



MuleSoft®

Anypoint Platform Development: Fundamentals

Student Manual

Mule runtime 4.1
September 22, 2018

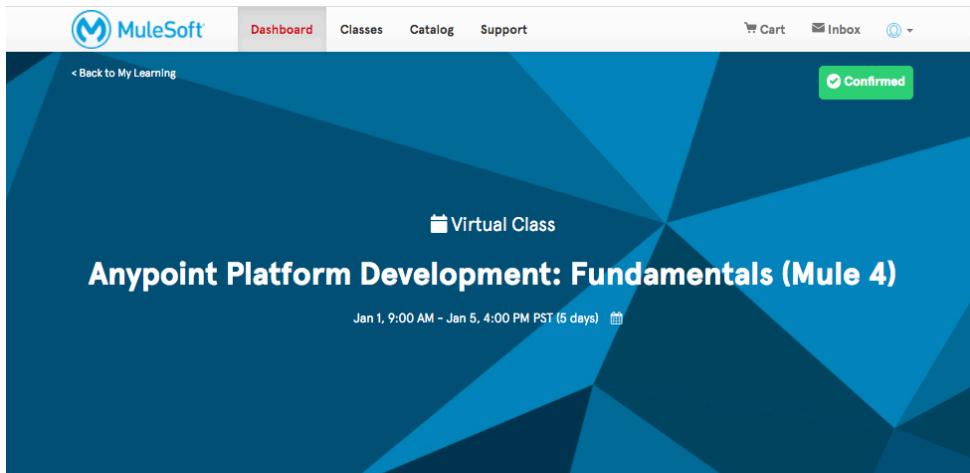
Table of Contents

INTRODUCING THE COURSE	5
Walkthrough: Set up your computer for class	6
PART 1: BUILDING APPLICATION NETWORKS WITH ANYPOINT PLATFORM	12
MODULE 1: INTRODUCING APPLICATION NETWORKS AND API-LED CONNECTIVITY	13
Walkthrough 1-1: Explore an API directory and an API portal	14
Walkthrough 1-2: Make calls to an API	20
MODULE 2: INTRODUCING ANYPOINT PLATFORM	31
Walkthrough 2-1: Explore Anypoint Platform and Anypoint Exchange	32
Walkthrough 2-2: Create a Mule application with flow designer	39
Walkthrough 2-3: Create an integration application with flow designer that consumes an API	48
MODULE 3: DESIGNING APIS	61
Walkthrough 3-1: Use API designer to define an API with RAML.....	62
Walkthrough 3-2: Use the mocking service to test an API	68
Walkthrough 3-3: Add request and response details	72
Walkthrough 3-4: Add an API to Anypoint Exchange.....	85
Walkthrough 3-5: Share an API	97
MODULE 4: BUILDING APIS	105
Walkthrough 4-1: Create a Mule application with Anypoint Studio	106
Walkthrough 4-2: Connect to data (MySQL database)	112
Walkthrough 4-3: Transform data	124
Walkthrough 4-4: Create a RESTful interface for a Mule application.....	134
Walkthrough 4-5: Use Anypoint Studio to create a RESTful API interface from a RAML file	142
Walkthrough 4-6: Implement a RESTful web service.....	151
MODULE 5: DEPLOYING AND MANAGING APIS.....	155
Walkthrough 5-1: Deploy an application to CloudHub	156
Walkthrough 5-2: Create and deploy an API proxy	163
Walkthrough 5-3: Restrict API access with policies and SLAs.....	175
Walkthrough 5-4: Request and grant access to a managed API	183
Walkthrough 5-5: Add client ID enforcement to an API specification.....	191

PART 2: BUILDING APPLICATIONS WITH ANYPOINT STUDIO	199
MODULE 6: ACCESSING AND MODIFYING MULE EVENTS.....	201
Walkthrough 6-1: View event data	202
Walkthrough 6-2: Debug a Mule application	209
Walkthrough 6-3: Track event data as it moves in and out of a Mule application	217
Walkthrough 6-4: Set request and response data.....	224
Walkthrough 6-5: Get and set event data using DataWeave expressions.....	228
Walkthrough 6-6: Set and get variables	236
MODULE 7: STRUCTURING MULE APPLICATIONS.....	240
Walkthrough 7-1: Create and reference subflows and private flows.....	241
Walkthrough 7-2: Pass messages between flows using the VM connector.....	245
Walkthrough 7-3: Encapsulate global elements in a separate configuration file.....	254
Walkthrough 7-4: Use property placeholders in connectors	263
Walkthrough 7-5: Create a well-organized Mule project	267
Walkthrough 7-6: Manage metadata for a project.....	277
MODULE 8: CONSUMING WEB SERVICES	284
Walkthrough 8-1: Consume a RESTful web service that has a connector in Exchange.....	285
Walkthrough 8-2: Consume a RESTful web service	296
Walkthrough 8-3: Consume a SOAP web service	305
Walkthrough 8-4: Transform data from multiple services to a canonical format	316
MODULE 9: CONTROLLING EVENT FLOW	325
Walkthrough 9-1: Multicast an event.....	326
Walkthrough 9-2: Route events based on conditions.....	333
Walkthrough 9-3: Validate events	345
MODULE 10: HANDLING ERRORS.....	351
Walkthrough 10-1: Explore default error handling.....	352
Walkthrough 10-2: Handle errors at the application level	360
Walkthrough 10-3: Handle specific types of errors	370
Walkthrough 10-4: Handle errors at the flow level	375
Walkthrough 10-5: Handle errors at the processor level.....	384
Walkthrough 10-6: Map an error to a custom error type	391
Walkthrough 10-7: Review and integrate with APIkit error handlers.....	397
Walkthrough 10-8: Set a reconnection strategy for a connector	409

MODULE 11: WRITING DATAWEAVE TRANSFORMATIONS.....	410
Walkthrough 11-1: Create transformations with the Transform Message component	411
Walkthrough 11-2: Transform basic JSON, Java, and XML data structures.....	422
Walkthrough 11-3: Transform complex data structures with arrays.....	427
Walkthrough 11-4: Transform to and from XML with repeated elements.....	436
Walkthrough 11-5: Define and use variables and functions.....	443
Walkthrough 11-6: Coerce and format strings, numbers, and dates.....	449
Walkthrough 11-7: Define and use custom data types	456
Walkthrough 11-8: Use DataWeave functions	460
Walkthrough 11-9: Look up data by calling a flow.....	465
PART 3: BUILDING APPLICATIONS TO SYNCHRONIZE DATA	469
MODULE 12: TRIGGERING FLOWS	471
Walkthrough 12-1: Trigger a flow when a new file is added to a directory.....	472
Walkthrough 12-2: Trigger a flow when a new record is added to a database and use automatic watermarking.....	480
Walkthrough 12-3: Schedule a flow and use manual watermarking	491
Walkthrough 12-4: Publish and listen for JMS messages.....	502
MODULE 13: PROCESSING RECORDS	508
Walkthrough 13-1: Process items in a collection using the For Each scope.....	509
Walkthrough 13-2: Process records using the Batch Job scope.....	515
Walkthrough 13-3: Use filtering and aggregation in a batch step	522

Introducing the Course



This instructor-led course is for developers and architects who want to get hands-on experience using Anypoint Platform to build APIs and integrations. In the first part, students use Anypoint Platform discover, consume, design, build, deploy, manage, and govern APIs. In the second part, students focus on using Mule and Anypoint Studio to build applications for use as API implementations and/or integrations.

The course includes a voucher to take a new MuleSoft Certified Developer exam for Mule 4 to be released later this year.

A downloadable data sheet for the course can be found [here](#).

INSTRUCTORS

MATERIALS

- APDevFundamentals4.1
Student Slides (ZIP)
216B ZIP
- APDevFundamentals4.1
Student Manual (PDF)
81.3KB PDF
- APDevFundamentals4.1
Student Files (ZIP)
214B ZIP

Objectives:

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

Walkthrough: Set up your computer for class

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

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

Download student files

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

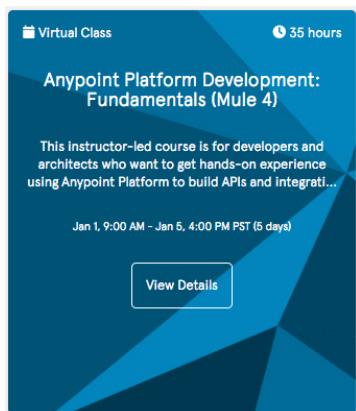
The screenshot shows the MuleSoft website's navigation bar with links for Products, Solutions, Industries, Services, Resources, Partners, Company, Try now, and Developers. Below the navigation, a breadcrumb trail reads "MuleSoft Training & Certification > Training > Home". On the right side, there is a "My training account" section with a "Register for upcoming classes" button. The main content area features a dark background with the text "Training & Certification" and a "Register for upcoming classes" button.

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

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

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

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



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

This instructor-led course is for developers and architects who want to get hands-on experience using Anypoint Platform to build APIs and integrations. In the first part, students use Anypoint Platform discover, consume, design, build, deploy, manage, and govern APIs. In the second part, students focus on using Mule and Anypoint Studio to build applications for use as API implementations and/or integrations.

The course includes a voucher to take a new MuleSoft Certified Developer exam for Mule 4 to be released later this year.

A downloadable data sheet for the course can be found [here](#).

INSTRUCTORS

MATERIALS

- APDevFundamentals4.1 Student Slides (ZIP) 216B ZIP
- APDevFundamentals4.1 Student Manual (PDF) 81.3KB PDF
- APDevFundamentals4.1 Student Files (ZIP) 214B ZIP

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

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

Make sure you have JDK 1.8

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

```
java -version
```

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

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

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

Note: JDK 1.9 is NOT supported.

Make sure you have Java in your PATH environment variable

15. In a command-line interface, type \$PATH (Mac) or %PATH% (Windows) and press enter.

- Mac: \$PATH
- Windows: %PATH%

16. After installing the correct JDK version, add or update an environment variable named JAVA_HOME that points to the installation location and then add JAVA_HOME/bin to your PATH environment variable.

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

Start Anypoint Studio

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

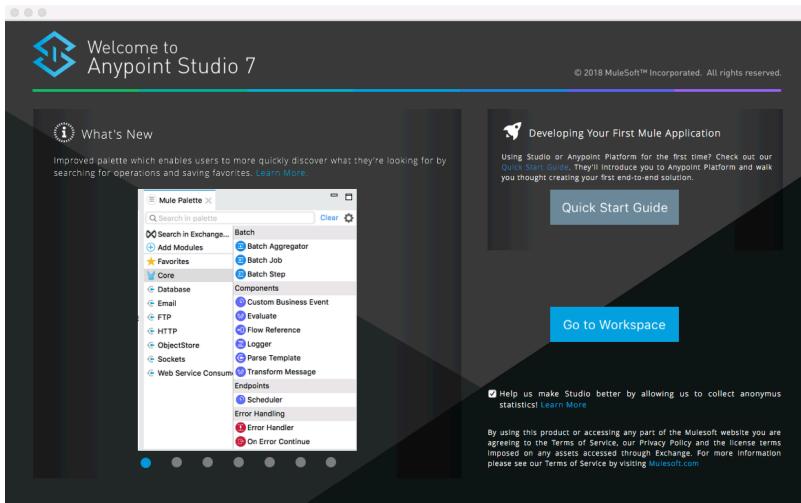
Note: If you do not have Anypoint Studio, you can download it from <https://www.mulesoft.com/lp/dl/studio>.

18. In the Workspace Launcher dialog box, look at the location of the default workspace; change the workspace location if you want.

19. Click OK to select the workspace; Anypoint Studio should open.

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

20. If you get a Welcome Page, click the X on the tab to close it.



21. If you get an Updates Available pop-up in the lower-right corner of the application, click it and install the available updates.

Open Advanced REST Client

22. Open Advanced REST Client.

Note: If you do not have Advanced REST Client (or another REST API client) installed, download it now from <https://install.advancedrestclient.com/> and install it.

23. Leave Advanced REST client open; you will use it throughout class.

A screenshot of the Advanced REST Client software. The main window has a blue header bar with the title "Advanced REST client". Below the header is a toolbar with a "Request" button, a "New request" button, a "+" button, and a "SEND" button. The main area is divided into sections: "Method" (set to GET), "Request URL", "Parameters", "Headers", "Authorization", "Variables", and "Actions". Under "Headers", there is a table with a single row: "Header name" (empty) and "Header value" (empty). A red "ADD HEADER" button is below the table. In the bottom right corner, there is a message "Headers size: bytes". At the very bottom of the window, there is a status bar with the text "Selected environment: Default" and an information icon.

Make sure you have an active Anypoint Platform account

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

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

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



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

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

A screenshot of the Anypoint Platform main menu. The left sidebar shows several options: Design Center, Exchange, Management Center, Access Management (which is highlighted with a blue background), API Manager, Runtime Manager, and Data Gateway. The main content area displays three main services: Design Center, Exchange, and Management Center, each with a brief description and a "Design", "Discover & share", and "Manage" button respectively.

27. In the left-side navigation, click the Runtime Manager link under Subscription.

28. Check your subscription level and if it is a trial account, make sure it is not expired.

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

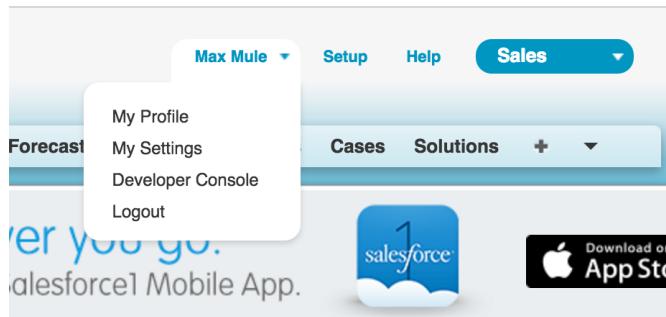
A screenshot of the Access Management page. The left sidebar has sections for ACCESS MANAGEMENT (Organization, Users, Roles, Environments, External Identity, Audit Logs) and SETTINGS (Runtime Manager). The main content area shows "Subscription Information" with details: Organization Name: Training, Subscription Tier: Trial, and Expiration Date: Expires on 12/12/2017. Below this, there are sections for "Production" (Environments for production applications) and "Sandboxes" (Test environments for QA of applications). At the bottom, there are two progress bars: "vCores" (0 / 0) and "Static IP" (0 / 1). The "Runtime Manager" link in the sidebar is underlined, indicating it is selected.

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

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

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

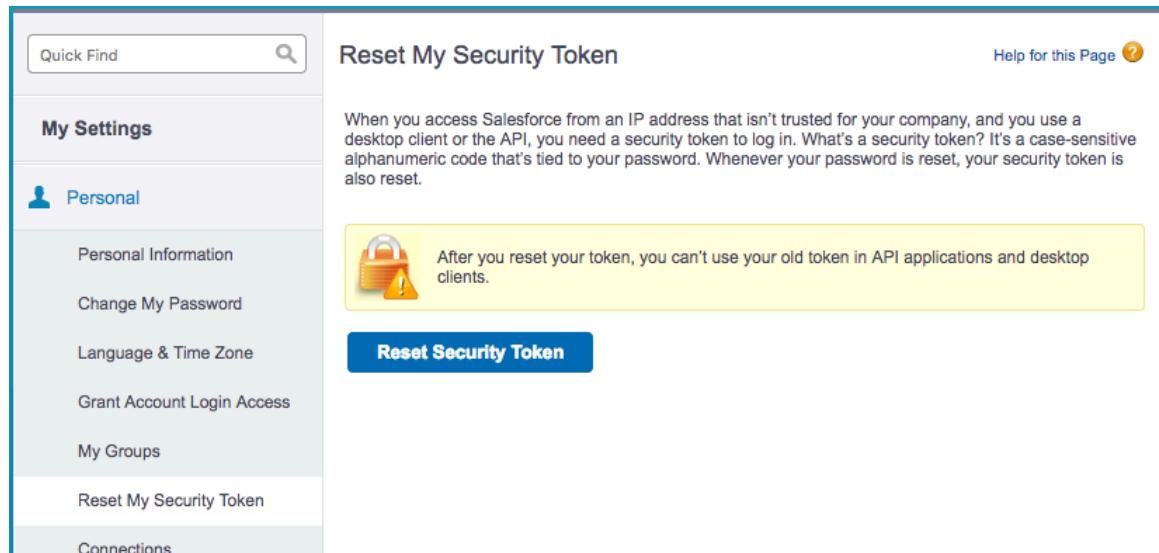
30. In Salesforce, click your name at the top of the screen and select My Settings.



31. In the left-side navigation, select Personal > Reset My Security Token.

32. If you did not already request a security token, click the Reset Security Token button.

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



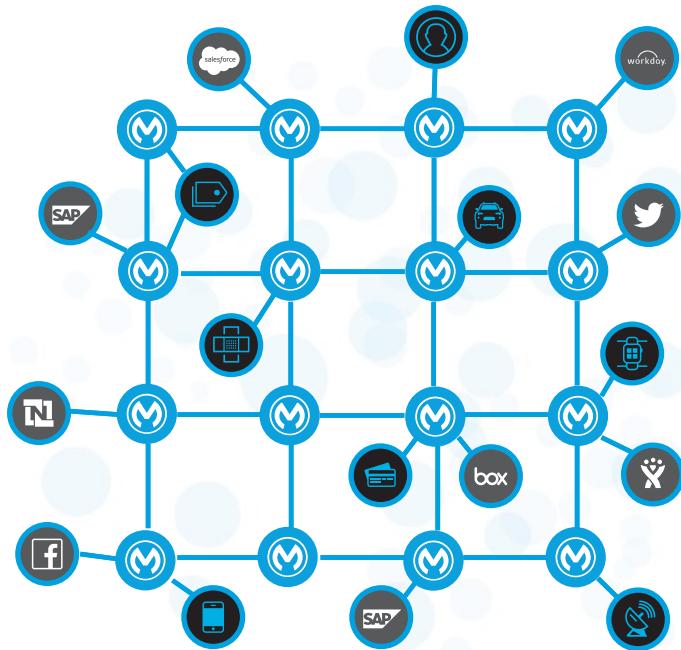
Reset My Security Token

When you access Salesforce from an IP address that isn't trusted for your company, and you use a desktop client or the API, you need a security token to log in. What's a security token? It's a case-sensitive alphanumeric code that's tied to your password. Whenever your password is reset, your security token is also reset.

After you reset your token, you can't use your old token in API applications and desktop clients.

Reset Security Token

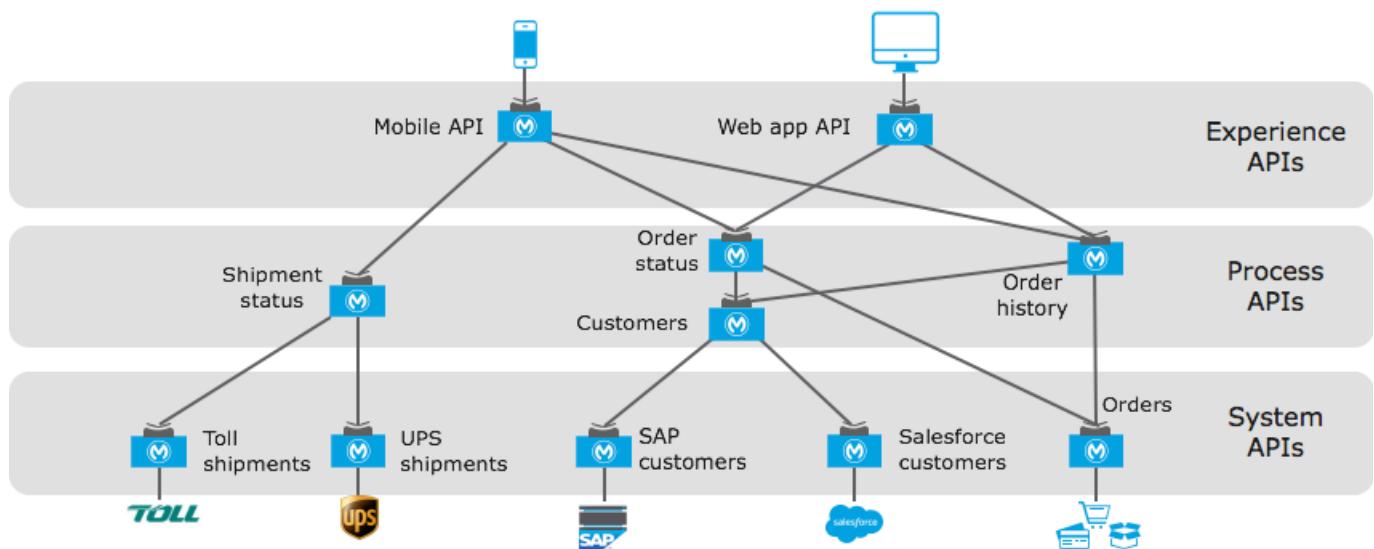
PART 1: Building Application Networks with Anypoint Platform



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

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

Module 1: Introducing Application Networks and API-Led Connectivity



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

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

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

In this walkthrough, you locate and explore documentation about APIs. You will:

- Browse the ProgrammableWeb API directory.
- Explore the API reference for an API (like Twitter).
- Explore the API portal for an API to be used in the course.

The left screenshot shows the ProgrammableWeb API directory homepage. It features a search bar with "Search Over 19,516 APIs" and a "SEARCH APIs" button. Below the search bar is a "Filter APIs" section with a dropdown menu set to "By Category". A table lists two APIs: Google Maps and Twitter. Google Maps is described as "[This API is no longer available. Google Maps' services have been split into multiple APIs, including the Static Maps API]". Twitter is described as "[This API is no longer available. It has been split into multiple APIs, including the Twitter Ads API, Twitter Search Tweets...]".

The right screenshot shows the MuleSoft // Training portal for the American Flights API (v2). It includes an "Assets list" sidebar with options like "API summary", "Types", "Resources", and "/flights". Under "/flights", there are four methods: GET (selected), POST, PUT, and DELETE. The "Request" section shows a GET request to "/flights/{ID}". The "Parameters" section shows a parameter "ID" with a type of "string" and a note "(required)".

Browse the ProgrammableWeb API directory

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

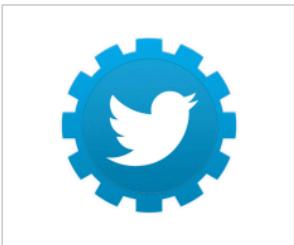
The screenshot shows the ProgrammableWeb API directory homepage. At the top, there's a navigation bar with links for "API NEWS" (dropdown), "API DIRECTORY" (dropdown), "BECOME MEMBER", and "LOGIN". Below the navigation bar is a search bar with the placeholder "Search over 15,047 APIs and much more" and a magnifying glass icon. The main content area features a large image of a smartphone displaying a map with a location pin labeled "SDK". To the right of the image is an advertisement for MuleSoft: "Build a better API strategy A 7 step blueprint for success". Below the advertisement are social media sharing buttons for RSS, Facebook, Twitter, Google+, and LinkedIn.

3. Browse the list of popular APIs.

API Name	Description	Category	Submitted
Google Maps	[This API is no longer available. Google Maps' services have been split into multiple APIs, including the Static Maps API ,	Mapping	12.05.2005
Twitter	[This API is no longer available. It has been split into multiple APIs, including the Twitter Ads API , Twitter Search Tweets... .	Social	12.08.2006
YouTube	The Data API allows users to integrate their program with YouTube and allow it to perform many of the operations available on the website. It provides the capability to search for videos, retrieve...	Video	02.08.2006
Flickr	The Flickr API can be used to retrieve photos from the Flickr photo sharing service using a variety of feeds - public photos and videos, favorites, friends, group pools, discussions, and more. The...	Photos	09.04.2005
Facebook	[This API is no longer available. Its functions have been split	Social	08.16.2006

Explore the API reference for the Twitter API

- Click the link for the Twitter API (or some other) API.
- In the Specs section, click the API Portal / Home Page link.



Twitter API

Social Blogging

[This API is no longer available. It has been split into multiple APIs, including the [Twitter Ads API](#), [Twitter Search Tweets API](#), and [Twitter Direct Message API](#).
This profile is maintained for historical, research, and reference purposes only.]

The Twitter micro-blogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with Twitter Search and trends data. The API presently supports the following data formats: XML, JSON, and the RSS and Atom syndication formats, with some methods only accepting a subset of these formats.

Summary	SDKs (72)	Articles (267)	How To (11)	Sample Source Code (25)	Libraries (38)	Developers (820)	Followers (1919)	Comments (4)
---------	-----------	----------------	-------------	-------------------------	----------------	------------------	------------------	--------------

SPECS	
API Endpoint	http://twitter.com/statuses/
API Portal / Home Page	https://dev.twitter.com/rest/public
Primary Category	Social

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

The screenshot shows the Twitter Developer API documentation. At the top, there's a purple header bar with links for 'Developer', 'Use cases', 'Products', 'Docs', 'More', 'Apply', a search icon, and 'Sign In'. Below the header, a search bar says 'Search all documentation...'. The main title is 'Post, retrieve and engage with Tweets' in large bold letters. Underneath it, there's a 'Basics' section, followed by 'Accounts and users' and 'Tweets'. The 'Tweets' section has a sub-section 'Post, retrieve and engage with Tweets' which lists various API endpoints like 'Get Tweet timelines', 'Curate a collection of Tweets', 'Optimize Tweets with Cards', 'Search Tweets', 'Filter realtime Tweets', 'Sample realtime Tweets', and 'Get batch historical Tweets'. To the right of these, there are three columns: 'Tweets' (listing POST statuses/update, POST statuses/destroy/:id, GET statuses/show/:id), 'Retweets' (listing POST statuses/retweet/:id, POST statuses/unretweet/:id, GET statuses/retweets/:id, GET statuses/retweets/_of_me), and 'Likes (formerly favorites)' (listing POST favorites/create/:id, POST favorites/destroy/:id, GET favorites/list). Below the table, a note says 'The following API endpoints can be used to programmatically create, retrieve and delete Tweets, Retweets and Likes:'.

- Browse the list of requests you can make to the API.
- Select the API Reference tab beneath the title of the page.
- Review the information for POST statuses/update including parameters, example request, and example response.

The screenshot shows the 'POST statuses/update' endpoint details. At the top, there's a purple header bar with links for 'Developer', 'Use cases', 'Products', 'Docs', 'More', 'Apply', a search icon, and 'Sign In'. Below the header, there's a table for the 'fail_dm_commands' parameter. It shows 'fail_dm_commands' as optional, with a description: 'When set to true, causes any status text that starts with shortcode commands to return an API error. When set to false, allows shortcode commands to be sent in the status text and acted on by the API.' The value is listed as 'true' and 'false'.

Example Request

```
POST https://api.twitter.com/1.1/statuses/update.json?
status=Maybe%20he%27ll%20finally%20find%20his%20keys.%20%23peterfall
```

Example Response

```
{
  "coordinates": null,
  "favorited": false,
  "created_at": "Wed Sep 05 00:37:15 +0000 2012",
  "truncated": false,
  "id_str": "243145735212777472",
  "entities": {
    "urls": [
      ...
    ]
  }
}
```

- Close the browser tab.

Explore an API portal for an API to be used in the course

12. Return to the course snippets.txt file.
13. Copy the URL for the MuleSoft Training API portal.
14. Return to a browser window and navigate to that URL:

[https://anypoint.mulesoft.com/exchange/portals/muletraining/.](https://anypoint.mulesoft.com/exchange/portals/muletraining/)

The screenshot shows the MuleSoft Training API portal. At the top, there's a navigation bar with the MuleSoft logo, a "Training" link, a "Home" link, and a "Login" link. The main heading is "Welcome to your MuleSoft Training portal!". Below it, a sub-headline reads: "Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience." On the left side, there's a sidebar titled "Assets" with a "All types" dropdown, a search bar, and a "X" button. Below the sidebar, there's a card for the "American Flights API". The card has a "REST API" icon, a five-star rating, and the text "American Flights API".

15. Click the American Flights API.
16. Browse the API Summary on the left-side and discover the resources that are available for the API.

The screenshot shows the "American Flights API" summary page. At the top, there's a navigation bar with the MuleSoft logo, a "Training" link, a "Home" link, and a "Login" link. To the left, there's a sidebar with a back arrow to "Assets list", the "American Flights API" name, and a "Request access" button. The main content area has a large green "API summary" section with a house icon, the API name "American Flights API v2", and a five-star rating with "(0 reviews)". Below this, a note states: "NOTE: Version v2 of this API is for use with Mule 4.1 courses. It has a 4.X.X proxy that uses HEADERS for client_id authentication." Further down, it says: "The American Flights API is a system API for operations on the `american` table in the `training database`". Under "Supported operations", there's a list of endpoints with their HTTP methods: "GET /flights", "POST /{ID}", "GET /{ID}", "DELETE /{ID}", and "PUT /{ID}". At the bottom, there's a "API instances" section.

17. Select the GET method.
18. Review the information about the GET method; you should see there is an optional query parameter called destination.

American Flights API | v2

`/flights : get`

Request

GET `/flights`

Parameters

Parameter	Type	Description
destination	string (enum)	Possible values: SFO, LAX, CLE

19. Scroll down and review the Headers and Response sections.

Headers

Parameter	Type	Description
client_id (required)	string	
client_secret (required)	string	

Response

200	Type application/json	Examples
	Type	
	<pre>[{"ID": "integer", "code": "string", "price": "number", "departureDate": "string", "origin": "string", "destination": "string"}]</pre>	

20. In the left-side navigation, select the DELETE method.
21. Locate information about the required ID URI parameter.

The screenshot shows a left sidebar with a tree view of API resources and methods, and a main panel displaying detailed information for the `/flights/{ID}` resource.

API summary

- > Types
- ▽ Resources
 - ▽ /flights
 - GET**
 - POST**
 - ▽ /{ID}
 - GET**
 - DELETE**
 - PUT**
 - API instances

/ID : delete

Request

DELETE `/flights/{ID}`

Parameters

Parameter	Type	Description
URI parameters		
ID (required)	string	

22. Review the information for the other resources.

Walkthrough 1-2: Make calls to an API

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

- Use Advanced REST Client to make calls to an unsecured API (an implementation).
- Make GET, DELETE, POST, and PUT calls.
- Use Advanced REST Client to make calls to a secured API (an API proxy).
- Use the API console in an API portal to make calls to a managed API using a mocking service.
- Use the API console to make calls to an API proxy endpoint.

Use Advanced REST Client to make GET requests to retrieve data

1. Return to or open Advanced REST Client.
2. Make sure the method is set to GET.
3. Return to the course snippets.txt file.
4. Copy the URL for the American Flights web service:
<http://training4-american-ws.cloudhub.io/api/flights>.

Note: This is the URL for the API implementation, not the managed API proxy. The -ws stands for web service.

5. Return to Advanced REST Client and paste the URL in the text box that says Request URL.

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

200 OK 1283.27 ms DETAILS ▾

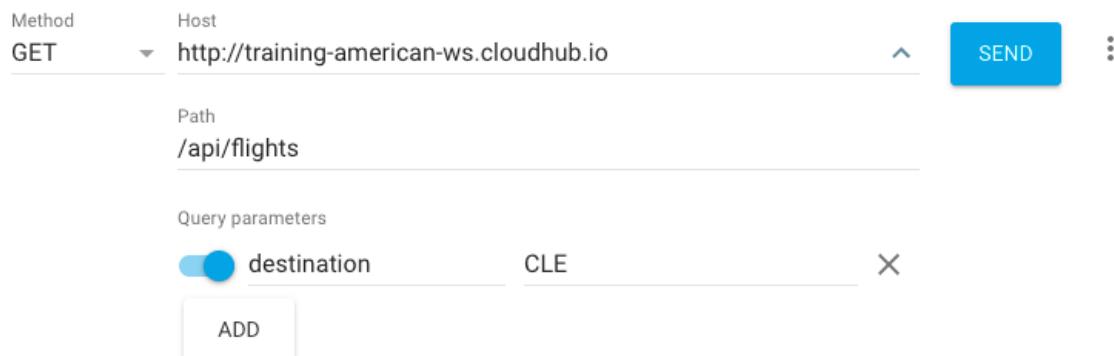
```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-1: {
  "ID": 2,
  "code": "eefd0123",
```

9. Click the Toggle raw response view button.

200 OK 1283.27 ms DETAILS ▾

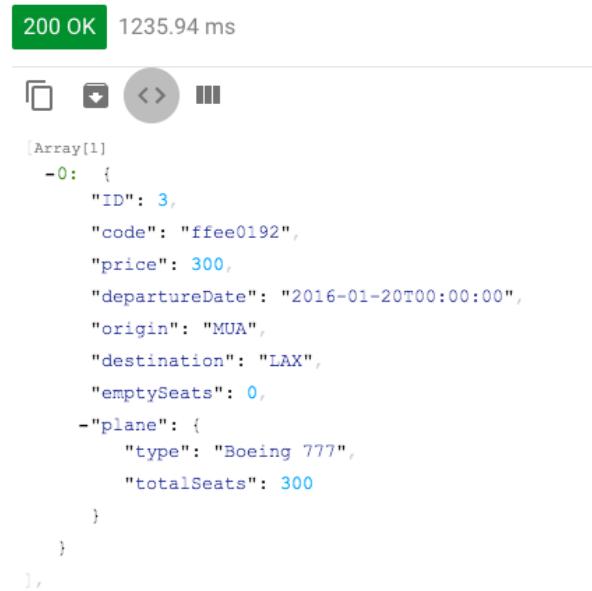
```
[{
  {
    "ID": 1,
    "code": "rree0001",
    "price": 541,
    "departureDate": "2016-01-20T00:00:00",
    "origin": "MUA",
    "destination": "LAX",
    "emptySeats": 0,
    "plane": {
      "type": "Boeing 787",
      "totalSeats": 200
    }
  },
}
```

10. Click the arrow to the right of the URL.
11. In the area that appears, set the Param name to destination and the value to CLE.



The screenshot shows a REST client interface. At the top, it displays a 'Method' dropdown set to 'GET' and a 'Host' field containing 'http://training-american-ws.cloudhub.io'. Below this, the 'Path' is set to '/api/flights'. Under 'Query parameters', there is a row for 'destination' with the value 'CLE'. A blue selection bar highlights the 'destination' label. An 'ADD' button is visible below the parameter row. To the right of the interface, there is a 'SEND' button and a vertical ellipsis menu icon.

12. Click the Send button; you should get just flights to CLE returned.
13. Click the X next to the parameter to delete it.
14. Click the arrow to the right of the URL to collapse the parameters section.
15. Change the request URL to use a uri parameter to retrieve the flight with an ID of 3:
<http://training4-american-ws.cloudhub.io/api/flights/3>.
16. Click the Send button; you should see only the flight with that ID returned.



The screenshot shows a REST client interface with a green '200 OK' status bar at the top. Below it, the response time is listed as '1235.94 ms'. The main area displays the JSON response body. The JSON structure is as follows:

```
[Array[1]
-0: {
  "ID": 3,
  "code": "ffee0192",
  "price": 300,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 777",
    "totalSeats": 300
  }
}
]
```

Make DELETE requests to delete data

17. Change the method to DELETE.

18. Click the Send button; you should see a 200 response with a message that the Flight was deleted (but not really).

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

Method Request URL

DELETE <http://training-american-ws.cloudhub.io/api/flights/3>

SEND ::

Parameters ▾

200 OK 253.51 ms DETAILS ▾

□ ↻ ⌂

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

19. Remove the URI parameter from the request: <http://training4-american-ws.cloudhub.io/api/flights>.
20. Click the Send button; you should get a 405 response with a message of method not allowed.

405 Method Not Allowed 200.86 ms DETAILS ▾

□ ↻ ⌂

```
{  
    "message": "Method not allowed"  
}
```

Make a POST request to add data

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

Method Request URL

POST <http://training-american-ws.cloudhub.io/api/flights>

SEND ::

Parameters ▾

415 Unsupported Media Type 261.25 ms DETAILS ▾

□ ↻ ⌂

```
{  
    "message": "Unsupported media type"  
}
```

23. Click the arrow next to Parameters beneath the request URL.
24. Click in the Header name field, type C, and then select Content-Type.
25. In the Header value field, select application/json.

Parameters ^

Headers	Authorization	Body	Variables	Actions
<input type="checkbox"/>  Toggle source mode + Insert headers set				X  
Header name Content-Type	Header value application/json			

26. Select the Body tab.
27. Return to the course snippets.txt file and copy the value for American Flights API post body.
28. Return to Advanced REST Client and paste the code in the body text area.

Parameters ^

Headers	Authorization	Body	Variables	Actions
Body content type application/json				

FORMAT JSON MINIFY JSON

```
{
  "code": "GQ574",
  "price": 399,
  "departureDate": "2016/12/20",
  "origin": "ORD",
  "destination": "SFO",
  "emptySeats": 200,
  "plane": {
    "type": "Boeing 747",
    "totalSeats": 400
  }
}
```

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

201 Created 274.80 ms DETAILS ▾

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

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

Headers Authorization **Body** Variables Actions

Body content type
application/json

FORMAT JSON MINIFY JSON

```
{  
    "code": "GQ574",  
    "price": 399,  
    "departureDate": "2016/12/20",  
    "origin": "ORD",  
    "destination": "SFO",  
    "emptySeats": 200  
}
```

32. Send the request; the message should still post successfully.
33. In the request body, remove the emptySeats key/value pair.
34. Delete the comma after the destination key/value pair.

Headers Authorization **Body** Variables Actions

Body content type
application/json

FORMAT JSON MINIFY JSON

```
{  
    "code": "GQ574",  
    "price": 399,  
    "departureDate": "2016/12/20",  
    "origin": "ORD",  
    "destination": "SFO"  
}
```

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

400 Bad Request 291.33 ms DETAILS ▾

□ ↴ <> ⌂

```
{  
    "message": "Bad request"  
}
```

Make a PUT request to update data

36. Change the method to PUT.
37. Add a flight ID of 3 to the URL.
38. Click the Send button; you should get a 400 Bad Request.

400 Bad Request 190.50 ms DETAILS ▾

□ ↴ ⌂ ⌂

```
{  
    "message": "Bad request"  
}
```

39. In the request body field, press Cmd+Z or Ctrl+Z so the emptySeats field is added back.
40. Send the request; you should get a 200 OK response with the message Flight updated (but not really).

200 OK 225.24 ms DETAILS ▾

□ ↴ ⌂ ⌂

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

Make a request to a secured API

41. Change the method to GET.
42. Change the request URL to <http://training4-american-api.cloudhub.io/flights/3>.

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

43. Click the Send button; you should get a 401 Unauthorized response with a message about an invalid client_id or secret.

401 Unauthorized 393.32 ms DETAILS ▾

□ ↴ ⌂ ⌂

```
Unable to retrieve client_id from message
```

44. Return to the course snippets.txt file and copy the value for the American Flights API client_id.
45. Return to Advanced REST Client and add a header called client_id.
46. Set client_id to the value you copied from the snippets.txt file.

47. Return to the course snippets.txt file and copy the value for the American Flights API client_secret.
48. Return to Advanced REST Client and add a second header called client_secret.
49. Set client_secret to the value you copied from the snippets.txt file.

Parameters ^

Headers	Authorization	Variables	Actions
<input type="checkbox"/> <> Toggle source mode + Insert headers set			X
Header name Content-Type	Header value application/json		X
Header name client_id	Header value d1374b15c6864c3682ddbed2a247a826		X
Header name client_secret	Header value 4a87fe7e2e43488c927372AEF981F066		X

50. Click the Send button; you should get data for flight 3 again.

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

200 OK 1399.56 ms DETAILS ▾

```
Array[1]
-0: {
  "ID": 3,
  "code": "ffee0192",
  "price": 300,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 777",
    "totalSeats": 300
  }
}
```

51. Click the Send button three more times; you should get a 429 Too Many Requests response with a Quota has been exceeded message.

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

429 Too Many Requests 209.24 ms DETAILS ▾

```
{
  "error": "Quota has been exceeded"
}
```

Use the API console in the API portal to make requests to the API using a mocking service

52. Return to the browser window with the American Flights API portal at
<https://anypoint.mulesoft.com/exchange/portals/muletraining>.

53. In the left-side navigation click the GET method for /flights.

54. Review the Headers and Response sections.

55. In the Response section, select Examples.

Response

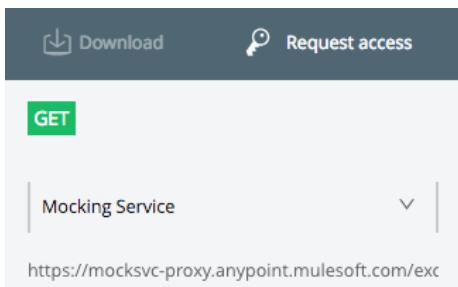
Type application/json

200 Examples

```
[{"ID": 1, "code": "ER38sd", "price": 400, "departureDate": "2017/07/26", "origin": "CLE", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 2, "code": "ER45if", "price": 540.99, "departureDate": "2017/07/27", "origin": "SFO", "destination": "ORD", "emptySeats": 54, "plane": {"type": "Boeing 777", "totalSeats": 300}}]
```

56. In the API console located on the right side of the page, make sure the Mocking Service endpoint is selected.

57. Look at the endpoint URL that is displayed.



58. Click the Show optional parameters checkbox.

59. Select LAX in the destination drop-down menu.

60. Click Send.

61. Look at the response; you should get the example flights that are to SFO.

Parameters Headers

Query parameters Show optional parameters

LAX

Send

200 OK 754.20 ms Details ↴

✓ ↴ </> □

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emotvSeats": 0.
}
```

Make requests to the API using an API proxy endpoint

62. At the top of the API console, change the endpoint to Production – Rate limiting SLA based policy.

63. Look at the endpoint URL that is displayed.

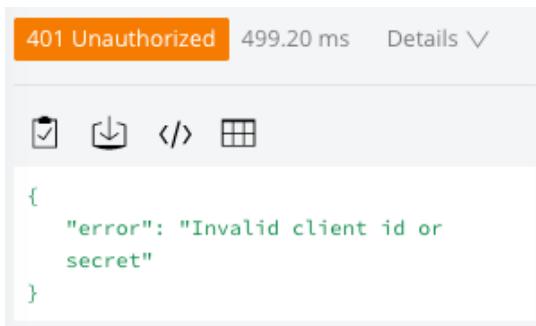
Download Request access

GET

Production - Rate limiting SLA based pol... ↴

http://training4.american-api.cloudhub.io/flights?i

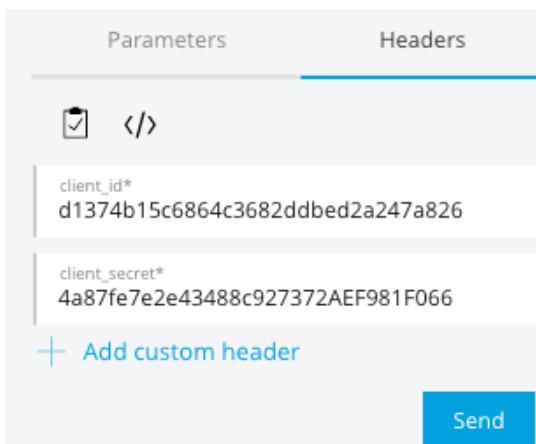
64. Click Send; you should get a 401 Unauthorized response.



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

65. Select the Headers tab.

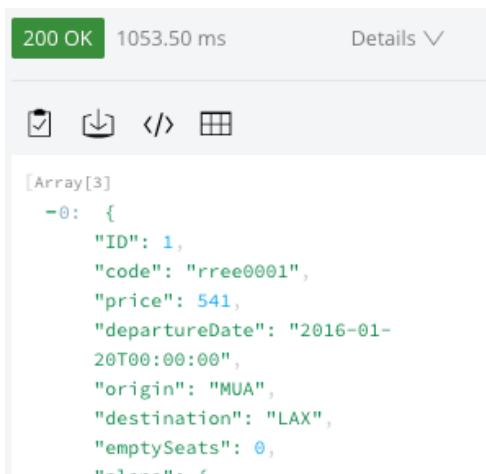
66. Copy and paste the client_id and client_secret values from the course snippets.txt file.



Parameters	Headers
<input checked="" type="checkbox"/> </>	
	client_id* d1374b15c6864c3682ddbed2a247a826
	client_secret* 4a87fe7e2e43488c927372AEF981F066
	+ Add custom header

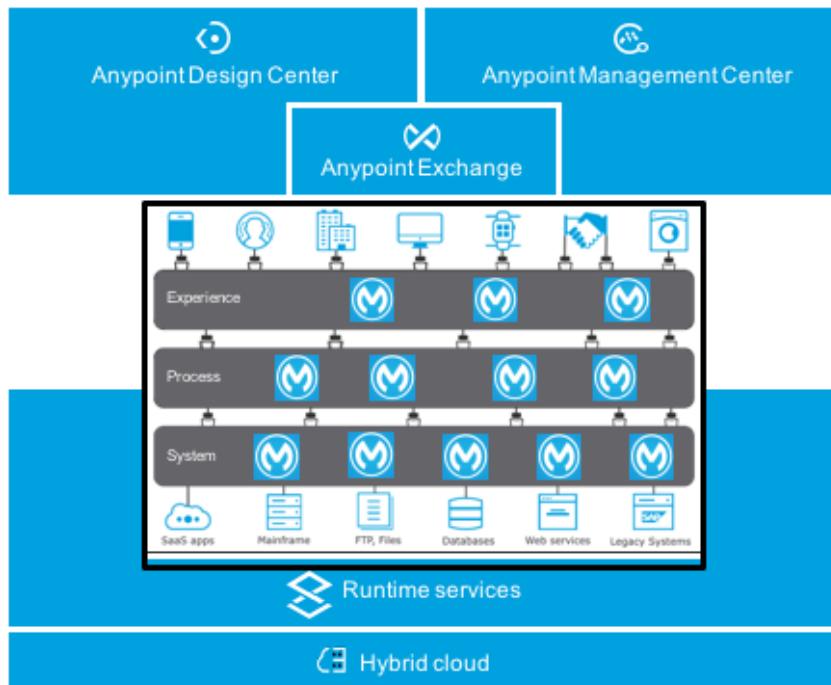
Send

67. Click Send; you should get a 200 OK response with only flights to LAX.



```
200 OK 1053.50 ms Details ▾  
  
[  
  Array[3]  
  -0: {  
    "ID": 1,  
    "code": "rree0001",  
    "price": 541,  
    "departureDate": "2016-01-  
20T00:00:00",  
    "origin": "MUA",  
    "destination": "LAX",  
    "emptySeats": 0,  
    -"plane": {}  
  }  
]
```

Module 2: Introducing Anypoint Platform



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

- Describe the benefits of Anypoint Platform and MuleSoft's approach to be successful with it.
- Describe the role of each component in building application networks.
- Navigate Anypoint Platform.
- Locate APIs and other assets needed to build integrations and APIs in Anypoint Exchange.
- Build basic integrations to connect systems using flow designer.

Walkthrough 2-1: Explore Anypoint Platform and Anypoint Exchange

In this walkthrough, you get familiar Anypoint Platform. You will:

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

The screenshot shows the Anypoint Exchange interface. The left sidebar has sections for Assets, Organizations, MuleSoft, Training, My applications, and Public portal. The main area is titled 'Assets' with a search bar containing 'american'. Below the search bar, it says 'Showing results for "american". Save this search'. There are three cards displayed: 1. REST API: Training: American Flights API by MuleSoft, rated 5 stars. 2. RAML Fragment: Training: American Flight Data Type by MuleSoft, rated 4 stars. 3. RAML Fragment: Training: American Flights Example by MuleSoft, rated 5 stars.

Return to Anypoint Platform

1. Return to Anypoint Platform at <https://anypoint.mulesoft.com> (not the public API portal you used last module!) in a web browser.

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

2. Click the menu button located in the upper-left in the main menu bar.
3. In the menu that appears, select Anypoint Platform; this will return you to the home page.

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

The screenshot shows the Anypoint Platform main menu. The left sidebar lists: Anypoint Platform (selected), Design Center, Exchange (selected), Management Center, Access Management, API Manager, Runtime Manager, and Data Gateway. The main content area shows 'Information' for the 'Training' tier, which expires on 12/12/2017. It also shows 'Sandboxes' (Test environments for QA of applications) with 0 vCores and 0 Static IP. At the bottom, there are navigation links for 'Home', 'Logout', and 'Help'.

Explore Anypoint Platform

4. In the main menu, select Access Management.
5. In the left-side navigation, select Users.
6. In the left-side navigation, select Environments.

The screenshot shows the 'Access Management' interface with the 'Environments' tab selected. On the left, a sidebar lists 'Organization', 'Users', 'Roles', 'Environments' (which is selected and highlighted in blue), 'External Identity', and 'Audit Logs'. Under 'SETTINGS', it lists 'Runtime Manager'. Under 'SUBSCRIPTION', it also lists 'Runtime Manager'. The main area displays a table titled 'Environments' with two rows:

Name	Type
Design	Design
Sandbox	Sandbox

A blue button labeled 'Add environment' is located at the top left of the main area.

7. In the main menu, select Design Center.
8. Click the Create button and look at the options in the drop-down menu.

The screenshot shows the 'Design Center' interface. On the left, a sidebar lists 'Projects' (selected and highlighted in blue), 'Search...', and a magnifying glass icon. Below this is a table with columns 'Name', 'Project Type', and 'Last Update'. A blue button labeled '+ Create' is highlighted in a dropdown menu, which also includes options: 'Mule Application', 'API specification', 'API fragment', and 'Get Anypoint Studio'. To the right, there is a large button with a mountain icon and the text 'Get Started'.

9. In the main menu, select Runtime Manager.
10. If you get a Choose environment page, select Design.

The screenshot shows the 'Runtime Manager' interface with the 'DESIGN' tab selected (highlighted in blue). On the left, a sidebar lists 'Applications' (selected and highlighted in blue), 'Servers', 'Alerts', 'VPCs', and 'Load Balancers'. The main area features a large gray 'MuleSoft' logo. Below it, the text 'There are no applications to show' is displayed. At the bottom, there is a button labeled 'Deploy application'.

11. In the main menu, select API Manager.

The screenshot shows the MuleSoft API Manager interface. The top navigation bar includes 'Training', '?', and 'MM' buttons. The left sidebar has a 'Sandbox' tab selected, along with 'API Administration', 'Client Applications', 'Custom Policies', and 'Analytics'. The main content area displays a message: 'No APIs to display. Get started by adding your first API.' It features a chart icon and a note: 'Select an API version to see more details'.

Explore Anypoint Exchange

12. In the main menu, select Exchange.

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

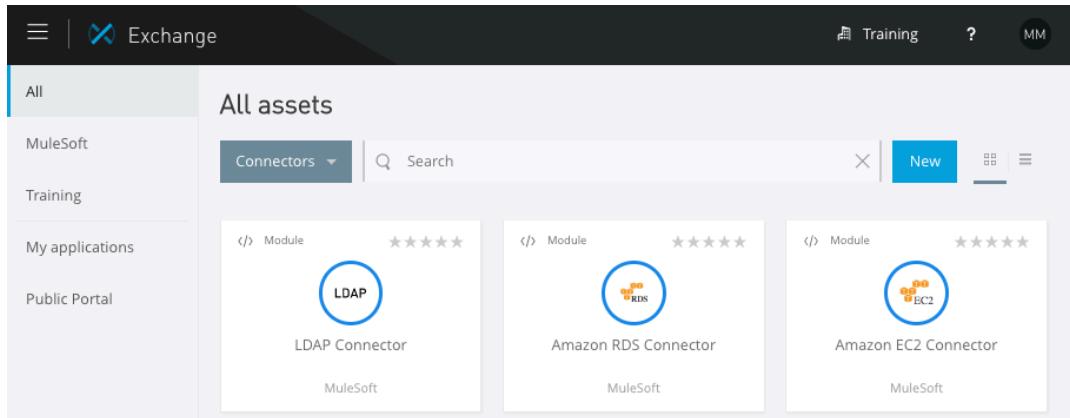
The screenshot shows the MuleSoft Exchange interface. The top navigation bar includes 'Training', '?', and 'MM' buttons. The left sidebar shows 'Assets' selected, followed by 'Organizations', 'MuleSoft', 'Training', 'My applications', and 'Public portal'. The main content area displays three asset cards: 'Microsoft Dynamics CRM Connector' (Module), 'TRADACOMS EDI Connector' (Module), and 'DevRel-Quick Start Products API Connector' (Connector).

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

The screenshot shows the MuleSoft Exchange interface with 'Training' selected in the left sidebar. The main content area is empty, indicating no private content is available.

15. In the left-side navigation, select MuleSoft.

16. In the types menu, select Connectors.



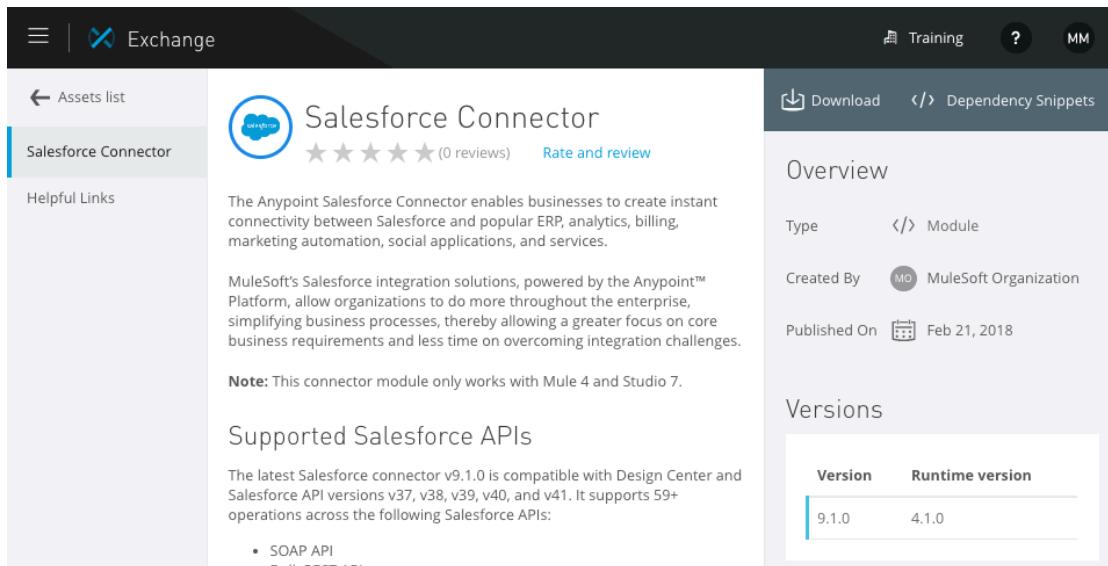
The screenshot shows the Anypoint Exchange interface. The left sidebar has a 'Connectors' option selected. The main area displays three connector modules: 'LDAP Connector' (MuleSoft), 'Amazon RDS Connector' (MuleSoft), and 'Amazon EC2 Connector' (MuleSoft). Each module has a blue circular icon and a five-star rating.

17. Select one of the connectors and review its information.

18. In the left-side navigation, click the Assets list link.

19. Search for the Salesforce Connector and review its details.

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



The screenshot shows the details page for the 'Salesforce Connector'. The left sidebar shows 'Assets list' and 'Salesforce Connector' selected. The main content area includes:

- Salesforce Connector**: A circular icon with the word 'Salesforce' and a star rating of 5 stars (0 reviews).
- Overview**: Includes fields for Type (Module), Created By (MuleSoft Organization), and Published On (Feb 21, 2018).
- Versions**: A table with two columns: Version (9.1.0) and Runtime version (4.1.0).
- Supported Salesforce APIs**: A list of supported APIs including SOAP API and REST API.

20. In the left-side navigation, click Assets list.

21. In the types menu, select Templates.

22. Remove salesforce from the search field and press Enter/Return.

Browse REST APIs in Anypoint Exchange

23. In the types menu, select REST APIs.

24. Browse the APIs.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a sidebar with categories: All, MuleSoft, Training, My applications, and Public Portal. The main area is titled "All assets" and has a "REST APIs" dropdown. A search bar and a "New" button are at the top right. Three API cards are listed:

- Optymyze API** (MuleSoft) - 5 stars
- CardConnect REST API** (MuleSoft) - 5 stars
- Nexmo SMS API** (MuleSoft) - 5 stars

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

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

This screenshot shows a single API card for the "Training: American Flights API". It has a green circular icon with a house symbol, a 5-star rating, and the title "Training: American Flights API" followed by "MuleSoft".

26. Review the API portal.

This screenshot shows the detailed view of the "Training: American Flights API". The left sidebar lists resources like "/flights", "/{ID}", and "API instances". The main content area shows the API summary, a preview of the RAML 1.0 file, and the "Asset versions for 1.0" section.

Asset versions for 1.0

Version	Instances
1.0.1	Mocking Service
1.0.0	

27. In the left-side navigation, expand and review the list of available resources.

28. Click the GET link for the /flights resource.

29. On the /flights: Get all flights page, review the information for the optional destination query parameter; you should see the API is similar to the one you explored in the public Muletraining portal.

The screenshot shows the MuleSoft API Exchange interface. On the left, there's a sidebar with navigation links like 'Assets list', 'Training: American Flights API', 'API summary', 'Types', 'Resources', '/flights' (which is expanded), and 'API instances'. Under '/flights', there are four items: 'GET Get all fl...', 'POST Add a flight', 'GET Get a fl...', and 'DELETE Del...'. The 'GET Get all fl...' item is highlighted. The main content area has a title 'Training: American Flights API | 1.0' with a 5-star rating. Below it, the endpoint '/flights : Get all flights' is listed with a 'Request' section containing a GET method and URL. To the right, a detailed view of the 'Get all flights' endpoint is shown. It includes a 'Mocking Service' dropdown set to 'Mocking Service', a 'Parameters' tab (selected), and a 'Headers' tab. Under 'Parameters', there's a table for 'Query parameters' with one entry: 'destination' (string, enum) which is described as 'Destination airport code' with possible values 'SFO, LAX, CLE'. A 'Send' button is at the bottom right of this panel.

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

30. In the API console, review the options for the instances you can test.

The screenshot shows the API console interface. At the top, there's a 'Download' button. Below it, the 'Get all flights' endpoint is selected. On the right, there's a dropdown menu titled 'Mocking Service' which lists three options: 'Mocking Service' (selected), 'Mocking Service', and 'RAML Base URI'.

31. Select Mocking Service.

32. Select a destination and click Send; you should get the two example flights.

The screenshot shows the MuleSoft Anypoint Platform Mocking Service interface. At the top, it says "GET Get all flights". Below that, it says "Mocking Service" and provides the URL "https://mocksvc-proxy.anypoint.mulesoft.com/exc". There are tabs for "Parameters" and "Headers", with "Parameters" being active. Under "Query parameters", there is a dropdown menu set to "LAX". A "Send" button is below the dropdown. At the bottom, it shows a successful response: "200 OK" with a duration of "295.75 ms". A "Details" link is available. The response body is shown as an array of two flight objects:

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
```

33. Change the API instance from Mocking Service to RAML Base URI.

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

The screenshot shows the MuleSoft Anypoint Platform Mocking Service interface. At the top, it says "GET Get all flights". Below that, it says "RAML Base URI" and provides the URL "http://training-american-ws.cloudhub.io/api/flights". There are tabs for "Parameters" and "Headers", with "Parameters" being active. Under "Query parameters", there is a dropdown menu set to "LAX". A "Send" button is below the dropdown. At the bottom, it shows a successful response: "200 OK" with a duration of "1310.77 ms". A "Details" link is available. The response body is shown as an array of three flight objects:

```
[Array[3]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
```

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

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

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

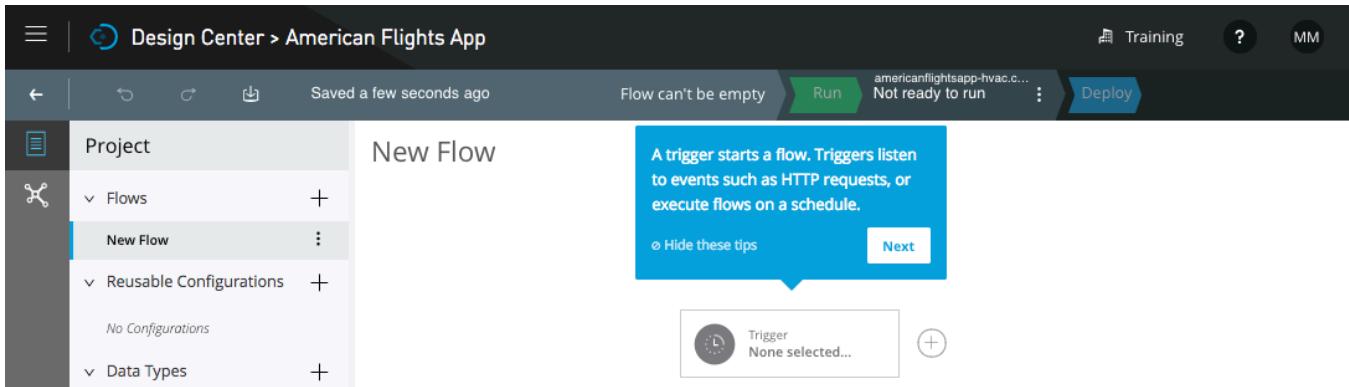
The screenshot shows the Anypoint Platform Design Center interface. At the top, it says "Design Center > American Flights App". Below that, there's a toolbar with "Run" (green), "Running", and "Deploy" buttons. The main area shows a "Project" tree on the left with "Flows", "Reusable Configurations", and "Data Types". A "Get flights" flow is selected. The flow diagram shows an "HTTP Listener" component connected to a "Logger" component. The "Logs" panel at the bottom shows several INFO-level log entries related to the application's startup and tracking handlers.

Create a Mule application project in Design Center

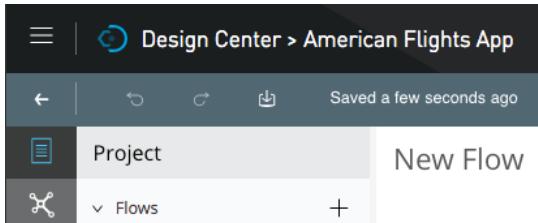
1. Return to Anypoint Platform.
2. In the main menu, select Design Center.
3. Click the Create button and select Mule Application.

The screenshot shows the Anypoint Platform Design Center interface with the "Create" dialog open. The dialog has "Mule Application" selected. Other options shown are "API specification" and "API fragment". There's also a link to "Get Anypoint Studio".

- In the New Mule Application dialog box, set the project name to American Flights App.
- Click Create; flow designer should open.



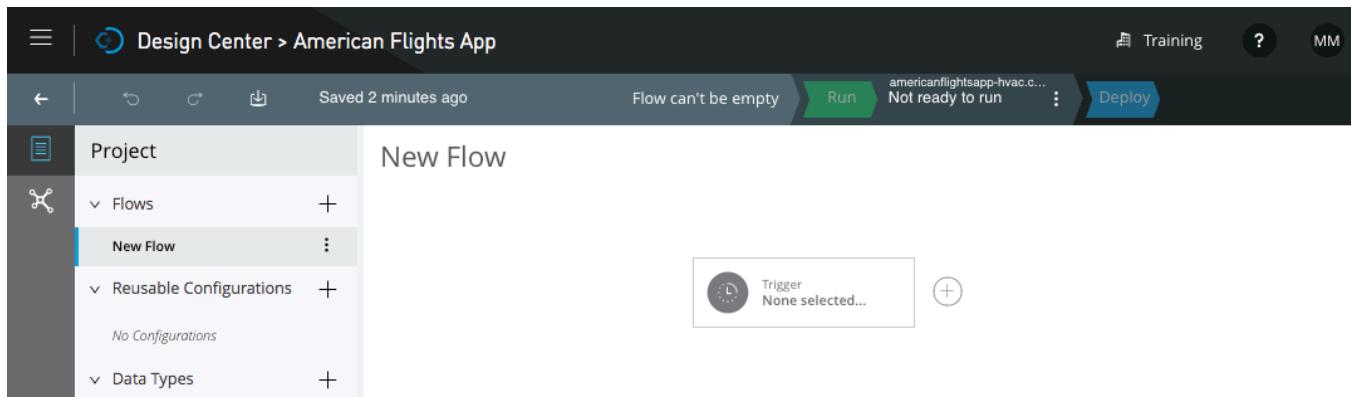
- In the pop-up box in flow designer, click Hide these tips.
- Click the arrow icon in the upper-left corner; you should return to the Design Center.



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

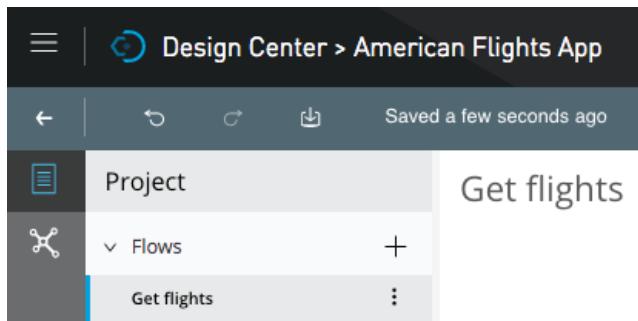
Name	Project Type	Last Update
American Flights App	Mule Application	April 18th, 2018

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



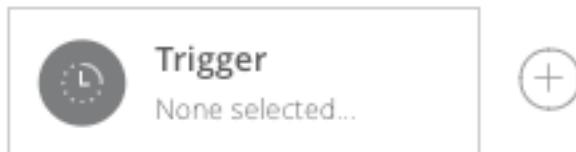
Rename the flow

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

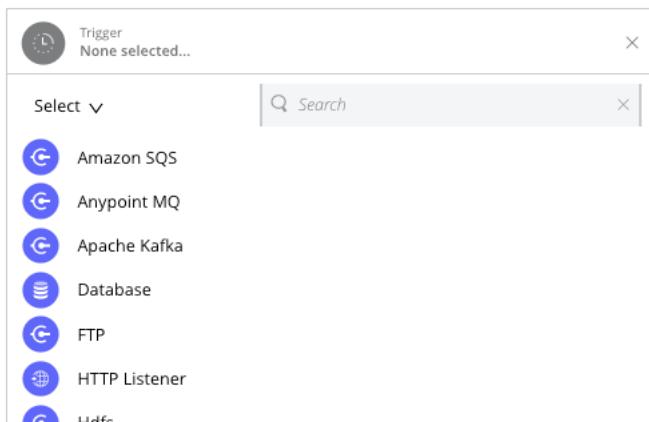


Create an HTTP trigger for a flow in the application

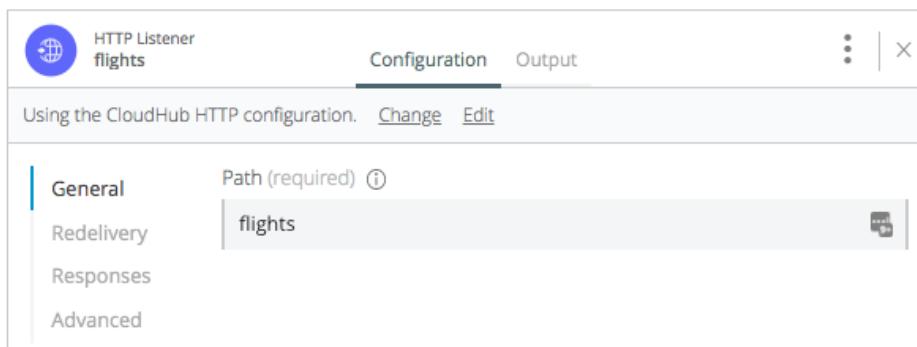
13. In flow designer, click the Trigger card.



14. In the Trigger card, select HTTP Listener.

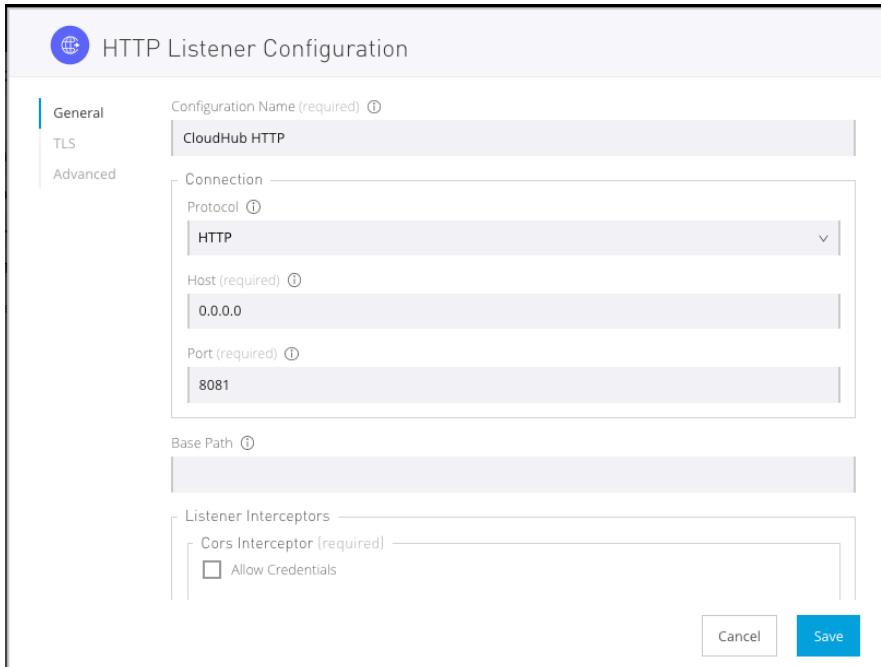


15. In the HTTP Listener dialog box, set the path to flights.

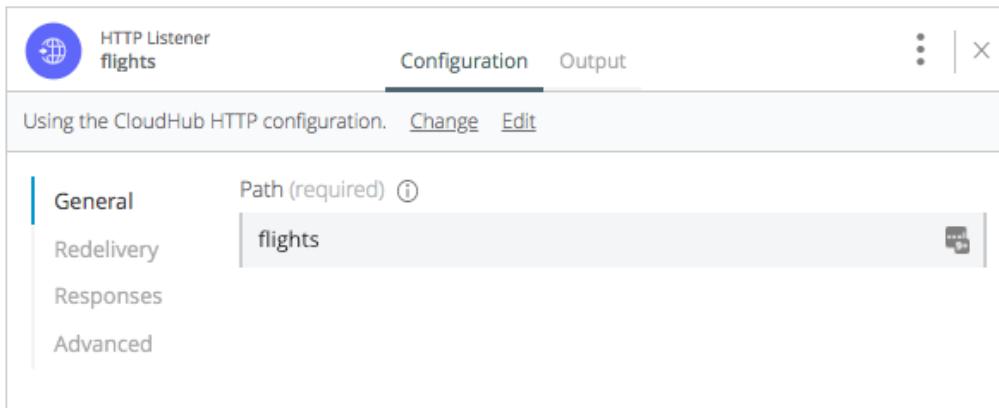


16. Click the Edit link for the CloudHub HTTP configuration.

17. In the HTTP Listener Configuration dialog box, review the information and click Cancel.

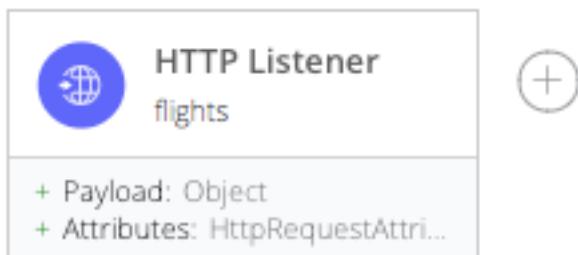


18. In the HTTP Listener dialog box, click the close button in the upper-right corner.

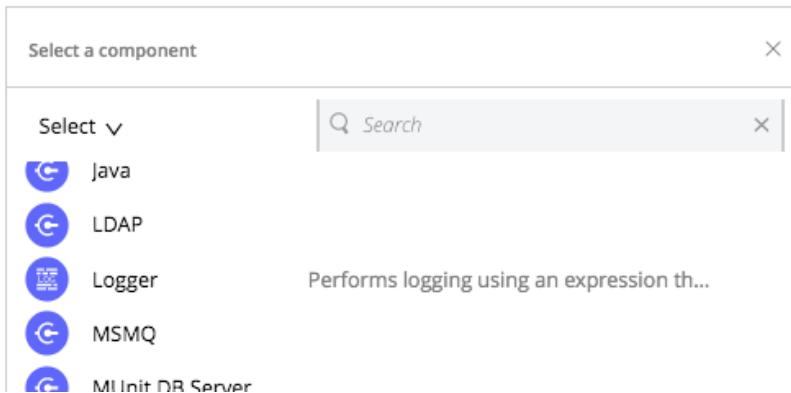


Add a Logger

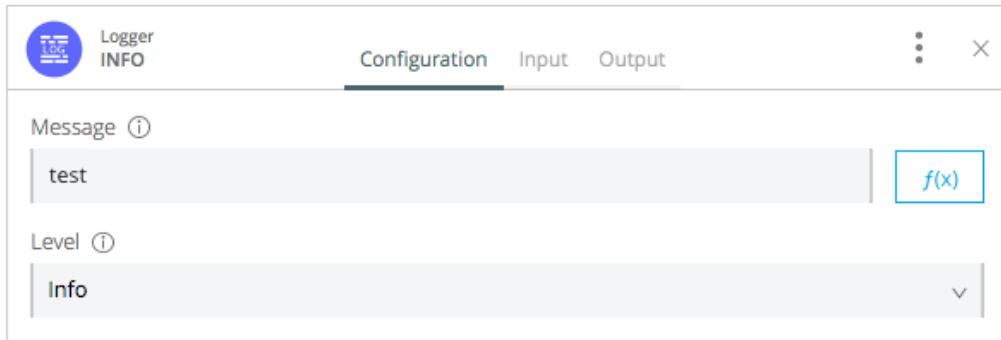
19. Click the add button next to the HTTP Listener card.



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

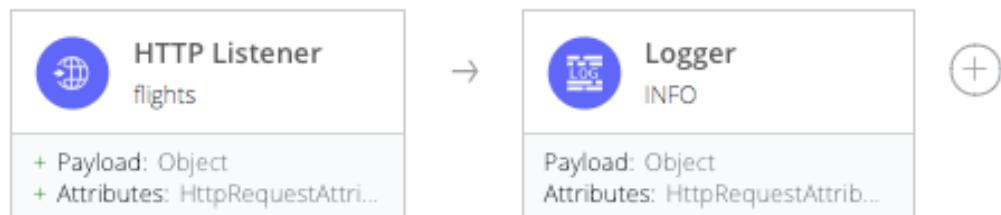


21. In the Logger dialog box, set the message to test.



22. Close the card.

23. Notice that there are gray lines across the middle of both cards.



Deploy the application

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

```
INFO 10:59:11 ****
* Application: americanflightsapp-dgzb
* OS encoding: UTF-8, Mule encoding: UTF-8
*
*****
SYSTEM 10:59:12 Worker(54.173.168.187): Your application has started successfully.
SYSTEM 10:59:12 Your application is started.
```

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

Design Center > American Flights App Training ? MM

Run americanflightsapp-dgzb.c... Deploy

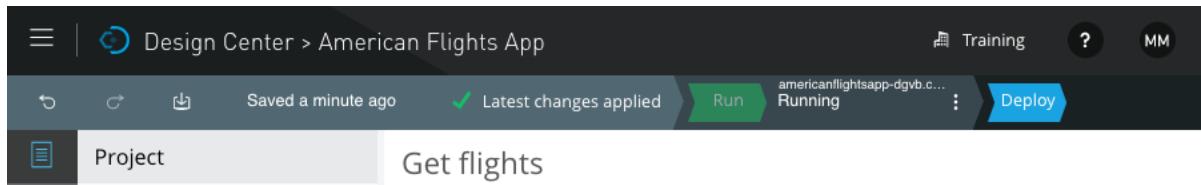
Project Get flights

26. Look at the generated URL for the application.

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

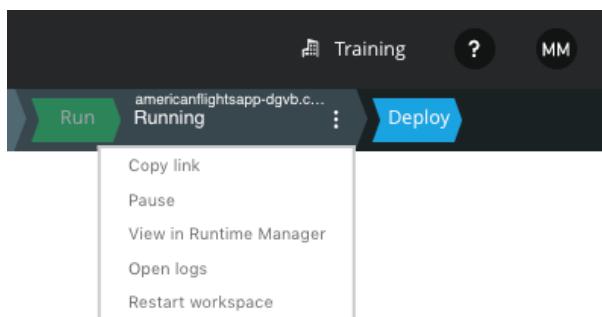
27. Click the Run button.

28. Watch the application status; it should change from Ready to run to Starting application to Running.



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

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



Test the application

30. Return to Advanced REST Client, paste the copied link, and click Send; you should get a 404 Not Found status with a No listener for endpoint: / message.

Method: GET Request URL: http://americanflightsapp-dgzb.cloudhub.io/

Parameters: **404 Not Found** 184.90 ms DETAILS

No listener for endpoint: /

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

Method: GET Request URL: http://americanflightsapp-dgzb.cloudhub.io/flights

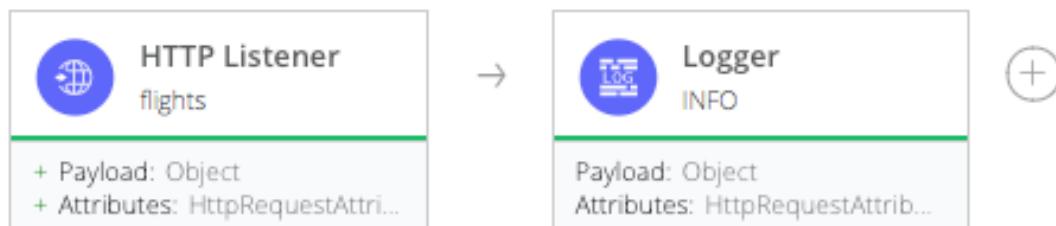
Parameters: ▾

200 OK 286.15 ms DETAILS ▾

32. Click Send again to make a second request.

33. Return to flow designer.

34. Notice that there are now green lines across both cards.

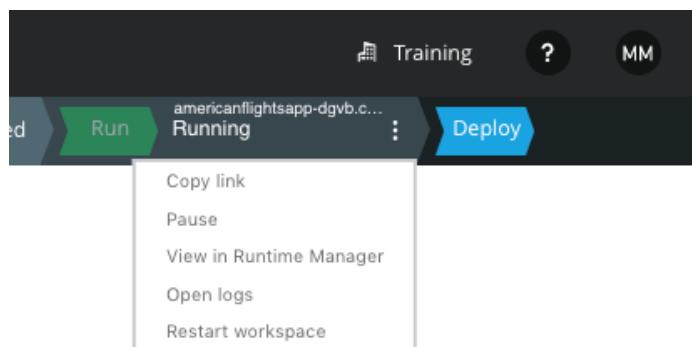


35. Look at the logs; you should see your Logger message displayed twice.

Level	Time	Message
INFO	11:10:16	Skipping the initialization of the mule.agent.tracking.handler.splunk Internal Handler because it's disabled.
INFO	11:10:16	Skipping the initialization of the tracking.notification.internal.message.handler Internal Handler because it's disabled.
INFO	11:10:16	Skipping the initialization of the mule.agent.tracking.handler.log Internal Handler because it's disabled.
INFO	11:10:16	Initializing the mule.agent.tracking.handler.cloudu... mule.agent.tracking.handler.cloudu... initialized successfully.
INFO	11:10:16	test
INFO	11:12:13	test

View the application in Runtime Manager

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



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

The screenshot shows the Runtime Manager interface. The left sidebar has a 'Logs' option selected. The main area shows the 'Live Console' for the application 'americanflightsapp-dgzb'. The console output includes several log entries:

```
[americanflightsapp-dgzb].get_flights.ring-buffer.01      INFO  
Skipping the initialization of the  
tracking.notification.internal.message.handler Internal Handler  
because it's disabled.  
  
11:10:16.887    03/25/2018    Worker-0  
[americanflightsapp-dgzb].get_flights.ring-buffer.01      INFO  
Skipping the initialization of the mule.agent.tracking.handler.log  
Internal Handler because it's disabled.  
  
11:10:16.888    03/25/2018    Worker-0  
[americanflightsapp-dgzb].get_flights.ring-buffer.01      INFO  
Initializing the mule.agent.tracking.handler.cloudhub.event ...  
  
11:10:16.895    03/25/2018    Worker-0  
[americanflightsapp-dgzb].get_flights.ring-buffer.01      INFO  
mule.agent.tracking.handler.cloudhub.event initialized successfully.
```

To the right, there is a 'Deployments' section showing a deployment entry for '10:59 - Deployment'.

38. In the left-side navigation, click Settings.

39. Review the settings page and locate the following information for the application:

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

The screenshot shows the Runtime Manager interface with 'Settings' selected in the sidebar. The main area displays the application file details for 'americanflightsapp-dgzb' and its runtime settings:

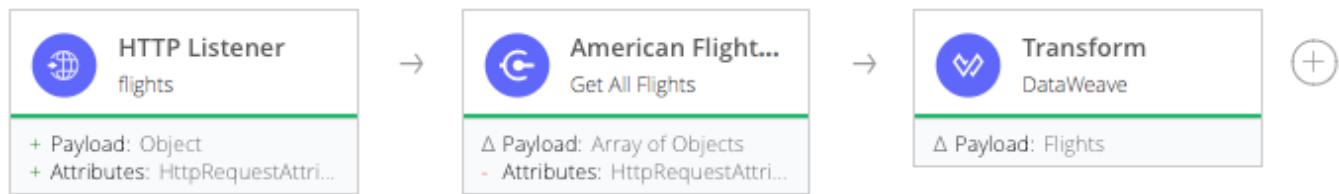
Application File
File: americanflightsapp-dgzb.jar
Last Updated: 2018-03-25 10:59:12AM
App url: americanflightsapp-dgzb.cloudhub.io

Runtime	Properties	Insight	Logging	Static IPs	
Runtime version: 4.1.0	Worker size: 0.2 vCores	Workers: 1			
<input type="checkbox"/> Automatically restart application when not responding					
<input type="checkbox"/> Persistent queues <input type="checkbox"/> Encrypt persistent queues					

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

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

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



Review Mule event data for the calls to the application

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

History
Mar 25, 2018 11:12am
Mar 25, 2018 11:10am

6. Locate your two calls to the application in the History panel; there should be no message payload for either call.
7. Change the Show drop-down menu to Attributes.

8. Review the attributes for the Mule event leaving the HTTP Listener processor.

```
{  
    "listenerPath": "/flights",  
    "relativePath": "/flights",  
    "version": "HTTP/1.1",  
    "scheme": "http",  
    "method": "GET",  
    "requestUri": "/flights",  
    "queryString": "",  
    "localAddress": "ip-172-16-22-97/172.16.22.97:8081",  
    "remoteAddress": "/54.144.217.7:31134",  
    "queryParams": {},  
    "uriParams": {},  
    "requestPath": "/flights",  
    "headers": {  
        "x-forwarded-port": "80",  
        "host": "americanflightsapp-dgzb.cloudhub.io",  
        "x-forwarded-for": "73.231.218.202",  
        "x-real-ip": "73.231.218.202",  
        "x-forwarded-proto": "http"  
    }  
}
```

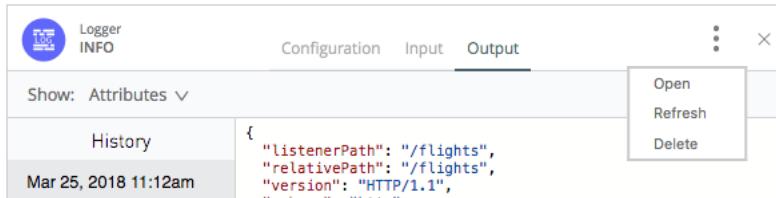
9. Close the card.
10. Open the Logger card.
11. Select the Input tab.
12. Review the payload and attributes values for the two calls.

```
{  
    "listenerPath": "/flights",  
    "relativePath": "/flights",  
    "version": "HTTP/1.1",  
    "scheme": "http",  
    "method": "GET",  
    "requestUri": "/flights",  
    "queryString": "",  
    "localAddress": "ip-172-16-22-97/172.16.22.97:8081",  
    "remoteAddress": "/54.144.217.7:31134",  
    "queryParams": {},  
    "uriParams": {},  
    "requestPath": "/flights",  
    "headers": {  
        "x-forwarded-port": "80",  
        "host": "americanflightsapp-dgzb.cloudhub.io",  
        "x-forwarded-for": "73.231.218.202",  
        "x-real-ip": "73.231.218.202",  
        "x-forwarded-proto": "http"  
    }  
}
```

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

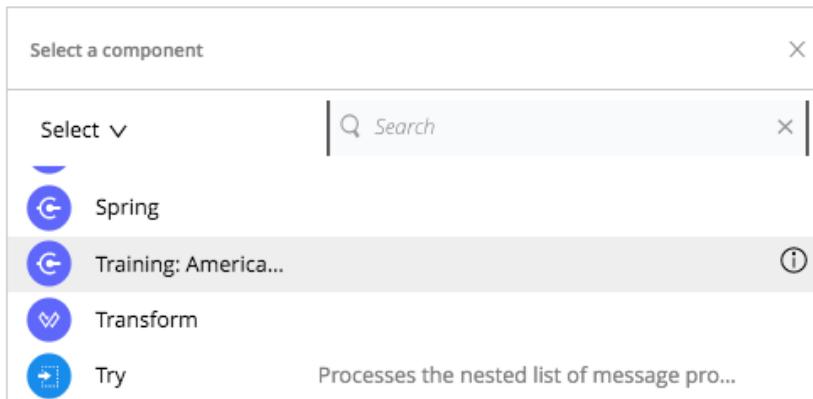
Delete a card

14. Click the options menu for the card and select Delete.

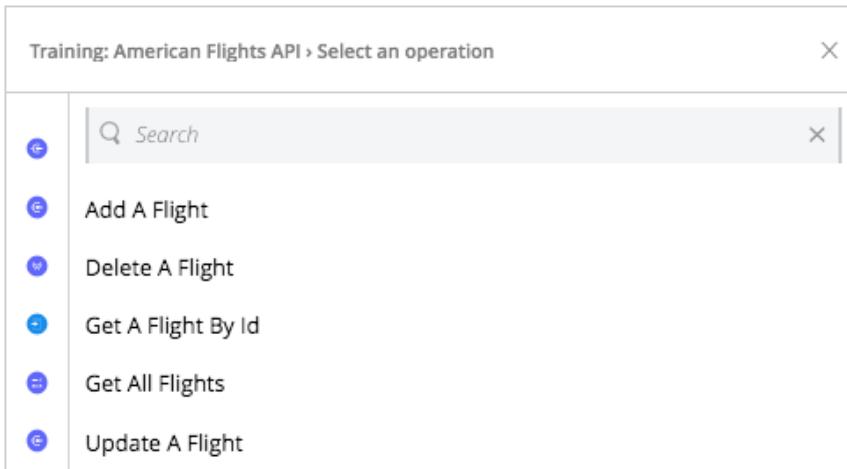


Use the American Flights API in Anypoint Exchange to get all flights

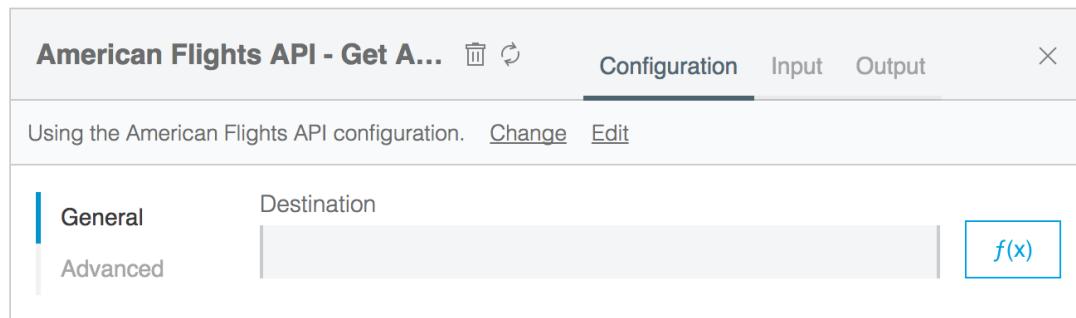
15. Click the Add button next to the HTTP Listener card.
16. In the Select a component dialog box, select the Training: American Flights API.



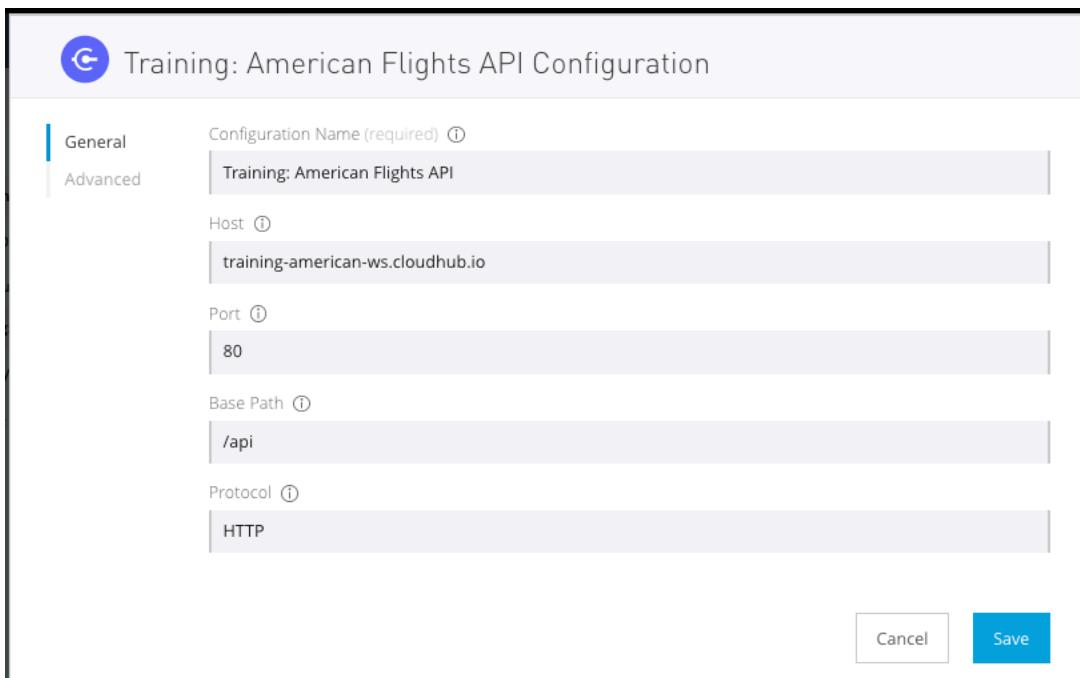
17. In the Training: American Flights API > Select an operation dialog box, select Get All Flights.



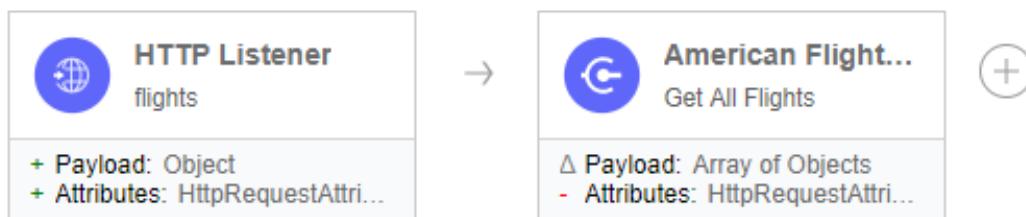
18. In the Get All Flights dialog box, click the Edit link for the Training: American Flights API configuration.



19. Review the information and click Cancel.



20. Close the American Flights API card.

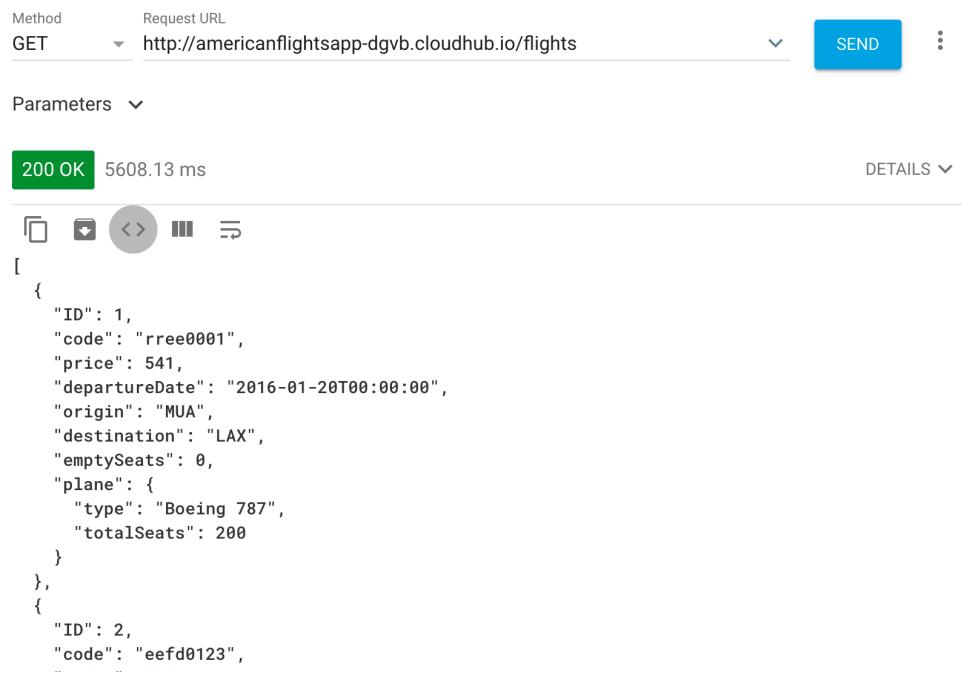


21. Click the Run button in the main menu bar.

22. Wait until the application is running.

Test the application

23. Return to Advanced REST Client and click Send; you should see flight data.



Method Request URL
GET http://americanflightsapp-dgzb.cloudhub.io/flights

Parameters

200 OK 5608.13 ms DETAILS

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

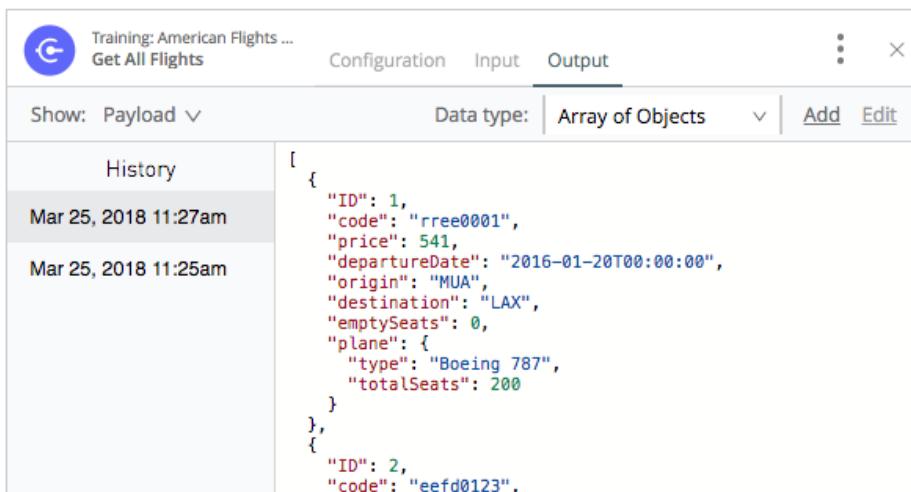
24. Click Send again to make a second request.

Review Mule event data

25. Return to flow designer and open the American Flights API card.

26. Select the Input tab and examine the Mule event data.

27. Select the Output tab; you should see payload data.



Training: American Flights ...
Get All Flights Configuration Input Output

Show: Payload Data type: Array of Objects Add Edit

History Mar 25, 2018 11:27am Mar 25, 2018 11:25am

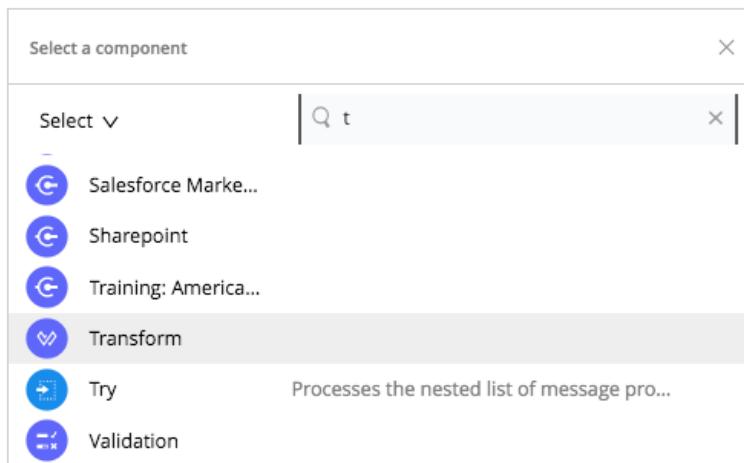
```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2, "code": "eefd0123", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination":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

28. Close the card.

Add and configure a component to transform the data

29. Click the add button in the flow.

30. In the Select a component dialog box, select Transform.



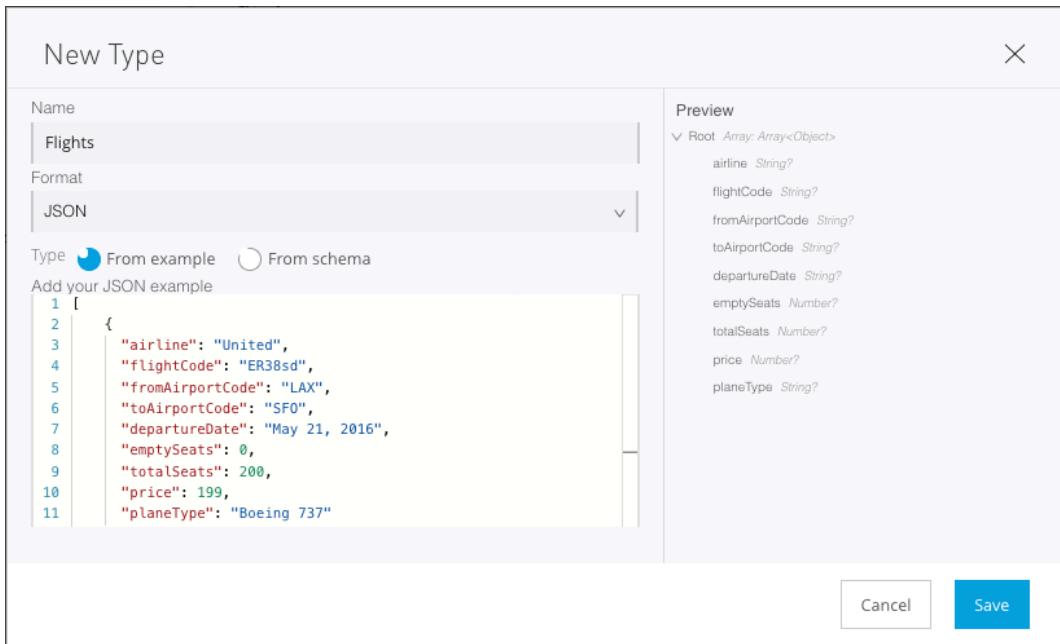
31. In the Transform card, look at the Mule event structure in the input section.

32. In the output section, click the Create new Data Type button.

33. In the New Type dialog box, set the following values:

- Name: Flights
- Format: JSON
- Type: From example

34. In the computer's file explorer, return to the student files folder and locate the flights-example.json file in the examples folder.
35. Open the file in a text editor and copy the code.
36. Return to flow designer and paste the code in the section to add your JSON example.



37. Click Save.
38. In the input section, expand the plane object.

Transform DataWeave

Configuration Input Output

Input

Output payload

Preview

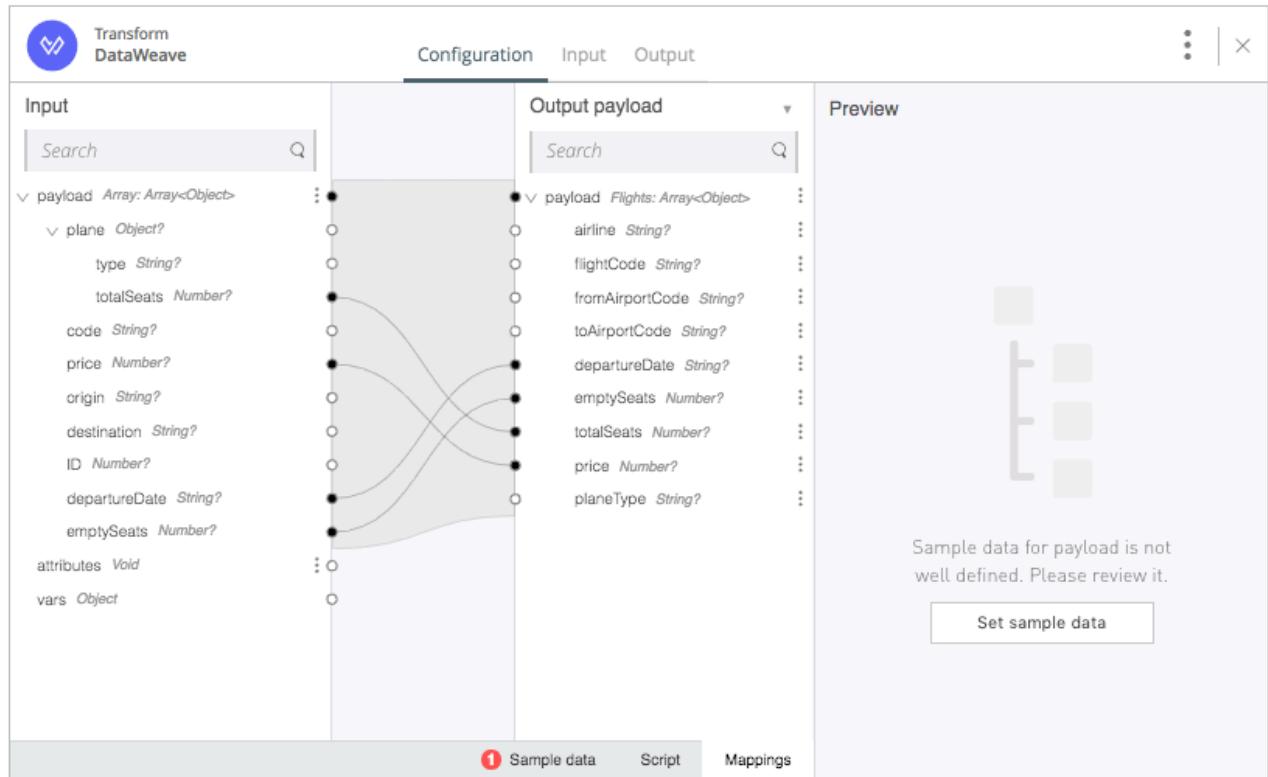
No data available, please perform some mappings and fill required sample data

Mappings

Create the transformation

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

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



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

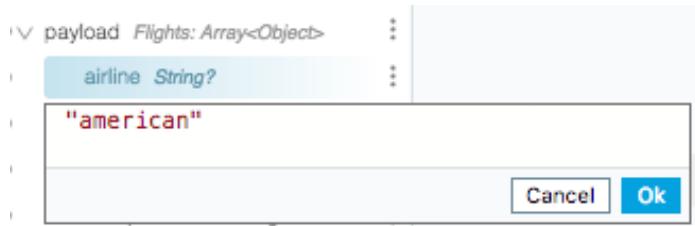
- plane > type to planeType
- code to flightCode
- origin to fromAirport
- destination to toAirport

The screenshot shows the Mule DataWeave Transform tool interface. The top navigation bar includes 'Transform', 'DataWeave', 'Configuration' (selected), 'Input', and 'Output'. The 'Input' panel on the left lists fields such as payload (Array<Object>), plane (Object?), type (String?), totalSeats (Number?), code (String?), price (Number?), origin (String?), destination (String?), ID (Number?), departureDate (String?), emptySeats (Number?), attributes (Void), and vars (Object). The 'Output payload' panel on the right lists fields for Flights (Array<Object>): airline (String?), flightCode (String?), fromAirportCode (String?), toAirportCode (String?), departureDate (String?), emptySeats (Number?), totalSeats (Number?), price (Number?), and planeType (String?). A large number of lines connect fields from the input to the output, indicating a many-to-one mapping. The 'Preview' panel on the right shows a small icon of a document with three columns. Below it, a message says 'Sample data for payload is not well defined. Please review it.' with a 'Set sample data' button. The bottom navigation bar includes 'Sample data' (with a red exclamation mark), 'Script', and 'Mappings' (selected).

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

The screenshot shows the 'Output payload' section of the Mule DataWeave Transform tool. The payload is defined as 'Flights: Array<Object>' containing fields: airline (String?), flightCode (String?), fromAirportCode (String?), and toAirportCode (String?). The 'airline' field is currently selected and highlighted in blue. A context menu is open over this field, listing 'Data type actions': Create, Edit, Set, Detach, and Set Expression. The 'Set Expression' option is highlighted with a gray border.

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



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

Note: You learn to write DataWeave transformations later in the Development Fundamentals course.

The screenshot shows the MuleSoft Anypoint Studio interface for a "Transform DataWeave" card. The top navigation bar includes tabs for Configuration, Input, Output, and a three-dot menu. The main area is divided into three sections: Input, Transformation script, and Preview.

Input: Shows the structure of the payload, which is an array of objects representing flights. Each flight object has fields like flightCode, fromAirportCode, toAirportCode, departureDate, emptySeats, totalSeats, price, planeType, and airline.

Transformation script: Displays the following DataWeave script:

```
1 %dw 2.0
2 output application/json
3 ---
4 (payload map (value0, index0) -> {
5   flightCode: value0.code,
6   fromAirportCode: value0.origin,
7   toAirportCode: value0.destination,
8   departureDate: value0.departureDate,
9   emptySeats: value0.emptySeats,
10  totalSeats: value0.plane.totalSeats,
11  price: value0.price,
12  planeType: value0.plane.type,
13  airline: "american"
14 })
```

Preview: A preview section on the right shows a small icon of a document with a tree-like structure. Below it, a message says "Sample data for payload is not well defined. Please review it." and a "Set sample data" button.

At the bottom of the card, there are three tabs: Sample data (disabled), Script (selected), and Mappings.

Add sample data

44. Click the Set sample data button in the preview section.
45. In the computer's file explorer, return to the student files folder and locate the american-flights-example.json file in the examples folder.
46. Open the file in a text editor and copy the code.

47. Return to flow designer and paste the code in the sample data for payload section.

The screenshot shows the Mule DataWeave Transform tool interface. The top navigation bar includes 'Transform', 'DataWeave', 'Configuration' (selected), 'Input', and 'Output'. The 'Input' section on the left shows a tree view of the 'payload' object with various properties like 'ID', 'code', 'price', 'departureDate', 'origin', 'destination', 'emptySeats', 'plane', etc. The 'Sample data for payload (application/json)' section in the center contains the following JSON code:

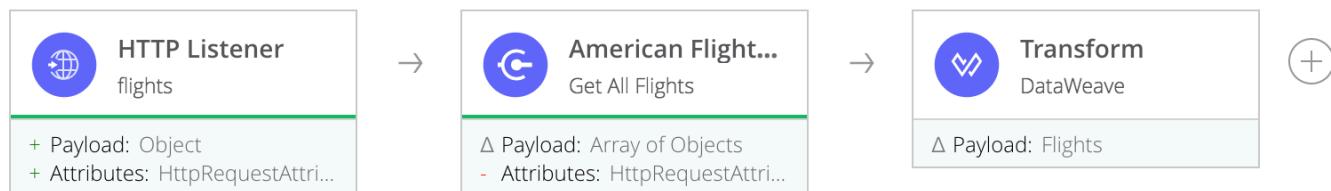
```
1 [{  
2     "ID": 1,  
3     "code": "ER38sd",  
4     "price": 400.00,  
5     "departureDate": "2016/03/20",  
6     "origin": "MUA",  
7     "destination": "SFO",  
8     "emptySeats": 0,  
9     "plane": {  
10        "type": "Boeing 737",  
11        "totalSeats": 150  
12    }, {  
13        "ID": 2,  
14        "code": "ER45if",  
15        "price": 345.99,  
16        "departureDate": "2016/02/11",  
17        "origin": "MUA",  
18        "destination": "LAX",  
19        "emptySeats": 52,  
20        "plane": {  
21            "type": "Boeing 777",  
22            "totalSeats": 300  
23        }  
24    }  
}
```

The 'Preview' section on the right shows the resulting transformed JSON output:

```
1 [ {  
2     "flightCode": "ER38sd",  
3     "fromAirportCode": "MUA",  
4     "toAirportCode": "SFO",  
5     "departureDate": "2016/03/20",  
6     "emptySeats": 0,  
7     "totalSeats": 150,  
8     "price": 400.00,  
9     "planeType": "Boeing 737",  
10    "airline": "american"  
11 }, {  
12     "flightCode": "ER45if",  
13     "fromAirportCode": "MUA",  
14     "toAirportCode": "LAX",  
15     "departureDate": "2016/02/11",  
16     "emptySeats": 52,  
17     "totalSeats": 300,  
18     "price": 345.99,  
19     "planeType": "Boeing 777",  
20     "airline": "american"  
21 }  
]
```

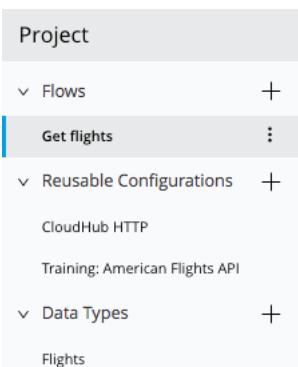
48. Look at the preview section, you should see a sample response for the transformation.

49. Close the card.



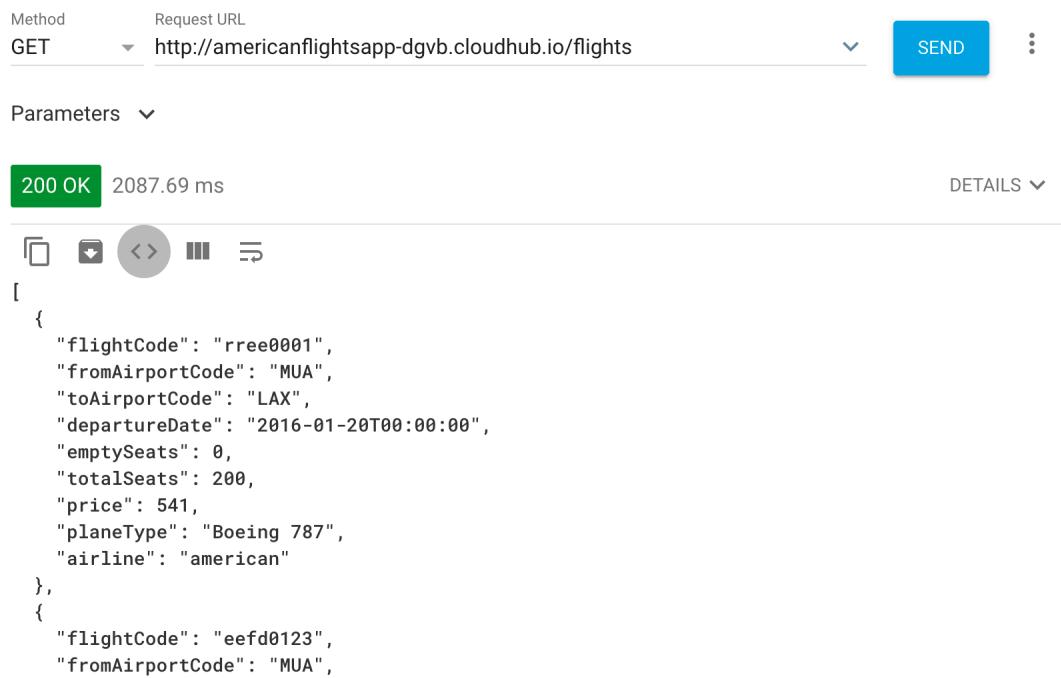
Locate the data type and configuration definitions

50. Locate the connector configuration and the new Flights data type in the project explorer.



Test the application

51. Run the project.
52. Return to Advanced REST Client and click Send to make another request to <http://americanflightsapp-xxxx.cloudhub.io/flights>; you should see all the flight data as JSON again but now with a different structure.



Method Request URL
GET <http://americanflightsapp-dgzb.cloudhub.io/flights>

SEND ⋮

Parameters ▼

200 OK 2087.69 ms DETAILS ▼

[
 {
 "flightCode": "rree0001",
 "fromAirportCode": "MUA",
 "toAirportCode": "LAX",
 "departureDate": "2016-01-20T00:00:00",
 "emptySeats": 0,
 "totalSeats": 200,
 "price": 541,
 "planeType": "Boeing 787",
 "airline": "american"
 },
 {
 "flightCode": "eefd0123",
 "fromAirportCode": "MUA",
 "toAirportCode": "LAX",
 "departureDate": "2016-01-20T00:00:00",
 "emptySeats": 0,
 "totalSeats": 200,
 "price": 541,
 "planeType": "Boeing 787",
 "airline": "american"
 }]

Stop the application

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

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

The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'DESIGN' selected, followed by 'Applications', 'Servers', 'Alerts', 'VPCs', and 'Load Balancers'. The main area has tabs for 'Deploy application' and 'Search Applications'. A table lists applications with columns for Name, Server, Status, and File. One row is selected for 'americanflightsapp-dgzb' on 'CloudHub' with a green 'Started' status. To the right, a detailed view shows the application name, status (green 'Started'), file path ('americanflightsapp-dgzb.jar'), and deployment details like runtime version 4.1.0, worker size 0.2 vCores, and 1 worker. Buttons for 'Manage Application', 'Logs', and 'Insight' are at the bottom.

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

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

This screenshot shows the Runtime Manager after changing the application status. The application 'americanflightsapp-dgzb' is now listed with a white circle icon and the text 'Undeployed'. The detailed view on the right shows the same application information but with a white circle icon and the text 'Undeployed'. The runtime version, worker size, and workers count remain the same.

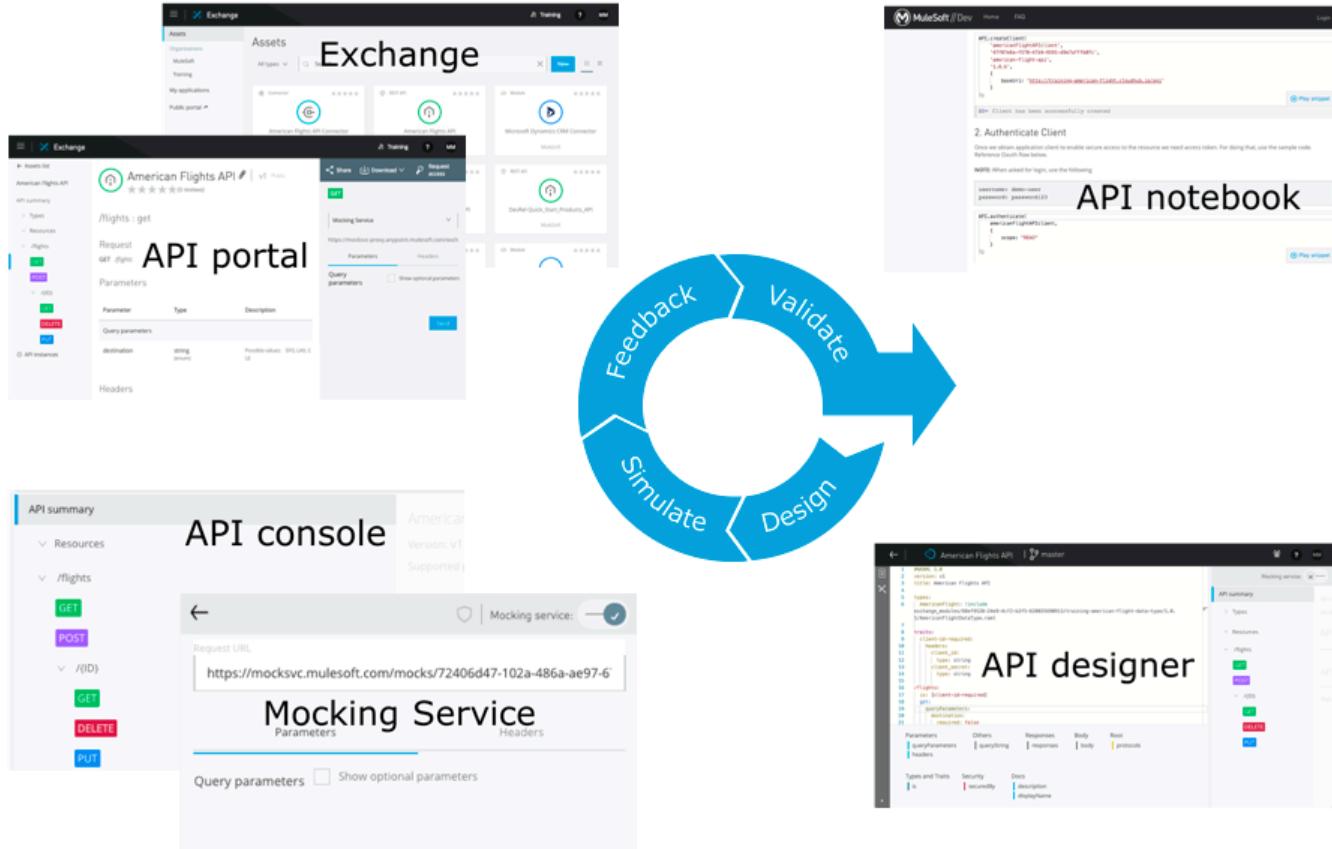
57. Close Runtime Manager.

58. Return to flow designer; you should see the application is not running.

The screenshot shows the Design Center interface. At the top, it says 'Design Center > American Flights App'. Below that is a toolbar with icons for back, forward, and search. The status bar shows 'Saved 7 minutes ago'. In the center, there's a green 'Run' button with a play icon, and to its right, a status message 'americanflightsapp-dgzb.c... Not running'. To the right of the status message is a 'Deploy' button.

59. Return to Design Center.

Module 3: Designing APIs



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

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

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

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

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

The screenshot shows the Anypoint Studio interface with the title bar "American Flights API" and "master". On the left, there's a sidebar with "Files" and a file named "american-flights-api.raml" selected. The main area displays the RAML code:

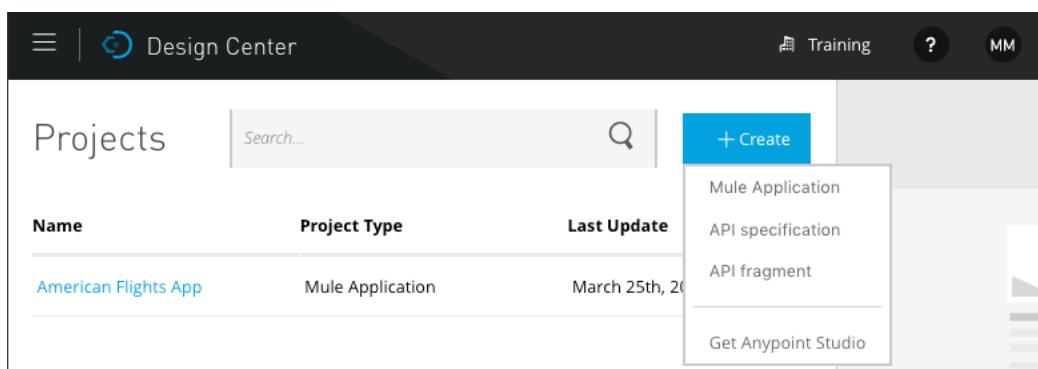
```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
5    get:
6      queryParameters:
7        destination:
8          required: false
9          enum:
10            - SFO
11            - LAX
12            - CLE
13    post:
14      /{ID}:
15        get:
16
17
```

Below the code editor are buttons for "Docs" (with "displayName" and "example" sub-options) and "Others" (with "facets" and "type" sub-options). To the right, the "API summary" pane shows the API structure:

- API summary
- Resources
 - /flights
 - GET
 - POST
 - /ID
 - GET

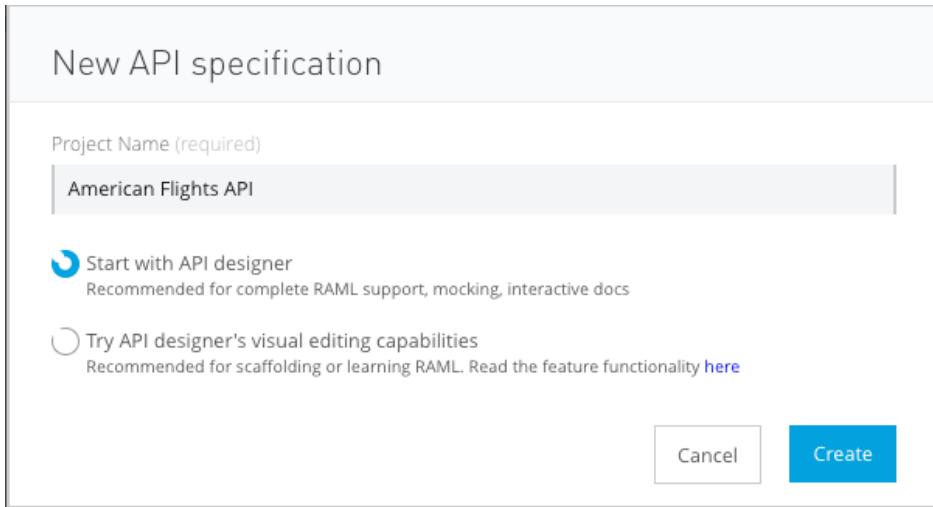
Create a new Design Center project

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

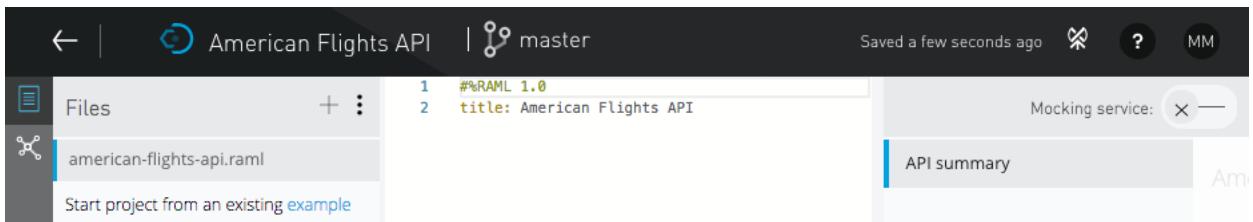


3. In the New API specification dialog box, set the project name to American Flights API.

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



5. In API designer, click the Hide these tips link in the popup window.
6. Review the three sections of API designer: the file browser, the editor, and the API console.



Add a RAML resource

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

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

View the API console

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

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

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, there's a code editor with the following RAML code:

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

To the right of the editor is the API designer shelf, which displays the following information:

- API summary**: Shows the title "American Flights API" and the endpoint "/flights".
- Resources**: A list containing the endpoint "/flights".

Add RAML methods

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

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

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

```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
5
```

The screenshot shows the API designer shelf with the following updated information:

- Types and Traits**: Options "is" and "type".
- Docs**: Options "description" and "displayName".
- Security**: Option "securedBy".
- Parameters**: Option "uriParameters".
- Methods**: Options "get", "put", and "post".

12. Click the get method in the shelf.

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

```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
5    get:
6    post:
7
```

Mocking service: [X](#)

API summary

Resources

/flights

GET

POST

Add a nested RAML resource

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

/ {ID}:

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

American Flights API | master

Files + :

american-flights-api.raml

```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
5    get:
6    post:
7
8  /{ID}:
9    get:
```

Saved a few seconds ago | ? | MM

Mocking service: X —

API summary

Resources

/flights

GET

POST

/ID

GET

Add an optional query parameter

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

```
1  %%RAML 1.0
2  title: American Flights API
3
4  /flights:
5    get:
6      queryParameters:
7        destination:
8    post:
9
10   /{ID}:
```

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

```
4  /flights:
5    get:
6      queryParameters:
7        destination:
8          required: false
9          enum:
10         - SFO
11         - LAX
12         - CLE
```

Try to call an API method using the API console

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

32. Review the information.

The screenshot shows a RAML API definition for a `/flights` endpoint. At the top, there's a header bar with a back arrow, the text "Mocking service:", and a close button. Below the header, the endpoint path `/flights : get` is displayed next to a blue "Try it" button. A "Request" section follows, showing the method `GET /flights`. Under "Parameters", there's a "Properties" section with a "Query parameters" entry. This entry has a key `destination` with a type of `string(enum)`. Below this, the possible values `SFO, LAX, CLE` are listed. Another blue "Try it" button is located at the bottom right of this section.

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

The screenshot shows a RAML interface with a "Request URL" input field highlighted in red. Below the field, a message says "Fill the URI parameters before making a request". There are tabs for "Parameters" and "Headers", with "Parameters" currently selected. Under "Parameters", there's a "Query parameters" section and a checkbox labeled "Show optional parameters". At the bottom, a red message "Request URL is invalid." is displayed next to a blue "Send" button.

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

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

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

The screenshot shows the Anypoint Platform interface. On the left, the 'Files' panel displays the 'american-flights-api.raml' file. The code editor shows the following RAML definition:

```
1  #RAML 1.0
2  baseUrl: https://mocksvc.mulesoft.com/mocks/7a9130df
3  -b911-4d24-a66d-6e906da868b8 #
4  title: American Flights API
5
6  /flights:
7    get:
8      queryParameters:
9        destination:
10       required: false
11       enum:
12         - SFO
13         - LAX
14         - CLE
15
16    /{ID}:
17      get:
```

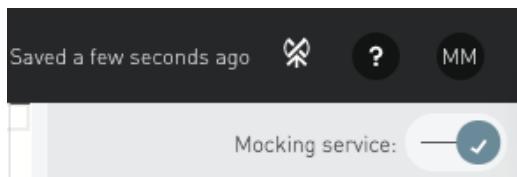
On the right, the 'Mocking service' slider is turned on. The 'Request URL' field contains 'https://mocksvc.mulesoft.com/mocks/7a'. The 'Parameters' tab is selected, showing a dropdown menu with 'SFO'. Below the request area, a response is displayed:

200 OK 123.10 ms

```
{ "message": "RAML had no response information for application/json" }
```

Turn on the mocking service

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



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

```
1  #RAML 1.0
2  baseUrl: https://mocksvc.mulesoft.com/mocks/f02c935c-66b1-4a68-9bf2-d7beb3affe53 #
3  title: American Flights API
```

Test the /flights:get resource

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

The screenshot shows the MuleSoft API Console interface. At the top, there's a back arrow, a shield icon, and a 'Mocking service:' toggle switch which is turned on (indicated by a checkmark). Below that is a 'Request URL' field containing 'https://mocksvc.mulesoft.com/mocks/7a'. There are two tabs: 'Parameters' (which is selected) and 'Headers'. Under 'Parameters', there's a 'Query parameters' section with a checkbox labeled 'Show optional parameters' which is unchecked. A large blue 'Send' button is centered below these sections. Below the send button, the response status is shown as '200 OK' in a green box with a timestamp of '119.20 ms'. To the right of the status is a dropdown arrow. Below the status are several small icons: a copy icon, a refresh icon, a link icon, and a list icon. The main content area displays a JSON response: { "message": "RAML had no response information for application/json" }.

6. Select the Show optional parameters checkbox.
7. In the destination text field, select SFO and click Send; you should get the same response.

Test the /flights/{ID} resource

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

The screenshot shows the MuleSoft API Console interface. At the top, there's a 'Mocking service:' toggle switch which is turned on (indicated by a checkmark). Below that is a 'API summary' section. Under 'API summary', there's a 'Resources' section with a 'flights' item. Under 'flights', there are two methods: 'GET' and 'POST'. Below 'flights' is another 'Resources' section with a '{ID}' item. Under '{ID}', there is a single 'GET' method.

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

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

The screenshot shows the 'Mocking service' interface. At the top, there's a back arrow, a shield icon, and the text 'Mocking service:'. Below that, the 'Request URL' field contains 'https://mocksvc.mulesoft.c' (note the trailing 'c'). A red error message 'Fill the URI parameters before making a requ...' is displayed below the URL field. There are tabs for 'Parameters' and 'Headers', with 'Parameters' being active. Under 'URI parameters', there's a single input field labeled 'ID*' containing 'ID*'. At the bottom right is a blue 'Send' button with a red message 'Request URL is invalid.' to its left.

11. In the ID text box, enter a value of 10.

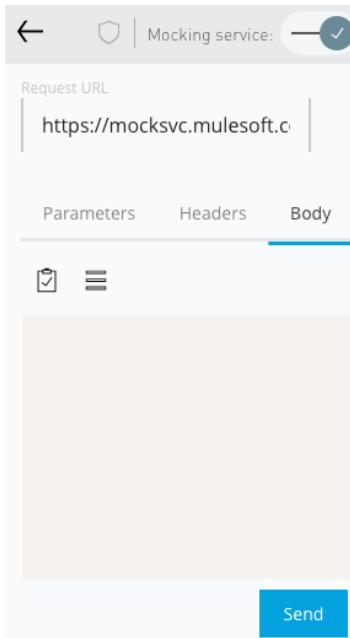
12. Click the Send button.

13. Look at the response; you should get the same default response with a 200 status code, a content-type of application/json, and the general RAML message placeholder.

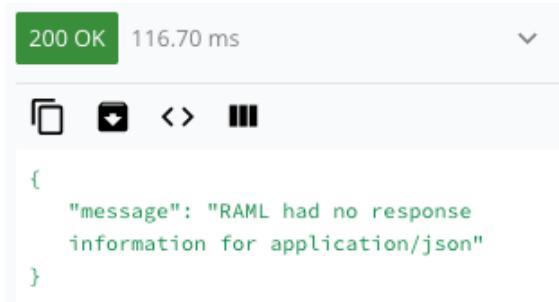
The screenshot shows the 'Mocking service' interface after entering '10' in the 'ID*' field. The 'Send' button is now greyed out. Below the interface, a summary shows a green '200 OK' button and '174.90 ms' response time. At the bottom, there are icons for copy, download, and refresh. The response body is shown as a JSON object: { "message": "RAML had no response information for application/json" }.

Test the /flights:post resource

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



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



Walkthrough 3-3: Add request and response details

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

- Use API fragments from Exchange.
 - Add a data type and use it to define method requests and responses.
 - Add example JSON requests and responses.
 - Test an API and get example responses.

The screenshot shows a GitHub repository for the American Flights API. The repository has a single commit, 'Mocking service', saved 5 minutes ago. The code is written in RAML 1.0 and defines an API for flight data. It includes endpoints for getting flights by ID and destination, and for creating new flights. The test results show a successful 200 OK response with a duration of 125.69 ms. The test data is an array of flight records, each with an ID, code, price, departure date, origin, destination, and seat information.

```
#%RAML 1.0
baseUri: https://mocksvc.mulesoft.com/mocks/6822ede5-6246-4462-a380-6a64a9d4e1c2 #
title: American Flights API

types:
| AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flight-data-type.raml

/flights:
get:
queryParameters:
destination:
required: false
enum:
- SFO
- LAX
- CLE
responses:
200:
body:
application/json:
type: AmericanFlight[]
example: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flights-example/1.0.1/AmericanFlightsExample.raml

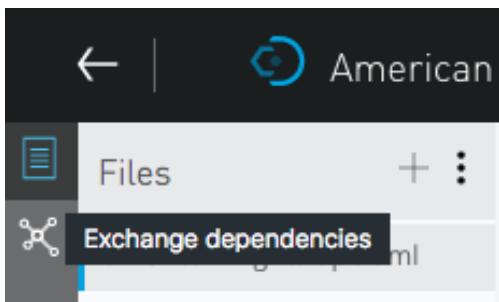
post:
body:
application/json:
type: AmericanFlight
example: !include examples/AmericanFlightNoIDEExample.raml
responses:
201:
body:
```

200 OK 125.69 ms

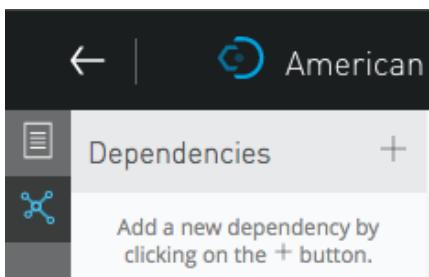
```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
},
-1: {
  "ID": 2,
  "code": "ER45if",
  "price": 540.99,
  "departureDate": "2017/07/27",
  "origin": "SFO",
  "destination": "ORD",
  "emptySeats": 54,
  "plane": {
    "type": "Boeing 777",
    "totalSeats": 300
  }
}]
```

Add data type and example fragments from Exchange

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



- Click the Add button.



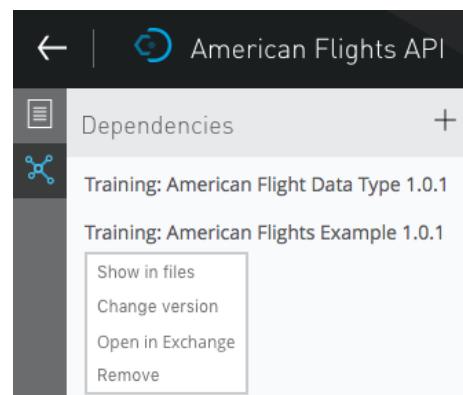
- In the Consume API Fragment dialog box, select the Training: American Flight Data Type and the Training: American Flights Example.

The screenshot shows a modal dialog titled 'Consume API Fragment'. At the top are filter buttons for 'All' and a search bar. The main area is a table with columns: Name, Date modified, Rating, Created by, and Business group. Two items are selected: 'Training: American Flight Data Type' and 'Training: American Flights Example'. The table rows are as follows:

Name	Date modified	Rating	Created by	Business group
FHIR DSTU-2 Data Types	Mar 10, 2018	★★★★★	ND Nial Darbey	MuleSoft
REST Connect Library	Feb 19, 2018	★★★★★	MO MuleSoft Organization	MuleSoft
<input checked="" type="checkbox"/> Training: American Flight Data Type	Feb 19, 2018	★★★★★	MO MuleSoft Organization	MuleSoft
<input checked="" type="checkbox"/> Training: American Flights Example	Feb 19, 2018	★★★★★	MO MuleSoft Organization	MuleSoft
Training: Cacheable Trait	Feb 19, 2018	★★★★★	MO MuleSoft Organization	MuleSoft
Training: OAuth2.0 Security Scheme	Feb 19, 2018	★★★★★	MO MuleSoft Organization	MuleSoft

At the bottom right are 'Cancel' and 'Add 2 dependencies' buttons.

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



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

```

1  #%%RAML 1.0 DataType
2
3  type: object
4  properties:
5    ID?: integer
6    code: string
7    price: number
8    departureDate: string
9    origin: string
10   destination: string
11   emptySeats: integer
12   plane:
13     type: object
14     required: false
15     properties:
16       type: string
17       totalSeats: integer
18

```

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



Define an AmericanFlight data type for the API

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

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

```

1  #%%RAML 1.0
2  baseUri: https://mocksvc.mulesoft.com/mocks/3220238a-17c6-4e31-b5fa-413e8129e12b #
3  title: American Flights API
4
5  types:
6    AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/tr
7
8  /flights:
9    get:

```

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

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

```
8  /flights:  
9    get:  
10   queryParameters:  
11     destination:  
12       required: false  
13       enum:  
14         - SF0  
15         - LAX  
16         - CLE  
17   responses:  
18     200:  
19       body:  
20         application/json:  
21           type: AmericanFlight
```

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

```
17   responses:  
18     200:  
19       body:  
20         application/json:  
21           type: AmericanFlight[]
```

Add an example response for the /flights:get method

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

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the file browser displays a project structure under 'Files'. It includes 'exchange_modules' (with 'training-american-flight-data-type' and '1.0.1' subfolders), 'training-american-flights-example' (with '1.0.1' subfolder), and 'american-flights-api.raml'. The 'AmericanFlightsExample.raml' file is currently selected and its content is displayed on the right. The content of the file is as follows:

```
1  %%RAML 1.0 NamedExample  
2  value:  
3  
4  -  
5  ID: 1  
6  code: ER38sd  
7  price: 400  
8  departureDate: 2017/07/26  
9  origin: CLE  
10 destination: SF0  
11 emptySeats: 0  
12 plane:  
13   type: Boeing 737  
14   totalSeats: 150  
15 -  
16 ID: 2  
17 code: ER45if  
18 price: 540.99  
19 departureDate: 2017/07/27  
20 origin: SF0  
21 destination: ORD  
22 emptySeats: 54  
23 plane:  
24   type: Boeing 777  
25   totalSeats: 300
```

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

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

```

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

```

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

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

Parameter	Type	Description
item.ID	integer	
item.code (required)	string	

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

The screenshot shows a REST API documentation interface. At the top, there's a green header bar with the status code "200". Below it, a navigation bar has tabs for "Type" (selected), "Examples" (highlighted in blue), and other options. The main content area displays a JSON array of flight objects:

```
[{"ID": 1, "code": "ER38sd", "price": 400, "departureDate": "2017/07/26", "origin": "CLE", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 2, "code": "ER45if", "price": 540.99, "departureDate": "2017/07/27", "origin": "SFO", "destination": "ORD", "emptySeats": 54, "plane": {"type": "Boeing 777", "totalSeats": 300}}]
```

Below the JSON, there's a table for parameters:

Parameter	Type	Description
Array of AmericanFlight items		
item. ID	integer	
item. code (required)	string	

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

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

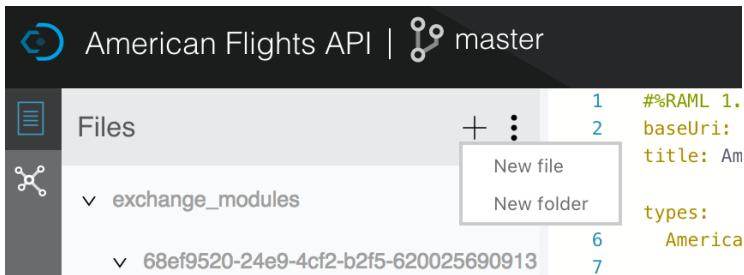
28. In the editor, indent under the `/{ID}` resource get method.

29. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

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

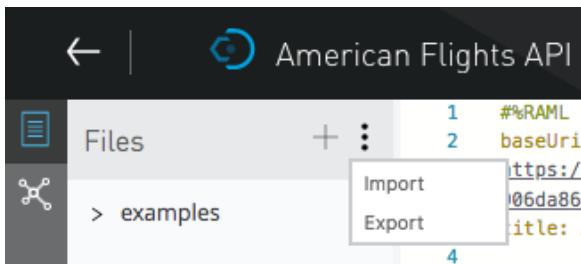
Define an example response for the /flights:get method in a new folder

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



31. In the Add new folder dialog box, set the name to examples and click Create.

32. In the file browser, click the menu button and select Import.



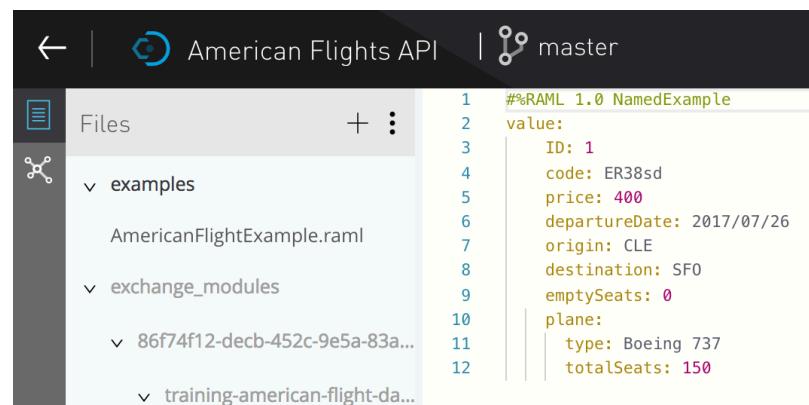
33. In the Import dialog box, leave File or ZIP selected and click the Choose file button.

34. Navigate to your student files and select the AmericanFlightExample.raml file in the examples folder.

35. In the Import dialog box, click Import; you should see the new file in API designer.

36. Review the code.

37. In the file browser, drag AmericanFlightExample.raml into the examples folder.



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

38. Return to american-flights-api.raml.

39. In the editor, go to a new line after the type declaration in {ID}:/get (at the same indentation).

40. In the shelf, click example.

41. Add an include statement and include the example in examples/AmericanFlightExample.raml.

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

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

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

The screenshot shows the API console interface for the /{ID}:get method. The top navigation bar has tabs for 'Type' and 'Examples'. The 'Type' tab is selected, showing the response schema:

```
{  
  "ID": 1,  
  "code": "ER38sd",  
  "price": 400,  
  "departureDate": "2017/07/26",  
  "origin": "CLE",  
  "destination": "SFO",  
  "emptySeats": 0,  
  "plane": {  
    "type": "Boeing 737",  
    "totalSeats": 150  
  }  
}
```

Below the schema, there are two tables:

Parameter	Type	Description
ID	integer	
code (required)	string	

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

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

The screenshot shows the Mocking service interface. The Request URL is <https://mocksvc.mulesoft.com/m>. A parameter **ID*** is set to **10**. The **Send** button is visible. The response status is **200 OK** with a response time of **384.00 ms**. The response body is a JSON object:

```
{  
  "ID": 1,  
  "code": "ER38sd",  
  "price": 400,  
  "departureDate": "2017/07/26",  
  "origin": "CLE",  
  "destination": "SFO",  
  "emptySeats": 0,  
  "plane": {  
    "type": "Boeing 737",  
    "totalSeats": 150  
  }  
}
```

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

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

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

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

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

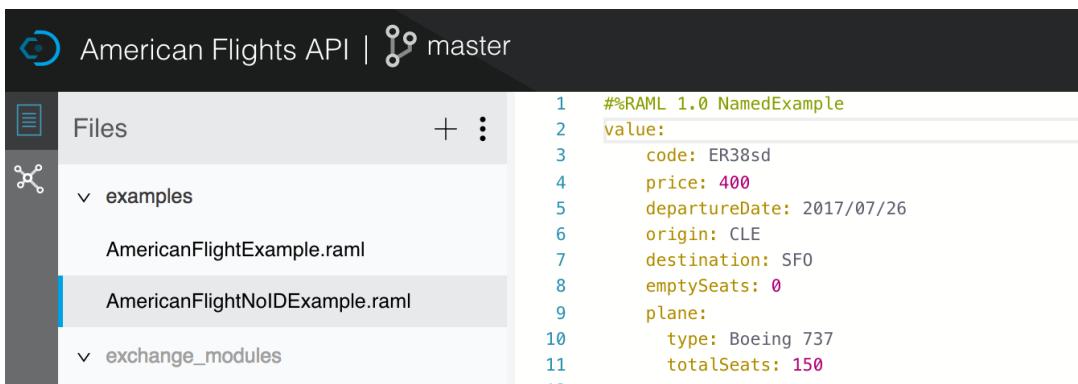
49. In the Add new file dialog box, set the following values:

- Version: RAML 1.0
- Type: Example
- File name: AmericanFlightNoIDExample.raml

50. Click Create.

51. Delete any code in the new file and then paste the code you copied.

52. Delete the line of code containing the ID.



The screenshot shows the MuleSoft Anypoint Studio interface. At the top, it says "American Flights API | master". Below that is a sidebar with icons for files, projects, and logs. The main area shows a file tree under "Files": "examples/AmericanFlightExample.raml" and "examples/AmericanFlightNoIDExample.raml" (which is selected). Under "examples", there is also a folder "exchange_modules". To the right of the file tree is the RAML code for "AmericanFlightNoIDExample.raml".

```
1  #%RAML 1.0 NamedExample
2  value:
3    code: ER38sd
4    price: 400
5    departureDate: 2017/07/26
6    origin: CLE
7    destination: SFO
8    emptySeats: 0
9    plane:
10   type: Boeing 737
11   totalSeats: 150
12
```

53. Return to american-flights-api.raml.

54. In the post method, go to a new line under type and add an example element.

55. Use an include statement to set the example to examples/AmericanFlightNoIDExample.raml.

```
24  post:
25    body:
26      application/json:
27        type: AmericanFlight
28        example: !include examples/AmericanFlightNoIDExample.raml
```

Specify an example response for the /flights:post method

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

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

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

58. In the shelf, click example.

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

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

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

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

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

/flights : post

Try it

Request

POST https://mocksvc.mulesoft.com/mocks/6822ede5-6246-4462-a380-6ae4a9d4e1c2/flights

Body

Type AmericanFlight

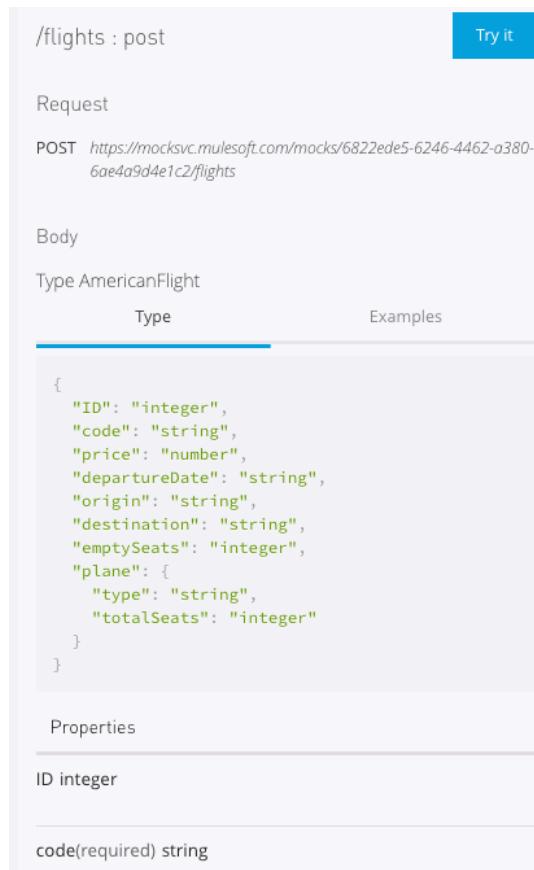
Type Examples

```
{  
    "ID": "integer",  
    "code": "string",  
    "price": "number",  
    "departureDate": "string",  
    "origin": "string",  
    "destination": "string",  
    "emptySeats": "integer",  
    "plane": {  
        "type": "string",  
        "totalSeats": "integer"  
    }  
}
```

Properties

ID integer

code(required) string



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

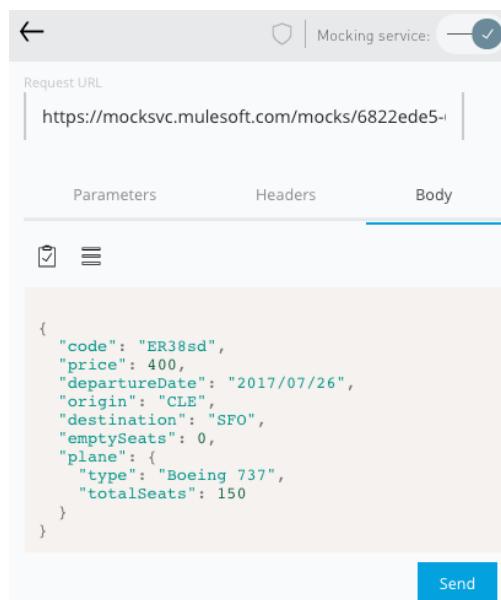
← Mocking service:

Request URL: https://mocksvc.mulesoft.com/mocks/6822ede5-6246-4462-a380-6ae4a9d4e1c2/flights

Parameters Headers Body

```
{  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2017/07/26",  
    "origin": "CLE",  
    "destination": "SFO",  
    "emptySeats": 0,  
    "plane": {  
        "type": "Boeing 737",  
        "totalSeats": 150  
    }  
}
```

Send



63. Click the Send button; you should now get a 201 response with the example message.

A screenshot of a REST API testing interface. At the top, a green bar displays "201 Created" and "494.58 ms". Below this, there are icons for copy, refresh, and navigation. The main area shows a JSON response:

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

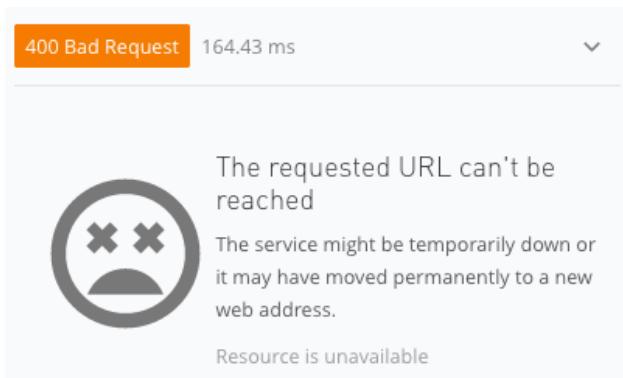
64. In the body, remove the emptySeats properties.

A screenshot of a REST API testing interface. The "Body" tab is selected. The request body contains the following JSON:

```
{
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
}
```

At the bottom right is a blue "Send" button.

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



66. Add the emptySeats property back to the body.

Walkthrough 3-4: Add an API to Anypoint Exchange

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

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

The screenshot shows the Anypoint Exchange interface with the 'American Flights API' asset selected. The left sidebar shows the API structure: 'American Flights API' (with a house icon), 'API summary', 'Types' (selected), 'Resources' (expanded), and 'flights' (expanded). Under 'flights', there are 'GET' and 'POST' methods, and a link to '/{ID}'. The main content area displays the 'American Flights API' details, including its description ('The American Flights API is a system API for operations on the american table in the training database.'), supported operations (Get all flights, Get a flight with a specific ID, Add a flight, Delete a flight, Update a flight), reviews (none), and asset versions for v1 (version 1.0.1 with one instance: Mocking Service). It also lists dependencies: 'Training: American Flights Example' (version 1.0.1) and 'Training: American Flight Data Type' (version 1.0.1).

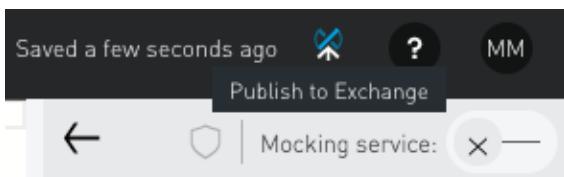
Remove the `baseUri` by turning off the mocking service

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

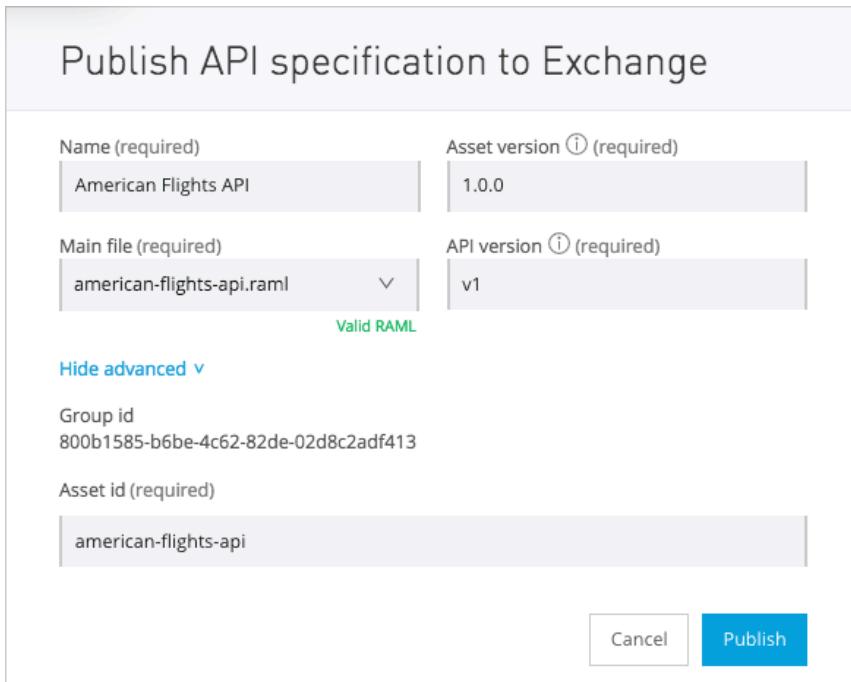
```
1  #%RAML 1.0
2  title: American Flights API
3
```

Publish the API to Anypoint Exchange from API designer

3. Click the Publish to Exchange button.



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



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

```

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

Locate your API in Anypoint Exchange

11. Return to Design Center.

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

The screenshot shows the MuleSoft Exchange interface. The left sidebar has 'Assets' selected. The main area is titled 'Assets' with a search bar and a 'New' button. Three items are listed: 'American Flights API Connector' (Connector), 'American Flights API' (REST API), and 'Microsoft Dynamics CRM Connector' (Module). All three have a green circular icon with a white house-like symbol.

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

The screenshot shows the MuleSoft Exchange interface with 'REST APIs' selected from the dropdown in the top navigation bar. The main area is titled 'Assets' with a search bar and a 'New' button. It displays results for REST APIs, showing three items: 'American Flights API', 'DevRel-Quick_Start_Products_API', and 'Customer API for Visual Editing'. Each item has a green circular icon with a white house-like symbol.

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

The screenshot shows the MuleSoft Exchange interface with 'MuleSoft' selected from the dropdown in the top navigation bar. The main area is titled 'Assets' with a search bar and a 'New' button. It displays results for REST APIs, showing three items: 'DevRel-Quick_Start_Products_API', 'Customer API for Visual Editing', and 'Account Information (AISP) API - RAML Definition'. Each item has a green circular icon with a white house-like symbol.

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

Review the auto-generated API portal

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

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

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

The screenshot shows the MuleSoft Exchange interface. At the top, there's a navigation bar with a menu icon and the word "Exchange". Below it is a sidebar with a back arrow labeled "Assets list" and a selected item "American Flights API". The main content area displays the API entry for "American Flights API". It features a green circular icon with a house symbol, a star rating of 5 stars with "(0 reviews)", and a "Rate and review" button. To the right of the icon, the API name is followed by a pencil icon and "v1 ▾".

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

The screenshot shows the "Asset versions for v1" section. It has a header with "Version" and "Instances" columns. A single row is listed: "1.0.0" under Version, "Mocking Service" under Instances, and three vertical dots under the URL column.

Version	Instances	
1.0.0	Mocking Service	⋮

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

The screenshot shows the "API instances" section. On the left, the sidebar is expanded to show "API instances" as the active item. The main content area shows a table of API instances. The table has columns for "Instances", "Environment", "URL", and "Visibility". One instance is listed: "Mocking Service" with "https://mocksvc-proxy.anypoint.mulesoft.com/exchange/800b1585-b6be-4c62-82de-02d8c2adf413/american-flights-api/1.0.0" under URL and "Public" under Visibility. There's also a "+ Add new instance" button.

Instances	Environment	URL	Visibility
Mocking Service		https://mocksvc-proxy.anypoint.mulesoft.com/exchange/800b1585-b6be-4c62-82de-02d8c2adf413/american-flights-api/1.0.0	Public

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

23. Select AmericanFlight and review the information.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a navigation sidebar with options like 'Assets list', 'American Flights API', 'API summary', 'Types' (selected), 'AmericanFlight', 'Resources', '/flights' (selected), and 'API instances'. Under '/flights', there are 'GET' and 'POST' buttons. The main content area displays the 'American Flights API' page with a green house icon, a rating of 0 reviews, and a title 'Type AmericanFlight'. It shows a JSON schema for the 'AmericanFlight' type and a table of parameters:

Parameter	Type	Description
ID	integer	
code (required)	string	

Test the API in its API portal in Exchange

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

The screenshot shows the MuleSoft Exchange interface with the API console open for the '/flights' GET method. The left sidebar is identical to the previous screenshot. The main area shows the endpoint '/flights : get' and a 'Request' section with 'GET /flights'. Below it is a 'Parameters' section with a table:

Parameter	Type	Description
Query parameters		
destination	string (enum)	Possible values: SFO, LAX, CLE

To the right, the API console shows a 'GET' button, a dropdown for 'Mocking Service' (set to 'Mocking Service'), a URL field with 'https://mocksvc-proxy.anypoint.mulesoft.com/exc', and tabs for 'Parameters' (selected) and 'Headers'. Under 'Parameters', there's a 'Query parameters' section with a dropdown set to 'LAX' and a 'Send' button.

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

The screenshot shows the MuleSoft Anypoint Exchange API designer interface. At the top, there is a green button labeled "GET". Below it, the URL is set to "Mocking Service" and the endpoint is "https://mocksvc-proxy.anypoint.mulesoft.com/exc". The "Parameters" tab is selected, showing a dropdown menu with "LAX" as the value for the query parameter. To the right of the dropdown is a checkbox labeled "Show optional parameters". A "Send" button is located below the dropdown. The response section shows a "200 OK" status with a response time of "582.08 ms". The "Details" link is expanded, showing a JSON response:

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
},
-1: {
  "ID": 2,
  "code": "ER45if".
}
```

Add information about the API

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

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

The American Flights API is a system API for operations on the **american** table** in the training database.
Supported operations
Get all flights
Get a flight with a specific ID
Add a flight

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

The American Flights API is a system API for operations on the **american** table** in the training database.
Supported operations

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

The American Flights API is a system API for operations on the **american** table** in the training database.
Supported operations

35. Select Get all flights and click the bulleted list button.

36. Repeat for the other operations.

The American Flights API is a system API for operations on the **american** table** in the training database.
Supported operations

- Get all flights
- Get a flight with a specific ID
- Add a flight

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

The screenshot shows the Exchange API documentation editor interface. On the left, there's a sidebar with options like 'Assets list', 'American Flights API' (selected), '+ Add new page', 'API summary', '+ Add terms and conditions', and 'API instances'. The main content area has a title 'American Flights API' with a green edit icon and version 'v1'. Below the title is a toolbar with various icons. The 'Visual' tab is selected. The content area contains the following text:

The American Flights API is a system API for operations on the **american table** in the *training database*.

Supported operations

- Get all flights
- Get a flight with a specific ID
- Add a flight

At the bottom right are 'Discard changes' and 'Save as draft' buttons.

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

The screenshot shows the Exchange API documentation editor interface after saving as a draft. The top bar includes the MuleSoft logo, a question mark icon, and a user profile icon. The main content area shows the published state of the API documentation. A message at the top says 'This is a draft that has not been published yet.' with a pencil icon. There are 'Exit Draft', 'Publish', and 'Edit' buttons. The content area is identical to the previous screenshot, showing the API title, supported operations, and the 'Save as draft' buttons at the bottom.

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

The screenshot shows the Mule Exchange interface. On the left, there's a sidebar with navigation links like 'Assets list', 'American Flights API', 'API summary', 'Types', 'Resources', and '/flights'. The main content area displays the 'American Flights API' details. It includes a green icon of a house with an 'i' inside, the API name, a version 'v1', a star rating of 0 reviews, and a 'Rate and review' link. Below this, a description states: 'The American Flights API is a system API for operations on the **american** table in the *training database*'. Under 'Supported operations', there's a list: 'Get all flights', 'Get a flight with a specific ID', and 'Add a flight'. On the right side, there's an 'Overview' panel with sections for 'Type' (REST API), 'Created By' (Max Mule), and 'Published On' (Nov 18, 2017). There are also 'Share', 'Download', and 'Edit' buttons at the top right.

Modify the API and publish the new version to Exchange

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

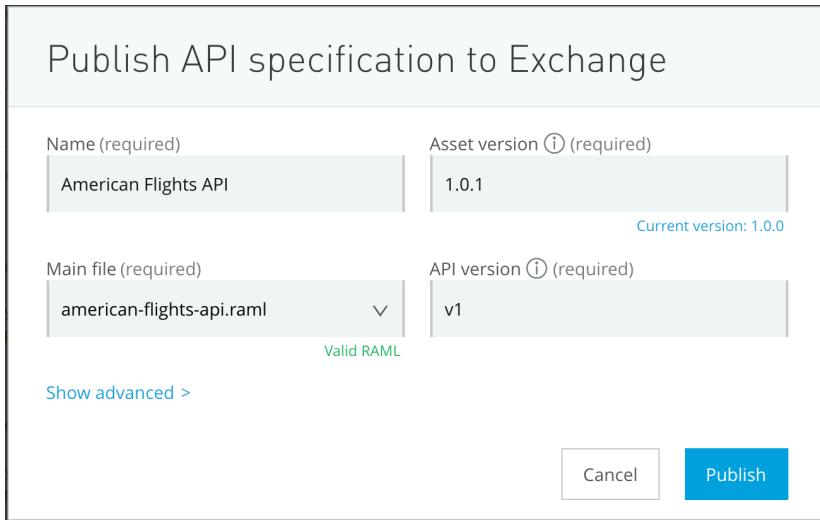
```

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

```

45. Click the Publish to Exchange button.

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



47. Click Publish.

48. In the Publish API specification to Exchange dialog box, click Exchange; Exchange should open in a new browser tab.

49. Locate the asset versions listed for the API; you should see both asset versions of the API listed with an associated API instance using the mocking service for the latest version.

Asset versions for v1	
Version	Instances
1.0.1	Mocking Service
1.0.0	

50. Click the Edit button.

51. Add two new operations that delete a flight and update a flight.

A screenshot of a rich text editor interface. The toolbar includes buttons for bold (B), italic (I), code (code), list (list), table (table), header (H), and other document-related functions. To the right of the toolbar are 'Markdown' and 'Visual' buttons. The content area contains the following text:

The American Flights API is a system API for operations on the **american** table** in the _training database_.

Supported operations

- Get all flights
- Get a flight with a specific ID
- Add a flight
- Delete a flight
- Update a flight

52. Click Save as draft and then Publish.

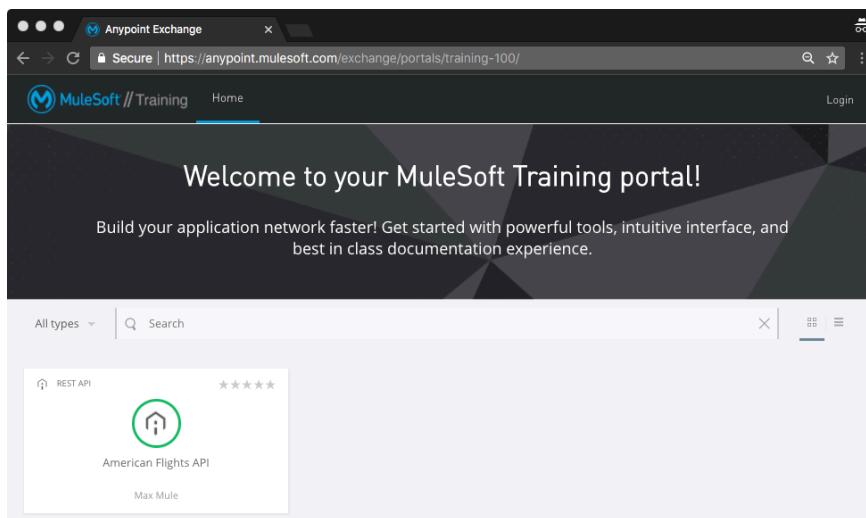
The screenshot shows the MuleSoft Exchange interface with the following details:

- Left Sidebar:** Assets list, American Flights API, API summary, Types, Resources, /flights (highlighted), GET, POST, /{ID}, API instances.
- Header:** Exchange, Training, Help, MM.
- Asset Details:** American Flights API, v1, 0 reviews, Rate and review. Description: "The American Flights API is a system API for operations on the **american** table in the *training* database." Supported operations: Get all flights, Get a flight with a specific ID, Add a flight, Delete a flight, Update a flight.
- Right Panel:**
 - Overview:** Type: REST API, Created By: Max Mule, Published On: Nov 18, 2017, Visibility: Private.
 - Asset versions for v1:** Version 1.0.1 (Mocking Service), Version 1.0.0.
 - Tags:** + Add a tag.
 - Dependencies:** Training: American Flights Example 1.0.1 (RAML Fragment), Training: American Flight Data Type 1.0.1 (RAML Fragment).

Walkthrough 3-5: Share an API

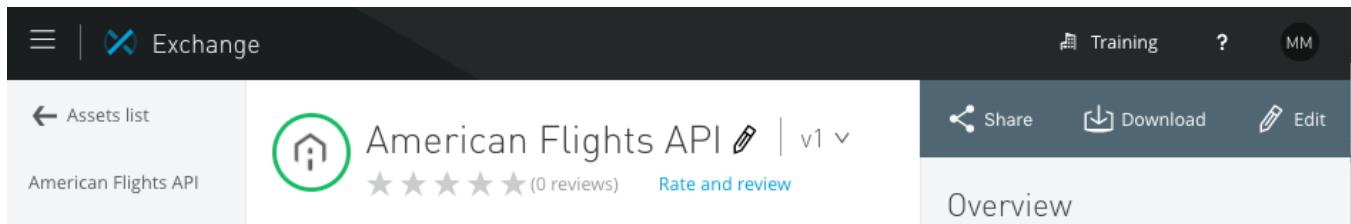
In this walkthrough, you share an API with both internal and external developers to locate, learn about, and try out the API. You will:

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

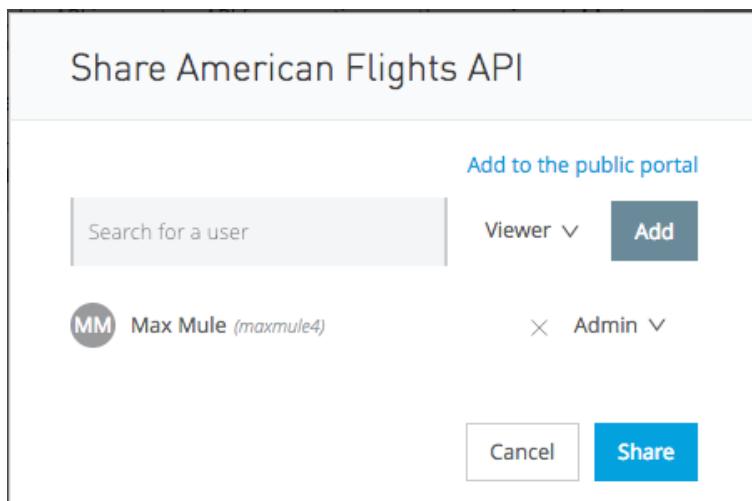


Share the API in the private Exchange with others

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



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



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

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

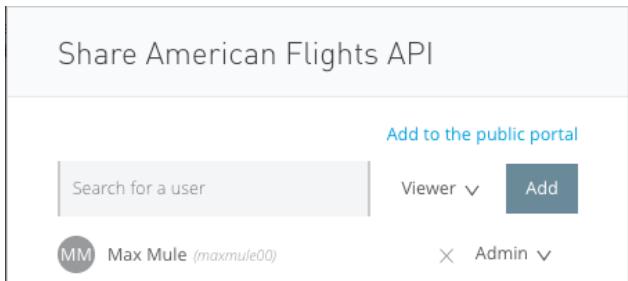
5. Click Cancel.
6. In the left-side navigation, click Assets list.
7. In the left-side navigation, select the name of your organization.
8. Click your American Flights API.

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

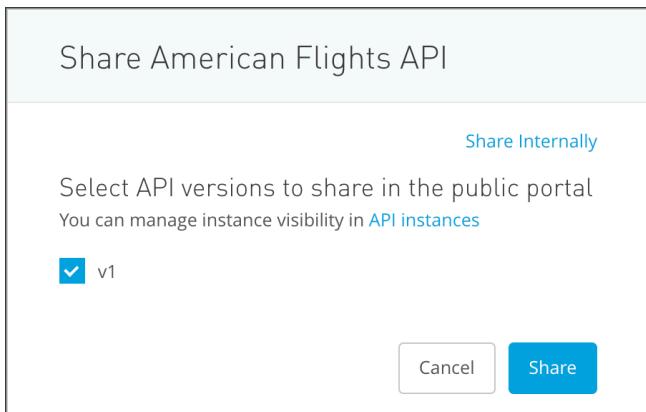
A screenshot of the MuleSoft Exchange interface. The left sidebar shows 'Assets' selected, with other options like 'Organizations', 'MuleSoft', 'Training', 'My applications', and 'Public portal' available. The main area is titled 'Assets' and shows two items: 'American Flights API Connector' and 'American Flights API'. Both items have a green circular badge with a white 'i' over them, indicating they are shared. The connector was created by 'Max Mule' and has a 5-star rating. The API also has a 5-star rating and was created by 'Max Mule'. A 'New' button is visible at the top right of the list.

Create a public API portal

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



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



Explore the API in the public portal

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

A screenshot of the MuleSoft Exchange interface. The left sidebar shows a navigation tree with "Assets" selected, followed by "Organizations", "MuleSoft", "Training", "My applications", and "Public portal". The main area is titled "Assets" and shows two items: "American Flights API Connector" and "American Flights API". Both items have a "Connector" icon, five-star ratings, and the name "American Flights API" followed by "Max Mule". The "New" button is visible in the top right corner of the main area.

15. Review the public portal that opens in a new browser tab.

16. Look at the URL for the public portal.

The screenshot shows a web browser with two tabs open. The active tab is titled 'Secure | https://anypoint.mulesoft.com/exchange/portals/training-100/'. The page content is a 'Welcome to your developer portal!' message with a subtext: 'Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience.' Below this, there's a search bar and a card for the 'American Flights API'. The card includes a 'REST API' icon, a 5-star rating, a house icon, the API name 'American Flights API', and the developer 'Max Mule'.

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

The screenshot shows the 'American Flights API' details page. On the left, there's a sidebar with navigation links: 'Assets list', 'American Flights API' (which is highlighted), 'API summary', 'Types' (with a 'GET' method listed), 'Resources' (with a 'POST' method listed), '/flights' (with a 'GET' method listed), and 'API instances'. The main content area displays the API title 'American Flights API | v1', its rating (0 reviews), and a 'Rate and review' button. It also includes a 'GET' method under 'Supported operations' with a list of operations: 'Get all flights', 'Get a flight with a specific ID', 'Add a flight', 'Delete a flight', and 'Update a flight'. A 'Rate and review' button is also present in this section.

18. In the left-side navigation, click the POST method for the flights resource.

19. Review the body and response information.

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

The screenshot shows the MuleSoft Anypoint Platform interface. On the left, the navigation bar includes 'Home' and 'My applications'. The main area displays the 'American Flights API' with a green icon and a rating of 5 stars (0 reviews). The API summary shows the endpoint '/flights : post'. The 'Request' section details a POST request to '/flights'. The 'Body' section specifies the type 'AmericanFlight' and provides a JSON schema example:

```
{
  "ID": "integer",
  "code": "string",
  "price": "number",
  "departureDate": "string",
  "origin": "string",
  "destination": "string",
  "emptySeats": "integer",
  "plane": {
    "type": "string",
    "totalSeats": "integer"
  }
}
```

The 'Mocking Service' dropdown is set to 'Mocking Service' with the URL <https://mocksvc-proxy.anypoint.mulesoft.com/>. The 'Parameters', 'Headers', and 'Body' tabs are visible. A 'Send' button is present. The response section shows a 201 Created status with a timestamp of 271.27 ms and a 'Details' link. The response body contains a message: '{ "message": "Flight added (but not really)" }'.

Customize the public portal

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

The screenshot shows the developer portal's home page. The top navigation bar includes 'Home' and 'My applications'. A blue 'Customize' button is located in the top right corner. The main content area features a large 'Welcome to your developer portal!' heading and a descriptive text: 'Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience.'

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

The screenshot shows the MuleSoft Anypoint Platform interface for configuring a new application. The main area displays a preview of the application's landing page. The header includes a logo, a 'Home' button, and a 'My applications' link. The sidebar on the right contains settings for various UI elements:

- Top bar:** Includes fields for 'Logo' and 'Favicon' with 'Choose file' buttons.
- Background color:** A color swatch set to #262728.
- Text color:** A color swatch set to #FFFFFF.
- Text color (active):** A color swatch set to #00A2DF.
- Welcome section:** Includes a 'Hero image' field containing 'image-default.png' with a 'Choose file' button.
- Text color:** A color swatch set to #FFFFFF.
- Custom pages:** A link to '+ Add new page'.

24. In the logo field, click Choose file.

25. In the file browser dialog box, navigate to the student files and locate the MuleSoft_training_logo.png file in the resources folder and click Open.

26. Locate the new logo in the preview.

27. In the hero image field, click Choose file.

28. In the file browser dialog box, navigate to the student files and locate the banner.jpg file in the resources folder and click Open.

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

The screenshot shows the MuleSoft Training portal builder interface. On the left, there's a preview of the portal with a dark header containing the MuleSoft logo and the text "Welcome to your MuleSoft Training portal!". Below the header is a banner with the text "Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience.". Underneath the banner is a search bar and a sidebar listing "All types" and "Max Mule". On the right side of the interface, there are several customization sections: "Top bar" (Logo set to "MuleSoft_training_logo.png", "Choose file" button), "Favicon" (empty "Choose file" button), "Background color" (#262728), "Text color" (#FFFFFF), "Text color (active)" (#00A2DF), "Welcome section" (Hero image set to "banner.jpg", "Choose file" button), "Text color" (#FFFFFF), and "Custom pages" (+ Add new page).

30. Change any colors that you want.

31. Click the Done editing button.

32. In the Publish changes dialog box, click Yes, publish; you should see your customized public portal.

Explore the public portal as an external developer

33. In the browser, copy the URL for the public portal.

34. Open a new private or incognito window in your browser.

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

The screenshot shows the MuleSoft Training portal. At the top, it says "Welcome to your MuleSoft Training portal!". Below that, a sub-headline reads "Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience." A search bar at the top has "All types" and a search icon. Below the headline, there's a card for the "American Flights API". The card has a green house icon, a 5-star rating, and the text "American Flights API" and "Max Mule".

36. Click the American Flights API.

37. Explore the API portal.

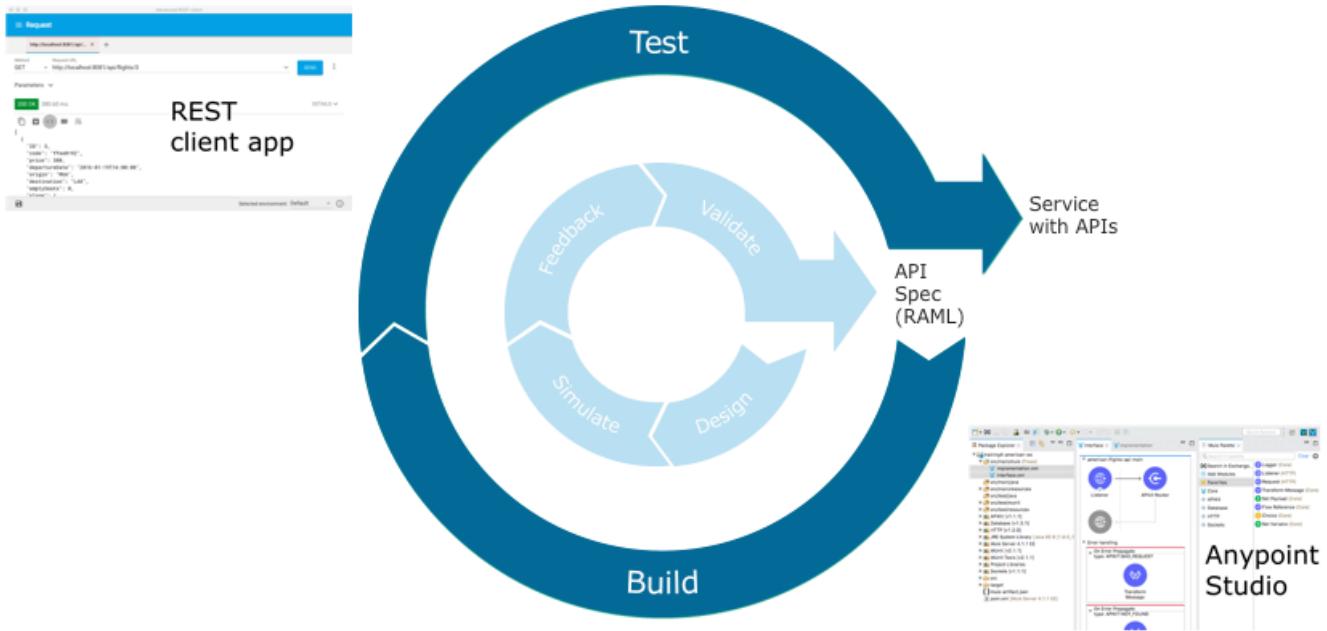
38. Make a call to one of the resource methods.

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

The screenshot shows the "American Flights API" details page. On the left, there's a sidebar with "Assets list" and "API instances". Under "API instances", there's a "GET /flights" entry. The main panel shows the API endpoint "/flights : get" with a "Request" section. It includes a "GET /flights" button and a "Parameters" table. The table has a header row with "Parameter", "Type", and "Description". Under "Query parameters", there's a row for "destination" with type "string (enum)" and possible values "SFO, LAX, CLE". To the right, there's a "GET" request configuration panel. It shows "Mocking Service" set to "https://mocksvc-proxy.anypoint.mulesoft.com/", "Parameters" tab selected, and a "Send" button. Below that, a "200 OK" response is shown with a timestamp of "260.37 ms" and a "Details" link. The response body is an array of two flight objects:

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE"
}]
```

Module 4: Building APIs



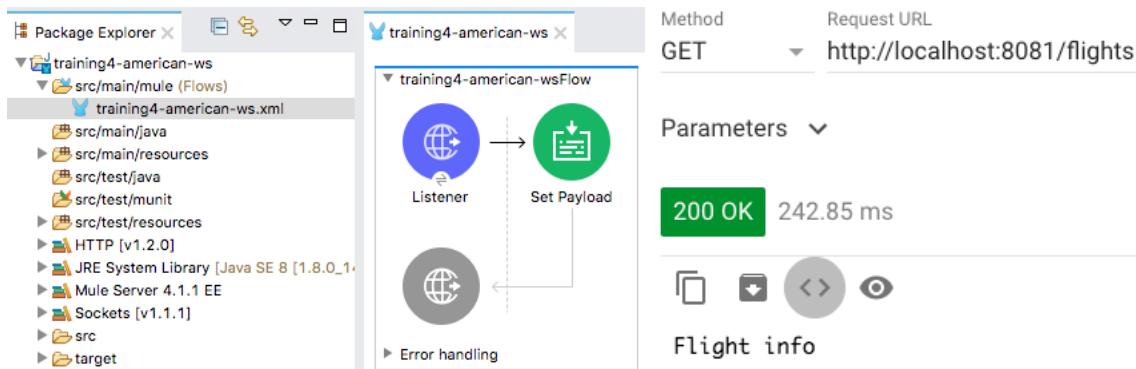
At the end of this module, you should be able to:

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

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

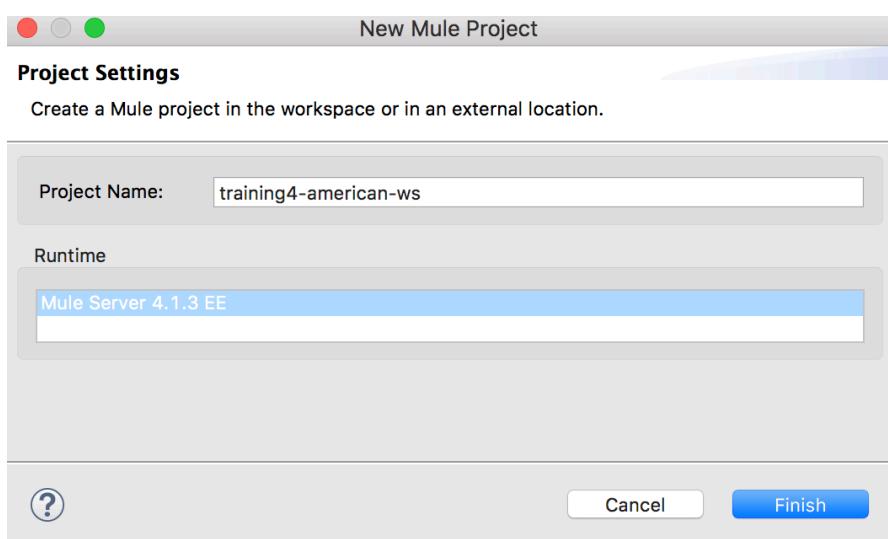
In this walkthrough, you build a Mule application. You will:

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



Create a Mule project

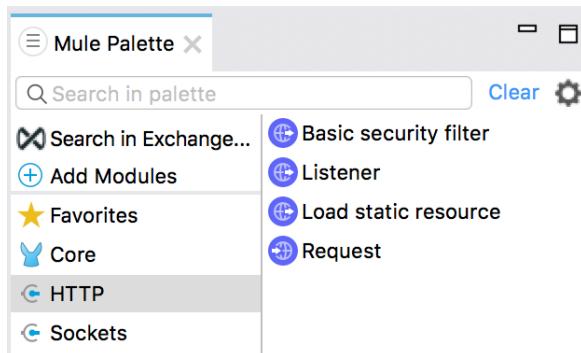
1. Open Anypoint Studio.
2. Select File > New > Mule Project.
3. In the New Mule Project dialog box, set the Project Name to training4-american-ws.
4. Ensure the Runtime is set to the latest version of Mule.



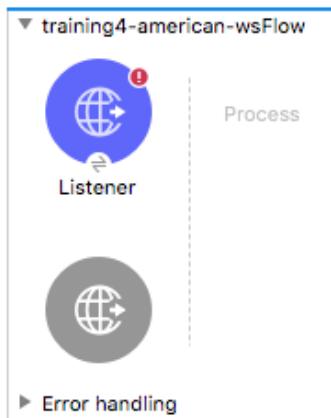
5. Click Finish.

Create an HTTP connector endpoint to receive requests

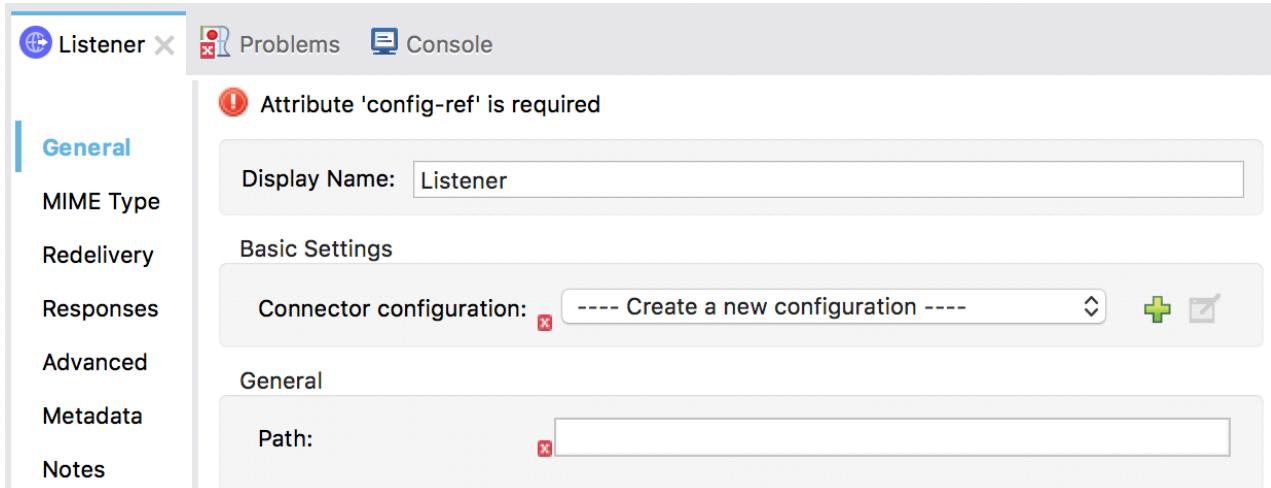
6. In the Mule Palette, select the HTTP module.



7. Drag the Listener operation from the Mule Palette to the canvas.

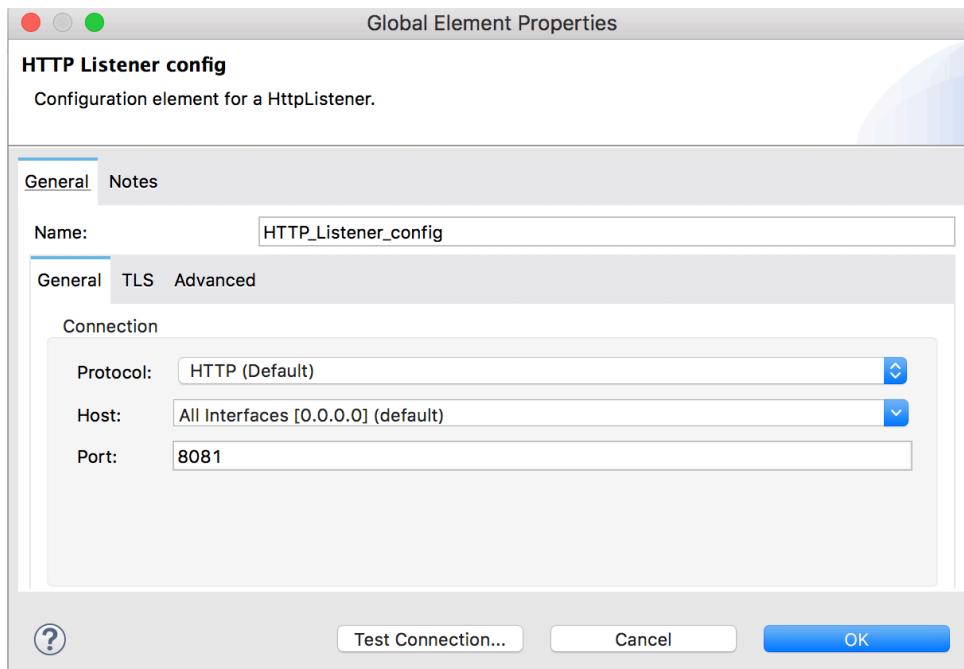


8. Double-click the HTTP Listener endpoint.
9. In the Listener properties view that opens at the bottom of the window, click the Add button next to connector configuration.



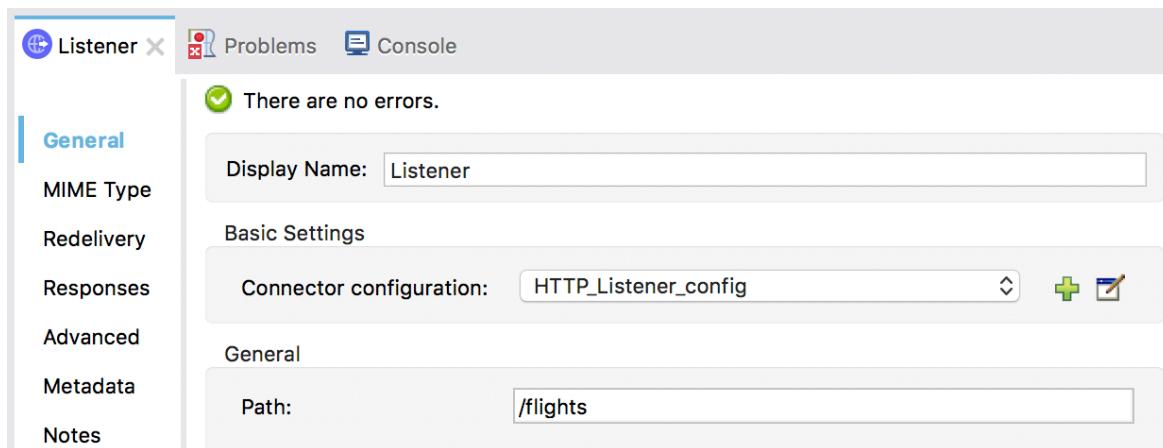
10. In the Global Element Properties dialog box, verify the following default values are present.

- Host: 0.0.0.0
- Port: 8081

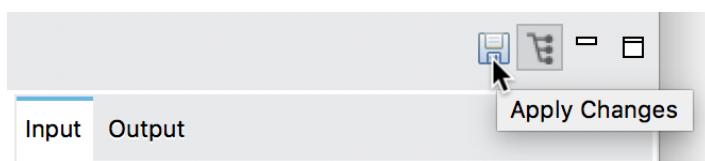


11. Click OK.

12. In the Listener properties view, set the path to /flights.

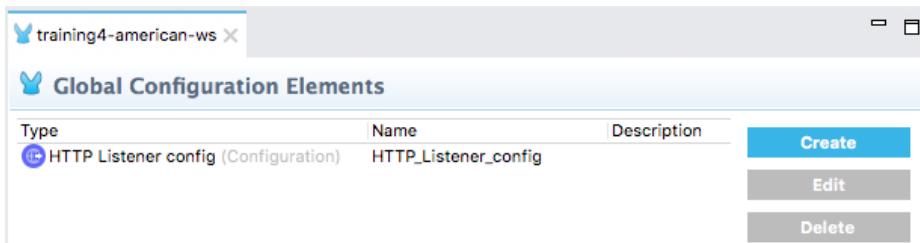


13. Click the Apply Changes button to save the file.



Review the HTTP Listener global element

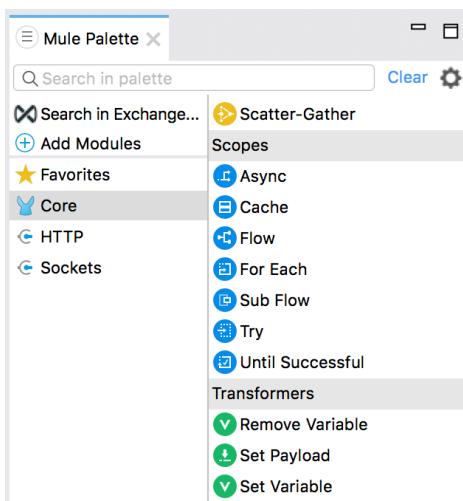
14. Select the Global Elements tab at the bottom of the canvas.
15. Double-click the HTTP Listener config.



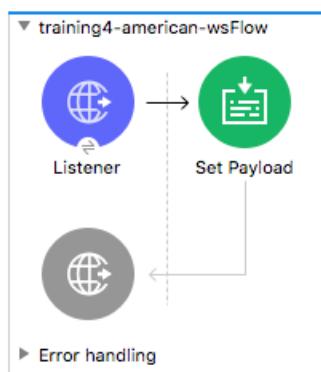
16. Review the information in the Global Element Properties dialog box and click Cancel.
17. Select the Message Flow tab to return to the canvas.

Display data

18. In the Mule Palette, select Core.
19. Scroll down in the right-side of the Mule Palette and locate the Transformers section.

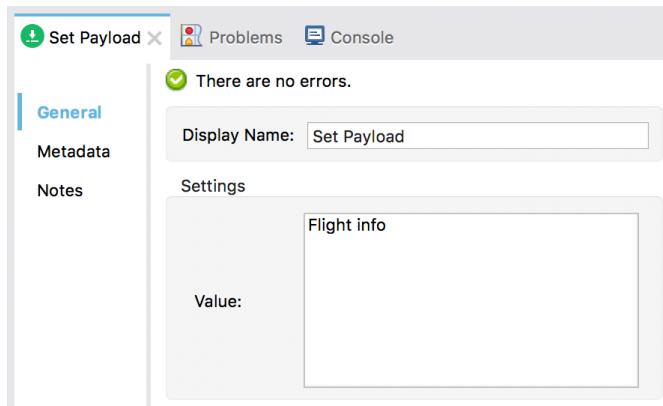


20. Drag the Set Payload transformer from the Mule Palette into the process section of the flow.



Configure the Set Payload transformer

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



22. Select the Configuration XML tab at the bottom of the canvas and examine the corresponding XML.

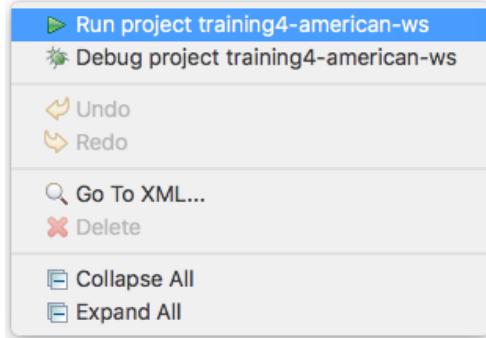
```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns="http://www.mulesoft.org/schema/mule/documentation"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.mulesoft.org/schema/mule/http/c
      <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener config">
        <http:listener-connection host="0.0.0.0" port="8081" />
      </http:listener-config>
      <flow name="training-american-wsFlow" doc:id="f3b0df18-0181-4935-86f2-d4dfba39f
        <http:listener doc:name="Listener" doc:id="8e9f2503-8230-4525-bca3-d7dd28ea
          <set-payload value="Flight info" doc:name="Set Payload" doc:id="7196a475-0f
        </flow>
      </mule>
```

23. Select the Message Flow tab to return to the canvas.

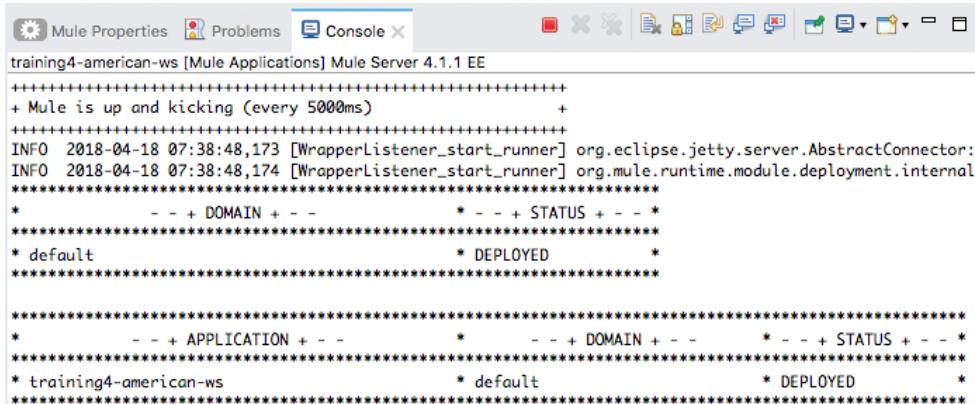
24. Click the Save button or press Cmd+S or Ctrl+S.

Run the application

25. Right-click in the canvas and select Run project training4-american-ws.



26. Watch the Console view; it should display information letting you know that both the Mule runtime and the training4-american-ws application started.



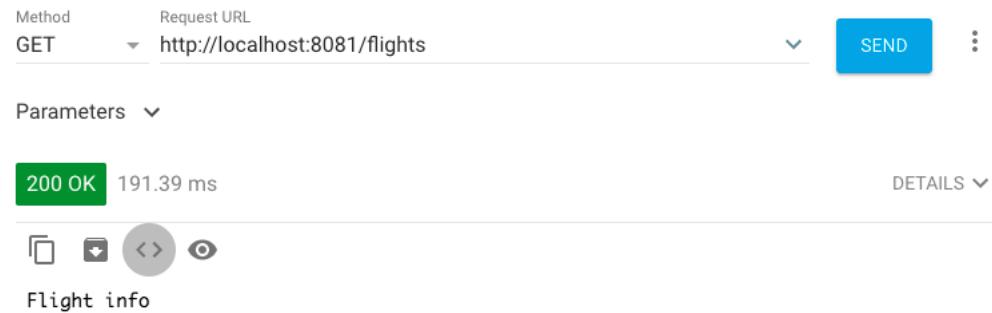
```
Mule Properties Problems Console
training4-american-ws [Mule Applications] Mule Server 4.1.1 EE
+-----+
+ Mule is up and kicking (every 5000ms) +
+-----+
INFO 2018-04-18 07:38:48,173 [WrapperListener_start_runner] org.eclipse.jetty.server.AbstractConnector:
INFO 2018-04-18 07:38:48,174 [WrapperListener_start_runner] org.mule.runtime.module.deployment.internal
*-----*
*      - - + DOMAIN + - -          * - - + STATUS + - - *
*-----*
* default                                * DEPLOYED          *
*-----*
*-----*
*      - - + APPLICATION + - -          *      - - + DOMAIN + - -          * - - + STATUS + - - *
*-----*
* training4-american-ws                  * default          * DEPLOYED          *
*-----*
```

Test the application

27. Return to Advanced REST Client.

28. Make sure the method is set to GET and that no headers or body are set for the request.

29. Make a GET request to <http://localhost:8081/flights>; you should see Flight info displayed.



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

SEND ::

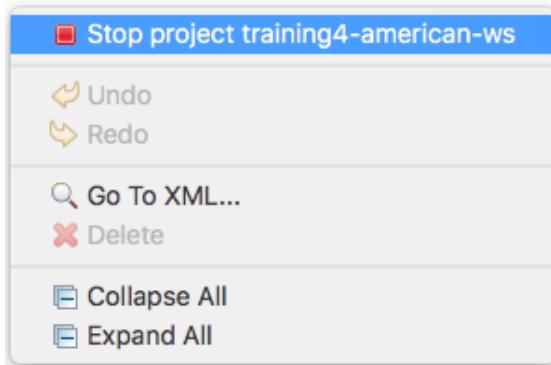
Parameters ▾

200 OK 191.39 ms DETAILS ▾

Flight info

30. Return to Anypoint Studio.

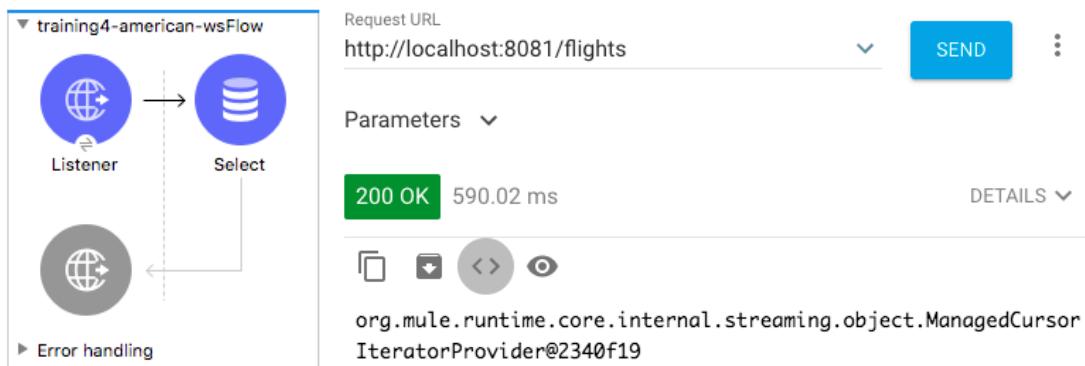
31. Right-click in the canvas and select Stop project training4-american-ws.



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

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

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



Locate database information

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

```
* MySQL database
db:
  host: "mudb.learn.mulesoft.com"
  port: "3306"
  user: "mule"
  password: "mule"
  database: "training"

American table: american
American table version2: flights
Account table: accounts
Account list URL: http://mu.learn.mulesoft.com/accounts/show
or if using mulesoft-training-services.jar application:
http://localhost:9090/accounts/show.html

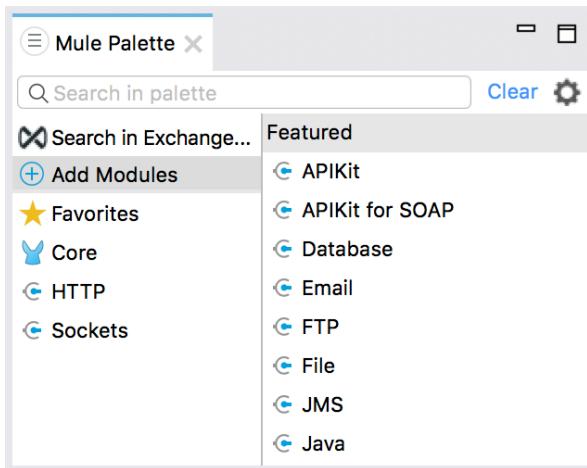
* MySQL database as URL and driver name
URL: jdbc:mysql://mudb.learn.mulesoft.com:3306/training?user=mule&password=mule
Driver class name: com.mysql.jdbc.Driver

* Derby database
URL: jdbc:derby://localhost:1527/memory:training
Driver class name: org.apache.derby.jdbc.ClientDriver
```

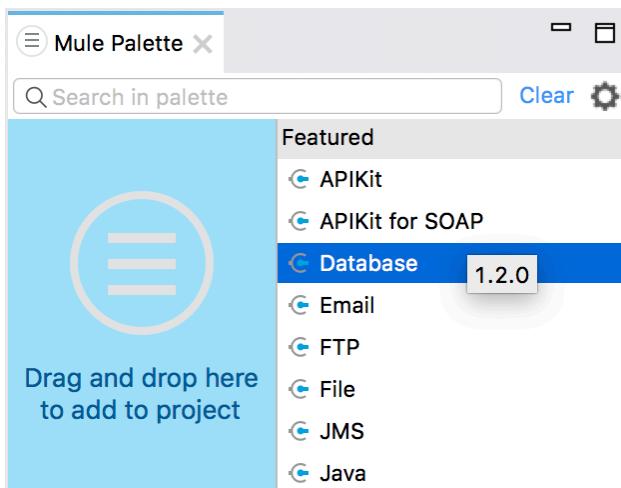
Note: The database information you see may be different than what is shown here; the values in the snippets file differ for instructor-led and self-study training classes.

Add a Database connector endpoint

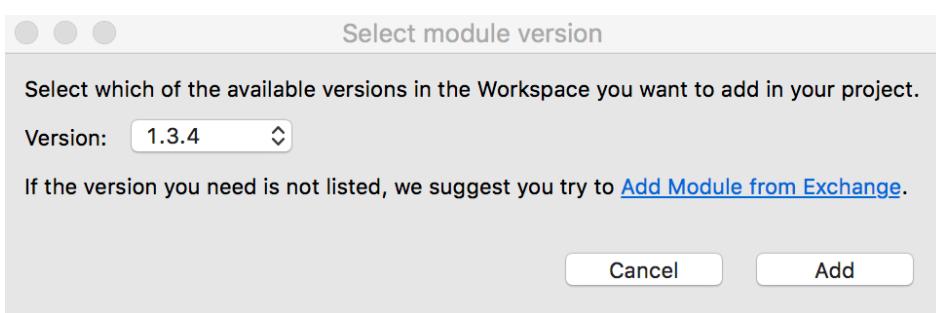
2. Return to Anypoint Studio.
3. Right-click the Set Payload message processor and select Delete.
4. In the Mule Palette, select Add Modules.



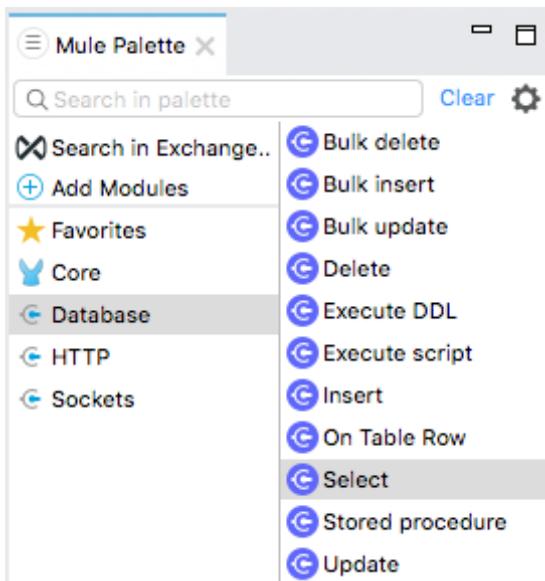
5. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.



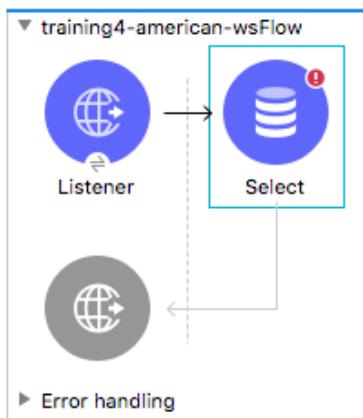
6. If you get a Select module version dialog box, select the latest version and click Add.



7. Locate the new Database connector in the Mule Palette.

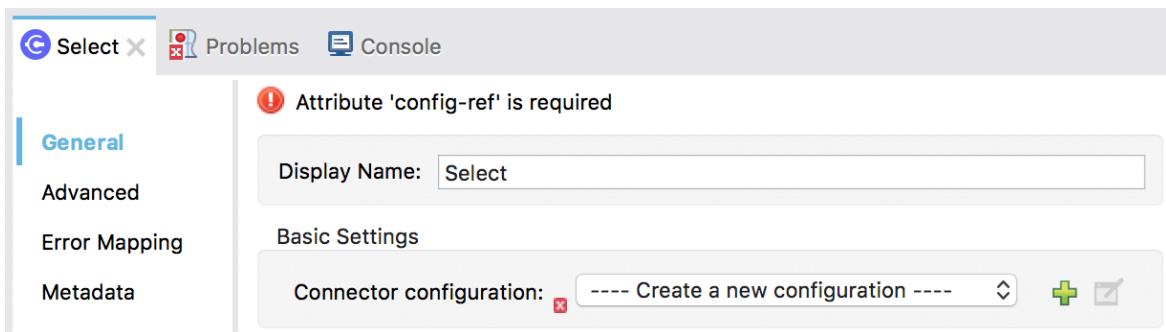


8. Drag and drop the Select operation in the process section of the flow.



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

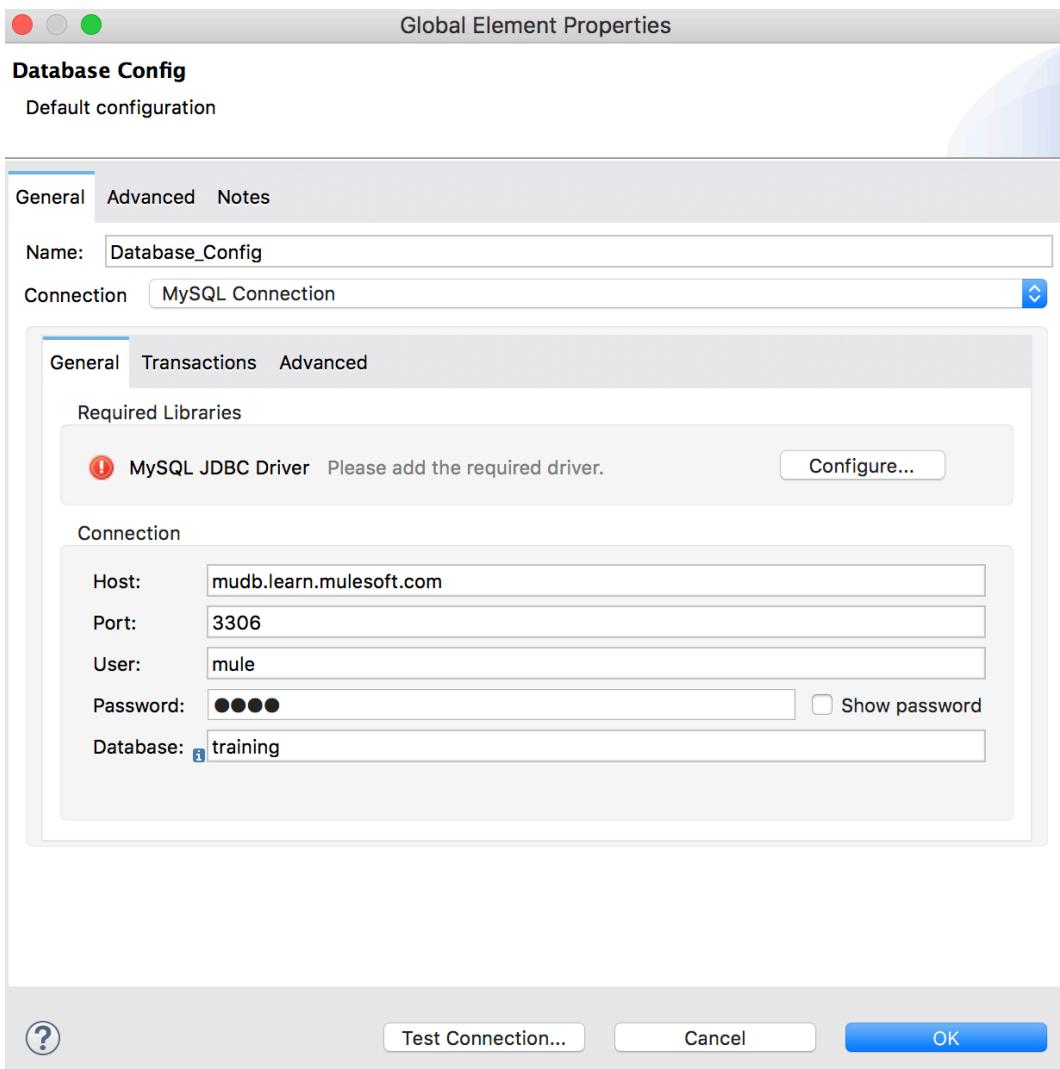
9. In the Select properties view, click the Add button next to connector configuration.



10. In the Global Element Properties dialog box, set the Connection to MySQL Connection.

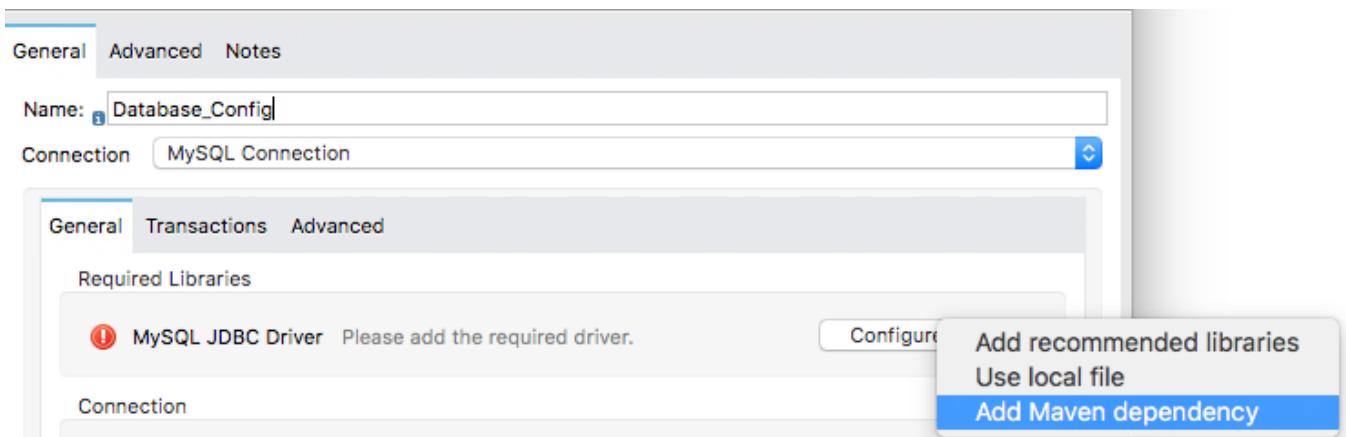


11. Set the host, port, user, password, and database values to the values listed in the course snippets.txt file.

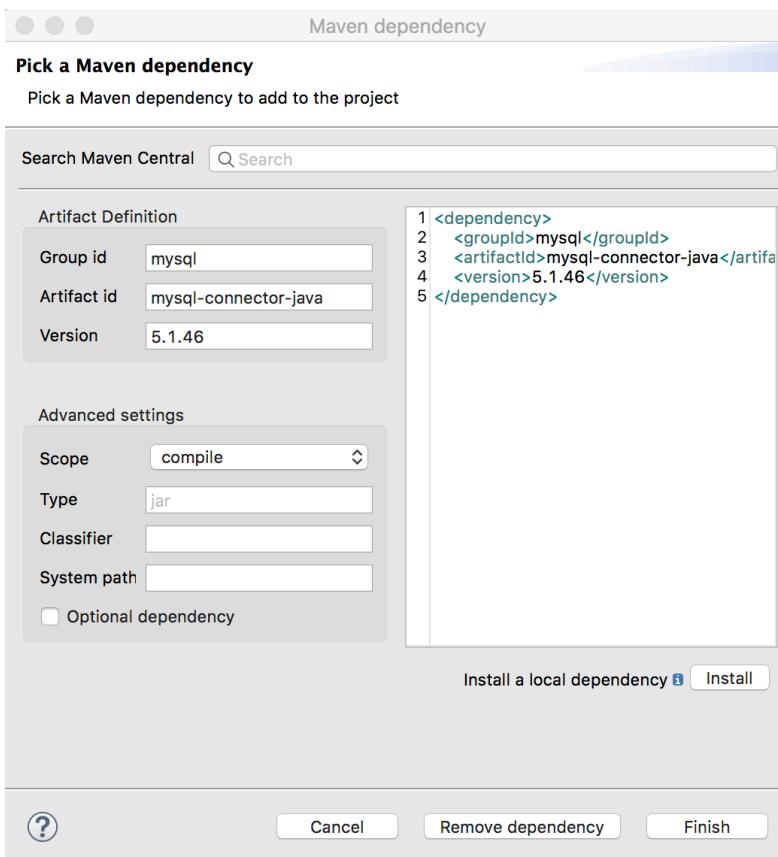


12. Click the Configure button next to MySQL JDBC Driver.

13. In the configure drop-down menu, select Add Maven dependency.

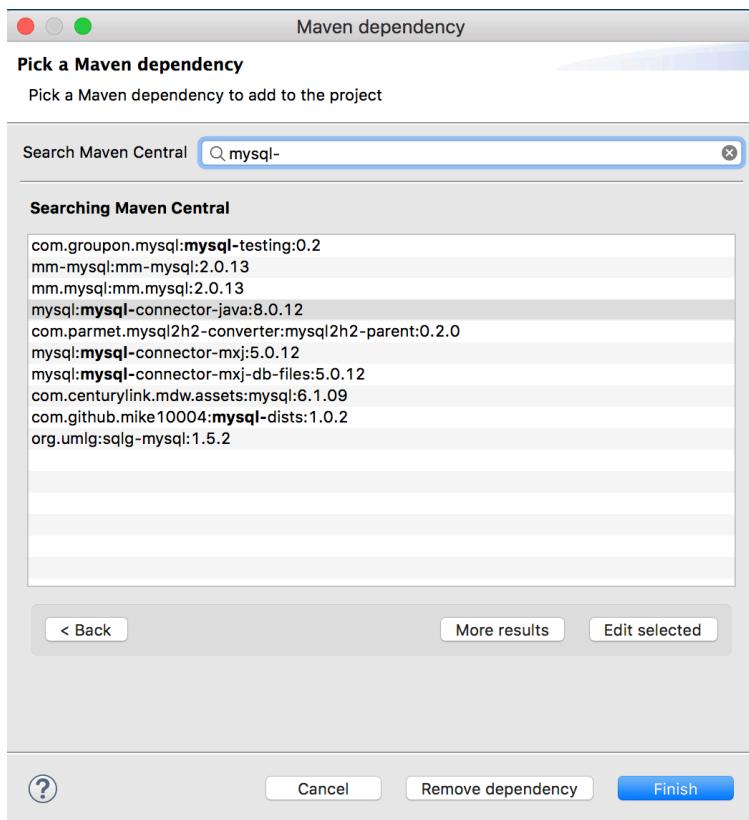


14. In the Maven dependency dialog box, locate the Search Maven Central text field.



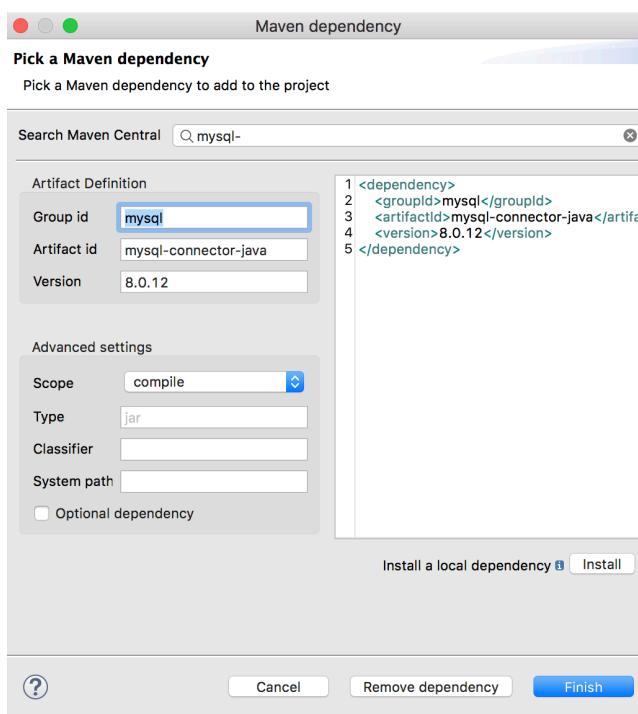
15. Enter mysql- in the Search Maven Central text field.

16. Select mysql:mysql-connector-java in the results that are displayed.



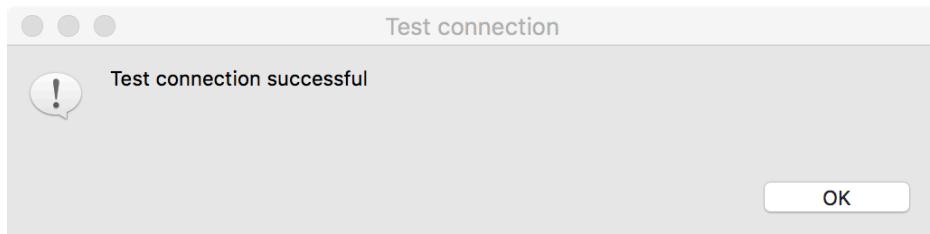
17. Click Edit selected.

18. Click Finish.



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

Note: Make sure the connection succeeds before proceeding.



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

20. Click OK to close the dialog box.
21. Click OK to close the Global Element Properties dialog box.

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

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

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

Note: Replace X.X.X with the version of the JAR file, for example 1.6.2.

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

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

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

```
(\_) M U L E S O F T   T R A I N I N G   S E R V I C E S
/ \ *** Version 1.6.2 ***

Starting resources:
- Message Broker started
- American database started
- American flights database ready
- Delta flights web service started
- Essentials Delta flights web service started
- Order web service started
- Accounts REST API published
- American flights API published
- Banking REST API published
- Essentials Accounts REST API published
- Essentials American flights API published
- Essentials JMS API published
- Essentials United flights web service started
- JMS API published
- United flights web service started

Available resources:
- Welcome page : http://localhost:9090
- American database URL : jdbc:derby://localhost:1527/memory:training
- JMS broker URL : tcp://localhost:61616
- Essentials American REST API : http://localhost:9090/essentials/american/flights
- Essentials American REST API RAML : http://localhost:9090/essentials/american/flights-api.raml
- Essentials United REST service : http://localhost:9090/essentials/united/flights
- Essentials Delta SOAP WSDL : http://localhost:9191/essentials/delta?wsdl
- Essentials Accounts API : http://localhost:9090/essentials/accounts/api
- Essentials Accounts form : http://localhost:9090/essentials/accounts/show.html
- Essentials JMS form : http://localhost:9090/essentials/jmsform.html
- Essentials JMS topic name : ap essentials

- Fundamentals American REST API : http://localhost:9090/american/flights
- Fundamentals American REST API RAML : http://localhost:9090/american/flights-api.raml
- Fundamentals United REST service : http://localhost:9090/united/flights
- Fundamentals Delta SOAP WSDL : http://localhost:9191/delta?wsdl
- Fundamentals Accounts API : http://localhost:9090/accounts/api
- Fundamentals Accounts form : http://localhost:9090/accounts/show.html
- Fundamentals JMS form : http://localhost:9090/jmsform.html
- Fundamentals JMS topic name : training

- Advanced Order SOAP service : http://localhost:9191/advanced/orders
- Advanced Order SOAP WSDL : http://localhost:9191/advanced/orders?wsdl
- Advanced Maven settings.xml : http://localhost:9191/advanced/settings.xml

- Banking API : http://localhost:9090/api/...
- Banking API RAML : http://localhost:9090/api/banking-api.raml

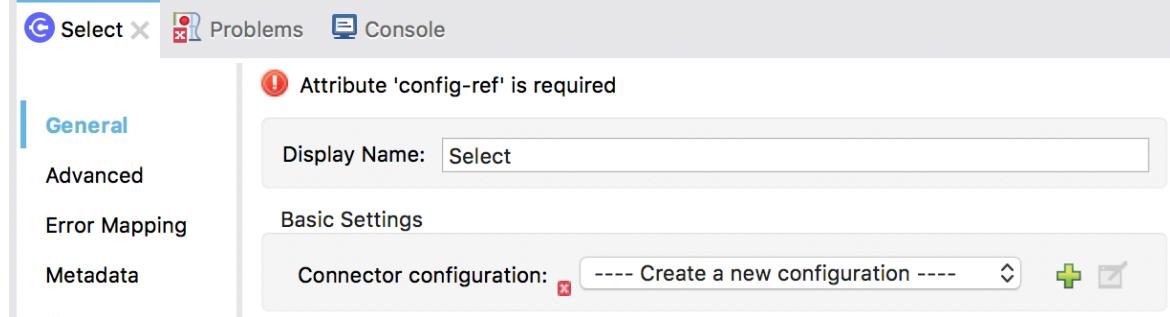
Press CTRL-C to terminate this application...
```



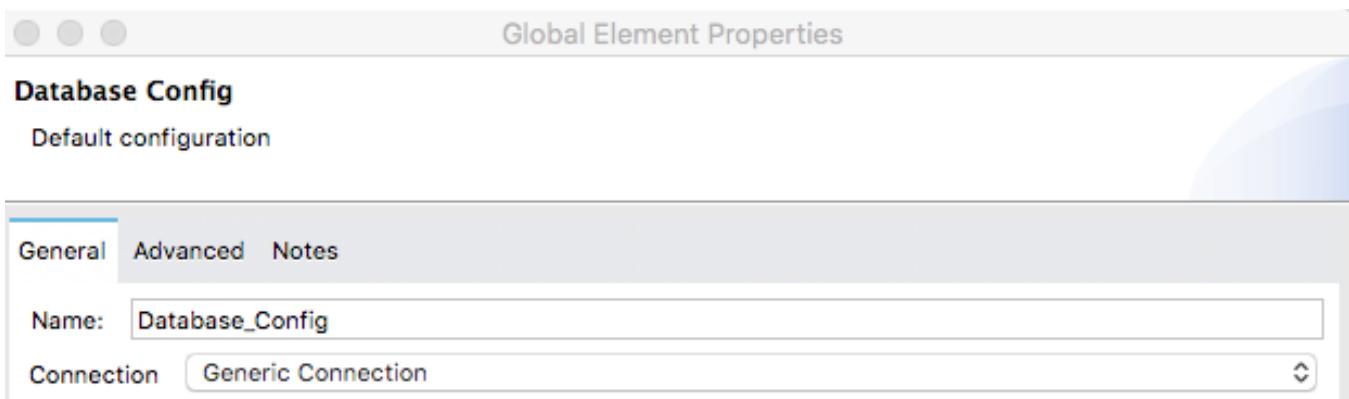
Note: When you want to stop the application, return to this window and press Ctrl+C.

25. Return to Anypoint Studio.

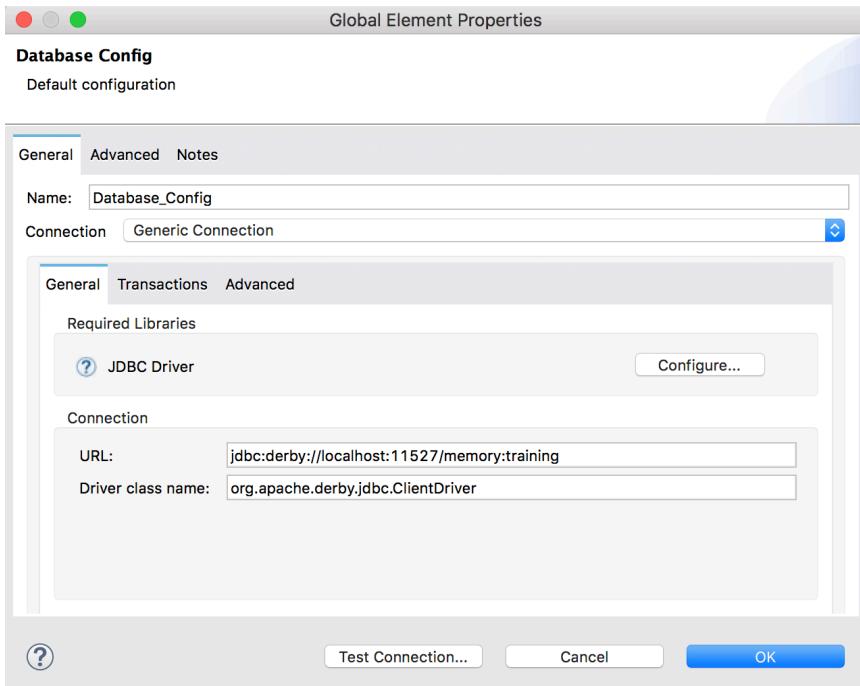
26. In the Select properties view, click the Add button next to connector configuration.



27. In the Global Element Properties dialog box, set the Connection to Generic Connection.



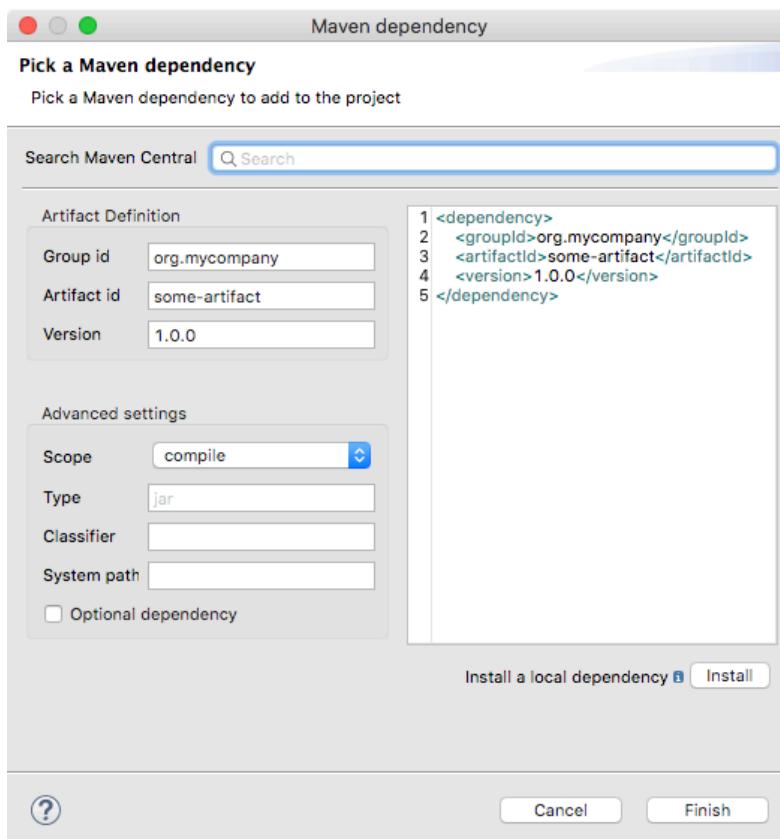
28. Set the URL and driver class name values to the values listed in the course snippets.txt file.



29. Click the Configure button next to JDBC Driver.

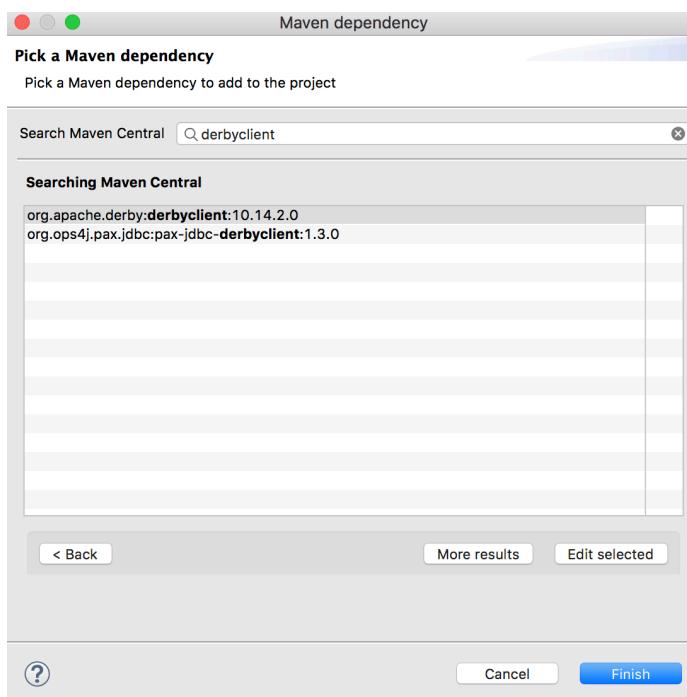
30. In the configure drop-down menu, select Add Maven dependency.

31. In the Maven dependency dialog box, locate the Search Maven Central text field.

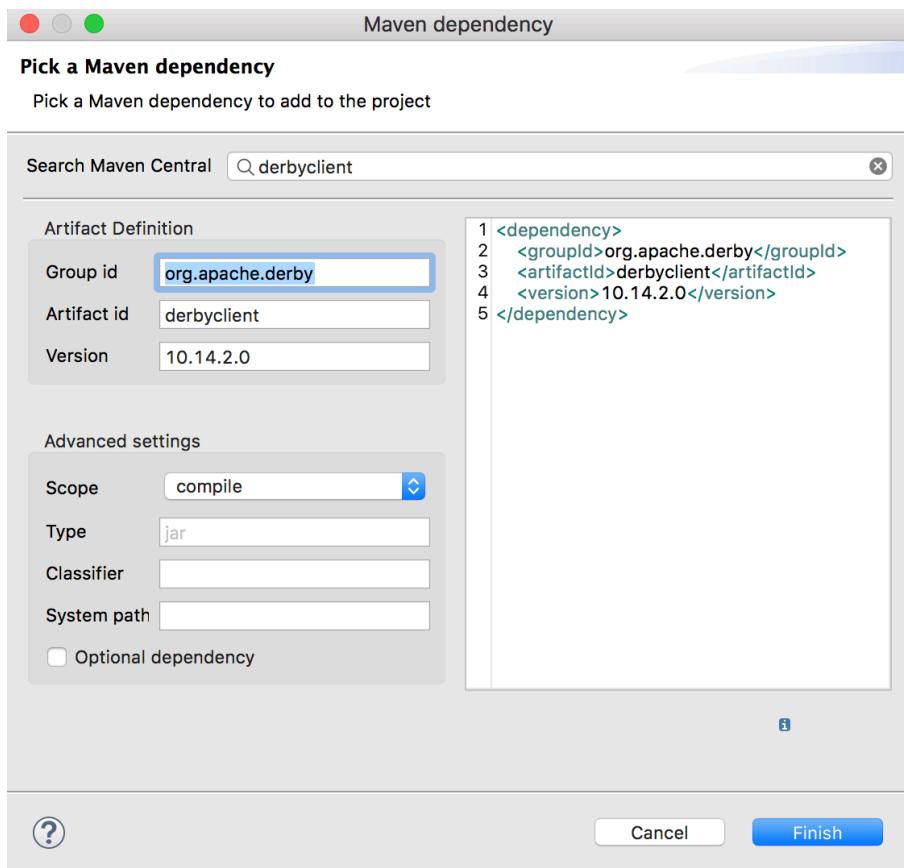


32. Enter derbyclient in the Search Maven Central text field.

33. Select org.apache.derby:derbyclient in the results that are displayed.



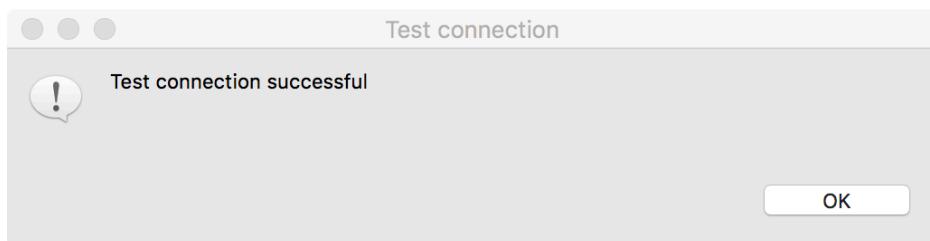
34. Click Edit selected.



35. Click Finish.

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

Note: Make sure the connection succeeds before proceeding.



37. Click OK to close the dialog box.

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

Write a query to return all flights

39. In the Select properties view, add a query to select all records from the american table.

```
SELECT *\nFROM american
```

Select X Problems Console

General

Advanced

Error Mapping

Metadata

Notes

Connector configuration: Database_Config

SQL Query Text:

```
SELECT *\nFROM american
```

Test the application

40. Run the project.
41. In the Save changes dialog box, select Yes.
42. Watch the console and wait for the application to start.
43. Once the application has started, return to Advanced REST Client.
44. In Advanced REST Client, make another request to <http://localhost:8081/flights>; you should get some type of Mule object.

Method Request URL

GET http://localhost:8081/flights

SEND

Parameters

200 OK 570.39 ms DETAILS

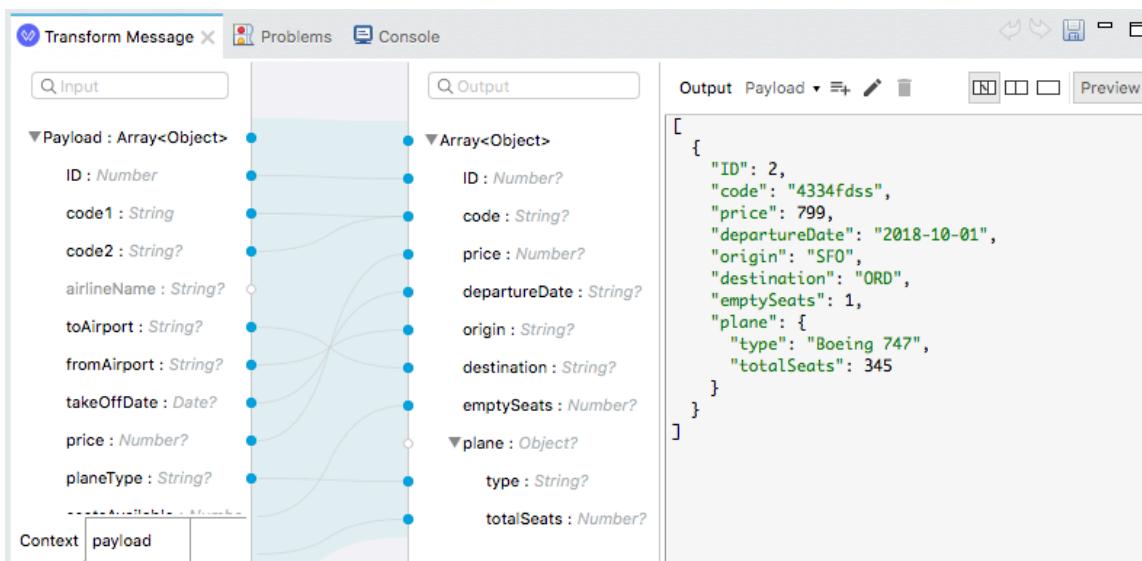
Parameters

org.mule.runtime.core.internal.streaming.object.ManagedCursorIteratorProvider@6ef4f8f8

Walkthrough 4-3: Transform data

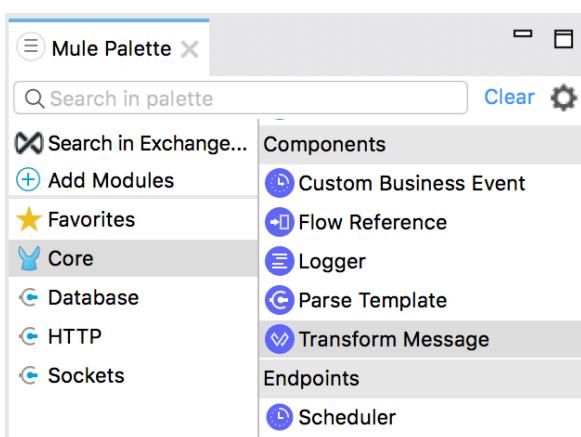
In this walkthrough, you transform and display the flight data into JSON. You will:

- Use the Transform Message component.
- Use the DataWeave visual mapper to change the response to a different JSON structure.

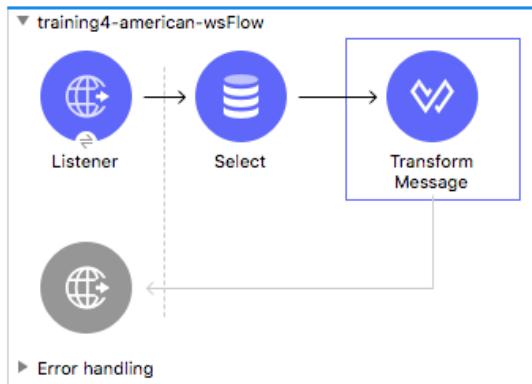


Add a Transform Message component

1. Return to Anypoint Studio.
2. In the Mule Palette, select Core and locate the Transform Message component in the Components section.



- Drag the Transform Message and drop it after the Select processor.



Review metadata for the transformation input

- In the Transform Message properties view, look at the input section and review the payload metadata.

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

```

1 %dw 2.0
2 output application/java
3 ---
4 {
5 }

```

Return the payload as JSON

- In the Transform Message properties view, change the output type from application/java to application/json and change the {} to payload.

```

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

```

Test the application

6. Save the file to redeploy the project.
7. In Advanced REST Client, send the same request; you should see the American flight data represented as JSON.

The screenshot shows the Advanced REST Client interface. At the top, there are fields for 'Method' (set to 'GET') and 'Request URL' (set to 'http://localhost:8081/flights'). Below these are buttons for 'SEND' and a more options menu. Underneath, a 'Parameters' dropdown is shown. The main area displays a green '200 OK' status bar with '1299.94 ms' latency. To the right is a 'DETAILS' button. Below the status bar, there are several icons: a copy icon, a refresh icon, a compare icon, and a full screen icon. The response body is a JSON array of 11 elements, each representing a flight. The first element is expanded to show its properties:

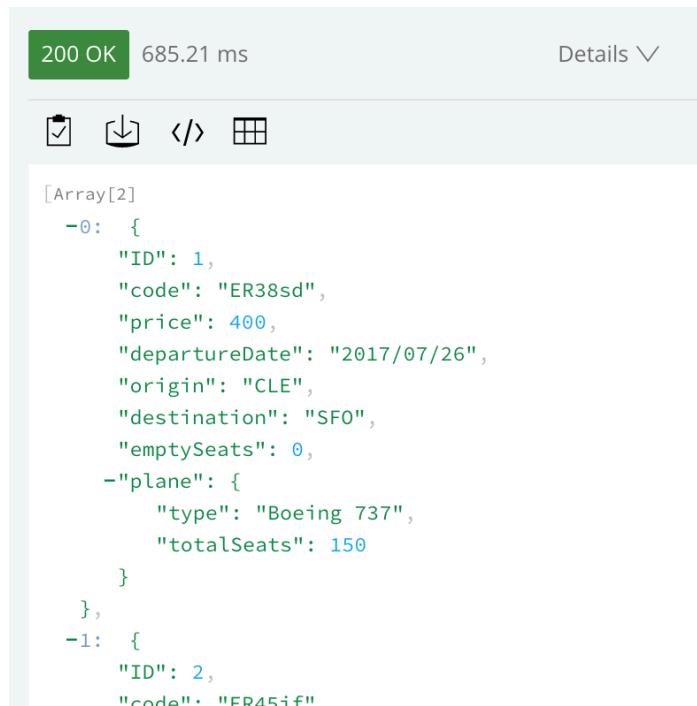
```
[Array[11]
-0: {
  "code2": "0001",
  "planeType": "Boeing 787",
  "totalSeats": 200,
  "toAirport": "LAX",
  "takeOffDate": "2016-01-19T16:00:00",
  "fromAirport": "MUA",
  "price": 541,
  "airlineName": "American Airlines",
  "seatsAvailable": 0,
  "ID": 1,
  "code1": "rree"
},
-1: {
  "code2": "0123",
}
```

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

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

8. Return to your American Flights API in Exchange.

9. Look at the example data returned for the /flights GET method.



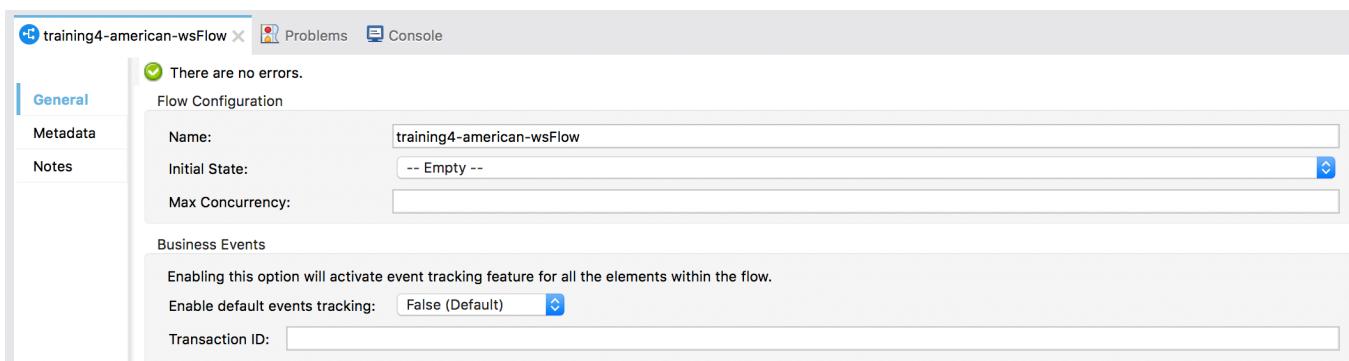
The screenshot shows a network request details view from a browser. At the top, it says "200 OK" and "685.21 ms". To the right, there's a "Details" button with a dropdown arrow. Below this, there are icons for copy, download, and refresh. The main content area displays a JSON array with two elements. The first element has an ID of 1, a code of "ER38sd", a price of 400, a departure date of "2017/07/26", an origin of "CLE", a destination of "SFO", and 0 empty seats. It also includes a nested "plane" object with a type of "Boeing 737" and 150 total seats. The second element has an ID of 2 and a code of "FR451F".

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
},
-1: {
  "ID": 2,
  "code": "FR451F"
}]
```

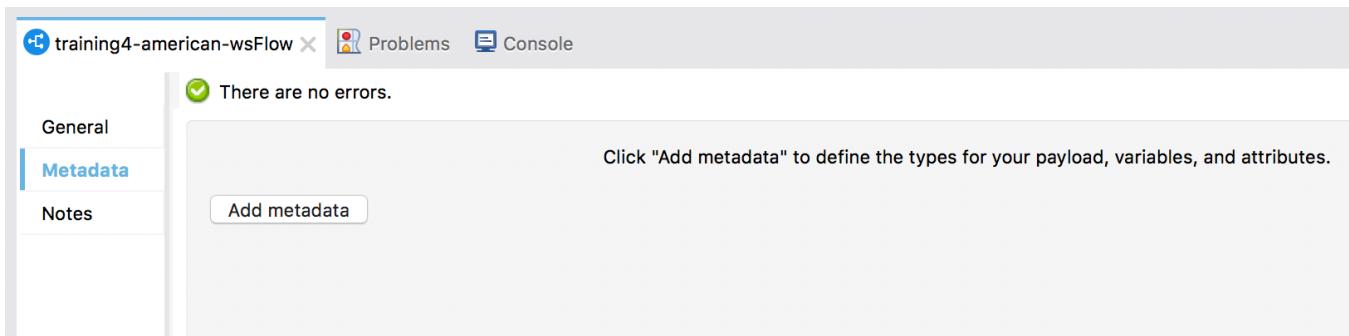
10. Notice that the structure of the JSON being returned by the Mule application does not match this JSON.

Define metadata for the data structure to be returned by the American flights API

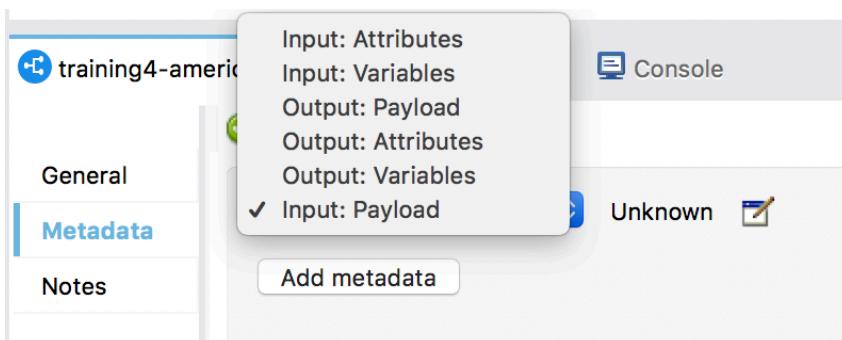
11. Return to Anypoint Studio.
12. In the canvas, click training4-american-wsFlow name.
13. In the training-american-wsFlow properties view, select the Metadata tab.



14. Click the Add metadata button.



15. In drop-down menu, select Output: Payload.



16. Click the Edit button.

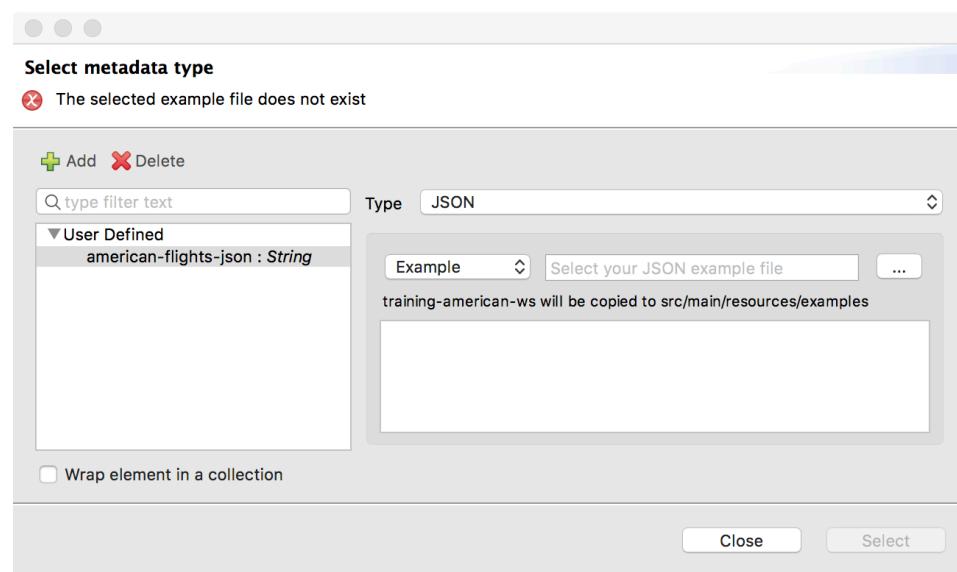
17. In the Select metadata type dialog box, click the Add button.

18. In the Create new type dialog box, set the type id to american_flights_json.

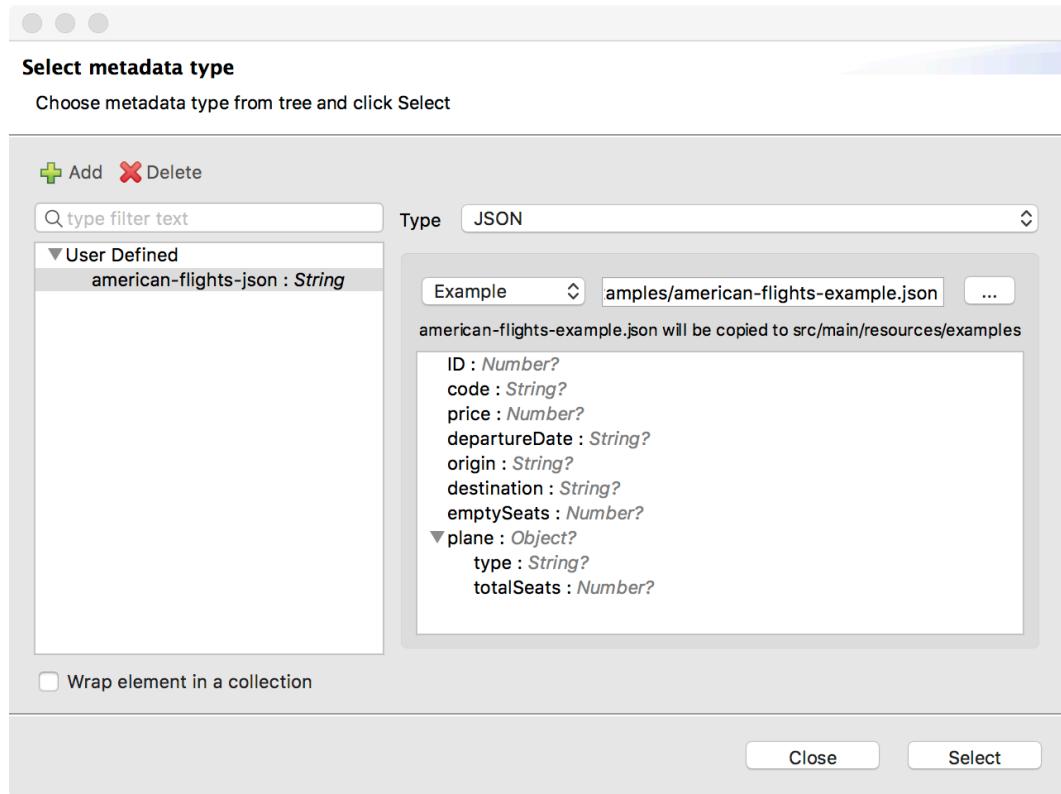
19. Click Create type.

20. Back in the Set metadata type dialog box, set the type to JSON.

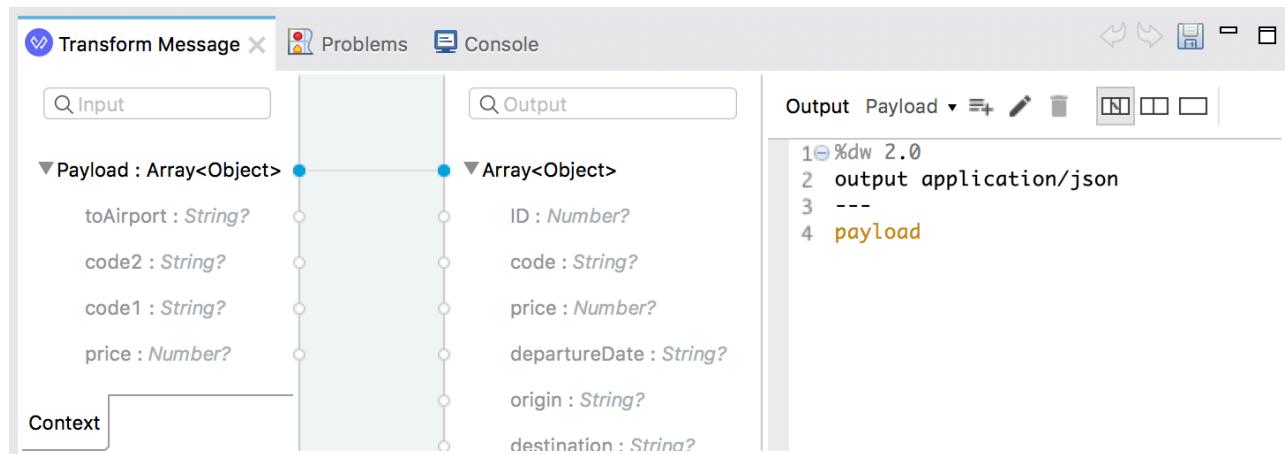
21. Change the Schema selection to Example.



22. Click the browse button and navigate to the course student files.
23. Select american-flights-example.json in the examples folder and click Open; you should see the example data for the metadata type.



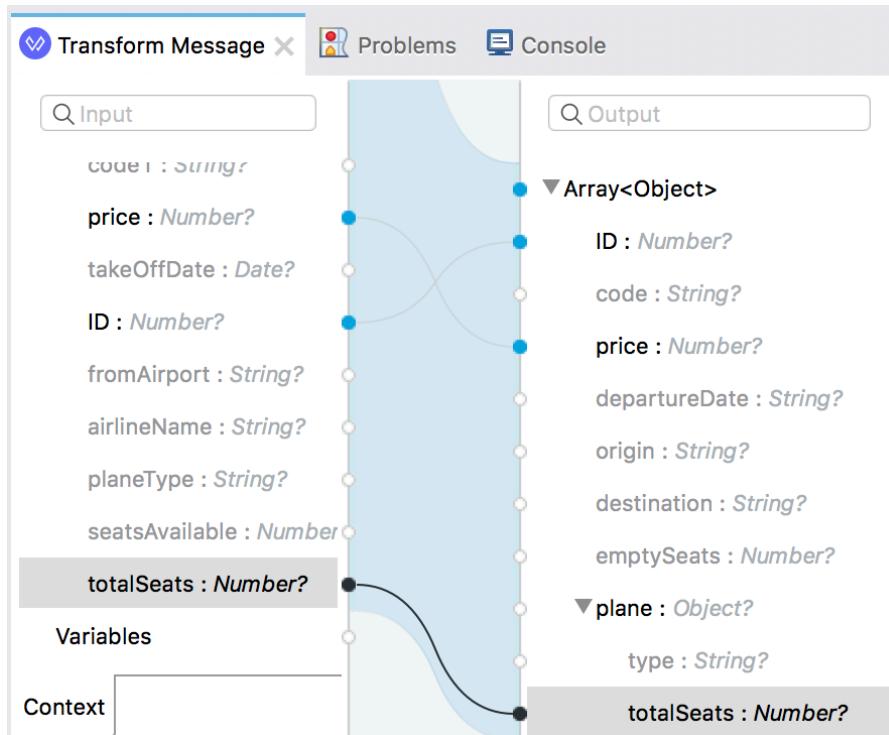
24. Click Select.
25. In training4-american-wsFlow, click the Transform Message component; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

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

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

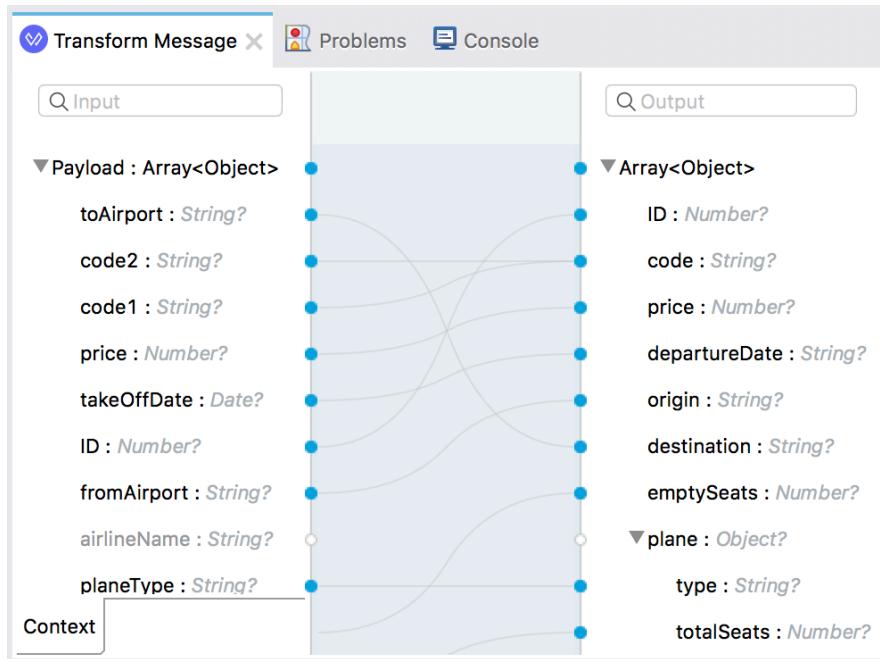


27. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

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

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

- code1 to code
- code2 to code



Add sample data (optional)

29. Click the Preview button in the output section.

30. In the preview section, click the Create required sample data to execute preview link.

The screenshot shows the 'Preview' section of the 'Transform Message' component. On the left, there is a code editor with DWScript code. On the right, there is a preview area with a link labeled 'Create required sample data to execute preview'.

```
1 %dw 2.0
2 output application/json
3 ---
4 payload map ( payload01 , indexOfPc )
5   ID: payload01.ID,
6   code: (payload01.code1 default
7     price: payload01.price,
8     departureDate: payload01.takeOffDate,
9     origin: payload01.fromAirport,
10    destination: payload01.toAirport,
11    emptySeats: payload01.emptySeats,
12    plane: {
13      "type": payload01.planeType,
14      totalSeats: payload01.totalSeats
15    }
16 }
```

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

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. On the left, the 'list_.dwl' file is open, displaying a Mule configuration with an array of objects as input. The middle pane shows a mapping grid where each input object is transformed into an output object. The output structure is defined on the right, showing an array of objects with properties like ID, code, price, departureDate, origin, destination, emptySeats, and a nested plane object. The 'payload' tab at the bottom contains sample data for the input, and the 'Output' tab shows the resulting JSON structure.

```
%dw 2.0
output application/java
---
[{
    toAirport: "????",
    code2: "????",
    code1: "????",
    price: 2,
    takeOffDate: "2003-10-01",
    ID: 2,
    fromAirport: "????",
    airlineName: "????",
    planeType: "????",
    seatsAvailable: 2,
    totalSeats: 2
}]
```

```
[
  {
    "ID": 2,
    "code": "????????",
    "price": 2,
    "departureDate": "2003-10-01",
    "origin": "????",
    "destination": "????",
    "emptySeats": 2,
    "plane": {
      "type": "????",
      "totalSeats": 2
    }
  }
]
```

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

This screenshot shows the same 'Transform Message' editor after step 33. The input section now contains sample values for all fields. The output section shows the transformed data with these specific values. The 'payload' tab at the bottom shows the original input data, and the 'Output' tab shows the transformed JSON with the replaced placeholder values.

```
%dw 2.0
output application/java
---
[{
    toAirport: "ORD",
    code2: "fdss",
    code1: "4334",
    price: 799,
    takeOffDate: "2018-10-01",
    ID: 1,
    fromAirport: "SFO",
    airlineName: "american",
    planeType: "Boeing 747",
    seatsAvailable: 1,
    totalSeats: 345
}]
```

```
[
  {
    "ID": 1,
    "code": "4334fdss",
    "price": 799,
    "departureDate": "2018-10-01",
    "origin": "SFO",
    "destination": "ORD",
    "emptySeats": 1,
    "plane": {
      "type": "Boeing 747",
      "totalSeats": 345
    }
  }
]
```

Test the application

35. Save the file to redeploy the project.

36. In Advanced REST Client, make another request to <http://localhost:8081/flights>; you should see all the flight data as JSON again but now with a different structure.

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

Parameters:

200 OK 1123.12 ms DETAILS

[Array[11]]
-0: {
 "ID": 1,
 "code": "rreee0001",
 "price": 541,
 "departureDate": "2016-01-19T16:00:00",
 "origin": "MUA",
 "destination": "LAX",
 "emptySeats": 0,
 "plane": {
 "type": "Boeing 787",
 "totalSeats": 200
 }
},

Try to retrieve information about a specific flight

37. Add a URI parameter to the URL to make a request to <http://localhost:8081/flights/3>; you should get a 404 Not Found response with a no listener message.

Method: GET Request URL: http://localhost:8081/flights/3

Parameters:

404 Not Found 21.24 ms DETAILS

No listener for endpoint: /flights/3

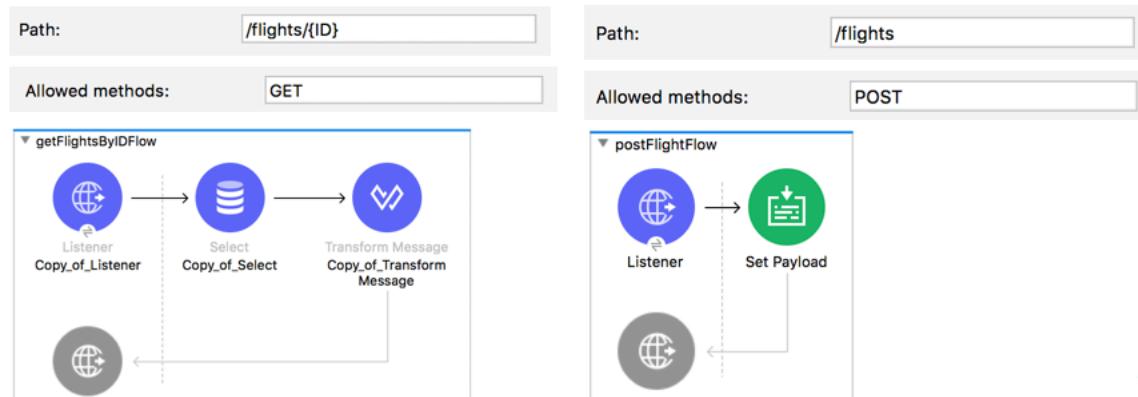
38. Return to Anypoint Studio.

39. Look at the console; you should get a no listener found for request (GET)/flights/3.

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

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

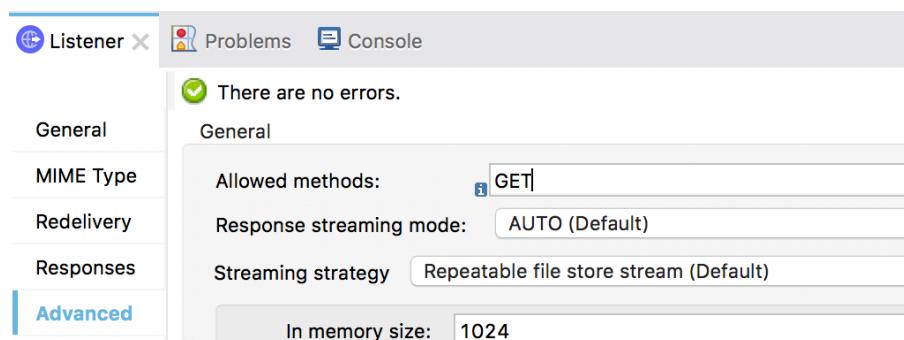
- Route based on path.
- Use a URI parameter in the path of a new HTTP Listener.
- Route based on HTTP method.



40

Restrict method calls to GET

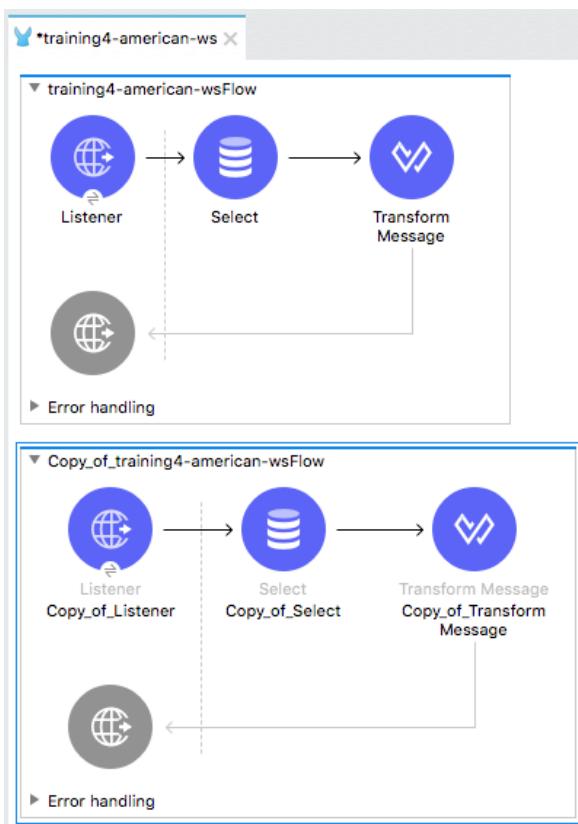
1. Return to Anypoint Studio.
2. Double-click the HTTP Listener in the flow.
3. In the left-side navigation of the Listener properties view, select Advanced.
4. Set the allowed methods to GET.



Make a copy of the existing flow

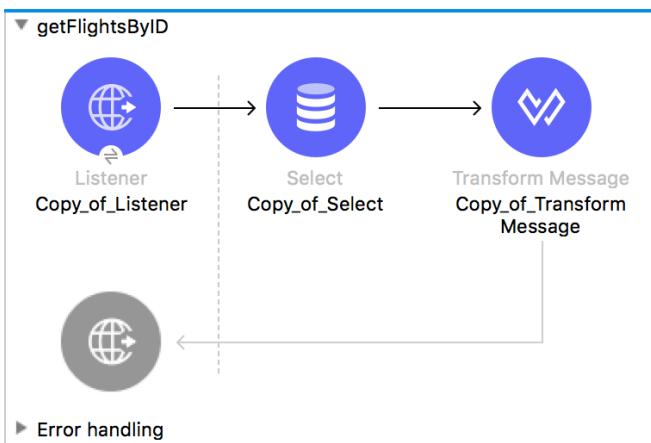
5. Click the flow in the canvas to select it.
6. From the main menu bar, select Edit > Copy.

7. Click in the canvas beneath the flow and select Edit > Paste.



Rename the flows

8. Double-click the first flow.
9. In the properties view, change its name to getFlights.
10. Change the name of the second flow to getFlightsByID.



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

Specify a URI parameter for the new HTTP Listener endpoint

11. Double-click the HTTP Listener in getFlightsByID.
12. Modify the path to have a URI parameter called ID.

The screenshot shows a configuration dialog with a 'General' tab selected. Under the 'Path:' label, there is a text input field containing the value '/flights/{ID}'.

Modify the Database endpoint

13. Double-click the Select operation in getFlightsByID.
14. Modify the query WHERE clause, to select flights with the ID equal to 1.

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

Test the application

15. Save the file to redeploy the project.
16. In Advanced REST Client, make another request to <http://localhost:8081/flights/3>; you should see details for the flight with an ID of 1.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'GET'. The 'Request URL' field contains 'http://localhost:8081/flights/3'. Below the URL, there's a 'Parameters' dropdown set to 'Parameters'. The response status is '200 OK' and the time taken is '813.37 ms'. The response body is displayed as JSON:

```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-19T16:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}]
```

Modify the database query to use the URI parameter

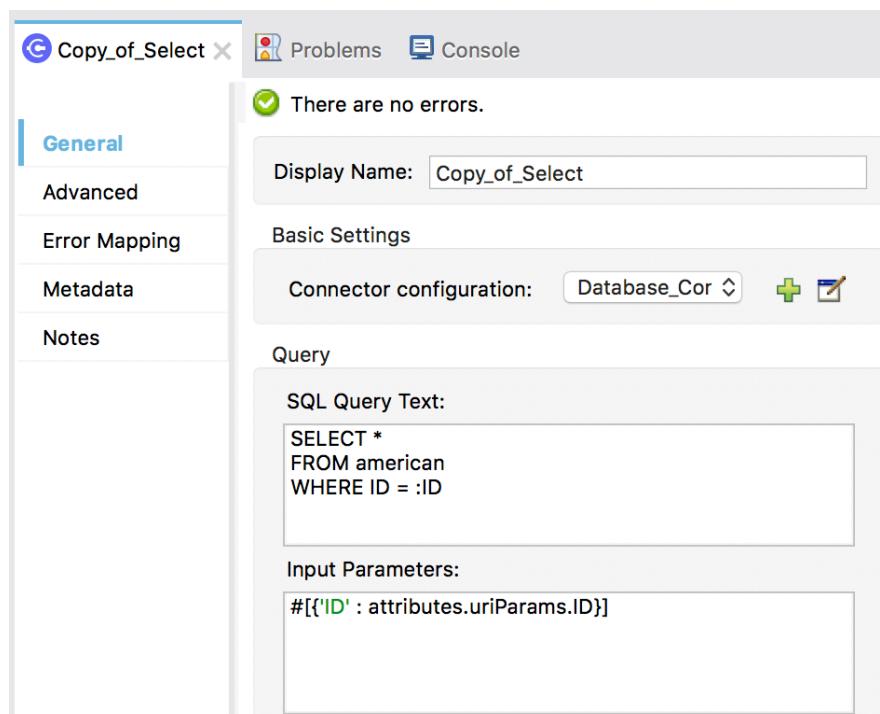
17. Return to the course snippets.txt file and copy the SQL input parameter expression.
18. Return to the getFlightsByID flow in Anypoint Studio.
19. In the Select properties view, locate the Query Input Parameters section and paste the expression you copied.

```
#[{ 'ID' : attributes.uriParams.ID}]
```

Note: You learn to write expressions in a later module in the Development Fundamentals course.

20. Change the WHERE clause in the SQL Query Text to use this input parameter.

```
SELECT *
FROM American
WHERE ID = :ID
```



Test the application

21. Save the file to redeploy the project.

22. In Advanced REST Client, make another request to <http://localhost:8081/flights/3>; you should now see the info for the flight with an ID of 3.

Method: GET Request URL: http://localhost:8081/flights/3

Parameters:

200 OK 704.34 ms DETAILS

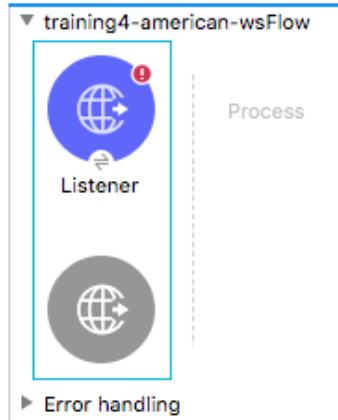
[Array[1]]
-0: {
"ID": 3,
"code": "ffee0192",
"price": 300,
"departureDate": "2016-01-19T16:00:00",

23. Return to Anypoint Studio.

Make a new flow to handle post requests

24. In the Mule Palette, select HTTP.

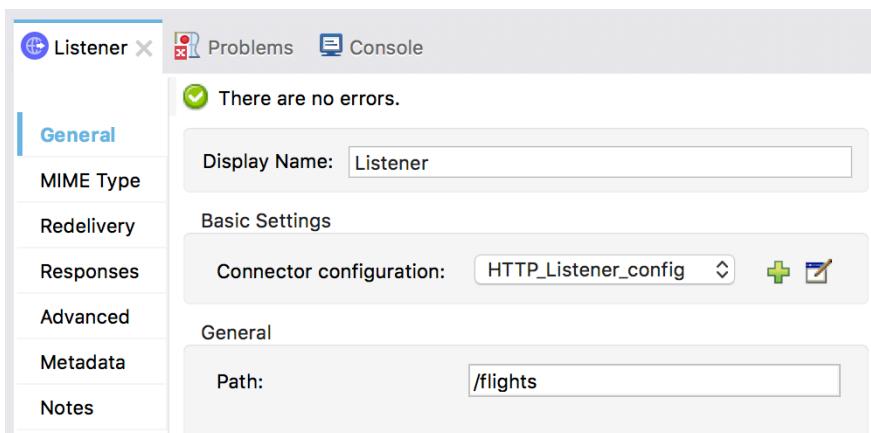
25. Drag Listener from the Mule Palette and drop it in the canvas below the two existing flows.



26. Change the name of the flow to postFlight.

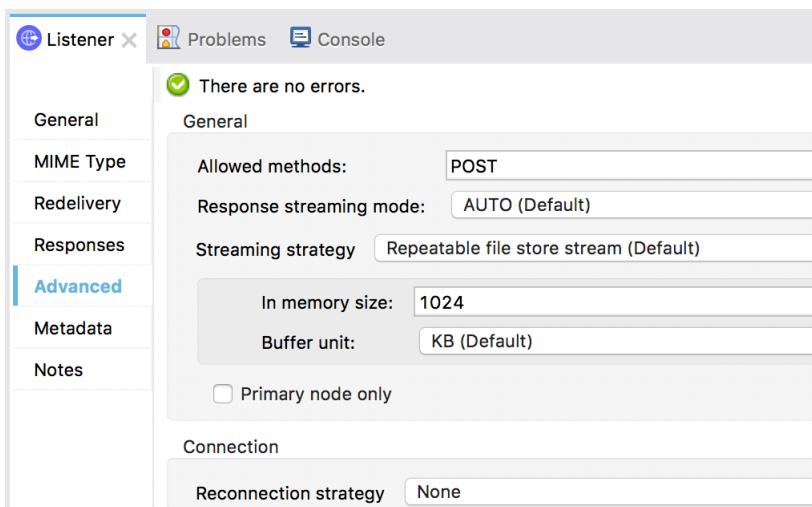
27. In the Listener properties view, set the connector configuration to the existing `HTTP_Listener_config`.

28. Set the path to /flights.

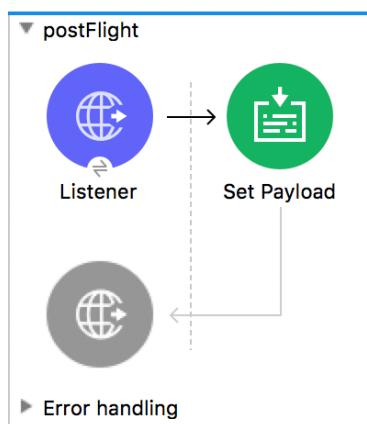


29. In the left-side navigation of the Listener properties view, select Advanced.

30. Set the allowed methods to POST.



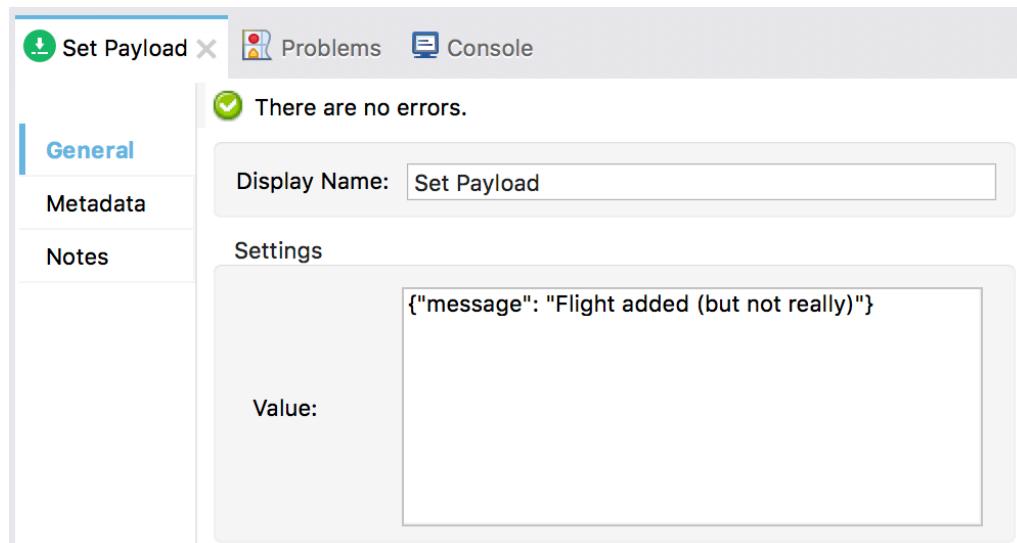
31. Drag the Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



32. Return to the course snippets.txt file and copy the American Flights API - /flights POST response example.

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

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



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

Test the application

34. Save the file to redeploy the project.
35. In Advanced REST Client, change the request type from GET to POST.
36. Click Send; you should get a 405 Method Not Allowed response.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'POST' and the 'Request URL' is 'http://localhost:8081/flights/3'. A large orange button labeled 'SEND' is visible. Below the URL, there is a 'Parameters' dropdown. The response section shows an orange box with the text '405 Method Not Allowed' and '12.83 ms'. At the bottom, the error message 'Method not allowed for endpoint: /flights/3' is displayed, along with several small icons.

37. Remove the URI parameter from the request URL: <http://localhost:8081/flights>.
38. Send the request; you should now see the message the flight was added – even though you did not send any flight data to add.

Method Request URL
POST ▾ <http://localhost:8081/flights> ▾ **SEND** ⋮

Parameters ▾

200 OK 29.53 ms DETAILS ▾

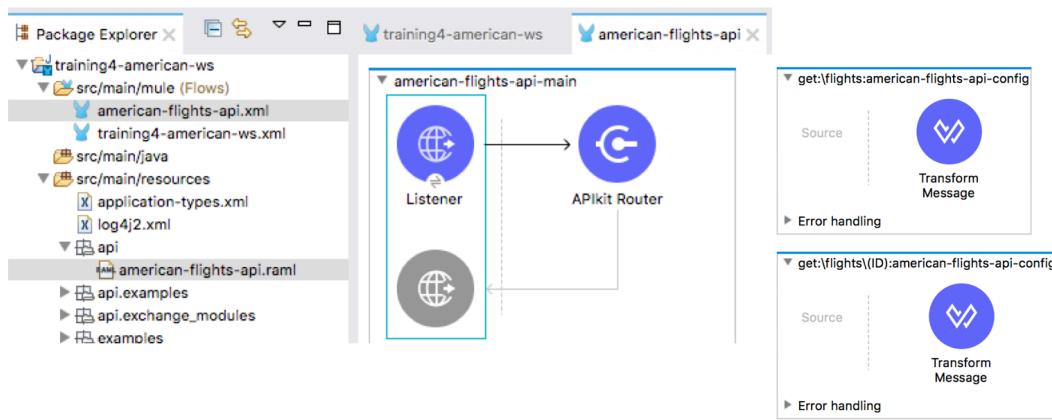
✖️ ↴ ↵ ⌂

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

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

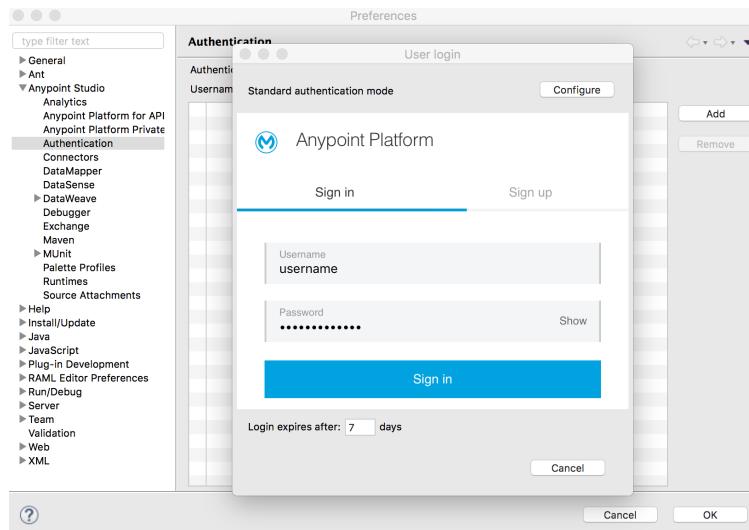
In this walkthrough, you generate a RESTful interface from the RAML file. You will:

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



Add Anypoint Platform credentials to Anypoint Studio

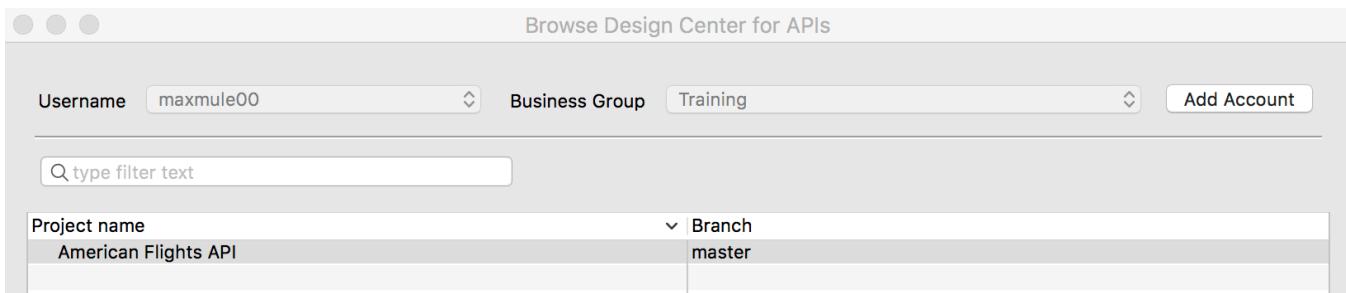
1. In Anypoint Studio, right-click training4-american-ws and select Anypoint Platform > Configure Credentials.
2. In the Authentication page of the Preferences dialog box, click the Add button.
3. In the Anypoint Platform Sign In dialog box, enter your username & password and click Sign In.



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

Add an API from Design Center to the Anypoint Studio project

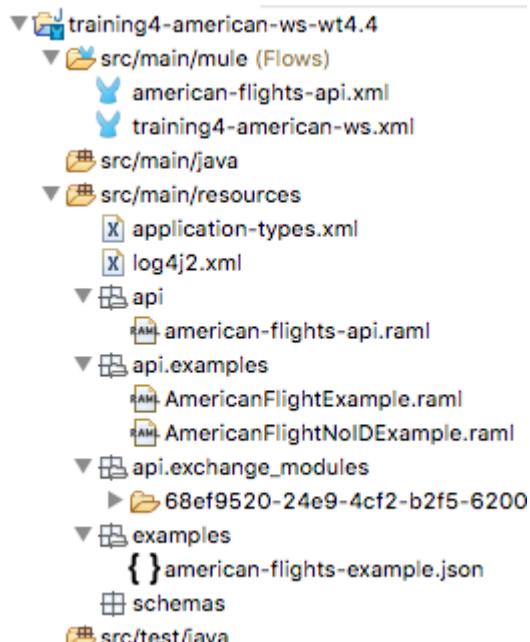
6. In the Package Explorer, locate the src/main/resources/api folder; it should not contain any files.
7. Right-click the folder (or anywhere in the project in the Package Explorer) and select Anypoint Platform > Import from Design Center.
8. In the Browse Design Center for APIs dialog box, select the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



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

Locate the API files added to the project

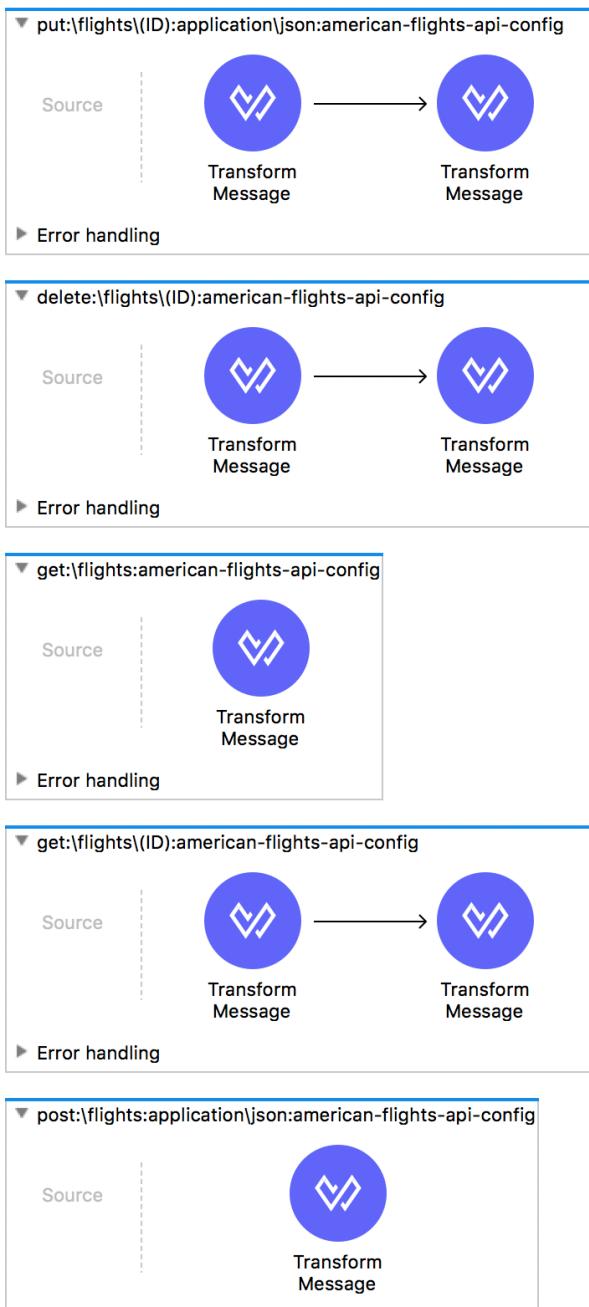
10. In the Package Explorer, locate and expand the src/main/resources folder; it should now contain api folders.
11. Expand the api folder; you should see the RAML file imported from Design Center.



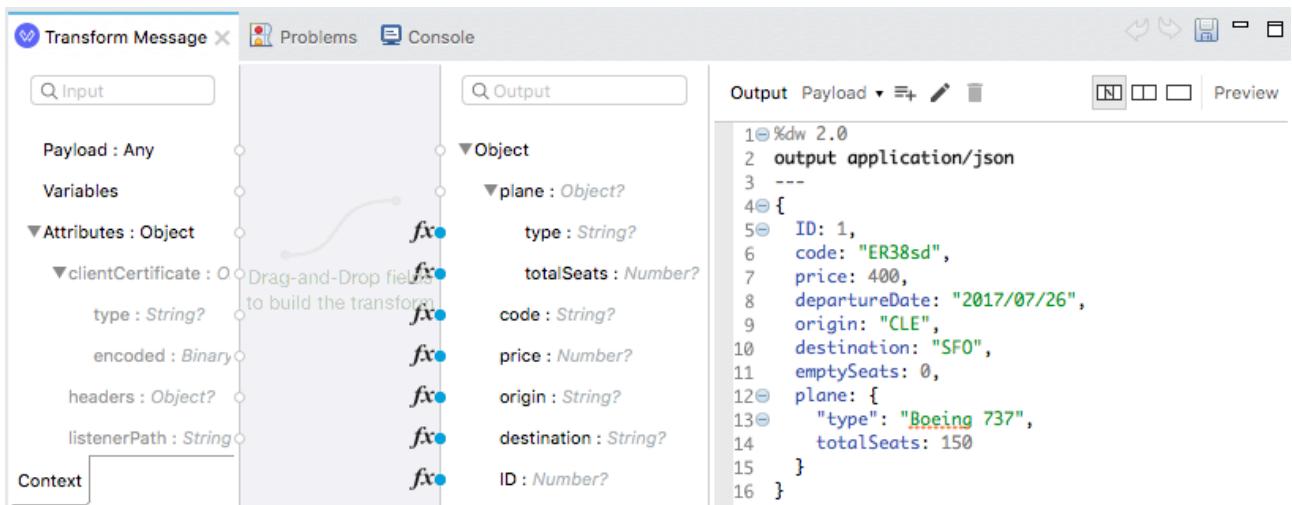
Examine the XML file created

12. Examine the generated american-flights-api.xml file and locate the following five flows:

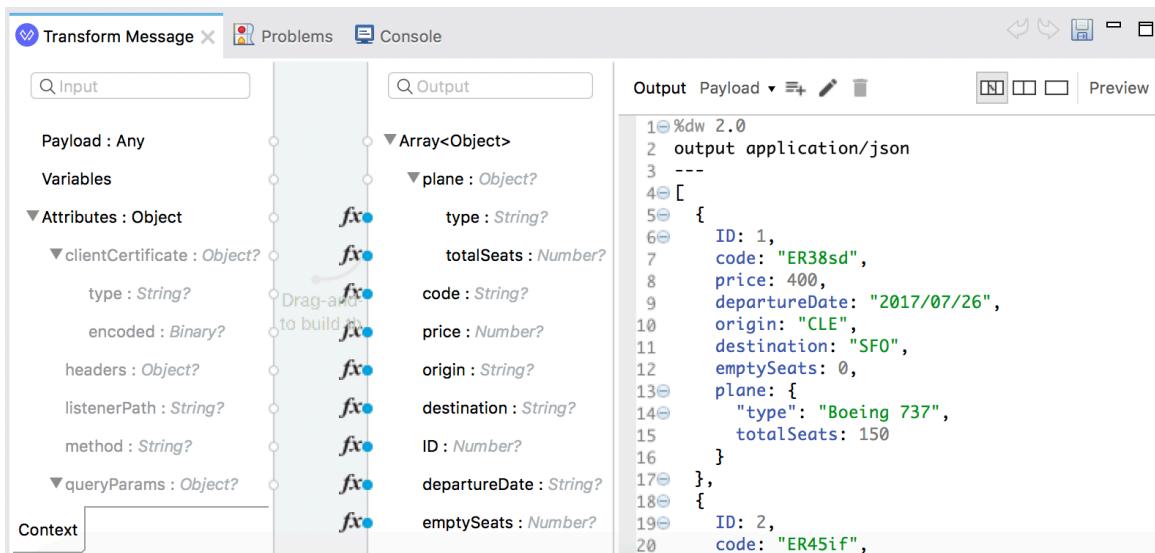
- get:/flights
- get:/flights/{ID}
- post:/flights
- delete:/flights/{ID}
- put:/flights/{ID}



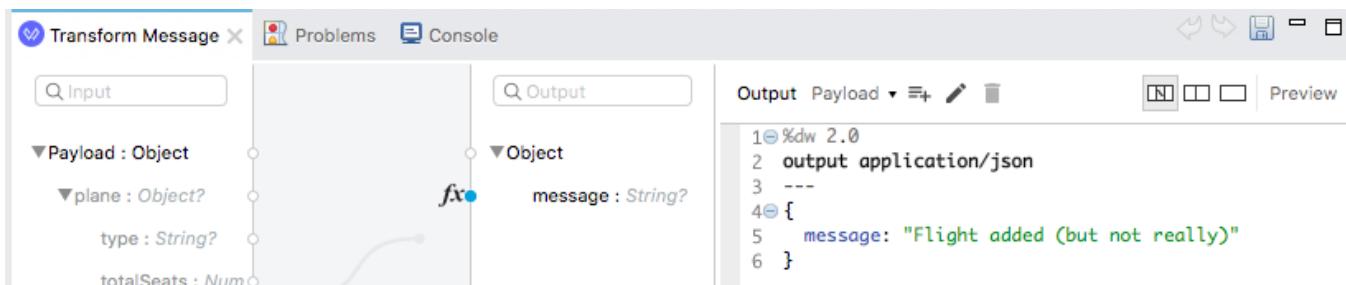
13. In the get:/flights/{ID} flow, double-click the second Transform Message component and look at the value in the properties view.



14. In the get:/flights flow, double-click the Transform Message component and look at the output JSON in the properties view.

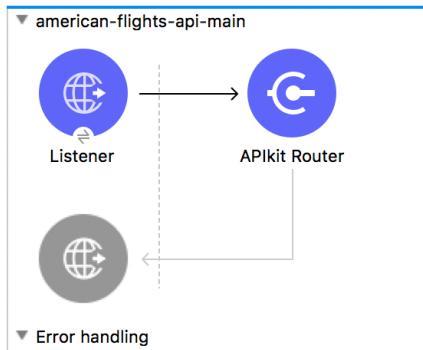


15. In the post:/flights flow, double-click the Transform Message component and look at the output JSON in the properties view.



Examine the main flow and the new HTTP Listener endpoint

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



18. In the Listener properties view, notice that the path is set to /api/*.

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

Listener

General

MIME Type

Redelivery

Responses

Advanced

Metadata

Notes

Display Name: Listener

Basic Settings

Connector configuration: american-flights-api-

General

Path: /api/*

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

Remove the other HTTP configuration and listeners

21. Return to training4-american-ws.xml.
22. In the Global Elements view, select the HTTP Listener config and click Delete.

Global Configuration Elements

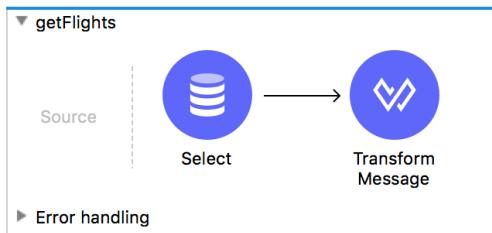
Type	Name	Description
HTTP Listener config (Configuration)	HTTP_Listener_config	
Database Config (Configuration)	Database_Config	

Create

Edit

Delete

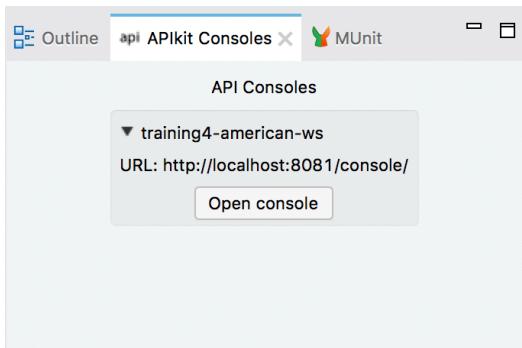
23. Return to the Message Flow view.
24. Right-click the HTTP Listener in getFlights and select Delete.



25. Delete the other two HTTP Listeners.

Test the web service using APIkit console

26. Save the files.
27. Look at the console; you should see the application was not redeployed.
28. Stop the project.
29. Run the project and wait until Mule and the application restart.
30. Locate the new APIkit Consoles view that is created and opened in Anypoint Studio.



31. Click the Open console button; a browser window should be open with an API console.

The screenshot shows a browser window with the following details:

- Title bar: localhost:8081/console/
- Main header: American Flights API
- Left sidebar (API summary):
 - Types: Version: v1, Supported protocols: HTTP
 - Resources:
 - /flights: API base URI http://localhost:8081/api/
 - Method buttons: GET, POST
 - Resource details: /flights, /{ID}

32. Select the GET method and click the TRY IT button.

The screenshot shows the API console interface for the American Flights API. The top navigation bar says "API console" and "American Flights API". On the left, there's a sidebar with "API summary", "Types", "Resources" (with "/flights" expanded), and a "Request" section. The main area shows the endpoint "/flights : get" with a "TRY IT" button. Below it, under "Request", is the URL "GET http://localhost:8081/api/flights". The "Parameters" section shows a table with one row: "destination" (string enum) with possible values SFO, LAX, CLE. The "GET" button is highlighted in blue.

33. Click Send; you should get a 200 response with the example flight data – not all the flights.

The screenshot shows the API console interface for the American Flights API. The top navigation bar says "API console" and "American Flights API". The sidebar shows "API summary", "Types", "Resources" (with "/flights" expanded), and a "Request" section. The "Request URL" is set to "http://localhost:8081/api/flights". The "Parameters" section shows a table with one row: "destination" (string enum) with possible values SFO, LAX, CLE. The "GET" button is highlighted in blue. The "SEND" button is visible on the right. The response section shows a green "200 OK" status with a "417.50 ms" latency. Below it, there's a JSON preview of the response data:

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26"
}...]
```

34. Close the browser window.

Test the web service using Advanced REST Client

35. Return to Advanced REST Client.
36. Change the method to GET and click Send to make a request to <http://localhost:8081/flights>; you should get a 404 Not Found response.

Method Request URL
GET <http://localhost:8081/flights>

[SEND](#) [⋮](#)

Parameters [▼](#)

404 Not Found 85.23 ms

[DETAILS](#) [▼](#)



No listener for endpoint: /flights

37. Change the URL to <http://localhost:8081/api/flights> and send the request; you should get a 200 response with the example flight data.

200 OK 31.10 ms

[DETAILS](#) [▼](#)



```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
},
-1: {
  "ID": 2,
  "code": "ER45if",
```

38. Make a request to <http://localhost:8081/api/flights/3>; you should see the example data returned for a flight with an ID of 1.

200 OK 45.04 ms DETAILS ▾

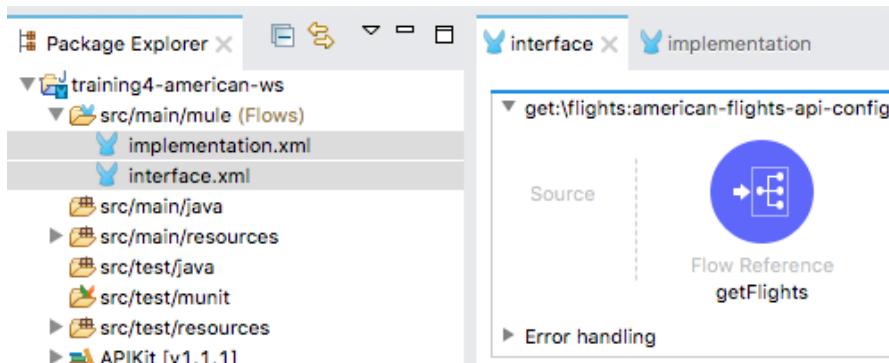
□ ▾ <> ⌂

```
{  
    "ID": 1,  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2017/07/26"
```

Walkthrough 4-6: Implement a RESTful web service

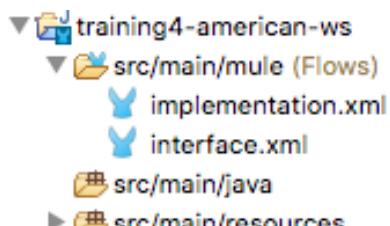
In this walkthrough, you wire the RESTful web service interface up to your back-end logic. You will:

- Pass an event from one flow to another.
- Call the backend flows.
- Create new logic for the nested resource call.
- Test the web service using Advanced REST Client.



Rename the configuration files

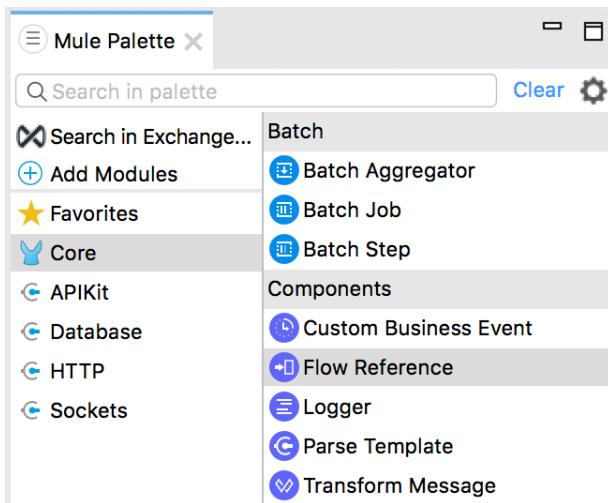
1. Return to Anypoint Studio.
2. Right-click `american-flights-api.xml` in the Package Explorer and select Refactor > Rename.
3. In the Rename Resource dialog box, set the new name to `interface.xml` and click OK.
4. Right-click `training4-american-ws.xml` and select Refactor > Rename.
5. In the Rename Resource dialog box, set the new name to `implementation.xml` and click OK.



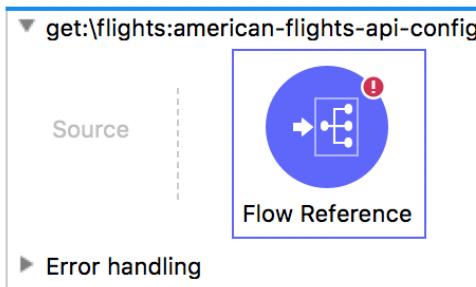
Use a Flow Reference in the /flights resource

6. Open `interface.xml`.
7. Delete the Transform Message component in the `get:/flights` flow.

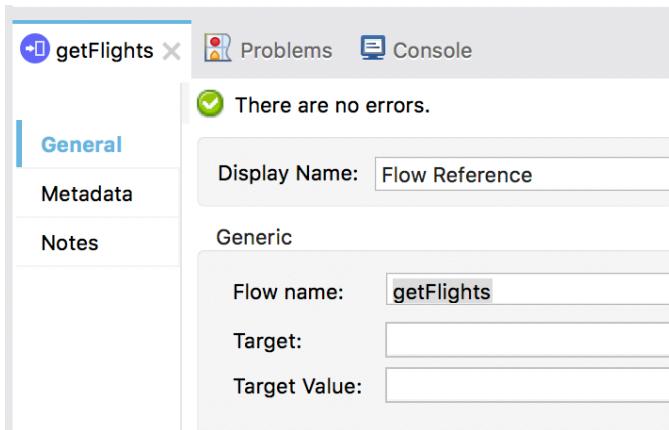
8. In the Mule Palette, select Core and locate the Components section in the right-side.



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



10. In the Flow Reference properties view, select getFlights for the flow name.

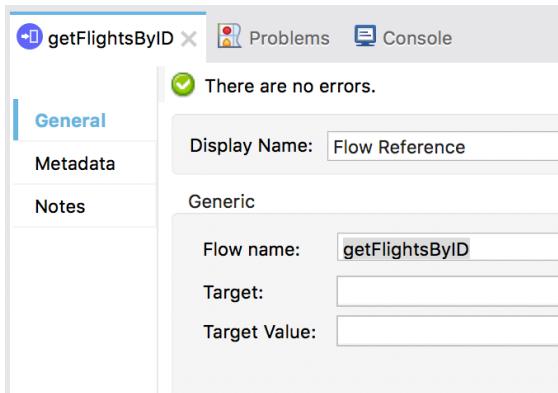


11. Change the display name to getFlights.

Use a Flow Reference in the /flights/{ID} resource

12. Delete both the Transform Message components in the get:/flights/{ID} flow.

13. Drag a Flow Reference component from the Mule Palette and drop it into the flow.
14. In the Flow Reference properties view, select getFlightsByID for the flow name.



15. Change the display name to getFlightsByID.

Test the web service using Advanced REST Client

16. Save the file to redeploy the project.
17. In Advanced REST Client, make a request to <http://localhost:8081/api/flights>; you should now get the data for all the flights from the database instead of the sample data.

The screenshot shows a successful API call with a status of '200 OK' and a response time of '2036.90 ms'. The response body is a JSON array of flight data:

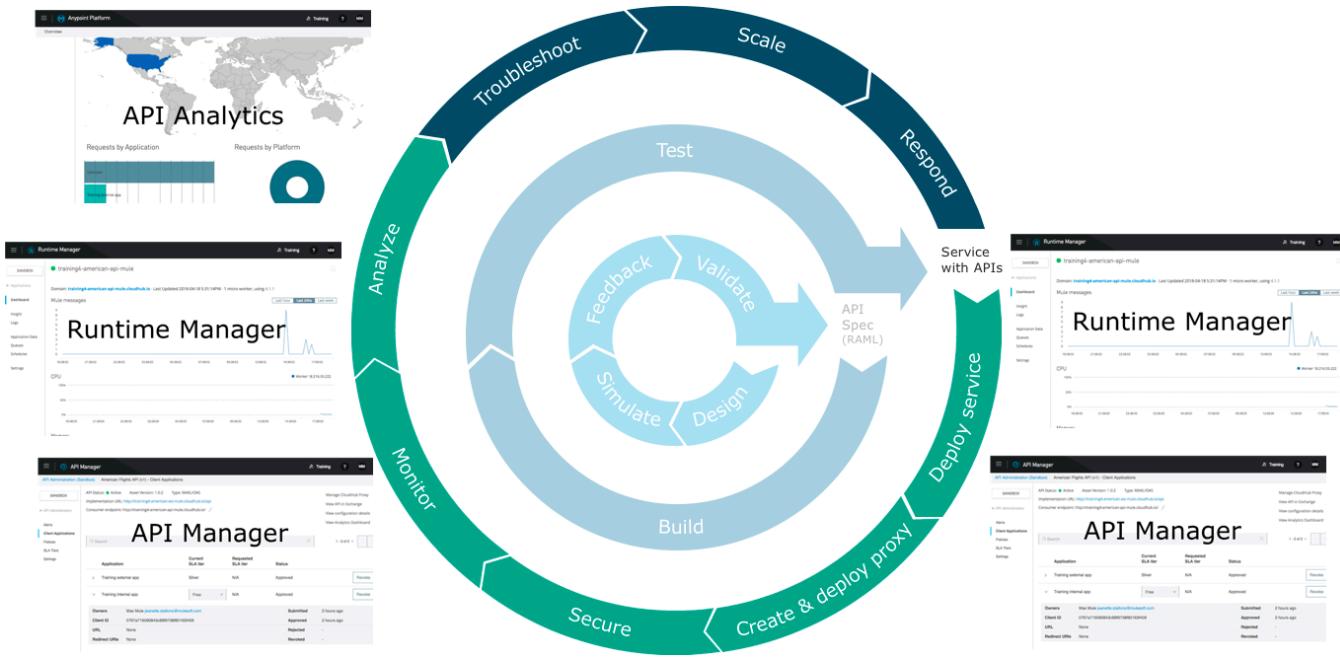
```
[Array[11]
 -0: { ... }
 -1: { ... }
 -2: { ... }
 -3: { ... }
 -4: { ... }
 -5: { ... }
 -6: { ... }
 -7: { ... }
 -8: { ... }
 -9: { ... }
 -10: {
   "ID": 11,
   "code": "rree4567",
   "price": 456,
   "departureDate": "2016-01-19T16:00:00",
   "origin": "MUA",
   "destination": "SFO",
   "emptySeats": 100,
   "plane": {
     "type": "Boeing 737",
     "totalSeats": 150
   }
 }]
```

18. Make a request to <http://localhost:8081/api/flights/3>; you should now get the data that flight from the database instead of the sample data.

19. Return to Anypoint Studio.

20. Stop the project.

Module 5: Deploying and Managing APIs



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

- Describe the options for deploying Mule applications.
- Deploy Mule applications to CloudHub.
- Use API Manager to create and deploy API proxies.
- Use API Manager to restrict access to API proxies.

Walkthrough 5-1: Deploy an application to CloudHub

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

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

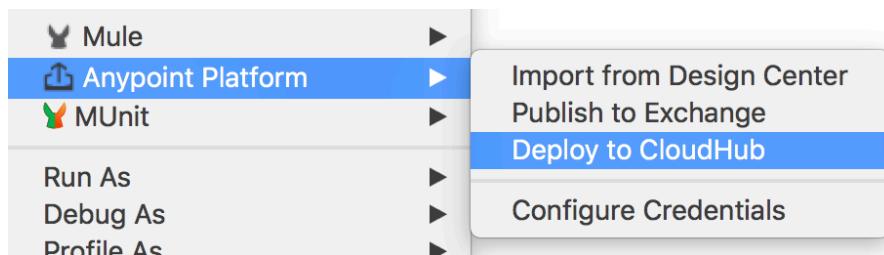
The screenshot shows the 'Runtime Manager' interface. On the left, there's a sidebar with tabs for 'SANDBOX', 'Applications' (which is selected), 'Servers', and 'Alerts'. The main area has a 'Deploy application' button and a search bar. A table lists one application: 'training4-american-ws-mule' running on 'CloudHub' with a 'Started' status and the file 'training4-american-ws-v2.jar'.

Name	Server	Status	File
training4-american-ws-mule	CloudHub	● Started	training4-american-ws-v2.jar

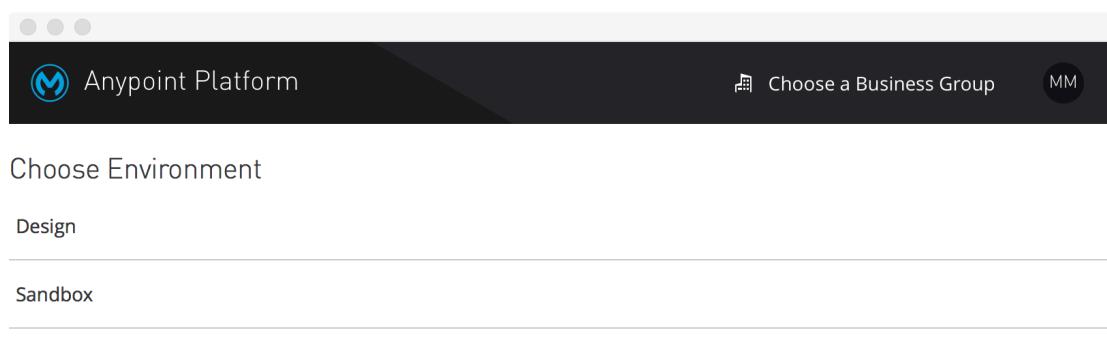
Note: If you do not have a working application at this point, import the training4-american-ws-wt4-6.jar solution into Anypoint Studio and work with that project.

Deploy the application to CloudHub

1. Return to Anypoint Studio.
2. In the Package Explorer, right-click the project and select Anypoint Platform > Deploy to CloudHub.

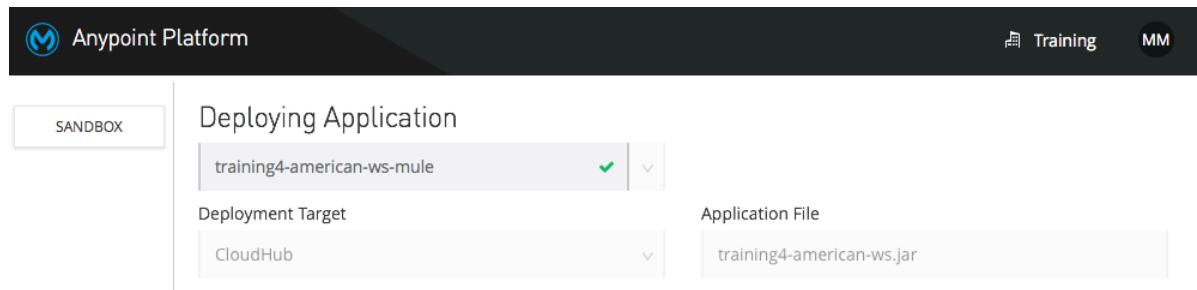


3. In the Choose Environment dialog box, select Sandbox.



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.

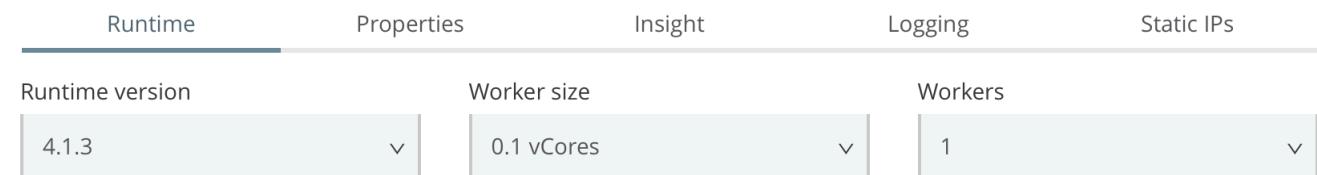


The screenshot shows the 'Deploying Application' dialog box. The application name dropdown contains 'training4-american-ws-mule' with a green checkmark. The deployment target dropdown is set to 'CloudHub'. The application file dropdown contains 'training4-american-ws.jar'.

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.

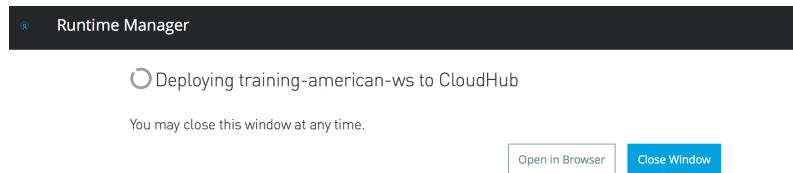
6. Make sure the worker size to 0.1 vCores.



Runtime	Properties	Insight	Logging	Static IPs
Runtime version 4.1.3	Worker size 0.1 vCores	Workers 1		

7. Click the Deploy Application button.

8. Click the Open in Browser button.



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

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

The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'Sandbox' selected. The main area has tabs for 'Deploy application' and 'Search Applications'. A table lists an application named 'training4-american-ws-mule' running on 'CloudHub' with a status of 'Deploying'. The file path is listed as 'training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar'.

Watch the logs and wait for the application to start

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

The screenshot shows the Runtime Manager interface. The sidebar now includes 'Applications', 'Servers', 'Alerts', 'VPCs', and 'Load Balancers'. The main table shows the application 'training4-american-ws-mule' on 'CloudHub' with a status of 'Started'. On the right, a detailed view for 'training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar' is displayed, showing it was last updated on 2018-04-18 at 1:48:21PM and has an app URL of 'training4-american-ws-mule.cloudhub.io'. Configuration details like Runtime version (4.1.1), Worker size (0.1 vCores), and Workers (1) are shown. Buttons for 'Manage Application', 'Logs', and 'Insight' are at the bottom.

- Click the Logs button.
- Watch the logs as the application is deployed.

13. Wait until the application starts (or fails to start).

The screenshot shows the Mule Runtime Manager interface. On the left, a sidebar navigation includes options like Applications, Dashboard, Insight, Logs (which is selected), Application Data, Queues, Schedules, and Settings. The main area displays the logs for the application "training4-american-ws-mule". The logs show the application starting up, with messages indicating it has started successfully. To the right, a "Deployments" panel shows a deployment log for "13:46 - Deployment" on "Worker-0", with entries for the system log.

```
Starting Bean: listener
13:48:21.870 04/18/2018 Worker-0 qtp1298881551-39 INFO
*****
* Application: training4-american-ws-mule
* OS encoding: UTF-8, Mule encoding: UTF-8
*
*****
13:48:21.955 04/18/2018 Deployment system SYSTEM
Worker(18.217.62.85): Your application has started successfully.

13:48:22.622 04/18/2018 Deployment system SYSTEM
Your application is started.
```

Deployments

Today

13:46 - Deployment

System Log

Worker-0

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

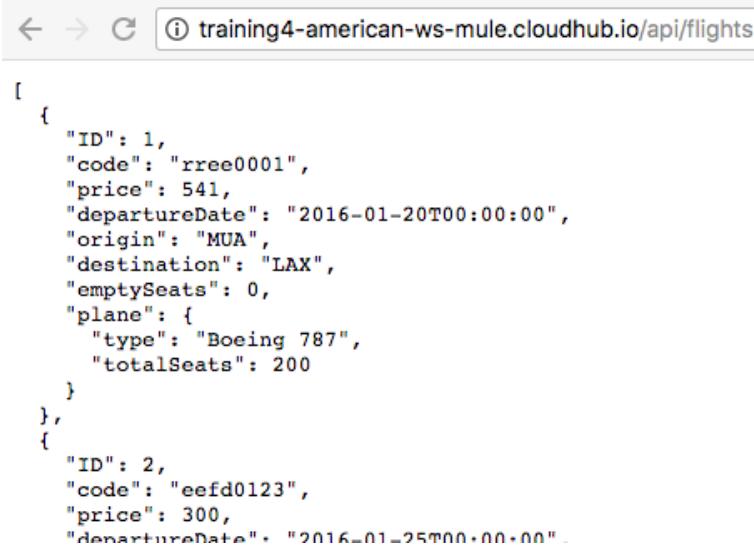
14. In the left-side navigation, select Dashboard.
15. Locate the link for the application on its new domain:
training4-american-ws-{lastname}.{region}.cloudbhub.io.

The screenshot shows the Mule Runtime Manager dashboard for the application "training4-american-ws-mule". It displays the application name, the deployed domain ("Domain: training4-american-ws-mule.cloudbhub.io"), and a "Mule messages" section.

Note: {region} represents the worker region to which the Mule application is deployed. In North America, the default region is US East and is denoted by us-e2 in the application URL.

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-{lastame}.cloudbu.io/api/flights>; you should see the flights data.

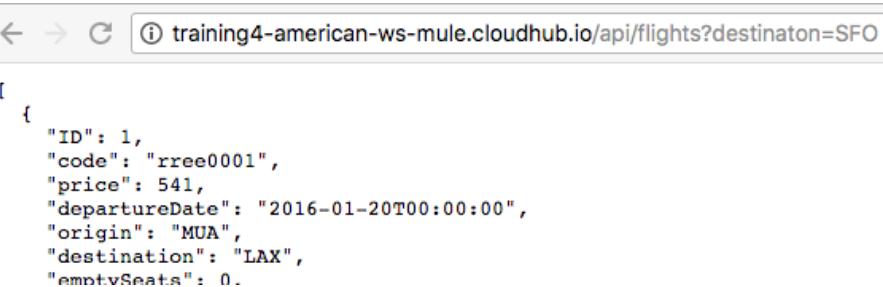


```
[  
  {  
    "ID": 1,  
    "code": "rree0001",  
    "price": 541,  
    "departureDate": "2016-01-20T00:00:00",  
    "origin": "MUA",  
    "destination": "LAX",  
    "emptySeats": 0,  
    "plane": {  
      "type": "Boeing 787",  
      "totalSeats": 200  
    }  
  },  
  {  
    "ID": 2,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

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

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

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



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

20. Leave this browser tab open.

Update the API implementation deployed to CloudHub

21. Return to the browser tab with Runtime Manager.
22. In the left-side navigation, select Settings.
23. Click the Choose file button and select Upload file.
24. Browse to the jars folder in the course student files.

25. Select training4-american-ws-v2.jar and click Open.

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

26. Click the Apply Changes button.

The screenshot shows the Runtime Manager interface. On the left, a sidebar menu includes options like Applications, Dashboard, Insight, Logs, Application Data, Queues, Schedules, and Settings (which is currently selected). The main area displays the application 'training4-american-ws-mule'. It has an 'Application File' section with a file input field containing 'training4-american-ws-v2.jar', a 'Choose file' button, a 'Get from sandbox' button, and a 'Stop' button. Below this is an 'App url' field with the value 'training4-american-ws-mule.cloudhub.io'. There are tabs for 'Runtime', 'Properties', 'Insight', 'Logging', and 'Static IPs', with 'Runtime' being active. Under 'Runtime', there are dropdowns for 'Runtime version' (set to 4.1.1), 'Worker size' (set to 0.1 vCores), and 'Workers' (set to 1). A warning message states '⚠ Your current subscription allows only one worker per application'. Below these are checkboxes for 'Automatically restart application when not responding' (checked), 'Persistent queues' (unchecked), and 'Encrypt persistent queues' (unchecked). At the bottom right is a large blue '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.



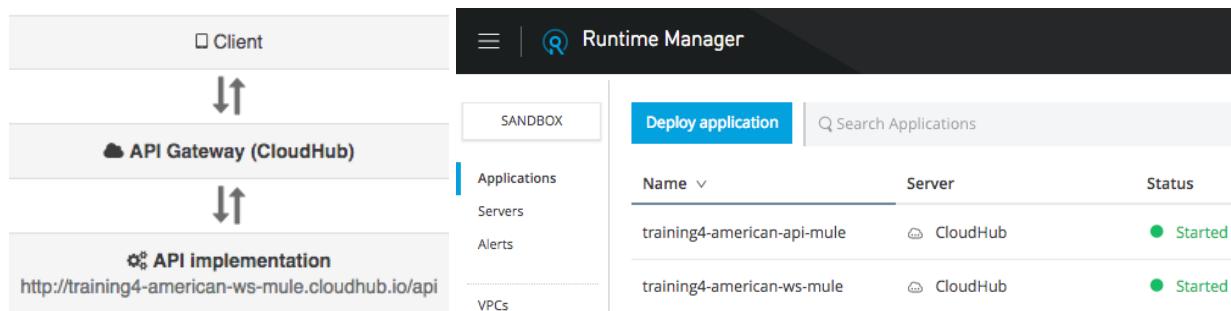
```
[  
  {  
    "ID": 5,  
    "code": "rree1093",  
    "price": 142,  
    "departureDate": "2016-02-11T00:00:00",  
    "origin": "MUA",  
    "destination": "SFO",  
    "emptySeats": 1,  
    "plane": {  
      "type": "Boeing 737",  
      "totalSeats": 150  
    }  
  },  
  {  
    "ID": 7,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

30. Close this browser tab.

Walkthrough 5-2: Create and deploy an API proxy

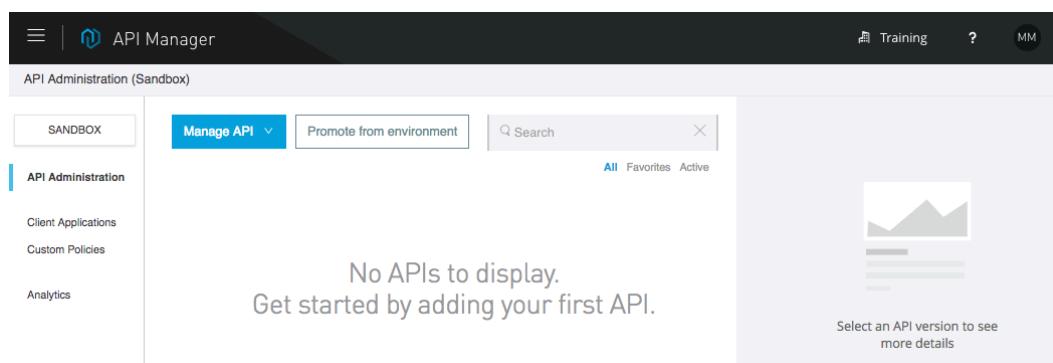
In this walkthrough, you create and deploy an API proxy for your API implementation on CloudHub. You will:

- 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.

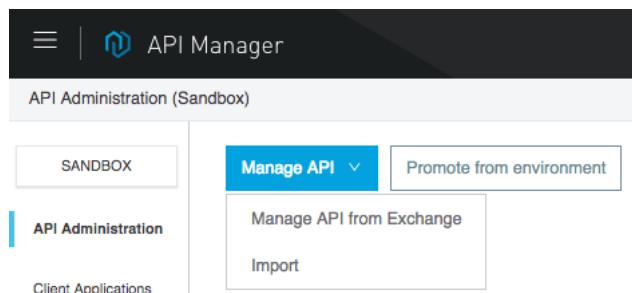


Create and deploy a proxy application

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}.{region}.cloudbus.io/api`
 - Proxy deployment target: CloudHub

The screenshot shows the MuleSoft API Manager interface. The top navigation bar includes 'Training' and 'MM' buttons. The main header says 'API Manager'. Below it, 'API Administration (Sandbox)' and 'Get from Exchange' are visible. A sidebar on the left has a 'SANDBOX' tab selected. The main content area is titled 'Manage API from Exchange' under 'API Configurations'. It contains the following fields:

- API name:** American Flights API
- Asset type:** RAML/OAS
- API version:** v1
- Asset version:** 1.0.1
- Managing type:** Endpoint with Proxy (selected)
- Implementation URI:** `http://training4-american-ws-mule.cloudbus.io/api`
- Proxy deployment target:** CloudHub (selected)
- Path:** /
- Check this box if you are managing this API in Mule 4 or above.** (checkbox checked)

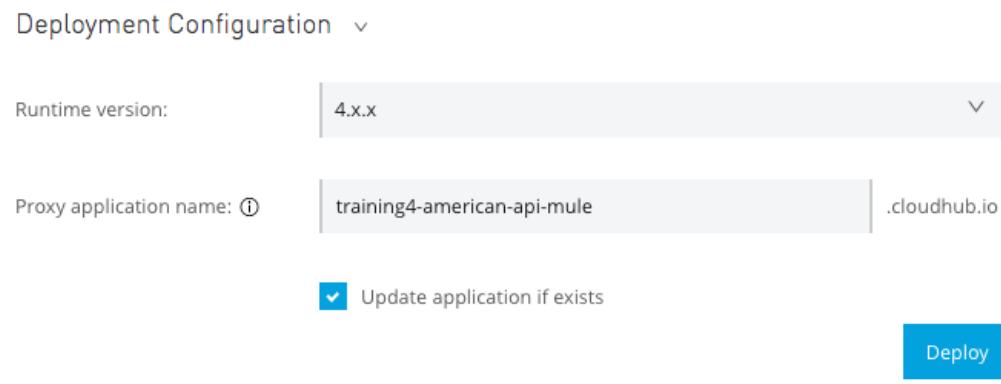
6. Select the checkbox: Check this box if you are managing this API in Mule 4 or above.

Path: /

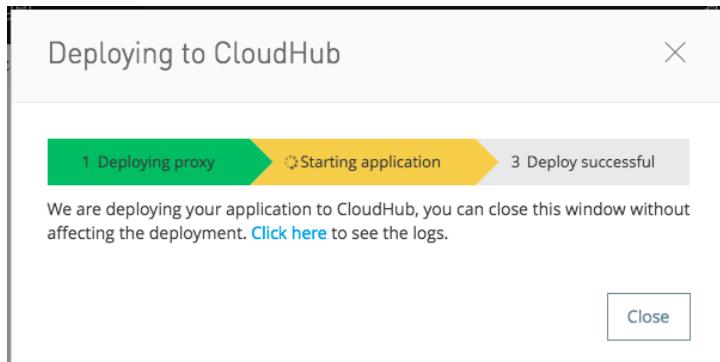
Check this box if you are managing this API in Mule 4 or above.

[Advanced options >](#)

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



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

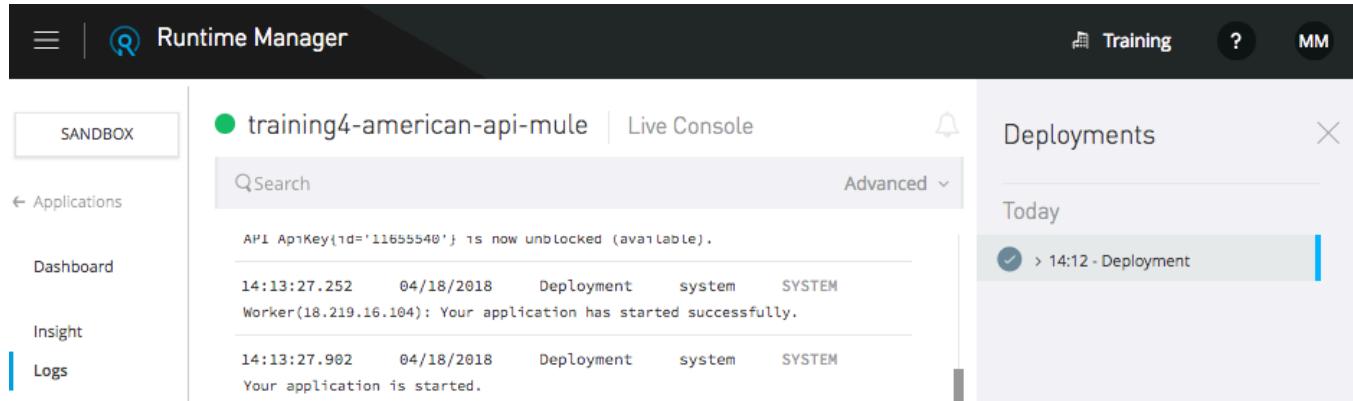


12. In the new browser tab that opens, watch the logs in Runtime Manager.

The screenshot shows the 'Runtime Manager' interface. The left sidebar has tabs for 'Sandbox' (selected), 'Applications' (with a back arrow), and 'Dashboard'. The main area shows the application 'training4-american-api-mule' with a 'Live Console' button. Below it is a search bar and an 'Advanced' dropdown. The deployment log table shows a single row: 14:12:08.656 04/18/2018 Deployment system SYSTEM Deploying application to 1 workers. To the right is a 'Deployments' sidebar with a 'Today' section showing a recent deployment entry with a checkmark and the timestamp 14:12 - Deployment.

13. Wait until the proxy application starts.

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



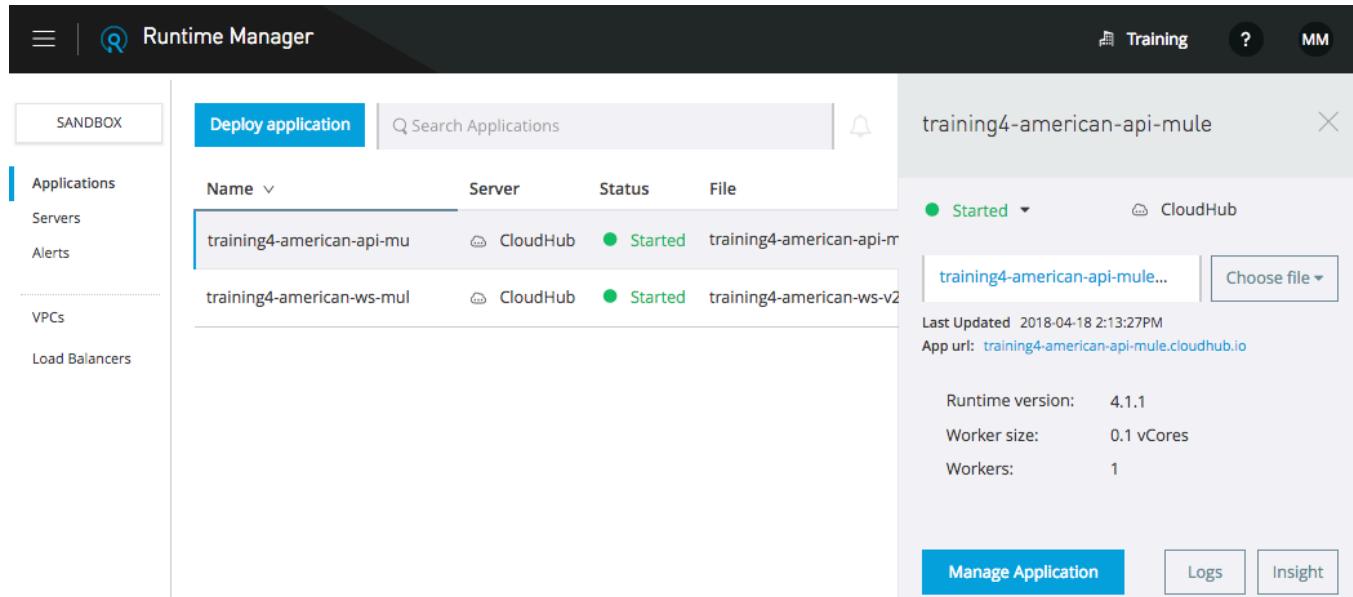
The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'Sandbox' selected, followed by 'Applications', 'Dashboard', 'Insight', and 'Logs'. The main area has a title '● training4-american-api-mule | Live Console'. It includes a search bar and an 'Advanced' dropdown. Below the search bar, the logs show:

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

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

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

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



The screenshot shows the Runtime Manager interface. The left sidebar has 'Applications' selected, along with 'Servers', 'Alerts', 'VPCs', and 'Load Balancers'. The main area has a title 'Deploy application' and a search bar. A table lists applications:

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

The right panel shows details for the application 'training4-american-api-mule':

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

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

16. Close the browser tab.

View API details in API Manager

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

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

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

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

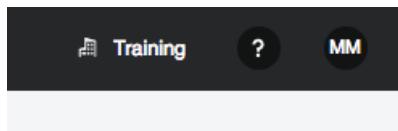
20. Click in the row for the v1 version – but not on the v1 link.

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

The screenshot shows the API Manager API Administration list. The left sidebar includes 'Sandbox', 'Manage API', 'Promote from environment', a search bar, and navigation buttons. The 'API Administration' link is selected. The main table lists the 'American Flights API' with one version, 'v1'. The table columns are API Name, Version, Status, Client Applications, and Creation Date. The 'v1' row shows it is Active and was created on 04-18-2018 14:11. To the right of the table, there is a summary for 'American Flights API v1' with links for 'Manage CloudHub Proxy', 'View API in Exchange', and 'View Analytics Dashboard'. Below these are tabs for 'Applications', 'Policies', and 'SLA tiers', with a note stating 'There are no policies configured for this API version.'

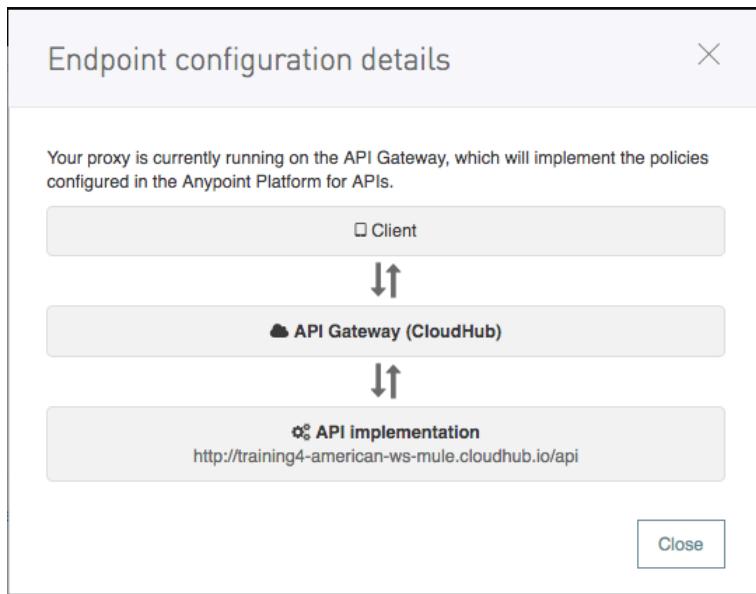
22. In the API list, click the v1 link for the API; you should be returned to the Settings page for the API.

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



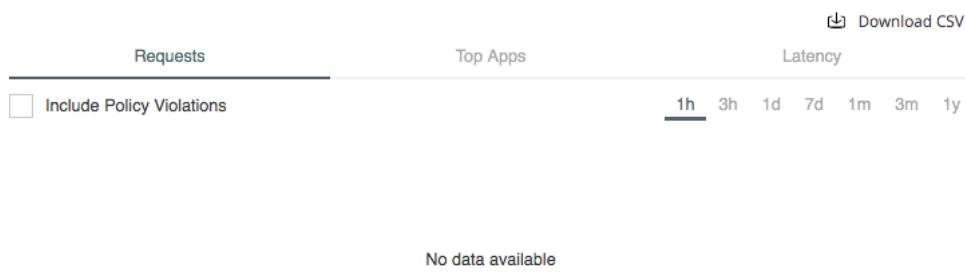
- Manage CloudHub Proxy >
- View API in Exchange >
- View configuration details >
- View Analytics Dashboard >

24. Click the View configuration details link.



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

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



View the new API proxy instance in Exchange

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

Version	Instances
1.0.1	Mocking Service Sandbox - v1:5831632
1.0.0	

Note: You may need to refresh your browser page.

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

GET

Mocking Service

Mocking Service

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

Assets list

American Flights API

API summary

> Types

▽ Resources

▽ /flights

 GET

 POST

 > /{ID}

API instances

Instances	Environment	URL	Visibility
Mocking Service		https://mocksvc-proxy.anypoint.mulesoft.com/exchange/800b1585-b6be-4c62-82de-02d8c2adf413/american-flights-api/1.0.1	Public
v1:5831632	Sandbox		Private

+ Add new instance

Set a friendly label for the API instance in API Manager

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

API Instance ⓘ

ID: 5831632

Label: No policy 

Set a consumer endpoint for the proxy in API Manager

36. Locate the proxy URL.

Proxy

Proxy Application: training4-american-api-mule

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

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

38. Click the Add consumer endpoint link.

American Flights API v1

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

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

39. Paste the value of the proxy URL.

40. Press Enter/Return.

American Flights API v1

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

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

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

Make requests to the API proxy from Exchange

41. Return to the browser tab with Exchange.

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

The screenshot shows the Exchange interface with the 'American Flights API' selected. On the left, the navigation sidebar shows 'Assets list', 'American Flights API', 'API summary', 'Types', 'Resources', '/flights' (selected), 'GET' (highlighted), 'POST', and '/{ID}'. The main area displays the 'American Flights API' details with version v1 Public. Below is a table of API instances:

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

Buttons for '+ Add new instance' and a pencil icon are also visible.

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

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

The screenshot shows the 'Asset versions for v1' page. It lists two instances under version 1.0.1: 'Mocking Service' (Public) and 'Sandbox - No policy' (Private). A dropdown menu icon is shown next to each instance.

Version	Instances
1.0.1	Mocking Service Sandbox - No policy
1.0.0	

45. Click the GET method for the flights resource.

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

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

47. Select the Sandbox - No policy instance.

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



200 OK | 9102.50 ms | Details ▾

Array[11]

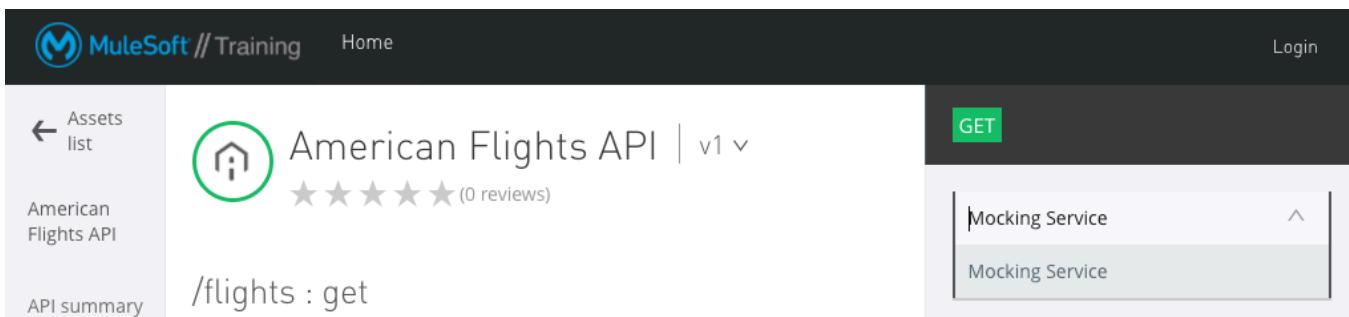
- 0: { ... }
- 1: { ... }
- 2: { ... }
- 3: {
 "ID": 4,
 "code": "rree1000",
 "price": 200,
 "departureDate": "2016-01-
 20T00:00:00",
 "origin": "MUA",
 "destination": "CLE",
 "emptySeats": 5,
 "-plane": {
 "type": "Boeing 737",
 "totalSeats": 150
 }
},
-4: {
 "ID": 5,

49. Make several more calls to this endpoint.

50. Make calls to different methods.

Make requests to the API proxy from the public portal

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



MuleSoft // Training | Home | Login

Assets list | American Flights API | v1 ▾

American Flights API | /flights : get

GET | Mocking Service | Mocking Service

Make an API instance visible in the public portal

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

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

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

[+ Add new instance](#)

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

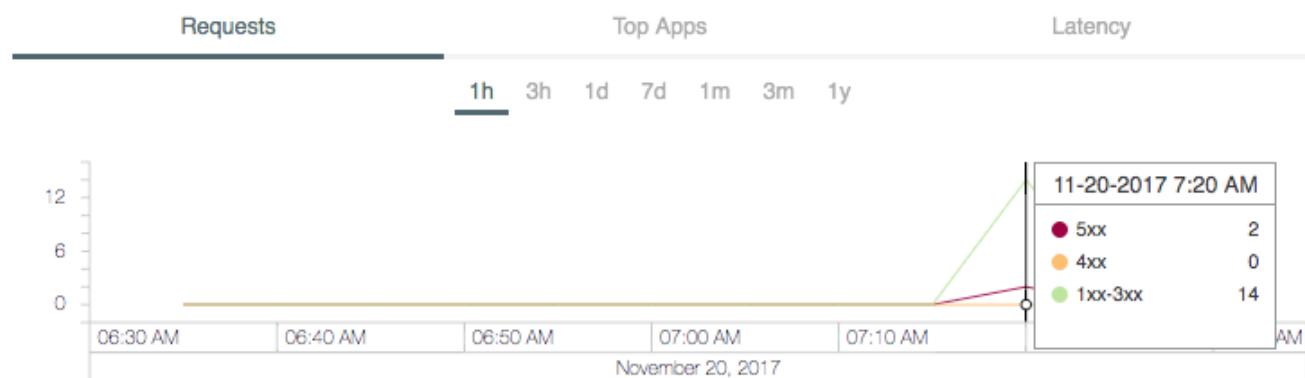
The screenshot shows an API request configuration. The method is 'GET'. The URL is <http://training4-american-api-mule.cloudhub.io/flights>. The 'Instance' dropdown is set to 'Sandbox - No policy'.

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

Look at the API request data

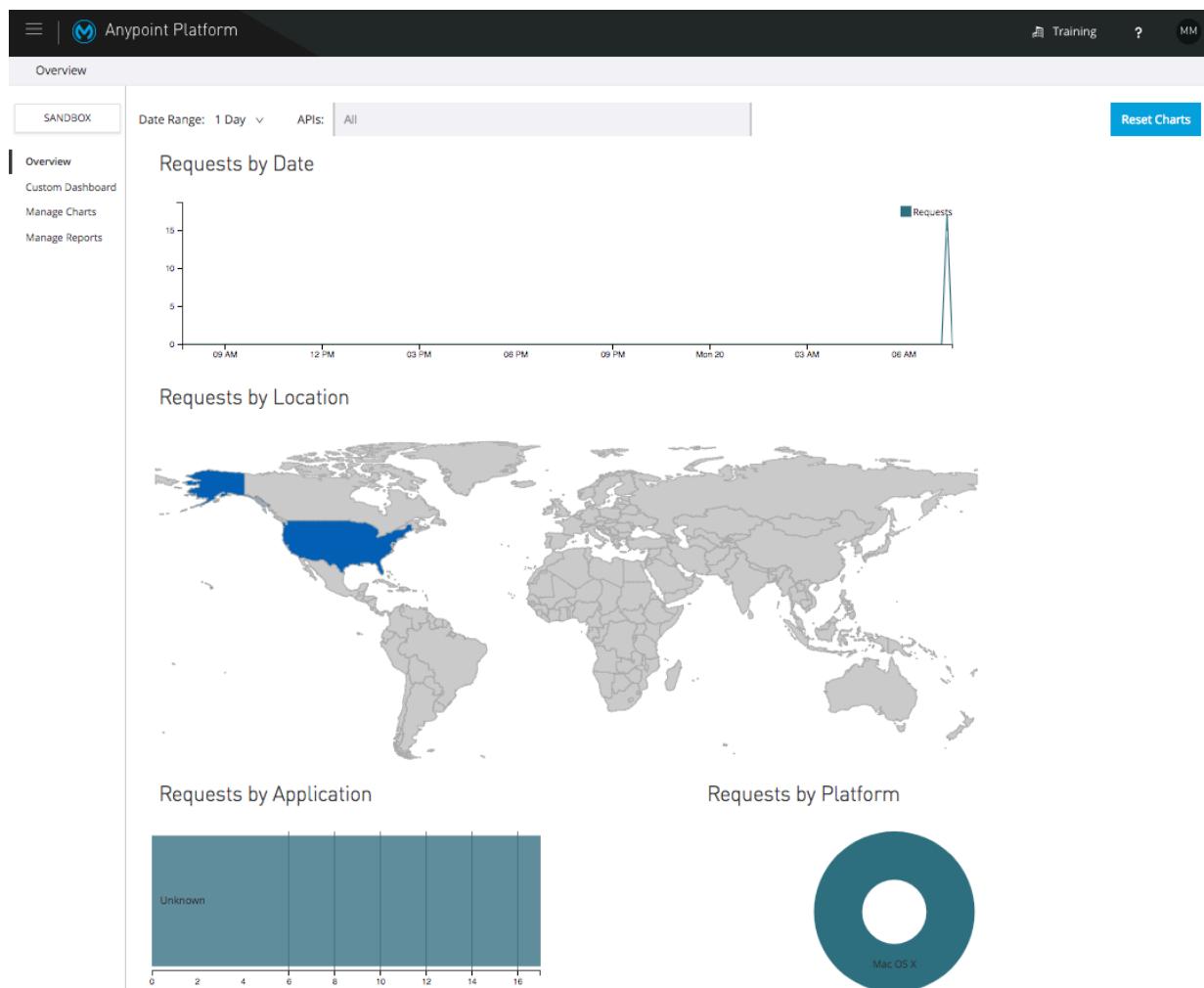
61. Return to the browser tab with API Manager.
62. Refresh the Settings page for your American Flights API.

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



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

65. Review the data in the dashboard.



66. Close the browser tab.

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

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

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

The screenshot shows the API Manager interface with the title "API Administration (Sandbox) American Flights API (v1) - Policies". On the left sidebar, the "Policies" option is selected. In the main content area, there is a table with one row. The row contains a column labeled "Name" with the value "Rate limiting - SLA based", a column labeled "Category" with the value "Quality of service", a column labeled "Fulfils" with the value "SLA Rate Limiting, Client ID required", and a column labeled "Requires" with the value "RAML snippet". Below the table, there is a section titled "Order Method Resource URI" with the values "1 All API Methods All API Resources". At the top right of the main content area, there is a blue button labeled "Apply New Policy" and a link "Edit policy order".

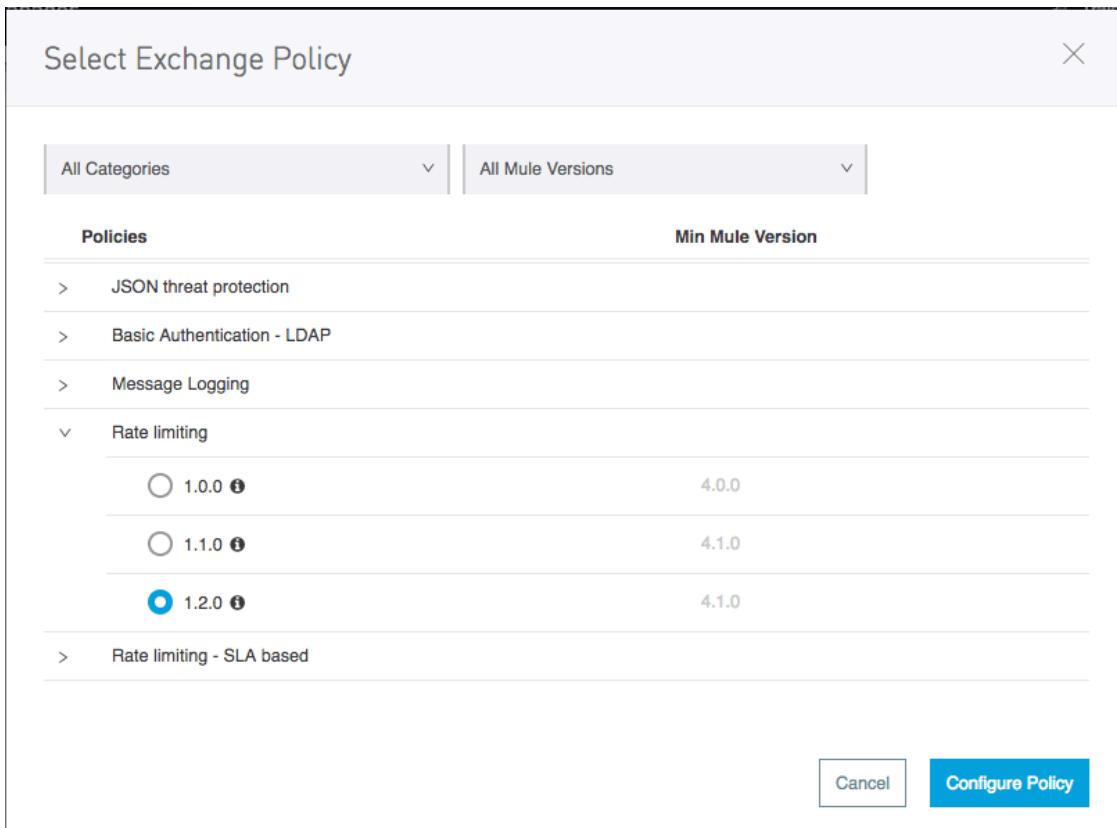
Create a rate limiting policy

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

The screenshot shows the API Manager interface with the title "API Administration (Sandbox) American Flights API (v1) - Policies". On the left sidebar, the "Policies" option is selected. In the main content area, there is a summary for the "American Flights API v1" API. It shows "API Status: Active", "Asset Version: 1.0.1", "Type: RAML/OAS", "Implementation URL: http://training4-american-ws-mule.cloudhub.io/api", and "Consumer endpoint: http://training4-american-api-mule.cloudhub.io/". To the right of the summary, there is a "Actions" dropdown menu with options: "Manage CloudHub Proxy", "View API in Exchange", "View configuration details", and "View Analytics Dashboard". At the bottom of the main content area, there is a blue button labeled "Apply New Policy".

3. Click the Apply New Policy button.

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



5. Click Configure Policy.

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

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

Apply Rate limiting policy

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

Identifier

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

--	--	--	--	--	--

Limits

Pairs of maximum quota allowed and time window.

# of Reqs *	Time Period *	Time Unit *
3	1	Minute

Add Limit

Clusterizable

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

Expose Headers

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

Method & Resource conditions

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

Cancel

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

The screenshot shows the MuleSoft API Manager interface. The top navigation bar includes 'API Manager', 'Training', a question mark icon, and a 'MM' icon. The left sidebar has a 'Sandbox' tab selected, followed by 'API Administration (Sandbox)', 'American Flights API (v1) - Policies', 'Alerts', 'Client Applications', 'Policies' (which is currently selected), 'SLA Tiers', and 'Settings'. The main content area displays the 'American Flights API v1' details, including its status as 'Active', asset version '1.0.1', and type 'RAML/OAS'. It also shows the implementation URL 'http://training4-american-ws-mule.cloudhub.io/api' and consumer endpoint 'http://training4-american-api-mule.cloudhub.io/'. Below this, there is a large blue button labeled 'Apply New Policy'. To the right, there are several actions: 'Manage CloudHub Proxy', 'View API in Exchange', 'View configuration details', and 'View Analytics Dashboard'. At the bottom, there is a table with columns 'Name', 'Category', 'Fulfils', and 'Requires'. One row is visible: 'Rate limiting' (with a help icon) under 'Category', 'Quality of service' under 'Fulfils', and 'Baseline Rate Limiting' under 'Requires'. There is also a 'Edit policy order' button.

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

API Instance ⓘ

ID: 5831632

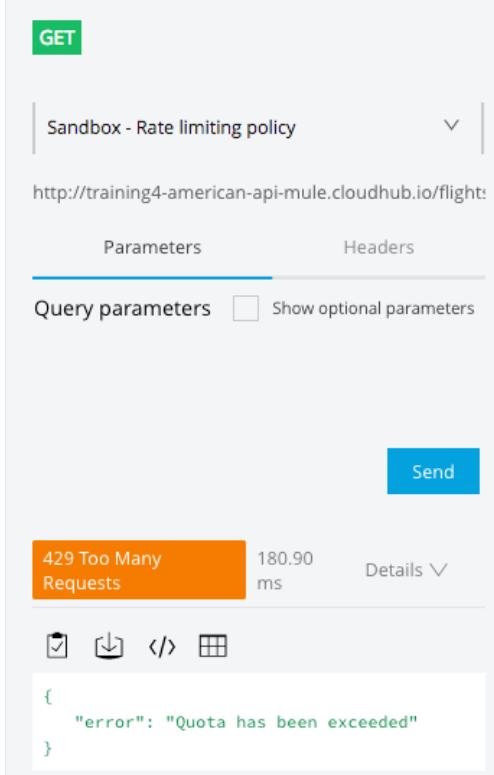
Label: Rate limiting policy 

Test the new rate limiting policy

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

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

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



The screenshot shows the API console interface for a GET request to the 'Sandbox - Rate limiting policy' instance. The URL is `http://training4-american-api-mule.cloudhub.io/flights`. The 'Parameters' tab is selected. A 'Send' button is at the bottom. The response is displayed in a box: '429 Too Many Requests' (highlighted in orange), '180.90 ms', and 'Details'. The details show a JSON error message: `{ "error": "Quota has been exceeded" }`.

Create SLA tiers

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

16. Click the Add SLA tier button.

The screenshot shows the API Manager interface for the American Flights API (v1). On the left, there's a sidebar with options like 'Sandbox', 'API Administration', 'Alerts', 'Client Applications', 'Policies', 'SLA Tiers' (which is selected and highlighted in blue), and 'Settings'. The main content area displays the API details: 'American Flights API v1', 'API Status: Active', 'Asset Version: 1.0.1', 'Type: RAML/OAS', 'Implementation URL: http://training4-american-ws-mule.cloudhub.io/api', and 'Consumer endpoint: http://training4-american-api-mule.cloudhub.io/'. Below these details is a search bar and an 'Actions' dropdown menu with options like 'Manage CloudHub Proxy', 'View API in Exchange', 'View configuration details', and 'View Analytics Dashboard'. At the bottom, it says 'There are no SLA tiers for this API version.' A prominent blue button labeled 'Add SLA tier' is located in the center of the main content area.

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

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

The 'Add SLA tier' dialog box is shown. It has fields for 'Name' (set to 'Free'), 'Description' (empty), 'Approval' (set to 'Automatic'), and a 'Limits' section. In the 'Limits' section, the '# of Reqs' is 1, 'Time Period' is 1, and 'Time Unit' is Minute. There are buttons for 'Cancel' and 'Add' at the bottom.

18. Click the Add button.

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

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

The screenshot shows a table titled 'SLA tiers' with columns: Name, Limits, Applications, Status, and Approval. There are two rows: 'Free' (Limits 1, Applications 0, Active, Auto) and 'Silver' (Limits 1, Applications 0, Active, Manual). Each row has 'Edit' and 'Delete' buttons.

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

Change the policy to rate limiting – SLA based

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

21. Expand the Rate limiting policy.

22. Click the Actions button and select Remove.

The screenshot shows the 'Rate limiting' policy details. It includes sections for 'Category' (Quality of service), 'Fulfils' (Baseline Rate Limiting), and 'Resource URI' (All API Methods, All API Resources). A 'Actions' dropdown menu is open, showing options: View Detail, Disable, Edit, and Remove.

Name	Category	Fulfils	Requires
Rate limiting ⓘ	Quality of service	Baseline Rate Limiting	

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

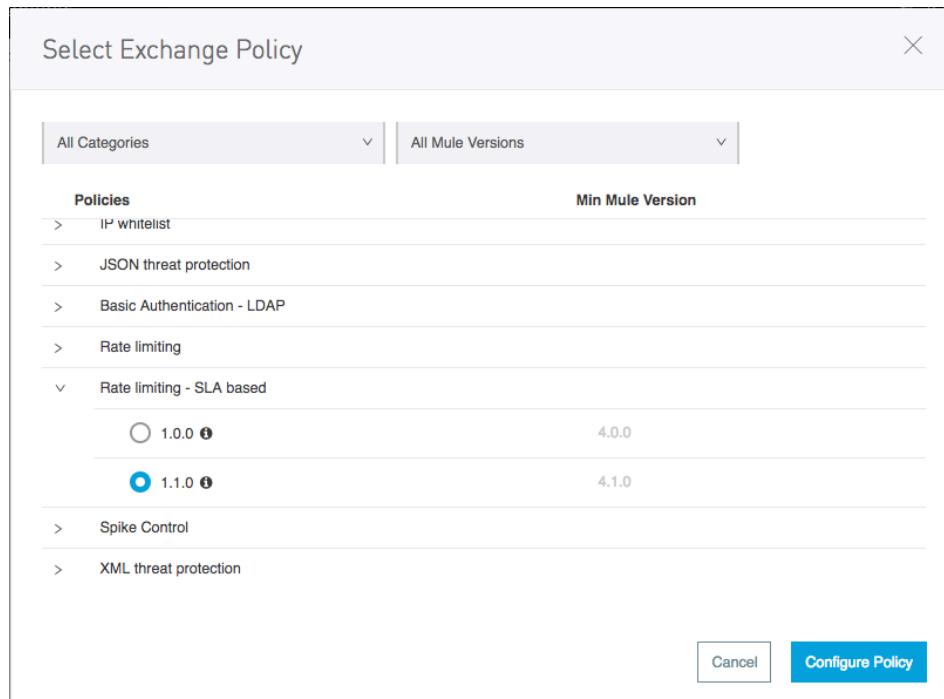
Disable
Edit
Remove

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

24. Click the Apply New Policy button.

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

26. Click Configure Policy.



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

The page header shows 'API Manager' and the current location 'API Administration (Sandbox) / American Flights API (v1) - Policies / Apply Rate limiting - SLA based policy'. A sidebar on the left includes 'Sandbox' and 'Policies' sections.

The main content area is titled 'Apply Rate limiting - SLA based policy' and contains the following configuration fields:

- Client ID Expression ***: Mule Expression to extract Client ID from API requests. Value: `#[attributes.headers['client_id']]`
- Client Secret Expression**: Mule Expression to extract Client Secret from API requests. Value: `#[attributes.headers['client_secret']]`
- Clusterizable**: A checked checkbox with the note: "When using a clustered runtime with this flag enabled, configuration will be shared among all nodes."
- Expose Headers**: A checked checkbox with the note: "Defines if headers should be exposed in the response to the client. These headers are: x-ratelimit-remaining, x-ratelimit-limit and x-ratelimit-reset."
- Method & Resource conditions**:
 - Apply configurations to all API methods & resources
 - Apply configurations to specific methods & resources

At the bottom right are 'Cancel' and 'Apply' buttons.

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

API Instance ⓘ
ID: 5831632
Label: 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.

The screenshot shows the Anypoint Platform interface for testing APIs. A modal window is open for a GET request to the endpoint `http://training4-american-api-mule.cloudhub.io/flight`. The request is set to the 'Sandbox - Rate limiting - SLA based policy'. The 'Parameters' tab is selected, and the 'Send' button is visible. Below the request details, an error response is displayed: a 401 Unauthorized status with a response time of 819.80 ms. The error message is: `{ "error": "Invalid client id or secret" }`.

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

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

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

The screenshot shows the 'Client Applications' section of the API Manager. On the left, a sidebar lists 'Sandbox', 'API Administration', 'Alerts', 'Client Applications' (which is selected), 'Policies', 'SLA Tiers', and 'Settings'. The main content area displays a table with two rows of data. The first row represents a 'Training external app' with a current SLA tier of 'Silver' and a requested tier of 'N/A', both marked as 'Approved'. The second row represents a 'Training internal app' with a current SLA tier of 'Free' and a requested tier of 'N/A', also marked as 'Approved'. Each row includes a 'Revoke' button.

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

Request access to the API as an internal consumer

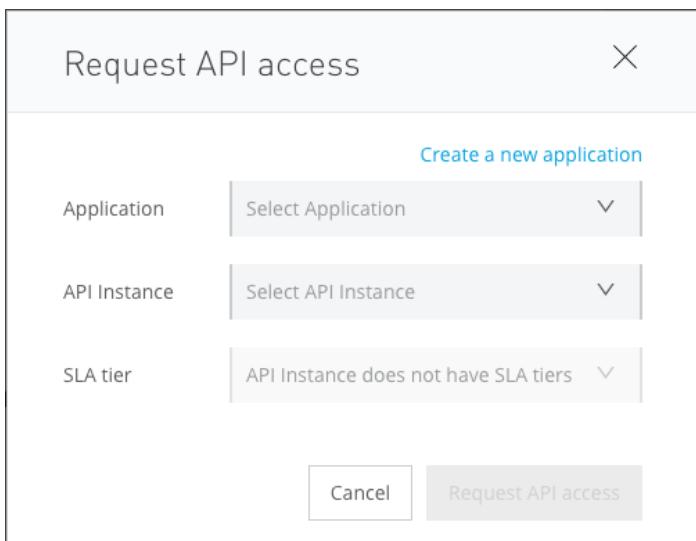
1. Return to the browser tab with Anypoint Exchange.
2. In the left-side navigation, select the name of the API to return to its home page.
3. Click the more options button in the upper-right corner and select Request access.

The screenshot shows the 'American Flights API' page on Anypoint Exchange. The left sidebar has 'Assets list' and 'American Flights API' selected. The main content area features the API's logo, name ('American Flights API'), version ('v1'), a 5-star rating with '(0 reviews)', and 'Rate and review' link. To the right, there are buttons for 'Share', 'Download', 'Edit', and 'Request access'. A 'Overview' tab is selected.

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

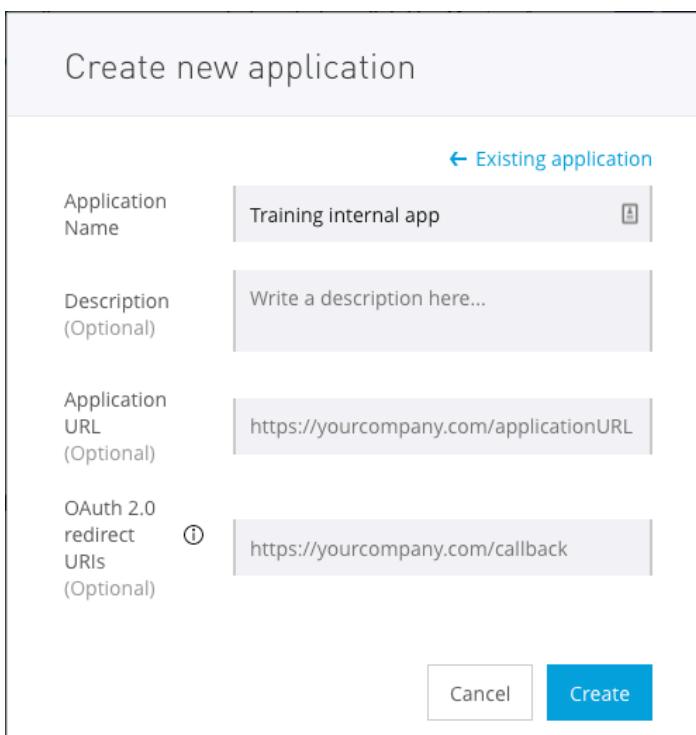
This screenshot shows the same 'American Flights API' page on Anypoint Exchange, but for a user with limited permissions. The 'Edit' button is replaced by a 'Download' and 'Request access' button. The 'Overview' tab is still selected.

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



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

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



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

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

7. Set the SLA tier to Free.

Request API access

Create a new application

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

# of Reqs	Time period	Time Unit
1	1	Minute

[Cancel](#) [Request API access](#)

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

Request API access [X](#)

✓ API access has been successful!

Client ID	e708026bb0cf4c3e8594ca39138b9a00
Client secret	e10A1faF382D47DEA6A90cA8d4FC3891

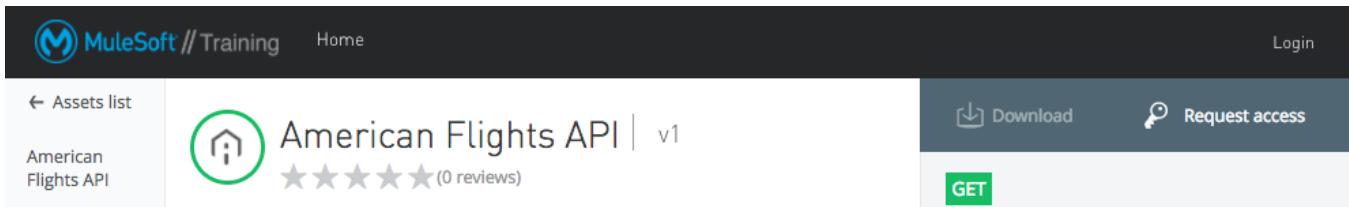
Application details have been opened in a new tab.

[Close](#)

10. Click Close.

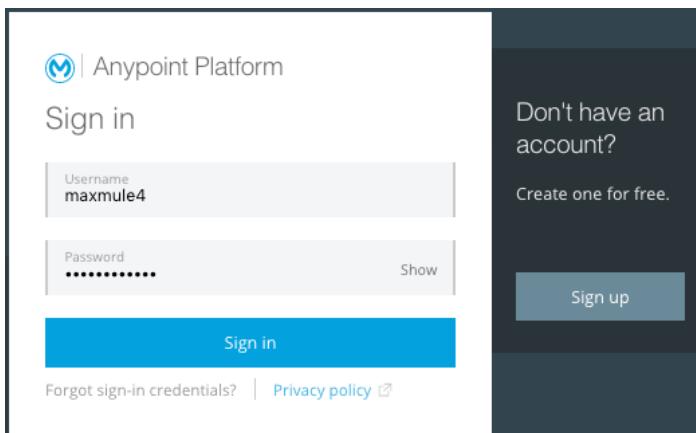
Request access to the API as an external consumer

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.



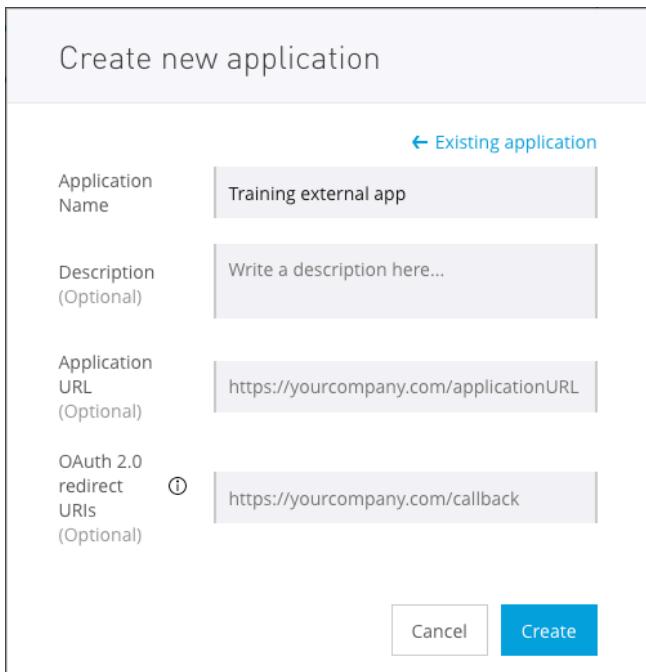
13. Click the Request access button; you should get a page to sign in or create an Anypoint Platform account.
14. Enter your existing credentials and click Sign in.

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.

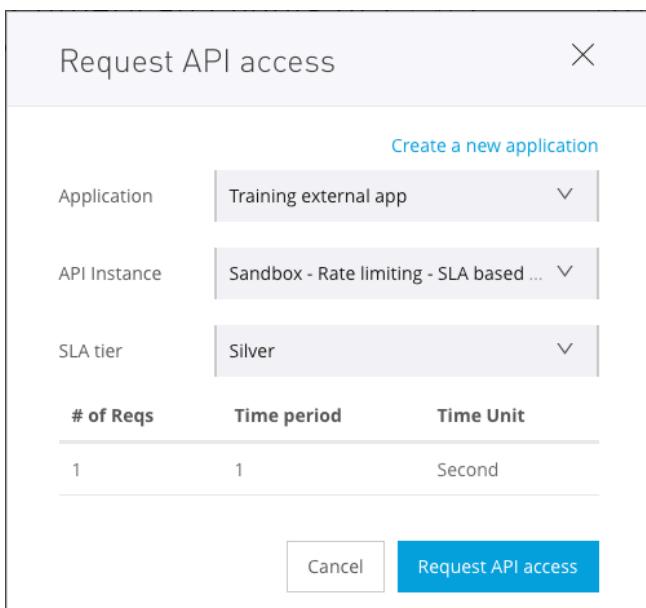


The dialog box has a title 'Create new application' and a back button 'Existing application'. It contains four input fields: 'Application Name' (Training external app), 'Description (Optional)' (Write a description here...), 'Application URL (Optional)' (https://yourcompany.com/applicationURL), and 'OAuth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

18. Click Create.

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

20. Set the SLA tier to Silver.



The dialog box has a title 'Request API access' and a close button 'X'. It contains three dropdowns: 'Application' (Training external app), 'API Instance' (Sandbox - Rate limiting - SLA based ...), and 'SLA tier' (Silver). Below these are three input fields: '# of Reqs' (1), 'Time period' (1), and 'Time Unit' (Second). At the bottom are 'Cancel' and 'Request API access' buttons.

21. Click Request API access.

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

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

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

Name	Description
Training internal app	
Training external app	

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

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

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

Grant an application access

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

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

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

- Sandbox:** A button labeled "SANDBOX".
- Search Bar:** A search bar with placeholder text "Search" and a clear button "X".
- Pagination:** "1 - 2 of 2" with previous and next buttons.
- Table Headers:** Application, Current SLA tier, Requested SLA tier, Status.
- Table Data:**
 - Row 1: Application: Training external app, Current SLA tier: N/A, Requested SLA tier: Silver, Status: Pending. Buttons: Approve, Reject, Delete.
 - Row 2: Application: Training internal app, Current SLA tier: Free, Requested SLA tier: N/A, Status: Approved. Button: Revoke.
- Left Sidebar:** Includes links for API Administration, Alerts, Client Applications (which is selected), Policies, SLA Tiers, and Settings.

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.

Application	Current SLA tier	Requested SLA tier	Status	
Training external app	Silver	N/A	Approved	Revoke
Owners	Max Mule		Submitted	2 minutes ago
Client ID	7623dbcc2e1949d7a861160fe4a3a1e6		Approved	a few seconds ago
URL	None		Rejected	-
Redirect URIs	None		Revoked	-
Training internal app	Free	N/A	Approved	Revoke

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

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

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

GET

Sandbox - Rate limiting - SLA based policy

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

Parameters Headers

</>

Header name	Value
client_id	7623dbcc2e1949d7

+ Add custom header

Send

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

38. Copy the value of the client_secret.

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

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

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

Parameters Headers

</>

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

+ Add custom header

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

200 OK 1119.00 ms Details

⏪ </> ⌂

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

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

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

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

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

The screenshot shows a RAML 1.0 specification for the American Flights API. On the left, the RAML code includes a trait named 'client-id-required' which defines 'client_id' and 'client_secret' headers. On the right, a modal dialog for a GET request to 'Mocking Service' shows the 'Headers' tab selected. It lists the 'client_id*' and 'client_secret*' headers with checkboxes next to them. A 'Send' button is at the bottom right of the dialog.

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

types:
  AmericanFlight: !include https://mocksvc-proxy.anypoint.mulesoft.com/exc

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

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

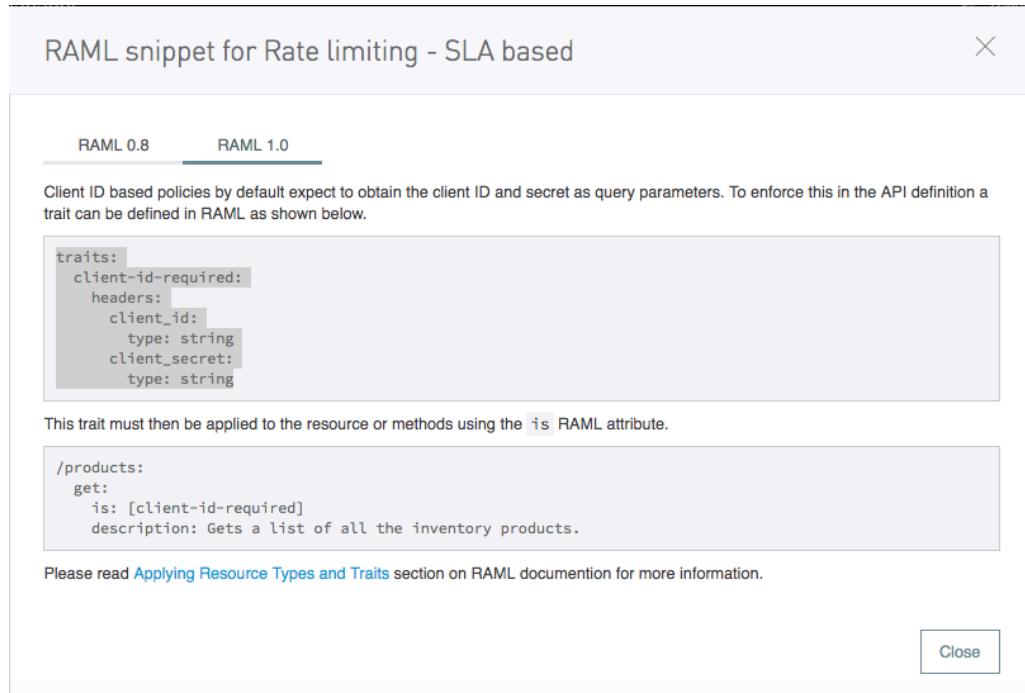
Copy the traits required to add authentication to the API specification

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

The screenshot shows the API Manager Policies page for the American Flights API (v1). The left sidebar has 'Policies' selected. The main area displays a table of policies. One policy, 'Rate limiting - SLA based', is highlighted. To its right, there are buttons for 'Apply New Policy', 'Edit policy order', and a 'RAML snippet' link. The top right of the screen shows 'Training' and 'MM' icons.

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

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



- Click Close.

Add authentication headers to the API specification

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

```

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

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

10. In the shelf, select is.

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

Types and Traits Docs

is	description
type	displayName

11. Add empty array brackets.

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

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

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

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

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

Test the API in the API console in Design Center

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

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

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

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

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

The screenshot shows a 400 Bad Request response. At the top, it says "400 Bad Request" and "499.00 ms". Below this is a large error icon featuring a sad face with crossed-out eyes. To the right of the icon, the text reads: "The requested URL can't be reached. The service might be temporarily down or it may have moved permanently to a new web address. Resource is unavailable". At the bottom, there is a JSON error message: {"error": "headers: client_id: required"}. Below the JSON message are several small icons: a square with a circle, a downward arrow, a left-right arrow, and a three-line menu.

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

The screenshot shows the MuleSoft Anypoint Platform Mocking service interface. At the top, it says "Mocking service: —✓". Below that is a "Request URL" field containing "https://mocksvc.mulesoft.com/mocks". Under the "Parameters" tab, there are two fields: "client_id*" with value "432" and "client_secret*" with value "765". A "Send" button is located below these fields. At the bottom, the response is shown as a green box indicating "200 OK" and "506.59 ms". Below the status, there are icons for copy, download, and refresh. The response body is a JSON array: [Array[2], -0: { "ID": 1, "code": "ER38sd", "price": 400 }].

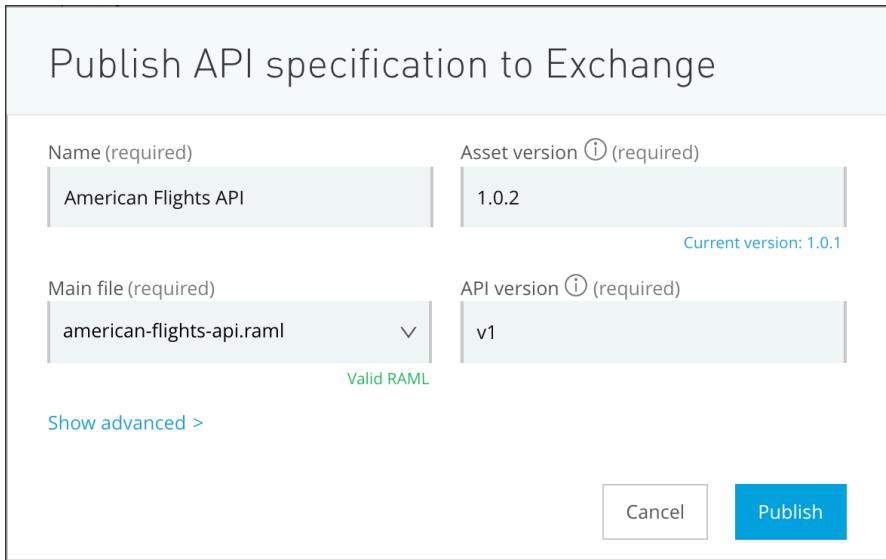
Publish the new version of the API to Exchange

19. Turn off the mocking service.

The screenshot shows the MuleSoft Anypoint Platform Mocking service interface. At the top, it says "Mocking service: X —". Below that is a "Request URL" field. The "Mocking service" status has changed from green with a checkmark to red with an 'X'.

20. Click the Publish to Exchange button.

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



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

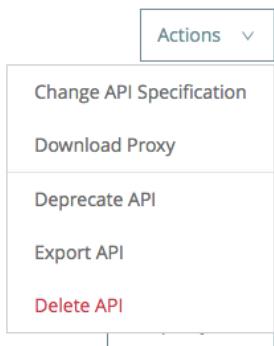
Update the managed API instance to use the new version of the API specification

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

American Flights API v1

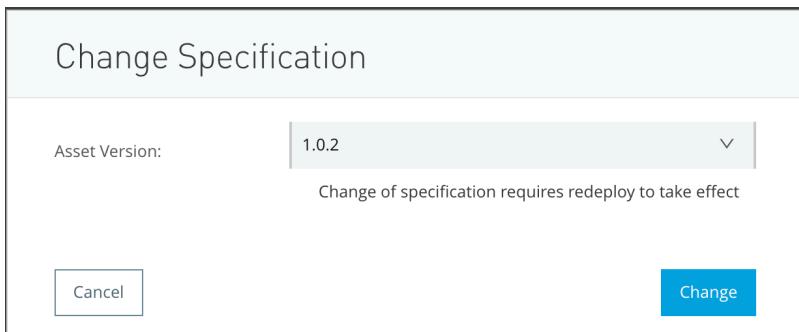
API Status: ● Active Asset Version: 1.0.1 Type: RAML/OAS
Implementation URL: <http://training4-american-ws-mule.cloudhub.io/api>
Consumer endpoint: <http://training4-american-api-mule.cloudhub.io/>

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



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

27. Click Change.



Redeploy a new proxy

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

The screenshot shows the "Deployment Configuration" section of the interface. It includes a dropdown menu for "Runtime version" set to "4.x.x", a text input for "Proxy application name" containing "training4-american-api-mule", and a text input for "Domain" containing ".cloudhub.io". At the bottom is a prominent blue "Redeploy" button.

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

The screenshot shows the "Runtime Manager" interface with the "Logs" tab selected. On the left, there's a sidebar with "Sandbox", "Applications", "Dashboard", "Insight", and "Logs" (which is selected). The main area shows the logs for the application "training4-american-api-mule". The logs display deployment and startup messages. To the right, there's a "Deployments" panel showing deployment logs for specific times like 17:28 and 14:12.

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

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

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

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

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

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

GET

Mocking Service

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

Parameters Headers

</>

client_id*

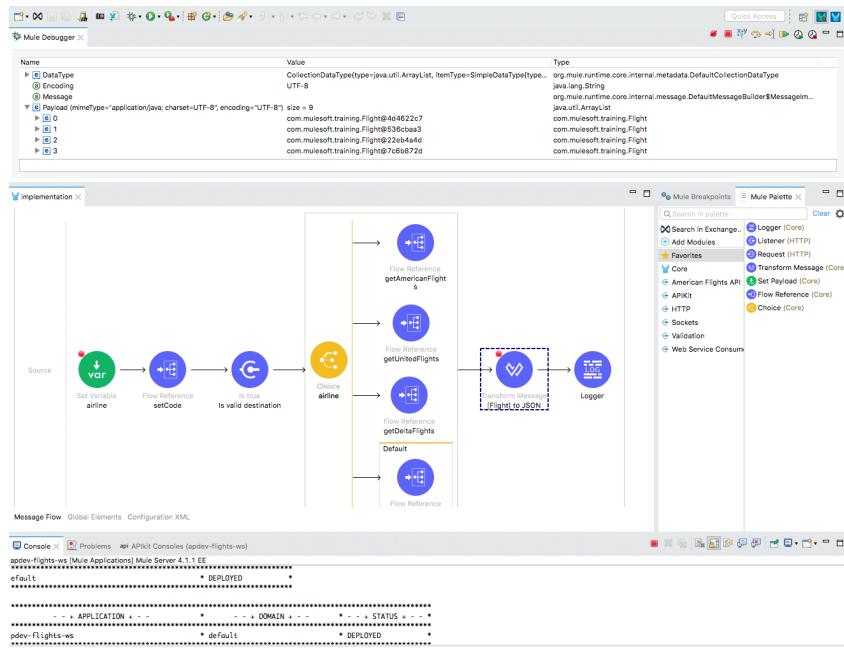
client_secret*

+ Add custom header

Send

39. Close all Anypoint Platform browser windows and tabs.

PART 2: Building Applications with Anypoint Studio

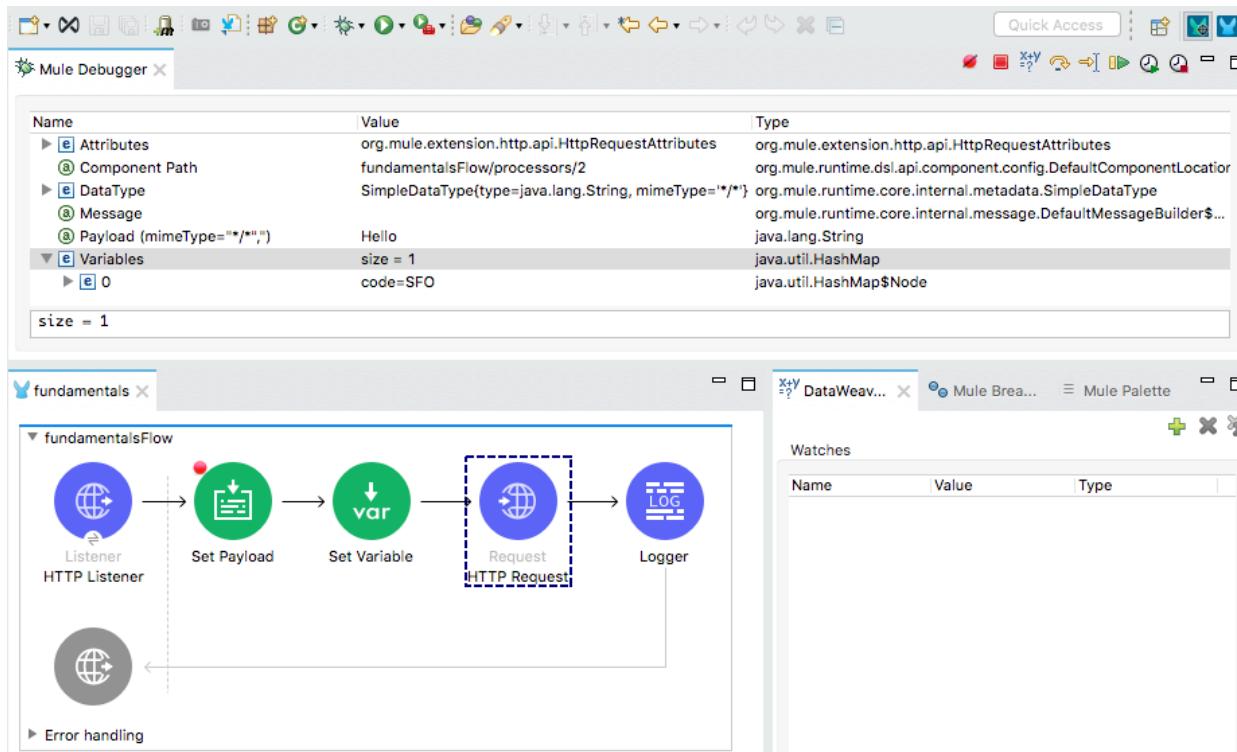


At the end of this part, you should be able to:

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

- Write DataWeave scripts for transformations.

Module 6: Accessing and Modifying Mule Events



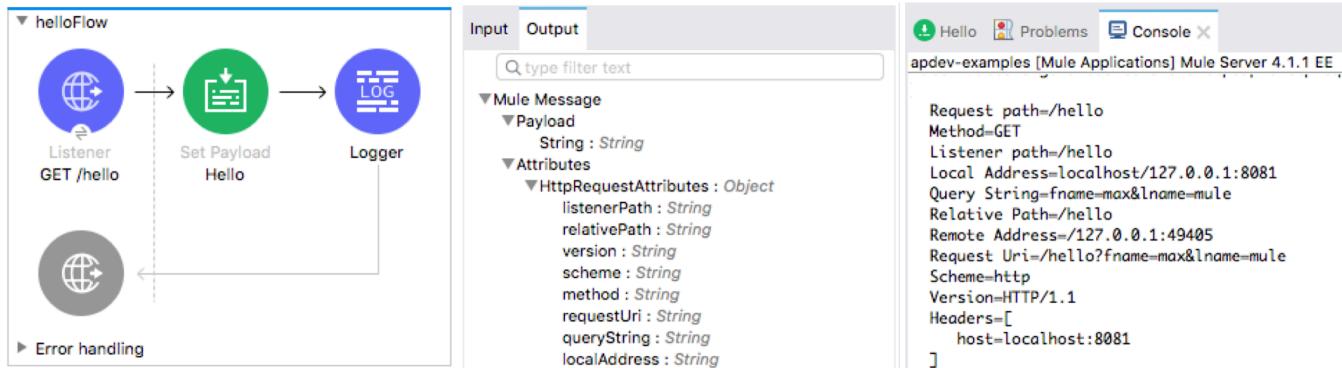
At the end this module, you should be able to:

- 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

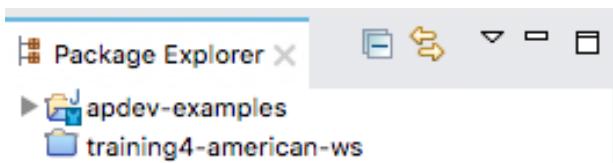
In this walkthrough, you create a new project to use in the next two modules to learn about Mule events and Mule applications. You will:

- 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

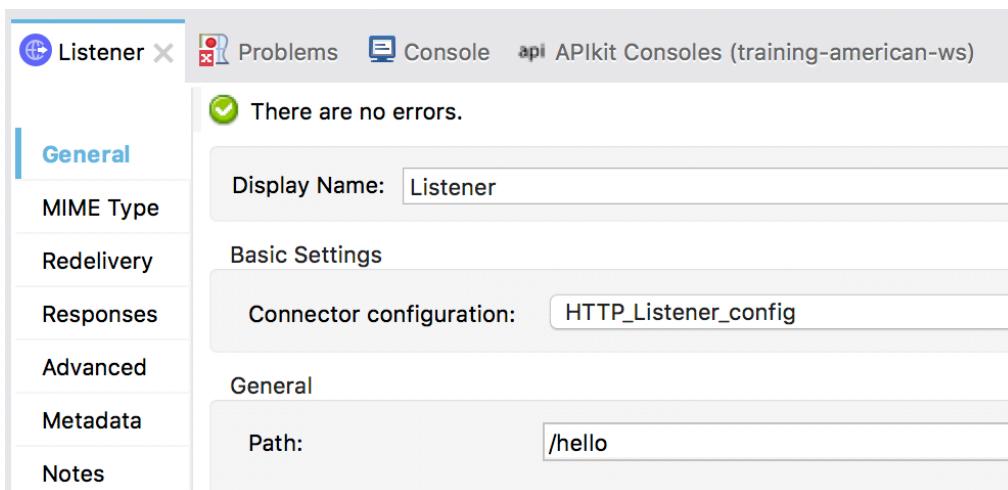
1. Return to Anypoint Studio.
2. Right-click training4-american-ws and select Close Project.
3. Select File > New > Mule Project.
4. Set the project name to apdev-examples and click Finish.



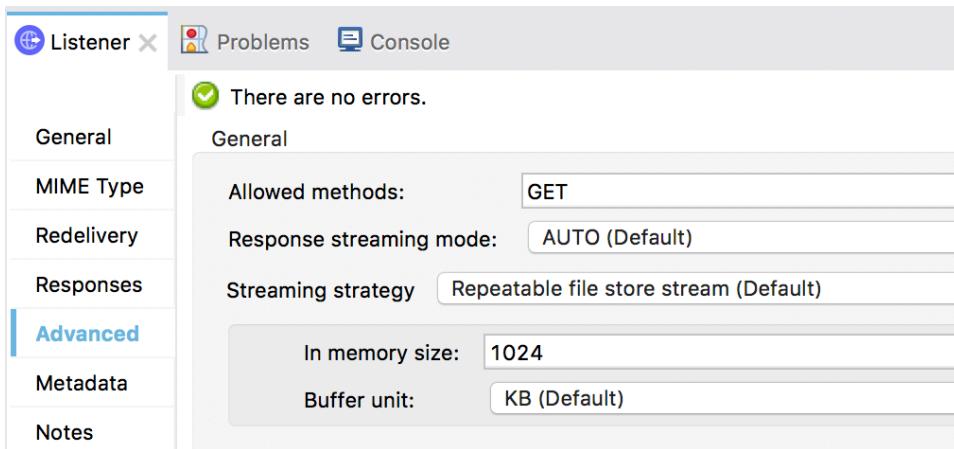
Create an HTTP Listener to receive requests

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, verify the host value is set to 0.0.0.0 and the port value to 8081.
9. Click OK.

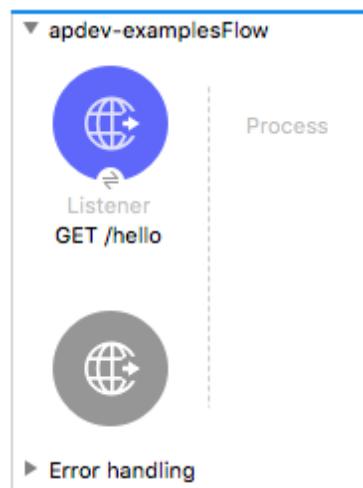
10. In the Listener properties view, set the path to /hello.



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



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

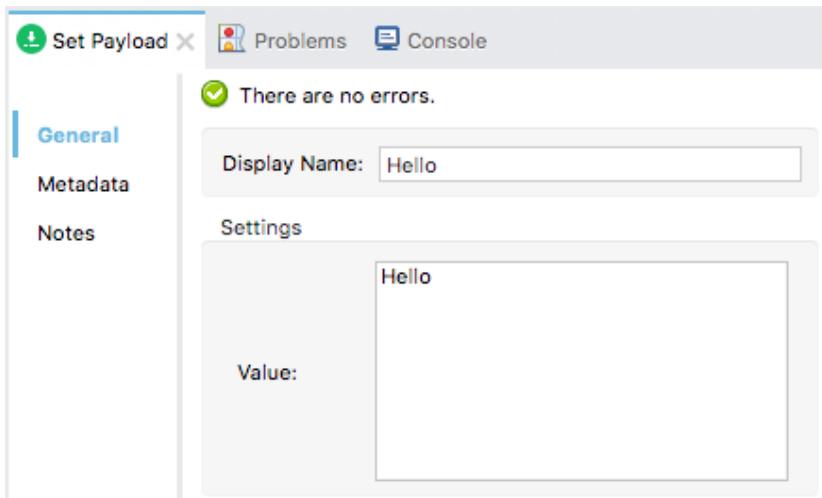


Change the flow name

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

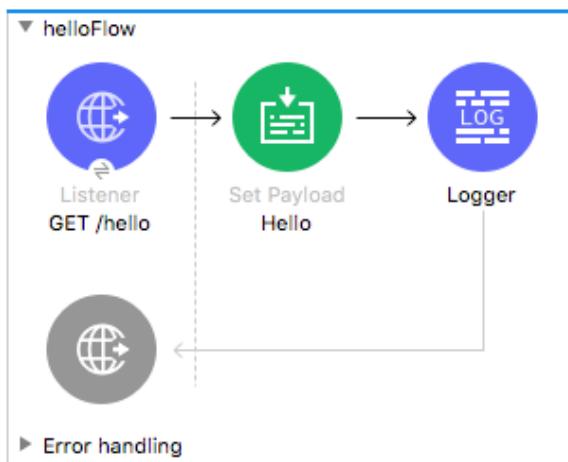
Set the message payload

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



Add a Logger

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



View event structure and metadata in the DataSense Explorer

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

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. At the top, there are tabs for 'Input' and 'Output'. Below them is a search bar labeled 'type filter text'. The main area displays a hierarchical tree structure under the 'Mule Message' node. The 'Payload' node is expanded, showing the value 'Undefined : Unknown'. The 'Attributes' node is also expanded, showing the value 'Undefined : Unknown'. A 'Variables' section is present at the bottom of the tree.

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

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. The tree structure under 'Mule Message' shows the 'Payload' node expanded, indicating a 'Binary : Binary' type. The 'Attributes' node is expanded, revealing a complex object named 'HttpRequestAttributes : Object' which contains numerous properties such as 'listenerPath', 'relativePath', 'version', 'scheme', 'method', 'requestUri', 'queryString', 'localAddress', 'remoteAddress', 'clientCertificate', 'queryParams', 'uriParams', 'requestPath', and 'headers', each with its respective type (e.g., String, Object). A 'Variables' section is also visible.

22. Select the Set Payload component in helloFlow.

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

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. The tree structure under 'Mule Message' shows the 'Payload' node expanded, indicating an 'Any : Any' type. The 'Attributes' node is expanded, showing a reference to 'HttpRequestAttributes : Object'. A 'Variables' section is present at the bottom.

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

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top says 'type filter text'. Below it is a tree structure under 'Mule Message': 'Payload' (with 'String : String') and 'Attributes' (with 'HttpRequestAttributes : Object'). There is also a 'Variables' section.

25. Select the Logger component in helloFlow.

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

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

Run the application and review response data

28. Save the file and run the project.

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

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

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

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

View event data in the Anypoint Studio console

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

32. Locate the data displayed by using the Logger.

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

34. Review the event attributes.

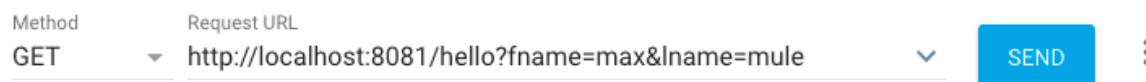


```
*****
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*****
* apdev-examples * default * DEPLOYED *
*****
```

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

Send query parameters with a request

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



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

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

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

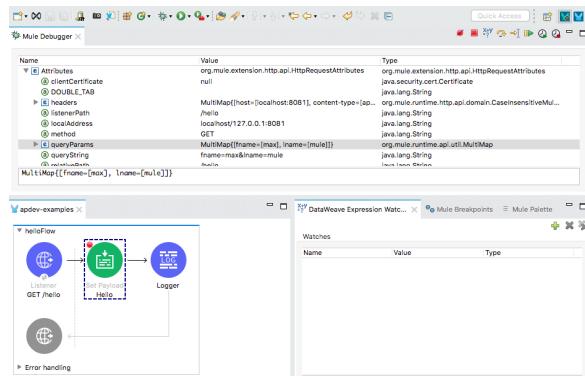
```
{  
    payload=java.lang.String  
    mediaType=/*  
    attributes=org.mule.extension.http.api.Http  
{  
    Request path=/hello  
    Method=GET  
    Listener path=/hello  
    Local Address=localhost/127.0.0.1:8081  
    Query String=fname=max&lname=mule  
    Relative Path=/hello  
    Remote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

40. Stop the project.

Walkthrough 6-2: Debug a Mule application

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

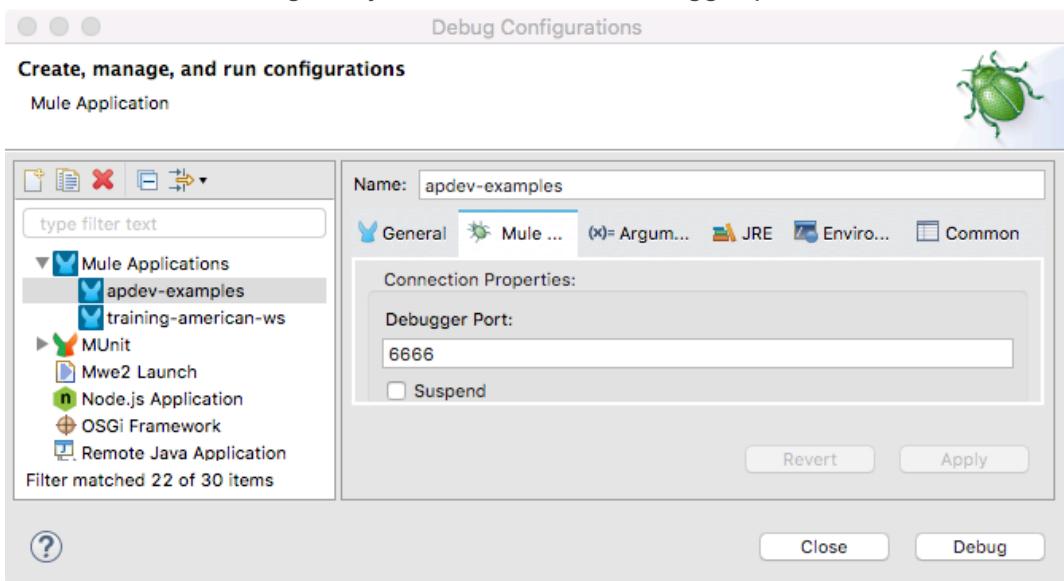
- Locate the port used by the Mule Debugger.
- Add a breakpoint, debug an application, and step through the code.
- Use the Mule Debugger to view event properties.
- Pass query parameters to a request and locate them in the Mule Debugger.
- Increase the request timeout for Advanced REST Client.



Locate the port used by the Mule Debugger

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

4. Select the Mule Debug tab; you should see the debugger port is set to 6666 for the project.

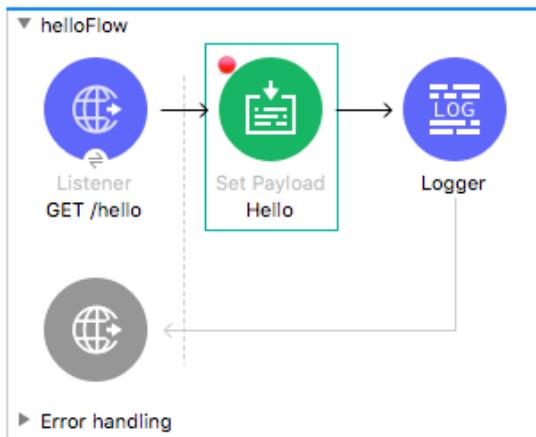


Note: If you know you have another program using port 6666 like McAfee on Windows, change this to a different value. Otherwise, you can test the Debugger first and come back and change the value here later if there is a conflict.

5. Click Close.

Add a breakpoint

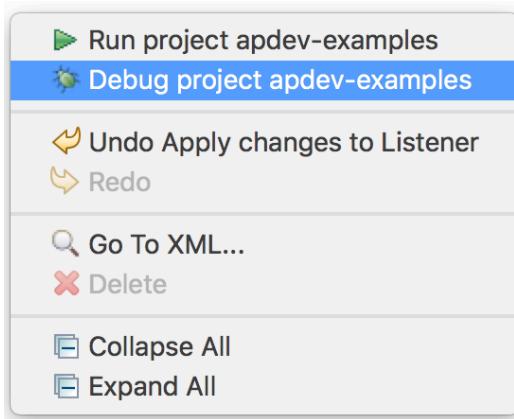
6. Right-click the Set Payload component and select Toggle breakpoint.



Debug the application

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

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



8. If you get a Confirm Perspective Switch dialog box, select Remember my decision and click Yes.
9. In the Debug perspective, close the MUnit view.
10. Look at the console and wait until the application starts.
11. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.

View event data in the Mule Debugger

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

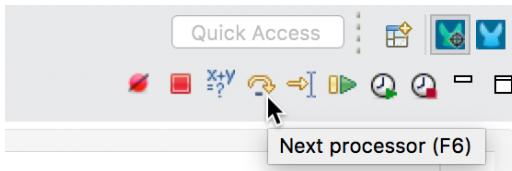
15. Locate the queryParams object.

The screenshot shows the Mule Studio interface with several open windows:

- Mule Debugger**: Shows a tree view of the current request attributes. One node, "queryParams", is expanded, revealing its value: "MultiMap{{fname=[max], lname=[mule]}}".
- apdev-examples**: Shows the "helloFlow" message flow. It consists of three components: Listener, Set Payload (with payload "Hello"), and Logger.
- Console**: Displays deployment logs for Mule Server 4.1.3 EE. Key entries include:
 - INFO 2018-09-20 15:01:36,909 [WrapperListener_start_runner] com.mulesoft.agent.configuration.postconfigure.DefaultPostConfigureRunner: Initializing the scheduling.notification.internal.message.handler ...
 - INFO 2018-09-20 15:01:36,909 [WrapperListener_start_runner] com.mulesoft.agent.configuration.postconfigure.DefaultPostConfigureRunner: scheduling.notification.internal.message.handler initialized successfully.
 - INFO 2018-09-20 15:01:36,912 [WrapperListener_start_runner] org.mule.runtime.module.deployment.internal.DeploymentDirectoryWatcher: + Mule is up and kicking (every 500ms) +
 - INFO 2018-09-20 15:01:36,921 [WrapperListener_start_runner] org.eclipse.jetty.server.AbstractConnector: Started ServerConnector@5f3f80df{HTTP/1.1,[http/1.1]}{0.0.0.0:59123}
 - INFO 2018-09-20 15:01:36,923 [WrapperListener_start_runner] org.mule.runtime.internal.StartupSummaryDeploymentListener: + - + DOMAIN + - - STATUS + - - *
- DataWeave Expression Watches**: An empty window for watching DataWeave expressions.

Step through the application

16. Click the Next processor button.



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

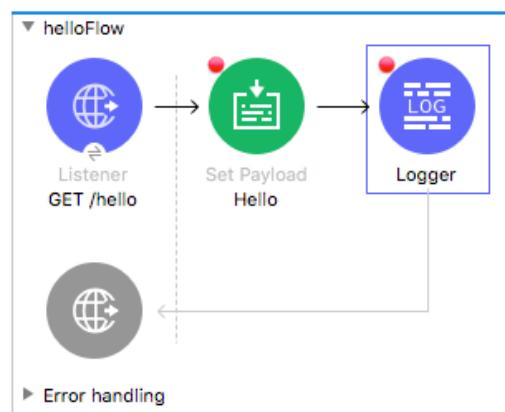
The screenshot shows the Mule Debugger interface. In the top panel, under the 'attributes' section, the 'payload' attribute is expanded to show its value as "Hello". Below this, other attributes like 'correlationId' and 'vars' are listed. In the bottom panel, the 'apdev-examples' application is shown with a flow named 'helloFlow'. The flow consists of three components: a 'Listener' (represented by a globe icon), a 'Set Payload' processor (represented by a clipboard icon), and a 'Logger' (represented by a log icon). The 'Set Payload' processor has a configuration step where it is set to 'Hello'. The flow is currently at the 'Set Payload' step, indicated by a red dot on the wire.

18. Expand Attributes and review the values.

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

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

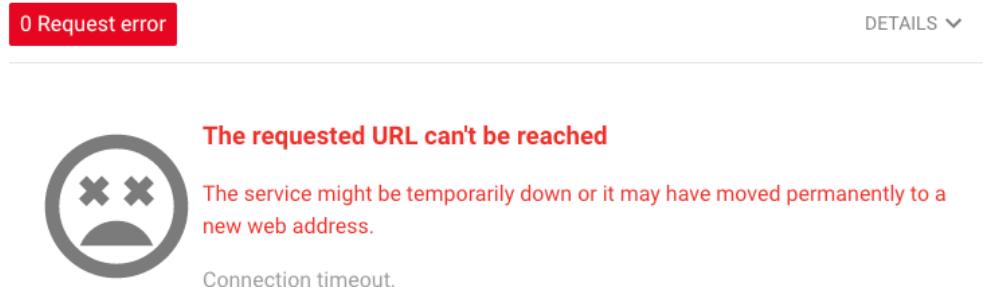
20. Add a breakpoint to the Logger.



21. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.
22. In the Mule Debugger, click the Resume button; you should step to the Logger.
23. Click the Resume button again; you should step through the rest of the application.

Cause the HTTP request to timeout

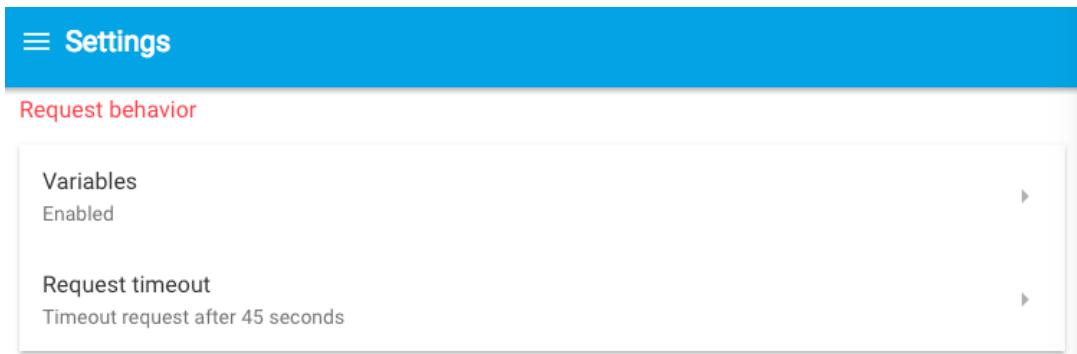
24. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.
25. Do not step through the application and wait (45 seconds) for the request to timeout.



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

Increase the request timeout for Advanced REST Client

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



The screenshot shows the "Settings" menu of the Advanced REST Client. The title bar is blue with the text "≡ Settings". Below it, under "Request behavior", there are two items: "Variables" (with "Enabled" checked) and "Request timeout" (set to "Timeout request after 45 seconds"). Each item has a small arrow icon to its right.

28. Change the request timeout to 300 seconds.

The screenshot shows the 'Settings' screen with the 'Timeout settings' section selected. At the top, there is a blue header bar with the title 'Settings'. Below it, a red arrow points left to the text 'Timeout settings'. The main content area has a light gray background. It contains a table with two columns: 'Request timeout' and 'seconds'. The 'Request timeout' column has a value of '300'. A note below the table states: 'When set to "0" (zero) then the request will never timeout. If the server does not close the connection and sends no response then the request will never end.' Another note at the bottom says: 'Set the value to positive number to set the time (in seconds) after which the request will timeout.'

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

The screenshot shows the 'Settings' screen with the 'HTTP request' drawer open. The drawer has a light gray background. It contains tabs for 'History' (which is selected and highlighted in green), 'Saved', and 'APIs'. Below the tabs, there is a note: 'When set to "0" (zero) then the request will never timeout. If the server does not close the connection and sends no response then the request will never end.' At the bottom of the drawer, there is a 'GET' button followed by a URL: 'http://localhost:8081/hello?fname=max&lname=mule'.

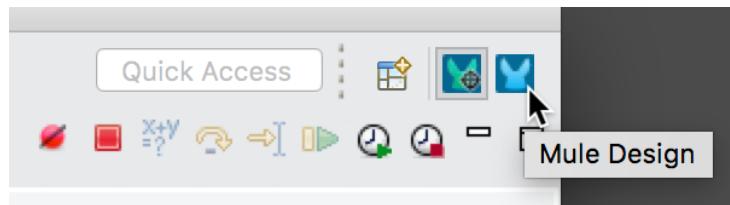
30. Click the Toggle Drawer button again to hide the drawer; you should now see the Request screen again.

The screenshot shows the 'Request' screen. At the top, there is a blue header bar with the title 'Request'. Below it, there is a toolbar with three tabs: 'http://training4-american...', 'http://localhost:8081/hell...', and a '+' icon. The second tab is currently active. The main content area has a light gray background. It contains fields for 'Method' (set to 'GET') and 'Request URL' (set to 'http://localhost:8081/hello?fname=max&lname=mule'). To the right of these fields are a 'SEND' button and a vertical ellipsis '...'. Above the content area, there is a note: 'When set to "0" (zero) then the request will never timeout. If the server does not close the connection and sends no response then the request will never end.'

Switch perspectives

31. Return to Anypoint Studio.
32. Click the Resume button.

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



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

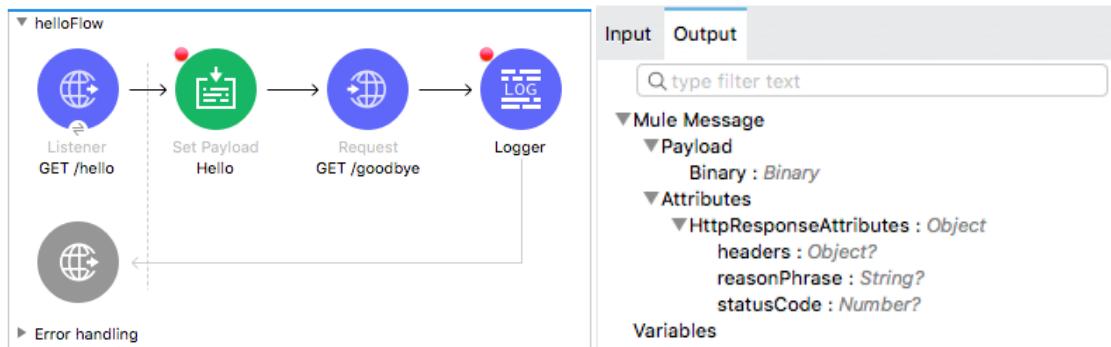
34. Leave the project running.

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

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

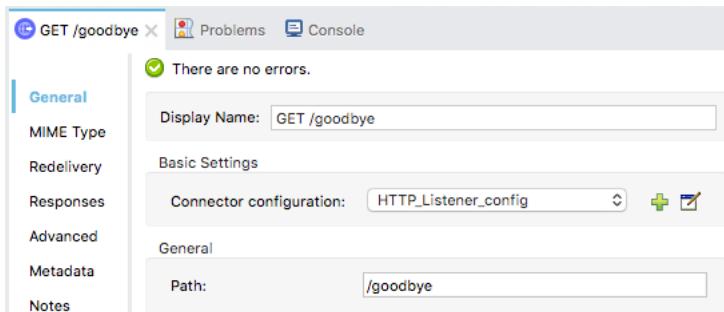
- Create a second flow with an HTTP Listener.
- Make an HTTP request from the first flow to the new HTTP Listener.
- View the event data as it moves through both flows.

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

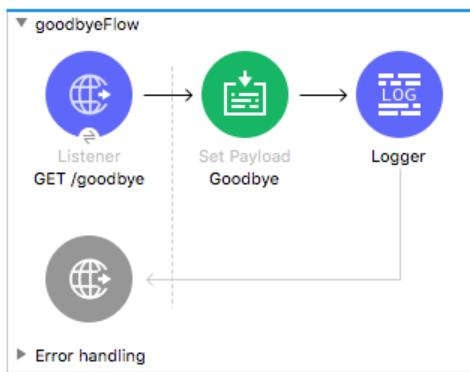


Create a second flow with an HTTP Listener

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

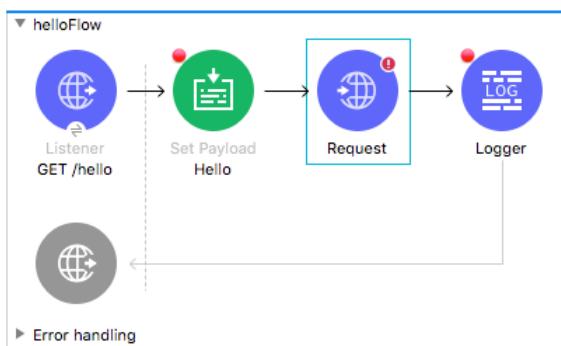


7. Add a Set Payload transformer and a Logger to the flow.
8. In the Set Payload properties view, set the display name and value to Goodbye.

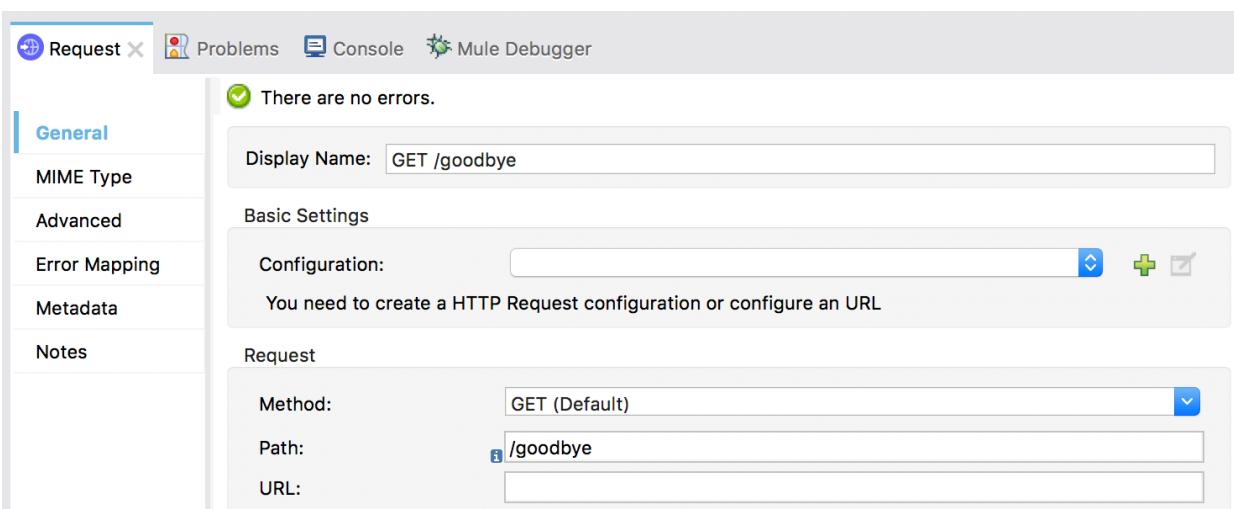


Make an HTTP request

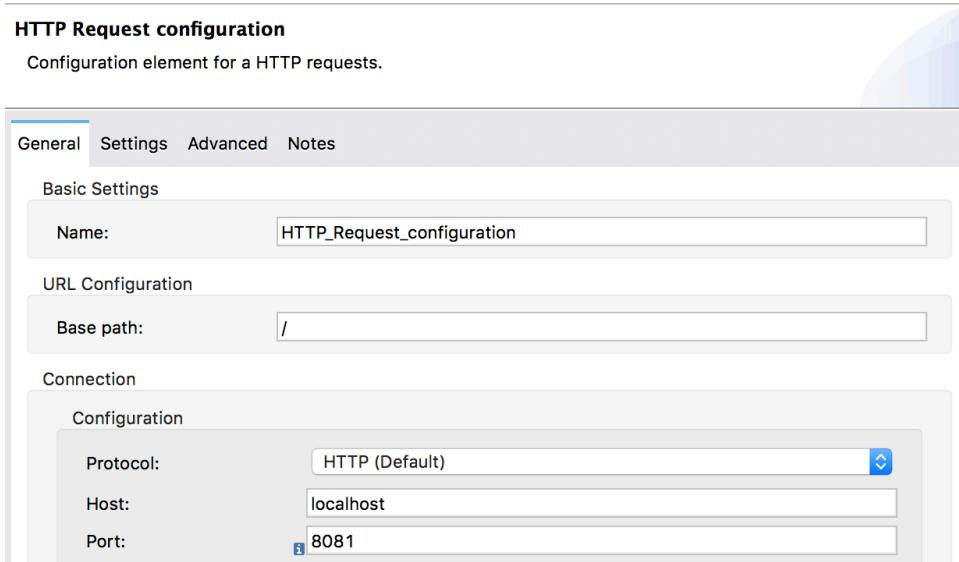
9. From the Mule Palette, drag an HTTP Request to the canvas and drop it before the Logger in helloFlow.



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



12. Click the Add button next to configuration.
13. In the dialog box, set the host to localhost and the port to 8081 and click OK.



View event structure and metadata in the DataSense Explorer

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

Input Output

Q type filter text

▼ Mule Message

 ▼ Payload

 (**Actual**) String : String
 (**Expected**) Object : Object

 ▼ Attributes

 ▼ HttpRequestAttributes : Object

- listenerPath : String
- relativePath : String
- version : String
- scheme : String
- method : String
- requestUri : String
- queryString : String
- localAddress : String
- remoteAddress : String

 ► clientCertificate : Object?
 queryParams : Object
 uriParams : Object
 requestPath : String
 headers : Object

Variables

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

```
Input Output
Search: type filter text
▼ Mule Message
  ▼ Payload
    Any : Any
  ▼ Attributes
    ▼ HttpServletResponseAttributes : Object
      statusCode : Number
      reasonPhrase : String
      headers : Object
Variables
```

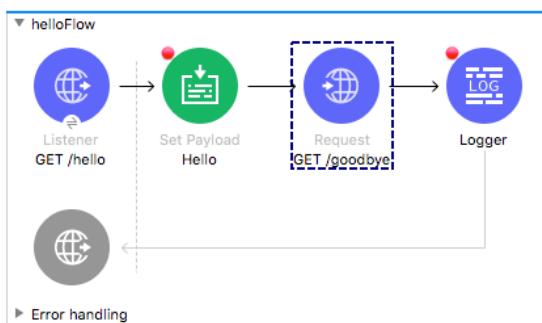
Change timeout for HTTP Request response

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

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

Debug the application

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

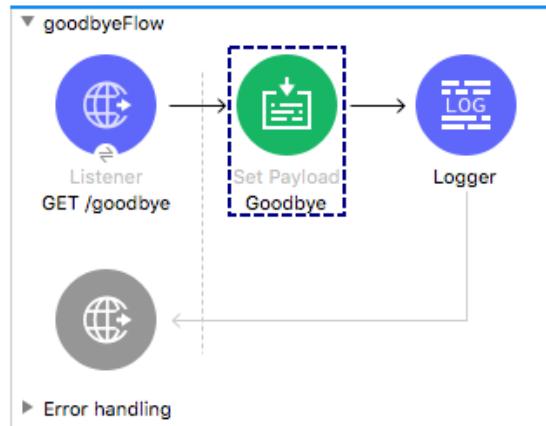


21. Look at the values of the payload and the attributes, including the queryParams.

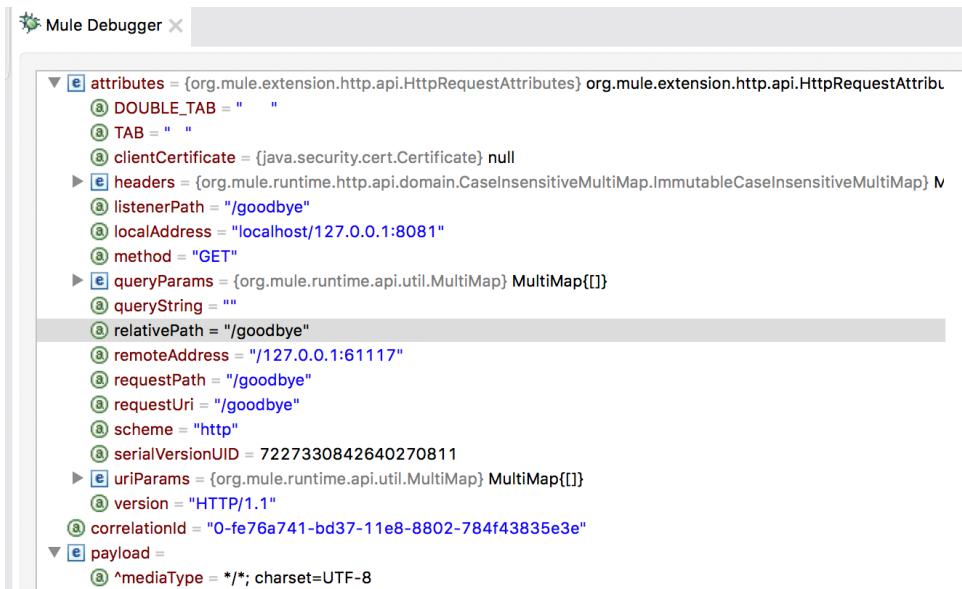


```
attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttribu
  @⑧ DOUBLE_TAB = "    "
  @⑧ TAB = "  "
  @⑧ clientCertificate = {java.security.cert.Certificate} null
  > @⑧ headers = {org.mule.runtime.http.api.domain.CaseInsensitiveMultiMap.ImmutableCaseInsensitiveMultiMap} M
    @⑧ listenerPath = "/hello"
    @⑧ localAddress = "localhost/127.0.0.1:8081"
    @⑧ method = "GET"
  > @⑧ queryParams = {org.mule.runtime.api.util.MultiMap} MultiMap[[fname=[max], lname=[mule]]]
    @⑧ queryString = "fname=max&lname=mule"
    @⑧ relativePath = "/hello"
    @⑧ remoteAddress = "/127.0.0.1:61113"
    @⑧ requestPath = "/hello"
    @⑧ requestUri = "/hello?fname=max&lname=mule"
```

22. Step into goodbyeFlow.

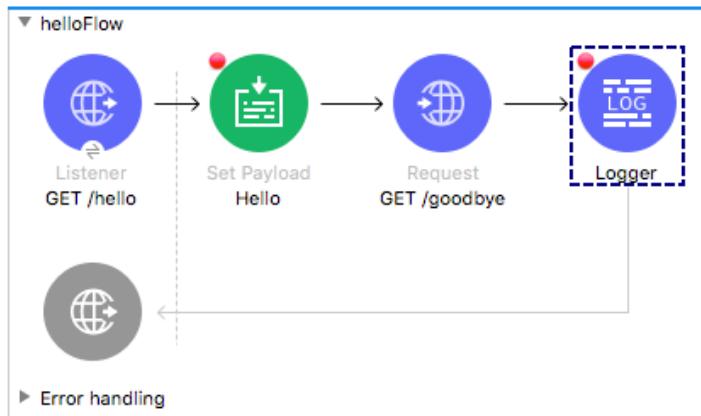


23. Look at the values of the payload and the attributes.



```
attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttribu
  @⑧ DOUBLE_TAB = "    "
  @⑧ TAB = "  "
  @⑧ clientCertificate = {java.security.cert.Certificate} null
  > @⑧ headers = {org.mule.runtime.http.api.domain.CaseInsensitiveMultiMap.ImmutableCaseInsensitiveMultiMap} M
    @⑧ listenerPath = "/goodbye"
    @⑧ localAddress = "localhost/127.0.0.1:8081"
    @⑧ method = "GET"
  > @⑧ queryParams = {org.mule.runtime.api.util.MultiMap} MultiMap[{}]
    @⑧ queryString = ""
    @⑧ relativePath = "/goodbye"
    @⑧ remoteAddress = "/127.0.0.1:61117"
    @⑧ requestPath = "/goodbye"
    @⑧ requestUri = "/goodbye"
    @⑧ scheme = "http"
    @⑧ serialVersionUID = 7227330842640270811
  > @⑧ uriParams = {org.mule.runtime.api.util.MultiMap} MultiMap[{}]
    @⑧ version = "HTTP/1.1"
  @⑧ correlationId = "0-fe76a741-bd37-11e8-8802-784f43835e3e"
  > @⑧ payload =
    @⑧ ^mediaType = */*; charset=UTF-8
```

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



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

```
attributes = {org.mule.extension.http.api.HttpResponseAttributes} org.mule.extension.http.api.HttpResponseAttributes
  @ DOUBLE_TAB = "
  @ TAB = "
headers = {org.mule.runtime.http.api.domain.CaseInsensitiveMultiMap.ImmutableCaseInsensitiveMultiMap} M
  ▶ @ 0 = {java.util.AbstractMap$SimpleEntry} date=Fri, 21 Sep 2018 00:50:34 GMT
  ▶ @ 1 = {java.util.AbstractMap$SimpleEntry} content-length=7
  @ reasonPhrase =
  @ serialVersionUID = -3131769059554988414
  @ statusCode = 200
  @ correlationId = "0-fe76a741-bd37-11e8-8802-784f43835e3e"
  ▶ @ payload = Goodbye
  @ vars = {java.util.Map} size = 0
```

A screenshot of the Mule Debugger tool. The title bar says "Mule Debugger". The main pane shows a tree view of variable "e". Under "attributes", it shows "DOUBLE_TAB" and "TAB" with their respective values. Under "headers", it shows a map with entries for "date", "content-length", "reasonPhrase", "serialVersionUID", "statusCode", and "correlationId". Under "payload", it shows the value "Goodbye". Under "vars", it shows a map with size 0.

26. Step through the rest of the application.

Review response data

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

200 OK 86431.06 ms

DETAILS ^

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

Response headers 3

Request headers 0

Redirects 0

Timings

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



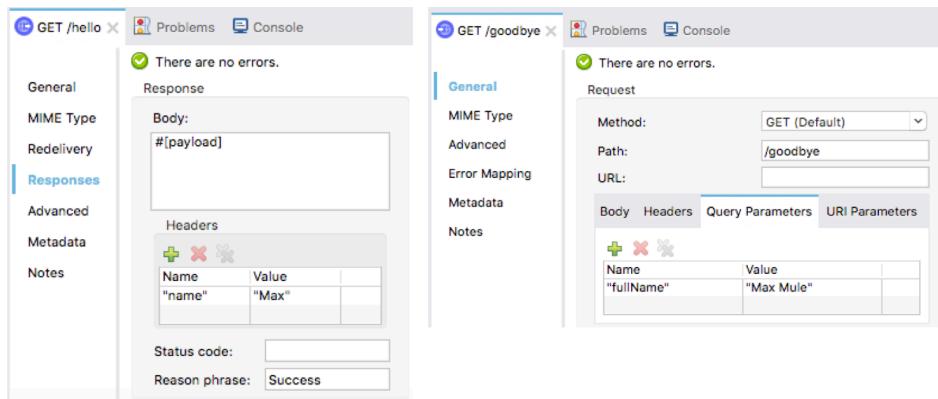
Goodbye

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

Walkthrough 6-4: Set request and response data

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

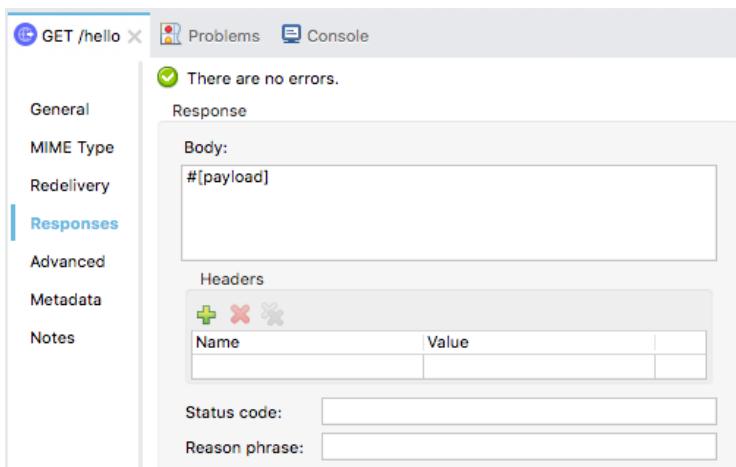
- View the default setting for a response body.
- Set a response header.
- View the default setting for a request body.
- Set a request query parameter.



View default response body

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

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



Set a response header

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

GET /hello X Problems Console

General

MIME Type

Redelivery

Responses

Advanced

Metadata

Headers

Name	Value
"name"	"Max"

Status code:

Reason phrase: Success

There are no errors.

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.

200 Success 1759.59 ms DETAILS ^

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

Response headers 4 Request headers 0 Redirects 0 Timings

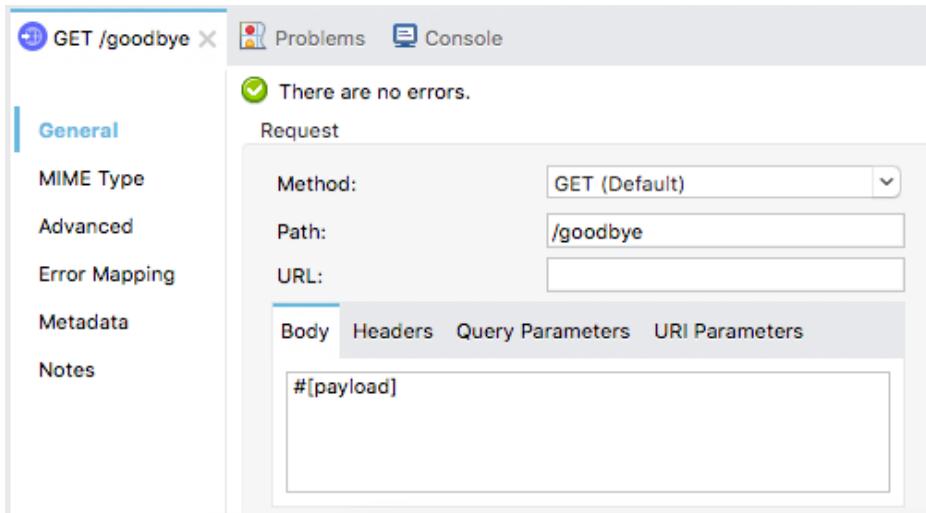
name: Max
content-type: application/octet-stream; charset=UTF-8
content-length: 7
date: Thu, 19 Apr 2018 20:46:15 GMT

Goodbye

View default request body

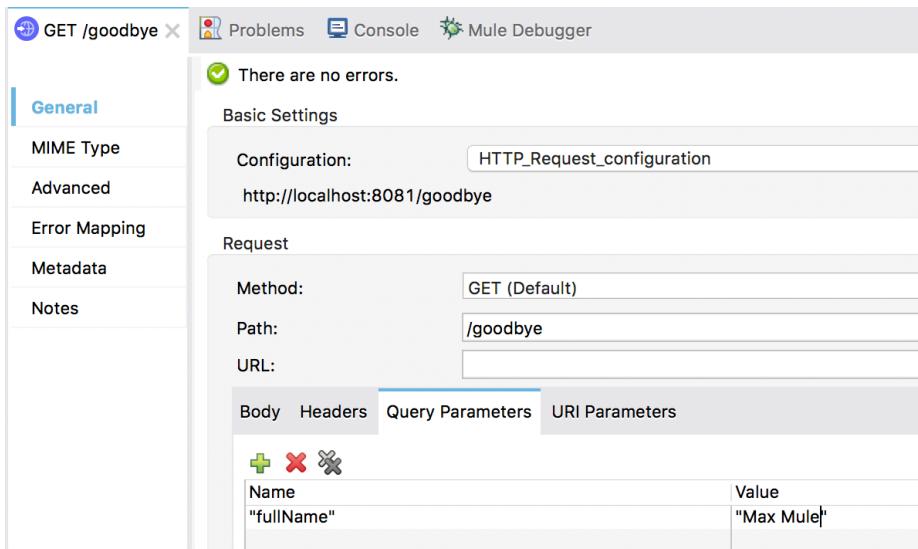
14. Return to Anypoint Studio and switch perspectives.
15. Double-click the GET /goodbye HTTP Request in helloFlow.

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.



The screenshot shows the Mule Debugger interface with a stack trace. The stack trace is as follows:

```
▼ [E] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes\n  Requests\n    [E] DOUBLE_TAB = "  "\n    [E] TAB = "  "\n    [E] clientCertificate = {java.security.cert.Certificate} null\n  ► [E] headers = {org.mule.runtime.http.api.domain.CaseInsensitiveMultiMap.ImmutableCaseInsensitiveMultiMap} MultiMap[[x-correlator]]\n    [E] listenerPath = "/goodbye"\n    [E] localAddress = "localhost/127.0.0.1:8081"\n    [E] method = "GET"\n  ▼ [E] queryParams = {org.mule.runtime.api.util.MultiMap} MultiMap[[fullName=[Max Mule]]]\n    ► [E] 0 = {java.util.AbstractMap$SimpleEntry} fullName=Max Mule\n      [E] key = "fullName"\n      [E] value = "Max Mule"\n    [E] queryString = "fullName=Max Mule"\n    [E] relativePath = "/goodbye"\n    [E] remoteAddress = "/127.0.0.1:61278"\n    [E] requestPath = "/goodbye"\n    [E] requestUri = "/goodbye?fullName=Max Mule"
```

24. Step through the rest of the application.

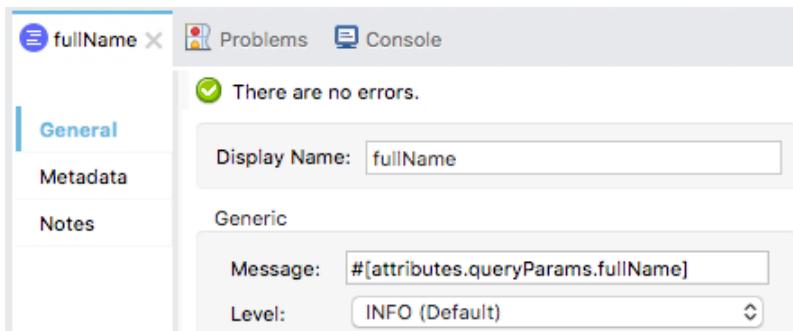
25. Switch to the Mule Design perspective.

26. Stop the project.

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

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

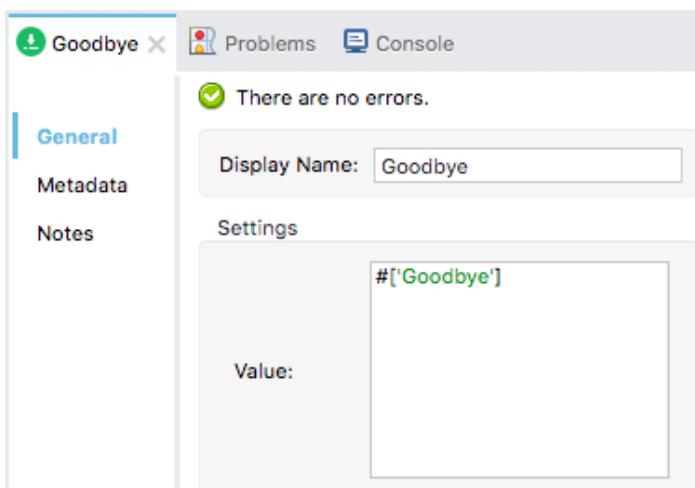
- Use expressions to set the payload and a logged value.
- Use expressions to set a response header and a request query parameter.
- In expressions, reference values for the event payload and attributes.
- Use the DataWeave upper() function and the concatenation, as, and default operators.



Use an expression to set the payload

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

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



4. Run the project.

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

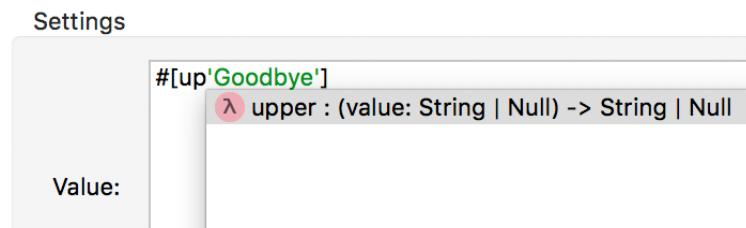
200 Success 3165.37 ms



Goodbye

Use a DataWeave function

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



9. Add parentheses around the Goodbye string.

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

10. Save the file to redeploy the application.

11. In Advanced REST Client, send the same request; the return string should now be in upper case.

200 Success 2253.95 ms

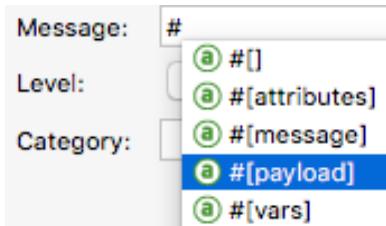


GOODBYE

Use an expression that references the payload in a Logger

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

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



16. Save the file to redeploy the application.
17. In Advanced REST Client, send the same request.
18. Return to the console in Anypoint Studio; you should see GOODBYE displayed for the second Logger instead of the entire event object.

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

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

19. Return to the properties view for the Logger in helloFlow.
20. Change the value to display the string Message in front of the payload.

Message: #[payload]

21. Save the file to redeploy the application.
22. In Advanced REST Client, send the same request.
23. Return to the console in Anypoint Studio; you should see the new string displayed.

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

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

Use the DataWeave concatenation operator

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

#['Message: ' ++ payload]

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

#['\nMessage: ' ++ payload]

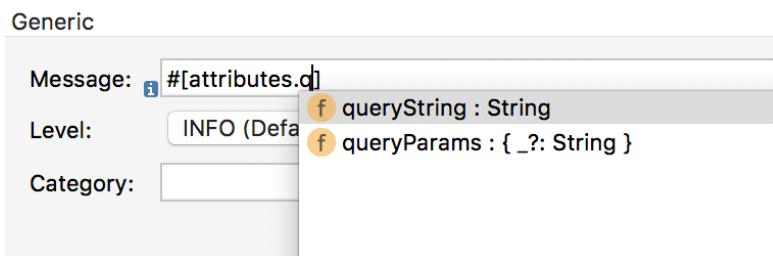
27. Save the file to redeploy the application.

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

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

Use an expression that references an attribute in a Logger

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



`#[attributes.queryParams]`

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

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

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

`#[attributes.queryParams.fullName]`

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

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

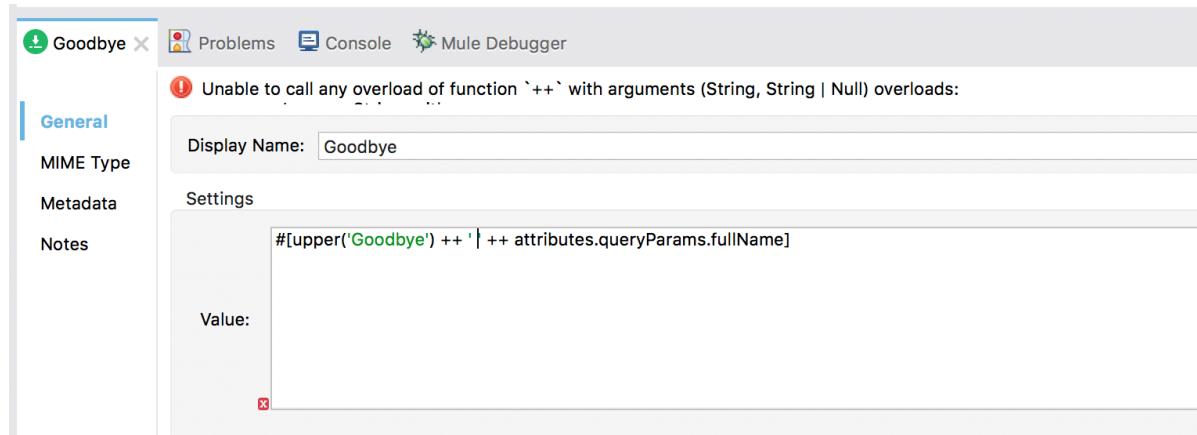
Use an expression that references an attribute when setting the payload

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

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

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

43. Review the error.



Use the as operator to coerce the attribute to a String

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

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

Note: This step coerces a null value to a String, to avoid UI errors. The recommended way to handle null values is by using a default value which is explained later in this walkthrough.

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.

200 Success 329.76 ms



GOODBYE Max Mule

Use an expression to set a request header

47. Return to the Anypoint Studio and navigate to the properties view for the GET /goodbye HTTP Request in helloFlow.

48. In the Request section, select the Query Parameters tab.

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

Name	Value
"fullname"	attributes.queryParams.fname

50. Click Apply Changes to redeploy the application.

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

200 Success 312.24 ms

GOODBYE max

Make a request and do not send a query parameter

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

500 Server Error 164.60 ms DETAILS ▾

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

Set a default value in an expression

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

54. Remove as String.

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

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

56. Click Apply Changes to redeploy the application.
57. In Advanced REST Client, send the same request; you should now see the default value Maxine displayed.

200 Success 307.33 ms



GOODBYE Maxine

Use a query parameter in the expression for a response header

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

attributes.queryParams.fname

Name	Value
"name"	attributes.queryParams.fname

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

200 Success 323.58 ms DETAILS ^

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

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

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

GOODBYE max

Debug and verify the query parameter is sent with the request

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



```
attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.ap
  ↗ DOUBLE_TAB = "    "
  ↗ TAB = "  "
  ↗ clientCertificate = {java.security.cert.Certificate} null
  ▶ headers = {org.mule.runtime.http.api.domain.CaseInsensitiveMultiMap.ImmutableCaseIn
  ↗ listenerPath = "/hello"
  ↗ localAddress = "localhost/127.0.0.1:8081"
  ↗ method = "GET"
  ▶ queryParams = {org.mule.runtime.api.util.MultiMap} MultiMap{[fname=[max]]}
  ↗ queryString = "fname=max"
  ↗ relativePath = "/hello"
  ↗
```

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

```
▶ queryParams = {org.mule.runtime.api.util.MultiMap} MultiMap{[fullName=[max]]}
  ↗ queryString = "fullName=max"
  ↗ relativePath = "/goodbye"
  ↗
```

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

```
▼ attributes = {org.mule.extension.http.api.HttpResponseAttributes} org.mule.extension.http.i
  ↗ DOUBLE_TAB = "    "
  ↗ TAB = "  "
  ▶ headers = {org.mule.runtime.http.api.domain.CaseInsensitiveMultiMap.ImmutableCaseIn
  ↗ reasonPhrase = ""
  ↗ serialVersionUID = -3131769059554988414
  ↗ statusCode = 200
  ↗
```

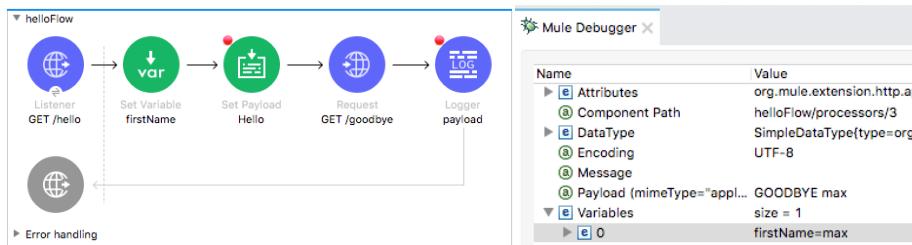
70. Step through the rest of the application.

71. Switch to the Mule Design perspective.

Walkthrough 6-6: Set and get variables

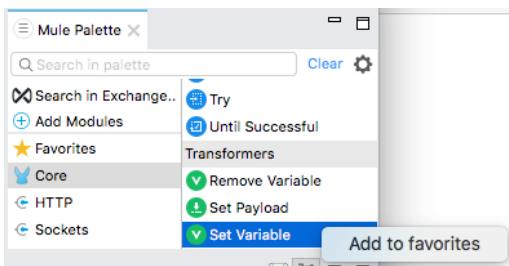
In this walkthrough, you create variables associated with an event. You will:

- 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.



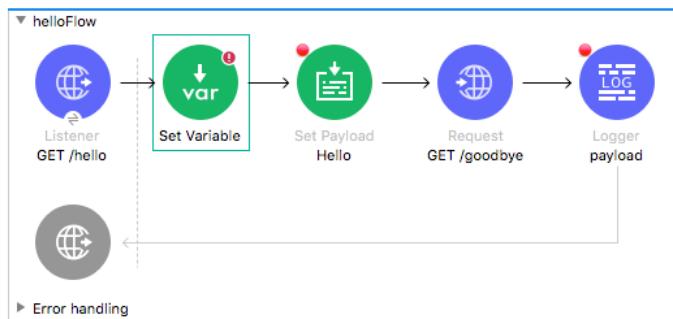
Add the Set Variable transformer to the Favorites section of the Mule Palette

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.

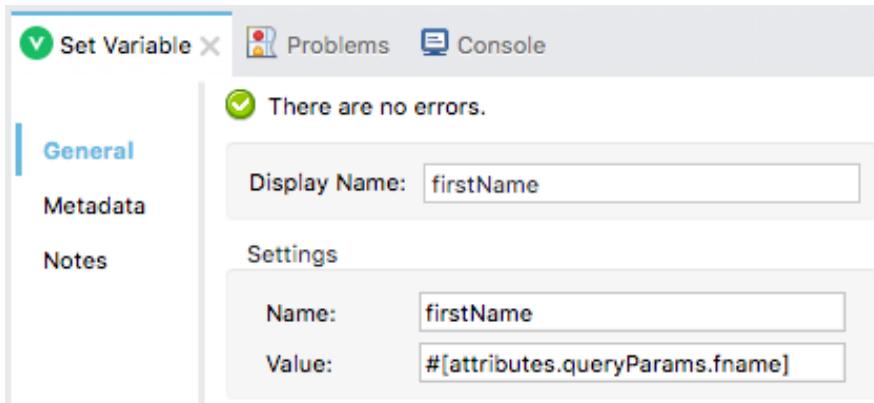


Create a variable

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.



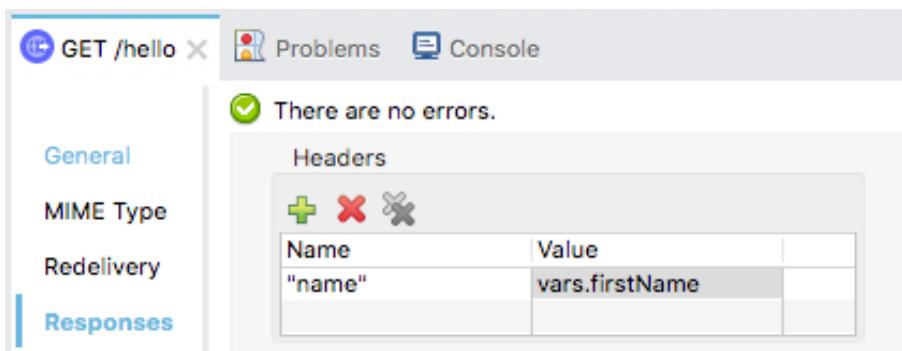
5. In the Set Variable properties view, set the display name and name to firstName.
6. Set the value to your query parameter, fname.



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.

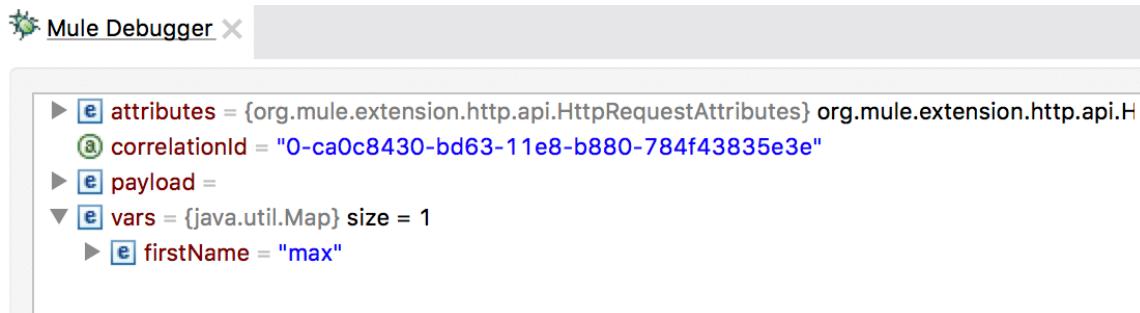
`vars.firstName`



View variables in the Mule Debugger

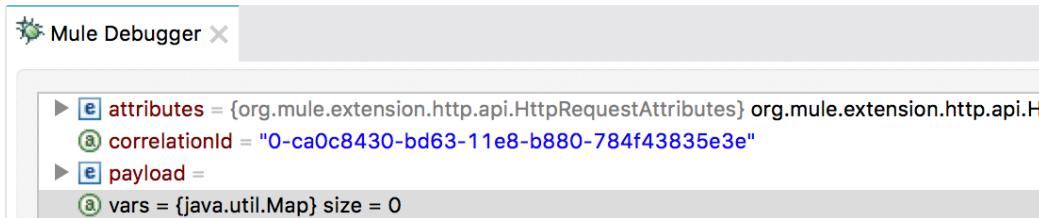
10. Save the file to redeploy the project in debug mode.
11. In Advanced REST Client, send the same request.

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



```
▶ [e] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.H
@ correlationId = "0-ca0c8430-bd63-11e8-b880-784f43835e3e"
▶ [e] payload =
▼ [e] vars = {java.util.Map} size = 1
  ▶ [e] firstName = "max"
```

13. Step into goodbyeFlow; you should see no variables listed in the vars section.



```
▶ [e] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.H
@ correlationId = "0-ca0c8430-bd63-11e8-b880-784f43835e3e"
▶ [e] payload =
@ vars = {java.util.Map} size = 0
```

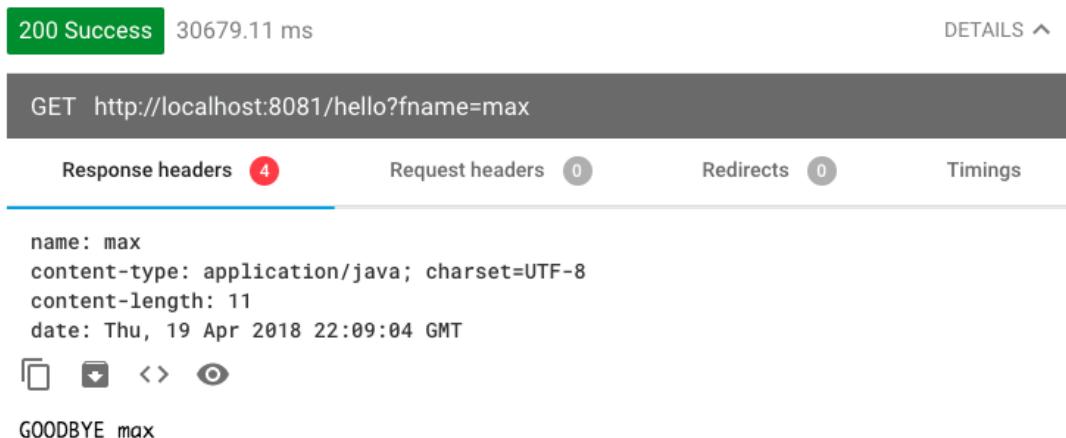
14. Step back to helloFlow; you should see the Variables section again with your firstName variable.



```
▶ [e] attributes = {org.mule.extension.http.api.HttpResponseAttributes} org.mule.extension.http.api.
@ correlationId = "0-ca0c8430-bd63-11e8-b880-784f43835e3e"
▶ [e] payload = GOODBYE max
▼ [e] vars = {java.util.Map} size = 1
  ▶ [e] firstName = "max"
```

15. Step through the rest of the application.

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



200 Success 30679.11 ms DETAILS ^

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

Response headers	4	Request headers	0	Redirects	0	Timings
name: max						
content-type: application/java; charset=UTF-8						
content-length: 11						
date: Thu, 19 Apr 2018 22:09:04 GMT						

GOODBYE max

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

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

Parameters ▾

200 Success 7967.76 ms DETAILS ^

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

Response headers 4 Request headers 0 Redirects 0 Timings

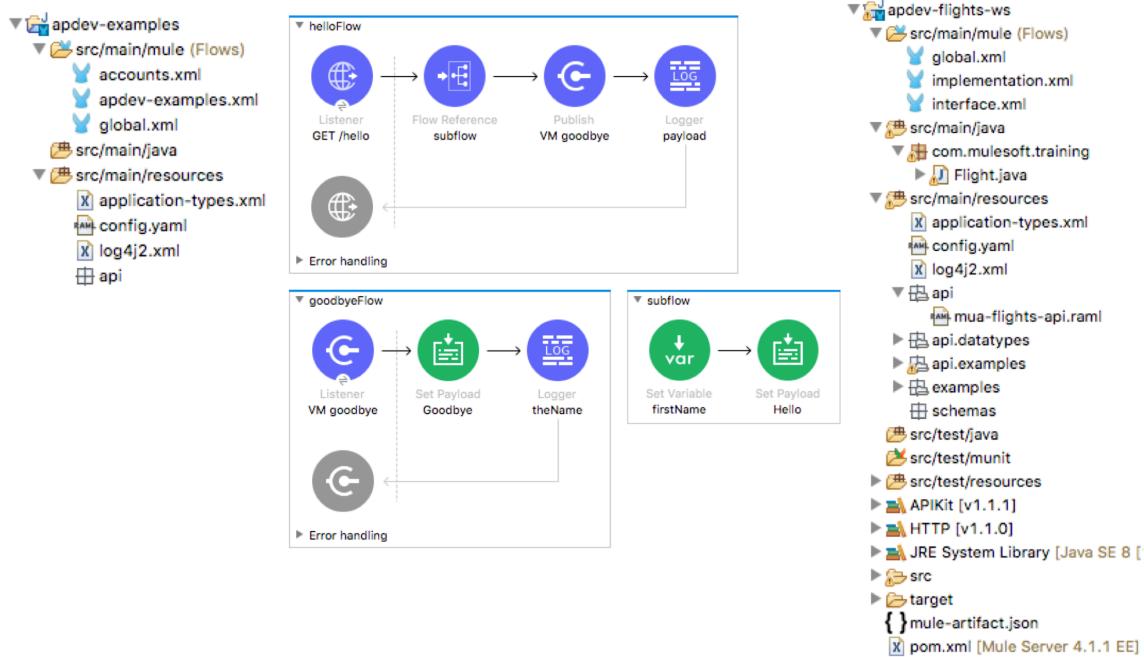
```
name: Maxwell
content-type: application/java; charset=UTF-8
content-length: 15
date: Thu, 19 Apr 2018 22:10:04 GMT
```

□ ↗ <> ○

GOODBYE Maxwell

20. Return to Anypoint Studio and switch perspectives.

Module 7: Structuring Mule Applications



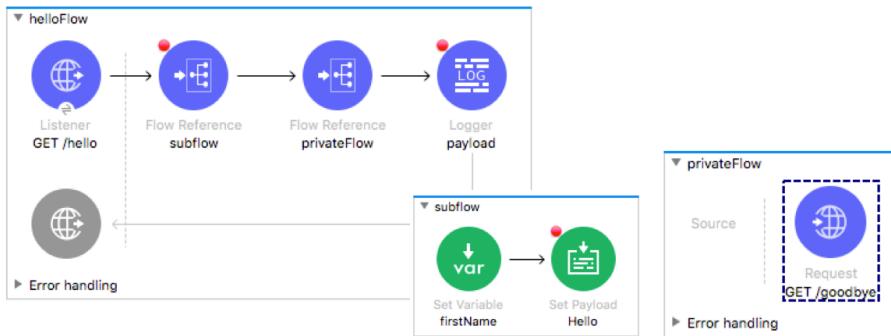
At the end of this module, you should be able to:

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

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

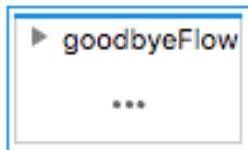
In this walkthrough, you continue to work with apdev-examples.xml. You will:

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



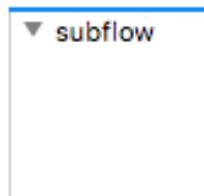
Collapse a flow

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

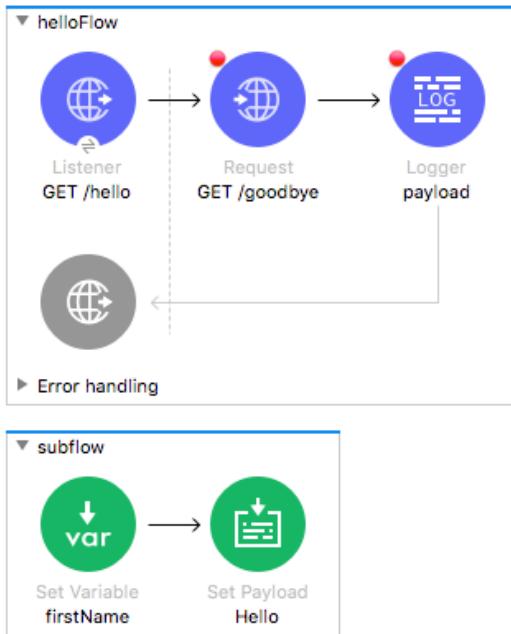


Create a subflow

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

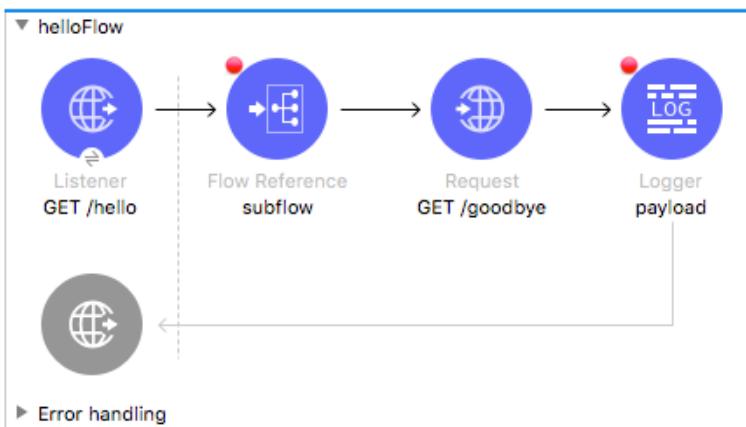


- Select the Set Variable and Set Payload transformers in helloFlow and drag them into the subflow.



Reference a subflow

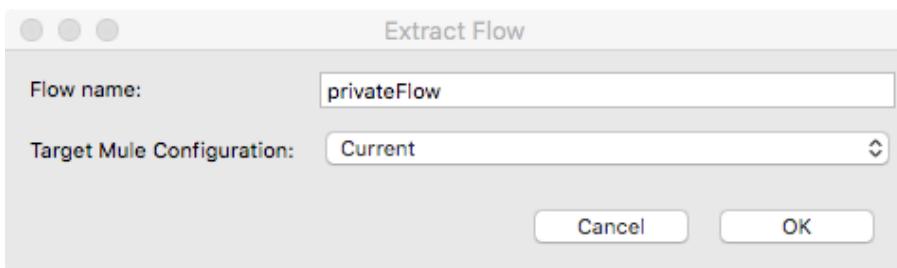
- Drag a Flow Reference component from the Mule Palette and drop it into helloFlow between the GET /hello HTTP Listener and the GET /goodbye HTTP Request.
- In the Flow Reference properties view, set the flow name and display name to subflow.
- Add a breakpoint to the subflow Flow Reference.
- Remove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



Extract processors into another flow

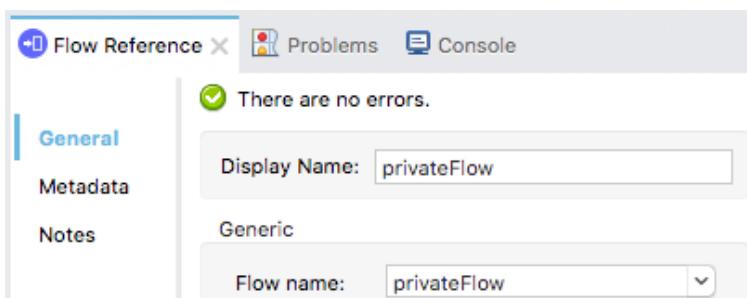
- Right-click the GET /goodbye HTTP Request and select Extract to > Flow.

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



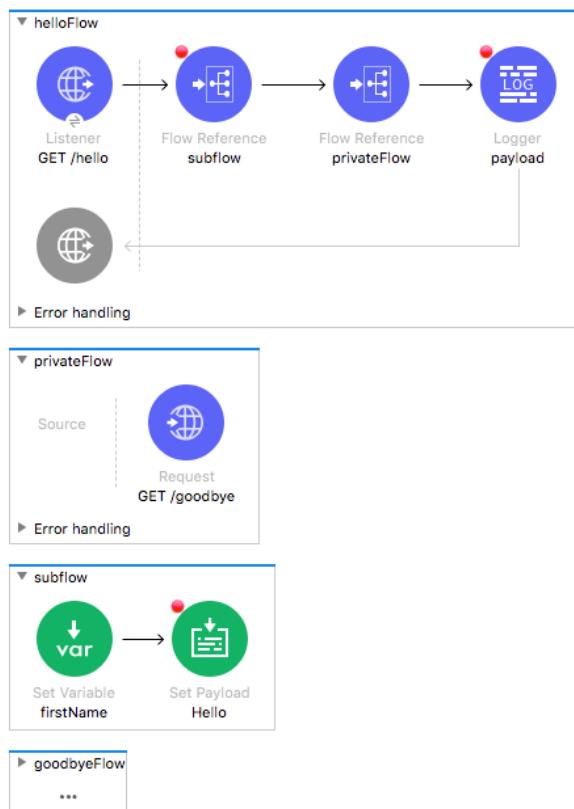
13. Leave the target Mule configuration set to current and click OK.

14. Look at the new Flow Reference properties view; set the display name to privateFlow.



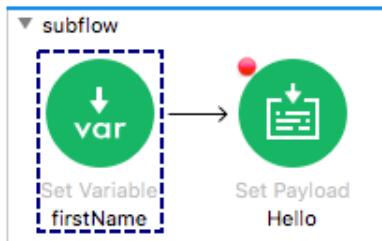
14.

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.

Mule Debugger

```
▶ [e] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ correlationId = "0-d52ab3d0-bda1-11e8-8fef-784f43835e3e"
▶ [e] payload = "Hello"
▶ [e] vars = {java.util.Map} size = 1
  ▶ [e] firstName = "Maxwell"
```

apdev-examples X pom.xml

Error handling

privateFlow

```
Source --- Request [GET /goodbye]
```

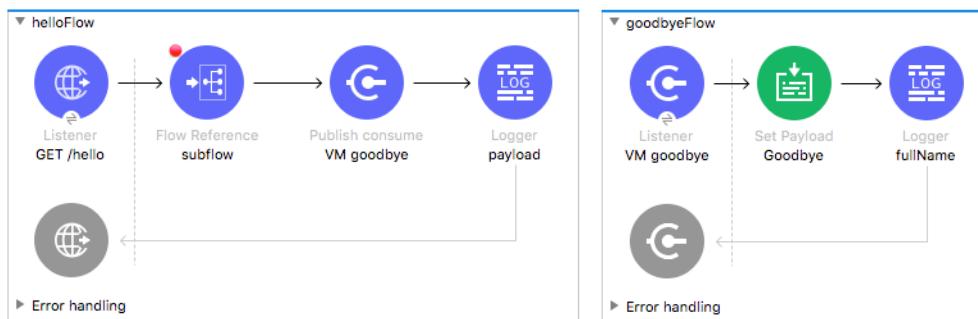
Message Flow Global Elements Configuration XML

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

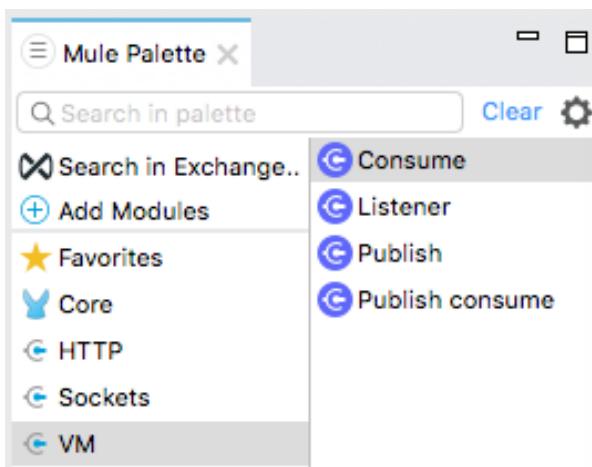
In this walkthrough, you pass messages between flows using asynchronous queues. You will:

- 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.

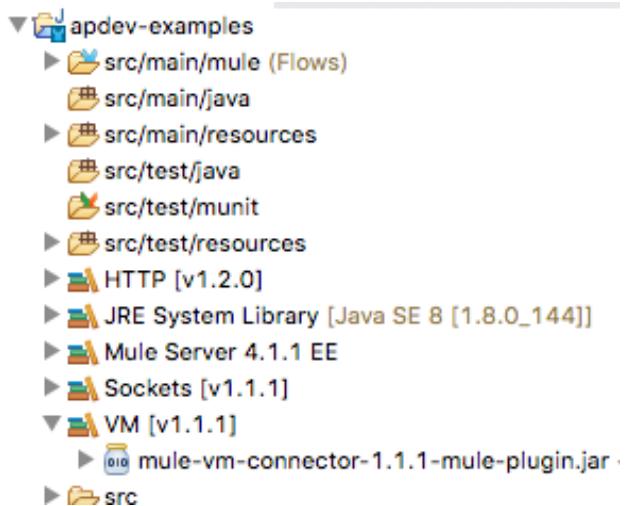


Add the VM module to the project

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.

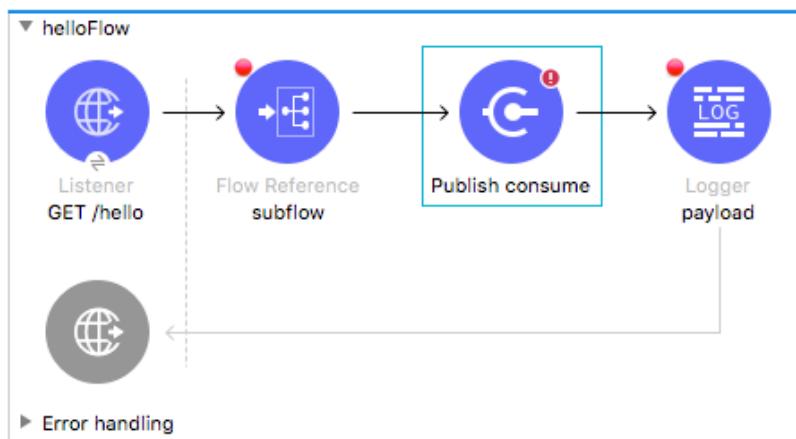


4. In the Project Explorer, locate the JAR file for the VM connector.

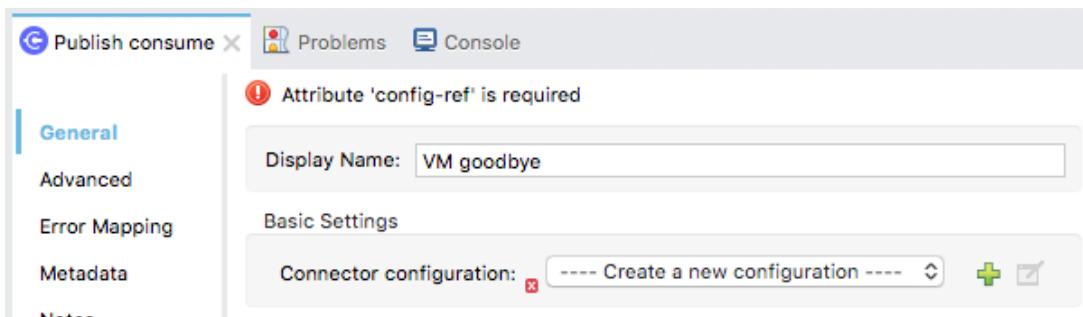


Add a VM Publish Consume operation

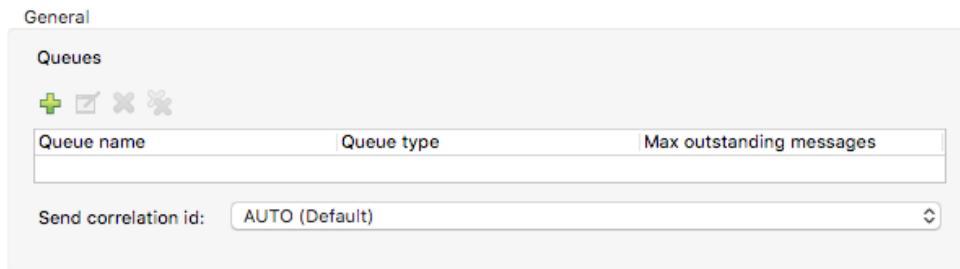
5. In helloFlow, delete the privateFlow Flow Reference.
6. Select VM in the Mule Palette.
7. Select the Publish consume operation and drag and drop it before the Logger in helloFlow.



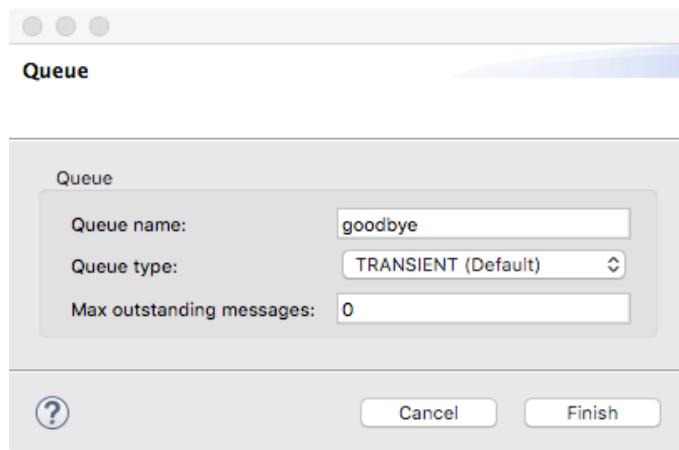
8. In the Publish consume properties view, change the display name to VM goodbye.



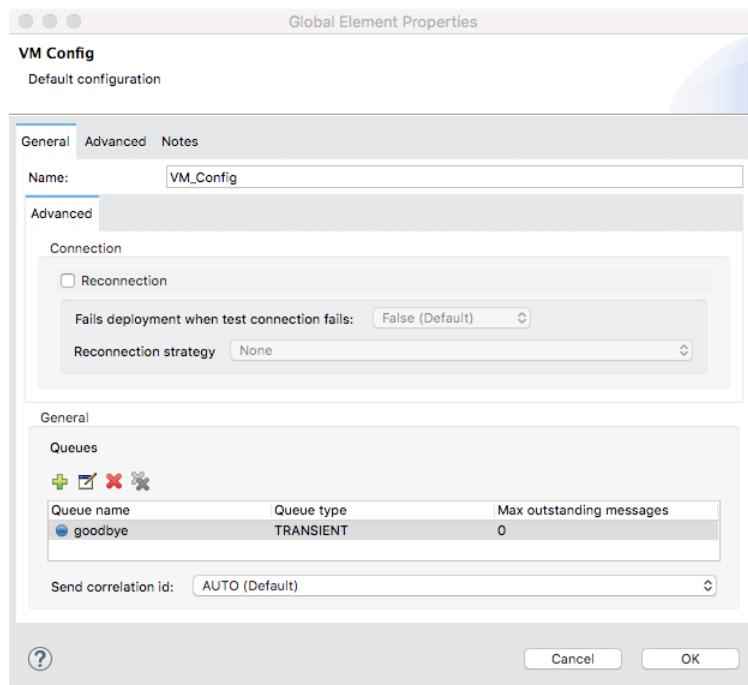
9. Click the Add button next to connector configuration.
10. In the Global Element Properties dialog box, click the Add Queue button.



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



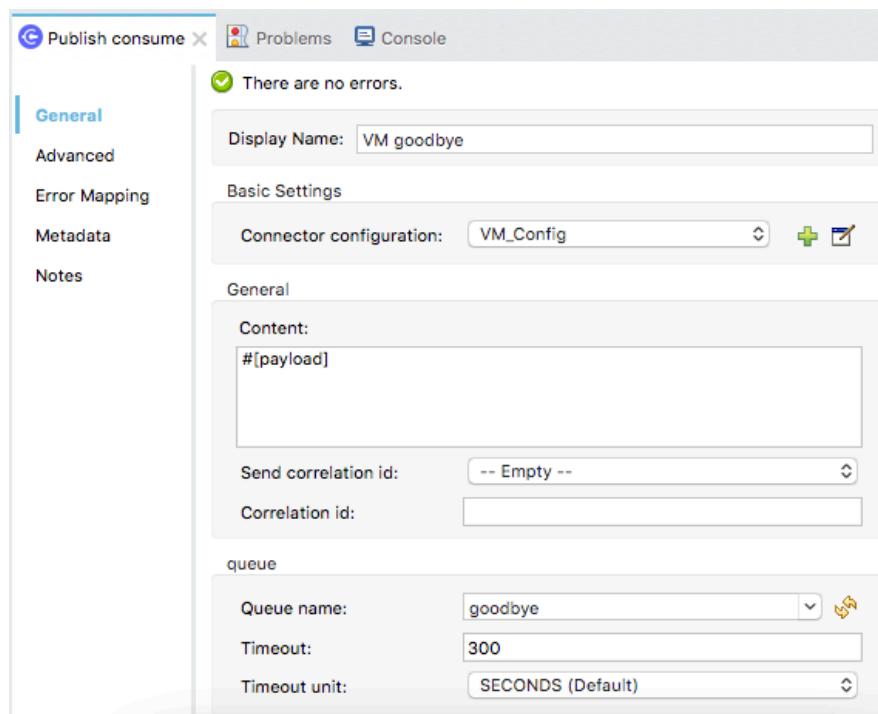
12. In the Global Element Properties dialog box, click OK.



13. In the VM goodbye Publish consume properties view, set the queue name to goodbye.

Note: If you do not see the queue name in the drop-down menu, click the refresh icon for it to be populated and then set it.

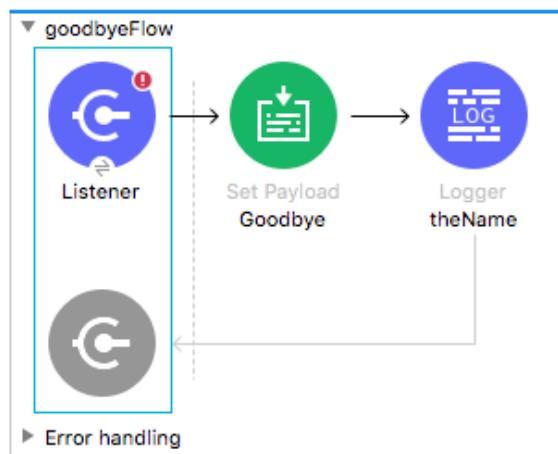
14. Set the timeout to 300 seconds for debugging purposes.



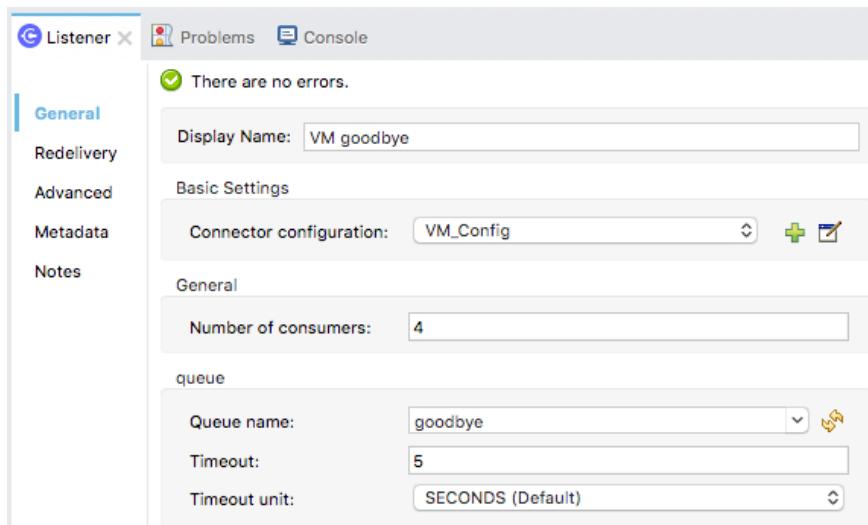
Add a VM Listener

15. Expand goodbyeFlow and delete its HTTP Listener.

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



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



Debug the application

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

```

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

```

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

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

The screenshot shows the Mule Debugger interface. The top window is titled "Mule Debugger" and displays the following variable information:

```
▶ [e] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ correlationId = "0-c1e7ddb0-bda2-11e8-a654-784f43835e3e"
  ▶ [e] payload = "Hello"
    @ ^mediaType = */*
  ▶ [e] vars = {[java.util.Map] size = 1}
    ▶ [e] firstName = "Maxwell"
```

The bottom window is titled "apdev-examples" and shows a flow diagram:

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

The "Listener" component has a blue border, indicating it is selected. The "VM goodbye" component is enclosed in a dashed blue box.

26. Step into goodbyeFlow.

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

The screenshot shows the Mule Debugger interface. The top window is titled "Mule Debugger" and displays the following variable information:

```
▶ [e] attributes = {org.mule.extensions.vm.api.VMMessageAttributes} org.mule.extensions.vm.api.VMMessageAttributes
  @ correlationId = "0-c1e7ddb0-bda2-11e8-a654-784f43835e3e"
  ▶ [e] payload = "Hello"
    @ ^mediaType = */*; charset=UTF-8
  ▶ [e] vars = {[java.util.Map] size = 0}
```

The bottom window is titled "apdev-examples" and shows the "goodbyeFlow" component:

```
graph LR
    Listener[Listener  
goodbyeFlow] --> SetPayload[Set Payload  
Goodbye]
    SetPayload --> Logger[Logger  
fullName]
```

The "Set Payload Goodbye" component is enclosed in a dashed blue box.

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

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

The screenshot shows two main sections. The top section is titled "Mule Debugger" and displays event details:

- attributes = {org.mule.extensions.vm.api.VMMessagesAttributes} org.mule.extensions.vm.api.VMMessagesAttributes
- correlationId = "0-c1e7ddb0-bda2-11e8-a654-784f43835e3e"
- payload = "GOODBYE Maxine"
- ^mediaType = application/java; charset=UTF-8
- vars = {java.util.Map} size = 1
 - firstName = "Maxwell"

The bottom section is titled "apdev-examples" and shows the application structure:

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

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

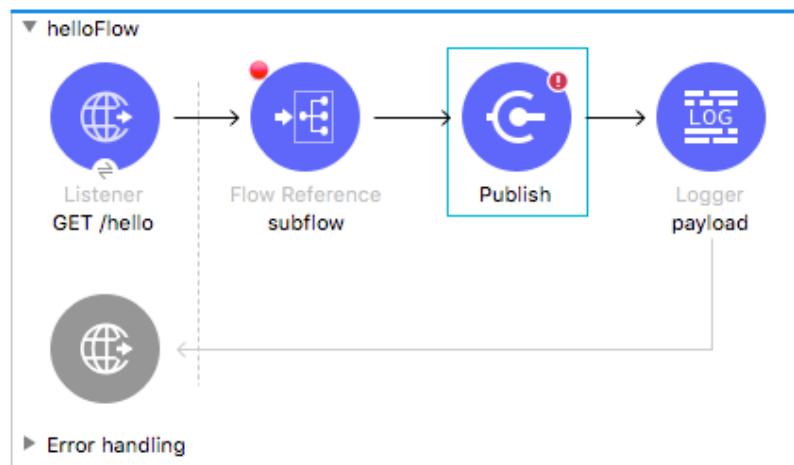
30. Step through the rest of the application.

31. Switch perspectives.

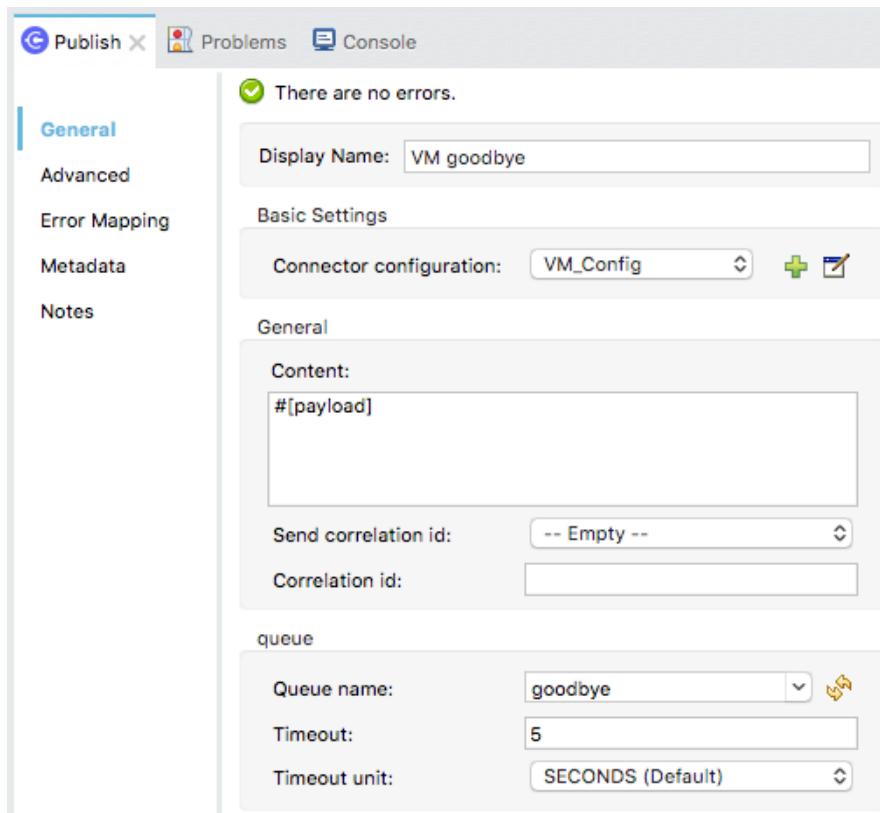
Change the VM Publish Consume operation to Publish

32. Delete the VM Publish consume operation in helloFlow.

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



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



Debug the application

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

41. Look at the value of the payload.

The screenshot shows the Mule Studio interface. At the top, there is a 'Mule Debugger' window showing the current state of variables:

```
▶ [E] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ correlationId = "0-87b2c8c0-bda3-11e8-a654-784f43835e3e"
▶ [E] payload = "Hello"
▼ [E] vars = {java.util.Map} size = 1
  ▶ [E] firstName = "Maxwell"
```

Below the debugger is the project navigation bar with 'apdev-examples' selected. The main area displays a flow diagram:

```
graph LR
    Listener((Listener  
GET /hello)) --> Subflow((Flow Reference  
subflow))
    Subflow --> Publish((Publish  
VM goodbye))
    Publish --> Logger((Logger))
```

The 'Logger' component is highlighted with a dashed blue border.

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

43. Step to the end of the application.

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

200 Success 160792.30 ms



Hello

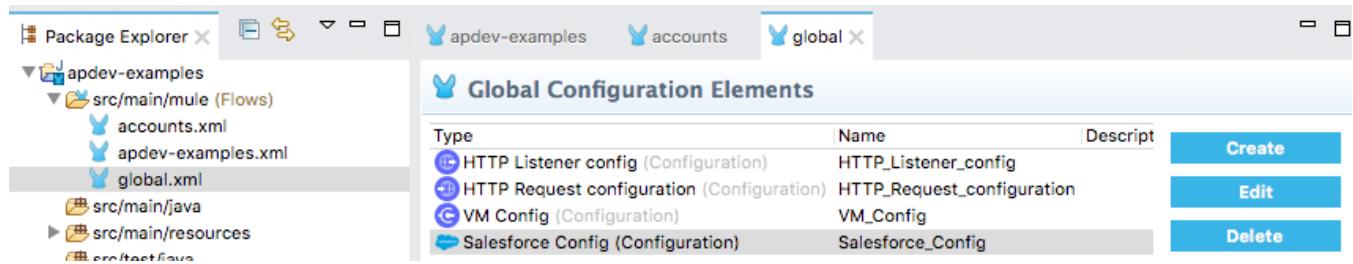
45. Return to Anypoint Studio and switch perspectives.

46. Stop the project.

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

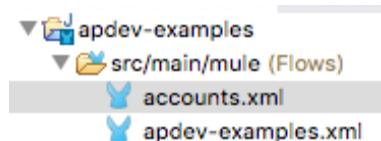
In this walkthrough, you refactor your apdev-examples project. You will:

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

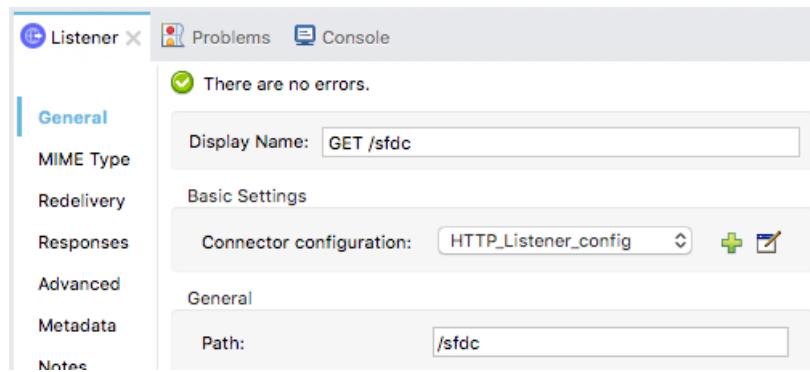


Create a new configuration file

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



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



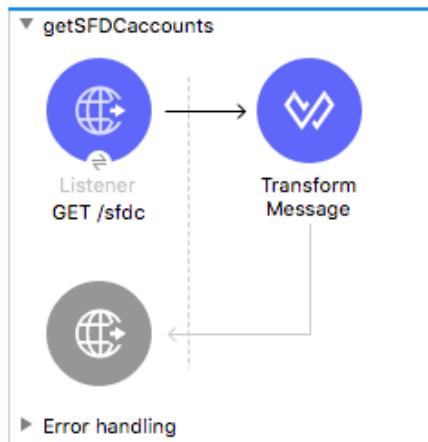
8. Add a Transform Message component to the flow.
9. In the Transform Message properties view, change the output type to application/json and set the expression to payload.

```

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

```

10. Change the name of the flow to getSFDCCaccounts.



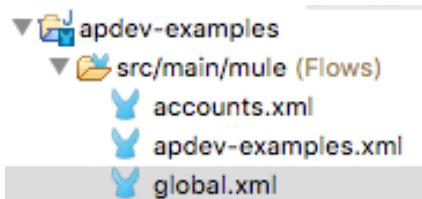
11. Switch to the Global Elements view; you should not see any global elements.

The screenshot shows the 'Global Configuration Elements' view. At the top, there are tabs for 'apdev-examples' and 'accounts'. The 'accounts' tab is selected. Below the tabs is a header bar with 'Create', 'Edit', and 'Delete' buttons. The main area is a table with columns 'Type', 'Name', and 'Description'.

Type	Name	Description

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.

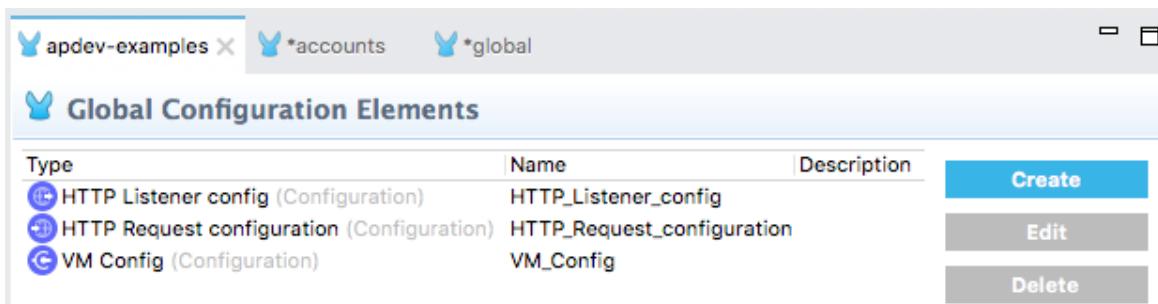


```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule http://www.mulesoft.org/schema/mule/core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

Move the existing global elements to the new global configuration file

15. Return to apdev-examples.xml.

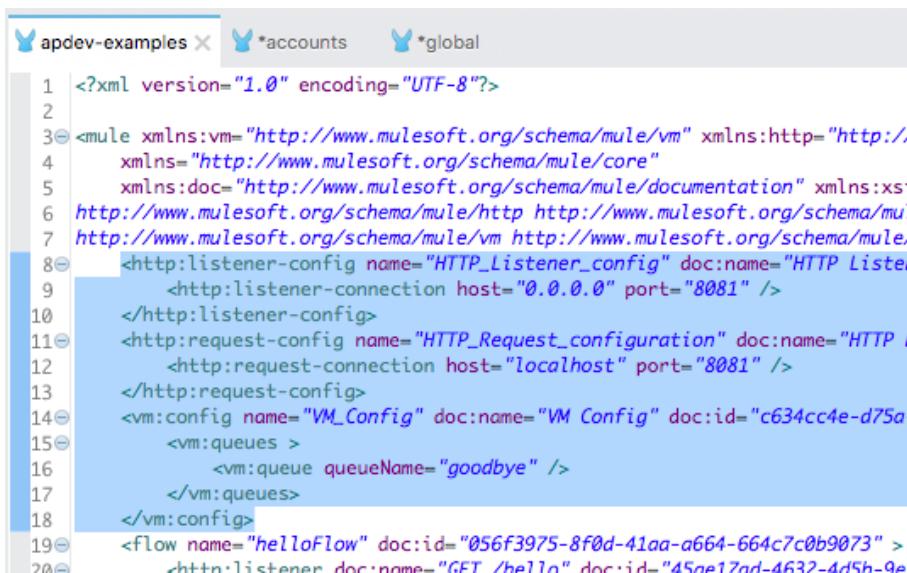
16. Switch to the Global Elements view and see there are three configurations.



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

17. Switch to the Configuration XML view.

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



```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:vm="http://www.mulesoft.org/schema/mule/vm" xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:doc="http://www.mulesoft.org/schema/mule/documentation" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule http://www.mulesoft.org/schema/mule
      http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm"
      <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener">
        <http:listener-connection host="0.0.0.0" port="8081" />
      </http:listener-config>
      <http:request-config name="HTTP_Request_configuration" doc:name="HTTP Request">
        <http:request-connection host="localhost" port="8081" />
      </http:request-config>
      <vm:config name="VM_Config" doc:name="VM Config" doc:id="c634cc4e-d75a-43d0-8f0d-41aa-a664-664c7c0b9073">
        <vm:queues>
          <vm:queue queueName="goodbye" />
        </vm:queues>
      </vm:config>
      <flow name="helloFlow" doc:id="056f3975-8f0d-41aa-a664-664c7c0b9073">
        <http:listener doc:name="HTTP / hello" doc:id="45ae17ad-4f32-4d5h-9a53" />
```

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

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

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3<mule xmlns:http="http://www.mulesoft.org/schema/mule/http"
4   xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
5   xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mulesoft.org/schema/mule/core/description"
6   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xsi:schemaLocation="
8     http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http.xsd
9     http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm.xsd
10    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core.xsd">
11<http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener Configuration">
12  <http:listener-connection host="0.0.0.0" port="8081" />
13</http:listener-config>
14<http:request-config name="HTTP_Request_configuration" doc:name="HTTP Request Configuration">
15  <http:request-connection host="localhost" port="8081" />
16</http:request-config>
17<vm:config name="VM_Config" doc:name="VM Config" doc:id="c634cc4e-d1d1-43f2-833a-1a2a2a2a2a2a">
18  <vm:queues>
19    <vm:queue queueName="goodbye" />
20  </vm:queues>
21</vm:config>
22
23</mule>

```

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

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

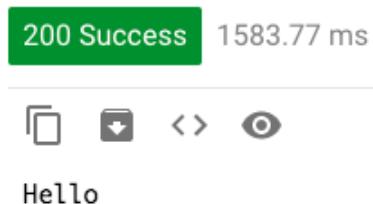
Test the application

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

The screenshot shows the configuration for the GET /hello endpoint. The 'Connector configuration' dropdown is set to 'HTTP_Listener_config'. Other visible settings include 'Display Name: GET /hello' and 'Basic Settings'.

25. Run the project.

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

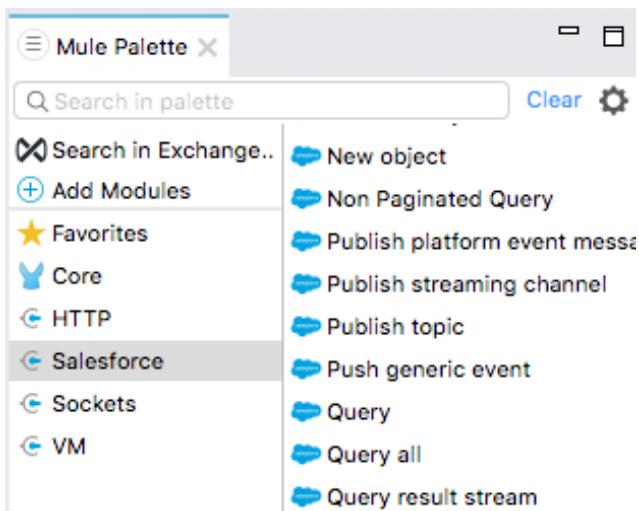


Create a new global element for the Salesforce component in global.xml

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.

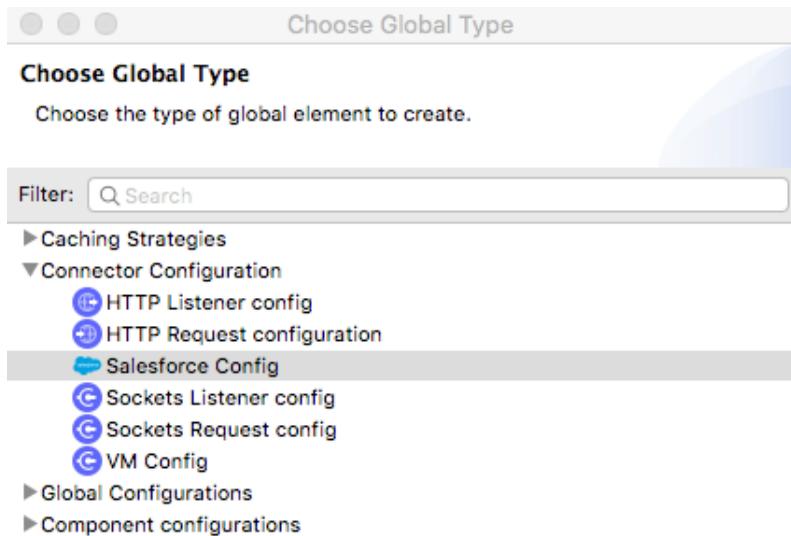


30. Locate the new Salesforce JAR in the project.

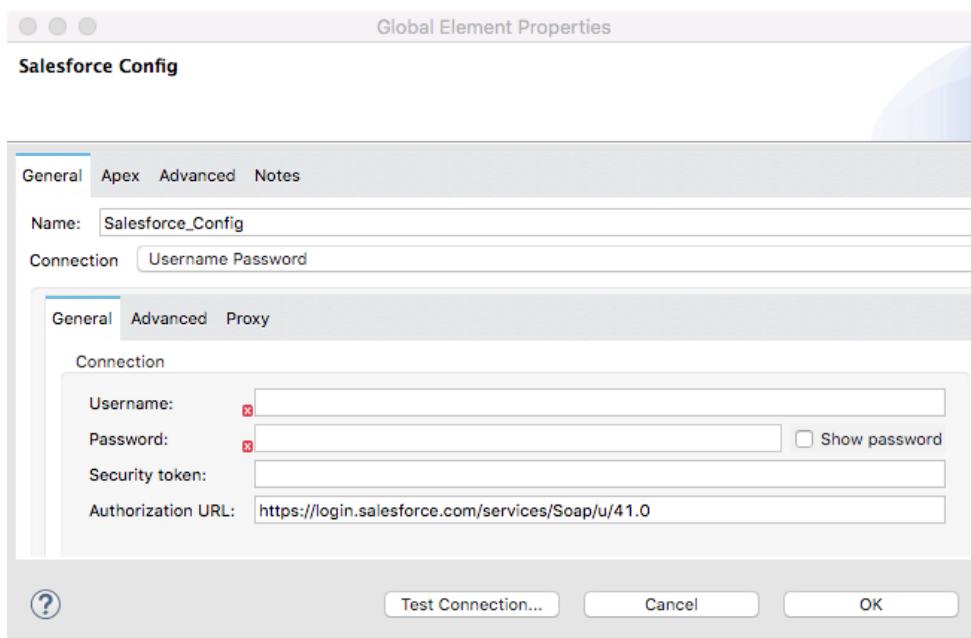
31. Return to global.xml.

32. Click Create.

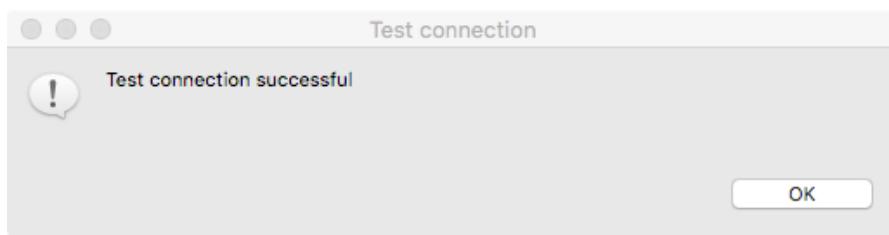
33. In the Choose Global Type dialog box, select Connector Configuration > Salesforce Config and click OK.



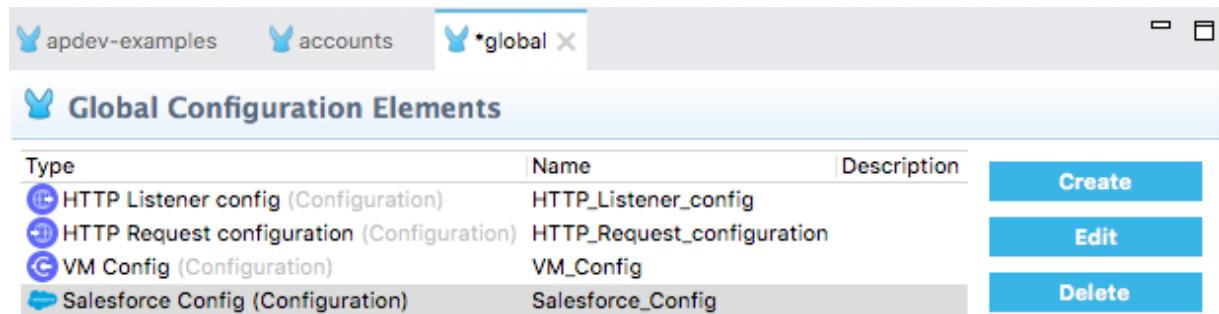
34. In the Global Element Properties dialog box enter your Salesforce username, password, and security token.



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.



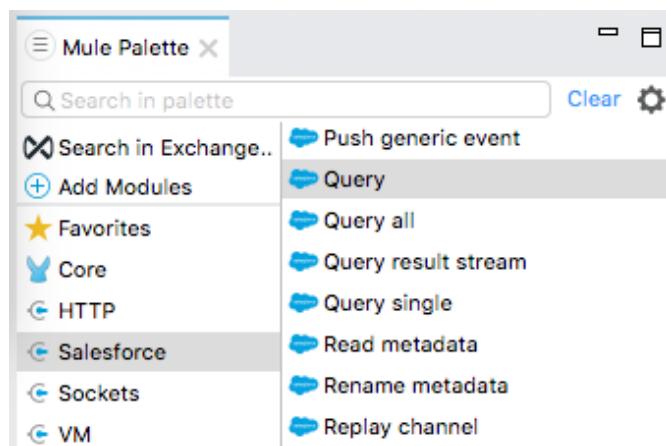
The screenshot shows the 'Global Configuration Elements' screen in the Mule Studio interface. At the top, there are tabs for 'apdev-examples', 'accounts', and '*global'. The '*global' tab is selected. Below the tabs is a title 'Global Configuration Elements'. A table lists four configuration elements:

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

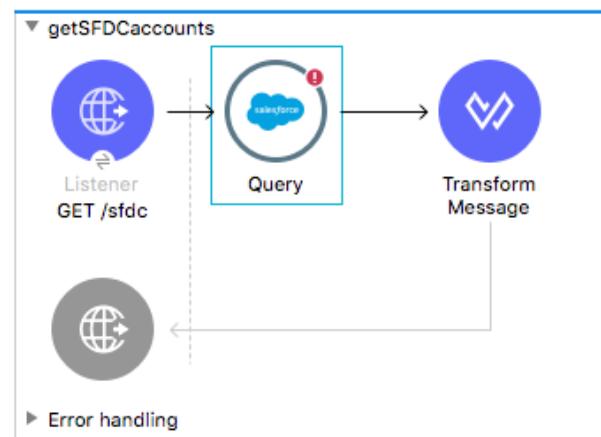
On the right side of the table are three buttons: 'Create', 'Edit', and 'Delete'.

Add a new Salesforce operation that uses the global configuration

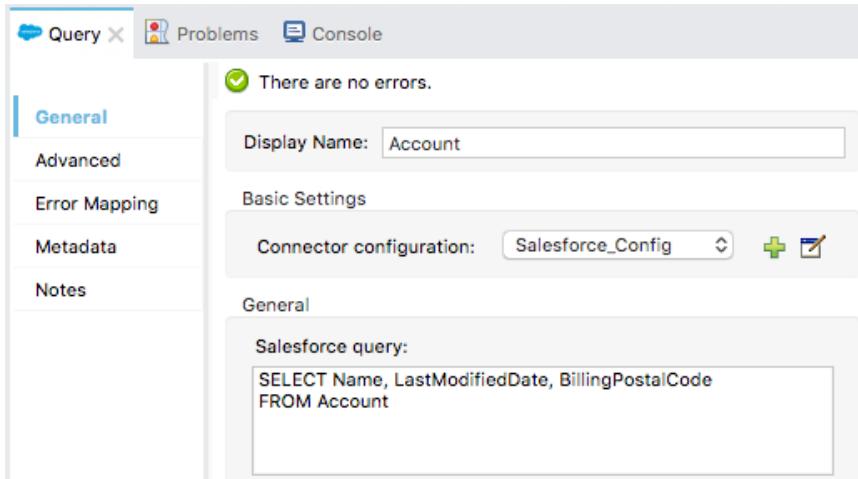
38. Return to the Message Flow view in accounts.xml.
39. In the Mule Palette, select Salesforce.



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

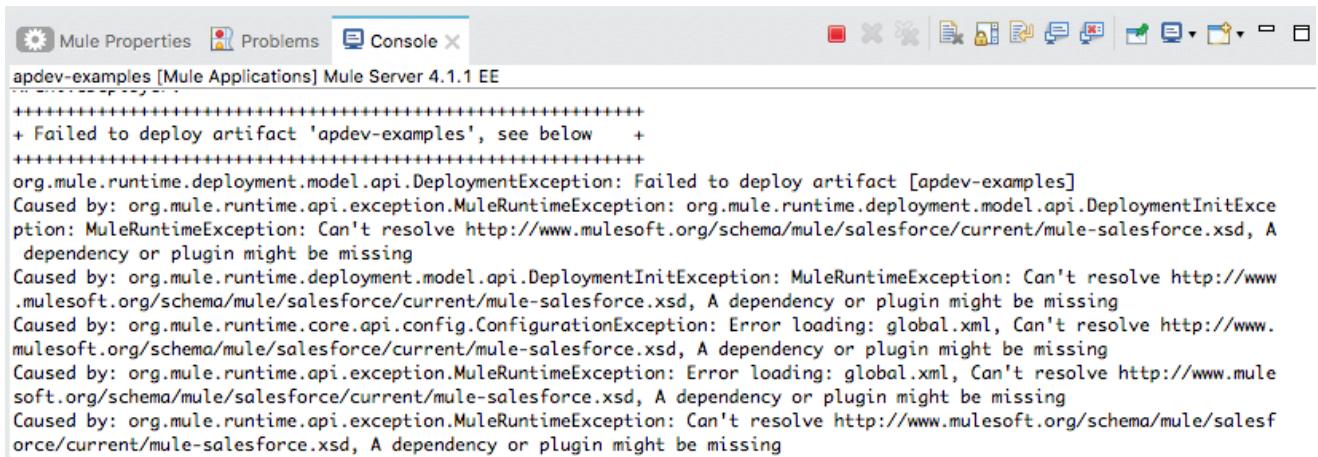


41. In the Query properties view, set the display name to Account.
42. Set the connector configuration to the existing Salesforce_Config.
43. Return to the course snippets.txt file and copy the Salesforce query.
44. Return to Anypoint Studio and paste the query in the Salesforce query section of the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



Test the application

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



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

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

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

SEND :
Parameters ▾

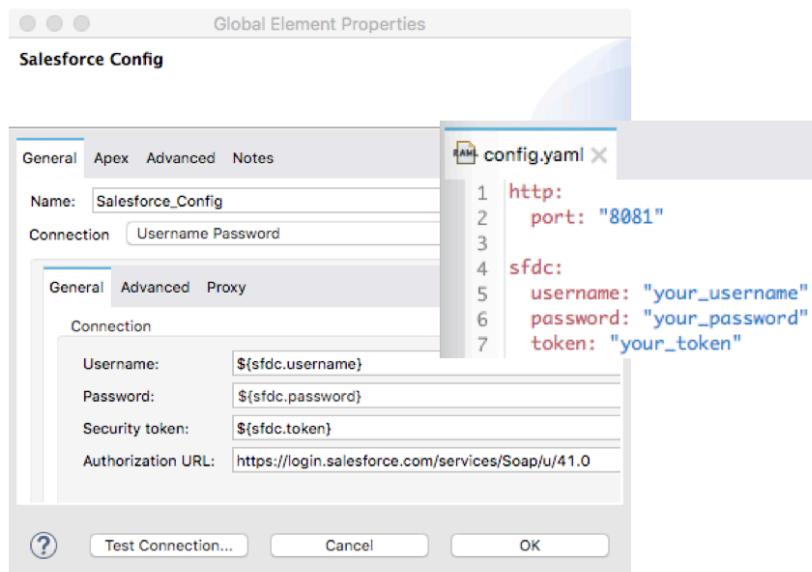
200 OK 1757.84 ms DETAILS ▾

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

Walkthrough 7-4: Use property placeholders in connectors

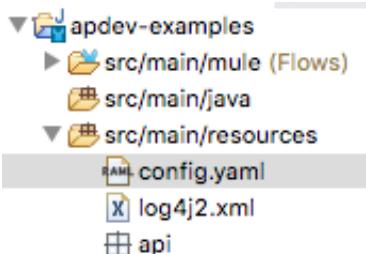
In this walkthrough, you introduce properties into your API implementation. You will:

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



Create a properties file

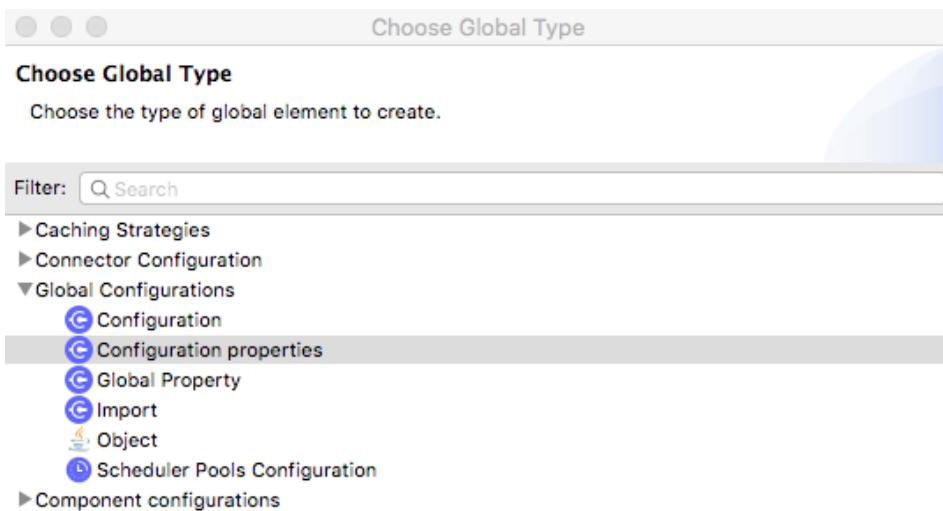
1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > File.
3. Set the file name to config.yaml and click Finish.



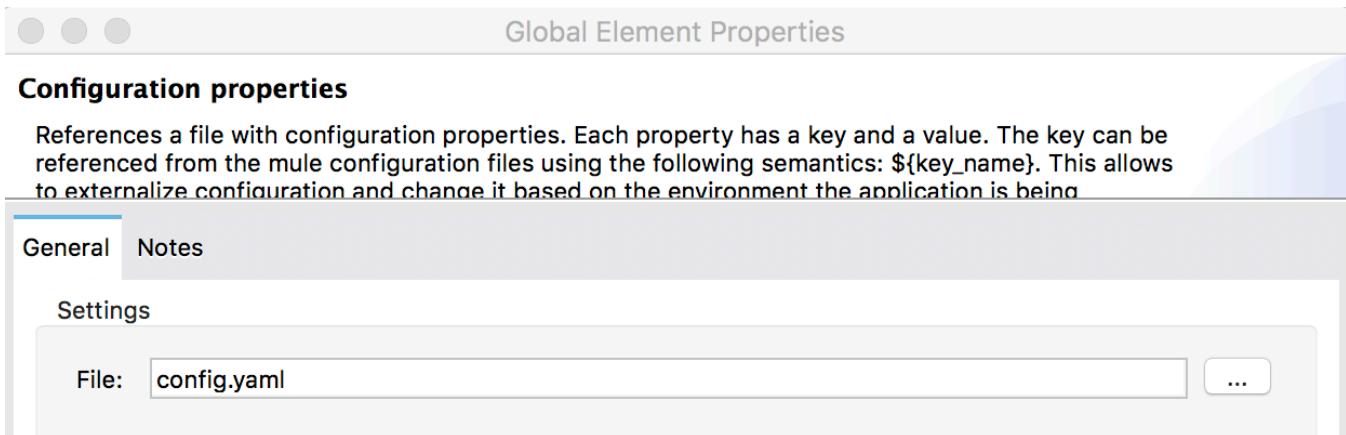
Create a Configuration properties global element

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

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



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



Parameterize the HTTP Listener port

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

The screenshot shows the Mule Studio interface with several tabs at the top: 'apdev-examples', 'accounts', '*global', and '*config.yaml'. The '*config.yaml' tab is selected and shows the following YAML code:

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

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

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

HTTP Listener config
Configuration element for a HttpListener.

General Notes

Name: **HTTP_Listener_config**

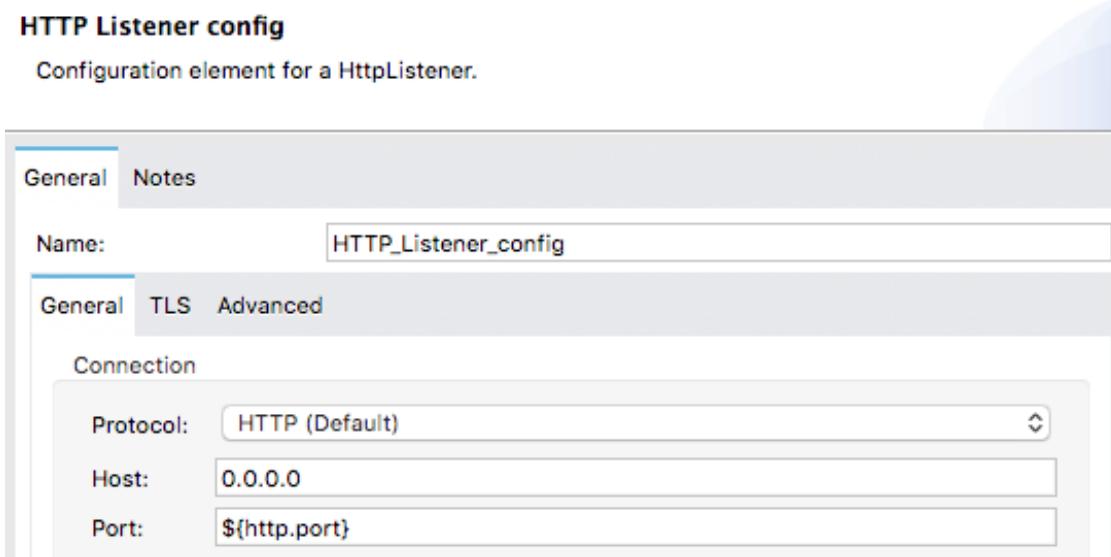
General TLS Advanced

Connection

Protocol: **HTTP (Default)**

Host: **0.0.0.0**

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



14. Click OK.

Parameterize the Salesforce credentials

15. Double-click the Salesforce Config global element.

16. Copy the token value and click OK.

17. Return to config.yaml .

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

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

apdev-examples accounts *global *config.yaml X

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

20. Save the file.

21. Return to global.xml.

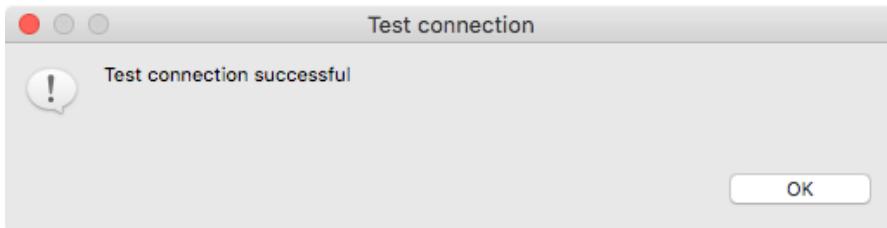
22. Double-click the Salesforce Config global element.

23. Change the values to use the application properties.

Connection

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

24. Click Test Connection and make sure it succeeds.



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

25. Click OK.

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

Test the application

27. Save all files to redeploy the application.

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

Method Request URL

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

Parameters ▾

200 OK 1757.84 ms DETAILS ▾

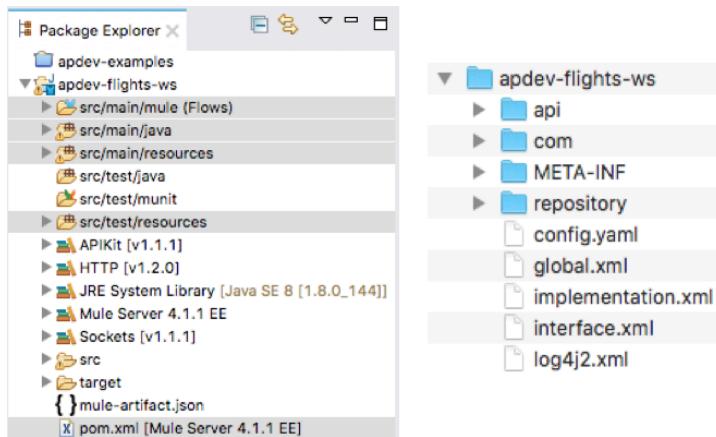
[
 {
 "LastModifiedDate": "2015-07-12T21:00:51.000Z",
 "BillingPostalCode": null,
 "Id": null,
 "type": "Account",
 "Name": "GenePoint"
 },
 {
 "LastModifiedDate": "2015-07-12T21:00:51.000Z",
 "BillingPostalCode": null
 }

29. Return to Anypoint Studio and stop the project.

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

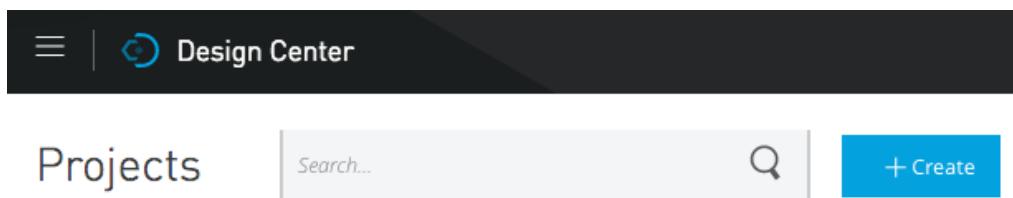
In this walkthrough, you create a new project for the Mule United Airlines (MUA) flights application that you will build during the course and then review and organize its files and folders. You will:

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



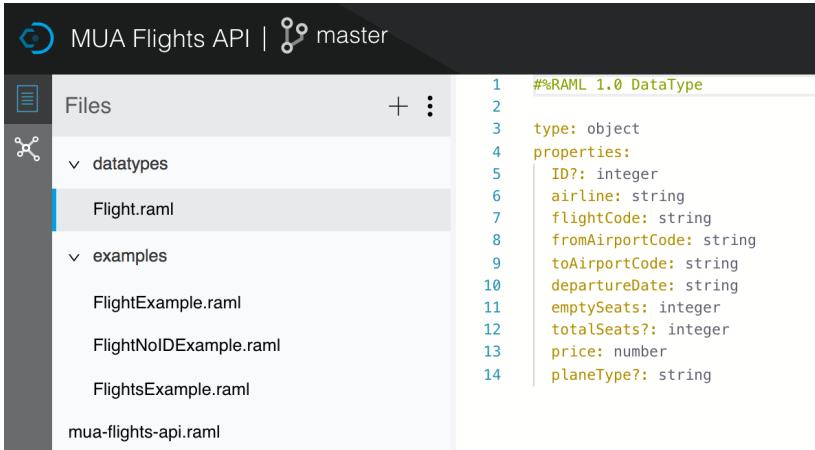
Create a new API in Design Center

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



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

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

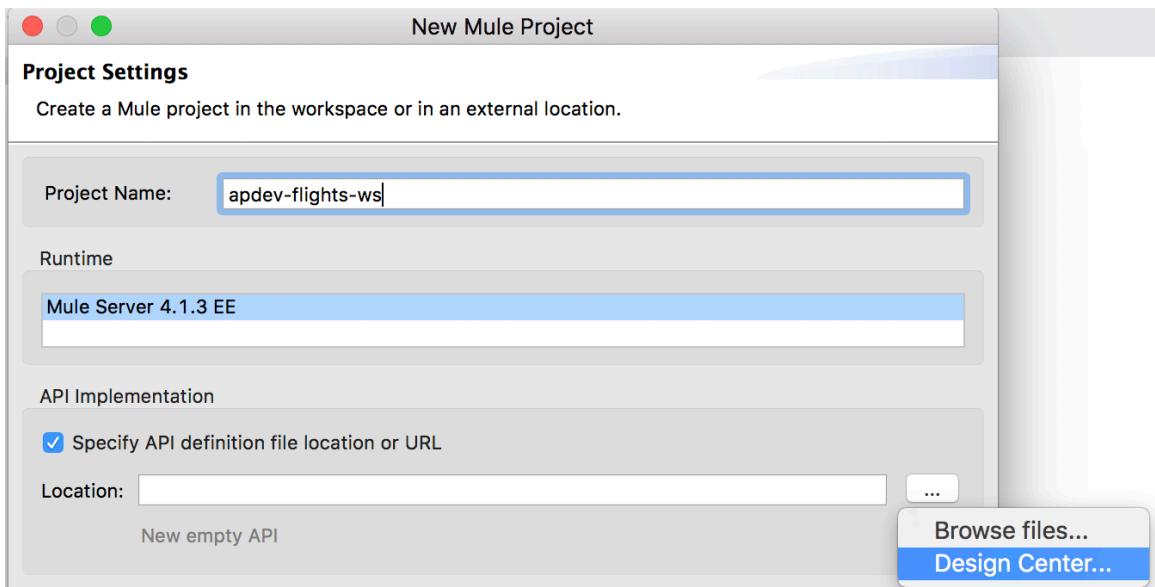


The screenshot shows a file browser interface for the 'MUA Flights API' repository. The left sidebar lists 'Files' and 'datatypes'. Under 'datatypes', 'Flight.raml' is selected and highlighted. The right pane displays the RAML code for the 'Flight' data type:

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

Create a new project to implement this API in Anypoint Studio

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



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

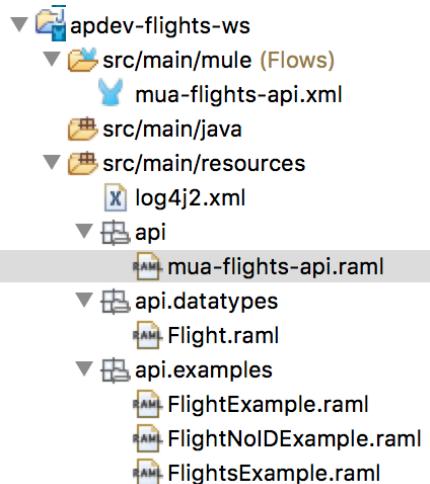
Project name	Branch
American Flights API	master
MUA Flights API	master

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.

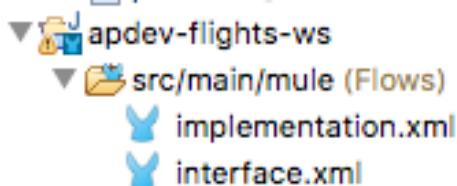
18. Leave the file open.

Review project configuration files

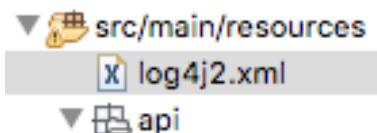
19. Look at the mua-flights-api.xml file that was created; it should have a get:\flights flow and a post:\flights flow.

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

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

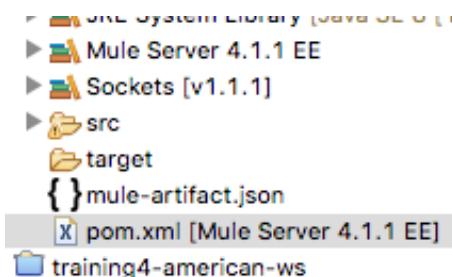


22. Locate log4j2.xml in src/main/resources and open it.



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.

A screenshot of a code editor showing two side-by-side XML files. Both files are titled 'pom.xml'. The left file is for the 'apdev-flights-ws' project and the right file is for the 'apdev-examples' project. The code is identical, showing dependencies for org.mule.modules:mule-apikit-module, org.mule.connectors:mule-vm-connector, com.mulesoft.connectors:mule-salesforce-connector, and mule-plugin. The code is color-coded with syntax highlighting for tags and attributes.

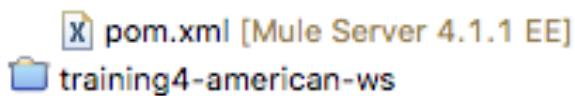
```
<dependency>
<groupId>org.mule.modules</groupId>
<artifactId>mule-apikit-module</artifactId>
<version>1.1.1</version>
<classifier>mule-plugin</classifier>
</dependency>
</dependencies>
<repositories>
<repository>
<id>anypoint-exchange</id>
<name>Anypoint Exchange</name>
<url>https://maven.anypoint.mulesoft.com/</url>

```

```
<dependency>
<groupId>org.mule.connectors</groupId>
<artifactId>mule-vm-connector</artifactId>
<version>1.1.1</version>
<classifier>mule-plugin</classifier>
</dependency>
<dependency>
<groupId>com.mulesoft.connectors</groupId>
<artifactId>mule-salesforce-connector</artifactId>
<version>9.1.0</version>
<classifier>mule-plugin</classifier>
</dependency>
</dependencies>
```

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".

A screenshot of a code editor showing the 'config.yaml' file. The content is:

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

The file is currently empty except for these two lines.

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.

A screenshot of the Mule Studio interface showing the 'Global Configuration Elements' view. It displays a table with one row:

Type	Name	Description	Actions
Configuration properties (Configuration)	Configuration properties		Create Edit

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

HTTP Listener config
Configuration element for a HttpListener.

General Notes

Name: **HTTP_Listener_config**

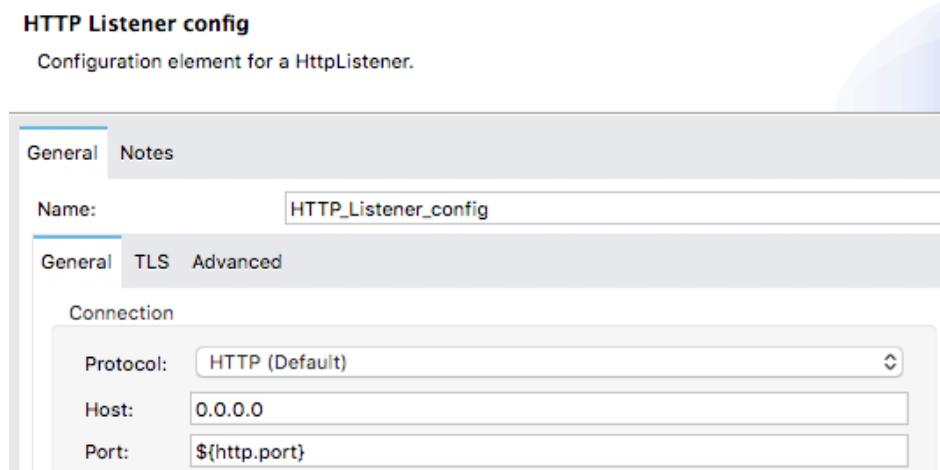
General TLS Advanced

Connection

Protocol: **HTTP (Default)**

Host: **0.0.0.0**

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

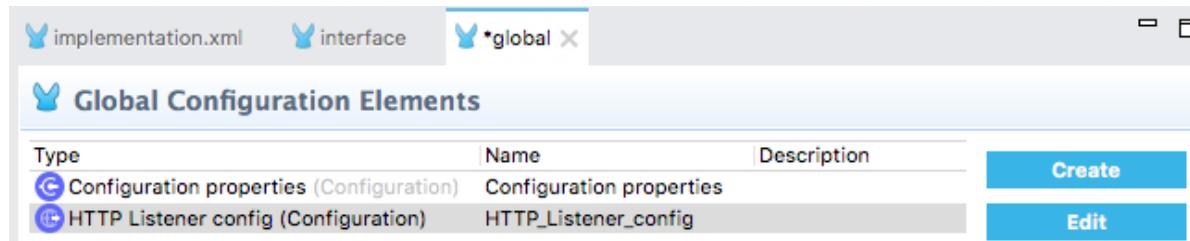


38. Confirm you now have two global elements defined in global.xml.

implementation.xml interface *global X

Global Configuration Elements

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



39. Save and close the file.

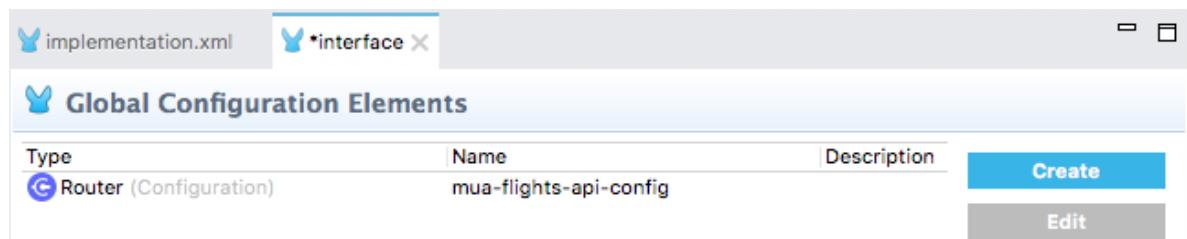
40. Go to the Global Elements view in interface.xml.

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

implementation.xml interface X

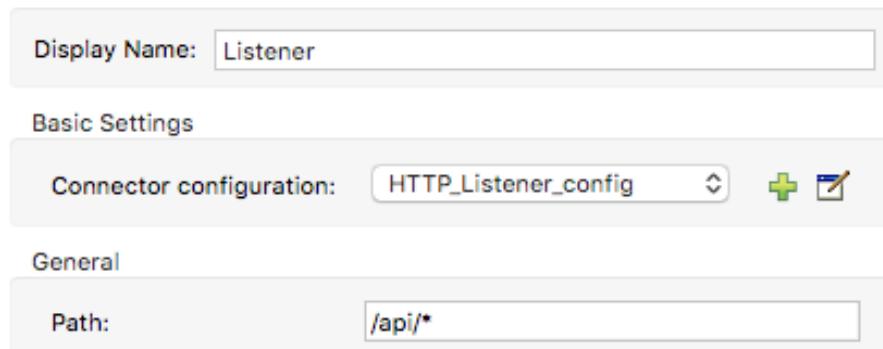
Global Configuration Elements

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



42. Return to the Message Flow view.

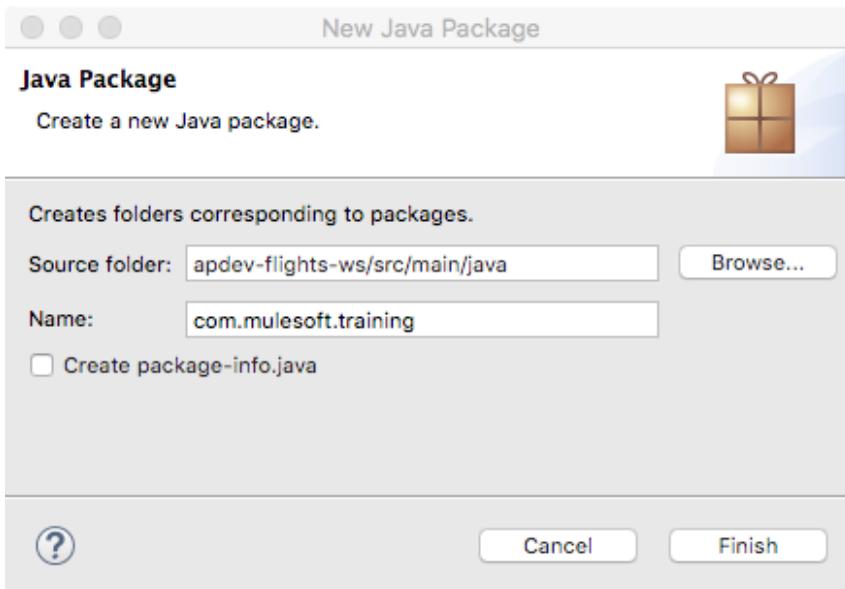
43. Select the HTTP Listener in mua-flights-api-main, and in the Listener properties view set the Connector configuration to HTTP_Listener_config.



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.

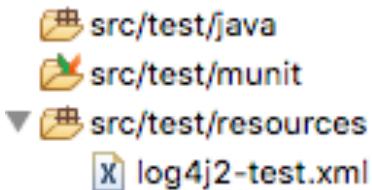
51. Review the code.



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.

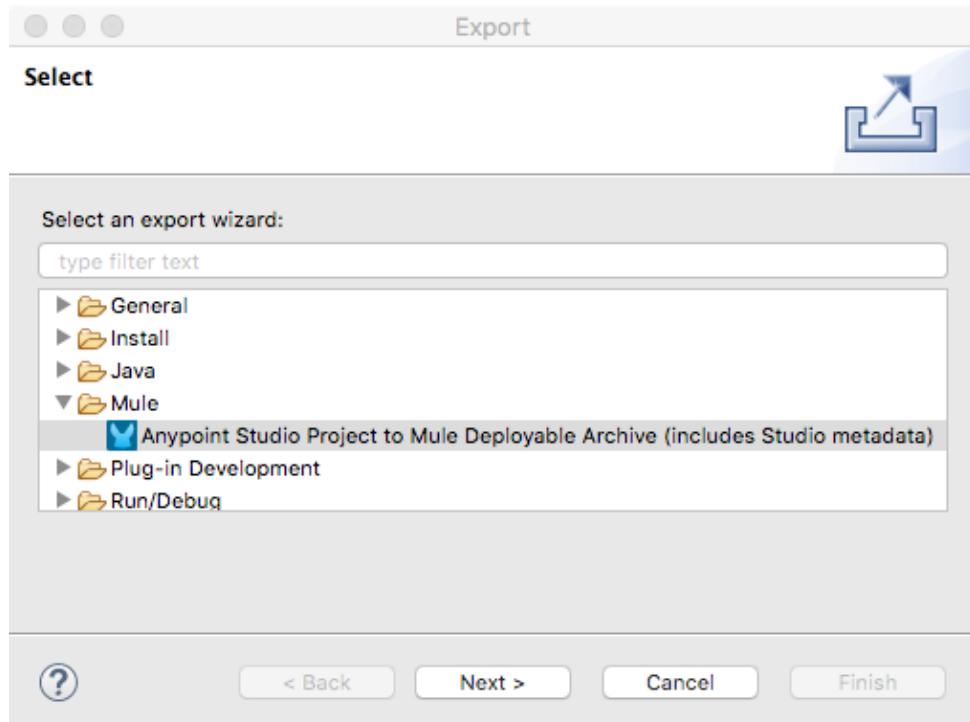
55. Select the three flights and one united-flights files (JSON and XML) and drag them into the src/test/resources folder in the Anypoint Studio apdev-flights-ws project.



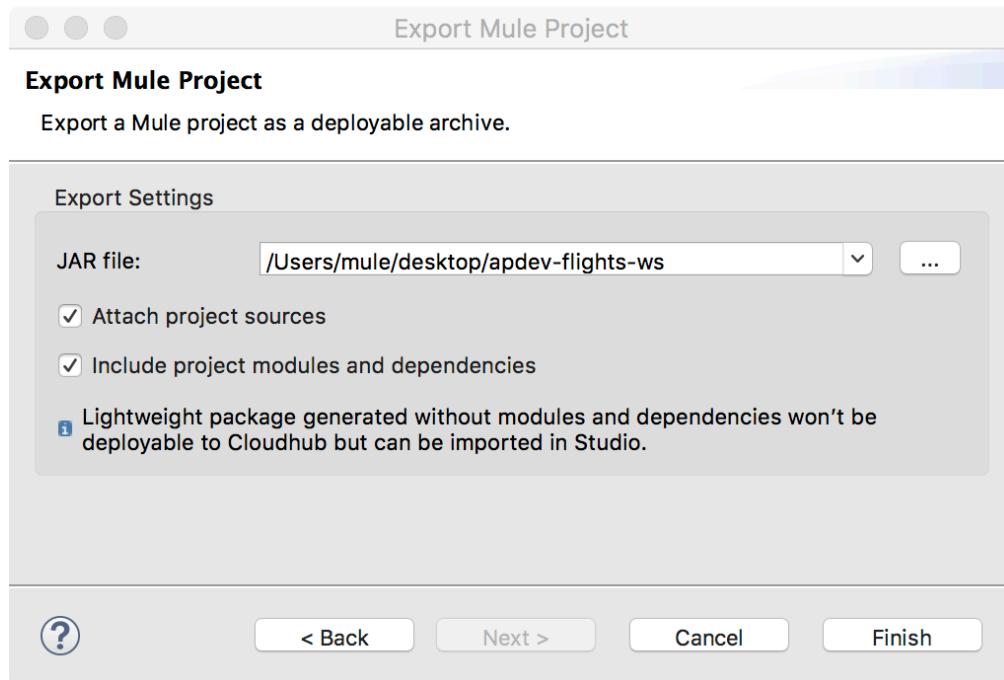
Examine the contents of a deployable archive

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.

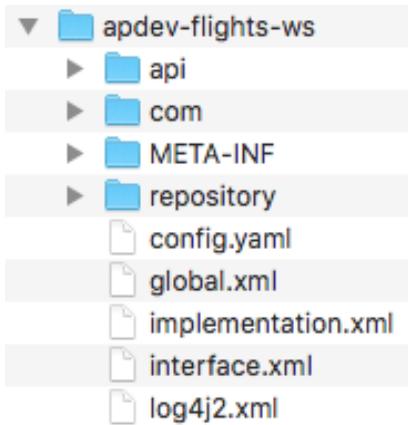


58. In the Export Mule Project dialog box, set the JAR file to a location that you can find and click Finish.



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

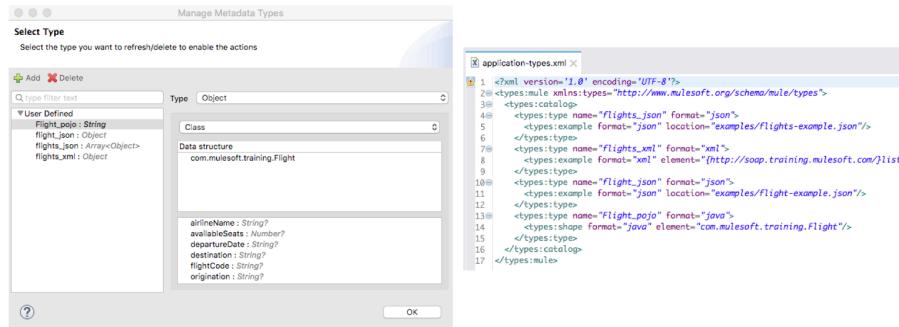
60. Open the resulting apdev-flights-ws folder and examine the contents.



Walkthrough 7-6: Manage metadata for a project

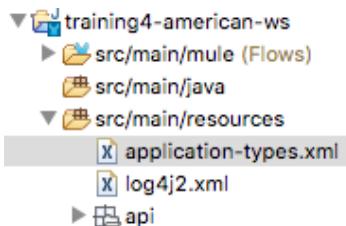
In this walkthrough, you define metadata for the new apdev-flights-ws project. You will:

- 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

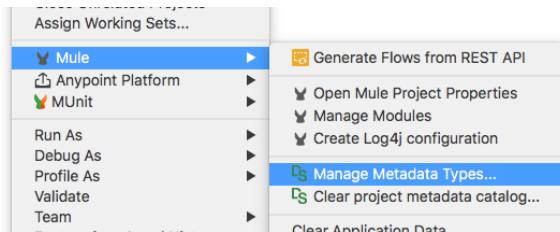
1. Open the training-american-ws project.
2. Locate application-types.xml in src/main/resources.



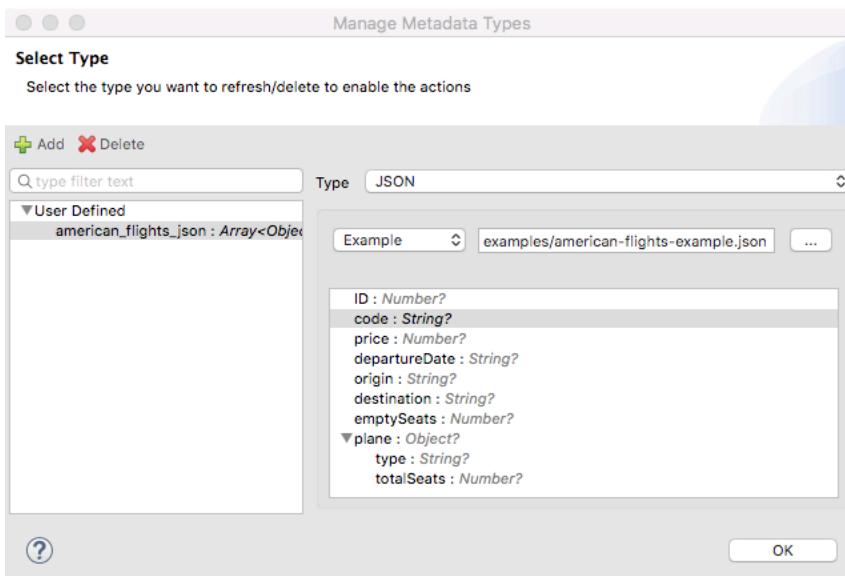
3. Open the file and review the code.

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

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



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



- Click OK.
- Close the project.

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

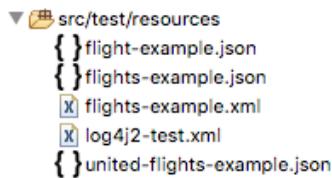
- Return to the apdev-flights-ws project.
- Return to mua-flights-api.raml and review the specified response and request types.

```

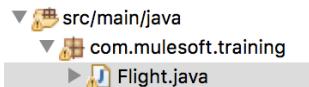
responses:
  200:
    body:
      application/json:
        type: Flight
        example: !include examples/FlightsExample.raml

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

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



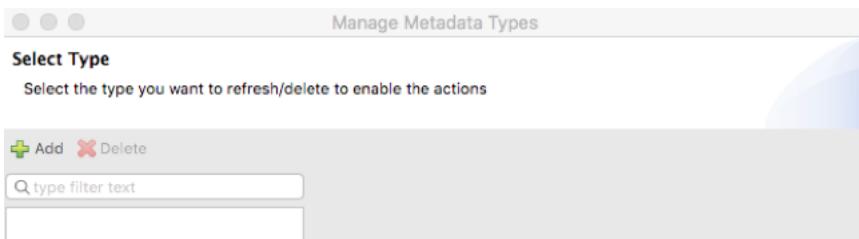
12. Locate the files you added to the src/main/java folder.



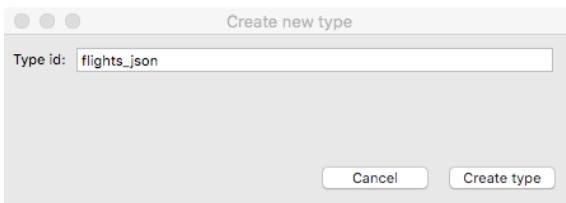
Create flights_json metadata type

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

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



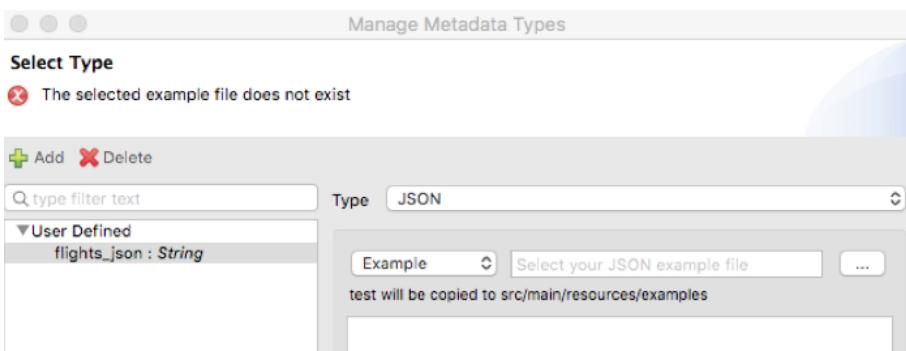
15. In the Create new type dialog box, set the type id to flights_json.



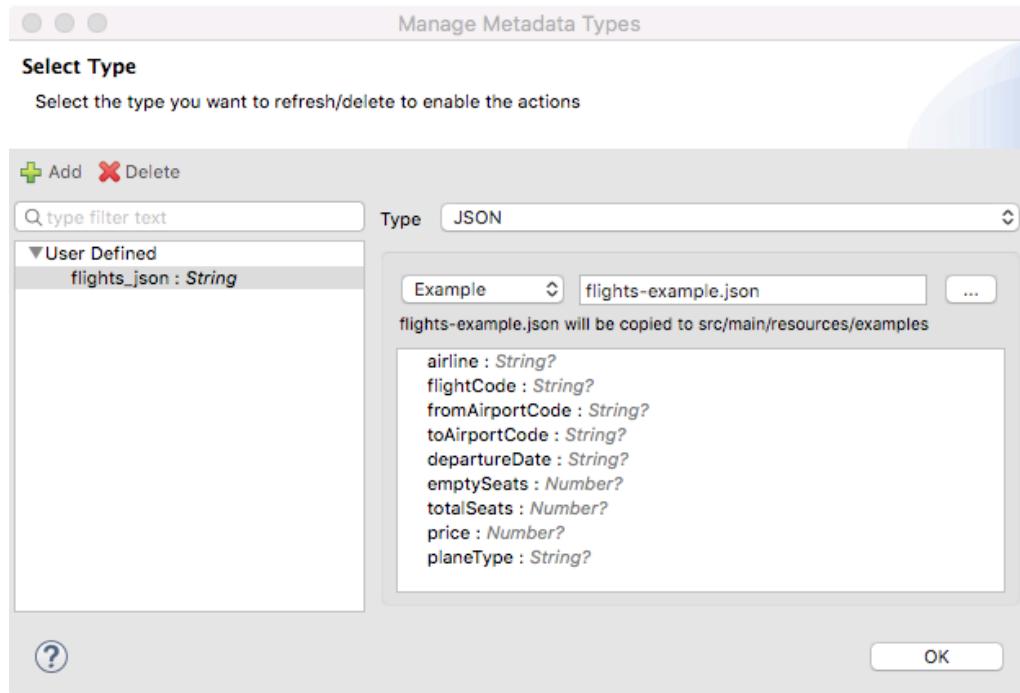
16. Click Create type.

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

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



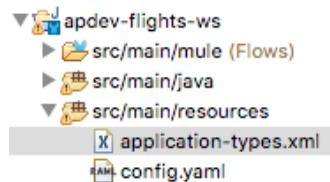
19. Click the browse button and select the flights-example.json file in src/test/resources.



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

Locate the new metadata definition file

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

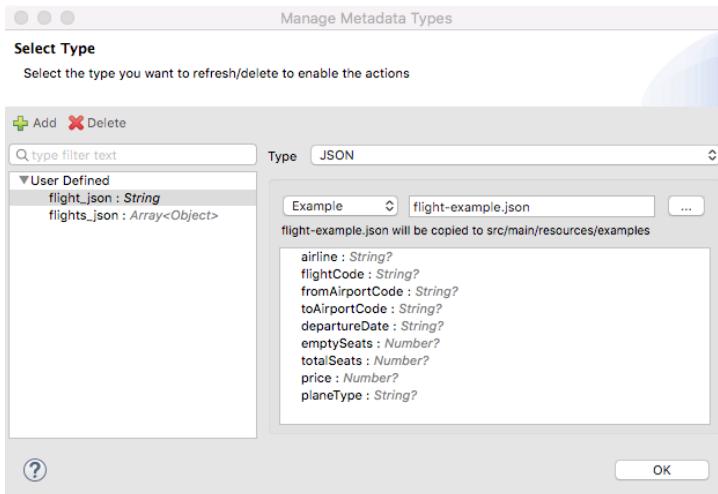


22. Open the file and review the code.

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

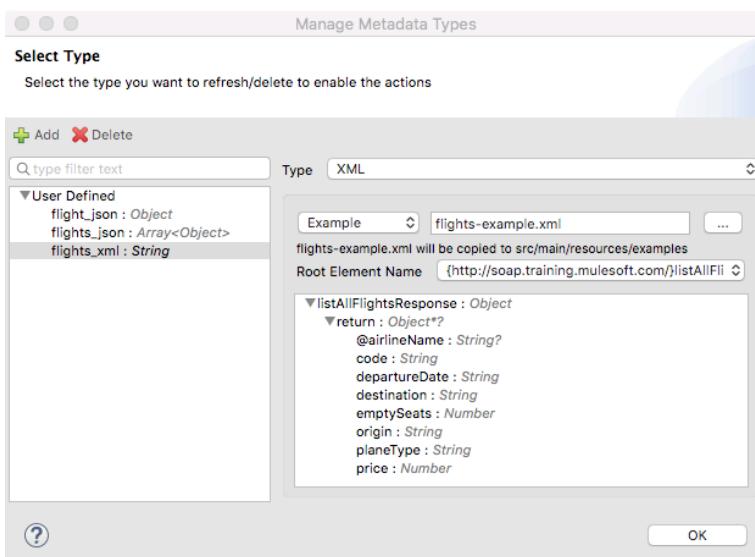
Create flight_json metadata type

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



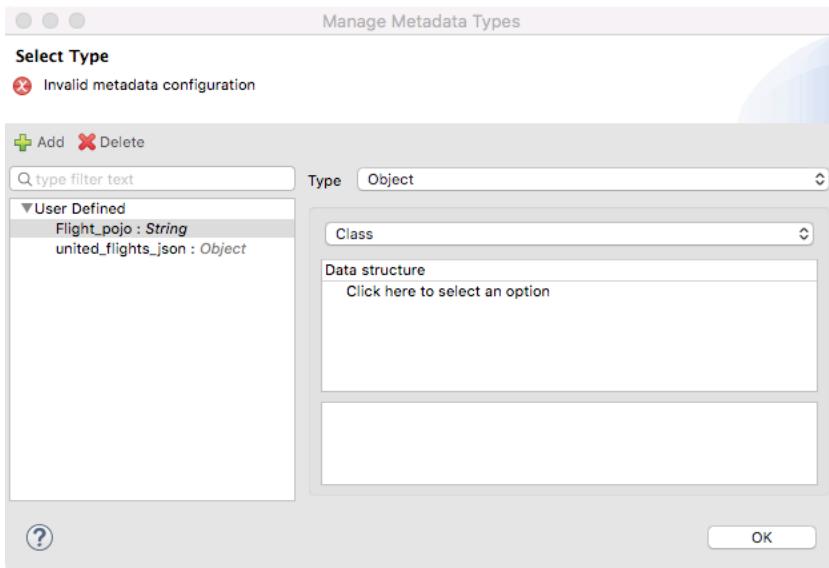
Create flights_xml metadata type

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

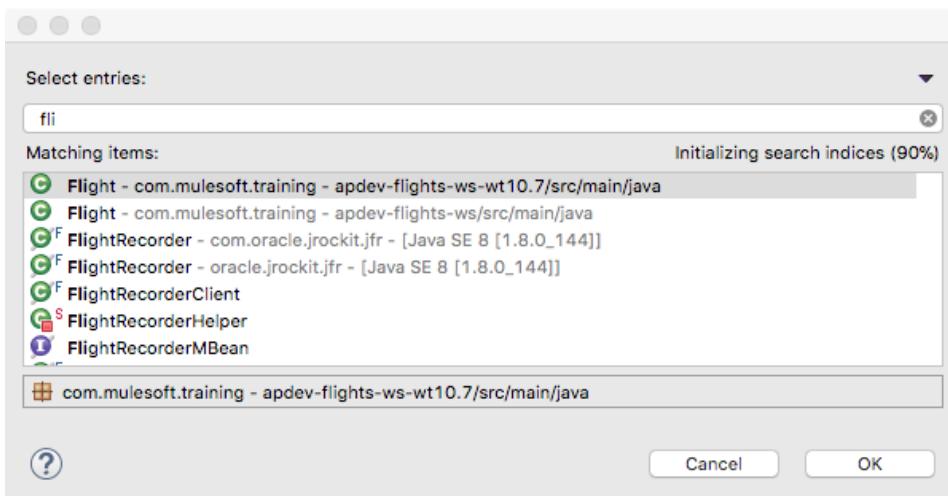


Create Flight metadata type

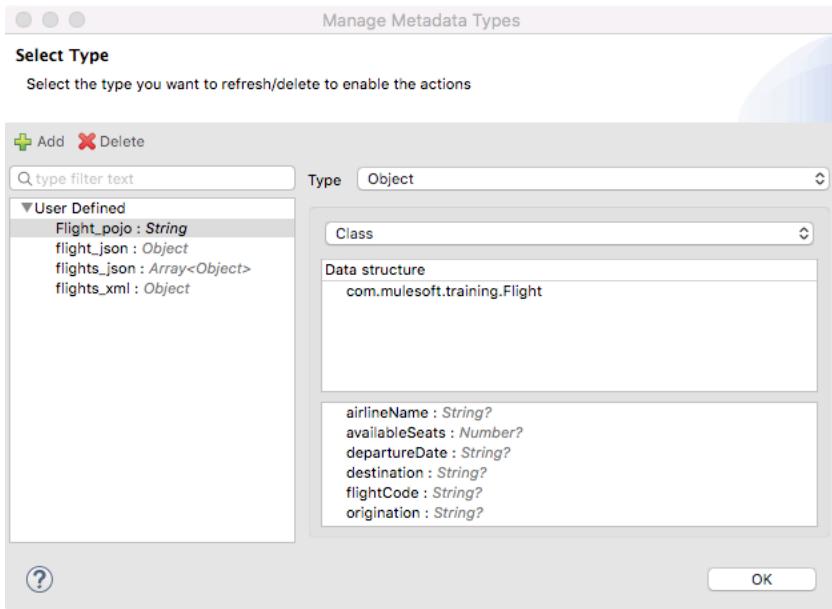
32. In the Select metadata type dialog box, click the Add button.
33. In the Create new type dialog box, set the type id to Flight_pojo and click Create type
34. In the Select metadata type dialog box, set the type to Object.



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



38. Click OK.



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

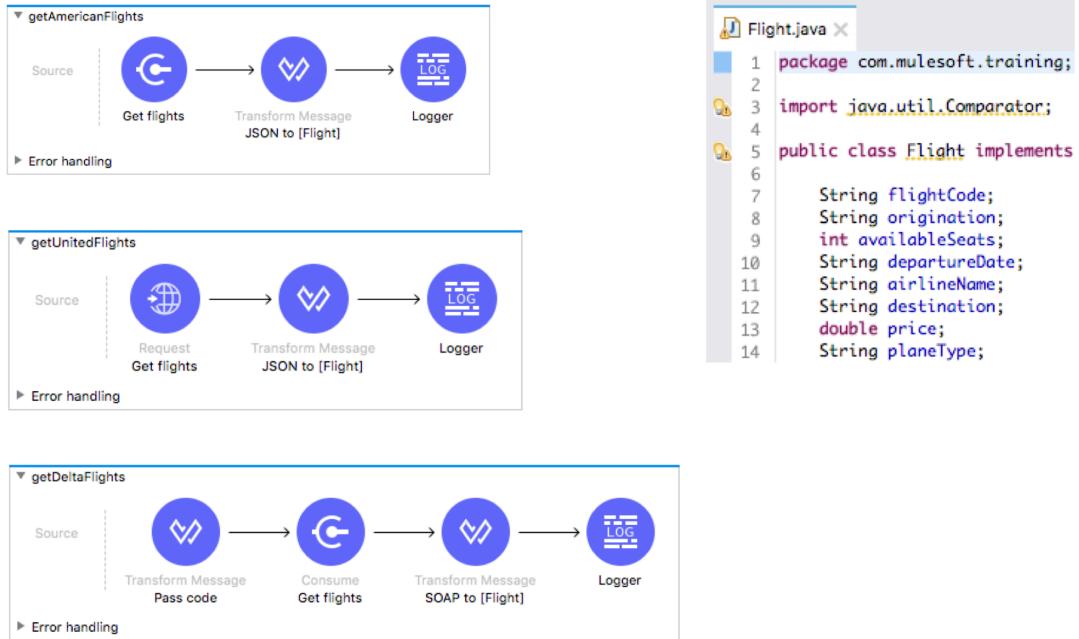
Review the metadata definition file

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

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

41. Close the file.

Module 8: Consuming Web Services



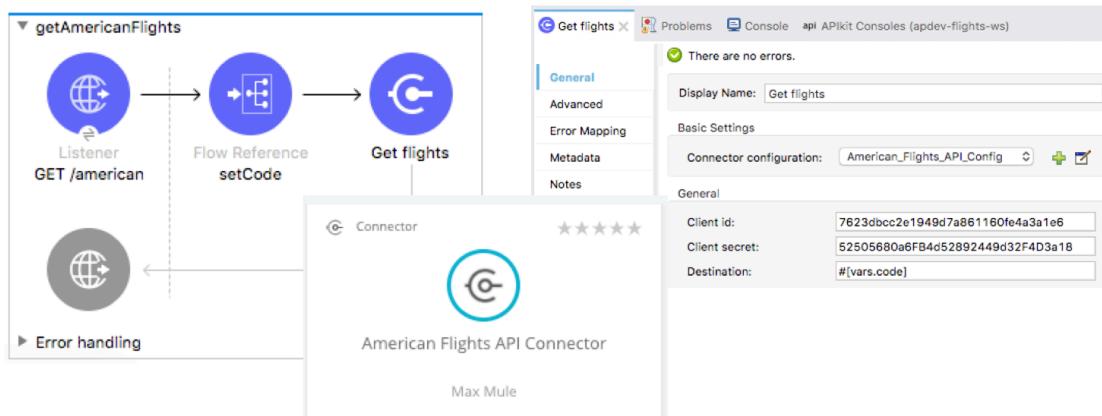
At the end of this module, you should be able to:

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

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

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

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



Review the auto-generated REST connector in Exchange

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

The screenshot shows the Exchange interface in Anypoint Platform. The left sidebar shows navigation options like Assets, Organizations, MuleSoft, Training, My applications, and Public portal. The main area is titled "Assets" and shows two items:

- A Connector named "American Flights API Connector" with a rating of 5 stars and created by "Max Mule".
- A REST API named "American Flights API" with a rating of 5 stars and created by "Max Mule".

3. Select the connector and review its information.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a sidebar with a navigation menu. The main content area is titled "American Flights API Connector". It features a circular icon with a gear and a star rating of 0 reviews. Below the title is a placeholder text: "This page currently doesn't contain a description. Click Edit to add text, images, videos, code blocks, etc...". There are "Share", "Download", "Edit", and more options buttons at the top right. Under "Overview", it says "Type: Connector", "Created by: Max Mule", and "Published on: Apr 18, 2018". A "Versions" section lists three versions: 1.0.2 (Runtime version 3.6.0), 1.0.1 (Runtime version 3.6.0), and 1.0.0 (Runtime version 3.6.0).

Make a request to the web service

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

The screenshot shows the Advanced REST Client interface. At the top, it has a "Method: GET" dropdown, a "Request URL: http://training4-american-api-mule.cloudbhub.io/flights", a "SEND" button, and a more options button. Below that is a "Parameters" section with an "Headers" tab selected. It shows two header entries: "client_id" with value "7623dbcc2e1949d7a861160fe4a3a1e6" and "client_secret" with value "52505680a6FB4d52892449d32F4D3a18". There are "ADD HEADER" and "A" buttons. The response section shows a green "200 OK" status with a "2125.58 ms" latency. The response body is a JSON array of flight details:

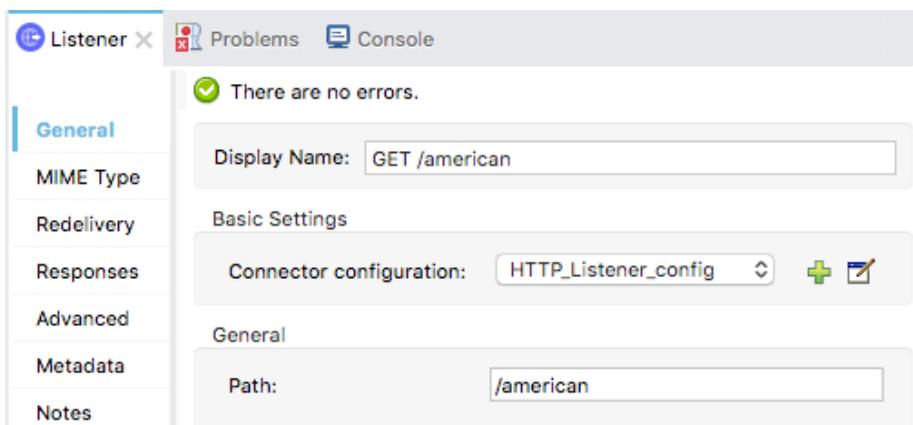
```

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

```

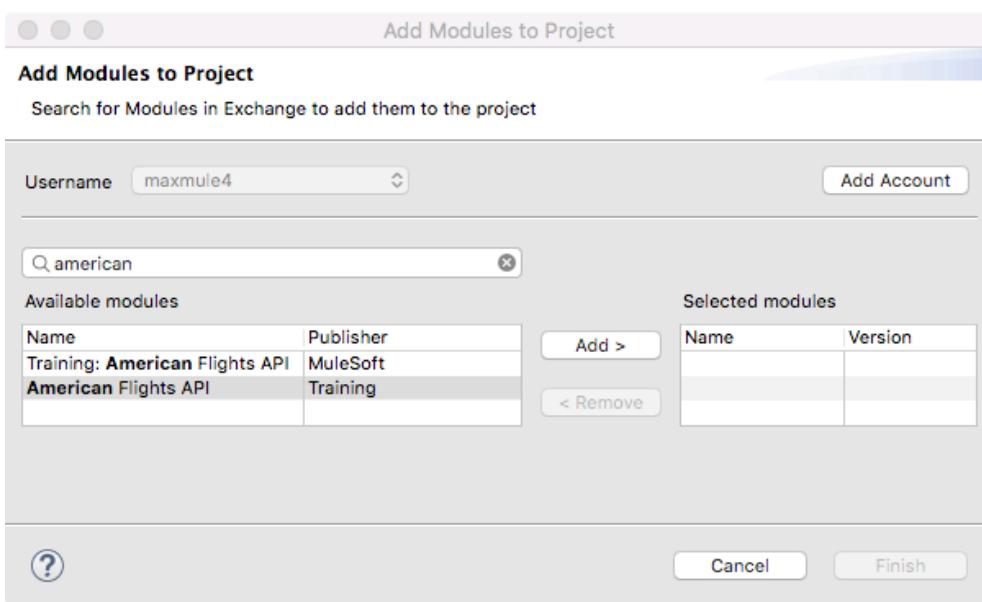
Add a new flow with an HTTP Listener

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



Add the American Flights API module to Anypoint Studio

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



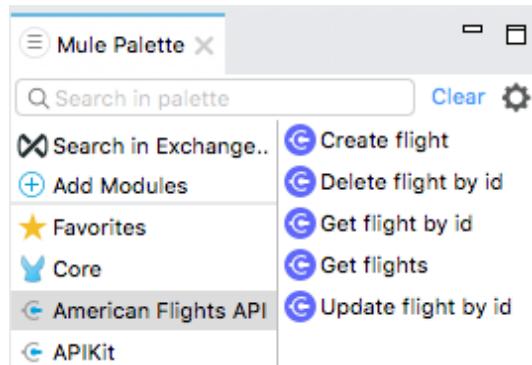
17. Select your American Flights API (**not** the Training: American Flights API) and click Add.

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

18. Click Finish.

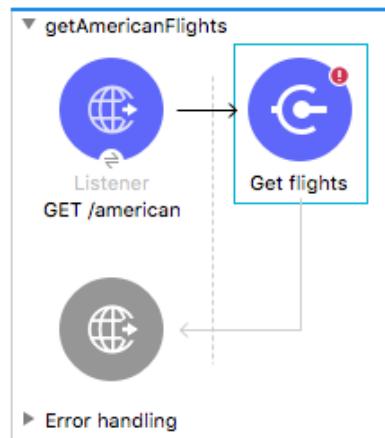
19. Wait for the module to be downloaded.

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

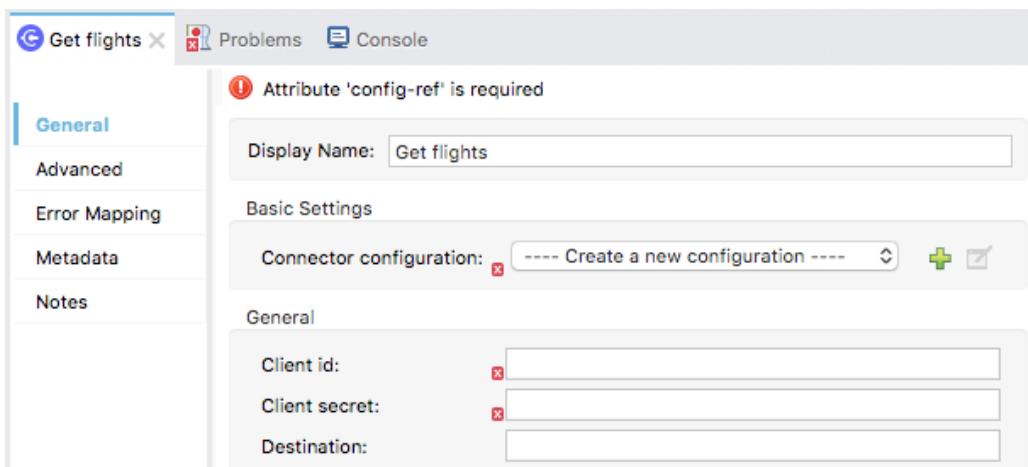


Add a Get all flights operation

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

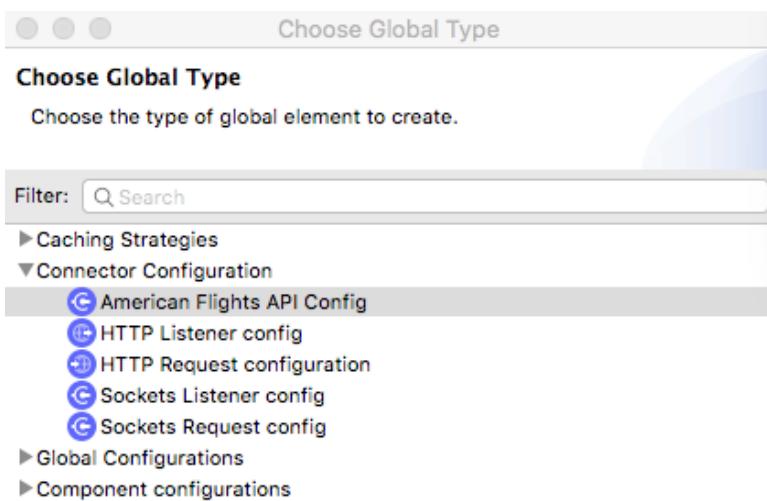


22. Select the Get all flights operation in the canvas and in its properties view, see that you need to need to create a new configuration and enter a client_id and client_secret.



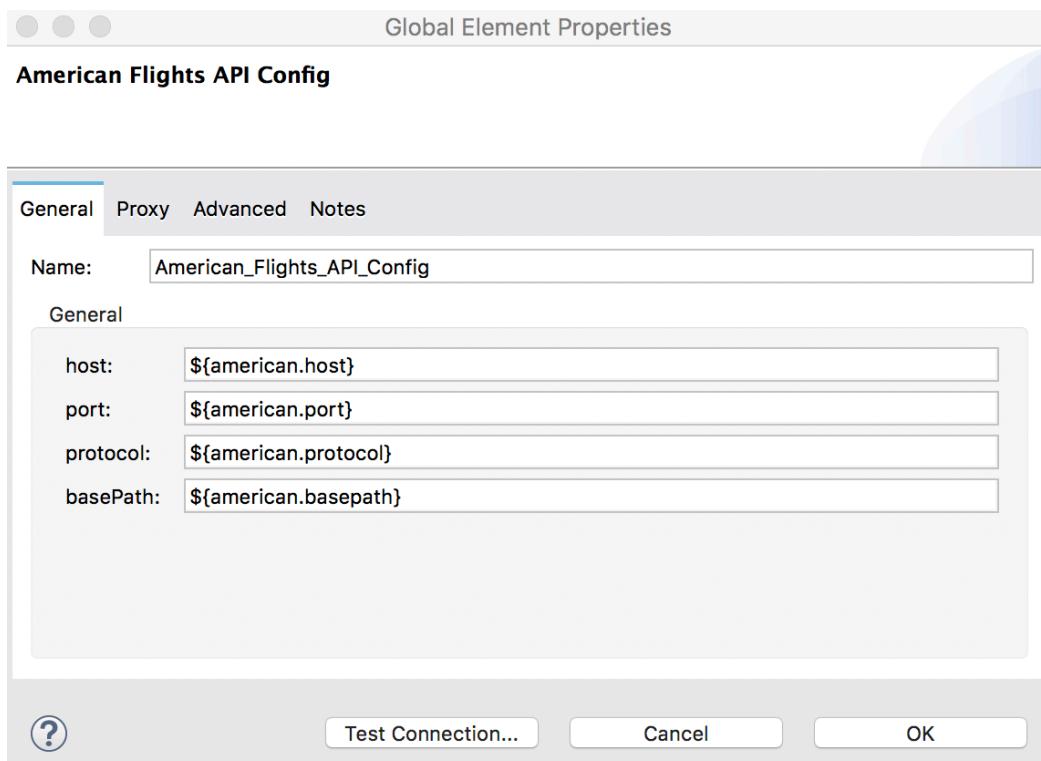
Configure the American Flights API connector

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



26. In the Global Element Properties dialog box, set each field to a corresponding property placeholder (that you will define and set next):

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



27. Click OK.

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

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

```
implementation *config.yaml
1 http:
2   port: "8081"
3
4 american:
5   host: "training4-american-api-{lastname}.cloudbus.io"
6   port: "80"
7   basePath: "/"
8   protocol: "HTTP"
9   client_id:
10  client_secret:
```

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

Configure the Get flights operation

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

The screenshot shows the 'Get flights' operation configuration in Mule Studio. The 'General' tab is selected. The 'Display Name' is 'Get flights'. The 'Connector configuration' is set to 'American_Flights_API_Config'. Under 'General', the 'Client id' is set to \${american.client_id} and the 'Client secret' is set to \${american.client_secret}. The 'Destination' field is empty.

Review metadata associated with the operation

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

The screenshot shows the DataSense Explorer with the 'Output' tab selected. The 'Payload' section is expanded, showing an array of objects with the following properties:

- plane : Object?
- code : String?
- price : Number?
- origin : String?
- destination : String?
- ID : Number?
- departureDate : String?
- emptySeats : Number?

Test the application

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

The screenshot shows the Advanced REST Client interface. At the top, there's a header with 'Method' set to 'GET' and 'Request URL' set to 'http://localhost:8081/american'. Below the header is a large green button labeled 'SEND'. To the right of the 'SEND' button is a vertical ellipsis menu icon. Underneath the header, there's a section titled 'Parameters' with a dropdown arrow. The main content area displays a green box with '200 OK' and '2691.91 ms' next to it. To the right of this box is a 'DETAILS' button with a dropdown arrow. Below this, there are several icons: a clipboard, a download arrow, a copy icon, a refresh/circular arrow icon, and a list icon. The response body is shown as a JSON array of 11 flight objects. One object is expanded to show its details:

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

Review the specification for the API you are building

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

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

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

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

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

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

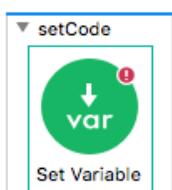
Create a variable to set the destination airport code

46. Return to implementation.xml.

47. Drag a Sub Flow scope from the Mule Palette and drop it at the top of the canvas.

48. Change the name of the flow to setCode.

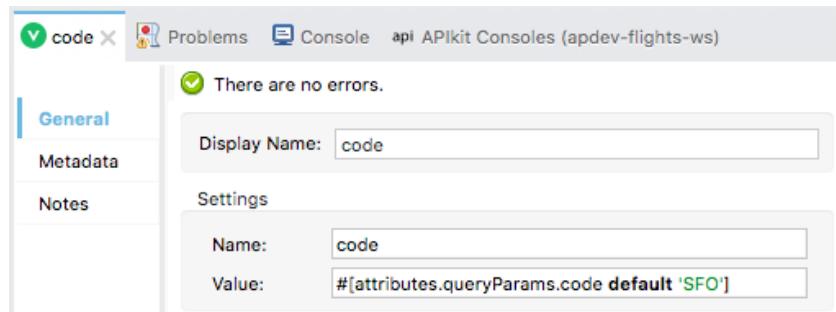
49. Drag a Set Variable transformer from the Mule Palette and drop it in the setCode subflow.



50. In the Set Variable properties view, set the display name and name to code.

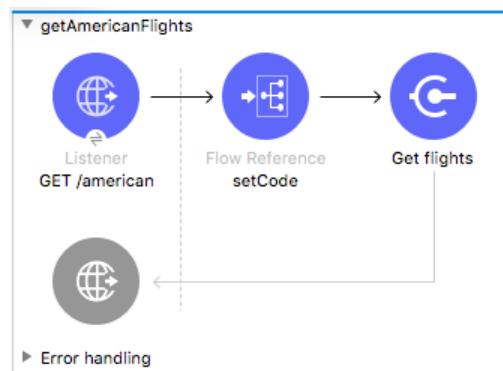
51. Set the value to a query parameter called code and give it a default value of SFO if no query parameter is passed to the flow.

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



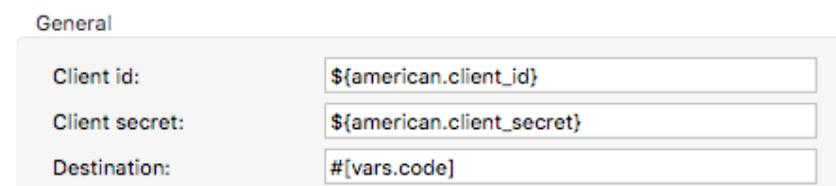
Dynamically set a query parameter for the web service call

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

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

SEND

Parameters

200 OK 1743.09 ms DETAILS

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

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

Method Request URL
GET http://localhost:8081/american?code=LAX

SEND

Parameters

200 OK 1204.27 ms DETAILS

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

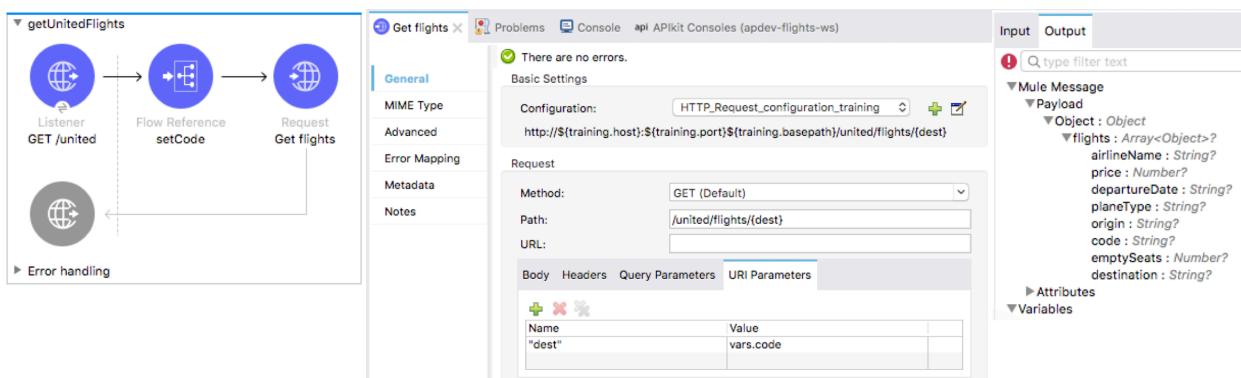
58. Examine the data structure of the JSON response.

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.

Method Request URL
GET http://mu.learn.mulesoft.com/united/flights

Parameters

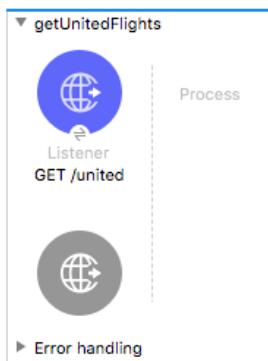
200 OK 278.45 ms DETAILS

```
{  
  "flights": [Array[12],  
    0: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    1: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    2: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    3: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    4: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    5: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    6: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    7: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    8: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    9: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    10: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    },  
    11: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "airlineName": "United",  
      "planeType": "Boeing 737",  
      "emptySeats": 0  
    }]
```

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

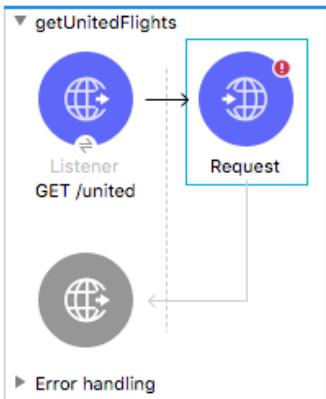
Add a new flow with an HTTP Listener operation

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



Add an HTTP Request operation

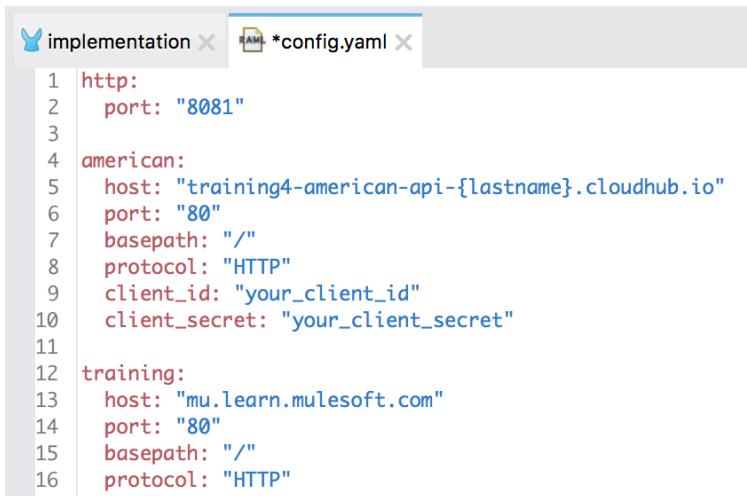
- Drag out an HTTP Request from the Mule Palette and drop it into the process section of getUnitedFlights.



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

Create an HTTP Request configuration

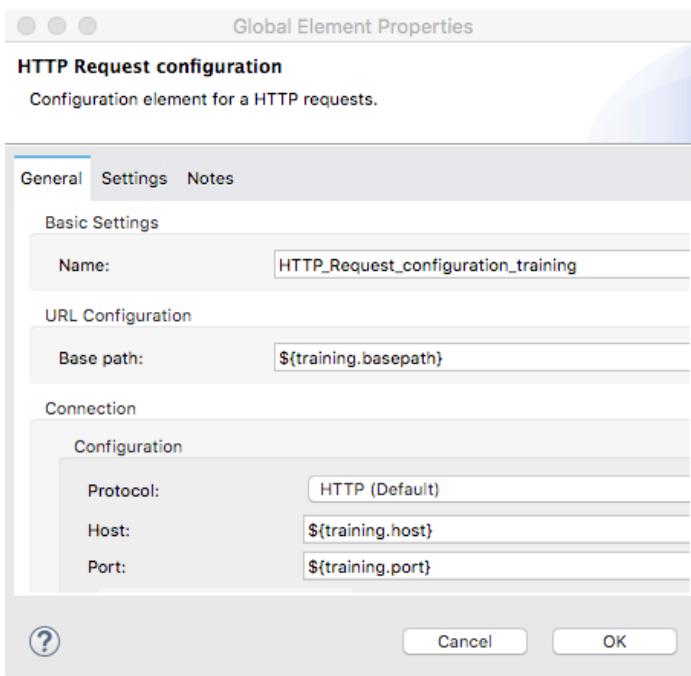
17. Return to the course snippets.txt file and copy the text for the Training web service properties.
18. Return to config.yaml and paste the code at the end of the file.



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

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

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



21. Click OK.

Configure the HTTP Request operation

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

The screenshot shows the Mule Studio interface with the 'GET flights' request operation selected. The 'General' tab is active, displaying the following configuration:

- Display Name:** GET flights
- Configuration:** HTTP_Request_configuration_training (with a '+' icon to add more)
- Path:** http://\${training.host}:\${training.port}\${training.basepath}/united/flights
- Request Method:** GET (Default)
- Request Path:** /united/flights

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

Review metadata associated with the United Get flights operation response

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

The screenshot shows the DataSense Explorer with the 'Output' tab selected. The payload section is expanded, showing the following structure:

- Mule Message
- Payload
 - Any : Any
- Attributes
 - HttpResponseAttributes : Object
- Variables

Test the application

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

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

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

Parameters:

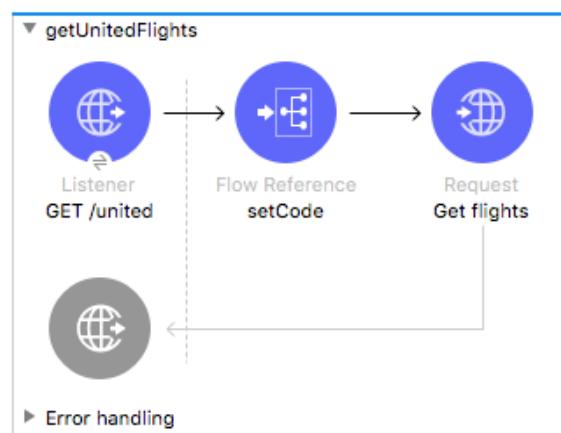
200 OK 446.93 ms DETAILS

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

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

Add a URI parameter to the web service call

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



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

36. In the Request section, change the path to /united/flights/{dest}.

The screenshot shows the 'Get flights' request properties in the Mule Studio interface. The 'Path' field in the 'Request' section is set to '/united/flights/{dest}'. The 'Body' tab contains the placeholder '#[payload]'. The 'URI Parameters' tab is selected.

37. Select the URI Parameters tab.

38. Click the Add button.

39. Set the name to dest.

40. Set the value to the value of the code variable.

`vars.code`

The screenshot shows the 'Get flights' request properties in the Mule Studio interface. A new URI parameter 'dest' has been added, mapped to the variable 'vars.code'. The 'URI Parameters' tab is selected.

Name	Value
"dest"	vars.code

Test the application

41. Save the file to redeploy the application.
42. In Advanced REST Client, make the same request; you should now only get flights to CLE.

Method Request URL
GET <http://localhost:8081/united?code=CLE>

Parameters

200 OK 513.70 ms DETAILS

{
 "flights": [Array[2]]
 -0: {
 "code": "ER9fje",
 "price": 845,
 "origin": "MUA",
 "destination": "CLE",
 "departureDate": "2015/07/11"
 }
}

43. Remove the code parameter and make the request; you should now only get results for SFO.

Note: You will transform the data to the format specified by the mua-flights-api in a later walkthrough.

Add metadata for the United Get flights operation response

44. Return to Anypoint Studio.
45. Navigate to the properties view for the Get flights operation in getUnitedFlights.
46. Select the Output tab in the DataSense Explorer and expand Payload; you should see no metadata for the payload.

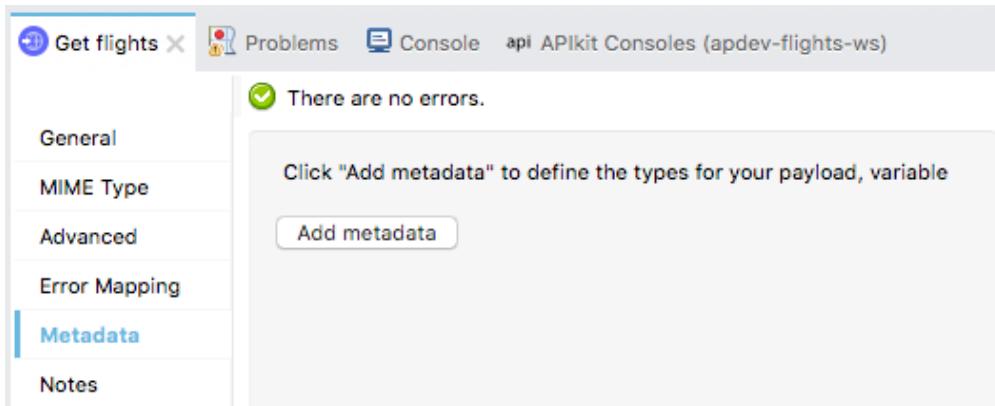
Input Output

Q type filter text

▼ Mule Message
 ▼ Payload
 Any : Any
 ▼ Attributes
 ► HttpResponseAttributes : Object
▼ Variables
 ▼ code
 Nothing : Nothing

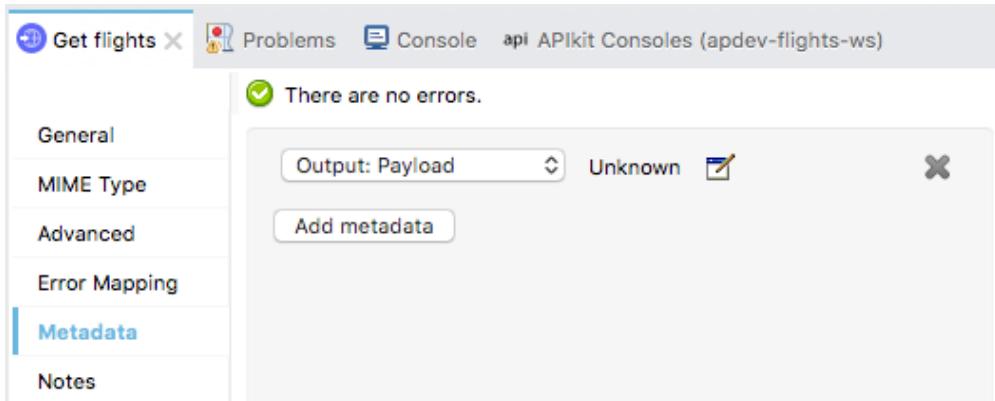
47. In the Get flights properties view, select the Metadata tab.

48. Click the Add metadata button.



49. Change the drop-down menu to Output: Payload.

50. Click the Edit button.



51. In the Select metadata type dialog box, click the Add button.

52. In the Create new type dialog box, set the type id to united_flights_json.

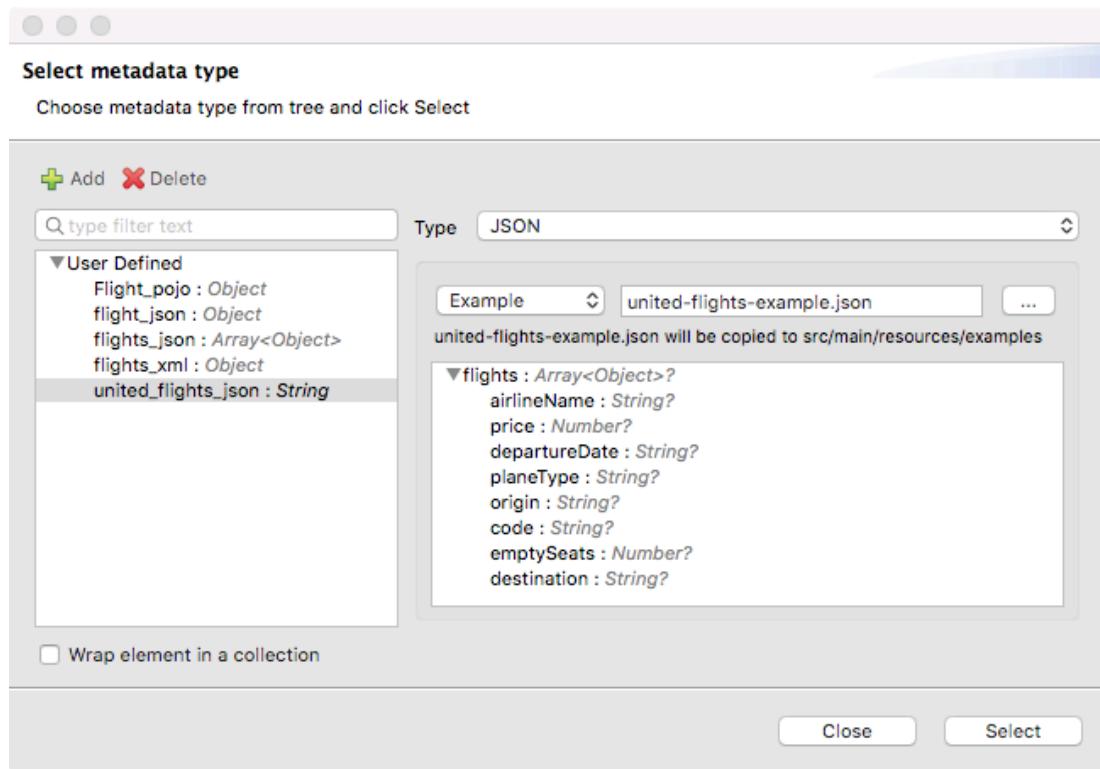
53. Click Create type.

54. In the Select metadata type dialog box, set the type to JSON.

55. Change the Schema selection to Example.

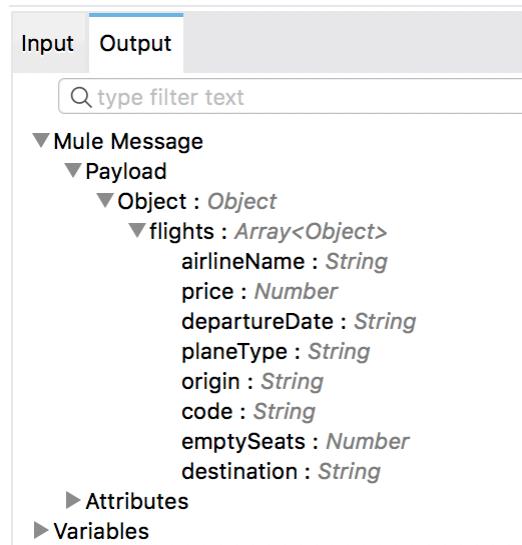
56. Click the browse button and navigate to the project's src/test/resources folder.

57. Select `united_flights-example.json` and click Open; you should see the example data for the metadata type.



58. Click Select.

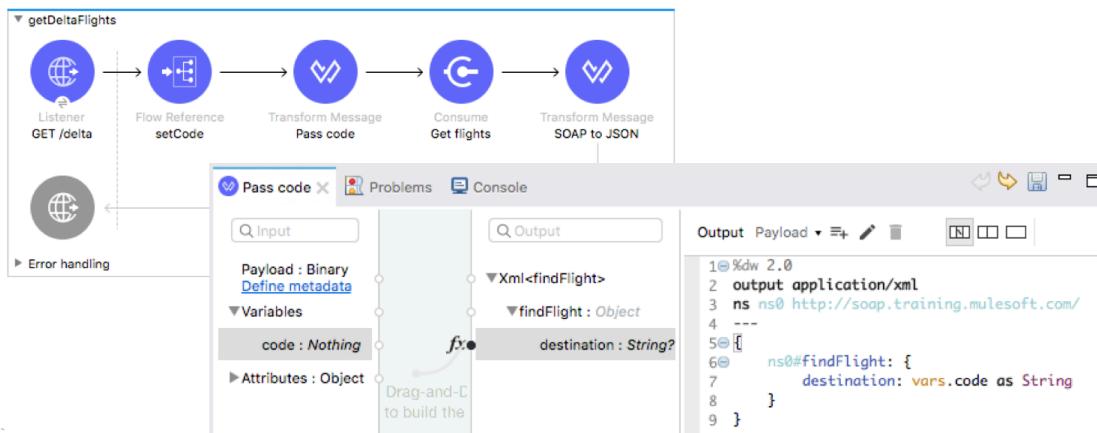
59. In the DataSense Explorer, select the Output tab and expand Payload; you should now see the structure for the payload.



Walkthrough 8-3: Consume a SOAP web service

In this walkthrough, you consume a Delta SOAP web service. You will:

- Create a new flow to call the Delta SOAP web service.
- Use a Web Service Consumer connector to consume a SOAP web service.
- Use the Transform Message component to pass arguments to a SOAP web service.



Browse the WSDL

1. Return to the course snippets.txt file and copy the Delta SOAP web service WSDL.
2. In Advanced REST Client, return to the third tab, the one with the learn.mulesoft request.
3. Paste the URL and send the request; you should see the web service WSDL returned.
4. Browse the WSDL; you should find references to operations listAllFlights and findFlight.

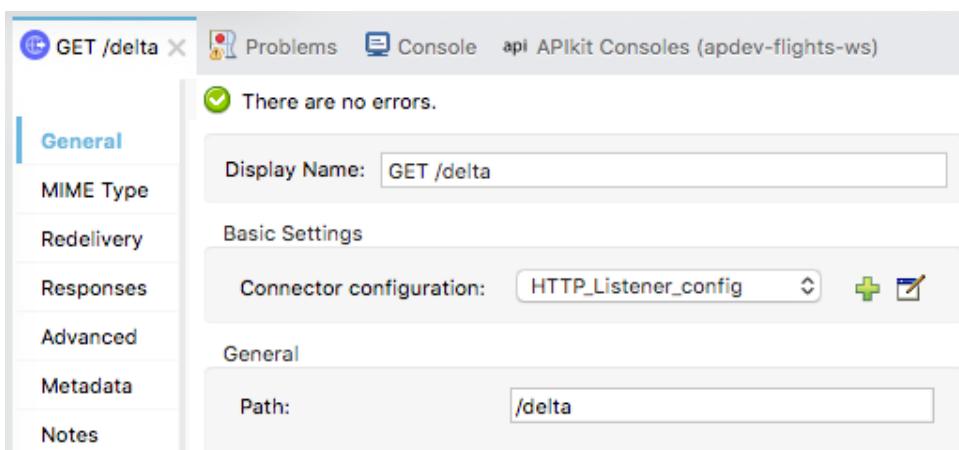
```
<?xml version='1.0' encoding='UTF-8' ?>
^ <wsdl:definitions name="TicketServiceService"
targetNamespace="http://soap.training.mulesoft.com/">
<wsdl:types>
^ <xsschema elementFormDefault="unqualified"
targetNamespace="http://soap.training.mulesoft.com/" version="1.0">
<xss:element name="findFlight" type="tns:findFlight" />
<xss:element name="findFlightResponse"
type="tns:findFlightResponse" />
<xss:element name="listAllFlights" type="tns:listAllFlights" />
<xss:element name="listAllFlightsResponse"
```

Create a new flow with an HTTP Listener connector endpoint

5. Return to Anypoint Studio.
6. Drag out another HTTP Listener from the Mule Palette and drop it in the canvas after the existing flows.
7. Rename the flow to getDeltaFlights.



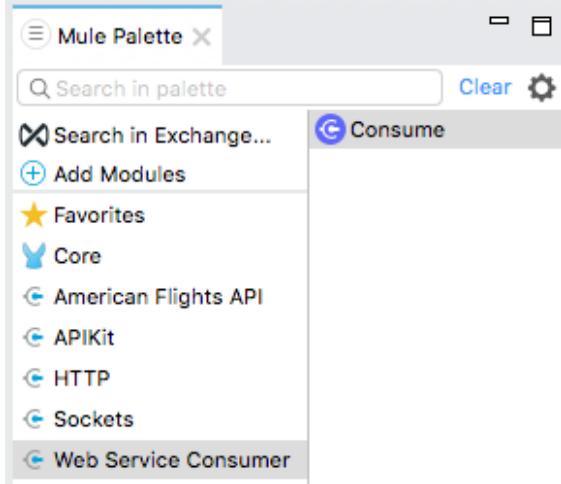
8. In the Listener properties view, set the display name to GET /delta.
9. Set the connector configuration to the existing HTTP_Listener_config.
10. Set the path to /delta and the allowed methods to GET.



Add the Web Service Consumer module to the project

11. In the Mule Palette, select Add Modules.

12. Select the Web Service Consumer connector in the right side of the Mule Palette and drag and drop it into the left side.
13. If you get a Select module version dialog box, select the latest version and click Add.



Configure the Web Service Consumer connector

14. Return to the course snippets.txt file and copy the text for the Delta web service properties.
15. Return to config.yaml in src/main/resources and paste the code at the end of the file.

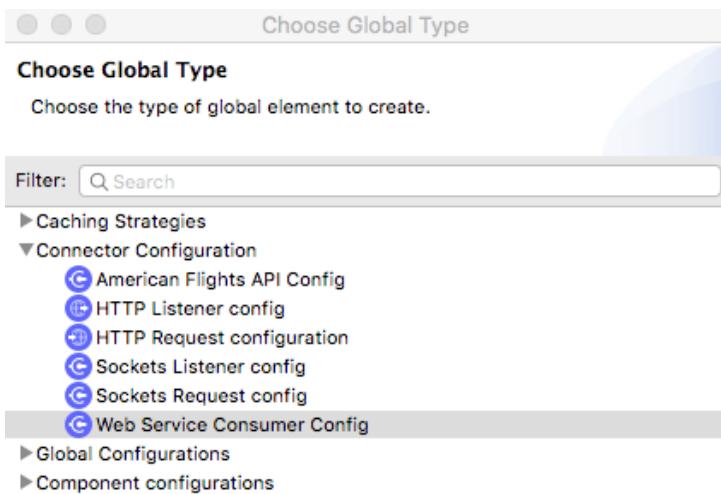
```

*implementation *config.yaml global
1 http:
2   port: "8081"
3
4 american:
5   host: "training4-american-api-[lastname].cloudbhub.io"
6   port: "80"
7   basepath: "/"
8   protocol: "HTTP"
9   client_id: "your_client_id"
10  client_secret: "your_client_secret"
11
12 training:
13   host: "mu.learn.mulesoft.com"
14   port: "80"
15   basepath: "/"
16   protocol: "HTTP"
17
18 delta:
19   wsdl: "http://mu.learn.mulesoft.com/delta?wsdl"
20   service: "TicketServiceService"
21   port: "TicketServicePort"

```

16. Save the file.
17. Return to global.xml.
18. Click Create.

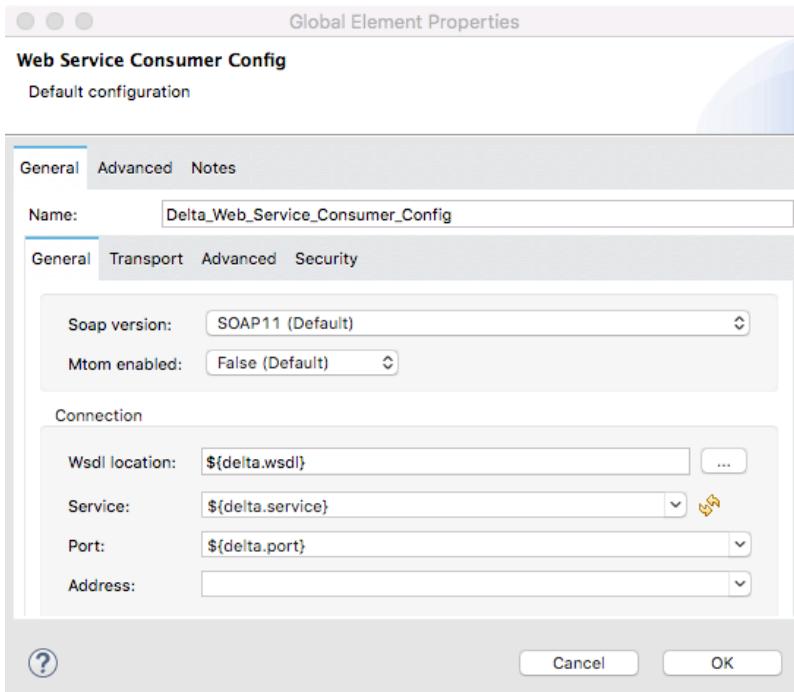
19. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer Config and click OK.



20. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer_Config.

21. Set the

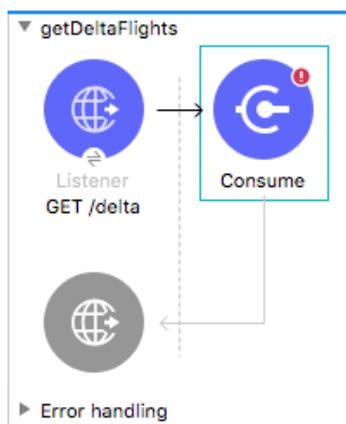
- Wsdl location: \${delta.wsdl}
- Service: \${delta.service}
- Port: \${delta.port}



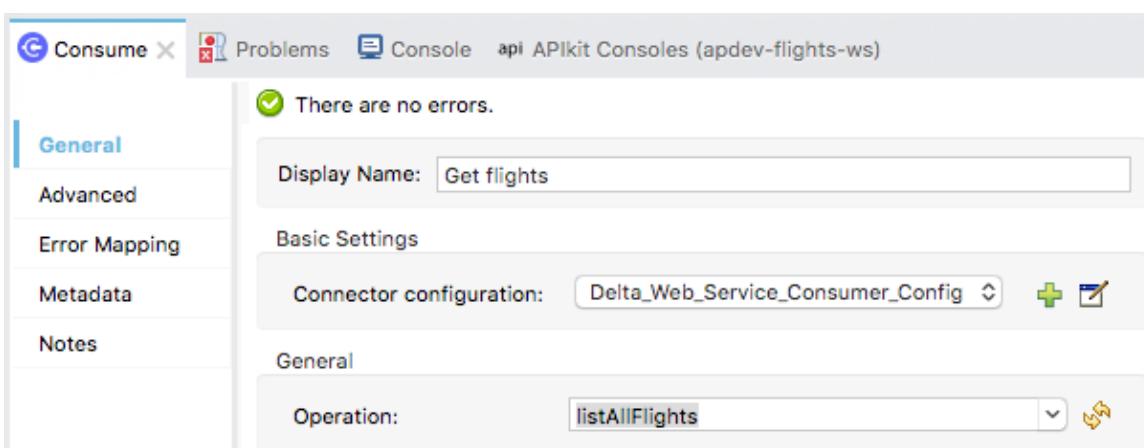
22. Click OK.

Add and configure a Consume operation

23. Return to implementation.xml.
24. Locate the Consume operation for the Web Service Consumer connector in the Mule Palette and drag and drop it in the process section of getDeltaFilghts.



25. In the Consume properties view, change its display name to Get flights.
26. Set the connector configuration to the existing Delta_Web_Service_Consumer_Config.
27. Click the operation drop-down menu button; you should see all of the web service operations listed.
28. Select the listAllFlights operation.



Review metadata associated with the United Get flights operation response

29. Select the Output tab in the DataSense Explorer and expand Payload; you should see payload metadata but with no detailed structure.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. At the top, there are tabs for 'Input' and 'Output'. Below them is a search bar with a warning icon and a placeholder 'type filter text'. Under the 'Mule Message' section, the 'Payload' item is expanded, showing its structure:

- SoapOutputPayload : Object
 - attachments : Object?
 - body : Binary?
 - headers : Object?

Other collapsed sections include 'Attributes' and 'Variables'. At the bottom of the panel is a 'Refresh Metadata' button.

30. Save the file.

31. Select the Output tab in the DataSense Explorer and expand Payload again; you should now see the payload structure.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. The payload structure is now fully expanded:

- Mule Message
 - Payload
 - Object : Object
 - body : Object?
 - listAllFlightsResponse : Object
 - return : Object*?
 - airlineName : String?
 - code : String?
 - departureDate : String?
 - destination : String?
 - emptySeats : Number
 - origin : String?
 - planeType : String?
 - price : Number

Test the application

32. Save the files to redeploy the project.
33. In Advanced REST Client, return to the middle tab – the one with the localhost requests.

34. Make a request to <http://localhost:8081/delta>; you should get a response that is object of type SoapOutputPayload.

Method Request URL
GET <http://localhost:8081/delta> ABORT ::

Parameters ▾

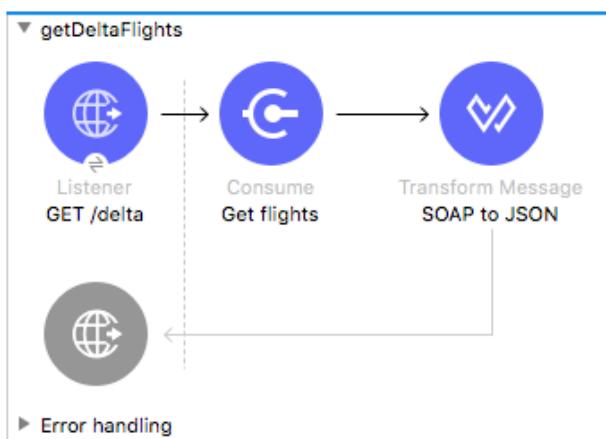
200 OK 9975.36 ms DETAILS ▾

✖️ ↻ ⟲ ⟳

org.mule.runtime.extension.api.soap.SapOutputPayload@26bdc7dc

Transform the response to JSON

35. Return to Anypoint Studio.
36. Add a Transform Message component to the end of the flow.
37. Set the display name to SOAP to JSON.



38. In the expression section of the Transform Message properties view, change the output type to json and the output to payload.

Output Payload ▾

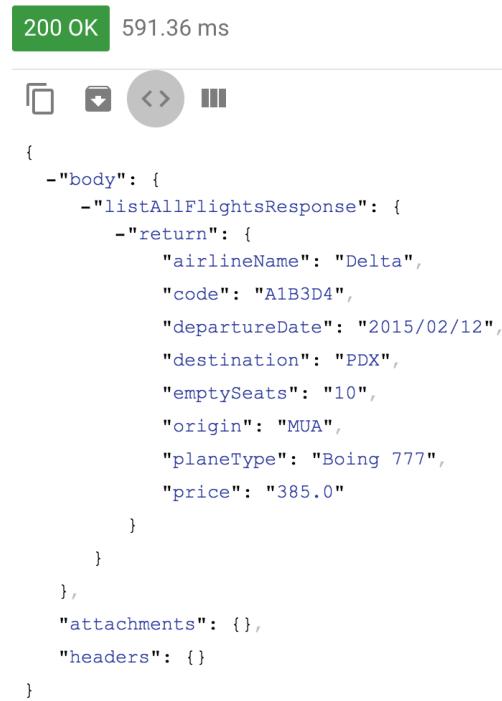
```
1 @%dw 2.0
2 output application/json
3 ---
4 payload
```

Test the application

39. Save the file to redeploy the project.

40. In Advanced REST Client, make another request to <http://localhost:8081/delta>; you should get JSON returned.

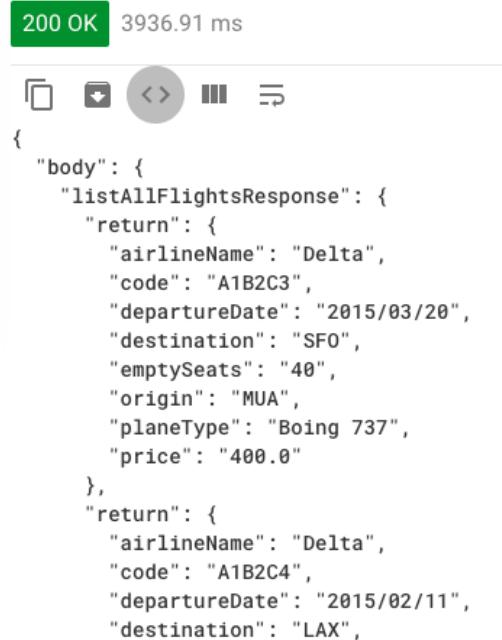
200 OK 591.36 ms



```
{  
  "body": {  
    "listAllFlightsResponse": {  
      "return": {  
        "airlineName": "Delta",  
        "code": "A1B3D4",  
        "departureDate": "2015/02/12",  
        "destination": "PDX",  
        "emptySeats": "10",  
        "origin": "MUA",  
        "planeType": "Boing 777",  
        "price": "385.0"  
      }  
    }  
  }  
}  
"attachments": {},  
"headers": {}  
}
```

41. Click the Toggle raw response view button; you should see all the flights.

200 OK 3936.91 ms



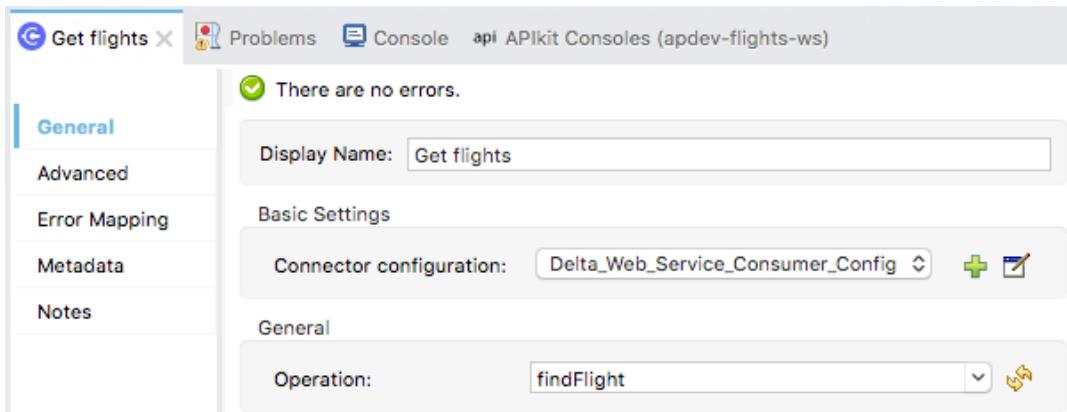
```
{  
  "body": {  
    "listAllFlightsResponse": {  
      "return": {  
        "airlineName": "Delta",  
        "code": "A1B2C3",  
        "departureDate": "2015/03/20",  
        "destination": "SFO",  
        "emptySeats": "40",  
        "origin": "MUA",  
        "planeType": "Boing 737",  
        "price": "400.0"  
      },  
      "return": {  
        "airlineName": "Delta",  
        "code": "A1B2C4",  
        "departureDate": "2015/02/11",  
        "destination": "LAX",  
        "emptySeats": "40",  
        "origin": "MUA",  
        "planeType": "Boing 737",  
        "price": "400.0"  
      }  
    }  
  }  
}
```

42. Add a query parameter called code and set it equal to LAX.

43. Send the request; you should still get all flights.

Call a different web service operation

44. Return to getDeltaFlights in Anypoint Studio.
45. In the properties view for Get flights Consume operation, change the operation to findFlight.



46. Save all files to redeploy the application.

Review metadata associated with the United Get flights operation response

47. Return to the Get flights properties view.
48. Select the Output tab in the DataSense Explorer and expand Payload; you should now see different payload metadata.

A screenshot of the DataSense Explorer. The top navigation bar has tabs for 'Input' and 'Output', with 'Output' selected. Below is a search bar with placeholder 'type filter text'. The main area shows a tree structure under 'Mule Message':

- ▼ Mule Message
 - ▼ Payload
 - ▼ Object : Object
 - ▼ body : Object?
 - ▼ findFlightResponse : Object
 - ▼ return : Object*?
 - airlineName : String?
 - code : String?
 - departureDate : String?
 - destination : String?
 - emptySeats : Number
 - origin : String?
 - planeType : String?
 - price : Number

49. Select the Input tab and expand Payload; you should see this operation now expects a destination.

```

Input Output
type filter text

▼ Mule Message
  ▼ Payload
    (Actual) Binary : Binary
    ▼ (Expected) Xml<findFlight> : Object
      ▼ findFlight : Object
        destination : String?
  ▶ Attributes
  Variables
  
```

Test the application

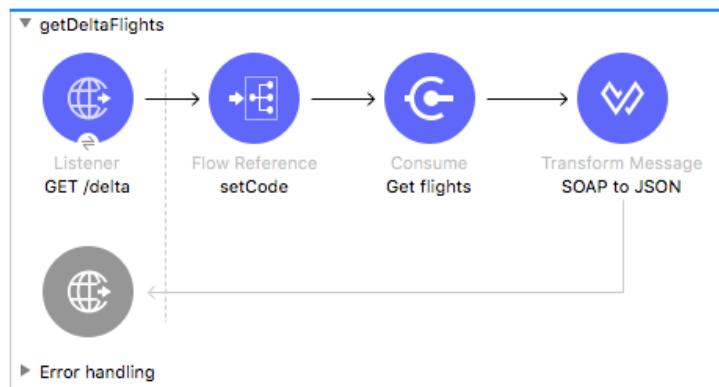
50. In Advanced REST Client, send the same request with the query parameter; you should get a 500 Server Error with a message that the operation requires input parameters.

500 Server Error 395.86 ms DETAILS ▾

Cannot build default body request for operation [findFlight], the operation requires input parameters

Use the set airport code subflow

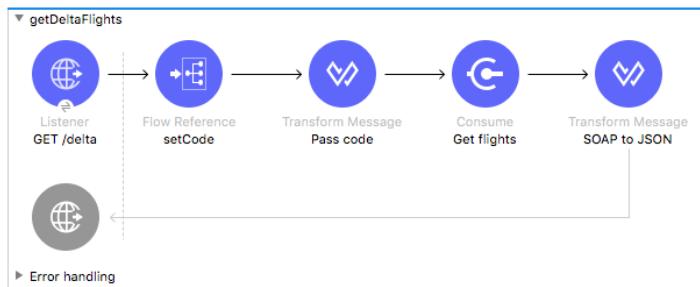
51. Return to getDeltaFlights in Anypoint Studio.
 52. Add a Flow Reference component after the GET /delta Listener.
 53. In the Flow Reference properties view, set the flow name and display name to setCode.



Use the Transform Message component to pass a parameter to the web service

54. Add a Transform Message component after the Flow Reference component.

55. Change its display name to Pass Code.



56. In the Pass code properties view, look at the input and output sections.

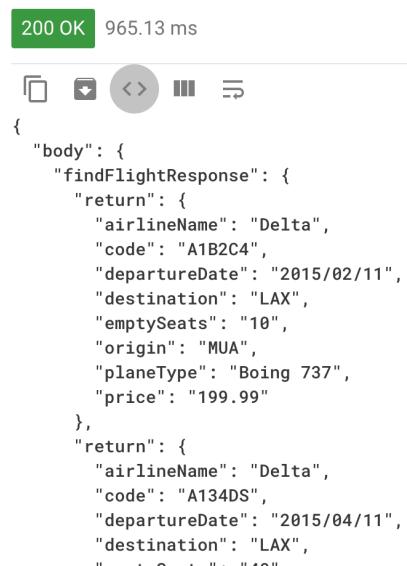
57. Drag the code variable in the input section to the destination element in the output section.

The screenshot shows the 'Pass code' properties view. On the left, under 'Input', there is a 'Payload : Binary' section with a 'Define metadata' link, and a 'Variables' section containing 'code : Nothing'. On the right, under 'Output', there is an 'Output' tab showing an XML payload definition. The XML code includes a 'findFlight' element with a 'destination' attribute. A green arrow points from the 'code' variable in the input section to the 'destination' attribute in the output section, indicating a drag-and-drop transformation.

Test the application

58. Save the file to redeploy the application.

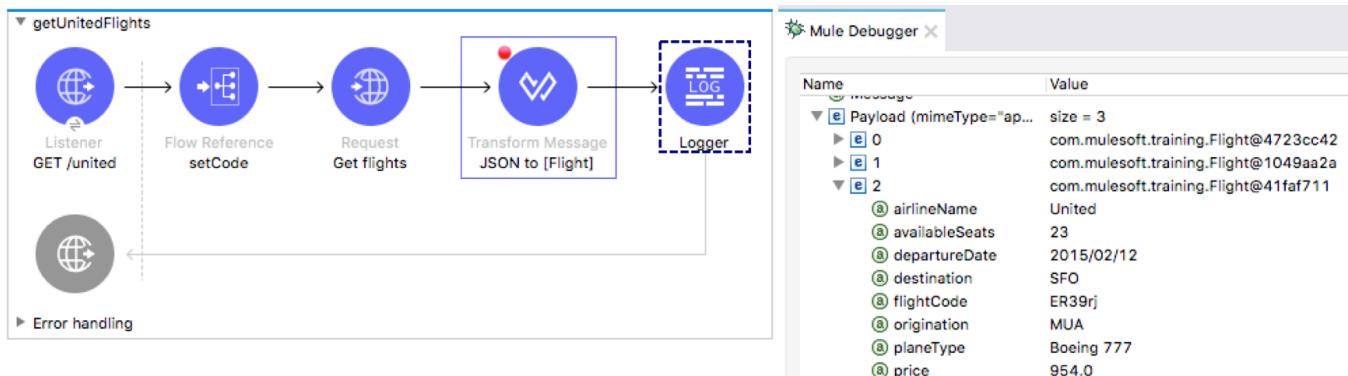
59. In Advanced REST Client, make another request.; you should now only see flights to LAX.



Walkthrough 8-4: Transform data from multiple services to a canonical format

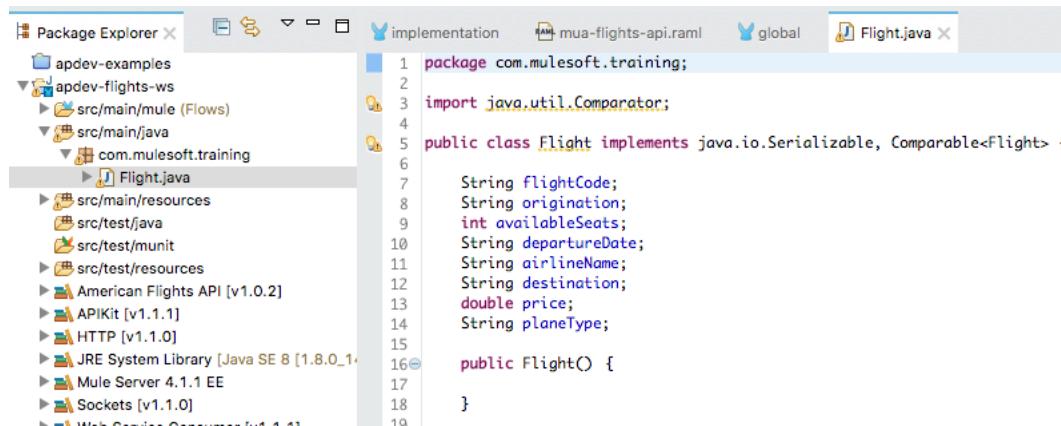
In this walkthrough, you will transform the JSON returned from the American and United web services and the SOAP returned from the Delta web service to the same format. You will:

- Define a metadata type for the Flight Java class.
- Transform the results from RESTful and SOAP web service calls to a collection of Flight objects.



Review Flight.java

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open Flight.java in src/main/java and review the file.



3. Close the file.

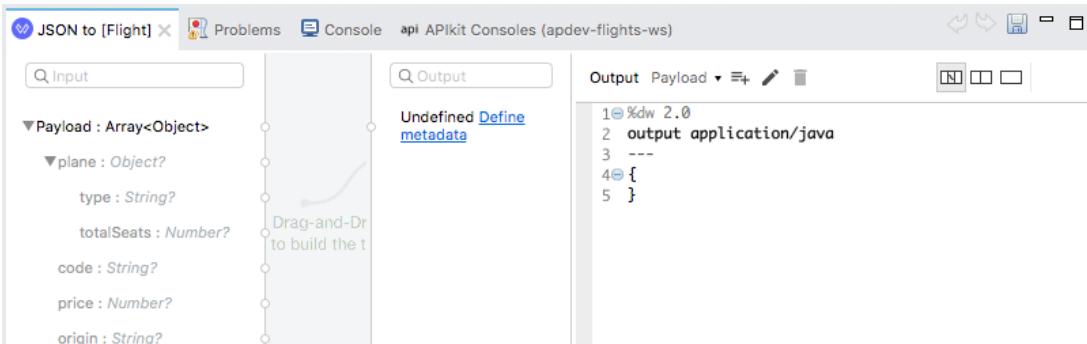
Change the American flow to return Java Flight objects instead of JSON

4. Return to getAmericanFlights in implementation.xml.
5. Add a Transform Message component to the end of the flow.
6. Add a Logger at the end of the flow.

7. Change the name of the Transform Message component to JSON to [Flight].



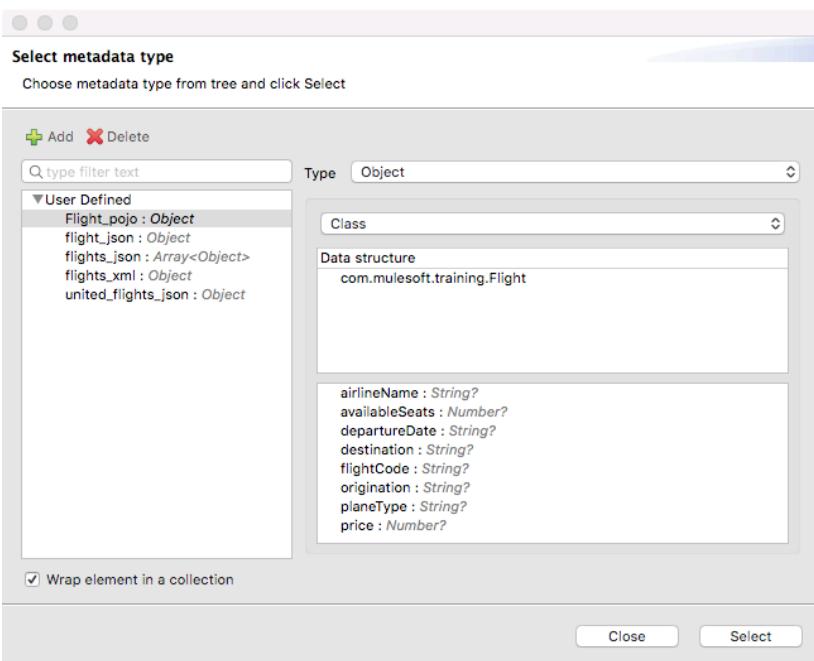
8. In the Transform Message properties view, look at the input section in the Transform Message properties view; you should see metadata already defined.



9. In the output section of the Transform Message properties view, click the Define metadata link.

10. In the Select metadata type dialog box, select the user-defined Flight_pojo.

11. Select Wrap element in a collection in the lower-left corner.



12. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
13. Map fields (except ID and totalSeats) by dragging them from the input section and dropping them on the corresponding field in the output section.

```
%dw 2.0
output application/java
payload map ( payload01 , indexOfPayload01 ) ->
{
    airlineName: "American",
    availableSeats: payload01.emptySeats,
    departureDate: payload01.departureDate,
    destination: payload01.destination,
    flightCode: payload01.code,
    origination: payload01.origin,
    planeType: payload01.plane.type,
    price: payload01.price
} as Object {
    class : "com.mulesoft.training.Flight"
}
```

14. Double-click the airlineName field in the output section.
15. In the generated DataWeave expression, change the airlineName value from null to "American".

```
%dw 2.0
output application/java
payload map ( payload01 , indexOfPayload01 ) ->
{
    airlineName: "American",
    availableSeats: payload01.emptySeats.
```

Test the application

16. Save to redeploy the project.
17. In Advanced REST Client, change the URL and make a request to <http://localhost:8081/american>; you should see a representation of a collection of Java objects.

200 OK 3370.05 ms DETAILS ▾

```
[{"id": 1, "airlineName": "American", "availableSeats": 1994, "departureDate": "2016-01-01T00:00:00Z", "destination": "New York", "flightCode": "AA1093", "origination": "Chicago", "planeType": "Boeing 737"}, {"id": 2, "airlineName": "American", "availableSeats": 2000, "departureDate": "2016-02-20T00:00:00Z", "destination": "Los Angeles", "flightCode": "AA1077", "origination": "New York", "planeType": "Boeing 737"}, {"id": 3, "airlineName": "American", "availableSeats": 3000, "departureDate": "2016-02-01T00:00:00Z", "destination": "Chicago", "flightCode": "AA1073", "origination": "Los Angeles", "planeType": "Boeing 737"}, {"id": 4, "airlineName": "American", "availableSeats": 4567, "departureDate": "2016-01-20T00:00:00Z", "destination": "Chicago", "flightCode": "AA1073", "origination": "New York", "planeType": "Boeing 737x"}]
```

Debug the application

18. Return to Anypoint Studio.
19. Stop the project.
20. Add a breakpoint to the Get flights operation.
21. Debug the project.
22. In Advanced REST Client, make another request to <http://localhost:8081/american>.
23. In the Mule Debugger, step to the Transform message component and examine the payload.

The screenshot shows the Mule Debugger interface. On the left, the message structure is displayed with various components expanded:

- attributes**: org.mule.extension.http.api.HttpResponseAttributes
- correlationId**: "0-72c81690-bdc6-11e8-b84a-784f43835e3e"
- payload**: [{ "ID": 5, "code": "rree1093", "price": 142, "departureDate": "2016-02-11T00:00:00", "origin": "MUA", "destination": "SFO", "emptySeats": 1, "plane": { "type": "Boeing 737", "totalSeats": 150 } }]
- vars**: java.util.Map size = 1
- code**: "SFO"

On the right, the expanded **payload** component shows the JSON structure:

```
[ { "ID": 5, "code": "rree1093", "price": 142, "departureDate": "2016-02-11T00:00:00", "origin": "MUA", "destination": "SFO", "emptySeats": 1, "plane": { "type": "Boeing 737", "totalSeats": 150 } } ]
```

24. Step to the Logger and look at the payload it should be a collection of Flight objects.

The screenshot shows the Mule Debugger interface. The message structure on the left includes a **payload** component expanded to show five flight objects:

- 0**: com.mulesoft.training.Flight@2615c58b
- 1**: com.mulesoft.training.Flight@3ffa16ad
- 2**: com.mulesoft.training.Flight@410afdc9
- 3**: com.mulesoft.training.Flight@02c75f8fa
- 4**: com.mulesoft.training.Flight@24c45962

Below the debugger, the **implementation** perspective shows the flow diagram:

```
graph LR; Listener[Listener  
GET /american] --> FlowRef[Flow Reference  
setCode]; FlowRef --> GetFlights[Get flights]; GetFlights --> Transform[Transform Message  
JSON to [Flight]]; Transform --> Logger[Logger]
```

25. Step through the rest of the application and switch perspectives.

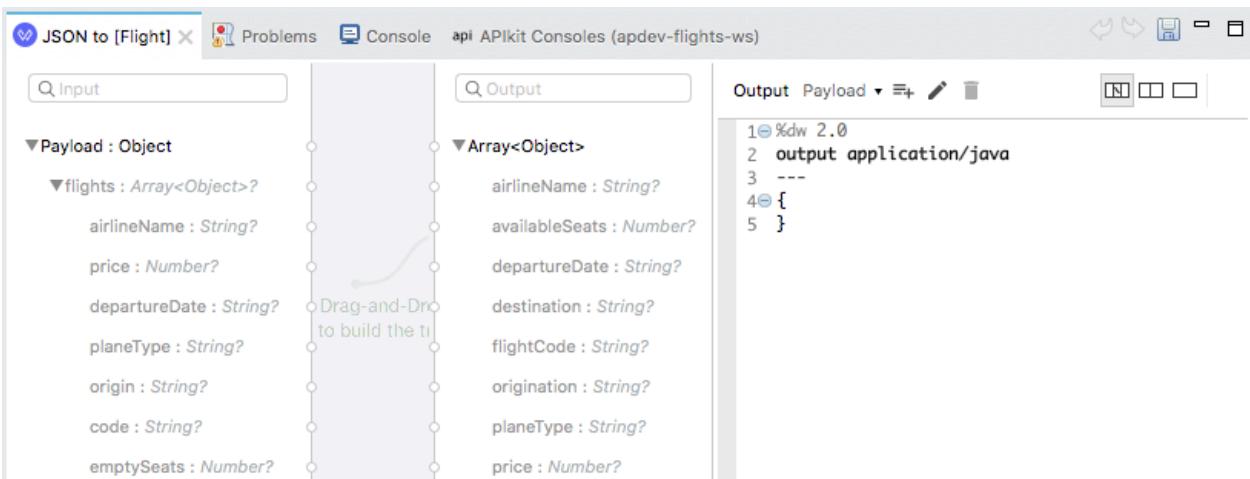
Change the United airline flows to return Java Flight objects instead of JSON

26. Return to geUnitedFlights in implementation.xml.

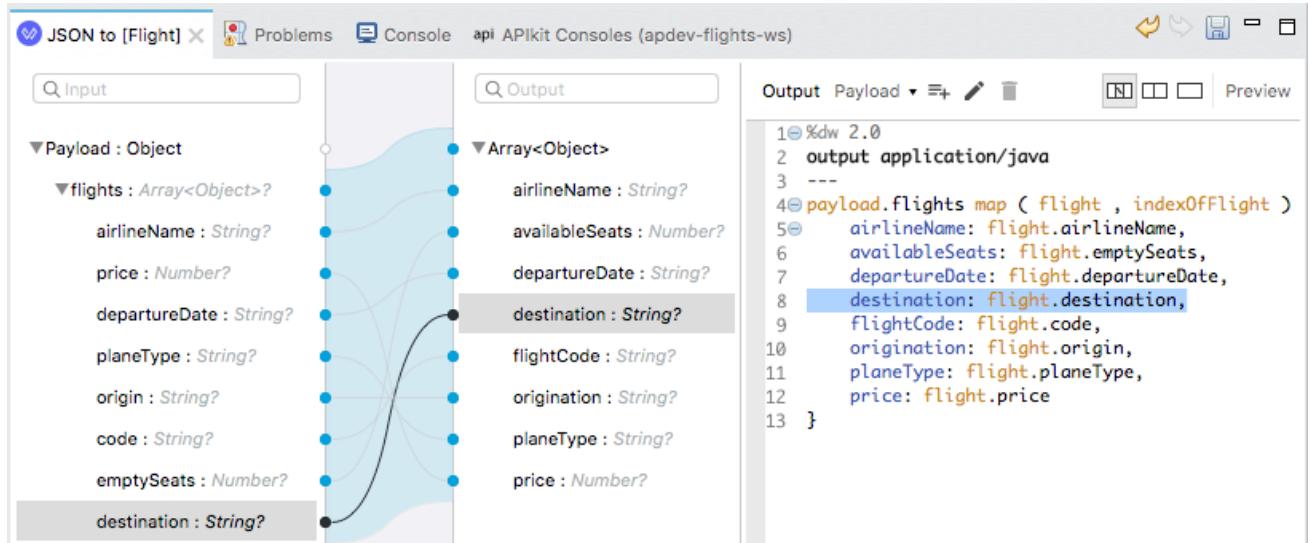
27. Add a Transform Message component at the end of the flow.
28. Change the name of the Transform Message component to JSON to [Flight].
29. Add a Logger to the end of the flow.



30. In the Transform Message properties view, look at the input section; you should see metadata already defined.
31. In the output section of the Transform Message properties view, click the Define metadata link.
32. In the Select metadata type dialog box, select the user-defined Flight_pojo.
33. Select Wrap element in a collection in the lower-left corner.
34. Click Select; you should now see output metadata in the output section of the Transform Message properties view.



35. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.



Debug the application

36. Add a breakpoint to the Transform Message component.
37. Save the files to redeploy the project.
38. In Advanced REST Client, change the URL to make a request to <http://localhost:8081/united>.
39. In the Mule Debugger, examine the payload.



40. Step to the Logger and look at the payload it should be a collection of Flight objects.

The screenshot shows the Mule Debugger interface. On the left, there is a tree view of the payload structure:

```
▶ [e] attributes = {org.mule.extension.http.api.HttpResponse}
  @ correlationId = "0-ab6bd5c0-bdc8-11e8-844f-784fc"
  ▶ [e] payload = {java.util.ArrayList} size = 3
    ▶ [e] 0 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@77cd2a88
    ▶ [e] 1 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@...
    ▶ [e] 2 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@...
      @^mediaType = application/java; charset=UTF-8
  ▶ [e] vars = {java.util.Map} size = 1
    ▶ [e] code = "SFO"
```

On the right, the payload value is displayed as `com.mulesoft.training.Flight@77cd2a88`. Below the debugger, the application navigation bar shows `implementation`, `config.yaml`, `global`, and `application-types.xml`. A flow diagram for the `getUnitedFlights` endpoint is visible, showing the sequence of components: Listener, Flow Reference, Request, Transform Message, and Logger.

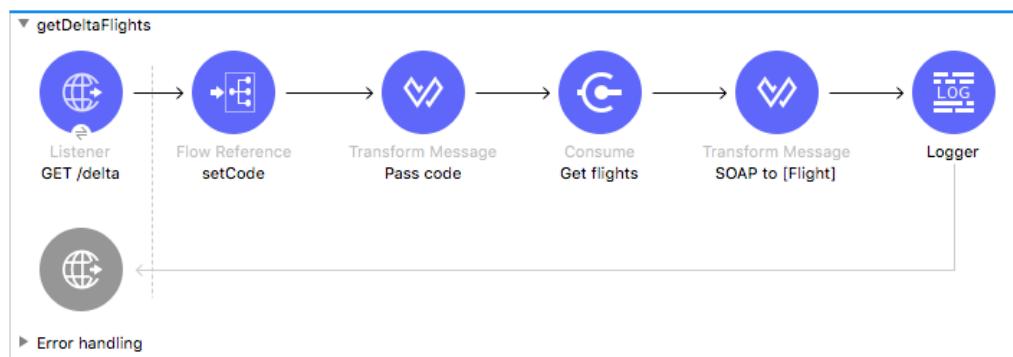
41. Step through the rest of the application and switch perspectives.

Change the Delta flow to return Java Flight objects

42. Return to `getDeltaFlights` in `implementation.xml`.

43. Change the name of the Transform Message component to SOAP to [Flight].

44. Add a Logger to the end of the flow.

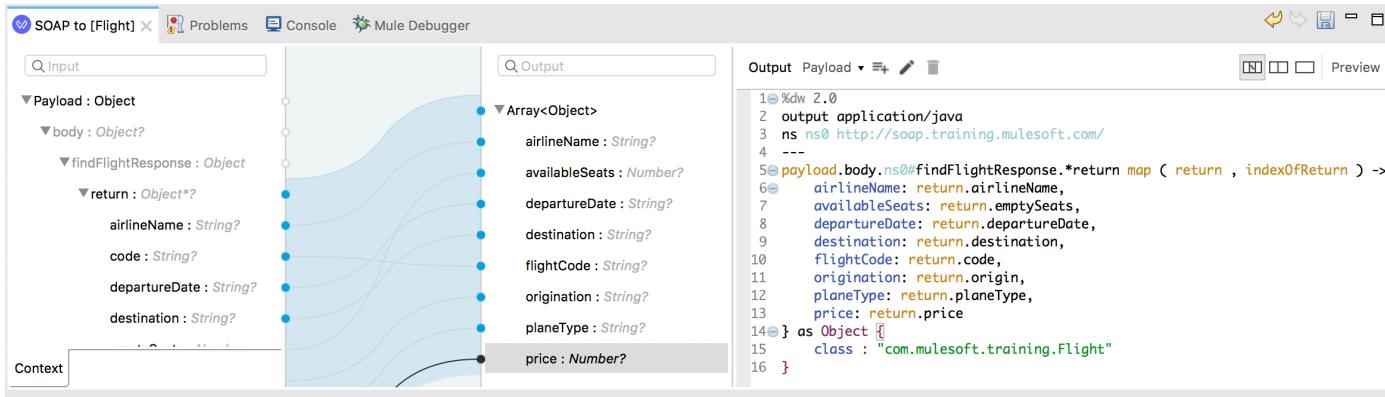


45. Look at the input section in the Transform Message properties view; you should see metadata already defined.

46. In the output section of the Transform Message properties view, click the Define metadata link.

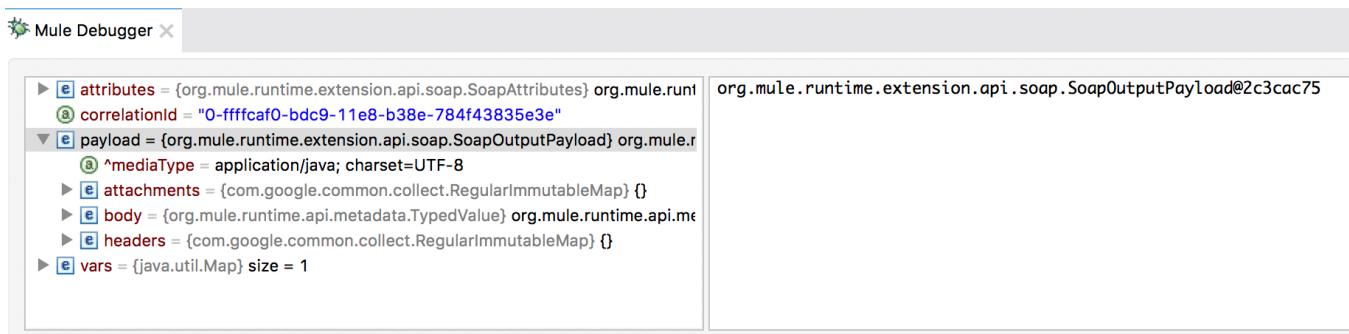
47. In the Select metadata type dialog box, select the user-defined Flight_pojo.

48. Select Wrap element in a collection in the lower-left corner.
49. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
50. Map the fields by dragging them from the input section and dropping them on the corresponding field in the output section.



Debug the application

51. Add a breakpoint to the Transform Message component.
52. Save the file to redeploy the project.
53. In Advanced REST Client, change the URL to make a request to <http://localhost:8081/delta>.
54. In the Mule Debugger, examine the payload.



55. Step to the Logger and look at the payload it should be a collection or Flight objects.

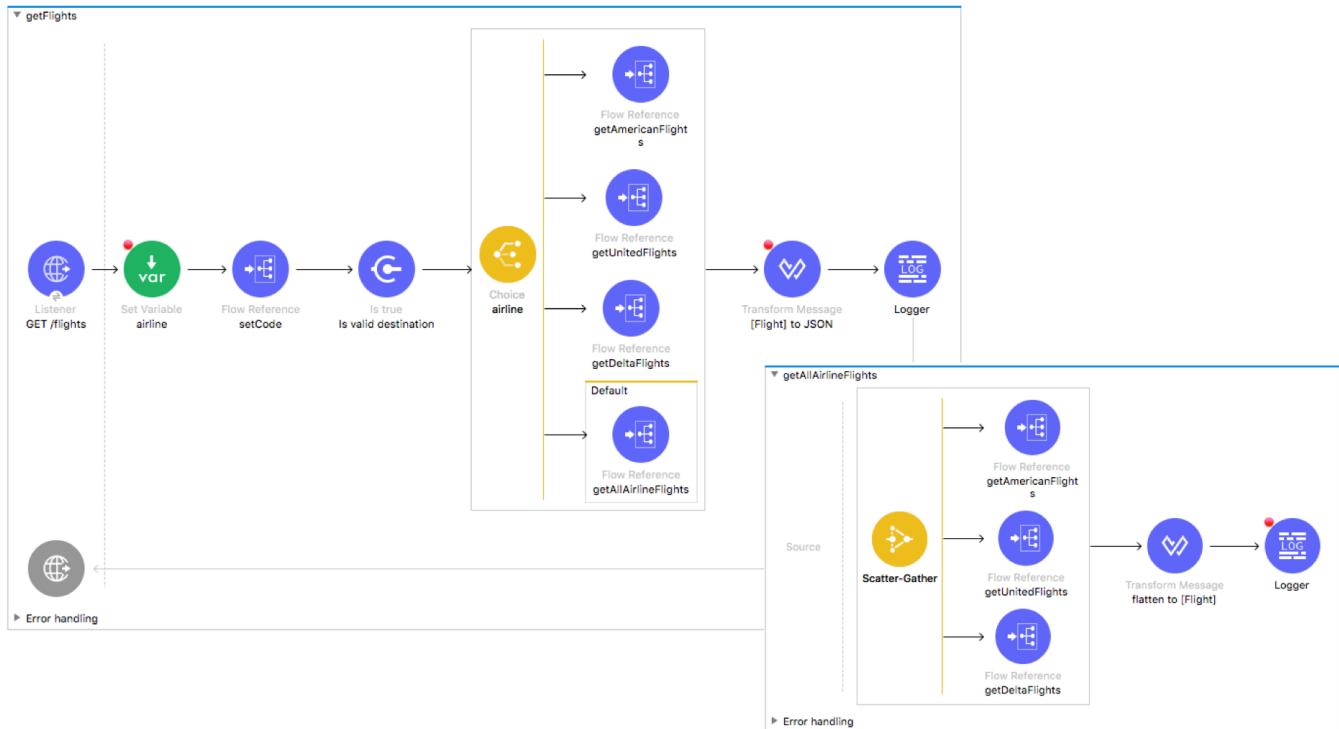
The screenshot shows the Mule Debugger interface. On the left, the 'implementation' tab is selected, displaying a flow named 'getDeltaFlights'. The flow consists of six components: 'Listener GET /delta', 'Flow Reference setCode', 'Transform Message Pass code', 'Consume Get flights', 'Transform Message SOAP to [Flight]', and 'Logger'. The 'Logger' component is highlighted with a dashed blue box. On the right, the 'Mule Debugger' window shows the payload structure:

```
▶ [E] attributes = {org.mule.runtime.extension.api.soap.SoapAttributes} org.mule.runtime.extension.api.soap.SoapAttributes
  @ correlationId = "0-ffffcaf0-bdc9-11e8-b38e-784f43835e3e"
  ▶ [E] payload = {[java.util.ArrayList] size = 3}
    ▶ [E] 0 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@32fd1646
    ▶ [E] 1 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@5f78c3c
    ▶ [E] 2 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@2e9e1997
    @ ^mediaType = application/java; charset=UTF-8
  ▶ [E] vars = {[java.util.Map] size = 1}
```

The 'application-types.xml' file is also visible in the implementation tab.

56. Step through the rest of the application and switch perspectives.

Module 9: Controlling Event Flow



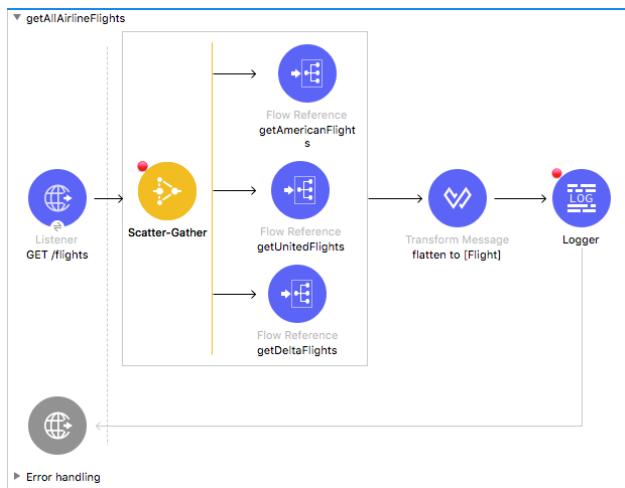
At the end of this module, you should be able to:

- Multicast events.
- Route events based on conditions.
- Validate events.

Walkthrough 9-1: Multicast an event

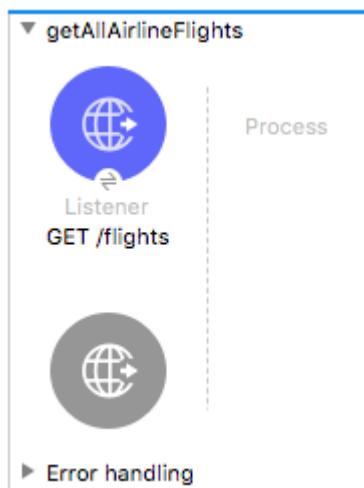
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.



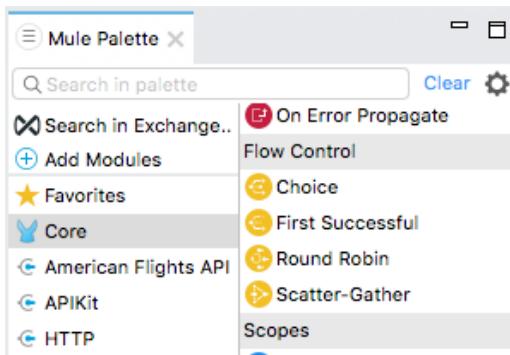
Create a new flow

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.



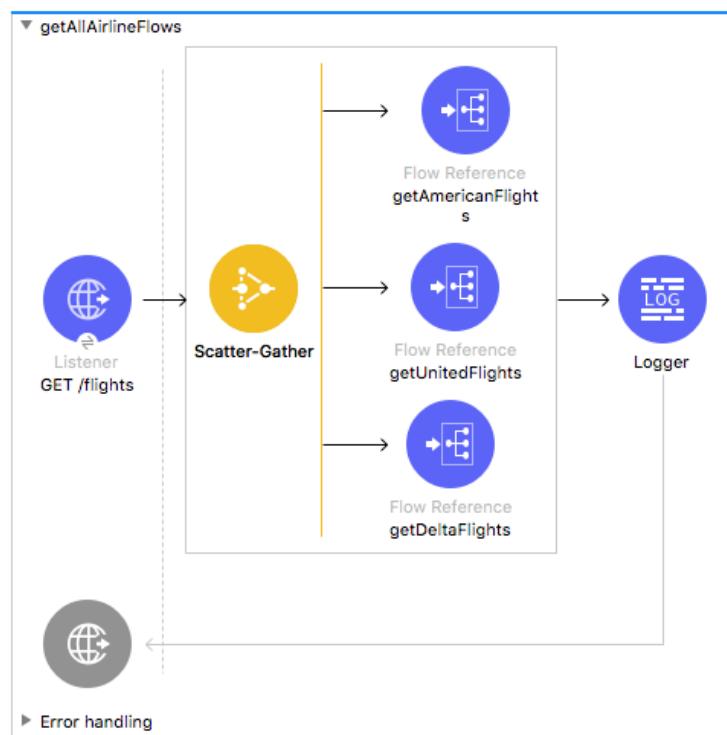
Browse the flow control elements in the Mule Palette

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 and display name to getAmericanFlights.
11. Set the flow name and display name of the second Flow Reference to getUnitedFlights.
12. Set the flow name and display name of the third Flow Reference to getDeltaFlights.
13. Add a Logger after the Scatter-Gather.



Review the metadata for the Scatter-Gather output

14. In the Logger properties view, explore the input payload structure in the DataSense Explorer.

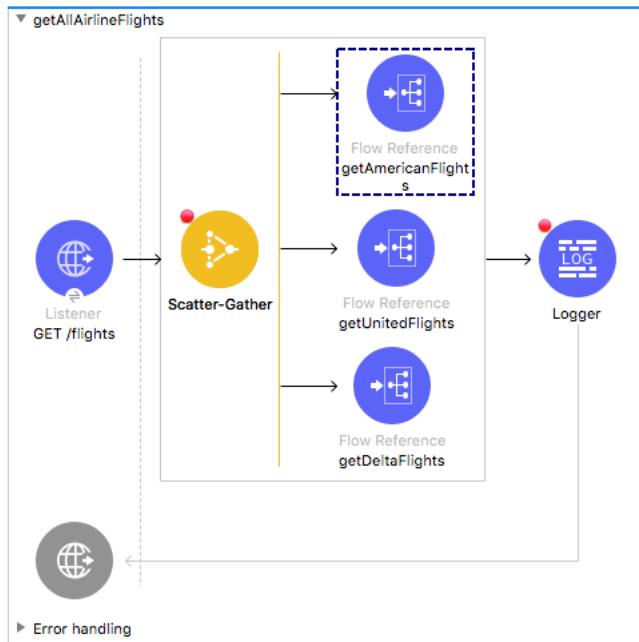
The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top contains the placeholder 'type filter text'. Below it, the 'Mule Message' structure is displayed as a tree:

- Mule Message
 - Payload
 - Object : Object
 - 0 : Object
 - 1 : Object
 - payload : Array<Object>
 - attributes : Object
 - 2 : Object
 - payload : Array<Object>
 - airlineName : String?
 - availableSeats : Number?
 - departureDate : String?
 - destination : String?
 - flightCode : String?
 - origination : String?
 - planeType : String?
 - price : Number?
 - attributes : Object
 - Attributes
 - Void : Void
 - Variables
 - code
 - Nothing : Nothing

Debug the application

15. Add a breakpoint to the Scatter-Gather.
16. Add a breakpoint to the Logger.
17. Save the file to redeploy the project in debug mode.
18. In Advanced REST Client, change the URL to make a request to <http://localhost/flights>.

19. In the Mule Debugger, step through the application; you should step through each of the airline flows.



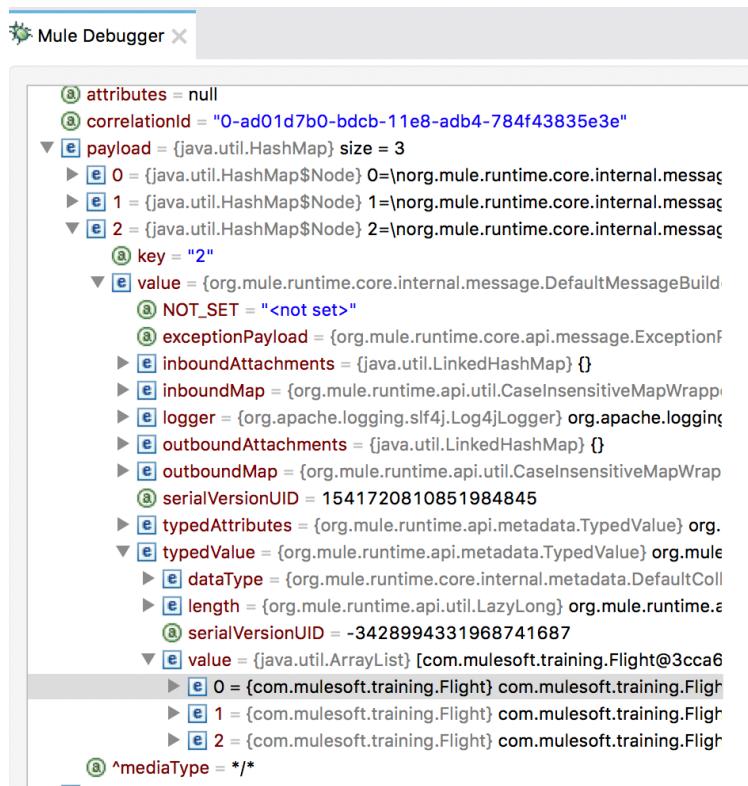
20. Stop at the Logger after the Scatter-Gather and explore the payload.

The screenshot shows the Mule Debugger interface. At the top, there's a "Mule Debugger" window displaying the payload details:

```
attributes = null
correlationId = "0-ad01d7b0-bdcb-11e8-adb4-784f43835e3e"
payload = {java.util.HashMap} size = 3
vars = {java.util.Map} size = 1
```

On the right, the value "size = 3" is highlighted. Below this, there's an "implementation" window showing the Mule flow. The flow starts with a "Listener GET /flights" component, followed by a "Scatter-Gather" component. Three parallel paths lead from the "Scatter-Gather" to "Flow Reference" components: "getAmericanFlight(s)", "Flow Reference getUnitedFlights", and "getDeltaFlights". These paths converge back to a "Logger" component, which is highlighted with a dashed blue box. The "implementation" window also includes tabs for "config.yaml", "global", and "application-types.xml".

21. Drill-down into one of the objects in the payload.



The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". The payload object is expanded to show three items (0, 1, 2) in a list. Item 2 is selected, revealing its internal structure:

- key = "2"
- value = {org.mule.runtime.core.internal.message.DefaultMessageBuilder}
 - NOT_SET = "<not set>"
 - exceptionPayload = {org.mule.runtime.core.api.message.Exception}
 - inboundAttachments = {java.util.LinkedHashMap} {}
 - inboundMap = {org.mule.runtime.api.util.CaseInsensitiveMapWrapper} {}
 - logger = {org.apache.logging.slf4j.Log4jLogger} org.apache.logging
 - outboundAttachments = {java.util.LinkedHashMap} {}
 - outboundMap = {org.mule.runtime.api.util.CaseInsensitiveMapWrapper} {}
 - serialVersionUID = 1541720810851984845
 - typedAttributes = {org.mule.runtime.api.metadata.TypedValue} org.mule
 - typedValue = {org.mule.runtime.api.metadata.TypedValue} org.mule
 - dataType = {org.mule.runtime.core.internal.metadata.DefaultColl}
 - length = {org.mule.runtime.api.util.LazyLong} org.mule.runtime.e
 - serialVersionUID = -3428994331968741687
 - value = [java.util.ArrayList] [com.mulesoft.training.Flight@3cca6]
 - 0 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight
 - 1 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight
 - 2 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight
- ^mediaType = "*/*"

22. Step through the rest of the application and switch perspectives.

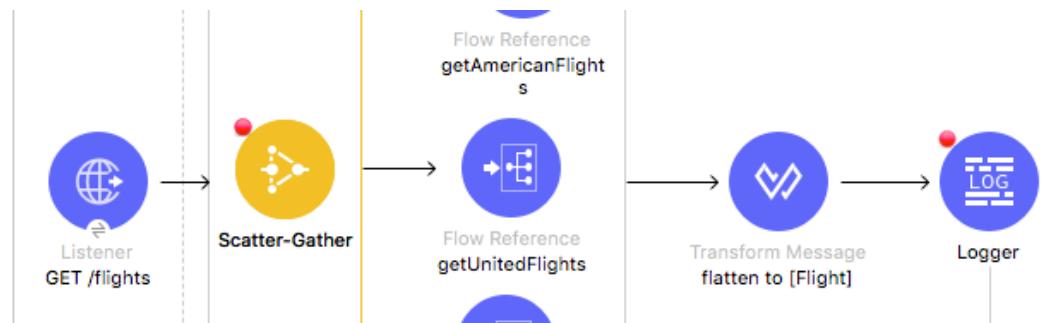
23. Return to Advanced REST Client and review the response; although the result is Java and you see a lot of characters, you should also see the Java for flight data for the three airlines.

Flatten the combined results

24. Return to Anypoint Studio.

25. In getAllAirlineFlights, add a Transform Message component before the Logger.

26. Change the display name to flatten to [Flight].



27. In the input section of the Transform Message properties view, review the payload data structure.

The screenshot shows the 'Transform Message' properties view in Mule Studio. The 'Input' tab is selected. Under 'Payload : Object', there are three objects labeled 0, 1, and 2. Object 2 contains a 'payload' array with six fields: airlineName, availableSeats, departureDate, destination, flightCode, and origination.

```
▼Payload : Object
  ▶0 : Object
  ▶1 : Object
  ▶2 : Object
    ▼payload : Array<Object>
      airlineName : String?
      availableSeats : Number?
      departureDate : String?
      destination : String?
      flightCode : String?
      origination : String?
```

28. In the expression section, use the DataWeave flatten function to flatten the collection of objects into a single collection.

```
1 %dw 2.0
2   output application/java
3   ---
4   flatten(payload..payload)
```

Debug the application

29. Save the file to redeploy the project.
30. In Advanced REST Client, make the same request to <http://localhost:8081/flights>.

31. In the Mule Debugger, press Resume until you are stopped at the Logger at the end of the Scatter-Gather; you should see the payload is now one ArrayList of Flights.

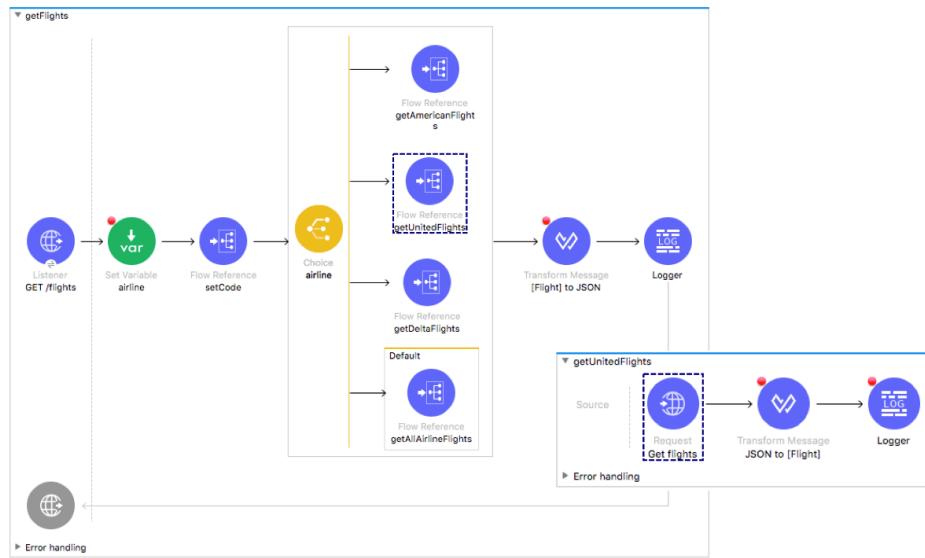
The screenshot shows the Mule Debugger interface. At the top, there's a toolbar with icons for Stop, Run, Break, Step Into, Step Out, Step Over, and Refresh. Below the toolbar, the title bar says "Mule Debugger". The main area has two panes. The left pane displays the debugger's tree view with expanded sections for attributes, correlationId, and payload. The payload section shows 11 elements, each representing a Flight object. The right pane shows the value of the size variable, which is 11. Below the debugger, the "implementation" tab is selected in the navigation bar, showing the Mule flow diagram. The flow starts with a Listener component (GET /flights) connected to a Scatter-Gather component. The Scatter-Gather component has three outgoing paths, each labeled with a Flow Reference: "getAmericanFlights", "Flow Reference getUnitedFlights", and "Flow Reference getDeltaFlights". These paths lead to three separate message routers. The final path from the routers leads to a Transform Message component, which is configured with the transformation "flatten to [Flight]". This is followed by a Logger component, which is highlighted with a dashed blue border. The configuration tab "config.yaml" is also visible in the navigation bar.

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

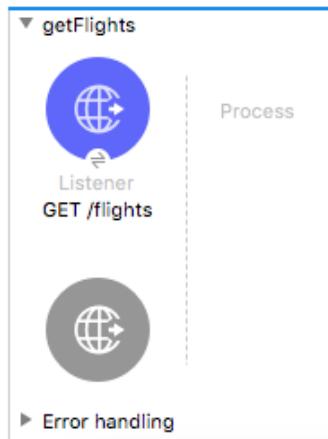
1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Open mua-flights-api.raml in src/main/resources/api.
3. Locate the airline query parameter and its possible values.

```
airline:
  displayName: Airline
  required: false
  enum:
    - united
    - delta
    - american
```

Create a new flow

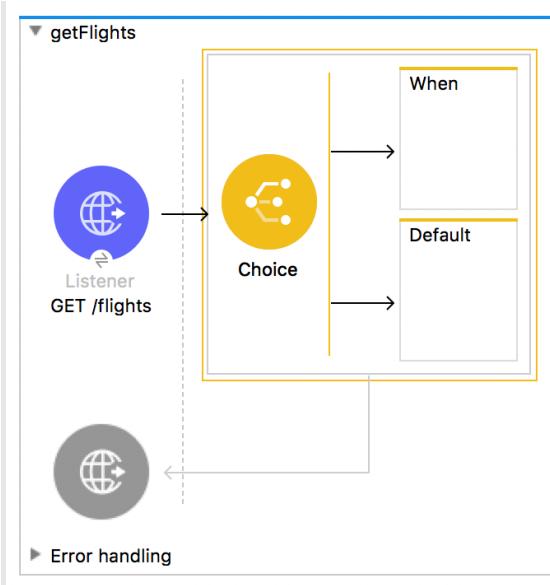
4. Return to implementation.xml.
5. Drag a Flow scope from the Mule Palette and drop it at the top of the canvas above all the other flows.
6. Change the name of the flow to getFlights.

7. Move the GET /flights HTTP Listener from getAllAirlineFlights to the source section of getFlights.



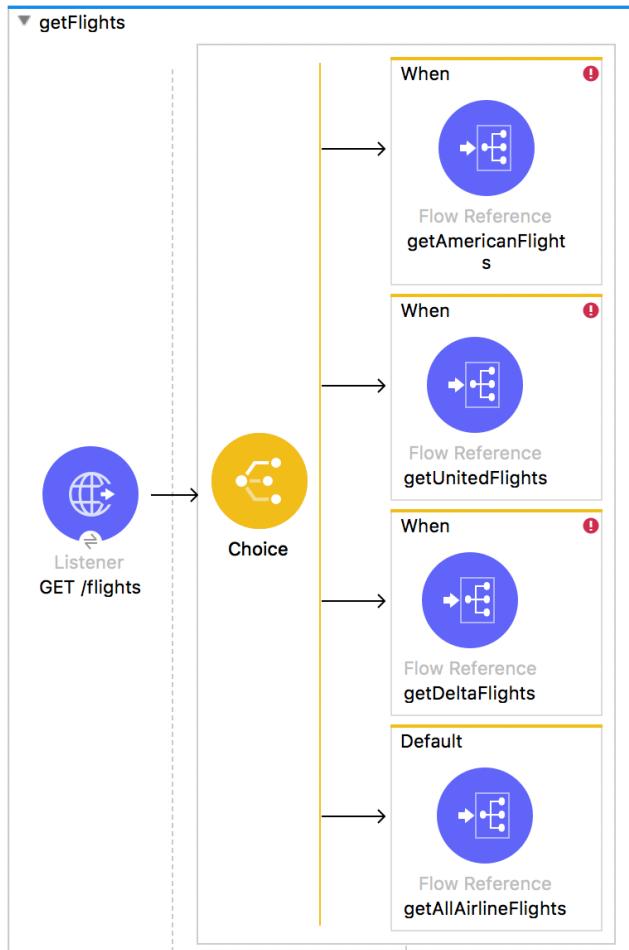
Add a Choice router

8. Drag a Choice flow control element from the Mule Palette and drop it in process section of getFlights.



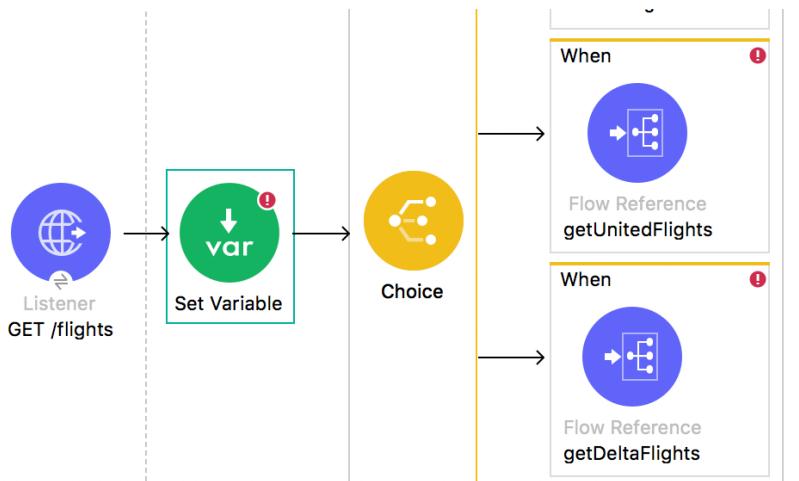
9. Add three Flow Reference components to the Choice router.
10. Add a Flow Reference component to the default branch of the router.
11. In the first Flow Reference properties view, set the flow name and display name to getAmericanFlights.
12. Set the flow names and display names for the other two Flow References to getUnitedFlights and getDeltaFlights.

13. For the Flow Reference in the default branch, set the flow name and display name to getAllAirlineFlights.



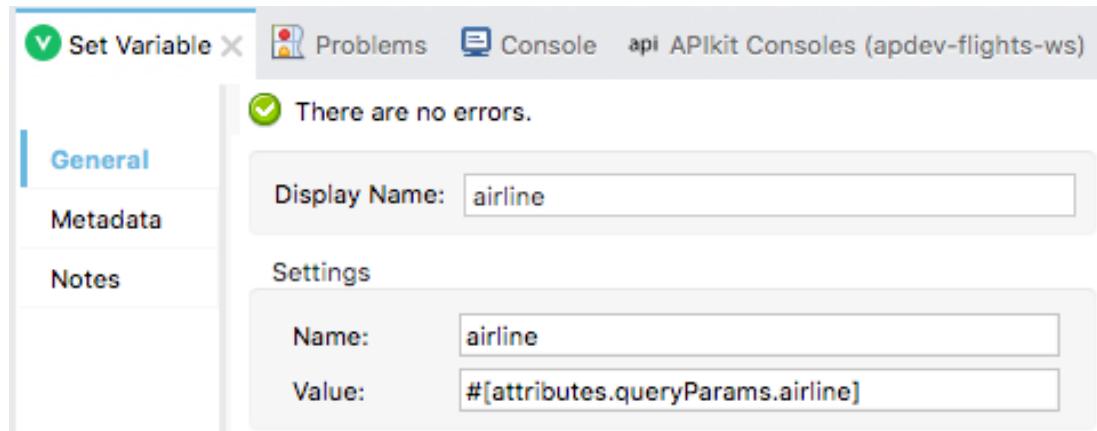
Store the airline query parameter in a variable

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.

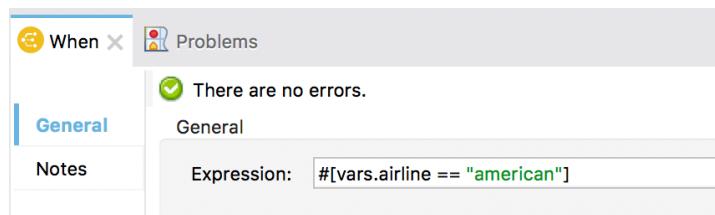
```
##[attributes.queryParams.airline]
```



Configure the Choice router

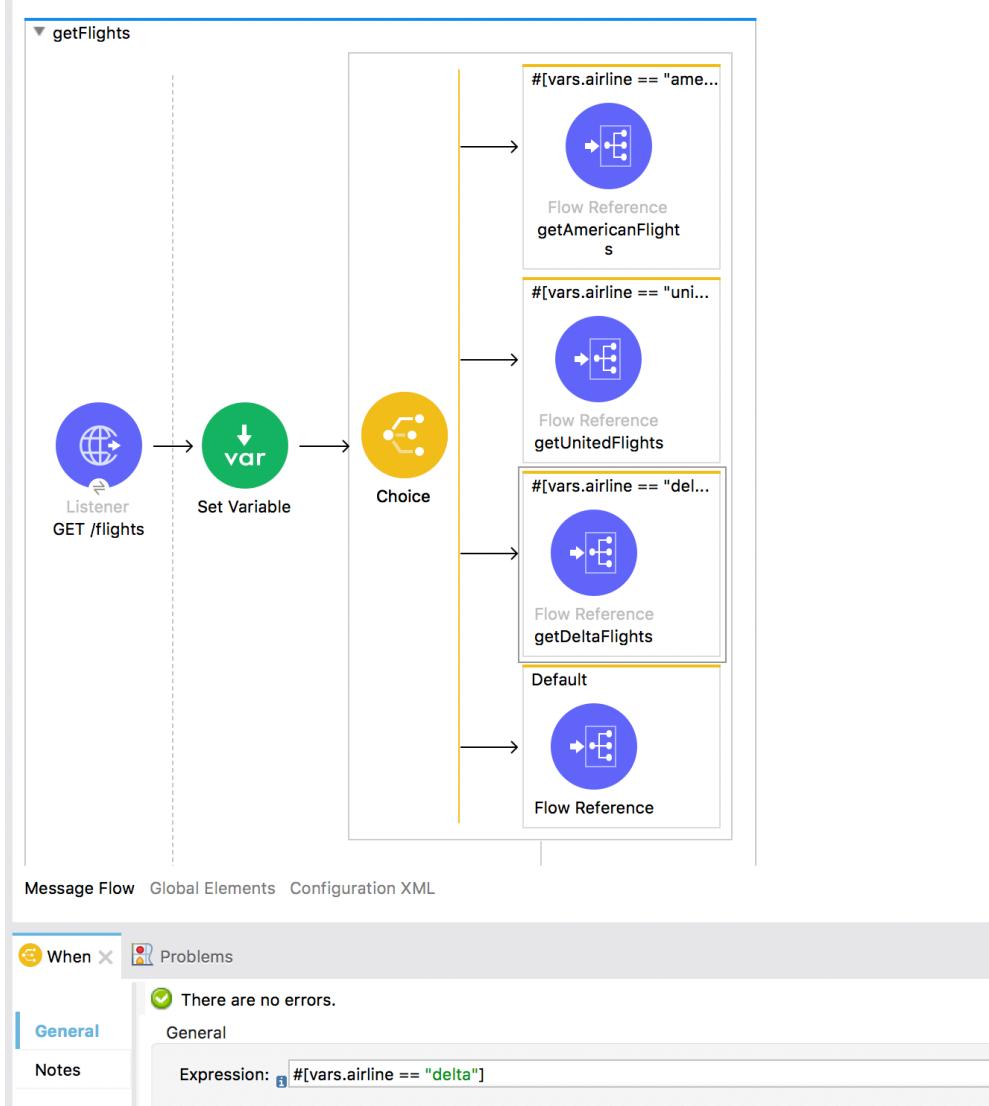
17. In the Choice router getAmericanFlights flow reference branch, click the When scope.
18. In the When properties view, add an expression to check if the airline variable is equal to american.

```
##[vars.airline == "american"]
```



19. Set a similar expression for the United route, routing to it when vars.airline is equal to united.

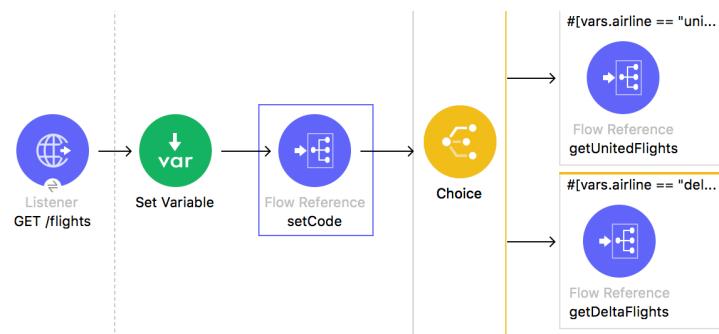
20. Set a similar expression for the Delta route, routing to it when vars.airline is equal to delta.



Route all requests through the router

21. Add a Flow Reference to the flow before the Choice router.

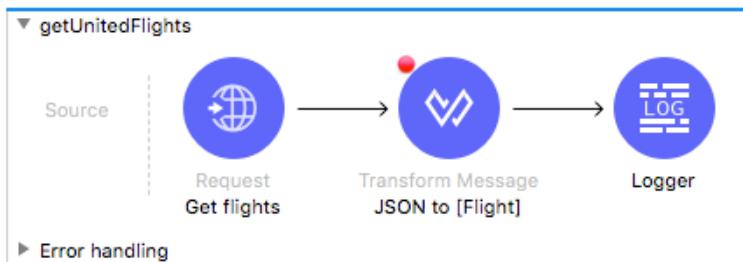
22. Set the flow name and display name to setCode.



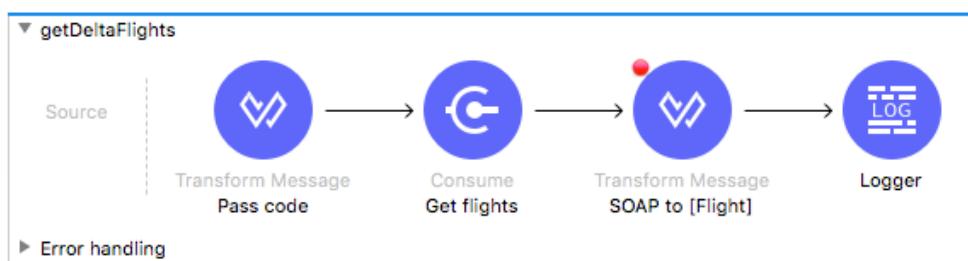
23. In getAmericanFlights, delete the HTTP Listener and the setCode Flow Reference.



24. In getUnitedFlights, delete the HTTP Listener and the setCode Flow Reference.



25. In getDeltaFlights, delete the HTTP Listener and the setCode Flow Reference.

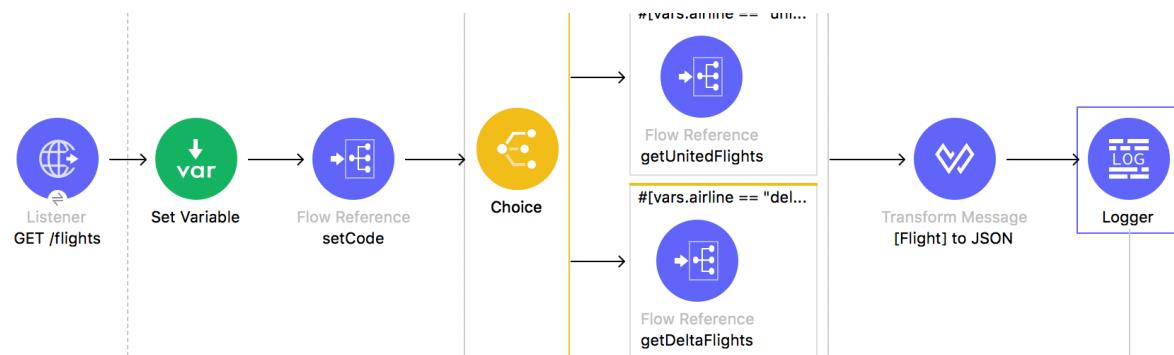


Return JSON from the flow

26. Add a Transform Message component after the Choice router.

27. Change its display name to [Flight] to JSON.

28. Add a Logger at the end of the flow.



29. In the Transform Message properties view, set the output type to JSON and the output to payload.

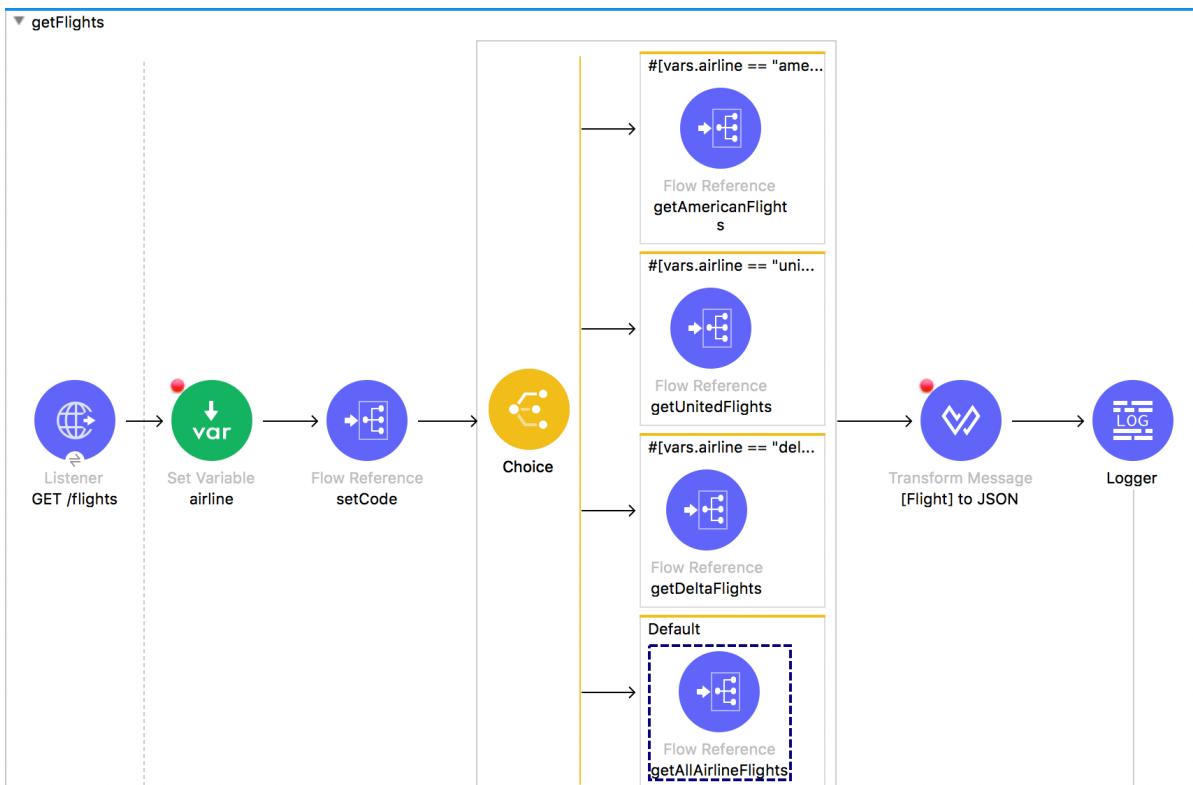
```

1 %dw 2.0
2 output application/json
3 ---
4 payload

```

Debug the application

30. Add a breakpoint to the Set Variable transformer at the beginning of getFlights.
31. Add a breakpoint to the Transform Message component after the Choice router.
32. Save the file to redeploy the project in debug mode.
33. In Advanced REST Client, make a request to <http://localhost:8081/flights>.
34. In the Mule Debugger, step through the application; you should see the Choice router pass the event to the default branch.



35. Resume to the Transform Message component after the Choice router; the payload should be an ArrayList of Flight objects
36. Step to the Logger; the payload should be JSON.
37. Step to the end of the application.

38. Return to Advanced REST Client; you should see American, United, and Delta flights to SFO (the default airport code).

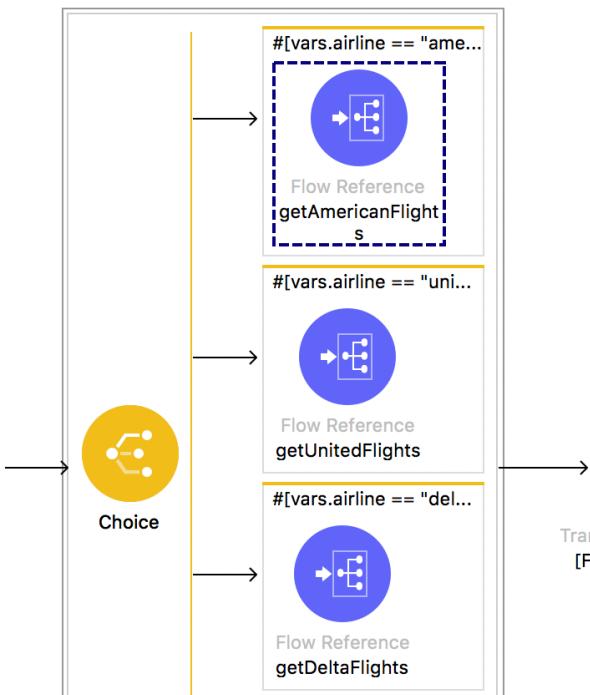
The screenshot shows the Advanced REST Client interface. At the top, it displays the Method (GET) and Request URL (http://localhost:8081/flights). Below the URL, there are buttons for SEND and more options. Under the parameters section, there's a dropdown menu set to "Parameters". The main area shows the response details: a green box indicates "200 OK" and "94251.28 ms". To the right, there's a "DETAILS" button with a dropdown arrow. Below this, there are several icons: a square, a downward arrow, a circular arrow, and a bar chart. The response body is displayed as a JSON array of 11 flight objects. Each object has properties like price, flightCode, availableSeats, planeType, departureDate, origination, airlineName, and destination. The first few flights are from American, and the last one is from United.

```
[Array[11]
-0: {
  "price": 142,
  "flightCode": "rree1093",
  "availableSeats": 1,
  "planeType": "Boeing 737",
  "departureDate": "2016-02-11T00:00:00",
  "origination": "MUA",
  "airlineName": "American",
  "destination": "SFO"
},
-1: { ... }
,-2: { ... }
,-3: { ... }
,-4: { ... }
,-5: {
  "price": 400,
  "flightCode": "ER38sd",
  "availableSeats": 0,
  "planeType": "Boeing 737",
  "departureDate": "2015/03/20",
  "origination": "MUA",
  "airlineName": "United",
  "destination": "SFO"
}
]
```

39. Add an airline query parameter set to american and send the request:

<http://localhost:8081/flights?airline=american>.

40. In the Mule Debugger, step through the rest of the application; you should see the message passed to getAmericanFlights and then back to getFlights.



41. Return to Advanced REST Client; you should see only American flights to SFO returned.

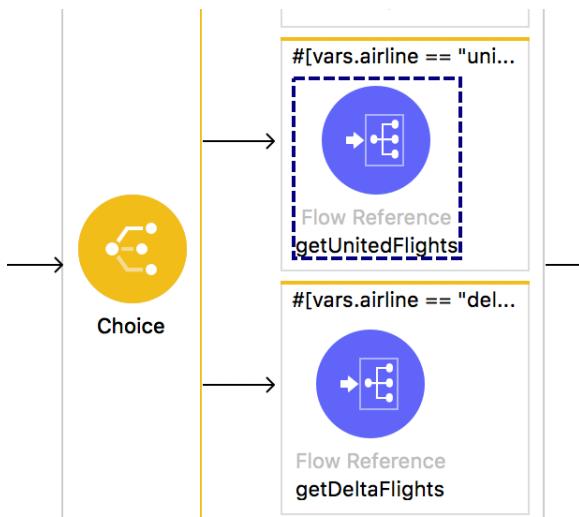
```
200 OK 19641.09 ms
[{"price": 142.0,
 "flightCode": "rree1093",
 "availableSeats": 1,
 "planeType": "Boeing 737",
 "departureDate": "2016-02-11T00:00:00",
 "origination": "MUA",
 "airlineName": "American",
 "destination": "SFO"}]
```

42. Change the airline to delta and add a second query parameter code set to LAX:

<http://localhost:8081/flights?airline=united&code=LAX>.

43. Send the request.

44. In the Mule Debugger, step through the application; the message should be routed to the United branch.



45. Return to Advanced REST Client; you should see only United flights to LAX are returned.

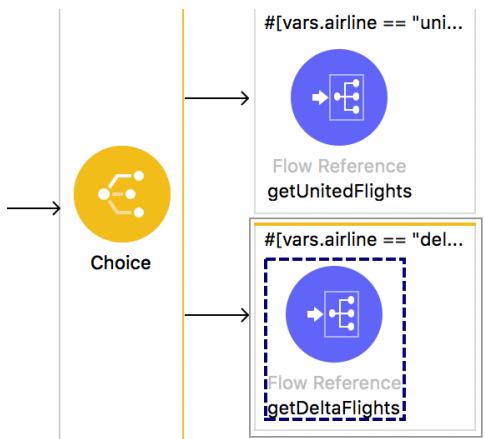
```
200 OK 47214.31 ms
{
  {
    "price": 345.99,
    "flightCode": "ER45if",
    "availableSeats": 52,
    "planeType": "Boeing 737",
    "departureDate": "2015/02/11",
    "origination": "MUA",
    "airlineName": "United",
    "destination": "LAX"
  },
}
```

46. Change the airline to delta and the code to PDX:

<http://localhost:8081/flights?airline=delta&code=PDX>

47. Send the request.

48. In the Mule Debugger, step through the application; the message should be routed to the Delta branch.



49. Return to Advanced REST Client; you should see only Delta flights to PDX are returned.

200 OK 40936.12 ms

[
 {
 "price": 958.0,
 "flightCode": "A1FGF4",
 "availableSeats": 80,
 "planeType": "Boing 777",
 "departureDate": "2015/02/13",
 "origination": "MUA",
 "airlineName": "Delta",
 "destination": "PDX"
 },

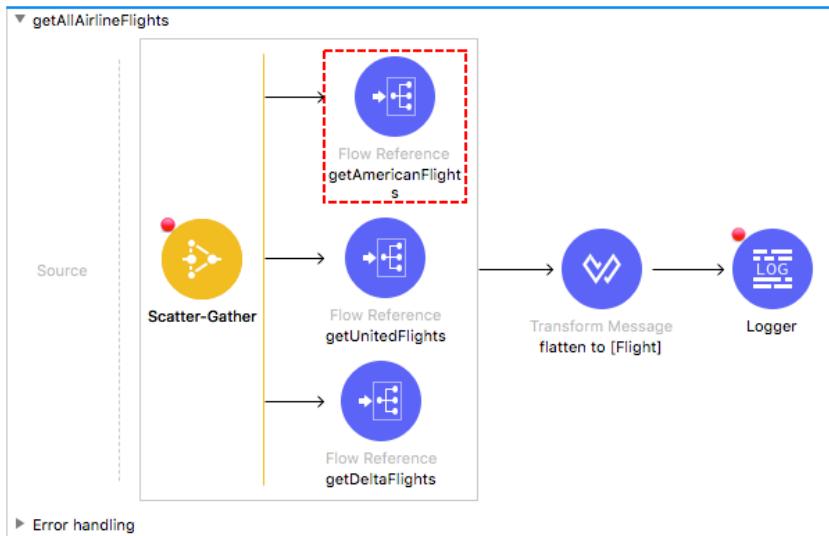
*Note: This JSON structure does not match that specified in the API: `mua-flights-api.raml` in `src/main/resources/api`. The data **should**, of course, be transformed so the response from the API matches this specification – but we are going to hold off doing that right now so that the scenario stays 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

50. Remove the airline parameter and set the code to FOO:

<http://localhost:8081/flights?code=FOO>.

51. In the Mule Debugger, step through the application; you should see multiple errors.



52. Return to Advanced REST Client; you should get a 500 Server Response and an exception.

Method Request URL

GET <http://localhost:8081/flights?code=FOO>

SEND :

Parameters ▾

500 Server Error 43387.29 ms DETAILS ▾

Exception(s) were found for route(s):

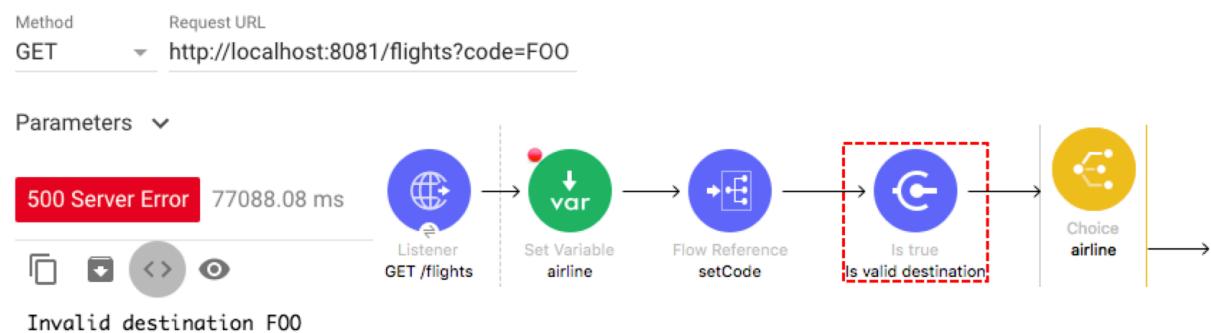
0: org.mule.extension.http.api.request.validator.ResponseValidatorTypedException
n: HTTP GET on resource '<http://training4-american-api-mule.cloudhub.io:80/flights>'
failed: bad request (400).

1: org.mule.runtime.core.api.expression.ExpressionRuntimeException: "You called
the function 'Value Selection' with these arguments."

Walkthrough 9-3: Validate events

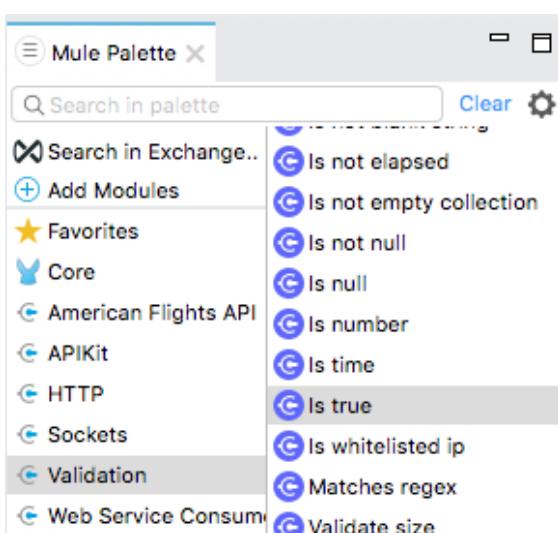
In this walkthrough, you use a validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request and to throw an error if it is not. You will:

- Add the Validation module to a project.
- Use an Is true validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request.
- Return a custom error message if the condition is not met.



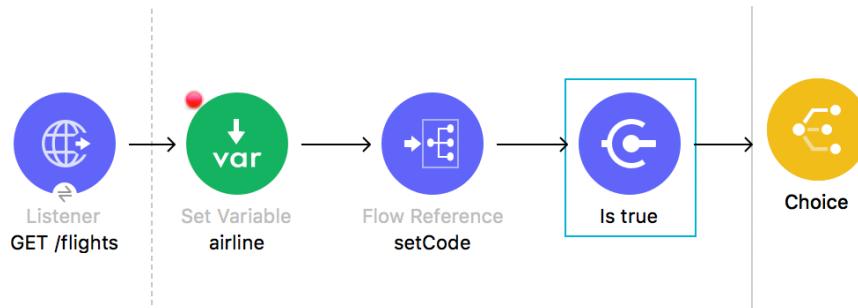
Add the Validation module to the project

1. Return to implementation.xml.
2. In the Mule Palette, select Add Modules.
3. Select the Validation module in the right side of the Mule Palette and drag and drop it into the left side.
4. If you get a Select module version dialog box, select the latest version and click Add.



Use the Is true validator to check for a valid destination code

5. Locate the Is true validator in the right side of the Mule Palette.
6. Drag and drop the Is true validator after the setCode Flow Reference in the getFlights flow.



7. In the Is true properties view, set the display name to: Is valid destination.
8. Change the expression from False (Default) to Expression.
9. Add a DataWeave expression to check if the code variable is one of the five destination codes.

```
#[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]
```

Note: You can copy this expression from the course snippets.txt file.

10. Set the error message to Invalid destination followed by the provided code.

```
#['Invalid destination' ++ ' ' ++ (vars.code default '')]
```

Note: You can copy this expression from the course snippets.txt file.

The screenshot shows the 'Is valid destination' component in the Mule Studio Properties view. The 'General' tab is selected. The 'Display Name' field is set to 'Is valid destination'. The 'Expression' field contains the DataWeave expression `#[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]`. The 'Message' field contains the error message `#['Invalid destination' ++ ' ' ++ (vars.code default '')]`.

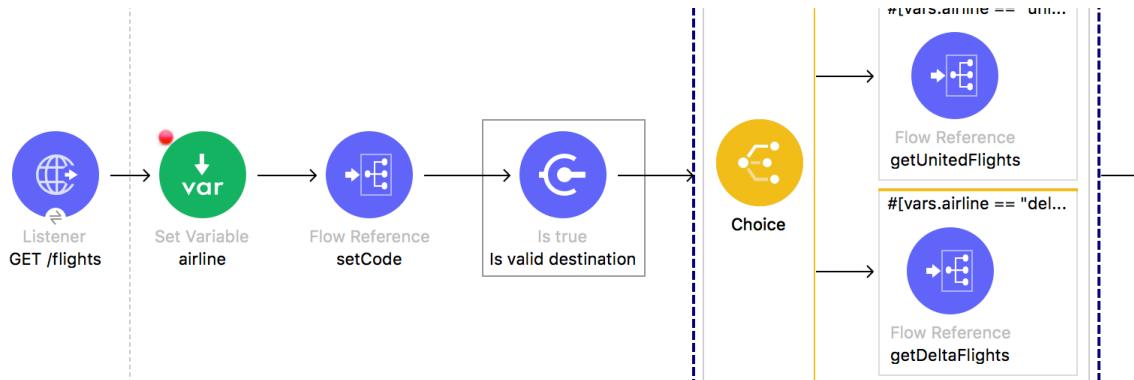
Debug the application

11. Save the file to redeploy the project in debug mode.

12. In Advanced REST Client, change the code to make a request to

<http://localhost:8081/flights?code=CLE>.

13. In the Mule Debugger, step past the validator; you should see the code is valid and application execution steps to the Choice router.



14. Resume through the rest of the application.

15. In Advanced REST Client, you should see flights.

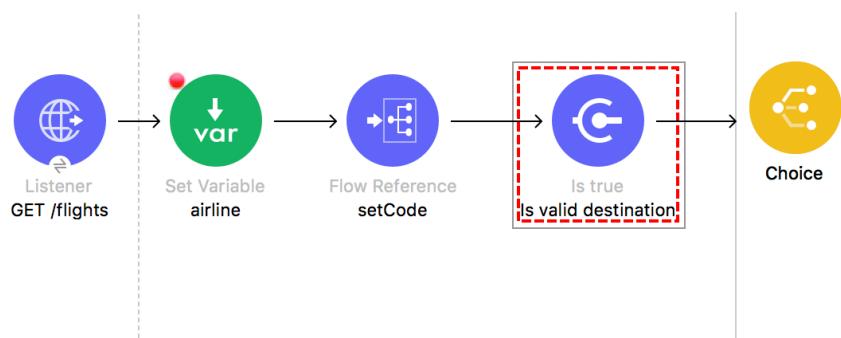
200 OK 65605.62 ms



```
[Array[8]
-0: {
  "airline": "American",
  "flightCode": "eefd0123",
  "fromAirportCode": "MUA",
  "toAirportCode": "CLE",
  "departureDate": "2016-01-25T00:00:00",
  "emptySeats": 7,
```

16. Change the code and make a request to <http://localhost:8081/flights?code=FOO>.

17. In the Mule Debugger, step to the validator; you should see an error.



18. Step again; you should see your Invalid destination message in the console.

```
ERROR 2018-04-20 11:55:44,836 [[MuleRuntime].cpuLight.15: [apdev-flights-ws].getFlights.CPU_LITE @65ffdff1] [event: 0-48  
bd68f0-44cc-11e8-a62a-8c85900da7e5] org.mule.runtime.core.internal.exception.OnErrorPropagateHandler:  
*****  
Message : Invalid destination FOO.  
Error type : VALIDATION:INVALID_BOOLEAN  
Element : getFlights/processors/2 @ apdev-flights-ws:implementation.xml:15 (Is valid destination)  
Element XML : <validation:is=true doc:name="Is valid destination" doc:id="e6d96ea1-71ee-45f1-9661-9048ce5382c9"  
" expression="#[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]" message="#['Invalid destination' ++ ' ' ++ vars.cod  
e]"></validation:is=true>
```

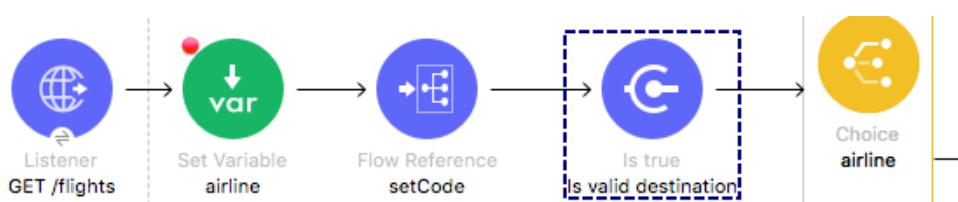
19. Resume through the rest of the application.

20. Return to Advanced REST Client; you should get a 500 Server Error with your Invalid destination message.

The screenshot shows the Advanced REST Client interface. At the top, there's a header with 'Method: GET' and 'Request URL: http://localhost:8081/flights?code=FOO'. Below the URL is a 'SEND' button and a more options menu. Underneath, there's a 'Parameters' dropdown set to 'GET'. The main content area shows a red box indicating a '500 Server Error' with a duration of '62965.80 ms'. To the right is a 'DETAILS' dropdown. Below the error message are several icons: a copy icon, a refresh icon, a link icon, and an eye icon. The error message itself is 'Invalid destination FOO'. At the bottom, there's a note: 'Note: You will catch this error and send a JSON response with a different status code in the next module.'

21. Remove the code and make a request to <http://localhost:8081/flights>.

22. In the Mule Debugger, step through the application; you should not get any errors.



23. Resume through the rest of the application.

24. Return to Advanced REST Client; you should get a 200 response with all airline flights to SFO.

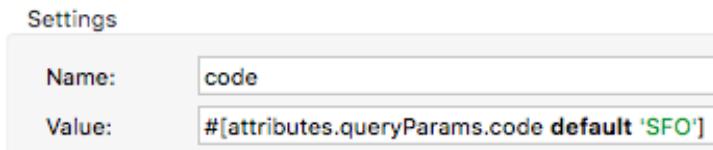
```
200 OK 9674.62 ms  
[  
 {  
   "price": 142.0,  
   "flightCode": "rree1093",  
   "availableSeats": 1,  
   "planeType": "Boeing 737",  
   "departureDate": "2016-02-11T00:00:00",  
   "origination": "MUA",  
   "airlineName": "American",  
   "destination": "SFO"  
 },
```

25. Return to Anypoint Studio and switch to the Mule Design perspective.

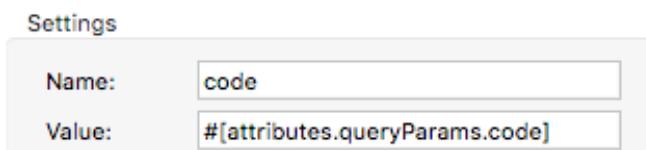
Remove the default destination

26. Locate the setCode subflow.

27. In the Set Variable properties view, review the default value.



28. Remove the default value from the value.

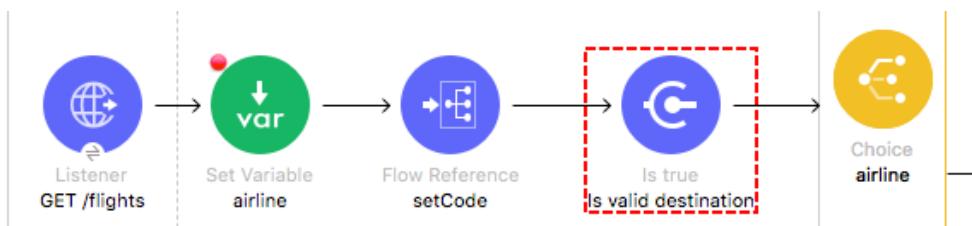


Debug the application

29. Save the file to redeploy the project.

30. In Advanced REST Client, make another request to <http://localhost:8081/flights>.

31. In the Mule Debugger, step to the validator; you should now get an error.



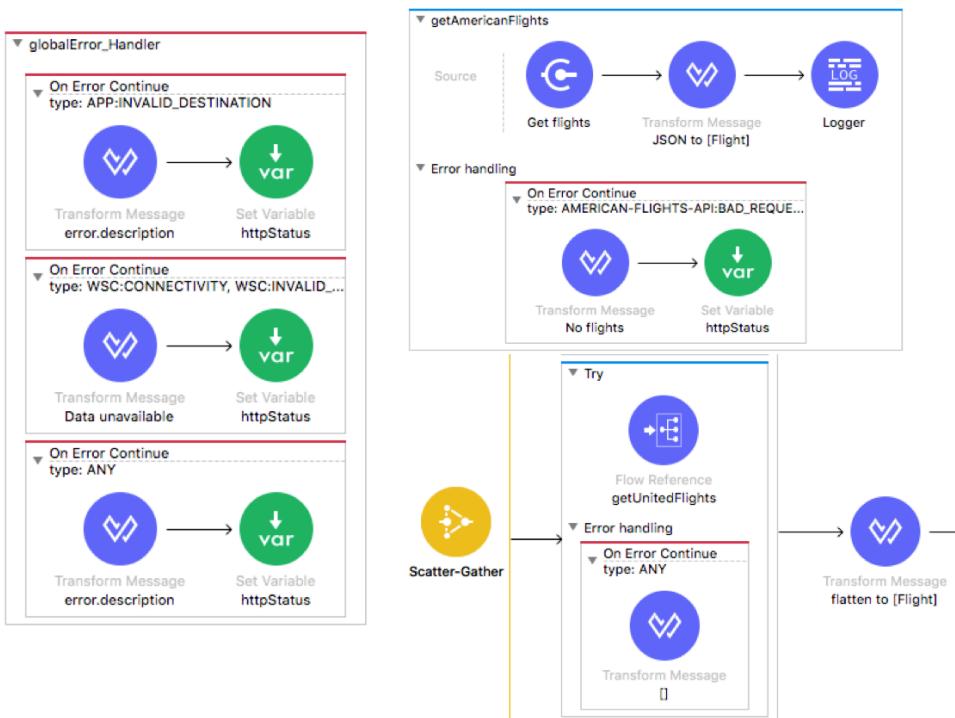
32. Resume through the rest of the application.
33. Return to Advanced REST Client; you should get a 500 Server Error with your Invalid destination message.

The screenshot shows the Advanced REST Client interface. At the top, there are fields for 'Method' (set to 'GET') and 'Request URL' (set to 'http://localhost:8081/flights'). To the right of these are a 'SEND' button and a three-dot menu icon. Below this, a 'Parameters' dropdown is shown. The main area displays an error response: a red box at the top says '500 Server Error' and '26591.16 ms'. To the right of this is a 'DETAILS' link. Below the error message are several small icons: a clipboard, a download arrow, a copy/paste icon, and a refresh/circular arrow icon. The text 'Invalid destination' is displayed below these icons.

Note: You will catch this error and send a JSON response with a different status code in the next module.

34. Return to Anypoint Studio and stop the project.

Module 10: Handling Errors



At the end of this module, you should be able to:

- Handle messaging errors at the application, flow, and processor level.
- Handle different types of errors, including custom errors.
- Use different error scopes to either handle an error and continue execution of the parent flow or propagate an error to the parent flow.
- Set the success and error response settings for an HTTP Listener.
- Set reconnection strategies for system errors.

Walkthrough 10-1: Explore default error handling

In this walkthrough, you get familiar with the three errors you will learn to handle this module. You will:

- Explore information about different types of errors in the Mule Debugger and the console.
- Review the default error handling behavior.
- Review and modify the error response settings for an HTTP Listener.

The screenshot shows two side-by-side interfaces. On the left is the 'Mule Debugger' showing a table of exception details. On the right is the 'APIKit Consoles' interface for a 'GET /flights' endpoint. The 'Responses' tab is selected in the APIKit interface, which displays the error response body as '#[output application/json --- error.errorType]'. Below it, the status code is set to 400 and the reason phrase is empty. The response headers section is also visible. The overall status message in the APIKit interface is 'There are no errors.'

Create a connectivity error

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open config.yaml.
3. Change the delta.wsdl property from /delta?wsdl to /deltas?wsdl.

```
delta:  
  wsdl: "http://mu.learn.mulesoft.com/deltas?wsdl"  
  service: "TicketServiceService"  
  port: "TicketServicePort"
```

4. Save the file.
5. Return to implementation.xml and debug the project.

Review a validation error in the Mule Debugger and console

6. In Advanced REST Client, add a code and make a request to
<http://localhost:8081/flights?code=FOO>.

7. In the Mule Debugger, step to where the error is thrown and expand the Exception object; you should see an error description, errorType, and other properties.

```

Mule Debugger X
VALIDATION:INVALID_BOOLEAN

▶ [e] attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ correlationId = "0-d924c630-bde0-11e8-b501-784f43835e3e"
  ▼ [e] error = {org.mule.runtime.core.internal.message.ErrorBuilder.ErrorImplementation} \norg.mule
    @ description = "Invalid destination FOO"
    @ detailedDescription = "Invalid destination FOO"
    ▶ [e] errorType = {org.mule.runtime.core.internal.message.ErrorTypeBuilder.ErrorTypeImplementation}
    ▶ [e] errors = {java.util.Collections.UnmodifiableRandomAccessList} []
    ▶ [e] exception = {org.mule.extension.validation.api.ValidationException} org.mule.extension.validation.ValidationException
    ▶ [e] muleMessage = {org.mule.runtime.core.internal.message.DefaultMessageBuilder.MessageBuilder}
    ▶ [e] payload =
    ▶ [e] vars = {java.util.Map} size = 1
  
```

8. Step again and review the error information logged in the console.

```

Console X Mule Properties
apdev-flights-wt10-1 [Mule Applications] Mule Server 4.1.1 EE
ERROR 2018-04-04 08:40:39,919 [[MuleRuntime].cpuLight.09: [apdev-flights-wt10-1].getFlights.CPU_LITE @244c11ce]
[event: 0-7f835d50-381e-11e8-8401-8c85900da7e5] org.mule.runtime.core.internal.exception.OnErrorPropagateHandler
:
*****
Message : Invalid destination FOO.
Error type : VALIDATION:INVALID_BOOLEAN
Element : getFlights/processors/2 @ apdev-flights-wt10-1:implementation.xml:16 (Is valid destination)
Element XML : <validation:is-true doc:name="Is valid destination" doc:id="e16b1467-395c-4020-8638-88dd0cfb705a" expression="#[['SFO','LAX','CLE','PDX','PDF']] contains vars.code]" message="#['Invalid destination' ++ ' ' ++ vars.code]"></validation:is-true>
(set debug level logging or '-Dmule.verbose.exceptions=true' for everything)
*****
```

9. Step through the rest of the application.

10. In Advanced REST Client, locate the response status code, status reason, and body; you should see a 500 status code, a status reason of Server Error, and a response body equal to the error description.

Method	Request URL
GET	http://localhost:8081/flights?airline=delta&code=FOO
SEND	

Parameters ▾

500 Server Error 12652.57 ms DETAILS ▾

Invalid destination FOO

Review a connectivity error in the Mule Debugger and console

11. Add an airline and change the code to make a request to

<http://localhost:8081/flights?airline=delta&code=PDX>.

12. In the Mule Debugger, step to where the error is thrown and review the Exception object.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". A table lists the exception details:

Name	Value
⑧ Encoding	UTF-8
▼ E Exception	
⑧ description	Error processing WSDL file [http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at 'http://mu.mulesoft-training.com/deltas?wsdl'.
⑧ detailedDescription	Error processing WSDL file [http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at 'http://mu.mulesoft-training.com/deltas?wsdl'.
▶ E errors	[]
E errorType	WSC:CONNECTIVITY
E exception	org.mule.runtime.api.connection.ConnectionException: Error processing WSDL fi...
⑧ muleMessage	null
⑧ Message	

13. Review the error information logged in the console.

14. Step through the rest of the application.

15. Return to Advanced REST Client; you should see a 500 Server Error with the error description.

The screenshot shows the Advanced REST Client interface. The request method is "GET" and the URL is "http://localhost:8081/flights?airline=delta&code=PDX". The response status is "500 Server Error" with a duration of "4450.12 ms". The error message is: "Error processing WSDL file [http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at 'http://mu.mulesoft-training.com/deltas?wsdl'.".

Review a bad request error in the Mule Debugger and console

16. Change the airline to make a request to

<http://localhost:8081/flights?airline=american&code=PDX>.

17. In the Mule Debugger, step to where the error is thrown and review the Exception object.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". A table lists the exception details, with the "error" field expanded to show its full stack trace:

▶ E attributes = {org.mule.extension.http.api.HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes	Request path=/flights
⑧ correlationId = "0-0aa6c00-bde1-11e8-b501-784f43835e3e"	
▼ E error = {org.mule.runtime.core.internal.message.ErrorBuilder\$ErrorImplementation} \norg.mule.runtime.core.internal.message.ErrorBuilder\$ErrorImplementation	
⑧ description = "org.mule.wsdl.parser.exception.WsdlParsingException-->Error processing WSDL file [http://mu.learn.mulesoft.com/deltas?wsdl]: Unable to locate document at 'http://mu.learn.mulesoft.com/deltas?wsdl'."	
⑧ detailedDescription = "org.mule.wsdl.parser.exception.WsdlParsingException-->Error processing WSDL file [http://mu.learn.mulesoft.com/deltas?wsdl]: Unable to locate document at 'http://mu.learn.mulesoft.com/deltas?wsdl'."	
▶ E errorType = {org.mule.runtime.core.internal.message.ErrorTypeBuilder\$ErrorTypeImplementation} WSC:CONNECTIVITY	
▶ E errors = {[java.util.Collections.UnmodifiableRandomAccessList] []}	
▶ E exception = {org.mule.runtime.api.connection.ConnectionException} org.mule.runtime.api.connection.ConnectionException: org.mule.wsdl.parser.exception.WsdlParsingException	
⑧ muleMessage = {org.mule.runtime.api.message.Message} null	
▶ E payload = <?xml version='1.0' encoding='UTF-8'?>\n<ns0:findFlight xmlns:ns0="http://soap.training.mulesoft.com/">\n <destination>PDX</destination>	
▶ E vars = {[java.util.Map] size = 2}	

18. Step through the rest of the application.
19. Review the error information logged in the console.
20. Return to Advanced REST Client; you should see a 500 Server Error with the error description.

The screenshot shows the Advanced REST Client interface. At the top, it displays the method as 'GET' and the request URL as 'http://localhost:8081/flights?airline=american&code=PDX'. Below the URL, there is a 'SEND' button and a more options menu. Underneath the URL, there is a 'Parameters' dropdown. The main content area shows a red box around the status code '500 Server Error' and the time '1980.21 ms'. To the right of this, there is a 'DETAILS' dropdown. Below the status code, there are several icons: a clipboard, a download arrow, a refresh arrow, and an eye icon. The detailed error message reads: 'HTTP GET on resource \'<http://training4-american-api.cloudhub.io:80/flights>\' failed: bad request (400).'

21. Return to Anypoint Studio and switch to the Mule Design perspective.
22. Stop the project.

Review the default error response settings for an HTTP Listener

23. Navigate to the properties view for the GET /flights Listener in getFlights.
24. Select the Responses tab.
25. Locate the Error Response section and review the default settings for the body, status code, and reason phrase.

The screenshot shows the Anypoint Studio properties view for a 'GET /flights' listener. The left sidebar lists tabs: General, MIME Type, Redelivery, **Responses**, Advanced, Metadata, and Notes. The 'Responses' tab is selected. In the main panel, there is a message 'There are no errors.' A green checkmark icon is present. Below this, the 'Error Response' section is expanded. It contains a 'Body:' field with the value '#[output text/plain --- error.description]'. Below the body field is a 'Headers' section with a table having columns 'Name' and 'Value'. There are three rows in the table, each with a plus sign icon to add more. At the bottom of the error response section are fields for 'Status code:' and 'Reason phrase:'.

Remove the default error response settings for the HTTP Listener

26. Select and cut the body expression (so it has no value, but you can paste it back later).



Test the application

27. Save the file and run the project.
28. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=PDX>; you should get the same response.

A screenshot of the Advanced REST Client. At the top, it shows 'Method: GET' and 'Request URL: http://localhost:8081/flights?airline=american&code=PDX'. Below that is a 'Parameters' dropdown. The main area shows a red box around '500 Server Error' and '1980.21 ms'. To the right is a 'DETAILS' button. Below the error message are icons for copy, download, and refresh. The error message itself says: 'HTTP GET on resource \'<http://training4-american-api.cloudhub.io:80/flights>\' failed: bad request (400)'.

Modify the default error response settings for the HTTP Listener

29. Return to Anypoint Studio.
30. Return to the Responses tab in the properties view for the GET /flights Listener.
31. In the error response body, paste back the original value.

```
##[output text/plain --- error.description]
```

32. Change the output type of the error response to application/json and display the error.errorType.

```
##[output application/json --- error.errorType]
```

33. Set the error response status code to 400.

The screenshot shows the APIkit Consoles interface for a 'GET /flights' endpoint. On the left, a sidebar lists 'General', 'MIME Type', 'Redelivery', 'Responses' (which is selected), 'Advanced', 'Metadata', and 'Notes'. The main area displays an 'Error Response' configuration. It includes a 'Body' section with the code '#[output application/json --- error.errorType]', a 'Headers' section with a table for adding headers (with three icons: plus, minus, and gear), and fields for 'Status code' (set to 400) and 'Reason phrase' (empty). A green checkmark indicates 'There are no errors.'

Test the application

34. Save the file to redeploy the application.
35. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=PDX>; you should get the new status code and reason and the response body should be the errorType object.
36. Look at the error namespace and identifier.

The screenshot shows the Advanced REST Client interface. At the top, it displays 'Method: GET' and 'Request URL: http://localhost:8081/flights?airline=american&code=PDX'. Below this is a 'Parameters' dropdown. The response section shows a red box around '400 Bad Request' and '620.16 ms'. To the right is a 'DETAILS' button. Below the response text, there are several icons: a clipboard, a download arrow, a comparison icon, a refresh icon, and a copy icon. The response body is a JSON object:

```
{  
  "parentErrorType": {  
    "parentErrorType": null,  
    "namespace": "MULE",  
    "identifier": "ANY"  
  },  
  "namespace": "AMERICAN-FLIGHTS-API",  
  "identifier": "BAD_REQUEST"  
}
```

37. Change the airline to make a request to <http://localhost:8081/flights?airline=delta&code=PDX>.

38. Look at the error namespace and identifier.

Method: GET Request URL: http://localhost:8081/flights?airline=delta&code=PDX

Parameters: { }

400 Bad Request 489.53 ms

DETAILS ▾

```
{ "parentErrorType": { "parentErrorType": { "parentErrorType": null, "namespace": "MULE", "identifier": "ANY" }, "namespace": "MULE", "identifier": "CONNECTIVITY" }, "namespace": "WSC", "identifier": "CONNECTIVITY" }
```

39. Change the code to make a request to <http://localhost:8081/flights?airline=delta&code=FOO>.

40. Look at the namespace and identifier.

Method: GET Request URL: http://localhost:8081/flights?airline=delta&code=FOO

Parameters: { }

400 Bad Request 283.39 ms

DETAILS ▾

```
{ "parentErrorType": { "parentErrorType": { "parentErrorType": { "parentErrorType": null, "namespace": "MULE", "identifier": "ANY" }, "namespace": "MULE", "identifier": "VALIDATION" }, "namespace": "VALIDATION", "identifier": "VALIDATION" }, "namespace": "VALIDATION", "identifier": "INVALID_BOOLEAN" }
```

Return the default error response settings for the HTTP Listener

41. Return to Anypoint Studio.

42. Return to the Responses tab in the properties view for the GET /flights Listener.

43. Change the error response output type back to text/plain and display the error.description.

```
##[output text/plain --- error.description]
```

44. Remove the status code.

The screenshot shows the Anypoint Studio interface for configuring a REST API endpoint. The left sidebar lists tabs: General, MIME Type, Redelivery, Responses (which is selected), Advanced, Metadata, and Notes. The main panel shows a configuration for a 'GET /flights' endpoint. In the 'Responses' tab, under 'Error Response', the 'Body' field contains the code '##[output text/plain --- error.description]'. The 'Headers' section has three icons (+, X, <>) and a table with columns 'Name' and 'Value', both currently empty. Below that, 'Status code:' and 'Reason phrase:' fields are also empty. A message at the top says 'There are no errors.'

Test the application

45. Save the file to redeploy the application.

46. In Advanced REST Client, change the code to make a request to

<http://localhost:8081/flights?airline=american&code=PDX>; you should get the 500 Server Error again with the plain text error description.

The screenshot shows the Advanced REST Client interface. At the top, it displays 'Method: GET' and 'Request URL: http://localhost:8081/flights?airline=american&code=PDX'. Below that is a 'Parameters' dropdown. On the right, there are 'SEND' and more options buttons. The main area shows a red box indicating a '500 Server Error' with a duration of '6009.75 ms'. Below the error message are several small icons: a square, a triangle, a double arrow, and a magnifying glass. The error message itself reads: 'HTTP GET on resource \'<http://training4-american-api.cloudhub.io:80/flights>\' failed: bad request (400)'.

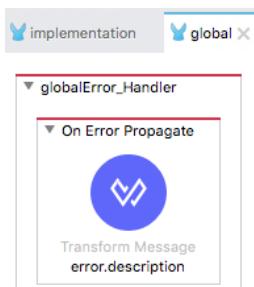
47. Return to Anypoint Studio.

48. Stop the project.

Walkthrough 10-2: Handle errors at the application level

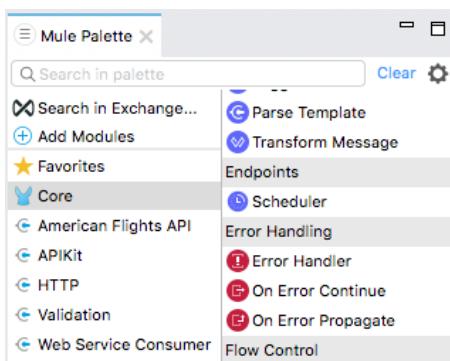
In this walkthrough, you create a default error handler for the application. You will:

- Create a global error handler in an application.
- Configure an application to use a global default error handler.
- Explore the differences between the On Error Continue and On Error Propagate scopes.
- Modify the default error response settings for an HTTP Listener.



Browse the error handling elements in the Mule Palette

1. Return to global.xml and switch to the Message Flow view.
2. In the Core section of the Mule Palette, locate the Error Handling elements.

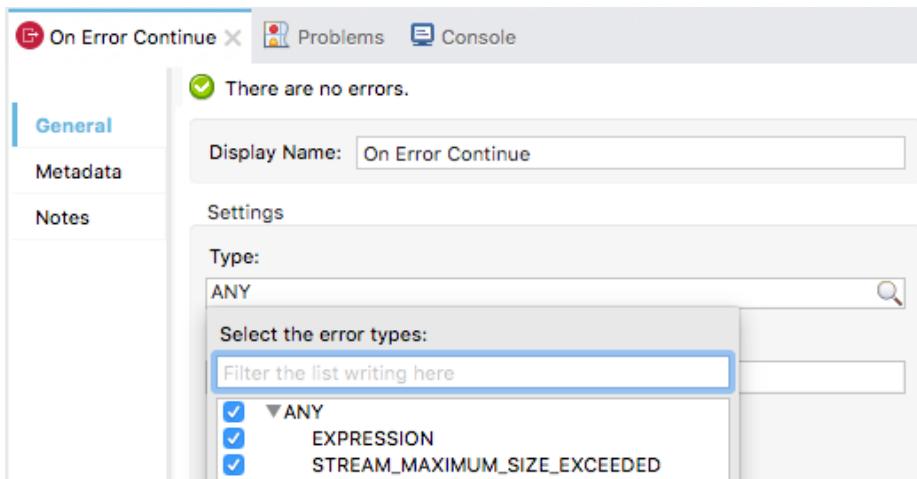


Create a global error handler with an On Error Continue scope

3. Drag out an Error Handler element and drop it in the canvas of global.xml.
4. Drag out an On Error Continue element and drop it in globalErrorHandler.



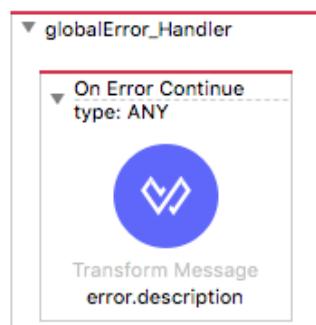
- In the On Error Continue properties view, click the Search button for type.
- In the drop-down menu that appears, select ANY.



Note: If the drop-down menu does not populate, type ANY into the text field.

Set the payload in the error handler to a JSON message

- Add a Transform Message component to the On Error Continue.
- Set the Transform Message display name to error.description.



- In the Transform Message properties view, change the output type to application/json.
- Add a message property to the output JSON and give it a value of the error.description.

```

Output Payload ▾ + 🖊 🗑️ IN Preview
1 %dw 2.0
2 output application/json
3 ---
4 [
5   "message": error.description
6 ]

```

The screenshot shows the 'Payload' tab in the Transform Message properties view. It displays the MEL code for the output transformation:

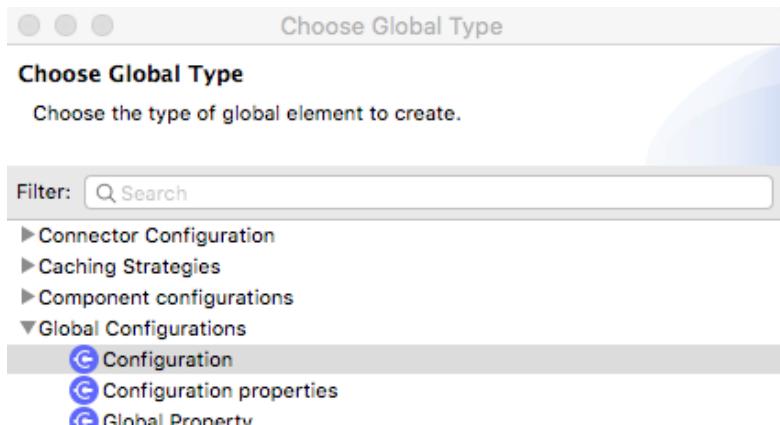
```

1 %dw 2.0
2 output application/json
3 ---
4 [
5   "message": error.description
6 ]

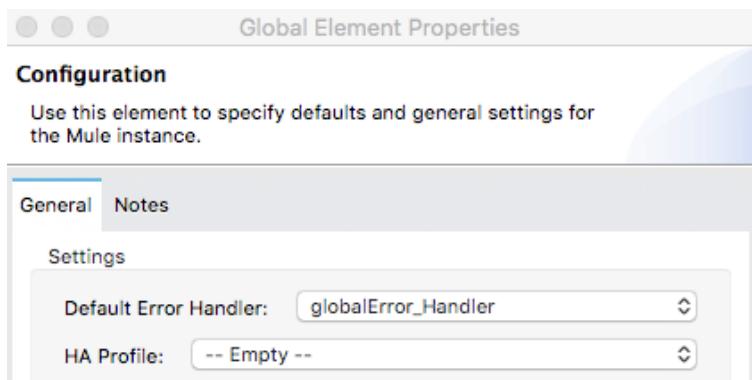
```

Set a default error handler for the application

11. Switch to the Global Elements view of global.xml.
12. Click Create.
13. In the Choose Global Type dialog box, select Global Configurations > Configuration and click OK.



14. In the Global Element Properties dialog box, set the default error handler to globalError_Handler and click OK.



15. Switch to the Message Flow view.

Review the default response settings for an HTTP Listener

16. Return to implementation.xml.
17. Navigate to the properties view for the GET /flights Listener in getFlights.
18. Select the Responses tab.

19. Locate the Response section (not the Error Response section) and review the default settings for the body, status code, and reason phrase.

The screenshot shows the Anypoint Studio interface for a REST API endpoint named 'GET /flights'. The left sidebar has tabs for General, MIME Type, Redelivery, **Responses**, Advanced, Metadata, and Notes. The 'Responses' tab is selected. The main area shows the 'Response' configuration. It includes a message box stating 'There are no errors.' A 'Body' field contains the placeholder '#[payload]'. Below it is a 'Headers' section with a table for adding header pairs. Underneath are fields for 'Status code:' and 'Reason phrase:'.

20. Locate the Error Response section and review the default settings for the body, status code, and reason phrase.

Test the On Error Continue behavior

21. Save all files and debug the project.
22. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=PDX>.
23. In the Mule Debugger, step through the application until the event is passed to globalError_Handler.

Note: In Anypoint Studio 7.1.X, focus will switch to global.xml but you will not see the application execution step through an individual error handler.

24. In the Mule Debugger, locate and expand Exception.

The screenshot shows the Mule Debugger interface. At the top, there's a table with columns 'Name' and 'Value'. An expanded entry under 'Exception' shows the following details:

Name	Value
encoding	UTF-8
Exception	
@description	HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400).
@detailedDescription	HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400).
@errors	[]
@errorType	AMERICAN-FLIGHTS-API:BAD_REQUEST
@exception	org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET on resour...
@muleMessage	

Below this, a message box displays the full exception message: "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."

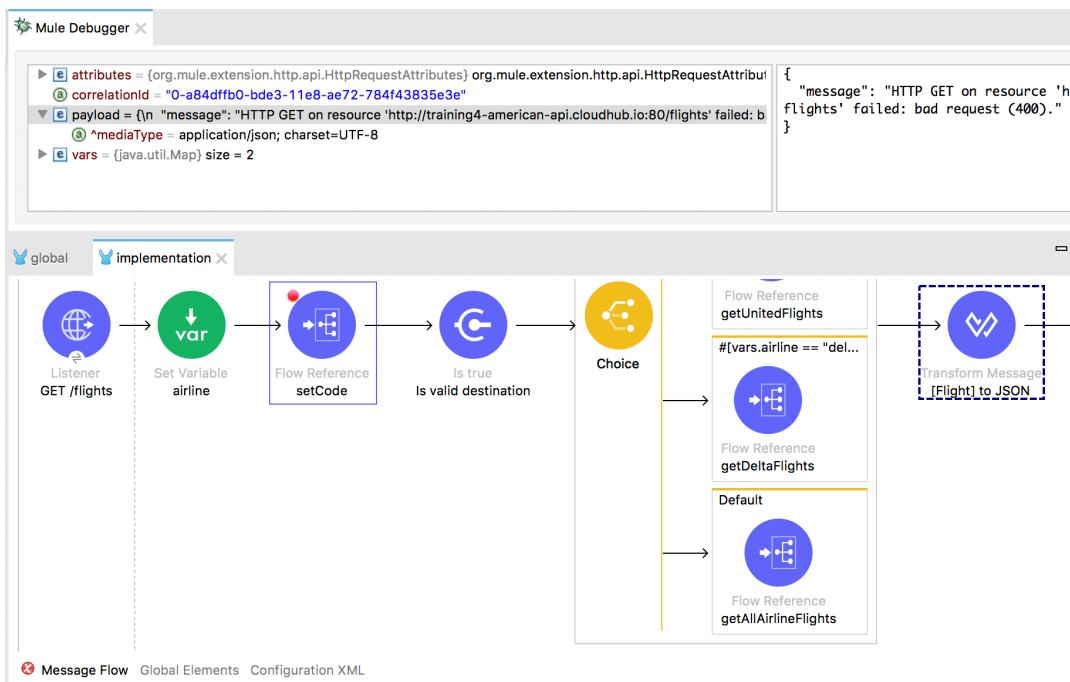
At the bottom, there are tabs for 'global' and 'implementation', with 'global' selected. A red box highlights a section of the configuration:

```
globalError_Handler
  On Error Continue
    type: ANY
      Transform Message
        error.description
```

This section contains a 'Transform Message' component with the expression 'error.description'.

25. Step again; the event should be passed back to getFlights, which continues to execute after the error occurs.

26. In the Mule Debugger, look at the payload and the absence of an exception object in the Mule Debugger.



27. Step through the rest of the application.

28. Return to Advanced REST Client; you should see get a 200 OK response with the response body equal to the payload value set in the error handler.

The screenshot shows the Advanced REST Client interface. It has the following fields:

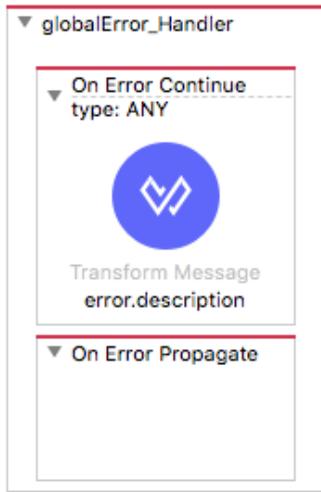
- Method: GET
- Request URL: http://localhost:8081/flights?airline=american&code=PDX
- Parameters dropdown: Parameters ▾
- Status: 200 OK 1813.83 ms
- Details: DETAILS ▾

The response body is displayed below the status:

```
{ "message": "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."}  
vars = {size = 2}
```

Change the default error handler to use an On Error Propagate scope

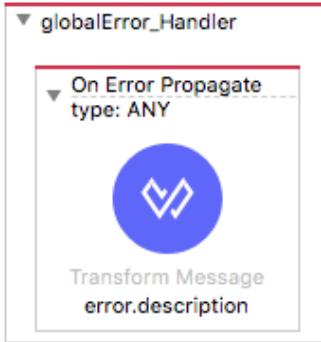
29. Return to Anypoint Studio and switch to the Mule Design perspective.
30. Return to global.xml.
31. Drag an On Error Propagate element from the Mule Palette and drop it to the right or left of the On Error Continue to add it to globalError_Handler.



32. In the On Error Propagate properties view, set the type to ANY.

Note: If the drop-down menu does not populate, type ANY into the text field.

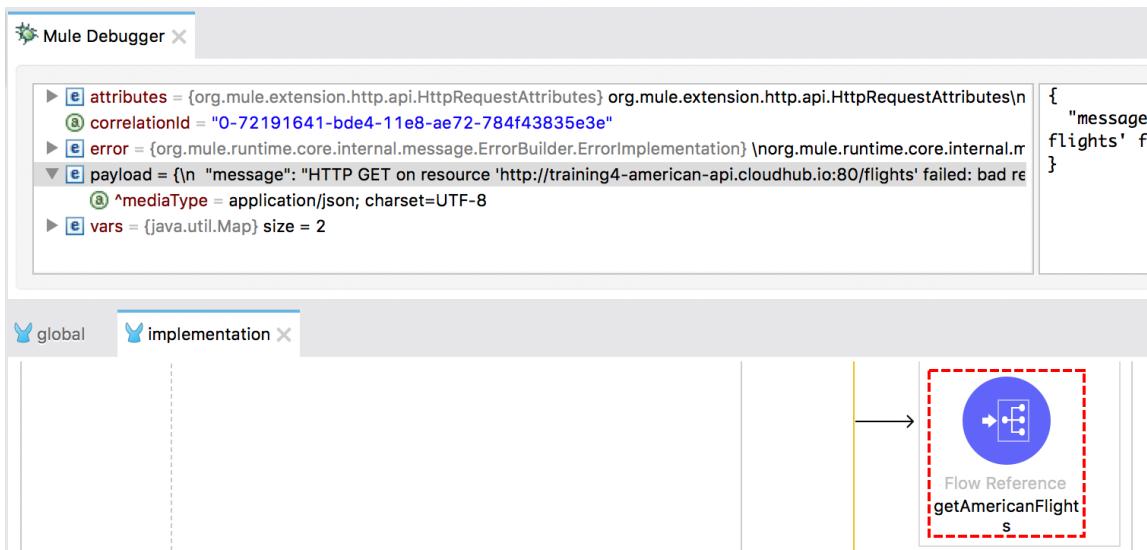
33. Move the Transform Message component from the On Error Continue to the On Error Propagate.
34. Delete the On Error Continue scope.



Test the On Error Propagate behavior

35. Save the files to redeploy the project.
36. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=PDX>.
37. Step through the application until the event is passed to globalError_Handler.
38. Look at the payload and the exception in the Mule Debugger; they should be the same as before.
39. Step again; the error is propagated up to the getFlights parent flow.

40. Look at the payload and the exception in the Mule Debugger.



41. Step again; the error is handled by the application's default error handler again.

42. Look at the payload and the exception in the Mule Debugger.

43. Step through the rest of the application.

44. Return to Advanced REST Client; you should see get a 500 Server Error response with the response body equal to the plain text error description – not the payload.

The screenshot shows the Advanced REST Client interface. At the top, it displays the method (GET), request URL (http://localhost:8081/flights?airline=american&code=PDX), and a 'SEND' button. Below that is a 'Parameters' dropdown.

The main area shows a red box around the status '500 Server Error' and the response time '11373.79 ms'. To the right is a 'DETAILS' button. Below the status are several icons: a copy icon, a refresh icon, a comparison icon, and a search icon.

The response body is displayed below the status:

```
HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400).
```

45. Return to Anypoint Studio and switch to the Mule Design perspective.

46. Stop the project.

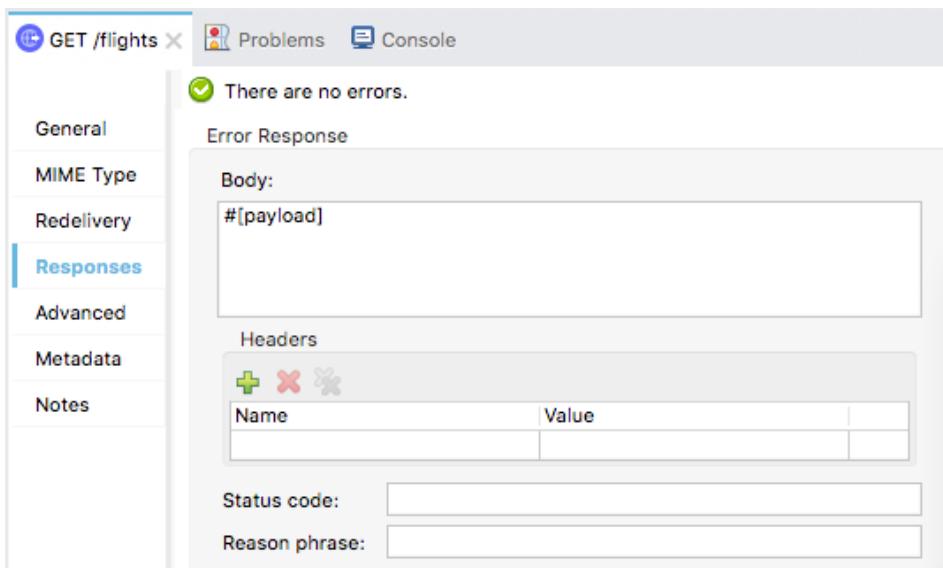
Modify the default error response settings for the HTTP Listener

47. Return to implementation.xml.

48. Navigate to the Responses tab in the properties view for the GET /flights Listener.

49. Change the error response body to return the payload instead of error.description.

```
##[payload]
```

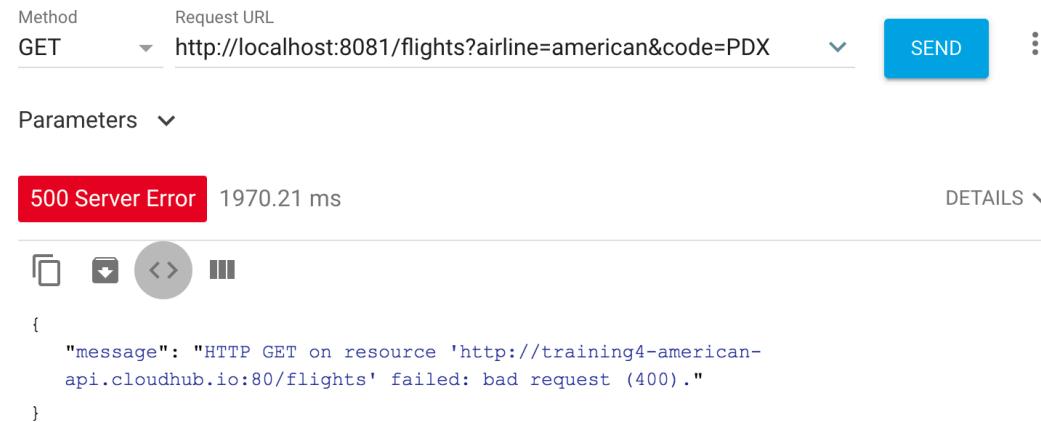


Test the application

50. Save the file and run the project.

51. In Advanced REST Client, make another request to

<http://localhost:8081/flights?airline=american&code=PDX>; you should now get the message that you set in the error handler.



Method	Request URL
GET	http://localhost:8081/flights?airline=american&code=PDX
SEND	

Parameters ▾

500 Server Error 1970.21 ms DETAILS ▾

{
 "message": "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."
}

Note: You will handle this error differently in a later walkthrough, so it does not return a server error. It is a valid request; there are just no American flights to PDX.

52. Change the airline to make a request to <http://localhost:8081/flights?airline=delta&code=PDX>; the error is handled by the same default handler.

The screenshot shows a web interface for sending a request. At the top, there are fields for 'Method' (set to 'GET') and 'Request URL' (set to 'http://localhost:8081/flights?airline=delta&code=PDX'). Below these are buttons for 'SEND' and a more options menu. A 'Parameters' dropdown is also present. The main content area displays a red box indicating a '500 Server Error' with a response time of '528.72 ms'. To the right of the error message is a 'DETAILS' button. Below the error message, there are several small icons: a square, a downward arrow, a circular arrow, and a vertical ellipsis. The detailed error message is shown in a code block:

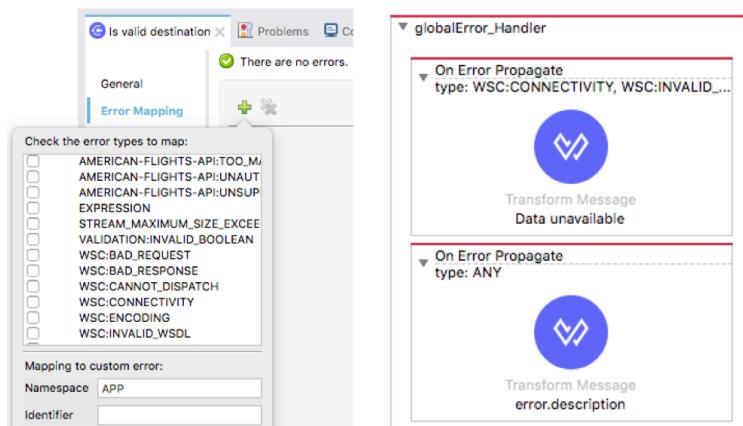
```
{  
  "message": "Error processing WSDL file [http://mu.mulesoft-training.com/deltas?wsdl]:  
  Unable to locate document at 'http://mu.mulesoft-training.com/deltas?wsdl'."  
}
```

53. Return to Anypoint Studio.

Walkthrough 10-3: Handle specific types of errors

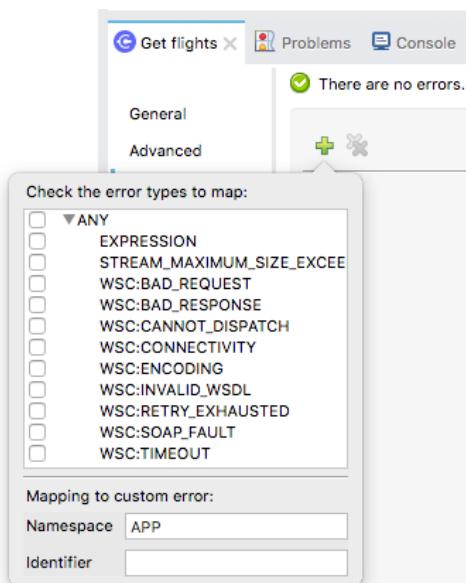
In this walkthrough, you continue to work with the application's default error handler. You will:

- Review the possible types of errors thrown by different processors.
- Create error handler scopes to handle different error types.



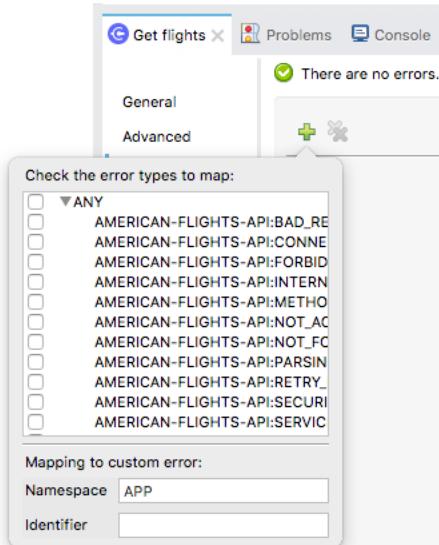
Review the possible types of errors thrown by a Web Service Consumer operation

1. Return to implementation.xml.
2. Navigate to the properties view for the Get flights Consume operation in getDeltaFlights.
3. Select the Error Mapping tab.
4. Click the Add new mapping button and review (but don't select!) the WSC error types.



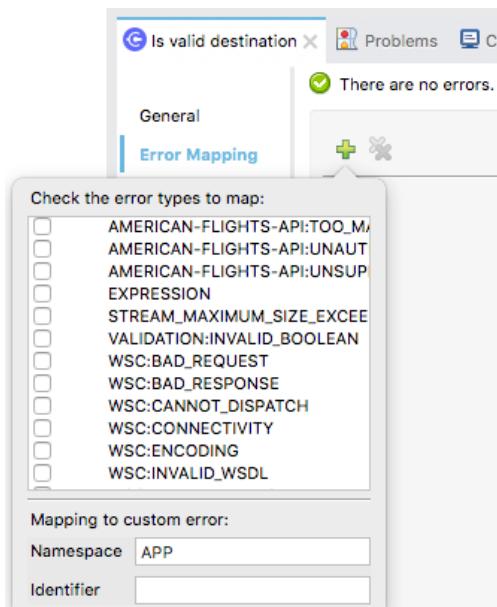
Review the possible types of errors thrown by a REST Connector operator

5. Navigate to the properties view for the Get flights operation in getAmericanFlights.
6. Select the Error Mapping tab.
7. Click the Add new mapping button, and review (but don't select!) the AMERICAN-FLIGHTS-API error types.



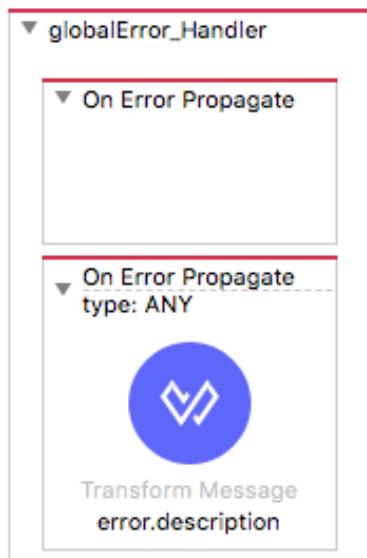
Review the possible types of errors thrown by a Validator operation

8. Navigate to the properties view for the Is true validator in getFlights.
9. Select the Error Mapping tab.
10. Click the Add new mapping button and locate (but don't select!) the VALIDATION error type.

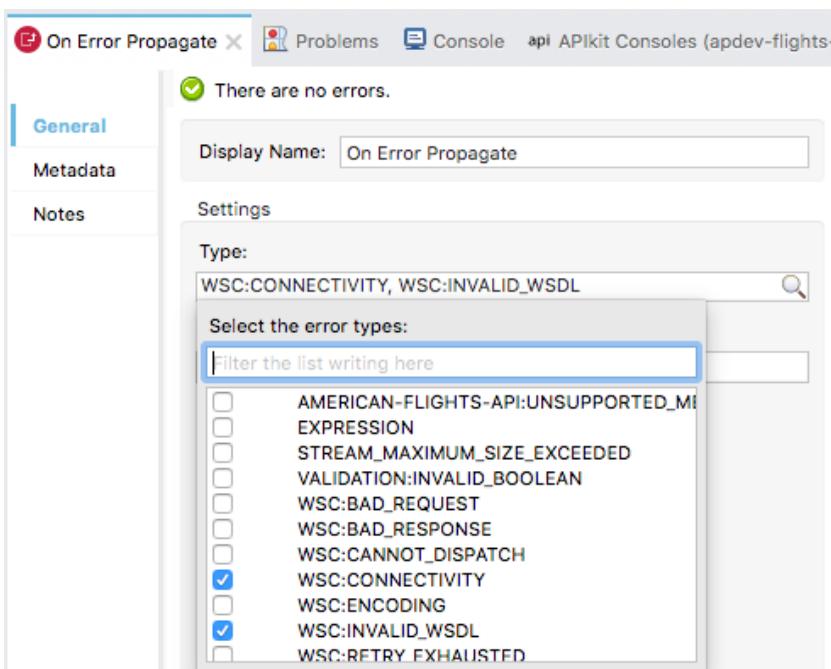


Add a second error handler to catch Web Service Consumer connectivity errors

11. Return to global.xml.
12. Add a second On Error Propagate to globalError_Handler.
13. If necessary, drag and drop it so it is first scope inside the error handler.

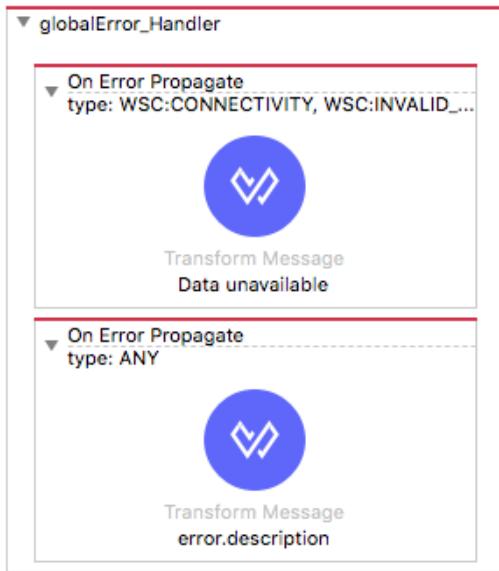


14. In the properties view for the new On Error Propagate, click the Search button.
15. Select WSC:CONNECTIVITY and WSC:INVALID_WSDL.



Note: If the drop-down menu does not show up, type this information instead.

16. Add a Transform Message component to the new On Error Propagate and set its display name to Data unavailable.



17. In the Transform Message properties view, change the output type to application/json.
18. Add a message property to the output JSON and set give it a value of "Data unavailable. Try later.".
19. For development purposes, concatenate the error.description to the message; be sure to use the auto-completion menu.

```
Output Payload ▾ + 🖊️ 🗑️
```

```
1%dw 2.0
2output application/json
3---
4{
5  "message": "Data unavailable. Try later. " ++ error.description
6 }
```

Preview

Test the application

20. Save the file to redeploy the project.

21. In Advanced REST Client, make another request to <http://localhost:8081/flights?airline=delta&code=PDX>; you should now get a 500 Server Error response with your new Data unavailable message.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'GET' and the 'Request URL' is 'http://localhost:8081/flights?airline=delta&code=PDX'. Below the URL is a 'Parameters' dropdown. The response status is '500 Server Error' with a duration of '869.73 ms'. The response body is a JSON object:

```
{ "message": "Data unavailable. Try later. Error processing WSDL file [http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at 'http://mu.mulesoft-training.com/deltas?wsdl'." }
```

22. Change the code to make a request to <http://localhost:8081/flights?airline=delta&code=FOO>; you should still get a 500 Server Error response with the Invalid destination JSON message.

Note: This should also not be a server error, but either a 4XX bad request or a 200 OK with an appropriate message. You will handle this error differently in the next walkthrough.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'GET' and the 'Request URL' is 'http://localhost:8081/flights?airline=delta&code=FOO'. Below the URL is a 'Parameters' dropdown. The response status is '500 Server Error' with a duration of '113.11 ms'. The response body is a JSON object:

```
{ "message": "Invalid destination FOO" }
```

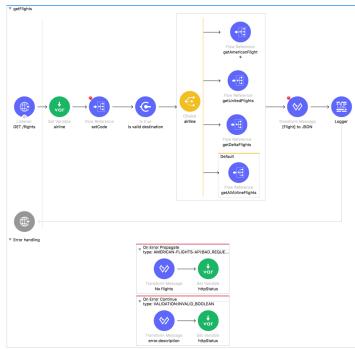
23. Return to Anypoint Studio.

24. Stop the project.

Walkthrough 10-4: Handle errors at the flow level

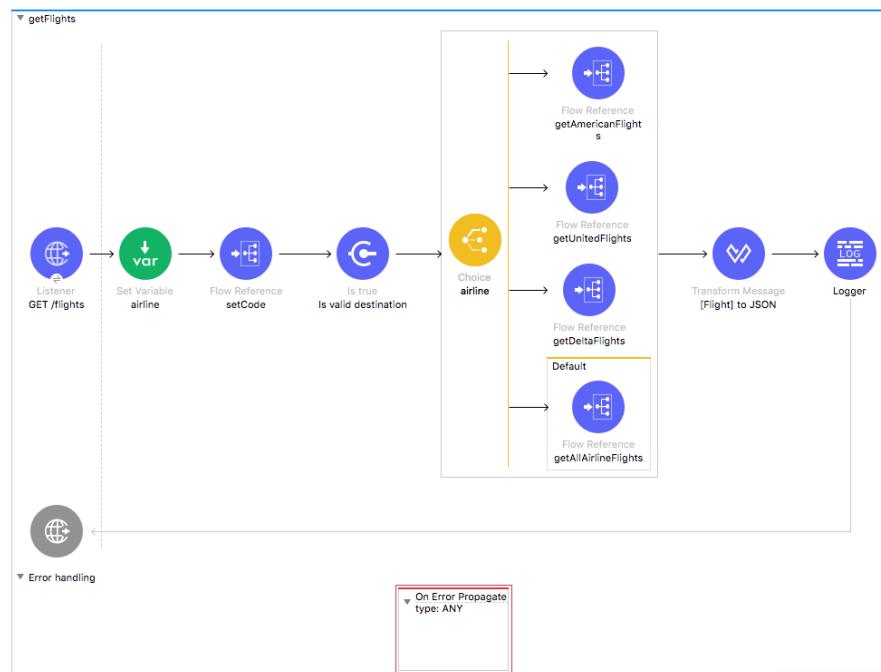
In this walkthrough, you add multiple error handlers to a flow. You will:

- Add error handlers to a flow.
- Test the behavior of errors thrown in a flow and by a child flow.
- Compare On Error Propagate and On Error Continue scopes in a flow.
- Set an HTTP status code in an error handler and modify an HTTP Listener to return it.



Add an On Error Propagate error handler to a flow

1. Return to implementation.xml.
2. Expand the Error handling section of the getFlights flow.
3. Add an On Error Propagate scope.
4. Set the error type to ANY so it will initially catch both the validation and American flights errors.



Set the payload in the error handler to the error description

5. Add a Transform Message component to the scope and set the display name to No flights.
6. In the Transform Message properties view, set the payload to a JSON message property equal to the string No flights to and concatenate the code variable.
7. Coerce the variable to a String.

```
1@ %dw 2.0
2   output application/json
3   ---
4@ []
5     "message": "No flights to " ++ vars.code as String
6 }
```

Set the HTTP status code in the error handler to 200

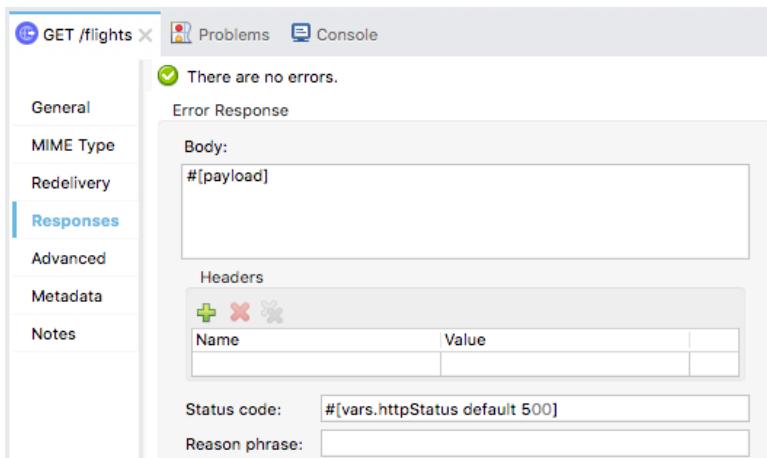
8. Add a Set Variable transformer to the On Error Propagate.
9. In the Set Variable properties view, set the display name and name to httpStatus.
10. Set the value to 200.



Set the HTTP Listener error response status code to use a variable

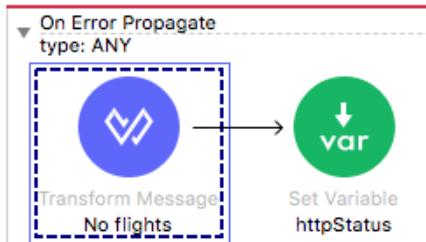
11. Navigate to the Responses tab in the properties view for the GET /flights Listener.
12. Change the error response status code (not the response status code!) to an expression that sets it equal to the value of an httpStatus variable with a default value of 500.

```
# [vars.httpStatus default 500]
```



Test the flow's On Error Propagate with an error in the flow

13. Save the file and debug the project.
14. In Advanced REST Client, change the airline to make a request to
<http://localhost:8081/flights?airline=american&code=FOO>.
15. In the Mule Debugger, step until the event is passed to the flow's error handler.



16. Step through the rest of the application.
17. Return to Advanced REST Client; you should get the 200 status code and the JSON message.

```
200 OK 81289.88 ms

{
  "message": "No flights to FOO"
}
```

Test the flow's On Error Propagate with an error in a child flow

18. In Advanced REST Client, change the code to make a request to
<http://localhost:8081/flights?airline=american&code=PDX>.

19. Step until the event is passed to the globalError_Handler.

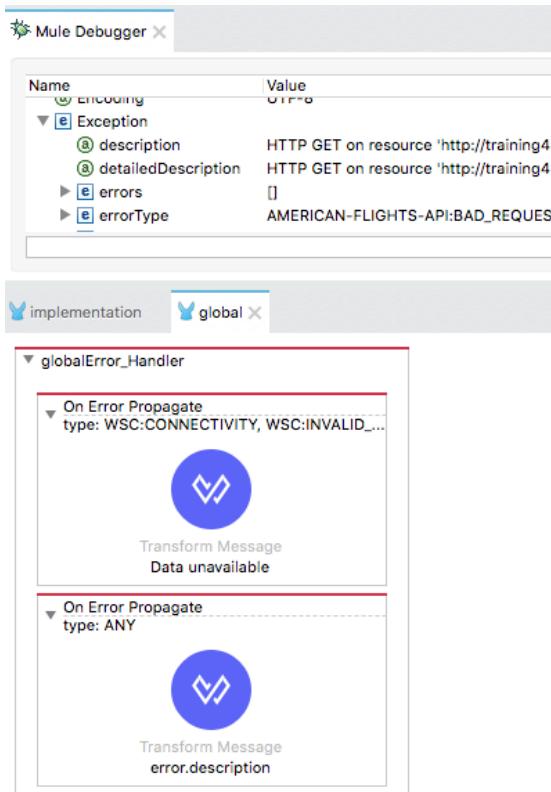
Mule Debugger

Name	Value
Encoding	UTF-8
Exception	<ul style="list-style-type: none">⑧ description: HTTP GET on resource 'http://training4-a...⑧ detailedDescription: HTTP GET on resource 'http://training4-a...► ⑨ errors: []► ⑨ errorType: AMERICAN-FLIGHTS-API:BAD_REQUEST

implementation global

globalError_Handler

- On Error Propagate type: WSC:CONNECTIVITY, WSC:INVALID_...
- Transform Message Data unavailable
- On Error Propagate type: ANY
- Transform Message error.description



20. Step until the event is passed to the parent flow's error handler.

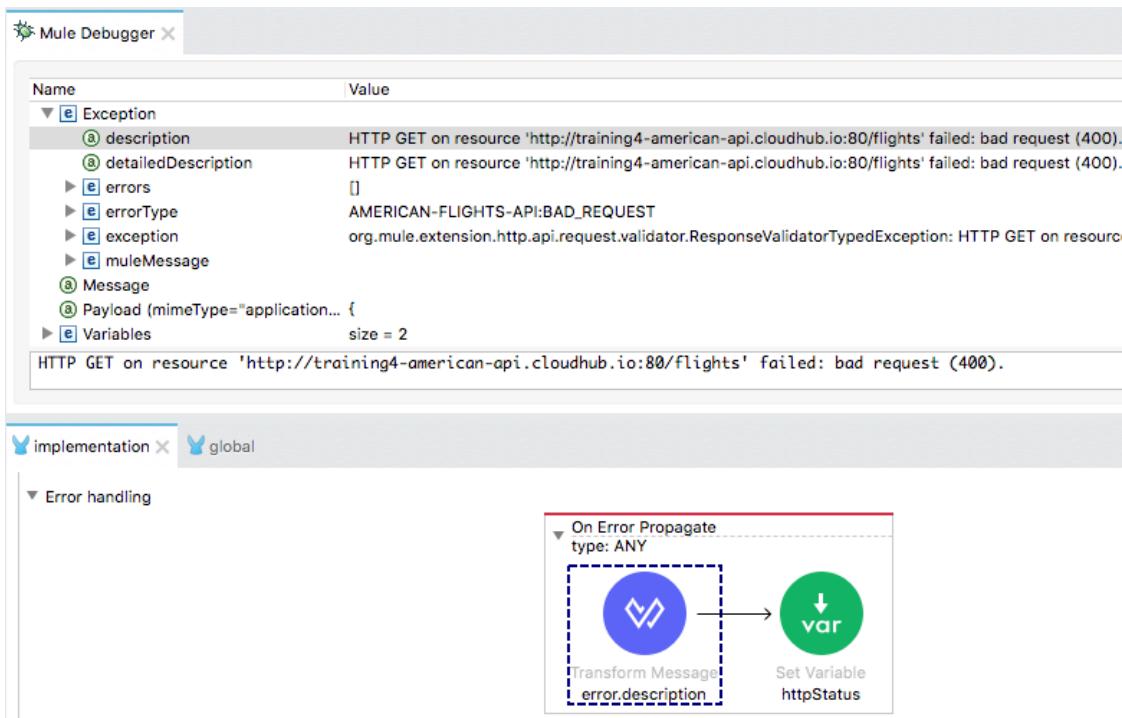
Mule Debugger

Name	Value
Exception	<ul style="list-style-type: none">⑧ description: HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400).⑧ detailedDescription: HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400).► ⑨ errors: []► ⑨ errorType: AMERICAN-FLIGHTS-API:BAD_REQUEST► ⑨ exception: org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET on resource...► ⑨ muleMessage:
Message	
Payload (mimeType="application..." {	size = 2
Variables	

implementation global

Error handling

- On Error Propagate type: ANY
- Transform Message error.description
- Set Variable httpStatus



21. Step through the rest of the application.

22. Return to Advanced REST Client; you should get the 200 status code and the JSON error message.

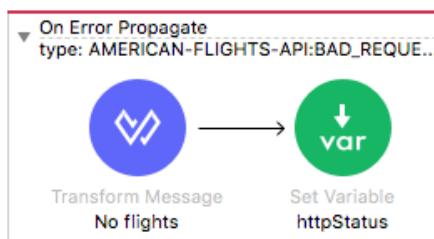
```
200 OK 109017.67 ms

{
  "message": "No flights to PDX"
}
```

23. Return to Anypoint Studio and switch to the Mule design perspective.

Specify the type of error to be handled by the flow's error handler

24. Navigate to the properties view for the On Error Propagate in getFlights.
25. Change the type from ANY to AMERICAN-FLIGHTS-API:BAD_REQUEST.



Test the application

26. Save the file to redeploy the project.
27. In Advanced REST Client, make another request to <http://localhost:8081/flights?airline=american&code=PDX>.
28. In the Mule Debugger, step through the application; the error should still be handled by globalError_Handler and then the getFlights flow error handler.

```
200 OK 109017.67 ms

{
  "message": "No flights to PDX"
}
```

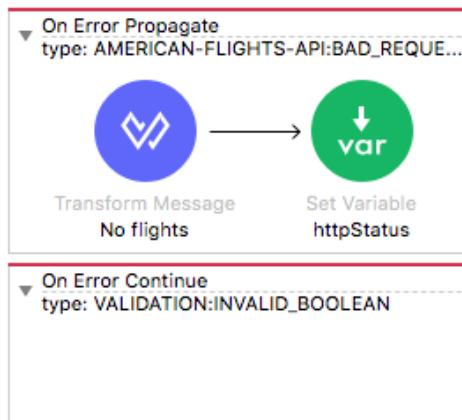
29. In Advanced REST Client, change the code to make a request to <http://localhost:8081/flights?airline=american&code=FOO>.
30. In the Mule Debugger, step through the application; you should get a 500 Server Error and no message because the error is not handled by the getFlights flow error handler or by globalError_handler.

The screenshot shows the Advanced REST Client interface. At the top, it displays "Method: GET" and "Request URL: http://localhost:8081/flights?airline=american&code=FOO". Below the URL is a "SEND" button and a more options menu. Underneath, there's a "Parameters" dropdown. The main response area shows a red box containing "500 Server Error" and "9631.16 ms". To the right of this, there's a "DETAILS" link.

31. Return to Anypoint Studio and switch to the Mule Design perspective.
32. Stop the project.
33. Navigate to the Responses tab in the properties view for the GET /flights Listener.
34. Review the error response body (not the response status code!) and ensure you know why you got the last response for <http://localhost:8081/flights?airline=american&code=FOO>.

Use an On Error Continue error handler in a flow

35. Add an On Error Continue scope to getFlights.
36. In the On Error Continue properties view, set the type to VALIDATION:INVALID_BOOLEAN.



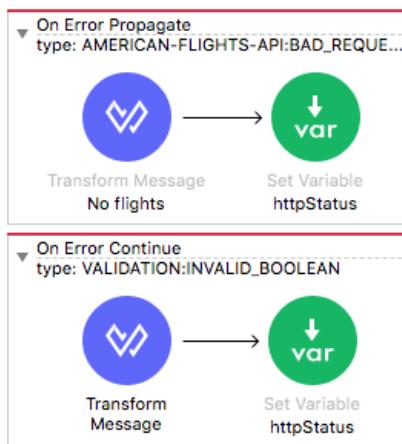
37. Add a Transform Message component to the scope and set the display name to error.description.

38. In the Transform Message properties view, set the payload to a JSON message property equal to the error.description.

```
1 %dw 2.0
2   output application/json
3   ---
4   [
5     "message": error.description
6 }
```

39. Add a Set Variable transformer to the On Error Continue scope.

40. Set the display name and name to httpStatus and set the value to 400.



Test the application

41. Save the file and run the project.

42. In Advanced REST Client, make another request to

<http://localhost:8081/flights?airline=american&code=FOO>; you should get a 200 response and the invalid destination message again – not the 400 response.



Set the HTTP Listener response status code

43. Return to Anypoint Studio.

44. Navigate to the Responses tab in the properties view for the GET /flights Listener.

45. Set the response status code (not the error response status code!) to an expression that sets it equal to the value of an httpStatus variable with a default value of 200.

```
# [vars.httpStatus default 200]
```

Test the application

46. Save the file to redeploy the project.
47. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=FOO>; you should now get the 400 response.

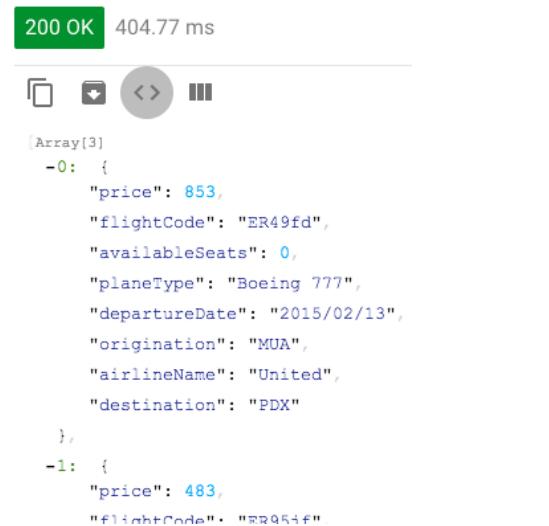


A screenshot of the Advanced REST Client interface. At the top, there's a green button labeled "400 Bad Request" and the text "396.58 ms". Below this is a toolbar with icons for copy, paste, refresh, and others. The main area shows a JSON response:

```
{  
    "message": "Invalid destination FOO"  
}
```

Test the application to see what happens with the Scatter-Gather

48. In Advanced REST Client, change the code and make a request to
<http://localhost:8081/flights?airline=american&code=PDX>; you should get a 200 response and no flights to PDX.
49. Change the airline to make a request to <http://localhost:8081/flights?airline=united&code=PDX>; you should get flights to PDX.



A screenshot of the Advanced REST Client interface. At the top, there's a green button labeled "200 OK" and the text "404.77 ms". Below this is a toolbar with icons for copy, paste, refresh, and others. The main area shows a JSON response representing an array of three flight objects:

```
[  
    {  
        "price": 853,  
        "flightCode": "ER49fd",  
        "availableSeats": 0,  
        "planeType": "Boeing 777",  
        "departureDate": "2015/02/13",  
        "origination": "MUA",  
        "airlineName": "United",  
        "destination": "PDX"  
    },  
    {  
        "price": 483,  
        "flightCode": "ER95if"  
    }  
]
```

50. Remove the airline to make a request to <http://localhost:8081/flights?code=PDX>; you should get flights to PDX because United has flights, but you get none.

500 Server Error 175.36 ms DETAILS ▾

□ ↻ ⌂ ⌃

```
{  
    "message": "Exception(s) were found for route(s): 0:  
    org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET  
    on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: too many  
    requests (429). 2: org.mule.runtime.api.connection.ConnectionException: Error  
    processing WSDL file [http://mu.learn.mulesoft.com/deltas?wsdl]: Unable to locate  
    document at 'http://mu.learn.mulesoft.com/deltas?wsdl'.  
}
```

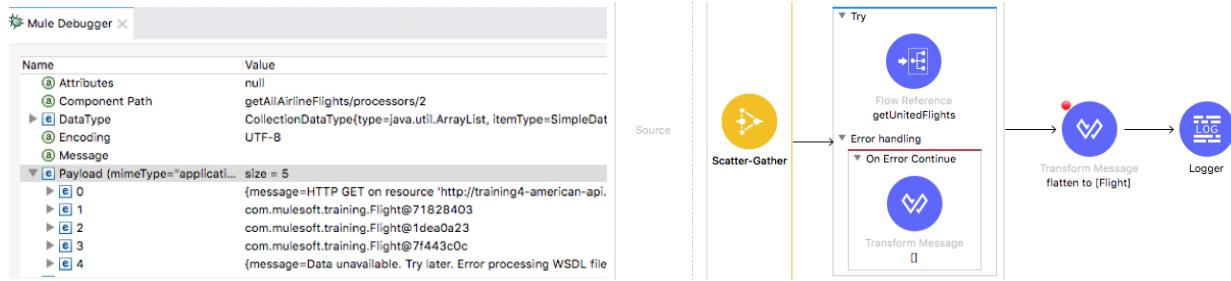
51. Return to Anypoint Studio.

52. Stop the project.

Walkthrough 10-5: Handle errors at the processor level

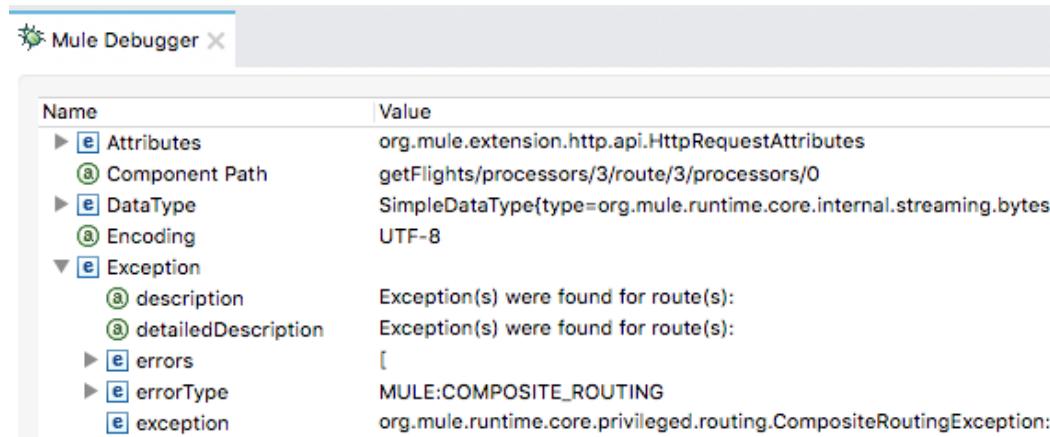
In this walkthrough, you work with the Scatter-Gather in getAllAirlineFlights so that it correctly returns results when one but not all airlines have flights. You will:

- Wrap the Flow Reference in each branch of a Scatter-Gather in a Try scope.
- Use an On Error Continue scope in each Try scope error handler to provide a valid response so flow execution can continue.



Debug the application when a Scatter-Gather branch has an error

1. Return to implementation.xml in Anypoint Studio.
2. Debug the project.
3. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>.
4. Return to the Mule Debugger and step through the application; you should see a routing exception.



5. Return to Advanced REST client, you should get the 500 Server Error that there were exceptions in routes 0 and 2.

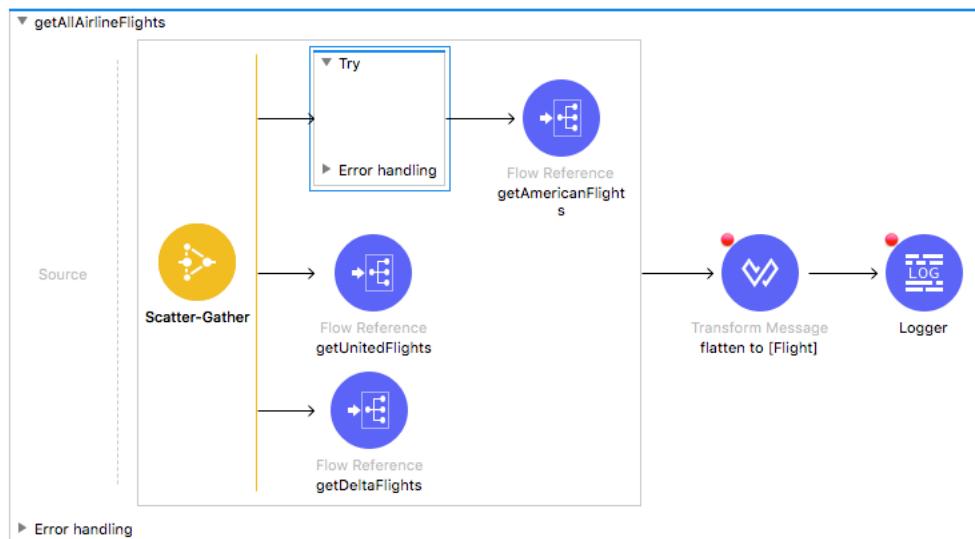
500 Server Error 49390.78 ms DETAILS ▾

```
{
  "message": "Exception(s) were found for route(s): 0:
org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET
on resource 'http://training4-american-api-mule.cloudhub.io:80/flights' failed: bad
request (400). 2: org.mule.runtime.api.connection.ConnectionException: Error processing
WSDL file [http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at
'http://mu.mulesoft-training.com/deltas?wsdl'."
}
```

6. Return to Anypoint Studio and switch to the Mule Design perspective.

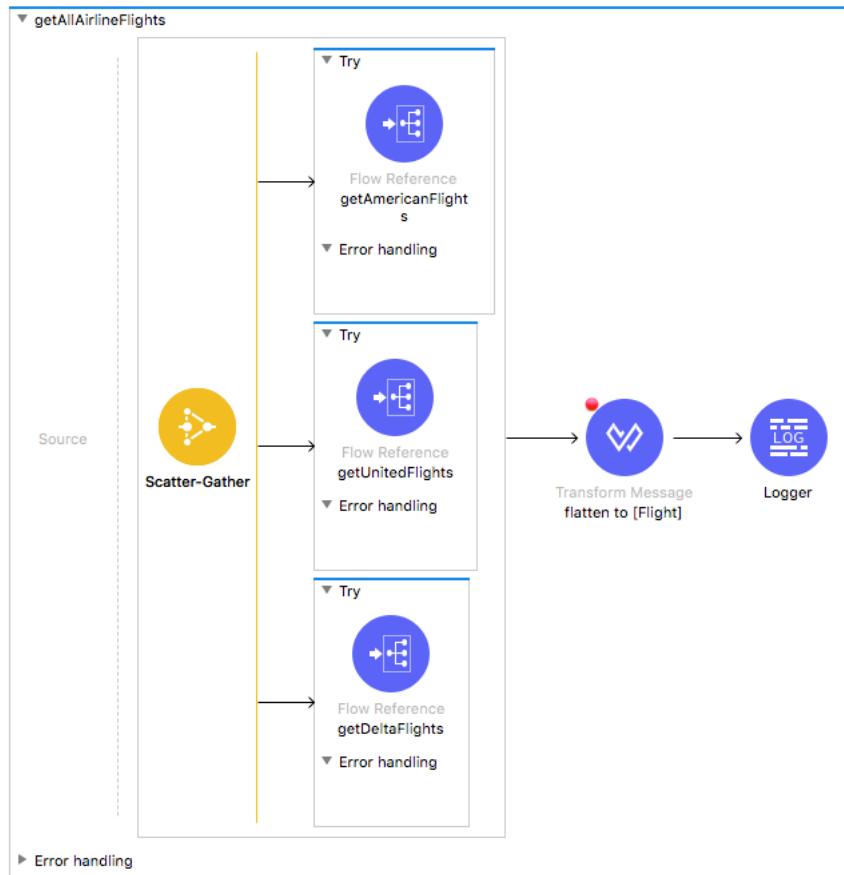
Place each Flow Reference in the Scatter-Gather in a Try scope

7. Locate getAllAirlineFlights.
8. Drag a Try scope from the Mule Palette and drop it in the Scatter-Gather.



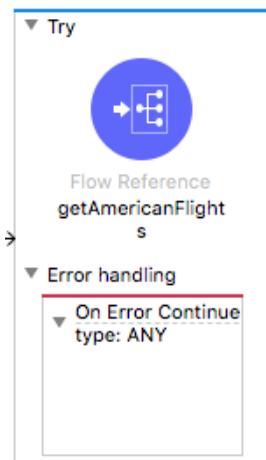
9. Drag the getAmericanFlights Flow Reference into the Try scope.
10. Add two more Try scopes to the Scatter-Gather.

11. Move the getUnitedFlights Flow Reference into one of the Try scopes and the getDeltaFlights Flow Reference into the other.



Add an error handler to each branch that passes through the error message

12. Expand the error handling section of the first Try scope.
13. Add an On Error Continue scope and set the types of errors it handles to ANY.
14. Repeat these steps to add the same error handler to the two other Try scopes.



Debug the application

15. Save the file to redeploy the project.

Note: If you get an error when deploying the project about a duplicate flow-ref name attribute, review the error and then switch to the Configuration XML view to fix it.

16. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>.
17. In the Mule Debugger, step through the application until you are at the Logger after the Scatter-Gather in getAllAirlineFlights.
18. Expand Payload; you should see three flights (from United) and two error messages.

The screenshot shows the Mule Debugger interface. At the top, there's a table with columns 'Name' and 'Value'. Below it, the 'Payload' section is expanded, showing a list of five items. The first four items are flight objects from United, and the fifth item is an error message. Below the payload table, there's a Mule flow diagram titled 'implementation'. The flow starts with a 'Scatter-Gather' component, followed by a 'Try' block containing an 'HTTP' connector with a flow reference to 'getUnitedFlights'. An 'Error handling' section within the Try block includes an 'On Error Continue' option. After the Try block, the flow goes through a 'Transform Message' component (flattening the Flight message) and finally reaches a 'Logger' component.

19. Step through the rest of the application.

20. Return to Advanced REST Client; you should see both United flights and error messages.

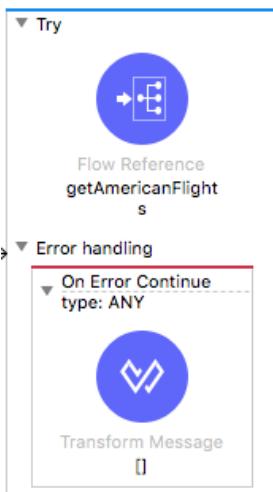
200 OK 98570.85 ms DET

[Array[5]

-0: {
"message": "HTTP GET on resource 'http://training4-american-api-mule.cloudhub.io:80/flights' failed: bad request (400)."
},
-1: { ... },
-2: { ... },
-3: { ... },
-4: {
"message": "Data unavailable. Try later. Error processing WSDL file
[http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at
'http://mu.mulesoft-training.com/deltas?wsdl'.
"}
}

Modify each error handler to set the payload to an empty array

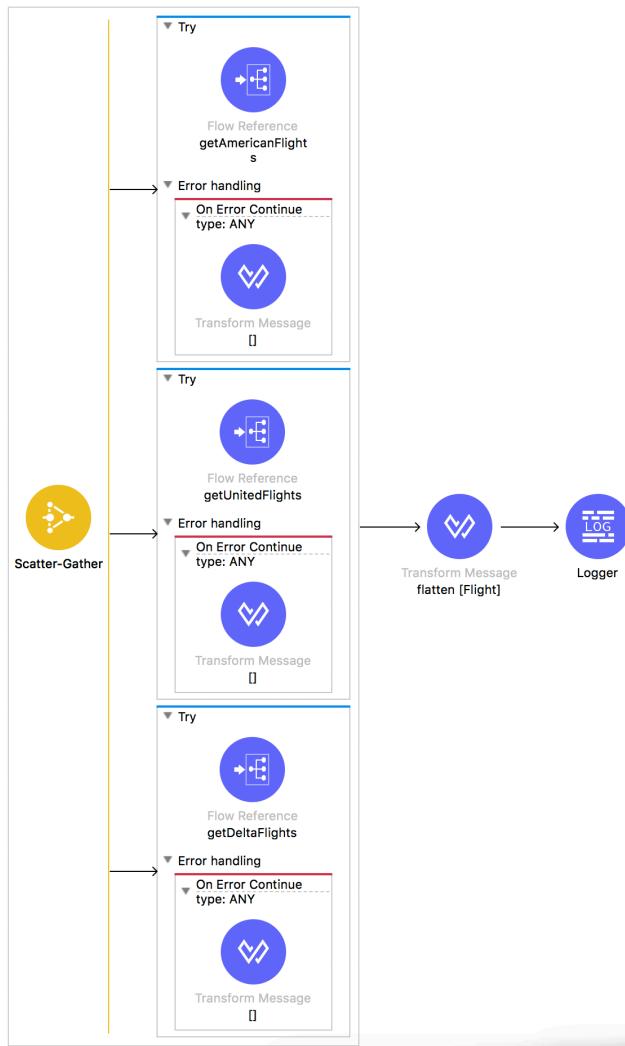
21. Return to Anypoint Studio and switch to the Mule Design perspective.
22. In getAllAirlineFlights, add a Transform Message component to one of the On Error Continue scopes.
23. Set the display name to [].



24. In the Transform Message properties view, set the payload to an empty array.

```
1 %dw 2.0
2 output application/java
3 ---
4 ()
```

25. Repeat the steps to add the same Transform Message component to the two other error handlers.



Debug the application

26. Save the file to redeploy the project.
27. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>.
28. In the Mule Debugger, step through the application until you are at the Logger after the Scatter-Gather in `getAllAirlineFlights`.

29. Expand Payload; you should now see the three flights and no error messages.

The screenshot shows the Mule Debugger interface. At the top, there's a tree view with nodes like Name, Datatype, Encoding, Message, and Payload. Under Payload, it shows a size of 3 and three entries: com.mulesoft.training.Flight@43f6840, com.mulesoft.training.Flight@4a827695, and com.mulesoft.training.Flight@6862340f. Below this is a flow diagram titled 'implementation'. It starts with a 'Source' (Scatter-Gather) component, followed by a 'Flow Reference getUnitedFlights'. This is followed by an 'Error handling' section with 'On Error Continue type: ANY'. The flow then continues through a 'Transform Message flatten [Flight]' component and finally a 'Logger' component.

30. Step through the rest of the application.

31. Return to Advanced REST Client; you should now only see the three flights.

The screenshot shows the Advanced REST Client results page. It displays a green '200 OK' status bar with a response time of 46634.68 ms. Below this is a table with columns for Headers, Status, and Body. The Headers and Status rows are empty. The Body row contains the following JSON output:

```
[Array[3]
-0: { ...
-1: { ...
-2: {
  "price": 532,
  "flightCode": "ER04kf",
  "availableSeats": 30,
  "planeType": "Boeing 777",
  "departureDate": "2015/02/12",
  "origination": "MUA",
  "airlineName": "United",
  "destination": "PDX"
}}
```

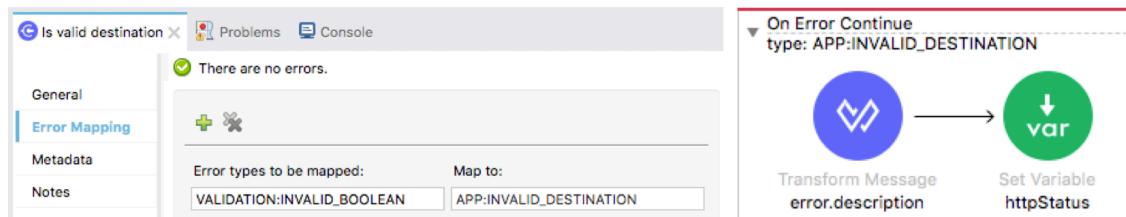
32. Return to Anypoint Studio and switch to the Mule Design perspective.

33. Stop the project.

Walkthrough 10-6: Map an error to a custom error type

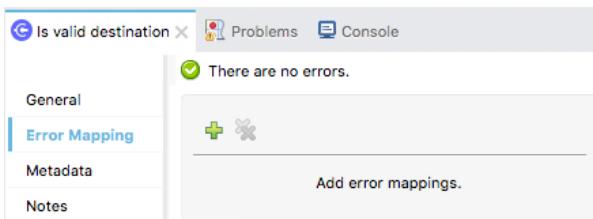
In this walkthrough, you map the Validation error to a custom error type for the application. You will:

- Map a module error to a custom error type for an application.
- Create an event handler for the custom error type.

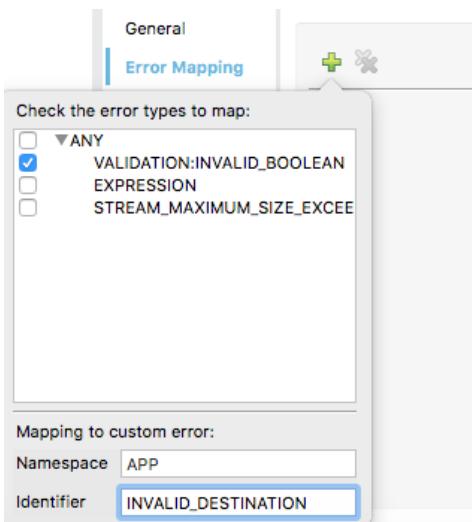


Map a validation module error to a custom error type

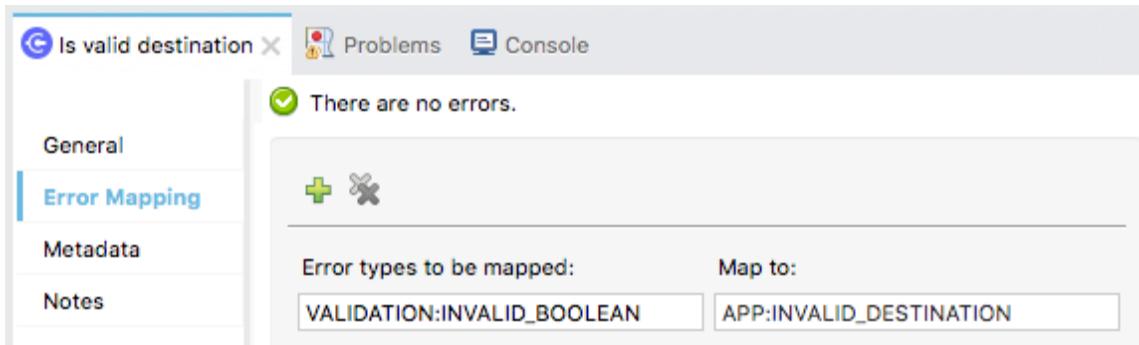
1. Return to implementation.xml.
2. Navigate to the properties view for the validator in getFlights.
3. Select the Error Mapping tab.
4. Click the Add new mapping button.



5. Select the VALIDATION:INVALID_BOOLEAN error type.
6. Leave the namespace of the custom error to map to set to APP.
7. Set the identifier to INVALID_DESTINATION.

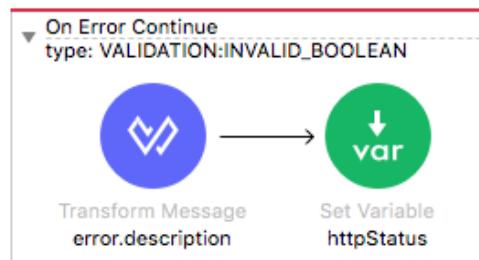


8. Press Enter; you should see your new error mapping.

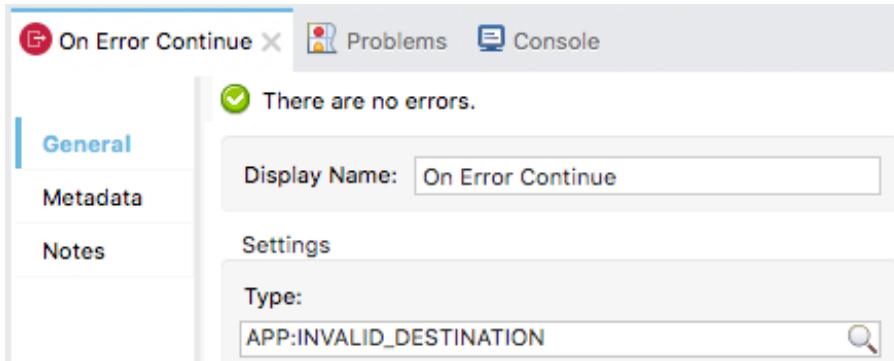


Change the existing validation error handler to catch the new custom type

9. Navigate to the properties view for the validation getFlights On Error Continue error handler.



10. Change the type from VALIDATION:INVALID_BOOLEAN to the new APP:INVALID_DESTINATION that should now appear in the type drop-down menu.



Test the application

11. Save the file and run the project.

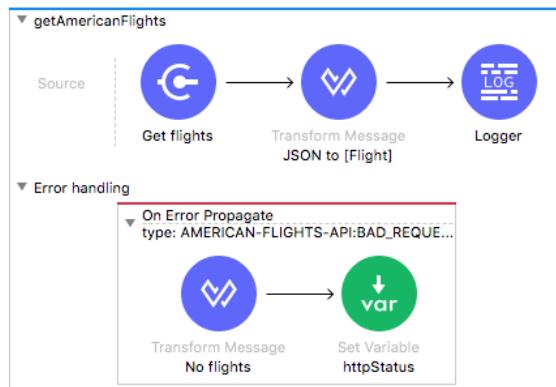
12. In Advanced REST Client, change the code and make a request to <http://localhost:8081/flights?code=FOO>; you should still get a 400 response with the invalid destination error.

400 Bad Request 30.10 ms

```
{  
    "message": "Invalid destination FOO"  
}
```

Move the American error handler into the American flow

13. Collapse the getAllAirlineFlights and setCode flows.
14. Move the AMERICAN-FLIGHTS-API error scope from getFlights into getAmericanFlights.

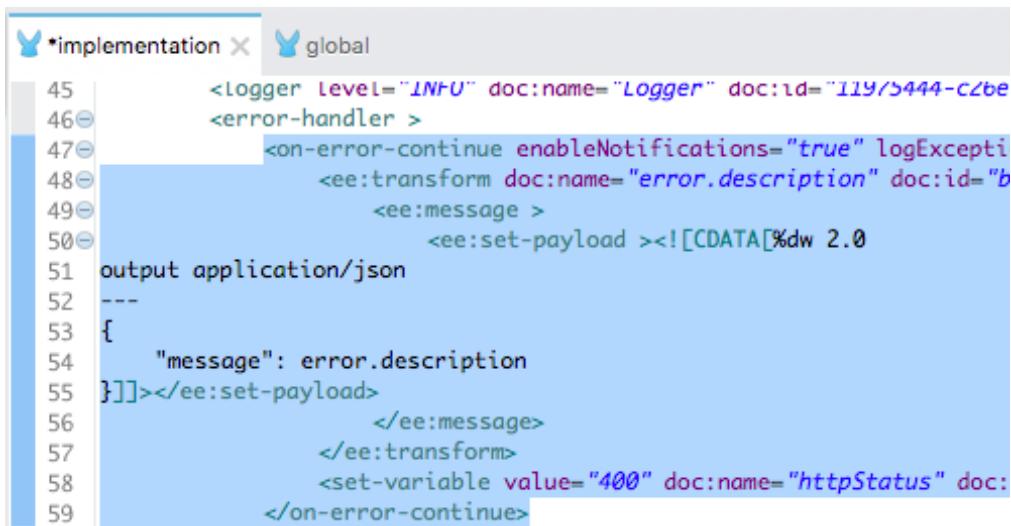


Move the error handler to the global error handler

Note: This description has instructions to make changes in the XML. If you prefer, you can copy and paste the error handler to move it between files and then delete the original.

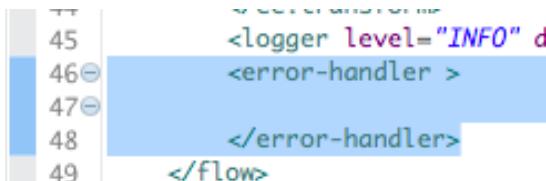
15. Return to implementation.xml.
16. Right-click the validation error scope in getFlights and select Go to XML.

17. Select and cut the APP:INVALID_DESTINATION on-error-continue.



```
45     <Logger level="INFO" doc:name="Logger" doc:id="119/5444-c2be
46     <error-handler>
47         <on-error-continue enableNotifications="true" logException="true">
48             <ee:transform doc:name="error.description" doc:id="b3333333-3333-4333-8888-888888888888">
49                 <ee:message>
50                     <ee:set-payload><![CDATA[%dw 2.0
51 output application/json
52 --->
53 {
54     "message": error.description
55 }]]></ee:set-payload>
56             </ee:message>
57             <ee:transform>
58                 <set-variable value="400" doc:name="httpStatus" doc:id="c3333333-3333-4333-8888-888888888888">
59             </on-error-continue>
```

18. Delete the remaining, empty error-handler tag set.



```
45     <logger level="INFO" doc:id="119/5444-c2be
46     <error-handler>
47         </error-handler>
48     </flow>
```

19. Return to global.xml.

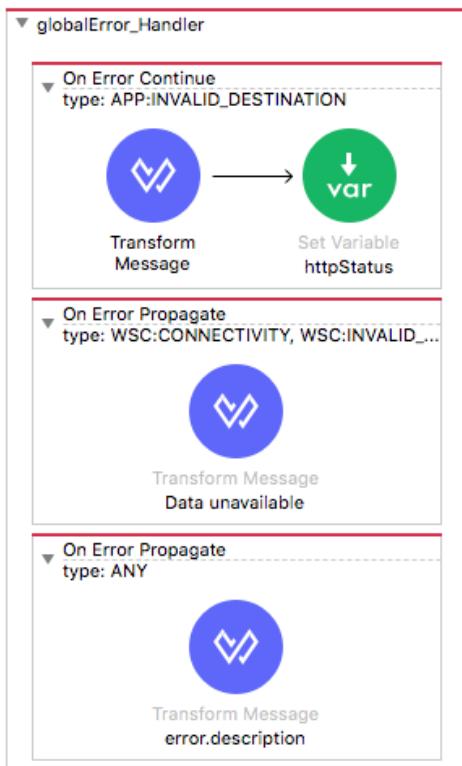
20. Right-click the globalError_Handler and select Go to XML.

21. Place the cursor on a new line inside and at the top of the error-handler.

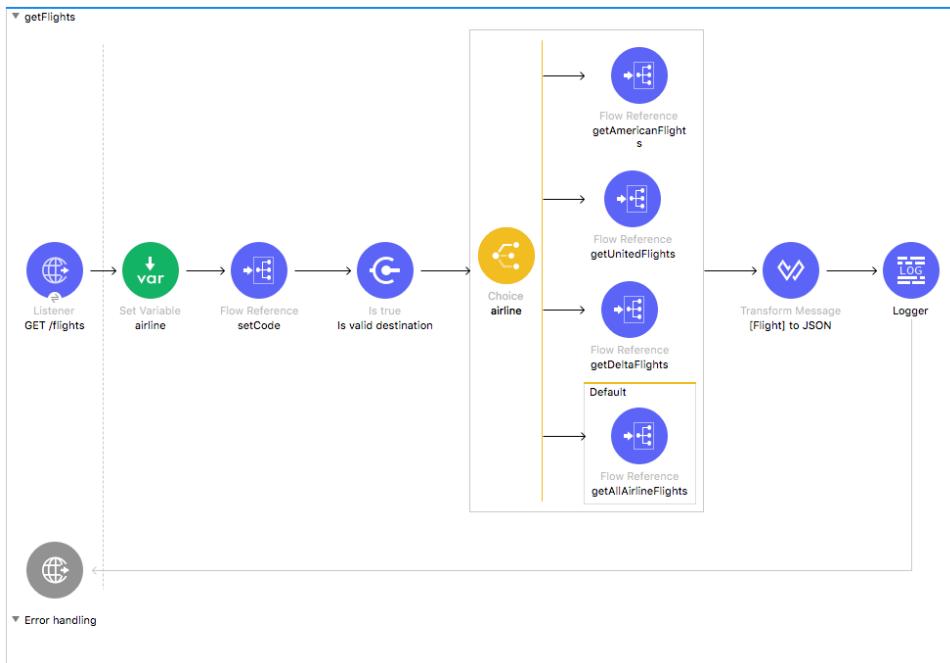


```
22     <error-handler name="globalError_Handler" doc:id="f7acbe
23         <on-error-continue enableNotifications="true" logException="true">
24             <ee:transform doc:name="error.description" doc:id="b3333333-3333-4333-8888-888888888888">
25                 <ee:message>
26                     <ee:set-payload><![CDATA[%dw 2.0
27 output application/json
28 --->
29 {
30     "message": error.description
31 }]]></ee:set-payload>
32             </ee:message>
33             <ee:transform>
34                 <set-variable value="400" doc:name="httpStatus" doc:id="c3333333-3333-4333-8888-888888888888">
35             </on-error-continue>
36             <on-error-propagate enableNotifications="true" logException="true">
37                 <ee:transform doc:name="Data unavailable" doc:id="d3333333-3333-4333-8888-888888888888">
38                     <ee:message>
```

22. Switch back to the Message Flow view; you should see the INVALID_DESTINATION handler.



23. Return to implementation.xml and switch to the Message Flow view; you should no longer see an error handler in getFlights.



Test the application

24. Save the files to redeploy the project.
25. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=FOO>; you should still get a 400 response with the invalid destination error.

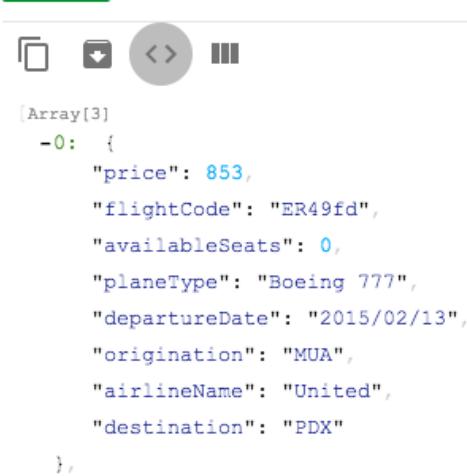
400 Bad Request 1534.19 ms



```
{  
    "message": "Invalid destination FOO"  
}
```

26. Change the code and make a request to <http://localhost:8081/flights?code=PDX>; you should still get only United flights.

200 OK 622.49 ms

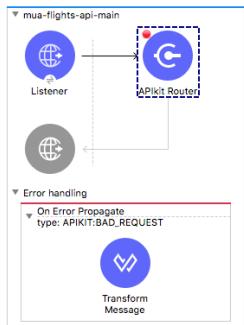


```
[Array[3]  
-0: {  
    "price": 853,  
    "flightCode": "ER49fd",  
    "availableSeats": 0,  
    "planeType": "Boeing 777",  
    "departureDate": "2015/02/13",  
    "origination": "MUA",  
    "airlineName": "United",  
    "destination": "PDX"  
},  
1: {  
    "price": 853,  
    "flightCode": "ER49fd",  
    "availableSeats": 0,  
    "planeType": "Boeing 777",  
    "departureDate": "2015/02/13",  
    "origination": "MUA",  
    "airlineName": "United",  
    "destination": "PDX"  
},  
2: {  
    "price": 853,  
    "flightCode": "ER49fd",  
    "availableSeats": 0,  
    "planeType": "Boeing 777",  
    "departureDate": "2015/02/13",  
    "origination": "MUA",  
    "airlineName": "United",  
    "destination": "PDX"  
}]
```

Walkthrough 10-7: Review and integrate with APIkit error handlers

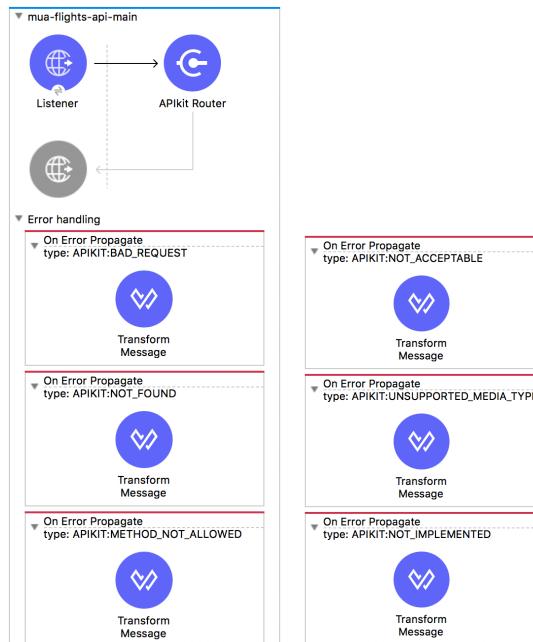
In this walkthrough, you connect the application's implementation to the interface and ensure all the error handling works. You will:

- Review the error handlers generated by APIkit.
- Review settings for the APIkit Router and HTTP Listener in the APIkit generated interface.
- Connect the implementation to the interface and test the error handling behavior.
- Modify implementation error scopes so they work with the APIkit generated interface.



Review APIkit generated error handlers

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open interface.xml.
3. Review the error handling section in mua-flights-api-main.



- Review the types of errors handled by each error handler scope.
- Navigate to the Transform Message properties view for the first error handling scope.
- Review the expression that sets the payload.

Output Payload    Preview

```

1 %dw 2.0
2 output application/json
3 ---
4 {message: "Bad request"}

```

- Use the output drop-down menu to change the output to Variable – httpStatus.

Output Payload  

Payload
 Variable - httpStatus

```

1 %dw 2.0
2 output application/json
3 ---
4 {message: "Bad request"}

```

- Review the expression that sets an httpStatus variable; you should see for a bad request that the httpStatus variable is set to 400.

Output Variable - httpStatus    Preview

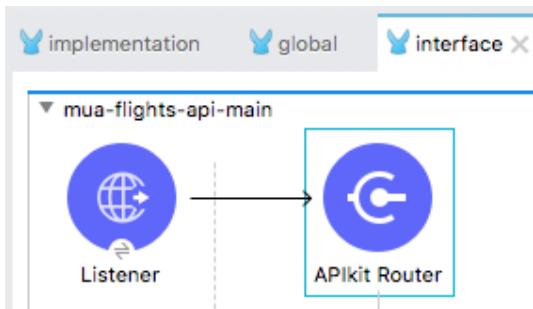
```

1 400

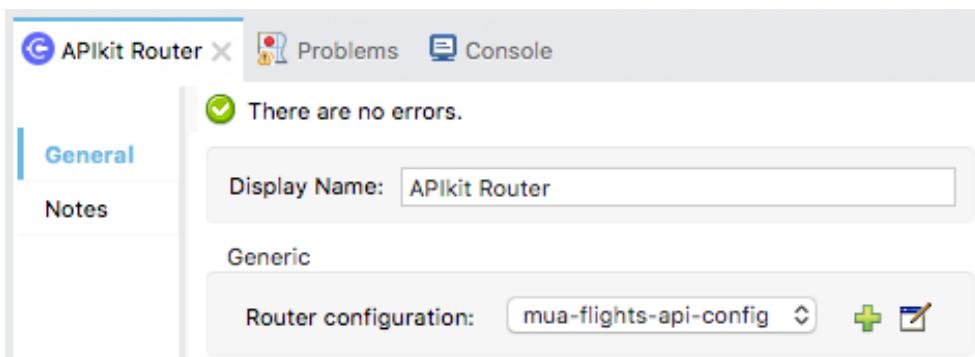
```

Review settings for the APIkit Router

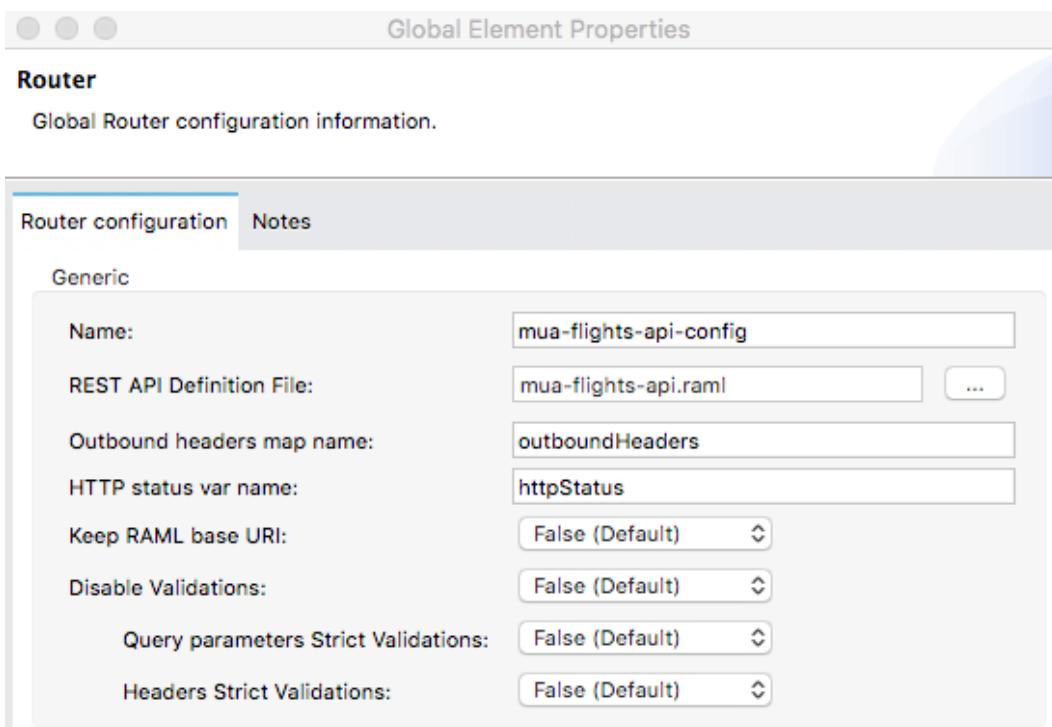
- Navigate to the properties view for the APIkit Router in mua-flights-api-main.



10. Click the Edit button next to router configuration.



11. In the Global Element Properties dialog box, locate the HTTP status var name setting; you should see `httpStatus`.

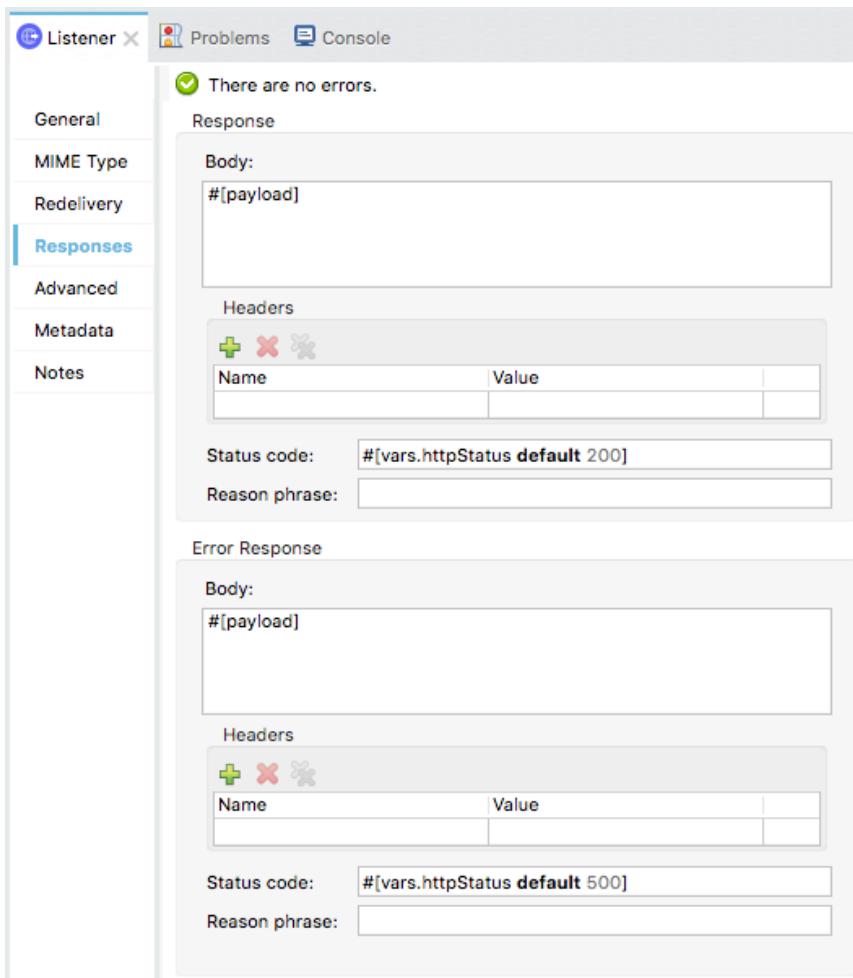


12. Click OK.

Review settings for the HTTP Listener

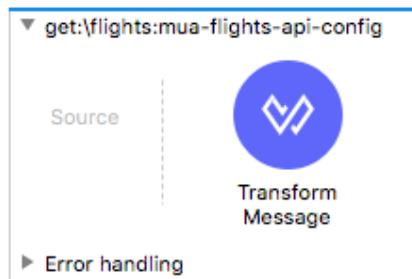
13. Navigate to the Responses tab in the properties view for the HTTP Listener in `mua-flights-api-main`.

14. Review the response body and status code.
15. Review the error response body and status code.



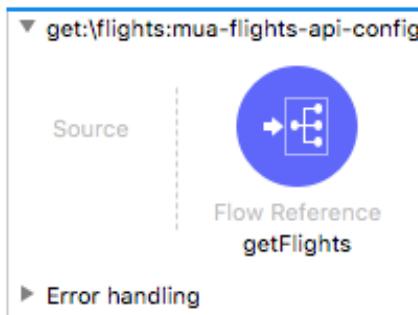
Connect the interface to the implementation

16. In interface.xml, locate the get:\flights flow.

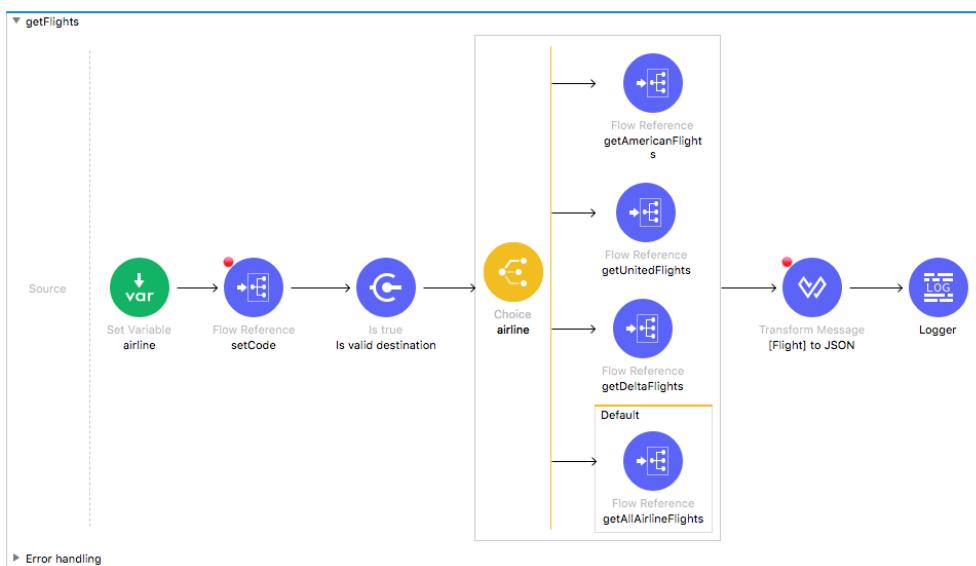


17. Review the default transformation in this flow.
18. Delete the Transform Message component.

19. Add a Flow Reference component to the flow.
20. In the Flow Reference properties view, set the flow name to getFlights.



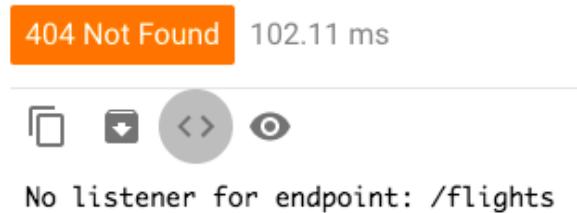
21. Return to implementation.xml.
22. Delete the GET /flights Listener in getFlights.



23. Save all the files to redeploy the application.

Test the application

24. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>; you should get a 404 response with a no listener message.



25. Add /api to the URL and make a request to <http://localhost:8081/api/flights?code=PDX>; you should get flights as before.

200 OK 896.95 ms



```
[Array[3]
-0: {
  "price": 853,
  "flightCode": "ER49fd",
  "availableSeats": 0,
  "planeType": "Boeing 777",
  "departureDate": "2015/02/13",
  "origination": "MUA",
  "airlineName": "United",
  "destination": "PDX"
},
```

26. Change the code to make a request to <http://localhost:8081/api/flights?code=FOO>; you should now get a 400 Bad Request response instead of your custom message.

400 Bad Request 39.57 ms



```
{
  "message": "Bad request"
}
```

27. Remove the code to make a request to <http://localhost:8081/api/flights>; you should now get your custom error message.

400 Bad Request 22660.33 ms



```
{
  "message": "Invalid destination"
}
```

28. Add the airline and code to make a request to

<http://localhost:8081/api/flights?airline=american&code=PDX>; you should now get a 500 Server error with no message instead of your 200 no flights to PDX response.

The screenshot shows the Advanced REST Client interface. At the top, it displays 'Method: GET' and 'Request URL: http://localhost:8081/api/flights?airline=american&code=PDX'. Below the URL is a 'SEND' button and a three-dot menu icon. Underneath the URL, there's a 'Parameters' dropdown. The main response area shows a red box containing '500 Server Error' and '283.89 ms'. To the right of this, there's a 'DETAILS' link with a dropdown arrow. The entire interface has a light blue background with white text and red highlights for errors.

Review the API

29. Return to Anypoint Studio and stop the project.

30. Return to mua-flights-api.raml and review the code; you should see the code query parameter is not required but it has allowed values enumerated.

```
queryParameters:  
  code:  
    displayName: Destination airport code  
    required: false  
    enum:  
      - SFO  
      - LAX  
      - PDX  
      - CLE  
      - PDF
```

Debug the application

31. Return to interface.xml.

32. Add a breakpoint to the APIkit Router in mua-flights-api-main.

33. Debug the project.

34. In Advanced REST Client, make another request to

<http://localhost:8081/api/flights?airline=american&code=PDX>.

35. In the Mule Debugger, watch the payload and variables and step through the application.

36. Step back to the APIkit router.

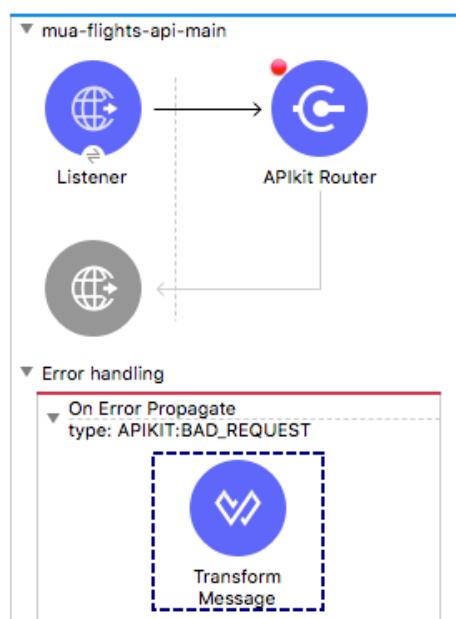
37. Review the exception, the payload, and the variables; you should no longer see any variables or your JSON payload message.

The screenshot shows the Mule Debugger interface with two main sections. The top section is a table titled 'Name' and 'Value'. It contains an entry for 'Exception' which includes details like 'description' (HTTP GET on resource), 'detailedDescription' (HTTP GET on resource), 'errors' (empty array), 'errorType' (AMERICAN-FLIGHTS-API:BAD_REQUEST), 'exception' (org.mule.extension.http.api.request.valida), 'muleMessage' (null), and 'Payload' (mimeType="*/"; cha...). The bottom section shows a flow diagram with a 'Listener' component connected to an 'APIkit Router' component. The 'APIkit Router' component has a red dashed border around it, indicating it is the current focus.

38. In Advanced REST Client, change the code to make a request to <http://localhost:8081/api/flights?airline=american&code=FOO>.

39. In the Mule Debugger, step through the application; the APIkit router should immediately throw an error.

40. Step again; you should see the error is handled by the that is handled by its APIKIT:BAD_REQUEST handler.



41. Click Resume to finish stepping through the application.

42. In Advanced REST Client, remove the airline and code to make a request to <http://localhost:8081/api/flights>.
43. In the Mule Debugger, step through the application; the validator should throw an error and execution should **not** return to the APIkit router.
44. Return to Advanced REST Client; you should successfully get a 400 response with the custom message.

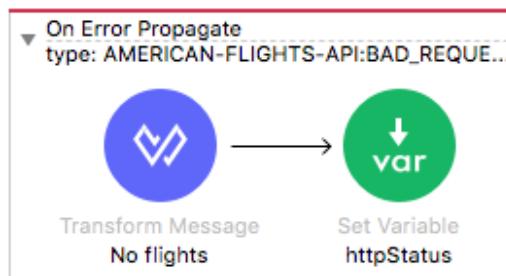
400 Bad Request 22660.33 ms

```
{
  "message": "Invalid destination"
}
```

Note: If you had specified the code query parameter to be required in the RAML file from which the interface was generated, the APIkit router would catch this immediately and respond with a 400 Bad Request response. The event would never get to the validator in your implementation.

Change the American flights error scope to On Error Continue

45. Return to Anypoint Studio and switch to the Mule Design perspective.
46. Return to implementation.xml.
47. Locate the error handler in the getAmericanFlights flow.



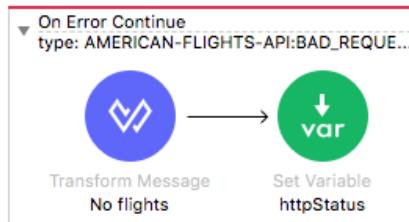
48. Right-click it and select Go To XML.

49. Change the on-error-propagate start and end tags in the getAmericaFlights flow to on-error-continue.

```
<error-handler>
    <on-error-continue ena...
        <ee:transform doc:>
            <ee:message>
                <ee:set-pa...
output application/json
---
{
    "message": "No flights to " ++
}]]><ee:set-payload>
        </ee:message>
    </ee:transform>
    <set-variable value...
    </on-error-continue>
</error-handler>
```

50. Change the doc:name in the start tag to On Error Continue.

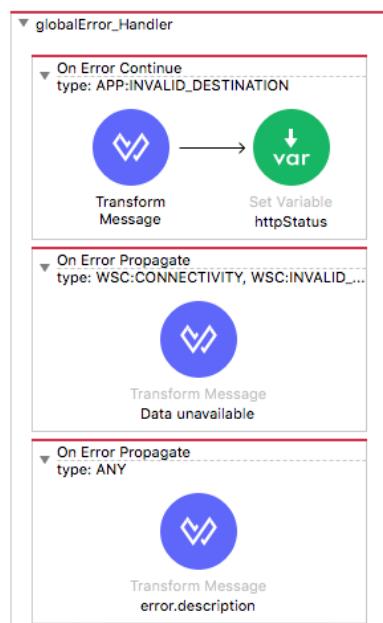
51. Switch back to the Message Flow view; you should now see an On Error Continue.



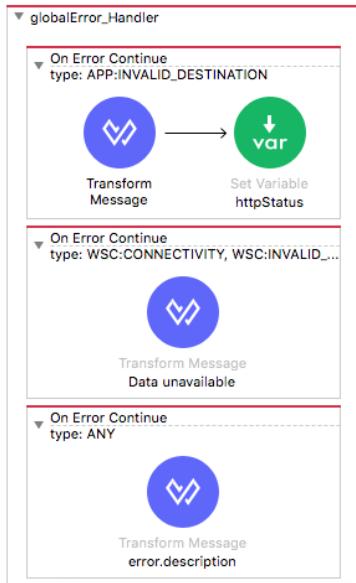
Change the global default error scopes to On Error Continue

52. Return to global.xml.

53. Review the types of error handler scopes.

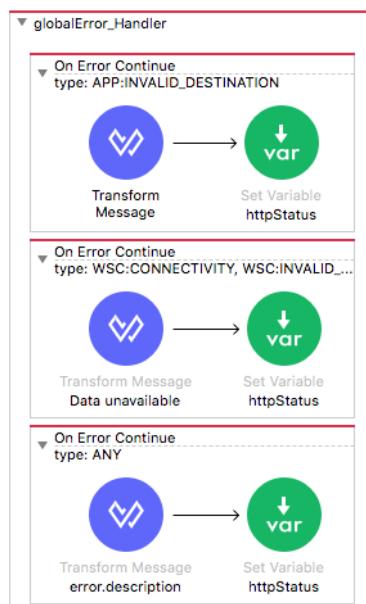


54. Right-click the WSC error handler and select Go To XML.
55. Change the four on-error-propagate start and end tags to on-error-continue.
56. Change the doc:names in both start tags to On Error Continue.
57. Switch back to the Message Flow view; you should now see both are On Error Continue scopes.



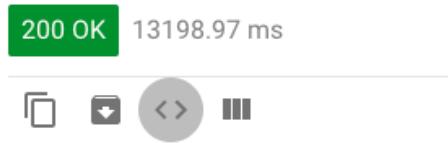
Set the HTTP status code for the On Error Continue scopes so you do not get 200

58. Add a Set Variable transformer to the WSC error handler.
59. In the Set Variable properties view, set the display name and name to httpStatus.
60. Set the value to 500.
61. Copy the Set Variable transformer and add it to the ANY error scope.



Test the application

62. Save the files to redeploy the project.
63. In Advanced REST Client, add the airline and code to make a request to <http://localhost:8081/api/flights?airline=american&code=PDX>.
64. In the Mule Debugger, step through the application; the application now does not return to the APIkit router.
65. Return to Advanced REST Client; you should now get a 200 response with the No flights to PDX message.



200 OK 13198.97 ms

{
 "message": "No flights to PDX"
}

66. In Advanced REST Client, change the airline to make a request to <http://localhost:8081/api/flights?airline=delta&code=PDX>.
67. In the Mule Debugger, step through the application.
68. Return to Advanced REST Client; you should now get the 500 response with the data unavailable message.



500 Server Error 5862.59 ms

{
 "message": "Data unavailable. Try later. Error processing WSDL file
 [http://mu.mulesoft-training.com/deltas?wsdl]: Unable to locate document at
 'http://mu.mulesoft-training.com/deltas?wsdl'.
}

69. Return to Anypoint Studio and switch perspectives.

Walkthrough 10-8: Set a reconnection strategy for a connector

In this walkthrough, you will:

- Set a reconnection strategy for the Web Service Consumer connector.

Set a reconnection strategy for a connector

1. Return to global.xml in Anypoint Studio.
2. Switch to the Global Elements view.
3. Double-click the Web Service Consumer Config.

The screenshot shows the 'Global Configuration Elements' view in Anypoint Studio. At the top, there are tabs for 'implementation', 'global' (which is selected), and 'interface'. Below the tabs is a table with columns 'Type', 'Name', and 'Description'. The table contains several entries, including 'Configuration properties', 'HTTP Listener config', 'American Flights API Config', 'HTTP Request configuration', 'Web Service Consumer Config' (which is highlighted in blue), and 'Configuration'. To the right of the table are three buttons: 'Create', 'Edit', and 'Delete'.

Type	Name	Description
Configuration properties (Configuration)	Configuration properties	
HTTP Listener config (Configuration)	HTTP_Listener_config	
American Flights API Config (Configuration)	American_Flights_API_Config	
HTTP Request configuration (Configuration)	HTTP_Request_configuration_training	
Web Service Consumer Config (Configuration)	Delta_Web_Service_Consumer_Config	
Configuration (Configuration)	Configuration	

4. In the Global Element Properties dialog box, select the Advanced tab in the second tab bar.
5. Select the Reconnection checkbox.
6. Change the reconnection strategy to standard.
7. Review the default frequency and attempts.

The screenshot shows the 'Global Element Properties' dialog box for the 'Web Service Consumer Config'. The title bar says 'Global Element Properties' and the sub-title is 'Web Service Consumer Config'. Below that is a note 'Default configuration'. The dialog has three tabs: 'General' (selected), 'Advanced' (selected), and 'Notes'. The 'Advanced' tab contains fields for 'Name' (set to 'Delta_Web_Service_Consumer_Config'), 'General', 'Security', 'Transport', and 'Advanced'. Under the 'Advanced' tab, there is a 'Connection' section with a checked 'Reconnection' checkbox. Below it is a dropdown for 'Fails deployment when test connection fails:' set to 'False (Default)'. Another dropdown for 'Reconnection strategy' is set to 'Standard'. Under 'Standard', there are fields for 'Frequency (ms):' (set to '2000') and 'Reconnection Attempts:' (set to '2'). At the bottom are 'Cancel' and 'OK' buttons.

8. Click OK.
9. Save the file.

Module 11: Writing DataWeave Transformations

The screenshot shows a DataWeave editor interface. On the left, the code is written in DataWeave syntax. On the right, the output is displayed as JSON documents.

```
1@ %dw 2.0
2 output application/json
3 type Currency = String {format: "##,.00"}
4 type Flight = Object {class: "com.mulesoft.training.Flight"}
5 fun getNumSeats(planeType: String) =
6    if (planeType contains ("737"))
7      150
8    else
9      300
10 import dasherize from dw::core::Strings
11 ---
12 @using (flights=
13   payload.*return map (object,index) -> {
14     destination: object.destination,
15     availableSeats: object.emptySeats as Number,
16     price: object.price as Number as Currency,
17     totalSeats: getNumSeats(object.planeType as String),
18     // totalSeats: lookup('getTotalSeats',{planeType: object.planeType}),
19     planeType: dasherize(replace(object.planeType,/(Boing)/) with "Boeing"),
20     departureDate: object.departureDate as Date {format: "yyyy/MM/dd"}
21       as String {format: "MMM dd, yyyy"}
22   } as Flight
23 )
24 @flights orderBy $.departureDate
25   orderBy $.price
26   distinctBy $
27   filter ($.availableSeats !=0)
```

[{"destination": "LAX", "availableSeats": 10, "price": "199.99", "totalSeats": 150, "planeType": "boeing-737", "departureDate": "Oct 21, 2015"}, {"destination": "PDX", "availableSeats": 23, "price": "283.00", "totalSeats": 300, "planeType": "boeing-777", "departureDate": "Oct 20, 2015"}, {"destination": "PDX", "availableSeats": 30, "price": "283.00", "totalSeats": 300, "planeType": "boeing-777", "departureDate": "Oct 21, 2015"}, {"destination": "SFO", }]

At the end of this module, you should be able to:

- Write DataWeave expressions for basic XML, JSON, and Java transformations.
- Write DataWeave transformations for complex data structures with repeated elements.
- Define and use global and local variables and functions.
- Use DataWeave functions.
- Coerce and format strings, numbers, and dates.
- Define and use custom data types.
- Call Mule flows from DataWeave expressions.
- Store DataWeave scripts in external files.

Walkthrough 11-1: Create transformations with the Transform Message component

In this walkthrough, you create a new flow that receives flight POST requests that you will use as the input for writing transformations in the next several walkthroughs. You will:

- Create a new flow that receives POST requests of JSON flight objects.
- Add sample data and use live preview.
- Create a second transformation that stores the output in a variable.
- Save a DataWeave script in an external file.
- Review DataWeave script errors.

The screenshot shows the Anypoint Studio interface. On the left, there is a code editor window titled "Output Variable - DWoutput" containing a DataWeave script:

```
1 %dw 2.0
2   output application/xml
3   ---
4   payload
```

To the right of the code editor is a "Preview" button. Below the code editor, the DataWeave script output is shown as XML:

```
{ "airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "JFK" }
```

On the right side of the interface, there is a tree view of project resources:

- src/test/resources:
 - flight-example.json
 - flights-example.json
 - flights-example.xml
 - log4j2-test.xml
 - united-flights-example.json
- src/main/resources:
 - application-types.xml
 - config.yaml
 - json_flight_playground.dwl
 - log4j2.xml
- sample_data:
 - json.json

Create a new flow that receives POST requests and returns the payload as Java

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open config.yaml.
3. Change the delta.wsdl property from /deltas?wsdl to /delta?wsdl.
4. Return to implementation.xml.
5. Right-click in the canvas and select Collapse All.
6. Drag a Transform Message component from the Mule Palette to the bottom of the canvas; a new flow should be created.
7. Change the name of the flow to postFlight.
8. In the Transform Message properties view, set the expression to the payload.

The screenshot shows the Anypoint Studio interface with a Transform Message component selected. The properties view shows the following configuration:

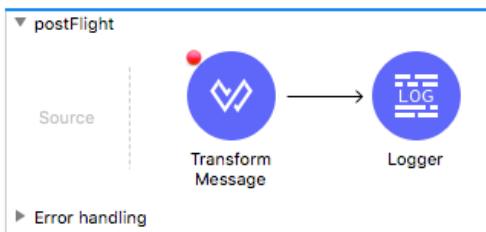
Output Payload

1 %dw 1.0
2 %output application/java
3 ---
4 payload

Preview

9. Add a breakpoint to the Transform Message component.

10. Add a Logger to the flow.

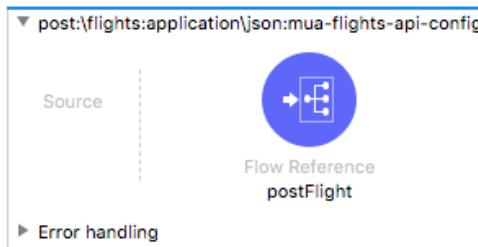


11. Return to interface.xml.

12. Locate the post:\flights:application\json:mua-flights-api-config flow.

13. Delete the Transform Message component and replace it with a Flow Reference.

14. In the Flow Reference properties view, set the flow name and display name to postFlight.



15. Save the files to redeploy the project in debug mode.

16. Close interface.xml.

Post a flight to the flow

17. Open flight-example.json in src/test/resources.

18. Copy the code and close the file.

19. In Advanced REST Client, change the method to POST and remove any query parameters.

20. Add a request header called Content-Type and set it equal to application/json.

21. Set the request body to the value you copied from flight-example.json.

22. Make the request to <http://localhost:8081/api/flights>.

Method: POST Request URL: http://localhost:8081/api/flights

Parameters ^

Headers	Authorization	Body	Variables	Actions
Body content type application/json				

FORMAT JSON MINIFY JSON

```
{  
  "airline": "United",  
  "flightCode": "ER38sd",  
  "fromAirportCode": "LAX",  
  "toAirportCode": "SFO",  
  "departureDate": "May 21, 2016",  
  "emptySeats": 0,  
  "totalSeats": 200,  
  "price": 199,  
  "planeType": "Boeing 737"  
}
```

23. In the Mule Debugger, expand Attributes and locate the content-type header.

24. Review the payload; you should see it has a mime-type of application/json.

Name	Value
Encoding	UTF-8
Message	
Payload (mimeType="application/json; charset=UTF-8", enc... {	
Variables	size = 2

```
{  
  "airline": "United",  
  "flightCode": "ER38sd",  
  "fromAirportCode": "LAX",  
  "toAirportCode": "SFO".
```

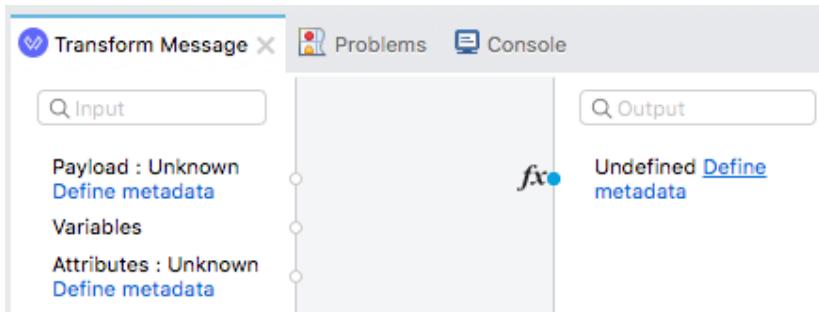
25. Step to the Logger; you should see the payload now has a mime-type of application/java and is a LinkedHashMap.

Name	Value	Type
e Payload (mimeType="application/java; ...	size = 9	java.util.LinkedHashMap
e 0	airline=United	java.util.LinkedHashMap\$Entry
e 1	flightCode=ER38sd	java.util.LinkedHashMap\$Entry
e 2	fromAirportCode=LAX	java.util.LinkedHashMap\$Entry
e 3	toAirportCode=SFO	java.util.LinkedHashMap\$Entry
e 4	departureDate=May 21, 2016	java.util.LinkedHashMap\$Entry
e 5	emptySeats=0	java.util.LinkedHashMap\$Entry
e 6	totalSeats=200	java.util.LinkedHashMap\$Entry
e 7	price=199	java.util.LinkedHashMap\$Entry
e 8	planeType=Boeing 737	java.util.LinkedHashMap\$Entry

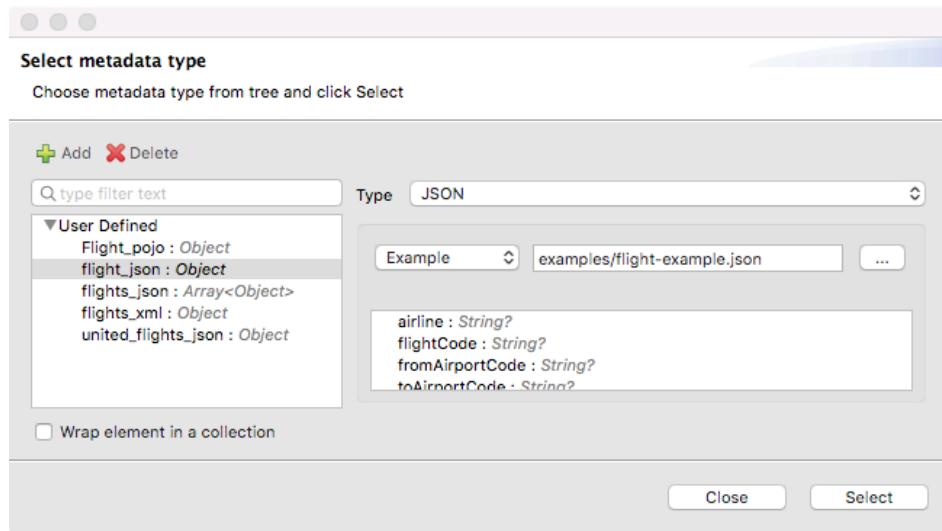
26. Step to the end of the application and switch perspectives.

Add input metadata for the transformation

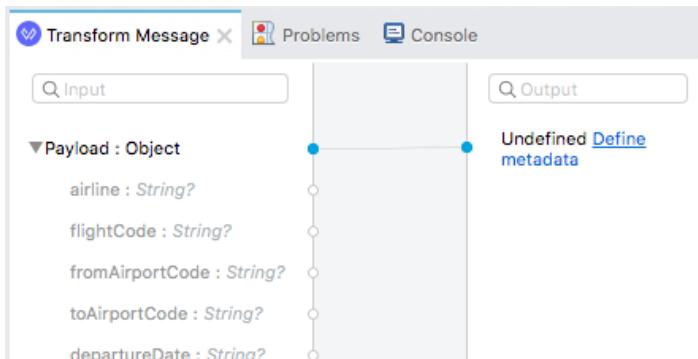
27. In the Transform Message properties view, click Define metadata in in the input section.



28. In the Select metadata type dialog box, select flight_json and click Select.



29. In the Transform Message Properties view, you should now see metadata for the input.



30. Open application-types.xml in src/main/resources.

31. Review the enrichment elements.

```
63     ~ types:processor-declaration>
64   </types:enrichment>
65   <types:enrichment select="#f963e3e4-5b3a-41bf-8c32-ef79bdce4a32">
66     <types:processor-declaration>
67       <types:input-event>
68         <types:message>
69           <types:payload type="flight_json"/>
70         </types:message>
71       </types:input-event>
72     </types:processor-declaration>
73   </types:enrichment>
74 </types:mule>
```

32. Close the file.

Preview sample data and sample output

33. In the Transform Message properties view, click the Preview button.
34. Click the Create required sample data to execute preview link.
35. Look at the input section; you should see a new tab called payload and it should contain the sample data.
36. Look at the preview section; you should see the sample output there of type Java.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. On the left, there's a tree view under 'json.json' containing flight information. Below it, a 'Context' tab and a 'payload' tab are visible. On the right, the 'Output' tab is selected, showing the generated Java code for the flight object:

```
{  
    airline: "United" as String [class: "java.lang.String"],  
    flightCode: "ER38sd" as String [class: "java.lang.String"],  
    fromAirportCode: "LAX" as String [class: "java.lang.String"],  
    toAirportCode: "SFO" as String [class: "java.lang.String"],  
    departureDate: "May 21, 2016" as String [class:  
        "java.lang.String"],  
    emptySeats: 0 as Number [class: "java.lang.Integer"],  
    totalSeats: 200 as Number [class: "java.lang.Integer"],  
    price: 199 as Number [class: "java.lang.Integer"],  
    planeType: "Boeing 737" as String [class: "java.lang.String"]  
} as Object [class: "java.util.LinkedHashMap", encoding: "UTF-8",  
mime_type: "*/*", raw: {  
    airline: "United" as String [class: "java.lang.String"],  
    flightCode: "ER38sd" as String [class: "java.lang.String"],  
    fromAirportCode: "LAX" as String [class: "java.lang.String"],  
    toAirportCode: "SFO" as String [class: "java.lang.String"],  
    departureDate: "May 21, 2016" as String [class: "java.lang.String"]},  
    ...}
```

37. Click the Source Only button to the left of the Preview button.

The screenshot shows the 'Transform Message' preview window again. The 'Output' tab is selected, displaying the Java code for the flight object. To the left, the 'Payload' tab is selected, showing the raw JSON input:

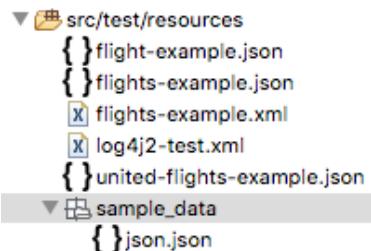
```
1@%dw 2.0  
2  output application/java  
3  ---  
4  payload
```

Below the tabs, the Java code for the flight object is displayed:

```
{  
    airline: "United" as String [class: "java.lang.String"],  
    flightCode: "ER38sd" as String [class: "java.lang.String"],  
    fromAirportCode: "LAX" as String [class: "java.lang.String"],  
    toAirportCode: "SFO" as String [class: "java.lang.String"],  
    departureDate: "May 21, 2016" as String [class: "java.lang.String"]},  
    ...}
```

Locate the new sample data file

38. In the project explorer, locate the new sample_data folder in src/test/resources.

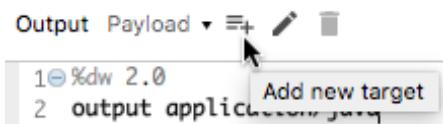


39. Open json.json.

40. Review the code and close the file.

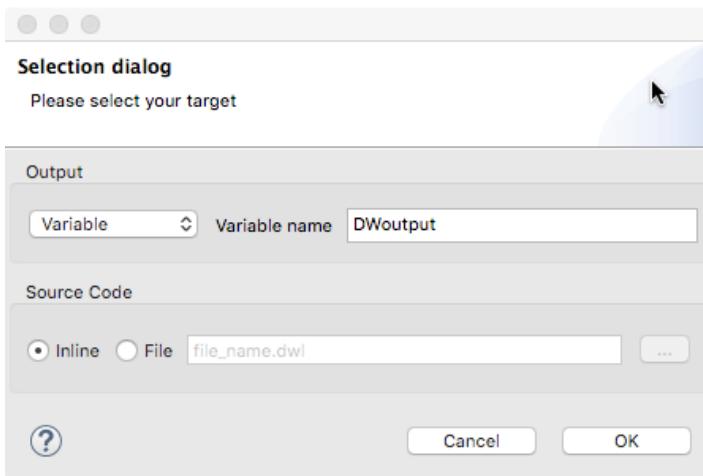
Create a second transformation with the same component

41. Click the Add new target button.



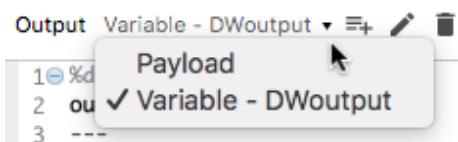
42. In the Selection dialog box, change the output to variable.

43. Set the variable name to DWoutput.



44. Click OK.

45. In the Transform Message properties view, click the drop-down menu button for the output; you should see and can switch between the two transformations.



46. For the new transformation, set the output type to json and set the expression to payload.

```
Output Variable - DWoutput ▾ + ✎ 🗑  
1@ %dw 2.0  
2   output application/json  
3   ---  
4   payload
```

Test the application

47. Save the file to redeploy the project in debug mode.
48. In Advanced REST Client, post the same request to <http://localhost:8081/api/flights>.
49. In the Mule Debugger, step to the Logger.
50. Review Payload; it should be Java as before.
51. Expand Variables; you should see the new DWoutput variable.

Name	Value	Type
▼ [E] Payload (mimeType="appl... size = 9		java.util.LinkedHashMap
► [E] 0	airline=United	java.util.LinkedHashMap\$Entry
► [E] 1	flightCode=ER38sd	java.util.LinkedHashMap\$Entry
► [E] 2	fromAirportCode=LAX	java.util.LinkedHashMap\$Entry
► [E] 3	toAirportCode=SFO	java.util.LinkedHashMap\$Entry
► [E] 4	departureDate=May 21, 2016	java.util.LinkedHashMap\$Entry
► [E] 5	emptySeats=0	java.util.LinkedHashMap\$Entry
► [E] 6	totalSeats=200	java.util.LinkedHashMap\$Entry
► [E] 7	price=199	java.util.LinkedHashMap\$Entry
► [E] 8	planeType=Boeing 737	java.util.LinkedHashMap\$Entry
▼ [E] Variables		java.util.HashMap
► [E] 0	outboundHeaders={}	java.util.HashMap\$Node
► [E] 1	httpStatus=201	java.util.HashMap\$Node
► [E] 2	DWoutput=org.mule.runtime.core.internal.streaming.bytes.ManagedCursorStreamProvider@198ee533	java.util.HashMap\$Node

52. Step through the rest of the application and switch perspectives.

Review the Transform Message XML code

53. Right-click the Transform Message component and select Go to XML.

54. Locate and review the code for both DataWeave transformations.

```
<flow name="postFlight" doc:id="493369a6-2d0d-4dfd-a241-210504c4d5eb" >
    <ee:transform doc:name="Transform Message" doc:id="be12bc7d-22f8-49...
        <ee:message >
            <ee:set-payload ><! [CDATA[%dw 2.0
output application/java
---
payload]]></ee:set-payload>
        </ee:message>
        <ee:variables >
            <ee:set-variable variableName="DWoutput" ><! [CDATA[%dw 2.0
output application/json
---
payload]]></ee:set-variable>
            </ee:variables>
        </ee:transform>
        <logger level="INFO" doc:name="Logger" doc:id="03c0bf95-38e9-45d2-a...
    </flow>
```

55. Switch back to the Message Flow view.

Save a DataWeave script to an external file

56. In the Transform Message Properties view, make sure the payload output expression is selected.

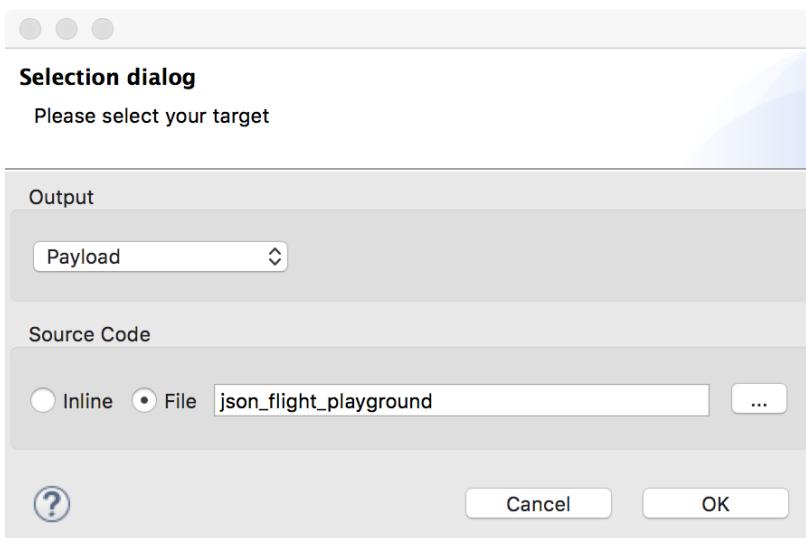
57. Click the Edit current target button (the pencil) above the code.

Output Payload ▾  

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload
```

58. In the Selection dialog box, change the source code selection from inline to file.

59. Set the file name to json_flight_playground.



60. Click OK.

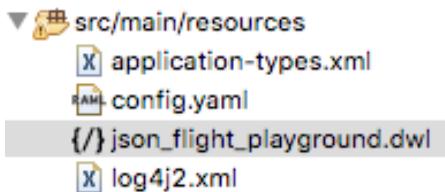
Locate and review the code that uses an external DataWeave script

61. Switch to Configuration XML view.

62. Locate the new code for the transformation in postFlight.

```
<flow name="postFlight" doc:id="847ead89-f898-4cd5-9ec0-1a1fdde0ed6d" >
    <ee:transform doc:name="Transform Message" doc:id="f963e3e4-5b3a-411
        <ee:message >
            <ee:set-payload resource="json_flight_playground.dwl" />
        </ee:message>
        <ee:variables >
            <ee:set-variable variableName="DWoutput" ><! [CDATA[%dw 2.0
output application/json
---
payload]]></ee:set-variable>
        </ee:variables>
    </ee:transform>
    <logger level="INFO" doc:name="Logger" doc:id="360ce24e-4dd3-4cef-bc
</flow>
```

63. In the Package Explorer, expand src/main/resources.



64. Open json_flight_playground.dwl.

65. Review and then close the file.

```
1 %dw 2.0
2 output application/java
3 ---
4 payload
```

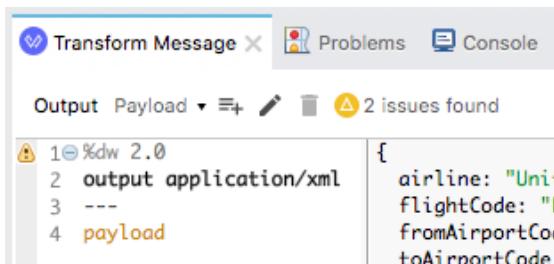
66. Return to Message Flow view in implementation.xml.

Change the output type to XML and review errors

67. In the Transform Message properties view, select the Variable – DWoutput output.

68. Change the output type from application/json to application/xml.

69. Locate the warning icons indicating that there is a problem.



The screenshot shows the 'Transform Message' editor in Mule Studio. At the top, there are tabs for 'Transform Message', 'Problems', and 'Console'. Below the tabs, there are buttons for 'Output', 'Payload', and 'Edit sample data'. A message box indicates '2 issues found'. The code area contains the following XML:

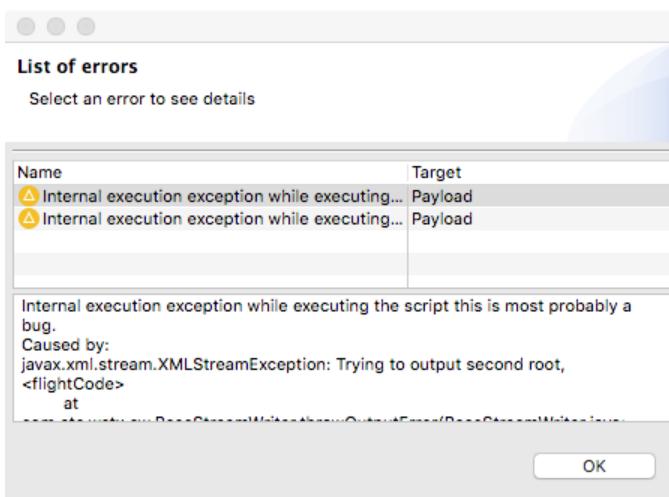
```
1 %dw 2.0
2 output application/xml
3 ---
4 payload
```

On the right side, a preview pane shows a JSON object:

```
{airline: "Unit", flightCode: "E", fromAirportCode: null, toAirportCode: null}
```

Note: If you do not see the warnings, click the Preview button. If you do not see a data preview, select the payload tab in the input section. If you don't see the payload tab, right-click payload in the input section and select Edit sample data.

70. Mouse over the icon located to the left of the code; you should see a message that there was an exception trying to output the second root <flightCode>.
71. Click the icon above the code; a List of errors dialog box should open.
72. Select and review the first issue.



73. In the List of errors dialog box, click OK.

Note: You will learn how to successfully transform to XML in the next walkthrough.

Test the application

74. Save the file to redeploy the project in debug mode.
75. In Advanced REST Client, post the same request to <http://localhost:8081/api/flights>.

76. In the Mule Debugger, step once; you should get an error.
77. Expand Exception and review the error information; you should see there is a DataWeave error when trying to output the second root.

The screenshot shows two perspectives of the Mule application. The top part is the 'Mule Debugger' perspective, which displays a table of exception details. The exception is a 'Internal execution exception while executing the script this is most probably a bug.' It was caused by a 'javax.xml.stream.XMLStreamException: Trying to output second root, <flightCode>' at 'com.ctc.wstx.sw.BaseStreamWriter.throwOutputError(BaseStreamWriter.java:1564)'. The bottom part is the 'implementation' perspective, showing a process flow named 'postFlight'. It consists of a 'Source' component followed by a 'Transform Message' component (which has a red dashed box around it), and finally a 'Logger' component.

Name	Value
Exception	<p>① description Internal execution exception while executing the script this is most probably a bug.</p> <p>② detailedDescription Internal execution exception while executing the script this is most probably a bug.</p> <p>③ errors []</p> <p>④ errorType MULE:EXPRESSION</p> <p>"Internal execution exception while executing the script this is most probably a bug. Caused by: javax.xml.stream.XMLStreamException: Trying to output second root, <flightCode> at com.ctc.wstx.sw.BaseStreamWriter.throwOutputError(BaseStreamWriter.java:1564)</p>

implementation global

postFlight

Source → Transform Message → Logger

78. Step to the end of the application and switch perspectives.

Walkthrough 11-2: Transform basic JSON, Java, and XML data structures

In this walkthrough, you continue to work with the JSON flight data posted to the flow. You will:

- Write scripts to transform the JSON payload to various JSON and Java structures.
- Write scripts to transform the JSON payload to various XML structures.

The screenshot shows a transformation script in the left pane and its XML preview in the right pane. The script is as follows:

```
1@%dw 2.0
2 output application/xml
3 ---
4@ data: {
5@   hub: "MUA",
6@   flight @{(airline: payload.airline): {
7@     code: payload.toAirportCode
8@   }
9@ }}
```

The XML preview shows the transformed structure:

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="United">
    <code>SFO</code>
  </flight>
</data>
```

Write expressions to transform JSON to various Java structures

1. Return to the Transform Message properties view for the transformation in postFlight.
2. In the output drop-down menu, select Payload.
3. Change the output type from java to json.
4. Type a period after payload and select price from the auto-completion menu.

The screenshot shows a transformation script with the payload auto-completion dropdown open. The payload object contains various properties like airline, flightCode, fromAirportCode, etc. The 'price' property is highlighted in the dropdown.

```
1@%dw 2.0
2 output application/json
3 ---
4 payload.
```

5. Look at the preview.

The screenshot shows the transformation preview. The output is a simple JSON object with a single key 'price' containing the value '199'.

```
1@%dw 2.0
2 output application/json
3 ---
4 payload.price
```

6. Change the output type from json to java.

The screenshot shows the transformation preview for Java output. The output is a JSON object with two keys: '199 as Number {class: "java.lang.Integer", encoding: "UTF-8", mimeType: "*/*", raw: 199}' and '199 as Number {class: "java.lang.Integer"}'.

```
1@%dw 2.0
2 output application/java
3 ---
4 payload.price
```

7. Change the DataWeave expression to data: payload.price.

Output Payload ▾ 2 issues found Preview

```
1@%dw 2.0
2   output application/java
3   ---
4   data: payload.price
```

```
{  
    data: 199 as Number {class: "java.lang.Integer"}  
} as Object {class: "java.util.LinkedHashMap",  
encoding: "UTF-8", mimeType: "*/*", raw: {  
    data: 199 as Number {class: "java.lang.Integer"}  
} as Object {class: "java.util.LinkedHashMap"}}}
```

8. Change the output type from java to json.

Output Payload ▾ 2 issues found Preview

```
1@%dw 2.0
2   output application/json
3   ---
4   data: payload.price
```

```
{  
    "data": 199  
}
```

9. Change the DataWeave expression to data: payload.

Output Payload ▾ 2 issues found Preview

```
1@%dw 2.0
2   output application/json
3   ---
4   data: payload
```

```
{  
    "data": {  
        "airline": "United",  
        "flightCode": "ER38sd",  
        "fromAirportCode": "LAX",  
        "toAirportCode": "SFO",  
        "departureDate": "May 21, 2016",  
        "emptySeats": 0,  
        "totalSeats": 200,  
        "price": 199,  
        "planeType": "Boeing 737"  
    }  
}
```

10. Change the output type from json to java.

Output Payload ▾ 2 issues found Preview

```
1@%dw 2.0
2   output application/java
3   ---
4   data: payload
```

```
{  
    data: {  
        airline: "United" as String {class: "java.lang.String"},  
        flightCode: "ER38sd" as String {class: "java.lang.String"},  
        fromAirportCode: "LAX" as String {class: "java.lang.String"},  
        toAirportCode: "SFO" as String {class: "java.lang.String"},  
        departureDate: "May 21, 2016" as String {class:  
"java.lang.String"},  
        emptySeats: 0 as Number {class: "java.lang.Integer"},  
        totalSeats: 200 as Number {class: "java.lang.Integer"},  
        price: 199 as Number {class: "java.lang.Integer"},  
        planeType: "Boeing 737" as String {class: "java.lang.String"}  
    } as Object {class: "java.util.LinkedHashMap"}  
}
```

11. Change the DataWeave expression to data: {}.

12. Add a field called hub and set it to "MUA".

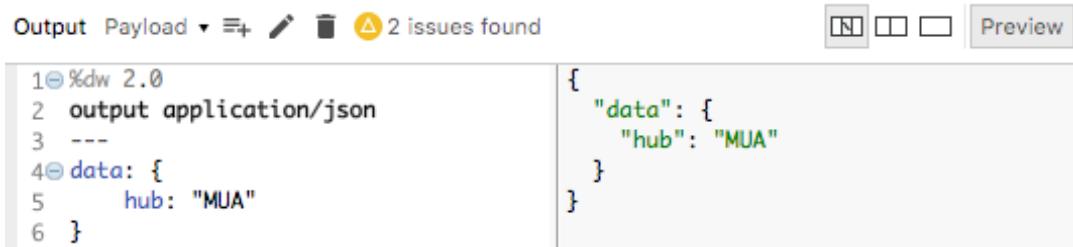


Output Payload ▾ 2 issues found

```
1@%dw 2.0
2  output application/java
3  ---
4@ data: {
5     hub: "MUA"
6 }
```

The code defines a DataWeave script version 2.0. It outputs application/java. The payload is a variable named 'data' which contains a key 'hub' with the value 'MUA'. The output is a LinkedHashMap object with 'hub' as a String and the rest as a LinkedHashMap.

13. Change the output type from java to json.



Output Payload ▾ 2 issues found

```
1@%dw 2.0
2  output application/json
3  ---
4@ data: {
5     hub: "MUA"
6 }
```

The code defines a DataWeave script version 2.0. It outputs application/json. The payload is a variable named 'data' which contains a key 'hub' with the value 'MUA'. The output is a JSON object with 'data' as a key containing a single entry 'hub' with value 'MUA'.

14. Add a field called code and use auto-completion to set it to the toAirportCode property of the payload.

15. Add a field called airline and set it to the airline property of the payload.



Output Payload ▾ 2 issues found

```
1@%dw 2.0
2  output application/json
3  ---
4@ data: {
5     hub: "MUA",
6     code: payload.toAirportCode,
7     airline: payload.airline
8 }
```

The code defines a DataWeave script version 2.0. It outputs application/json. The payload is a variable named 'data' which contains keys 'hub', 'code', and 'airline'. The 'code' value is set to the 'toAirportCode' property of the payload, and the 'airline' value is set to the 'airline' property of the payload. The output is a JSON object with 'data' as a key containing three entries: 'hub', 'code', and 'airline'.

Write an expression to output data as XML

16. In the output drop-down menu, select Variable – DW output.

17. Change the DataWeave expression to data: payload.

18. Look at the preview; you should now see XML.

Output Variable - DWoutput ▾  

1@%dw 2.0
2 output application/xml
3 ---
4 data: payload

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <airline>United</airline>
  <flightCode>ER38sd</flightCode>
  <fromAirportCode>LAX</fromAirportCode>
  <toAirportCode>SFO</toAirportCode>
  <departureDate>May 21, 2016</departureDate>
  <emptySeats>0</emptySeats>
  <totalSeats>200</totalSeats>
  <price>199</price>
  <planeType>Boeing 737</planeType>
</data>
```

Preview

19. In the output drop-down menu, select Payload.

20. Copy the DataWeave expression.

21. Switch back to Variable – DWoutput and replace the DataWeave expression with the one you just copied.

Output Variable - DWoutput ▾  

1@%dw 2.0
2 output application/xml
3 ---
4@ data: {
5@ hub: "MUA",
6@ code: payload.toAirportCode
7@ airline: payload.airline
8 }

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <code>SFO</code>
  <airline>United</airline>
</data>
```

Preview

22. Modify the expression so the code and airline properties are child elements of a new element called flight.

Output Variable - DWoutput ▾  

1@%dw 2.0
2 output application/xml
3 ---
4@ data: {
5@ hub: "MUA",
6@ flight: {
7@ code: payload.toAirportCode,
8@ airline: payload.airline
9@ }
10 }

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight>
    <code>SFO</code>
    <airline>United</airline>
  </flight>
</data>
```

Preview

23. Modify the expression so the airline is an attribute of the flight element.

Output Variable - DWoutput ▾  

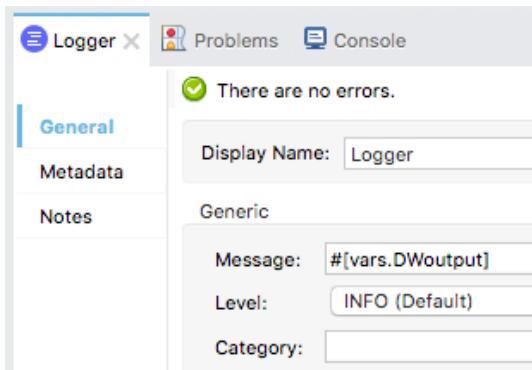
1@%dw 2.0
2 output application/xml
3 ---
4@ data: {
5@ hub: "MUA",
6@ flight @(airline: payload.airline): {
7@ code: payload.toAirportCode
8@ }
9 }

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="United">
    <code>SFO</code>
  </flight>
</data>
```

Preview

Debug the application

24. In the Logger properties view, set the message to the value of the DWoutput variable.



25. Save the file to redeploy the project in debug mode.

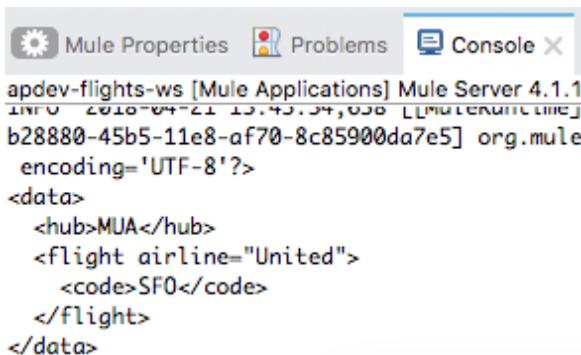
26. In Advanced REST Client, post the same request to <http://localhost:8081/flights>.

27. In the Mule Debugger, step to the Logger; you should see the transformed JSON payload and the DWoutput variable as a StreamProvider.

Name	Value
Attributes	org.mule.extension.http.api.HttpRequestAttributes
Component Path	postFlight/processors/1
DataType	SimpleDataType{type=org.mule.runtime.core.internal.streaming.bytes.ManagedCursorStreamProvider, r
Encoding	UTF-8
Message	
Payload (mimeType="application/json";...)	{ "data": { "hub": "MUA", "code": "SFO", "airline": "United" } }
Variables	size = 3 0: outboundHeaders={} 1: httpStatus=201 2: DWoutput=org.mule.runtime.core.internal.streaming.bytes.ManagedCursorStreamProvider@41b47dd8

28. Step through the application and switch perspectives.

29. Look at the Logger output in the console; you should see the XML.



Walkthrough 11-3: Transform complex data structures with arrays

In this walkthrough, you create a new flow that allows multiple flights to be posted to it, so you have more complex data with which to work. You will:

- Create a new flow that receives POST requests of a JSON array of flight objects.
- Transform a JSON array of objects to DataWeave, JSON, and Java.

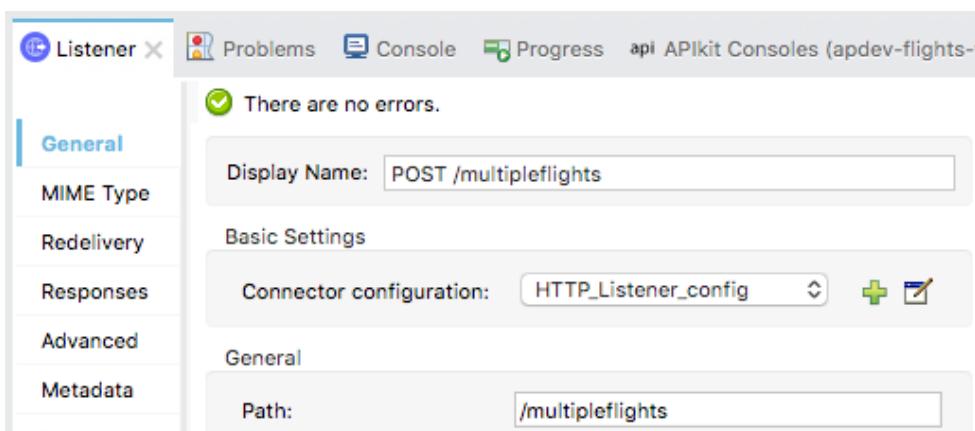


```
1@%dw 2.0
2  output application/dw
3  ---
4@ payload map (object,index) -> {
5   'flight$(index)': object
6 }
```

```
[ { flight0: { airline: "United", flightCode: "ER38sd", fromAirportCode: "LAX", toAirportCode: "SFO", departureDate: "May 21, 2016", emptySeats: 0, totalSeats: 200, price: 199, planeType: "Boeing 737" } }, { flight1: { airline: "Delta", ... } } ]
```

Create a new flow that receives POST requests of a JSON array of flight objects

1. Return to implementation.xml.
2. Drag out an HTTP Listener from the Mule Palette to the bottom of the canvas.
3. Change the flow name to postMultipleFlights.
4. In the HTTP Listener properties view, set the display name to POST /multipleflights.
5. Set the connector configuration to the existing HTTP_Listener_config.
6. Set the path to /multipleflights and set the allowed methods to POST.



Note: You are bypassing the APIkit router because a corresponding resource/method pair is not defined in the API.

7. Add a Transform Message component to the flow.
8. In the Transform Message properties view, set the expression to the payload.

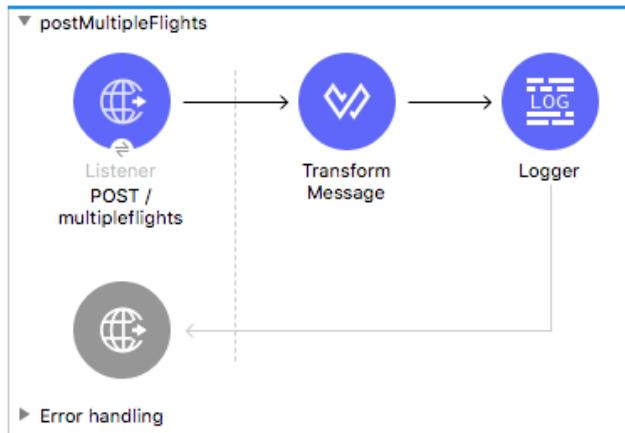
Output Payload ▾ Preview

```

1 %dw 1.0
2 %output application/java
3 ---
4 payload

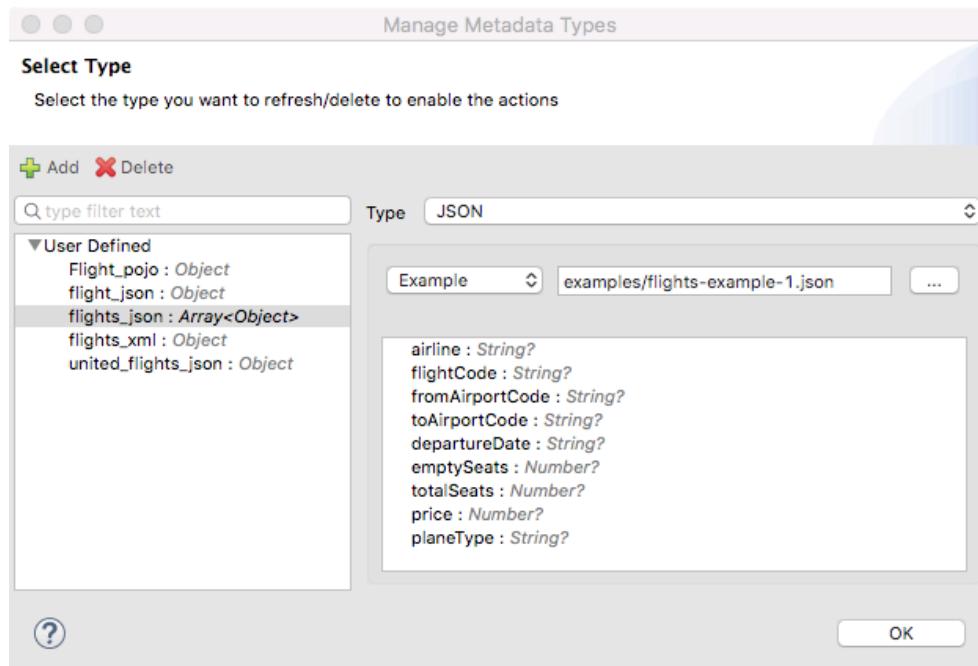
```

9. Add a Logger to the flow.

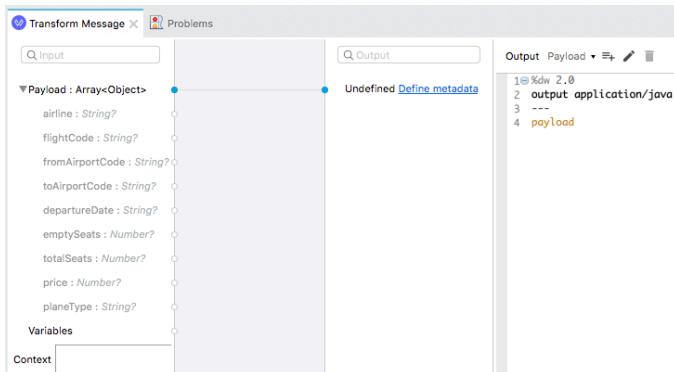


Add input metadata for the transformation

10. In the Transform Message properties view, click Define metadata in the input section.
11. In the Select metadata type dialog box, select flights_json and click Select.



12. In the Transform Message Properties view, you should now see metadata for the input.

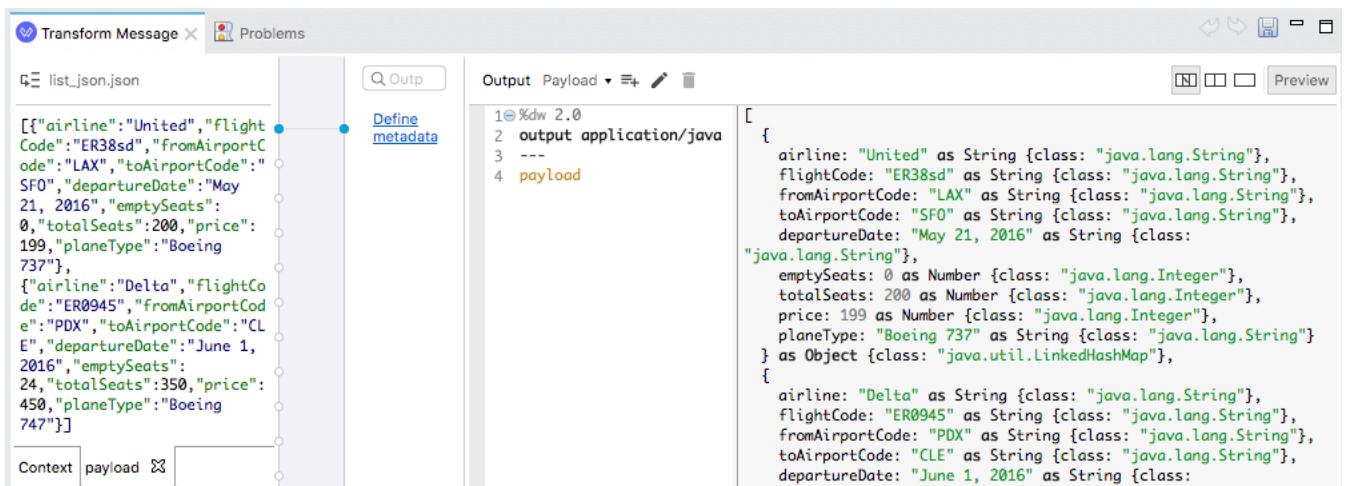


Preview sample data and sample output

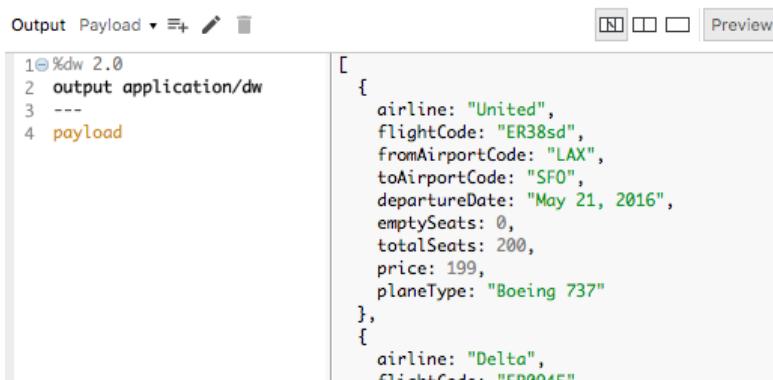
13. Click the Preview button.

14. In the preview section, click the Create required sample data to execute preview link.

15. Look at the preview section; the sample output should be an ArrayList of LinkedHashMaps.



16. Change the output type to application/dw.



Return values for the first object in the collection

17. Change the DataWeave expression to payload[0].

The screenshot shows the DataWeave editor interface. The payload is defined as a JSON object with various flight details. The expression being evaluated is payload[0], which is highlighted in orange.

```
1@%dw 2.0
2 output application/dw
3 ---
4 payload[0]
```

```
{
  airline: "United",
  flightCode: "ER38sd",
  fromAirportCode: "LAX",
  toAirportCode: "SFO",
  departureDate: "May 21, 2016",
  emptySeats: 0,
  totalSeats: 200,
  price: 199,
  planeType: "Boeing 737"
}
```

18. Change the DataWeave expression to payload[0].price.

The screenshot shows the DataWeave editor interface. The expression payload[0].price is highlighted in orange and results in the value 199.

```
1@%dw 2.0
2 output application/dw
3 ---
4 payload[0].price
```

```
199
```

19. Change the DataWeave expression to payload[0].*price.

The screenshot shows the DataWeave editor interface. The expression payload[0].*price is highlighted in orange and results in the array [199].

```
1@%dw 2.0
2 output application/dw
3 ---
4 payload[0].*price
```

```
[199]
```

Return collections of values

20. Change the DataWeave expression to return a collection of all the prices.

The screenshot shows the DataWeave editor interface. The expression payload.*price is highlighted in orange and results in the array [199, 450].

```
1@%dw 2.0
2 output application/dw
3 ---
4 payload.*price
```

```
[199,
 450]
```

21. Change the DataWeave expression to payload.price; you should get the same result.

The screenshot shows the DataWeave editor interface. The expression payload.price is highlighted in orange and results in the array [199, 450].

```
1@%dw 2.0
2 output application/dw
3 ---
4 payload.price
```

```
[199,
 450]
```

22. Change the DataWeave expression to return a collection of prices and available seats.

```
[payload.price, payload.emptySeats]
```

The screenshot shows the DataWeave editor interface. The left pane contains the DW script:

```
1@ %dw 2.0
2 output application/dw
3 ---
4 [payload.price, payload.emptySeats]
```

The right pane shows the preview of the resulting JSON array:

```
[ [ 199, 450 ], [ 0, 24 ] ]
```

Use the map operator to return object collections with different data structures

23. Change the DataWeave expression to payload map \$.

The screenshot shows the DataWeave editor interface. The left pane contains the DW script:

```
1@ %dw 2.0
2 output application/dw
3 ---
4 payload map $
```

The right pane shows the preview of the resulting array of objects:

```
[ { airline: "United", flightCode: "ER38sd", fromAirportCode: "LAX", toAirportCode: "SFO", departureDate: "May 21, 2016", emptySeats: 0, totalSeats: 200, price: 199, planeType: "Boeing 737" }, { airline: "Delta", flightCode: "DIAAWE" } ]
```

24. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.

The screenshot shows the DataWeave editor interface. The left pane contains the DW script:

```
1@ %dw 2.0
2 output application/dw
3 ---
4@ payload map {
5   flight: $$
6 }
```

The right pane shows the preview of the resulting array of objects with the 'flight' property added:

```
[ { flight: 0 }, { flight: 1 } ]
```

25. Change the DataWeave expression to create a property called destination for each object that is equal to the value of the toAirportCode field in the payload collection.

Output Payload ▾  

```
1@%dw 2.0
2  output application/dw
3  ---
4@payload map {
5@    flight: $$,
6      destination: $.toAirportCode
7 }
```

[
 {
 flight: 0,
 destination: "SFO"
 },
 {
 flight: 1,
 destination: "CLE"
 }
]

26. Change the DataWeave expression to dynamically add each of the properties present on the payload objects.

Output Payload ▾  

```
1@%dw 2.0
2  output application/dw
3  ---
4@payload map {
5@    flight: $$,
6      ($$): $  
7 }
```

[
 {
 flight: 0,
 "0": {
 airline: "United",
 flightCode: "ER38sd",
 fromAirportCode: "LAX",
 toAirportCode: "SFO",
 departureDate: "May 21, 2016",
 emptySeats: 0,
 totalSeats: 200,
 price: 199,
 planeType: "Boeing 737"
 }
 },
 {
 flight: 1,
 }

27. Remove the first flight property assignment.

28. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.

Output Payload ▾  

```
1@%dw 2.0
2  output application/dw
3  ---
4@payload map {
5@    'flight$$': $  
6 }
```

[
 {
 flight0: {
 airline: "United",
 flightCode: "ER38sd",
 fromAirportCode: "LAX",
 toAirportCode: "SFO",
 departureDate: "May 21, 2016",
 emptySeats: 0,
 totalSeats: 200,
 price: 199,
 planeType: "Boeing 737"
 }
 },
 {
 flight1: {
 }

Use explicit named parameters with the map operator

29. Change the map operator so that it uses two parameters called object and index and set the field name to just the index value.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/dw
3 ---
4@payload map (object,index) -> {
5     (index): object
6 }
```

The resulting payload is a JSON array containing two objects, indexed 0 and 1:

```
[{"0": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}, "1": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}}
```

30. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/dw
3 ---
4@payload map (object,index) -> {
5     'flight${index}': object
6 }
```

The resulting payload is a JSON array containing three objects, indexed 0, 1, and 2, with names flight0, flight1, and flight2 respectively:

```
[{"flight0": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}, "flight1": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}, "flight2": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}}
```

Note: If you want to test the application, change the output type to application/json and then in Advanced REST Client, make a request to <http://localhost:8081/multipleflights> and replace the request body with JSON from the flights-example.json file.

Change the expression to output different data types

31. Change the DataWeave expression output type from application/dw to application/json.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/json
3 ---
4@payload map (object,index) -> {
5     'flight$(index)': object
6 }
```

The resulting payload is a JSON array:

```
[{"flight0": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}}
```

32. Change the output type to application/java.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/java
3 ---
4@payload map (object,index) -> {
5     'flight$(index)': object
6 }
```

The resulting payload is a Java LinkedHashMap:

```
[{"flight0": {"airline": "United" as String {class: "java.lang.String"}, "flightCode": "ER38sd" as String {class: "java.lang.String"}, "fromAirportCode": "LAX" as String {class: "java.lang.String"}, "toAirportCode": "SFO" as String {class: "java.lang.String"}, "departureDate": "May 21, 2016" as String {class: "java.lang.String"}, "emptySeats": 0 as Number {class: "java.lang.Integer"}, "totalSeats": 200 as Number {class: "java.lang.Integer"}, "price": 199 as Number {class: "java.lang.Integer"}, "planeType": "Boeing 737" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"}, {"flight1": {}}]
```

33. Change the output type to application/xml; you should get an issue displayed.

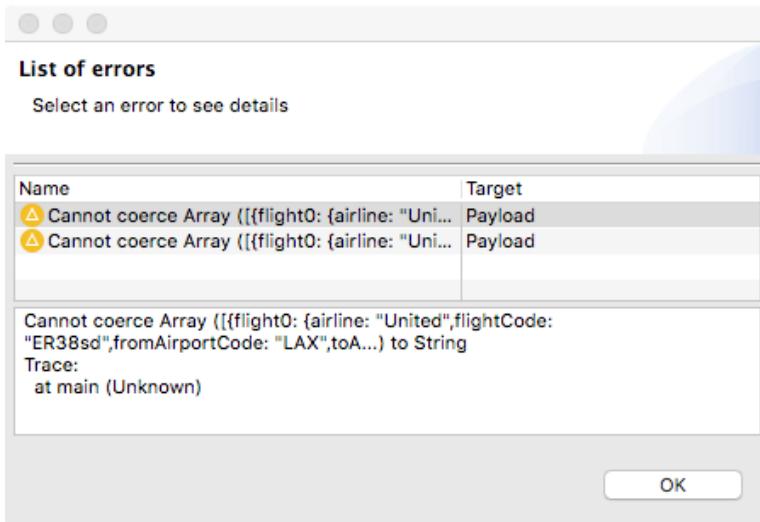
The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/xml
3 ---
4@payload map (object,index) -> {
5     'flight$(index)': object
6 }
```

A warning icon is shown next to the first line of code, indicating an issue. The resulting payload is an XML fragment:

```
[{"flight0": {"airline", "flight", "fromAir", "toAirr"}}
```

34. Click the warning icon and review the issues.

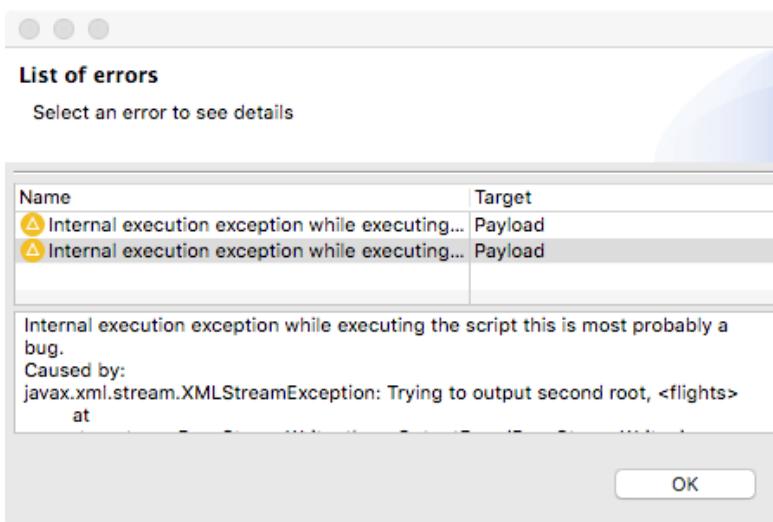


35. Change the DataWeave expression to create a root node called flights; you should still get an issue.

Output Payload ▾ ⚡ 2 issues found

```
1 %dw 2.0
2 output application/xml
3 ---
4 flights: payload map (object,index) -> {
5   'flight$(index)': object
6 }
```

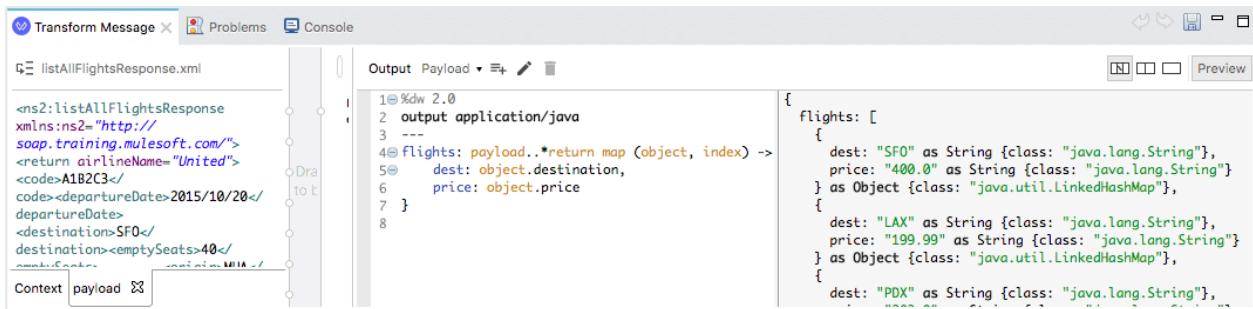
36. Click the warning icon and review the issues.



Walkthrough 11-4: Transform to and from XML with repeated elements

In this walkthrough, you continue to work with the JSON data for multiple flights posted to the flow. You will:

- Transform a JSON array of objects to XML.
- Replace sample data associated with a transformation.
- Transform XML with repeated elements to different data types.



Transform the JSON array of objects to XML

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Change the expression to map each item in the input array to an XML element.

```
flights: {(payload map (object,index) -> {
  'flight$(index)': object
})}
```

3. Look at the preview; the JSON should be transformed to XML successfully.



4. Modify the expression so the flights object has a single property called flight.

```

1@%dw 2.0
2  output application/xml
3  ---
4@ flights: {payload map (object,index) -> {
5    flight: object
6  }
7}}

```

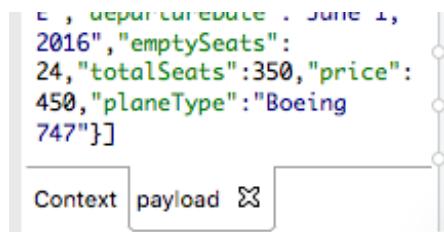
```

<?xml version='1.0' encoding='UTF-8'?>
<flights>
<flight>
<airline>United</airline>
<flightCode>ER38sd</flightCode>
<fromAirportCode>LAX</fromAirportCode>
<toAirportCode>SFO</toAirportCode>
<departureDate>May 21, 2016</departureDate>
<emptySeats>0</emptySeats>
<totalSeats>200</totalSeats>
<price>199</price>
<planeType>Boeing 737</planeType>
</flight>
<flight>
<airline>Delta</airline>
</flight>
</flights>

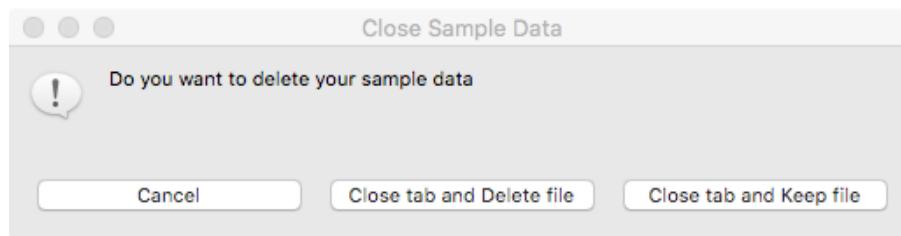
```

Change the input metadata to XML

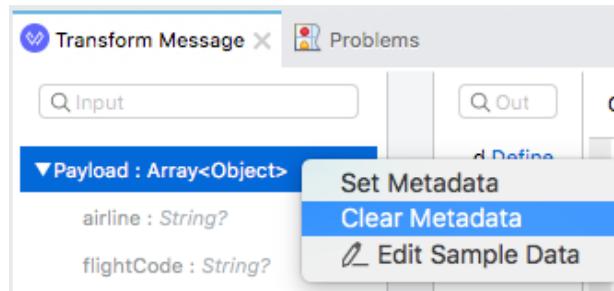
5. In the input section, click the x on the payload tab to close it.



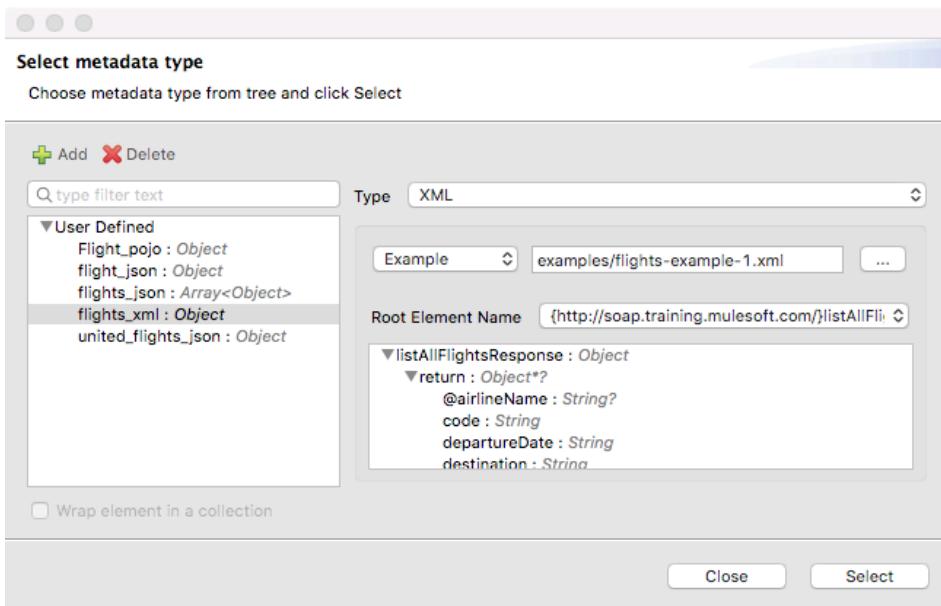
6. In the Close Sample Data dialog box, click Close tab and Keep file.



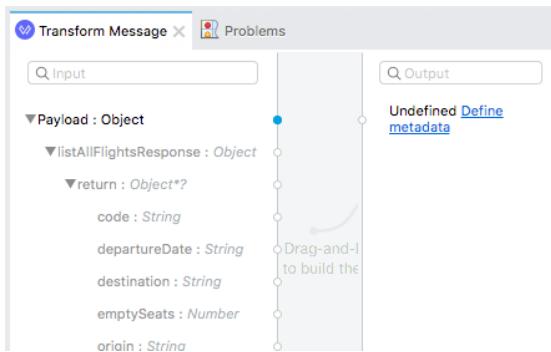
7. In the input section, right-click Payload and select Clear Metadata.



8. Click the Define metadata link.
 9. In the Select metadata type dialog box, select flights_xml and click Select.

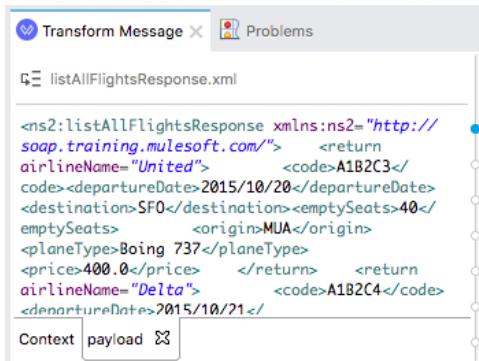


10. In the Transform Message Properties view, you should now see new metadata for the input.

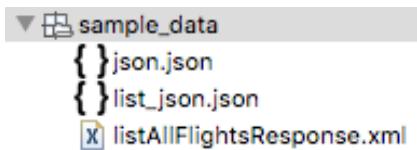


Preview sample data and sample output

11. In the preview section, click the Create required sample data to execute preview link.
 12. Look at the XML sample payload in the input section.



13. Look at the sample_data folder in src/test/resources.



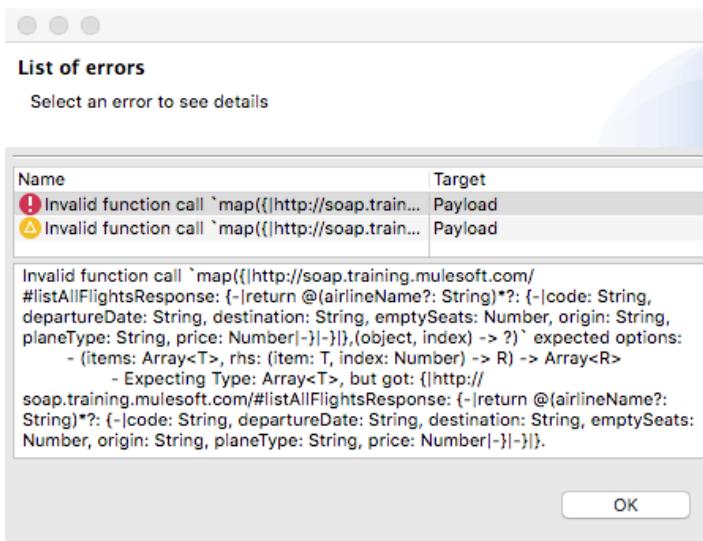
14. Review the transformation expression; you should get an error.

The screenshot shows the Mule Studio interface with a transformation step. The payload is set to application/xml. The transformation expression is:

```
1 %dw 2.0
2 output application/xml
3 ---
4 flights: {payload map (object, index) -> {
5   flight: object
6 }
7 }}
```

An error message is displayed: "X Invalid function call `map` expected options: - (items: Array<T>, rhs: (item: T, index: Number) -> R) -> Array<R> - Expecting Type: Array<T>, but got: {http://soap.training.mulesoft.com/#listAllFlightsResponse: {return @airlineName?: String}*?: {code: String, departureDate: String, destination: String, emptySeats: Number, origin: String, planeType: String, price: Number}-|}-} (object, index) -> ?`" with a yellow warning icon.

15. Look at the error.



Transform XML with repeated elements to a different XML structure

16. Look at the payload metadata in the input section.

The screenshot shows the "Transform Message" editor. The "Input" field is empty. The "Payload : Object" section is expanded, showing:

- listAllFlightsResponse : Object
 - return : Object*?
 - code : String
 - departureDate : String

17. Change the DataWeave expression so that the map is iterating over payload..return.

Output Payload ▾   Preview

```
1@ %dw 2.0
2  output application/xml
3  ---
4@ flights: {$(payload..return map (object,index) -> {
5    flight: object
6  })
7 })
```

<?xml version='1.0' encoding='UTF-8'?>
<flights>
<flight>
<code>A1B2C3</code>
<departureDate>2015/10/20</departureDate>
<destination>SFO</destination>
<emptySeats>40</emptySeats>
<origin>MUA</origin>
<planeType>Boing 737</planeType>
<price>400.0</price>
</flight>
</flights>

18. Change the DataWeave expression so that the map is iterating over the repeated return elements of the input.

Output Payload ▾   Preview

```
1@ %dw 2.0
2  output application/xml
3  ---
4@ flights: {$(payload..*return map (object,index) -> {
5    flight: object
6  })
7 })
```

<?xml version='1.0' encoding='UTF-8'?>
<flights>
<flight>
<code>A1B2C3</code>
<departureDate>2015/10/20</departureDate>
<destination>SFO</destination>
<emptySeats>40</emptySeats>
<origin>MUA</origin>
<planeType>Boing 737</planeType>
<price>400.0</price>
</flight>
<flight>
<code>A1B2C4</code>
<departureDate>2015/10/21</departureDate>

19. Change the DataWeave expression to return flight elements with dest and price elements that map to the corresponding destination and price input values.

Output Payload ▾   Preview

```
1@ %dw 2.0
2  output application/xml
3  ---
4@ flights: {$(payload..*return map (object,index) -> {
5@   flight: [
6    dest: object.destination,
7    price: object.price
8  ]
9 })
10 })
```

<?xml version='1.0' encoding='UTF-8'?>
<flights>
<flight>
<dest>SFO</dest>
<price>400.0</price>
</flight>
<flight>
<dest>LAX</dest>
<price>199.99</price>
</flight>
</flights>

Note: If you want to test the application with Advanced REST Client, be sure to change the content-type header to application/XML and replace the request body with XML from the flights-example.xml file.

Change the expression to output different data types

20. Change the output type from application/xml to application/dw.



```
1@ %dw 2.0
2  output application/dw
3  ---
4@ flights: {$(payload..*)return map (object,index) -> {
5@   flight: {
6@     dest: object.destination,
7@     price: object.price
8@   }
9@ }
10 )}
```

```
{ flights: [
  flight: {
    dest: "SFO",
    price: "400.0"
  },
  flight: {
    dest: "LAX",
    price: "199.99"
  }
], ...}
```

21. Change the output type to application/json.



```
1@ %dw 2.0
2  output application/json
3  ---
4@ flights: {$(payload..*)return map (object,index) -> {
5@   flight: {
6@     dest: object.destination,
7@     price: object.price
8@   }
9@ }
10 )}
```

```
{ "flights": [
  "flight": {
    "dest": "SFO",
    "price": "400.0"
  },
  "flight": {
    "dest": "LAX",
    "price": "199.99"
  }
], ...}
```

22. Change the output type to application/java.



```
1@ %dw 2.0
2  output application/java
3  ---
4@ flights: {$(payload..*)return map (object,index) -> {
5@   flight: {
6@     dest: object.destination,
7@     price: object.price
8@   }
9@ }
10 )}
```

```
{ flights: [
  flight: {
    dest: "PDX" as String {class: "java.lang.String"},
    price: "283.0" as String {class: "java.lang.String"}
  } as Object {class: "java.util.LinkedHashMap"}
] as Object {class: "java.util.LinkedHashMap"}
} as Object {class: "java.util.LinkedHashMap", encoding: "UTF-8", mimeType: "*/*", raw: [
  flights: [
    flight: {
      dest: "PDX" as String {class: "java.lang.String"},
      price: "283.0" as String {class: "java.lang.String"}
    } as Object {class: "java.util.LinkedHashMap"}
  ] as Object {class: "java.util.LinkedHashMap"}
] as Object {class: "java.util.LinkedHashMap"}}
], ...}
```

23. In the DataWeave expression, remove the {} around the map expression.

Output Payload ▾ dw Preview

```
1@%dw 2.0
2 output application/java
3 ---
4@ flights: payload..*return map (object,index) -> {
5@   flight: {
6@     dest: object.destination,
7@     price: object.price
8@   }
9}
10
```

{
flights: [
 {
flight: {
dest: "SFO" as String {class: "java.lang.String"},
price: "400.0" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"}
} as Object {class: "java.util.LinkedHashMap"},
 {
flight: {
dest: "LAX" as String {class: "java.lang.String"},
price: "199.99" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"}
} as Object {class: "java.util.LinkedHashMap"},
 {

24. Change the DataWeave expression to remove the flight property; you should get an ArrayList with five LinkedHashMaps.

Output Payload ▾ dw Preview

```
1@%dw 2.0
2 output application/java
3 ---
4@ flights: payload..*return map (object,index) -> {
5@   dest: object.destination,
6@   price: object.price
7 }
8
```

{
flights: [
 {
dest: "SFO" as String {class: "java.lang.String"},
price: "400.0" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"},
 {
dest: "LAX" as String {class: "java.lang.String"},
price: "199.99" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"},
 {
dest: "PDX" as String {class: "java.lang.String"},
price: "283.0" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"},
 {
dest: "PDX" as String {class: "java.lang.String"},
price: "283.0" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"},
 {
dest: "PDX" as String {class: "java.lang.String"},
price: "283.0" as String {class: "java.lang.String"}
} as Object {class: "java.util.LinkedHashMap"}
] as Array {class: "java.util.ArrayList"}
} as Object {class: "java.util.LinkedHashMap", encoding:

Walkthrough 11-5: Define and use variables and functions

In this walkthrough, you continue to work with the DataWeave transformation in postMultipleFlights. You will:

- Define and use a global constant.
- Define and use a global variable that is equal to a lambda expression.
- Define and use a lambda expression assigned to a variable as a function.
- Define and use a local variable.



The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2  output application/dw
3
4@fun getNumSeats (planeType: String) =
5@  if (planeType contains('737'))
6@    150
7@  else
8@    300
9 ---
10@using (flights =
11@  payload..*return map (object,index) -> {
12@    dest: object.destination,
13@    price: object.price,
14@    totalSeats: getNumSeats(object.planeType as String),
15@    plane: object.planeType
16@  }
17@)
18@flights
```

The output pane displays the resulting JSON array:

```
[{"dest": "SFO", "price": "400.0", "totalSeats": 150, "plane": "Boing 737"}, {"dest": "LAX", "price": "199.99", "totalSeats": 150, "plane": "Boing 737"}, {"dest": "PDX", "price": "283.0", "totalSeats": 300, "plane": "Boing 777"}]
```

Define and use a global constant

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Change the output type to application/dw.
3. In the header, define a global variable called numSeats that is equal to 400.

```
var numSeats = 400
```

4. In the body, add a totalSeats property that is equal to the numSeats constant.

```
totalSeats: numSeats
```

```
4  var numSeats = 400
5  ---
6@flights: payload..*return map (object,index) -> {
7@  dest: object.destination,
8@  price: object.price,
9@  totalSeats: numSeats
10 }
```

5. Look at the preview.

Output Payload ▾   Preview

```
1@ %dw 2.0
2  output application/dw
3
4  var numSeats = 400
5  ---
6@ flights: payload..*return map (object,index) -> {
7@   dest: object.destination,
8@   price: object.price,
9@   totalSeats: numSeats
10 }
11
```

{
 flights: [
 {
 dest: "SFO",
 price: "400.0",
 totalSeats: 400
 },
 {
 dest: "LAX",
 price: "199.99",
 totalSeats: 400
 },
]
}

6. Comment out the variable declaration in the header.

```
//var numSeats = 400
```

Define and use a global variable that is equal to a lambda expression that maps to a constant

7. In the header, define a global variable called numSeats that is equal to a lambda expression with an input parameter x equal to 400.
8. Inside the expression, set numSeats to x.

```
var numSeats = (x=400) -> x
```

9. Add parentheses after numSeats in the totalSeats property assignment.

```
totalSeats: numSeats()
```

10. Look at the preview.

Output Payload ▾   Preview

```
1@ %dw 2.0
2  output application/dw
3
4 //var numSeats = 400
5 var numSeats = (x = 400) -> x
6  ---
7@ flights: payload..*return map (object,index) -> {
8@   dest: object.destination,
9@   price: object.price,
10  totalSeats: numSeats()
11 }
12
```

{
 flights: [
 {
 dest: "SFO",
 price: "400.0",
 totalSeats: 400
 },
 {
 dest: "LAX",
 price: "199.99",
 totalSeats: 400
 },
]
}

Add a plane property to the output

11. Add a plane property to the DataWeave expression equal to the value of the input planeType values.
12. Look at the values of plane type in the preview.

The screenshot shows the DataWeave editor interface. The left pane contains the DW script:

```
1 %dw 2.0
2 output application/dw
3
4 //var numSeats = 400
5 var numSeats = (x = 400) -> x
6 ---
7 flights: payload..*return map (object,index) -> {
8   dest: object.destination,
9   price: object.price,
10  totalSeats: numSeats(),
11  plane: object.planeType
12 }
```

The right pane shows the preview of the resulting JSON output:

```
[{"dest": "LAX", "price": "199.99", "totalSeats": 400, "plane": "Boing 737"}, {"dest": "PDX", "price": "283.0", "totalSeats": 400, "plane": "Boing 777"}],
```

Define and use a global variable that is equal to a lambda expression with input parameters

13. Comment out the numSeats variable declaration in the header.
14. In the header, define a global variable called numSeats that is equal to a lambda expression with an input parameter called planeType of type String.

```
var numSeats = (planeType: String) ->
```

15. Inside the expression, add an if/else block that checks to see if planeType contains the string 737 and sets numSeats to 150 or 300.

```
6---- var numSeats = (planeType: String) ->
7      if (planeType contains('737'))
8          150
9      else
10         300
```

16. Change the totalSeats property assignment to pass the object's planeType property to numSeats.

```
totalSeats: numSeats(object.planeType)
```

17. Force the argument to a String.

```
totalSeats: numSeats(object.planeType as String)
```

18. Look at the preview; you should see values of 150 and 300.

The screenshot shows the Mule Studio interface. On the left, there is a code editor window titled "Output Payload" containing a Mule configuration file. The code defines a variable `numSeats` based on the `planeType`. If `planeType` contains '737', `numSeats` is 150; otherwise, it is 300. A function `flights` is defined to map objects to a map of destination, price, total seats, and plane type. The right side shows the "Preview" tab, which displays three flight records. The first flight has dest: "LAX", price: "400.0", totalSeats: 150, and plane: "Boing 737". The second flight has dest: "PDX", price: "199.99", totalSeats: 150, and plane: "Boing 737". The third flight has dest: "PDX", price: "283.0", totalSeats: 300, and plane: "Boing 777".

```
1@ %dw 2.0
2  output application/dw
3
4 //var numSeats = 400
5 //var numSeats = (x = 400) -> x
6@ var numSeats = (planeType: String) ->
7@   if (planeType contains('737'))
8     150
9   else
10    300
11 ---
12@ flights: payload..*return map (object,index) -> {
13@   dest: object.destination,
14@   price: object.price,
15@   totalSeats: numSeats(object.planeType as String),
16@   plane: object.planeType
17 }
```

```
price: "400.0",
totalSeats: 150,
plane: "Boing 737"
},
{
dest: "LAX",
price: "199.99",
totalSeats: 150,
plane: "Boing 737"
},
{
dest: "PDX",
price: "283.0",
totalSeats: 300,
plane: "Boing 777"
},
```

19. Surround the variable declaration in the header with /* and */ to comment it out.

Define and use a lambda expression assigned to a variable as a function

20. In the header, use the fun keyword to define a function called `getNumSeats` with a parameter called `planeType` of type String.

```
fun getNumSeats(planeType: String)
```

21. Copy the if/else expression in the `numSeats` declaration.

22. Set the `getNumSeats` function equal to the if/else expression.

```
fun getNumSeats(planeType: String) =
  if (planeType contains('737'))
    150
  else
    300
```

23. In the body, change the `totalSeats` property to be equal to the result of the `getNumSeats` function.

```
totalSeats: getNumSeats(object.planeType as String)
```

24. Look at the preview.

The screenshot shows the DataWeave editor interface. On the left, the code editor displays the following DataWeave script:

```
60 /* var numSeats = (planeType: String) ->
7     if (planeType contains('737'))
8         150
9     else
10        300
11 */
12 fun getNumSeats (planeType: String) =
13     if (planeType contains('737'))
14         150
15     else
16        300
17 ---
18 flights: payload..*return map (object,index) -> {
19     dest: object.destination,
20     price: object.price,
21     totalSeats: getNumSeats(object.planeType as String),
22     plane: object.planeType
23 }
```

On the right, the preview pane shows the resulting JSON output:

```
flights: [
{
    dest: "SFO",
    price: "400.0",
    totalSeats: 150,
    plane: "Boing 737"
},
{
    dest: "LAX",
    price: "199.99",
    totalSeats: 150,
    plane: "Boing 737"
},
{
    dest: "PDX",
    price: "283.0",
    totalSeats: 300,
    plane: "Boing 777"
}
```

Create and use a local variable

25. In the body, surround the existing DataWeave expression with parentheses and add the using keyword in front of it.

```
17 ---
18 using [flights: payload..*return map (object,index) -> {
19     dest: object.destination,
20     price: object.price,
21     totalSeats: getNumSeats(object.planeType as String),
22     plane: object.planeType
23 }
24 )
```

26. Change the colon after flights to an equal sign.

27. Modify the indentation to your liking.

28. After the local variable declaration, set the transformation expression to be the local flights variable.

```
18 using (flights =
19 payload..*return map (object,index) -> {
20     dest: object.destination,
21     price: object.price,
22     totalSeats: getNumSeats(object.planeType as String),
23     plane: object.planeType
24 }
25 )
26
27 flights
```

29. Look at the preview.

The screenshot shows a software interface with a top navigation bar containing 'Output' and 'Payload'. Below this is a code editor window with the following Java code:

```
10      300
11  */
12 fun getNumSeats (planeType: String) =
13     if (planeType contains('737'))
14         150
15     else
16         300
17 ---
18 using (flights =
19 payload..*return map (object,index) -> {
20     dest: object.destination,
21     price: object.price,
22     totalSeats: getNumSeats(object.planeType as String),
23     plane: object.planeType
24 }
25 )
26
27 flights
```

To the right of the code editor is a preview pane titled 'Preview' which displays the resulting JSON data:

```
[{"dest": "SFO", "price": "400.0", "totalSeats": 150, "plane": "Boing 737"}, {"dest": "LAX", "price": "199.99", "totalSeats": 150, "plane": "Boing 737"}, {"dest": "PDX", "price": "283.0", "totalSeats": 300, "plane": "Boing 777"}]
```

Walkthrough 11-6: Coerce and format strings, numbers, and dates

In this walkthrough, you continue to work with the XML flights data posted to the postMultipleFlights flow. You will:

- Coerce data types.
- Format strings, numbers, and dates.



The screenshot shows the DataWeave editor in Anypoint Studio. The code is as follows:

```
18@using (flights =
19@  payload.*return map (object,index) -> {
20@    dest: object.destination,
21@    price: object.price as Number as String {format: "##,.00"}, // Coerces price to a string
22@    totalSeats:getNumSeats(object.planeType as String),
23@    plane:upper(object.planeType as String), // Converts plane type to uppercase
24@    date: object.departureDate as Date {format: "yyyy/MM/dd"} // Coerces date to a string
25@      as String {format: "MMM dd, yyyy"} // Coerces date to a string
26@  }
27 )
28
29 flights
```

The resulting output is:

```
dest: "SFU",
price: "400.00" as String {format: "##,.00"}, // Coerces price to a string
totalSeats: 150,
plane: "BOING 737",
date: "Oct 20, 2015" as String {format: "MMM dd, yyyy"} // Coerces date to a string
},
{
dest: "LAX",
price: "199.99" as String {format: "##,.00"}, // Coerces price to a string
totalSeats: 150,
plane: "BOING 737",
date: "Oct 21, 2015" as String {format: "MMM dd, yyyy"} // Coerces date to a string
},
```

Format a string

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Use the upper function to return the plane value in uppercase.



The screenshot shows the DataWeave editor with several syntax errors underlined in red:

```
18@using (flights =
19@  payload.*return map (object.index) -> {
20@    dest: object.destination,
21@    price: object.price,
22@    totalSeats:getNumSeats(object.planeType as String),
23@    plane: upper(object.planeType)
24@  }
25 )
```

Note: While using Anypoint Studio 7 in Windows you might not see the red lines under the DataWeave code. You will only be able to see the red cross next to the line number – if that is true, then proceed to the next step.

3. Coerce the argument to a String.

```
plane: upper(object.planeType as String)
```

4. Look at the preview.

The screenshot shows the DataWeave editor interface. On the left, the code is written:

```
15    else
16      300
17  ---
18  using (flights =
19    payload..*return map (object,index) -> {
20      dest: object.destination,
21      price: object.price,
22      totalSeats: getNumSeats(object.planeType as String),
23      plane: upper(object.planeType as String)
24    }
25  )
26
27 flights
```

On the right, the preview pane shows the resulting JSON array:

```
[{"dest": "SFO", "price": "400.0", "totalSeats": 150, "plane": "BOING 737"}, {"dest": "LAX", "price": "199.99", "totalSeats": 150, "plane": "BOING 737"}],
```

Coerce a string to a number

5. In the preview, look at the data type of the prices; you should see that they are strings.
6. Change the DataWeave expression to use the as keyword to return the prices as numbers.

price: object.price as Number,

7. Look at the preview.

The screenshot shows the DataWeave editor interface. On the left, the code is identical to the previous one, except for the addition of the as Number keyword:

```
15    else
16      300
17  ---
18  using (flights =
19    payload..*return map (object,index) -> {
20      dest: object.destination,
21      price: object.price as Number,
22      totalSeats: getNumSeats(object.planeType as String),
23      plane: upper(object.planeType as String)
24    }
25  )
26
27 flights
```

On the right, the preview pane shows the resulting JSON array with numeric prices:

```
[{"dest": "SFO", "price": 400.0, "totalSeats": 150, "plane": "BOING 737"}, {"dest": "LAX", "price": 199.99, "totalSeats": 150, "plane": "BOING 737"}],
```

8. Change the output type from application/dw to application/java.

9. Look at the preview; you should see the prices are now either Integer or Double objects.

```

12 fun getNumSeats (planeType: String) =
13     if (planeType contains('737'))
14         150
15     else
16         300
17 ---
18 using (flights =
19     payload..*return map (object,index) -> {
20         dest: object.destination,
21         price: object.price as Number,
22         totalSeats: getNumSeats(object.planeType as String),
23         plane: upper(object.planeType as String)
24     }
25 )
26
27 flights

```

```

[{"dest": "SFO" as String {class: "java.lang.String"}, "price": 400 as Number {class: "java.lang.Integer"}, "totalSeats": 150 as Number {class: "java.lang.Integer"}, "plane": "BOING 737" as String {class: "java.lang.String"}} as Object {class: "java.util.LinkedHashMap"}, {"dest": "LAX" as String {class: "java.lang.String"}, "price": 199.99 as Number {class: "java.lang.Double"}, "totalSeats": 150 as Number {class: "java.lang.Integer"}, "plane": "BOING 737" as String {class: "java.lang.String"}} as Object {class: "java.util.LinkedHashMap"}, {"dest": "PDX" as String {class: "java.lang.String"}, "price": 283 as Number {class: "java.lang.Integer"}, "totalSeats": 300 as Number {class: "java.lang.Integer"}, "plane": "Airbus A320" as String {class: "java.lang.String"}} as Object {class: "java.util.LinkedHashMap"}]

```

Coerce a string to a specific type of number object

10. Change the DataWeave expression to use the class metadata key to coerce the prices to java.lang.Double objects.

```
price: object.price as Number {class:"java.lang.Double"},
```

11. Look at the preview; you should see the prices are now all Double objects.

```

12 fun getNumSeats (planeType: String) =
13     if (planeType contains('737'))
14         150
15     else
16         300
17 ---
18 using (flights =
19     payload..*return map (object,index) -> {
20         dest: object.destination,
21         price: object.price as Number {class: "java.lang.Double"}, // Coercion added here
22         totalSeats: getNumSeats(object.planeType as String),
23         plane: upper(object.planeType as String)
24     }
25 )
26
27 flights

```

```

[{"dest": "SFO" as String {class: "java.lang.String"}, "price": 400.0 as Number {class: "java.lang.Double"}, "totalSeats": 150 as Number {class: "java.lang.Integer"}, "plane": "BOING 737" as String {class: "java.lang.String"}} as Object {class: "java.util.LinkedHashMap"}, {"dest": "LAX" as String {class: "java.lang.String"}, "price": 199.99 as Number {class: "java.lang.Double"}, "totalSeats": 150 as Number {class: "java.lang.Integer"}, "plane": "BOING 737" as String {class: "java.lang.String"}} as Object {class: "java.util.LinkedHashMap"}, {"dest": "PDX" as String {class: "java.lang.String"}, "price": 283.0 as Number {class: "java.lang.Double"}, "totalSeats": 300 as Number {class: "java.lang.Integer"}, "plane": "Airbus A320" as String {class: "java.lang.String"}} as Object {class: "java.util.LinkedHashMap"}]

```

12. Change the output type from application/java back to application/dw.

The screenshot shows the DataWeave editor with the following code:

```
12 fun getNumSeats (planeType: String) =  
13     if (planeType contains('737'))  
14         150  
15     else  
16         300  
17 ---  
18 using (flights =  
19     payload..*return map (object,index) -> {  
20         dest: object.destination,  
21         price: object.price as Number {class: "java.lang.Double"},  
22         totalSeats: getNumSeats(object.planeType as String),  
23         plane: upper(object.planeType as String)  
24     }  
25 )  
26  
27 flights
```

The preview pane on the right shows the resulting JSON output:

```
[  
  {  
    dest: "SFO",  
    price: 400.0 as Number {class: "java.lang.Double"},  
    totalSeats: 150,  
    plane: "BOING 737"  
  },  
  {  
    dest: "LAX",  
    price: 199.99 as Number {class: "java.lang.Double"},  
    totalSeats: 150,  
    plane: "BOING 737"  
  },  
  {  
    dest: "PDX",  
    price: 283.0 as Number {class: "java.lang.Double"},  
    totalSeats: 300,  
    plane: "BOING 737"  
  }]
```

Format a number

13. Remove the coercion of the price to a java.lang.Double.

14. Coerce the price to a String and use the format schema property to format the prices to zero decimal places.

```
price: object.price as Number as String {format: "###"},
```

15. Look at the preview; the prices should now be formatted.

The screenshot shows the DataWeave editor with the following code:

```
17 ---  
18 using (flights =  
19     payload..*return map (object,index) -> {  
20         dest: object.destination,  
21         price: object.price as Number as String {format: "###"},  
22         totalSeats: getNumSeats(object.planeType as String),  
23         plane: upper(object.planeType as String)  
24     }  
25 )  
26  
27 flights
```

The preview pane on the right shows the resulting JSON output:

```
[  
  {  
    dest: "SFO",  
    price: "400" as String {format: "###"},  
    totalSeats: 150,  
    plane: "BOING 737"  
  },  
  {  
    dest: "LAX",  
    price: "200" as String {format: "###"},  
    totalSeats: 150,  
    plane: "BOING 737"  
  },  
  {  
    dest: "PDX",  
    price: "283" as String {format: "###"},  
    totalSeats: 300,  
    plane: "BOING 737"  
  }]
```

16. Change the DataWeave expression to format the prices to two decimal places.

```
price: object.price as Number as String {format: "##.##"},
```

17. Look at the preview.

The screenshot shows the DataWeave editor with the payload code:

```
17 ---  
18@using (flights =  
19@    payload.*return map {object,index} -> {  
20@        dest: object.destination,  
21@        price: object.price as Number as String {format: "###.##"},  
22@        totalSeats: getNumSeats(object.planeType as String),  
23@        plane: upper(object.planeType as String)  
24@    }  
25 )  
26  
27 flights
```

The preview pane displays the resulting JSON array:

```
[  
  {  
    dest: "SFO",  
    price: "400" as String {format: "###.##"},  
    totalSeats: 150,  
    plane: "BOING 737"  
  },  
  {  
    dest: "LAX",  
    price: "199.99" as String {format: "###.##"},  
    totalSeats: 150,  
    plane: "Airbus A320"  
  }]
```

18. Change the DataWeave expression to format the prices to two minimal decimal places.

```
price: object.price as Number as String {format: "###.00"},
```

19. Look at the preview; all the prices should now be formatted to two decimal places.

The screenshot shows the DataWeave editor with the payload code:

```
17 ---  
18@using (flights =  
19@    payload.*return map {object,index} -> {  
20@        dest: object.destination,  
21@        price: object.price as Number as String {format: "###.00"},  
22@        totalSeats: getNumSeats(object.planeType as String),  
23@        plane: upper(object.planeType as String)  
24@    }  
25 )  
26  
27 flights
```

The preview pane displays the resulting JSON array:

```
[  
  {  
    dest: "SFO",  
    price: "400.00" as String {format: "###.00"},  
    totalSeats: 150,  
    plane: "BOING 737"  
  },  
  {  
    dest: "LAX",  
    price: "199.99" as String {format: "###.00"},  
    totalSeats: 150,  
    plane: "Airbus A320"  
  }]
```

Note: If you are not a Java programmer, you may want to look at the Java documentation for the DecimalFormat class to review documentation on patterns.

<https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>

Coerce a string to a date

20. Add a date field to the return object.

```
date: object.departureDate
```

21. Look at the preview; you should see the date property is a String.

The screenshot shows the Mule Studio interface with the DataWeave editor and a preview pane. The DataWeave code is as follows:

```
17 ---  
18 @using flights =  
19@ payload.*return map (object,index) -> {  
20@   dest: object.destination,  
21@   price: object.price as Number as String {format: "###.00"},  
22@   totalSeats: getNumSeats(object.planeType as String),  
23@   plane: upper(object.planeType as String),  
24@   date: object.departureDate  
25@ }  
26 )  
27  
28 flights
```

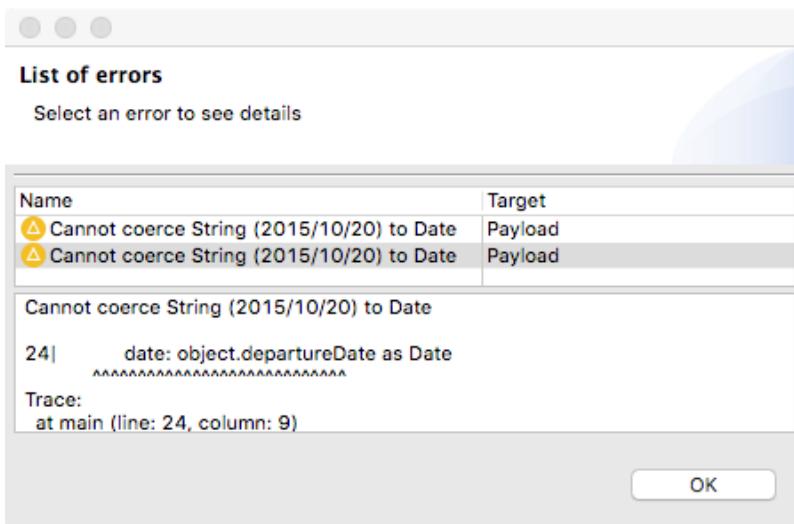
The preview pane shows the resulting JSON output:

```
price: "400.00" as String {format: "###.##"},  
totalSeats: 150,  
plane: "BOING 737",  
date: "2015/10/20"  
},  
{  
dest: "LAX",  
price: "199.99" as String {format: "###.##"},  
totalSeats: 150,  
plane: "BOING 737",  
date: "2015/10/21"  
},
```

22. Change the DataWeave expression to use the as operator to convert departureDate to a date object; you should get an exception.

```
date: object.departureDate as Date
```

23. Look at the exception.



24. Look at the format of the dates in the preview.

25. Change the DataWeave expression to use the format schema property to specify the pattern of the input date strings.

```
date: object.departureDate as Date {format: "yyyy/MM/dd"}
```

Note: If you are not a Java programmer, you may want to look at the Java documentation for the `DateTimeFormatter` class to review documentation on pattern letters.

<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

26. Look at the preview; you should see date is now a Date object.



```
Output Payload ▾ Preview
17 ---
18 @using (flights =
19     payload..*return map (object,index) -> {
20         dest: object.destination,
21         price: object.price as Number as String {format: "###.00"},
22         totalSeats: getNumSeats(object.planeType as String),
23         plane: upper(object.planeType as String),
24         date: object.departureDate as Date {format: "yyyy/MM/dd"}
25     }
26 )
27
28 flights
```

price: "400.00" as String {format: "###.00"},
totalSeats: 150,
plane: "BOING 737",
date: 12015-10-20 as Date {format: "yyyy/MM/dd"}
},
{
dest: "LAX",
price: "199.99" as String {format: "###.00"},
totalSeats: 150,
plane: "BOING 737",
date: 12015-10-21 as Date {format: "yyyy/MM/dd"}
},
]

Format a date

27. Use the as operator to convert the dates to strings, which can then be formatted.

```
date: object.departureDate as Date {format: "yyyy/MM/dd"} as String
```

28. Look at the preview section; the date is again a String – but with a different format.



```
Output Payload ▾ Preview
17 ---
18 @using (flights =
19     payload..*return map (object,index) -> {
20         dest: object.destination,
21         price: object.price as Number as String {format: "###.00"},
22         totalSeats: getNumSeats(object.planeType as String),
23         plane: upper(object.planeType as String),
24         date: object.departureDate as Date {format: "yyyy/MM/dd"} as String
25     }
26 )
27
28 flights
```

price: "400.00" as String {format: "###.00"},
totalSeats: 150,
plane: "BOING 737",
date: "2015-10-20"
},
{
dest: "LAX",
price: "199.99" as String {format: "###.00"},
totalSeats: 150,
plane: "BOING 737",
date: "2015-10-21"
},
]

29. Use the format schema property with any pattern letters and characters to format the date strings.

```
date: object.departureDate as Date {format: "yyyy/MM/dd"} as String  
{format: "MMM dd, yyyy"}
```

30. Look at the preview; the dates should now be formatted according to the pattern.



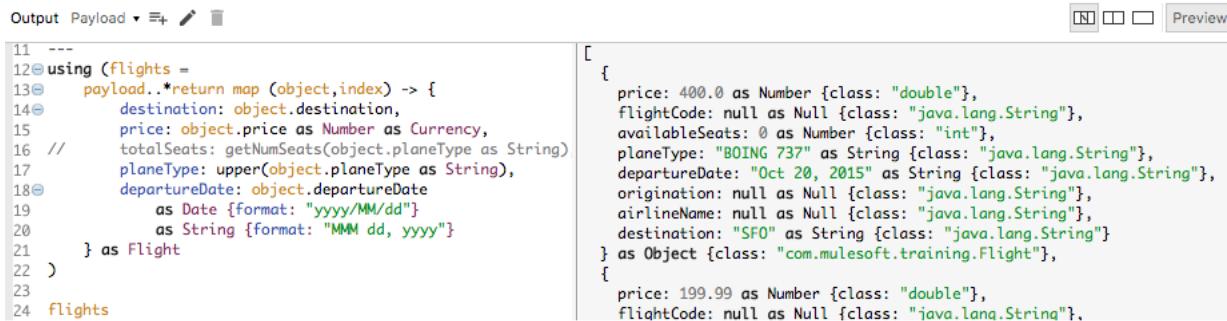
```
Output Payload ▾ Preview
18 @using (flights =
19     payload..*return map (object,index) -> {
20         dest: object.destination,
21         price: object.price as Number as String {format: "###.00"},
22         totalSeats: getNumSeats(object.planeType as String),
23         plane: upper(object.planeType as String),
24         date: object.departureDate as Date {format: "yyyy/MM/dd"}  
            as String {format: "MMM dd, yyyy"}
25     }
26 )
27
28 flights
```

dest: "SFO",
price: "400.00" as String {format: "###.00"},
totalSeats: 150,
plane: "BOING 737",
date: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}
},
{
dest: "LAX",
price: "199.99" as String {format: "###.00"},
totalSeats: 150,
plane: "BOING 737",
date: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}
},
]

Walkthrough 11-7: Define and use custom data types

In this walkthrough, you continue to work with the flight JSON posted to the flow. You will:

- Define and use custom data types.
- Transform objects to POJOs.



```
11 ---
12@using (flights =
13@ payload..*return map (object,index) -> {
14@     destination: object.destination,
15@     price: object.price as Number as Currency,
16@     // totalSeats: getNumSeats(object.planeType as String),
17@     planeType: upper(object.planeType as String),
18@     departureDate: object.departureDate
19@     as Date [format: "yyyy/MM/dd"]
20@     as String [format: "MMM dd, yyyy"]
21@   } as Flight
22@ }
23@ flights
```

```
{ price: 400.0 as Number {class: "double"}, flightCode: null as Null {class: "java.lang.String"}, availableSeats: 0 as Number {class: "int"}, planeType: "BOING 737" as String {class: "java.lang.String"}, departureDate: "Oct 20, 2015" as String {class: "java.lang.String"}, origination: null as Null {class: "java.lang.String"}, airlineName: null as Null {class: "java.lang.String"}, destination: "SFO" as String {class: "java.lang.String"} } as Object {class: "com.mulesoft.training.Flight"}, { price: 199.99 as Number {class: "double"}, flightCode: null as Null {class: "java.lang.String"} }
```

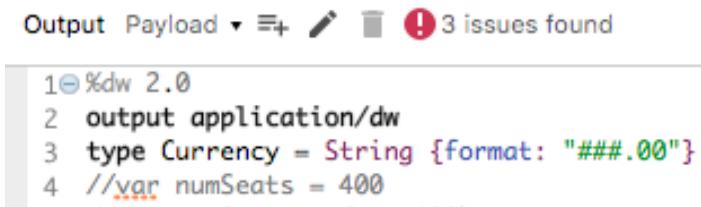
Define a custom data type

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Select and cut the string formatting expression for the prices.

```
18@using (flights =
19@ payload..*return map (object,index) -> {
20@     dest: object.destination,
21@     price: object.price as Number as String {format: "###.00"},  
22@     totalSeats: getNumSeats(object.planeType as String),
23@     plane: upper(object.planeType as String)
```

3. In the header section of the expression, define a custom data type called Currency.
4. Set it equal to the value you copied.

```
type Currency = String {format: "###.00"}
```



```
1@%dw 2.0
2@output application/dw
3@type Currency = String {format: "###.00"}
4@//var numSeats = 400
```

Use a custom data type

5. In the DataWeave expression, set the price to be of type currency.

```
price: object.price as Number as Currency,
```

6. Look at the preview; the prices should still be formatted as strings to two decimal places.

```

18@ using (flights =
19@   payload.*return map (object,index) -> {
20@     dest: object.destination,
21@     price: object.price as Number as Currency,
22@     totalSeats: getNumSeats(object.planeType as String),
23@     plane: object.planeType
24@   }
25@ )
26@ 
```

```

{
  dest: "SFO",
  price: "400.00" as Currency {format: "##.00"},
  totalSeats: 150,
  plane: "BOING 737",
}

```

Transform objects to POJOs

7. Open the Flight.java class in the project's src/main/java folder and look at the names of the properties.

```

1 package com.mulesoft.training;
2
3 import java.util.Comparator;
4
5 public class Flight implements java.io.Serializable,
6
7   String flightCode;
8   String origination;
9   int availableSeats;
10  String departureDate;
11  String airlineName;
12  String destination;
13  double price;
14  String planeType;
15
16@   public Flight() {
17
18  }

```

8. Return to postMultipleFlights in implementation.xml.
9. In the header section of the expression, define a custom data type called flight that is of type com.mulesoft.traning.Flight.

```
type Flight = Object {class: "com.mulesoft.training.Flight"}
```

```

1@ %dw 2.0
2@ output application/dw
3@ type Currency = String {format: "##.00"}
4@ type Flight = Object {class: "com.mulesoft.training.Flight"}
5@ 
```

10. In the DataWeave expression, set the map objects to be of type Flight.

```
20 using (flights =
21   payload.*return map (object,index) -> {
22     dest: object.destination,
23     price: object.price as Number as Currency,
24     totalSeats: getNumSeats(object.planeType as String),
25     plane: upper(object.planeType as String),
26     date: object.departureDate as Date {format: "yyyy/MM/dd"}
27       as String {format: "MMM dd, yyyy"}
28   } as Flight
29 )
```

11. Look at the preview.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave code is displayed:

```
18    300
19  ---
20 using (flights =
21   payload.*return map (object,index) -> {
22     dest: object.destination,
23     price: object.price as Number as Currency,
24     totalSeats: getNumSeats(object.planeType as String),
25     plane: upper(object.planeType as String),
26     date: object.departureDate as Date {format: "yyyy/MM/dd"}
27       as String {format: "MMM dd, yyyy"}
28   } as Flight
29 )
30
31 flights
```

On the right, the preview pane shows the resulting JSON output:

```
[{"dest": "SFO", "price": "400.00" as Currency {format: "###.00"}, "totalSeats": 150, "plane": "BOING 737", "date": "Oct 20, 2015" as String {format: "MMM dd, yyyy"} as Flight [class: "com.mulesoft.training.Flight"], {"dest": "LAX", "price": "199.99" as Currency {format: "###.00"}, "totalSeats": 150, "plane": "BOING 737", "date": "Oct 21, 2015" as String {format: "MMM dd, yyyy"} as Flight [class: "com.mulesoft.training.Flight"]}
```

12. Change the output type from application/dw to application/java.

```
1 %dw 2.0
2 output application/java
3 type Currency = String {format: "###.00"}
4 type Flight = Object {class: "com.mulesoft.training.Flight"}  
-
```

13. Look at the issue you get.

The screenshot shows the 'List of errors' dialog in MuleSoft Anypoint Studio. It displays two validation errors:

Name	Target
△ "Invalid property name: dest on class class..."	Payload
△ "Invalid property name: dest on class class..."	Payload

Below the table, a detailed error message is shown:

"Invalid property name: dest on class class com.mulesoft.training.Flight. Validate that the correct setters is present."

Trace:
at main (line:22, column:3) evaluating expression "%dw 2.0"

At the bottom right is an 'OK' button.

14. Change the name of the dest key to destination.

```
destination: object.destination,
```

15. Change the other keys to match the names of the Flight class properties:

- plane to planeType
- date to departureDate

16. Comment out the totalSeats property.

17. Look at the preview.

Output Payload ▾

```
18      300
19  ---
20@using (flights =
21@    payload..*return map (object,index) -> {
22@      destination: object.destination,
23@      price: object.price as Number as Currency,
24@      //      totalSeats: getNumSeats(object.planeType as String,
25@      planeType: upper(object.planeType as String),
26@      departureDate: object.departureDate as Date [form
27@          as String {format: "MM dd, yyyy"}]
28@      } as Flight
29@    )
30
31  flights
```

```
{
  price: 400.0 as Number {class: "double"},
  flightCode: null as Null {class: "java.lang.String"},
  availableSeats: 0 as Number {class: "int"},
  planeType: "BOING 737" as String {class: "java.lang.String"},
  departureDate: "Oct 20, 2015" as String {class: "java.lang.String"},
  origination: null as Null {class: "java.lang.String"},
  airlineName: null as Null {class: "java.lang.String"},
  destination: "SFO" as String {class: "java.lang.String"}
} as Object {class: "com.mulesoft.training.Flight"},

{
  price: 199.99 as Number {class: "double"},
  flightCode: null as Null {class: "java.lang.String"},
```

Note: If you want to test the application with Advanced REST Client, be sure to change the content-type header to application/XML and replace the request body with XML from the flights-example.xml file.

Mule Debugger X

Name	Value
⑧ Encoding	UTF-8
⑧ Message	
▼ ⑥ Payload (mimeType="application/xml") size = 5	
► ⑥ 0	com.mulesoft.training.Flight@29201745
► ⑥ 1	com.mulesoft.training.Flight@53ef4201
► ⑥ 2	com.mulesoft.training.Flight@19a23ebc
► ⑥ 3	com.mulesoft.training.Flight@3439bb4
▼ ⑥ 4	com.mulesoft.training.Flight@2a702ea6
⑧ airlineName	null
⑧ availableSeats	0
⑧ departureDate	Oct 20, 2015
⑧ destination	PDX
⑧ flightCode	null
⑧ origination	null
⑧ planeType	BOING 777
⑧ price	283.0

implementation X flights-example.xml

▼ postMultipleFlights

```
graph LR
    Listener((Listener)) --> Transform[Transform Message]
    Transform --> Logger[Logger]
```

Walkthrough 11-8: Use DataWeave functions

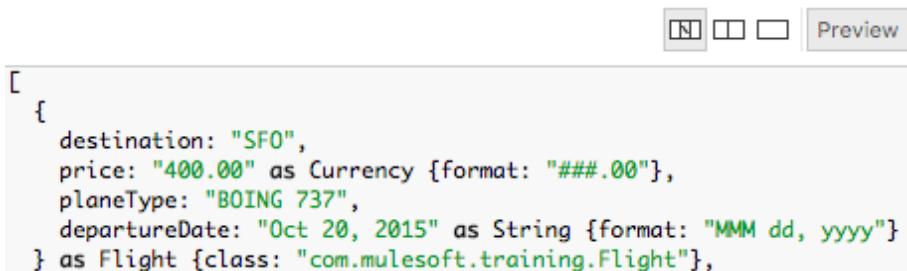
In this walkthrough, you continue to work with the flights JSON posted to the flow. You will:

- Use functions in the Core module that are imported automatically.
- Replace data values using pattern matching.
- Order data, remove duplicate data, and filter data.
- Use a function in another module that you must explicitly import into a script to use.
- Dasherize data.

```
34 @flights orderBy $.departureDate  
35     orderBy $.price  
36     distinctBy $  
37     filter ($.availableSeats != 0)
```

Use the replace function

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Change the output type from application/java to application/dw.
3. Look at the preview and see that Boeing is misspelled.

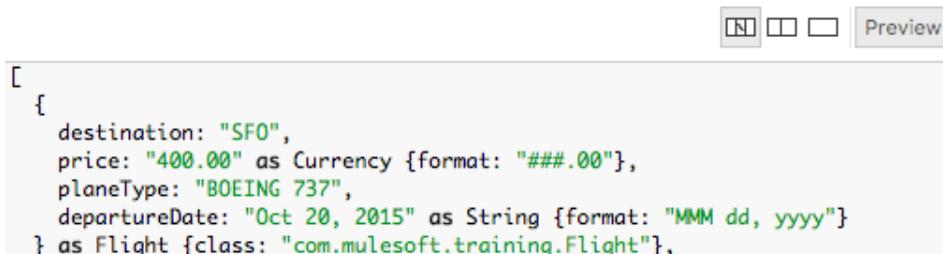


The screenshot shows the DataWeave preview pane with a list of flight objects. Each object is represented by a square icon and contains the following fields:
destination: "SFO",
price: "400.00" as Currency {format: "###.00"},
planeType: "BOING 737",
departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}
} as Flight {class: "com.mulesoft.training.Flight"},

4. For planeType, use the replace function to replace the string Boing with Boeing.

```
planeType: upper(replace(object.planeType,/(Boing)/) with "Boeing"),
```

5. Look at the preview; Boeing should now be spelled correctly.



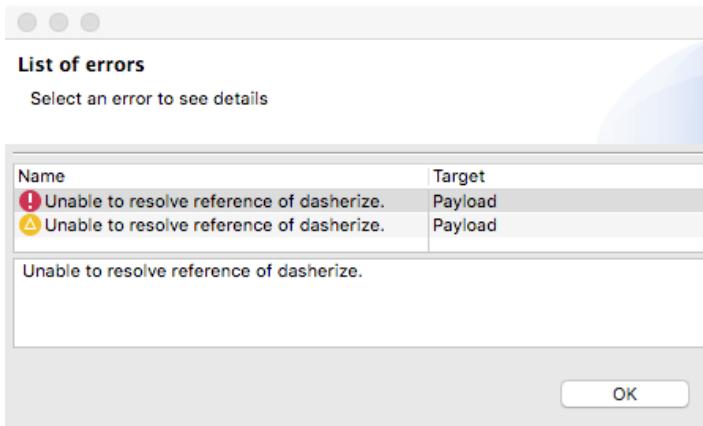
The screenshot shows the DataWeave preview pane with the same list of flight objects as before, but the 'planeType' field has been updated to "Boeing". The other fields remain the same: destination: "SFO", price: "400.00" as Currency {format: "###.00"}, departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}.

Use the dasherize function in the Strings module

- For planeType, replace the upper function with the dasherize function.

```
planeType: dasherize(replace(object.planeType,/(Boing)/) with  
"Boeing"),
```

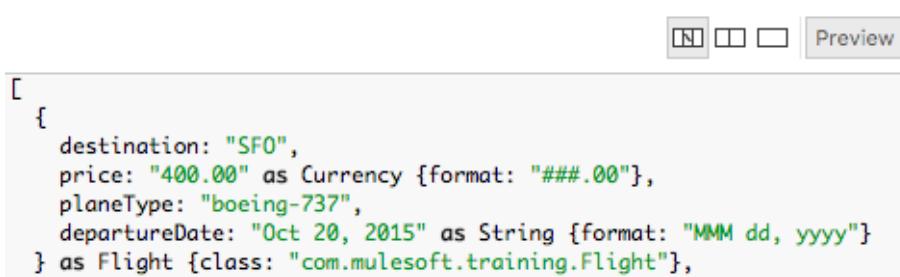
- Look at the error you get.



- In the script header, add an import statement to import dasherize from the Strings module.

```
import dasherize from dw::core::Strings  
  
1@%dw 2.0  
2 output application/dw  
3 import dasherize from dw::core::Strings  
4 type Currency = String {format: "###.00"}  
5 type Flight = Object {class: "com.mulesoft.training.Flight"}
```

- Look at the preview.



Use the orderBy function

- In the preview section, look at the flight prices; the flights should not be ordered by price.
- Use the orderBy function to order the flights object by price.

```
flights orderBy $.price
```

12. Look at the preview; the flights should now be ordered by price.

The screenshot shows the DataWeave editor with the following code:

```
13    300
14  */
15 fun getNumSeats (planeType: String) =
16   if (planeType contains('737')) 150
17   else 300
18 ---
19 using (flights =
20   payload.*return map (object,index) -> {
21     destination: object.destination,
22     price: object.price as Number as Currency,
23     // totalSeats: getNumSeats(object.planeType as String),
24     planeType: dasherize(replace(object.planeType,/(Boin,
25     departureDate: object.departureDate
26       as Date {format: "yyyy/MM/dd"}
27       as String {format: "MMM dd, yyyy"})
28   } as Flight
29 )
30 )
31 )
32
33 flights orderBy $.price
```

The preview pane shows the resulting JSON output:

```
[{"destination": "LAX", "price": "199.99" as Currency {format: "###.00"}, "planeType": "boeing-737", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "SFO", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}]
```

13. Look at the three \$283 flights; there is a duplicate and they are not ordered by date.

14. Change the DataWeave expression to sort flights by price and then date.

```
flights      orderBy $.departureDate
              orderBy $.price
```

15. Look at the preview; the flights should now be ordered by price and flights of the same price should be sorted by date.

The screenshot shows the DataWeave editor with the following code:

```
13 fun getNumSeats (planeType: String) =
14   if (planeType contains('737')) 150
15   else 300
16 ---
17 using (flights =
18   payload.*return map (object,index) -> {
19     destination: object.destination,
20     price: object.price as Number as Currency,
21     // totalSeats: getNumSeats(object.planeType as String),
22     planeType: dasherize(replace(object.planeType,/(Boin,
23     departureDate: object.departureDate
24       as Date {format: "yyyy/MM/dd"}
25       as String {format: "MMM dd, yyyy"})
26   } as Flight
27 )
28 )
29
30 flights orderBy $.departureDate
31
32
33 orderBy $.price
```

The preview pane shows the resulting JSON output:

```
[{"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}, {"destination": "SFO", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}} as Flight {class: "com.mulesoft.training.Flight"}]
```

Remove duplicate data

16. Use the distinctBy function to first remove any duplicate objects.

```
flights      orderBy $.departureDate
              orderBy $.price
              distinctBy $
```

17. Look at the preview; you should now see only two \$283 flights to PDX instead of three.

```
Output Payload ▾ Preview
15 fun getNumSeats (planeType: String) =  
16     if (planeType contains('737'))  
17         150  
18     else  
19         300  
20 ---  
21 using (flights =  
22 payload..*return map (object,index) -> {  
23     destination: object.destination,  
24     price: object.price as Number as Currency,  
25     // totalSeats: getNumSeats(object.planeType as String),  
26     planeType: dasherize(replace(object.planeType,/(Boin:  
27     departureDate: object.departureDate  
28         as Date [format: "yyyy/MM/dd"]  
29         as String [format: "MMM dd, yyyy"]  
30     } as Flight  
31 })  
32 flights orderBy $.departureDate  
33     orderBy $.price  
34     distinctBy $  
35  
36 price: "199.99" as Currency {format: "###.##"},  
37 planeType: "boeing-737",  
38 departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}  
39 } as Flight [class: "com.mulesoft.training.Flight"],  
40 {  
41     destination: "PDX",  
42     price: "283.00" as Currency {format: "###.##"},  
43     planeType: "boeing-777",  
44     departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}  
45 } as Flight [class: "com.mulesoft.training.Flight"],  
46 {  
47     destination: "PDX",  
48     price: "283.00" as Currency {format: "###.##"},  
49     planeType: "boeing-777",  
50     departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}  
51 } as Flight [class: "com.mulesoft.training.Flight"],  
52 {  
53     destination: "SFO",  
54     price: "400.00" as Currency {format: "###.##"},  
55     planeType: "boeing-737",  
56     departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}  
57 }
```

18. Add an `availableSeats` field that is equal to the `emptySeats` field and coerce it to a number.

availableSeats: object.emptySeats as Number

19. Look at the preview; you should get three \$283 flights to PDX again.

```
Output Payload ▾ + └ Preview
16 if (planeType contains('737'))
17   150
18 else
19   300
20 ---
21 using (flights =
22 payload.*return map (object,index) -> {
23   destination: object.destination,
24   price: object.price as Number as Currency,
25   // totalSeats: getNumSeats(object.planeType as String,
26   planeType: dasherize(replace(object.planeType,/(^|_|$)/g,
27   departureDate: object.departureDate
28     as Date {format: "yyyy/MM/dd"}
29     as String {format: "MMM dd, yyyy"},
30   availableSeats: object.emptySeats as Number
31 } as Flight
32 )
33
34 flights orderBy $.departureDate
35 orderBy $.price
36 distinctBy $
```

```
{
  destination: "PDX",
  price: "283.00" as Currency {format: "###.##"},  

  planeType: "boeing-777",
  departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},  

  availableSeats: 0
} as Flight {class: "com.mulesoft.training.Flight"},  

{
  destination: "PDX",
  price: "283.00" as Currency {format: "###.##"},  

  planeType: "boeing-777",
  departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},  

  availableSeats: 23
} as Flight {class: "com.mulesoft.training.Flight"},  

{
  destination: "PDX",
  price: "283.00" as Currency {format: "###.##"},  

  planeType: "boeing-777",
  departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},  

  availableSeats: 30
} as Flight {class: "com.mulesoft.training.Flight"}  


```

Use the filter function

20. In the preview, look at the values of the availableSeats properties; you should see some are equal to zero.

21. Use the filter function to remove any objects that have availableSeats equal to 0.

```
flights    orderBy $.departureDate  
           orderBy $.price  
           distinctBy $  
           filter ($.availableSeats !=0)
```



22. Look at the preview; you should no longer get the flight that had no available seats.

Output Payload ▾  

```
18    else
19      300
20  ---
21@using flights =
22@  payload.*return map (object,index) -> {
23@    destination: object.destination,
24    price: object.price as Number as Currency,
25 //  totalSeats: getNumSeats(object.planeType as String)
26    planeType: dasherize(replace(object.planeType,/(?!
27@      departureDate: object.departureDate
28        as Date {format: "yyyy/MM/dd"})
29        as String {format: "MMM dd, yyyy"}),
30      availableSeats: object.emptySeats as Number
31    } as Flight
32  )
33
34@ flights orderBy $.departureDate
35  orderBy $.price
36  distinctBy $
37  filter ($.availableSeats !=0)
38
```

destination: "LAX",
price: "199.99" as Currency {format: "###.00"},
planeType: "boeing-737",
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 10
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "###.00"},
planeType: "boeing-777",
departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 23
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "###.00"},
planeType: "boeing-777",
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 30
} as Flight {class: "com.mulesoft.training.Flight"},
{
...

Walkthrough 11-9: Look up data by calling a flow

In this walkthrough, you continue to work with the transformation in `getMultipleFlights`. You will:

- Call a flow from a DataWeave expression.

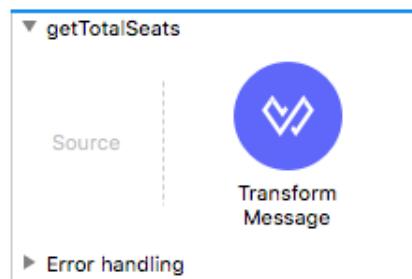


Create a lookup flow

1. Return to the Transform Message properties view for the transformation in `postMultipleFlights`.
2. Copy the `getNumSeats` function.

```
15① fun getNumSeats (planeType: String) =  
16②     if (planeType contains('737'))  
17③         150  
18④     else  
19⑤         300
```

3. Drag a Transform Message component from the Mule Palette and drop it at the bottom of the canvas to create a new flow.
4. Change the name of the flow to `getTotalSeats`.



5. In the Transform Message properties view of the component in `getTotalSeats`, paste the function you copied into the script header.

6. In the script body, call the function, passing to it the payload.

```
Output Payload ▾ ⌂
```

```
1@ %dw 2.0
2  output application/java
3
4@ fun getNumSeats(planeType: String) =
5@   if (planeType contains ("737"))
6     150
7   else
8     300
9
10 ---
11 getNumSeats(payload.planeType)
```

Call a flow from a DataWeave expression

7. Return to the Transform Message properties view of the component in postMultipleFlights.
8. Add a property called totalSeats inside the expression (keep the other one commented out).
9. Change the return type of the map function from Flight to Object.

```
-->
21@ using (flights =
22@   payload..*return map (object,index) -> {
23@     destination: object.destination,
24@     price: object.price as Number as Currency,
25@     // totalSeats: getNumSeats(object.planeType as String),
26@     totalSeats:
27@       planeType: dasherize(replace(object.planeType,/(Boin
28@       departureDate: object.departureDate
29@         as Date {format: "yyyy/MM/dd"})
30@         as String {format: "MMM dd, yyyy"}),
31@       availableSeats: object.emptySeats as Number
32@     } as Object
33@   )
```

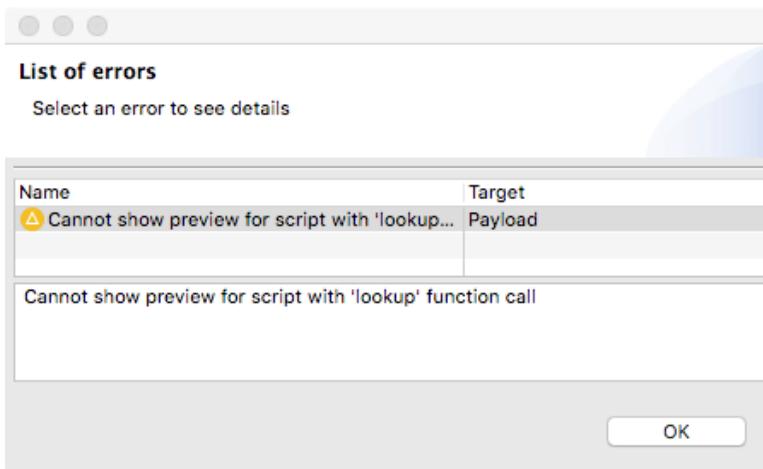
10. Set totalSeats equal to the return value from the DataWeave lookup() function.

```
totalSeats: lookup()
```

11. Pass to lookup() an argument that is equal to the name of the flow to call: "getTotalSeats".
12. Pass an object to lookup() as the second argument.
13. Give the object a field called type (to match what the flow is expecting) and set it equal to the value of the planeType field.

```
totalSeats: lookup("getTotalSeats",{planeType: object.planeType})
```

14. Look at the preview.
15. Look at the warning you get.



Test the application

16. Run or debug the project.
17. In Advanced REST Client, make sure the method is set to POST and the request URL is set to <http://localhost:8081/multipleflights>.
18. Set a Content-Type header to application/xml.
19. Set the request body to the value contained in the flights-example.xml file in the src/test/resources folder.

Method Request URL
POST <http://localhost:8081/multipleflights> **SEND** **⋮**

Parameters **^**

Headers	Authorization	Body	Variables	Actions
Body content type application/xml		<pre><ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"> <return airlineName="United"> <code>A1B2C3</code><departureDate>2015/10/20</departureDate> <destination>SFO</destination><emptySeats>40</emptySeats> <origin>MUA</origin> <planeType>Boing 737</planeType> <price>400.0</price> </return> <return airlineName="Delta"> <code>A1B2C4</code> <departureDate>2015/10/21</departureDate><destination>LAX</destination><emptyS... <origin>MUA</origin> <planeType>Boing 737</planeType> <price>199.99</price> </return> </ns2:listAllFlightsResponse></pre>		

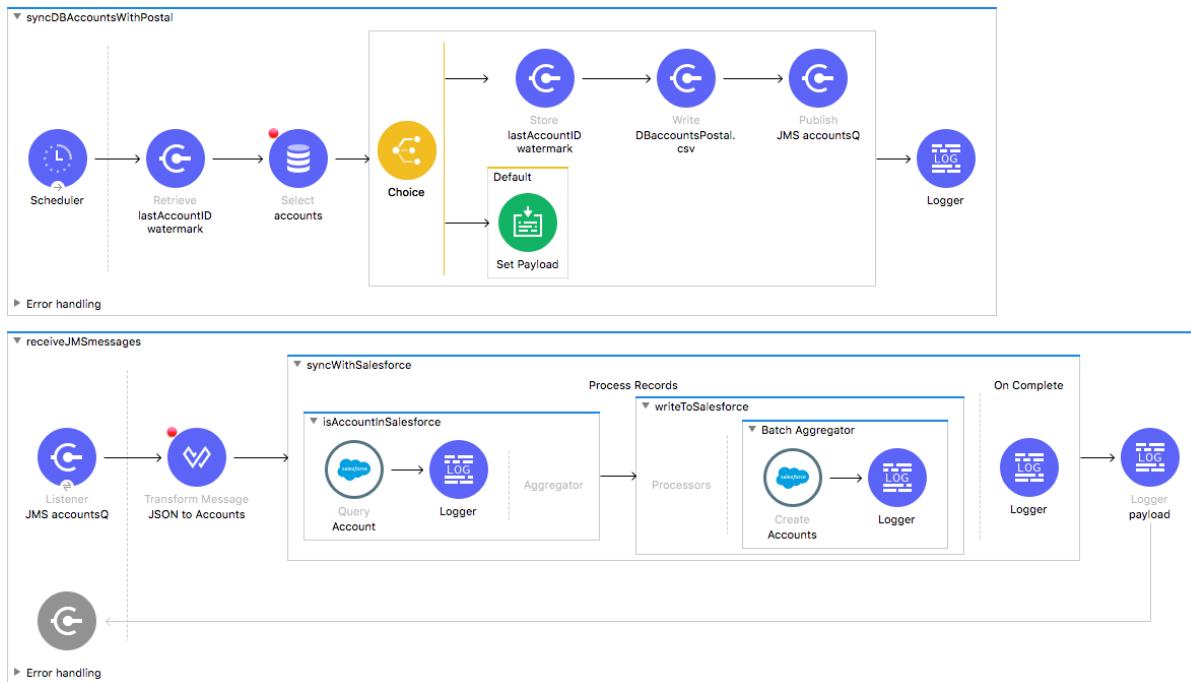
20. Send the request; you should get the DataWeave representation of the return flight data and each flight should have a totalSeats property equal to 150 or 300.

200 OK 14174.81 ms DETAILS ▾

[
 {
 destination: "LAX",
 price: "199.99" as Currency {format: "###.##"},
 totalSeats: 150 as Number {class: "java.lang.Integer", encoding: "UTF-8", mimeType: "application/java", raw: 150 as Number {class: "java.lang.Integer"}},
 planeType: "boeing-737",
 departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},
 availableSeats: 10
,
 {
 destination: "PDX",
 price: "283.00" as Currency {format: "###.##"},
 totalSeats: 300 as Number {class: "java.lang.Integer", encoding: "UTF-8", mimeType: "application/java", raw: 300 as Number {class: "java.lang.Integer"}},
 planeType: "boeing-777",
 departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},
 availableSeats: 23
,
 {

21. Return to Anypoint Studio.
22. Stop the project.
23. Close the project.

PART 3: Building Applications to Synchronize Data

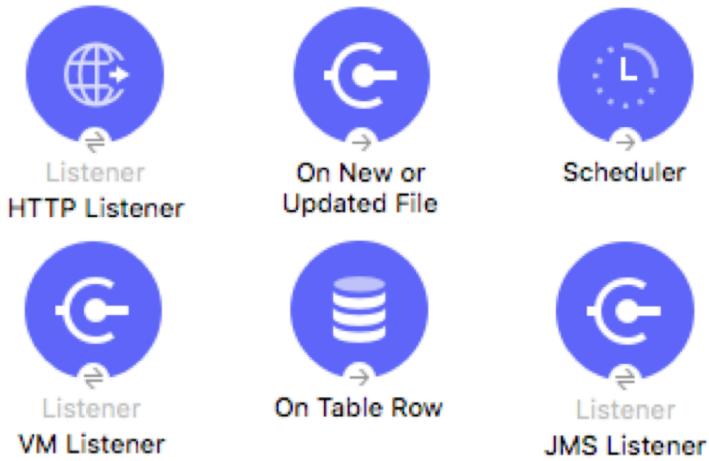


At the end of this part, you should be able to:

- Trigger flows when files or database records are added or updated.
- Schedule flows.
- Persist and share data across flow executions.
- Publish and consume JMS messages.

- Process items in a collection sequentially.
- Process records asynchronously in batches.

Module 12: Triggering Flows



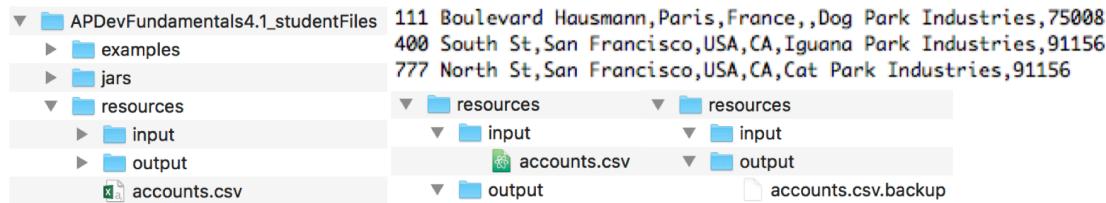
At the end of this module, you should be able to:

- Read and write files.
- Trigger flows when files are added, created, or updated.
- Trigger flows when new records are added to a database table.
- Schedule flows to run at a certain time or frequency.
- Persist and share data in flows using the Object Store.
- Publish and consume JMS messages.

Walkthrough 12-1: Trigger a flow when a new file is added to a directory

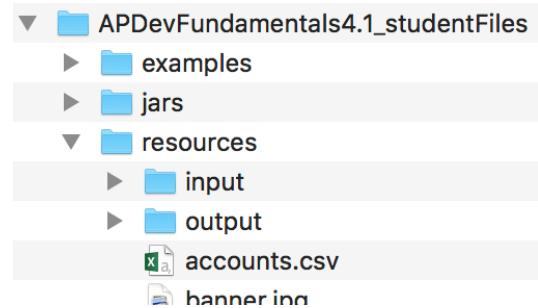
In this walkthrough, you load data from a local CSV file when a new file is added to a directory. You will:

- Add and configure a File listener to watch an input directory.
- Restrict the type of file read.
- Rename and move the processed files.



Locate files and folders

1. In your computer's file browser, return to the student files for the course.
2. Open the resources folder and locate the accounts.csv file and the input and output folders.

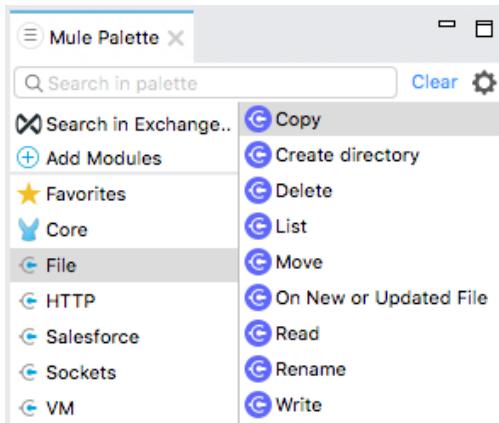


3. Leave this folder open.

Add the File module to the project

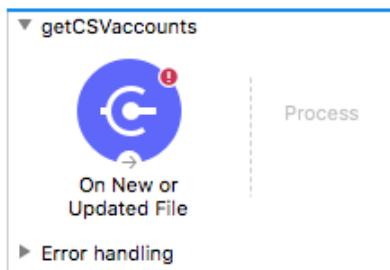
4. Return to Anypoint Studio and open the apdev-examples project.
5. Open accounts.xml.
6. In the Mule Palette, select Add Modules.
7. Select the File connector in the right side of the Mule Palette and drag and drop it into the left side.

8. If you get a Select module version dialog box, select the latest version and click Add.



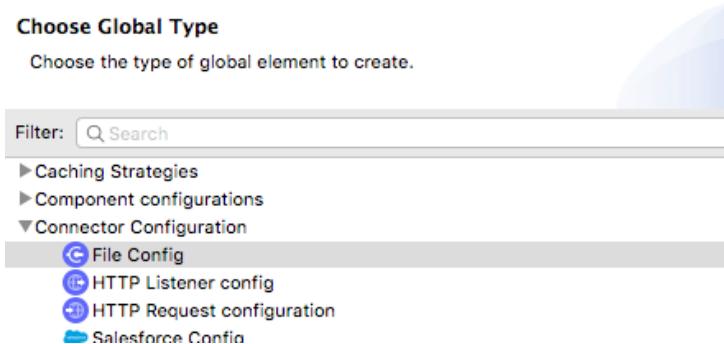
Create a flow that monitors a location for new files

9. Locate the On New or Updated File operation for the File connector in the right side of the Mule Palette and drag and drop it at the top of the canvas to create a new flow.
10. Rename the flow to getCSVaccounts.



Configure the File connector

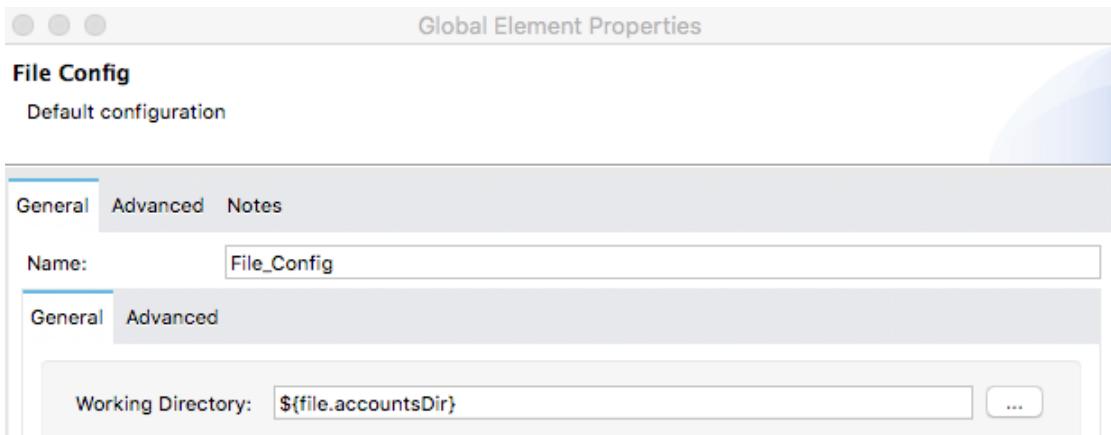
11. Open global.xml and switch to the Global Elements view.
12. Click Create.
13. In the Choose Global Type dialog box, select Connector Configuration > File Config and click OK.



14. In the Global Element Properties dialog box, click the browse button next to working directory.
15. Browse to and select the student files folder for the course.
16. Select the resources folder and click Open.



17. Select and cut the value of the working directory that got populated and replace it with a property \${file.accountsDir}.



18. In the Global Element Properties dialog box, click OK.
19. Open config.yaml in src/main/resources.
20. Create a new property file.accountsDir and set it equal to the value you copied.

```

accounts
global
config.yaml

9 file:
10   accountsDir: "/Users/mule/Desktop/APDevFundamentals4.1_studentFiles/resources"

```

The screenshot shows a code editor with a file named 'config.yaml'. The file contains the following YAML configuration:

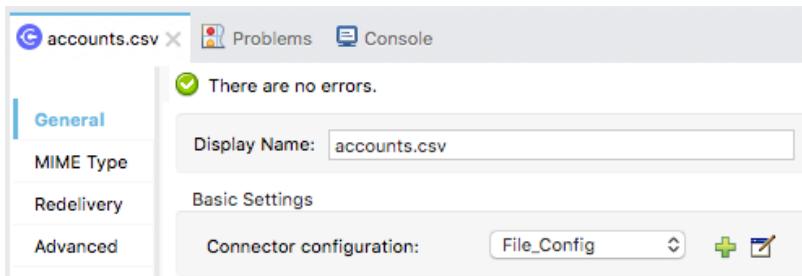
```

file:
  accountsDir: "/Users/mule/Desktop/APDevFundamentals4.1_studentFiles/resources"

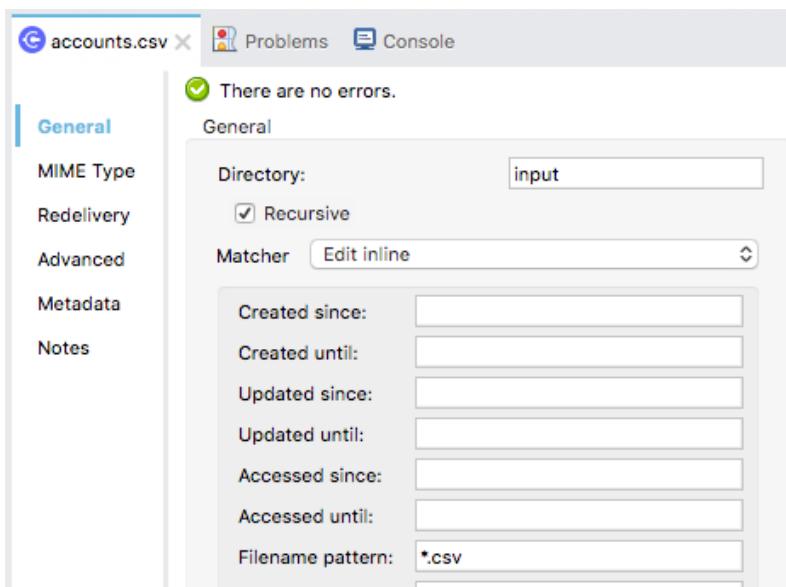
```

Configure the On New or Updated File operation

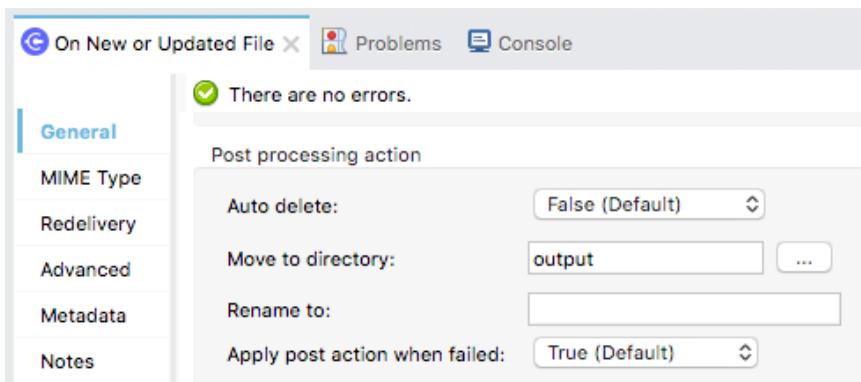
21. Return to accounts.xml.
22. In the On New or Updated File properties view, change the display name to accounts.csv.
23. Set the connector configuration to the existing File_Config.



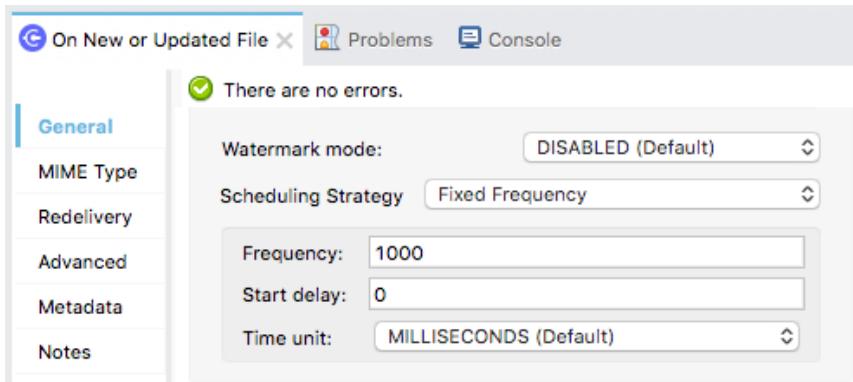
24. In the General section, set the directory to input.
25. Set the matcher to Edit inline.
26. Set filename pattern to *.csv.



27. In the post processing action section, set the move to directory to output.



28. In the General section, review the default scheduling information; the endpoint checks for new files every 1000 milliseconds.



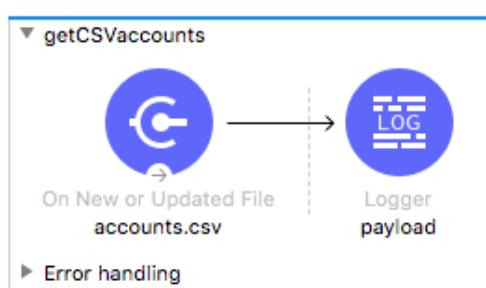
Review event metadata in the DataSense Explorer

29. Select the Output tab in the DataSense Explorer; you should see no metadata about the structure of the CSV file.

Display the file contents

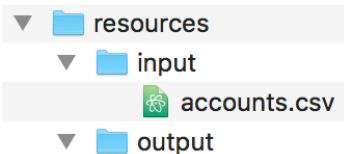
30. Add a Logger to the flow.
31. In the Logger properties view, set the display name to payload and the message to display the payload.

```
##[payload]
```



Debug the application

32. Add a breakpoint to the Logger.
33. Debug the project.
34. In your computer's file browser, return to the student files for the course.
35. Move the accounts.csv file to the input folder.



36. Return to the Mule Debugger and look at the payload.

Mule Debugger

Name	Value
③ Message	
③ Payload (mimeType="text/csv")	Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode 111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008 400 South St,San Francisco,USA,CA,Iguana Park Industries,91156 777 North St,San Francisco,USA,CA,Cat Park Industries,91156

accounts global config.yaml

getCSVaccounts

```
graph LR; File((On New or Updated File  
accounts.csv)) --> Logger[Logger]
```

Error handling

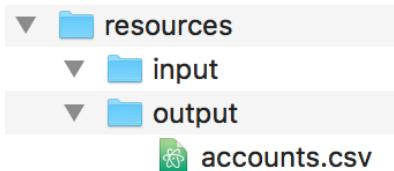
37. Expand Attributes and locate the fileName and path attributes.

Mule Debugger

Name	Value
③ Attributes	LocalFileAttributes[lastModifiedTime=2018-04-03T11:51:48,lastAccessTime=2018-04-2018-01-12T10:49:36]
③ creationTime	2018-01-12T10:49:36
③ directory	false
③ fileName	accounts.csv
③ lastAccessTime	2018-04-22T12:24:08
③ lastModifiedTime	2018-04-03T11:51:48
③ path	/Users/mule/Desktop/APDevFundamentals4.1_studentFiles/resources/input/accounts.csv

38. Step to the end of the application.

39. In your computer's file browser, return to the student files for the course; you should see accounts.csv has been moved to the output folder.



40. Return to Anypoint Studio and look at the console; you should see the file contents displayed.

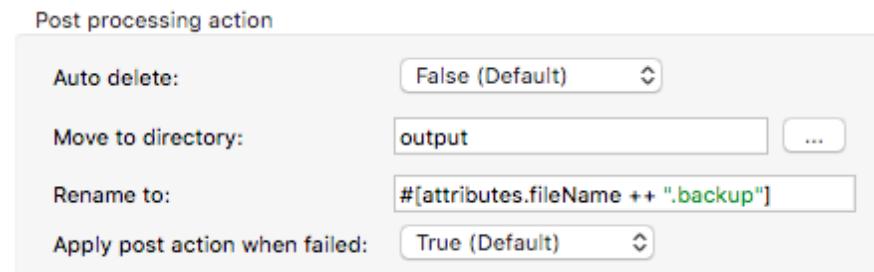
```
Console X Problems Logger apdev-examples [Mule Applications] Mule Server 4.1.1 EE
INFO 2018-04-22 13:04:01,776 [[MuleRuntime].cpuLight.07: [apdev-examples].getCSVaccountsFlow.CPU_LITE @49a07a2e] [event : 0-467f4890-4668-11e8-8ff7-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode
111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008
400 South St, San Francisco, USA, CA, Iguana Park Industries, 91156
777 North St, San Francisco, USA, CA, Cat Park Industries, 91156
```

The screenshot shows the Anypoint Studio interface with the "Console" tab selected. The log window displays several lines of text representing CSV data being processed by a flow named "getCSVaccountsFlow". The data includes columns like Billing Street, Billing City, Billing Country, Billing State, Name, and BillingPostalCode, with specific values for each row.

Rename the file

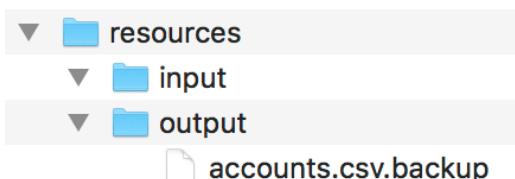
41. Return to the File properties view and in the Post processing action section, set the rename to property to an expression that appends .backup to the original filename.

```
##[attributes.fileName ++ ".backup"]
```

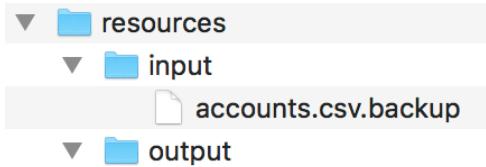


Test the application

42. Save the file to redeploy the project.
43. Remove the breakpoint from the Logger.
44. In your computer's file explorer, move the accounts.csv file from the output folder back to the input folder; you should see it appear in the output folder with its new name.



45. Move the accounts.csv.backup file from the output folder back to the input folder; it should not be processed and should stay in the input folder.



46. Return to Anypoint Studio and switch perspectives.

47. Stop the project.

Walkthrough 12-2: Trigger a flow when a new record is added to a database and use automatic watermarking

In this walkthrough, you work with the accounts table in the training database. You will:

- Add and configure a Database listener to check a table on a set frequency for new records.
- Use the listener's automatic watermarking to track the ID of the latest record retrieved and trigger the flow whenever a new record is added.
- Output new records to a CSV file.
- Use a form to add a new account to the table and see the CSV file updated.

MUA Accounts

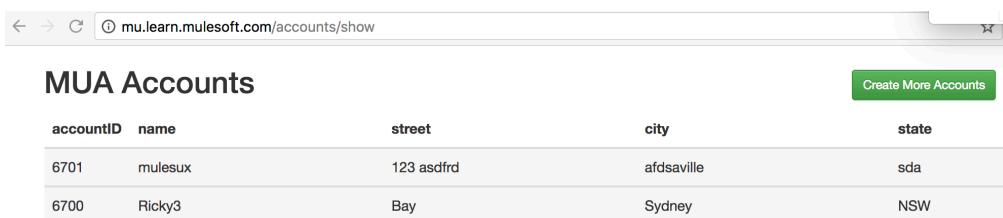
accountID	name	street	city	state	
6706	Max Mule	77 Geary Street	San Francisco	California	Create More Accounts
6705	Minnie Mule	77 Geary Street	San Francisco	California	

DBaccounts.csv ▾

```
usa,6684,a\,snksaa,emc_kcj,sumanth,sncksid,95113
United States,6704,bana,texas,AO_Smith,triple,94108
United States,6702,77 Geary Street,California,Max Mule ,San Francisco,94111
United States,6692,Dakila,Bulacan,Winlyn,Malolos,3000
Australia,6700,Bay,NSW,Ricky3,Sydney,2216
United States,6705,77 Geary Street,California,Minnie Mule,San Francisco,94108
United States,6706,77 Geary Street,California,Max Mule,San Francisco,94111
```

View accounts data

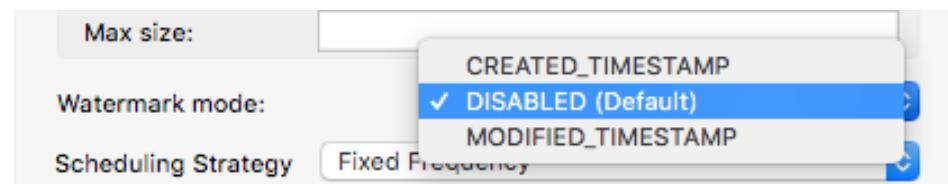
1. In a web browser, navigate to <http://mu.mulesoft-training.com/accounts/show>.
2. Review the data and the names of the columns—which match those of database table.



accountID	name	street	city	state	
6701	mulesux	123 asdfrd	afdsaville	sda	Create More Accounts
6700	Ricky3	Bay	Sydney	NSW	

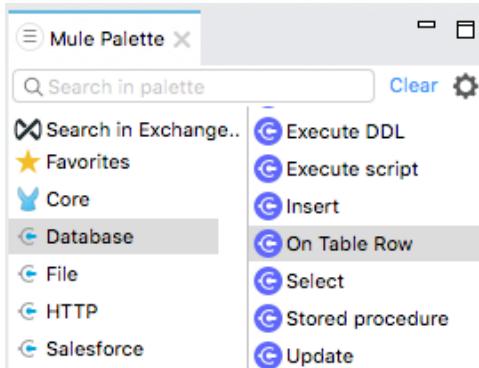
Look at the File listener settings for watermarking

3. Return to accounts.xml in Anypoint Studio.
4. Return to the On New or Updated File properties view for the listener in getCSVaccounts.
5. Locate the watermark mode setting in the General section and look at its possible values.



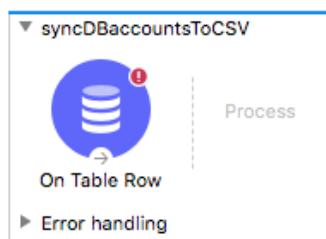
Add the Database module to the project

6. In the Mule Palette, select Add Modules.
7. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.
8. If you get a Select module version dialog box, select the latest version and click Add.



Create a flow to monitor a database

9. Locate the On Table Row operation for the Database connector in the right side of the Mule Palette and drag and drop it at the top of the canvas to create a new flow.
10. Rename the flow to syncDBaccountsToCSV.



Configure a Database connector

11. Return to the course.snippets.txt file.
12. Locate and copy the MySQL (or Derby) database properties in the Module 4 section.
13. Return to config.yaml and paste the properties.

```
*accounts *global *config.yaml
db:
  host: "mudb.learn.mulesoft.com"
  port: "3306"
  user: "mule"
  password: "mule"
  database: "training"
```

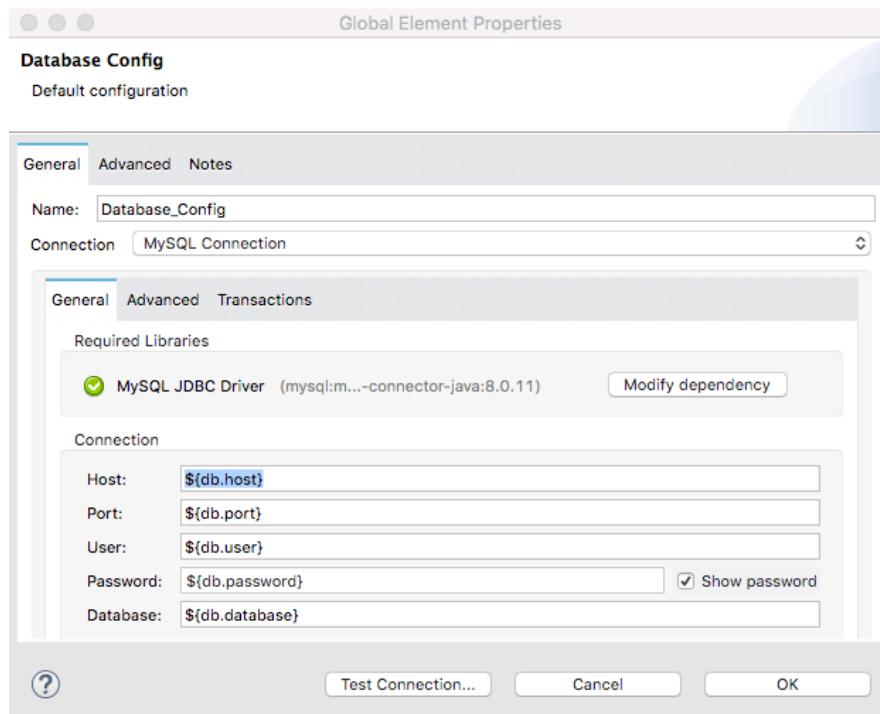
The image shows a code editor window with tabs for "accounts", "global", and "config.yaml". The "config.yaml" tab is active. The code contains a "db:" key with several properties: host, port, user, password, and database, all set to specific values.

14. Save the file.

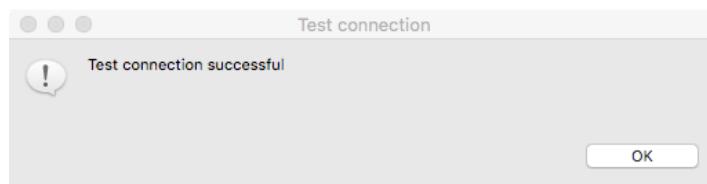
15. Return to global.xml and create a new Database Config that uses these properties.

Note: If necessary, refer to the detailed steps in Walkthrough 4-2.

16. Add the MySQL (or Derby) JDBC driver.



17. Test the connection and make sure it is successful.

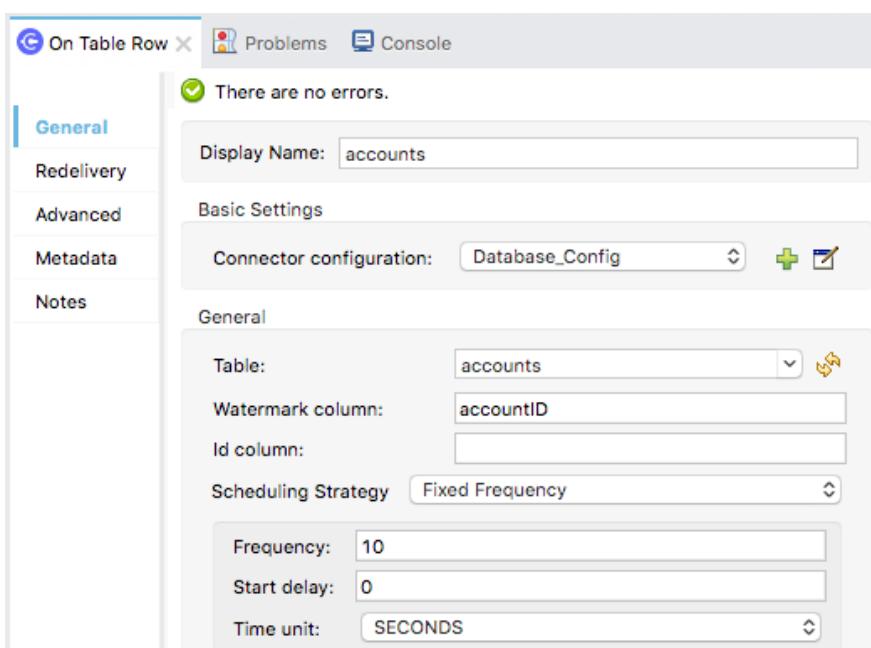


Configure the Database listener

18. Return to accounts.csv.

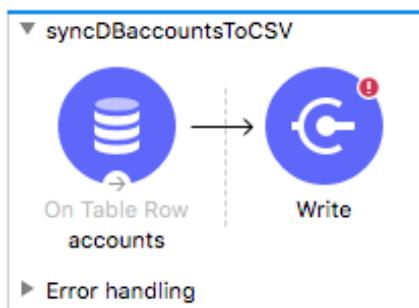
19. In the On Table Row properties view, set the following values:

- Display Name: accounts
- Connector configuration: Database_Config
- Table: accounts
- Watermark column: accountID
- Id column: accountID
- Frequency: 10
- Time unit: SECONDS



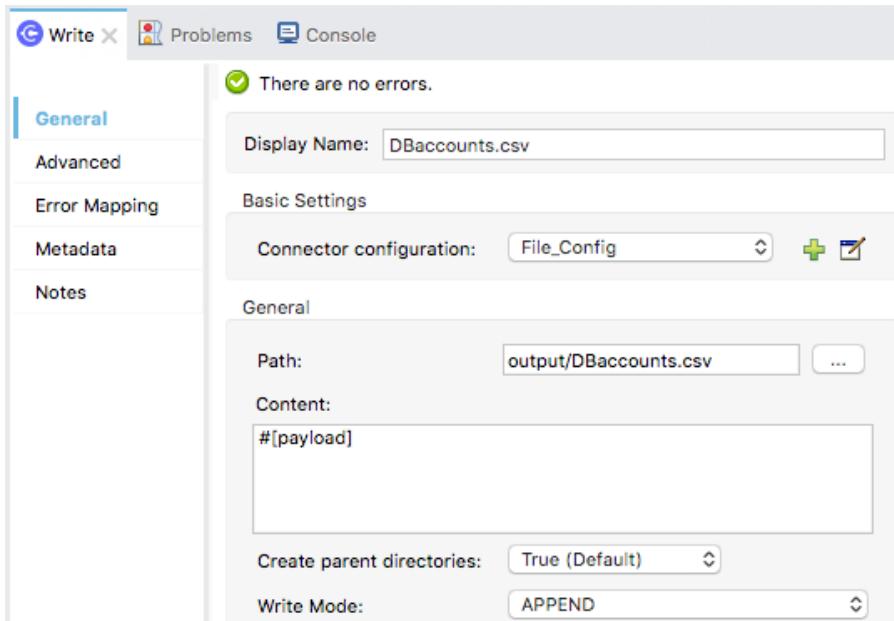
Write new records to a CSV file

20. Add a File Write operation to the flow.



21. In the Write properties view, set the following values:

- Display Name: DBaccounts.csv
- Connector configuration: File_Config
- Path: output/DBaccounts.csv
- Write mode: APPEND



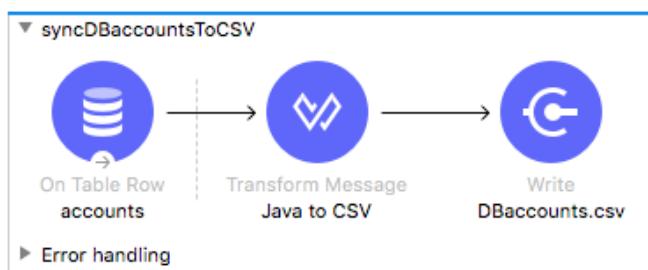
22. Select the input tab in the DataSense Explorer.

23. Expand Payload; you should see the operation will get an Array of Objects from the Database operation.

Transform the records to CSV

24. Add a Transform Message component before the Write operation.

25. Change the display name to Java to CSV.



26. In the Transform Message properties view, change the output type to application/csv and add a header property equal to false.

27. Set the body expression to [payload].



```
Output Payload ▾ ⌂ ⌂ ⌂ Preview  
1 %dw 2.0  
2 output application/csv header=false  
3 ---  
4 [payload]
```

Add a Logger to display the records

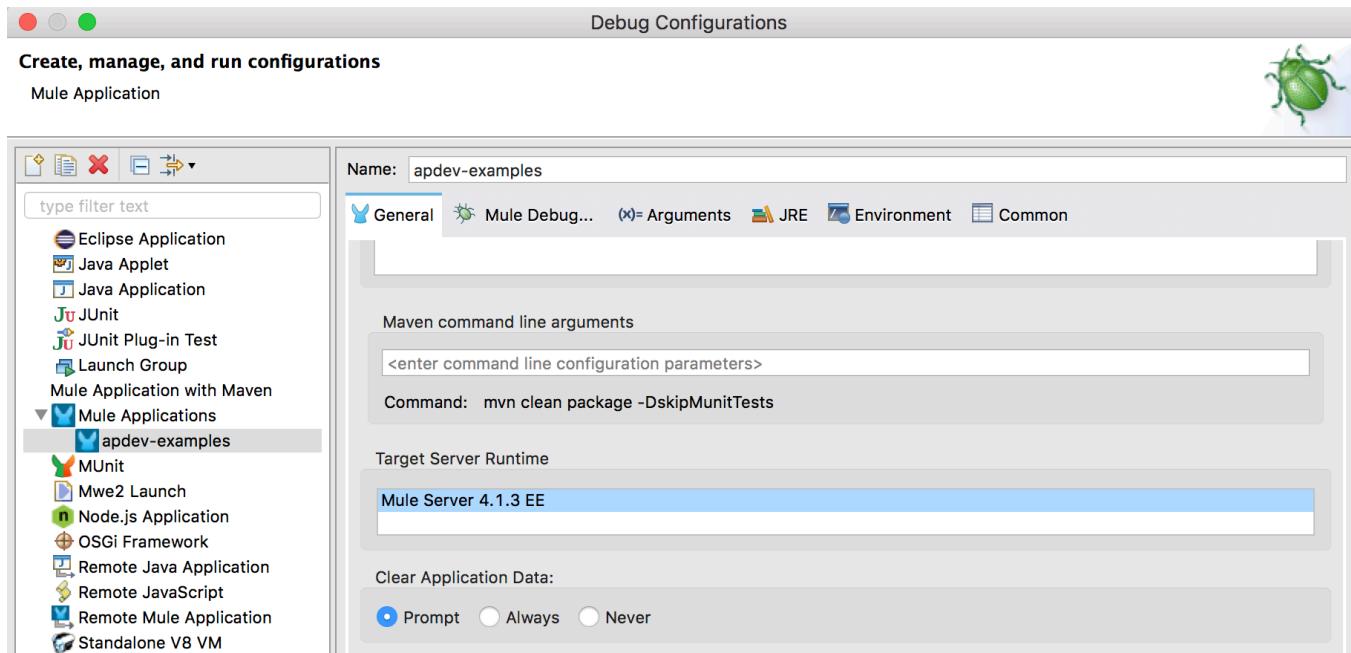
28. Add a Logger to the flow.
29. Change the display name to payload.
30. Set the message to display the payload.



Set the application to prompt to clear application data when it starts

31. Add a breakpoint to the Transform Message component.
32. In the main menu, select Run > Debug Configurations.

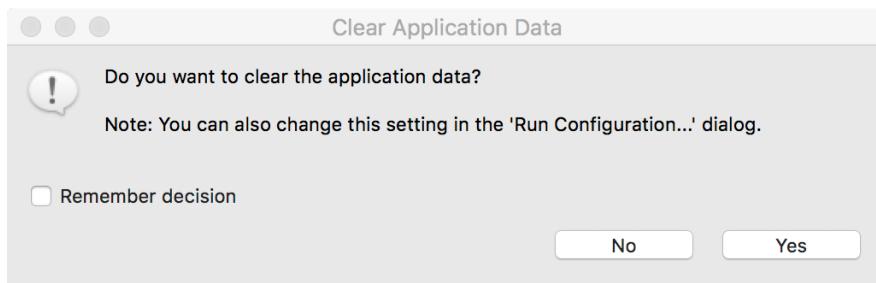
33. In the Debug Configurations dialog box, locate the Clear Application Data section in the General tab and change it to Prompt.



Note: You need to rerun or debug an application to get the prompt; you cannot just save to redeploy.

Debug the project

34. Click Debug.
35. In the Clear Application dialog box, select Yes.



36. In the Mule Debugger, expand Payload.

The screenshot shows the Mule Debugger interface. At the top, there's a table titled "Payload (mimeType='*/*')". The table has two columns: "Name" and "Value". Under "Name", there are seven entries labeled 0 through 6. Under "Value", each entry corresponds to a field: size=7, country=United States of America, accountID=6468, street=11111 Test Street, state=CA, name=TJC, city=San Diego, and postal=92116. Below the table, there's a large empty text area. At the bottom of the window, there are tabs for "accounts", "config.yaml", and "global".

accounts X config.yaml global

Name Value

Payload (mimeType="*/*")

0 size = 7
country=United States of America
accountID=6468
street=11111 Test Street
state=CA
name=TJC
city=San Diego
postal=92116

1

2

3

4

5

6

37. Click Resume and step through several of the records.

38. Look at the console, you should see records displayed.

The screenshot shows the Mule Studio Console. The title bar says "Console X Problems Mule Properties" and "apdev-examples [Mule Applications] Mule Server 4.1.1 EE". The main area displays log entries:

```
919}
INFO 2018-04-24 10:08:57,619 [[MuleRuntime].cpuLight.13: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @76a240d] [event : 0-2c6556e2-47e2-11e8-9be7-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=Unit ed States, accountID=6688, street=77 Geary Street, state=California, name=Maxwell Mule, city=San Francisco, postal=94108 }
INFO 2018-04-24 10:08:57,621 [[MuleRuntime].cpuLight.13: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @76a240d] [event : 0-2c670490-47e2-11e8-9be7-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=Aust ralia, accountID=6697, street=Strathfield, state=NSW, name=Lindsay, city=Sydney, postal=94108}
```

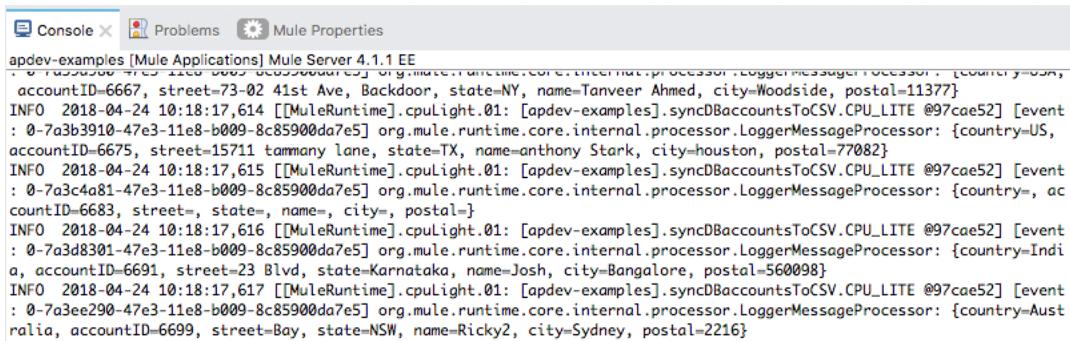
39. Stop the project and switch perspectives.

Clear application data and test the application

40. Run the project.

41. In the Clear Application dialog box, select Yes.

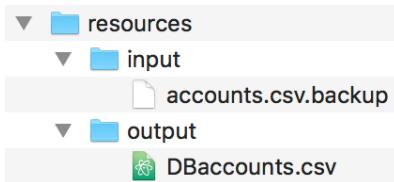
42. Look at the console; you should see many records.



```
apdev-examples [Mule Applications] Mule Server 4.1.1 EE
INFO 2018-04-24 10:18:17,610 [[MuleRuntime].cpulight.01: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @97cae52] [event : 0-7a3d8301-47e3-11e8-b009-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=US, accountID=6667, street=73-02 41st Ave, Backdoor, state=NY, name=Tanveer Ahmed, city=Woodside, postal=11377}
INFO 2018-04-24 10:18:17,614 [[MuleRuntime].cpulight.01: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @97cae52] [event : 0-7a3b3910-47e3-11e8-b009-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=US, accountID=6675, street=15711 tammany lane, state=TX, name=anthony Stark, city=houston, postal=77082}
INFO 2018-04-24 10:18:17,615 [[MuleRuntime].cpulight.01: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @97cae52] [event : 0-7a3c4a81-47e3-11e8-b009-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=, accountID=6683, street=, state=, name=, city=, postal=}
INFO 2018-04-24 10:18:17,616 [[MuleRuntime].cpulight.01: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @97cae52] [event : 0-7a3d8301-47e3-11e8-b009-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=India, accountID=6691, street=23 Blvd, state=Karnataka, name=Josh, city=Bangalore, postal=560098}
INFO 2018-04-24 10:18:17,617 [[MuleRuntime].cpulight.01: [apdev-examples].syncDBaccountsToCSV.CPU_LITE @97cae52] [event : 0-7a3ee290-47e3-11e8-b009-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {country=Australia, accountID=6699, street=Bay, state=NSW, name=Ricky2, city=Sydney, postal=2216}
```

43. Stop the project.

44. Return to the resource folder in your computer's file explorer; you should see a new DBaccounts.csv file appear in the output folder.



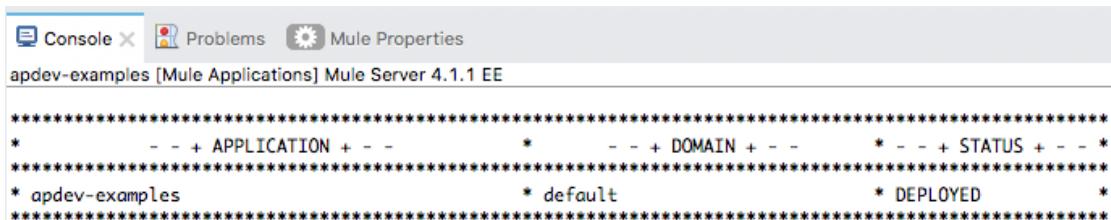
45. Open the DBaccounts.csv file with a text editor; you should see the records.

Test the application again without clearing application data

46. Return to Anypoint Studio and run the project.

47. In the Clear Application dialog box, select No.

48. Look at the console; you should see no new records output.



```
*****
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*****  
* apdev-examples * default * DEPLOYED *
```

Add a new account to the database

49. Return to the account data in the web browser at <http://mu.mulesoft-training.com/accounts/show>.

50. Click Create More Accounts.

51. Fill out the form with data and click Create Record.

mu.learn.mulesoft.com/accounts

Create New Account Records

Name: Max Mule

Street: 77 Geary Street

City: San Francisco

State: California

Postal: 94111

Country: United States

Note: Set the postal code to a specific value. In the next walkthrough, you will retrieve only new records with this specific postal code, so you do not get all of the records.

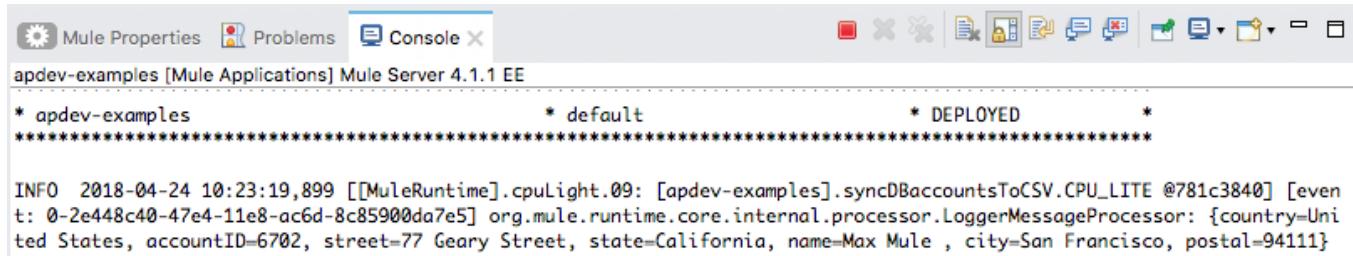
52. Click View Existing Accounts; you should see your new account.

← → ⌂ ⓘ mu.learn.mulesoft.com/accounts/show

MUA Accounts

accountID	name	street	city	state
6702	Max Mule	77 Geary Street	San Francisco	California

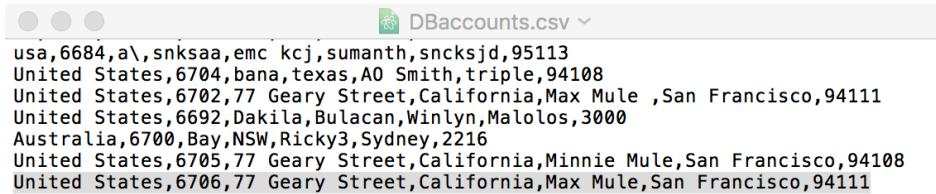
53. Return to the console in Anypoint Studio; you should see your new record.



54. Stop the project.

55. Return to your computer's file explorer; the modified date on the DBAccounts.csv file should have been updated.

56. Open DBaccounts.csv and locate your new record at the end of the file.



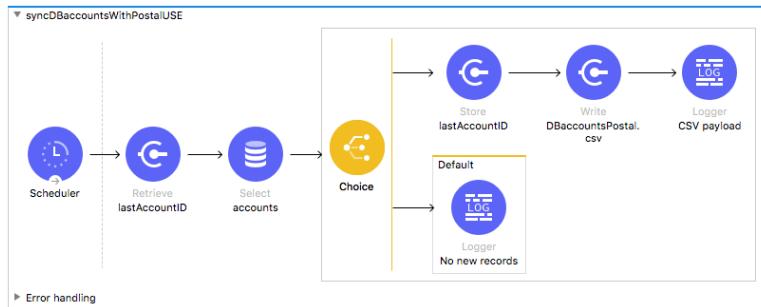
The screenshot shows a window titled "DBaccounts.csv" containing a list of account records. The data is as follows:

usa,6684,a\,snksaa,emc kcj,sumanth,sncksjd,95113	United States,6704,bana,texas,A0 Smith,triple,94108	United States,6702,77 Geary Street,California,Max Mule ,San Francisco,94111	United States,6692,Dakila,Bulacan,Winlyn,Malolos,3000	Australia,6700,Bay,NSW,Ricky3,Sydney,2216	United States,6705,77 Geary Street,California,Minnie Mule,San Francisco,94108	United States,6706,77 Geary Street,California,Max Mule,San Francisco,94111
usa,6684,a\,snksaa,emc kcj,sumanth,sncksjd,95113	United States,6704,bana,texas,A0 Smith,triple,94108	United States,6702,77 Geary Street,California,Max Mule ,San Francisco,94111	United States,6692,Dakila,Bulacan,Winlyn,Malolos,3000	Australia,6700,Bay,NSW,Ricky3,Sydney,2216	United States,6705,77 Geary Street,California,Minnie Mule,San Francisco,94108	United States,6706,77 Geary Street,California,Max Mule,San Francisco,94111

Walkthrough 12-3: Schedule a flow and use manual watermarking

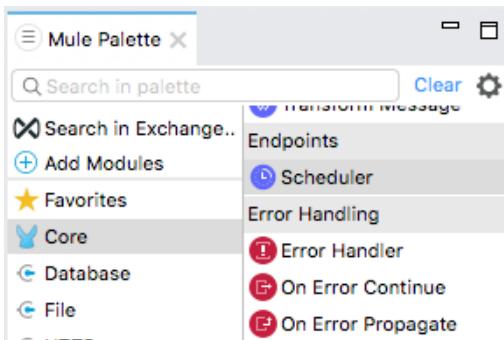
In this walkthrough, you continue to work with the accounts table in the training database. You will:

- Use the Scheduler component to create a new flow that executes at a specific frequency.
 - Retrieve accounts with a specific postal code from the accounts table.
 - Use the Object Store component to store the ID of the latest record and then use it to only retrieve new records.

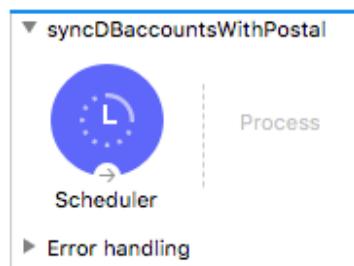


Create a flow that executes at a specific frequency

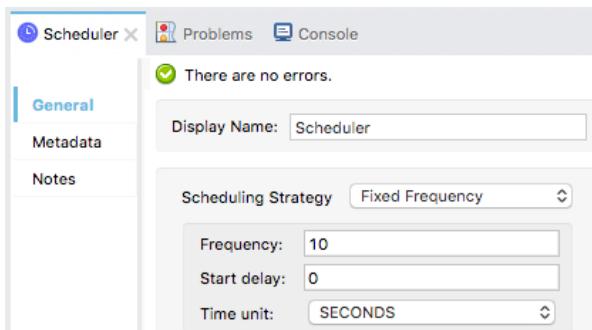
1. Return to accounts.xml in Anypoint Studio.
 2. Locate the Scheduler component in the Core section of the Mule Palette.



3. Drag a Scheduler component from the Mule Palette and drop it at the top of the canvas to create a new flow.
 4. Change the name of the flow to syncDBaccountsWithPostal.

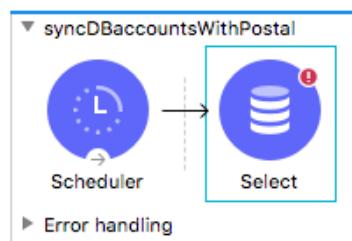


5. In the Scheduler properties view, set the frequency to 10 seconds.



Retrieve records with a specific postal code from the database

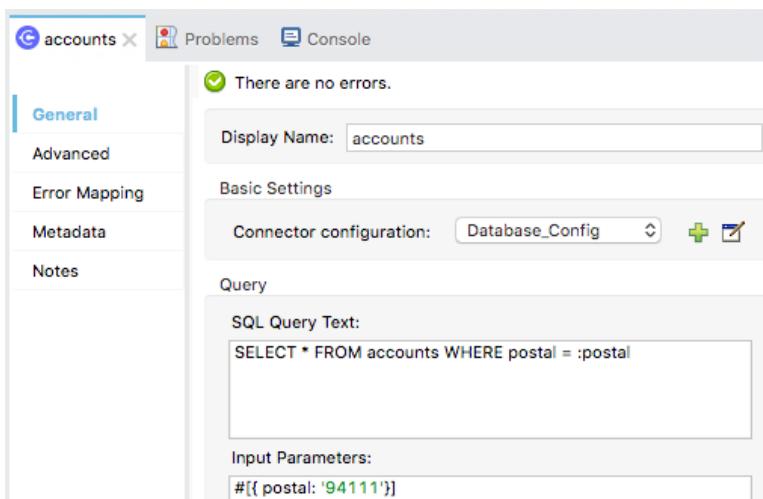
6. From the Mule Palette, drag a Database Select operation and drop it in the flow.



7. In the Select properties view, set the following:

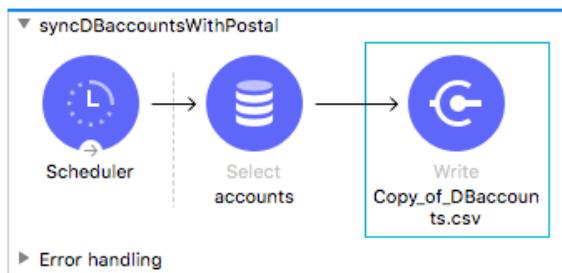
- Display Name: accounts
- Connector configuration: Database _Config
- SQL query text: SELECT * FROM accounts WHERE postal = :postal
- Input parameters: #[{ postal: 'yourPostalValue'}]

Note: If you want, you can store the postal code as a property in config.yaml and then reference it here in the DataWeave expression as #[{ postal: p('propertyName') }].



Output the records to a CSV file

8. Copy the Write operation in syncDBaccountsToCSV and paste it at the end of syncDBaccountsWithPostal.

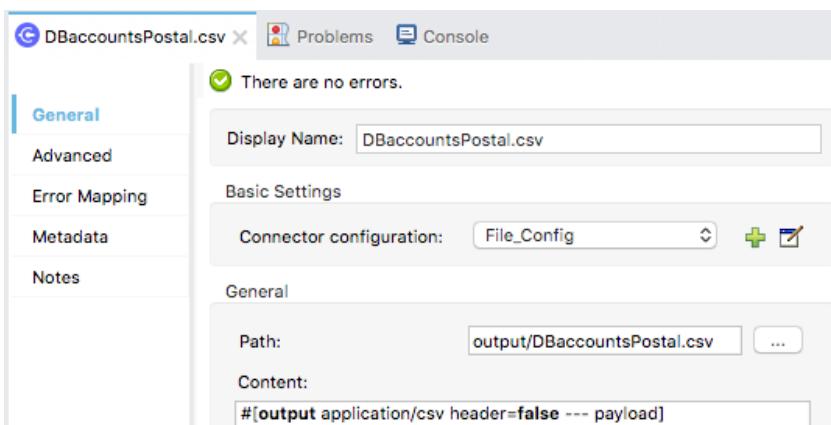


9. In the properties view for the Write operation, set the following values:

- Display Name: DBaccountsPostal.csv
- Path: output/DBaccountsPostal.csv

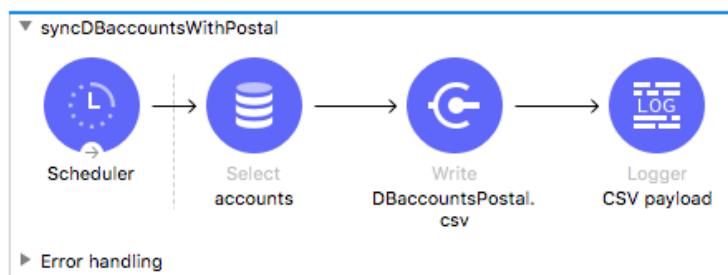
10. Change the content to output the payload as application/csv with a header property equal to false.

```
##[output application/csv header=false --- payload]
```

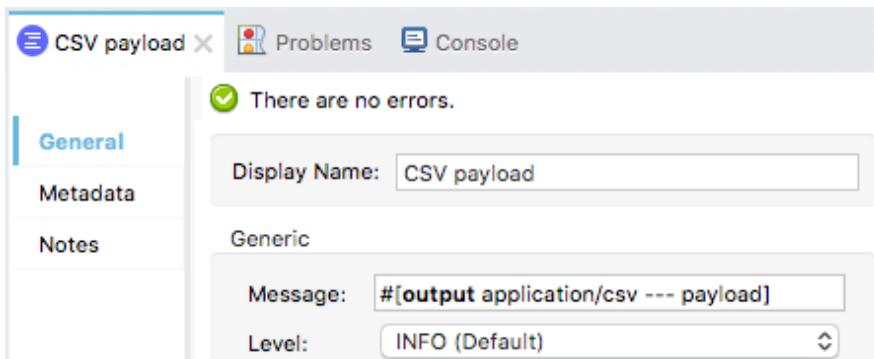


Log the payload

11. Add a Logger at the end of the flow and change the display name to CSV payload.

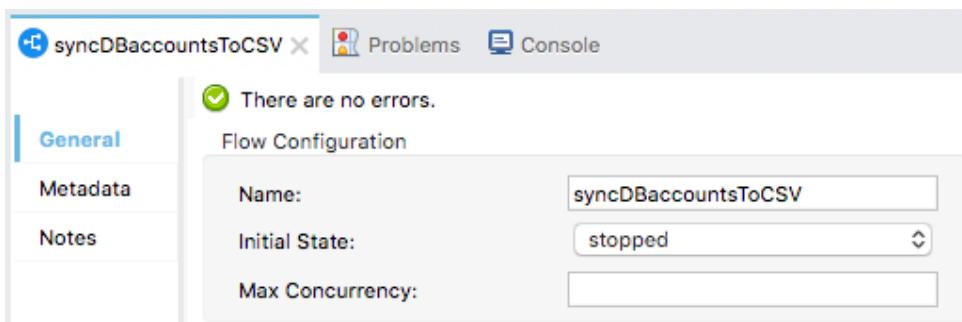


12. Set the message to the payload as type application/csv.



Stop the other syncDBaccountsToCSV flow so it does not run

13. In the properties view for the syncDBaccountsToCSV flow, set the initial state to stopped.



Test the application

14. Run the project.
15. In the Clear Application dialog box, select Yes.
16. Watch the console, you should see the same records displayed every 10 seconds.

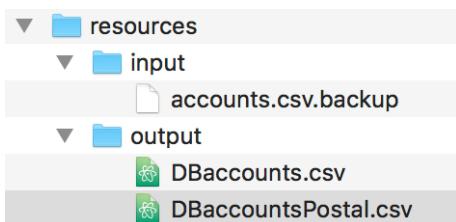
```
apdev-examples [Mule Applications] Mule Server 4.1.1 EE
ID,country,street,state,name,city,postal
6702,United States,77 Geary Street,California,Max Mule ,San Francisco,94111
6706,United States,77 Geary Street,California,Max Mule, San Francisco,94111
6707,United States,77 Geary Street,California,Molly Mule, San Francisco,94111

INFO 2018-04-24 12:50:37,076 [[MuleRuntime].cpulight.14: [apdev-examples].syncDBaccountsWithPostal.CPU_LITE @3ffc
[Event: 0-c207e940-47f8-11e8-b11e-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: a
ID,country,street,state,name,city,postal
6702,United States,77 Geary Street,California,Max Mule ,San Francisco,94111
6706,United States,77 Geary Street,California,Max Mule, San Francisco,94111
6707,United States,77 Geary Street,California,Molly Mule, San Francisco,94111
```

*Note: Right now, all records with matching postal code are retrieved – over and over again.
Next, you will modify this so only new records with the matching postal code are retrieved.*

17. Stop the project.

18. Return to the resources folder in your computer's file browser; you should see the new DBaccountsPostal.csv file.

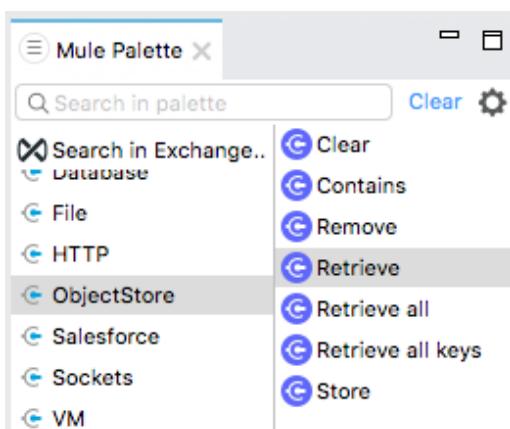


19. Open the file and review the contents.

20. Delete the file.

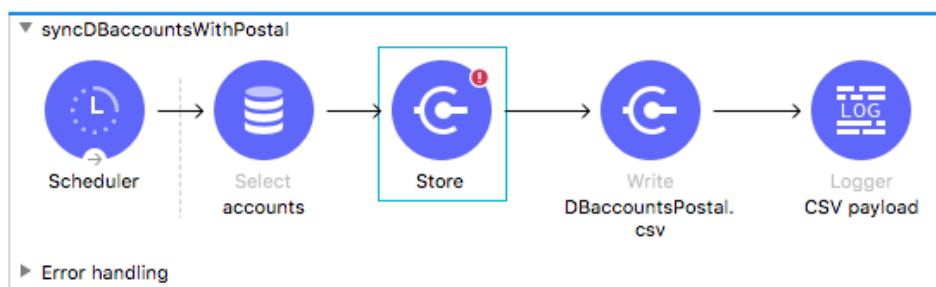
Add the ObjectStore module to the project

21. In the Mule Palette, select Add Modules.
22. Select the ObjectStore connector in the right side of the Mule Palette and drag and drop it into the left side.
23. If you get a Select module version dialog box, select the latest version and click Add.



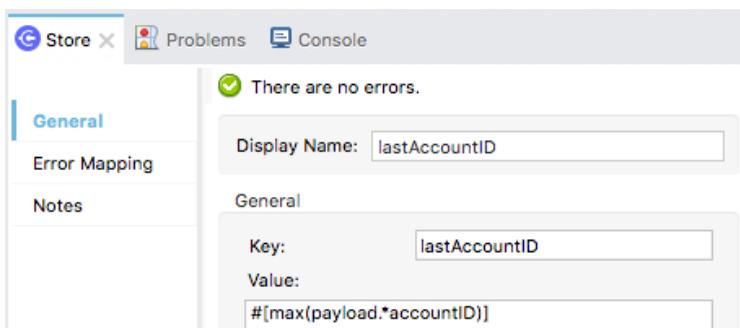
Store the ID of the last record retrieved

24. Drag the Store operation for the ObjectStore connector from the Mule Palette and drop it after the Database Select operation.



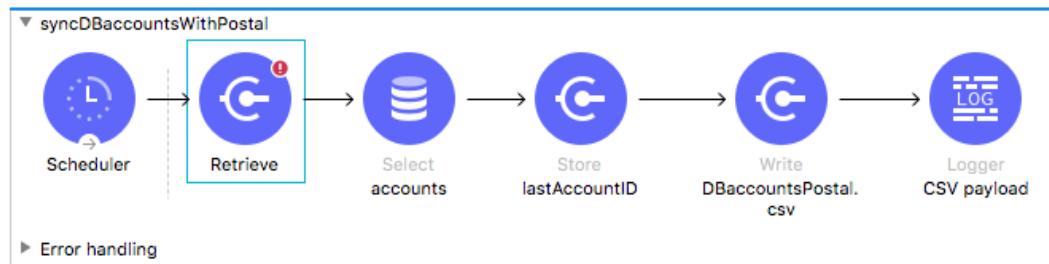
25. In the Store properties view, set the display name and key to lastAccountID.
26. Set the value to an expression for the maximum value of lastAccountID in the payload, the retrieved records.

```
##[max(payload.*accountID)]
```



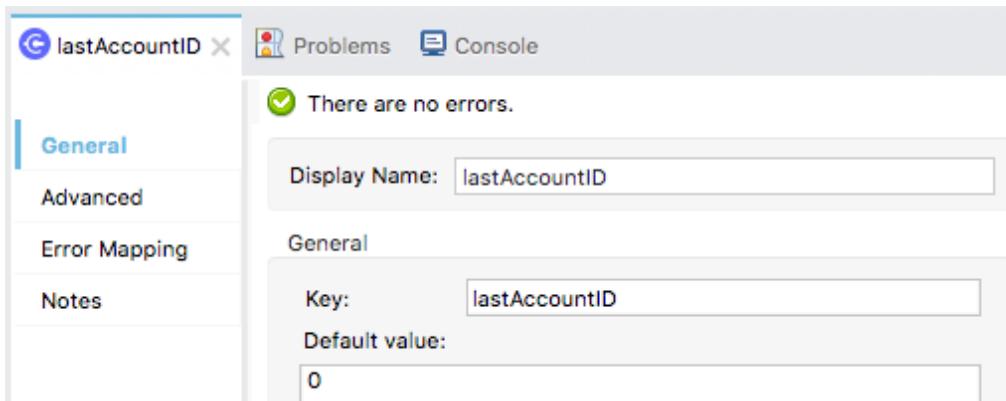
Before the query, retrieve the ID of the last record retrieved

27. Drag the Retrieve operation for the ObjectStore connector from the Mule Palette and drop it before the Database Select operation.

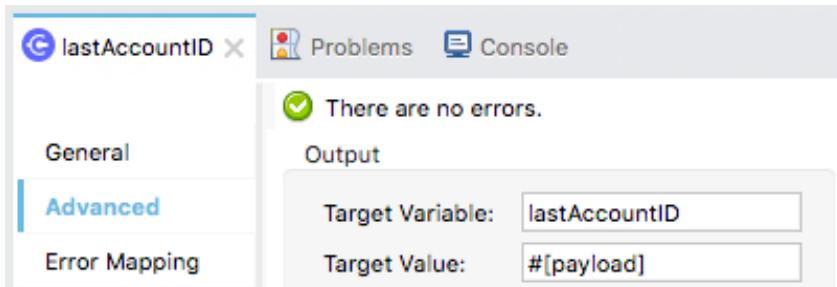


28. In the Retrieve properties view, set the following values:

- Display Name: lastAccountID
- Key: lastAccountID
- Default value: 0



29. Select the Advanced tab and set the target variable to lastAccountID so the value is stored in a variable called lastAccountID.



Modify the query to only retrieve new records with a specific postal code

30. In the accounts Select properties view, add a second query input parameter called lastAccountID that is equal to the watermark value.

```
#[{postal: 'yourValue', lastAccountID: vars.lastAccountID}]
```

31. Modify the SQL query text to use this parameter to only retrieve new records.

```
SELECT * FROM accounts WHERE postal = :postal AND accountID > :lastAccountID
```

SQL Query Text:

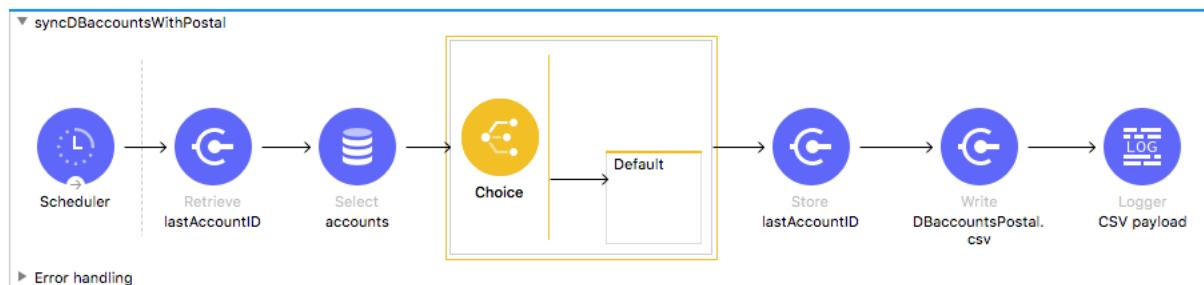
```
SELECT *
FROM accounts
WHERE postal = :postal AND accountID > :lastAccountID
```

Input Parameters:

```
#[{ postal: '94111', lastAccountID: vars.lastAccountID}]
```

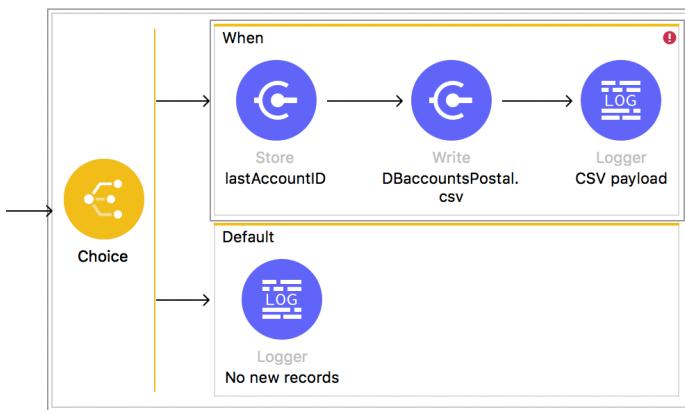
Only store the value if new records were retrieved

32. Add a Choice router after the Select operation.

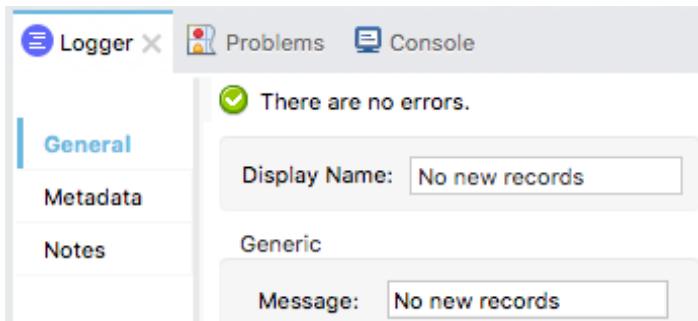


33. Move the Store, Write, and Logger operations into the when branch of the router.

34. Add a Logger to the default branch.

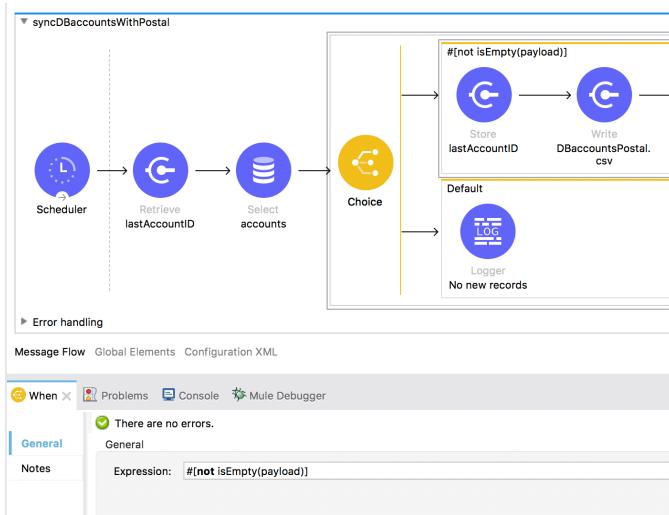


35. In the Logger properties view, set the display name and value to No new records.



36. In the Choice router, click the when scope and in the properties view, add an expression to route the event when the payload is not empty.

```
##[not isEmpty(payload)]
```



Clear the application data and debug the application

37. Add a breakpoint to the Retrieve operation.
38. Debug the project.
39. In the Clear Application dialog box, select Yes.
40. In the Mule Debugger, step to the Select operation; you should see a lastAccountID variable with a value of 0.

Mule Debugger

Name	Value
⑧ Message	
⑧ Payload (mimeType="*/")	null
▼ e Variables	size = 1
► e 0	lastAccountID=0
LastAccountID=0	

accounts X global config.yaml accounts

```
graph LR; Scheduler((Scheduler)) --> Retrieve((Retrieve)); Retrieve -- "lastAccountID" --> Select[Select accounts]; Select --> Choice((Choice))
```

41. Step to the Choice router; you should see record(s) were retrieved from the database.

Mule Debugger

Name	Value
⑧ Encoding	UTF-8
⑧ Message	
▼ e Payload (mimeType="application/json")	size = 3
► e 0	{accountID=6702, country=United States, street=77 Geary Street, st}
► e 1	{accountID=6706, country=United States, street=77 Geary Street, st}
► e 2	{accountID=6707, country=United States, street=77 Geary Street, st}
► e Variables	size = 1

accounts X global config.yaml

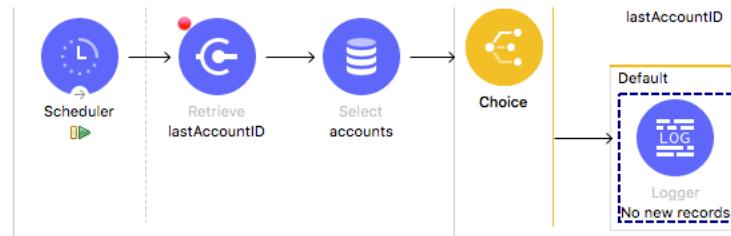
```
graph LR; alert[alert] --> Choice((Choice)); Choice --> Store[Store lastAccountID]; Store --> Write[Write DBaccountsPostal.csv]; Write --> Logger[Logger payload]
```

42. Step through the rest of the application and click resume; the flow should be executed again, and execution should be stopped back at the beginning of the flow.

43. Step to the Select operation; you should see the lastAccountID variable now has a non-zero value.

Name	Value
Message	null
Payload (mimeType="*/")	null
Variables	size = 1
e 0	lastAccountID=6707
lastAccountID=6707	

44. Step into the Choice router; you should see no records were retrieved from the database (unless someone else just added one with the same postal code!).



45. Stop the project and switch perspectives.

46. Return to your computer's file explorer and locate and open the new DBaccountsPostal.csv file.

Note: If you see the same records more than once, it was because during debugging, the flow was executed again before the watermark was stored.

Debug the application and do not clear application data

47. Return to Anypoint Studio and debug the project.
 48. In the Clear Application dialog box, select No.
 49. In the Mule Debugger, step to the Select operation; you should see a lastAccountID variable with the same non-zero value.

Name	Value
Message	null
Payload (mimeType="*/")	null
Variables	size = 1
e 0	lastAccountID=6707

50. Step to the Choice router; no new records should have been returned.
51. Remove the breakpoint from the Retrieve operation.
52. Click Resume until application execution is no longer stopped anywhere.
53. Make sure there are no other breakpoints in the flow.

Add a new account to the database

54. Return to the account data in the web browser at <http://mu.mulesoft-training.com/accounts/show>.
55. Click Create More Accounts.
56. Fill out the form with data and use the same postal code.

Create New Account Records

Name:
Maxwell Mule

Street
77 Geary Street

57. Click Create Record.
58. Click View Existing Accounts; you should see your new account.
59. Return to Anypoint Studio; you should see your new record in the console.

```

Console X Problems CSV payload
apdev-examples [Mule Applications] Mule Server 4.1.1 EE
*****
INFO 2018-04-24 14:44:26,002 [[MuleRuntime].cpuLight.15: [apdev-examples].syncDBaccountsWithPostal] USE.CPU_LITE @7983b49f [event: 0-a7241c60-4808-11e8-b9b5-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: No new records
INFO 2018-04-24 14:44:34,025 [[MuleRuntime].cpuLight.09: [apdev-examples].syncDBaccountsWithPostal] USE.CPU_LITE @7983b49f [event: 0-ad11c000-4808-11e8-b9b5-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: accountID, country, street, state, name, city, postal
6709, United States, 77 Geary Street, California, Maxwell Mule, San Francisco, 94111

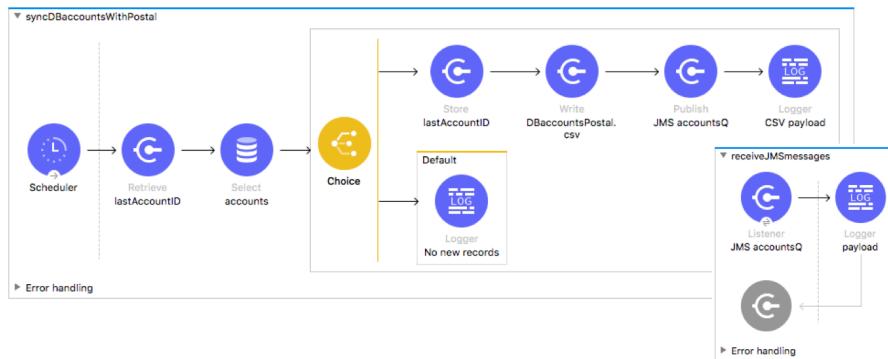
```

60. Return to your computer's file explorer; the modified date on the DBAccountsPostal.csv file should have been updated.
61. Open DBaccountsPostal.csv and locate your new record at the end of the file.
62. Return to Anypoint Studio and switch perspectives.
63. Stop the project.

Walkthrough 12-4: Publish and listen for JMS messages

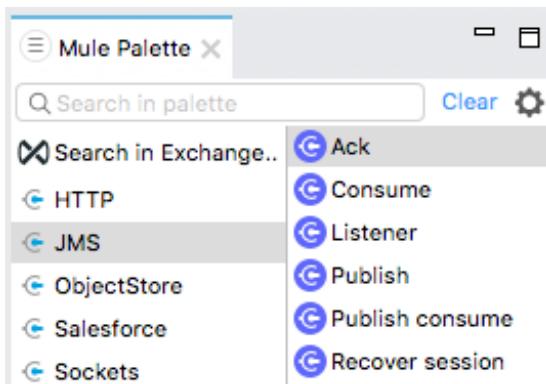
In this walkthrough, you send a JMS message for each new database record so it can be processed asynchronously. You will:

- Add and configure a JMS connector for ActiveMQ (that uses an in-memory broker).
- Send messages to a JMS queue.
- Listener for and process messages from a JMS queue.



Add the JMS module to the project

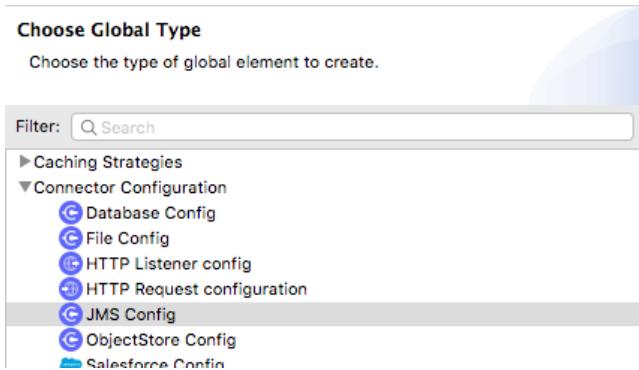
1. Return to accounts.xml.
2. In the Mule Palette, select Add Modules.
3. Select the JMS connector in the right side of the Mule Palette and drag and drop it into the left side.
4. If you get a Select module version dialog box, select the latest version and click Add.



Configure the JMS connector

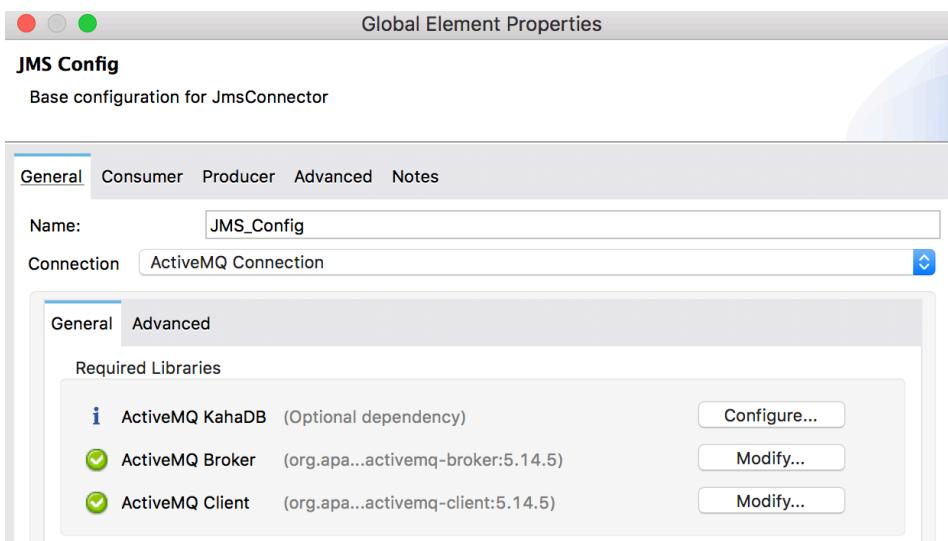
5. Return to the Global Elements view of global.xml.
6. Click Create.

- In the Choose Global Type dialog box, select Connector Configuration > JMS Config and click OK.

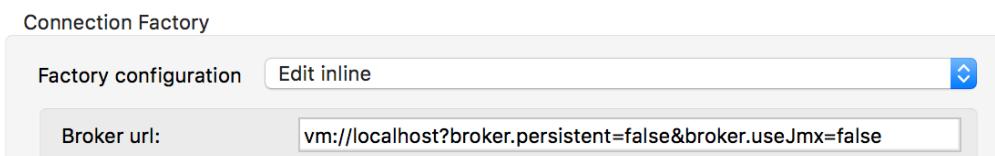


- In the Global Element Properties dialog box, make sure the connection is set to ActiveMQ Connection.
- Click the Configure > Add Maven dependency option for the ActiveMQ Broker.
- In the Maven dependency dialog box, enter activemq-broker in the Search Maven Central field.
- Select the org.apache.activemq:activemq-broker entry and select Finish.

Note: There are local versions of the drivers available in the student files.



- Scroll down and locate the Connection Factory section.
- In the Factory configuration, select Edit inline; the Broker url should already be populated.

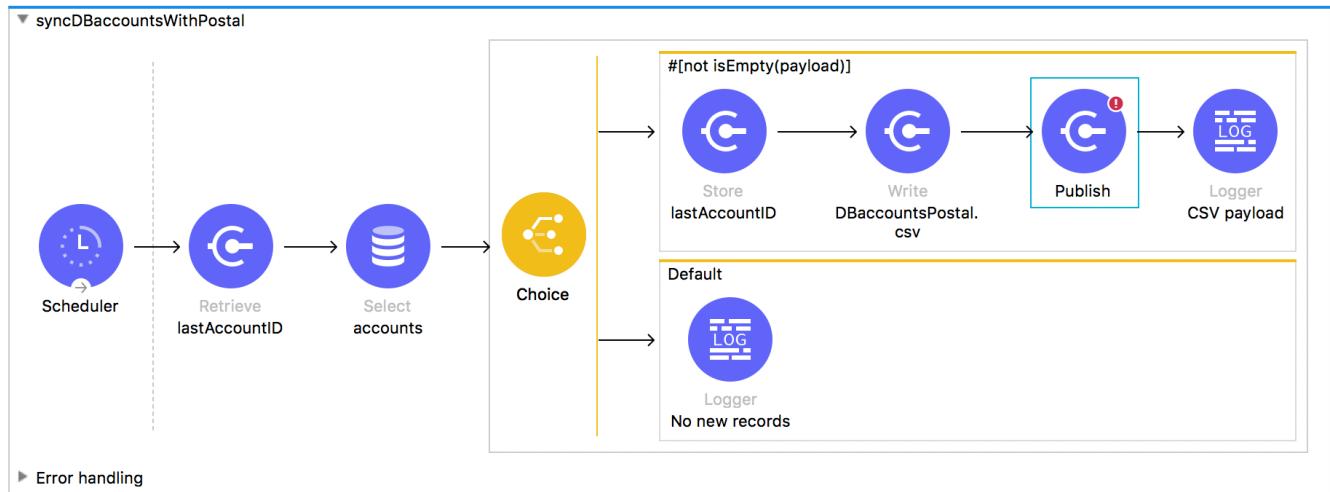


- Click OK.

Use the JMS Publish operation to send a message to a JMS queue

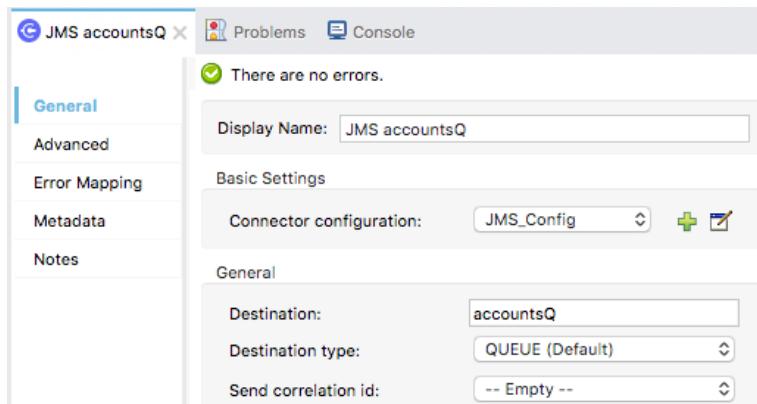
15. Return to accounts.xml.

16. Drag out a JMS Publish operation from the Mule Palette and drop it after the Write operation in syncDBaccountsWithPostal.



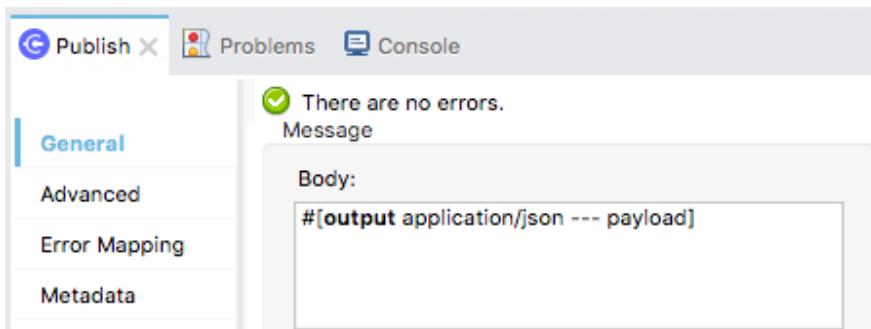
17. In the Publish properties view, set the following values:

- Display Name: JMS accountsQ
- Connector configuration: JMS_Config
- Destination: accountsQ
- Destination type: QUEUE



18. Modify the message body to be of type application/json.

```
##[output application/json --- payload]
```



19. Scroll down and locate the User Properties field.

20. Set it equal to an object a key called publisher with a value of training.

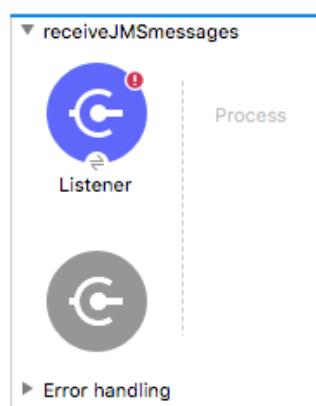
```
#>{"publisher":"training"}
```



Create a flow to listen to a JMS topic

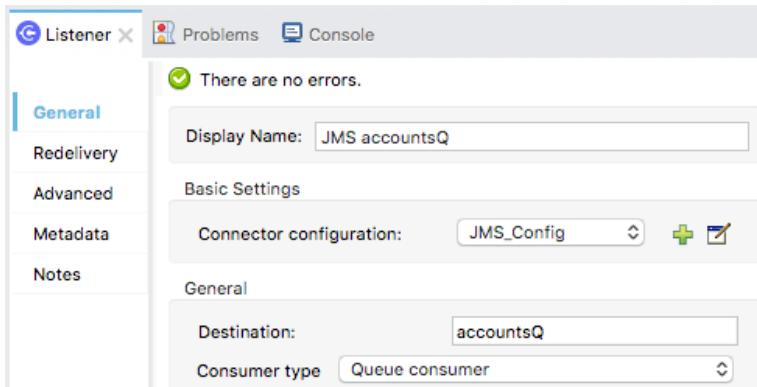
21. Drag out the JMS Listener operation from the Mule Palette and drop it in the canvas to create a new flow.

22. Give the flow a new name of receiveJMSMessages.



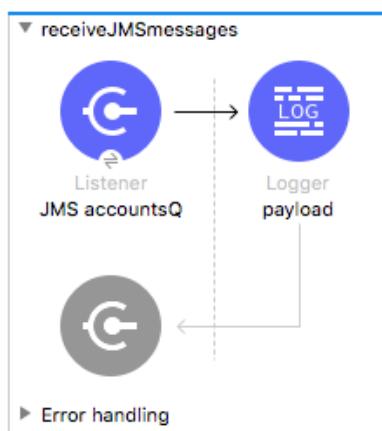
23. In the Listener properties view, set the following values:

- Display Name: JMS accountsQ
- Connector configuration: JMS_Config
- Destination: accountsQ
- Consumer type: Queue consumer



24. Add a Logger to the flow.

25. Set its display name to payload and its message to #[payload].



Clear application data and debug the application

26. Add a breakpoint to the Logger in receiveJMSmessages.

27. Debug the project.

28. In the Clear Application Data dialog box, select Yes.

29. Wait until the project deploys; application execution should stop in receiveJMSMessages.
30. In the Mule Debugger view, look at the value of the payload.

The screenshot shows the Mule Debugger interface. At the top, there's a header bar with tabs for 'accounts', 'global', and 'config.yaml'. Below this is a table with two columns: 'Name' and 'Value'. Under 'Name', there are entries for 'Message' and 'Payload (mimeType="application/json;)'. The 'Payload' entry is expanded, showing a JSON object with fields: 'accountID': 6702, 'country': 'United States', 'street': '77 Geary Street', and 'state': 'California'. Below the table is a diagram of a Mule flow. It starts with a 'Listener JMS accountsQ' component, followed by a dashed arrow pointing to a 'Logger payload' component. The 'Logger payload' component is highlighted with a red box.

31. Expand the Attributes and locate the publisher property in the userProperties.

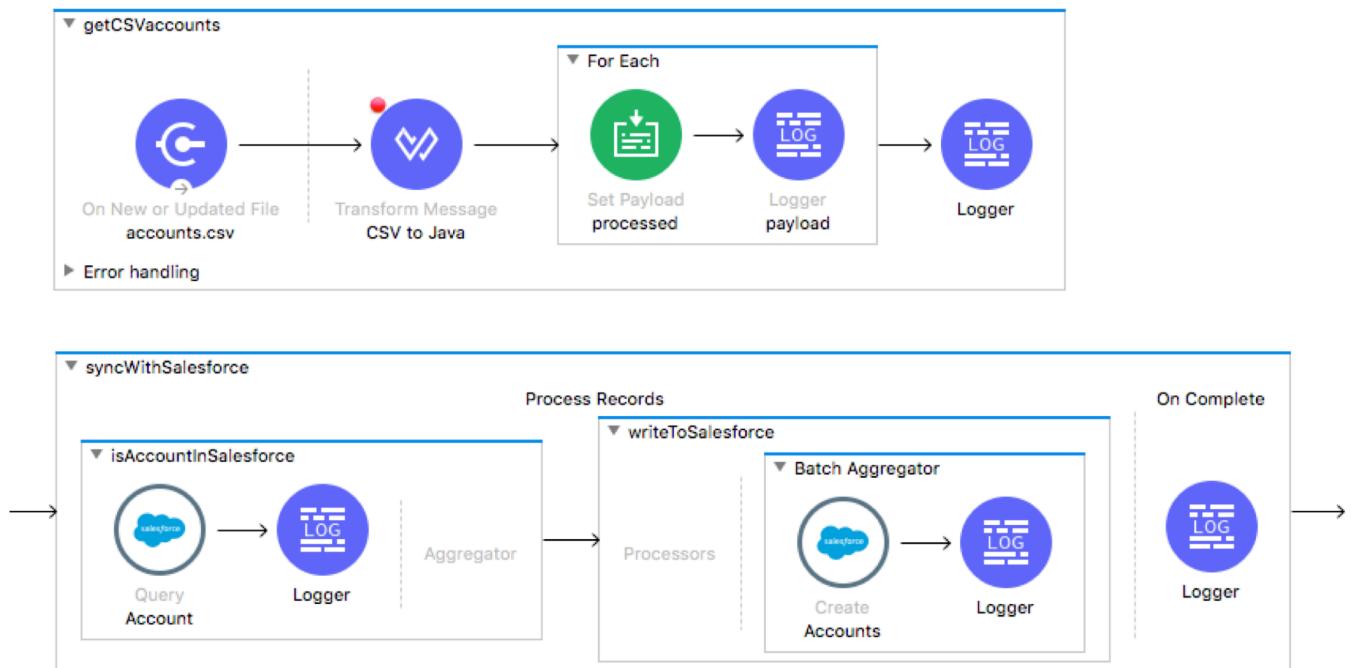
Name	Value
Attributes	org.mule.extensions.jms.api.message.JmsAttributes@33934a68
ackId	null
headers	org.mule.extensions.jms.api.message.JmsHeaders@714e8d79
properties	org.mule.extensions.jms.api.message.JmsMessageProperties@7e
all	{MM_MESSAGE_ENCODING=UTF-8, MM_MESSAGE_CONTENT_TY
JMS_PREFIX	JMS
jmsProperties	{}
JMSX_PREFIX	JMSX
jmsxProperties	org.mule.extensions.jms.api.message.JmsxProperties@5abc457c
userProperties	{MM_MESSAGE_ENCODING=UTF-8, MM_MESSAGE_CONTENT_TY
0	MM_MESSAGE_ENCODING=UTF-8
1	MM_MESSAGE_CONTENT_TYPE=application/json; charset=UTF-8
2	publisher=training

32. Step through the rest of the application; you should see the JSON message in the console.

Note: If you want to test further, return to the account data in the web browser at <http://mu.mulesoft-training.com/accounts/show> and add a new record with the same postal code.

33. Stop the project and switch perspectives.

Module 13: Processing Records



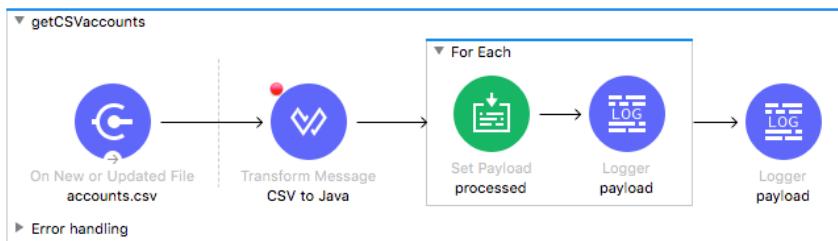
At the end of this module, you should be able to:

- Process items in a collection using the For Each scope.
- Process records using the Batch Job scope.
- Use filtering and aggregation in a batch step.

Walkthrough 13-1: Process items in a collection using the For Each scope

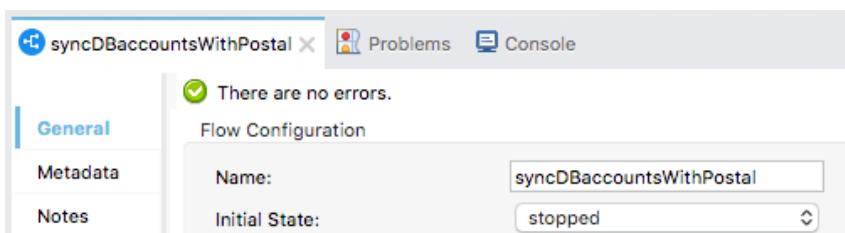
In this walkthrough, you split a collection and process each item in it. You will:

- Use the For Each scope element to process each item in a collection individually.
- Change the value of an item inside the scope.
- Examine the payload before, during, and after the scope.
- Look at the thread used to process each item.



Stop the syncDBaccountsWithPostal flow so it does not run

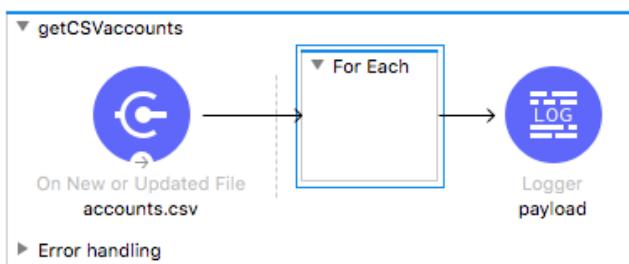
1. Return to accounts.xml.
2. In the properties view for the syncDBaccountsWithPostal flow, set the initial state to stopped.



3. Right-click in the canvas and select Collapse All.

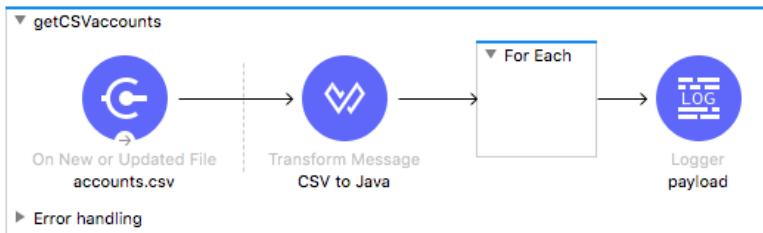
Add a For Each scope

4. Expand getCSVaccounts.
5. In the Mule Palette, select Core.
6. Locate the For Each scope and drag and drop it before the Logger.

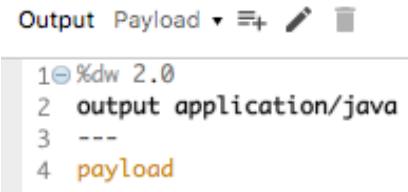


Transform the input to a collection

7. Add a Transform Message component before the For Each scope.
8. Set its display name to CSV to Java.



9. In the Transform Message properties view, leave the output type set to java and set the expression to payload.

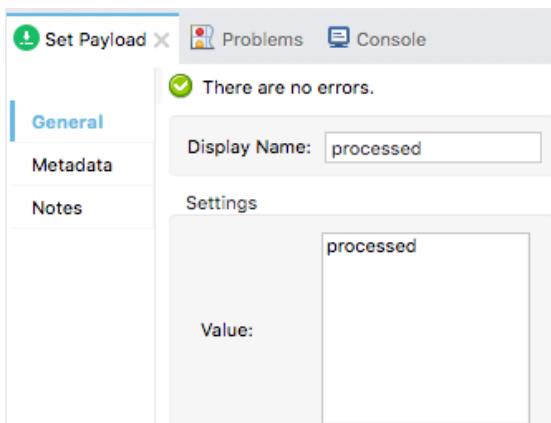


Process each element in the For Each scope

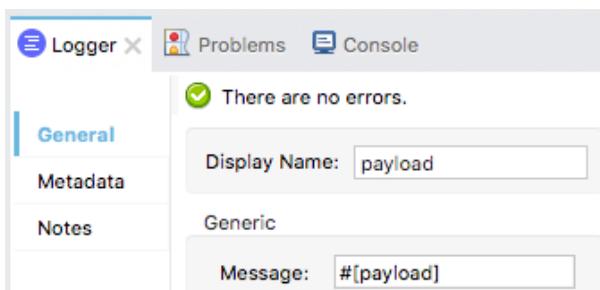
10. Add a Set Payload transformer and a Logger to the For Each scope.



11. In the Set Payload properties view, set the display name and value to the string: processed.

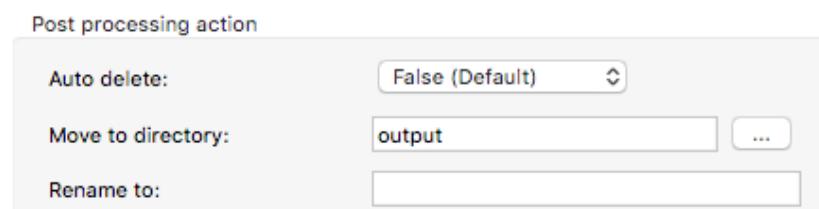


12. Set the Logger display name to payload and have it display the payload.



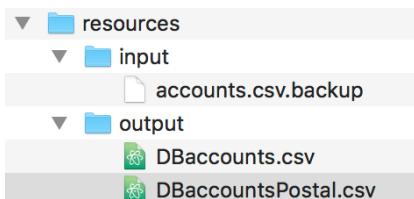
Change the File listener so it does not rename files

13. In the On New or Updated File properties view, delete the rename to value.



Debug the application

14. Add a breakpoint to the Transform Message component.
15. Debug the project and do not clear application data.
16. Return to the course student files in your computer's file browser.



17. Rename accounts.csv.backup to accounts.csv.
18. Return to Anypoint Studio; application execution should have stopped at the Transform Message component.

19. In the Mule Debugger view, look at the payload type and value.

The screenshot shows the Mule Debugger interface. At the top, there's a table with columns 'Name' and 'Value'. The 'Payload (mimeType="text/csv")' row is expanded, showing the CSV data:
Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode
Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode
111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008
400 South St, San Francisco,USA,CA,Iguana Park Industries,91156
777 North St, San Francisco,USA,CA,Cat Park Industries,91156

Below the table, the Mule flow configuration is visible. It starts with an 'On New or Updated File' connector reading 'accounts.csv'. This is followed by a 'Transform Message' step labeled 'CSV to Java'. The output of this step enters a 'For Each' scope. Inside the scope, there are three steps: 'Set Payload processed', 'Logger payload', and another 'Logger' step. The flow then continues with another 'Logger' step.

20. Step to the For Each scope; the payload should now be an ArrayList of LinkedHashMaps.

Name	Value	Type
Payload (mimeType="text/csv")	size = 3	java.util.ArrayList
e 0	{Billing Street=111 Boule...	java.util.LinkedHashMap
e 1	{Billing Street=400 South...	java.util.LinkedHashMap
e 2	{Billing Street=777 North...	java.util.LinkedHashMap
e 0	Billing Street=777 North St	java.util.LinkedHashMap\$...
e 1	Billing City=San Francisco	java.util.LinkedHashMap\$...

21. Step into the For Each scope; the payload should be a LinkedHashMap.

22. Expand Variables; you should see a counter variable.

The screenshot shows the Mule Debugger interface with expanded variables. The 'Variables' section shows a 'counter' variable with a value of 1. Below the debugger, the Mule flow configuration is shown again, identical to the one in step 19.

23. Step again; you should see the payload for this record inside the scope has been set to the string, processed.

Name	Value	Type
Payload (mimeType="*/*")	processed	java.lang.String
Variables		
▶ 0	size = 2	java.util.HashMap
▶ 1	rootMessage= counter=1	java.util.HashMap\$Node
processed		

accounts X global config.yaml

getCSVaccounts

```

graph LR
    C((C)) --> TM[Transform Message  
CSV to Java]
    TM --> FE[For Each]
    FE --> SP[Set Payload  
processed]
    SP --> LP[Logger  
payload]
    LP --> L[Logger]

```

On New or Updated File accounts.csv
Transform Message CSV to Java
For Each
Set Payload processed
Logger payload
Logger
Error handling

24. Step again and look at the payload and counter for the second record.

Name	Value	Type
Payload (mimeType="application/x-www-form-urlencoded")	java.util.LinkedHashMap	
▶ 0	Billing Street=400 South St	java.util.LinkedHashMap\$Entry
▶ 1	Billing City=San Francisco	java.util.LinkedHashMap\$Entry
▶ 2	Billing Country=USA	java.util.LinkedHashMap\$Entry
▶ 3	Billing State=CA	java.util.LinkedHashMap\$Entry
▶ 4	Name=Iguana Park Industries	java.util.LinkedHashMap\$Entry
▶ 5	BillingPostalCode=91156	java.util.LinkedHashMap\$Entry
Variables		
▶ 0	size = 2	java.util.HashMap
▶ 1	rootMessage= counter=2	java.util.HashMap\$Node

25. Step through the application to the Logger after the For Each scope; the payload should be equal to the original ArrayList of HashMaps and not a list of processed strings.

Name	Value
Message	
Payload (mimeType="application/x-www-form-urlencoded")	size = 3
▶ 0	{Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France,
▶ 1	{Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing
▶ 2	{Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing
{Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing	Stat

accounts X

getCSVaccounts

```

graph LR
    C((C)) --> TM[Transform Message  
CSV to Java]
    TM --> FE[For Each]
    FE --> SP[Set Payload  
processed]
    SP --> LP[Logger  
payload]
    LP --> L[Logger]

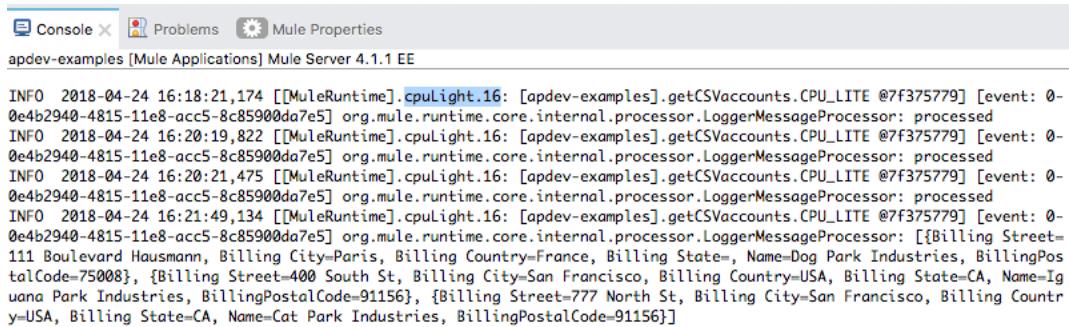
```

On New or Updated File accounts.csv
Transform Message CSV to Java
For Each
Set Payload processed
Logger payload
Logger
Error handling

26. Step to the end of the application.
27. Stop the project and switch perspectives.

Look at the processing threads

28. In the console, locate the thread number used to process each item in the collection; the same thread should be used for each: cpuLight.16 in the following screenshot.



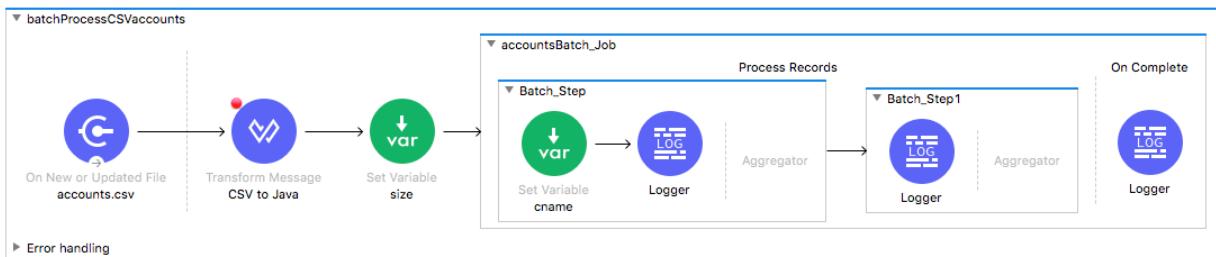
```
Console X Problems Mule Properties
apdev-examples [Mule Applications] Mule Server 4.1.1 EE

INFO 2018-04-24 16:18:21,174 [[MuleRuntime].cpuLight.16: [apdev-examples].getCSVaccounts.CPU_LITE @7f375779] [event: 0-0e4b2940-4815-11e8-acc5-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: processed
INFO 2018-04-24 16:20:19,822 [[MuleRuntime].cpuLight.16: [apdev-examples].getCSVaccounts.CPU_LITE @7f375779] [event: 0-0e4b2940-4815-11e8-acc5-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: processed
INFO 2018-04-24 16:20:21,475 [[MuleRuntime].cpuLight.16: [apdev-examples].getCSVaccounts.CPU_LITE @7f375779] [event: 0-0e4b2940-4815-11e8-acc5-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: processed
INFO 2018-04-24 16:21:49,134 [[MuleRuntime].cpuLight.16: [apdev-examples].getCSVaccounts.CPU_LITE @7f375779] [event: 0-0e4b2940-4815-11e8-acc5-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: [{Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}, {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}, {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Cat Park Industries, BillingPostalCode=91156}]
```

Walkthrough 13-2: Process records using the Batch Job scope

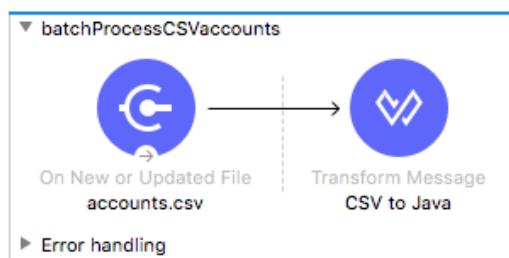
In this walkthrough, you create a batch job to process the records in a CSV file. You will:

- Use the Batch Job scope to process items in a collection.
- Examine the payload as it moves through the batch job.
- Explore variable persistence across batch steps and phases.
- Examine the payload that contains information about the job in the on complete phase.
- Look at the threads used to process the records in each step.



Create a new flow to read CSV files

1. Return to accounts.xml.
2. Drag a Flow scope from the Mule Palette and drop it above getCSVaccounts.
3. Change the flow name to batchProcessCSVaccounts.
4. Select the On New or Updated File operation in getCSVaccounts and select Edit > Copy.
5. Click the Source section of batchProcessCSVaccounts and select Edit > Paste.
6. Modify the display name.
7. Add a Transform Message component to the flow.
8. Set the display name to CSV to Java.

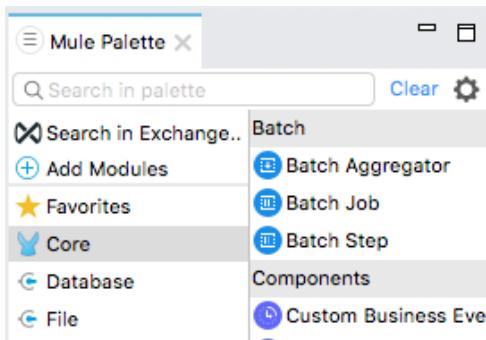


9. In the Transform Message properties view, change the body expression to payload.

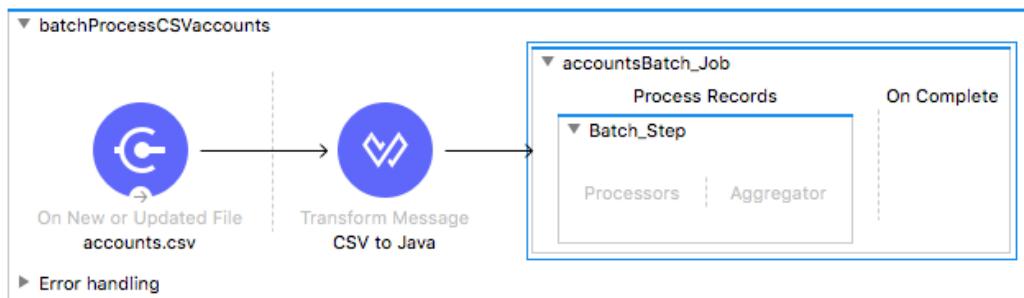
```
Output Payload ▾ + ✎ ━  
1 @%dw 2.0  
2 output application/java  
3 ---  
4 payload
```

Add a Batch Job scope

10. In the Mule Palette, select Core and locate the Batch elements.

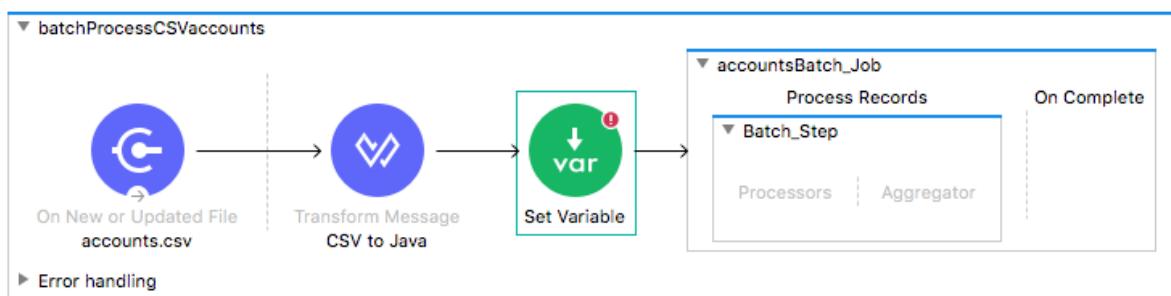


11. Drag a Batch Job scope and drop it after the Transform Message component.



Set a variable before the Batch Job scope

12. Add a Set Variable transformer before the Batch Job scope.

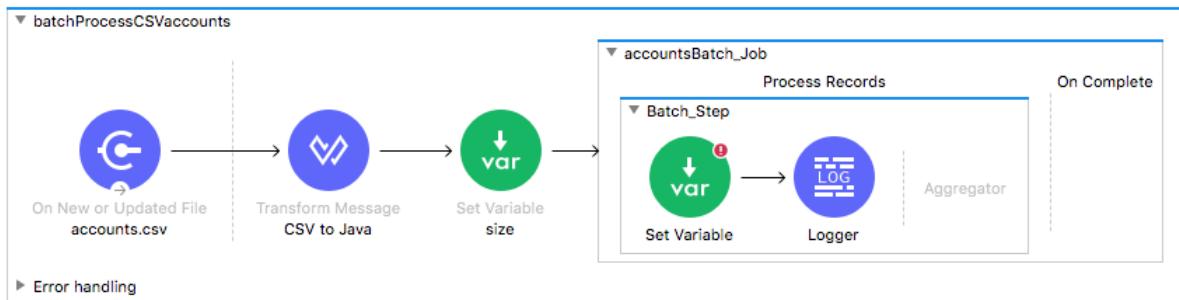


13. In the Set Variable properties view, set the following:

- Display Name: size
- Name: size
- Value: #[sizeOf(payload)]

Set a variable inside the Batch Job scope

14. Add a Set Variable transformer to the batch step in the processors phase of the batch step.
15. Add a Logger component to the batch step after the transformer.

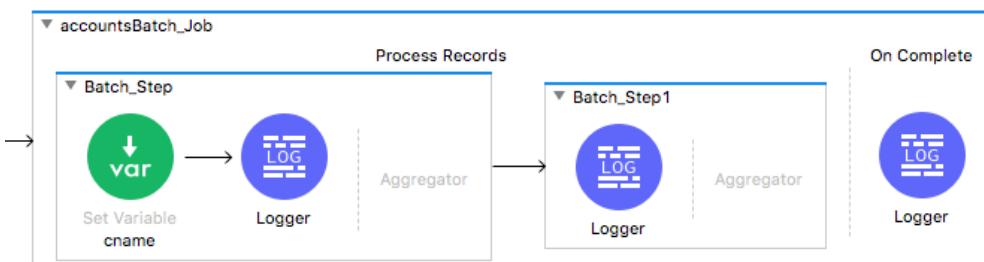


16. In the Set Variable properties view, set the following:

- Display Name: cname
- Name: cname
- Value: #[payload.Name]

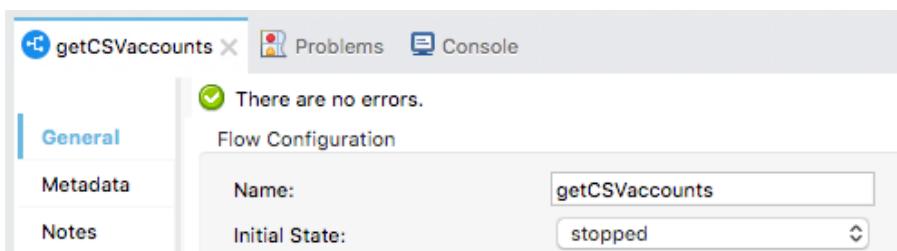
Create a second batch step

17. Drag a Batch Step scope from the Mule Palette and drop it in the process records phase of the Batch Job scope after the first batch step.
18. Add a Logger to the processors section of the second batch step.
19. Add a Logger to the On Complete phase.



Stop the other flow watching the same file directory from being executed

20. In the properties view for the getCSVaccounts flow, set the initial state to stopped.



Debug the application

21. In batchProcessCSVaccounts, add a breakpoint to the Transform Message component.
22. Save the file to redeploy the application in debug mode.
23. Return to the course student files in your computer's file browser and move accounts.csv from the output folder to the input folder.
24. In the Mule Debugger view, watch the payload as you step to the Batch Job scope; it should go from CSV to an ArrayList.
25. In the Mule Debugger view, expand Variables; you should see the size variable.

The screenshot shows the Mule Debugger interface at the top, displaying a table of variables. One variable, 'Payload', is expanded to show its contents as an ArrayList of three elements, each being a LinkedHashMap. Below the debugger is the Mule Studio interface, showing a flow diagram. A variable named 'size' is being passed into a 'Batch_Step' component. Inside the 'Batch_Step' component, there is a 'Set Variable' node named 'cname' and a 'Logger' node. The flow then continues through an 'Aggregator' and another 'Batch_Step1' component, which also contains a 'Logger' node. On the left side of the Mule Studio interface, there is a green circle icon with a downward arrow and the word 'variable' below it, indicating the current context of the variable being debugged.

26. Step to the Logger in the first batch step.
27. In the Mule Debugger view, look at the value of the payload; it should be a HashMap.
28. Expand Variables; you should see the size variable and the cname variable specific for that record.

The screenshot shows the Mule Debugger and Mule Studio interface. The Mule Debugger window displays the payload and variables for a specific record. The payload is a LinkedHashMap with entries for Billing Street, City, Country, State, Name, and Postal Code. The Variables section shows a 'size' variable and a 'cname' variable, which is highlighted. The Mule Studio interface below shows a 'accountsBatch_Job' process. It consists of two main steps: 'Batch_Step' and 'Batch_Step1'. 'Batch_Step' contains a 'Set Variable cname' node and a 'Logger' node. 'Batch_Step1' also contains a 'Logger' node. Between the two steps is an 'Aggregator' node. Arrows indicate the flow from 'Batch_Step' to 'Batch_Step1' through the aggregator.

Name	Value	Type
Payload (mimeType="*/")	size = 6	java.util.LinkedHashMap
▶ e 0	Billing Street=111 Boulevard Hausmann	java.util.LinkedHashMap\$Entry
▶ e 1	Billing City=Paris	java.util.LinkedHashMap\$Entry
▶ e 2	Billing Country=France	java.util.LinkedHashMap\$Entry
▶ e 3	Billing State=	java.util.LinkedHashMap\$Entry
▶ e 4	Name=Dog Park Industries	java.util.LinkedHashMap\$Entry
▶ e 5	BillingPostalCode=75008	java.util.LinkedHashMap\$Entry
▼ e Variables	size = 4	java.util.HashMap
▶ e 0	size=3	java.util.HashMap\$Node
▶ e 1	cname=Dog Park Industries	java.util.HashMap\$Node
▶ e 2	batchJobInstanceId=f08e73d0-4819-11e...	java.util.HashMap\$Node
▶ e 3	_mule_batch_INTERNAL_record=com.mule...	java.util.HashMap\$Node

cname=Dog Park Industries

accounts global config.yaml

```

graph LR
    subgraph accountsBatch_Job [accountsBatch_Job]
        subgraph Batch_Step [Batch_Step]
            direction TB
            var((var)) --> log1[Logger]
            log1 --> aggregator1[Aggregator]
        end
        subgraph Batch_Step1 [Batch_Step1]
            direction TB
            aggregator1 --> log2[Logger]
        end
    end

```

29. Step through the rest of the records in the first batch step and watch the payload and the cname variable change.

The screenshot shows the Mule Debugger interface. The 'Variables' section is expanded, and the 'cname' variable is highlighted for the second record (index 1). The value for 'cname' is now 'Iguana Park Industries', indicating a change from the first record's 'Dog Park Industries'.

Name	Value	Type
▼ e Variables	size = 4	
▶ e 0	size=3	
▶ e 1	cname=Iguana Park Industries	
▶ e 2	batchJobInstanceId=f08e73d0-4819-11e...	
▶ e 3	_mule_batch_INTERNAL_record=com.mule...	

30. Step into the second batch step and look at the payload and the cname variable; you should see the cname variable is defined and has a value.

The screenshot shows the Mule Debugger interface. At the top, there is a table of variables:

Name	Value
Payload (mimeType="*/*")	size = 6
Variables	size = 4
0	size=3
1	cname=Dog Park Industries
2	batchJobInstanceId=f08e73d0-4819-11e8-acc5-8c85900da7e5
3	_mule_batch_INTERNAL_record=com.mulesoft.mule.runtime.module.batch.ap

Below the table, the value for variable '1' is expanded to show 'cname=Dog Park Industries'. At the bottom of the debugger window, there are tabs for 'accounts', 'global', and 'config.yaml'.

Below the debugger, the Mule flow diagram is shown:

```

graph LR
    Start(( )) --> SetVar[Set Variable cname]
    SetVar --> Logger1[Logger]
    subgraph Process_Records [Process Records]
        SetVar
        Logger1
        Agg1[Aggregator]
        Agg1 --> BatchStep1[Batch Step 1]
        subgraph BatchStep1 [Batch Step 1]
            Agg1
            Log1[Logger]
        end
    end
    BatchStep1 --> OnComplete[On Complete]
    OnComplete --> Logger2[Logger]

```

The flow starts with an 'accounts' source, followed by a 'Set Variable cname' step, then a 'Logger' step. This is followed by an 'Aggregator' step, which then leads to a 'Batch Step 1' step. Inside 'Batch Step 1', there is another 'Aggregator' step followed by a 'Logger' step. Finally, the flow ends with an 'On Complete' phase containing a 'Logger' step.

31. Step through the rest of the records in the second batch step and watch the value of cname.
32. Step into the on complete phase; you should see the payload is an ImmutableBatchJobResult.
33. Expand the payload and locate the values for totalRecords, successfulRecords, and failedRecords.

The screenshot shows the Mule Debugger interface. At the top, there is a table of variables:

Name	Value
Payload (mimeType="*/*")	com.mulesoft.mule.runtime.module.batch.internal.ImmutableBatchJobResult@f08e73d0-4819-11e8-acc5-8c85900da7e5
batchJobInstanceId	321033
elapsedTimeInMillis	false
failedOnCompletePhase	false
failedOnInputPhase	false
failedOnLoadingPhase	false
failedRecords	0
inputPhaseException	null
loadedRecords	3
loadingPhaseException	null
onCompletePhaseException	null
processedRecords	3
serialVersionUID	4323747859995526737
stepResults	{Batch Step=com.mulesoft.mule.runtime.module.batch.internal.ImmutableBatchJobResult@f08e73d0-4819-11e8-acc5-8c85900da7e5}
successfulRecords	3
totalRecords	3
Variables	size = 2
0	size=3

Below the table, the value for variable '0' is expanded to show 'size=3'. At the bottom of the debugger window, there are tabs for 'accounts', 'global', and 'config.yaml'.

Below the debugger, the Mule flow diagram is shown:

```

graph LR
    Start(( )) --> SetVar[Set Variable cname]
    SetVar --> Logger1[Logger]
    subgraph Process_Records [Process Records]
        SetVar
        Logger1
        Agg1[Aggregator]
        Agg1 --> BatchStep1[Batch Step 1]
        subgraph BatchStep1 [Batch Step 1]
            Agg1
            Log1[Logger]
        end
    end
    BatchStep1 --> OnComplete[On Complete]
    OnComplete --> Logger2[Logger]

```

The flow starts with an 'accounts' source, followed by a 'Set Variable cname' step, then a 'Logger' step. This is followed by an 'Aggregator' step, which then leads to a 'Batch Step 1' step. Inside 'Batch Step 1', there is another 'Aggregator' step followed by a 'Logger' step. Finally, the flow ends with an 'On Complete' phase containing a 'Logger' step.

34. Step through the rest of the application and switch perspectives.

35. Stop the project.

Look at the processing threads

36. In the console, locate the thread number used to process each record in the collection in each step of the batch process; you should see more than one thread used.

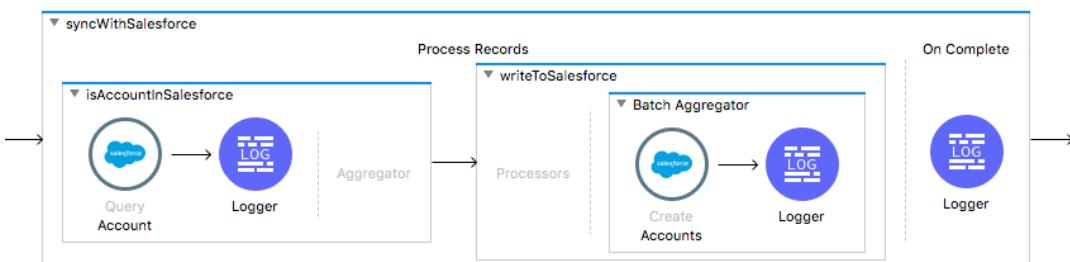


```
Console X Problems Mule Properties
apdev-examples [Mule Applications] Mule Server 4.1.1 EE
}
INFO 2018-04-24 17:06:06,907 [[MuleRuntime].io.02: [apdev-examples].batch-job-accountsBatch_Job-work-manager @722ce329]
[event: 0-6222eba0-481c-11e8-aa15-8c85900da7e5] com.mulesoft.mule.runtime.module.batch.internal.DefaultBatchStep: Step
Batch_Step finished processing all records for instance 65904d00-481c-11e8-aa15-8c85900da7e5 of job accountsBatch_Job
INFO 2018-04-24 17:06:07,963 [[MuleRuntime].io.01: [apdev-examples].batch-job-accountsBatch_Job-work-manager @722ce329]
[event: 0-6222eba0-481c-11e8-aa15-8c85900da7e5] org.mule.runtime.core.internal.processor.LoggerMessageProcessor:
org.mule.runtime.core.internal.message.DefaultMessageBuilder$MessageImplementation
```

Walkthrough 13-3: Use filtering and aggregation in a batch step

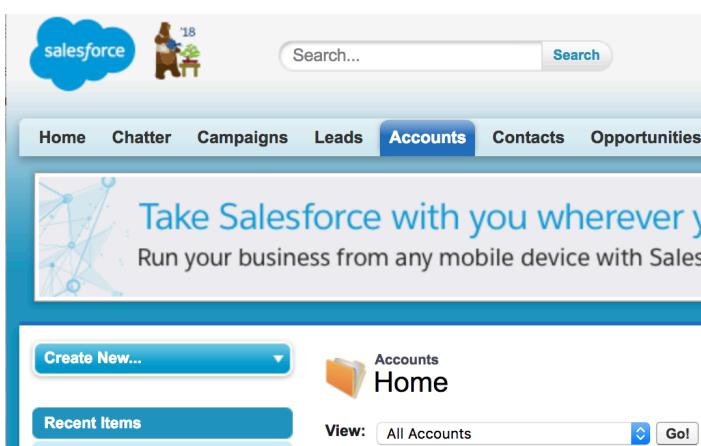
In this walkthrough, you use a batch job to synchronize account database records to Salesforce. You will:

- Use a batch job to synchronize database records (with your postal code) to Salesforce.
- In a first batch step, check to see if the record already exists in Salesforce.
- In a second batch step, add the record to Salesforce.
- Use a batch step filter so the second batch step is only executed for specific records.
- Use a Batch Aggregator scope to commit records in batches.



Look at existing Salesforce account data

1. In a web browser, navigate to <http://login.salesforce.com/> and log in with your Salesforce Developer account.
2. Click the Accounts link in the main menu bar.
3. In the view drop-down menu, select All Accounts and click the Go button.



- Look at the existing account data; a Salesforce Developer account is populated with some sample data.

Action	Account Name	Billing State/Province	Phone
Edit Del +	Burlington Textiles Co...	NC	(336) 222-7000
Edit Del +	Dickenson plc	KS	(785) 241-6200
Edit Del +	Edge Communications	TX	(512) 757-6000
Edit Del +	Express Logistics a...	OR	(503) 421-7800
Edit Del +	GenePoint	CA	(650) 867-3450
Edit Del +	Grand Hotels & Res...	IL	(312) 596-1000

- Notice that countries and postal codes are not displayed by default.
- Click the Create New View link next to the drop-down menu displaying All Accounts.
- Set the view name to All Accounts with Postal Code.
- Locate the Select Fields to Display section.
- Select Billing Zip/Postal Code as the available field and click the Add button.
- Add the Billing Country field.
- Use the Up and Down buttons to order the fields as you prefer.

Step 3. Select Fields to Display

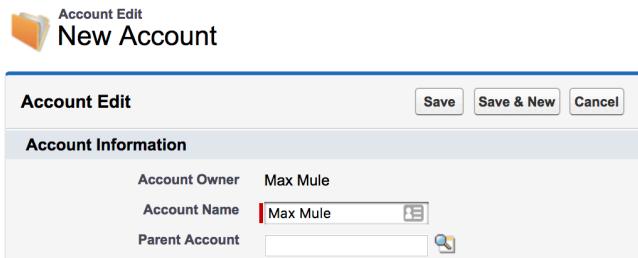
Available Fields	Selected Fields
Shipping State/Province Shipping Zip/Postal Code Shipping Country Fax Website Employees D&B Company Data.com Key SIC Code SIC Description NAICS Code NAICS Description D-U-N-S Number Tradestyle Year Started	Account Name Billing Zip/Postal Code Billing Country Billing State/Province Account Number Phone Type
Add	Top Up Down Bottom
Remove	

- Click the Save button; you should now see all the accounts with postal codes and countries.

Action	Account Name	Billing Zip/Postal Code	Billing Country	Billing State/Province	Account Number
Edit Del +	Burlington Textiles Co...	27215	USA	NC	CD656092
Edit Del +	Dickenson plc	66045	USA	KS	CC634267
Edit Del +	Edge Communications		TX		CD451796
Edit Del +	Express Logistics a...		OR		CC947211
Edit Del +	GenePoint		CA		CC978213

Add an account to Salesforce with a name matching one of the database records

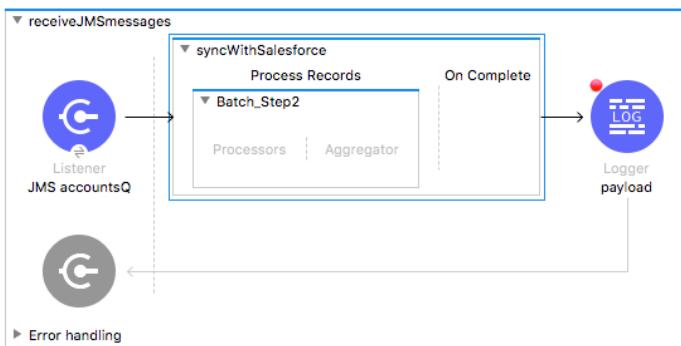
13. Click the New Account button.
14. Enter an account name (one that matches one of the accounts you added with your postal code to the database) and click Save.



15. Leave this window open.

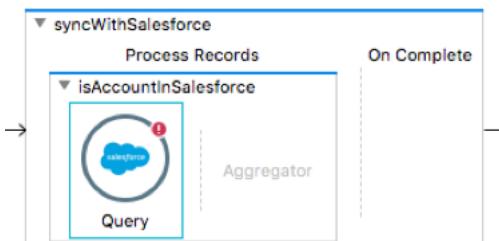
Add a Batch Job scope to the receiveJMSmessages flow

16. Return to accounts.xml in Anypoint Studio.
17. Drag a Batch Job scope from the Mule Palette and drop it before the Logger in the receiveJMSmessages flow.
18. Change the display name of the batch job to syncWithSalesforce.



Query Salesforce to see if an account already exists

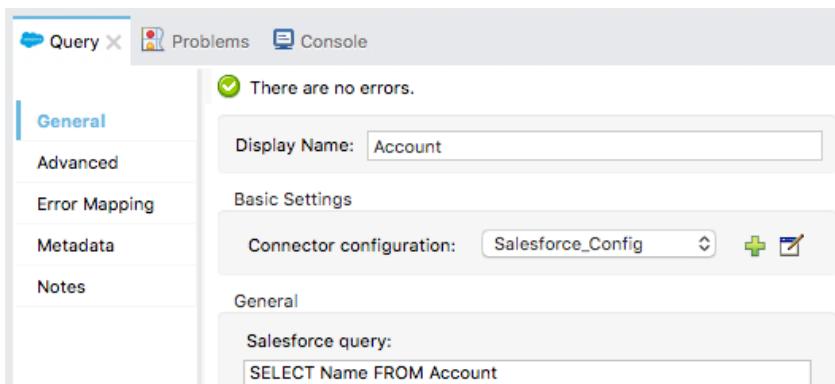
19. Change the name of the batch step inside the batch job to isAccountInSalesforce.
20. Drag a Salesforce Query operation from the Mule Palette and drop it in the processors phase in the batch step.



21. In the Query properties view, set the following values:

- Display Name: Account
- Connector configuration: Salesforce_Config
- Salesforce query: SELECT Name FROM Account

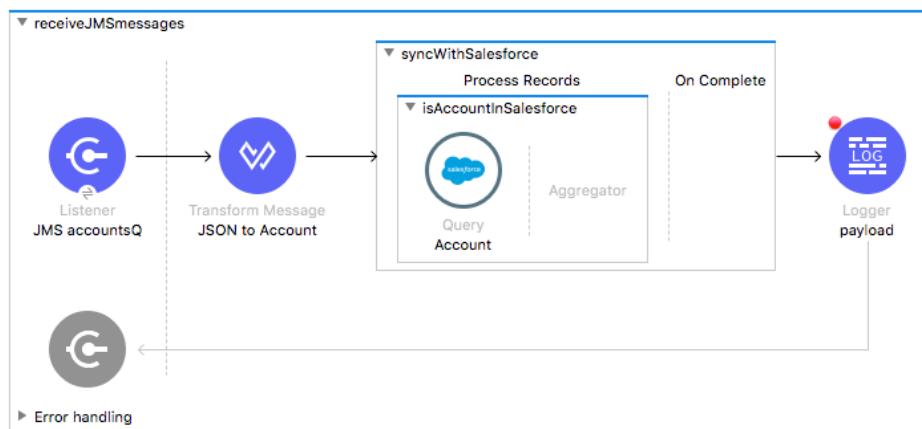
Note: You will add to this query shortly.



Transform the input JSON data to Salesforce Account objects

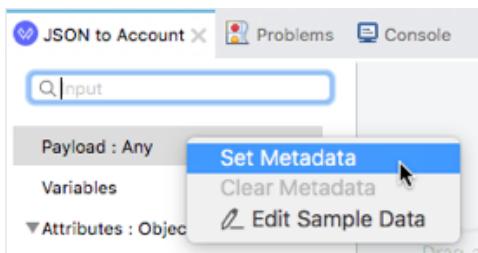
22. Add a Transform Message component before the Batch Job.

23. Change the display name to JSON to Accounts.



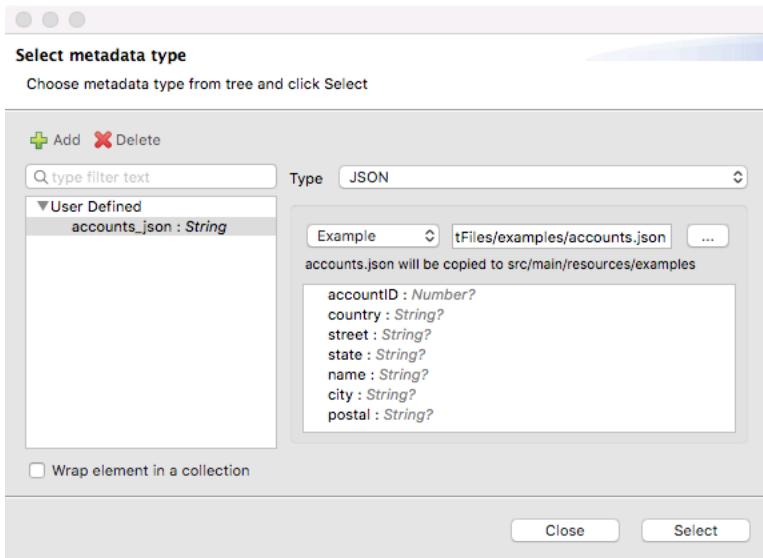
24. In the Transform Message properties view, look at the metadata in the input section.

25. Right-click Payload and select Set Metadata.



26. Create a new metadata type with the following information:

- Type id: accounts_json
- Type: JSON
- Example: Use the *accounts.json* file in the examples folder of the course student files



27. In the Transform Message properties view, map the following fields:

- country: BillingCountry
- street: BillingStreet
- state: BillingState
- name: BillingPostalCode
- city: BillingCity
- postal: BillingPostalCode

Note: If you do not get Salesforce metadata for the Account object, you can copy the DataWeave transformation from the course snippets.txt file.

```
%dw 2.0
output application/java
---
payload map ( payload01 , indexOfPayload01 ) -> {
    Name: payload01.name,
    BillingStreet: payload01.street,
    BillingCity: (payload01.city default ""),
    BillingState: payload01.state,
    BillingPostalCode: payload01.postal,
    BillingCountry: payload01.country
}
```

Finish the batch step to check if an account already exists in Salesforce

28. In the Query properties view, add an input parameter named cname.
29. Set the parameter value to payload.Name, ensure it is a String, and give it a default value.

```
payload.Name default "" as String
```

30. Modify the Salesforce query to look for accounts with this name.

```
SELECT Name  
FROM Account  
WHERE Name= ':cname'
```

Account X Problems Console

General

Salesforce query:

```
SELECT Name  
FROM Account  
WHERE Name = ':cname'
```

Parameters

Name	Value
"cname"	payload.Name default "" as String

There are no errors.

Store the result in a variable instead of overriding the payload

31. Select the Advanced tab.
32. Set the target variable to exists.
33. Set the target value to

```
#[(sizeOf(payload as Array) > 0)]
```

Account X Problems Console

General

Advanced

Error Mapping

Output

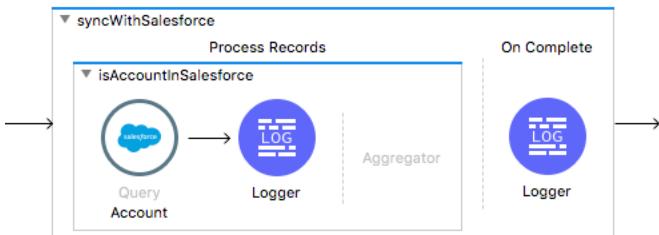
Target Variable: exists

Target Value: #[(sizeOf(payload as Array) > 0)]

There are no errors.

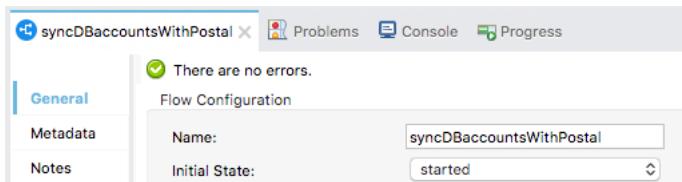
34. Add a Logger after the Query operation.

35. Add a Logger to the on complete phase.



Change the initial state of the flow

36. In the properties view for the syncDBaccountsWithPostal flow, change the initial state to started.



Debug the application

37. Add a breakpoint to the Transform Message component.

38. Debug the project.

39. Clear the application data.

40. In the Mule Debugger, wait until application execution stops in the receiveJMSmessages flow.

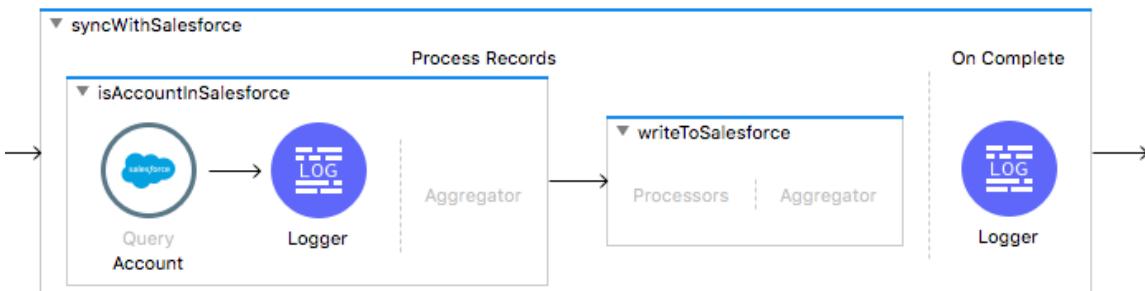
41. Step to the Logger in the first batch step and expand Variables; you should see the exists variable set to true or false.

Name	Value
Message	
Payload (mimeType="*/*")	size = 6
Variables	size = 3
0	batchJobInstanceId=c53d11a0-48be-11e8-a612-8c85900da7e5
1	exists=true
2	_mule_batch_INTERNAL_record=com.mulesoft.mule.runtime.module.batch.api.record.Record@418081e1
exists	true

42. Step through the application; you should see the exists variable set to false for records with names that don't exist in Salesforce and true for those that do.
43. Stop the project and switch perspectives.

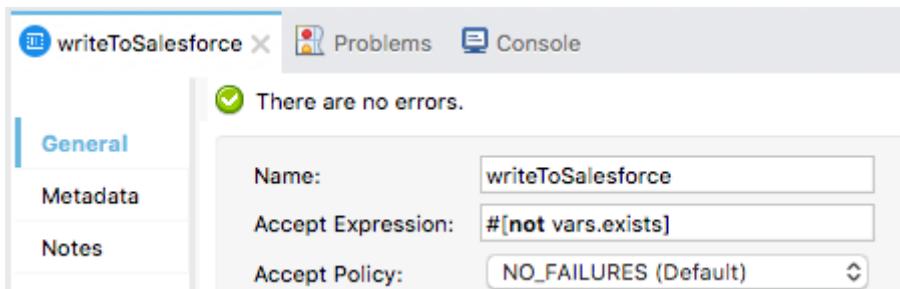
Set a filter for the insertion step

44. Add a second batch step to the batch job.
45. Change its display name to writeToSalesforce.



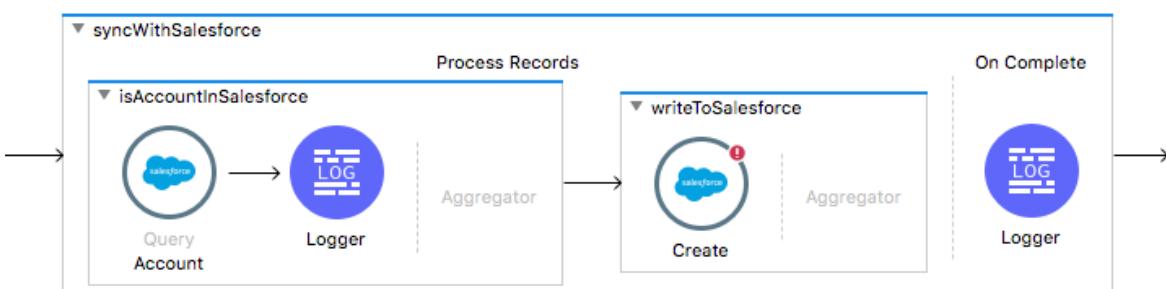
46. In the properties view for the second batch step, set the accept expression so that only records that have the variable exists set to false are processed.

```
##[not vars.exists]
```



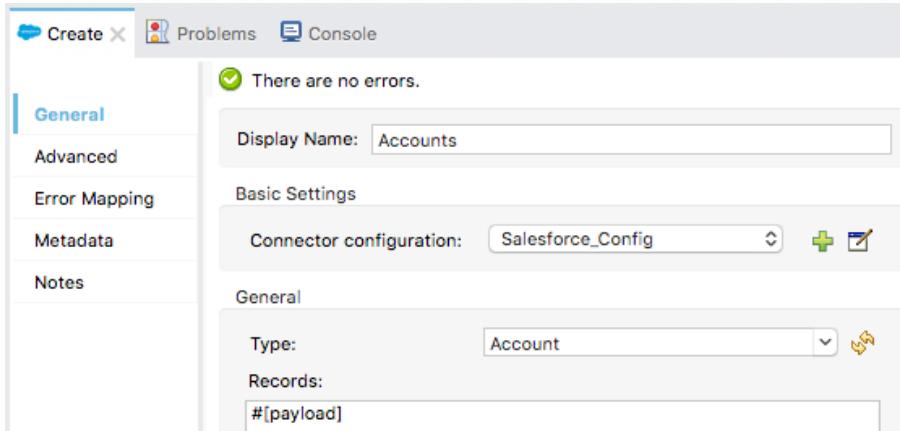
Use the Salesforce Create operation to add new account records to Salesforce

47. Add Salesforce Create operation to the second batch step in the processing phase.

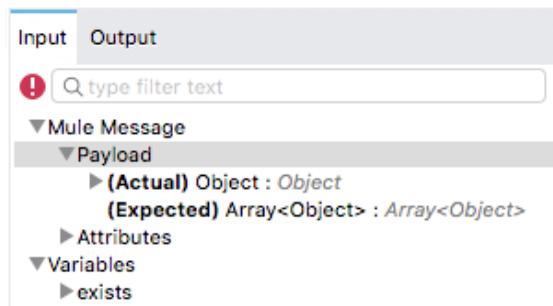


48. In the Create properties view, set the following:

- Display Name: Accounts
- Connector configuration: Salesforce_Config
- Type: Account
- Records: #[payload]



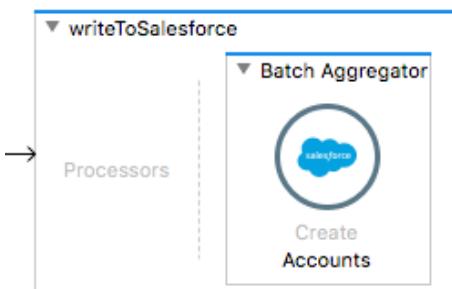
49. Select the Input tab in the DataSense Explorer; you should see this operation expects an Array of Account objects – not just one.



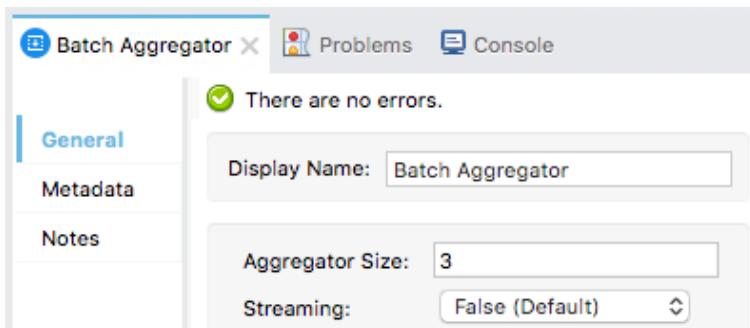
Create the Array of objects that the operation is expecting

50. Drag a Batch Aggregator scope from the Mule Palette and drop it in the aggregator section of the second batch step.

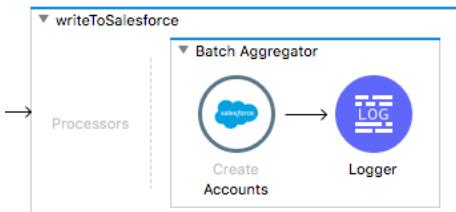
51. Move the Salesforce Create operation into the Batch Aggregator.



52. In the Batch Aggregator properties view, leave the aggregator size set to 3.



53. Add a Logger after the Create operation.



Test the application

54. Debug the project and clear the application data.

55. Step through the application until you step into the Batch Aggregator.

56. Expand Payload.

The screenshot shows the Mule Debugger and the application canvas. The debugger table shows a payload of three account records:

Name	Value
Message	
Payload (mimeType="*/")	size = 3 [redacted] [redacted] [redacted] size = 2
Variables	

The application canvas shows a flow starting with a 'writeToSalesforce' component, followed by a 'Batch Aggregator' component containing a 'Create Accounts' processor. An arrow points from the 'Batch Aggregator' to a 'Logger' component. The 'Logger' component is labeled 'Logger payload'.

57. Step through the rest of the flow.
58. Return to Salesforce and look at the accounts; you should see the records from the legacy MySQL database are now in Salesforce.

Note: You could also check for the records by making a request to <http://localhost:8081/sfdc>.

<input type="checkbox"/> Edit Del +	Express Logistics a...
<input type="checkbox"/> Edit Del +	GenePoint
<input type="checkbox"/> Edit Del +	Grand Hotels & Res...
<input type="checkbox"/> Edit Del +	Max Mule 94111
<input type="checkbox"/> Edit Del +	Max Muley
<input type="checkbox"/> Edit Del +	Maxwell Mule 94111
<input type="checkbox"/> Edit Del +	Mighty Mule 94111
<input type="checkbox"/> Edit Del +	Molly Mule 94111
<input type="checkbox"/> Edit Del +	Pyramid Constructi... 75251

59. Run the application again but do not clear the application data; no records should be processed.
60. Return to salesforce.com and locate your new record(s); they should have been inserted only once.
61. Return to Anypoint Studio and stop the project.