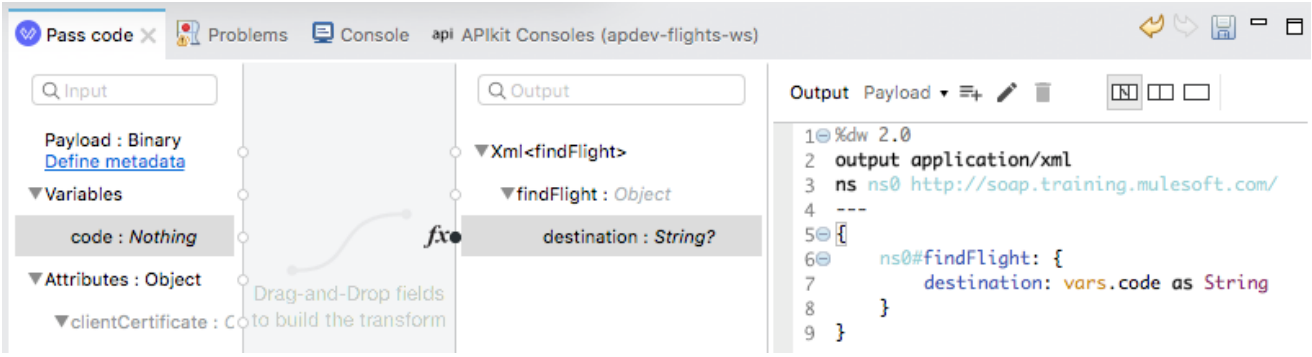


Use the Transform Message component to pass a parameter to the web service

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

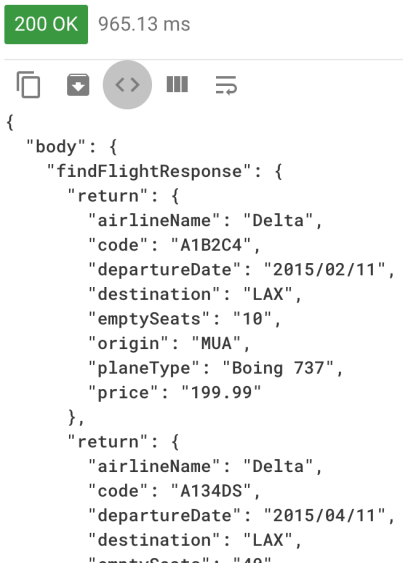


56. In the Pass code properties view, look at the input and output sections.
57. Drag the code variable in the input section to the destination element in the output section.



Test the application

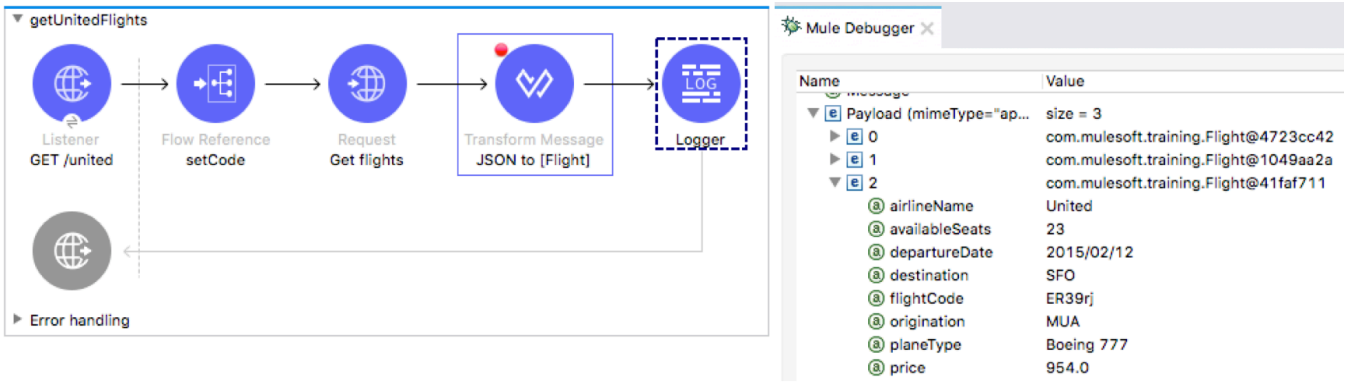
58. Save the file to redeploy the application.
59. In Advanced REST Client, make another request.; you should now only see flights to LAX.



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

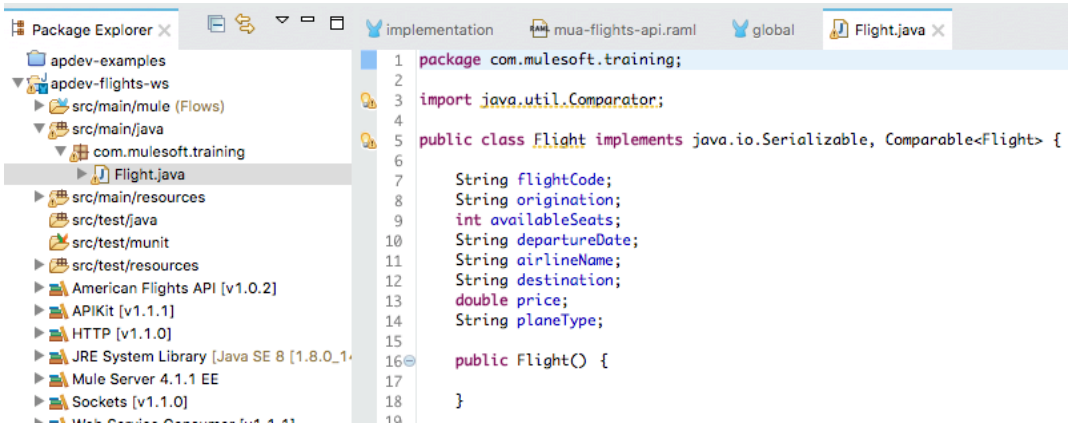
In this walkthrough, you will transform the JSON returned from the American and United web services and the SOAP returned form the Delta web service to the same format. You will:

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



Review Flight.java

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

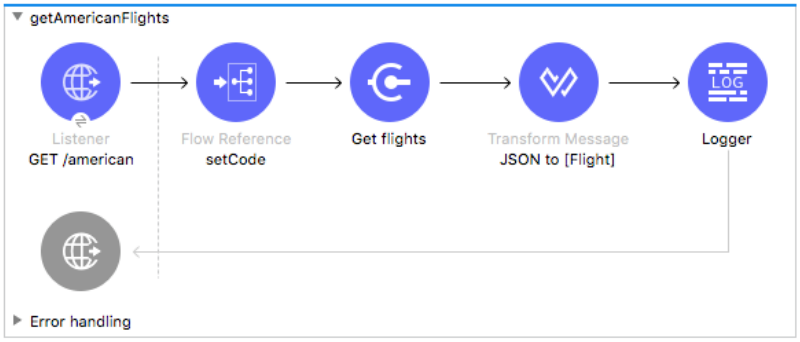


3. Close the file.

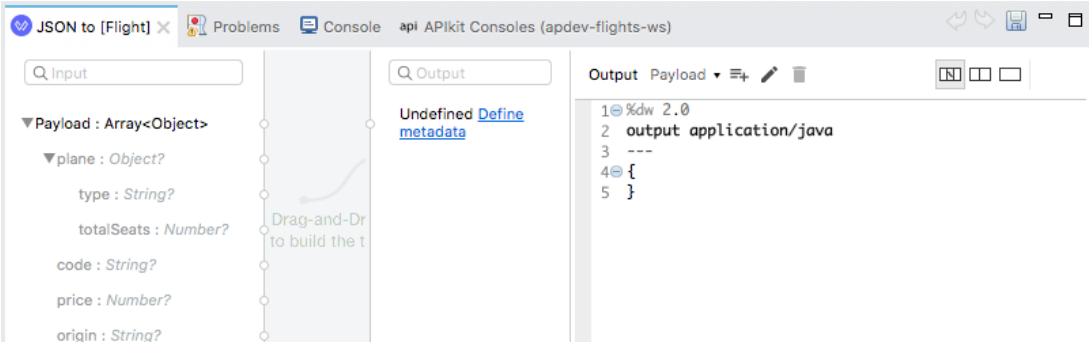
Change the American flow to return Java Flight objects instead of JSON

4. Return to getAmericanFlights in implementation.xml.
5. Add a Transform Message component to the end of the flow.
6. Add a Logger at the end of the flow.

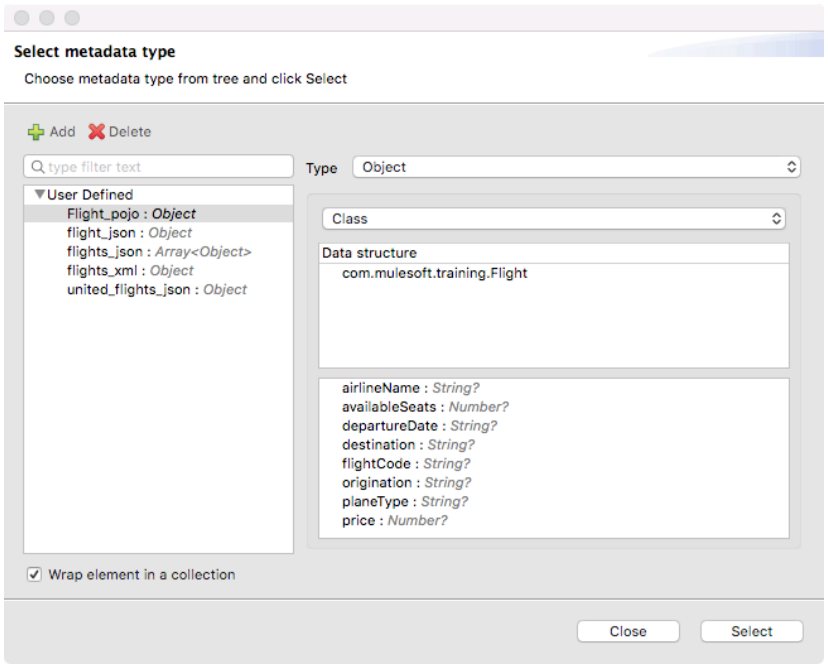
7. Change the name of the Transform Message component to JSON to [Flight].



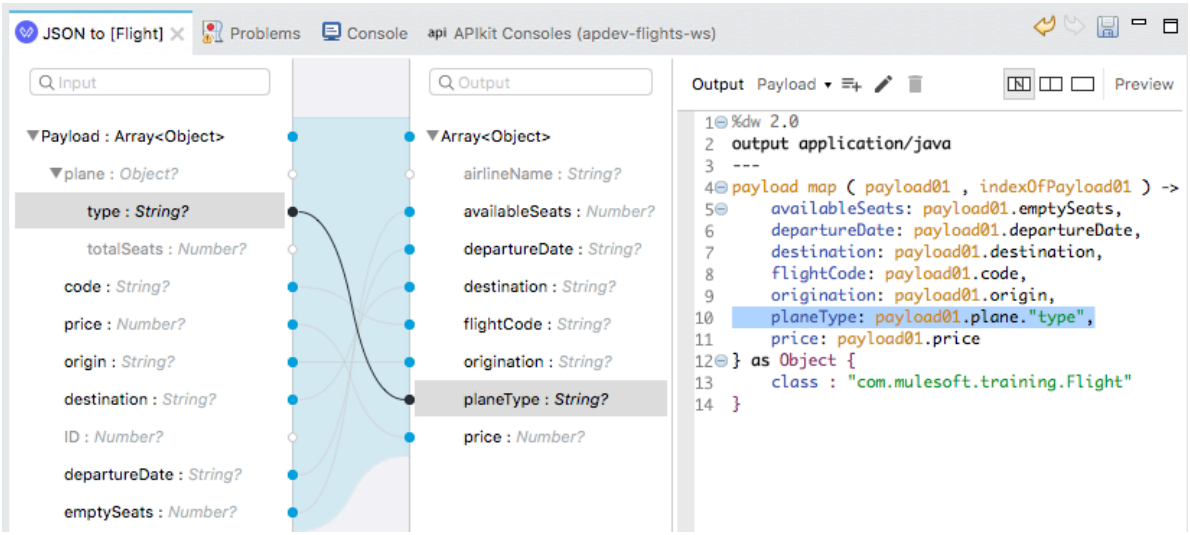
8. In the Transform Message properties view, look at the input section in the Transform Message properties view; you should see metadata already defined.



9. In the output section of the Transform Message properties view, click the Define metadata link.
10. In the Select metadata type dialog box, select the user-defined Flight_pojo.
11. Select Wrap element in a collection in the lower-left corner.



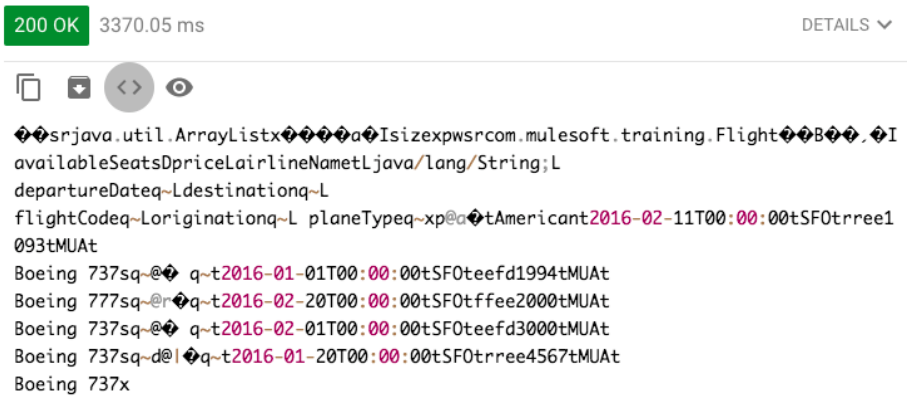
12. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
13. Map fields (except ID and totalSeats) by dragging them from the input section and dropping them on the corresponding field in the output section.



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

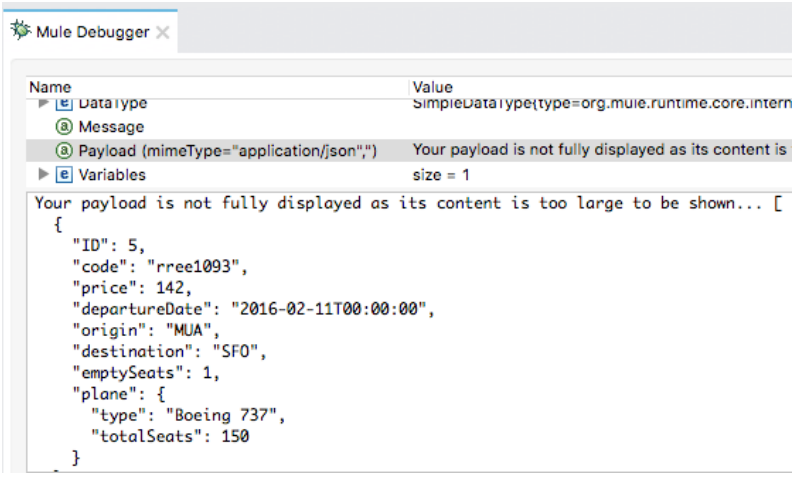
Test the application

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

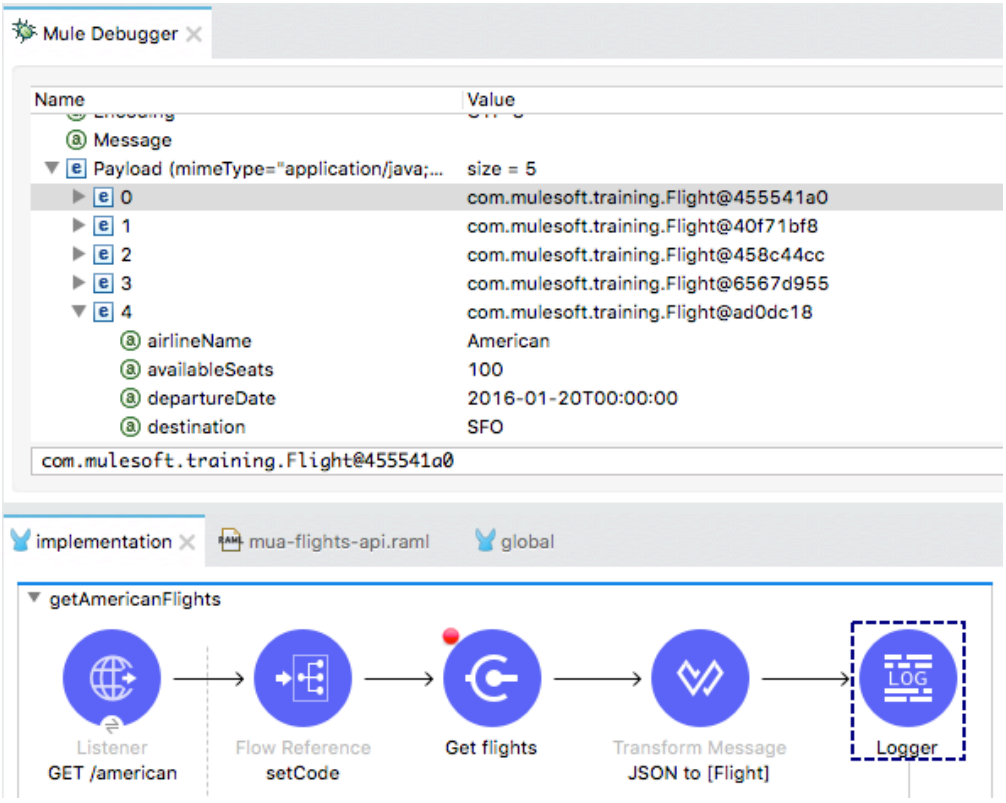


Debug the application

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



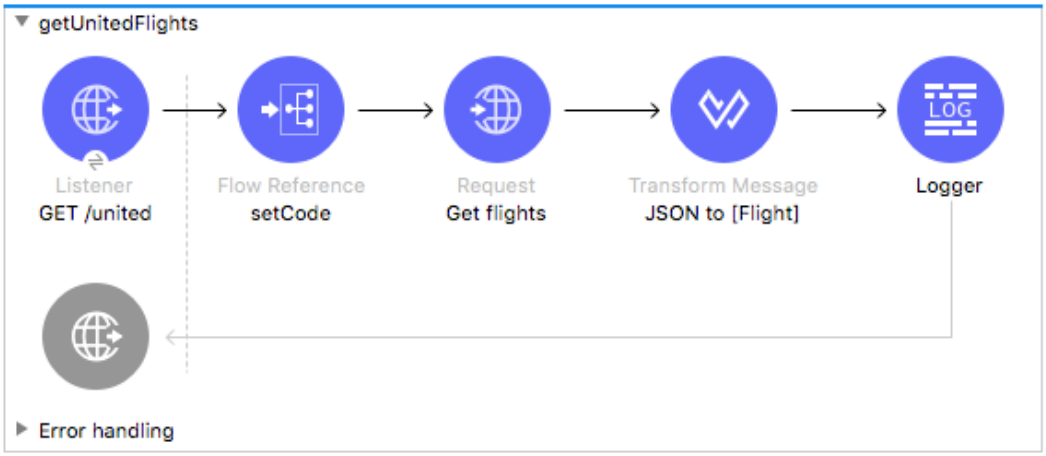
24. Step to the Logger and look at the payload it should be a collection or Flight objects.



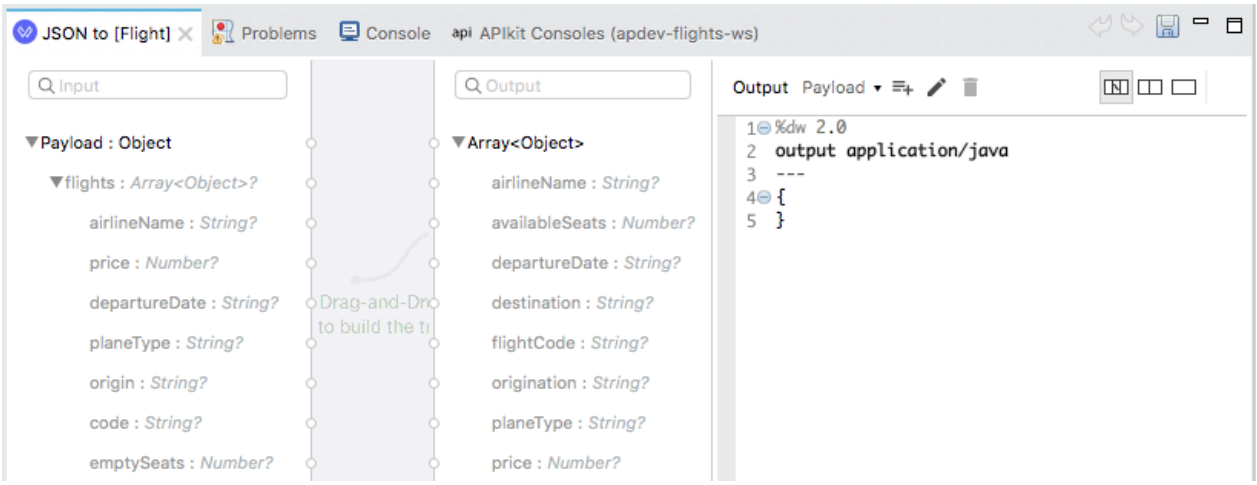
25. Step through the rest of the application and switch perspectives.

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

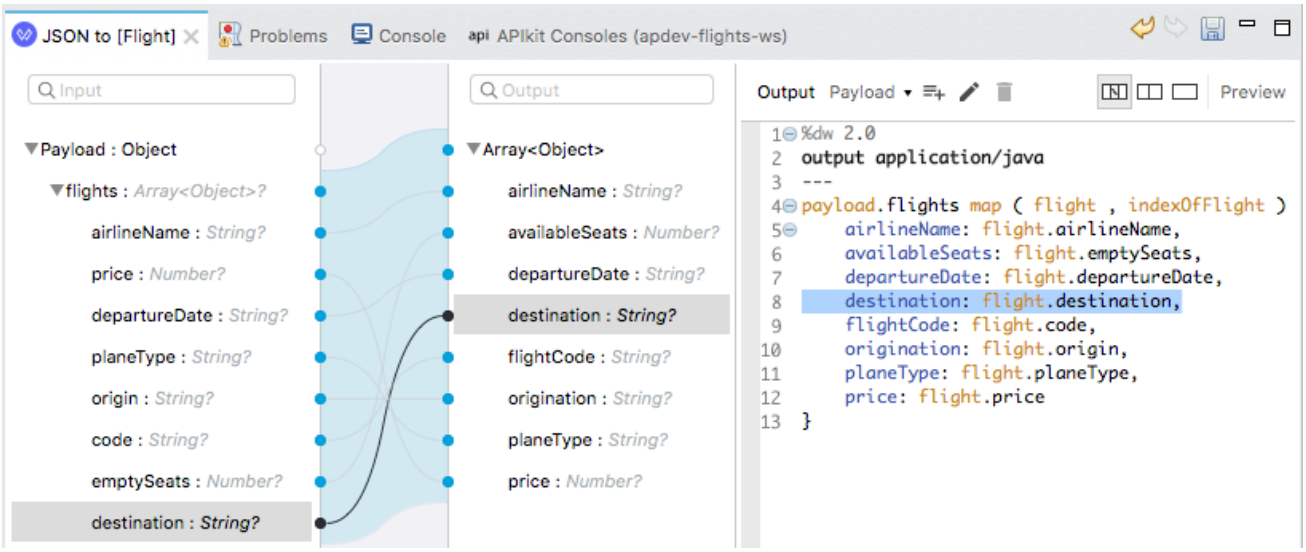
26. Return to geUnitedFlights in implementation.xml.
27. Add a Transform Message component at the end of the flow.
28. Change the name of the Transform Message component to JSON to [Flight].
29. Add a Logger to the end of the flow.



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



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



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.

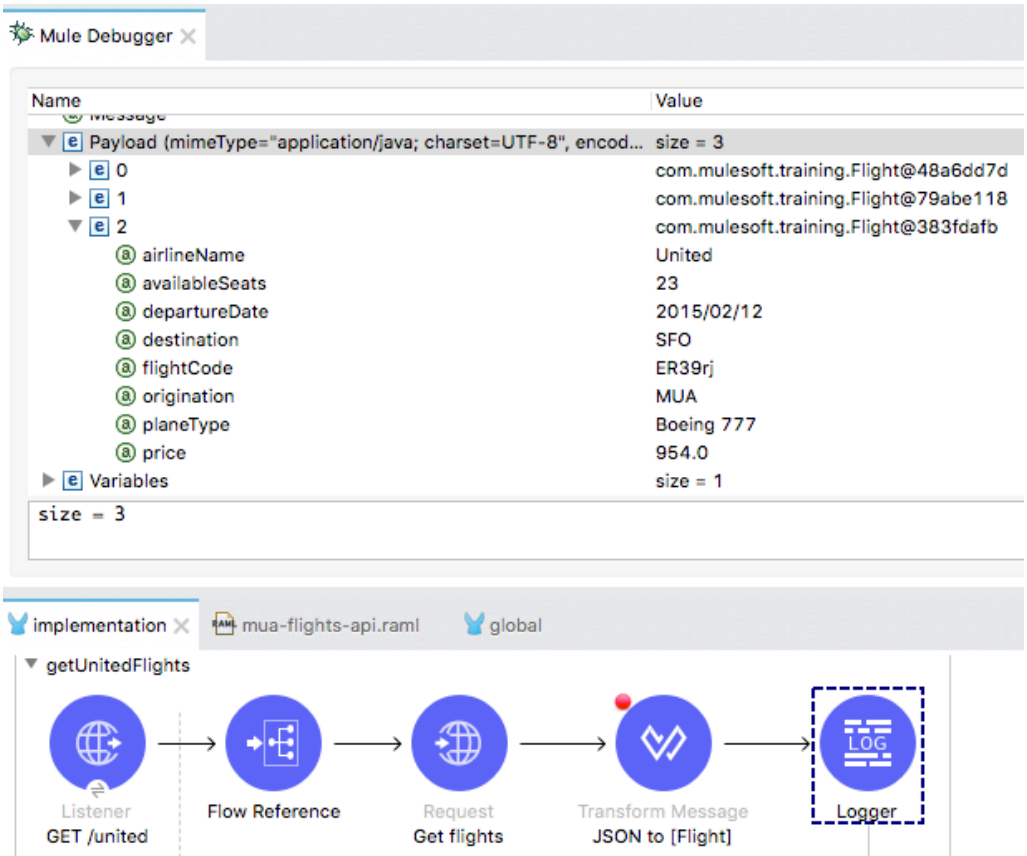
```
%dw 2.0
output application/java
---
payload.flights map ( flight , indexOfFlight ) -> {
    airlineName: flight.airlineName,
    availableSeats: flight.emptySeats,
    departureDate: flight.departureDate,
    destination: flight.destination,
    flightCode: flight.code,
    origination: flight.origin,
    planeType: flight.planeType,
    price: flight.price
} as Object {
    class : "com.mulesoft.training.Flight"
}

```

Debug the application

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

39. In the Mule Debugger, examine the payload.

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

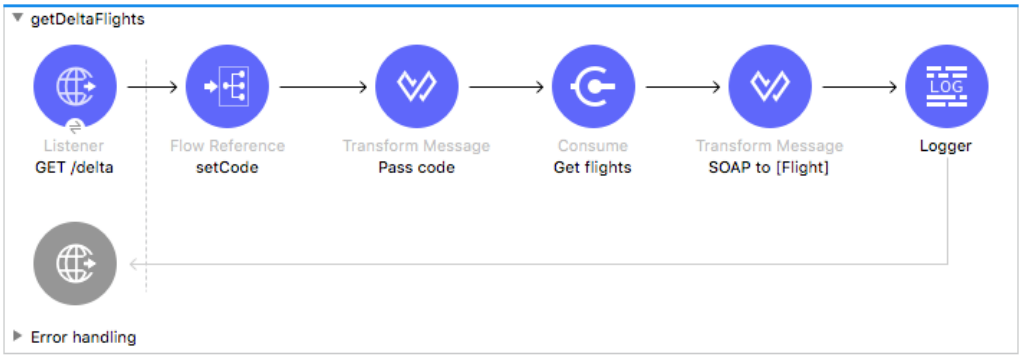


41. Step through the rest of the application and switch perspectives.

Change the Delta flow to return Java Flight objects

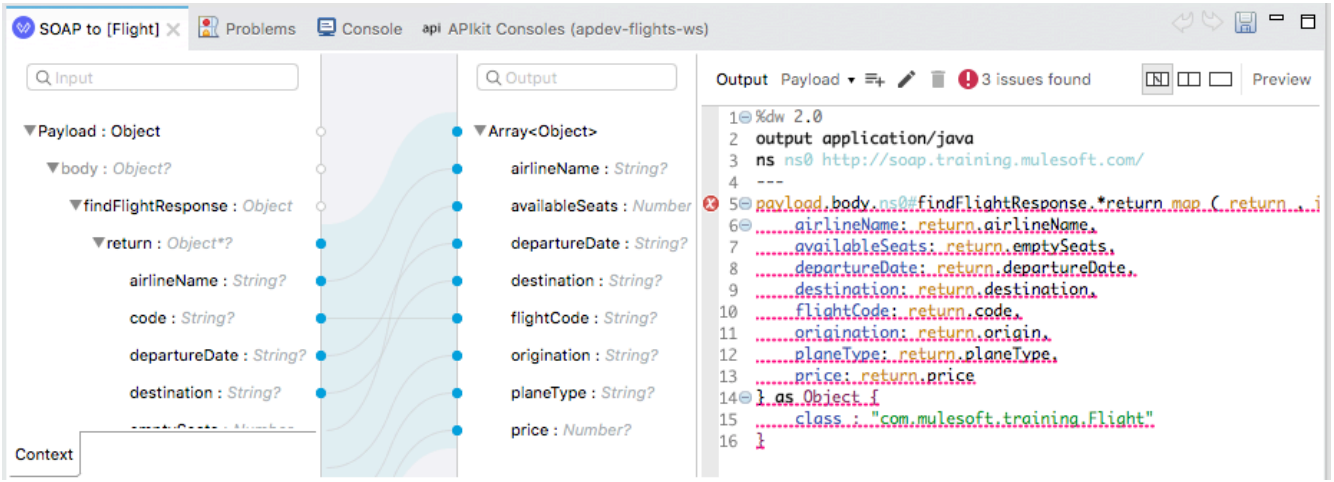
42. Return to geDeltaFlights in implementation.xml.
43. Change the name of the Transform Message component to SOAP to [Flight].

44. Add a Logger to the end of the flow.



45. Look at the input section in the Transform Message properties view; you should see metadata already defined.
46. In the output section of the Transform Message properties view, click the Define metadata link.
47. In the Select metadata type dialog box, select the user-defined Flight_pojo.
48. Select Wrap element in a collection in the lower-left corner.
49. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
50. Map the fields by dragging them from the input section and dropping them on the corresponding field in the output section.

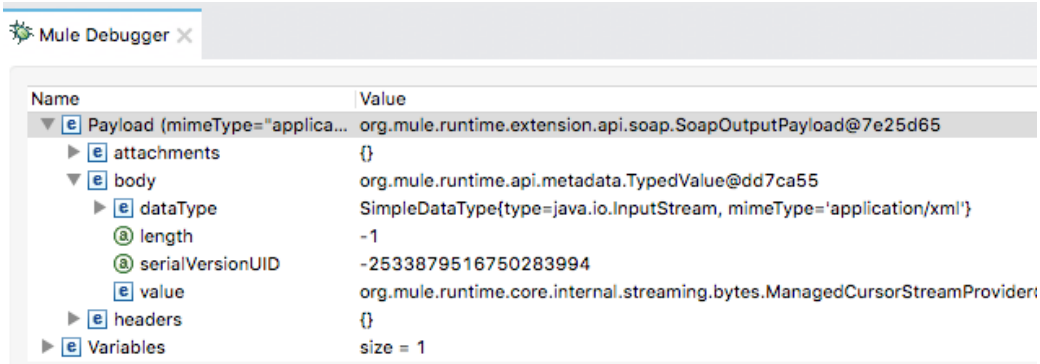
    Note: If you get a DataWeave error and cannot drag and drop the fields, delete all the existing DataWeave code, return to the course snippets.txt file and copy the DataWeave code for the Delta transformation, and then return to Anypoint Studio and paste it in the expression section.



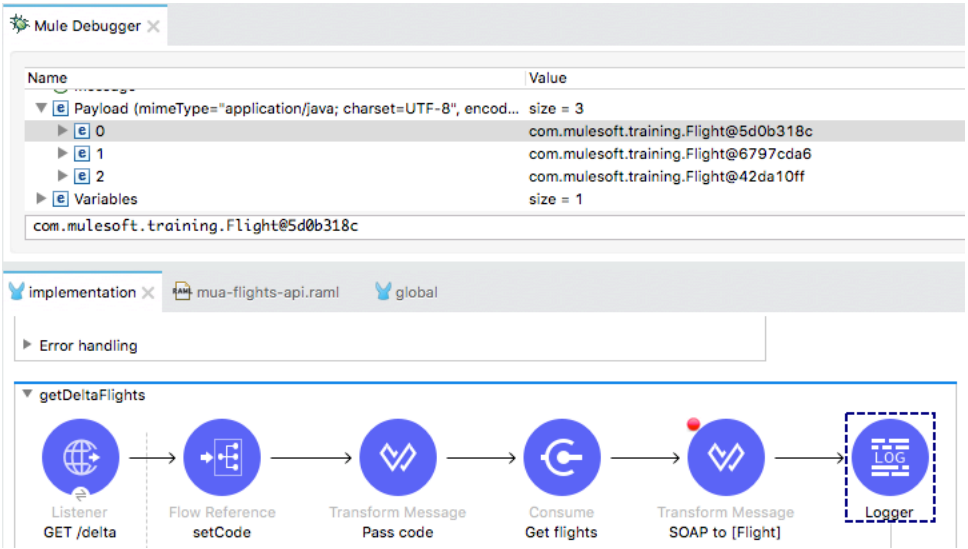
Debug the application

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.

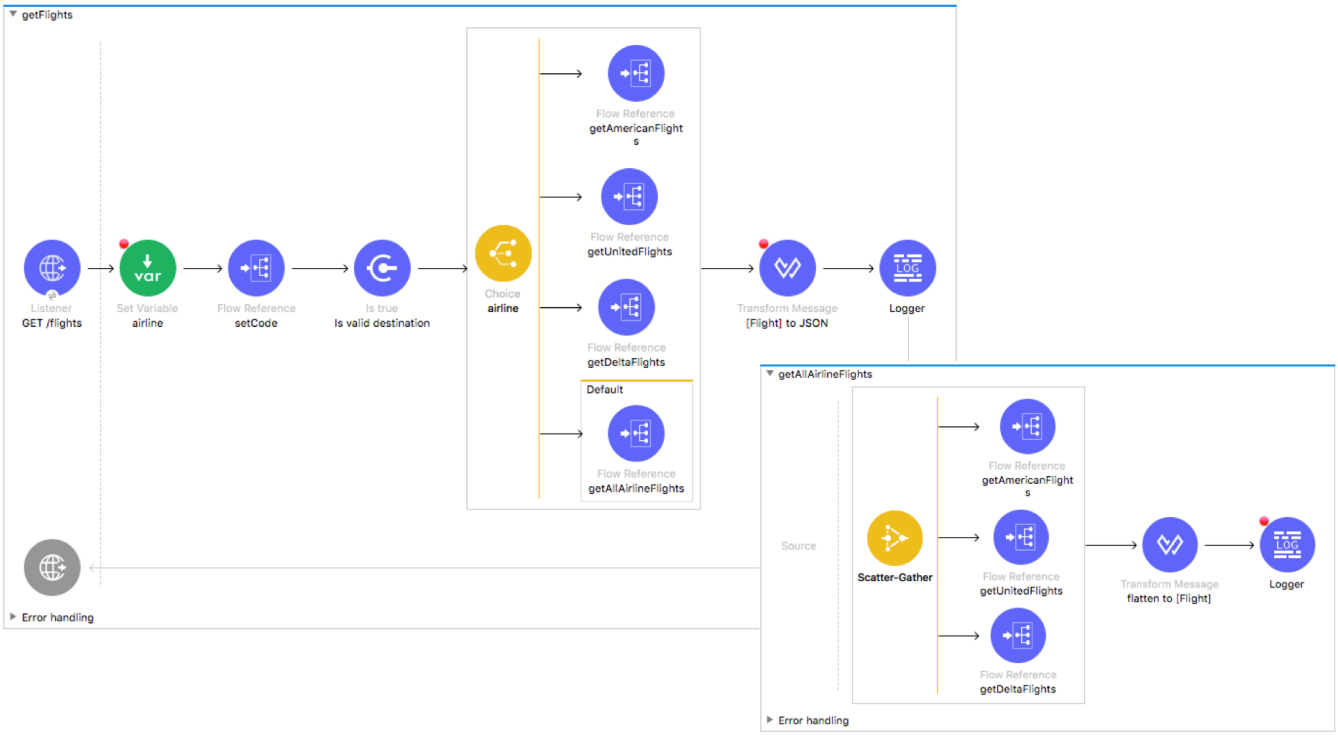


55. Step to the Logger and look at the payload it should be a collection or Flight objects.



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

Module 9: Controlling Event Flow



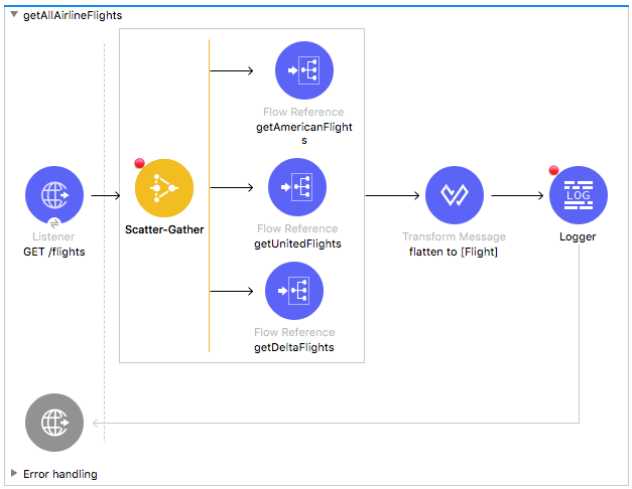
At the end of this module, you should be able to:

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

Walkthrough 9-1: Multicast an event

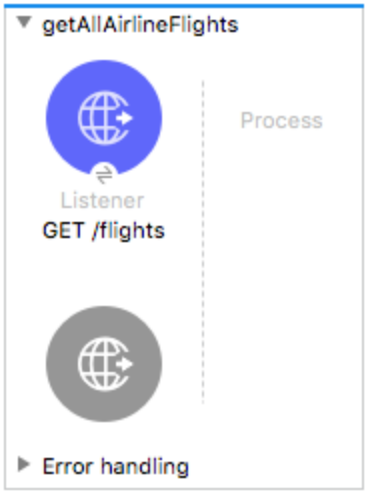
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

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



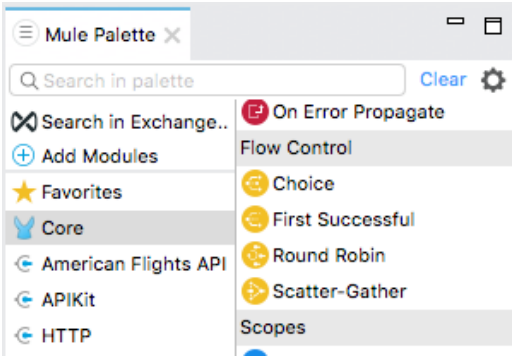
Create a new flow

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.



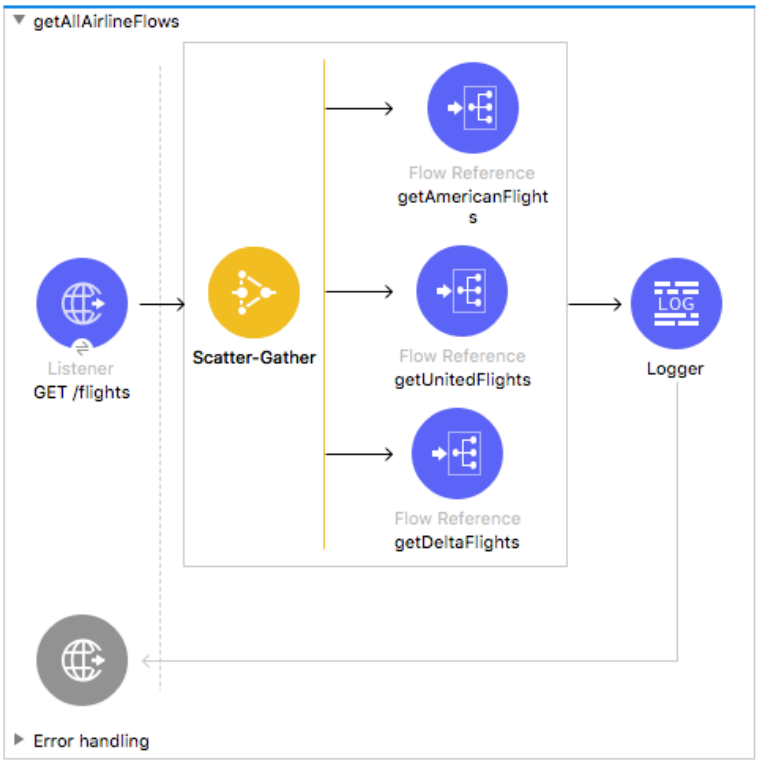
Browse the flow control elements in the Mule Palette

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



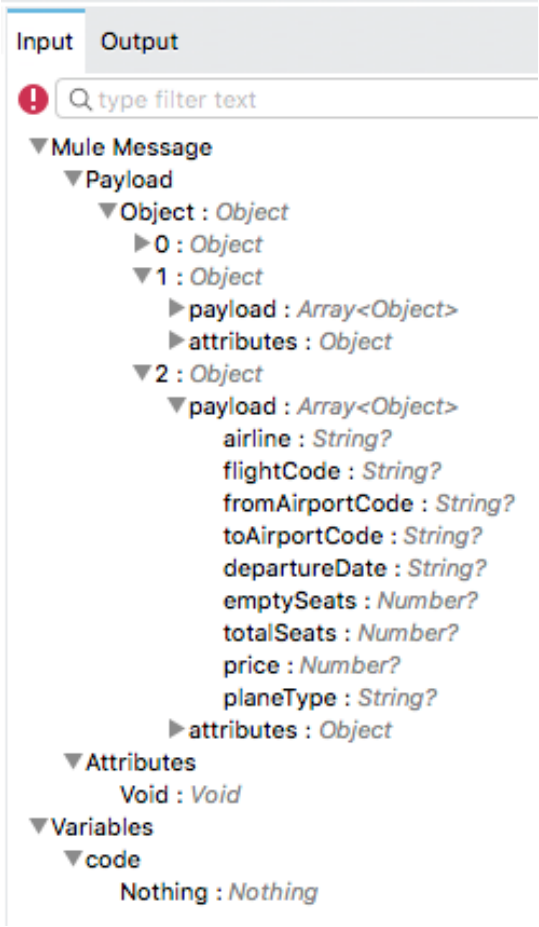
Add a Scatter-Gather to call all three airline services

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

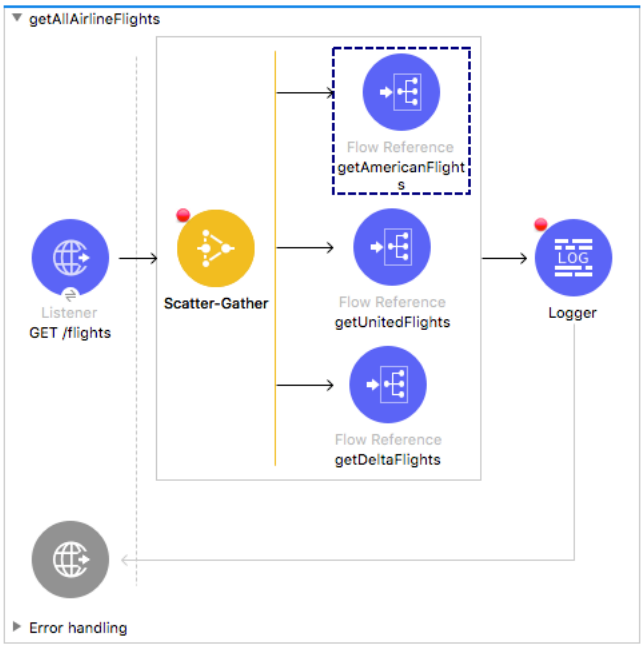
14. In the Logger properties view, explore the input payload structure in the DataSense Explorer.



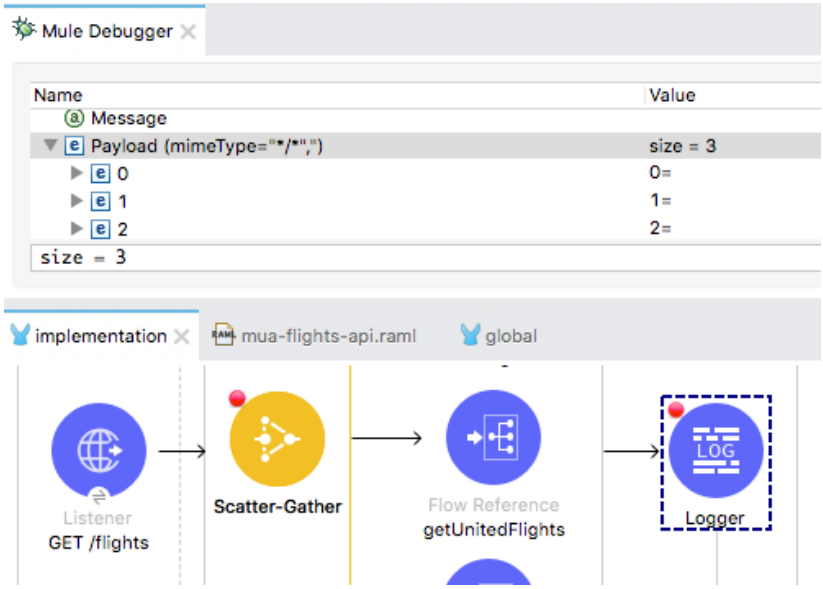
Debug the application

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.

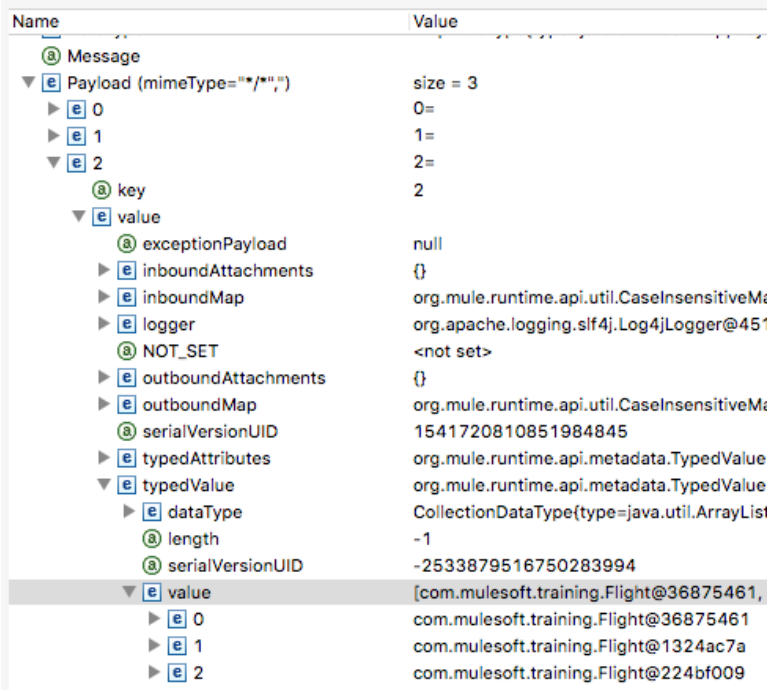
19. In the Mule Debugger, step through the application; you should step through each of the airline flows.



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



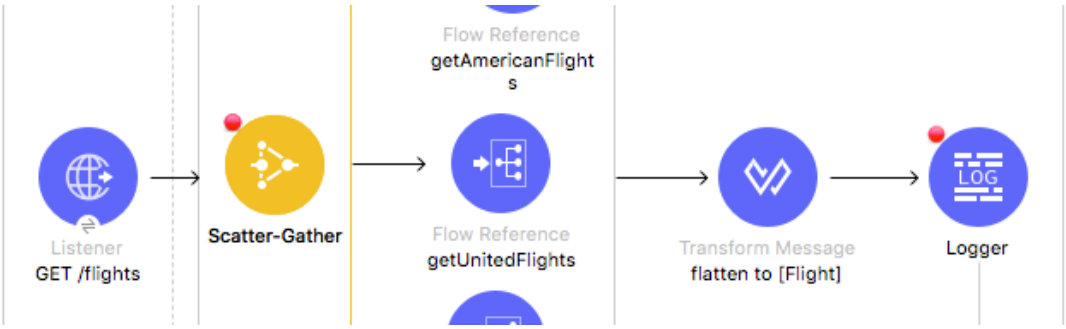
21. Drill-down into one of the objects in the payload.



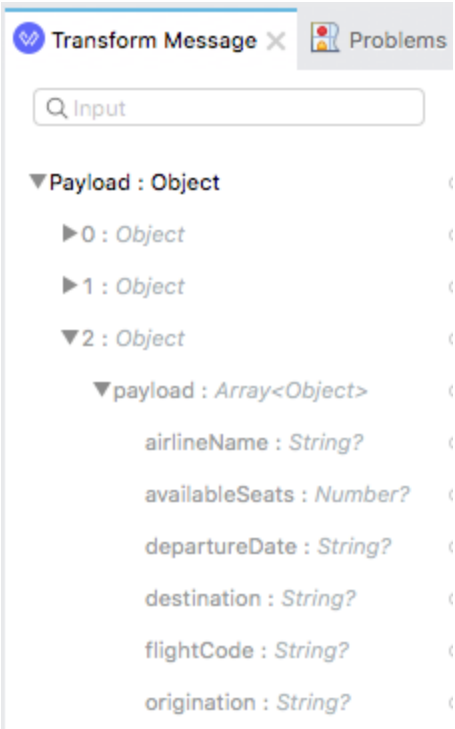
22. Step through the rest of the application and switch perspectives.
23. Return to Advanced REST Client and review the response; although the result is Java and you see a lot of characters, you should also see the Java for flight data for the three airlines.

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.



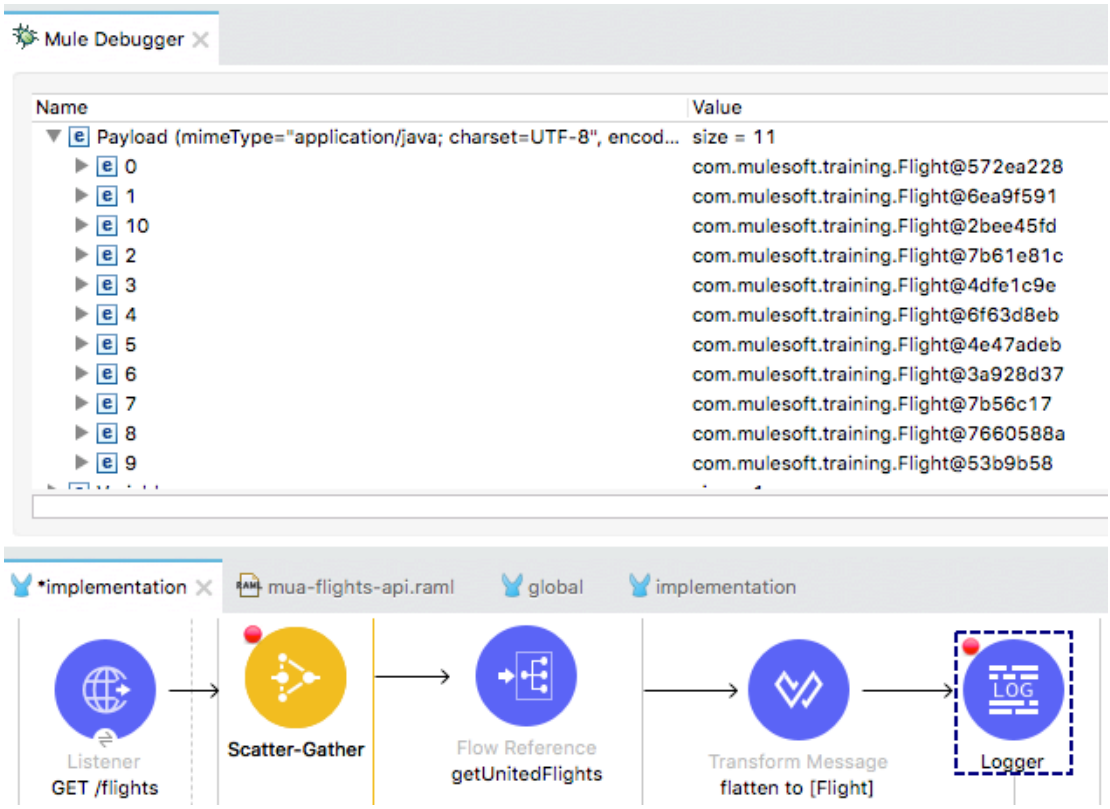
28. In the expression section, use the DataWeave flatten function to flatten the collection of objects into a single collection.

```
%dw 2.0
output application/java
---
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.

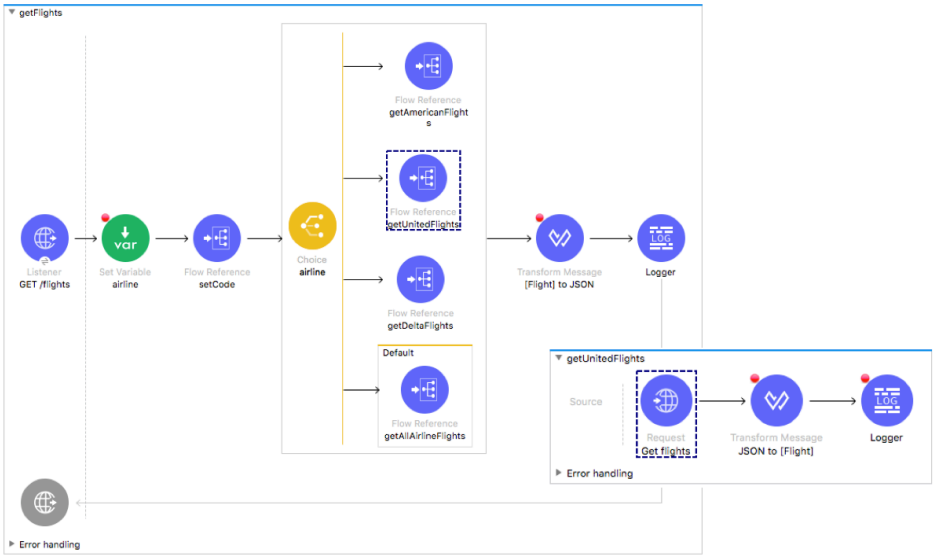


32. Step through the rest of the application and switch perspectives.

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

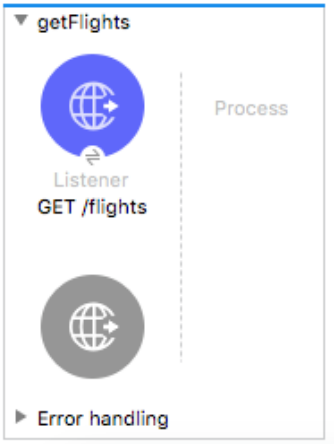
1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Open mua-flights-api.raml in src/main/resources/api.
3. Locate the airline query parameter and its possible values.

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

Create a new flow

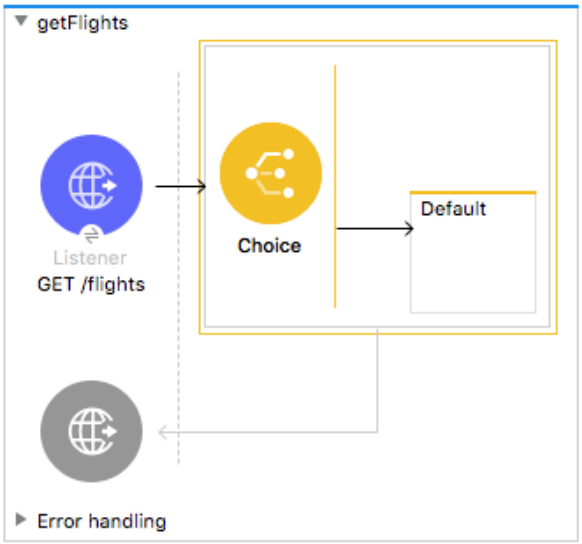
4. Return to implementation.xml.
5. Drag a Flow scope from the Mule Palette and drop it at the top of the canvas above all the other flows.
6. Change the name of the flow to getFlights.

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

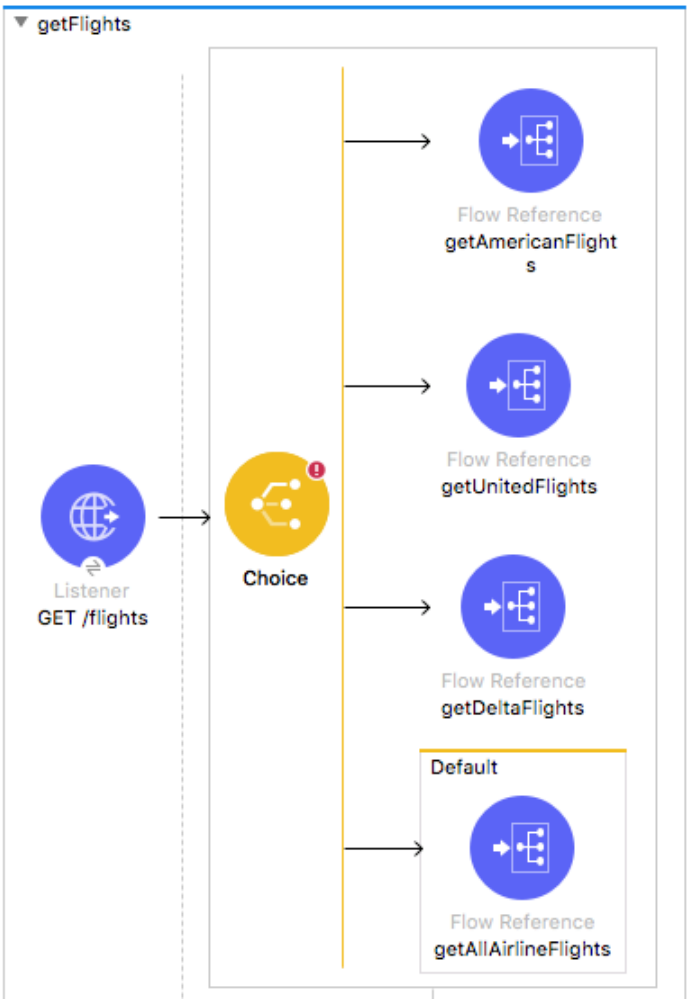


Add a Choice router

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

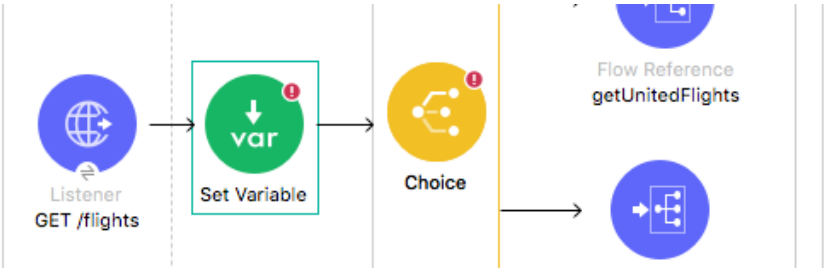


9. Add three Flow Reference components to the Choice router.
10. Add a Flow Reference component to the default branch of the router.
11. In the first Flow Reference properties view, set the flow name to getAmericanFlights.
12. Set the flow names for the other two Flow References to getUnitedFlights and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



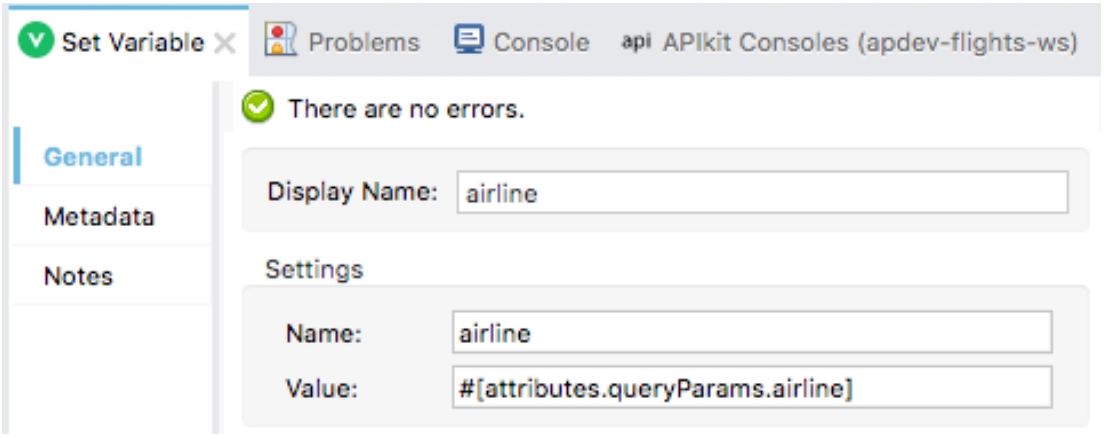
Store the airline query parameter in a variable

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



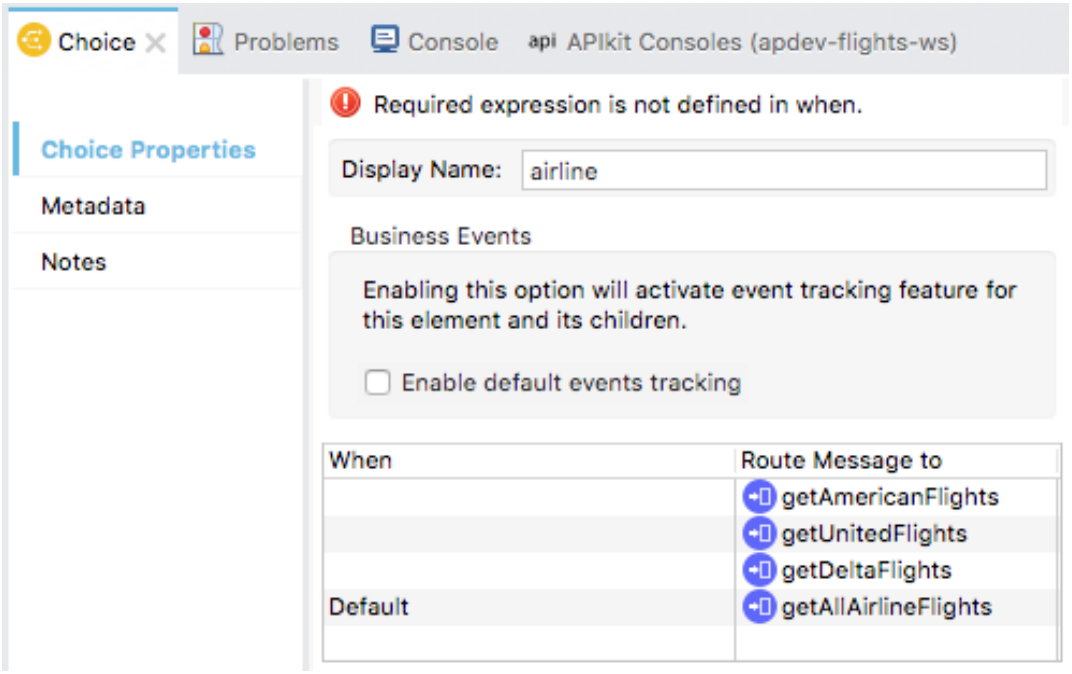
15. In the Set Variable properties view, set the display name and name to airline.
16. Set the value to a query parameter called airline.

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



Configure the Choice router

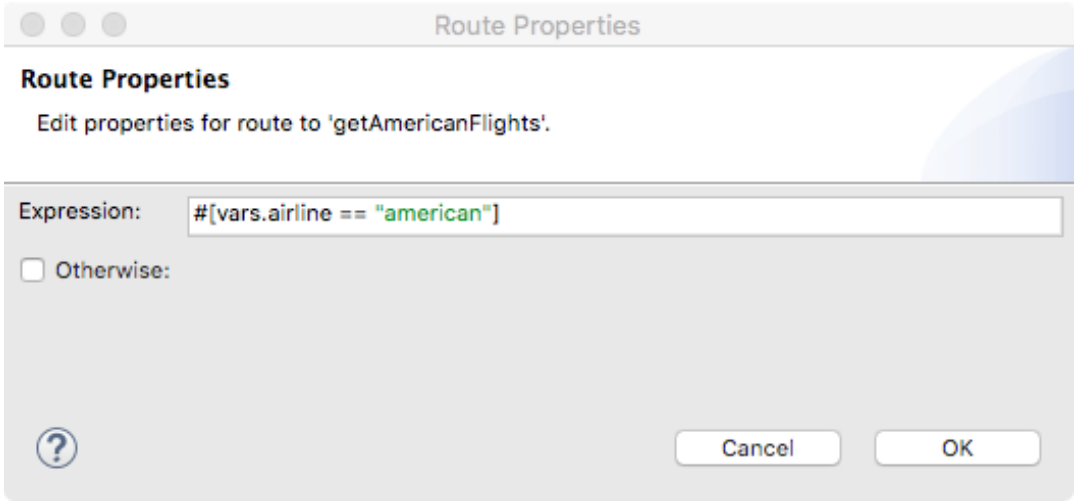
17. In the Choice properties view, set the display name to airline.



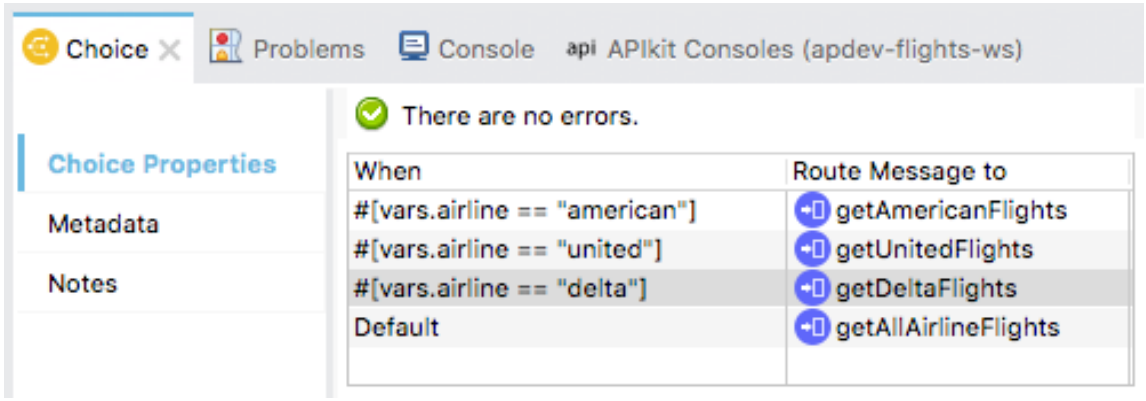
18. In the Choice properties view, double-click the getAmericanFlights route.

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"]
```

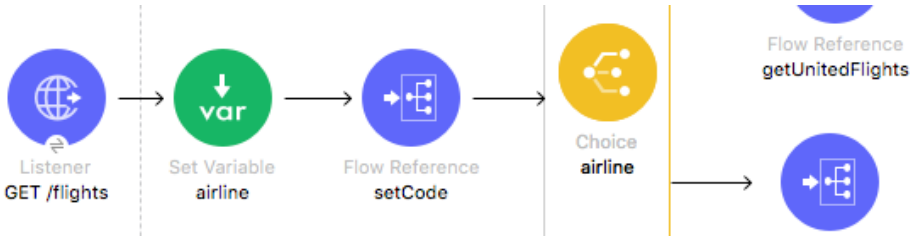


20. Set a similar expression for the United route, routing to it when vars.airline is equal to united.
21. Set a similar expression for the Delta route, routing to it when vars.airline is equal to delta.



Route all requests through the router

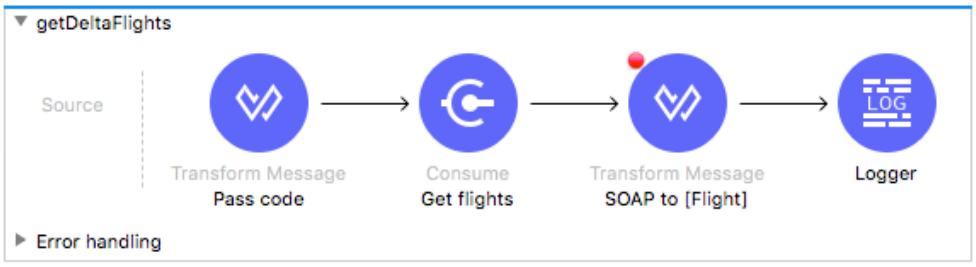
22. Add a Flow Reference to the flow before the Choice router.
23. Set the flow name to setCode.



24. In getAmericanFlights, delete the HTTP Listener and the setCode Flow Reference.

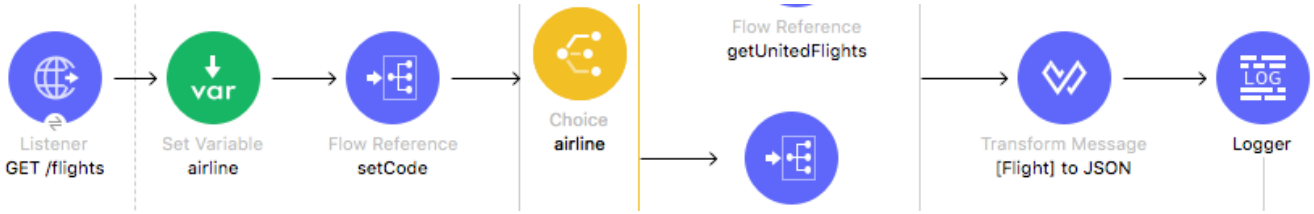
25. In getUnitedFlights, delete the HTTP Listener and the setCode Flow Reference.

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.

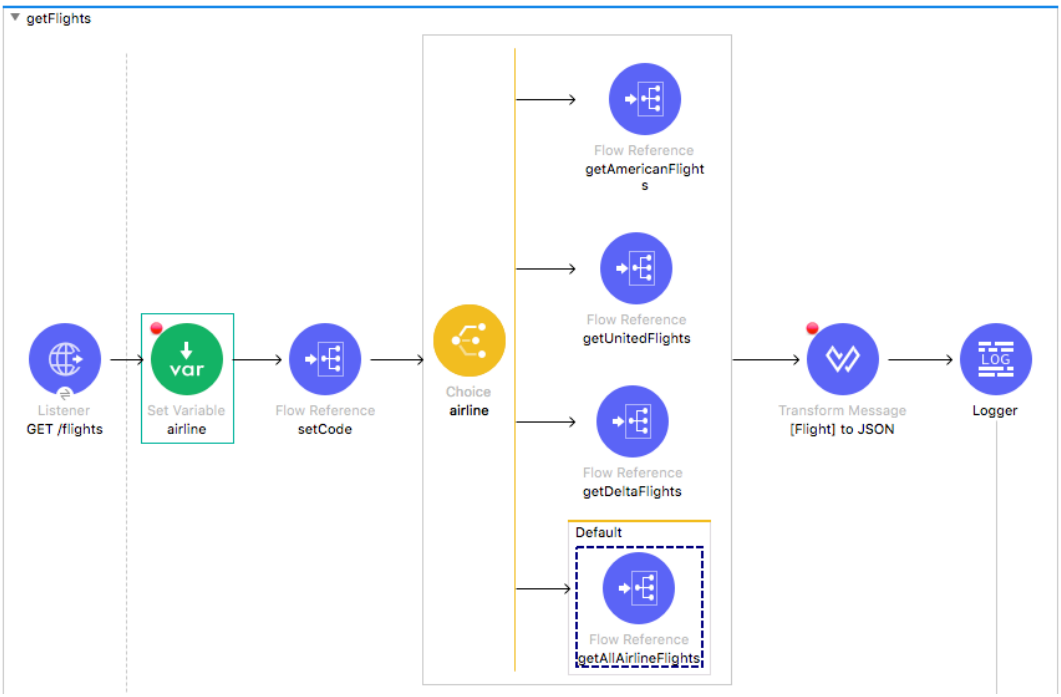


30. In the Transform Message properties view, set the output type to JSON and the output to payload.

```
%dw 2.0
output application/json
---
payload
```

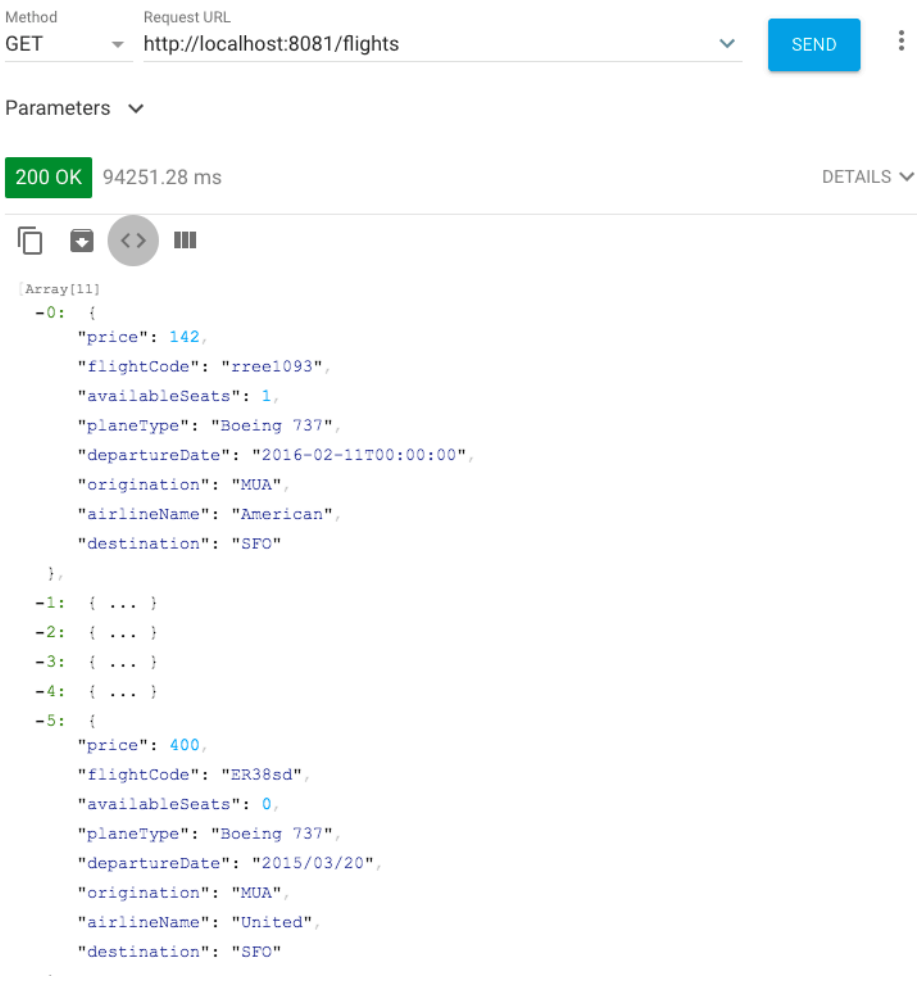
Debug the application

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.

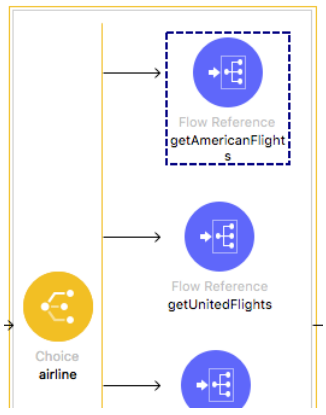


36. Resume to the Transform Message component after the Choice router; the payload should be an ArrayList of Flight objects
37. Step to the Logger; the payload should be JSON.
38. Step to the end of the application.

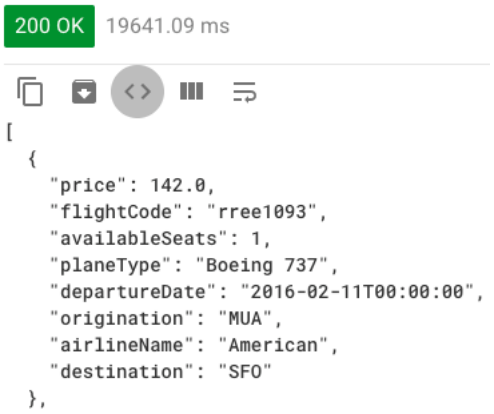
39. Return to Advanced REST Client; you should see American, United, and Delta flights to SFO (the default airport code).



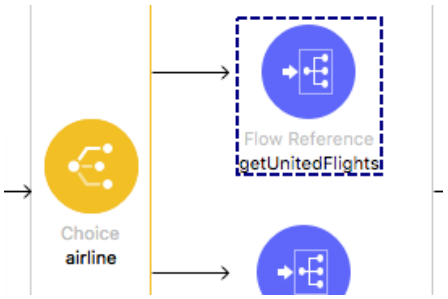
40. Add an airline query parameter set to american and send the request: http://localhost:8081/flights?airline=american.
41. In the Mule Debugger, step through the rest of the application; you should see the message passed to getAmericanFlights and then back to getFlights.



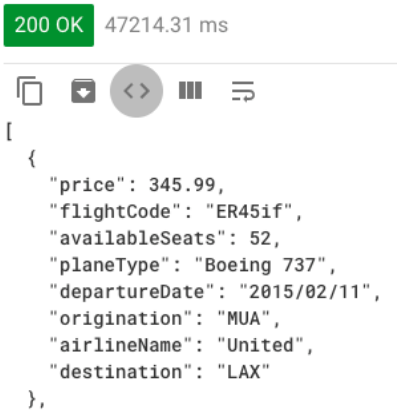
42. Return to Advanced REST Client; you should see only American flights to SFO returned.



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.

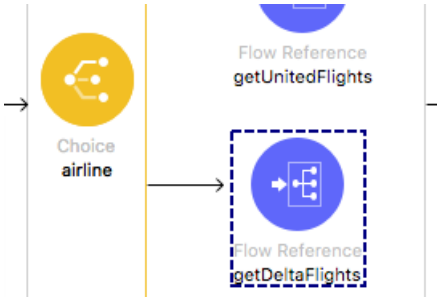


46. Return to Advanced REST Client; you should see only United flights to LAX are returned.

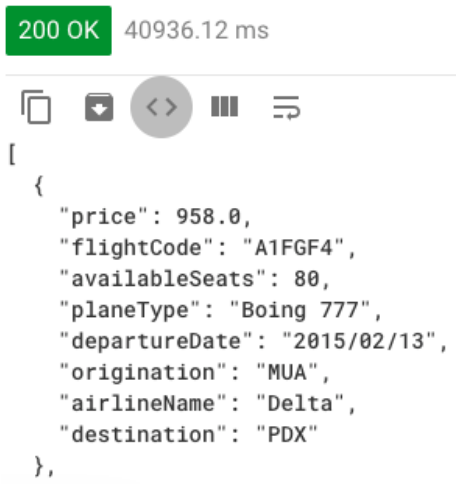


47. Change the airline to delta and the code to PDX: http://localhost:8081/flights?airline=united&code=PDX.
48. Send the request.

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.

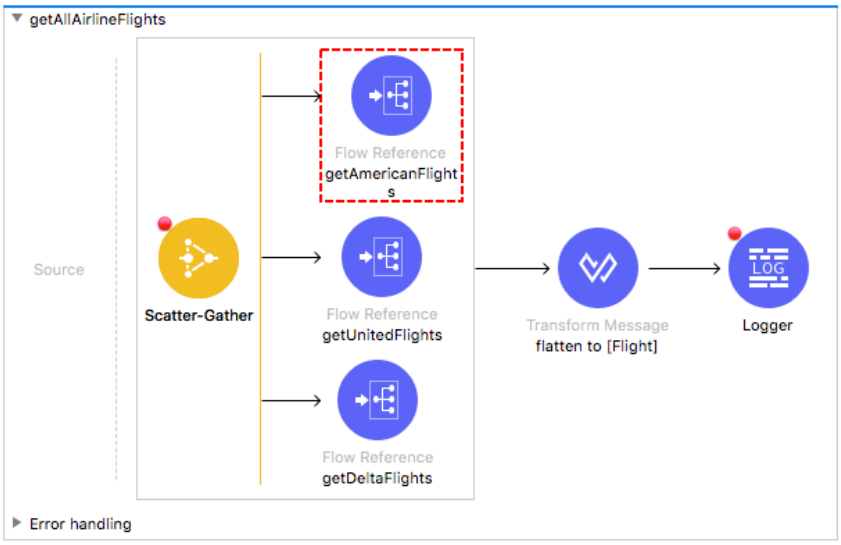


*Note: This JSON structure does not match that specified in the API: mua-flights-api.raml in src/main/resources/api. The data **should**, of course, be transformed so the response from the API matches this specification – but we are going to hold off doing that right now so that the scenario stays simpler 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.