



MuleSoft®

Anypoint Platform Development: Fundamentals

Student Manual

Mule runtime 3.8
June 20, 2016

Table of Contents

INTRODUCING THE COURSE.....	5
Walkthrough: Set up your computer for class	6
PART 1: IMPLEMENTING API-LED CONNECTIVITY WITH ANYPOINT PLATFORM 14	
MODULE 1: INTRODUCING API-LED CONNECTIVITY	15
Walkthrough 1-1: Explore API directories and portals	16
Walkthrough 1-2: Make calls to an API.....	23
Walkthrough 1-3: Explore Anypoint Platform	31
MODULE 2: DESIGNING APIS	35
Walkthrough 2-1: Use API Designer to define an API with RAML	36
Walkthrough 2-2: Use the mocking service to test an API.....	41
Walkthrough 2-3: Add request and response details	45
Walkthrough 2-4: Create an API portal	51
Walkthrough 2-5: Add an API to the Anypoint Exchange	58
MODULE 3: BUILDING APIS	63
Walkthrough 3-1: Create a Mule application with Anypoint Studio	64
Walkthrough 3-2: Connect to data (MySQL database).....	70
Walkthrough 3-3: Transform data	80
Walkthrough 3-4: Create a RESTful interface for a Mule application	90
Walkthrough 3-5: Use Anypoint Studio to create a RESTful API interface from a RAML file	96
Walkthrough 3-6: Implement a RESTful web service	103
MODULE 4: DEPLOYING AND MANAGING APIS.....	109
Walkthrough 4-1: Prepare an API for deployment using properties	110
Walkthrough 4-2: Deploy an application to the cloud	115
Walkthrough 4-3: Create and deploy an API proxy	122
Walkthrough 4-4: Restrict API access	128
PART 2: BUILDING APPLICATIONS WITH ANYPOINT STUDIO	140
MODULE 5: ACCESSING AND MODIFYING MULE MESSAGES	141
Walkthrough 5-1: Set and log message data.....	142
Walkthrough 5-2: Debug a Mule application	146
Walkthrough 5-3: Read and write message properties using MEL expressions	152
Walkthrough 5-4: Read and write variables	155

MODULE 6: STRUCTURING MULE APPLICATIONS	158
Walkthrough 6-1: Create and reference flows and subflows	159
Walkthrough 6-2: Pass messages between flows using the Java Virtual Machine (VM) transport ..	163
Walkthrough 6-3: Encapsulate global elements in a separate configuration file	168
Walkthrough 6-4: Create a well organized Mule project	171
MODULE 7: CONSUMING WEB SERVICES.....	180
Walkthrough 7-1: Consume a RESTful web service.....	181
Walkthrough 7-2: Pass arguments to a RESTful web service	192
Walkthrough 7-3: Consume a RESTful web service that has a RAML definition	198
Walkthrough 7-4: Consume a SOAP web service	206
Walkthrough 7-5: Pass arguments to a SOAP web service using DataWeave	212
MODULE 8: HANDLING ERRORS	216
Walkthrough 8-1: Handle a messaging exception	217
Walkthrough 8-2: Handle different types of messaging exceptions	222
Walkthrough 8-3: Create and use global exception strategies	229
Walkthrough 8-4: Specify a global default exception strategy	233
Walkthrough 8-5: Review a mapping exception strategy.....	236
MODULE 9: CONTROLLING MESSAGE FLOW	239
Walkthrough 9-1: Route messages based on conditions	240
Walkthrough 9-2: Multicast a message.....	248
Walkthrough 9-3: Filter messages	266
Walkthrough 9-4: Validate messages	272
MODULE 10: WRITING DATAWEAVE TRANSFORMATIONS	276
Walkthrough 10-1: Write your first DataWeave transformation	277
Walkthrough 10-2: Transform basic Java, JSON, and XML data structures	286
Walkthrough 10-3: Transform complex data structures with arrays	291
Walkthrough 10-4: Transform to and from XML with repeated elements	297
Walkthrough 10-5: Coerce and format strings, numbers, and dates	303
Walkthrough 10-6: Use DataWeave operators	311
Walkthrough 10-7: Define and use custom data types	315
Walkthrough 10-8: Call MEL functions and other flows	319

MODULE 11: CONNECTING TO ADDITIONAL RESOURCES.....	326
Walkthrough 11-1: Connect to a SaaS application (Salesforce).....	327
Walkthrough 11-2: Connect to a file (CSV).....	336
Walkthrough 11-3: Poll a resource	345
Walkthrough 11-4: Connect to a JMS queue (ActiveMQ).....	354
Walkthrough 11-5: Find and install not-in-the-box connectors	361
MODULE 12: PROCESSING RECORDS.....	366
Walkthrough 12-1: Process items in a collection individually	367
Walkthrough 12-2: Create a batch job for records in a file	370
Walkthrough 12-3: Create a batch job to synchronize records from a database to Salesforce	375

Introducing the Course

The screenshot shows a course page on the MuleSoft Learning Platform. At the top, there's a navigation bar with links for 'My Learning' (which is highlighted in red), 'Classes', 'Catalog', and 'Support'. On the right side of the header are 'Cart' and 'Inbox' icons. Below the header, a green button with a checkmark and the word 'Confirmed' is visible. The main content area has a blue background. It features a 'Virtual Class' icon, the course title 'Anypoint Platform Development: Fundamentals' in bold blue text, and the date 'June 15, 2016 from 8:00AM to 3:00PM PDT'. A 'Back to My Learning' link is at the bottom left.

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, you use Anypoint Platform to take an API through its complete lifecycle: design, build, deploy, manage, and govern. In the second larger part, you focus on using Mule and Anypoint Studio to build applications for use as API implementations and/or integrations. It includes a voucher code to take the *MuleSoft Certified Developer – Integration and API Associate* exam.

FACILITIES

- ✓ Courseware

INSTRUCTORS



MATERIALS

- APDevFundamentals3.8 Student Files (ZIP) 110.95MB ZIP
- APDevFundamentals3.8 Student Manual (PDF) 64.66MB PDF
- APDevFundamentals3.8 Student Slides (ZIP) 24.9MB ZIP

Objectives:

- Learn about 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 Postman (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, and Company. A search icon is also present. Below the bar, a banner for 'Training & Certification' is visible, featuring a close-up photo of a person's eyes. The 'My training account' link is highlighted in blue at the top right of the banner.

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 box 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.

The screenshot shows a course details page on the MuleSoft Learning platform. At the top, there is a navigation bar with links for 'My Learning', 'Classes', 'Catalog', and 'Support'. On the right side of the header, there are links for 'Cart', 'Inbox', and a user profile. A green button labeled 'Confirmed' is visible in the top right corner. Below the header, there is a breadcrumb link '[Back to My Learning](#)'. The main content area features a large blue background with white text. It includes a 'Virtual Class' icon, the course title 'Anypoint Platform Development: Fundamentals' in bold, the date 'June 13, 2016 from 8:00AM to 3:00PM PDT', and a 'Confirmed' status indicator. The bottom section of the page contains course descriptions, instructor information, and a list of materials available for download.

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, you use Anypoint Platform to take an API through its complete lifecycle: design, build, deploy, manage, and govern. In the second larger part, you focus on using Mule and Anypoint Studio to build applications for use as API implementations and/or integrations. It includes a voucher code to take the [MuleSoft Certified Developer - Integration and API Associate](#) exam.

FACILITIES

✓ Courseware

INSTRUCTORS



MATERIALS

	APDevFundamentals3.8 Student Files (ZIP)
	APDevFundamentals3.8 Student Manual (PDF)
	APDevFundamentals3.8 Student Slides (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
```

14. 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.

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.

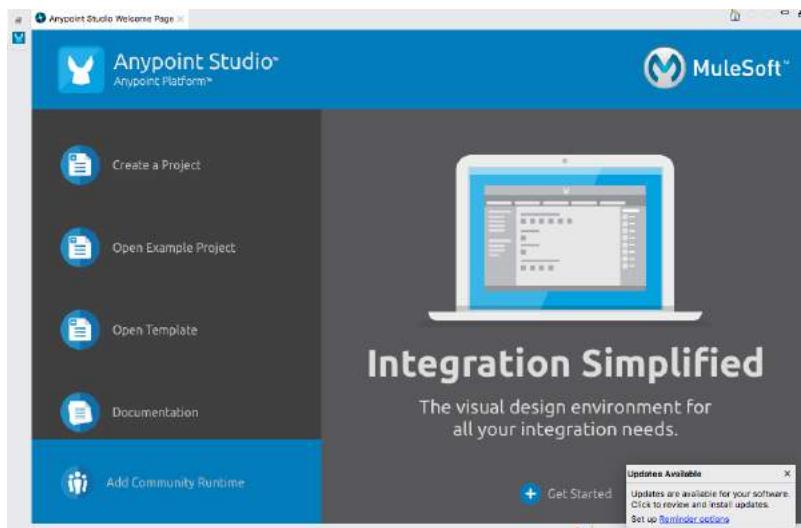
Note: If you do not have Anypoint Studio, you can download it from <https://www.mulesoft.com/lp/dl/studio>.

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.

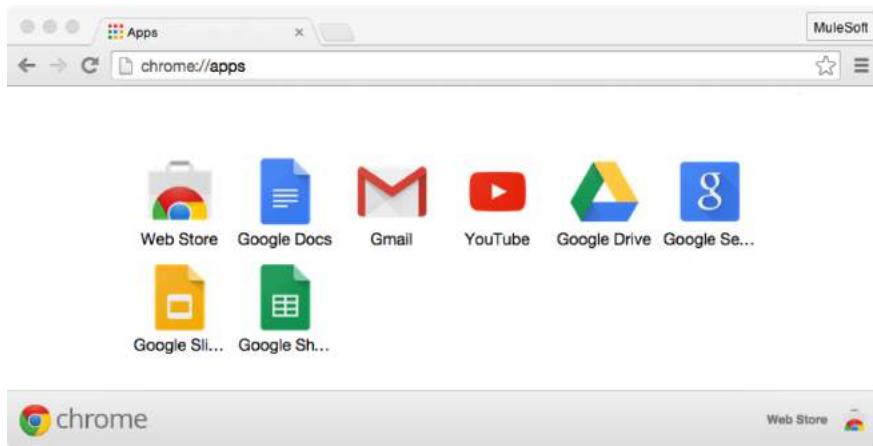


Install Postman (if you did not already)

22. Open the Google Chrome web browser.

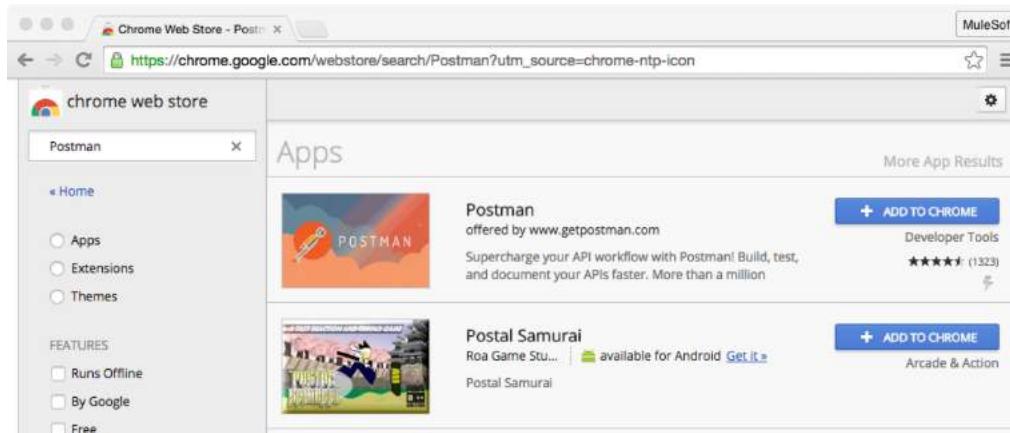
Note: If you do not have Chrome, download it from <http://chrome.google.com>.

23. Navigate to the URL <chrome://apps>; you should get a page showing icons for all installed Google Chrome apps.



24. Click the Web Store icon.

25. In the Chrome web store, select Apps and search for Postman; you should see a listing for Postman offered by www.getpostman.com.

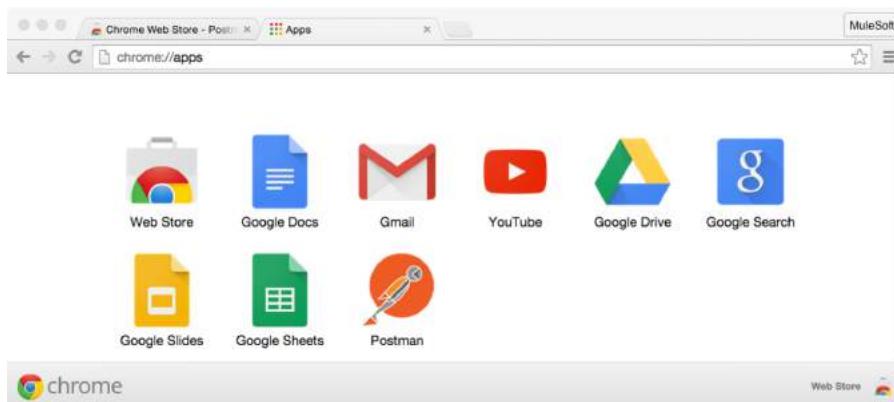


The screenshot shows the Chrome Web Store interface. The left sidebar has a search bar with "Postman" and filters for "Apps", "Extensions", and "Themes". Under "FEATURES", it shows "Runs Offline", "By Google", and "Free". The main area is titled "Apps" and lists the "Postman" app by "www.getpostman.com" with a rating of 4.5 stars (1323 reviews). Below it is the "Postal Samurai" game. A "More App Results" link is at the top right.

26. Click the ADD TO CHROME button for Postman.

27. In the Add Postman dialog box, click Add.

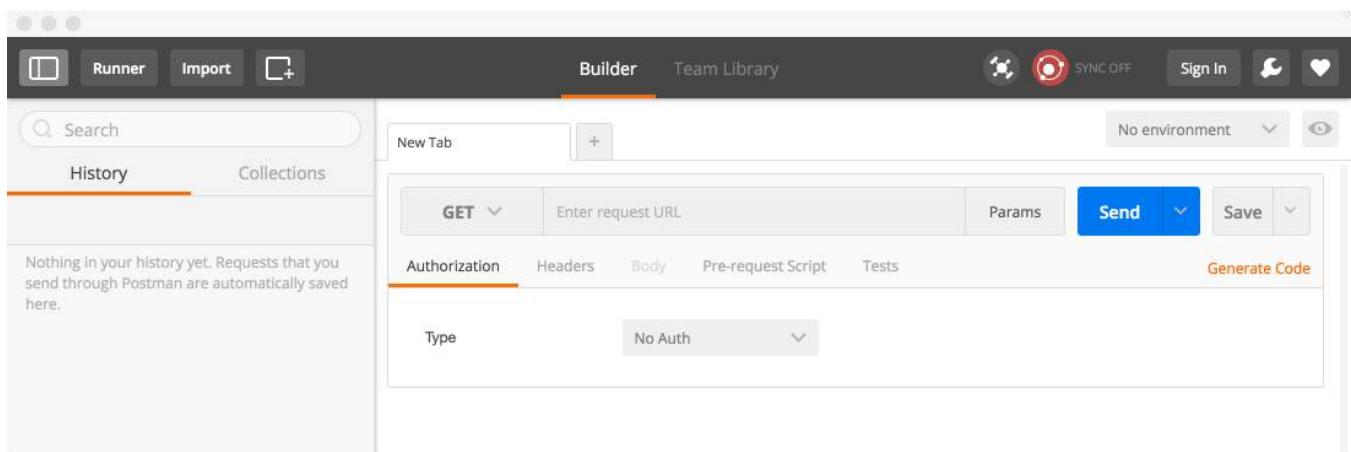
28. In Chrome, navigate to chrome://apps; you should see a Postman icon.



The screenshot shows the "chrome://apps" page. It displays various Google services as icons: Web Store, Google Docs, Gmail, YouTube, Google Drive, Google Search, Google Slides, Google Sheets, and Postman. The Postman icon is located in the bottom row.

29. Click the Postman icon; Postman should open.

30. Leave Postman open; you will use it throughout class.



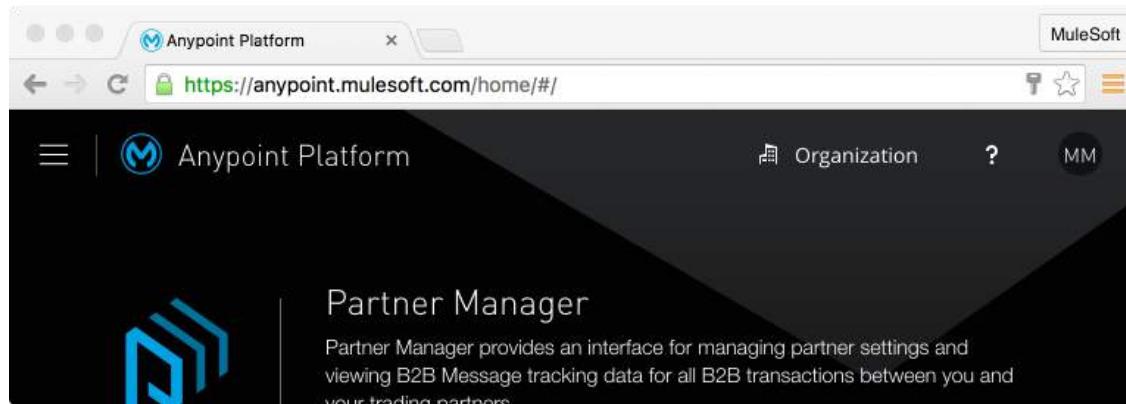
The screenshot shows the Postman application window. The top navigation bar includes "Runner", "Import", "Builder" (which is selected), "Team Library", "Sync Off", "Sign In", and various icons. The left sidebar has "History" (selected) and "Collections". The main workspace shows a "New Tab" with a "GET" dropdown, an "Enter request URL" field, "Params", "Send" (blue button), and "Save" buttons. Below this are tabs for "Authorization", "Headers", "Body", "Pre-request Script", "Tests", and "Generate Code". At the bottom, there's a "Type" dropdown set to "No Auth".

Make sure you have an active Anypoint Platform account

31. In a web browser, navigate to <http://anypoint.mulesoft.com>.

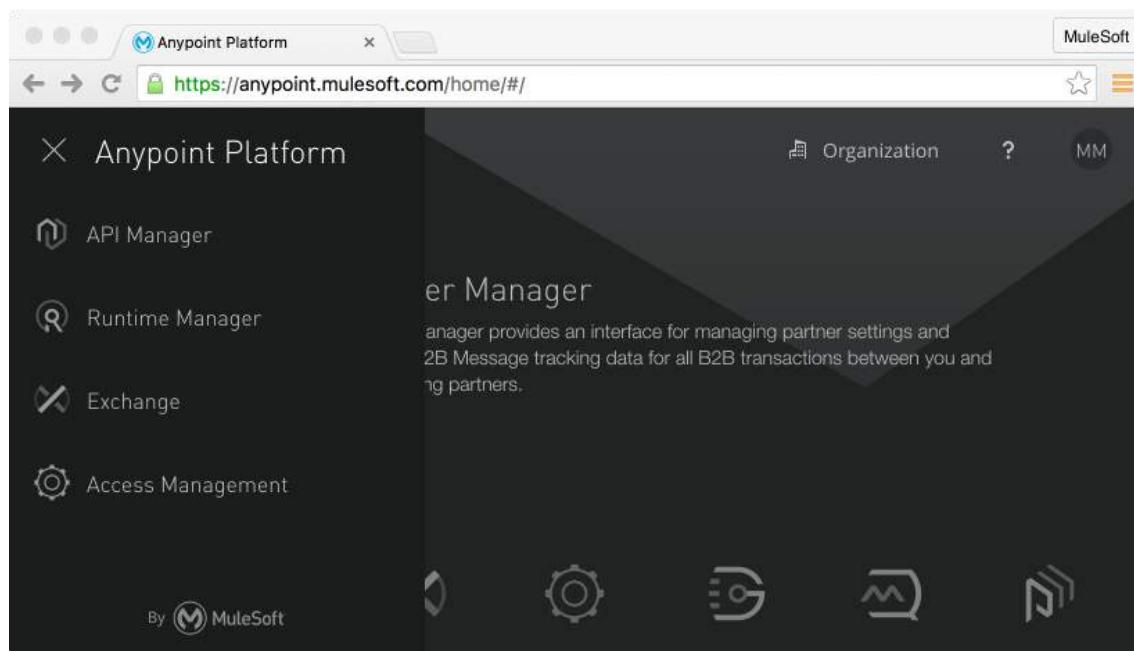
Note: You can use a trial account or your company account (if you already have one). If you do not have an account, sign up for a free, 30-day trial account now.

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



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

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



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

35. 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.

The screenshot shows the 'Subscription Level' section of the Anypoint Platform interface. On the left, a sidebar lists 'ACCESS MANAGEMENT' (Organization, Users, Roles, Environments, External Identity, Audit Logs), 'SETTINGS' (Runtime Manager, Exchange), and 'SUBSCRIPTION' (Runtime Manager). The 'Runtime Manager' option under 'SUBSCRIPTION' is currently selected. The main content area displays the 'Subscription Level' with a message: 'Trial expires on 06/02/2016'. Below this, sections for 'Production' (Environments for production applications) and 'Sandboxes' (Sandboxes are not available for trial accounts) are shown. A progress bar for 'vCores' is at 0 / 1. At the bottom, there is a 'Static IP' section.

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

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

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.

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

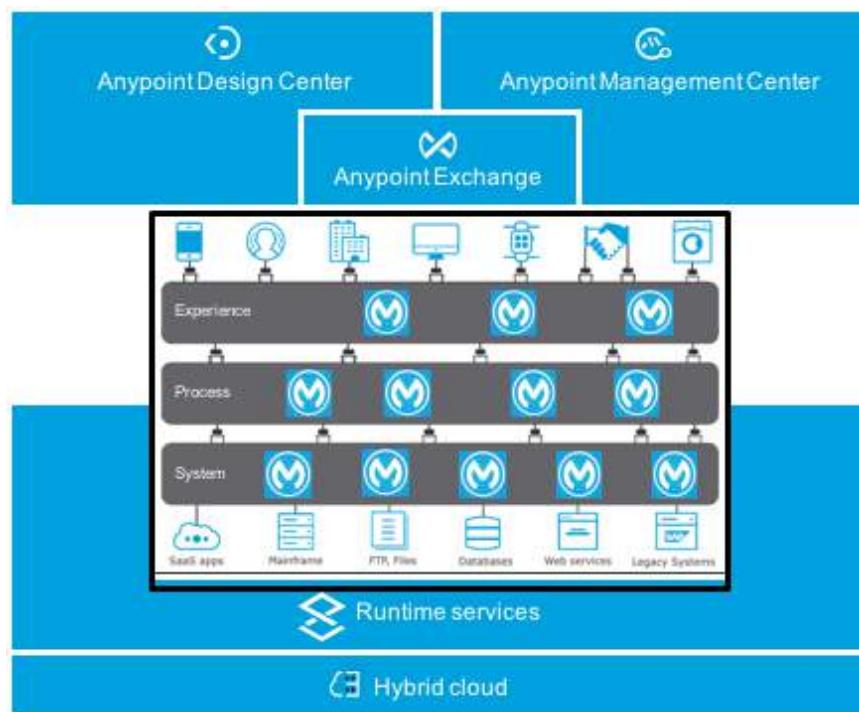


38. In the left-side navigation, select Personal > Reset My Security Token.
39. If you did not already request a security token, click the Reset Security Token button.

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.

The screenshot shows the 'Reset My Security Token' page within the Salesforce 'My Settings' section. The left sidebar lists options like Personal Information, Change My Password, Language & Time Zone, Grant Account Login Access, My Groups, Reset My Security Token (which is highlighted in blue), and Connections. The main content area has a heading 'Reset My Security Token' and a note: 'Clicking the button below invalidates your existing token. After resetting your token, you will have to use the new token in all API applications.' Below this is a warning message: 'Your security token is tied to your password and subject to any password policies your administrators have configured. Whenever your password is reset, your security token is also reset.' At the bottom is a blue button labeled 'Reset Security Token'. A 'Help for this Page' link is in the top right, and a 'Chat' icon is in the bottom right.

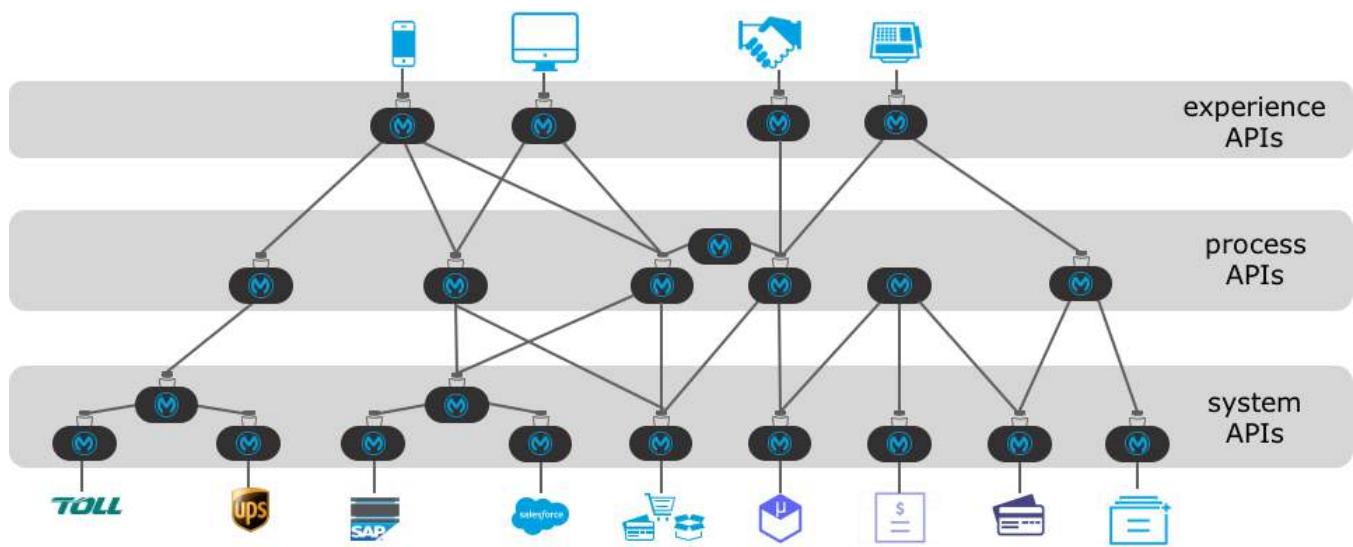
PART 1: Implementing API-Led Connectivity with Anypoint Platform



Objectives:

- Describe what API-led connectivity is and its benefits.
- Use Anypoint Platform to take an API through its complete lifecycle.
- Design, build, deploy, manage, and govern an API.

Module 1: Introducing API-Led Connectivity



Objectives:

- Identify the problems faced by IT today.
- Describe what API-led connectivity is and its benefits.
- Explain what web services and APIs are.
- Explore API directories and portals.
- Make calls to secure and unsecured APIs.
- Introduce API-led connectivity with Anypoint Platform.
- Explore Anypoint Platform.

Walkthrough 1-1: Explore API directories and portals

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

- Browse the ProgrammableWeb API directory.
- Explore the MuleSoft Developer portal for popular APIs.
- View an API definition file.
- Explore the MuleSoft Developer portal for Anypoint Platform.
- Use the API Console in an Anypoint Platform API Portal to make sample calls to an API.

The screenshot shows the MuleSoft API Platform 2.0.3 developer portal. The left sidebar has links for Overview, API reference (which is selected), Query API Platform applications, Query APIs by a tag name, Public search API, and Extended search API. The main content area displays several API endpoints with their methods and descriptions:

Endpoint	Method	Description
/organizations/{organizationId}/theme/default	GET	
/organizations/{organizationId}/portals/terms	DELETE PUT GET	
/organizations/{organizationId}/apis	Type: collection A collection of APIs.	POST GET
/organizations/{organizationId}/apis/by-name	GET	
/organizations/{organizationId}/apis/versions/by-endpoint	GET	
/organizations/{organizationId}/apis/versions/by-name	GET	
/organizations/{organizationId}/apis/versions/tiers	GET	

Explore 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 website. The top navigation bar includes links for ProgrammableWeb, API NEWS, API DIRECTORY, and a search bar. Below the navigation is a horizontal menu with links for API UNIVERSITY, RESEARCH, SPORTS, SECURITY, TRAVEL, and DESIGN. To the right of this menu is a button for ADD APIs & MORE, followed by social media icons for RSS, Facebook, Twitter, Google+, and LinkedIn. The main content area features a large image of a smartphone displaying a map with a location pin, and an advertisement for MuleSoft: "Build a better API strategy A 7 step blueprint for success".

3. Scroll down and click the link for the Twitter API.

Note: If Twitter is no longer displayed on the main page, search for it.

The screenshot shows a web browser window with the title 'API Directory | ProgrammableWeb'. The URL in the address bar is 'www.programmableweb.com/apis/directory'. Below the address bar, there are tabs for 'API NEWS', 'API DIRECTORY', 'API UNIVERSITY', and a blue 'ADD API' button. A search bar is on the right. The main content area displays a list of APIs. The first item is 'Twitter', which is described as '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...'. To the right of the description are the categories 'Social', a rating of '12.08.20', and a small profile icon.

4. In the Specs section, click the API Homepage link.

SPECS

API Provider

<http://twitter.com>

API Endpoint

<http://twitter.com/statuses/>

API Homepage

<https://dev.twitter.com/rest/public>

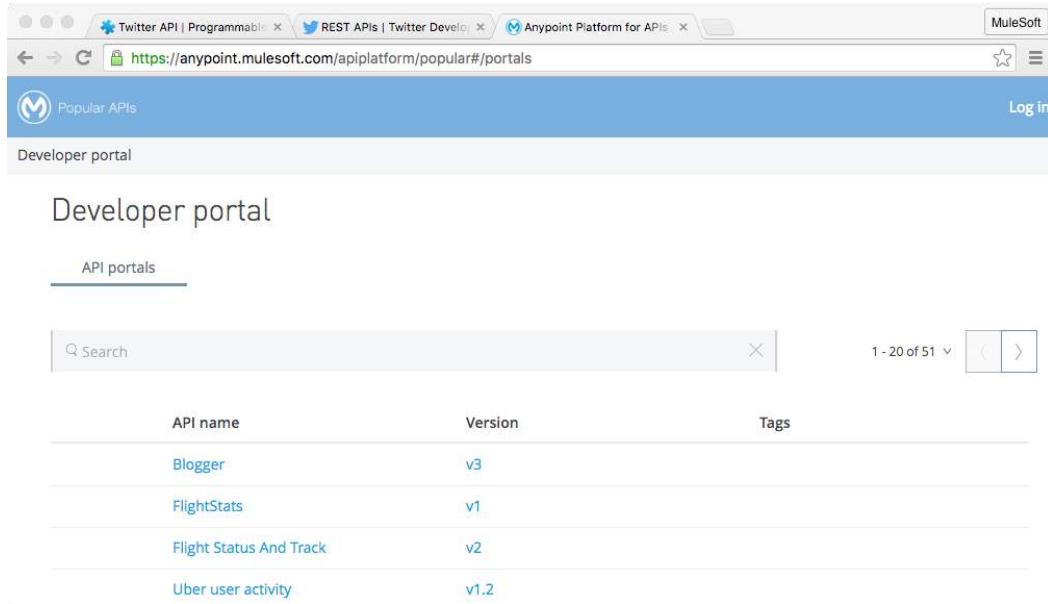
Primary Category

5. Browse the list of requests you can make to the API.

The screenshot shows a web browser window with the title 'Twitter API | ProgrammableWeb' and the URL 'https://dev.twitter.com/rest/public'. The page has a blue header with the Twitter logo and navigation links for 'Developers / Documentation / REST APIs'. On the left, there's a sidebar titled 'API Console Tool' with sections for 'Public API' and a list of endpoints like 'The Search API', 'Working with Timelines', etc. The main content area is titled 'REST APIs' and contains text about the REST API providing programmatic access to Twitter data. It also mentions OAuth and JSON responses. Below this is an 'Overview' section with a list of topics including 'API Rate Limiting', 'API Rate Limits', 'Working with Timelines', and 'Using the Twitter Search API'.

Explore the MuleSoft Developer portal for popular APIs

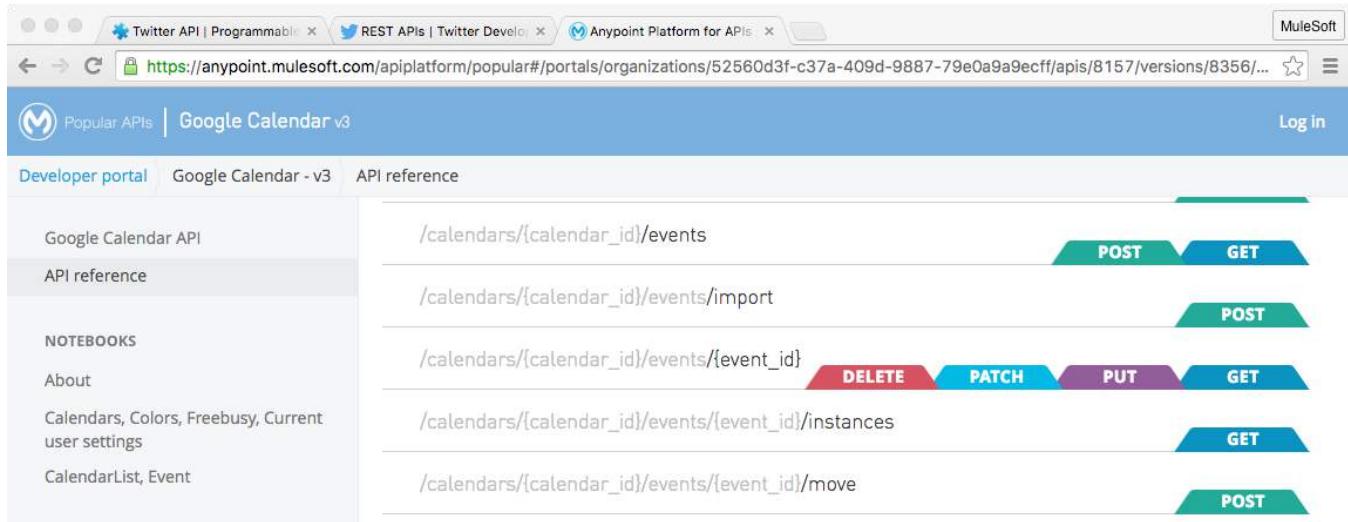
6. Return to the course snippets.txt file.
7. Copy the URL for the MuleSoft developer portal for popular APIs:
<https://anypoint.mulesoft.com/apiplatform/popular#/portals>.
8. In a web browser, navigate to that URL.



The screenshot shows a web browser window with the following details:

- Address bar: https://anypoint.mulesoft.com/apiplatform/popular#/portals
- Title bar: Twitter API | Programmable, REST APIs | Twitter Developers, Anypoint Platform for APIs, MuleSoft
- Header: Popular APIs, Log in
- Page title: Developer portal
- Section title: API portals
- Search bar: Search (with X button) and pagination: 1 - 20 of 51, ()
- Table: API name, Version, Tags
- Data rows:
 - Blogger, v3
 - FlightStats, v1
 - Flight Status And Track, v2
 - Uber user activity, v1.2

9. Scroll down and locate and click the Google Calendar link.
10. In the left-side navigation, click the API reference link.
11. Scroll the list of available resources.
12. Click the GET tab for the /calendars/{calendar_id}/events/{event_id} resource.



The screenshot shows a web browser window with the following details:

- Address bar: https://anypoint.mulesoft.com/apiplatform/popular#/portals/organizations/52560d3f-c37a-409d-9887-79e0a9a9ecff/apis/8157/versions/8356...
- Title bar: Twitter API | Programmable, REST APIs | Twitter Developers, Anypoint Platform for APIs, MuleSoft
- Header: Popular APIs | Google Calendar v3, Log in
- Page title: Developer portal | Google Calendar - v3 | API reference
- Left sidebar:
 - Google Calendar API
 - API reference (selected)
 - NOTEBOOKS
 - About
 - Calendars, Colors, Freebusy, Current user settings
 - CalendarList, Event
- Resource list:
 - /calendars/{calendar_id}/events
 - POST
 - GET
 - /calendars/{calendar_id}/events/import
 - POST
 - /calendars/{calendar_id}/events/{event_id}
 - DELETE
 - PATCH
 - PUT
 - GET
 - /calendars/{calendar_id}/events/{event_id}/instances
 - GET
 - /calendars/{calendar_id}/events/{event_id}/move
 - POST

13. Scroll down and look at the sample response.

The screenshot shows a browser window with the URL <https://anypoint.mulesoft.com/apiplatform/popular#/portals/organizations/52560d3f-c37a-409d-9887-79e0a9a9ecff/apis/>. The page title is "Popular APIs | Google Calendar v3". On the left, there's a sidebar with "Google Calendar API" and "API reference" sections, and a "NOTEBOOKS" section containing "About", "Calendars, Colors, Freebusy, Current user settings", and "CalendarList, Event". The main content area is titled "Response" and shows a "STATUS 200" response with a green "200" badge. It indicates the "Body" is in "application/json" format and provides an "Example" button. The example JSON is displayed in a code block:

```
{  
  "kind": "calendar#event",  
  "etag": "etag",  
  "id": "string",  
  "status": "string",  
  "htmlLink": "string",  
  "created": "datetime",  
  "updated": "datetime",  
  "summary": "string",  
  "description": "string",  
  "location": "string",  
}
```

14. Click the Try it button.

15. Scroll down and locate and click the GET button; you should get errors that client id, client secret, event id, and version are required.

The screenshot shows the same browser window and URL as the previous screenshot. The main content area is titled "Security Scheme" and displays the message "Custom Security Schemes are not supported in Try It". Below this, there are dropdown menus for "OAuth 2.0" and "Authorization Grant", both of which are currently empty. There are two red-highlighted input fields labeled "Client ID *" and "Client Secret *", both with the word "Required" below them. Below these fields is a "Scopes" section with a checkbox for "https://www.googleapis.com/auth/calendar". At the bottom of the form is a "URI PARAMETERS" section. A large red "Required" label is also present above the "Client ID" field.

16. Scroll up and click the Close button in the upper-right corner.

17. Click the DELETE tab for the /calendars/{calendar_id}/events/{event_id} resource.

18. Scroll down and look at the sample response.

The screenshot shows a web browser with three tabs open: 'Twitter API | Programmable...', 'REST APIs | Twitter Dev...', and 'Anypoint Platform for APIs'. The main content area is for the 'Google Calendar v3' API. On the left, there's a sidebar with links like 'Google Calendar API', 'API reference' (which is selected), 'NOTEBOOKS', 'About', and 'Calendars, Colors, Freebusy, Current user settings'. The right side has a 'Response' section showing a green box for '200' with the text: 'STATUS 200' and 'If successful, this method returns an empty response body.'

19. Scroll up and click the PUT tab for the /calendars/{calendar_id}/events/{event_id} resource.

20. Scroll down and look at the example body.

21. Scroll down and look at the sample response.

22. Scroll down to the bottom of the API reference page.

23. Click the Download API definition as a .zip file link.

The screenshot shows the same API documentation interface. The 'API reference' tab is selected. On the right, there are two main sections: one for the endpoint '/users/me/calendarList/{calendar_id}' (with methods POST, GET, DELETE, PATCH, PUT) and another for '/users/me/calendarList/watch' (with method POST). Below these, there's a 'ROOT RAML URL' field containing the URL 'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/52560d...' and a 'Download API definition as a .zip file' button.

24. In your computer's file browser, navigate to your downloads folder and unzip the API ZIP.

25. Open api.raml in a text editor.

26. Take a quick look at the API definition file and then close it.

```
◀ | ▶ | □ api.raml ▲ |  
  
/{event_id}:  
  get:  
    is: [ calendarEvent ]  
    description: Returns an event.  
    queryParameters:  
      timeZone:  
        description: Time zone used in the response. Optional. The default is the  
        example: UTC  
    responses:  
      200:  
        body:  
          application/json:  
            schema: eventResourceResponse  
            example: !include examples/eventResourceResponse-example.json  
      put:  
        is: [ calendarEvent , sendNotifications ]  
        description: Updates an event.  
        body:  
          application/json:  
            schema: createEventResourceRequest
```

Explore the MuleSoft Developer portal for Anypoint Platform

27. Return to the course snippets.txt file and copy the URL for the MuleSoft developer portal for Anypoint Platform: <https://anypoint.mulesoft.com/apiplatform/anypoint-platform/#/portals>.

28. In a web browser, navigate to that URL.

A screenshot of a web browser window showing the MuleSoft Developer portal. The address bar contains the URL "https://anypoint.mulesoft.com/apiplatform/anypoint-platform/#/portals". The page has a dark header with the MuleSoft logo and a "Log in" button. Below the header, there's a navigation bar with "Developer portal" and "API portals" tabs, where "API portals" is underlined. A search bar with placeholder text "Q Search" is followed by a pagination indicator "1 - 20 of 25" and navigation arrows. A table lists three API entries: "Industry Solution | FHIR Schedule API" (version v1, tags: healthcare, fhir, acceleration template), "Automation - Execution API" (version 0.1), and "Industry Solution | Salesforce Health Cloud Custom API" (version v1, tags: healthcare, fhir, acceleration template).

29. Scroll down and locate and click the API Platform link.

30. In the left-side navigation, click the API reference link.
31. Scroll the list of available resources.
32. Click the GET tab for the /public/apis resource.

The screenshot shows the MuleSoft API Platform 2.0.3 interface. The left sidebar has 'API reference' selected. The main area is titled 'API reference' and shows the 'Resources' section. Under 'Resources', there is a list item for '/public'. Expanding '/public' shows two entries: '/public/apis' and '/organizations'. The '/public/apis' entry has a 'Type: apiSearch' button and a 'GET' button. Below it is a search placeholder: 'Search through all public APIs within the Anypoint Platform.' The '/organizations' entry has a descriptive text: 'Organizations are the container entity for all API Portal-related resources.'

33. Click the Try it button.
34. Scroll down and locate and click the GET button.
35. Scroll down and look at the response body.

The screenshot shows the same MuleSoft API Platform interface as before, but now the response body for the '/public/apis' endpoint is displayed. The response is a JSON object with the following structure:

```

1 [
2   "total": 5940,
3   "apis": [
4     {
5       "audit": {
6         "created": {
7           "date": "2016-05-11T09:05:07.422Z"
8         },
9         "updated": {}
10       },
11       "masterOrganizationId": "4a2bd049-b822-4ad0-9023-877ebcb91b4b",
12       "organizationId": "4a2bd049-b822-4ad0-9023-877ebcb91b4b",
13       "id": 67323,
14       "name": "1.1Development",
15       "versions": [
16         {
17           "audit": {
18             "created": {
19               "date": "2016-05-11T09:05:07.422Z"
20             },
21             "updated": {
22               "date": "2016-05-11T09:07:48.258Z"
23             }
24         }
25       ]
26     }
27   ]
28 ]

```

Walkthrough 1-2: Make calls to an API

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

- Use Postman to make calls to an unsecured API.
- Make GET, DELETE, POST, and PUT calls.
- Use Postman to make calls to a secured API.

The screenshot shows the Postman interface with a PUT request to `http://apdev-american-api.cloudhub.io/flights/3`. The Body tab is selected, showing a JSON payload:

```
1 [ {  
2   "code": "GQ574",  
3   "price": 399,  
4   "departureDate": "2016/12/20",  
5   "origin": "ORD",  
6   "destination": "SFO",  
7   "emptySeats": 200  
8 } ]
```

The response status is 200 OK with a message: "Flight updated (but not really)".

Make GET requests to retrieve data

1. Return to or open Postman.
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://apdev-american-ws.cloudhub.io/api/flights>.
5. Return to Postman and paste the URL in the text box that says Enter request URL.

The screenshot shows the Postman interface with a GET request to `http://apdev-american-ws.cloudhub.io/api/flights`. The Authorization tab is selected, showing "No Auth".

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

The screenshot shows the Postman interface with a successful API call. The URL is `http://apdev-american-ws.cloudhub.io/api/flights`. The response status is 200, and the time taken is 1070 ms. The response body is displayed in JSON format:

```

1. [
2.   {
3.     "ID": 1,
4.     "code": "rree0001",
5.     "price": 541,
6.     "departureDate": "2016-01-20T00:00:00",
7.     "origin": "MUA",
8.     "destination": "LAX",
9.     "emptySeats": 0,
10.    "plane": {
11.      "type": "Boeing 787",
12.      "totalSeats": 200
13.    }
14.  },
15.  {
16.    "ID": 2,
17.    "code": "eefd0123",
18.    "price": 300.
  
```

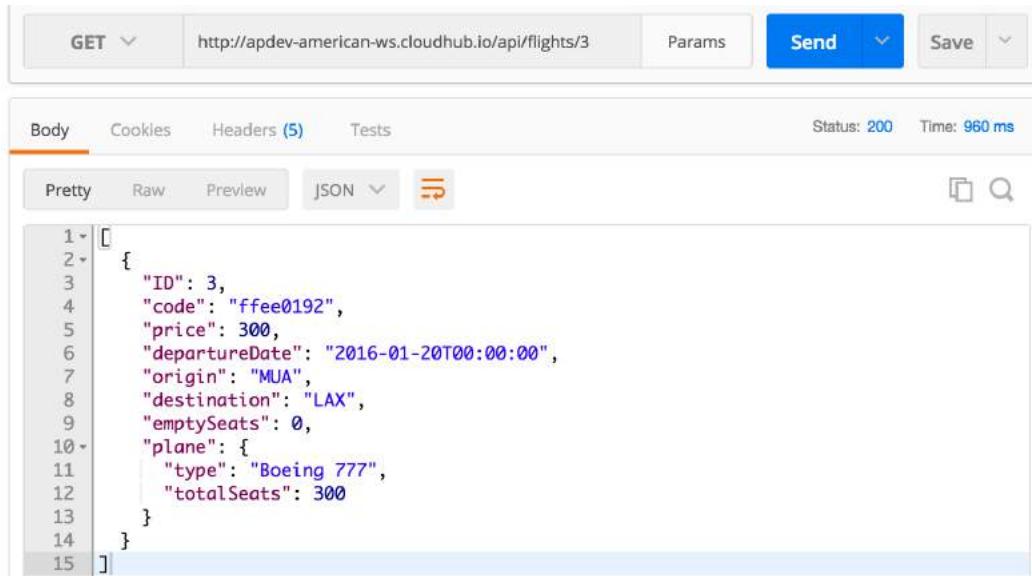
- Click the Params button next to the URL.
- In the area that appears, set the key to code and the value to CLE.
- Click the Send button; you should get just flights to CLE returned.

The screenshot shows the Postman interface with a filtered API call. The URL is `http://apdev-american-ws.cloudhub.io/api/flights?code=CLE`. The response status is 200, and the time taken is 1062 ms. The response body is displayed in JSON format:

```

1. [
2.   {
3.     "ID": 2,
4.     "code": "eefd0123",
5.     "price": 300,
6.     "departureDate": "2016-01-25T00:00:00",
7.     "origin": "MUA",
8.     "destination": "CLE",
9.     "emptySeats": 7.
  
```

12. Click the X next to the parameter to delete it.
13. Change the request URL to use a uri parameter to retrieve the flight with an ID of 3:
<http://apdev-american-ws.cloudhub.io/api/flights/3>
14. Click the Send button; you should see only the flight with that ID returned.



The screenshot shows a REST client interface. At the top, there's a header bar with 'GET' selected, the URL 'http://apdev-american-ws.cloudhub.io/api/flights/3', and buttons for 'Params', 'Send', and 'Save'. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Tests'. The 'Body' tab is active, showing a JSON response. The JSON is pretty-printed and looks like this:

```

1 [
2   {
3     "ID": 3,
4     "code": "ffee0192",
5     "price": 300,
6     "departureDate": "2016-01-20T00:00:00",
7     "origin": "MUA",
8     "destination": "LAX",
9     "emptySeats": 0,
10    "plane": {
11      "type": "Boeing 777",
12      "totalSeats": 300
13    }
14  }
15 ]

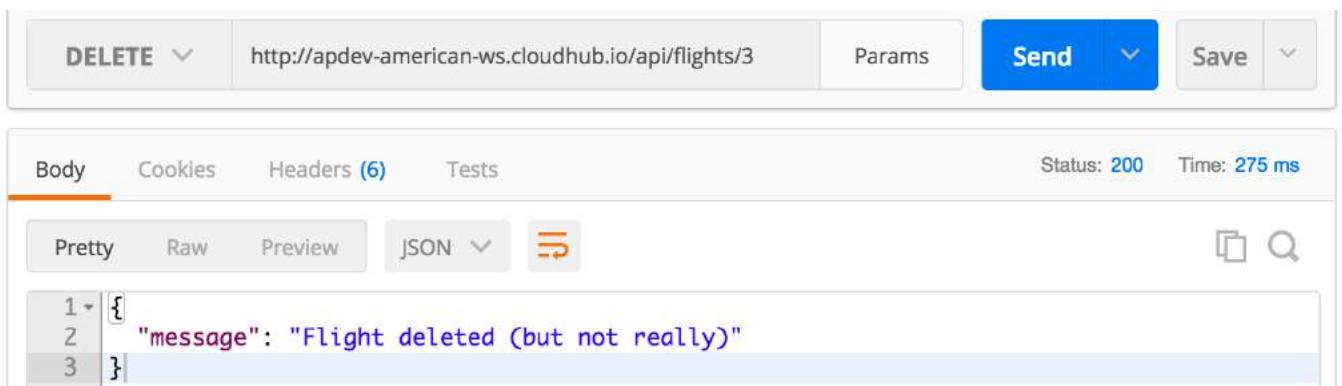
```

The status bar at the bottom right indicates 'Status: 200' and 'Time: 960 ms'.

Make DELETE requests to delete data

15. Change the method to DELETE.
16. Click the Send button; you should see a 200 response with a message that the was Flight deleted (but not really).

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



The screenshot shows a REST client interface. At the top, there's a header bar with 'DELETE' selected, the URL 'http://apdev-american-ws.cloudhub.io/api/flights/3', and buttons for 'Params', 'Send', and 'Save'. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Tests'. The 'Body' tab is active, showing a JSON response. The JSON is pretty-printed and looks like this:

```

1 {
2   "message": "Flight deleted (but not really)"
3 }

```

The status bar at the bottom right indicates 'Status: 200' and 'Time: 275 ms'.

17. Remove the URI parameter from the request: <http://apdev-american-ws.cloudhub.io/api/flights>.

18. Click the Send button; you should get a 405 response with a message of method not allowed.

The screenshot shows the Postman interface. At the top, there is a header bar with a 'DELETE' dropdown, the URL 'http://apdev-american-ws.cloudhub.io/api/flights', a 'Params' button, a 'Send' button, and a 'Save' button. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Tests'. The 'Body' tab is selected. On the right side of the body panel, it says 'Status: 405' and 'Time: 157 ms'. The body content is displayed in JSON format:

```
1 [{}  
2 "message": "Method not allowed"  
3 ]
```

Make a POST request to add data

19. Change the method to POST.

20. Click the Send button; you should get a 415 response with a message of unsupported media type.

Note: With the initial release of APIkit in Mule 3.8.0, you may get a different message of flow not found for resource.

The screenshot shows the Postman interface. At the top, there is a header bar with a 'POST' dropdown, the URL 'http://apdev-american-ws.cloudhub.io/api/flights', a 'Params' button, a 'Send' button, and a 'Save' button. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Tests'. The 'Body' tab is selected. On the right side of the body panel, it says 'Status: 415 OK' and 'Time: 158 ms'. The body content is displayed in JSON format:

```
1 [{}  
2 "message": "Unsupported media type"  
3 ]
```

21. Click the Headers link under the request URL.

22. Click in the Headers key field, type C, and then select Content-Type.

23. Click in the Value field, type A, and then select application/json.

The screenshot shows the Postman interface. At the top, there is a header bar with a 'POST' dropdown, the URL 'http://apdev-american-ws.cloudhub.io/api/flights', a 'Params' button, a 'Send' button, and a 'Save' button. Below the header, there are tabs for 'Authorization', 'Headers (1)', 'Body', 'Pre-request Script', and 'Tests'. The 'Headers (1)' tab is selected. On the right side of the headers panel, there is a 'Generate Code' button. Under the 'Headers (1)' tab, there is a table with one row. The first column is 'key' and the second column is 'value'. The 'key' column contains 'Content-Type' with a checked checkbox, and the 'value' column contains 'application/json'. There are also 'Bulk Edit' and 'Presets' buttons at the bottom right of the headers panel.

24. Click the Body link under the request URL.

25. Select the raw radio button.
26. Return to the course snippets.txt file and copy the value for American Flights API post body.
27. Return to Postman and paste the code in the body text area.



The screenshot shows the Postman interface with the 'Body' tab selected. The 'raw' radio button is checked. The request body is set to JSON (application/json). The JSON payload is:

```

1 {
2   "code": "GQ574",
3   "price": 399,
4   "departureDate": "2016/12/20",
5   "origin": "ORD",
6   "destination": "SFO",
7   "emptySeats": 200,
8   "plane": {"type": "Boeing 747", "totalSeats": 400}
9 }

```

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



The screenshot shows the Postman interface after sending the request. The status is 201 and the time is 208 ms. The response body is:

```

1 {
2   "message": "Flight added (but not really)"
3 }

```

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



The screenshot shows the Postman interface with the 'Body' tab selected. The 'raw' radio button is checked. The request body is set to JSON (application/json). The JSON payload is:

```

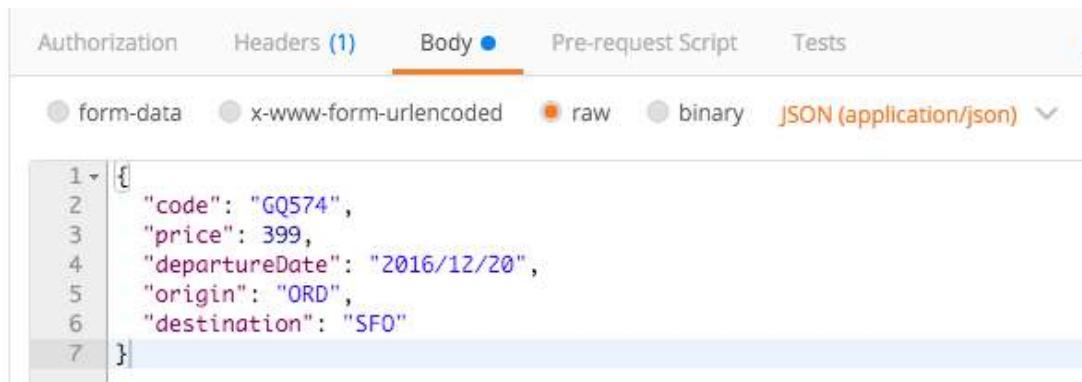
1 {
2   "code": "GQ574",
3   "price": 399,
4   "departureDate": "2016/12/20",
5   "origin": "ORD",
6   "destination": "SFO",
7   "emptySeats": 200
8 }

```

31. Send the request; the message should still post successfully.

32. In the request body, remove the emptySeats key/value pair.

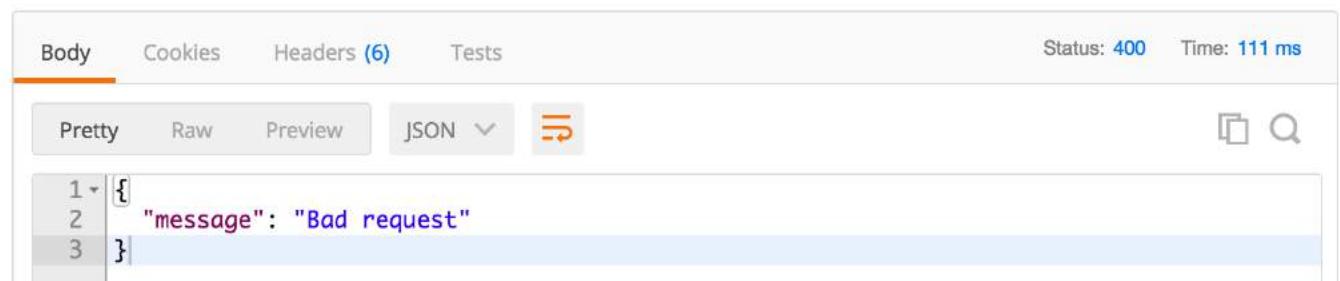
33. Delete the comma after the destination key/value pair.



The screenshot shows the Postman interface with the 'Body' tab selected. The 'JSON (application/json)' option is chosen. The request body contains the following JSON:

```
1 {  
2   "code": "GQ574",  
3   "price": 399,  
4   "departureDate": "2016/12/20",  
5   "origin": "ORD",  
6   "destination": "SFO"  
7 }
```

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



The screenshot shows the Postman interface with the 'Body' tab selected. The status is 400 and the time is 111 ms. The response body is JSON:

```
1 {  
2   "message": "Bad request"  
3 }
```

Make a PUT request to update data

35. Change the method to PUT.

36. Add a flight ID to the URL to modify a particular flight.

37. Click the Send button; you should get a bad request message.

38. In the request body field, press Cmd+Z or Ctrl+Z so the emptySeats field is added back.

39. Send the request; you should see the response Flight updated (but not really).

The screenshot shows the Postman interface. At the top, a 'PUT' method is selected, and the URL is set to <http://apdev-american-ws.cloudhub.io/api/flights/3>. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "code": "GQ574",  
3   "price": 399,  
4   "departureDate": "2016/12/20",  
5   "origin": "ORD",  
6   "destination": "SFO",  
7   "emptySeats": 200  
8 }
```

Below the body, the response is shown. The status is 200, and the time taken is 116 ms. The 'Body' tab is active, displaying the message:

```
1 {  
2   "message": "Flight updated (but not really)"  
3 }
```

Make a request to a secured API

40. Change the method to GET.

41. Change the request URL to <http://apdev-american-api.cloudhub.io/flights/3>.

42. Click the Send button; you should get a message about a missing client_id.

The screenshot shows the Postman interface. The method is changed to 'GET', and the URL is set to <http://apdev-american-api.cloudhub.io/flights/3>. The 'Body' tab is active, showing an HTML payload:

```
i 1 Unable to retrieve client_id from message
```

Below the body, the response is shown. The status is 401, and the time taken is 271 ms. The 'Body' tab is active, displaying the message:

```
i 1 Unable to retrieve client_id from message
```

43. Return to the course snippets.txt file and copy the value for the American Flights API client_id.

44. Return to Postman and add a request parameter called client_id.

45. Set client_id to the value you copied from the snippets.txt file.

46. Return to the course snippets.txt file and copy the value for the American Flights API client_secret.
47. Return to Postman and add a request parameter called client_secret.
48. Set client_secret to the value you copied from the snippets.txt file.
49. Click the Send button; you should get data for flight 3 again – or a message that API calls have been exceeded).

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

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://apdev-american-api.cloudhub.io/flights/3?client_id=d1374b15c686
- Params:** client_id: d1374b15c6864c3682ddbed2a247a826, client_secret: 4a87fe7e2e43488c927372AEF981F066
- Authorization:** No Auth
- Body:** JSON response (Pretty, Raw, Preview) showing flight details for flight ID 3.
- Response Headers:** Status: 200, Time: 1228 ms

```

1 [
2   {
3     "ID": 3,
4     "code": "ffee0192",
5     "price": 300,
6     "departureDate": "2016-01-20T00:00:00",
7     "origin": "MUA",
8     "destination": "LAX",
9     "emptySeats": 0,
10    "plane": {
11      "type": "Boeing 777",
12      "totalSeats": 300
13    }
14  }
15 ]
  
```

Walkthrough 1-3: Explore Anypoint Platform

In this walkthrough, you get familiar with the Anypoint Platform web application. You will:

- Explore Anypoint Platform.
- Add an API to Anypoint Platform.

The screenshot shows the Anypoint Platform API Manager interface. The URL in the browser is <https://anypoint.mulesoft.com/apiplatform/organization-5/admin/#/organizations/095b2011-d592-498e-9251-38442e7f6414/dash...>. The main panel displays the "American Flights API" version 1.0, which is described as "API for american flights table in training database". It shows that the API is owned by "Max Mule". The "Applications" tab is selected, indicating there are no applications for this API version.

Return to Anypoint Platform

1. Return to Anypoint Platform 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, click Anypoint Platform; this will return you to the home page.

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

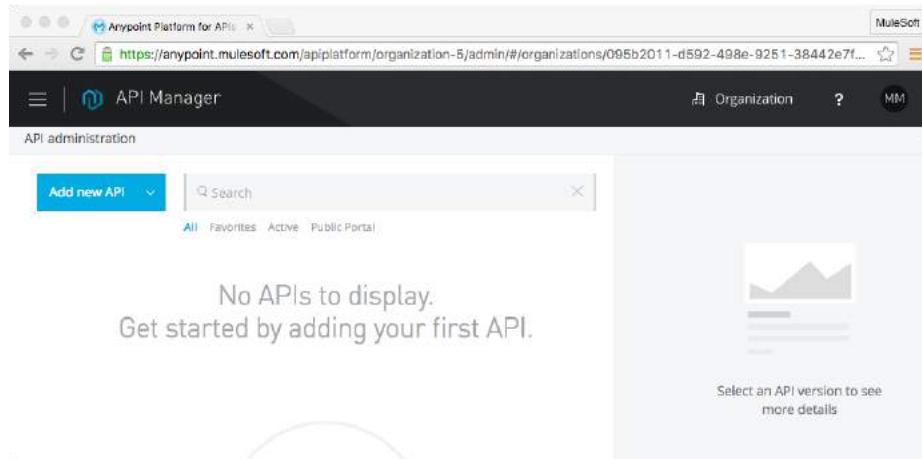
The screenshot shows the Anypoint Platform main menu. The URL in the browser is <https://anypoint.mulesoft.com/home/#/>. The left sidebar lists several options: "Anypoint Platform" (selected), "API Manager", "Runtime Manager", "Exchange", and "Access Management". The right panel has a dark background with a large, semi-transparent triangular graphic pointing towards the center. At the top right of the main area are buttons for "Organization", "?", and "MM". Below the graphic, the word "change" is visible, followed by a description of the Exchange feature: "Anypoint Exchange is a discovery interface where you can find Anypoint actors, Templates, examples, and API descriptions that speed project delivery." A blue "Open" button is highlighted at the bottom of this section. The footer of the page includes the MuleSoft logo and icons for "API Manager", "Runtime Manager", "Exchange", and "Access Management".

Explore Anypoint Platform

4. In the main menu, select Access Management.
5. In the main menu, select API Manager.
6. In the main menu, select Exchange.
7. In the main menu, select Runtime Manager.

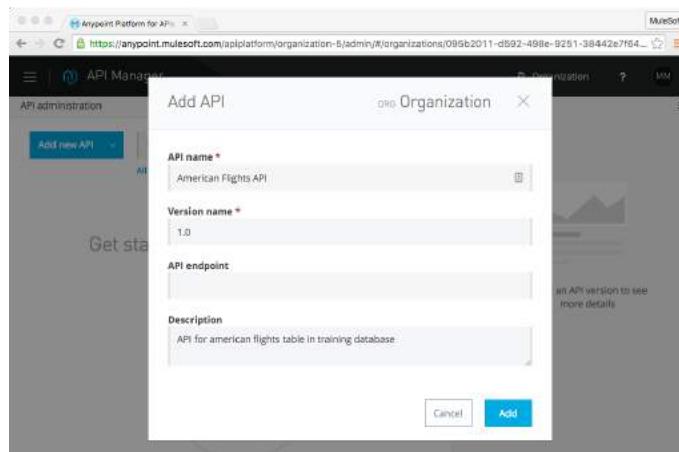
Add an API to Anypoint Platform

8. In the main menu, select API Manager again.
9. Click the Add new API button.



10. In the Add API dialog box, enter the following information.

- API name: American Flights API
- Version name: 1.0
- API endpoint: *Leave blank*
- Description: API for american flights table in training database



11. Click Add.

12. Look at the different sections and links for the API on the API administration page.

The screenshot shows the Anypoint Platform for APIs API Manager interface. At the top, the URL is https://anypoint.mulesoft.com/apiplatform/organization-5/admin/#/organizations/095b2011-d592-498e-9251-38442e7f64... The main title is "American Flights API" version 1.0. Below the title, there's a placeholder "Set the API URL...". A descriptive text says "API for american flights table in training database". There's a button "ADD A TAG". On the left, there are three cards: "API Definition" (with a "Define API in API designer" link), "API Portal" (with a dropdown showing "No portal"), and "API Status" (with a "Configure endpoint" link). Below these cards, tabs for "Applications", "Policies", "SLA tiers", and "Permissions" are visible. Under the "Applications" tab, there's a section for managing consumer applications with steps: "Set an API URL and deploy a proxy" and "Create & publish an API portal for this API version". It also mentions that application developers can request access from the portal. The "Applications" tab is currently selected.

13. Click the API administration link in the main menu; you should see your new API listed.

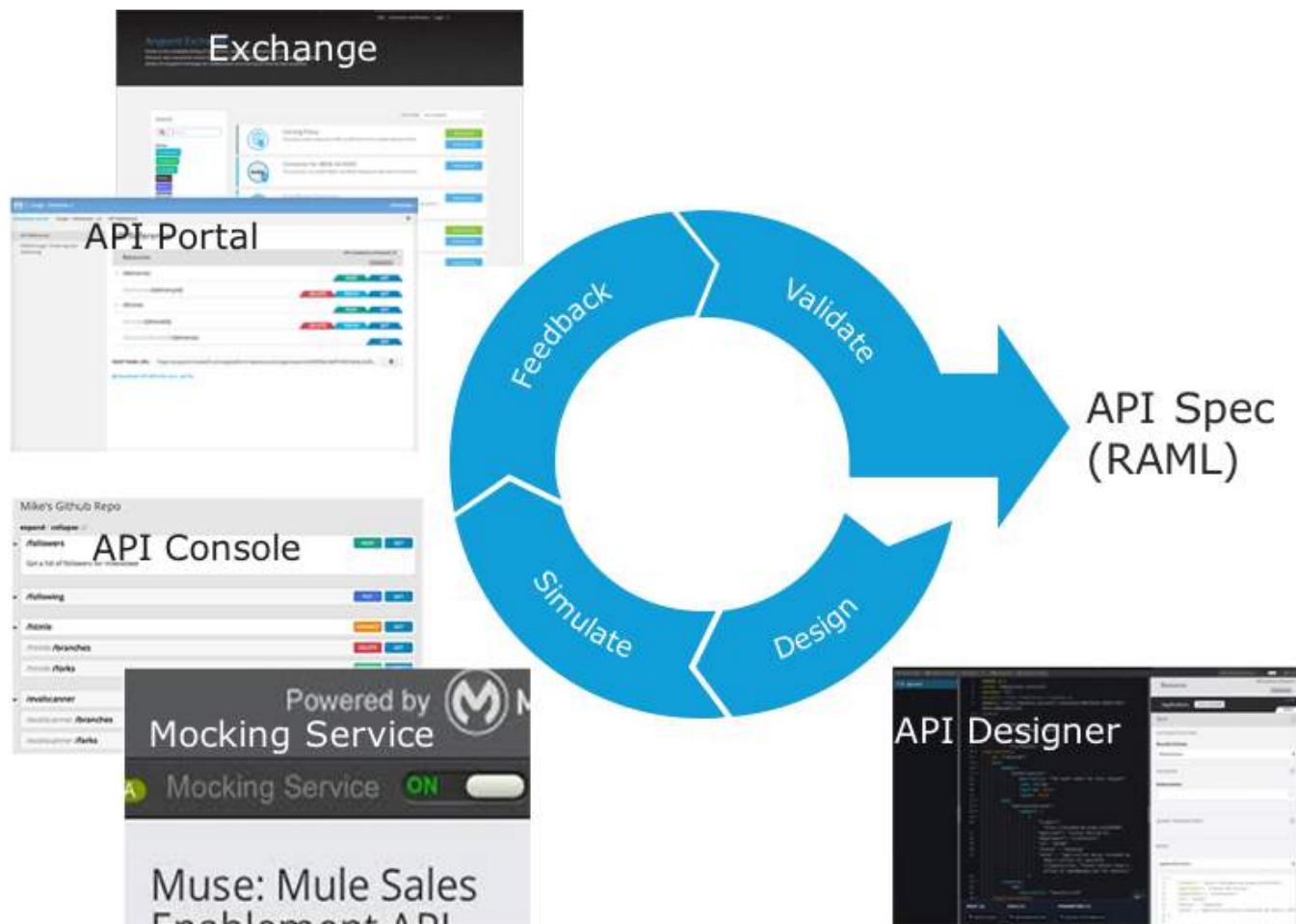
14. Click the row containing version 1.0 of your API; you should see details for it displayed on the right.

The screenshot shows the Anypoint Platform for APIs API Manager interface. The main navigation bar has "Add new API" highlighted. The search bar shows "Search" and "1 - 1 of 1". The filter buttons are "All", "Favorites", "Active", and "Public Portal". The results table shows one entry: "American Flights API" version 1.0, which is "No portal". To the right, a detailed view of the API version is shown. The API name is "American Flights API" and its version is "1.0". The description is "API for american flights table in training database". The "Owners" section lists "Max Mule". Below the API details, tabs for "Applications", "Policies", and "SLA tiers" are visible. The "Applications" tab is selected, showing the message "There are no applications for this API version."

15. Click the version 1.0 link for the API; you should return to the details page for that API.

The screenshot shows the Anypoint Platform for APIs API Manager interface. At the top, the URL is https://anypoint.mulesoft.com/apiplatform/organization-5/admin/#/organizations/095b2011-d592-498e-9251-38442e7f6414/dash... The title bar says "API Manager". The main header displays "American Flights API" and "1.0". Below the header, there's a sub-header "Set the API URL..." with a link icon. A descriptive text follows: "API for american flights table in training database" with a link icon. A "ADD A TAG" button is present. Three cards are shown below: "API Definition" (with a "Define API in API designer" button), "API Portal" (with a "No portal" message), and "API Status" (with a "Configure endpoint" button). The MuleSoft logo is in the top right corner.

Module 2: Designing APIs



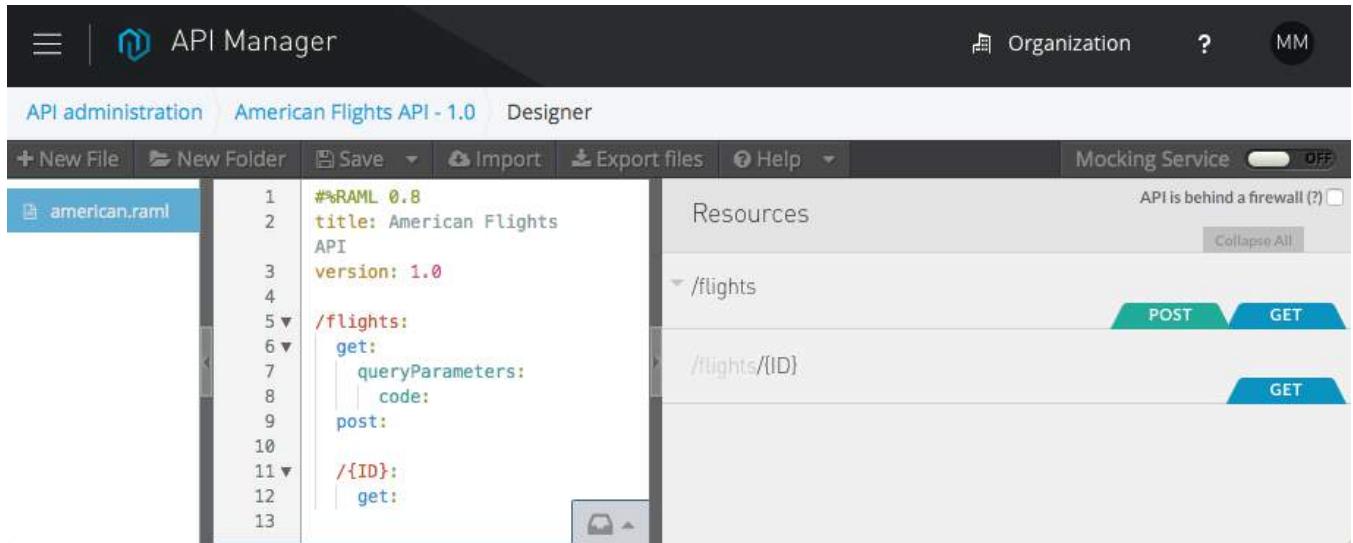
Objectives:

- Define an API with RAML, the Restful API Modeling Language.
- Mock an API to test its design before it is built.
- Create a portal for developers to learn how to use an API.
- Make an API discoverable by adding it to the private Exchange.

Walkthrough 2-1: Use API Designer to define an API with RAML

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

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



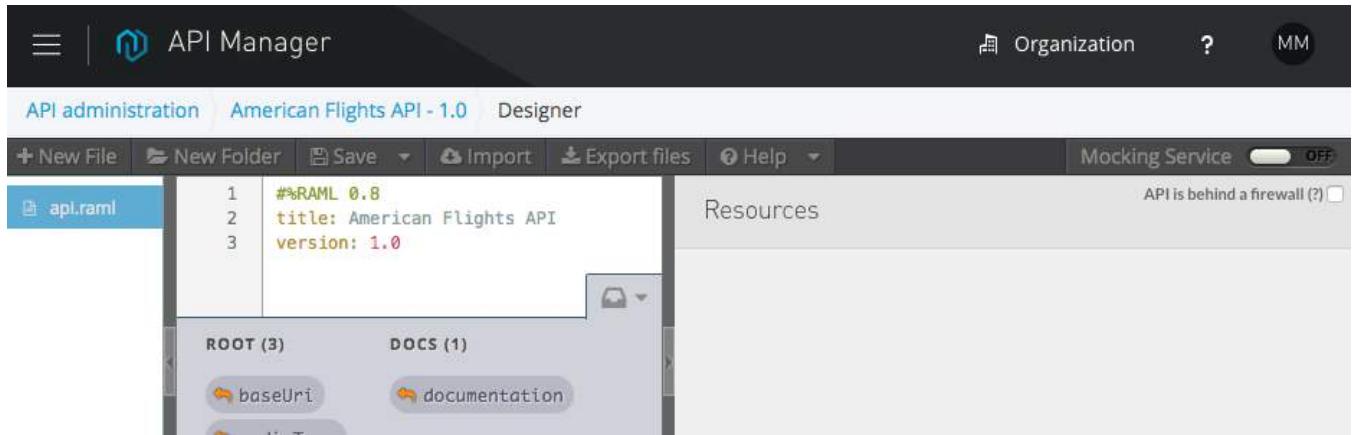
The screenshot shows the API Manager interface with the 'American Flights API - 1.0' selected in the navigation bar. The left sidebar shows a file named 'american.raml'. The main area displays the RAML code for the API:

```
%RAML 0.8
title: American Flights API
version: 1.0
/flights:
  get:
    queryParameters:
      code:
    post:
      /{ID}:
        get:
```

On the right, there is a 'Resources' panel showing a tree structure for the '/flights' resource, with a 'POST' button and a 'GET' button below it. A note says 'API is behind a firewall (?)' with a checkbox.

Open the API Designer

1. Return to the API details page for American Flights API in Anypoint Platform.
2. In the API Definition section, click the Define API in API designer link; the API Designer should open.



The screenshot shows the API Manager interface with the 'American Flights API - 1.0' selected in the navigation bar. The left sidebar shows a file named 'api.raml'. The main area displays the RAML code for the API:

```
%RAML 0.8
title: American Flights API
version: 1.0
```

At the bottom, there are buttons for 'ROOT (3)', 'DOCS (1)', 'baseUri', and 'documentation'.

*Note: To change the background color from black to white, press **Ctrl+Shift+T**.*

Add a RAML resource

3. Place the cursor on a new line of code at the end of the file.
4. Add a resource called flights.

```
/flights:
```

View the API Console

5. Look at the API Console on the right side of the window; you should see the flights resource.

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

The screenshot shows the API Manager interface. On the left, the RAML editor displays the following code:

```
#%RAML 0.8
title: American Flights API
version: 1.0
/flights:
```

On the right, the API console shows the resources listed under the /flights endpoint:

```
Resources
/flights
```

Add RAML methods

6. In the API Designer editor, go to a new line of code.
7. Indent by pressing the Tab key.
8. Press the G key and then the Enter key to add a get method.
9. Look at the API Console; you should see a GET tab for the flights resource.
10. In the editor, press the Enter key and then the P key.
11. In the popup that appears, select post and then press Enter.

The screenshot shows the API Designer editor with the following RAML code:

```
5 ▼ /flights:
6   get:
7   post:
8     patch: methods
9     + post: methods
10    put: methods
11
```

A dropdown menu is open at line 9, showing the options "post:" and "methods".

12. Look at the API Console; you should see GET and POST tabs for the flights resource.

The screenshot shows the API Manager Designer interface. On the left, the 'api.raml' file is open in the code editor, containing the following RAML code:

```
%RAML 0.8
title: American Flights API
version: 1.0

/flights:
  get:
  post:
```

On the right, the API Console displays the '/flights' resource. It shows two buttons at the bottom: 'POST' (green) and 'GET' (blue). The status bar at the top right indicates 'API is behind a firewall (?)' with a checkbox.

Add a nested RAML resource

13. In the editor, press Enter twice.
14. Make sure you are still under the flights resource (at the same indentation as the methods).
15. Add a nested resource for a flight with a particular ID.

/{{ID}}:

16. Add a get method to this resource.
17. Look at the API Console; you should see the nested resource with a GET tab.

The screenshot shows the API Manager Designer interface after adding the nested resource. The 'api.raml' file now includes:

```
%RAML 0.8
title: American Flights API
version: 1.0

/flights:
  get:
  post:
  /{{ID}}:
    get:
```

In the API Console, the '/flights' resource is expanded to show the nested '/{{ID}}' resource. It has a 'GET' button at the bottom. There is also a 'Collapse All' button above the expanded section.

Add a query parameter

18. Locate the API Designer shelf at the bottom of the editor and look at its contents.

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 its icon. If you see the icon, click it to expand it. If you don't see the icon and you also don't see the API Console, you probably have an error in your code, like a missing RAML definition.

19. In the editor, indent under the /flights get method (not the /flights/{ID} get method).
20. Look at the contents of the API Designer shelf again; the parameters shown should have changed.
21. In the API Designer shelf, click queryParameters.

The screenshot shows the MuleSoft API Manager interface. At the top, there's a navigation bar with 'API administration', 'American Flights API - 1.0', and 'Designer' tabs. Below the navigation bar is a toolbar with 'New File', 'New Folder', 'Save', 'Import', and 'Export files'. A file named 'api.raml' is open in the editor, containing the following RAML code:

```

1  #%RAML 0.8
2  title: American Flights API
3  version: 1.0
4
5  /flights:
6    get:
7
8    post:
9
10 /{ID}:
11   get:
12

```

Below the editor, the 'API Designer' shelf is visible. It includes sections for 'PARAMETERS (3)', 'RESPONSES (1)', 'SECURITY (1)', and buttons for 'baseUriParameters', 'headers', and 'queryParameters'. The 'queryParameters' button is highlighted.

22. Indent and add a key named code.

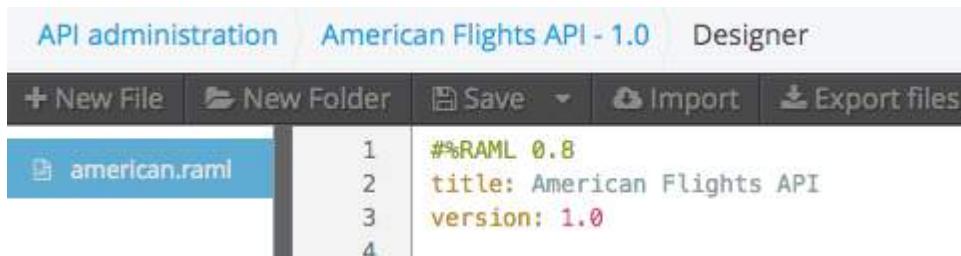
```

1  #%RAML 0.8
2  title: American Flights API
3  version: 1.0
4
5  /flights:
6    get:
7      queryParameters:
8        code:
9    post:
10
11 /{ID}:
12   get:
13

```

Save and rename the RAML file

23. Click the Save button.
24. Right-click api.raml and select Rename.
25. In the Rename a file dialog box, set the name to american.raml and click OK.

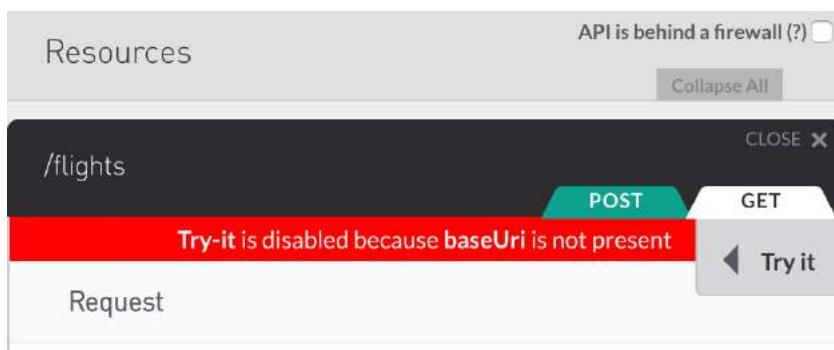


The screenshot shows the 'API administration' interface for the 'American Flights API - 1.0' project. The top navigation bar includes 'API administration', 'American Flights API - 1.0', and 'Designer'. Below the navigation is a toolbar with 'New File', 'New Folder', 'Save', 'Import', and 'Export files'. A file list on the left shows 'american.raml' selected. The main area displays the RAML code:

```
#%RAML 0.8
title: American Flights API
version: 1.0
```

Try to call an API method using the API Console

26. In the API Console, click the GET tab for the /flights resource; you should get a message that Try-it is disabled because baseUri is not present.
27. Click the CLOSE button in the upper-right corner.



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

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

- Add example responses to the API definition.
- Turn on the the mocking service.
- Use the API Console to make calls to the mocked API.

The screenshot shows the API Manager interface. On the left, there's a file tree with a selected file named "american.raml". The main area displays the RAML code for the American Flights API. On the right, there's an "API Console" section with a "Mocking Service" toggle switch set to "ON". Below it, there's a form with a "code" input field and three buttons: "GET", "Clear", and "Reset". At the bottom right of the console, there's a "GET" button and a URL placeholder "/flights/{ID}".

```
%RAML 0.8
baseUri: https://mocksvc.mulesoft.com/mocks/23d99088-f728-414f-adb7-e94005c182f0
title: American Flights API
version: 1.0

/flights:
  get:
    queryParameters:
      code:
    post:
  /{ID}:
    get:
```

Turn on the mocking service

1. Return to API Designer.
2. Locate the Mocking Service slider in the menu bar above the API Console.
3. Slide it to on.

The screenshot shows the API Designer interface. At the top, there's a "Mocking Service" toggle switch set to "ON". Below it, there's a "Resources" section with a "POST" and "GET" button for the "/flights" endpoint. Underneath, there's another "GET" button for the "/flights/{ID}" endpoint.

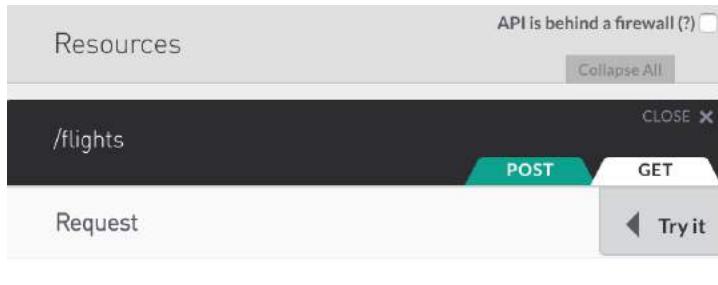
4. Look at the baseUri added to the RAML definition in the editor.

The screenshot shows the API Manager interface with the RAML code editor. The "baseUri" line has been updated to include the URL from the previous screenshot: "baseUri: https://mocksvc.mulesoft.com/mocks/23d99088-f728-414f-adb7-e94005c182f0".

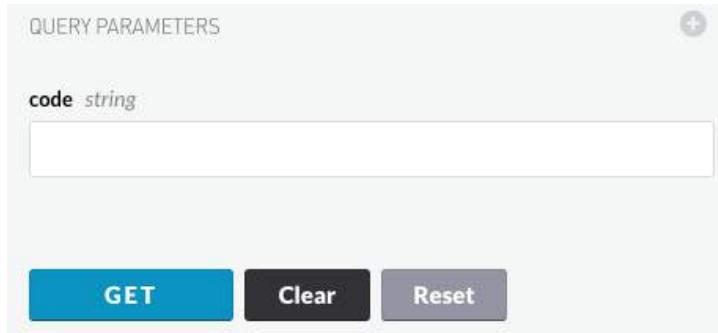
```
%RAML 0.8
baseUri: https://mocksvc.mulesoft.com/mocks/23d99088-f728-414f-adb7-e94005c182f0
title: American Flights API
version: 1.0
```

Test the /flights resource

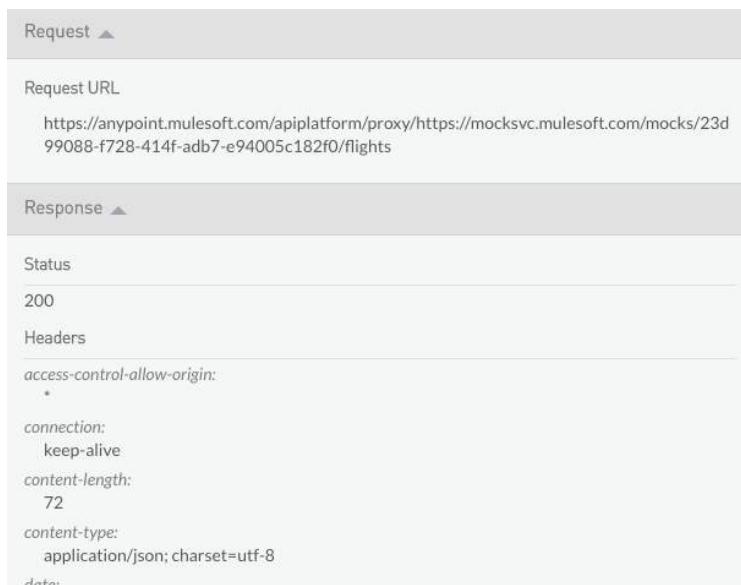
5. In the API Console, click the GET tab for the /flights resource again.
6. Click the Try it button.



7. Locate the text field for the code query parameter.
8. Notice it is not required and has no default or suggested values.
9. Click the GET button.



10. Look at the response; you should get a 200 status code and see the content-type is application/json by default.

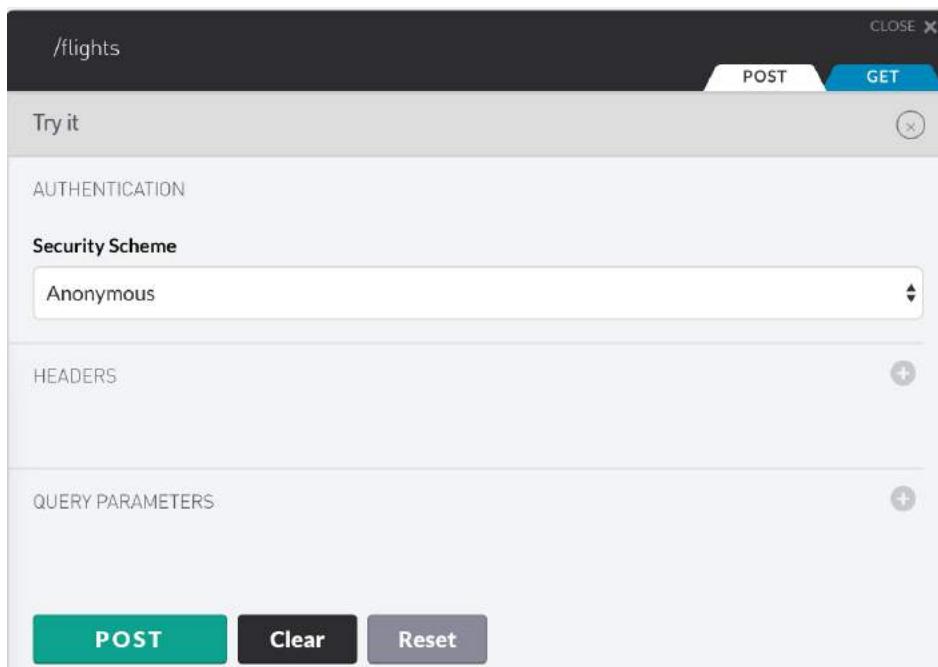


11. Scroll down and look at the response body; you should get a general RAML message placeholder.

```
vary:  
  Accept-Encoding  
Body  
1 | {  
2 |   "message": "RAML had no response information for application/json"  
3 | }
```

12. Scroll up and click the POST tab.

13. Notice that there is no body field in which to enter the data to post with the request.



14. Click the POST button.

15. Look at the response; you should get a 200 status code and see the content-type is application/json with a default message.

Test the /flights/{ID} resource

16. Scroll down to the bottom of the API Console.
17. Click the GET tab for the /flights/{ID} resource.
18. Click the Try it button.

19. Click the GET button; you should get a message that the ID is required.

The screenshot shows a user interface for a REST API test. At the top, it says "URI PARAMETERS" and displays a "GET /flights/{ID}" endpoint. Below this, there is a text input field labeled "ID * string". A red horizontal bar spans across the input field with the word "Required" written in white. Underneath the input field, there are sections for "HEADERS" and "QUERY PARAMETERS", each with a plus sign icon to add new items. At the bottom of the interface are three buttons: a red "Force GET" button, a black "Clear" button, and a black "Reset" button.

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

21. Click the GET button.

22. Look at the response; you should get a 200 status code and see the content-type is application/json with a default message.

Walkthrough 2-3: Add request and response details

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

- Specify data types for get and post method responses.
- Add example JSON responses.
- Specify possible query parameter values.
- Test the API and get the sample responses.

The screenshot shows the MuleSoft API Manager Designer interface. On the left, the RAML file 'american.raml' is open, displaying code for an API endpoint. On the right, the 'Body' section shows a JSON response example for the '/flights' endpoint's 'get' method. The JSON example includes fields like ID, code, price, departure date, origin, destination, empty seats, and plane type.

```
#%RAML 0.8
baseUri:
  https://mocksvc.mulesoft.com/mocks/23d9908
  8-f728-414f-adb7-e94005c182f0
title: American Flights API
version: 1.0

/flights:
  get:
    queryParameters:
      code:
        | enum: [SFO, LAX, CLE]
    responses:
      200:
        body:
          application/json:
            example: [
              {"ID":1, "code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": { "type": "Boeing 737", "totalSeats": 150 }},
              {"ID":2, "code": "ER45if", "price": 500, "departureDate": "2016/03/21", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": { "type": "Airbus A320", "totalSeats": 180 }}]
```

View the specification for the data to be returned by the API

- In your computer's file browser, navigate to your student files folder and open the schemas-and-examples folder.
- Open american-flights-example.json and look at the JSON.
- Copy the JSON.

The screenshot shows a browser window with two tabs: 'APDevFundamentals3.8_s...' and 'american-flights-exampl...'. The content area displays the JSON response example, which is identical to the one shown in the API Manager screenshot above.

```
[{"ID":1, "code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": { "type": "Boeing 737", "totalSeats": 150 }}, {"ID":2, "code": "ER45if", "price": 500, "departureDate": "2016/03/21", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": { "type": "Airbus A320", "totalSeats": 180 }}]
```

Specify details for the /flights get method response

- Return to API Designer.
- Indent under the /flights get method (at the same level as queryParameters).
- In the API Designer shelf, click responses.

7. Indent and add a 200 response.

```
6 ▼ /flights:  
7 ▼   get:  
8     | queryParameters:  
9     |   code:  
10 ▼    responses:  
11      |   200:
```

8. Indent and add a body parameter from the shelf.
9. Indent and add application/json from the shelf or the popup that appears.

```
6 ▼ /flights:  
7 ▼   get:  
8     | queryParameters:  
9     |   code:  
10 ▼    responses:  
11      |   200:  
12        |   body:  
13          |   application/json:  
14  
15         post:
```

Add sample data for the /flights get method response

10. Indent and add an example element.
11. Add a space and then a | after the example element.
12. Indent under example and paste the example code you copied from the american-flights-example.json file.

```
6 ▼ /flights:  
7 ▼   get:  
8     | queryParameters:  
9     |   code:  
10 ▼    responses:  
11      |   200:  
12        |   body:  
13          |   application/json:  
14            |   example: |  
15              [{"ID":1, "code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin":  
"MUA", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737",  
"totalSeats": 150}}, {"ID":2, "code": "ER45if", "price": 345.99, "departureDate":  
"2016/02/11", "origin": "MUA", "destination": "LAX", "emptySeats": 52, "plane":  
{"type": "Boeing 777", "totalSeats": 300}]]  
16
```

Specify details for the /flights/{ID} get method response

13. For the /flights/{ID} get method, add a 200 response with application/json data.

14. Indent under application/json and add an example element with a space and then a | after it.
15. Return to the course snippets.txt file and copy the American Flights API - /flights/{ID} get response example.
16. Return to API Designer and indent and paste the example code you copied.

```

19 ▼  | /{ID}:
20 ▼  |   get:
21 ▼  |     responses:
22 ▼  |       200:
23 ▼  |         body:
24 ▼  |           application/json:
25 ▼  |             example: |
26      |               {"ID":1, "code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}

```

Specify details for the /flights post method response

17. For the /flights post method, add a 201 response with application/json data.
18. Indent under application/json and add an example element with a space and then a | after it.
19. Return to the course snippets.txt file and copy the American Flights API - /flights post response example.
20. Return to API Designer and indent and paste the example code you copied.

```

18 ▼  | post:
19 ▼  |   responses:
20 ▼  |     201:
21 ▼  |       body:
22 ▼  |         application/json:
23 ▼  |           example: |
24      |             {"message": "Flight added (but not really)"}
25

```

21. Save the file.

Get the sample data when making calls to the mocked API

22. In the API Console, click the GET tab for the flights resource again.

23. Scroll down and locate the new response section with the 200 response information with the example data.

Response

200 STATUS 200

Body application/json

Examples: Example

```
[  
  {  
    "ID": 1,  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2016/03/20",  
    "origin": "MUA",  
    "destination": "SFO",  
    "emptySeats": 0,  
    "plane": {  
      "type": "Boeing 737",  
      "totalSeats": 150  
    }  
  },  
  {  
    "ID": 2,  
    "code": "ER45if".  
  }]
```

24. Scroll up and click the Try it button.
25. Click the GET button; you should see the example data returned.

vary:
Accept-Encoding

Body

```
1 [  
2   {  
3     "ID": 1,  
4     "code": "ER38sd",  
5     "price": 400,  
6     "departureDate": "2016/03/20",  
7     "origin": "MUA",  
8     "destination": "SFO",  
9     "emptySeats": 0.  
10}
```

26. Scroll up and click the POST tab for the flights resource.
27. Click the close button for Try it.
28. Look at the request section and notice there is no information about the data that should be sent with the request – the data to be posted.

Specify details for the /flights post method request

29. Return to /flights in the editor.
30. Indent directly under the /flights post method so you are above and at the same level as responses.
31. Add a body element.
32. Indent and add application/json.
33. Indent and add an example element with a space and then a | after it.
34. Indent.
35. Return to the course snippets.txt file and copy the American Flights API - /flights post request body example.
36. Return to API Designer and indent and paste the example code you copied.

```
18 ▼ | post:
19 ▼ |   body:
20 ▼ |     application/json:
21 |       example: |
22 |         {"code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin": "MUA",
23 |           "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats":
24 |             150}}
25 ▼ |   responses:
26 ▼ |     201:
27 |       body:
```

Specify details for the /flights get method request

37. In the /flights get method, indent under the code query parameter and look at the possible parameters in the shelf.
38. Click enum.
39. Set enum to an array of values including SFO, LAX, and CLE.

```
6 ▼ | /flights:
7 ▼ |   get:
8 ▼ |     queryParameters:
9 |       code:
10 |         enum: [SFO, LAX, CLE]
11 ▼ |   responses:
```

40. Save the file.

Use the sample data when making calls to the mocked API

41. In the API Console, click the GET tab for the /flights resource again.
42. Click the Try it button.

43. Locate the code query parameter; it should now have a drop-down menu with a list of possible values.
44. Select a value and click the GET button; you should see all the example data still returned.
45. Scroll up and click the Close button.
46. Click the POST tab for the /flights resource.
47. Look at the new body section.

BODY

The screenshot shows a JSON editor interface. At the top, there is a dropdown menu labeled "application/json". Below it, a "Examples" section contains a link "Example". The main area is a text input box containing the following JSON object:

```
{
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2016/03/20",
  "origin": "MUA",
  "destination": "SFO",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
}
```

48. Click the Try it button.
49. Locate the text area for entering JSON for the body to post; it should be prefilled with the example JSON.

The screenshot shows a configuration screen for a POST request. At the top, there is a "BODY" section with a dropdown menu set to "application/json". Below it is a text area containing the same JSON object as the previous screenshot:

```

1 {
2   "code": "ER38sd",
3   "price": 400,
4   "departureDate": "2016/03/20",
5   "origin": "MUA",
6   "destination": "SFO",
7   "emptySeats": 0,
8   "plane": {
9     "type": "Boeing 737",
10    "totalSeats": 150
11  }
12 }
```

At the bottom of the text area is a "Prefill with example" button. Below the text area are three buttons: "POST" (highlighted in green), "Clear", and "Reset".

50. Click the POST button; you should get a 201 response with a message that the flight was added (but not really).

Walkthrough 2-4: Create an API portal

In this walkthrough, you create an API portal for developers to locate, learn about, and try out the API. You will:

- Create an API portal.
- Add content to the portal.
- Customize the portal.
- View the resulting developer portal.

The screenshot shows the MuleSoft API Portal interface. At the top, it says "MuleSoft // Dev | American Flights API 1.0" and "username05". Below that is a navigation bar with "Developer portal", "American Flights API - 1.0", and "API reference". The main area has tabs for "Overview" and "API reference", with "API reference" being active. On the left, there's a sidebar with "Resources" and a "Collapse All" button. The main content area shows the "/flights" endpoint with "POST" and "GET" methods listed. There's also a "GET" button for the "/flights/{ID}" path.

Create a portal

1. Return to API Designer.
2. Click the American Flights API – 1.0 link in the top-left corner.

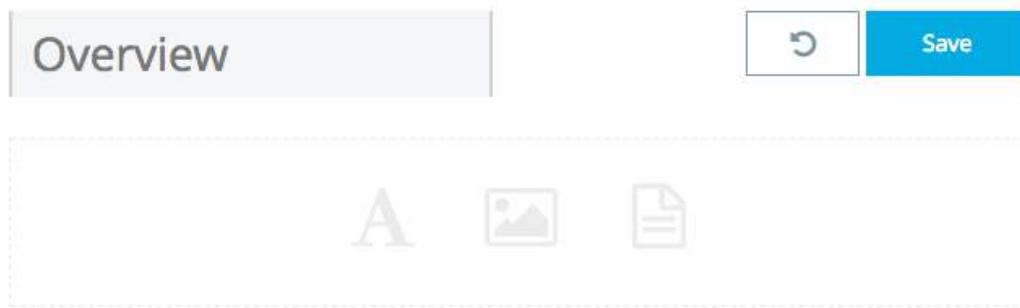
The screenshot shows the API Manager interface. The title bar says "API Manager". The menu bar includes "API administration", "American Flights API - 1.0", and "Designer". The toolbar has buttons for "New File", "New Folder", "Save", "Import", "Export files", and "Help". Below the toolbar, a file named "american.raml" is listed. The main workspace shows some RAML code: "#%RAML 0.8" and "baseUri: https://api.americanairlines.com/v1/flights".

3. On the API details page, locate the API Portal section.
4. In the drop-down menu, select Create new portal; the API Portal Designer should open.

The screenshot shows the API Portal Designer interface. It has three main sections: "API Definition" (with a checkmark), "API Portal", and "API Status". The "API Definition" section says "Use the API Designer to create a concise, human-readable API definition with RAML." It has links for "Edit in API designer" and "Download as a .zip file". The "API Portal" section says "Publishing an API portal allows you to expose documentation and other content that can help developers understand how to use your API." It has a dropdown menu currently set to "No portal", with "Create new portal" as an option. The "API Status" section says "Configuring the API endpoint allows you to manage your API with policies and SLA tiers." It has a link for "Configure endpoint".

Customize the portal's home page

5. In the API Portal Designer, click the Home text in the page and change it to Overview.
6. Return to the course snippets.txt file and copy the text for the API Portal overview text.
7. Return to the API Portal Designer and click the A symbol in the middle of the page.



8. In the text area that appears, paste the text.
9. Look at the markdown syntax being used to format the text.

- Get all flights
- Get a flight with a specific ID
- Add a flight
To the right of the text editor are two buttons: a blue 'Save' button and a white 'Cancel' button."/>

10. Click outside the text area and look at the formatted content.
11. Click the Save button.

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

Explore the API reference page

12. In the left-side navigation, click the API reference link.
13. Explore the API, just as you did in the API Console in API Designer.

The screenshot shows the API Manager interface. At the top, there's a navigation bar with 'API Manager' and other tabs like 'Organization', '?', and 'MM'. Below the navigation is a header for 'American Flights API' with icons for 'Private', 'Themes', and 'Live portal'. On the left, a sidebar has 'Add' and 'Overview' buttons, and the 'API reference' button is selected, indicated by a blue border. The main area is titled 'API reference' and shows 'Resources' with a 'flights' endpoint. Under 'flights', there are 'POST' and 'GET' methods. A 'ROOT RAML URL' field contains the URL <https://anypoint.mulesoft.com/apiplatform/repository/v2/>, with a 'Copy' button to its right. There's also a link to 'Download API definition as a .zip file'.

Save the RAML URL

14. Locate the ROOT RAML URL at the bottom of the API reference.
15. Click the Copy button located to the right.
16. Return to the course snippets.txt file and paste it under Your American Flights RAML URL.

```
46 * Your American Flights RAML URL  
47 https://anypoint.mulesoft.com/apiplatform/repository/v2/  
48  
49 * Your American Flights API Portal URL
```

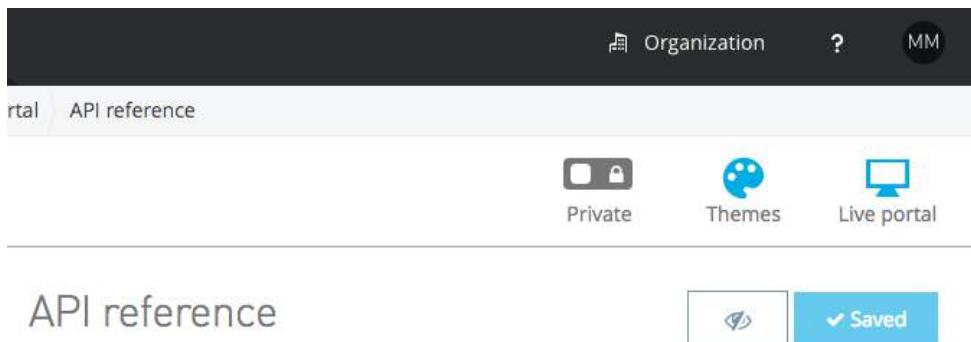
Make pages visible

17. Return to the API Portal Designer.
18. In the left-side navigation, check the checkboxes to the left of the Overview and API reference pages.
19. Click the Set to visible button (the eye); the eye icons next to both pages should now be green.

This screenshot shows the same API Manager interface as before, but with a different state. The 'Overview' and 'API reference' buttons in the sidebar now have blue checkboxes to their left, indicating they are selected. The main 'API reference' section still displays the 'flights' resource and its methods.

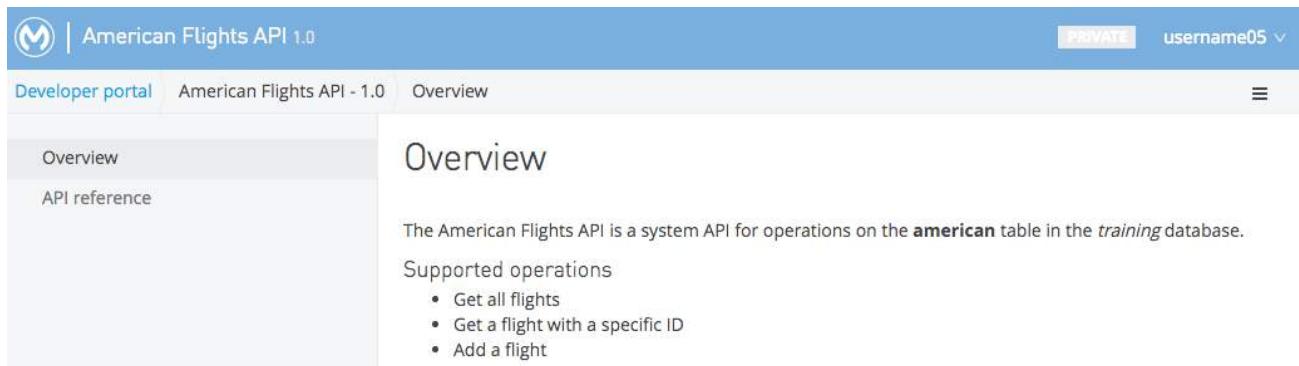
View the developer portal

20. Click the Live portal button.



The screenshot shows the top navigation bar of a developer portal. It includes links for 'Organization', a help icon, and user profile information ('MM'). Below the navigation is a horizontal menu bar with three items: 'Private' (represented by a lock icon), 'Themes' (represented by a circular icon with dots), and 'Live portal' (represented by a computer monitor icon). The 'Live portal' button is highlighted with a blue border. The main content area is titled 'API reference'. At the bottom right of this area is a blue button with a checkmark and the text 'Saved'.

21. Examine the API Portal that opens in a new browser tab or window.

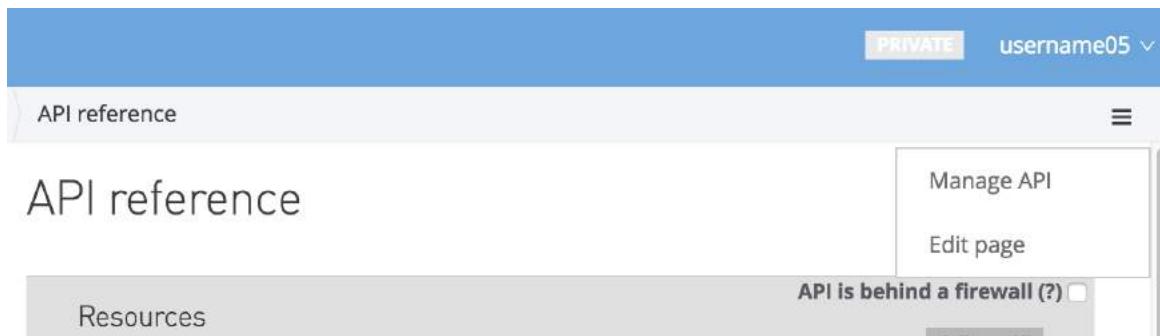


The screenshot shows the 'American Flights API 1.0' developer portal. The top header includes the MuleSoft logo, the API name, a 'PRIVATE' status indicator, and a user dropdown ('username05'). The main navigation bar has links for 'Developer portal', 'American Flights API - 1.0', and 'Overview'. On the left, there's a sidebar with 'Overview' and 'API reference' options. The main content area is titled 'Overview' and contains a brief description of the API, supported operations (Get all flights, Get a flight with a specific ID, Add a flight), and a 'Supported operations' section with a bulleted list.

22. In the left-side navigation, click API reference.

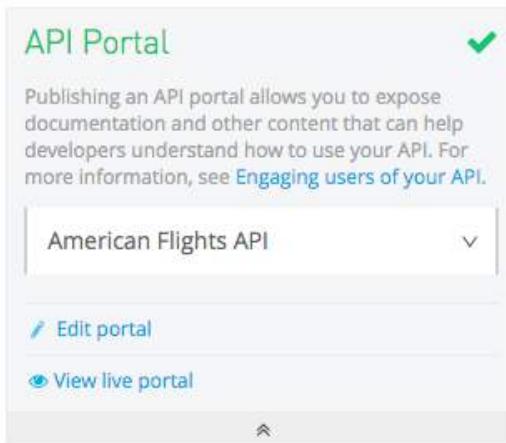
23. Explore the API.

24. In the main menu bar, click the menu button (the three horizontal lines) located all the way to the right and select Manage API; you should return to the API Detail page in Anypoint Platform.



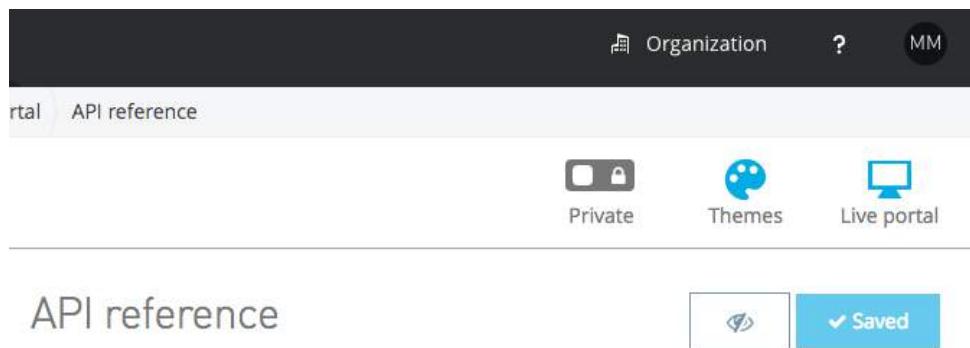
The screenshot shows the 'API reference' page. The top header includes a 'PRIVATE' status, a user dropdown ('username05'), and a menu icon. The main content area is titled 'API reference'. A context menu is open over the 'Manage API' link in the sidebar, listing 'Manage API' and 'Edit page'. A note at the bottom right says 'API is behind a firewall (?)' with a checkbox. The sidebar also includes a 'Resources' link.

25. In the API Portal section, click the Edit portal link.



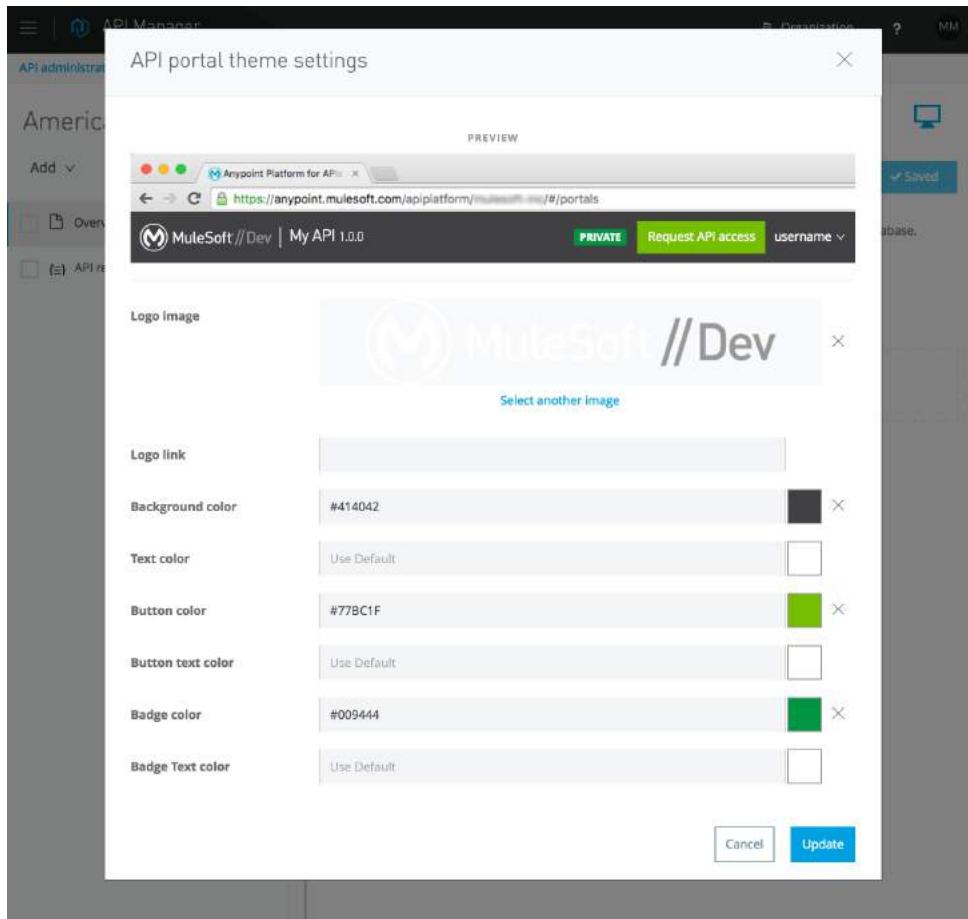
Skin the portal

26. In the API Portal Designer, click the Themes button; an API portal theme settings dialog box should open.



27. In your computer's file browser, navigate to the student files and locate the PNG file in the resources folder.
28. Drag the PNG file from the student files to the API Portal theme settings dialog box and drop it in Logo image area.
29. Click in the Background color text box.
30. In the color pop-up that appears, click in the color bar on the right to select a color and then click in the color box to select a shade.
31. Look at the preview at the top of the dialog box and make sure the color looks good with the logo.
32. Change any other colors you want.

33. Click the Update button.



34. In the API Portal Designer, click the Live portal button again; you should see the new portal skin.

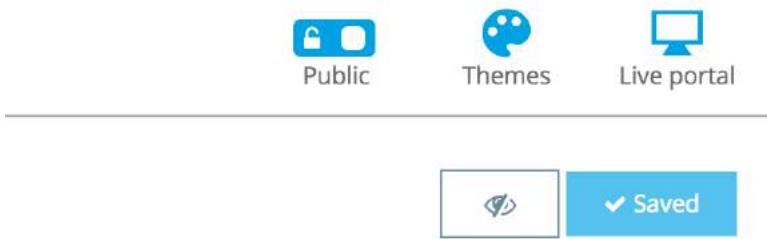
A screenshot of the API Portal Designer showing the 'American Flights API 1.0' live portal. The top navigation bar includes the MuleSoft logo, the portal name, a 'PRIVATE' badge, and a user session indicator ('username05'). The main content area features a sidebar with 'Overview' and 'API reference' tabs, and a main panel with the title 'Overview'. The text in the main panel states: 'The American Flights API is a system API for operations on the american table in the training database.' It also lists 'Supported operations' with three items: 'Get all flights', 'Get a flight with a specific ID', and 'Add a flight'.

35. Look at the upper-right corner and locate the PRIVATE badge.

36. Close the tab or web browser window.

Make the portal public

37. Return to the API Portal Designer.
38. Click the Private slider in the upper-right corner; it should now say Public.



or operations on the **american** table in the *training* database.

Save the portal URL

39. Click the API Administration link in the upper-left corner of the API Portal Designer.
40. Locate the Public portal link next to the version of the API.

A screenshot of the MuleSoft API Manager interface. The top navigation bar has a '☰' icon, the 'API Manager' logo, and the 'API administration' tab selected. Below the navigation is a search bar with 'Search' and a '×' button, and a page navigation section showing '1 - 1 of 1' with left and right arrows. Underneath, there are tabs for 'All', 'Favorites', 'Active', and 'Public Portal'. A list item for 'American Flights API' is shown with its version '1.0' and a 'Public portal' link with a copy icon.

41. Right-click the link and select Copy Link Address (or your browser equivalent).
42. Return to the course snippets.txt file and paste it under Your American Flights API Portal URL.
43. Save the file.

```
46 * Your American Flights RAML URL
47 https://anypoint.mulesoft.com/apiplatform/repository/v2/orga
48
49 * Your American Flights API Portal URL
50 https://anypoint.mulesoft.com/apiplatform/organization-5/#/p
```

Walkthrough 2-5: Add an API to the Anypoint Exchange

In this walkthrough, you enhance the discoverability of an API by adding it to the private Anypoint Exchange. You will:

- Give yourself permission to publish items on the Exchange.
- Add a new RAML API to the private Exchange.
- Submit an item for approval.
- Approve and publish an item to the private Exchange.

The screenshot shows the Anypoint Exchange interface. At the top, there's a navigation bar with icons for Organization, Help, and a user profile. Below the bar, a search bar contains the text "Search". On the left, a sidebar has a "Show content from:" dropdown set to "Everywhere" and a "Show:" dropdown with options: Connectors (selected), Templates, Examples, RAMLs (selected), WSDLs, and Others. In the main area, there are filters for Status and Scope, and a "Sort by: Last modified" dropdown. A search bar below these filters shows the text "Showing: RAMLs". The results list two items: "American Flights API" (Organization) and "Schedule FHIR RAML" (MuleSoft). Each item has a "View details" button and an "Open Portal" button. The "American Flights API" item also has a small blue circular icon next to its name.

Review private Exchange settings

1. Return to Anypoint Platform.
2. In the main menu, select Access Management.
3. In the left-side navigation, click Exchange.
4. Review the information.

Make yourself an Exchange Administrator

5. In the left-side navigation, click Roles.
6. View the Exchange roles.

Exchange Administrators	0	Exchange Administrators
Exchange Contributors	0	Exchange Contributors
Exchange Viewers	0	Exchange Viewers
Organization Administrators	1	Organization Administrators
Portals Viewer	0	Viewer of all portals in the organization

- Click Exchange Administrators.
- Click in the Add a user text box and select yourself from the drop-down menu that appears.
- Click the add button.

Access Management

Organization

Users

Roles

Environments

External Identity

Audit Logs

Exchange Administrators

Description: Exchange Administrators

Name	Username	Email
Max Mule		

+ (Add)

Create a new item to add to the Exchange

- In the main menu, select Exchange.
- In the Exchange, click the Add item button and select RAML.

Show content from:

Everywhere

Show:

RAMLs

Add item

Search

Sort by: Rating

Sending JSON to a JMS Queue (MuleSoft)
Learn how to connect to JMS Queues and Topics.
★★★★★ (4 votes)

Extracting Data from LDAP (MuleSoft)

Download

View details

Download

- Set the item name to American Flights RAML.

Back to list

American Flights RAML

american-flights-api

Icon URL:
Icon Url

Save changes

Discard

13. Scroll down to the versions section and click the Add version button.

14. Enter the following information:

- RAML version: 1.0
- API version: 1.0
- API Portal URL: *Use the URL for your portal that you saved in the course snippets.txt file*
- RAML URL: *Use the URL for your RAML that you saved in the course snippets.txt file*

Versions

RAML version	API version
RAML version *	API version
1.0	1.0
API Portal URL	<input type="text" value="https://anypoint.mulesoft.com/apiplatform/organization-05/#/portals/1"/>
RAML URL	<input type="text" value="https://anypoint.mulesoft.com/apiplatform/repository/v2/organization"/>
Documentation URL	<input type="text"/>
Done Discard	

15. Click Done.

16. Scroll up and click the Save new item button.

Locate the item

17. Click the Back to list link.

18. Click the RAMLs button in the upper-left corner of the Exchange.

19. In the Status menu beneath the Add item button, select Waiting for approval; you should see nothing listed.

20. Change the Status menu to Work in progress; you should your RAML.

The screenshot shows the Exchange interface with a search bar and a dropdown menu for 'Status'. The 'Status' dropdown is open, showing options: All, Published, Waiting for approval, and Work in progress. The 'Work in progress' option is highlighted. On the right, there's a card for 'American Flights API' with 'Organization' selected, and buttons for 'Open Portal' and 'View details'.

Publish the item

21. Click the View details button for the American Flights RAML.
22. Locate the Work in progress status.
23. Click the drop-down button on the Publish button and select your organization.

The screenshot shows the 'American Flights RAML' details page. It includes a 'Back to list' link, the title 'American Flights RAML' with an 'Organization' tag, and a 'By: Max Mule' section. On the right, there are several buttons: 'Open Portal', 'Share URL', 'Edit', 'Clone me!', and a 'Publish' button which has 'Waiting for approval' selected. A note at the bottom says: 'Please open the Exchange from Anypoint Studio to rate content.'

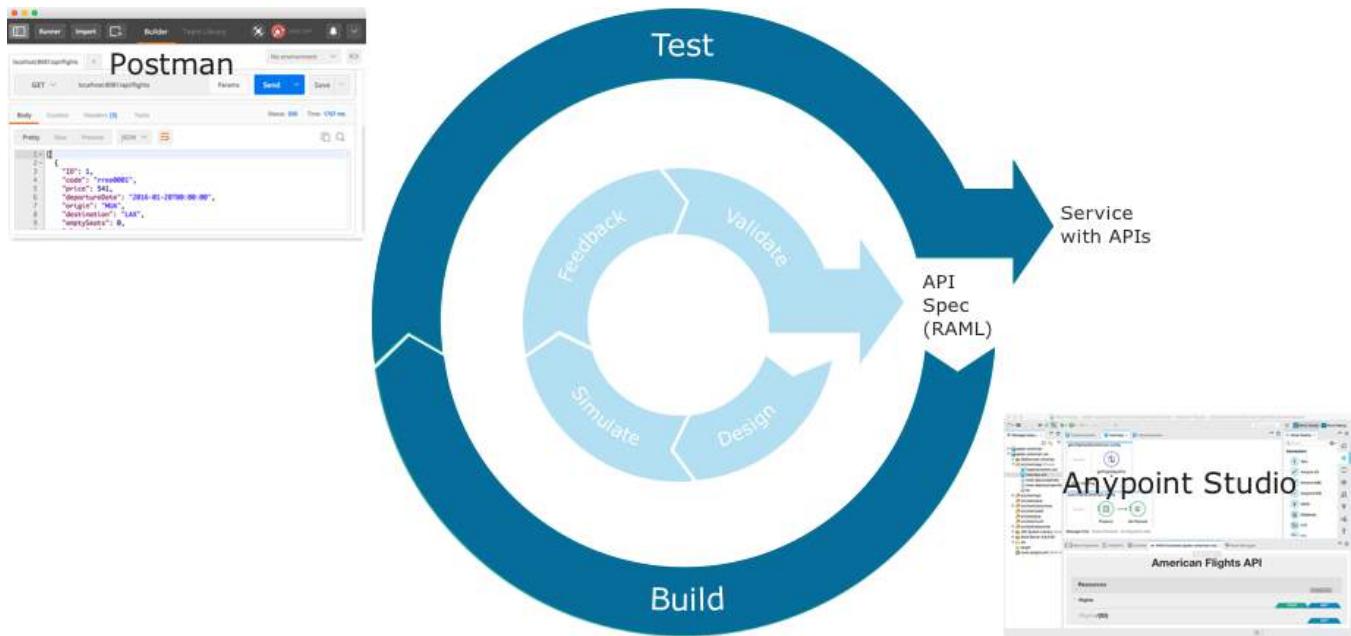
Note: Exchange contributors would have a Request to Publish instead of a Publish button and then an Exchange administrator would have to approve the item.

View the new item in the Exchange

24. In the Status menu beneath the Add item button, select Published; you should see your RAML listed.
25. Change the Status menu to All; you should see your RAML listed.

The screenshot shows the MuleSoft Exchange interface. At the top, there is a navigation bar with icons for Organization, Help, and MM. The main area has a search bar and a 'Add item' button. On the left, there are filters for 'Show content from:' (Everywhere), 'Status' (All), 'Scope' (All), and 'Sort by' (Last modified). A sidebar on the left lists categories: Connectors (highlighted in blue), Templates, Examples, RAMLs, WSDLs, and Others. Below these filters, the results are displayed under the heading 'Showing: RAMLs'. There are two items listed: 'American Flights API' and 'Schedule FHIR RAML'. Each item has a 'View details' button and an 'Open Portal' button.

Module 3: Building APIs



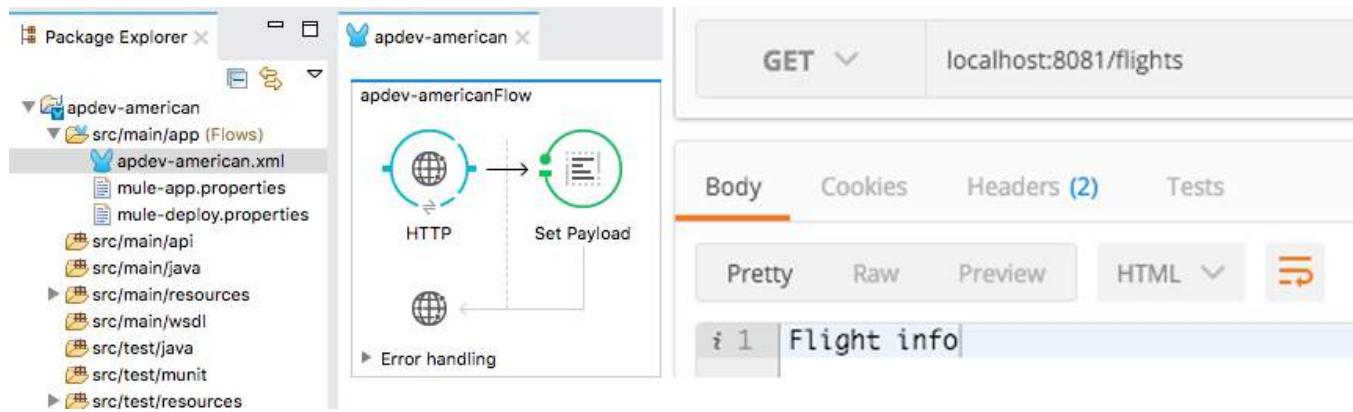
Objectives:

- Introduce Mule applications, flows, messages, and message processors.
- Use Anypoint Studio to create a flow graphically.
- Build, run, and test a Mule application.
- Use a connector to connect to a database.
- Use the graphical DataWeave editor to transform data.
- Create a RESTful interface for an application from a RAML file.
- Connect an API interface to the implementation.

Walkthrough 3-1: Create a Mule application with Anypoint Studio

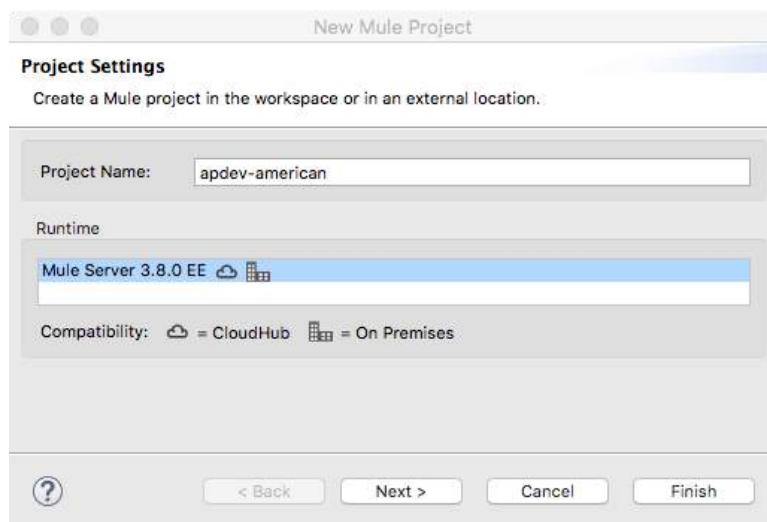
In this walkthrough, you build a Mule application. You will:

- 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 Postman.



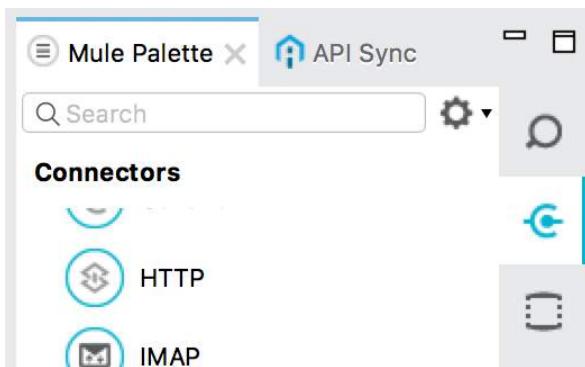
Create a Mule project

1. Open Anypoint Studio.
2. Select File > New > Mule Project.
3. Set the Project Name to apdev-american.
4. Ensure the Runtime is set to the latest version of Mule.
5. Click Finish.

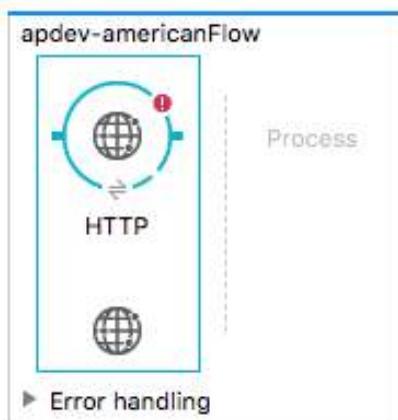


Create an HTTP connector endpoint to receive requests

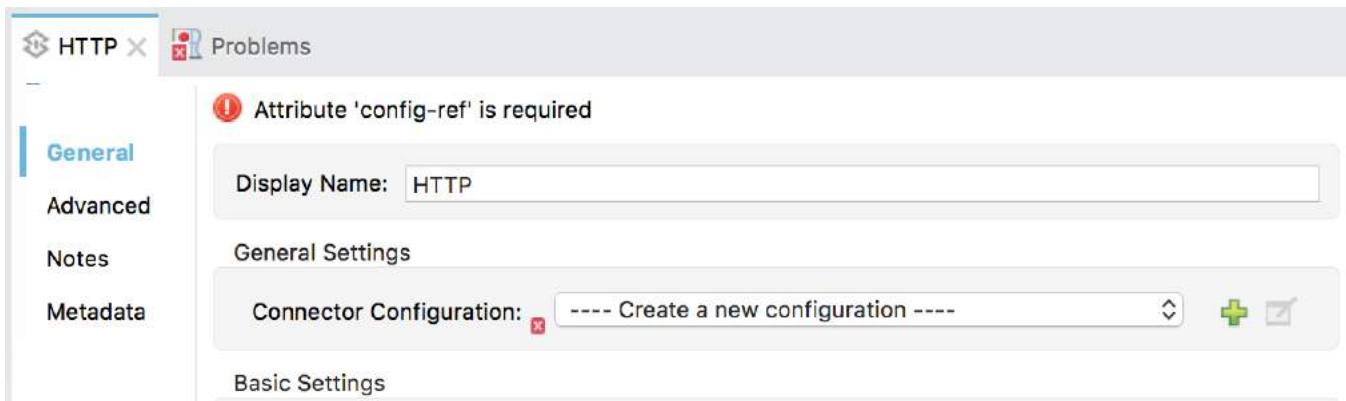
6. In the Mule Palette, select the Connectors tab.



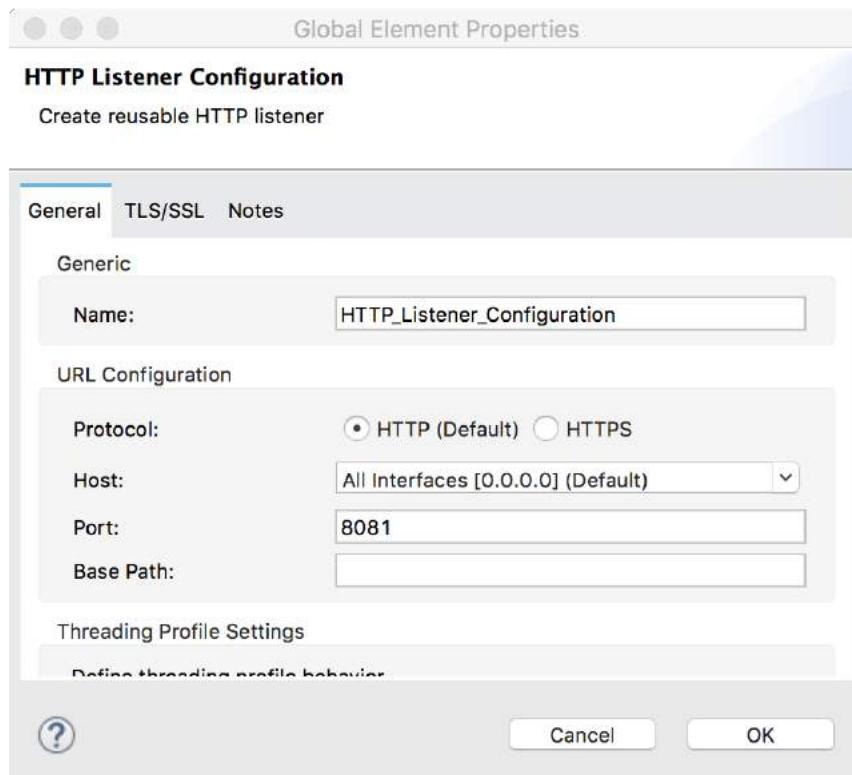
7. Drag an HTTP connector from the Mule Palette to the canvas.



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

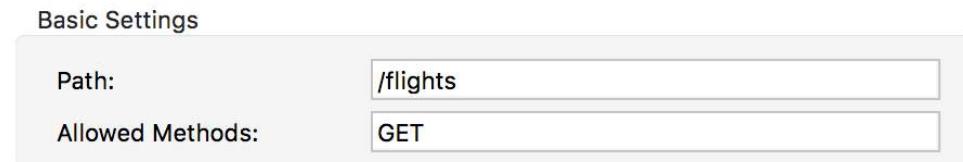


9. In the Global Element Properties dialog box, look at the default values and click OK.



10. In the HTTP properties view, set the path to /flights.

11. Set the allowed methods to GET.

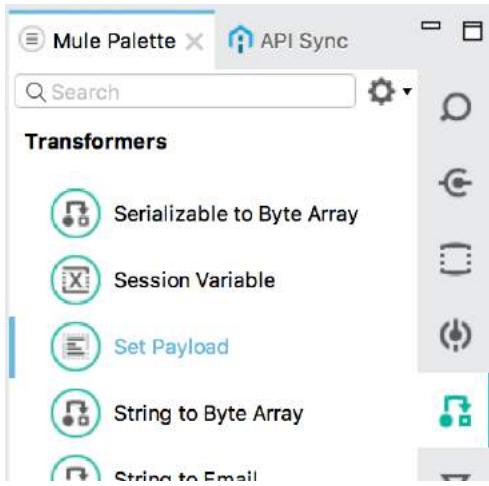


12. Click the Apply Changes button; the errors in the Problems view should disappear.

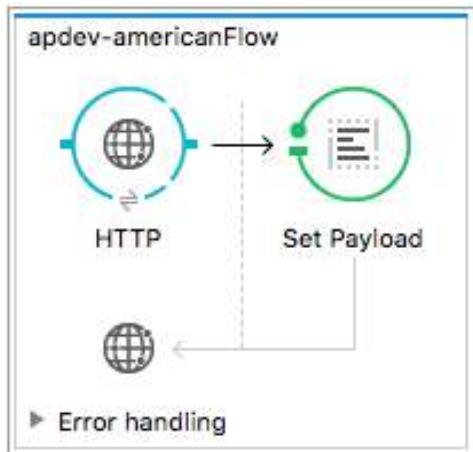


Display data

13. In the Mule Palette, select the Transformers tab.

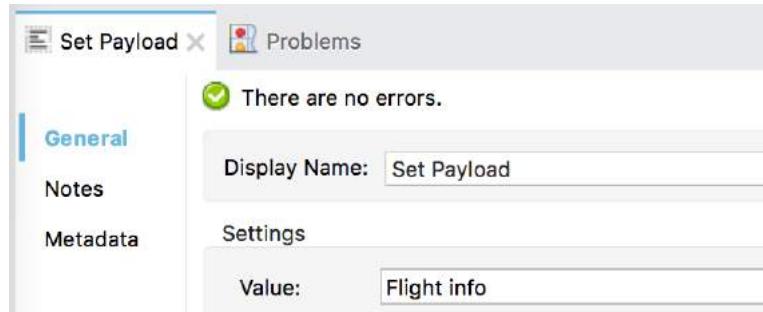


14. Drag a Set Payload transformer from the Mule Palette into the process section of the flow.



Configure the Set Payload transformer

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



16. Click the Configuration XML link at the bottom of the canvas and examine the corresponding XML.

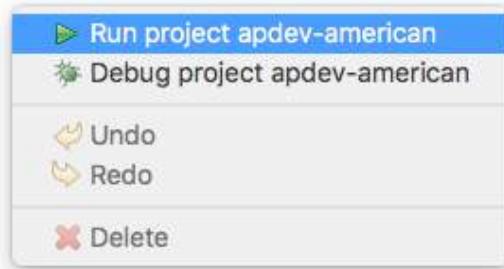


```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3<mule xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns="http://www.mulesoft.c
4     xmlns:spring="http://www.springframework.org/schema/beans"
5     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfr
7     http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/curren
8     http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/curren
9     <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081"
10    <flow name="apdev-americanFlow">
11        <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedN
12            <set-payload value="Flight info" doc:name="Set Payload"/>
13        </flow>
14    </mule>
```

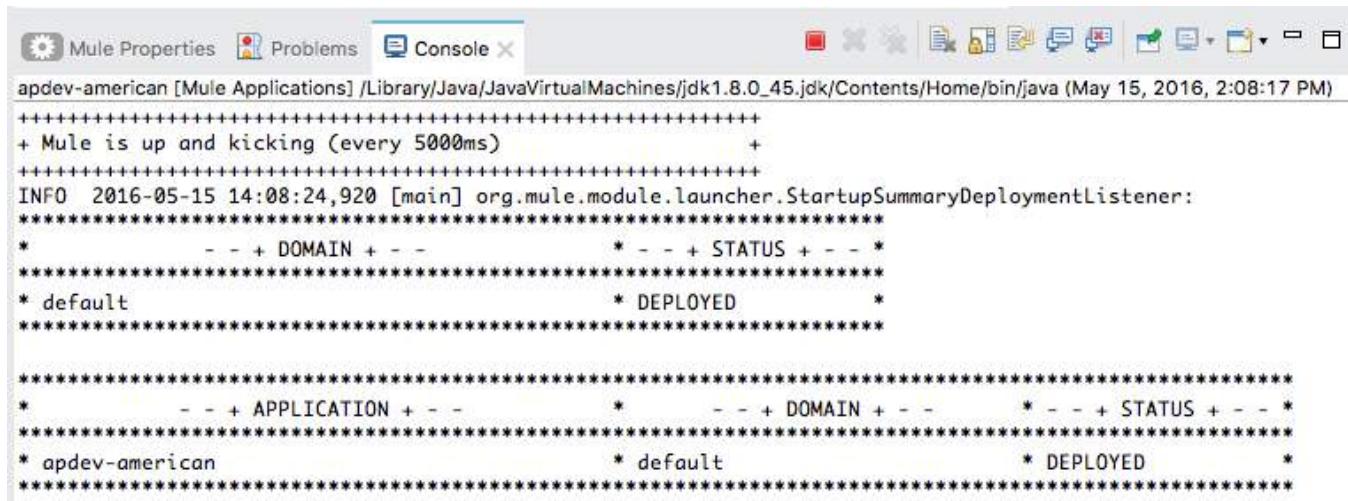
17. Click the Message Flow link to return to the canvas.
18. Click the Save button or press Cmd+S or Ctrl+S.

Run the application

19. Right-click in the canvas and select Run project apdev-american.



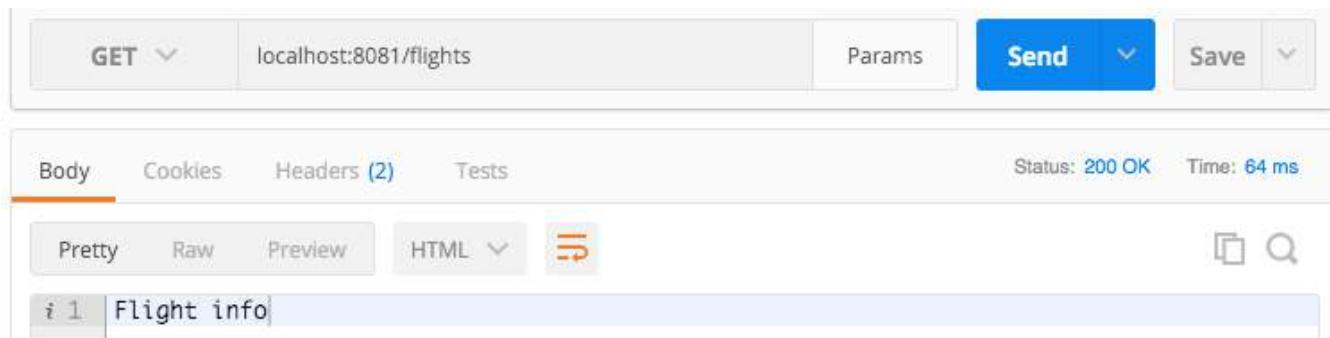
20. Watch the Console view; it should display information letting you know that both the Mule runtime and the apdev-american application started.



```
Mule Properties Problems Console
apdev-american [Mule Applications] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (May 15, 2016, 2:08:17 PM)
+ Mule is up and kicking (every 5000ms)
INFO 2016-05-15 14:08:24,920 [main] org.mule.module.launcher.StartupSummaryDeploymentListener:
*****
*      - - + DOMAIN + - - * - - + STATUS + - - *
*****
* default                                * DEPLOYED   *
*****
*      - - + APPLICATION + - - *      - - + DOMAIN + - - * - - + STATUS + - - *
*****
* apdev-american                         * default        * DEPLOYED   *
*****
```

Test the application

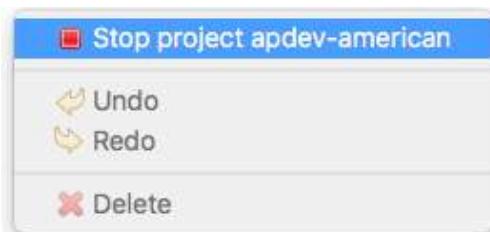
21. Return to Postman.
22. Make sure the method is set to GET and that no headers or body are set for the request.
23. Make a GET request to <http://localhost:8081/flights>; you should see Flight info displayed.



The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: localhost:8081/flights
- Params: None
- Send button: Blue button labeled "Send".
- Status: 200 OK
- Time: 64 ms
- Body tab selected, showing "Flight info" in the preview area.
- Headers tab: (2) entries
- Tests tab: None
- Pretty, Raw, Preview, HTML buttons
- Search and refresh icons

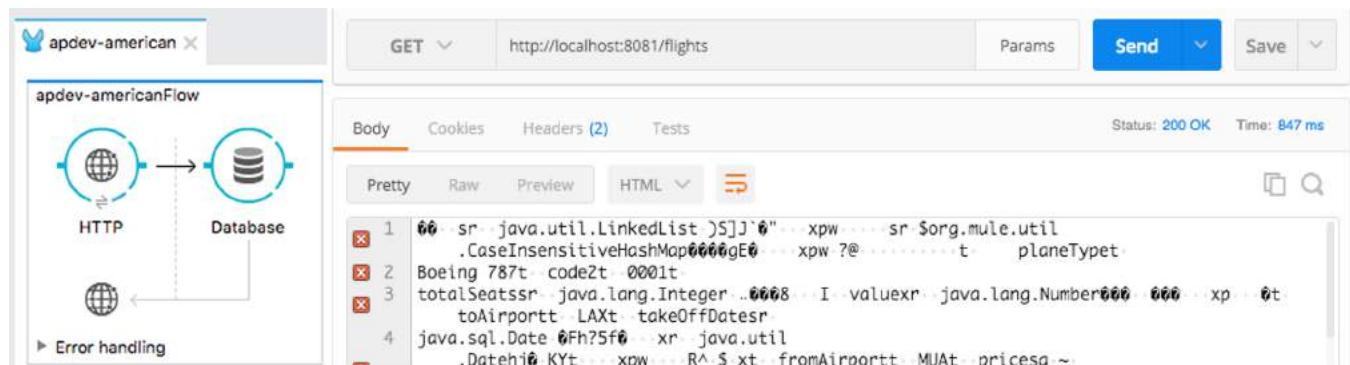
24. Return to Anypoint Studio.
25. Right-click in the canvas and select Stop project apdev-american.



Walkthrough 3-2: 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 connector endpoint.
- 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 endpoint 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 database information.

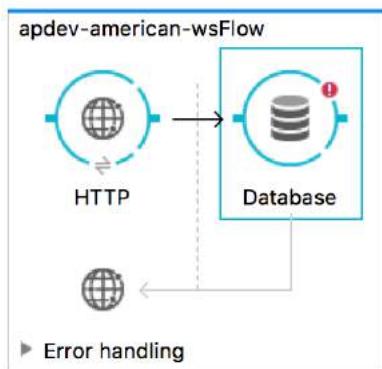
```
64  * MODULE 3 ****
65
66  * MySQL database
67  db.host = mudb.mulesoft-training.com
68  db.port = 3306
69  db.user = mule
70  db.password = mule
71  db.database = training
72  American table: american
```

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-paced 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 the Connectors tab.

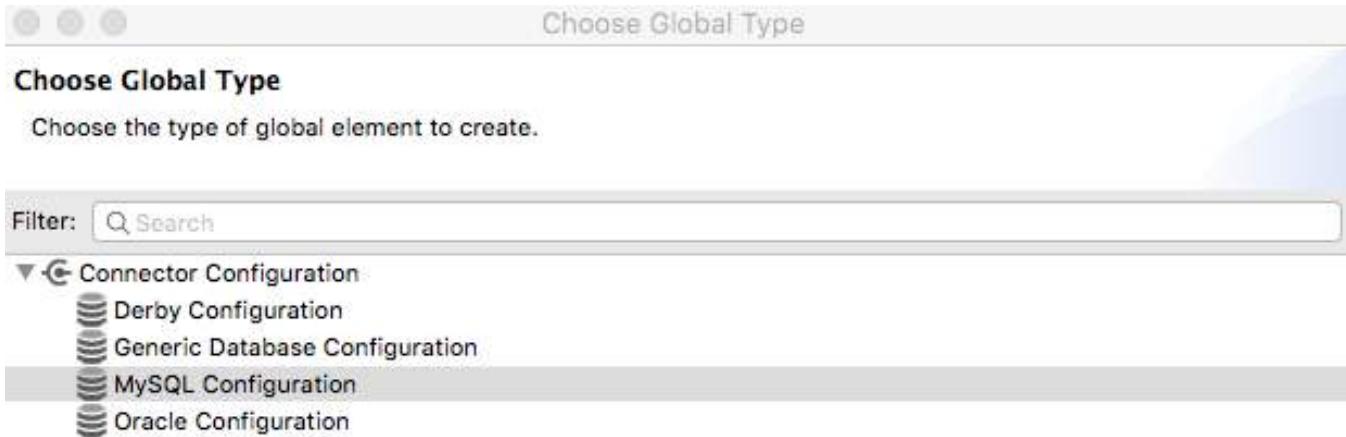
5. Drag a Database connector to the process section of the flow.



6. Double-click the Database endpoint.

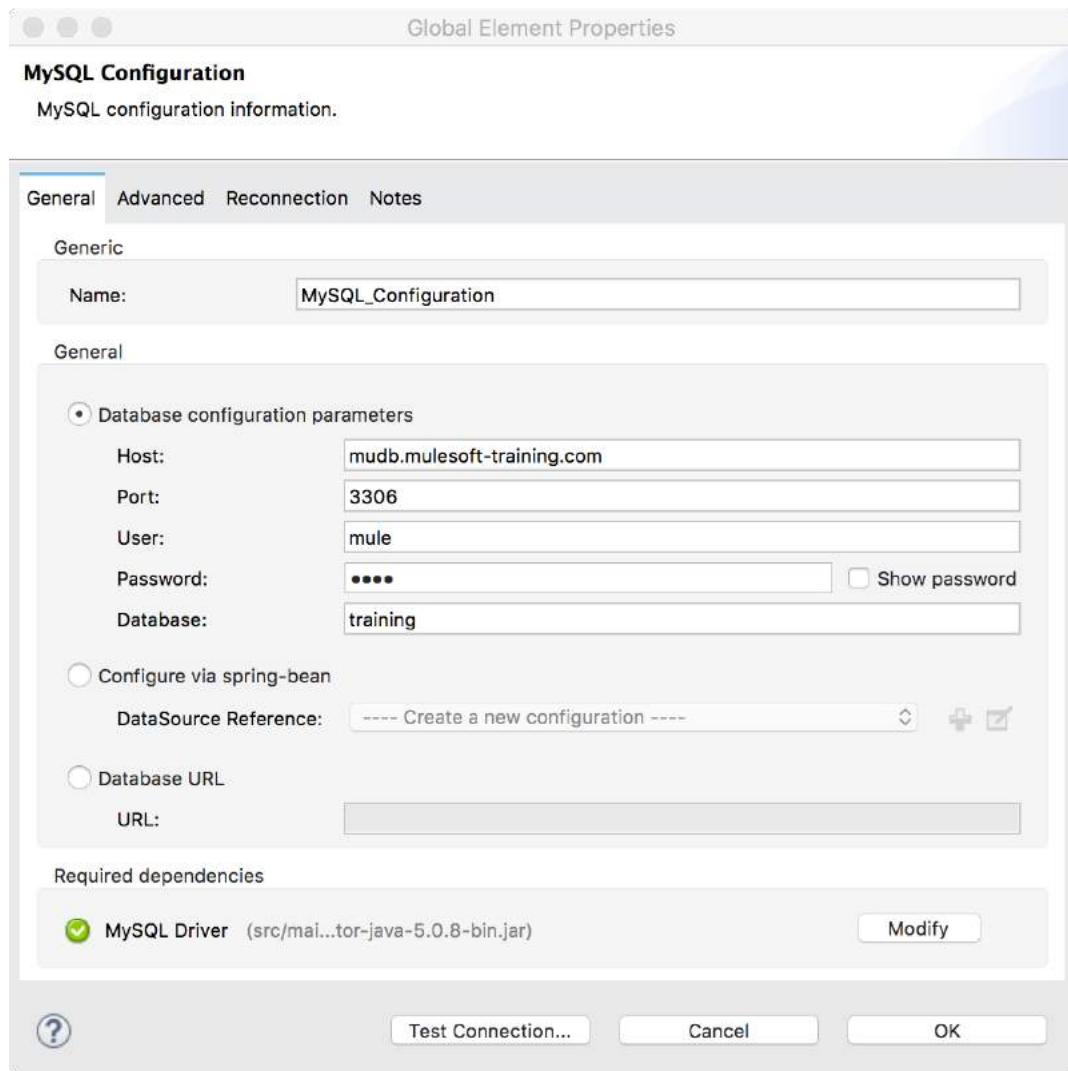
Option 1: Configure a MySQL Database connector (if you have access to port 3306)

7. In the Database properties view, click the Add button next to connector configuration.
8. In the Choose Global Type dialog box, select Connector Configuration > MySQL Configuration and click OK.



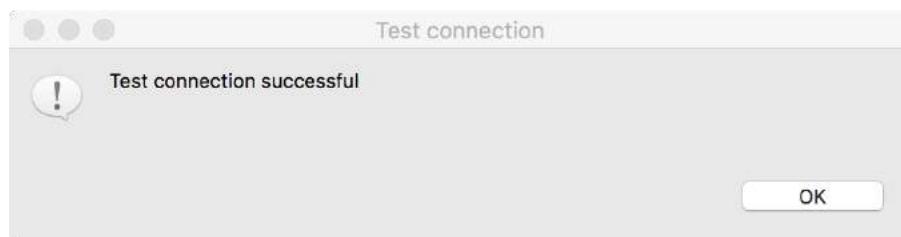
9. In the Global Element Properties dialog box, set the server, port, user, password, and database values to the values listed in the course snippets.txt file.
10. Under Required dependencies, click the Add File button next to MySQL Driver.

11. Navigate to the student files folder, select the MySQL JAR file located in the resources folder, and click Open.



12. 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.

13. Click OK to close the dialog box.
14. Click OK to close the Global Element Properties dialog box.

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

15. In a command-line interface, use the cd command to navigate to the folder containing the jars folder of the student files.
16. 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 1.3.0.

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
```

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

```
[MULESOFT TRAINING SERVICES]

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

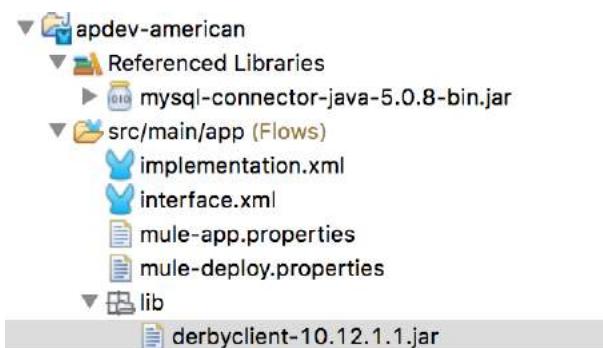
+-----+
| MuleSoft training services - Press CTRL-C to terminate this application |
|-----|
| Welcome page      : http://localhost:9090                           |
| Derby database URL : jdbc:derby://localhost:1527/memory:training       |
| JMS broker URL    : tcp://localhost:61616                            |
| JMS topic name    : opessentials                                     |
| United REST service: http://localhost:9090/essentials/united/flights/{destination} |
|                      NB: use http://0.0.0.0:9090/essentials/united in Mule apps |
| Delta SOAP service: http://localhost:9191/essentials/delta             |
| Delta SOAP WSDL     : http://localhost:9191/essentials/delta?wsdl        |
| Delta WS operations: listAllFlights, findFlights                   |
| Banking API        : http://localhost:9090/api/...                     |
| Banking API RAML   : http://localhost:9090/api/banking-service.raml    |
| Accounts form      : http://localhost:9090/essentials/accounts/show.html |
+-----+
```

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

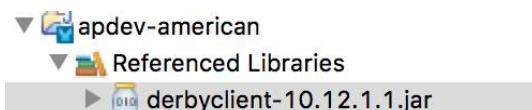
18. In the computer's file explorer, return to the student files folder and locate the derbyclient.jar file in the jars folder.



19. Copy and paste or drag this JAR file into the src/main/app/lib folder of the project in Anypoint Studio.



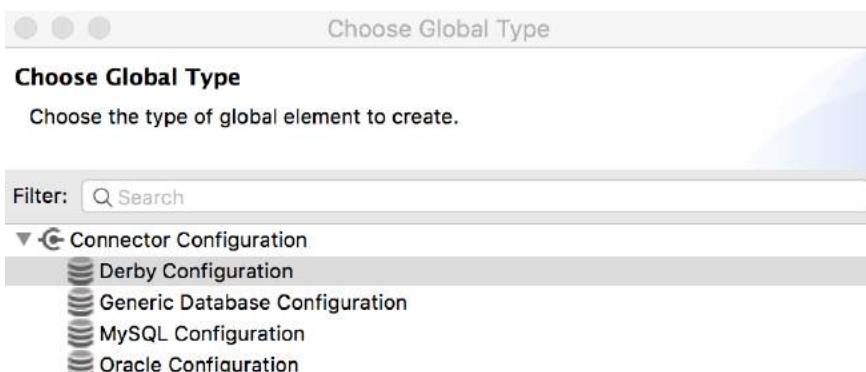
20. Right-click the JAR file and select Build Path > Add to Build Path; you should now see the JAR file in the project's Referenced Libraries.



21. Double-click the Database endpoint in the canvas.

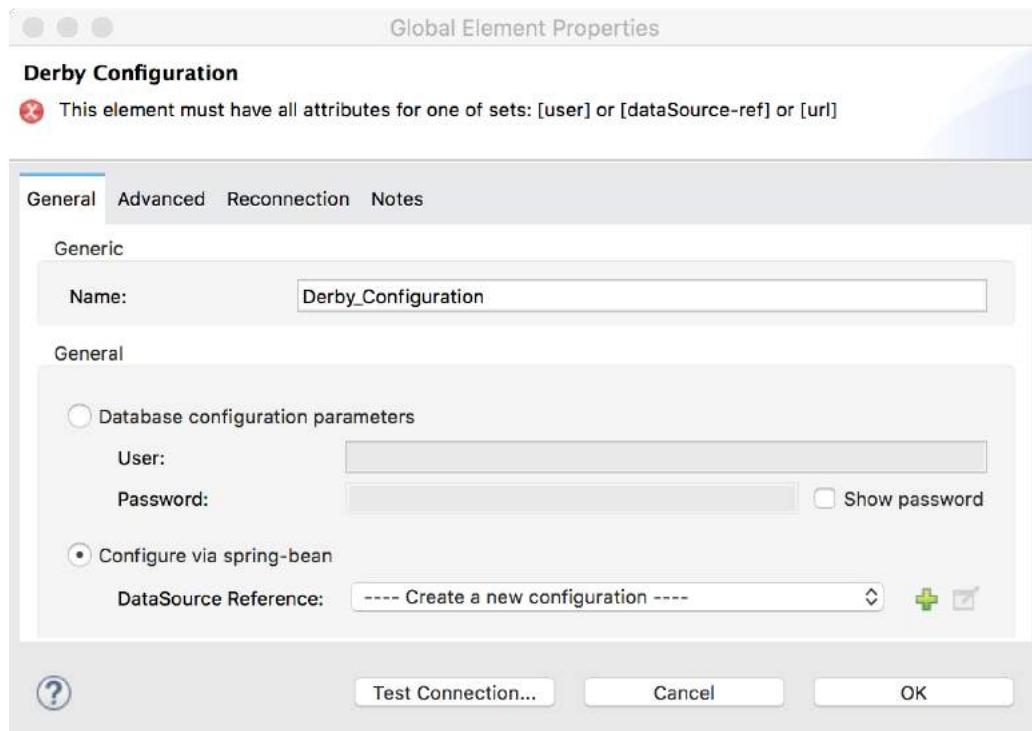
22. In the Database properties view, click the Add button next to connector configuration.

23. In the Choose Global Type dialog box, select Connector Configuration > Derby Configuration and click OK.

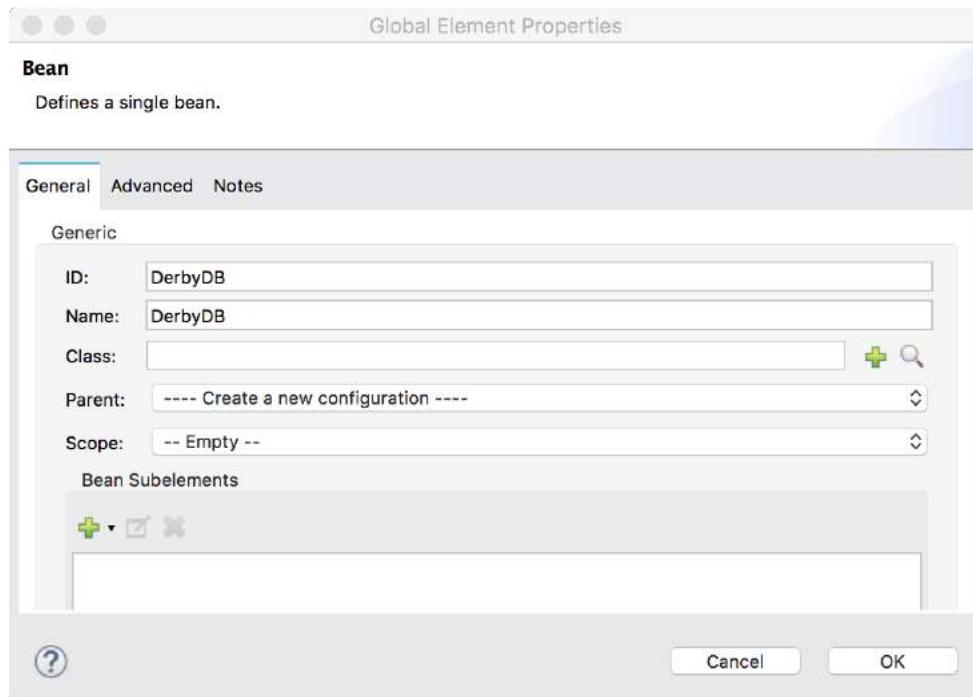


24. In the Global Element Properties dialog box, select Configure via spring-bean.

25. Click the Add button next to DataSource Reference.

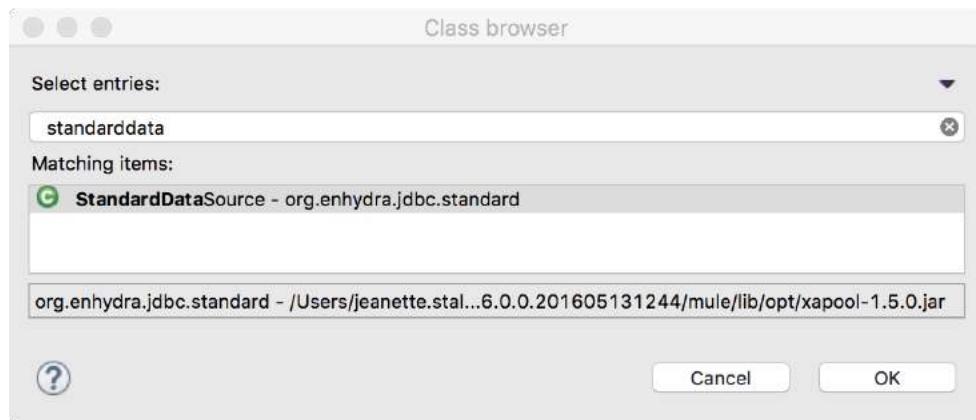


26. On the Bean page of the Global Element Properties dialog box, set the ID and name to DerbyDB.



27. Click the Browse for Java class button next to class.

28. In the Class browser dialog box, start typing StandardDataSource and select the matching item for StandardDataSource – org.enhydra.jdbc.standard.



29. Click OK.

30. On the Bean page of the Global Element Properties dialog box, click the Add button under Bean Subelements and select Add Property.

31. In the Property dialog box, set the following values:

- Name: driverName
- Value: org.apache.derby.jdbc.ClientDriver

Note: You can copy this value from the course snippets.txt file.

32. Click Finish.

33. Click the Add button under Bean Subelements again and select Add Property.

34. In the Property dialog box, set the following values:

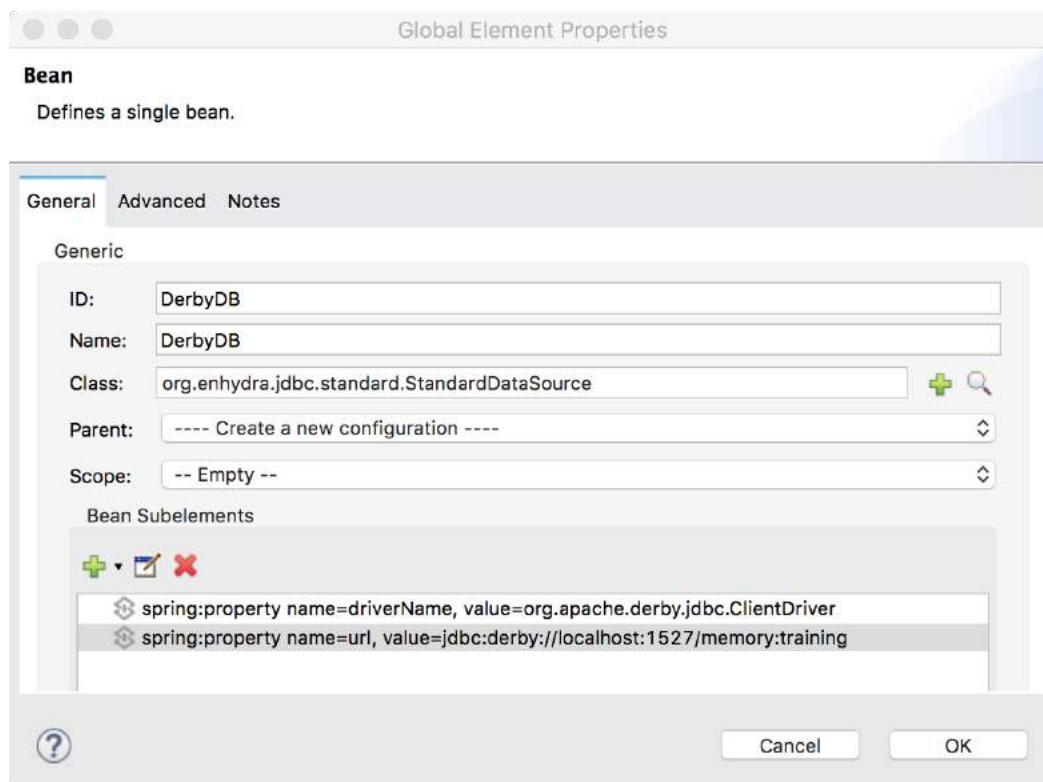
- Name: url
- Value: jdbc:derby://localhost:1527/memory:training

Note: You can copy this value from the course snippets.txt file.

Note: If you changed the database port when you started the mulesoft-training-services application, be sure to use the new value here.

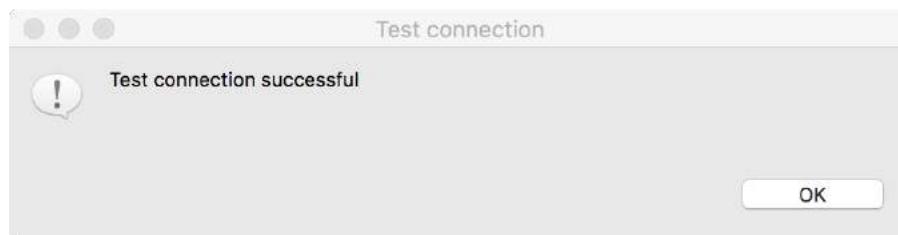
35. Click Finish.

36. On the Bean page of the Global Element Properties dialog box, click OK.



37. On the Derby Configuration page of the Global Element Properties dialog box, click Test Connection; you should get a successful test dialog box.

Note: Make sure the connection succeeds before proceeding.



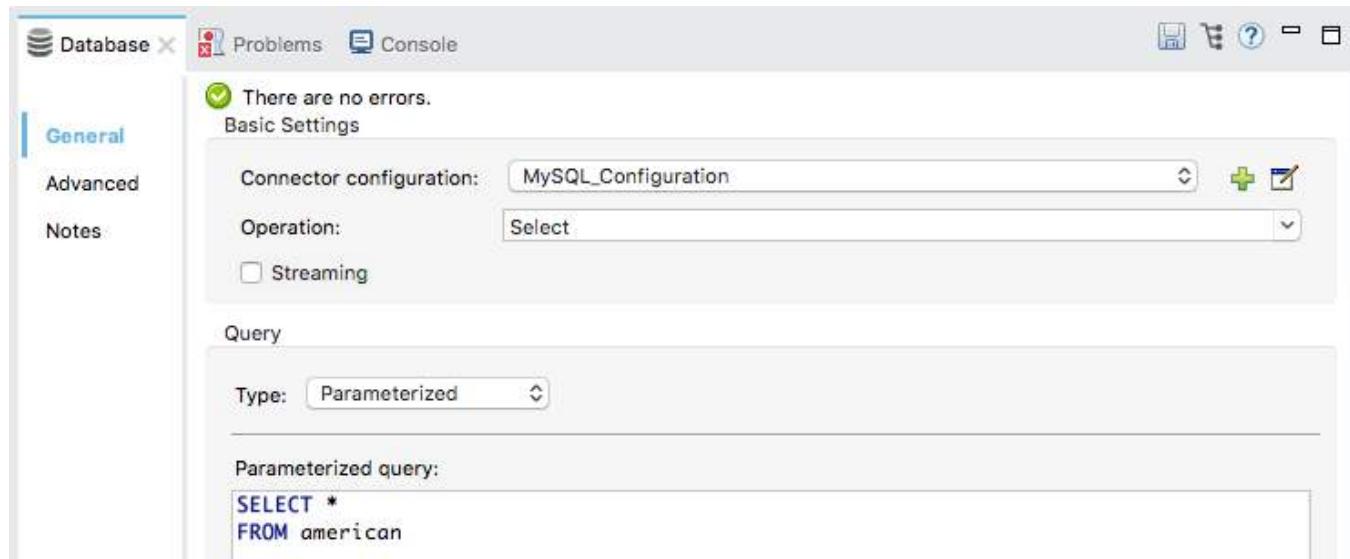
38. Click OK to close the dialog box.

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

Write a query to return all flights

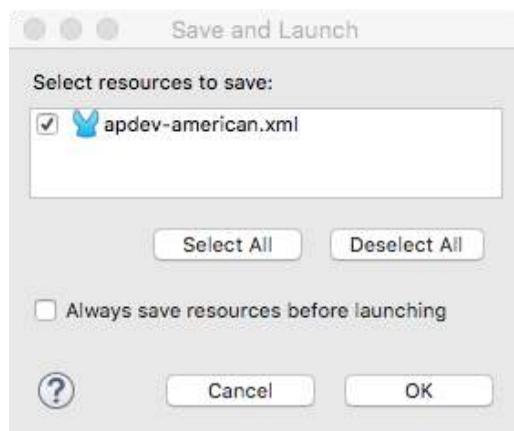
40. In the Database properties view, set the operation to Select.
41. Add a query to select all records from the american table.

```
SELECT *\nFROM american
```



Test the application

42. Run the project.
43. In the Save and launch dialog box, select Always save resources before launching and click OK.



44. Watch the console, and wait for the application to start.
45. Once it has started, return to Postman.
46. In Postman, make another request to <http://localhost:8081/flights>; you should get some garbled plain text displayed – the tool's best representation of Java objects.

POST <http://localhost:8081/flights> Params Send Save

Body Cookies Headers (2) Tests Status: 200 OK Time: 847 ms

Pretty Raw Preview HTML

```
1 sr java.util.LinkedList )S]J" xpw sr $org.mule.util  
.CaseInsensitiveHashMap@000gE0 xpw ?@ .....t planeType  
2 Boeing 787t code2t 0001t  
3 totalSeatssr java.lang.Integer ..0008 I valuexr java.lang.Number@000 000 xp ..@t  
toAirportt LAXt takeOffDatesr  
4 java.sql.Date @Fh?5f0 xr java.util  
.Datehj@ KYt xpw R^ $ xt fromAirportt MUAt pricesq ~
```

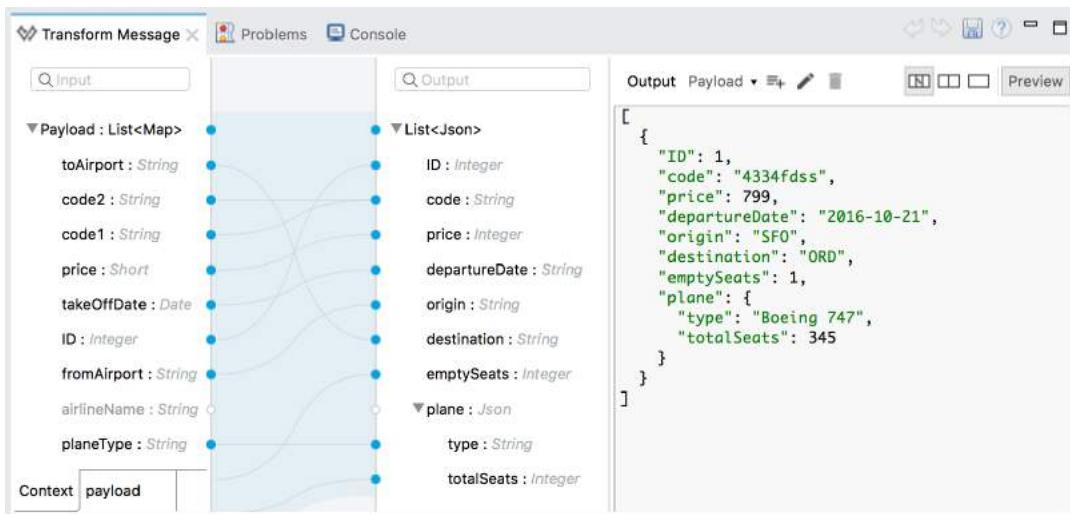
47. Return to Anypoint Studio.

48. Stop the project.

Walkthrough 3-3: Transform data

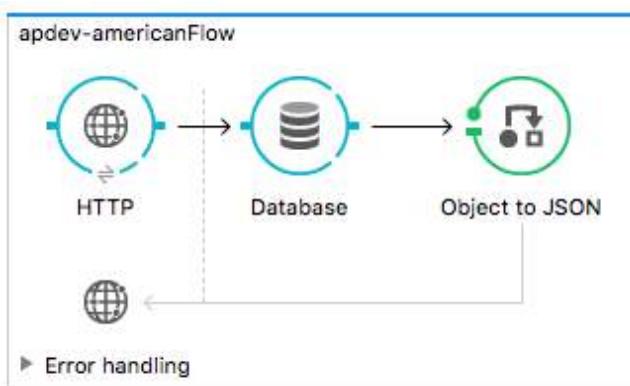
In this walkthrough, you transform and display the account data into JSON. You will:

- Use the Object to JSON transformer.
- Replace it with a Transform Message component.
- Use the DataWeave graphical editor to change the response to a different JSON structure.



Add an Object to JSON transformer

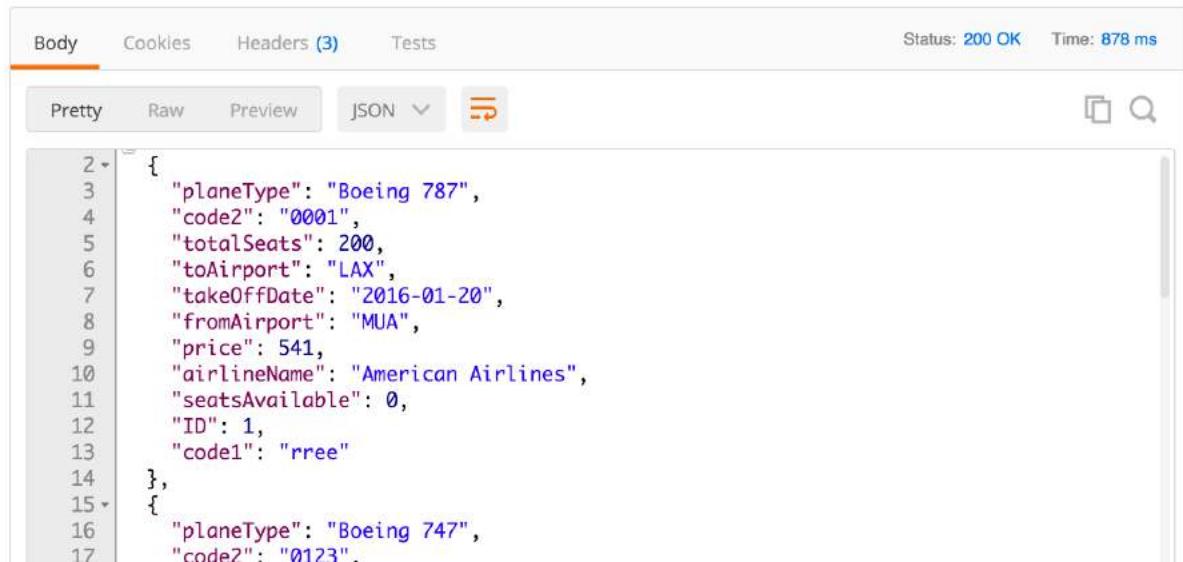
1. In the Mule Palette, select the Transformers tab.
2. Drag an Object to JSON transformer from the Mule Palette and drop it after the Database endpoint.



Test the application

3. Run the project.

- In Postman, send the same request; you should see the American flight data represented as JSON.



The screenshot shows the Postman interface with the following details:

- Body:** The tab is selected.
- Status:** 200 OK, Time: 878 ms
- JSON:** The response is displayed in a pretty-printed JSON format.
- Content:**

```

2  {
3   "planeType": "Boeing 787",
4   "code2": "0001",
5   "totalSeats": 200,
6   "toAirport": "LAX",
7   "takeOffDate": "2016-01-20",
8   "fromAirport": "MUA",
9   "price": 541,
10  "airlineName": "American Airlines",
11  "seatsAvailable": 0,
12  "ID": 1,
13  "code1": "rree"
14 },
15 {
16   "planeType": "Boeing 747",
17   "code2": "0123".

```

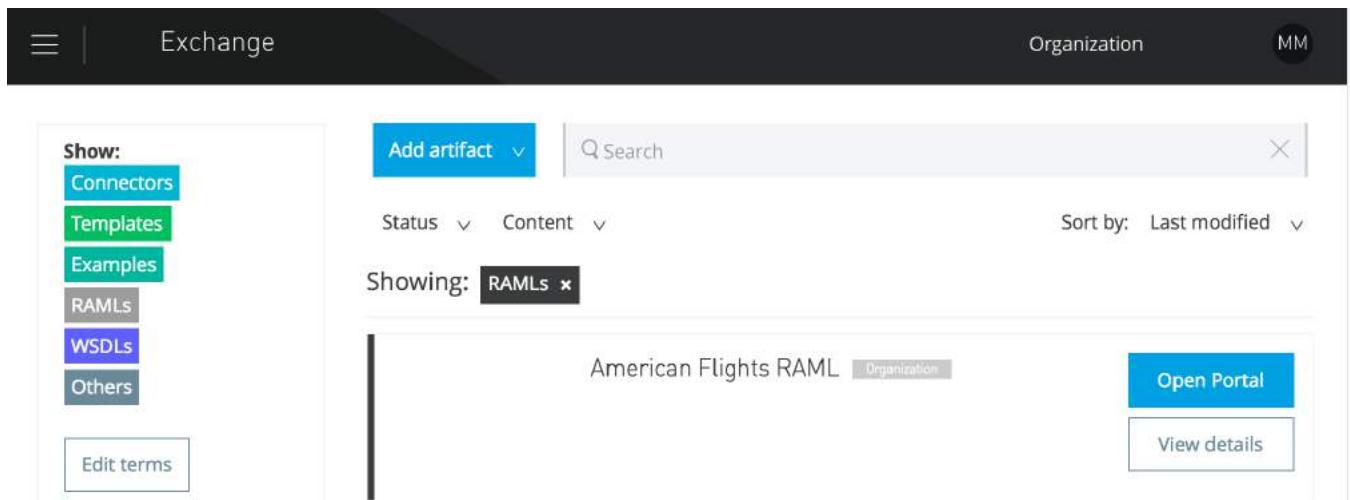
Note: If you are using the local Derby database, the properties will be uppercase instead.

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

- Return to your Anypoint Exchange open in a web browser.

Note: If you closed it, return to <http://anypoint.mulesoft.com> and from the main menu, select Exchange.

- Locate your American Flights RAML and click Open Portal.



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

- Header:** Exchange, Organization, MM
- Left Sidebar (Show dropdown):**
 - Connectors (selected)
 - Templates
 - Examples
 - RAMLs
 - WSDLs
 - Others
 - Edit terms
- Search Bar:** Add artifact, Search, X
- Filter Bar:** Status, Content, Sort by: Last modified
- Result Area:**
 - Showing: RAMLs
 - American Flights RAML (Organization)
 - Open Portal
 - View details

- Click the API reference link.

8. Look at the example data returned for the /flights get method.

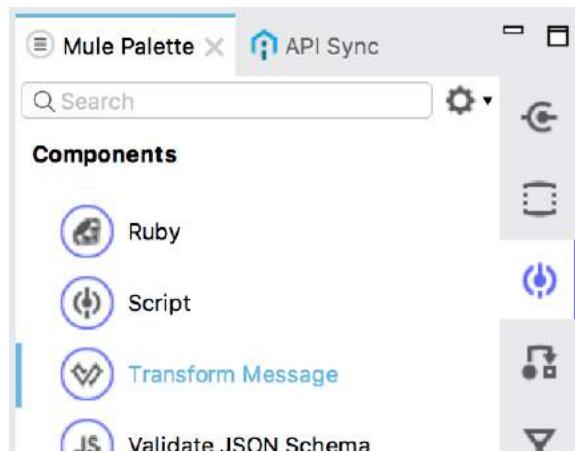
The screenshot shows the MuleSoft API Developer portal interface. The top navigation bar includes the MuleSoft logo, the text "MuleSoft // Dev | American Flights API 1.0", and a "Log in" button. Below the navigation, there are three tabs: "Developer portal", "American Flights API - 1.0", and "API reference". The "API reference" tab is selected. On the left, there's a sidebar with "Overview" and "API reference" sections. The main content area is titled "Response" and shows a green box indicating "STATUS 200" and "200". It also shows "Body application/json" and "Examples: Example". The example JSON is displayed in a code block:

```
[  
  {  
    "ID": 1,  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2016/03/20",  
    "origin": "MUA",  
    "destination": "SFO",  
    "emptySeats": 0,  
    "plane": {  
      "type": "Boeing 737",  
      "totalSeats": 150  
    }  
  },  
  {  
    "ID": 2,  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2016/03/20",  
    "origin": "MUA",  
    "destination": "SFO",  
    "emptySeats": 0,  
    "plane": {  
      "type": "Boeing 737",  
      "totalSeats": 150  
    }  
  }]
```

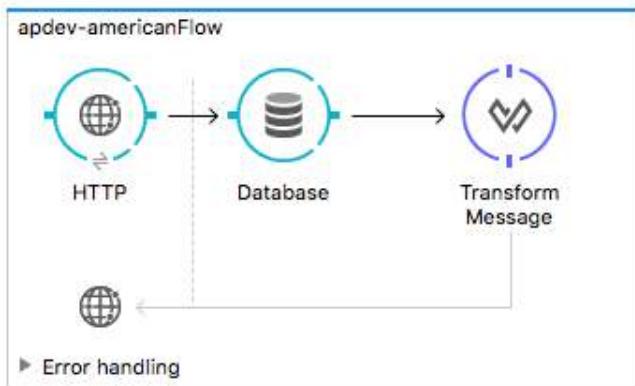
9. Notice that the structure of the JSON being returned by the Mule application does not match this JSON.

Add a Transform Message component

10. Return to Anypoint Studio and stop the project.
11. Right-click the Object to JSON transformer and select Delete.
12. In the Mule Palette, select the Components tab.



13. Drag a Transform Message component from the Mule Palette and drop it after the Database endpoint.



Review metadata for the transformation input

14. Double-click the Transform Message component in the canvas.
15. In the Transform Message properties view, look at the input section and review the payload metadata; it should match the data returned by the Database endpoint.

The screenshot shows the "Transform Message" properties view. The left panel displays the "Input" section, which lists fields such as "toAirport", "code2", "code1", "price", and "takeOffDate". The right panel shows the "Output" section, which includes a "Payload" tab displaying the following JSON code:

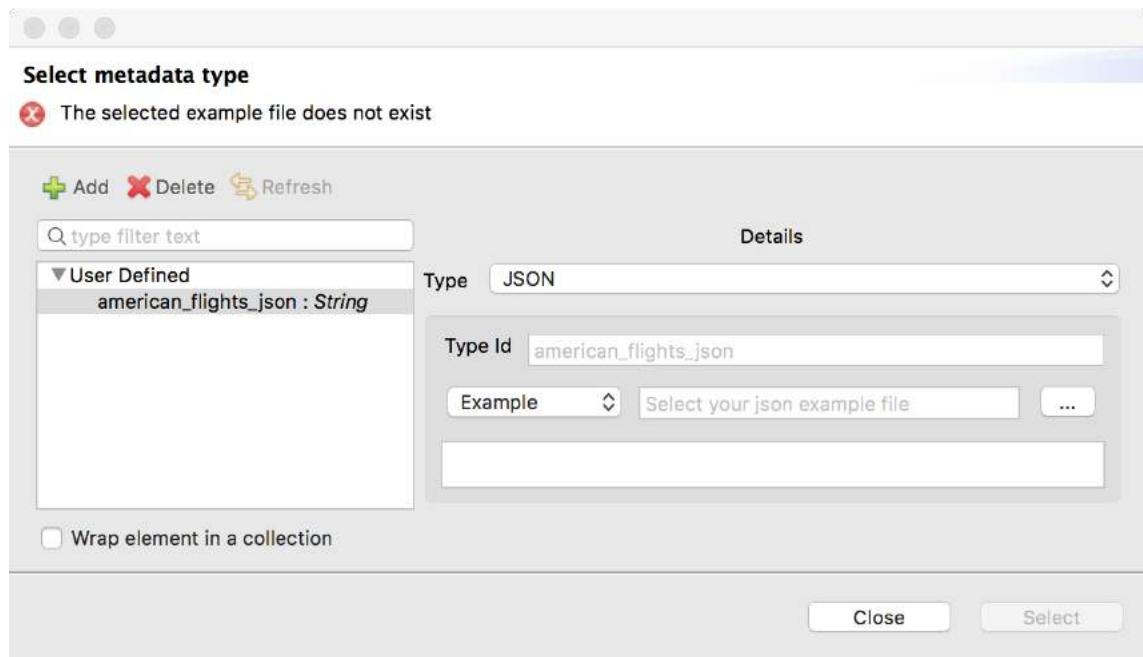
```
1 %dw 1.0
2 %output application/java
3 ---
4 {
5 }
```

Add metadata for the transformation output

16. Click the Define metadata link in the output section.
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.

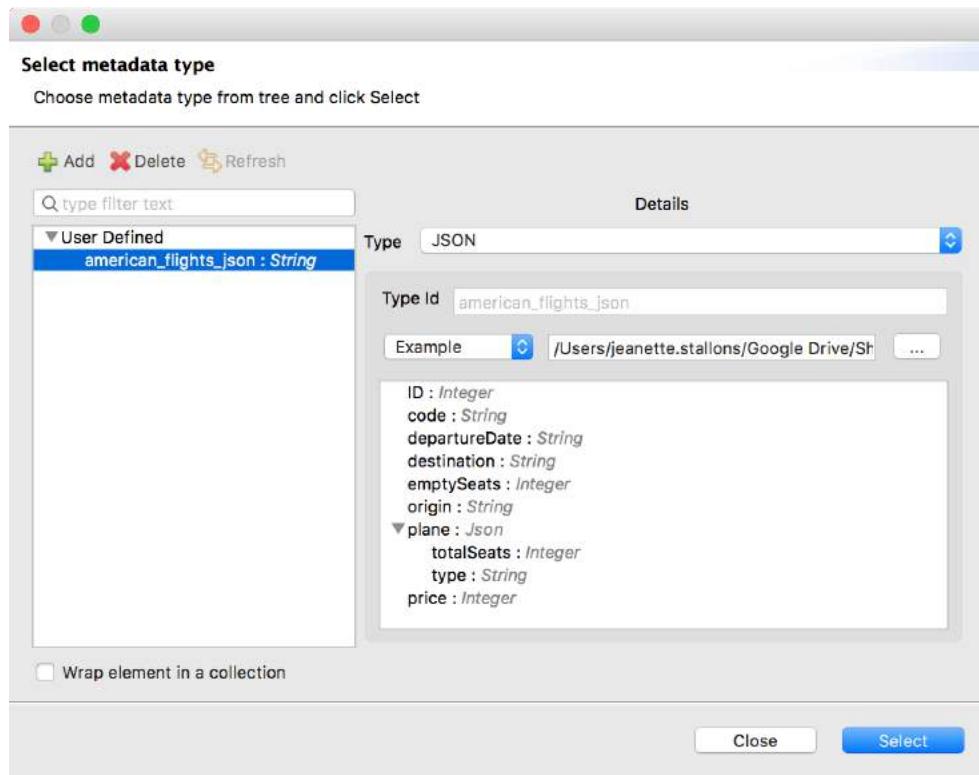
20. Back in the Set metadata type dialog box, set the type to JSON.

21. Change the Schema selection to Example.

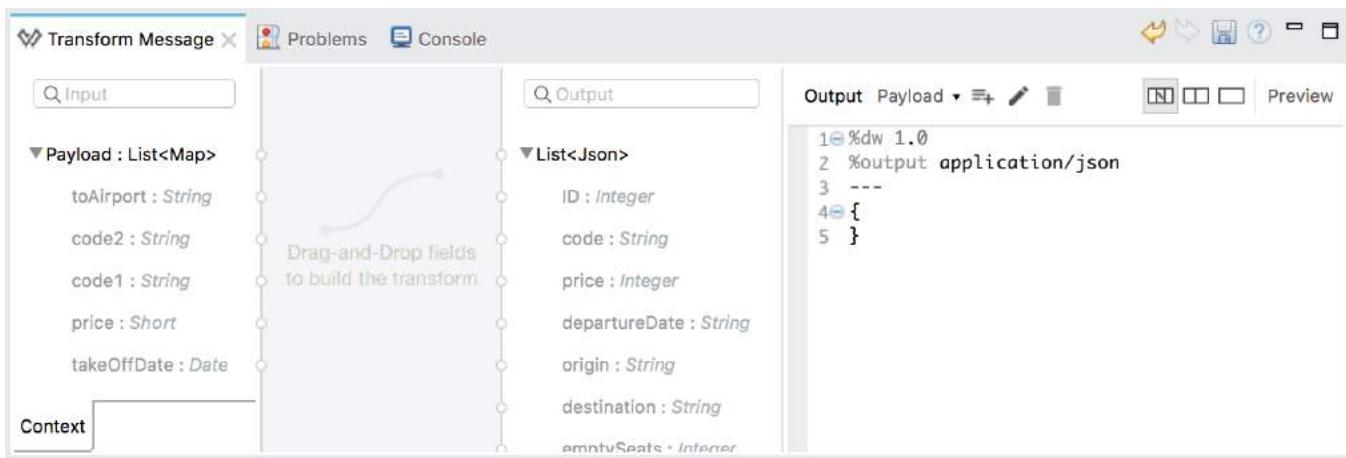


22. Click the browse button and navigate to the course student files.

23. Select american-flights-example.json in the schemas-and-examples folder and click Open; you should see the example data for the metadata type.



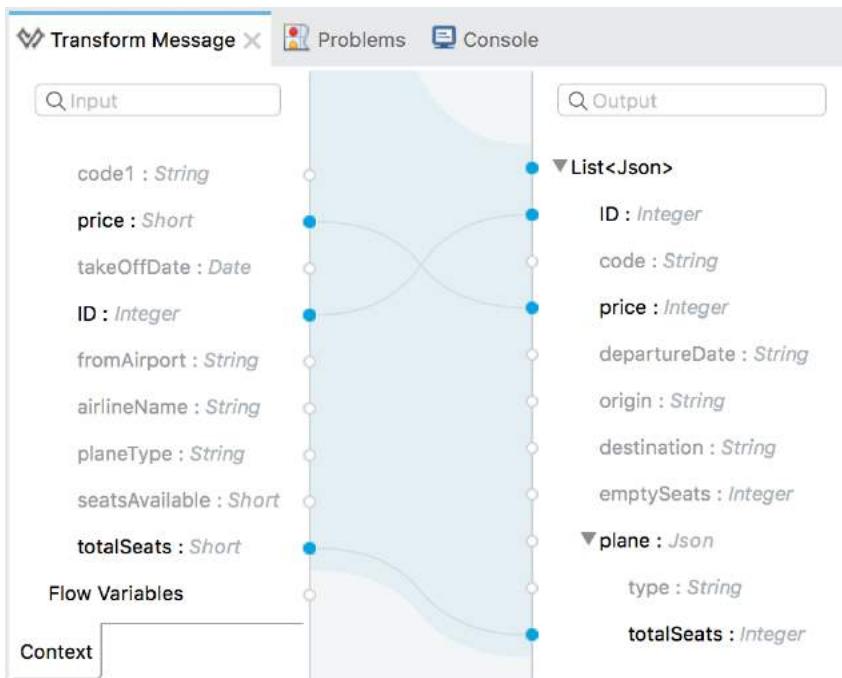
24. Click Select; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

25. 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

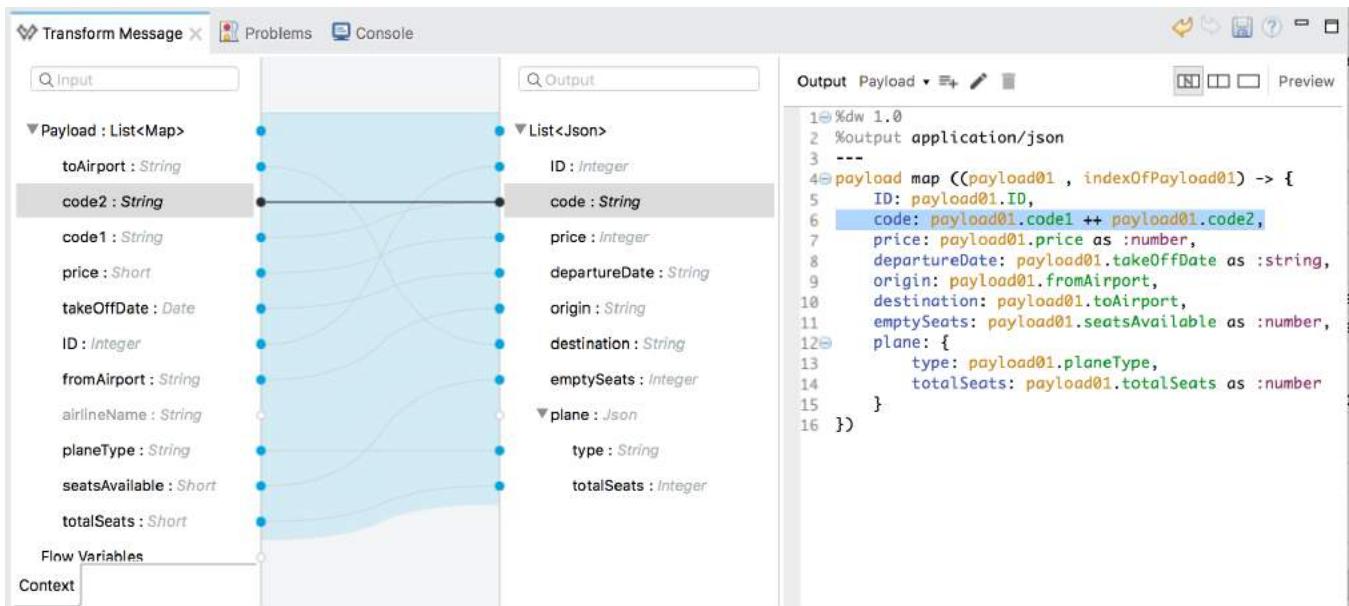


26. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

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

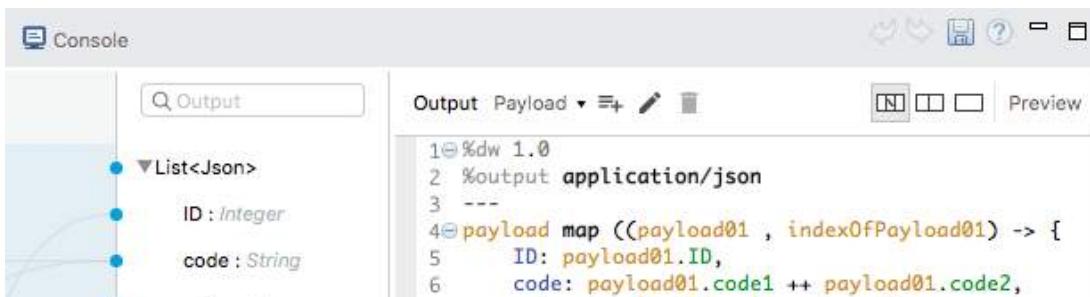
27. Concatenate two fields by dragging them from the input section and dropping them on the same field in the output section.

- code1 to code
- code2 to code



Add sample data

28. Click the Preview button in the output section.



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

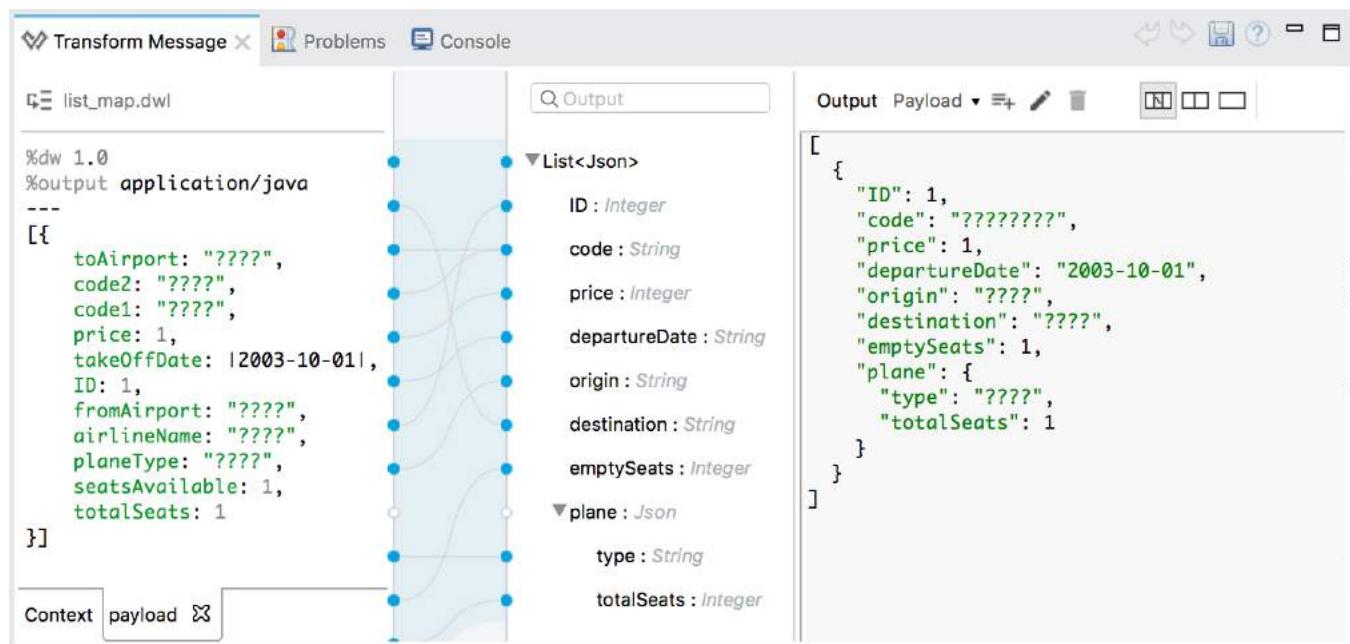
The screenshot shows the Mule Studio interface with the preview tab selected. At the top, there are tabs for 'Output' and 'Payload'. Below them are icons for copy, cut, and paste, followed by a 'Preview' button. The preview area contains the following code:

```
1 @%dw 1.0
2 %output application/json
3 ---
4 @payload map ((payload01 , indexOfPayload01) -> {
5     ID: payload01.ID,
6     code: payload01.code1 ++ payload01.code2,
7     price: payload01.price as :number,
8     departureDate: payload01.takeOffDate as :string
9     origin: payload01.fromAirport,
10    destination: payload01.toAirport,
11    emptySeats: payload01.seatsAvailable as :number
12 })
```

At the bottom of the preview area, there is a blue link: [Create required sample data to execute preview](#).

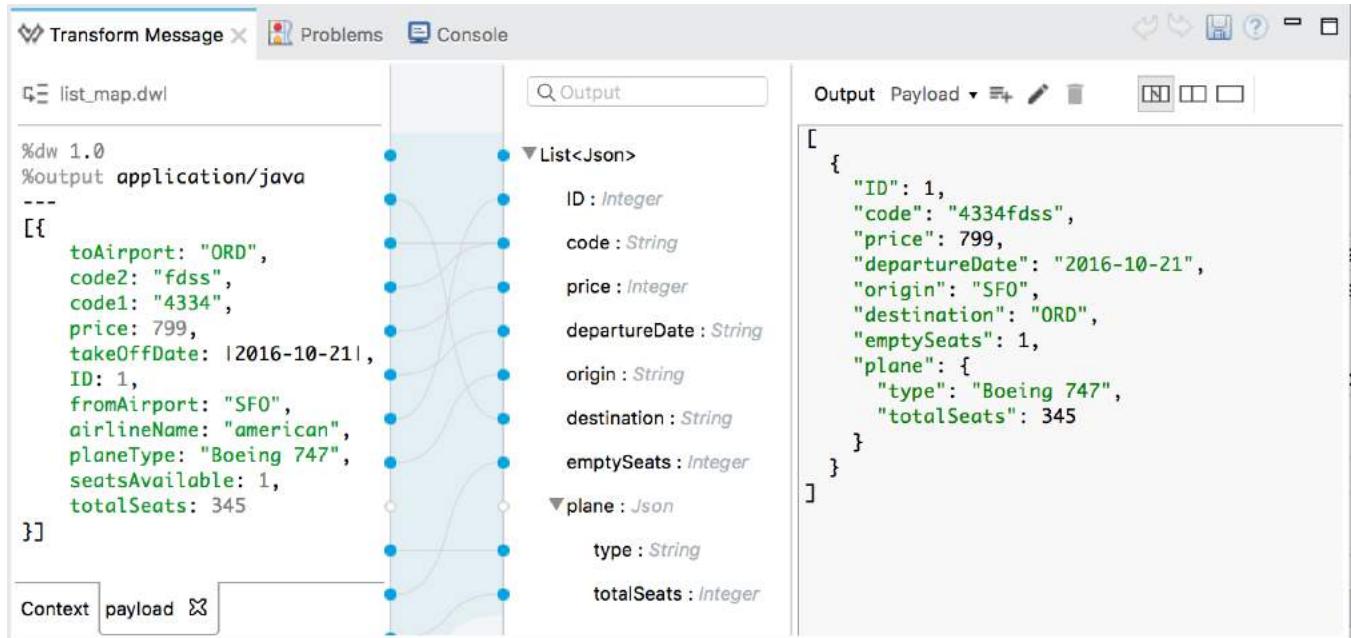
30. Look at the input section, you should see a new tab called payload with sample data generated from the input metadata.

31. Look at the output section, you should see a sample response for the transformation.



32. In the input section, replace all the ???? with sample values.

33. Look at the output section, you should see the sample values in the transformed data.



Test the application

34. Run the project.

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

Status: 200 OK Time: 1689 ms

Body

Pretty Raw Preview JSON

```
[ {
    "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": "peefd0123"
}]
```

Try to retrieve information about a specific flight

36. Add a URI parameter to the URL to make a request to <http://localhost:8081/flights/3>; you should get a 404 response with a resource not found message.

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

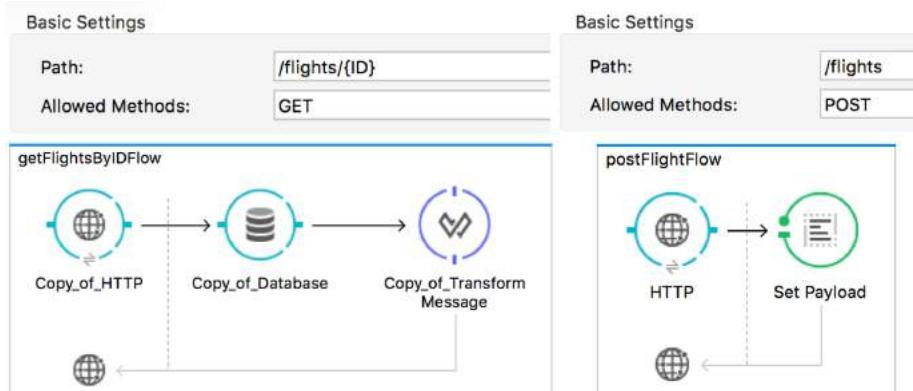
- Method: GET
- URL: localhost:8081/flights/3
- Params: None
- Send button
- Save button
- Body tab selected (highlighted in orange)
- Cookies, Headers (2), Tests tabs
- Status: 404 No listener for endpoint: /flights/3
- Time: 10 ms
- Pretty, Raw, Preview buttons
- HTML dropdown
- Copy icon
- Search icon
- Response body: i 1 | Resource not found.

37. Return to Anypoint Studio.
38. Look at the console; you should get a no listener found for request (GET)/flights/3.
39. Stop the project.

Walkthrough 3-4: Create a RESTful interface for a Mule application

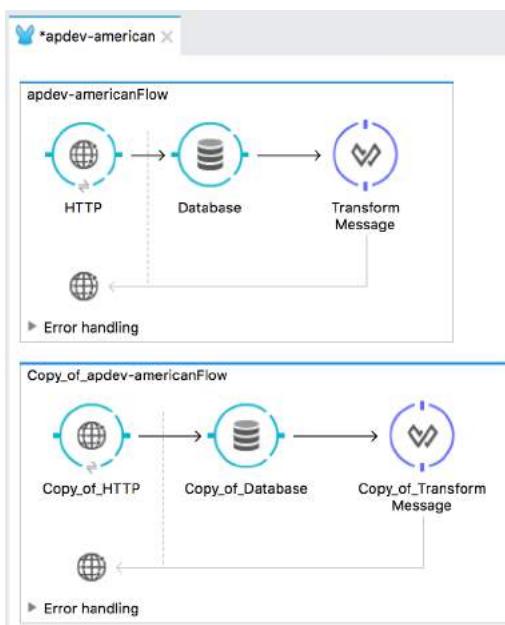
In this walkthrough, you continue to create a RESTful interface for the application. You will:

- Route based on path.
- Add a URI parameter to a new HTTP Listener endpoint path.
- Route based on HTTP method.



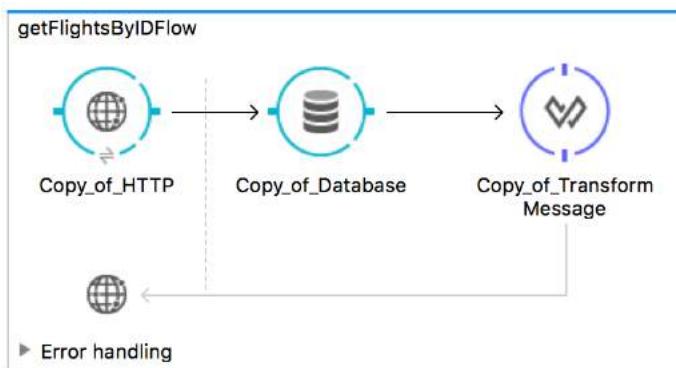
Make a copy of the existing flow

1. Return to apdev-american.xml.
2. Click the flow in the canvas to select it.
3. From the main menu bar, select Edit > Copy.
4. Click in the canvas beneath the flow and select Edit > Paste.



Rename the flows

5. Double-click the name of the first flow.
6. In the Properties view, change its name to getFlightsFlow.
7. Change the name of the second flow to getFlightsByIDFlow.



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

Specify a URI parameter for the new HTTP Listener endpoint

8. Double-click the HTTP Listener endpoint in getFlightsByIDFlow.
9. Change the path to have a URI parameter called ID.

Basic Settings	
Path:	/flights/{ID}
Allowed Methods:	GET

Modify the Database endpoint

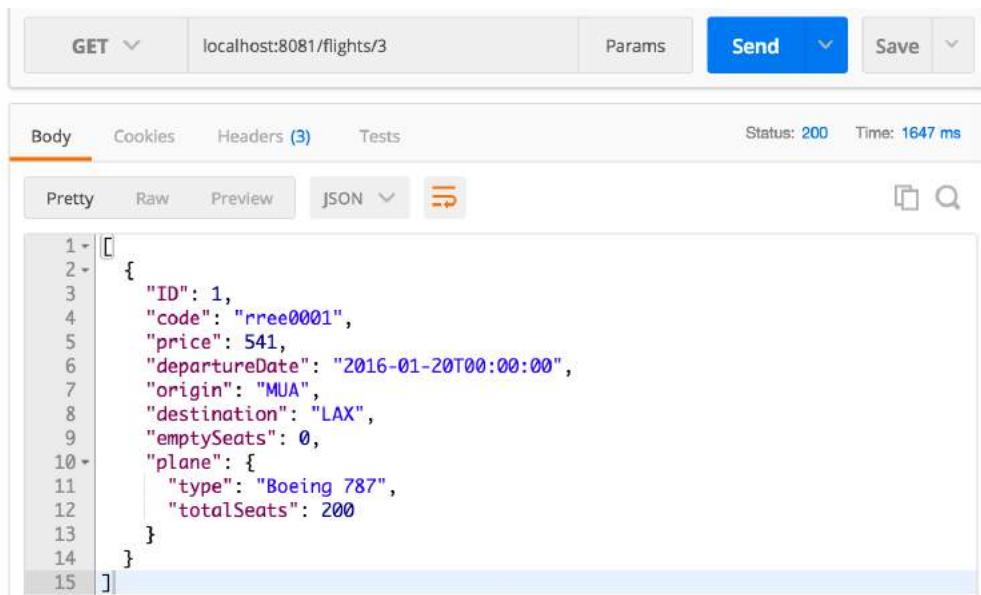
10. Double-click the Database endpoint in getFlightsByIDFlow.
11. Modify the query WHERE clause, to select flights with the ID equal to 1.

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

Test the application

12. Run the project.

13. In Postman, 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 Postman interface with a successful HTTP request. The method is set to GET, the URL is localhost:8081/flights/3, and the response status is 200 with a time of 1647 ms. The Body tab is selected, displaying the JSON response:

```
1 - [0
2 - {
3 -   "ID": 1,
4 -   "code": "rree0001",
5 -   "price": 541,
6 -   "departureDate": "2016-01-20T00:00:00",
7 -   "origin": "MUA",
8 -   "destination": "LAX",
9 -   "emptySeats": 0,
10 -  "plane": {
11 -    "type": "Boeing 787",
12 -    "totalSeats": 200
13 -  }
14 - }
15 - ]]
```

Modify the database query to use the URI parameter

14. Return to the course snippets.txt file and copy the SQL expression for American Flights API.
15. Return to Anypoint Studio and stop the project.
16. In the Database properties view, replace the existing WHERE clause with the value you copied.

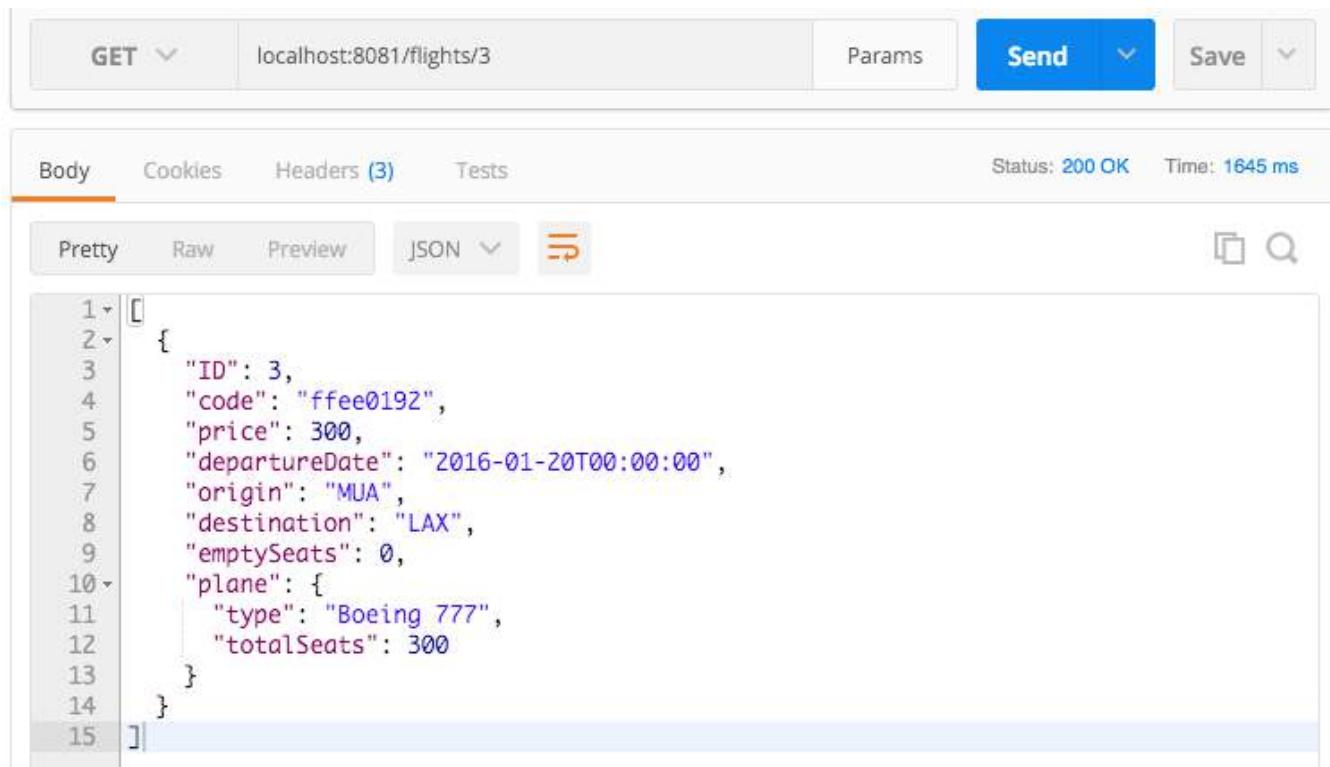
```
SELECT *
FROM american
WHERE ID = #[message.inboundProperties.'http.uri.params'.ID]
```

Note: You learn about reading and writing properties and variables in a later module in the developer courses.

Test the application

17. Run the project.

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



The screenshot shows the Postman interface with a successful HTTP request. The URL is set to `localhost:8081/flights/3`. The response status is `200 OK` and the time taken is `1845 ms`. The `Body` tab is selected, displaying the JSON response:

```
1 [ ]  
2 {  
3   "ID": 3,  
4   "code": "ffee0192",  
5   "price": 300,  
6   "departureDate": "2016-01-20T00:00:00",  
7   "origin": "MUA",  
8   "destination": "LAX",  
9   "emptySeats": 0,  
10  "plane": {  
11    "type": "Boeing 777",  
12    "totalSeats": 300  
13  }  
14 }  
15 ]
```

19. Return to Anypoint Studio and stop the project.

Make a new flow to handle post requests

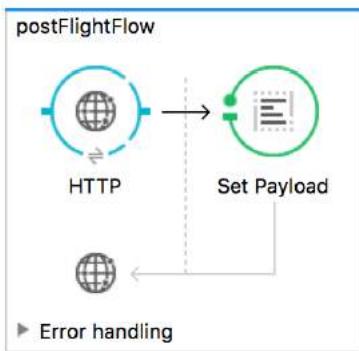
20. In the Mule Palette, select the Connectors tab.
21. Drag out an HTTP connector from the Mule Palette and drop it in the canvas below the two existing flows.
22. Change the name of the flow to `postFlightFlow`.
23. Double-click the HTTP Listener endpoint.
24. In the HTTP properties view, set the path to `/flights` and the allowed methods to POST.



The screenshot shows the `Basic Settings` section of the Mule Studio Properties view for an `HTTP` endpoint. The `Path` is set to `/flights` and the `Allowed Methods` are set to `POST`.

25. In the Mule Palette, select the Transformers tab.

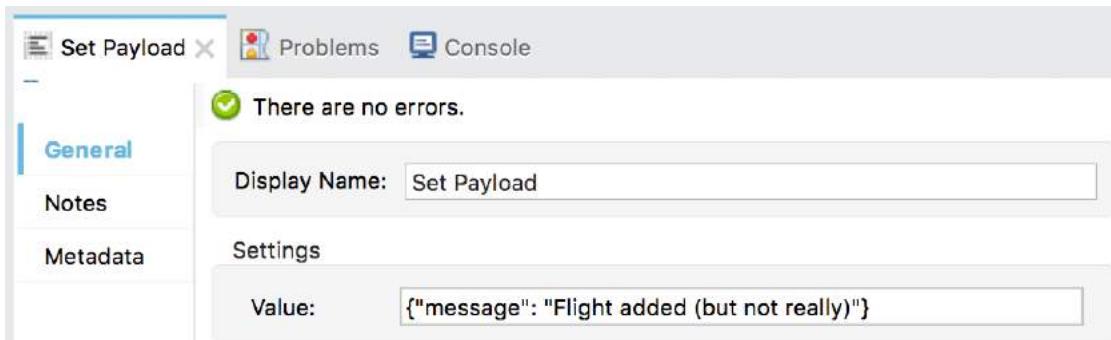
26. Drag out a Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



27. Double-click the Set Payload processor.
28. Return to the course snippets.txt file and copy the Module 2 American Flights API - /flights post response example.

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

29. 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

30. Run the project.
31. In Postman, change the request type from GET to POST.
32. Remove the URI parameter from the request URL: <http://localhost:8081/flights>.

33. Send the request; you should now see the message the flight was added – even though you did not send any flight data to add.

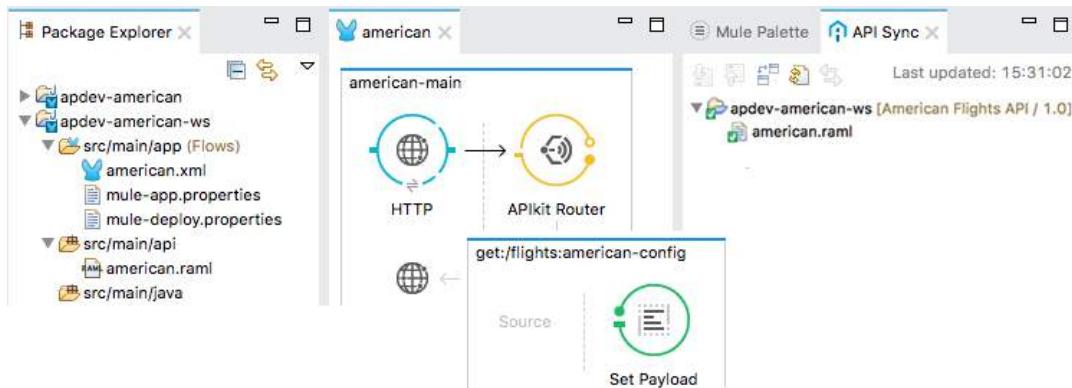
The screenshot shows a REST client interface. At the top, there are buttons for 'POST' (selected), 'http://localhost:8081/flights/' (the URL), 'Params' (button), 'Send' (button), and 'Save' (button). Below the URL, there are tabs for 'Body' (selected), 'Cookies', 'Headers (2)', and 'Tests'. On the right, it says 'Status: 200 OK' and 'Time: 156 ms'. Under the 'Body' tab, there are buttons for 'Pretty' (selected), 'Raw', 'Preview', 'HTML' (dropdown), and icons for copy and search. The main body area contains the JSON response: `i 1 | {"message": "Flight added (but not really)"}|`.

34. Return to Anypoint Studio and stop the project.

Walkthrough 3-5: Use Anypoint Studio to create a RESTful API interface from a RAML file

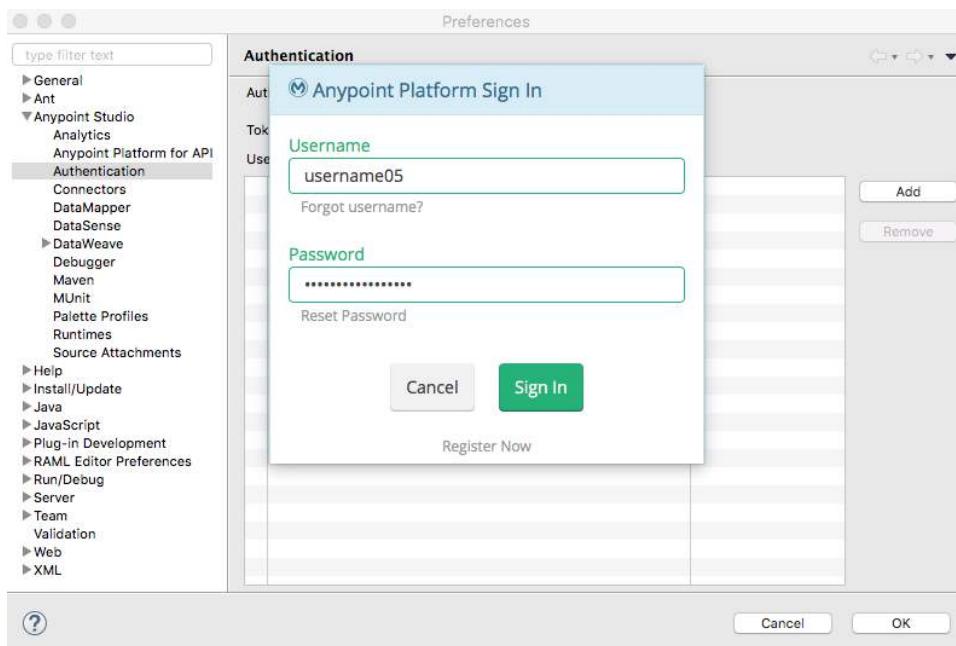
In this walkthrough, you generate a RESTful interface from the RAML file. You will:

- Add Anypoint Platform credentials to Anypoint Studio.
- Add a RAML file from Anypoint Platform to an Anypoint Studio project.
- Use Anypoint Studio and APIkit to generate a RESTful web service interface from a RAML file.
- Test the web service in the APIkit Consoles view and Postman.



Add Anypoint Platform credentials to Anypoint Studio

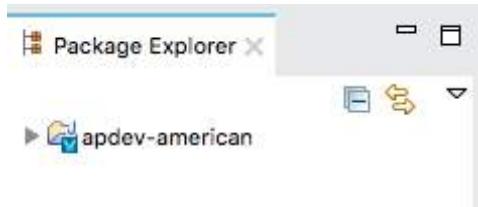
1. In Anypoint Studio, right-click apdev-american and select Anypoint Platform > Configure.
2. In the Authentication page of the Preferences dialog box, click the Add button.
3. In the Anypoint Platform Sign In dialog box, enter your username & password and click Sign In.



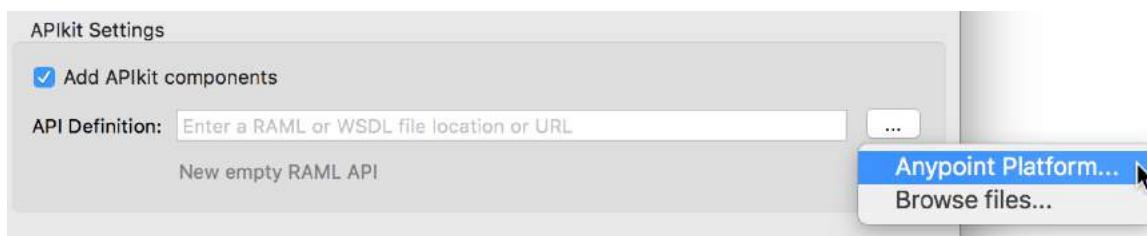
4. On the Authentication page, make sure your username is listed and selected.
5. Click OK.

Create a new project

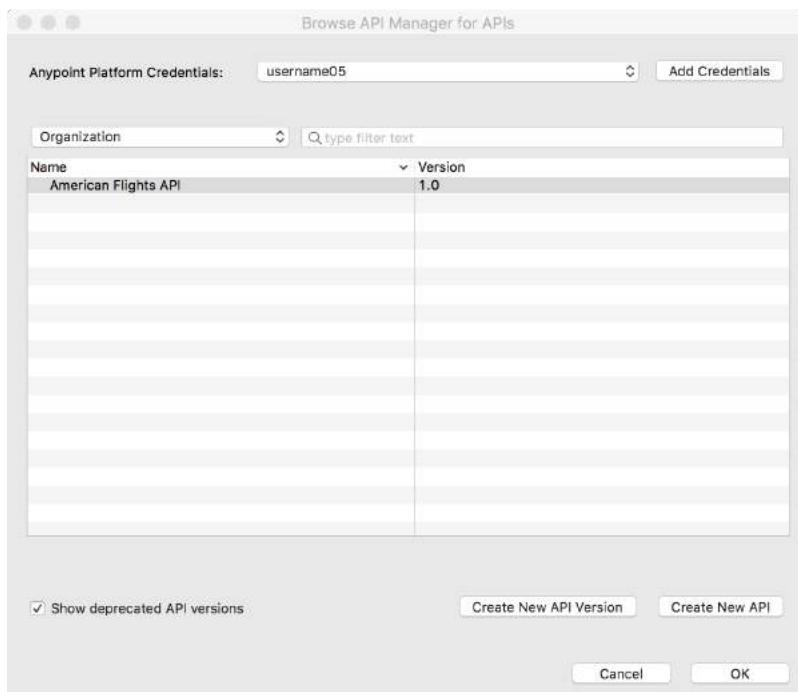
6. In the Package Explorer, click the arrow to the left of the apdev-american project to collapse it.



7. Right-click in the Package Explorer and select New > Mule Project.
8. In the New Mule Project dialog box, set the project name to apdev-american-ws.
9. Check Add APIkit components.
10. Click the browse button next to API Definition and select Anypoint Platform.



11. In the Browse API Manager for APIs dialog box, select the American Flights API and click OK.



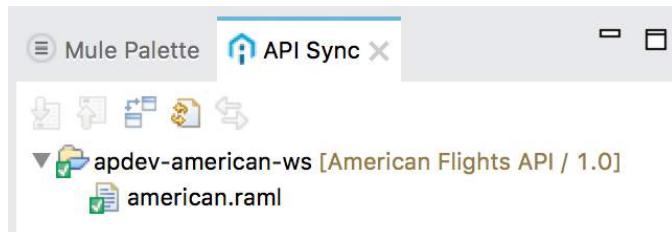
12. In the New Mule Project dialog box, click Finish.

Note: You are creating a new project because in Anypoint Studio 6.0.0, you cannot connect an existing project to an API on Anypoint Platform. You have to make this connection when you create the project. This will be possible in a future release.

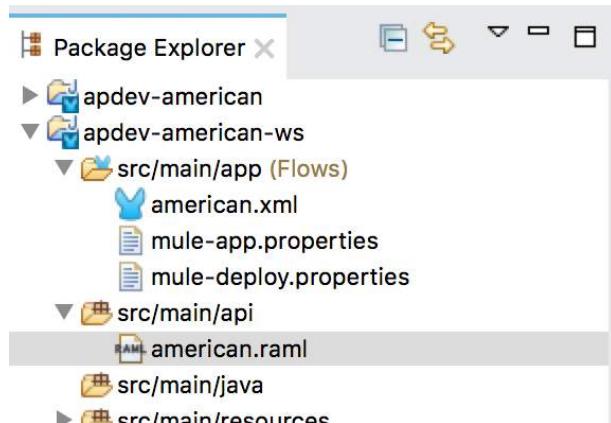
Locate the RAML file added to the project

13. Click the API Sync tab next to Mule Palette tab.

14. Expand the apdev-american-ws folder; you should see the API and the RAML file it contains.



15. In the Package Explorer, locate and expand the project's src/main/api folder; it should contain american.raml.



Note: If you did not successfully create the RAML file earlier, you can use the one included in the solution folder of the student files. Open it in a text editor, copy all the code, open american.raml in Anypoint Studio, replace the existing code, save the file, then click the Upload button in the API Sync view to upload it to your Anypoint Platform account.

Examine the XML file created

16. In american.xml, locate the following three flows:

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

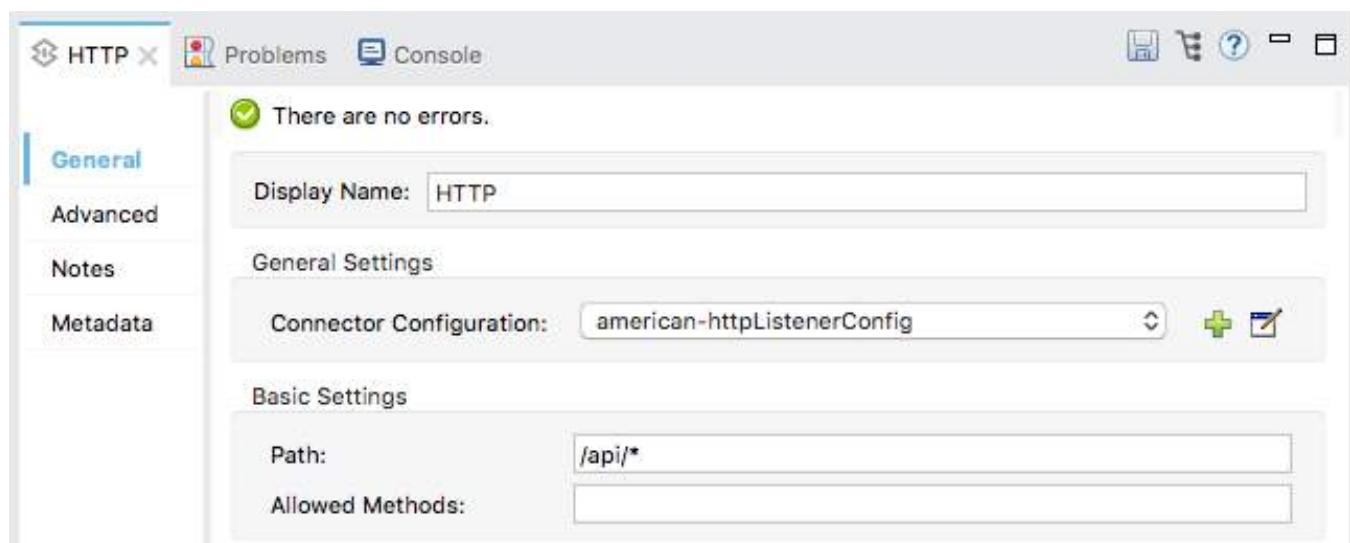


Examine the new HTTP Listener endpoint

17. Double-click the HTTP Listener endpoint in the american-main flow.

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

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

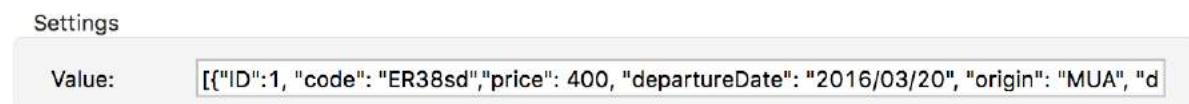


Look at the generated resource flow

19. Look at the get:/flights flow.



20. Double-click the Set Payload transformer and look at the value in the Set Payload properties view.



Test the web service in the APIkit Consoles view

21. Run the project.

22. Look at the APIkit Consoles view that opens.

The screenshot shows the Mule Studio interface with the 'api APikit Consoles (apdev-american-ws)' tab selected. The main area displays the 'American Flights API' resources. Under the 'Resources' section, there is a tree view with a single node: '/flights'. Below this, under '/flights', there are two operations: 'POST' and 'GET'. Further down, under '/flights/{ID}', there is another 'GET' operation. The 'POST' and 'GET' buttons are green, while the other 'GET' button is blue.

Note: If the API is not displayed in the APIkit Console, change your HTTP Listener Configuration host from 0.0.0.0 to localhost and then run the project again.

23. Click the GET button for /flights.
24. Click the Try It button.
25. Click in the code query parameter field and select one of the enumerated values, like LAX.
26. Click the GET button; you should get a 200 response with the example flight data.

The screenshot shows the 'Response' details for a flight search. On the left, the 'Code' field contains the JSON response:

```
[{"ID": 1, "code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 2, "code": "ER45if", "price": 345.99, "departureDate": "2016/02/11", "origin": "MUA", "destination": "LAX", "emptySeats": 52, "plane": {"type": "Boeing 777", "totalSeats": 300}}]
```

 On the right, the 'Status' field shows '200'. The 'Headers' section includes 'content-length: 364', 'content-type: application/json', and 'date: Wed, 01 Jun 2016 17:53:23 GMT'. The 'Body' section shows the same JSON response as the 'Code' field.

Test the web service in Postman

27. In Postman, make a GET request to <http://localhost:8081/flights>; you should get a 404 response with a message that the resource was not found because there is no longer a listener for that endpoint.

The screenshot shows the Postman interface. The top bar has 'GET' selected, the URL is 'http://localhost:8081/flights', and the 'Send' button is highlighted. Below the URL, tabs for 'Body', 'Cookies', 'Headers (2)', and 'Tests' are visible. The status bar indicates 'Status: 404 No listener for endpoint: /flights' and 'Time: 15 ms'. The 'Body' tab is active, showing a single line of text: 'i 1 Resource not found.'

28. Change the URL to <http://localhost:8081/api/flights> and send the request; you should see the example data returned.

Note: Be sure the method is set to GET.

The screenshot shows the Postman interface. The top bar has 'GET' selected, the URL is 'http://localhost:8081/api/flights', and the 'Send' button is highlighted. Below the URL, tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests' are visible. The status bar indicates 'Status: 200 OK' and 'Time: 14 ms'. The 'Body' tab is active, showing a JSON response with line numbers on the left. The JSON data is as follows:

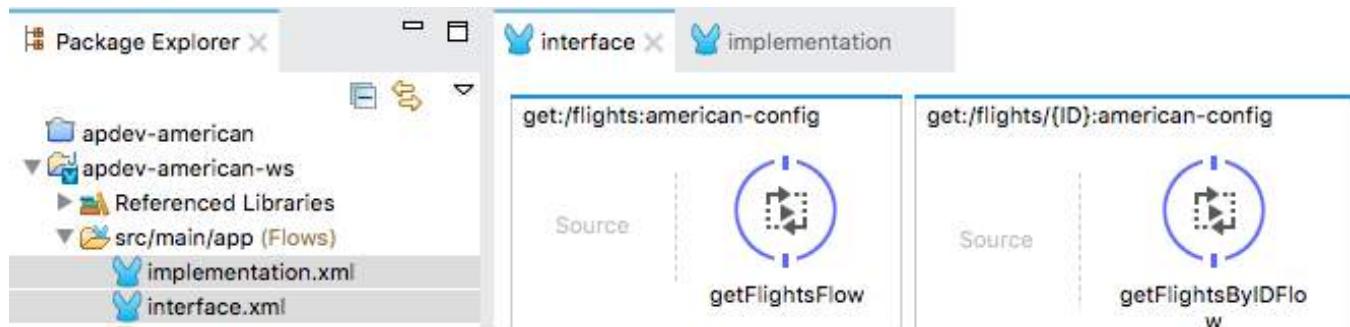
```
1  [
2   {
3     "ID": 1,
4     "code": "ER38sd",
5     "price": 400,
6     "departureDate": "2016/03/20",
7     "origin": "MUA",
8     "destination": "SFO",
9     "emptySeats": 0,
10    "plane": {
11      "type": "Boeing 737",
12      "totalSeats": 150
13    }
14  },
15  {
16    "ID": 2,
17    "code": "ER45if",
18    "price": 345.99
}
```

29. Make a request to <http://localhost:8081/api/flights/3>; you should see the example data returned.
30. Return to Anypoint Studio and stop the project.

Walkthrough 3-6: Implement a RESTful web service

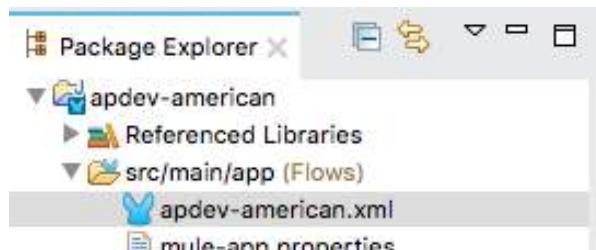
In this walkthrough, you wire the RESTful web service interface up to your back-end logic. You will:

- Pass a message from one flow to another.
- Create new logic for the nested resource call.
- Call the backend flows.
- Test the web service in the APIkit Consoles view and Postman.

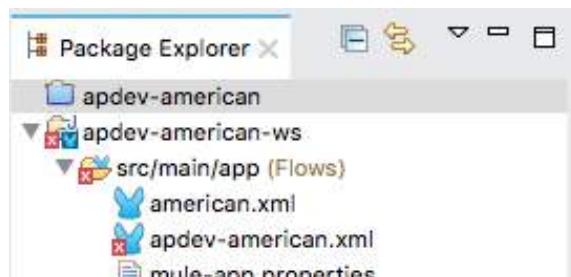


Add implementation code to the new project

1. Return to Anypoint Studio.
2. Expand the apdev-american project.
3. Right-click apdev-american.xml and select Copy.



4. In the apdev-american-ws project, right-click the src/main/app folder and select Paste.
5. Right-click the apdev-american project and select Close Project.



6. Open apdev-american.xml in the apdev-american-ws project.
7. Click the Global Elements link at the bottom of the canvas.

- Double-click the MySQL Configuration.

Type	Name	
MySQL Configuration (Configured)	MySQL_Configuration	Create
HTTP Listener Configuration (Configured)	HTTP_Listener_Configuration	Edit
		Delete

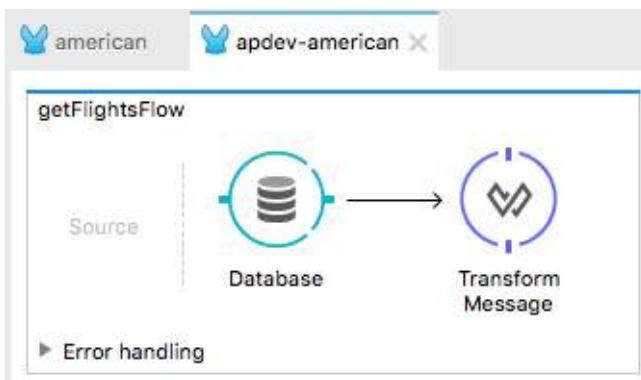
- In the Global Element Properties dialog box, click the Add File button next to MySQL Driver.
- Browse to and select the mysql.jar file located in the resources folder in the course student files.
- Click Open.
- In the Global Element Properties dialog box, click OK.

Remove the listeners

- In the Global Elements view, select the HTTP Listener and click Delete.

Type	Name	Description	
HTTP Listener Configuration (Configured)	HTTP_Listener_Configuration		Create
MySQL Configuration (Configured)	MySQL_Configuration		Edit
			Delete

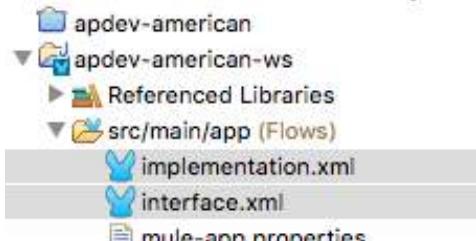
- Return to the Message Flow view.
- Right-click the HTTP Listener endpoint in one of the flows and select Delete.



- Delete the other two HTTP Listener endpoints.

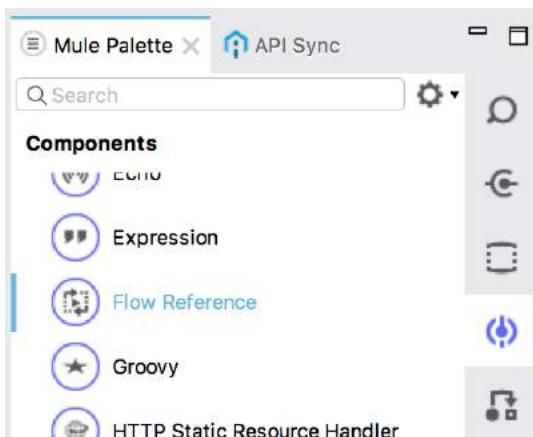
Rename the configuration files

17. Right-click american.xml in the Package Explorer and select Refactor > Rename.
18. In the Rename Resource dialog box, set the new name to interface.xml and click OK.
19. Right-click apdev-american.xml and select Refactor > Rename.
20. In the Rename Resource dialog box, set the new name to implementation.xml and click OK.

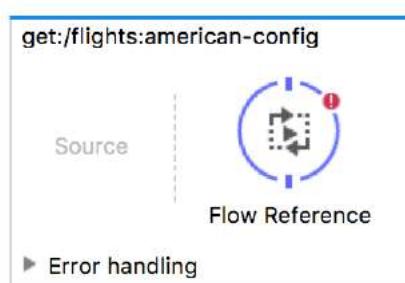


Set logic for the /flights resource

21. Return to interface.xml.
22. Delete the Set Payload transformer in the get:/flights flow.
23. In the Mule Palette, select the Components tab.



24. Drag a Flow Reference component from the Mule Palette and drop it into the process section of the flow.



25. In the Flow Reference properties view, select getFlightsFlow for the flow name.

The screenshot shows the 'Flow Reference' properties view in Mule Studio. The 'General' tab is selected. It displays the 'Display Name' as 'getFlightsFlow' and the 'Flow name' as 'getFlightsFlow'. A note at the top states 'There are no errors.'

Set logic for the /flights/{ID} resource

26. Delete the Set Payload transformer in the get:/flights/{ID} flow.
27. Drag a Flow Reference component from the Mule Palette and drop it into the flow.
28. In the Flow Reference properties view, select getFlightsByIDFlow for the flow name.



29. Return to implementation.xml.
30. Double-click the Database endpoint in getFlightsByIDFlow.
31. Change the query to use a variable instead of a query parameter.

WHERE ID = #[flowVars.ID]

The screenshot shows the configuration dialog for a Database endpoint. The 'Type' is set to 'Parameterized'. The 'Parameterized query' field contains the following SQL:

```
SELECT *  
FROM american  
WHERE ID = #[flowVars.ID]
```

Note: You learn about the different types of variables in later modules in the developer courses.

Test the web service in the APIkit Consoles view

32. Run the project.
33. In the APIkit Consoles view, click the GET button for /flights.
34. Click the Try It button.
35. Click the GET button; you should now see the real data from the database displayed instead of the example data.

The screenshot shows the APIkit Consoles interface. On the left, there is a large code editor window displaying a JSON array of flight records. Each record contains fields like ID, code, price, departure date, origin, destination, empty seats, and plane type. On the right, there is a panel titled "Response" which shows the status (200), headers (Content-Type: application/json; charset=UTF-8, Date: Wed, 01 Jun 2016 18:06:59 GMT, Transfer-Encoding: Identity), and the body of the response, which is identical to the data in the code editor.

```
[{"ID": 1, "code": "ER38sd", "price": 400, "departureDate": "2016/03/20", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 2, "code": "ER45if", "price": 345.99, "departureDate": "2016/02/11", "origin": "MUA", "destination": "LAX", "emptySeats": 52, "plane": {"type": "Boeing 777", "totalSeats": 300}}]
```

Response

Status
200

Headers
content-type: application/json; charset=UTF-8
date: Wed, 01 Jun 2016 18:06:59 GMT
transfer-encoding: Identity

Body

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

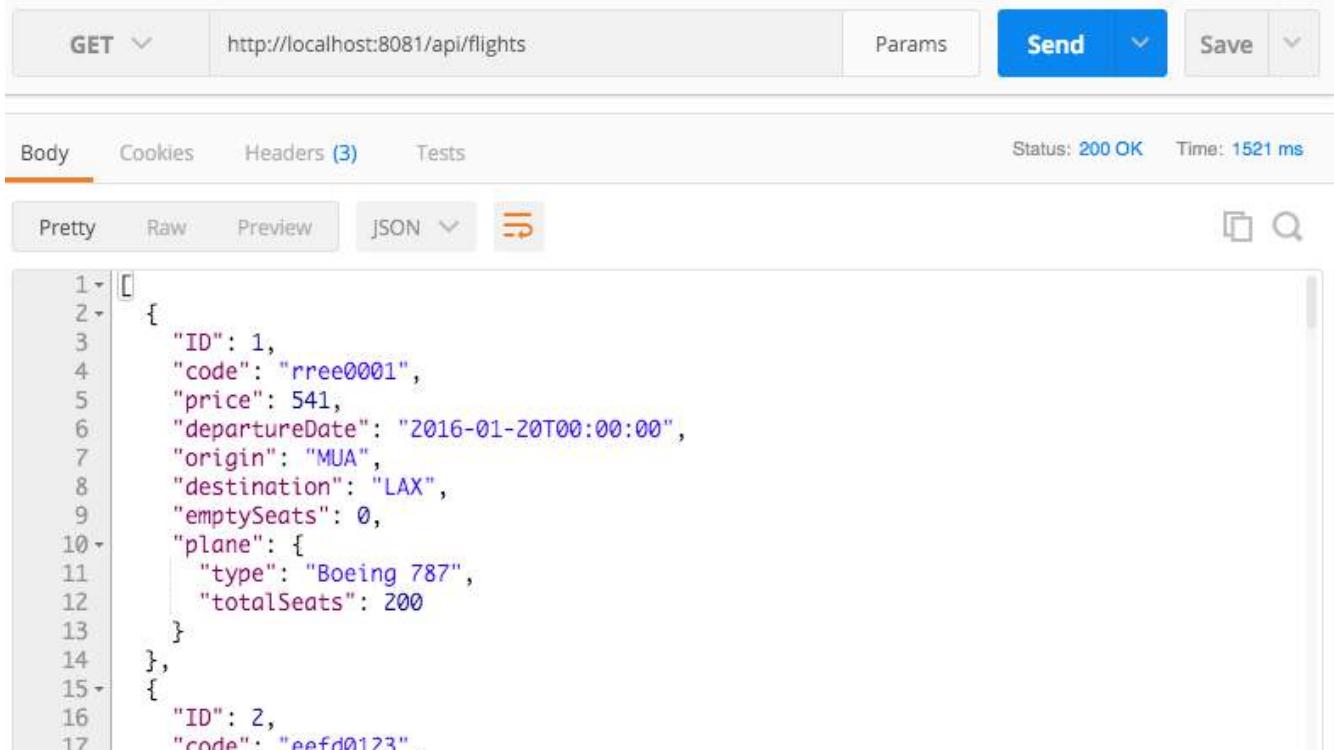
36. Scroll down and click the GET button for the /flights/{ID} resource.
37. Click the Try It button.
38. Enter an ID of 2 and click the GET button; you should get the data for that flight.

The screenshot shows the APIkit Consoles interface. On the left, there is a code editor window with a section labeled "Body". Inside the body, there is a JSON object representing a single flight record with ID 2, code "eefd0123", price 300, and departure date "2016-01-25T00:00:00".

```
1 [{"ID": 2, "code": "eefd0123", "price": 300, "departureDate": "2016-01-25T00:00:00"}]
```

Test the web service in Postman

39. In Postman, 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 the Postman interface with a successful HTTP request. The URL is <http://localhost:8081/api/flights>. The response status is 200 OK and the time taken is 1521 ms. The response body is displayed in Pretty JSON format:

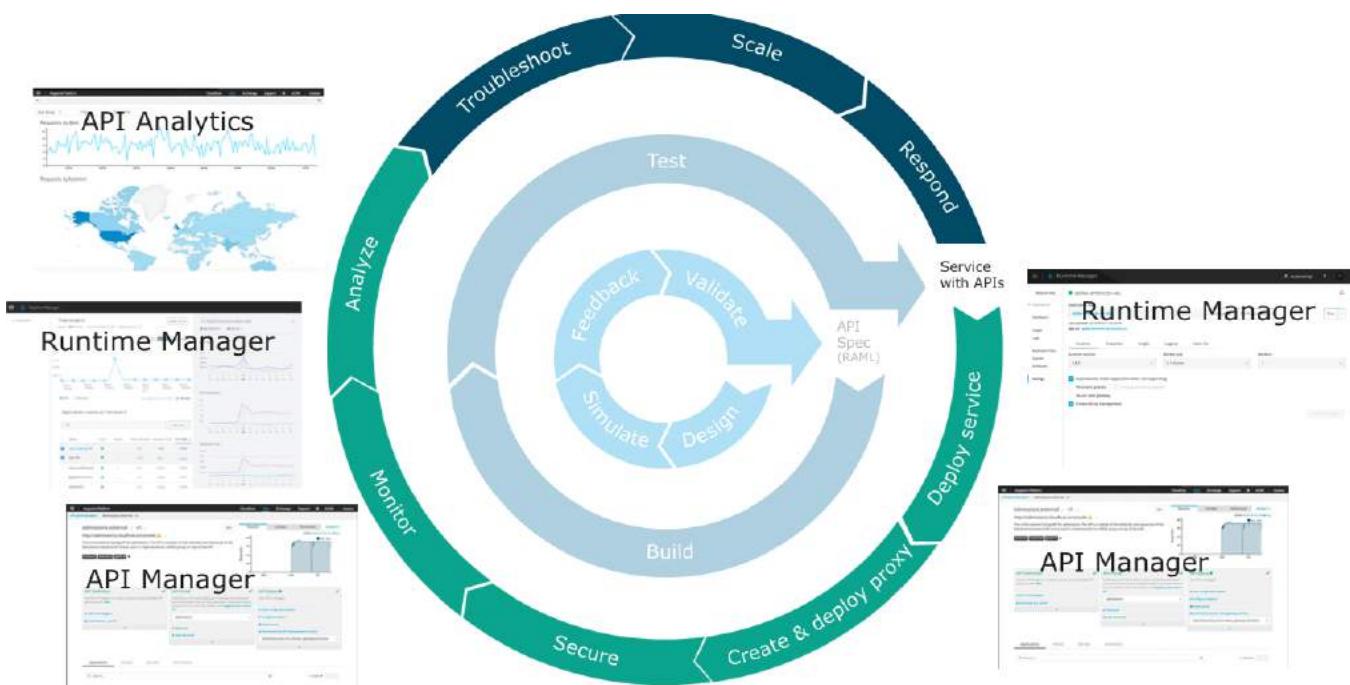
```
1 [ ] [
2   {
3     "ID": 1,
4     "code": "rree0001",
5     "price": 541,
6     "departureDate": "2016-01-20T00:00:00",
7     "origin": "MUA",
8     "destination": "LAX",
9     "emptySeats": 0,
10    "plane": {
11      "type": "Boeing 787",
12      "totalSeats": 200
13    }
14  },
15  {
16    "ID": 2,
17    "code": "eefd0123".

```

40. Make a request to <http://localhost:8081/api/flight/3>; you should now get the data that flight from the database instead of the sample data.

41. Return to Anypoint Studio and stop the project.

Module 4: Deploying and Managing APIs



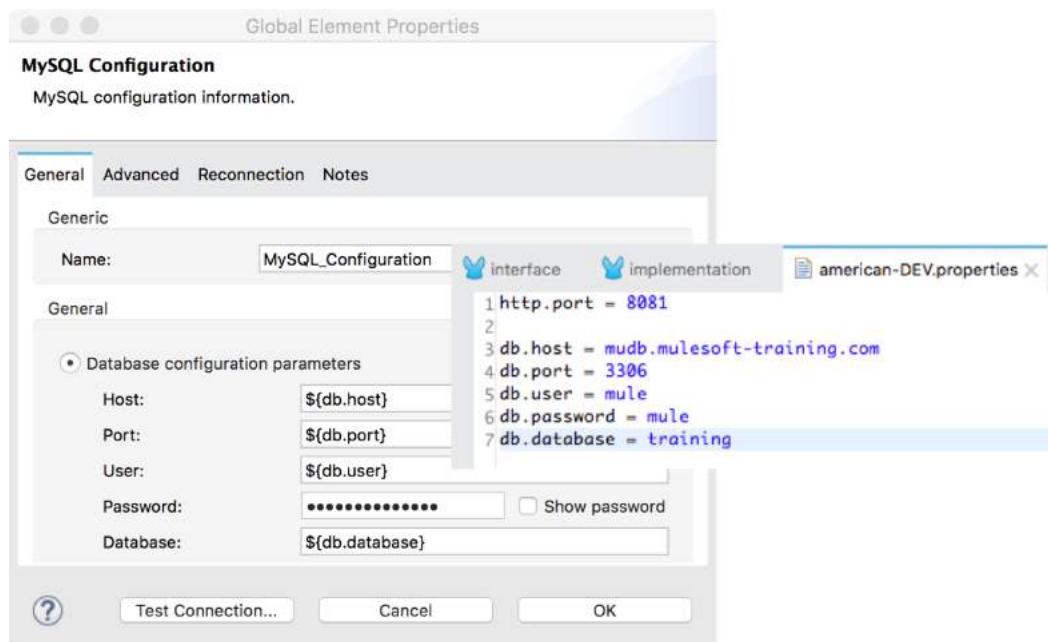
Objectives:

- Describe the options for deploying Mule applications.
- Use properties in Mule applications so they can be easily moved between environments.
- Deploy a Mule application to the cloud.
- Create and deploy a proxy for an API in the cloud.
- Restrict access to an API proxy.

Walkthrough 4-1: Prepare an API for deployment using properties

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

- Create a properties file for your application.
- Create a Properties Placeholder global element to specify the properties file.
- Define and use Database connector properties.
- Parameterize the HTTP Listener.
- Specify a properties file dynamically.



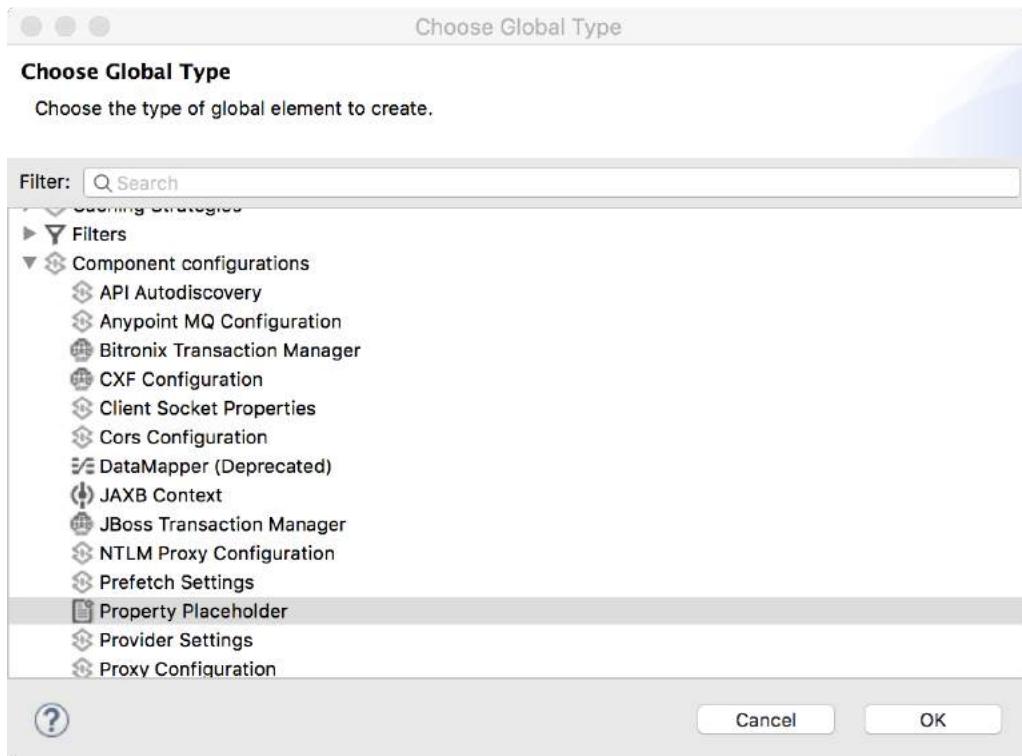
Create a properties file

1. Right-click the src/main/resources folder in the Package Explorer and select New > File.
2. Set the file name to american-DEV.properties and click Finish.



Create a Properties Placeholder global element

3. Return to interface.xml.
4. Click the Global Elements link at the bottom of the canvas.
5. Click Create.
6. In the Choose Global Type dialog box, select Component configurations > Property Placeholder and click OK.

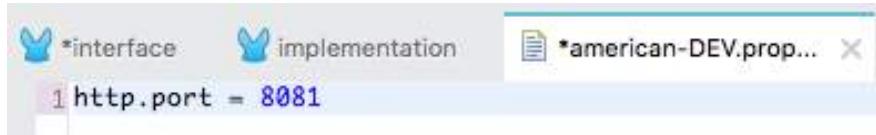


7. In the Global Element Properties dialog box, set the location to american-DEV.properties and click OK.

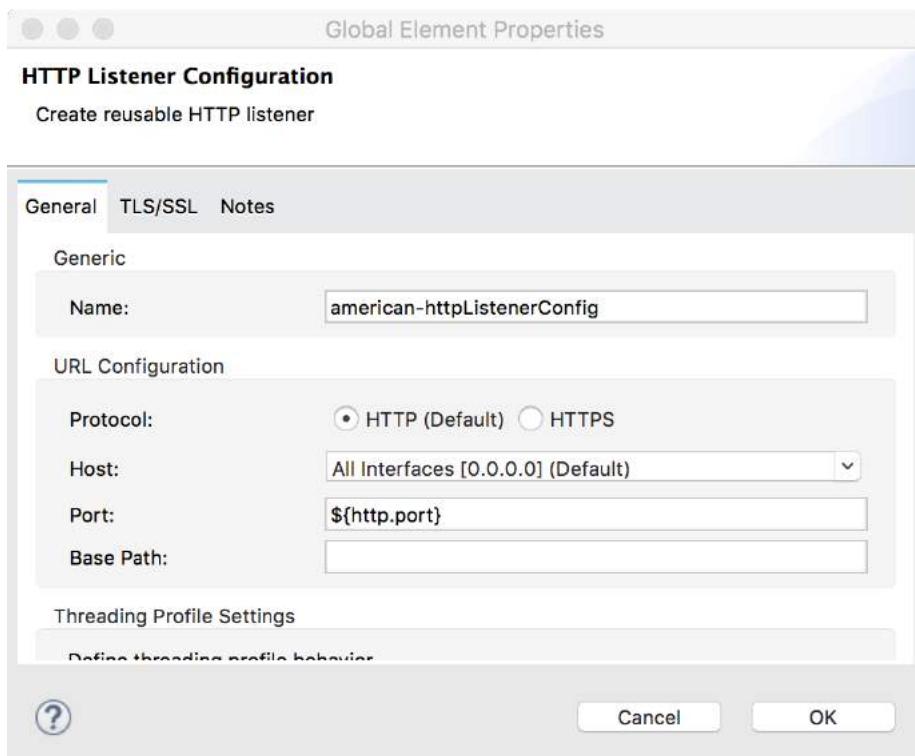


Parameterize the HTTP Listener port

8. Return to american-DEV.properties.
9. Create a property called http.port and set it to 8081.



10. Return to the Global Elements view in interface.xml.
11. Double-click the HTTP Listener Configuration global element.
12. Change the port from 8081 to the application property, \${http.port}.
13. Click OK.



Parameterize the database credentials

14. Return to the course snippets.txt file and copy the database parameters (the five starting with db).

15. Return to american-DEV.properties and paste the values.

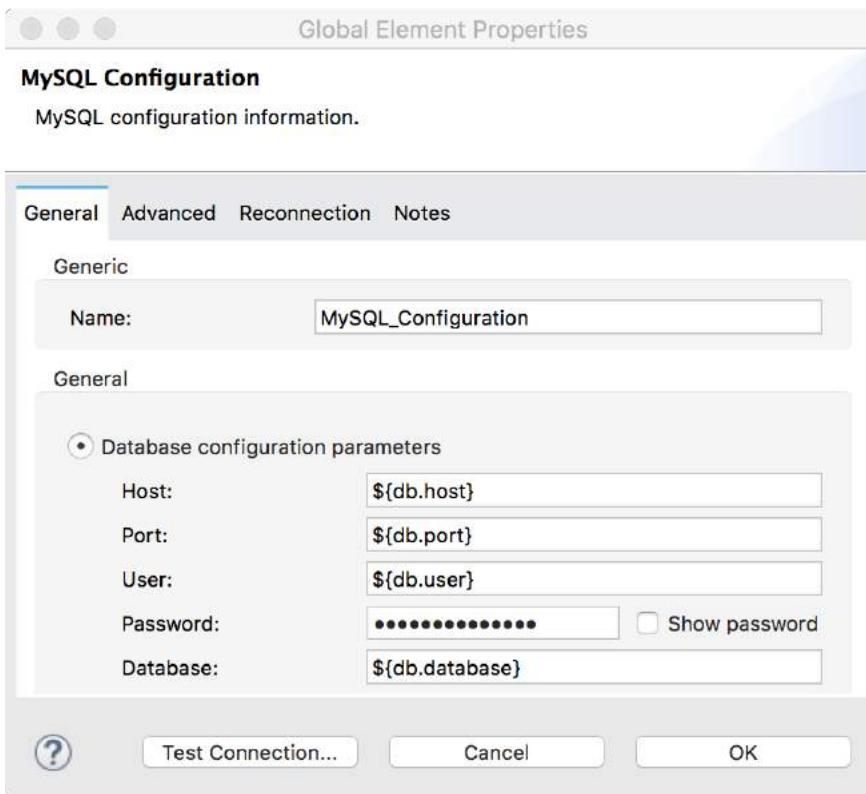
```
1 http.port = 8081
2
3 db.host = mudb.mulesoft-training.com
4 db.port = 3306
5 db.user = mule
6 db.password = mule
7 db.database = training
```

16. Save the file.

17. Return to the Global Elements view in implementation.xml.

18. Double-click the Database Configuration global element.

19. Change the values to use the application properties.



20. Click Test Connection and make sure it succeeds.

Note: If your connection fails, click OK and then go back and make sure you saved american-DEV.properties.

21. Click OK.

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

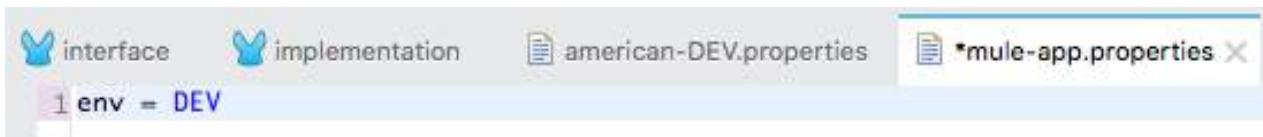
23. Switch to the Message Flow view.

Test the application

24. Run the project.
25. In Postman, make a request <http://localhost:8081/api/flights> and confirm you still get data.

Define an environment property value in mule-app.properties

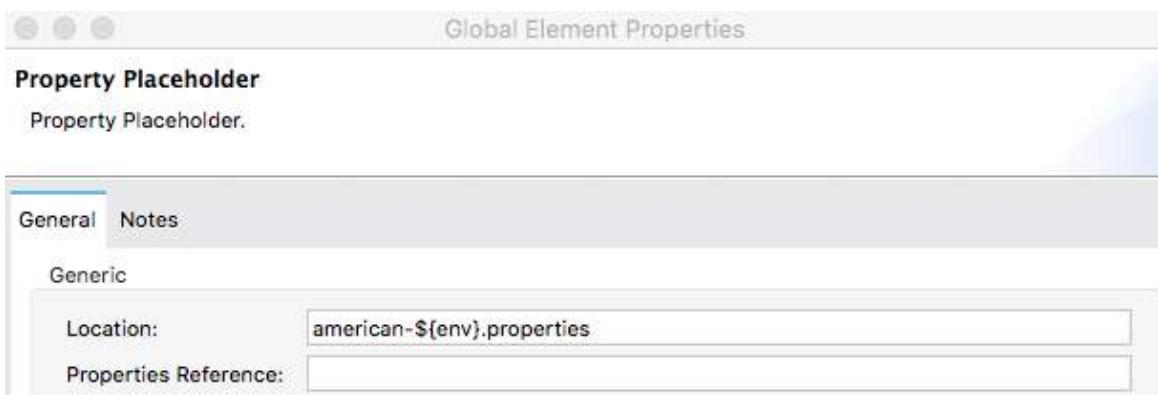
26. Return to Anypoint Studio and stop the project.
27. Open mule-app.properties located in the src/main/app folder.
28. Define a property called env and set it to DEV.



29. Save the file.

Use the environment property in the Property Placeholder

30. Return to the Global Elements view in interface.xml.
31. Double-click the Property Placeholder to edit it.
32. In the Global Element Properties dialog box, change the location to american-\${env}.properties and click OK.



Test the application

33. Return to the Message Flow view in interface.xml.
34. Run the project.
35. In Postman, make a request <http://localhost:8081/api/flights> and confirm you still get data.
36. Return to Anypoint Studio and stop the project.

Walkthrough 4-2: Deploy an application to the cloud

In this walkthrough, you deploy and run your application on Anypoint Platform in the cloud. You will:

- Deploy an application from Anypoint Studio to the cloud.
- Run the application on its new, hosted domain.
- Make calls to the web service.
- Update the API implementation deployed to the cloud.

The screenshot shows the Anypoint Platform Runtime Manager. At the top, there's a navigation bar with 'Runtime Manager' and other icons. Below it, a sidebar has 'PRODUCTION' selected and options for 'Applications', 'Servers', and 'Alerts'. The main area has a 'Deploy application' button and a search bar. A table lists applications: 'apdev-american-ws' is listed under 'Name', 'CloudHub' under 'Server', 'Started' under 'Status', and 'apdev-american-ws.zip' under 'File'.

Deploy the application to the cloud

1. Return to the apdev-american-ws project in Anypoint Studio.
2. In the Package Explorer, right-click the project and select Anypoint Platform > Deploy to Cloud.



3. At the top of the Anypoint Platform dialog box, set the application name to apdev-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 'apdev-american-ws' is entered in the field, which has a green checkmark icon next to it.

4. 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 3.8.0. EE.

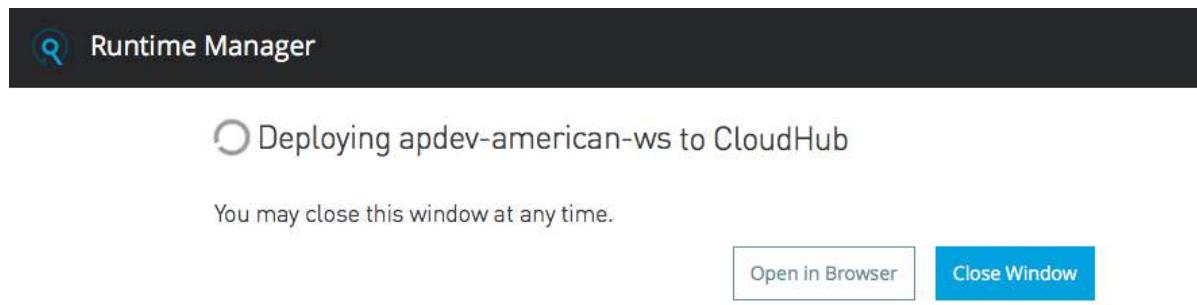
5. Set the worker size to 0.1 vCores.

Runtime	Properties	Insight	Logging	Static IPs
Runtime version	Worker size			Workers
3.8.0	0.1 vCores			1

6. Click the Properties tab; you should see the env variable set to DEV.

Runtime	Properties	Insight	Logging	Static IPs
	Text			
	List			
		env=DEV		

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



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

The screenshot shows the Anypoint Platform Runtime Manager. The left sidebar has tabs for "PRODUCTION", "Applications", "Servers", and "Alerts". The main area has a "Deploy application" button and a search bar. A table lists the deployment status of "apdev-american-ws" to "CloudHub".

Name	Server	Status	File
apdev-american-ws	CloudHub	Deploying	apdev-american-ws.zip

Watch the logs and wait for the application to start

10. Click the name of the application; you should see information about the application appear on the right-side of the window.

The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'PRODUCTION' selected, followed by 'Applications', 'Servers', and 'Alerts'. The main area has a 'Deploy application' button and a search bar. A table lists an application named 'apdev-american-ws' with 'CloudHub' as the server, 'Deploying' as the status, and 'apdev-american-ws.zip' as the file. To the right, a detailed view for 'apdev-american-ws' is shown, including a progress bar indicating 'Deploying', the file 'apdev-american-ws.zip', and deployment details like runtime version 3.8.0, worker size 0.1 vCores, and 1 worker. Buttons for 'Manage Application', 'Logs', and 'Insight' are at the bottom.

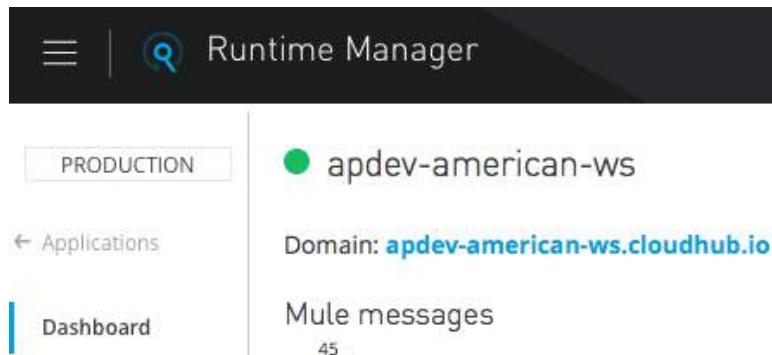
- 11. Click the Logs button.
- 12. Watch the logs as the application is deployed.
- 13. Wait until the application starts (or fails to start).

This screenshot shows the 'Logs' tab in the Runtime Manager. The left sidebar has 'PRODUCTION' selected and 'Logs' highlighted. The main area displays log entries for the application 'apdev-american-ws'. The first entry shows the application starting at 16:45:03.629 on 05/03/2016. Subsequent entries show worker logs and deployment details. To the right, a 'Deployments' panel shows a timeline of events: a deployment at 16:40, another at 16:33, and one at 16:28. Each deployment entry includes a 'System Log' and a 'Worker-0' entry.

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

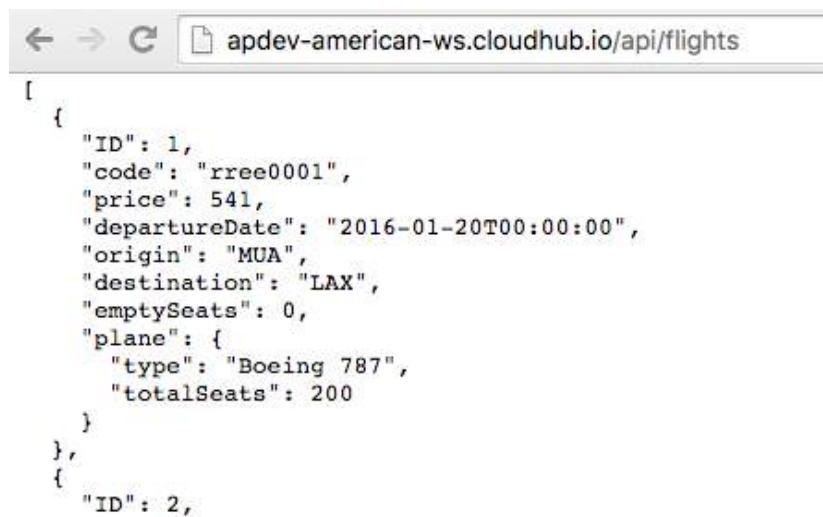
Test the application

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



The screenshot shows the Mule Runtime Manager interface. At the top, there's a dark header with a search icon and the text "Runtime Manager". Below it, a navigation bar has "PRODUCTION" selected. The main area displays the application "apdev-american-ws" with a green status dot. To the left, there's a sidebar with "Applications" and "Dashboard" buttons. On the right, the "Domain" is listed as "**apdev-american-ws.cloudbhub.io**". Below that, "Mule messages" are shown with a count of 45.

16. Click the link; a request will be made to that URL and you should get a resource not found message.
17. Modify the path to `http://apdev-american-ws-{lastname}.cloudbhub.io/api/flights`; you should see the flights data.



The screenshot shows a browser window with the URL `apdev-american-ws.cloudbhub.io/api/flights`. The page displays a JSON array of flight data:

```
[  
  {  
    "ID": 1,  
    "code": "rree0001",  
    "price": 541,  
    "departureDate": "2016-01-20T00:00:00",  
    "origin": "MUA",  
    "destination": "LAX",  
    "emptySeats": 0,  
    "plane": {  
      "type": "Boeing 787",  
      "totalSeats": 200  
    }  
  },  
  {  
    "ID": 2,  
    "code": "rree0002",  
    "price": 541,  
    "departureDate": "2016-01-20T00:00:00",  
    "origin": "MUA",  
    "destination": "LAX",  
    "emptySeats": 0,  
    "plane": {  
      "type": "Boeing 787",  
      "totalSeats": 200  
    }  
  }]
```

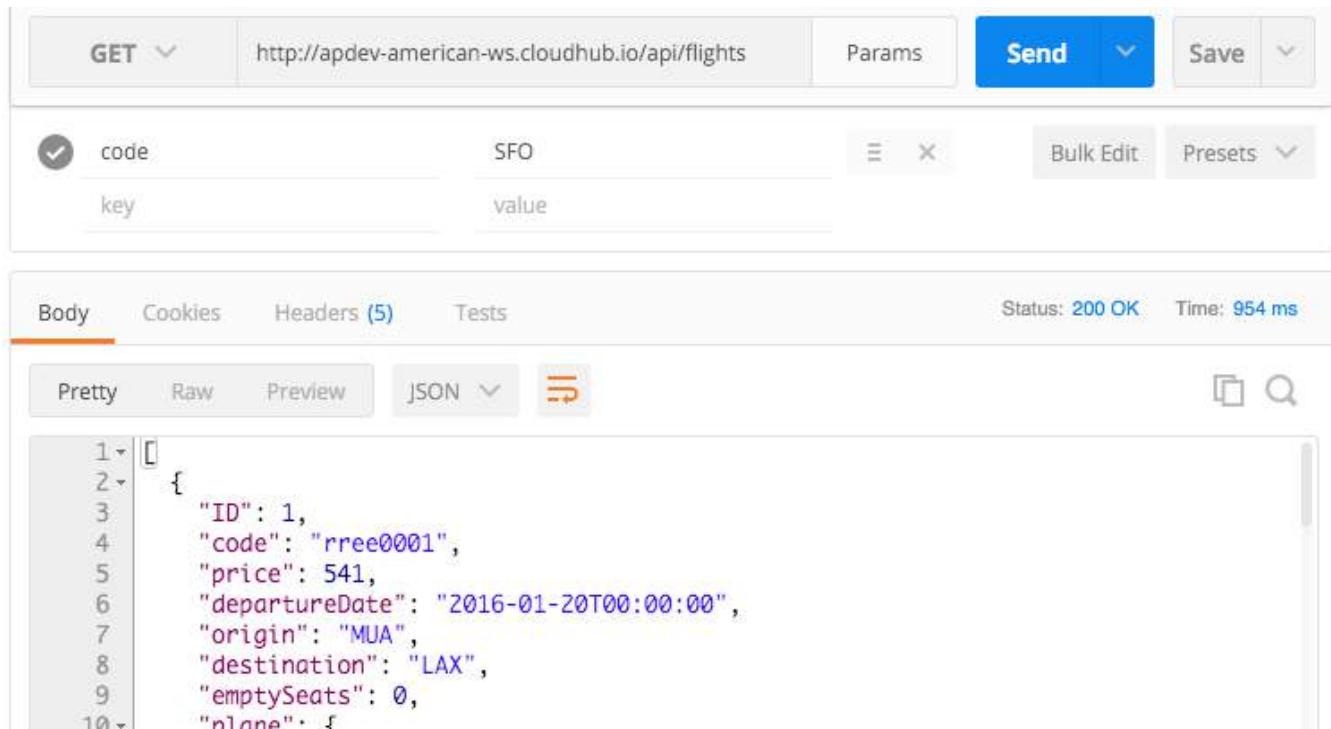
18. Copy the URL.
19. Return to Postman and paste the value in the URL.
20. Click Send; you should get the flights data.

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

21. Add a query parameter called code and set it equal to SFO.

22. 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.



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

- Method: GET
- URL: <http://apdev-american-ws.cloudhub.io/api/flights>
- Params: code=SFO
- Send and Save buttons
- Table for parameters: code (key), SFO (value)
- Body tab selected, showing Headers (5) and Tests. Status: 200 OK, Time: 954 ms.
- Body content (Pretty JSON):

```
1 [ { 2 "ID": 1, 3 "code": "rree0001", 4 "price": 541, 5 "departureDate": "2016-01-20T00:00:00", 6 "origin": "MUA", 7 "destination": "LAX", 8 "emptySeats": 0, 9 "plane": {} 10 }
```
- Pretty, Raw, Preview, JSON buttons
- Copy and Search icons

Update the API implementation in the cloud

23. Return to the apdev-american-ws application in the Runtime Manager in Anypoint Platform.
24. In the left-side navigation, click Settings.
25. Click the Choose file button.
26. Browse to the resources folder in the course student files.
27. Select apdev-american-ws.zip and click Open.

Note: This updated version of the application adds functionality to return results for a particular destination. It also adds flows for deleting and updating a flight, though they don't actually modify the database.

28. Click the Apply Changes button.

The screenshot shows the Mule Runtime Manager interface. On the left, a sidebar lists navigation options: PRODUCTION, Applications (with a back arrow), Dashboard, Insight, Logs, Application Data, Queues, Schedules, and Settings (which is currently selected). The main panel displays the application 'apdev-american-ws'. It includes fields for the Application File ('apdev-american-ws.zip'), a Stop button, and a bell icon. Below this, the App url is shown as 'apdev-american-ws.cloudhub.io'. A tab bar at the top of the main area includes Runtime (selected), Properties, Insight, Logging, and Static IPs. Under Runtime, the Runtime version is set to '3.8.0', Worker size to '0.1 vCores', and Workers to '1'. There are several checkboxes for settings: 'Automatically restart application when not responding' (checked), 'Persistent queues' (unchecked), 'Encrypt persistent queues' (unchecked), 'Secure data gateway' (unchecked), and 'Enhanced log management' (checked). At the bottom right of the main panel is a large blue 'Apply Changes' button.

29. Wait until the application is uploaded and then redeloys successfully.

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

Test the application

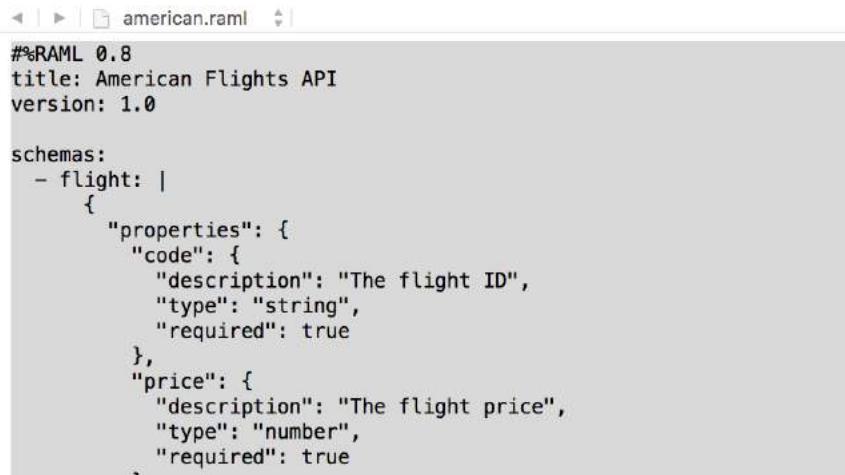
30. In Postman, make the same request with a code of SFO; you should now get only flights to SFO.

The screenshot shows a Postman request configuration and its response. The request method is 'GET' and the URL is 'http://apdev-american-ws-jms.cloudhub.io/api/flights?code=SFO'. The response status is '200 OK' and the time taken is '993 ms'. The response body is displayed in 'Pretty' format, showing a JSON array of flight details. The array contains one element, which is a flight object with the following properties:

```
1 | [ | { | "ID": 5, | "code": "rree1093", | "price": 142, | "departureDate": "2016-02-11T00:00:00", | "origin": "MUA", | "destination": "SFO", | "amountVents": 1 | }
```

Update the RAML file in the cloud

31. In your computer's file browser, locate and open the american.raml file in the student files.
32. Copy all the code in the file.



```
#%RAML 0.8
title: American Flights API
version: 1.0

schemas:
  - flight: |
    {
      "properties": {
        "code": {
          "description": "The flight ID",
          "type": "string",
          "required": true
        },
        "price": {
          "description": "The flight price",
          "type": "number",
          "required": true
        }
      }
    }
```

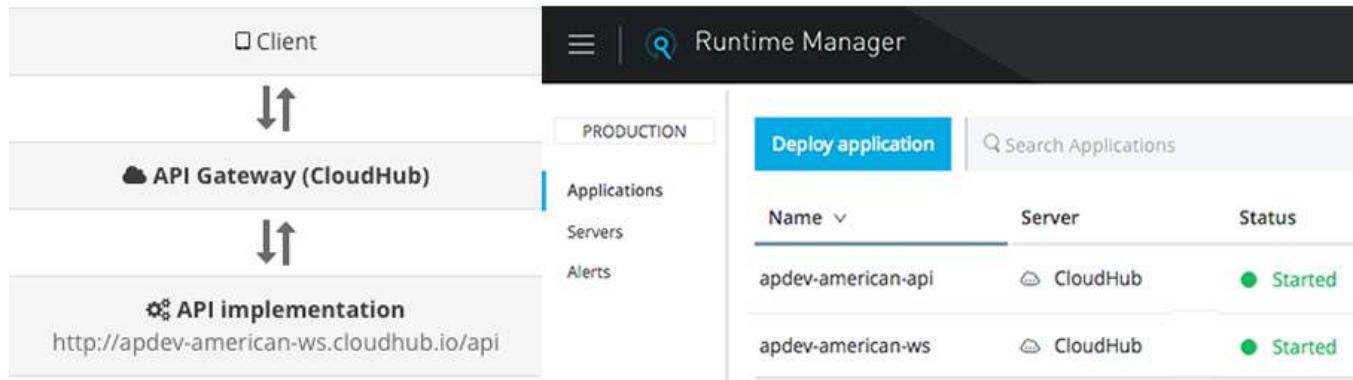
33. Return to Anypoint Platform in a web browser.
34. From the main menu, select API Manager.
35. Click the American Flights API version 1.0 link.
36. In the API Definition section, click the Edit in API designer link.
37. Turn off the mocking service.
38. Select all the text and delete it.
39. Paste the new text from the american.raml file.
40. Save the file.
41. Leave this page open.

Note: You updated the RAML file to match the updated, deployed application. At this time, you are not going to update the implementation source code in Anypoint Studio because it requires the addition of functionality that is taught later in the Developer courses.

Walkthrough 4-3: Create and deploy an API proxy

In this walkthrough, you create and deploy an API proxy for your API implementation in the cloud. You will:

- Set the API implementation URI in the RAML file.
- Use Anypoint Platform to automatically create an API proxy application.
- Deploy the API proxy application.



Set the implementation URI in the RAML file

1. Return to the american.raml definition page for American Flights API – 1.0 in Anypoint Platform.
2. Add a baseUri and set it equal to <http://apdev-american-ws-{lastname#}.cloudhub.io/api>.

```
1  #%RAML 0.8
2  baseUri: http://apdev-american-ws.cloudhub.io/api
3  title: American Flights API
4  version: 1.0
```

3. Save the file.
4. Click the American Flights API - 1.0 link.

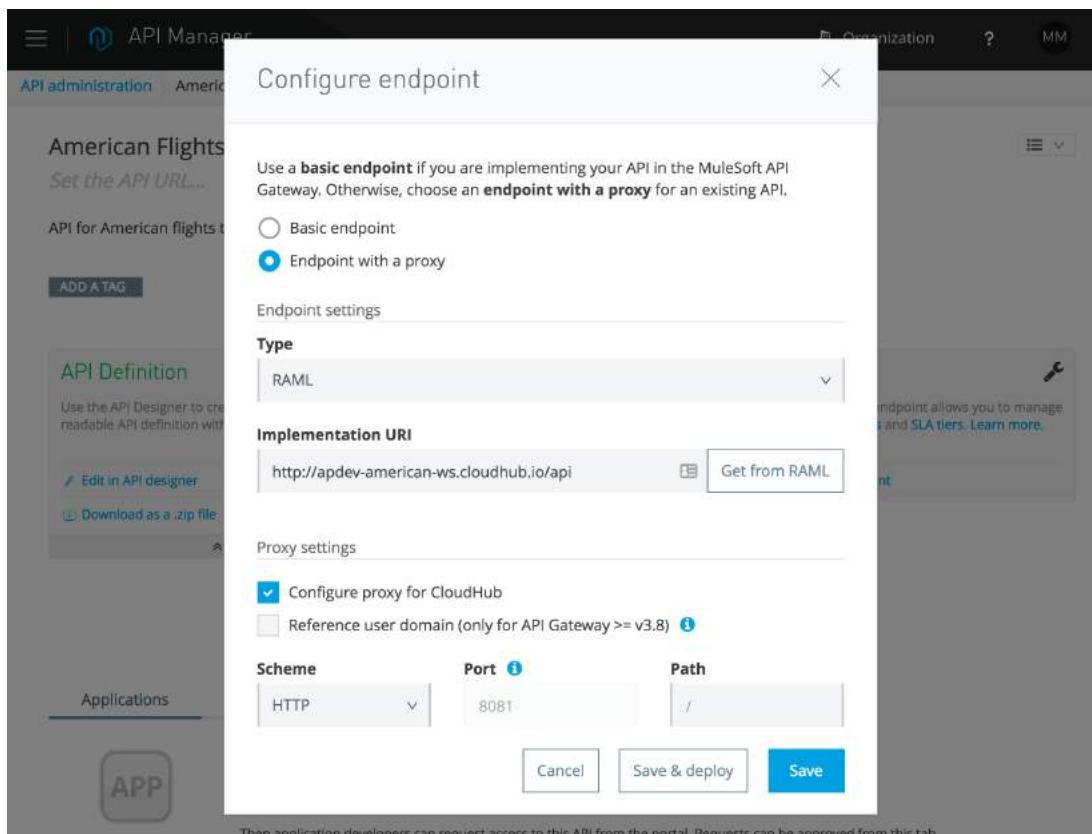
Create an API proxy

5. In the API Status section, click the Configure endpoint link.

The screenshot shows the 'API Status' section of the Anypoint Platform interface. It features three cards: 'API Definition' (with a green checkmark), 'API Portal' (with a green checkmark), and 'API Status' (with a wrench icon). The 'API Status' card contains text about configuring the API endpoint and managing policies and SLA tiers. Below the cards, there's a link to 'Configure endpoint'.

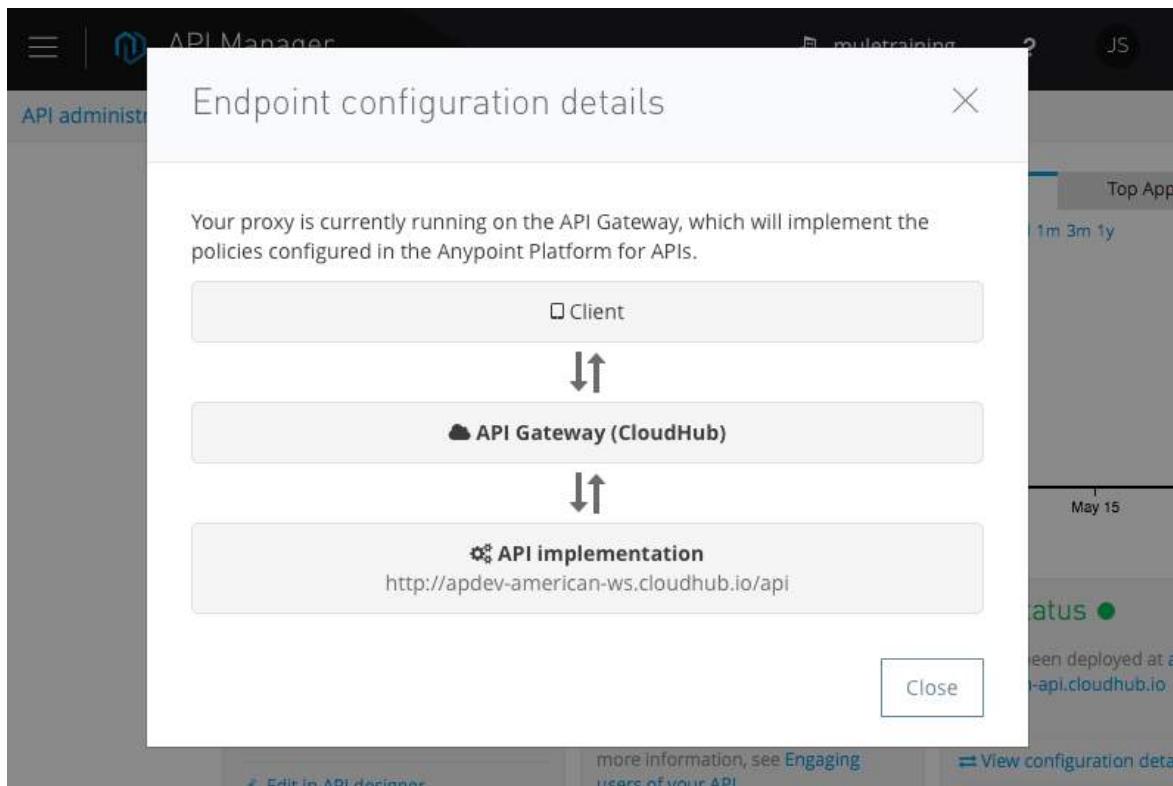
6. In the Configure endpoint dialog box, confirm the following values are set:

- Endpoint with a proxy
 - Type: RAML
 - Implementation URI: `http://apdev-american-ws-{lastname#}.cloudhub.io/api`
7. Select the Configure proxy for CloudHub checkbox.
8. Confirm the following values are set:
- Scheme: RAML
 - Port: 8081
 - Path: /



9. Click Save.

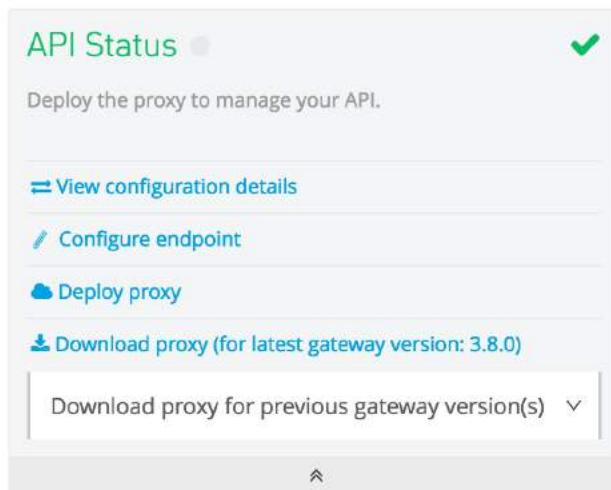
10. In the API Status section, click the View configuration details link.



11. Click Close.

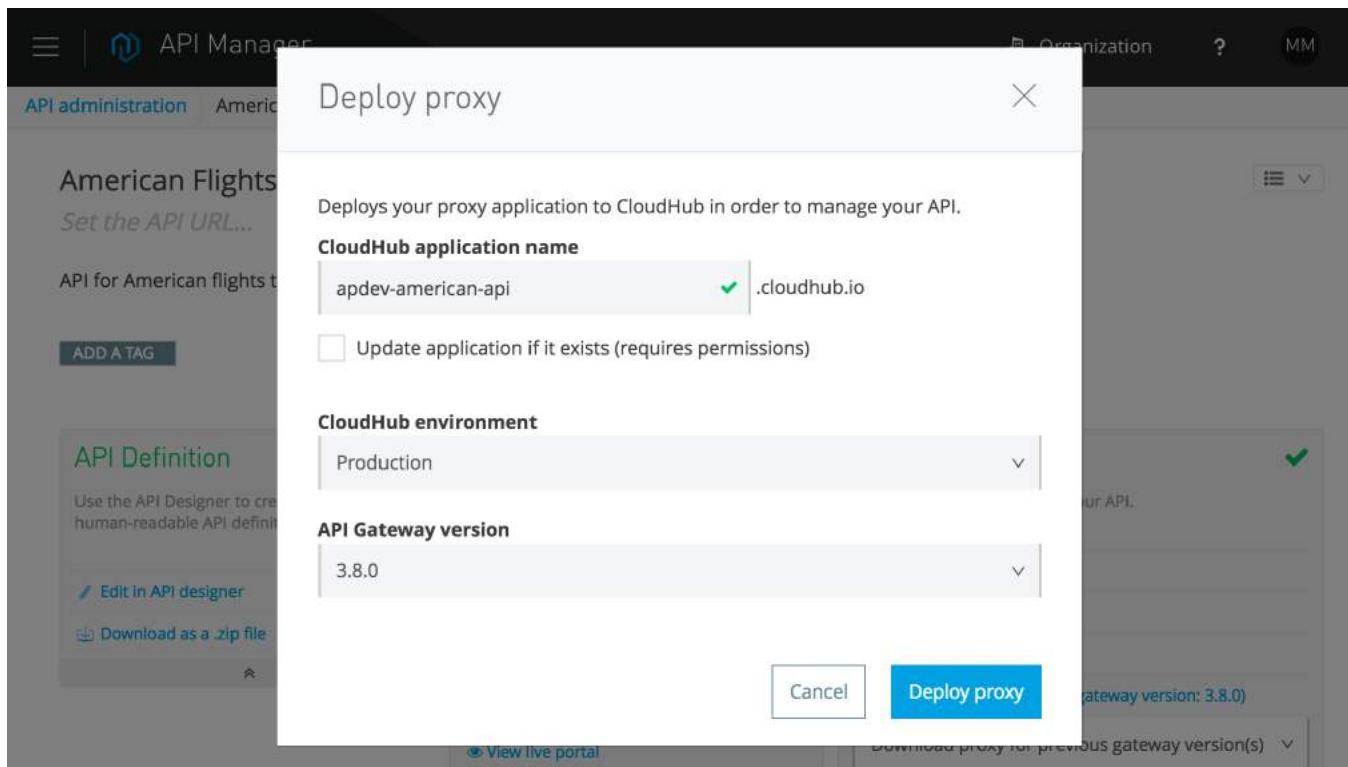
Deploy the proxy

12. In the API Status section, click the Deploy proxy link.

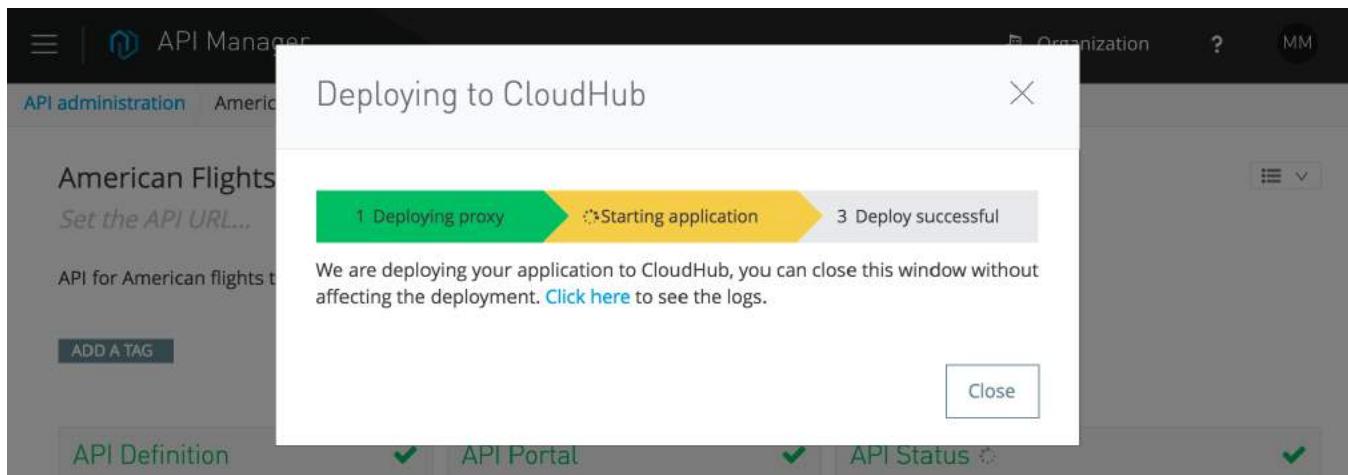


13. In the Deploy proxy dialog box, change the CloudHub application name to apdev-american-api-{lastname#}.cloudhub.io.

14. Click Deploy proxy.



15. In the Deploying to CloudHub dialog box, click the Click here link to watch the logs.



16. Watch the logs.

17. 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.

Locate the API proxy application

18. In the main menu in Anypoint Platform, select Runtime Manager.
19. Locate the new apdev-american-api-{lastname#} application.

The screenshot shows the 'Runtime Manager' interface. On the left, there's a sidebar with 'PRODUCTION' selected, and options for 'Applications', 'Servers', and 'Alerts'. The main area has a 'Deploy application' button and a search bar. A table lists two applications:

Name	Server	Status	File
apdev-american-api	CloudHub	Started	apdev-american-api-api-gateway.zip
apdev-american-ws	CloudHub	Started	apdev-american-ws.zip

Look at the API request data

20. From the main menu, select API Manager.
21. Click the American Flights API version 1.0 link.
22. Look at the Requests chart in the upper-right corner.

Call the API

23. Return to Postman.
24. Change the request URL to apdev-american-api-{lastname#}.cloudhub.io/flights; you should successfully get flights.

Note: Be sure to remove the /api from the URL.

The screenshot shows the Postman application interface. At the top, it says 'GET' and the URL is 'http://apdev-american-api-jms.cloudhub.io/flights'. Below that, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Tests'. The status is 'Status: 200 OK' and 'Time: 1043 ms'. Under the 'Body' tab, there are tabs for 'Pretty', 'Raw', 'Preview', and 'JSON'. The JSON response is displayed in a code editor-like view:

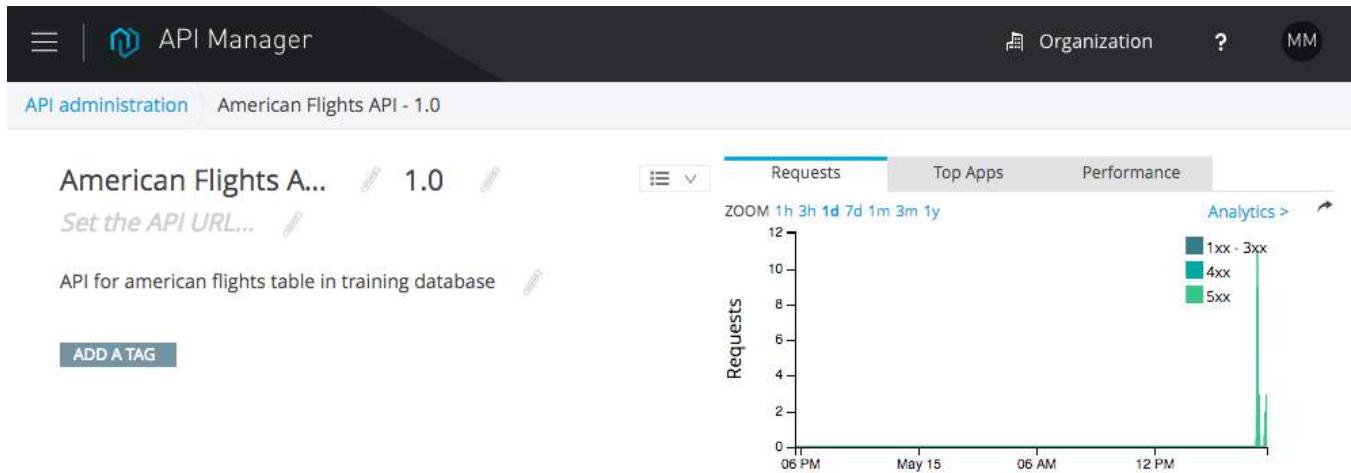
```
1. [
2.   {
3.     "ID": 1,
4.     "code": "rree0001",
5.     "price": 541,
6.     "departureDate": "2016-01-20T00:00:00",
7.     "origin": "MUA",
8.     "destination": "LAX",
9.     "amount": 541
10.  }
11. ]
```

25. Make several more calls.

Look at the API request data

26. Return to the API details page for American Flights API v1.0 in Anypoint Platform.

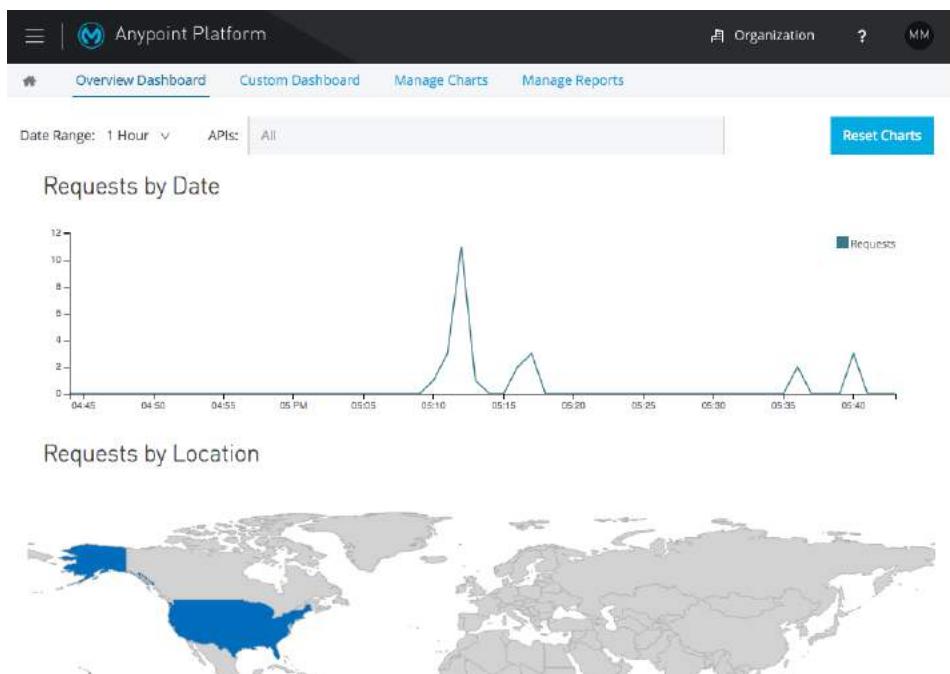
27. Look at the Request graph in the upper-right corner; you should see some calls.



28. Click the Analytics> link.

29. Take a quick look at the Overview Dashboard.

30. Change the Date Range to 1 hour.



31. Close the web browser tab or window.

Walkthrough 4-4: Restrict API access

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 a rate limiting SLA policy.
- Request access for an application to an SLA tier from the API Developer portal.
- Approve access for an application to an SLA tier.
- Call the governed API with client credentials.

The screenshot shows the 'APDev fictitious application' configuration page. At the top, there's a header with a user icon and 'username05'. Below it, the application name 'APDev fictitious application' is displayed with a 'Delete this app' button. The main area has tabs for 'My applications' and 'APDev fictitious application'. Under the application tab, there's a form for a 'GET' request to 'apdev-american-api.cloudhub.io/flights?client_id=d1374b15c6'. It includes fields for 'client_id' (d1374b15c6864c3682ddbed2a247a) and 'client_secret' (a4cb1787a04c41c3866FD31B95435), with 'Params' and 'Save' buttons. To the right, there's a sidebar with 'Client ID' (3e1f2b48751049de917b66dc95489cd3) and 'Client secret' (77b1414d1f464c8d989C5B6C0DF701AE), along with 'Reset client secret' and 'Request tier change' buttons. Below the request form, a table shows API details: API name 'American Flights API', API version '1.0', Current SLA tier 'Silver', Requested SLA tier 'N/A', and Status 'Approved'.

Create a rate limiting policy

1. Return to the API details page for American Flights API v1.0 in Anypoint Platform.
2. At the bottom of the page, locate and click the Policies tab.

The screenshot shows the 'Policies' tab selected in the navigation bar. There's a search bar at the top. Below it, a message states 'There are no registered applications for this API Version.' The rest of the page is mostly empty.

- Click the Apply button next to the rate limiting policy.

Applications Policies SLA tiers Permissions

Applied policies

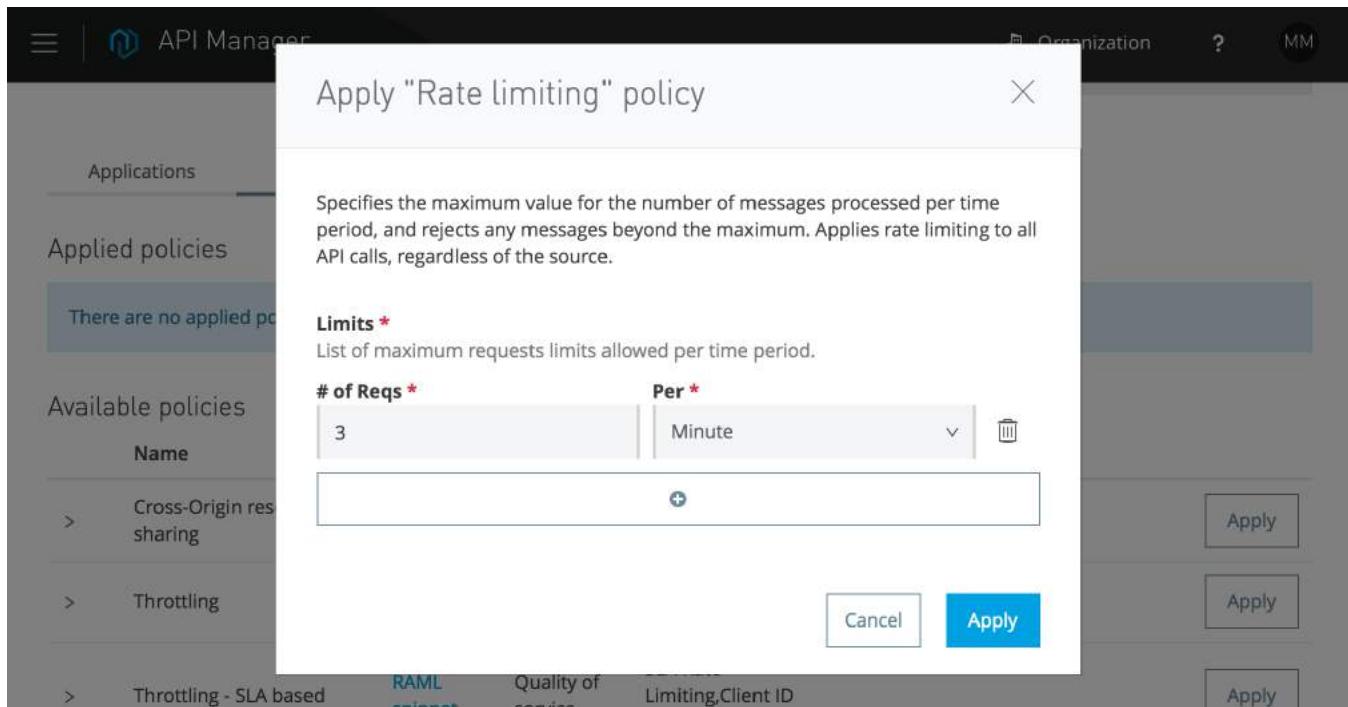
There are no applied policies.

Available policies

Name	Category	Fulfils	Requires
> Cross-Origin resource sharing	Compliance	CORS enabled	<button>Apply</button>
> Throttling	Quality of service	Baseline Rate Limiting	<button>Apply</button>
> Throttling - SLA based	RAML snippet	Quality of service SLA Rate Limiting, Client ID required	<button>Apply</button>
> Rate limiting	Quality of service	Baseline Rate Limiting	<button>Apply</button>

- In the Apply "Rate limiting" policy dialog box, set the following values and click Apply:

- # of Reqs: 3
- Per: Minute



Test the rate limiting policy

5. In Postman, make another request to <http://apdev-american-api-{lastname#}.cloudbhub.io/flights>; you should get flight results.
6. Click the Send button three more times until you get a response that the allowed number of API calls has been exceeded.

The screenshot shows a Postman request configuration for a GET request to 'apdev-american-api.cloudbhub.io/flights'. The 'Body' tab is selected, showing the response: 'Status: 429 Too Many Requests' and 'Time: 104 ms'. The response body contains the message 'API calls exceeded'.

Create SLA tiers

7. Return to the API administration page for American Flights API in Anypoint Platform.
8. At the bottom of the page, click the SLA tiers tab.
9. Click the Add SLA tier button.

The screenshot shows the 'SLA tiers' tab selected in the navigation bar. A blue button labeled 'Add SLA tier' is visible. A search bar is present. The message 'There are no SLA tiers for this API version.' is displayed.

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

- Name: Free
- Approval: Automatic
- # of Reqs: 1
- Per: Hour

The screenshot shows the 'Update SLA tier' dialog box. The 'Name' field is set to 'Free'. The 'Approval' field is set to 'Automatic'. Under the 'Limits' section, the '# of Reqs' field is set to '1' and the 'Per' field is set to 'Hour'. A 'visible' checkbox is checked, and a trash can icon is visible. At the bottom right are 'Cancel' and 'Update' buttons.

Update SLA tier

Name *

Free

Description

Description

Approval *

Automatic

Limits

of Reqs *

1

Per *

Hour

visible

Cancel Update

11. Click the Add button.

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

- Name: Silver
- Approval: Manual
- # of Reqs: 1
- Per: Second

Applications		Policies	SLA tiers	Permissions	
Add SLA tier		<input type="text"/> Search X		1 - 2 of 2 < >	
Name	Limits	Applications	Status	Approval	
Free	1	0	Active	Auto	Edit Delete
Silver	1	0	Active	Manual	Edit Delete

Change the policy to rate limiting – SLA based

13. Click the Policies tab.

14. Click the Remove button for the Rate limiting policy.

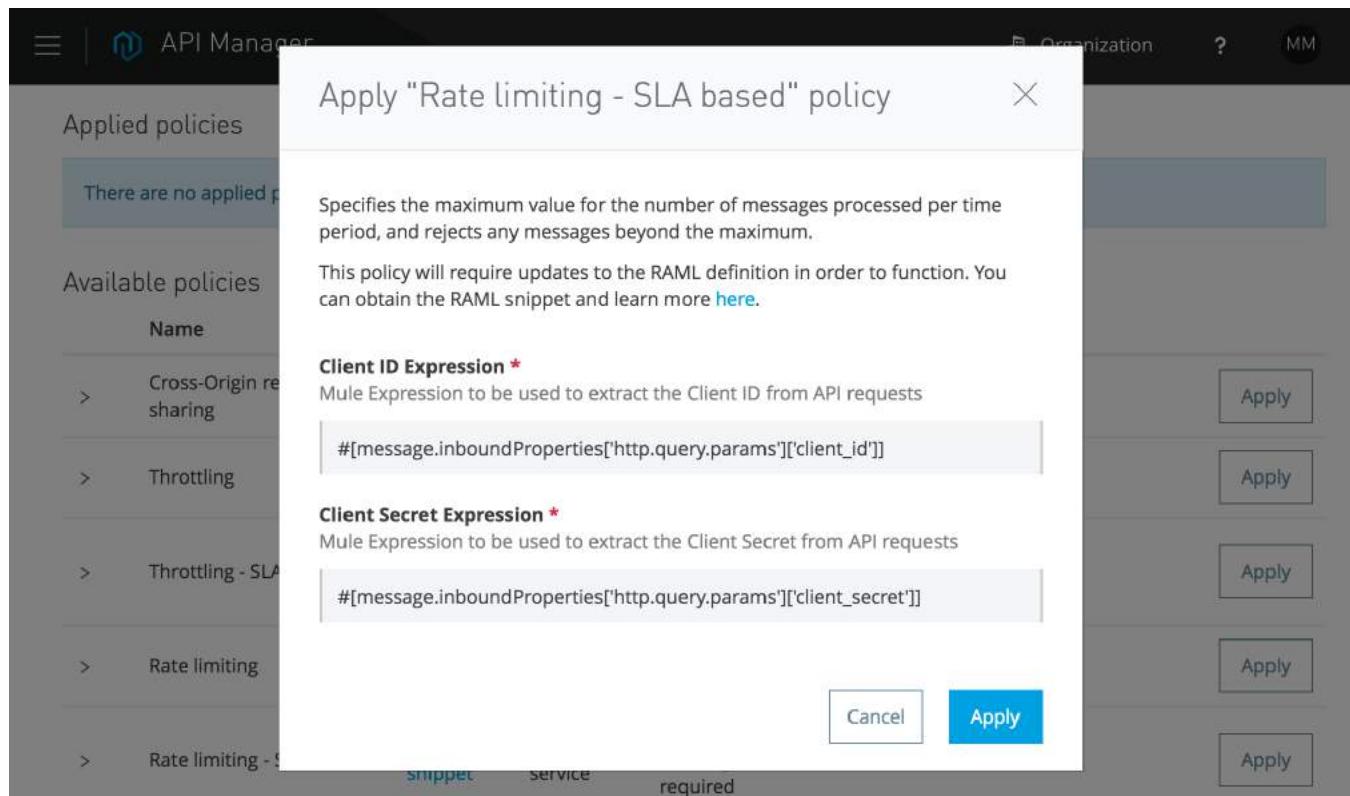
Applied policies				
		Edit policy order		
Name	Category	Fulfils	Requires	
> 1 Rate limiting	Quality of service	Baseline Rate Limiting		Edit Remove

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

16. Click the Apply button for the Rate limiting – SLA based policy.

>	Rate limiting - SLA based	RAML snippet	Quality of service	SLA Rate Limiting, Client ID required	Apply
---	---------------------------	------------------------------	--------------------	---------------------------------------	-----------------------

17. In the Apply "Rate-limiting – SLA based" policy dialog box, look at the expressions and see that a client ID and secret need to be sent with API requests as query parameters.



18. Click Apply.

Test the rate limiting – SLA based policy

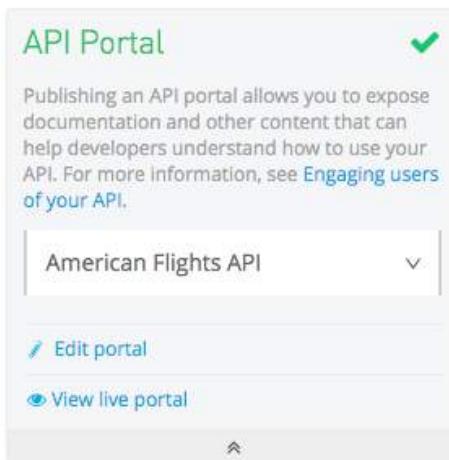
19. In Postman, make another request to your API; you should get a 401 response with a message that a client_id was not provided.

The screenshot shows the Postman interface. The top bar includes 'GET', the URL 'apdev-american-api.cloudhub.io/flights', 'Params', 'Send', and 'Save' buttons. Below the URL, tabs for 'Body', 'Cookies', 'Headers (5)', and 'Tests' are visible. The 'Tests' tab is selected. The status bar shows 'Status: 401 OK' and 'Time: 275 ms'. Under the 'Tests' tab, a message box displays the error: 'i 1 | Unable to retrieve client_id from message'.

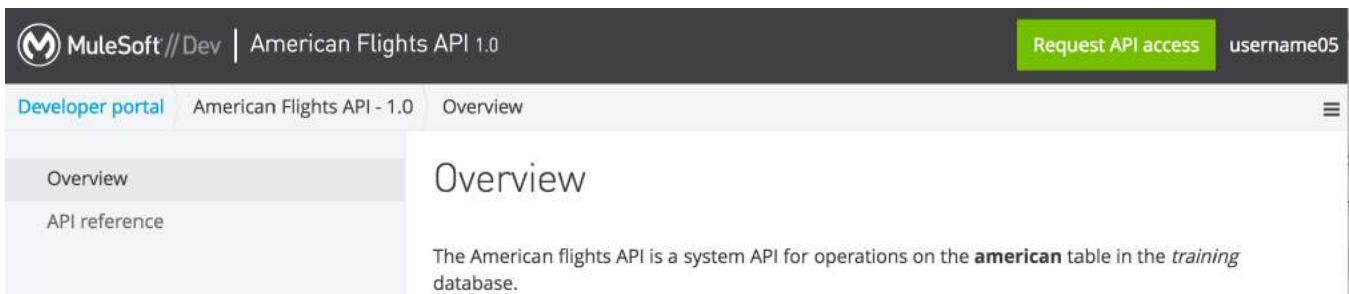
Request access to the API

20. Return to the API details page for American Flights API v1.0 in Anypoint Platform.

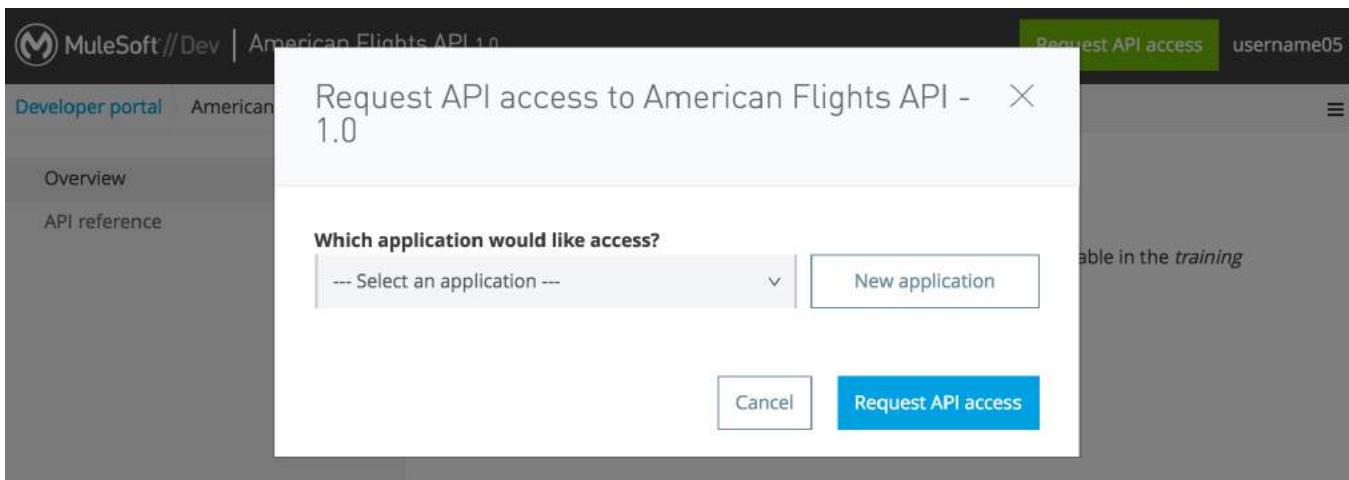
21. In the API Portal section, click the View live portal link.



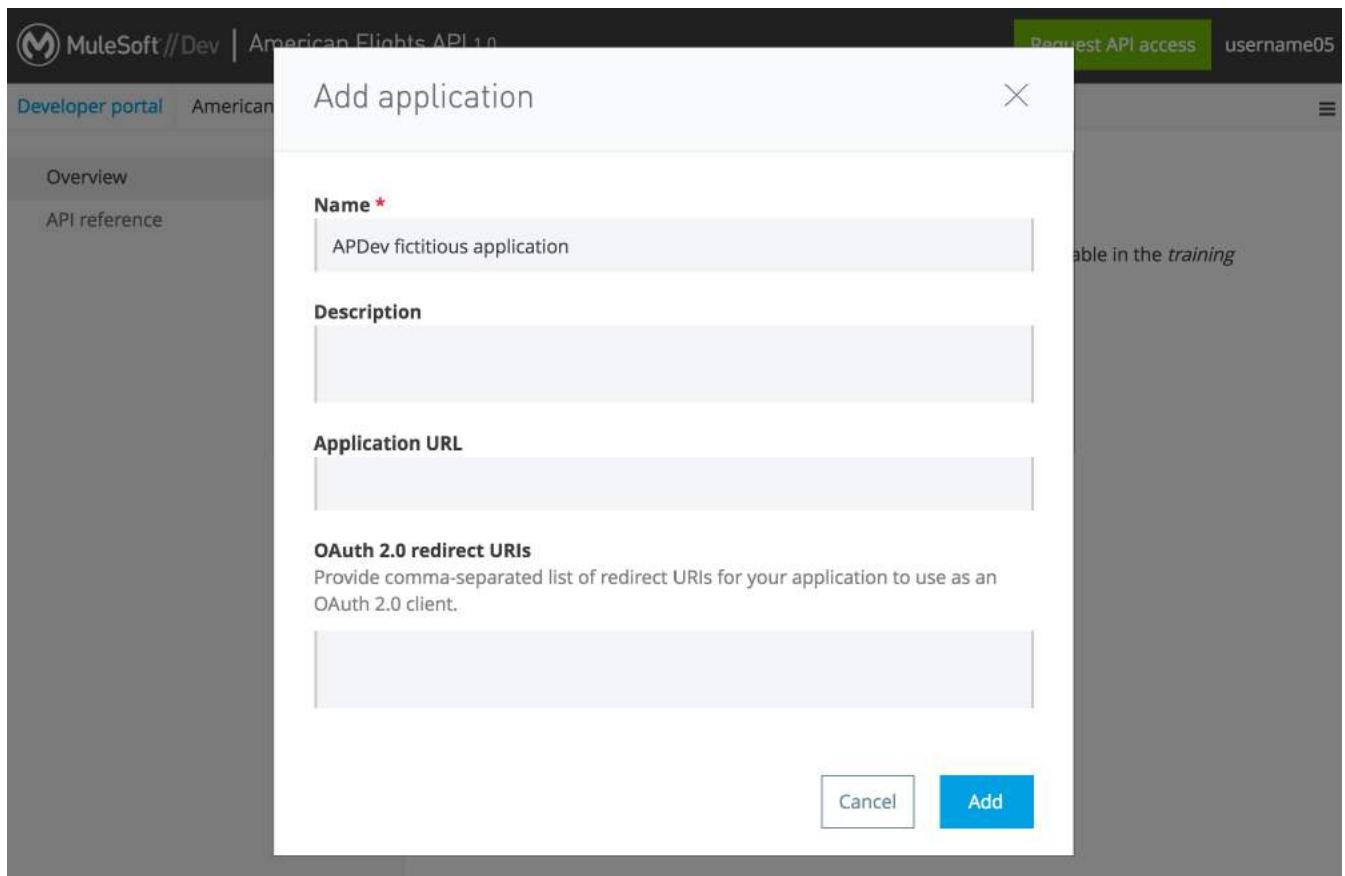
22. In the portal, click the Request API access button.



23. In the Request API access to American Flights API - 1.0 dialog box, click New application.



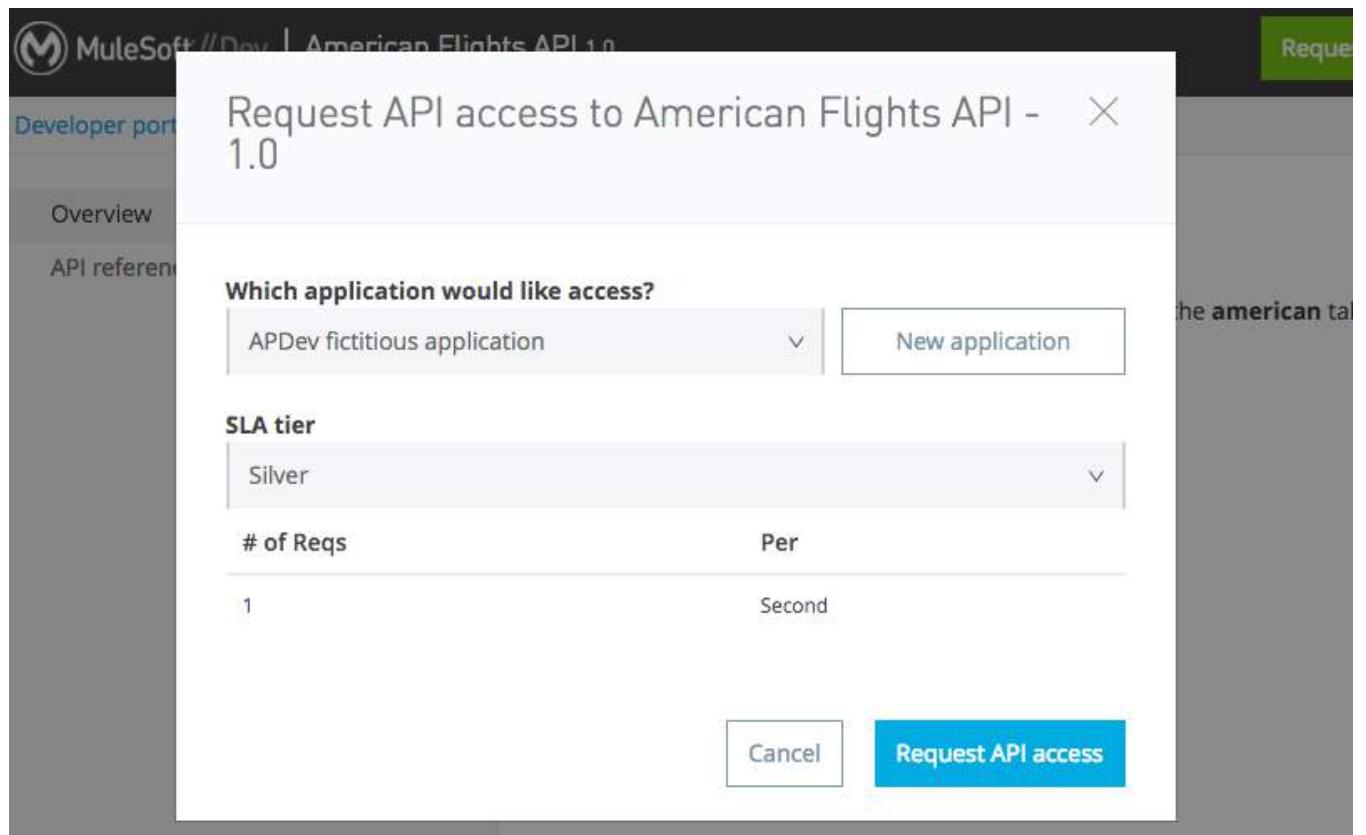
24. In the Add application dialog box, set the name to APDev fictitious application (or some other name).



25. Click Add.

26. In the Request API access to American Flights API – 1.0 dialog box, set the SLA tier to Silver.

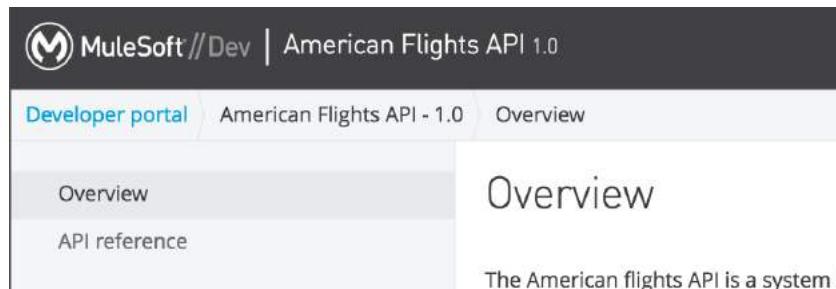
27. Click Request API access.



28. In the Request API access dialog box, click OK.

Retrieve client ID and secret

29. In the main menu bar, click the Developer portal link.



30. In the Developer portal, click My applications.

31. Click the APDev fictitious application link.

Name	Description
APDev fictitious application	

32. Locate the application client ID and client secret.

API name	API version	Current SLA tier	Requested SLA tier	Status	Action
American Flights API	1.0	Silver ⓘ	N/A	Approved	Request tier change

33. Copy the client ID and paste it in the course snippets.txt file under your client_id.

34. Copy the client secret and paste it in the course snippets.txt file under your client_secret.

Approve the request

35. Click the menu button and select Manage your APIs; this should return you to the API administration page in Anypoint Platform.

36. Click the 1.0 link for American Flights API.

37. Select the Applications tab; you should see the request from APDev fictitious application.

Applications	Policies	SLA tiers	Permissions	
Application	Current SLA tier	Requested SLA tier	Status	
> APDev fictitious application	N/A	Silver	Pending	<button>Approve</button> <button>Reject</button> <button>Delete</button>

38. Click Approve.

Test the API SLA

39. In Postman, make sure the URL is set to <http://apdev-american-api-{lastname#}.cloudhub.io/flights>.

40. Click the Params button next to the URL.

41. Set the following key and value values:

- key: client_id
- value: *Get the value for your application from your course snippets.txt file*

42. Add a second key/value pair with the following values:

- key: client_secret
- value: *Get the value for your application from your course snippets.txt file*

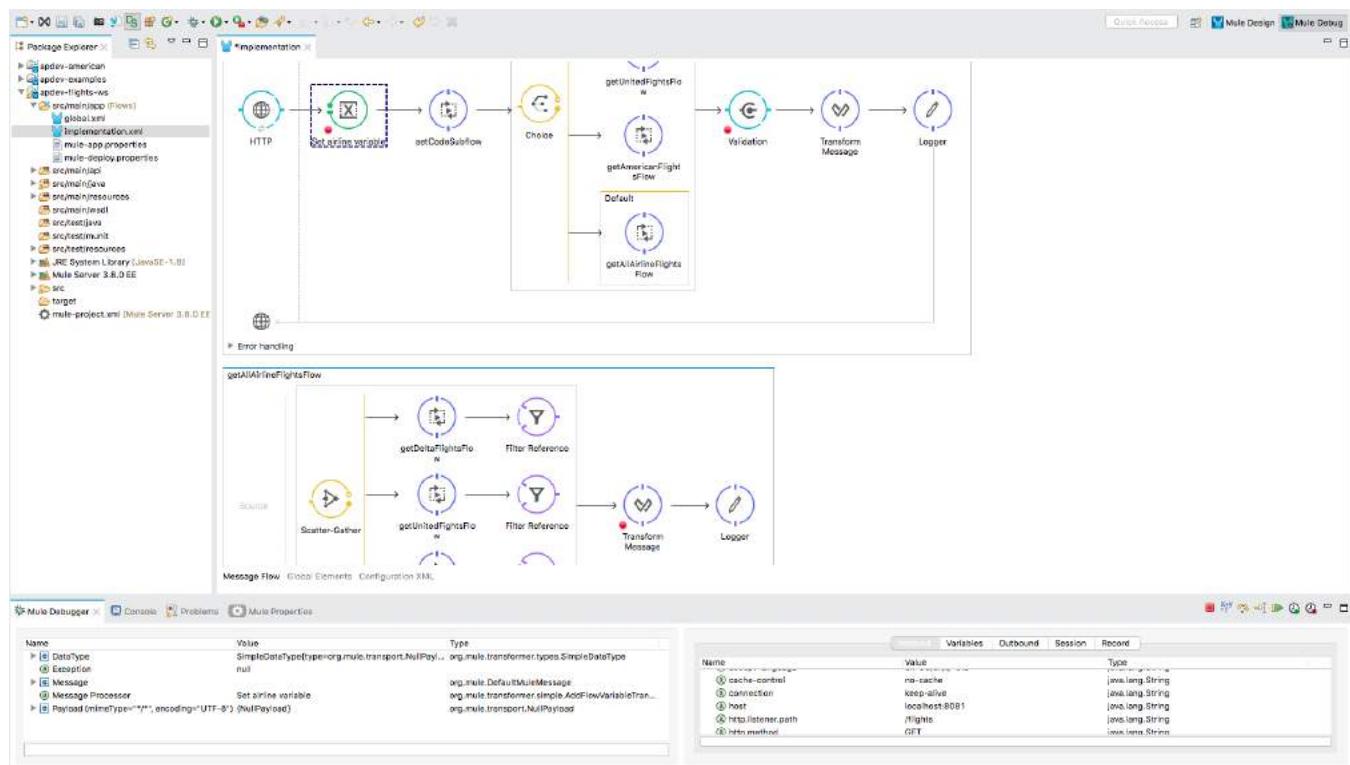
The screenshot shows a Postman interface with a GET request. The URL is set to `apdev-american-api.cloudhub.io/flights?client_id=d1374b15c6`. Below the URL, there are two parameters listed: `client_id` with value `d1374b15c6864c3682ddbed2a247a` and `client_secret` with value `a4cb1787a04c41c3866FD31B95435`. The "Send" button is visible at the top right, and a "Save" button is also present.

43. Click Send; you should get the flight data returned again.

The screenshot shows the Postman interface with a successful API call. The URL is `apdev-american-api.cloudhub.io/flights?client_id=d1374b15c6`. The request method is GET. The response status is 200 OK with a time of 1149 ms. The response body is a JSON array containing one flight record:

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

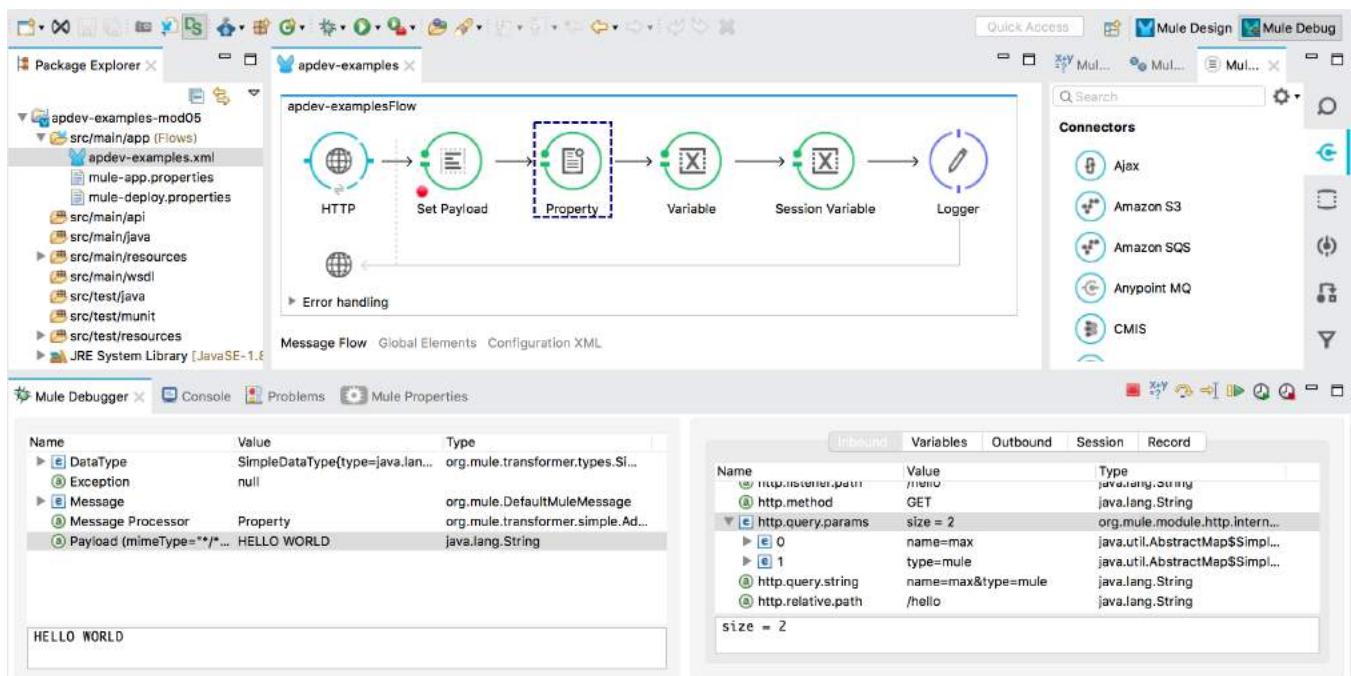
PART 2: Building Applications with Anypoint Studio



Objectives:

- Debug Mule applications.
- Read and write message payloads, properties, and variables using the Mule Expression Language.
- Structure Mule applications using flows, subflows, in-memory message queues, properties files, and configuration files.
- Connect to web services, SaaS applications, files, polled resources, JMS queues, and more.
- Route, filter, and validate messages and handle message exceptions.
- Write DataWeave expressions for more complicated transformations.
- Process individual records in a collection and synchronize data in databases to SaaS applications.

Module 5: Accessing and Modifying Mule Messages



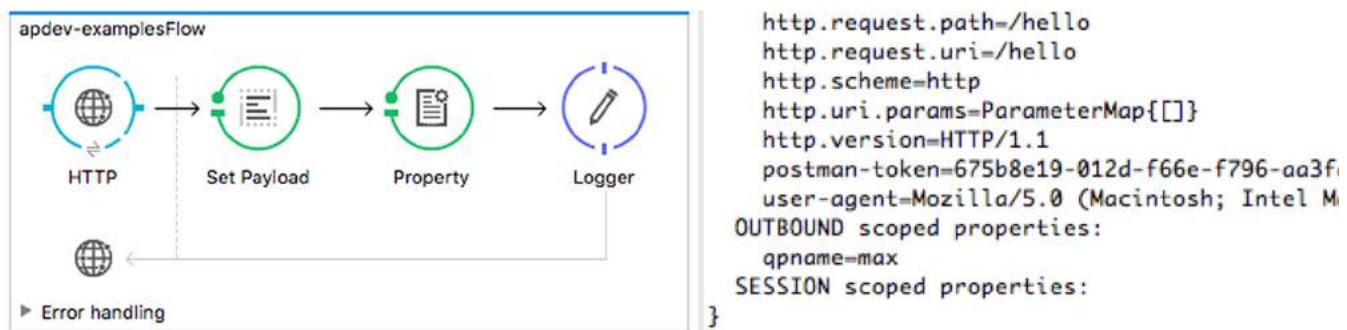
Objectives:

- Log message data.
- Debug Mule applications.
- Read and write message properties.
- Write expressions with Mule Expression Language (MEL).
- Create variables.

Walkthrough 5-1: Set and log message data

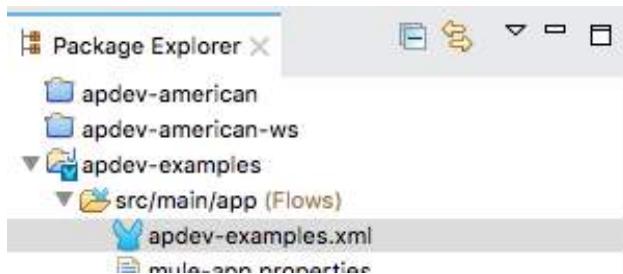
In this walkthrough, you create a new project to use in the next two modules for learning about Mule messages and Mule applications. You will:

- Create a new Mule project with an HTTP Listener endpoint.
- Set the message payload and an outbound property.
- Use a Logger to view message data in the Anypoint Studio console.
- Review message data in Postman.



Create a new Mule project

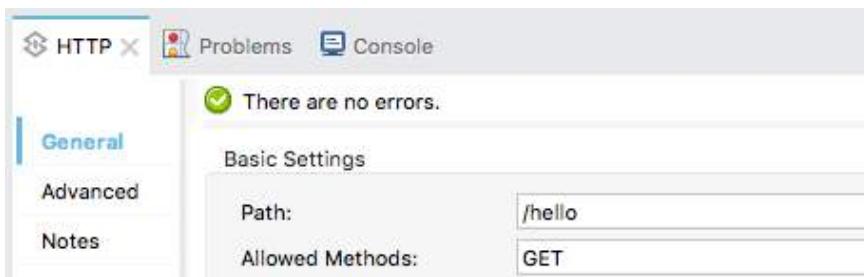
1. Return to Anypoint Studio.
2. Right-click apdev-american-ws and select Close Project.
3. Select File > New > Mule Project.
4. Set the Project Name to apdev-examples and click Finish.



Create an HTTP connector endpoint to receive requests

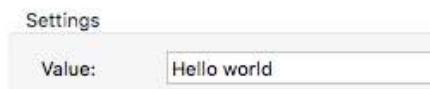
5. Drag an HTTP connector from the Mule Palette to the canvas.
6. In the HTTP properties view, click the Add button next to connector configuration.
7. In the Global Element Properties dialog box, look at the default values and click OK.
8. In the HTTP properties view, set the path to /hello.

9. Set the allowed methods to GET.



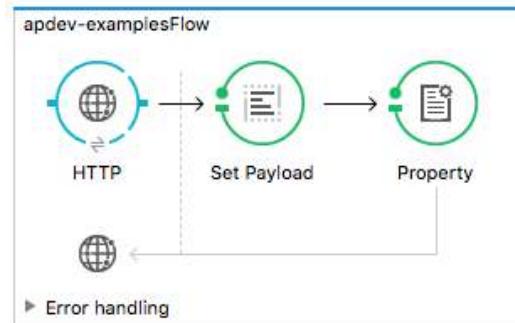
Set the message payload

10. Drag a Set Payload transformer from the Mule Palette into the process section of the flow.
11. In the Set Payload properties view, set the value field to Hello world.

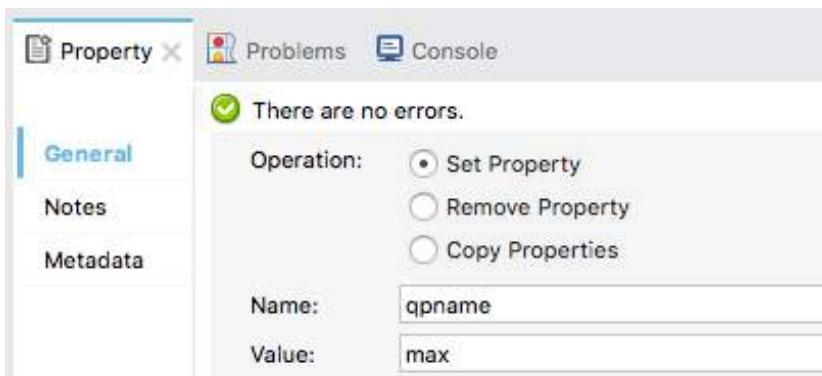


Set an outbound message property

12. Drag a Property transformer from the Mule Palette into the flow.



13. In the Properties view for the Property transformer, select Set Property.
14. Set the name to qpname and the value to max.



Add a Logger

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



Run the application and review message data

16. Run the project.
17. Return to Postman 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 the another tab for later use. Optionally, you could also save it to a Postman collection.

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

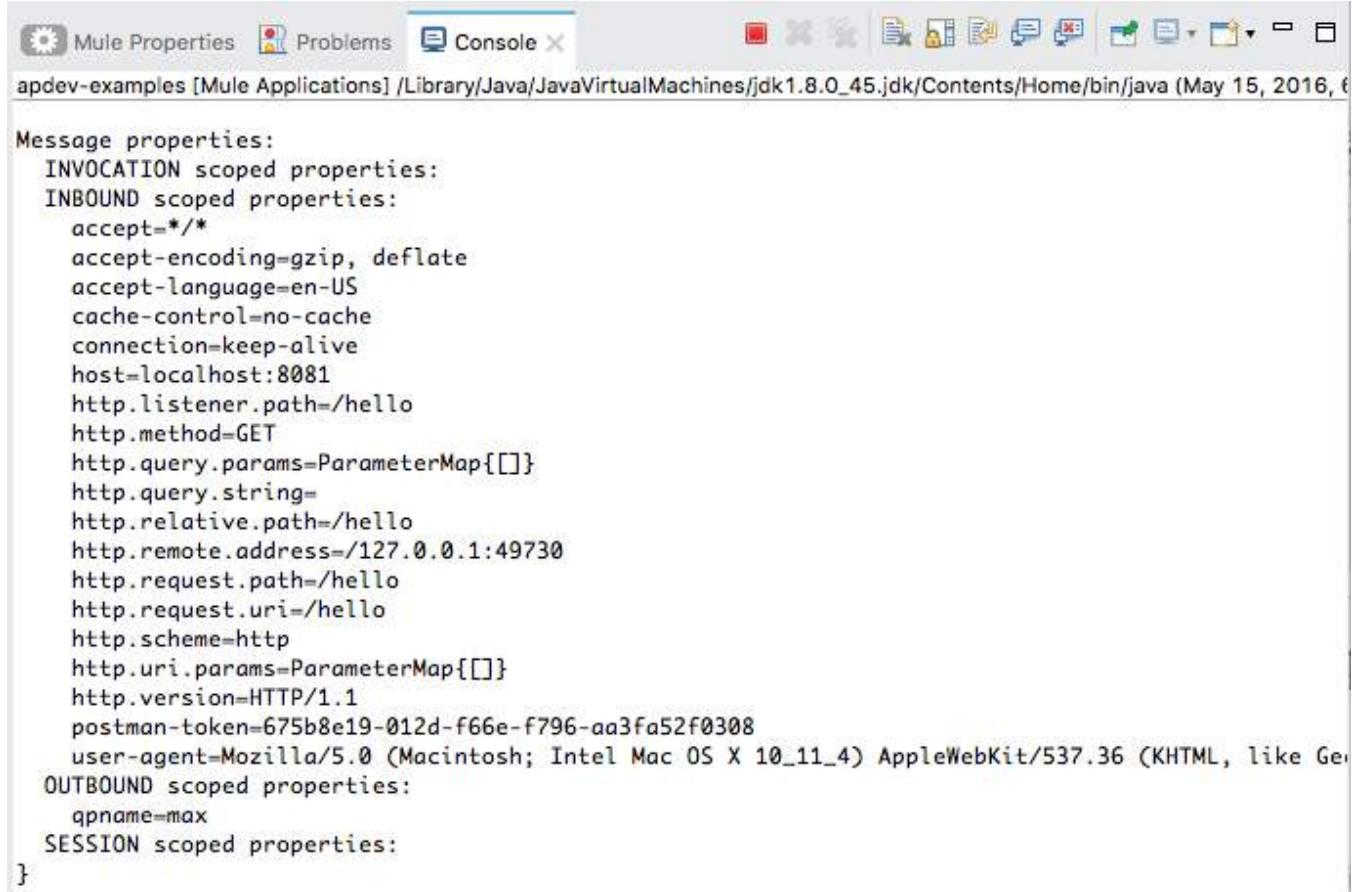
The screenshot shows the Postman interface with a new tab created for the request. The URL is set to `localhost:8081/hello`. The response status is 200 and the time taken is 280 ms. The response body is displayed as "Hello world".

19. Click the Headers link in the response; you should see the qpname property.

The screenshot shows the Headers section of the response. It lists three headers: Content-Length → 11, Date → Mon, 16 May 2016 01:14:41 GMT, and qpname → max.

View message data in the Anypoint Studio console

20. Return to Anypoint Studio and look at the console.
21. Locate the data displayed by using the Logger.
22. Find where the data type of the payload is specified.
23. Review the message inbound properties.
24. Review the message outbound properties.
25. Locate the other two scopes of properties that have no values set.



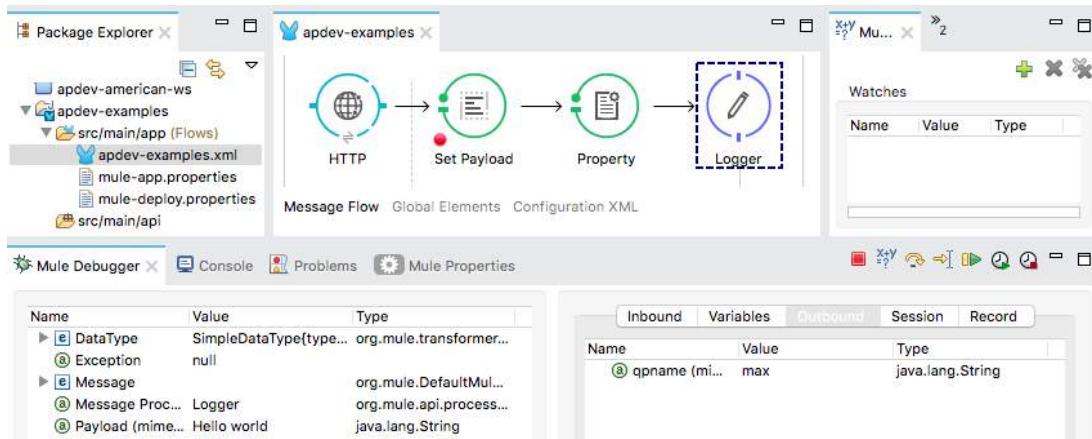
```
Mule Properties Problems Console apdev-examples [Mule Applications] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (May 15, 2016, 6:45:45 PM)  
  
Message properties:  
    INVOCATION scoped properties:  
    INBOUND scoped properties:  
        accept=/*  
        accept-encoding=gzip, deflate  
        accept-language=en-US  
        cache-control=no-cache  
        connection=keep-alive  
        host=localhost:8081  
        http.listener.path=/hello  
        http.method=GET  
        http.query.params=ParameterMap{[]}  
        http.query.string=  
        http.relative.path=/hello  
        http.remote.address=/127.0.0.1:49730  
        http.request.path=/hello  
        http.request.uri=/hello  
        http.scheme=http  
        http.uri.params=ParameterMap{[]}  
        http.version=HTTP/1.1  
        postman-token=675b8e19-012d-f66e-f796-aa3fa52f0308  
        user-agent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko)  
    OUTBOUND scoped properties:  
        qpname=max  
    SESSION scoped properties:  
}
```

26. Stop the project.

Walkthrough 5-2: Debug a Mule application

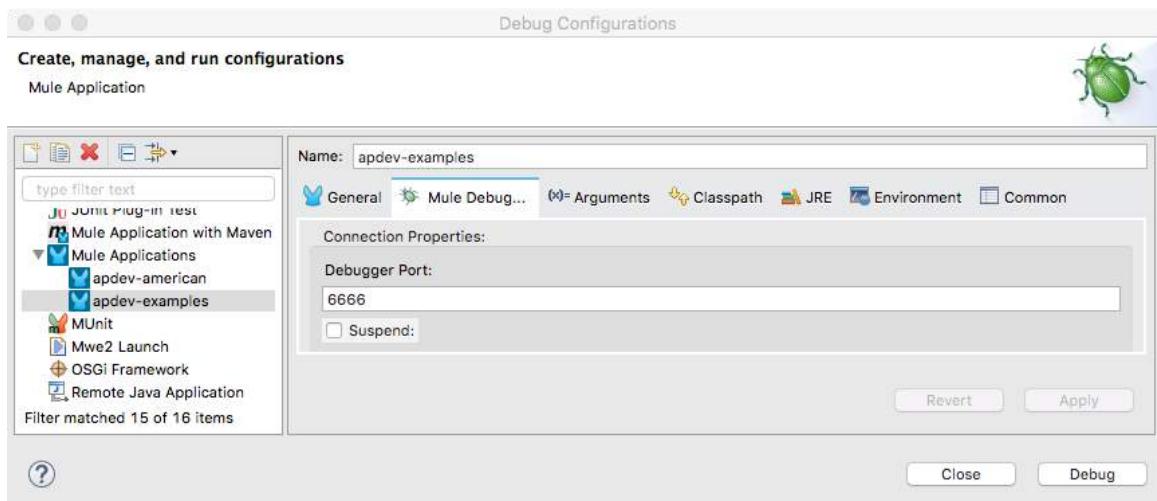
In this walkthrough, you debug and step through the code in a Mule application. You will:

- 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 message properties.
- Pass query parameters to a request and locate them in the Mule Debugger.



Locate the port used by the Mule Debugger

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

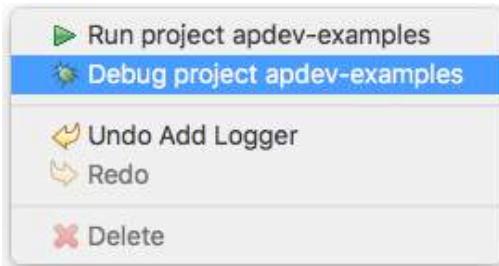
Debug the application

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



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 an Accept incoming network connections dialog box, click Allow.
9. If you get a Confirm Perspective Switch dialog box, select Remember my decision and click Yes.



10. In the Debug perspective, close the MUnit view.

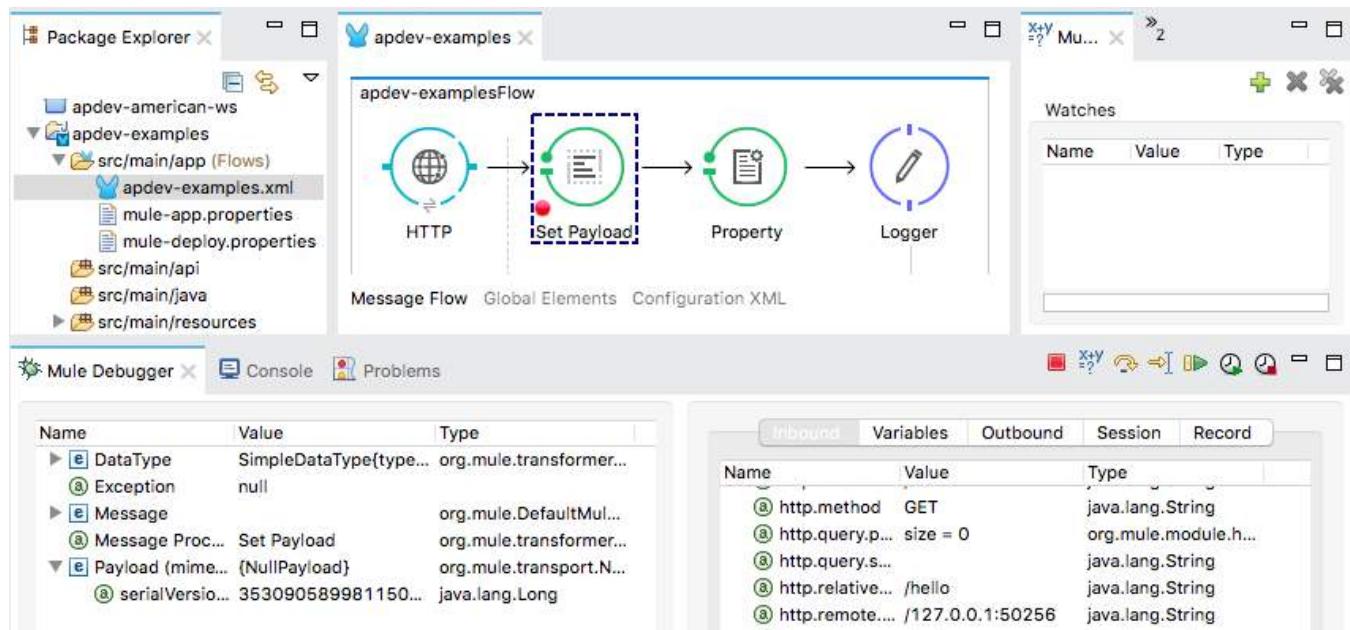
Note: You will not use MUnit in this course. MUnit is covered in the Anypoint Platform Development: Advanced course.

11. Look at the console and wait until the application starts.

12. In Postman, make another request to <http://localhost:8081/hello>.

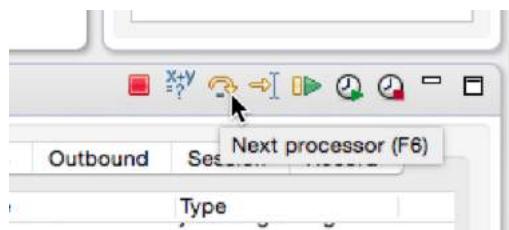
View message properties and variables in the Mule Debugger

13. Return to Anypoint Studio and locate the Mule Debugger view.
14. Drill-down and explore the variables on the left side of the Mule Debugger.
15. Look at the values of the inbound properties on the right side of the Mule Debugger.
16. Select each of the other tabs: Variables, Outbound, Session, Record; you should not see values for any other properties or variables.



Step through the application

17. Click the Next processor button.



18. Look at the new value of the payload; it should be Hello world.
19. Look at the inbound and outbound properties; you should see they have not changed.
20. Click the Next processor button; you should now see the outbound property `qpname`.

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

- Package Explorer:** Shows the project structure with files like `apdev-examples.xml`, `mule-app.properties`, and `mule-deploy.properties`.
- Mule Flow:** A sequence of four processors: `HTTP`, `Set Payload`, `Property`, and `Logger`. The `Logger` processor is highlighted with a dashed blue border.
- Mule Debugger:** A tool for inspecting message variables. It displays two tables:
 - Inbound:** Contains variables like `(DataType: SimpleDataType{type...})`, `(Exception: null)`, `(Message: org.mule.DefaultMul...)`, `(Message Proc...: Logger)`, and `(Payload (mime...): Hello world)`.
 - Outbound:** Contains the variable `(qpname (mi...): max)` of type `java.lang.String`.

21. Step through the rest of the application.

Send query parameters with a request

22. Return to Postman and add a parameter with a key of `name` and a value of `max`.
23. Add a second key / value pair of `type` and `mule`.
24. Click Send.

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** `localhost:8081/hello?name=max&type=mule`
- Params:** A table showing the query parameters:

Name	Value
name	max
type	mule
- Status:** Sending...

25. Return to the Mule Debugger view and locate your query parameters.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
http.method	GET	java.lang.String		
http.query.params	size = 2	org.mule.module.http.inter...		
0	name=max	java.util.AbstractMap\$Sim...		
key	name	java.lang.String		
value	max	java.lang.String		
1	type=mule	java.util.AbstractMap\$Sim...		
http.query.string	name=max&type=mule	java.lang.String		
http.relative.path	/	java.lang.String		
http.remote.address	0:0:0:0:0:0:0:1:58091	java.lang.String		

Use the Mule Expression evaluator

26. Click the Evaluate Mule Expression button.



27. Enter the following expression and press the Enter key.

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

28. Expand the results; you should see the query parameters.

#	[message.inboundProperties.'http.query.params']	▼
Name	Value	Type	
#[message.inbound...	size = 2	org.mule.module.http.inter...	
0	name=max	java.util.AbstractMap\$Sim...	
1	type=mule	java.util.AbstractMap\$Sim...	
key	type	java.lang.String	
value	mule	java.lang.String	

29. Click anywhere outside the expression window to close it.

30. Click the Resume button to execute the rest of the flow.

31. Stop the project.

Switch perspectives

32. Click the Mule Design tab in the upper-right corner of Anypoint Studio to switch perspectives.

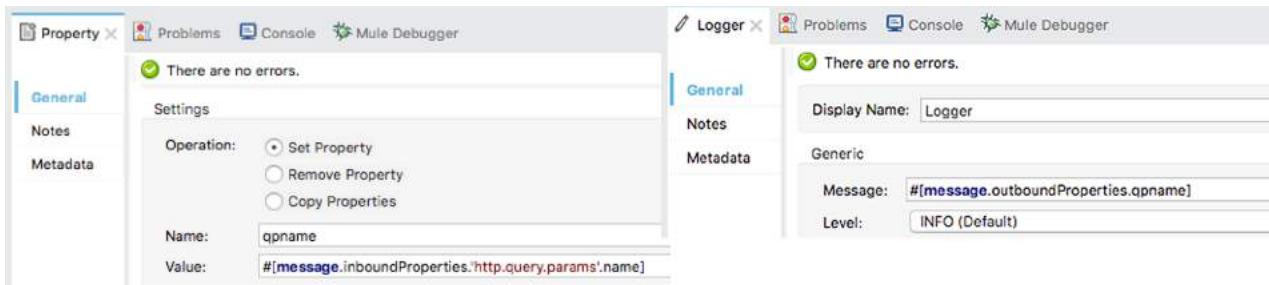


Note: In Eclipse, a perspective is a specific arrangement of views in specific locations. You can rearrange the perspective by dragging and dropping tabs and views to different locations. Use the Window menu in the main menu bar to save and reset perspectives.

Walkthrough 5-3: Read and write message properties using MEL expressions

In this walkthrough, you manipulate message properties. You will:

- Use an expression to set the payload.
- Use an expression to display specific info to the console.
- Use an expression to set an outbound property.
- Use an expression to read an outbound property.



Use an expression to set the payload

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

```
#['Hello world']
```



4. Run the project.
5. In Postman, make a request to <http://localhost:8081/hello> with the query parameters; the application should work as before.
6. In Anypoint Studio, return to the Set Payload expression and use autocomplete to use the `toUpperCase()` method to return the value in upper case.

```
#['Hello world'].toUpperCase()
```

Note: Press Ctrl+Space to trigger autocomplete if it does not appear.

7. Click the Apply Changes button in the upper-right corner of the properties view to redeploy the application.
8. In Postman, send the same request; the return string should now be in upper case.

The screenshot shows the Anypoint Studio Properties view for a Logger component. The URL is set to 'localhost:8081/hello?name=max&type=mule'. The Body tab is selected, showing the response 'HELLO WORLD' in the preview area.

Use an expression to display specific info to the console

9. In Anypoint Studio, navigate to the Properties view for the Logger component.
10. Set the message to display the http.query.params property of the message inbound properties.

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

The screenshot shows the Anypoint Studio Properties view for a Logger component. The message is set to '#[message.inboundProperties.'http.query.params']', level is set to INFO (Default), and category is empty.

11. Click Apply Changes to redeploy the application.
12. In Postman, send the same request.
13. Return to the Anypoint Studio console; you should see a ParameterMap object listed.

```
*****
* apdev-examples          * default           * DEPLOYED      *
*****
INFO 2016-05-15 18:48:06,358 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: ParameterMap{[name=[max], type=[mule]]}
```

14. Modify the Logger to display one of the query parameters.

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

15. Click Apply Changes to redeploy the application.
16. In Postman, send the same request.

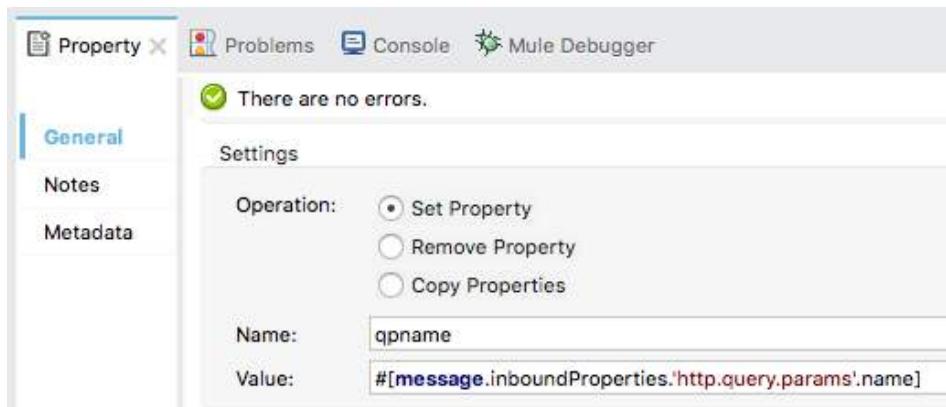
17. Return to the console; you should now see the value of the parameter displayed.

```
INFO 2016-05-15 18:48:41,451 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: max
```

Use an expression to set an outbound property

18. In the Properties view for the Property transformer, change the value to an expression that evaluates the name query parameter.

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



19. Modify the Logger to display the value of this outbound property.

```
##[message.outboundProperties.qpname]
```

20. Click Apply Changes to redeploy the application.

21. In Postman, send the same request.

22. Return to the console; you should see the value of your variable displayed.

```
INFO 2016-05-15 18:50:12,742 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: max
```

23. Stop the project.

Walkthrough 5-4: Read and write variables

In this walkthrough, you create flow and session variables. You will:

- Use the Variable transformer to create a flow variable.
- Use the Session transformer to create a session variable.
- Use the Mule Debugger to see their values.



Create a flow variable

1. Return to apdev-examples.xml.
2. Add a Variable transformer between the Property and Logger processors.



3. In the Variable properties view, select Set Variable.
4. Set the name to qptype and the value to your second query parameter, type.

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

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

5. Modify the Logger to also display the value of this new variable.

Name: #[message.outboundProperties.qpname] Type: #[flowVars.qptype]

Debug the application

6. Debug the project.
7. In Postman, send the same request with the parameters.
8. In the Mule Debugger, select the Variables tab.
9. Step to the Logger; you should see your new flow variable.

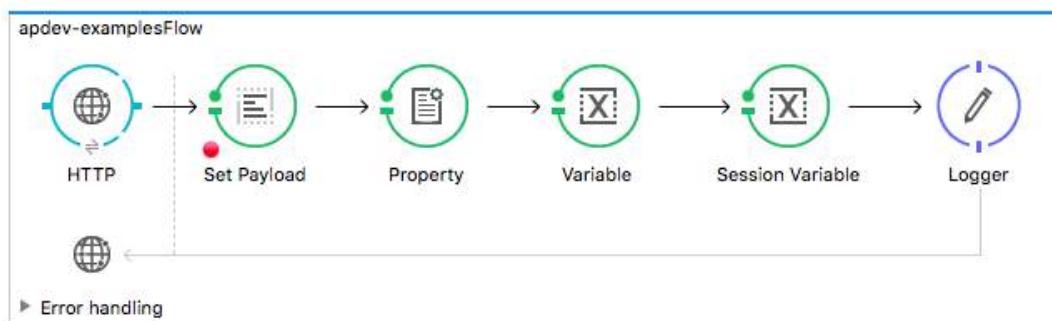
Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
③ qptype (mimeType="*/*", enc...)	mule	java.lang.String		

10. Click the Resume button.
11. Look at the console; you should see the value of your outbound property and the value of your new flow variable displayed.

```
INFO 2016-05-15 18:57:51,141 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: Name: max Type: mule
```

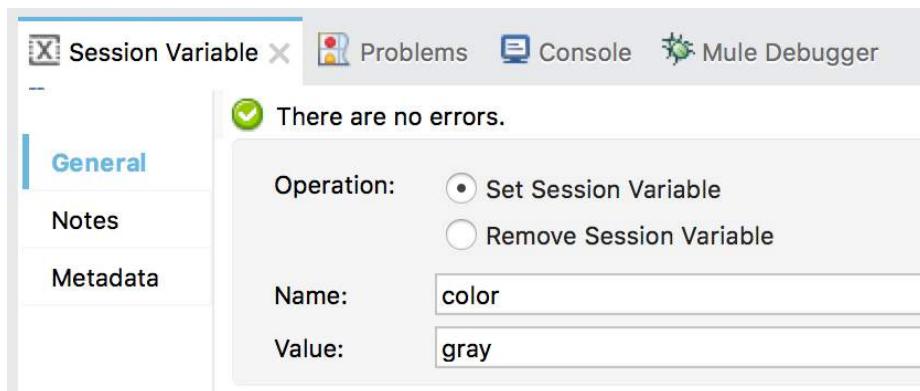
Create a session variable

12. Add a Session Variable transformer between the Variable and Logger processors.



13. In the Session Variable Properties view, select Set Session Variable.

14. Set the name to color and give it any value, static or dynamic.



15. Modify the Logger to also display the session variable.

```
Name: #[message.outboundProperties.qpname] Type: #[flowVars.qptype]  
Color: #[sessionVars.color]
```

Debug the application

16. Click Apply Changes to redeploy the application in debug mode.
17. In Postman, send the same request.
18. In the Mule Debugger, select the Session tab.
19. Step to the Logger; you should see your new session variable.

Inbound	Variables	Outbound	Session	Record
Name			Value	Type
	③ color (mimeType="*/*", encoding="null")		gray	java.lang.String

20. Click the Resume button.
21. Look at the console; you should see the value of your session variable displayed.

```
INFO 2016-05-15 19:00:03,872 [[apdev-examples].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: Name: max Type: mule Color: gray
```

22. Stop the project.

Note: You will explore the persistence of these variables in the next module.

Module 6: Structuring Mule Applications

```
apdev-examples
  src/main/app (Flows)
    accounts.xml
    apdev-examples.xml
    global.xml
    mule-app.properties
    mule-deploy.properties

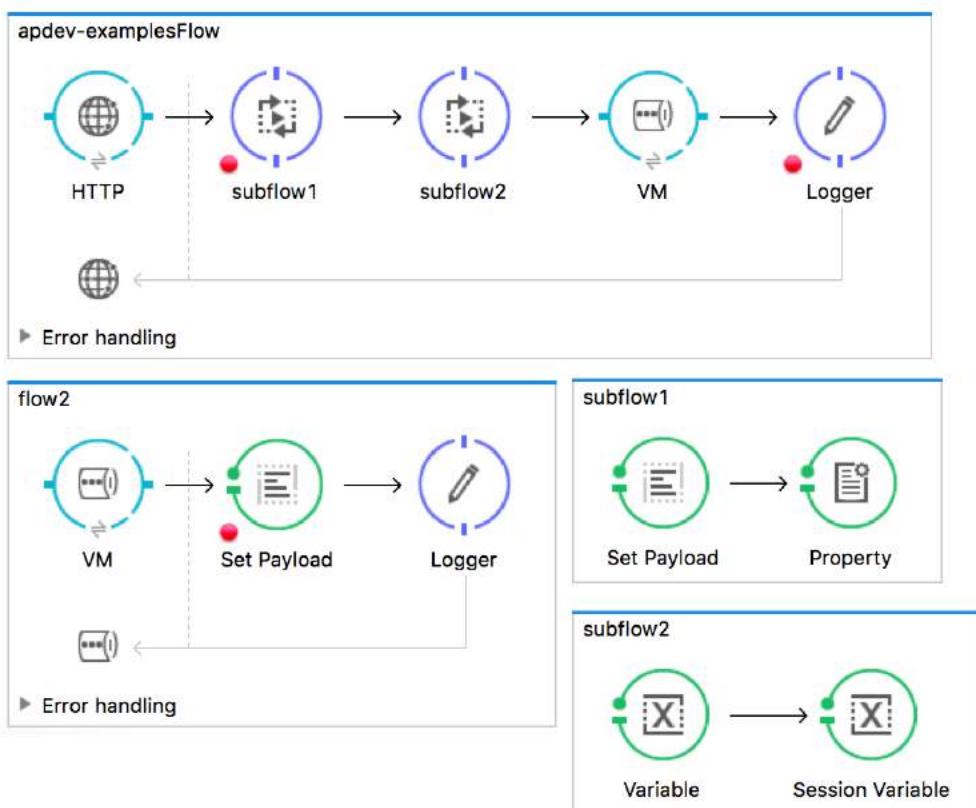
apdev-flights-ws
  src/main/app (Flows)
    global.xml
    implementation.xml
    mule-app.properties
    mule-deploy.properties

src/main/api
  api.raml

src/main/java
  com.mulesoft.training
    Flight.java

src/main/resources
  flights-DEV.properties
  log4j2.xml
  src/main/wsdl
  src/test/java
  src/test/munit

src/test/resources
  flight-example.json
  flights-example.json
```



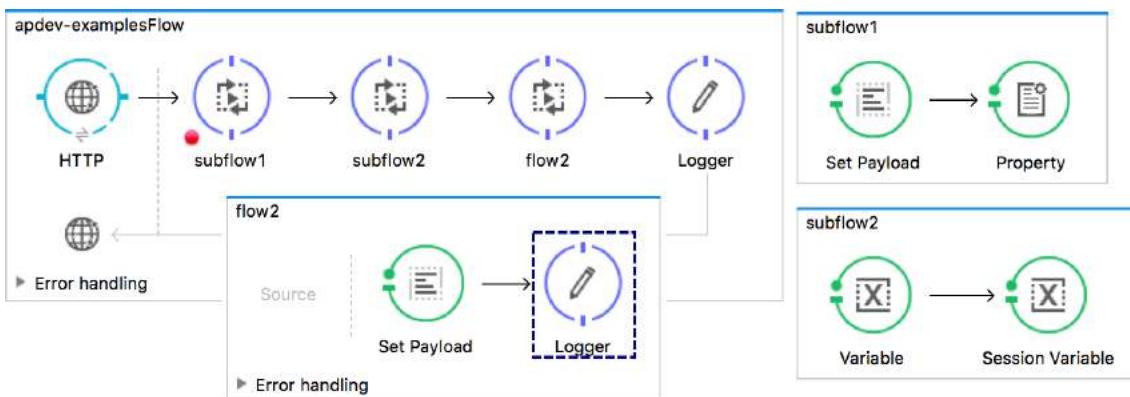
Objectives:

- Create and reference flows and subflows.
- Pass messages between flows using the Java Virtual Machine (VM) transport.
- Investigate variable persistence through subflows and flows and across transport barriers.
- Encapsulate global elements in separate configuration files.
- Explore the files and folder structure of a Mule project.

Walkthrough 6-1: Create and reference flows and subflows

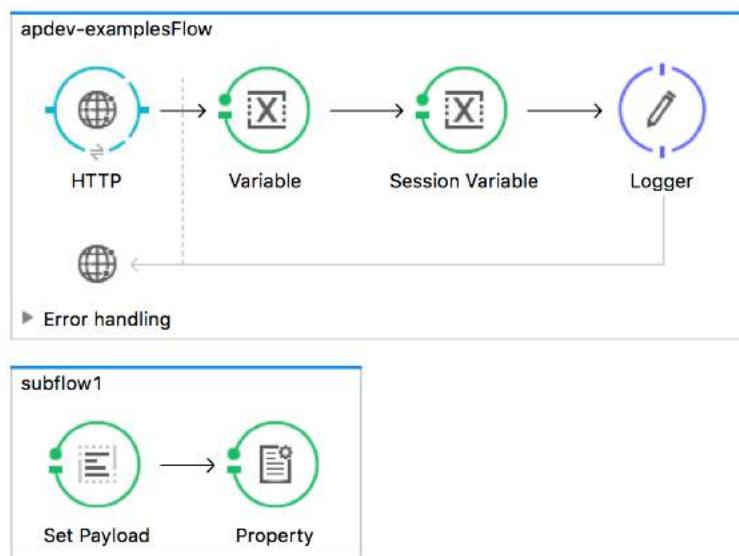
In this walkthrough, you continue to work with apdev-examples.xml. You will:

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



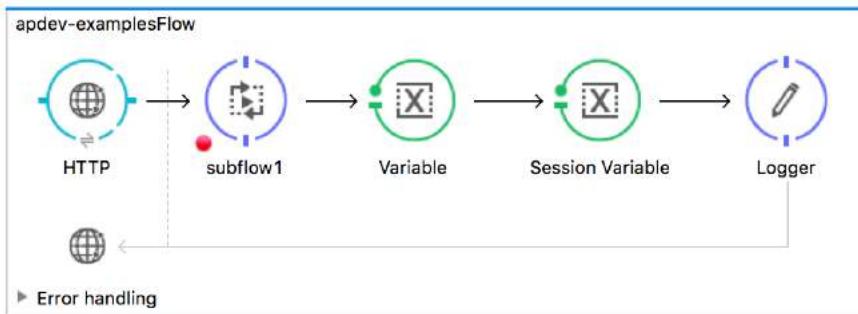
Create a subflow

- Return to apdev-examples.xml in Anypoint Studio.
- Drag a Sub Flow scope from the Mule Palette and drop it beneath the existing flow in the canvas.
- Select the Set Payload and Property transformers in apdev-examplesFlow and drag them into the subflow.
- Change the name of the subflow to subflow1.



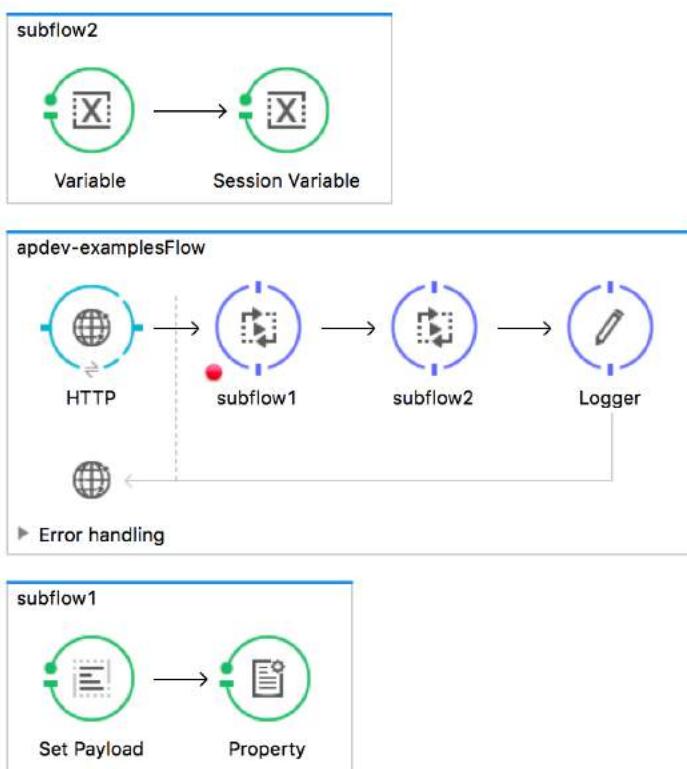
Reference a subflow

5. Drag a Flow Reference component from the Mule Palette and drop it into the apdev-examplesFlow between the HTTP Listener endpoint and the Variable transformer.
6. In the Flow Reference properties view, set the flow name to subflow1.



Extract processors into a subflow

7. Right-click the Variable and Session Variable transformers and select Extract to > Sub Flow.
8. In the Extract Flow dialog box, set the flow name to subflow2.
9. Leave the target Mule configuration set to current and click OK.
10. Look at the new Flow Reference Properties view; the flow name should already be set to subflow2.



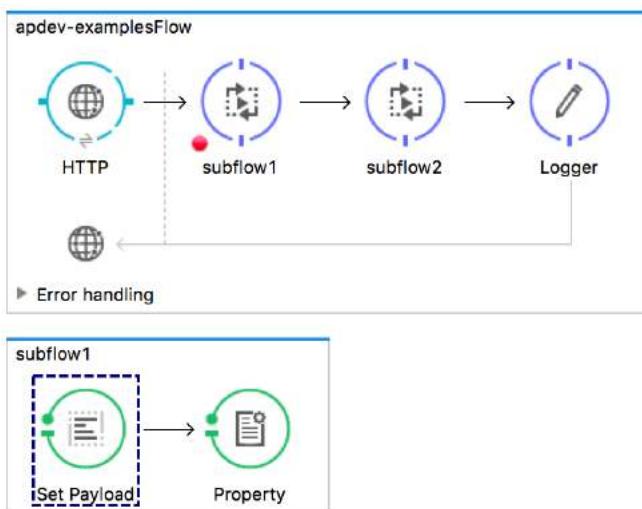
11. Drag subflow2 below subflow1 in the canvas.

Debug the application

12. Debug the project.

13. In Postman, send the same request to <http://localhost:8081/hello?name=max&type=mule>.

14. In the Mule Debugger, step through the application, watching messages move into and out of the subflows.



15. In Postman, send the same request again.

16. In the Mule Debugger, step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables in each of the flows.

Create a second flow

17. Drag a Flow scope element to the canvas and drop it beneath the existing flows.

18. Change the flow name to flow2.

19. Add a Set Payload transformer to the process section of the flow.

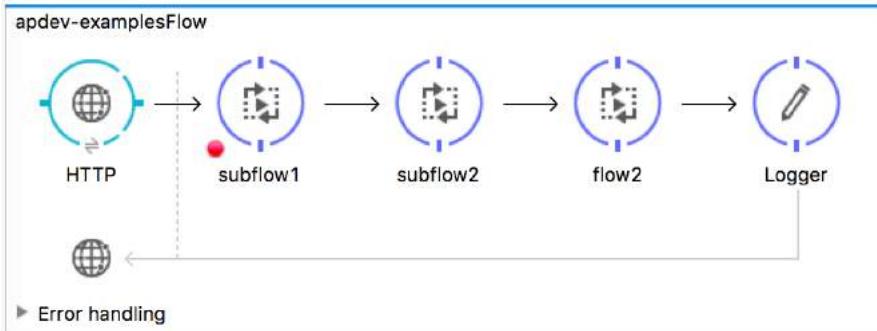
20. In the Set Payload properties view, set the value to Goodbye.

21. Add a Logger after the Set Payload transformer.



Reference a flow

22. In apdev-examplesFlow, add a Flow Reference component between the subflow2 Flow Reference and the Logger.
23. In the Flow Reference Properties view, set the flow name to flow2.



Debug the application

24. Save the file to redeploy the application in debug mode.
25. In Postman, send the same request.
26. In the Mule Debugger, step through the application, watching the flow move into and out of the second flow; at the end, you should see the payload is the value set in the second flow.

The screenshot shows the Mule Debugger interface with the flow 'apdev-examplesFlow' open. The flow structure is identical to the one in the previous diagram. The 'Logger' component is highlighted with a dashed blue box. Below the flow, the 'Message Flow' tab is selected, showing the following variable values:

Name	Value	Type
(DataType)	SimpleDataType{ty...}	org.mule.transfor...
Exception	null	
(Message)		org.mule.DefaultM...
(Message Pr...)	Logger	org.mule.api.proce...
(Payload (mi...)	Goodbye	java.lang.String

On the right, the 'Inbound' tab of the variable viewer shows:

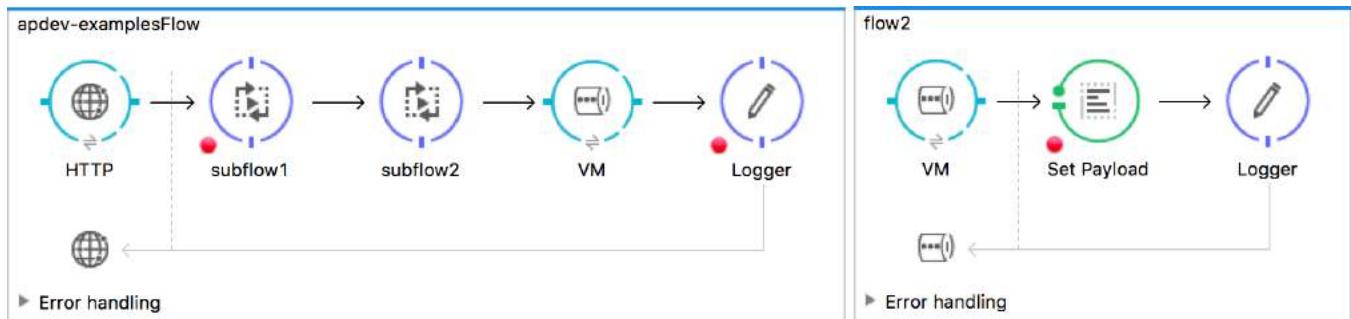
Name	Value	Type
qptype (mime...	mule	java.lang.Stri

27. In Postman, send the same request again.
28. In the Mule Debugger, step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables in each of the flows.
29. Stop the project.

Walkthrough 6-2: Pass messages between flows using the Java Virtual Machine (VM) transport

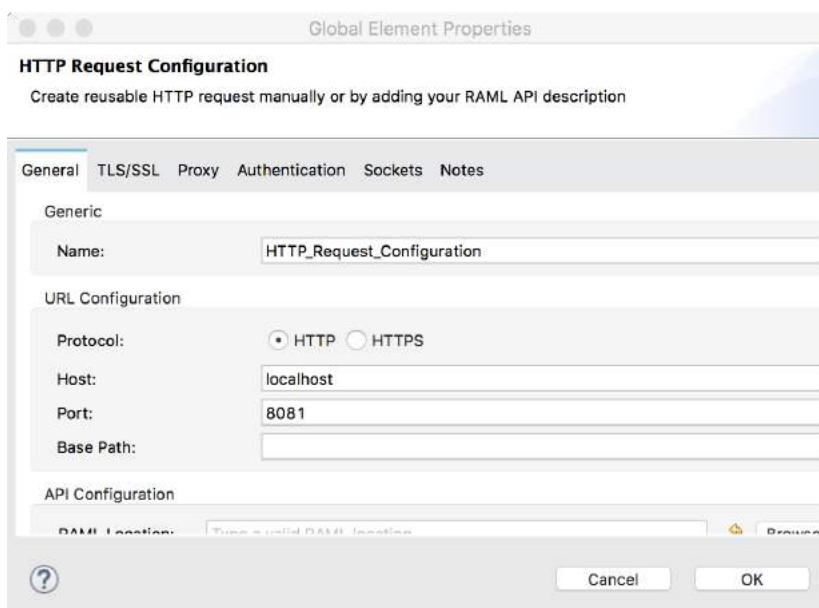
In this walkthrough, you continue to work with the apdev-examplesFlow. You will:

- Pass messages through an HTTP transport barrier.
- Pass messages between flows using the VM transport.
- Explore variable persistence across these transport barriers.

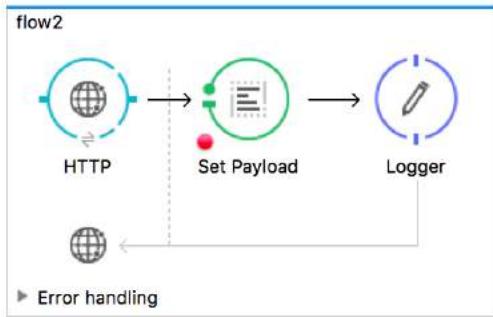


Create an HTTP transport barrier

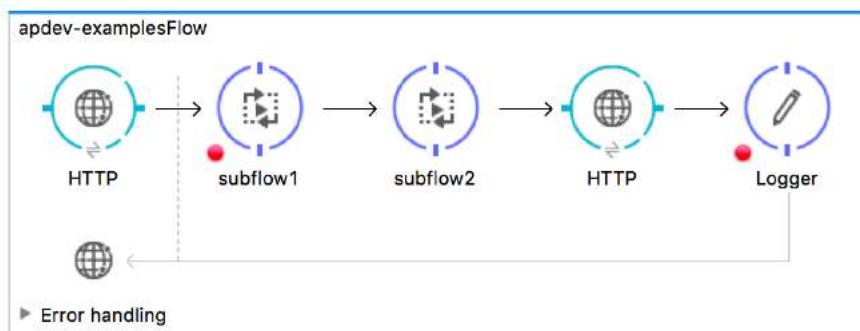
1. Return to apdev-examples.xml.
2. Switch to the Global Elements view.
3. Click Create.
4. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.
5. In the Global Elements dialog box, set the host to localhost, the port to 8081, and click OK.



6. Return to the Message Flow view.
7. Drag an HTTP connector into the Source section of flow2.
8. In the HTTP properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.
9. Set the path to `/flow2` and the allowed methods to GET.
10. Add a breakpoint to the Set Payload transformer in flow2.



11. Add an HTTP Request endpoint after the `flow2` reference in `apdev-examplesFlow`.
12. Delete the `flow2` reference.
13. In the HTTP properties view, set the connector configuration to `HTTP_Request_Configuration`.
14. Set the path to `/flow2` and the method to GET.
15. Add a breakpoint to the `Logger`.



Debug the application

16. Debug the application.
17. In Postman, send the same request.
18. In the Mule Debugger, step into `flow2`.

19. Locate the qpname property in the inbound properties.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
④ http.remote.add... /127.0.0.1:8080		java.lang.String		
③ http.request.path	/flow2	java.lang.String		
③ http.request.uri	/flow2	java.lang.String		
③ http.scheme	http	java.lang.String		
③ http.uri.params	size = 0	org.mule.module....		
③ http.version	HTTP/1.1	java.lang.String		
③ qpname	max	java.lang.String		
③ user-agent	AHC/1.0	java.lang.String		

20. Look at the outbound properties, the flow variables, and the session variables.

Note: Session variables are persisted across some but not all transport barriers. As you see here, they are not propagated across the HTTP transport barrier.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		

21. Step through the flow until the message returns to apdev-examplesFlow.

22. Look at the inbound and outbound properties and the flow and session variables.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
④ qpype (mim... mule		java.lang.String		

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
④ color (mime... gray		java.lang.String		

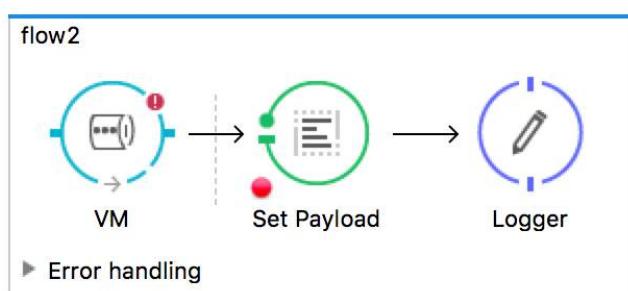
23. Click Resume.

24. Stop the project.

Create a VM transport barrier

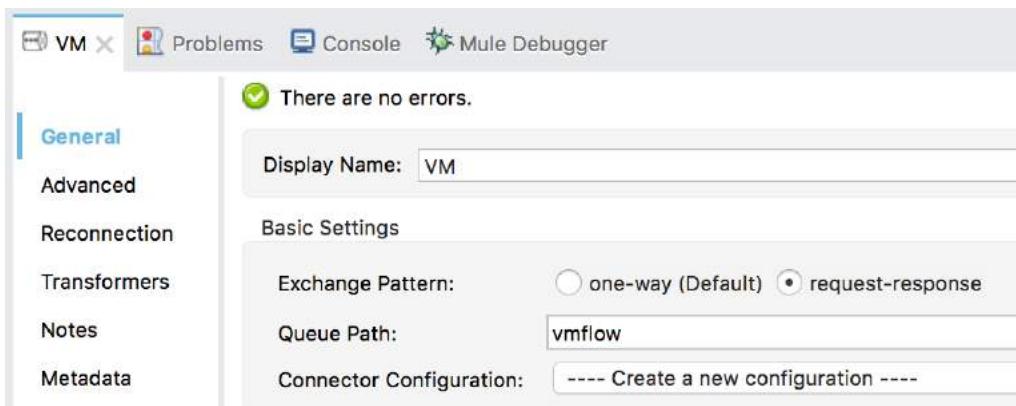
25. In flow2, delete the HTTP Listener endpoint.

26. Drag a VM connector from the Mule Palette into the source section of flow2.



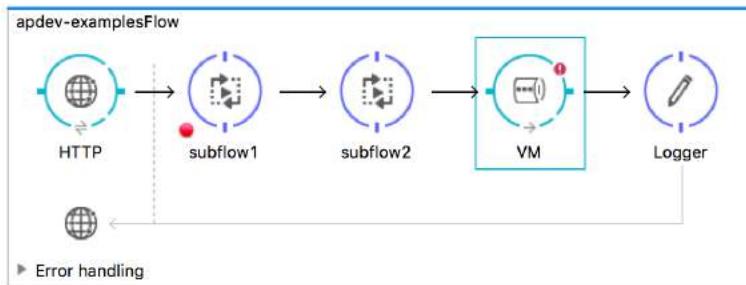
27. In the VM properties view, set the exchange pattern to request-response.

28. Set the queue path to vmflow.



29. In apdev-examplesFlow, add a VM connector endpoint after the HTTP Request endpoint.

30. Delete the HTTP Request endpoint.



31. In the VM properties view, set the exchange pattern to request-response.

32. Set the queue path to vmflow.

Debug the application

33. Debug the project.

34. In Postman, send the same request.

35. In the Mule Debugger, step through the application into flow2.

36. Look at the inbound and outbound properties and the flow and session variables.

The Mule Debugger displays four tabs: Inbound, Variables, Outbound, and Session. Below each tab is a table showing variable properties.

Name	Value	Type
MULE_ENDPOINT	vm://vmflow	java.lang.String
MULE_ORIGINATI...	endpoint.vm.vmflow	java.lang.String
MULE_SESSION	rOOABXNyACNvcm...	java.lang.String
qname	max	java.lang.String

Name	Value	Type
MULE_COR...	-1	java.lang.Integer
MULE_COR...	-1	java.lang.Integer

Name	Value	Type
color (mime...)	gray	java.lang.String

Note: Session variables are persisted across some but not all transport barriers. As you see here, they are propagated across the VM transport barrier.

37. Step through the flow until the message returns to apdev-examplesFlow.
38. Look at the inbound and outbound properties and the flow and session variables.

Inbound	Variables	Outbound	Session	Record
Session Variables				
Name		Value	Type	
@ MULE_CORRELATION_ID		-1	java.lang.Integer	
@ MULE_CORRELATION_SEQUENCE		-1	java.lang.Integer	
@ MULE_SESSION		rO0ABXNyACNvcm...	java.lang.String	
Inbound Variables				
Name		Value	Type	
@ MULE_CORRELATION_ID		-1	java.lang.Integer	
@ MULE_CORRELATION_SEQUENCE		-1	java.lang.Integer	
@ MULE_SESSION		rO0ABXNyACNvcm...	java.lang.String	
Outbound Variables				
Name		Value	Type	
@ qptype (mime-type)		mule	java.lang.String	
Session Variables				
Name		Value	Type	
@ color (mime-type)		gray	java.lang.String	

39. Step through the rest of the application and stop the project.

Walkthrough 6-3: Encapsulate global elements in a separate configuration file

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

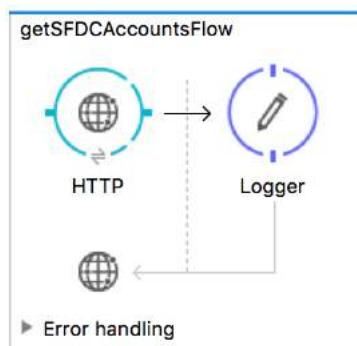
- Create a new configuration file with an endpoint that uses an existing global element.
- Create a configuration file global.xml for just global elements.
- Move the existing global elements to global.xml.



Create a new configuration file

1. Return to the apdev-examples project.
2. Create a new Mule configuration file called accounts.xml.
3. Drag out an HTTP Listener to the canvas.
4. In the HTTP properties view, set the connector configuration to the existing configuration.
5. Set the path to /sfdc and the allowed methods to GET.
6. Add a Logger component to the flow.
7. Change the name of the existing flow to getSFDCAccountsFlow.

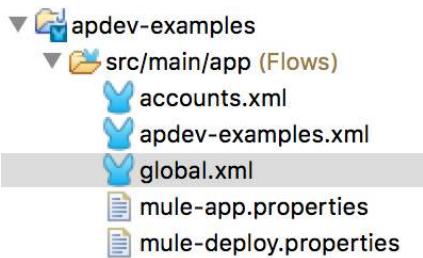
Note: You will use this flow in a later module to retrieve data from Salesforce.



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

Create a global configuration file

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



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

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

Remove the existing global elements

12. Return to apdev-examples.xml.

13. Switch to the Global Elements view and see there are two configurations.

The screenshot shows the 'Global Mule Configuration Elements' view in the Mule Studio interface. It lists two configurations under 'Type': 'HTTP Listener Configuration' and 'HTTP Request Configuration'. There are 'Create' and 'Edit' buttons at the top right.

14. Switch to the Configuration XML view.

15. Select and cut the two configuration elements defined before the flows.

The screenshot shows the Configuration XML view with the code editor. Lines 10 through 15 are selected, containing the definitions for the 'HTTP Listener Configuration' and 'HTTP Request Configuration' global elements.

```
<http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081"/>
<http:request-config name="HTTP_Request_Configuration" host="localhost" port="8081"/>
<flow name="apdev-examplesFlow">
    <http:listener config-ref="HTTP_Listener_Configuration" path="/hello" />
    <flow-ref name="subflow1" doc:name="subflow1"/>
</flow>
```

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

16. Return to the Message Flow view.

Move the global elements to a new configuration file

17. Return to global.xml.
18. Paste the global elements you cut to the clipboard between the start and end mule tags.



```
8  http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/hi
9  http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/c
10
11    <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" por
12      <http:request-config name="HTTP_Request_Configuration" host="localhost" por
13
14  </mule>
15
```

19. Switch to the Global Elements view; you should see the two configurations.

Test the application

20. Save all the files; any problems should disappear.
21. Return to apdev-examples.xml.
22. Double-click the HTTP Listener connector in apdev-examplesFlow; the connector configuration should still be set to HTTP_Listener_Configuration, which is now defined in global.xml.
23. Run the project.
24. In Postman, send the same request.; you should still get a response of Goodbye.

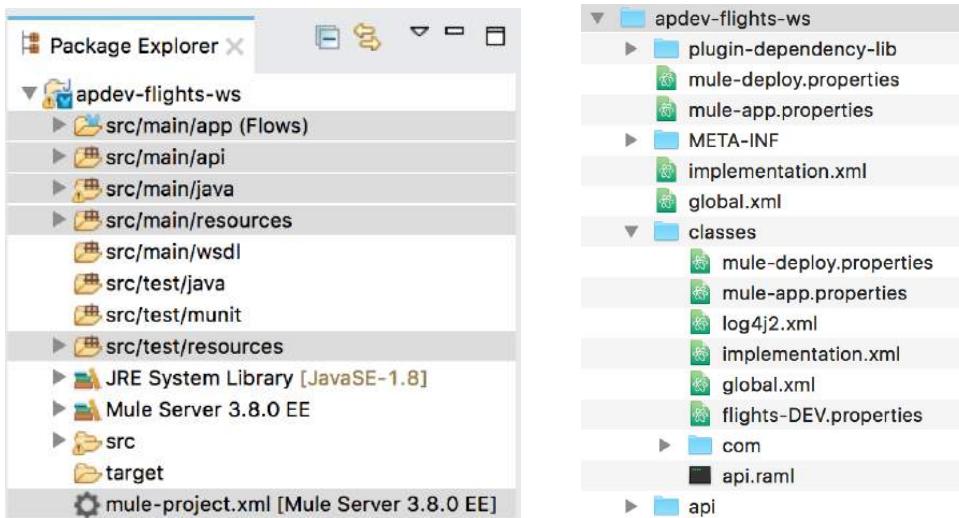
Close the project

25. Return to Anypoint Studio.
26. Stop the project.
27. In the Package Explorer, right-click apdev-examples and select Close Project.

Walkthrough 6-4: 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:

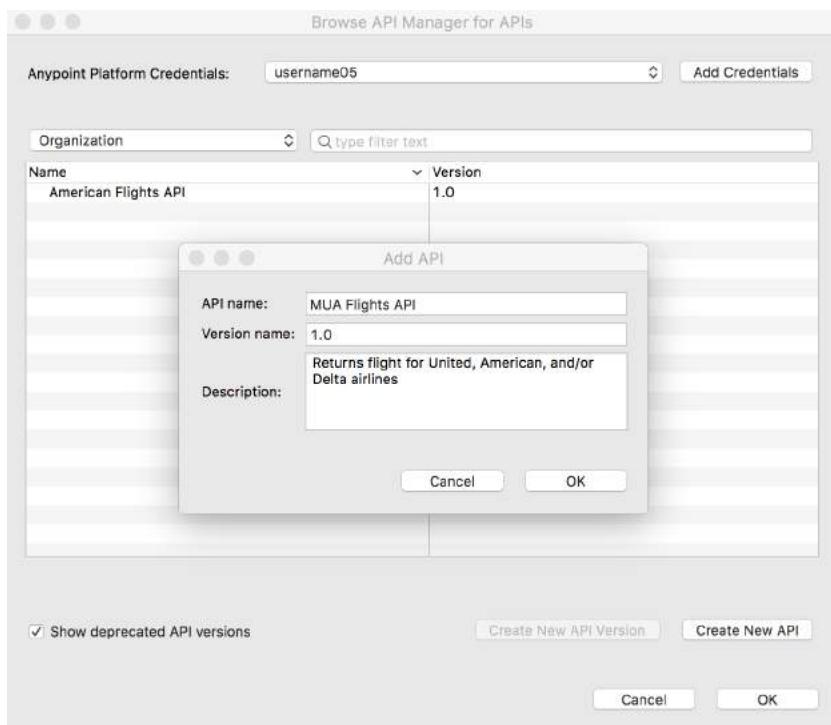
- Create a project with a new RAML file that is added to Anypoint Platform.
- Review the project's configuration and properties files.
- Create an application properties file and a global configuration file for the project.
- Add Java files and test resource files to the project.
- Create and examine the contents of a deployable archive for the project.



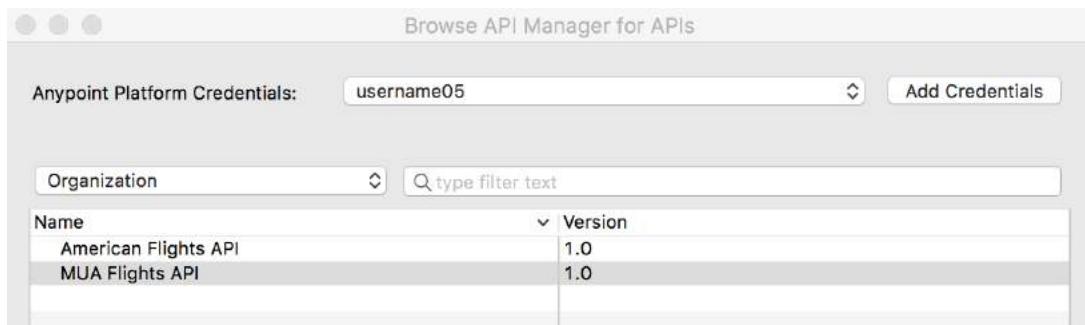
Create a project with a new RAML file that is added to Anypoint Platform

1. Right-click in the Package Explorer and select New > Mule Project.
2. In the New Mule Project dialog box, set the project name to apdev-flights-ws.
3. Check Add APIkit components.
4. Click the browse button next to API Definition and select Anypoint Platform.
5. In the Browse API Manager for APIs dialog box, click the Create New API button.
6. In the Add API dialog box, set the following values:
 - API name: MUA Flights API
 - Version name: 1.0
 - Description: Returns flights for United, American, and/or Delta airlines

7. Click OK.



8. In the Browse API Manager for APIs dialog box, make sure the new MUA Flights API is now listed and selected.



9. Click OK.

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

Locate the new API on Anypoint Platform

11. Return to Anypoint Platform in a web browser.

12. From the main menu, select API Manager; you should see the new MUA Flights API.

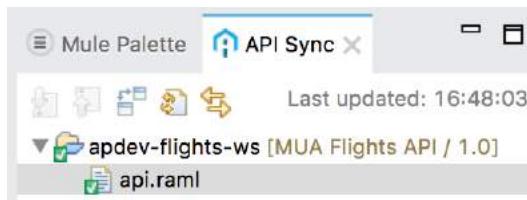
The screenshot shows the MuleSoft API Manager interface. At the top, there's a navigation bar with a menu icon, the title 'API Manager', and a search bar. Below the bar, there are buttons for 'Add new API' (highlighted in blue), 'Search', and navigation arrows. A status message '1 - 2 of 2' is displayed. Below the search bar, there are tabs for 'All', 'Favorites', 'Active', and 'Public Portal'. Two API entries are listed: 'MUA Flights API' (version 1.0, no portal, 'Add Version' button) and 'American Flights API' (version 1.0, active, 'Public portal' link, 'Add Version' button).

13. Click the 1.0 version of the API.

14. Click the Edit in API designer link and look at the starting RAML code.
15. Click the MUA Flights API – 1.0 link in the menu bar.
16. Leave this web browser window open.

Locate the new RAML file in Anypoint Studio

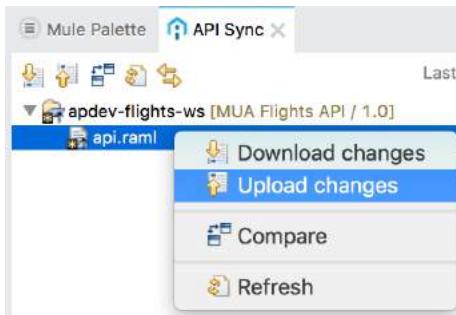
17. Return to the apdev-flights-ws project in Anypoint Studio.
18. Make sure you are in the Mule Design perspective.
19. In the API Sync view, locate the new connection to Anypoint Platform.



20. Double-click api.raml; you should see an empty RAML file.
21. In your computer's file browser, locate flights.raml in the resources folder of the student files.
22. Open the file in a text editor and copy the text.
23. Return to api.raml in Anypoint Studio.
24. Delete the existing text and paste the definition you copied.

Note: You are adding the RAML code to the existing api.raml file (instead of adding a new RAML file to the project and deleting the existing one) because an API's main RAML file can't be deleted on Anypoint Platform. You can change its name, but you cannot delete it.

25. Save the file.
26. In the API Sync view, right-click api.raml and select Upload changes.



Review the API on Anypoint Platform

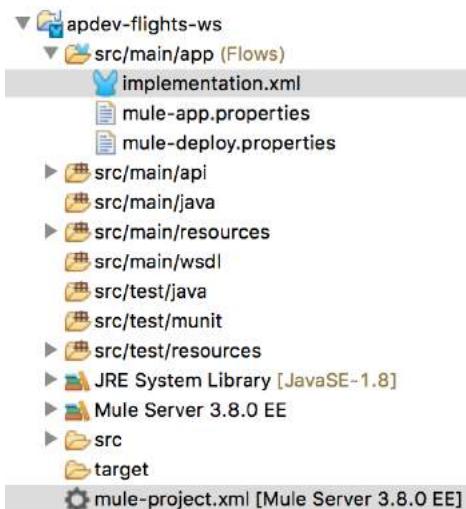
27. Return to Anypoint Platform in a web browser.
28. Click the Edit in API designer link; you should see the modified RAML.

Review project configuration files

29. Return to Anypoint Studio.
30. Look at the apdev-flights-ws.xml file that was created; it should have no contents.

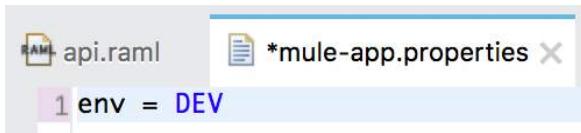
Note: No interface for the API was created because this was a project with a new RAML file with no resources. You can generate the interface later by right-clicking the project and selecting Mule > Generate Flows from RAML.

31. Rename apdev-flight-ws.xml to implementation.xml.
32. Locate mule-project.xml in the project and open it.
33. Review its contents and then close the file.



Review project properties files

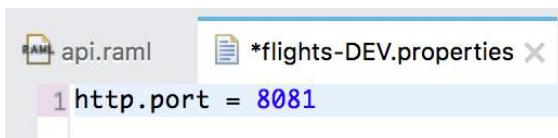
34. Locate mule-app.properties in the project and open it.
35. Add an environment variable called env equal to DEV.



36. Save the file and close it.
37. Locate and open mule-deploy.properties.
38. Review its contents and then close the file.

Create a properties file for application parameters

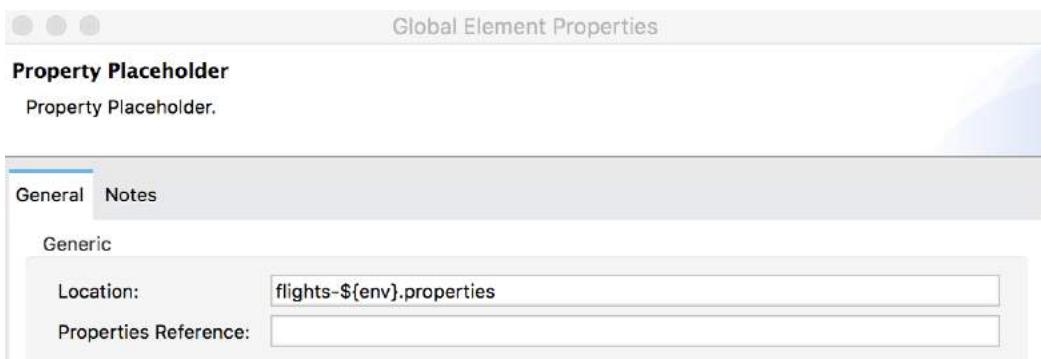
39. Right-click src/main/resources and select New > File.
40. In the New File dialog box, set the file name to flights-DEV.properties and click Finish.
41. Add a property called http.port equal to 8081.



42. Save and close the file.

Create a global configuration file

43. In src/main/app, create a new configuration file called global.xml.
44. Switch to the Global Elements view.
45. Create a new Property Placeholder configuration with a location set to flights-\${env}.properties.



46. Click OK.

47. Create a new HTTP Listener Configuration with a port set to \${http.port}.

The screenshot shows the 'Global Element Properties' dialog with the title 'HTTP Listener Configuration'. It has tabs for 'General', 'TLS/SSL', and 'Notes', with 'General' selected. Under 'General', the 'Name' field is set to 'HTTP_Listener_Configuration'. In the 'URL Configuration' section, the 'Protocol' is set to 'HTTP (Default)', 'Host' is 'All Interfaces [0.0.0.0] (Default)', 'Port' is '\${http.port}', and 'Base Path' is empty. There are three circular progress bars at the top left of the dialog.

48. Confirm you now have two global elements defined in the global.xml file.

The screenshot shows the 'Global Mule Configuration Elements' list. It has columns for 'Type' and 'Name'. Two items are listed: 'Property Placeholder (Config)' under 'Name' and 'HTTP Listener Configuration (HTTP_Listener_Configuration)' under 'Name'. There are 'Create' and 'Edit' buttons at the bottom right. The 'global' tab is selected in the top navigation bar, which also includes 'api.raml'.

49. Save and close the file.

Review src/main/resources

50. In src/main/resources, open log4j2.xml.

51. Review and then close the file.

The screenshot shows the 'src/main/resources' directory. It contains two files: 'flights-DEV.properties' and 'log4j2.xml'. The 'src/main/resources' folder is expanded, indicated by a downward arrow icon.

Add Java files to src/main/Java

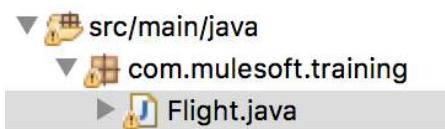
52. In your computer's file browser, locate the com folder in the java folder of the student files.

53. Drag the com folder into the src/main/java folder in the Anypoint Studio apdev-flights-ws project.

54. Expand the new com directory.

55. Open the Flight.java file.

56. Review the code and then close the file.



Review src/test folders

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

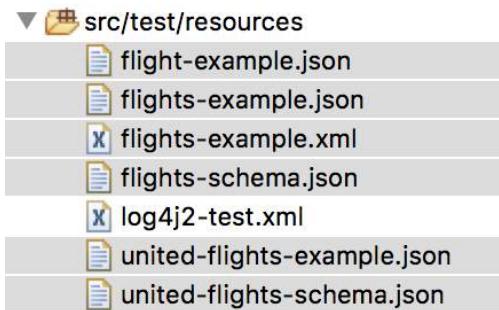
58. Expand the src/test/resources folder.



Add test resources

59. In your computer's file browser, expand the schema-and-examples folder in the student files.

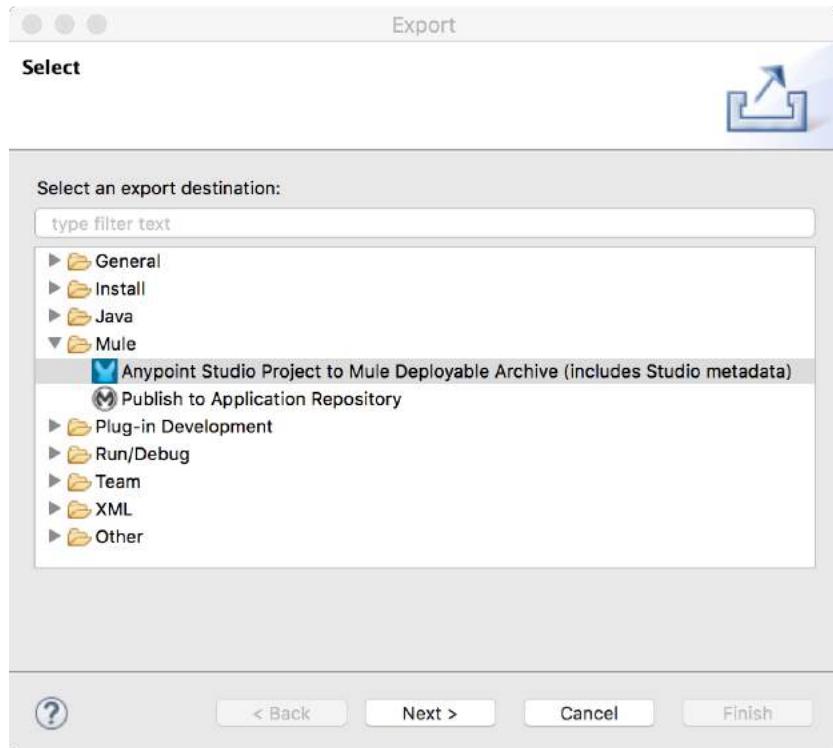
60. Select the four flights and two united-flights JSON files and drag them into the src/test/resources folder in the Anypoint Studio apdev-flights-ws project.



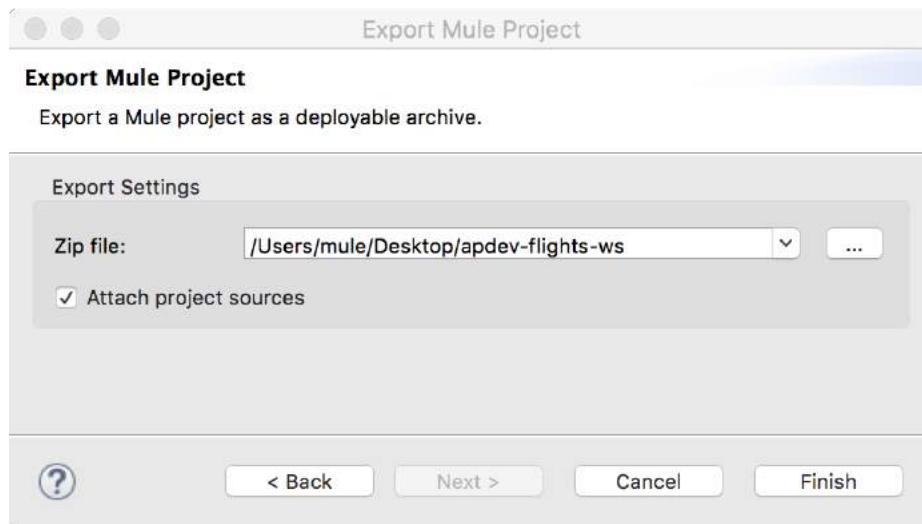
Examine the contents of a deployable archive

61. In Anypoint Studio, right-click the project and select Export.

62. In the Export dialog box, select Mule > Anypoint Studio to Mule Deployable Archive and click Next.



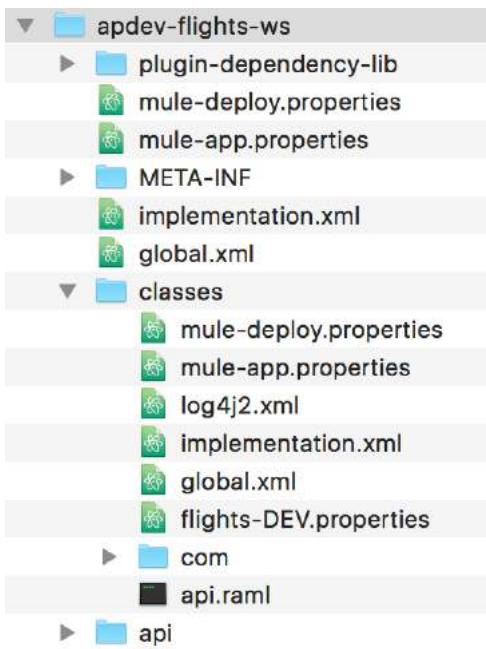
63. Set the Zip file to a location that you can find and click Finish.



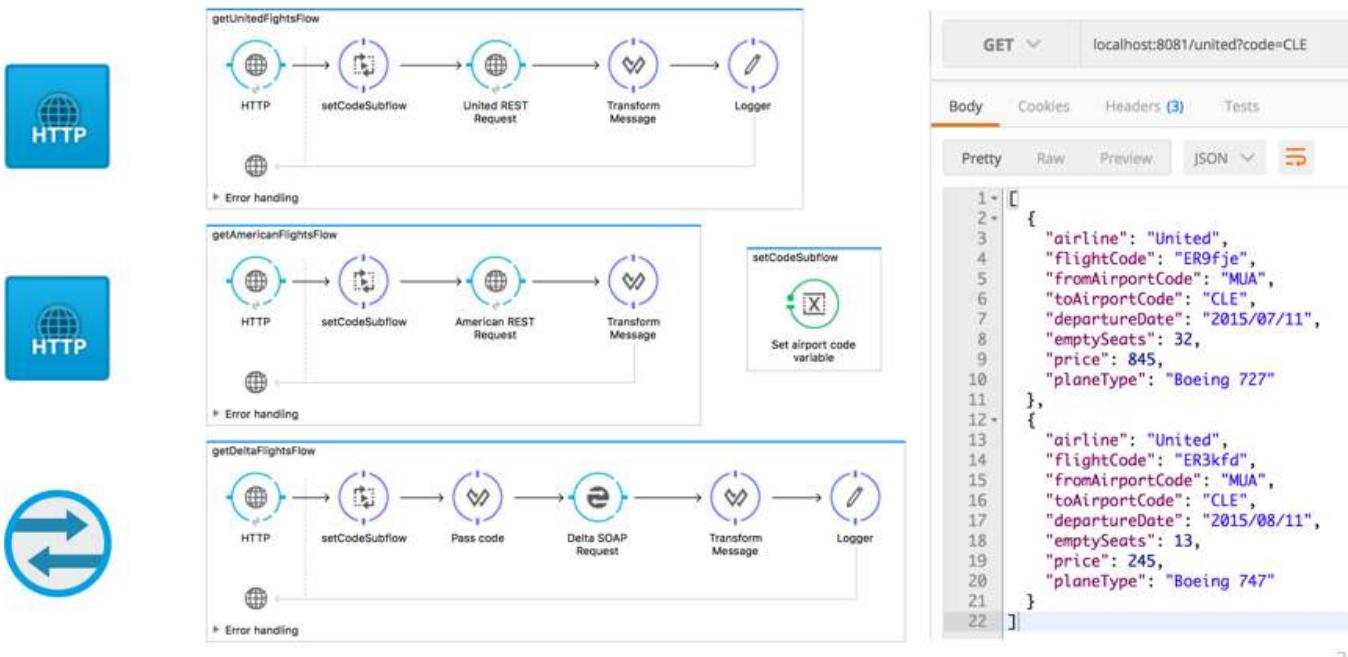
64. In your computer's file browser, locate the ZIP file and expand it.

65. Open the resulting apdev-flights-ws folder.

66. Expand the classes folder and examine the contents.



Module 7: Consuming Web Services



Objectives:

- Consume RESTful web services with and without parameters.
- Consume RESTful web services that have RAML definitions.
- Consume SOAP web services.
- Use DataWeave to pass parameters to SOAP web services.

Walkthrough 7-1: Consume a RESTful web service

In this walkthrough, you consume a RESTful web service that returns a list of all United flights as JSON. You will:

- Create a new flow to call a RESTful web service.
- Use an HTTP Request endpoint to consume a RESTful web service.
- Use DataWeave to transform the JSON response into JSON specified by an API.

The screenshot shows the Mule Studio interface with a flow named 'getUnitedFlightsFlow'. The flow consists of four components: 'HTTP', 'United REST Request', 'Transform Message', and 'Logger'. The 'HTTP' component has an outgoing arrow pointing to the 'United REST Request' component. The 'United REST Request' component has an outgoing arrow pointing to the 'Transform Message' component. The 'Transform Message' component has an outgoing arrow pointing to the 'Logger' component. The 'HTTP' component also has an incoming arrow from the left. The 'Transform Message' component has an outgoing arrow pointing back to the left. The 'Body' tab of the message preview shows a JSON response with flight details. The 'Tests' tab shows a successful test run with status 200 and time 8799 ms.

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

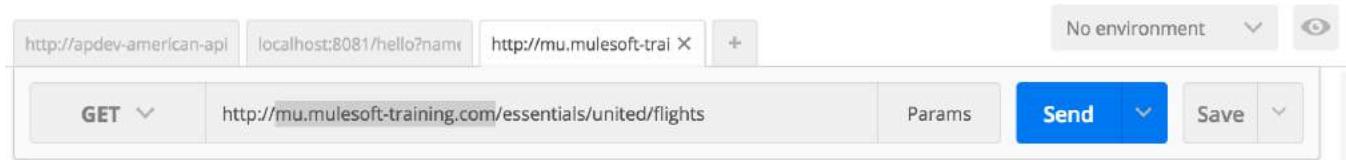
Make a request to the web service

1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Postman, make a new tab.
4. Make a GET request to this URL; you should see JSON data for the United flights as a response.

The screenshot shows the Postman interface with a tab for 'http://mu.mulesoft-training.com/essentials/united/flights'. The 'Body' tab of the message preview shows a JSON response with flight details. The 'Tests' tab shows a successful test run with status 200 OK and time 787 ms.

```
1+ [
2+   {
3+     "flights": [
4+       {
5+         "airlineName": "United",
6+         "price": 400,
7+         "departureDate": "2015/03/20",
8+         "planeType": "Boeing 737",
9+         "origin": "MUA",
10+        "code": "ER38sd",
11+        "emptySeats": 0,
12+        "destination": "SFO"
13+      },
14+      {
15+        "airlineName": "United",
16+        "price": 345.99
17+      }
18+    ]
19+  }
20+]
```

5. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.
6. Copy the host name from the URL; this should be something like mu.mulesoft-training.com.



Add a new flow with an HTTP Listener endpoint

7. Return to implementation.xml in Anypoint Studio.
8. Drag out an HTTP connector and drop it in the canvas.
9. Double-click the name of the flow in the canvas and give it a new name of getUnitedFlightsFlow.

Configure the HTTP Listener endpoint

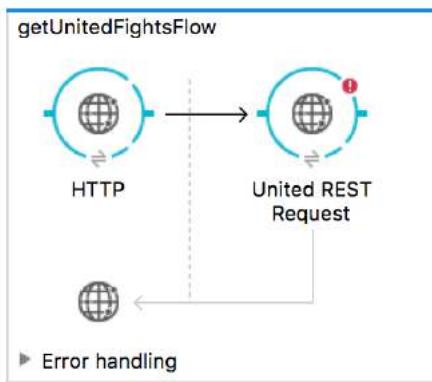
10. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
11. Set the path to /united.
12. Set the allowed methods to GET.

The screenshot shows the Anypoint Studio interface. At the top, there's a toolbar with various icons. Below it is the main workspace where a process named 'getUnitedFlightsFlow' is defined. This process contains an 'HTTP' connector. To the right of the canvas is the 'Properties' view for the 'HTTP' connector. Under the 'General' tab, the 'Path' is set to '/united' and the 'Allowed Methods' are set to 'GET'. A status message at the top of the properties view says 'There are no errors.'

Add an HTTP Request endpoint

13. Drag out another HTTP connector and drop it into the process section of the flow.

14. Change the HTTP Request endpoint display name to United REST Request.



Configure the HTTP Request connector

15. Return to flights-DEV.properties in src/main/resources.

16. Create a property called united.host and set it equal to the value you copied from the URL.

A screenshot of a code editor showing the contents of the file '*flights-DEV.properties'. The file contains the following configuration:

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

17. Save the file.

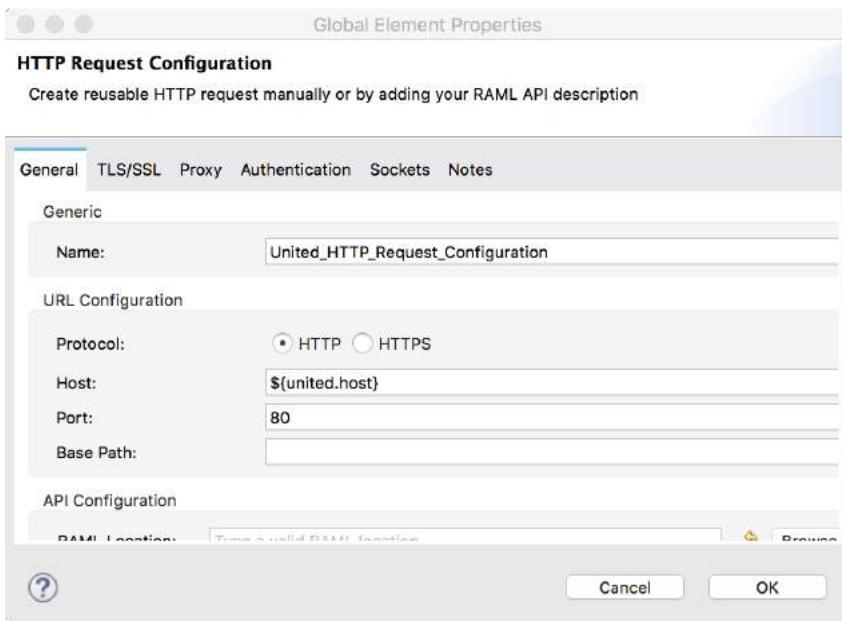
18. Return to global.xml.

19. In the Global Elements view, click Create.

20. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

21. In the Global Element Properties dialog box, set the following values and click OK.

- Name: United_REST_Request_Configuration
- Host: \${united.host}
- Port: 80
- Base Path: Leave blank



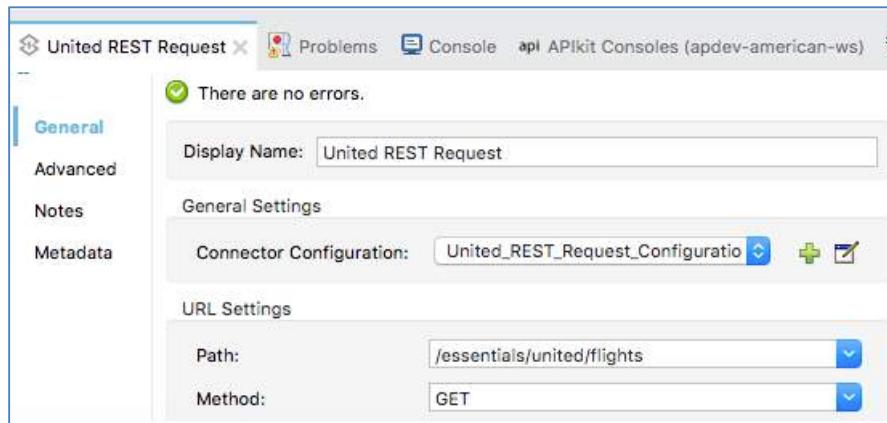
Configure the HTTP Request endpoint

22. Return to implementation.xml.

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

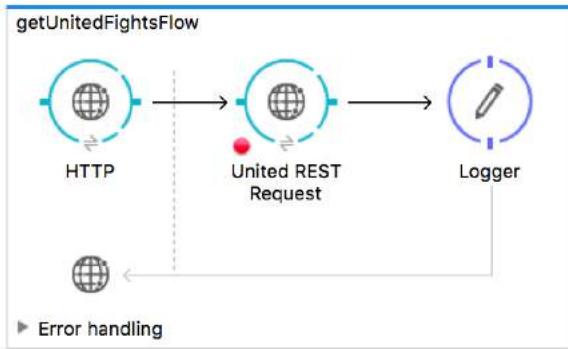
24. Set the path to /essentials/united/flights.

25. Set the method to GET.



Test the application

26. Make sure the United REST Request endpoint has a breakpoint.
27. Add a Logger after the United REST Request endpoint.



28. Debug the project.
29. In Postman, return to the tab with the localhost request.
30. Make a request to <http://localhost:8081/united>.
31. In the Mule Debugger, step to the Logger and look at the payload; it should be of type BufferInputStream.

Screenshot of the Mule Debugger interface. The top navigation bar includes tabs for 'Console', 'Problems', and 'Mule Properties'. The main area displays a table of message variables:

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

32. Step through the application.
33. Return to Postman; you should see the JSON flight data returned.

34. Examine the data structure of the JSON response.

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

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

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

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

Review the structure for the desired JSON response

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

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

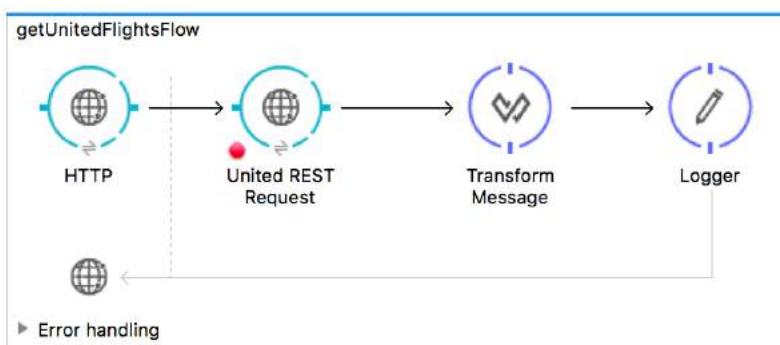
The screenshot shows the contents of the flights-example.json file in Anypoint Studio:

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

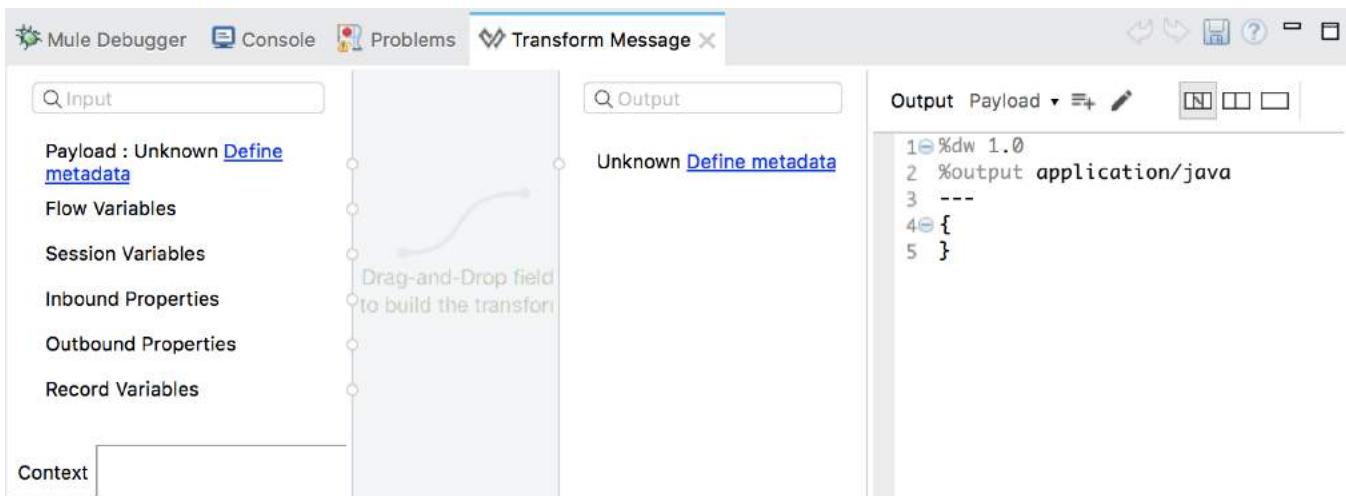
Add input metadata for the transformation

37. Return to implementation.xml.

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



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



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

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

42. Click Create type.

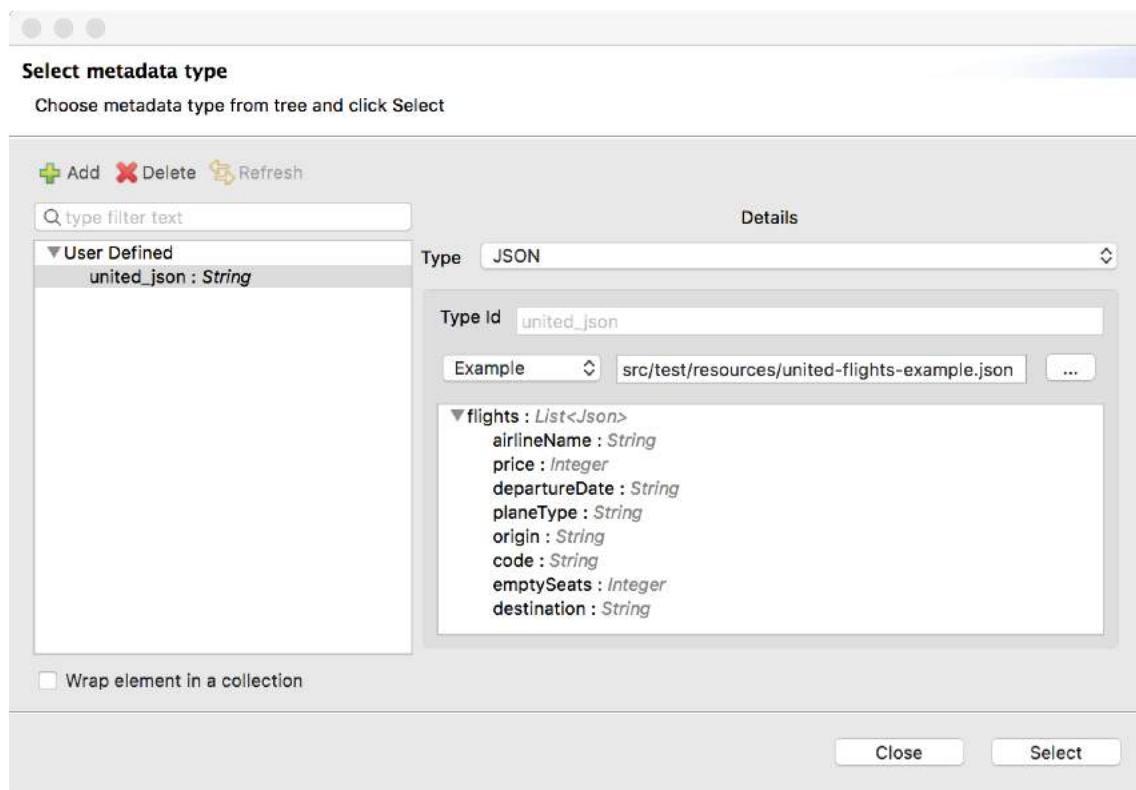
43. In the Select metadata type dialog box, set the type to JSON.

44. Change the Schema selection to Example.

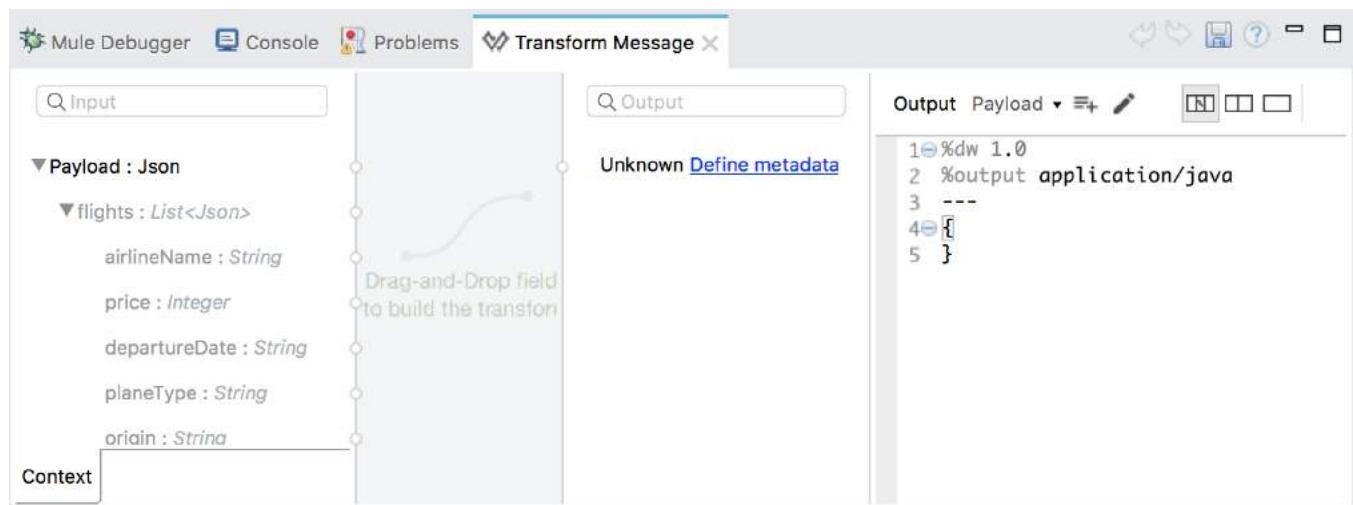
45. Click the browse button.

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

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



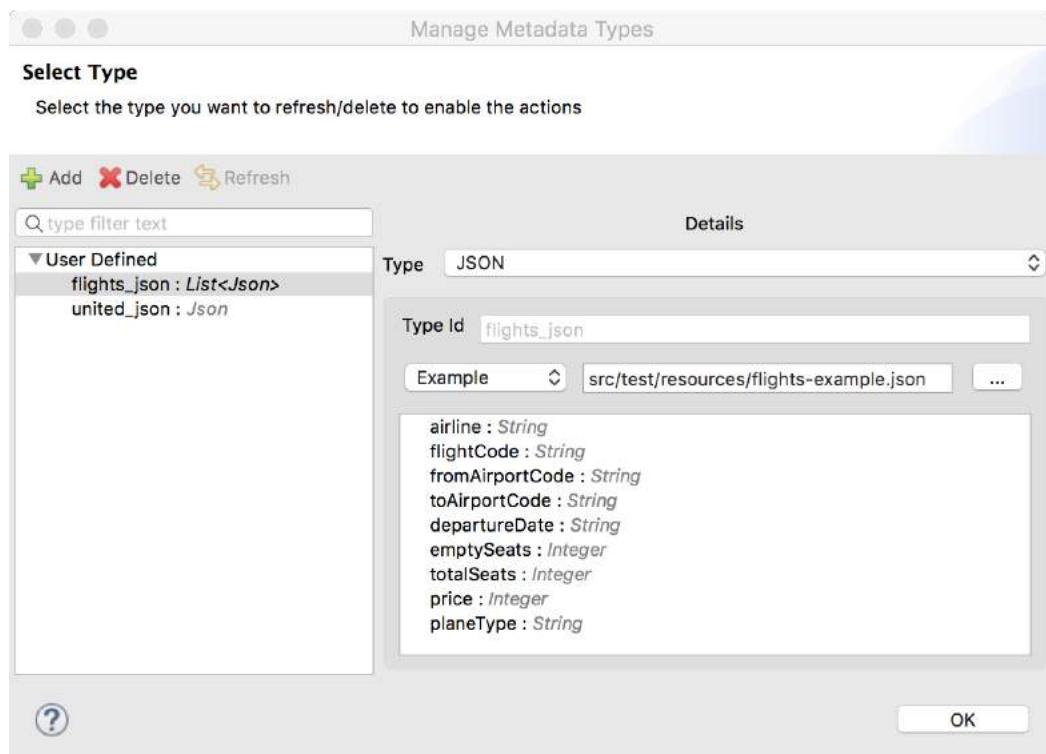
48. Click Select; you should now see output metadata in the input section of the Transform Message properties view.



Add output metadata for the transformation

49. In the output section of the Transform Message properties view, find where the output is set to application/java.
50. Click the Define metadata link.
51. In the Select metadata type dialog box, click the Add button.
52. In the Create new type dialog box, set the type id to flights_json.
53. Click Create type.
54. In the Select metadata type dialog box, set the type to JSON.
55. Change the Schema selection to Example.
56. Click the browse button and navigate to the projects's src/test/resources folder.
57. Select flights-example.json and click Open; you should see the example data for the metadata type.

Note: Be sure to select the JSON file with flights plural, not singular.



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

The screenshot shows the 'Transform Message' editor in Mule Studio. The left pane displays the 'Payload' section with a tree view of flight attributes. The right pane shows the 'Output' section with a 'List<Json>' node selected. A 'flights' node from the payload is being mapped to the 'List<Json>' node using a 'Drag-and-Drop' action. The output payload is defined as follows:

```
1 %dw 1.0
2 %output application/json
3 ---
4 [
5 ]
```

Create the transformation

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

Note: There is no input field to map to totalSeats.

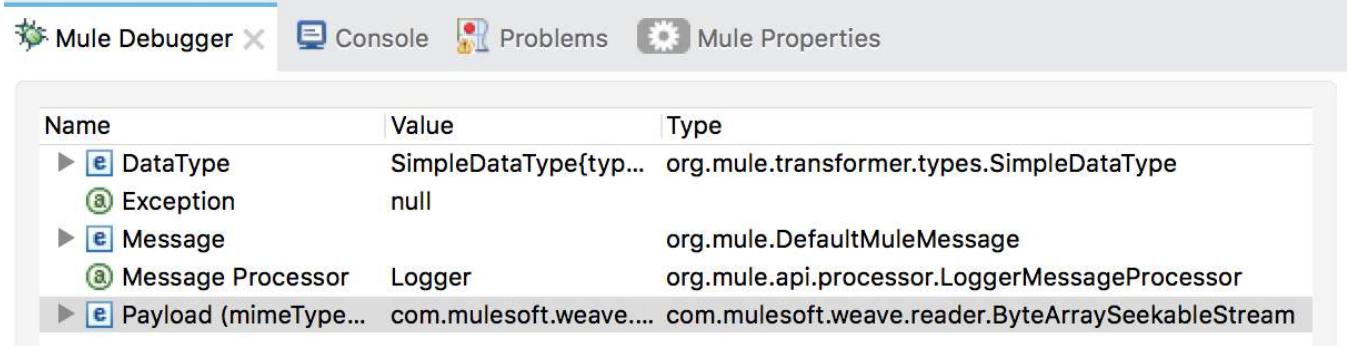
The screenshot shows the 'Transform Message' editor with fields being mapped from the payload to the output list. The payload contains flight attributes like airlineName, price, departureDate, etc. The output payload is defined with a map expression for the flights list:

```
1 %dw 1.0
2 %output application/json
3 ---
4 payload.flights map ((flight , indexOfflight) -> {
5   airline: flight.airlineName,
6   flightCode: flight.code,
7   fromAirportCode: flight.origin,
8   toAirportCode: flight.destination,
9   departureDate: flight.departureDate,
10  emptySeats: flight.emptySeats,
11  price: flight.price,
12  planeType: flight.planeType
13 })
```

Test the application

60. Debug the project.

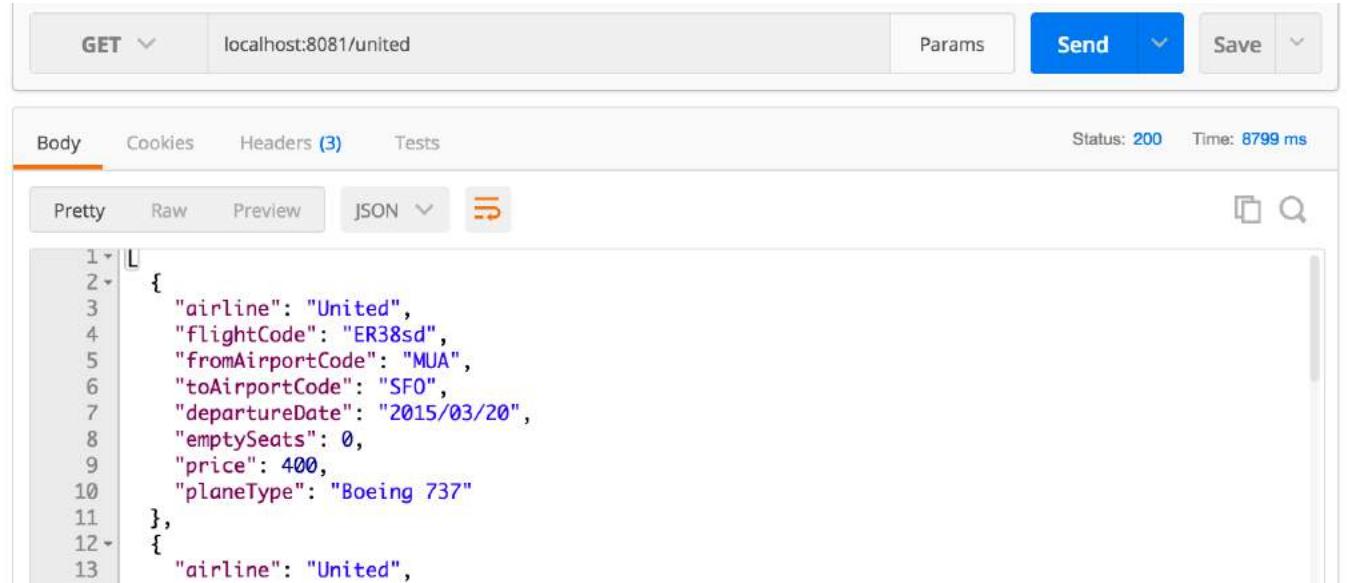
61. In Postman, make another request to <http://localhost:8081/united>.
62. In the Mule Debugger, step to the Logger; you should see the payload is now a DataWeave ByteArraySeekableStream.



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

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

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



The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: localhost:8081/united
- Params: None
- Send button: Enabled
- Save button: Enabled
- Body tab selected
- Pretty: Selected
- Raw: Unselected
- Preview: Unselected
- JSON: Selected
- Copy icon: Available
- Search icon: Available
- Status: 200
- Time: 8799 ms

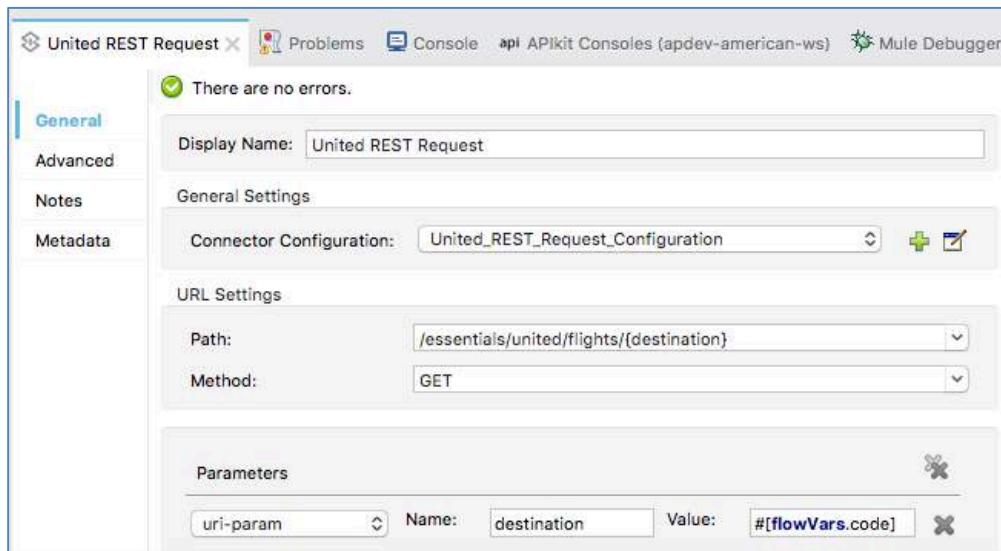
```
1 [L
2 {
3     "airline": "United",
4     "flightCode": "ER38sd",
5     "fromAirportCode": "MUA",
6     "toAirportCode": "SFO",
7     "departureDate": "2015/03/20",
8     "emptySeats": 0,
9     "price": 400,
10    "planeType": "Boeing 737"
11 },
12 [
13     {
14         "airline": "United",
15     }
16 ]
```

65. Return to Anypoint Studio and stop the project.

Walkthrough 7-2: Pass arguments to a RESTful web service

In this walkthrough, you retrieve United flights for a specific destination by setting the destination as a URI parameter. You will:

- Modify the HTTP Request endpoint to use a URI parameter for the destination.
- Set the destination to a static value.
- Create a flow variable to store the value of a query parameter with an airport code value.
- Set the destination to the dynamic value of this flow variable.

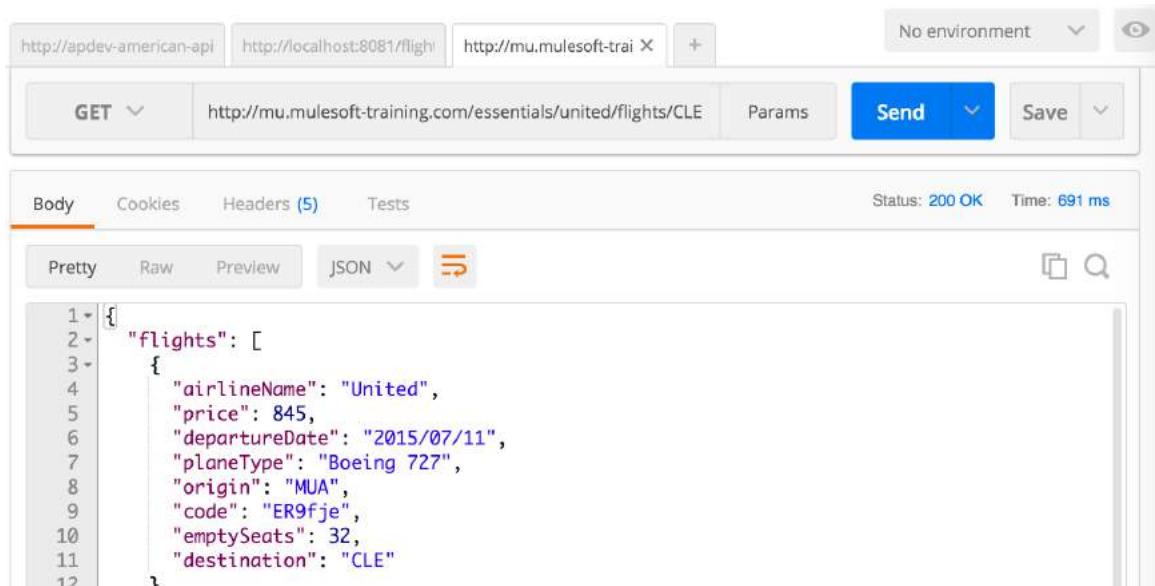


Make a request to the web service specifying a destination

1. Return to Postman and click the third tab, the one with the request to the United web service.
2. In the URL field, add the destination CLE as a URI parameter: <http://mu.mulesoft-training.com/essentials/united/flights/CLE>.

Note: The API you are building for flights.raml has a query parameter called code for the destination; this existing United web service uses a URI parameter for the destination.

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



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

- Header bar: http://apdev-american-api, http://localhost:8081/flight, http://mu.mulesoft-trai X, No environment
- Method: GET
- URL: http://mu.mulesoft-training.com/essentials/united/flights/CLE
- Buttons: Params, Send, Save
- Status: 200 OK, Time: 691 ms
- Body tab selected: Shows a JSON response with line numbers 1-12.

```
1- [ {  
2-   "flights": [  
3-     {  
4-       "airlineName": "United",  
5-       "price": 845,  
6-       "departureDate": "2015/07/11",  
7-       "planeType": "Boeing 727",  
8-       "origin": "MUA",  
9-       "code": "ER9fje",  
10-      "emptySeats": 32,  
11-      "destination": "CLE"  
12-    } ]
```
- Other tabs: Cookies, Headers (5), Tests

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

Add a URI parameter with a static value

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

12. Set the parameter value to SFO.

The screenshot shows the 'United REST Request' configuration in Anypoint Studio. The 'General' tab is active. In the 'URL Settings' section, the 'Path' is set to '/essentials/united/flights/{destination}' and the 'Method' is set to 'GET'. In the 'Parameters' section, there is one entry: 'uri-param' with 'Name: destination' and 'Value: SFO'. A green checkmark in the top left corner indicates that there are no errors.

Test the application

13. Run the project.
14. In Postman, return to the tab with the localhost requests.
15. Make a request to <http://localhost:8081/united/>; you should only get the flights to SFO.
16. Add a query parameter called code and set it to LAX.

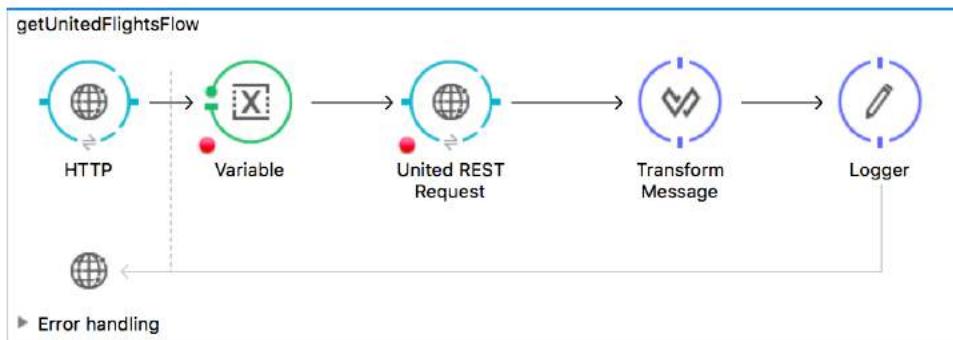
The screenshot shows a POSTMAN interface. The request method is set to 'GET'. The URL is 'localhost:8081/united?code=LAX'. There is one query parameter named 'code' with the value 'LAX'. The 'Send' button is highlighted in blue.

17. Send the request; of course, you should still get only flights to SFO.

Create a variable to set the destination airport code

18. Return to Anypoint Studio.

19. Add a Variable transformer before the United REST Request endpoint.



20. In the Variable Properties view, change the display name to Set airport code variable.

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

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

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

The screenshot shows the "Variable" tab in the Mule Studio interface. The "Display Name" is set to "Set airport code variable". Under "Settings", the "Operation" is selected as "Set Variable", the "Name" is "code", and the "Value" is "##[message.inboundProperties.'http.query.params'.code]".

Change the REST request URI parameter to a dynamic value

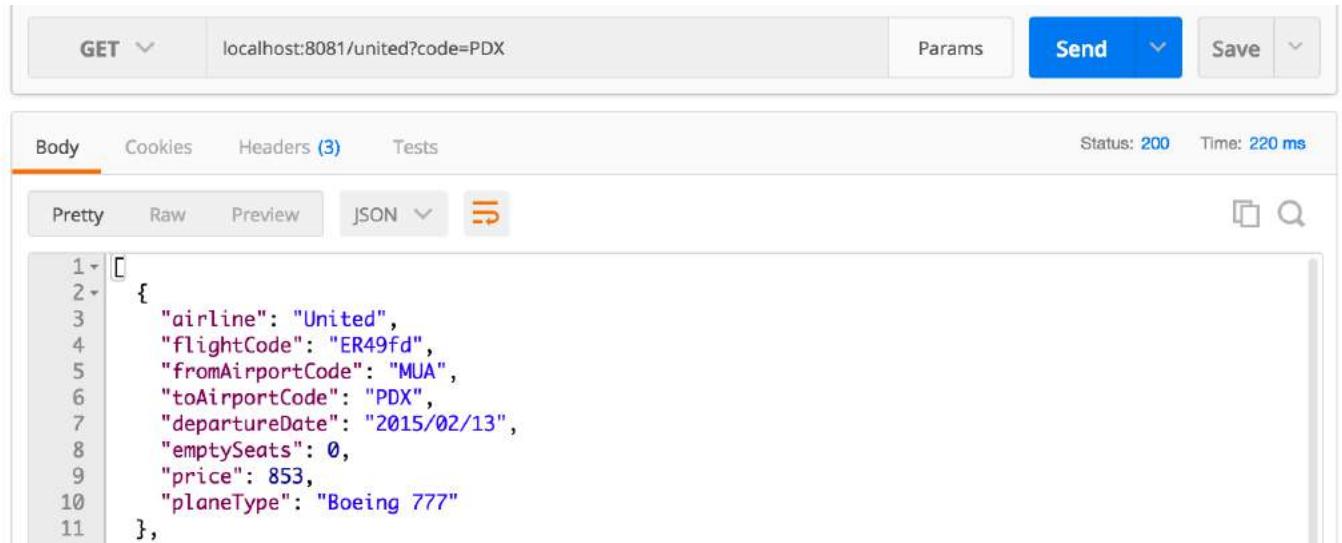
23. In the United REST Request properties view, change the value of the uri-param from SFO to the value of the flow variable containing the airport code.

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

The screenshot shows the "Parameters" section of the United REST Request properties. A parameter named "destination" is defined with a "Value" of "##[flowVars.code]".

Test the application

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



The screenshot shows a Postman request for `localhost:8081/united?code=PDX`. The response status is 200 and the time taken is 220 ms. The response body is a JSON object:

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

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



The screenshot shows a Postman request for `localhost:8081/united`. The response status is 500 and the time taken is 15 ms. The response body contains an exception message:

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

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

28. Return to Anypoint Studio.

29. Modify the Set variable transformer to use a ternary expression to assign a default value of SFO if no query parameter is passed to the flow.

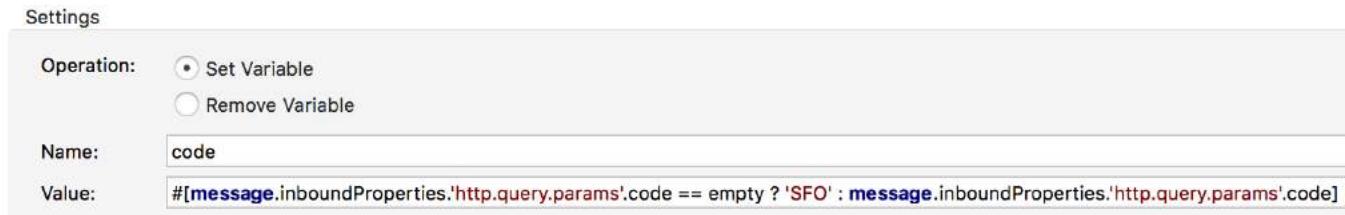
```
#[(message.inboundProperties.'http.query.params'.code == empty) ?  
'SFO' : message.inboundProperties.'http.query.params'.code]
```

Settings

Operation: Set Variable
 Remove Variable

Name: code

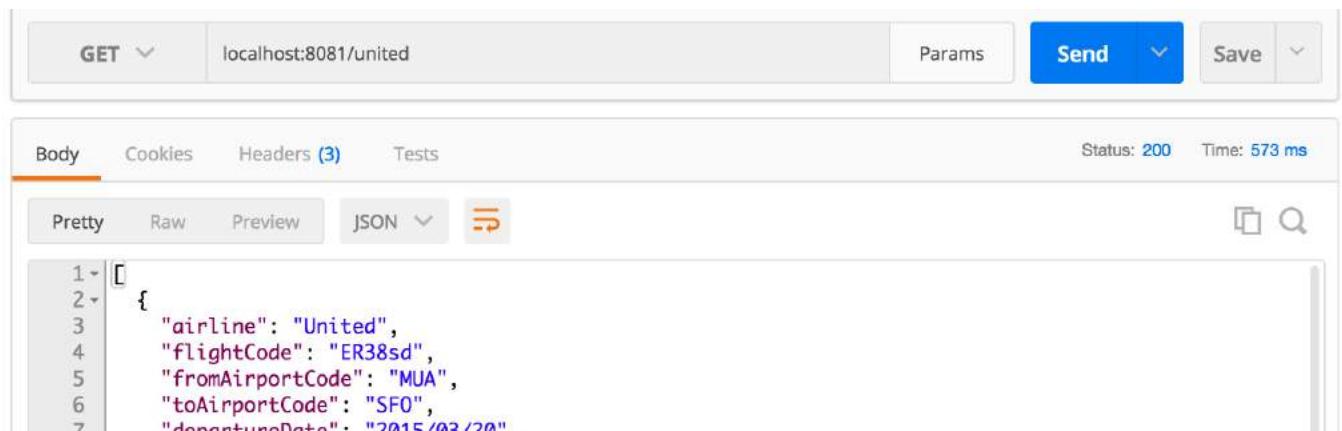
Value: #[message.inboundProperties.'http.query.params'.code == empty ? 'SFO' : message.inboundProperties.'http.query.params'.code]



Test the application

30. Save and redeploy the application.

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



The screenshot shows a Postman request to `localhost:8081/united`. The response status is `200` and the time taken is `573 ms`. The response body is a JSON object:

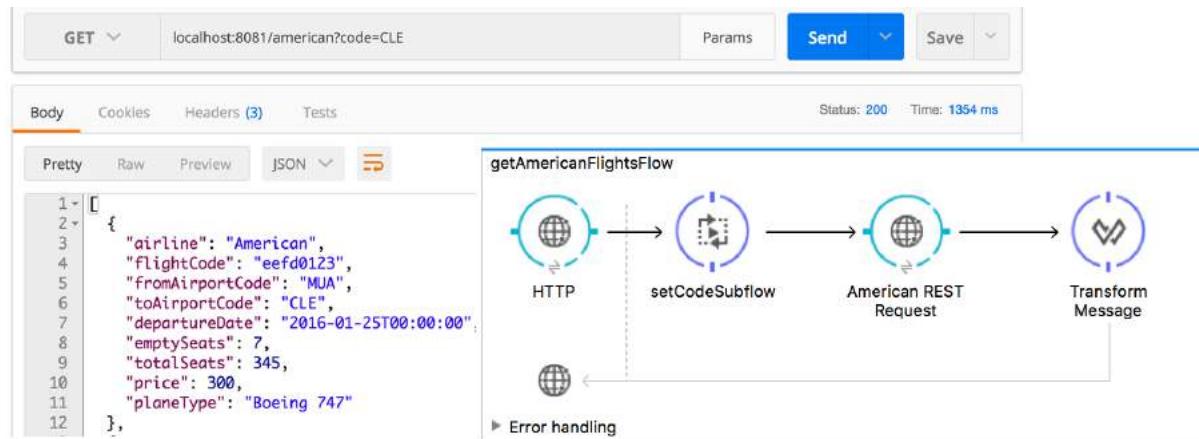
```
1 - [  
2 - {  
3 -   "airline": "United",  
4 -   "flightCode": "ER38sd",  
5 -   "fromAirportCode": "MUA",  
6 -   "toAirportCode": "SFO",  
7 -   "departureDate": "2015/03/20"
```

32. Return to Anypoint Studio and stop the project.

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

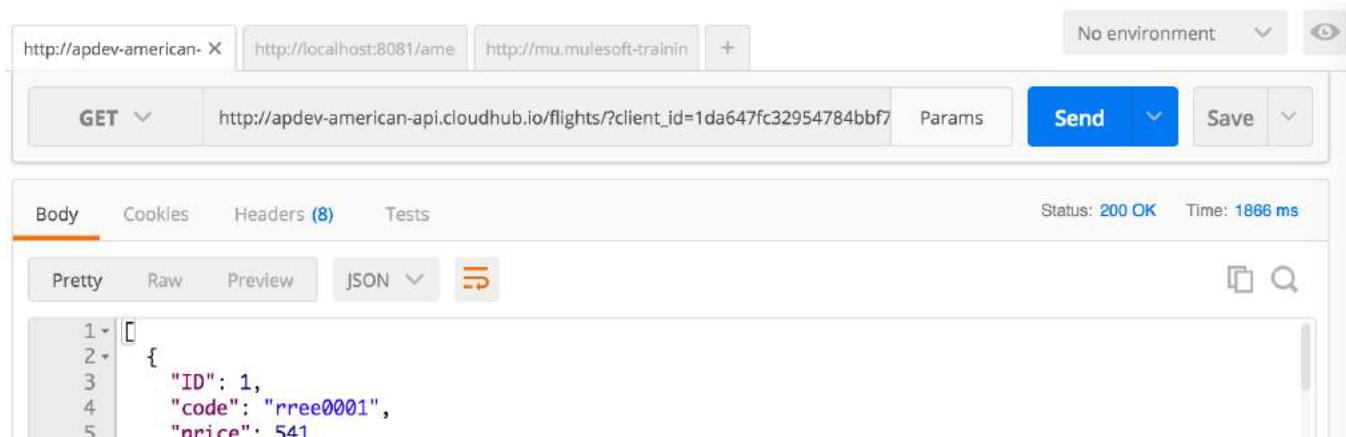
In this walkthrough, you consume the American flights RESTful web service that you built and deployed to the cloud. You will:

- Create a new flow to call a RESTful web service that has a RAML definition.
- Select the web service resource from the list provided by Anypoint Studio from the RAML file.
- Use DataWeave to transform the JSON response into JSON specified by an API.



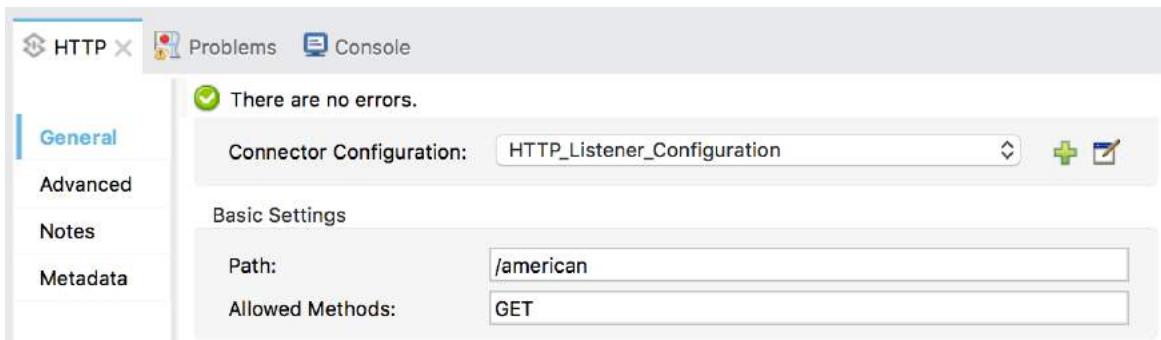
Make a request to the web service

1. In Postman, return to the first tab – the one with the request to your American Flights API <http://apdev-american-api-{lastname#}.cloudbhub.io/flights> and that passes a client_id and client_secret.
2. Send the request; you should still see JSON data for the American flights as a response.



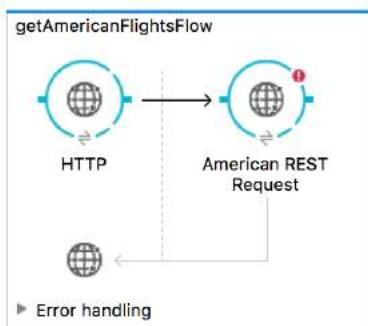
Add a new flow with an HTTP Listener endpoint

3. Return to implementation.xml.
4. Drag out another HTTP connector and drop it in the canvas.
5. Rename the flow to getAmericanFlightsFlow.
6. In the Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
7. Set the path to /american.
8. Set the allowed methods to GET.



Add an HTTP Request endpoint for a web service with a RAML definition

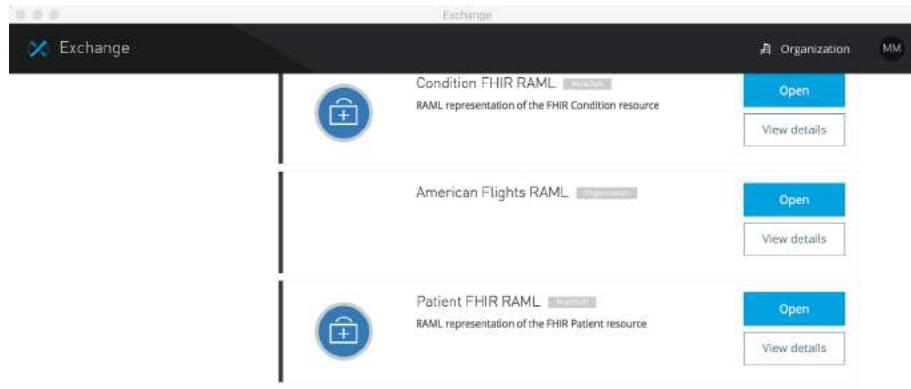
9. Drag out another HTTP connector and drop it in the process section of the new flow.
10. Change the endpoint display name to American REST Request.



Configure the HTTP Request connector

11. Return to global.xml.
12. In the Global Elements view, click Create.
13. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

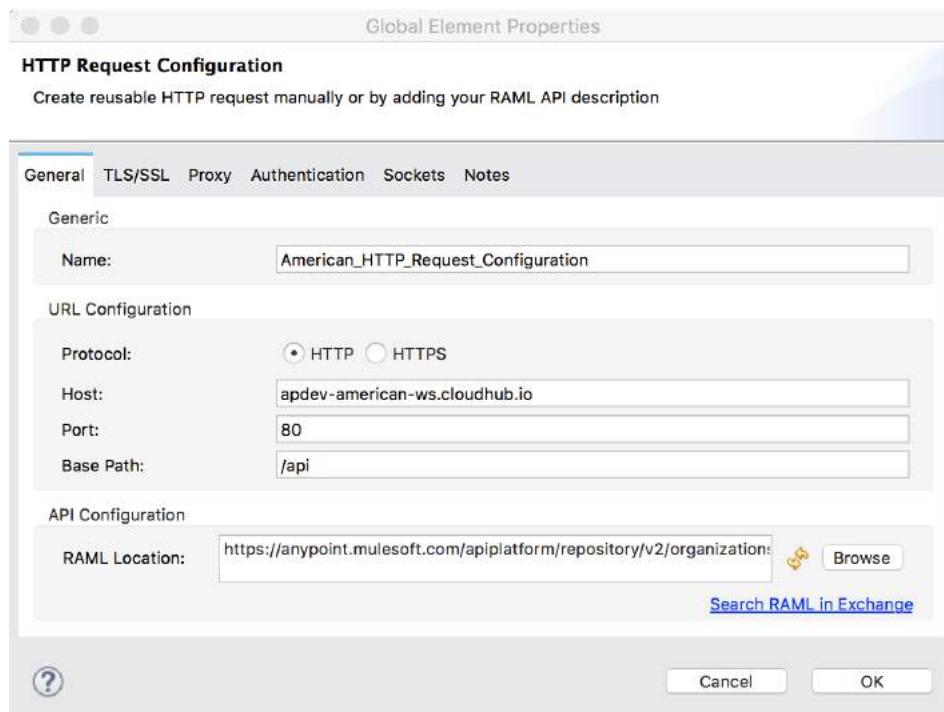
14. In the Global Element Properties dialog box, change the name to American_HTTP_Request_Configuration.
15. Click the Search RAML in Exchange link next to RAML location.
16. In the Exchange window that opens, scroll down and locate your American Flights RAML.



17. Click Open; the Exchange window should close.
18. Back in the Global Element Properties dialog box, wait for the RAML to be parsed and the host, port, and base path fields to be populated and then click OK.

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

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



19. Click OK.

Configure the HTTP Request endpoint

20. Return to implementation.xml.
21. In the American REST Request properties view, set the connector configuration to the existing American_HTTP_Request_Configuration.
22. Click the expand button for the path field; you should see all the available resources for the RESTful web service defined by the RAML file listed – in this case, there are two.
23. Select /flights/{ID}.
24. Look at the method drop-down menu; you should see DELETE, GET, and PUT.

Connector Configuration: American_HTTP_Request_Configuration   

URL Settings

Path: /flights/{ID}

Method:

DELETE
GET
PUT

Parameters

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

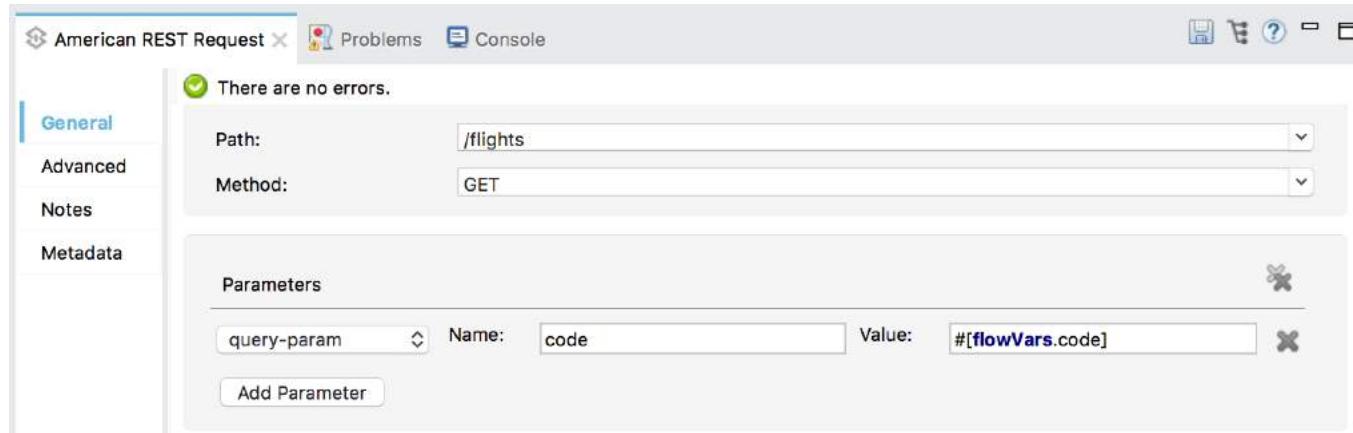
Connector Configuration: American_HTTP_Request_Configuration   

URL Settings

Path: /flights

Method: GET

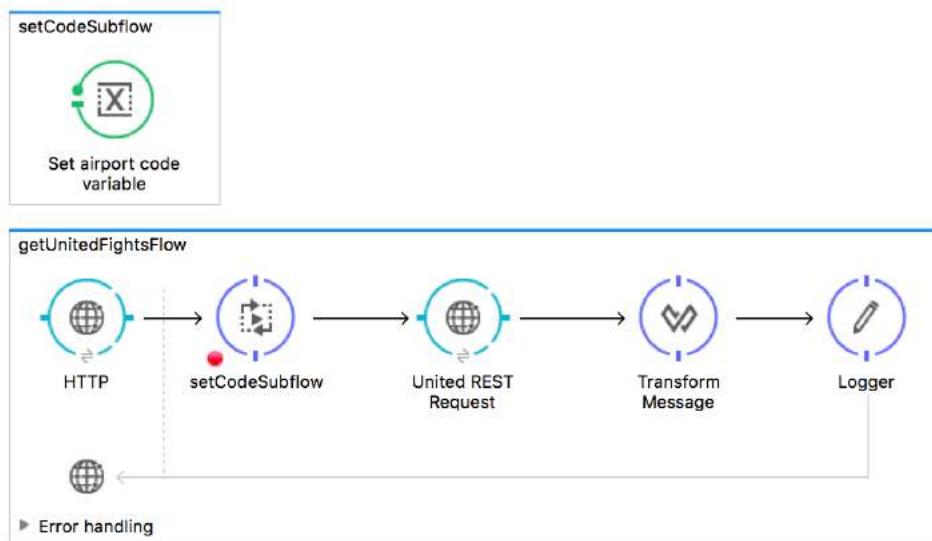
28. Scroll down and check to see if a query parameter called code has been added.
29. If the destination parameter was not automatically created, create it.
30. Set the value to a flow variable called code; you will add this to the flow next.



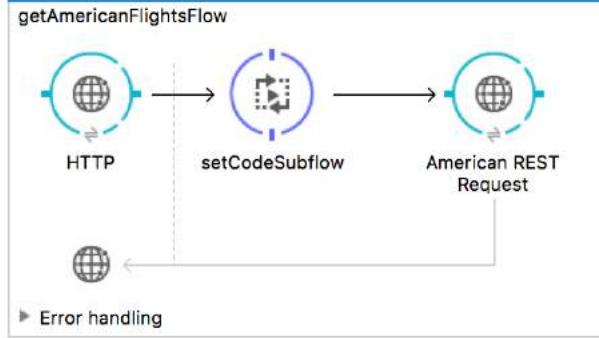
Extract the Set airport code variable processor into a subflow and use it

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

Note: If you do not see the subflow in the canvas, switch to the Configuration XML view and then back to the Message Flow view.



35. Drag a Flow Reference component from the Mule Palette before the American REST Request in getAmericanFlightsFlow.
36. In the Flow Reference properties view, set the flow name to setCodeSubflow.



Test the application

37. Save all the files to redeploy the application.
38. In Postman, return to the tab with the local requests.
39. Make a request to <http://localhost:8081/american>; you should get the American flights to SFO.
40. Add a query parameter called code with a value of CLE.
41. Send the request; you should get just the flights to CLE.

```

{
  "ID": 2,
  "code": "eefd0123",
  "price": 300,
  "departureDate": "2016-01-25T00:00:00",
  "origin": "MUA",
  "destination": "CLE",
  "emptySeats": 7,
  "plane": {
    "type": "Boeing 747",
    "totalSeats": 345
  }
}
  
```

Transform the data

42. Return to getAmericanFlightsFlow.
43. Add a Transform Message component after the American REST Request endpoint.
44. Double-click the Transform Message component.
45. Look at the input section; you should see metadata already defined.

46. In the output section of the Transform Message properties view, click the Define metadata link.
47. In the Define metadata type dialog box, select flights_json.
48. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
49. Map fields (except ID and airline) by dragging them from the input section and dropping them on the corresponding field in the output section.

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

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

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

Test the application

52. Save to redeploy the project.

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



The screenshot shows the Postman interface with a GET request to `localhost:8081/american?code=CLE`. The response body is displayed in a pretty-printed JSON format:

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

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

Walkthrough 7-4: Consume a SOAP web service

In this walkthrough, you consume a SOAP web service that returns a list of all Delta flights as XML. You will:

- Create a new flow to call a SOAP web service.
- Use a Web Service Consumer endpoint to consume a SOAP web service for Delta flight data.
- Use DataWeave to transform the XML response into JSON specified by the MUA Flights API.

The screenshot shows a Postman interface with a successful response. The response body is a JSON object representing a flight:

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

To the right of the response, a flow diagram titled "getDeltaFlightsFlow" is shown:

```
graph LR; A((HTTP)) --> B((Delta SOAP Request)); B --> C((Transform Message)); C --> D((Logger));
```

The flow consists of four components: HTTP, Delta SOAP Request, Transform Message, and Logger.

Browse the WSDL

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

The screenshot shows a Postman interface with a successful response. The response body is the WSDL XML definition:

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

Create a new flow with an HTTP Listener connector endpoint

5. Return to Anypoint Studio.
6. Drag out another HTTP connector and drop it in the canvas after the existing flows.
7. Rename the flow to getDeltaFlightsFlow.
8. In the Properties view for the endpoint, set the connector configuration to the existing HTTP_Listener_Configuration.
9. Set the path to /delta.
10. Set the allowed methods to GET.

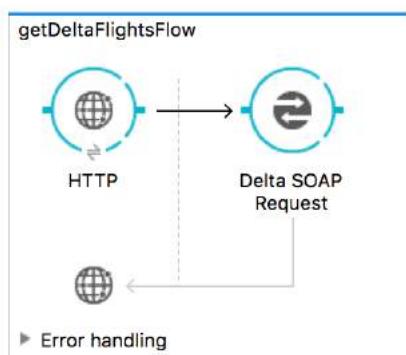
The screenshot shows the Anypoint Studio interface. At the top, there's a title bar with 'getDeltaFlightsFlow' and a 'Process' tab. Below the title bar is a diagram area containing an 'HTTP' connector icon. To the right of the diagram is an 'Error handling' section. At the bottom of the screen is a toolbar with tabs for 'Message Flow', 'Global Elements', and 'Configuration XML'. Below the toolbar is a status bar with 'HTTP X', 'Problems', and 'Console' buttons. The main workspace shows the flow definition. A properties panel on the right side displays the following configuration:

Path:	/delta
Allowed Methods:	GET

A note in the properties panel states: 'There are no errors.'

Add a Web Service Consumer connector endpoint

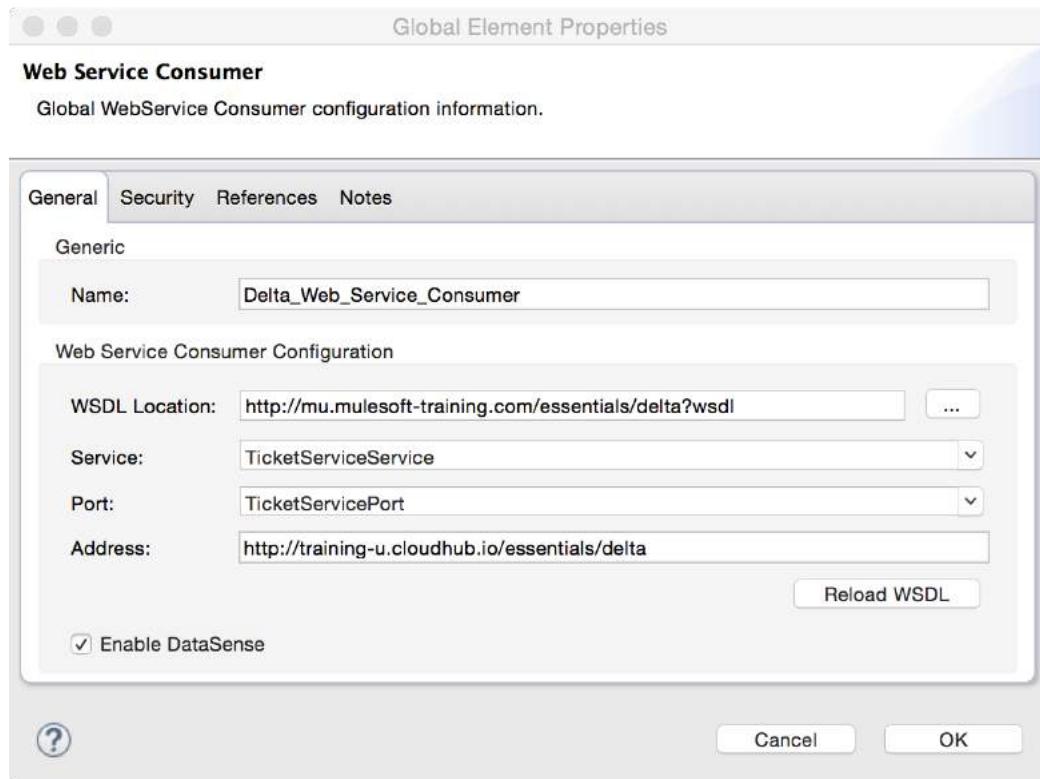
11. Drag out a Web Service Consumer connector and drop it in the process section of the flow.
12. Change its display name to Delta SOAP Request.



Configure the Web Service Consumer connector

13. Return to global.xml.
14. Click Create.
15. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer and click OK.
16. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer.
17. Set the WSDL location to the value you copied from the course snippets.txt file.
18. Wait for the service, port, and address fields to populate.

Note: If the fields do not populate, click the Reload WSDL button. If the fields still do not auto-populate, select TicketServiceService from the service drop-down menu and select TicketServicePort from the port drop-down menu.

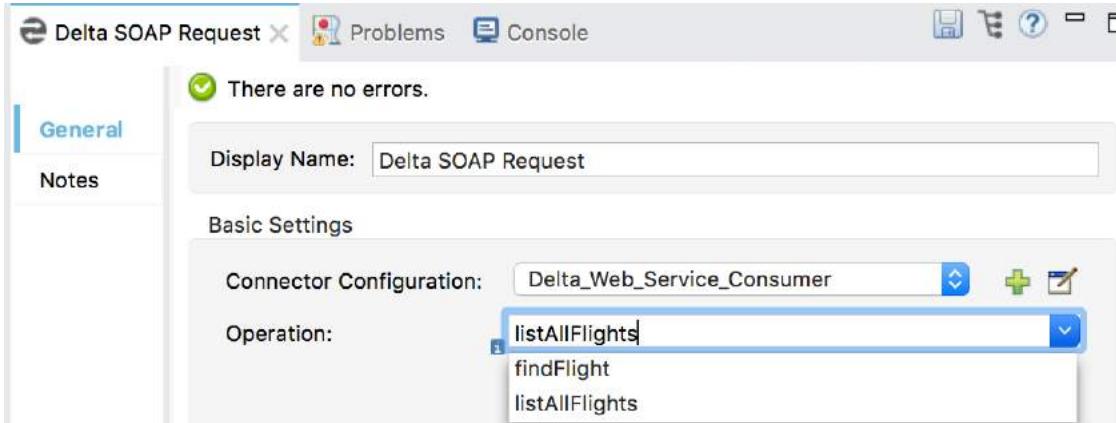


19. Click OK.

Configure the Web Service Consumer endpoint

20. Return to implementation.xml.

21. In the Delta SOAP Request properties view, set the connector configuration to the existing Delta_Web_Service_Consumer.
22. Click the operation drop-down menu button; you should see all of the web service operations listed.



23. Select the listAllFlights operation.

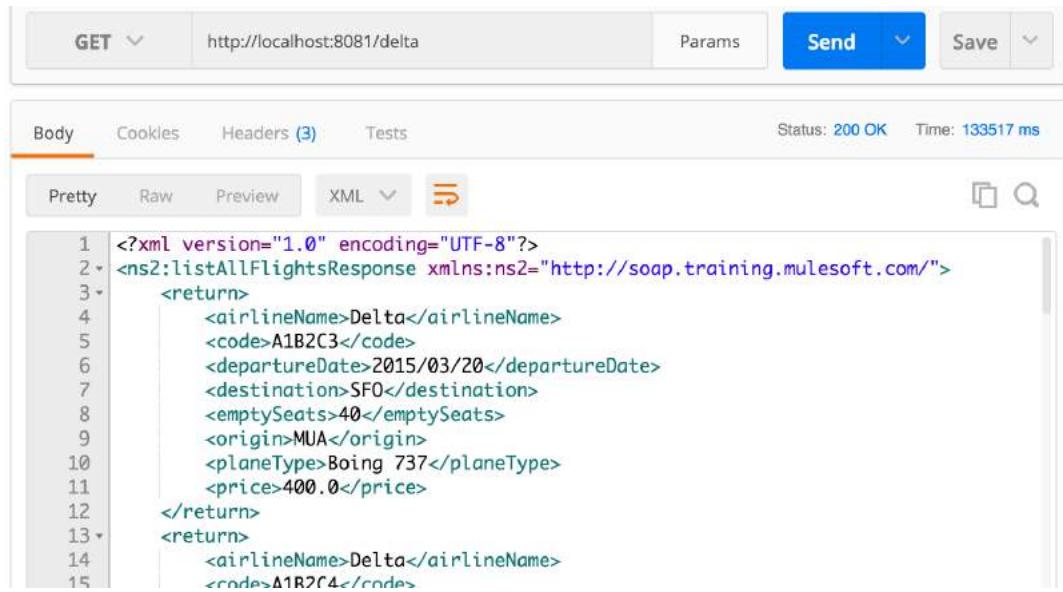
Test the application

24. Add a breakpoint to the Delta SOAP Request endpoint.
25. Add a Logger to the end of the flow.
26. Debug the project.
27. In Postman, return to the middle tab – the one with the localhost requests.
28. Make a request to <http://localhost:8081/delta>.
29. In the Mule Debugger, step to the Logger and look at the payload; it should be of type DepthXMLStreamReader.

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

30. Step through the application.

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



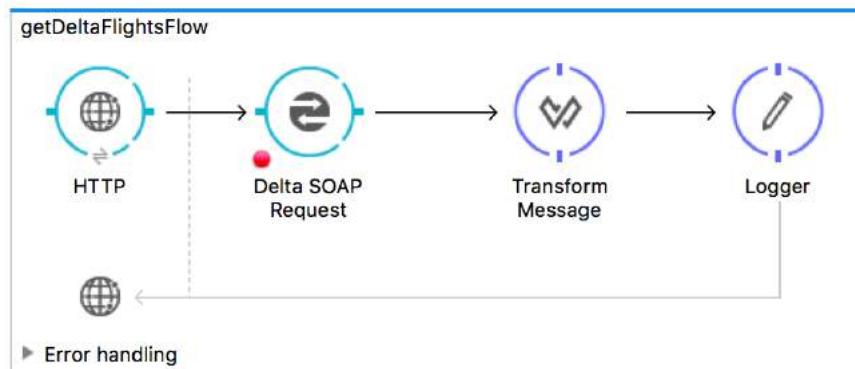
The screenshot shows the Postman interface with a successful HTTP request. The URL is `http://localhost:8081/delta`. The response status is `200 OK` and the time taken is `133517 ms`. The response body is displayed in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
  <return>
    <airlineName>Delta</airlineName>
    <code>A1B2C3</code>
    <departureDate>2015/03/20</departureDate>
    <destination>SFO</destination>
    <emptySeats>40</emptySeats>
    <origin>MUA</origin>
    <planeType>Boing 737</planeType>
    <price>400.0</price>
  </return>
  <return>
    <airlineName>Delta</airlineName>
    <code>A1R2C4</code>
```

Transform the data

32. Return to Anypoint Studio and stop the project.

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



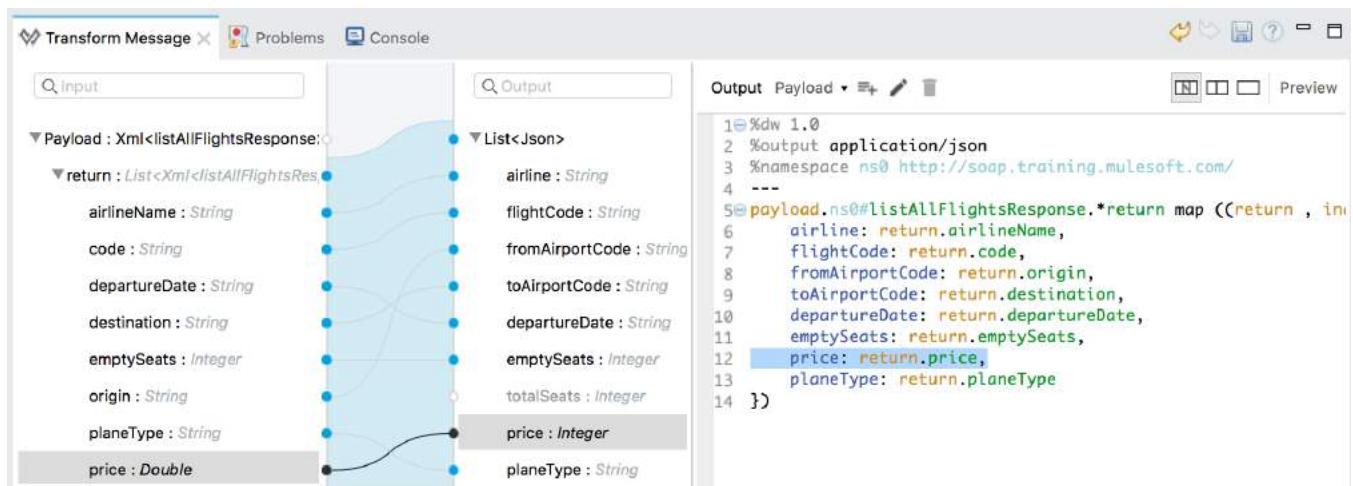
34. Look at the input section of the Transform Message properties view; you should see metadata already defined.

35. In the output section of the Transform Message properties view, click the Define metadata link.

36. In the Define metadata type dialog box, select flights_json.

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

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



Test the application

39. Run the project.
 40. In Postman, make another request to <http://localhost:8081/delta>; you should see all the flight data but now as JSON.

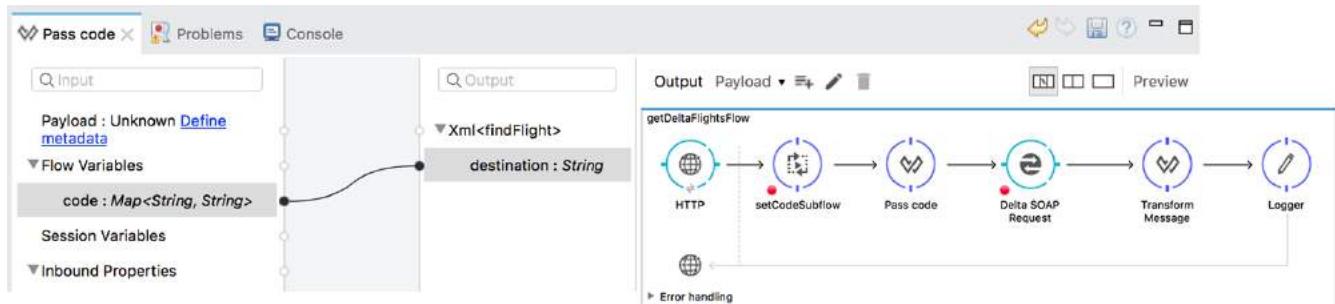
GET		localhost:8081/delta	Params	Send	Save
Body	Cookies	Headers (3)	Tests	Status: 200	Time: 1513 ms
Pretty	Raw	Preview	JSON	☰ □ 🔍	
<pre> 1 [2 { 3 "airline": "Delta", 4 "flightCode": "A1B2C3", 5 "fromAirportCode": "MUA", 6 "toAirportCode": "SFO", 7 "departureDate": "2015/03/20" </pre>					

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

Walkthrough 7-5: Pass arguments to a SOAP web service using DataWeave

In this walkthrough, you modify the Delta flow to return the flights for a specific destination instead of all the flights. You will:

- Change the web service operation invoked to one that requires a destination as an input argument.
- Set a flow variable to the desired destination.
- Use DataWeave to pass the flow variable to the web service operation.



Call a different web service operation

1. Return to getDeltaFlightsFlow.
2. In the Properties view for the Delta SOAP Request endpoint, change the operation to findFlight.



Test the application

3. Apply the changes to redeploy the application.

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

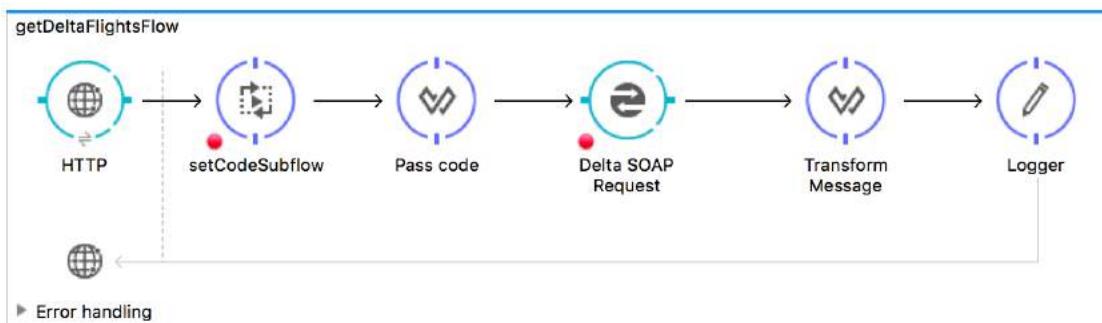
The screenshot shows a Postman interface. At the top, there's a header with 'GET' selected, the URL 'localhost:8081/delta?code=LAX', and a 'Send' button. Below the header, there's a status bar with 'Status: 500 No binding operation info while invoking unknown method with params unknown.' and 'Time: 734 ms'. The main area is titled 'Body' and contains a single line of text: 'i 1 | No binding operation info while invoking unknown method with params unknown..'. There are tabs for 'Pretty', 'Raw', 'Preview', and 'HTML'.

Use the set airport code subflow

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

Use DataWeave to pass parameters to the web service

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



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

The screenshot shows the DataWeave editor for the 'Pass code' component. On the left, under 'Input', there's a 'Payload : Unknown Define metadata' section and a 'Flow Variables' section containing a 'code : Map<String, String>' variable. On the right, under 'Output', there's a 'Payload' section with the following DataWeave code:

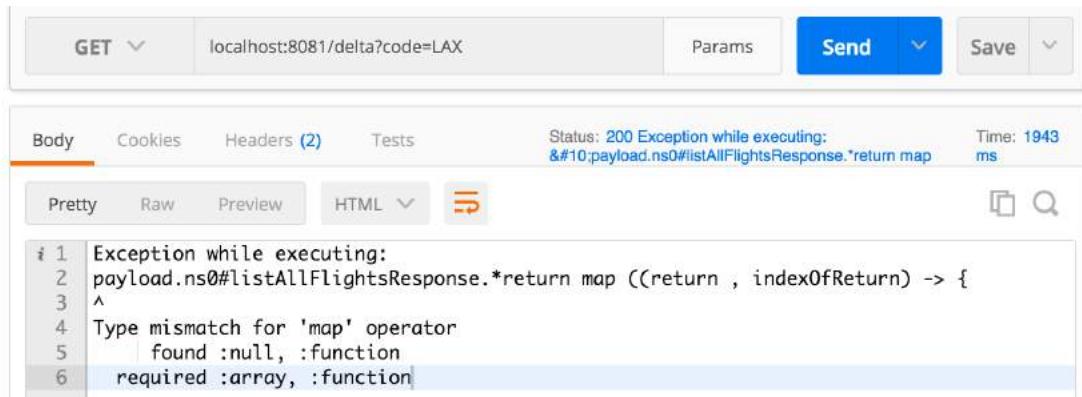
```

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

```

Test the application

12. Redeploy the application.
13. In Postman, make the same request; you should get a 200 status code with an exception message.



A screenshot of the Postman application interface. At the top, there's a header with 'GET' selected, the URL 'localhost:8081/delta?code=LAX', 'Params', a 'Send' button, and a 'Save' dropdown. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (2)', and 'Tests'. The 'Body' tab is active, showing a status message: 'Status: 200 Exception while executing:
payload.ns0#listAllFlightsResponse.*return map'. To the right of the status message, it says 'Time: 1943 ms'. Below the status message, there are buttons for 'Pretty', 'Raw', 'Preview', and 'HTML'. The 'Pretty' button is highlighted. The main body area contains a code block with lines 1 through 6. Line 1 starts with 'Exception while executing:', line 2 has 'payload.ns0#listAllFlightsResponse.*return map', line 3 has '(', line 4 has 'Type mismatch for 'map' operator', line 5 has '| found :null, :function', and line 6 has '| required :array, :function'.

14. Examine the exception message and figure out what is wrong with the transformation.

Modify the DataWeave expression

15. Return to getDeltaFlightsFlow.
16. Go to the Properties view for the Transform Message component after Delta SOAP Request.
17. Look at the transformation expression.
18. Change ns0#listAllFlightsResponse in the transformation code to ns0#findFlightResponse.



A screenshot of the MuleSoft Anypoint Studio DataWeave editor. The editor shows a code block with numbered lines from 1 to 14. Lines 1 through 4 are standard DataWeave setup: '%dw 1.0', '%output application/json', '%namespace ns0 http://soap.training.mulesoft.com/', and '---'. Lines 5 through 14 show a transformation rule: 'payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> { airline: return.airlineName, flightCode: return.code, fromAirportCode: return.origin, toAirportCode: return.destination, departureDate: return.departureDate, emptySeats: return.emptySeats, price: return.price, planeType: return.planeType })'. The 'Payload' dropdown is set to 'application/json'. There are three preview icons on the right: 'Text', 'XML', and 'JSON'. A 'Preview' button is also present.

Test the application

19. Redeploy the application.

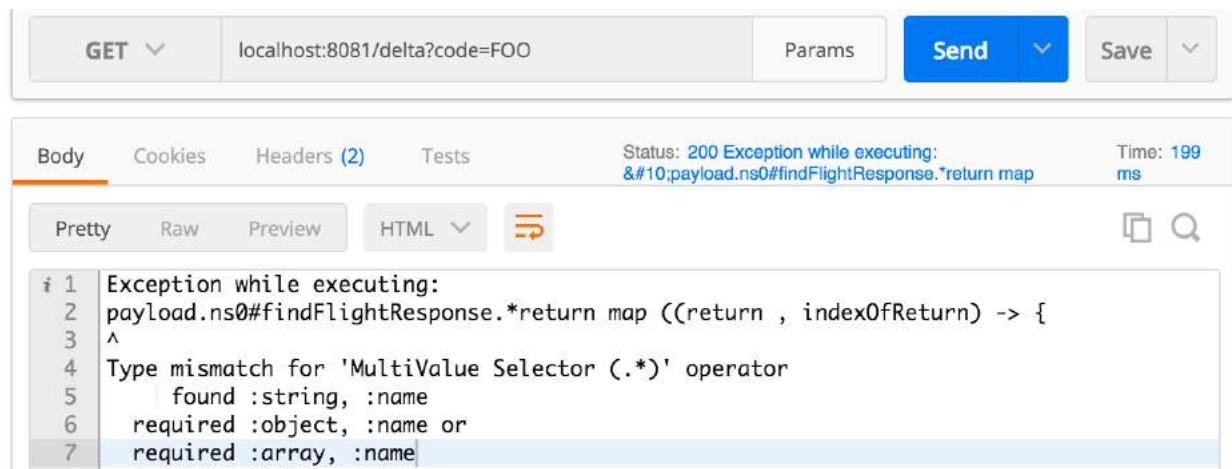
20. In Postman, make the same request; you should now get all the flights to LAX.



The screenshot shows the Postman interface with a successful HTTP request. The URL is `localhost:8081/delta?code=LAX`. The response status is 200 and the time taken is 450 ms. The response body is displayed in Pretty JSON format:

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

21. Change the code query parameter to have a value of CLE.
22. Send the request; you should now get only flights to CLE.
23. Change the code query parameter to have a value of FOO.
24. Send the request; you should get a 200 status code and a message with an exception.

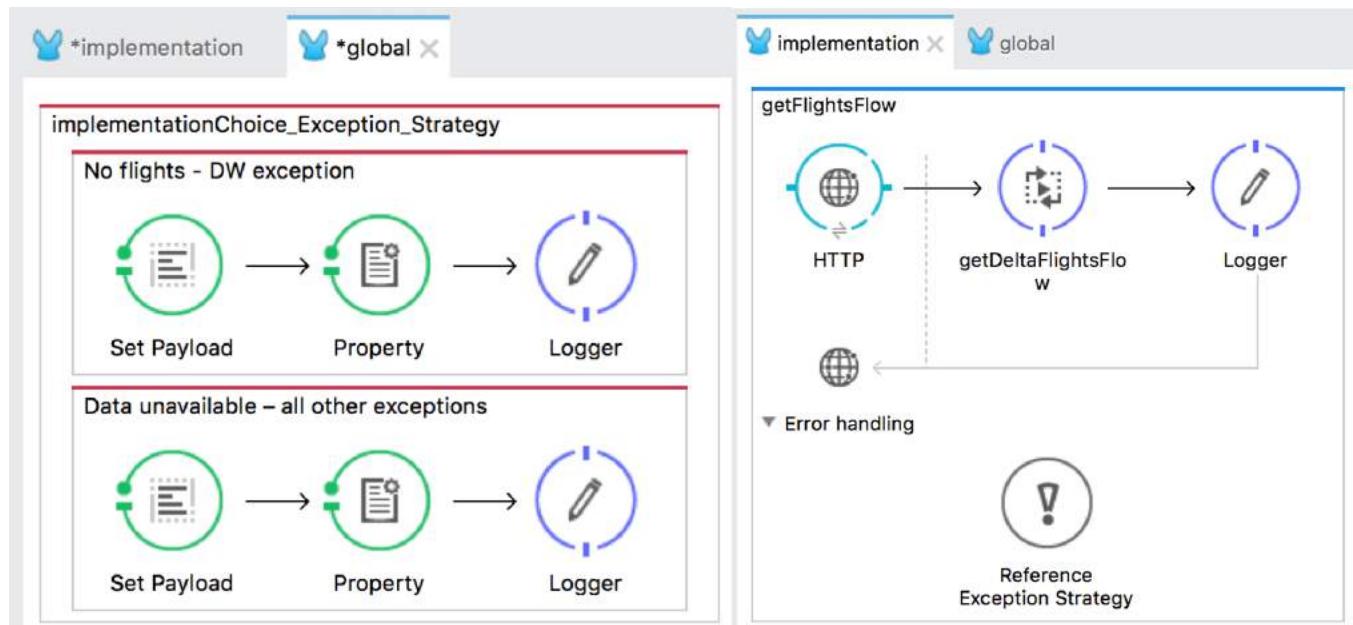


The screenshot shows the Postman interface with an error response. The URL is `localhost:8081/delta?code=FOO`. The status is 200, but there is an exception message: "Exception while executing: payload.ns0#findFlightResponse.*return map". The response body is displayed in HTML format:

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

25. Return to Anypoint Studio and stop the project.

Module 8: Handling Errors



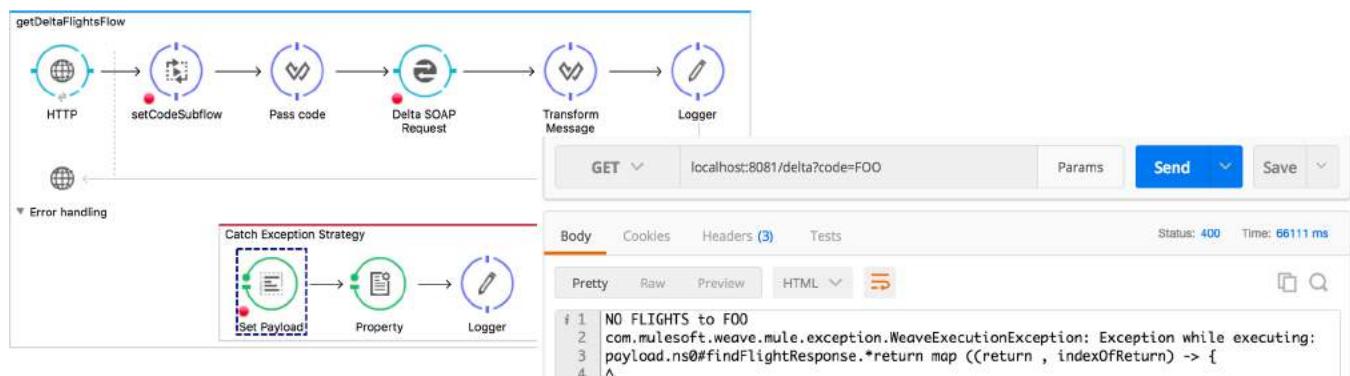
Objectives:

- Describe the different types of exception strategies.
- Handle messaging exceptions in flows.
- Create and use global exception handlers.
- Specify a global default exception strategy.

Walkthrough 8-1: Handle a messaging exception

In this walkthrough, you handle an exception thrown by the Delta flow when a destination with no flights is used. You will:

- Add a catch exception strategy to a flow.
- Catch the exception and set the payload to send an error message back.
- Reference the exception object inside an exception handler.
- Set an HTTP status code inside the exception handler.



Debug the application for a request with a non-existent destination

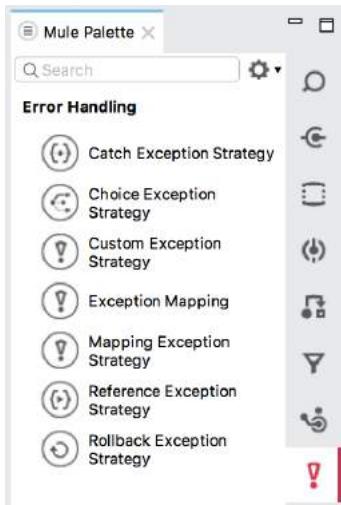
1. Return to getDeltaFlightsFlow.
2. Make sure there is a breakpoint on the flow reference component.
3. Debug the project.
4. In Postman, make another request to <http://localhost:8081/delta?code=FOO>.
5. In the Mule Debugger, step through the application and when you get the exception, drill-down into the exceptionThrown object.

Name	Value	Type
» E Exception	SimpleDataType{type=org.apache.cxf.staxuti... org.mule.transformer.types.SimpleDataType	org.mule.transformer.types.SimpleDataType
» E exceptionThrown	com.mulesoft.weave.mule.exception.WeaveE... com.mulesoft.weave.mule.exception.WeaveE...	com.mulesoft.weave.mule.exception.WeaveE...
» E cause	com.mulesoft.weave.engine.ast.dynamic.Une...	com.mulesoft.weave.engine.ast.dynamic.Une...
» A CAUSE_CAPTION	Caused by: null	java.lang.String
» A causeRollback	false	java.lang.Boolean
» A detailMessage	com.mulesoft.weave.engine.ast.dynamic.Une...	java.lang.String
» E EMPTY_THROWABLE_ARRAY	[Ljava.lang.Throwable;@1fc5724b	java.lang.Throwable[]
» A errorCode	-1	java.lang.Integer
» E event	MuleEvent: 0-28442770-1c52-11e6-b0a8... org.mule.DefaultMuleEvent	org.mule.DefaultMuleEvent
» A EXCEPTION_MESSAGE_DELIMITER	*****	java.lang.String
» A EXCEPTION_MESSAGE_SECTION...	-----	java.lang.String
» A failingMessageProcessor	null	org.mule.api.processor.MessageProcessor
» A handled	false	java.lang.Boolean

6. Click the Resume button.

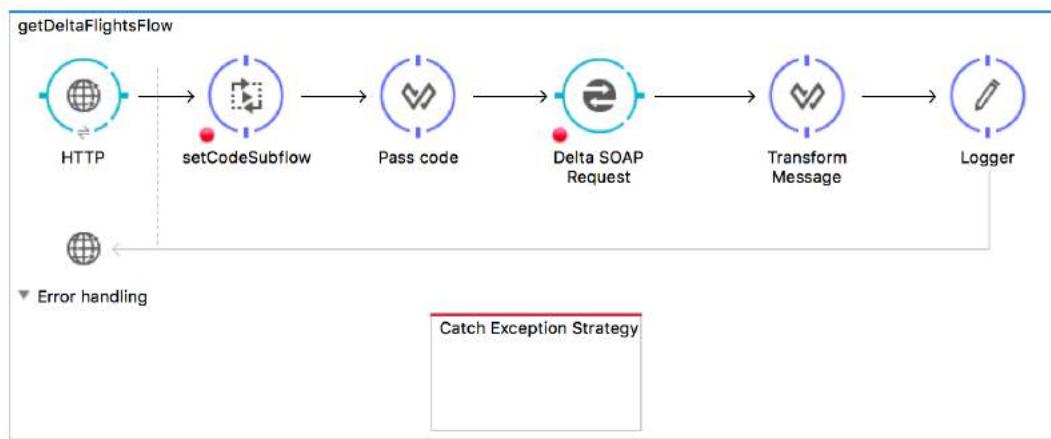
Browse the error handling elements in the Mule Palette

7. In the Mule Palette, select the Error Handling tab.
8. View the available error handling processors.



Add a catch exception strategy

9. In getDeltaFlightsFlow, click the arrow to expand the Error handling section.
10. Drag a Catch Exception Strategy from the Mule Palette into the error handling section of the flow.

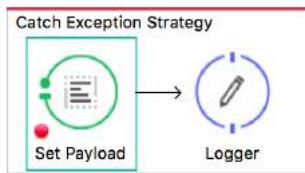


11. Add a Set Payload transformer to the catch exception strategy.
12. In the Set Payload properties view, set the value the following MEL expression:

```
NO FLIGHTS to #[flowVars.code + '\n' + exception]
```

13. Make sure there is a breakpoint on the transformer inside the catch exception.

14. Add a Logger after the transformer.

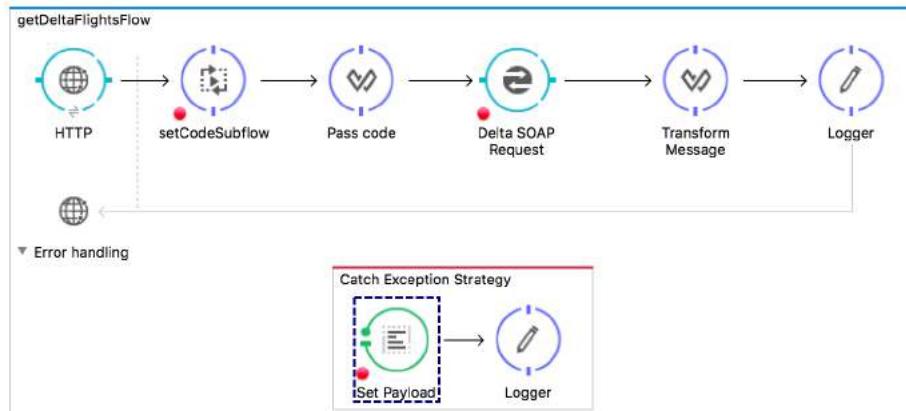


Test the application

15. Redeploy the application in debug mode.

16. In Postman, make another request to <http://localhost:8081/delta?code=FOO>.

17. Step through the application; you should see the exception thrown in getDeltaFlightsFlow and handled by the exception handler.



18. Step to the end of the application.

19. In Postman, you should get a 200 response with your custom message.

Screenshot of Postman showing a successful API call to `localhost:8081/delta?code=FOO`. The response body contains a custom error message:

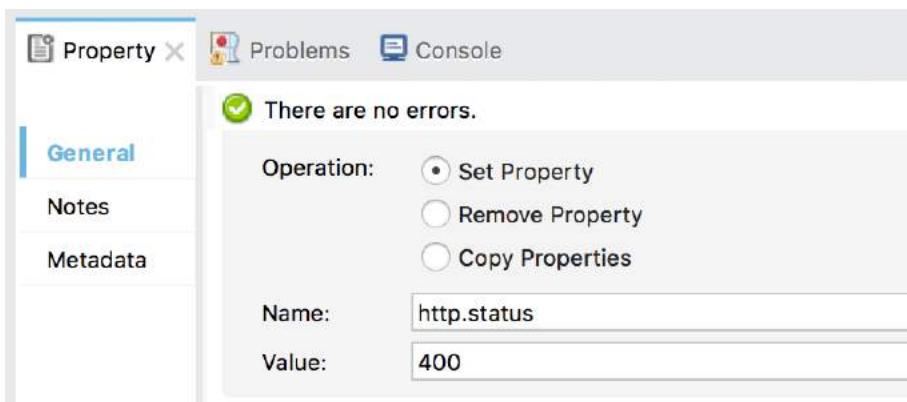
```
i 1 NO FLIGHTS to FOO
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {
4 ^
5 Type mismatch for 'MultiValue Selector (*.*)' operator
6     found :string, :name
7     required :object, :name or
8     required :array, :name|
```

Set the http status code

20. Return to getDeltaFlightsFlow.
21. In the catch exception strategy, add a Property transformer after the Set Payload transformer.

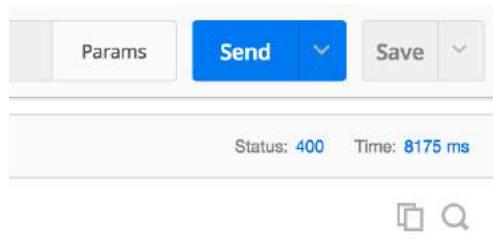


22. In the Property properties view, select Set Property.
23. Set the name to http.status and the value to 400.



Test the application

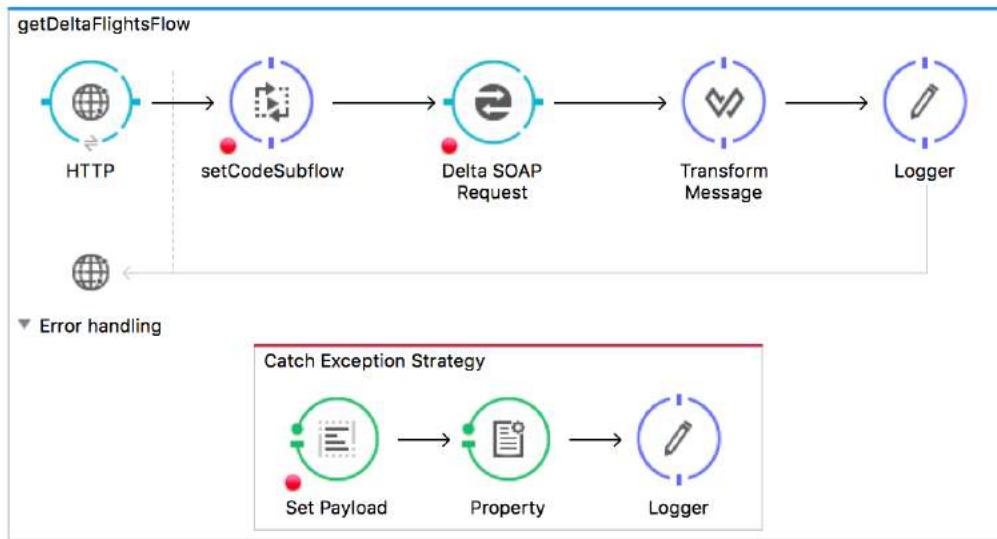
24. Redeploy the application.
25. In Postman, make another request to <http://localhost:8081/delta?code=FOO>.
26. In the Mule Debugger, step through the application.
27. Return to Postman; you should now get a 400 status code.



Create a different type of error

28. Return to getDeltaFlightsFlow.

29. Delete the Pass code processor.



Test the application

30. Redeploy the application.
31. In Postman, delete the query parameter and send the request.
32. In the Mule Debugger, step through the application until you get the exception and move into the exception handler; you should see the exception is handled by the same exception handler.
33. Drill-down into the Exception object in the debugger.
34. Click Resume.
35. In Postman, you should get the same message, but this time saying there are no flights to SFO even though that was not the problem.

GET localhost:8081/delta Params Send Save

Status: 400 Time: 8658 ms

Body Cookies Headers (3) Tests

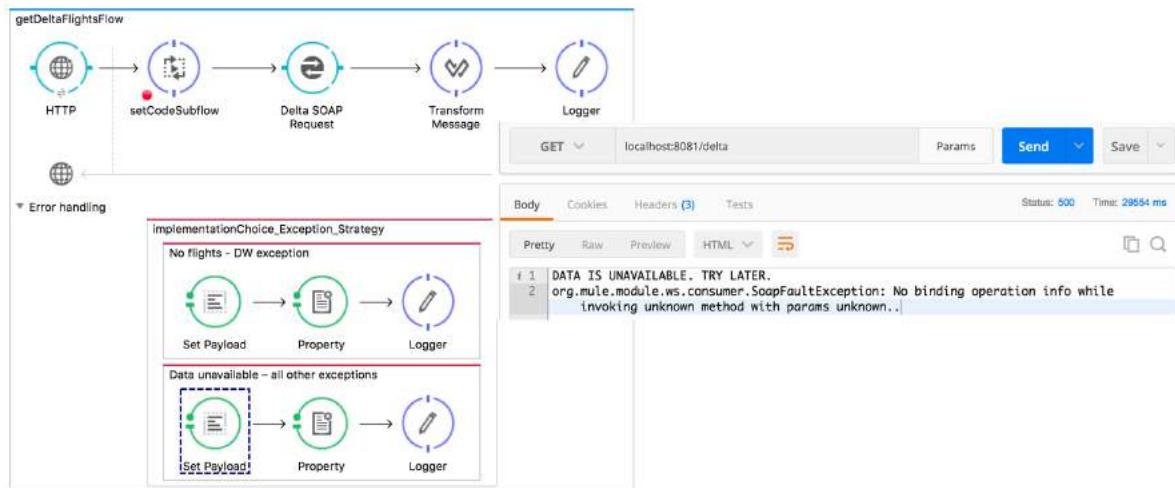
Pretty Raw Preview HTML HTML CSV JSON XML Text Copy Search

```
i 1 NO FLIGHTS to SFO
2 org.mule.module.ws.consumer.SoapFaultException: No binding operation info while
   invoking unknown method with params unknown...
```

Walkthrough 8-2: Handle different types of messaging exceptions

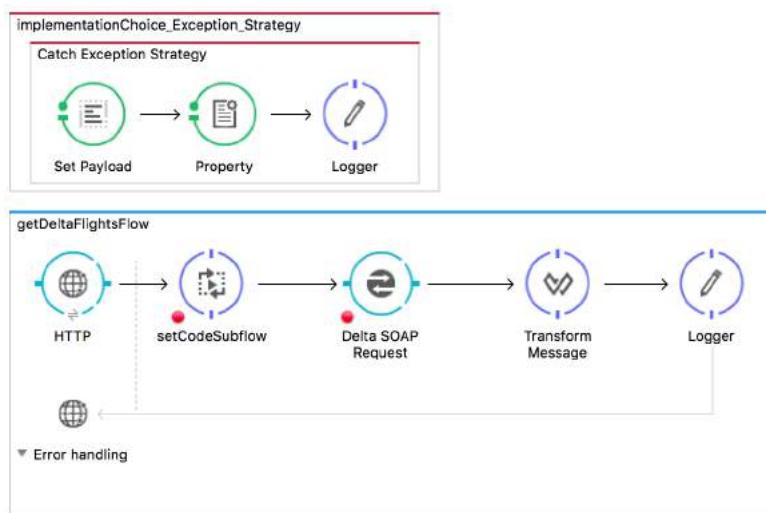
In this walkthrough, you handle multiple types of exceptions thrown by the Delta flow. You will:

- Add and configure a choice exception strategy.
- Get exceptions handled by both of the catch exception strategies in the choice strategy.
- Create a new flow that calls a flow that has an exception so the exception can bubble up and be handled by the calling flow.



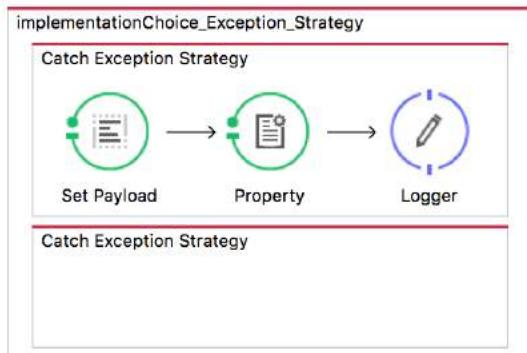
Add a choice exception strategy

1. Return to implementation.xml.
2. Drag a Choice Exception Strategy from the Mule Palette and drop it above getDeltaFlightsFlow.
3. Drag the catch exception strategy from getDeltaFlightsFlow and drop it in the choice exception strategy.



4. Drag a second Catch Exception Strategy from the Mule Palette and drop it in the choice exception strategy.

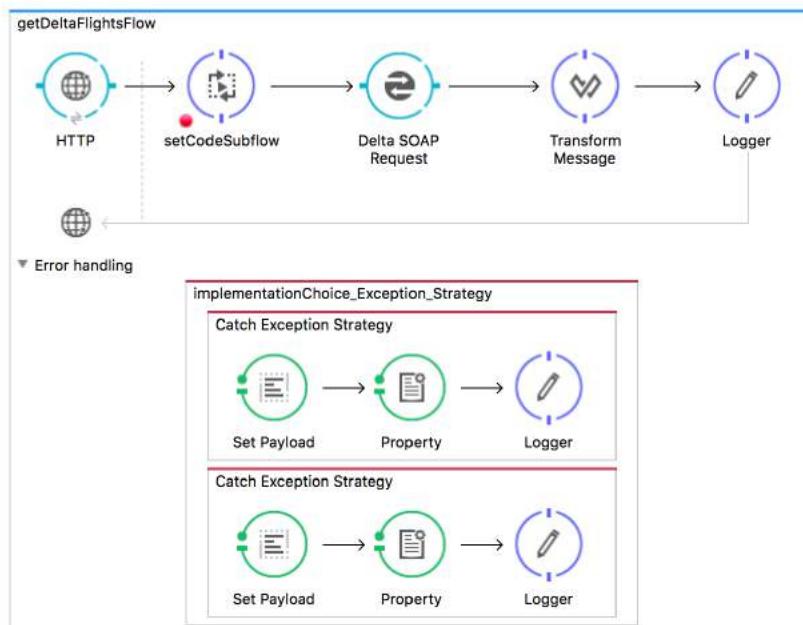
Note: If you are having difficulty adding it, try dropping it on the catch exception strategy already there.



5. Drag the choice exception strategy and drop it in the error handling section of getDeltaFlightsFlow.

Note: If you are having difficulty adding it because the canvas is scrolling, try changing the order of the flows on the canvas and then dragging and dropping.

6. Add a Set Payload transformer, a Property transformer, and a Logger to the new catch exception strategy.



7. In the Set Payload properties view, set the value the following MEL expression:

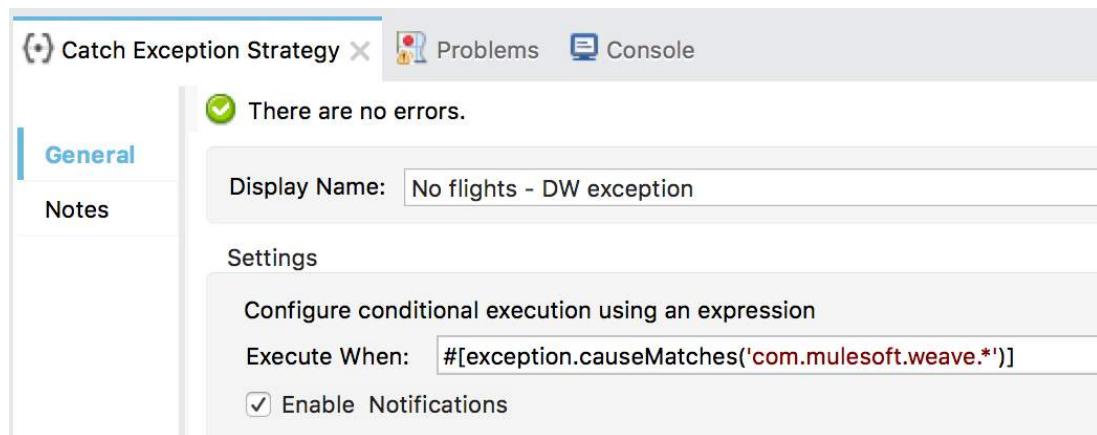
```
DATA IS UNAVAILABLE. TRY LATER. #[ '\n' + exception ]
```

8. In the Property properties view, select Set Property.
9. Set the name to http.status and the value to 500.

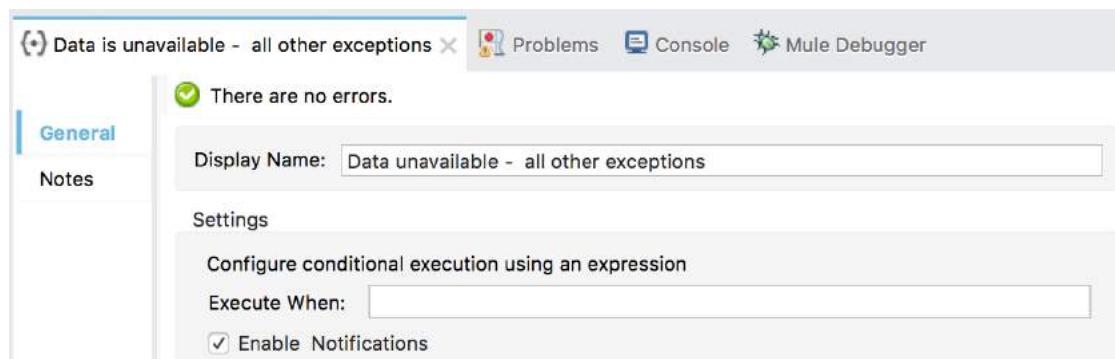
Configure the choice exception strategy

10. In the Properties view for the first catch exception strategy, set the display name to No flights – DW exception.
11. Set the execute when value to an expression for when a com.mulesoft.weave.* exception is thrown; use the causeMatches() method.

```
##[exception.causeMatches('com.mulesoft.weave.*')]
```



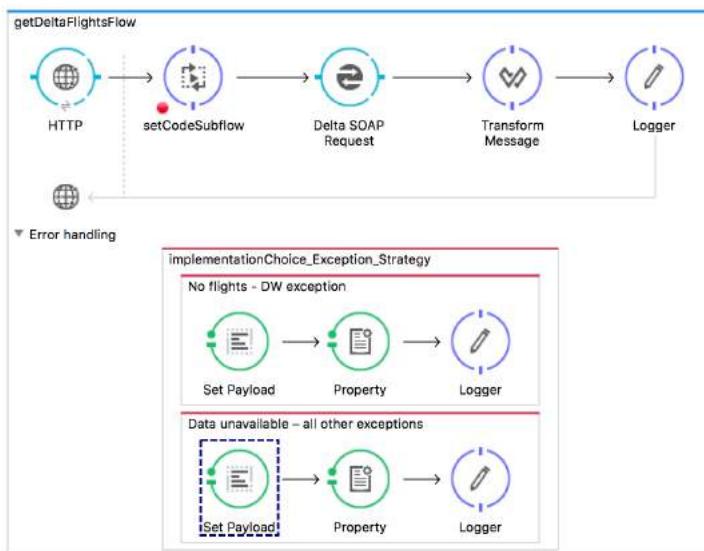
12. In the Properties view for the second catch exception strategy, set the display name to Data unavailable – all other exceptions.
13. Leave the execute when value blank.



Test the application

14. Debug the application.
15. In Postman, make another request to <http://localhost:8081/delta>.

16. In the Mule Debugger, step through the application; the exception should be handled by the second catch block.



17. Step to the end of the application.

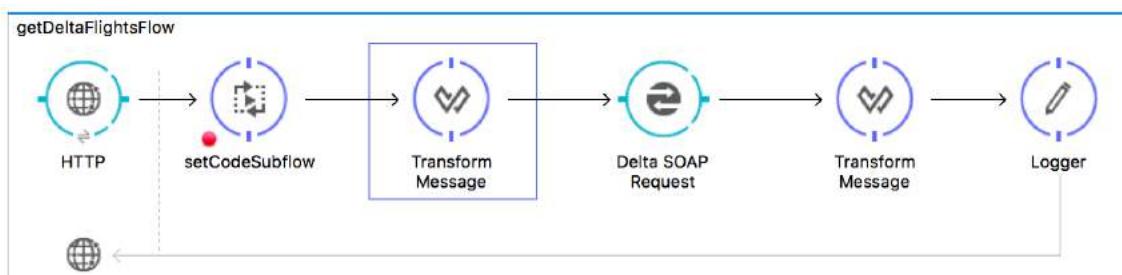
18. Return to Postman; you should see the second error message.

A screenshot of the Postman application interface. The top bar shows 'GET' selected, the URL 'localhost:8081/delta', and various buttons like 'Send' and 'Save'. Below the URL, tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests' are visible, along with status information 'Status: 500 Time: 29554 ms'. The 'Body' tab is active, showing a 'Pretty' view of the response. The response body contains the text 'DATA IS UNAVAILABLE. TRY LATER.' followed by a detailed error message: 'org.mule.module.ws.consumer.SoapFaultException: No binding operation info while invoking unknown method with params unknown..|'. There are also 'Raw' and 'Preview' buttons, and a search icon.

Fix the request error

19. Return to getDeltaFlightsFlow.

20. Add a Transform Message component before the Delta SOAP Request processor.



21. In the Transform Message properties view, drag the code flow variable in the input section to the destination element in the output section.

Test the application

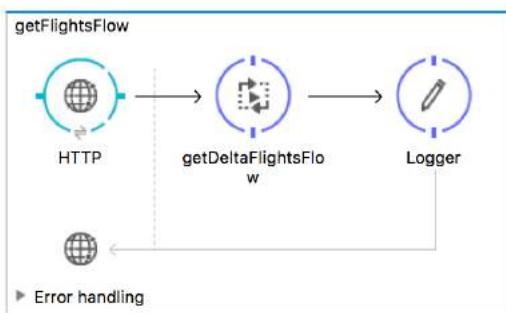
22. Run the project.
23. In Postman, make the same request to <http://localhost:8081/delta>; you should get the SFO results again.
24. Make a request to <http://localhost:8081/delta?code=FOO>; the exception is handled by the first exception strategy and you should get the NO FLIGHTS TO FOO message.

The screenshot shows a Postman interface. At the top, there's a header bar with 'GET' selected, a URL field containing 'localhost:8081/delta?code=FOO', a 'Params' button, a 'Send' button, and a 'Save' button. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests'. The 'Body' tab is active. On the right side of the interface, it says 'Status: 400' and 'Time: 185 ms'. Under the 'Body' tab, there are buttons for 'Pretty', 'Raw', 'Preview', and 'HTML'. The 'Pretty' button is selected. The response body is displayed in a code-like format:
1 NO FLIGHTS to FOO
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {

Call the Delta flow from another flow

25. Return to Anypoint Studio.
26. From the Mule Palette, drag an HTTP connector and drop it at the top of the canvas above all the other flows.
27. In the HTTP properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.
28. Set the path to `/flights` and the allowed methods to `GET`.
29. Change the flow name to `getFlightsFlow`.
30. Add a Flow Reference component to the flow.
31. In the Flow Reference properties view, set the flow to `getDeltaFlightsFlow`.

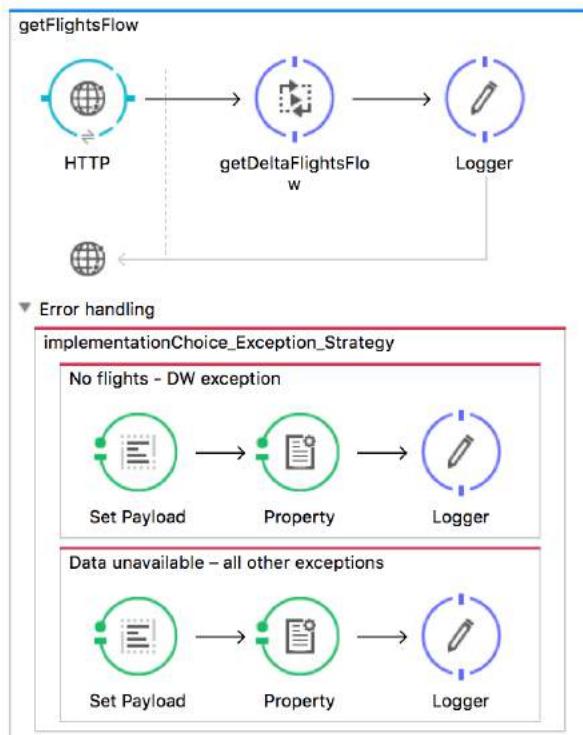
32. Add a Logger to the end of the flow.



Move a catch exception strategy to the calling flow

33. Expand the error handling section of getFlightsFlow.

34. Move the choice exception strategy from getDeltaFlightsFlow to getFlightsFlow.



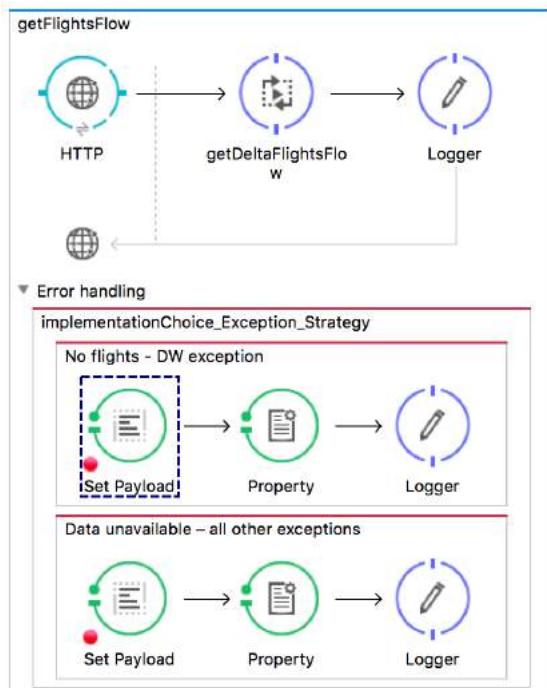
Test the application

35. Debug the project.

36. Make sure there is a breakpoint on each of the Set Payload transformers in the catch exception strategies.

37. In Postman, make a request to <http://localhost:8081/flights?code=FOO>.

38. In the Mule Debugger, step through the application; you should see the exception thrown by the Delta flow is caught by the calling flow.

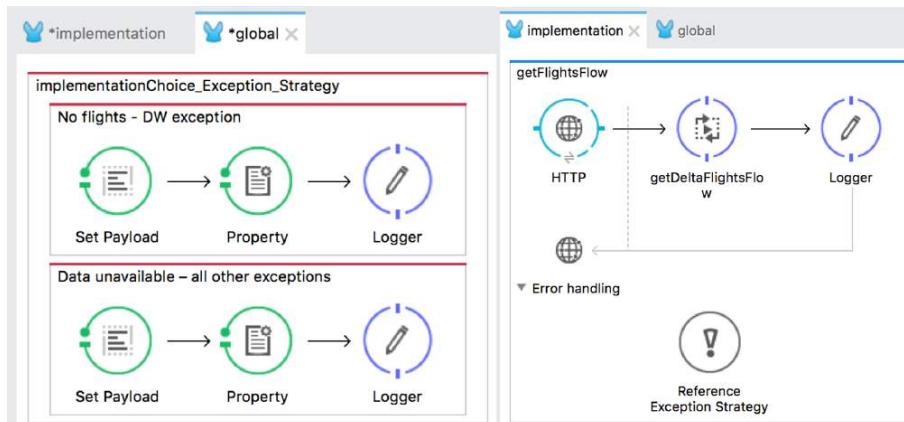


39. Step through the rest of the application and then stop the project.

Walkthrough 8-3: Create and use global exception strategies

In this walkthrough, you create a global exception handler for the project. You will:

- Create a global exception handler.
- Reference and use the global exception handler in flows.



Create a global exception handler

1. Return to implementation.xml.
2. Switch to the Configuration XML view.
3. Locate the choice exception strategy and select it and cut it.

```
<flow name="getFlightsFlow">
    <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" doc:name="HTTP"/>
    <flow-ref name="getDeltaFlightsFlow" doc:name="getDeltaFlightsFlow"/>
    <logger level="INFO" doc:name="Logger"/>
    <choice-exception-strategy doc:name="implementationChoice_Exception_Strategy">
        <catch-exception-strategy when="#[exception.causeMatches('com.mulesoft.weave.flows.core.MuleException')]">
            <set-payload value="NO FLIGHTS to #[flowVars.code + '\n' + exception]" doc:name="Set Payload"/>
            <set-property propertyName="http.status" value="400" doc:name="Property"/>
            <logger level="INFO" doc:name="Logger"/>
        </catch-exception-strategy>
        <catch-exception-strategy doc:name="Data unavailable &#8211; all other exceptions">
            <set-payload value="DATA IS UNAVAILABLE. TRY LATER. #['\n' + exception]" doc:name="Set Payload"/>
            <set-property propertyName="http.status" value="500" doc:name="Property"/>
            <logger level="INFO" doc:name="Logger"/>
        </catch-exception-strategy>
    </choice-exception-strategy>
</flow>
<sub-flow name="setCodeSubflow">
```

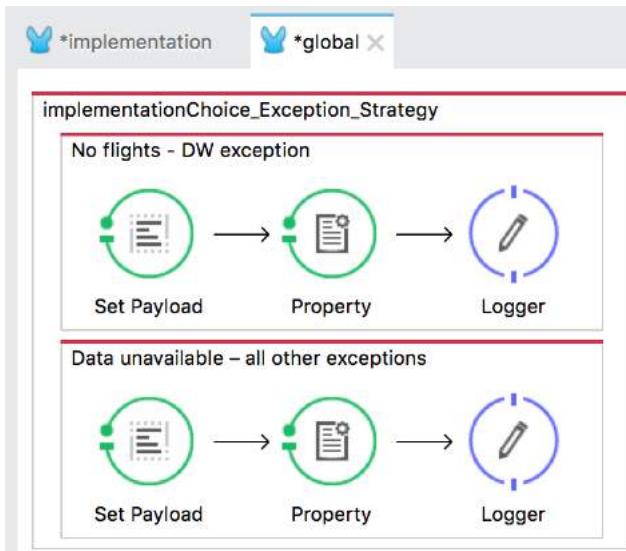
4. Go to the Configuration XML view in global.xml.
5. Place the cursor on a new line inside the start and end mule tags after the other tags.
6. Paste the choice exception strategy.

7. In the choice-exception-strategy tag, change doc:name to name; the problem should go away



```
17    </http:request-config>
18    <ws:consumer-config name="Delta_Web_Service_Consumer" service="Ticketing">
19
20    <choice-exception-strategy name="implementationChoice_Exception_Strategy">
21        <catch-exception-strategy when="#[exception.causeMatches('No FLIGHTS')]">
22            <set-payload value="NO FLIGHTS to #[flowVars.code + '\n']"/>
23            <set-property propertyName="http.status" value="400" doc:name="Set HTTP Status" />
24            <logger level="INFO" doc:name="Logger"/>
25        </catch-exception-strategy>
26        <catch-exception-strategy doc:name="Data unavailable &#8211; all other exceptions">
27            <set-payload value="DATA IS UNAVAILABLE. TRY LATER. #['"/>
28            <set-property propertyName="http.status" value="500" doc:name="Set HTTP Status" />
29            <logger level="INFO" doc:name="Logger"/>
30        </catch-exception-strategy>
31    </choice-exception-strategy>
32
33 </mule>
```

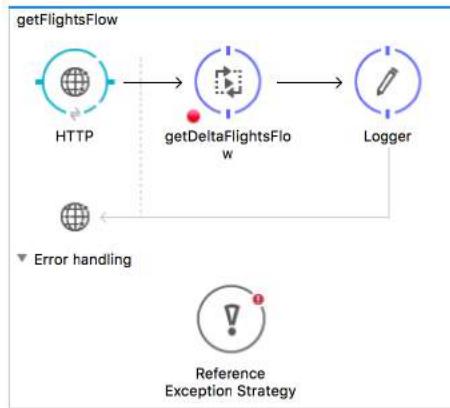
8. Switch to the Message Flow view; you should see the choice exception strategy.



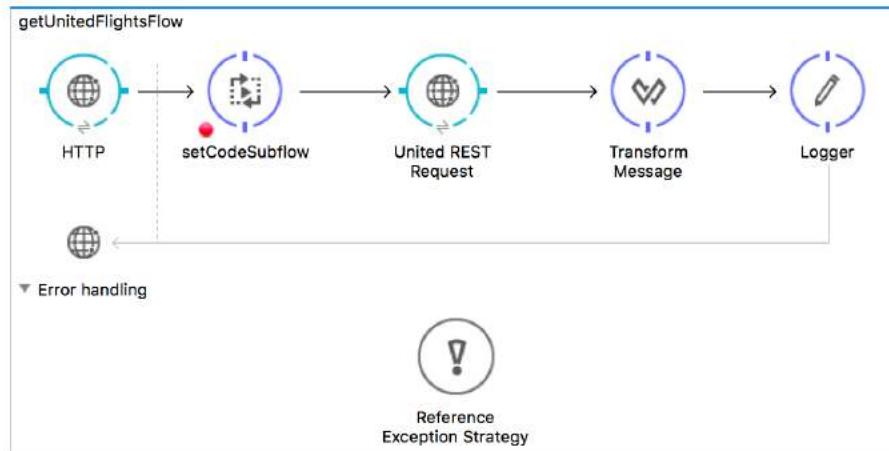
Use the global exception handler

9. Return to implementation.xml and switch to the Message Flow view.
10. Locate getFlightsFlow and expand its error handling section.

11. Drag a Reference Exception Strategy from the Mule Palette and drop it in the error handling section.



12. In the Reference Exception Strategy properties view, set the global exception strategy to the existing implementationChoice_Exception_Strategy.
13. Expand the error handling section of getUnitedFlightsFlow.
14. Drag a Reference Exception Strategy from the Mule Palette and drop it in the error handling section.
15. In the Properties view, set the global exception strategy to implementationChoice_Exception_Strategy.



Note: You could add a reference exception strategy to the rest of the flows, but instead, you will create a global default exception strategy in the next walkthrough.

Test the application

16. Run the project.

17. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>; you should get the same NO FLIGHTS to FOO message.

GET localhost:8081/delta?code=FOO Params Send Save

Body Cookies Headers (3) Tests Status: 400 Time: 185 ms

Pretty Raw Preview HTML `≡` □ 🔍

```
i 1 NO FLIGHTS to FOO
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {
```

18. Make a request to <http://localhost:8081/united?code=FOO>; you should see the same message.

19. Make a request to <http://localhost:8081/delta?code=FOO>; you should NOT see the same message because the exception is not being handled.

GET localhost:8081/delta?code=FOO Params Send Save

Body Cookies Headers (2) Tests Status: 200 Exception while executing:
payload.ns0#findFlightResponse.*return map Time: 118 ms

Pretty Raw Preview HTML `≡` □ 🔍

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

Walkthrough 8-4: Specify a global default exception strategy

In this walkthrough, you change the default exception handling for the application. You will:

- Create a global configuration element in the global.xml file.
- Specify a default exception strategy in the global configuration element.
- Remove the existing exception handling strategies.
- Use the default exception handling strategy.

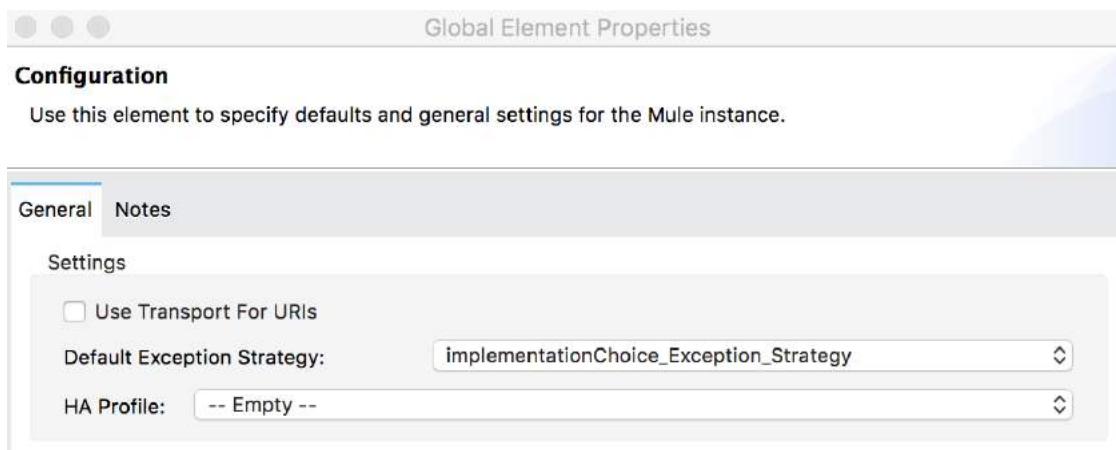
The screenshot shows the 'Global Mule Configuration Elements' view in the Mule Studio interface. The left pane lists configuration elements with their types and names. The right pane shows the 'Global Element Properties' for a selected configuration element, specifically the 'Configuration' type. It includes sections for General settings, Notes, and Settings, where the 'Default Exception Strategy' is set to 'implementationChoice_Exception_Strategy'.

Specify a global default exception strategy

1. Return to global.xml.
2. Switch to the Global Elements view.
3. Click the Create button.
4. In the Choose Global Type dialog box, select Global Configurations > Configuration and click OK.

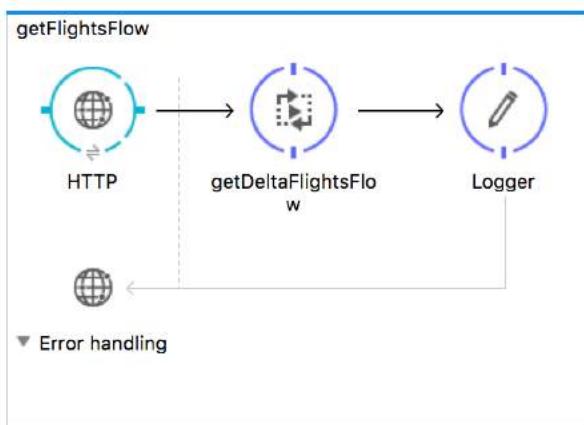
The screenshot shows the 'Choose Global Type' dialog box. It has a search bar at the top and a tree view below it. The 'Configuration' node under 'Global Configurations' is selected, highlighted with a grey background.

5. In the Global Element Properties dialog box, set the default exception strategy to implementationChoice_Exception_Strategy and click OK.

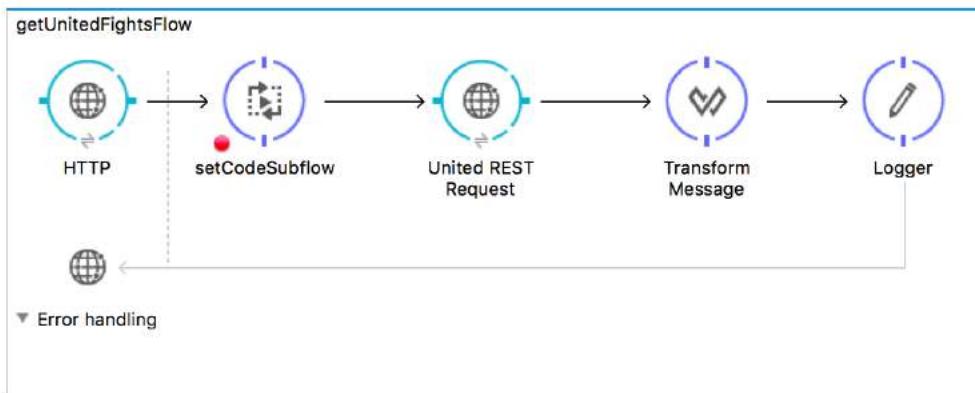


Remove the existing exception strategy references

6. Return to implementation.xml.
7. Delete the reference exception strategy in getFlightsFlow.



8. Delete the reference exception strategy in getUnitedFlightsFlow.



Test the application

9. Save all the files and run the project.
10. In Postman, make a request to <http://localhost:8081/flights?code=FOO>; you should still see the no flights error message displayed.
11. Make a request to <http://localhost:8081/delta?code=FOO>; you should now see the no flights error message displayed.

The screenshot shows the Postman interface with a GET request to `localhost:8081/delta?code=FOO`. The response is **400 OK** with a time of **132 ms**. The Body tab displays the following JSON response:

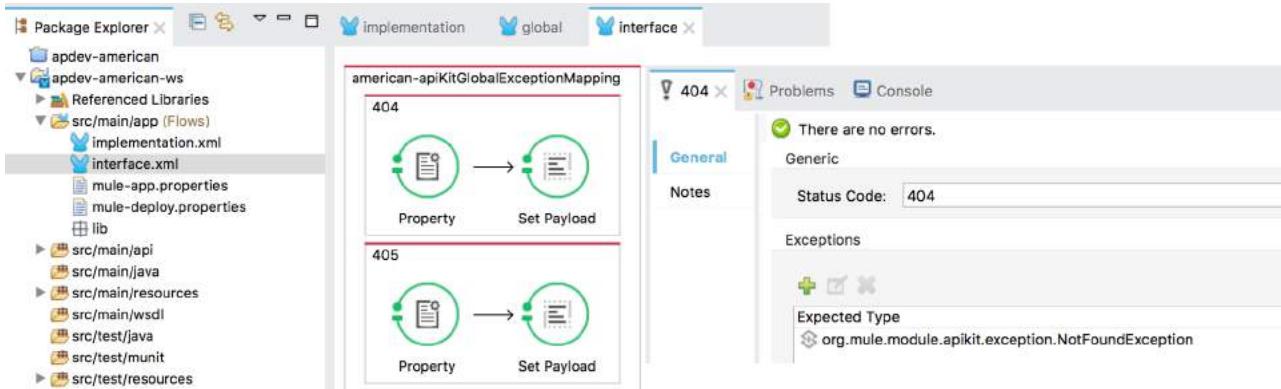
```
{  
  "error": "NO FLIGHTS to FOO",  
  "exception": "com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:"}
```

12. Return to Anypoint Studio and stop the project.

Walkthrough 8-5: Review a mapping exception strategy

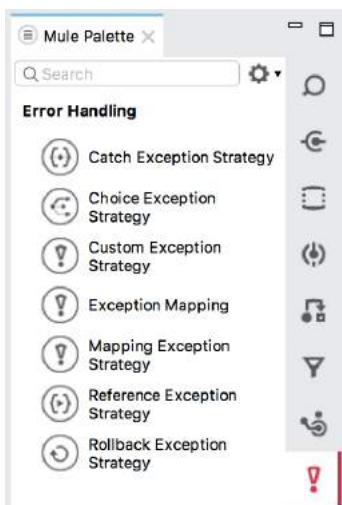
In this walkthrough, you review the mapping exception strategy that was created automatically by APIkit for the interface for the American flights web service. You will:

- Locate the mapping exception strategy.
- Review the exception mappings.



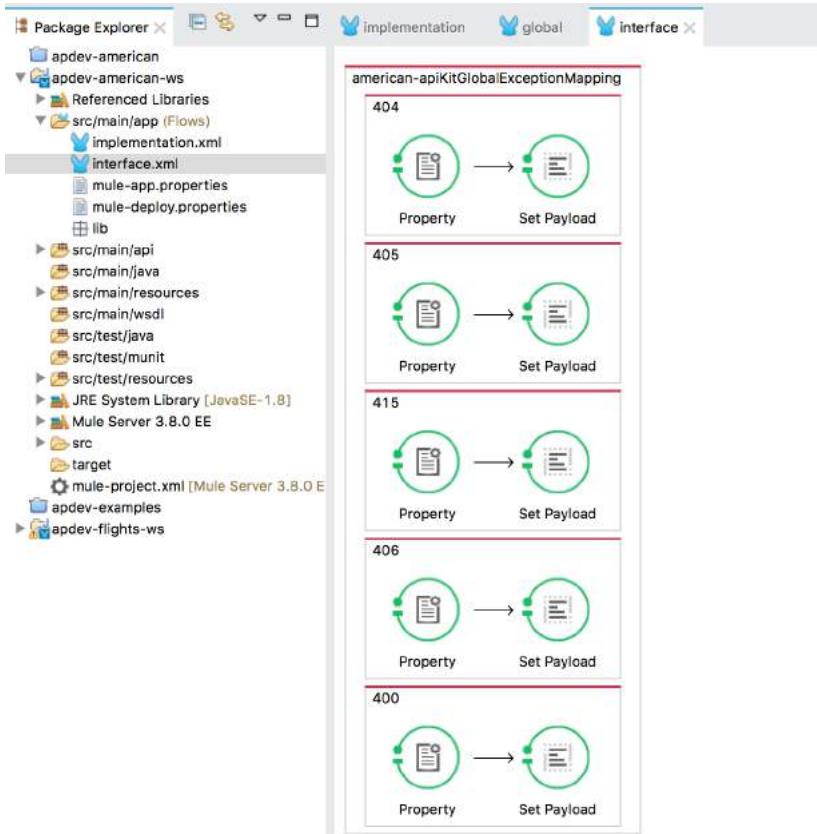
Browse the error handling elements in the Mule Palette

1. In the Package Explorer, double-click the apdev-american-ws project to open it.
2. Open interface.xml in src/main/app.
3. In the Mule Palette, select the Error Handling tab.
4. Locate the Mapping Exception Strategy.

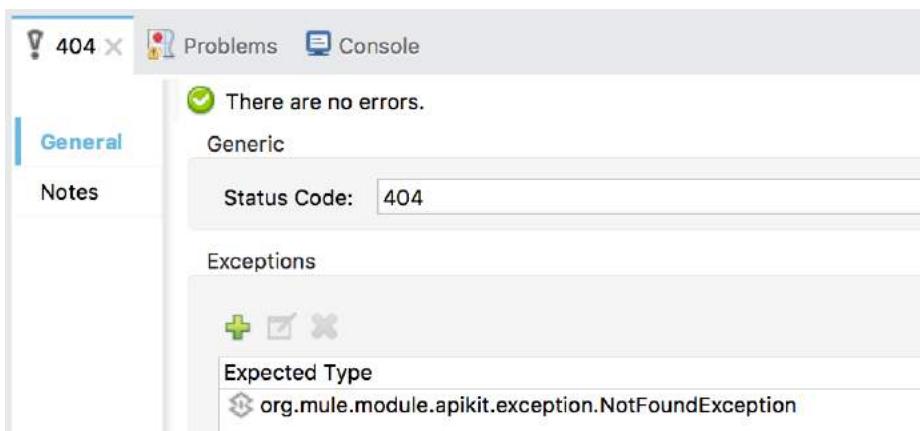


Review a mapping exception strategy

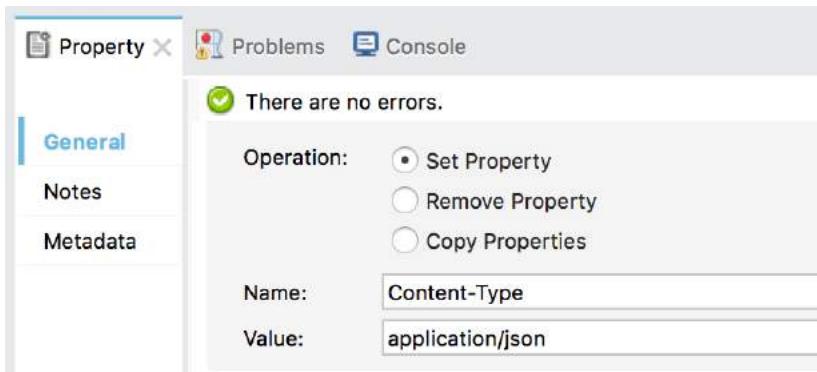
5. In interface.xml, locate the american-apiKitGlobalExceptionMapping Mapping exception strategy.



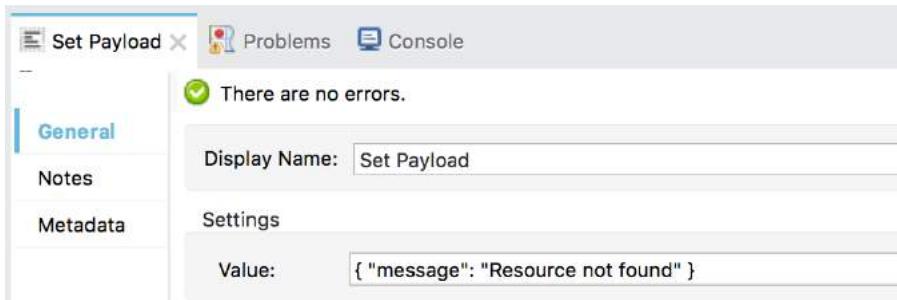
6. Double-click the 404 exception mapping and look at the settings in the 404 properties view.



7. Double-click the Property transformer in the 404 exception mapping and look at the settings in the Set Payload properties view.

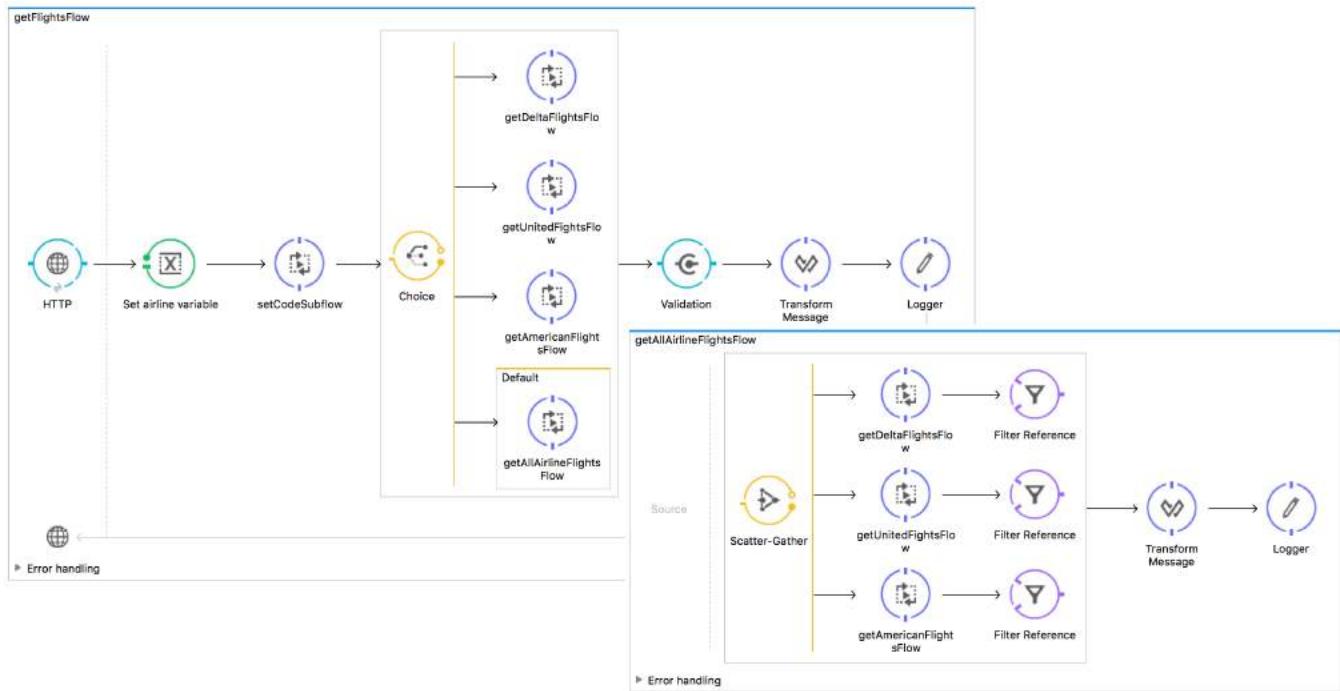


8. Double-click the Set Payload transformer in the 404 exception mapping and look at the settings in the Set Payload properties view.



9. Look at the values for the Set Payload transformers in the other exception mappings.
10. Close the project.

Module 9: Controlling Message Flow



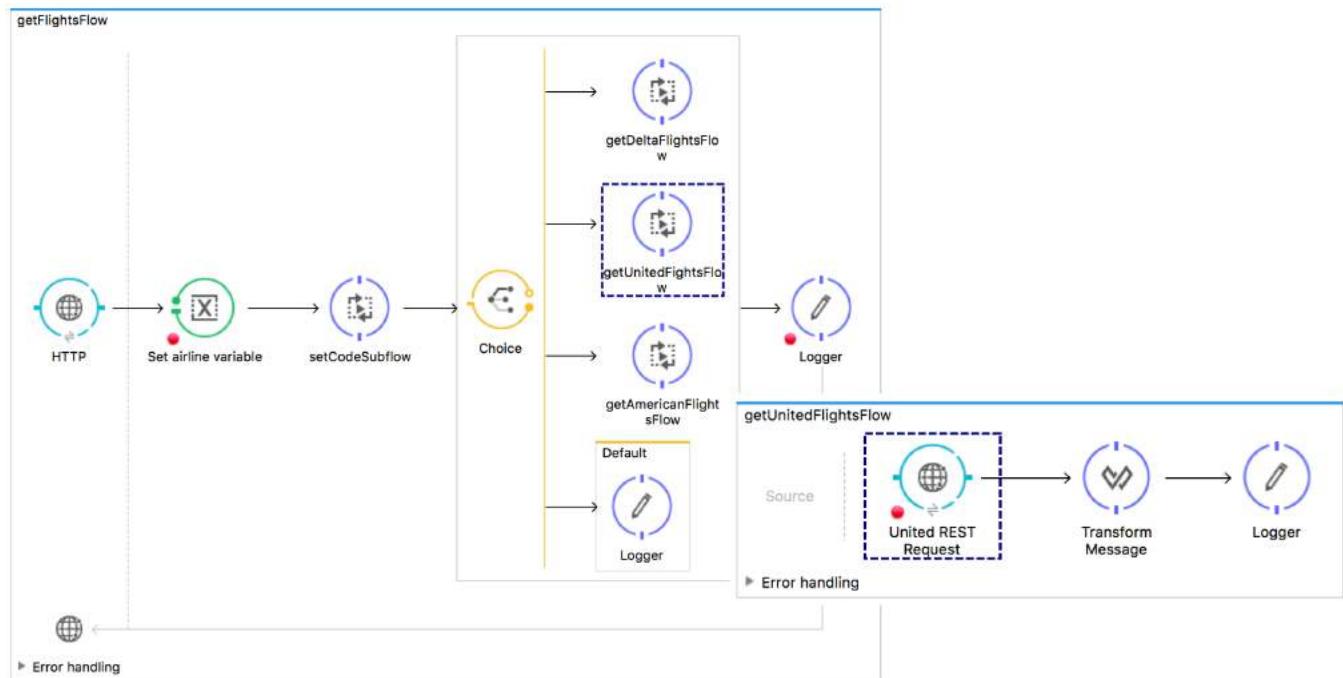
Objectives:

- Route messages based on conditions.
- Multicast messages.
- Filter messages.
- Validate messages.

Walkthrough 9-1: Route messages based on conditions

In this walkthrough, you modify gtAllFlightsFlow to route messages to either the United, Delta, or American flows based on the value of an airline query parameter. You will:

- Use a Choice router.
- Set the router paths.



Look at possible airline values specified in the API

1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Go to the API Sync view and open `api.raml`.
3. Locate the airline query parameter and its possible values.
4. Run the project.

Test the application

5. In Postman, make a request to `http://localhost:8081/flights?code=CLE`; you should see only Delta flights to CLE.
6. Add a query parameter called `airline` and set it equal to `united`.

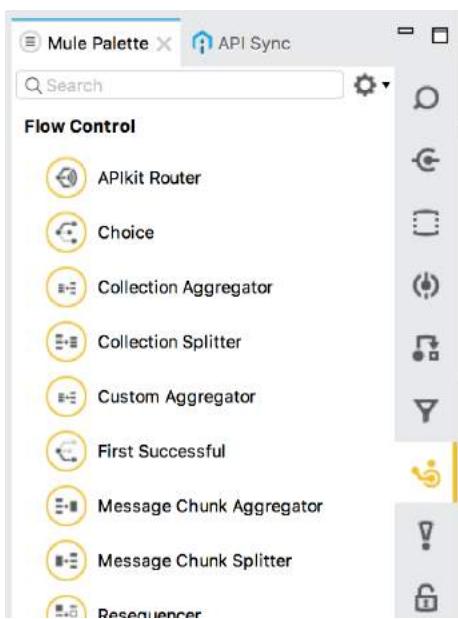
7. Send the request again; you should still only see the Delta flights.

The screenshot shows a REST client interface. At the top, there's a header bar with 'GET' selected, the URL 'localhost:8081/flights?code=CLE&airline=united', a 'Params' button, a 'Send' button, and a 'Save' button. Below the header are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests'. The 'Body' tab is active, showing a status of '200 OK' and a time of '185 ms'. The response body is displayed in a code editor-like area with syntax highlighting for JSON. The JSON object has the following structure:

```
12 {  
13   "airline": "Delta",  
14   "price": "308.0",  
15   "departureDate": "2015/07/11",  
16   "plane": "Boeing 777"
```

Browse the flow control elements in the Mule Palette

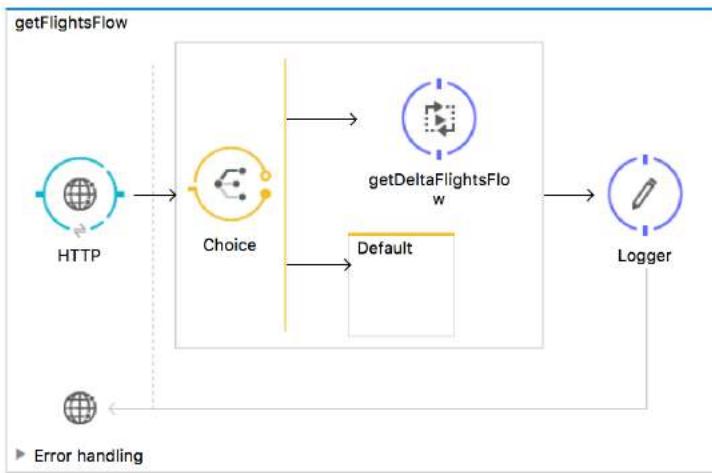
8. Return to Anypoint Studio.
9. In the Mule Palette, select the Flow Control tab.
10. View the available flow control processors.



Add a Choice router

11. Drag a Choice flow control element from the Mule Palette and drop it in getFlightsFlow before getDeltaFlightsFlow.

12. Drag the getDeltaFlightsFlow flow reference into the router.

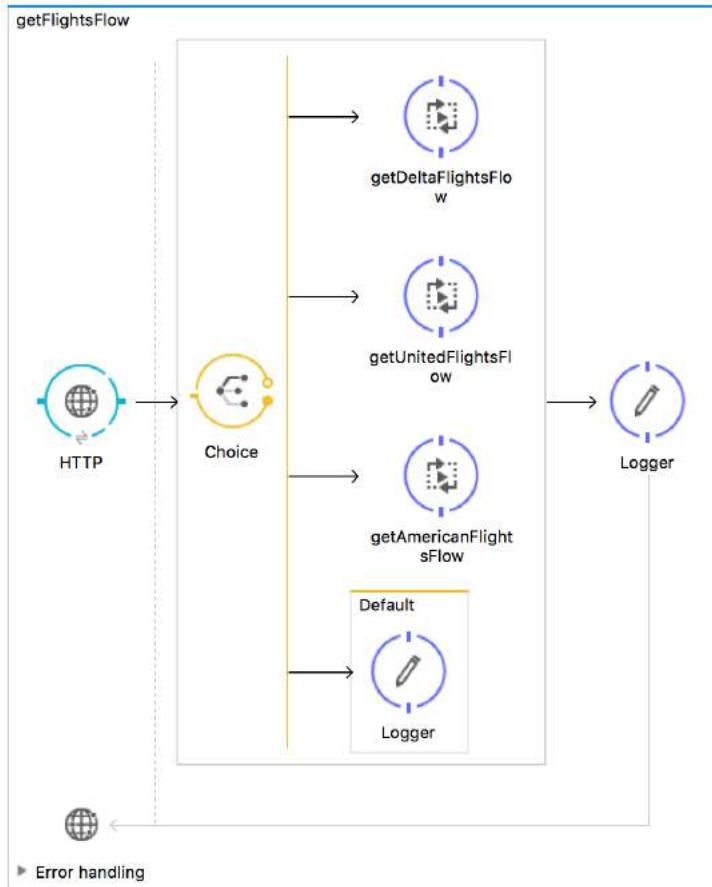


13. Add two additional Flow Reference components to the Choice router.

14. In the Properties view for the first flow reference, set the flow name to getUnitedFlightsFlow.

15. In the Properties view for the second flow reference, set the flow name to getAmericanFlightsFlow.

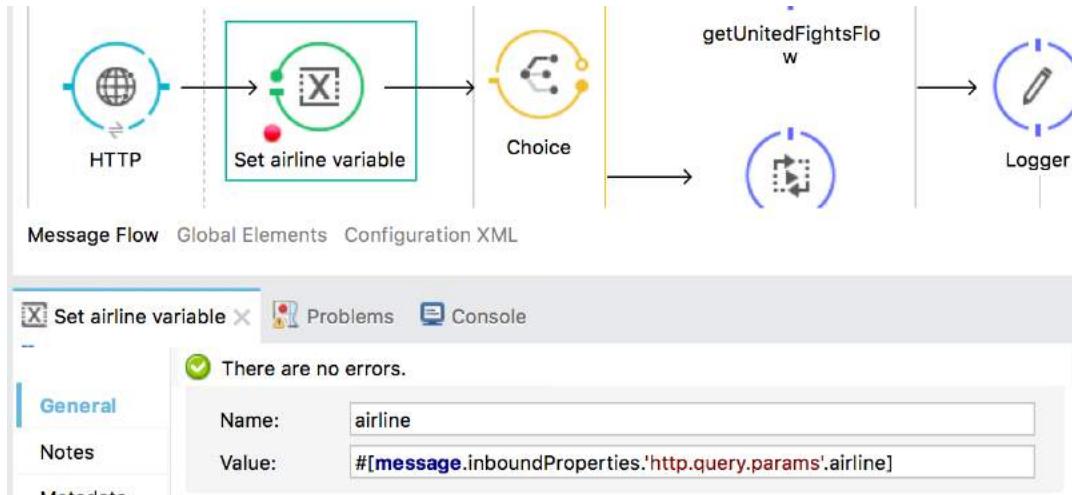
16. Drag a Logger component from the Mule Palette and drop it in the default branch.



Store the airline query parameter in a flow variable

17. Add a Variable transformer before the Choice router.
18. Change its display name to Set airline variable.
19. Set the flow variable name to airline and set it equal to a query parameter called airline.

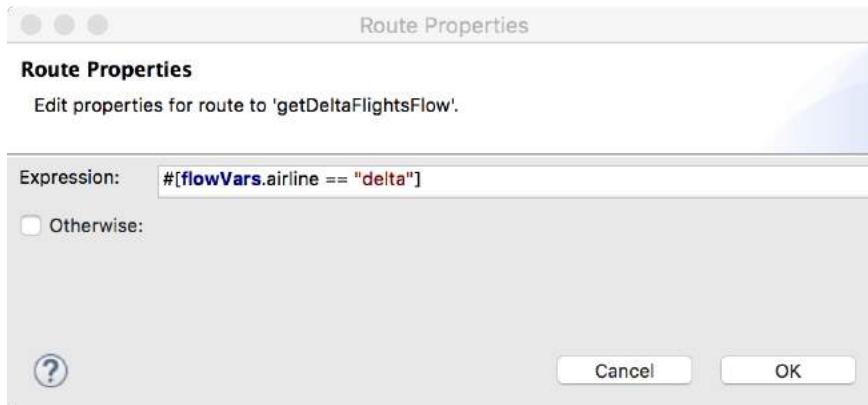
```
##[message.inboundProperties.'http.query.params'.airline]
```



Configure the Choice router

20. In the Choice properties view, double-click the getDeltaFlightsFlow route.
21. In the Route Properties dialog box, set the expression to true if the airline flow variable is equal to delta and click OK.

```
##[flowVars.airline == "delta"]
```



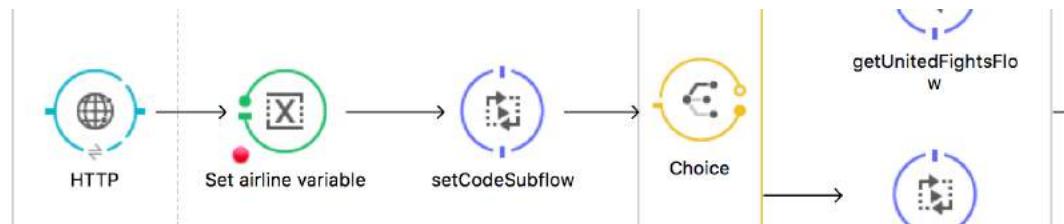
22. Set a similar expression for the United route, routing to it when flowVars.airline is equal to united.

23. Set a similar expression for the American route, routing to it when flowVars.airline is equal to american.

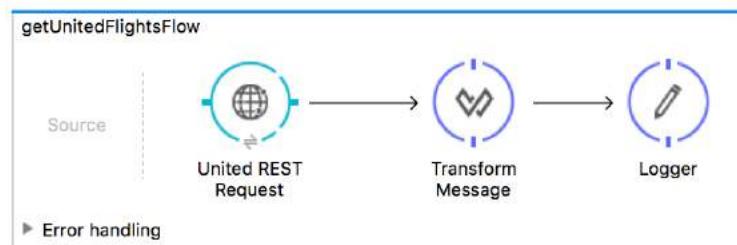
The screenshot shows the 'Choice' properties configuration in Anypoint Studio. The 'When' section contains three entries: '#[flowVars.airline == "delta"]', '#[flowVars.airline == "united"]', and '#[flowVars.airline == "american"]'). The 'Route Message to' section lists four options: 'getDeltaFlightsFlow', 'Logger', 'getUnitedFlightsFlow', and 'getAmericanFlightsFlow'. The 'getAmericanFlightsFlow' option is highlighted with a grey background.

Route all requests through the router

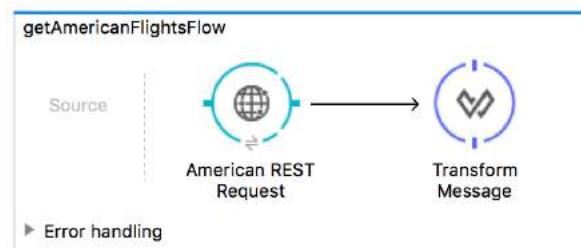
24. From getUnitedFlightsFlow, drag the setCodeSubflow flow reference into getFlightsFlow before the choice router.



25. In getUnitedFlightsFlow, delete the HTTP Listener endpoint.



26. In getAmericanFlightsFlow, delete the HTTP Listener endpoint and the setCodeSubflow flow reference.

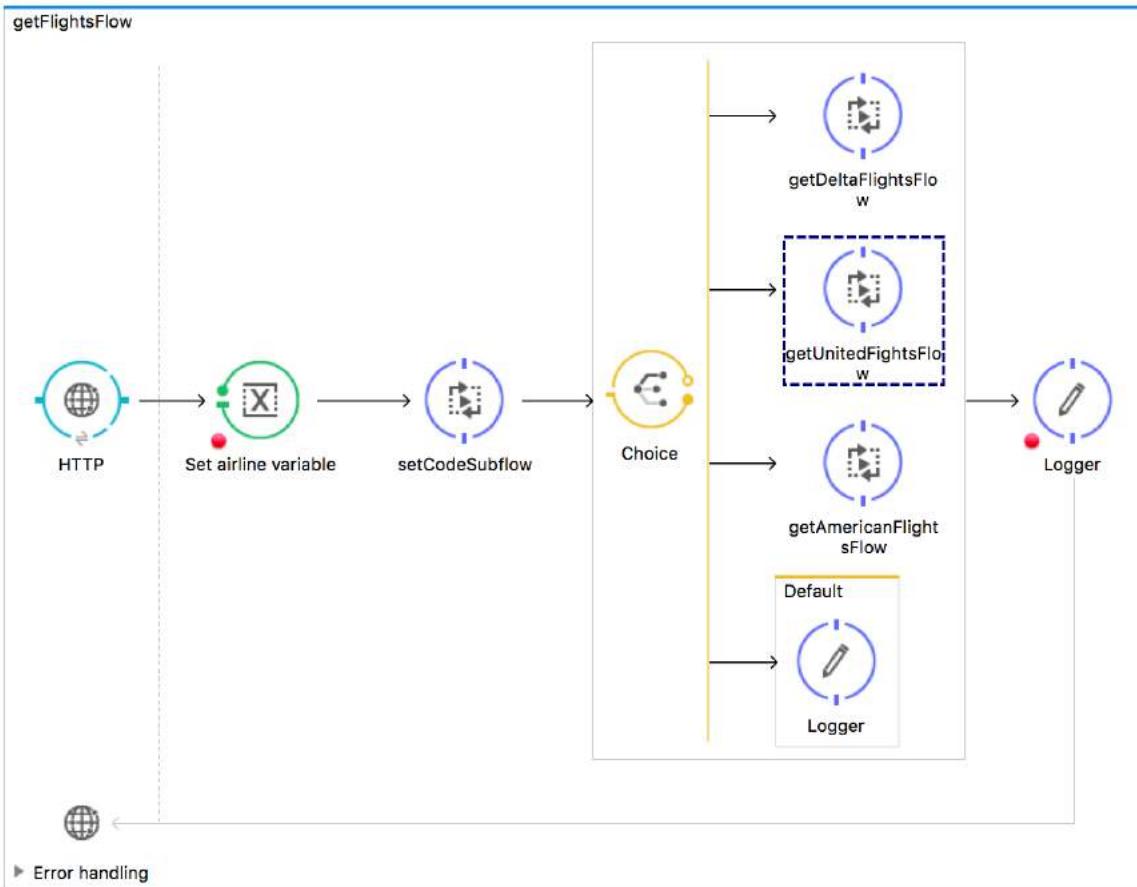


27. In getDeltaFlightsFlow, delete the HTTP Listener endpoint and the setCodeSubflow flow reference.

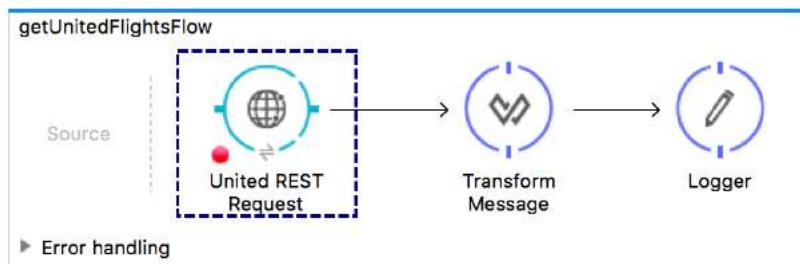


Test the application

28. In getFlightsFlow, make sure there is a breakpoint on one of the processors before the Choice router.
 29. Debug the project.
 30. In Postman, make the same request to <http://localhost:8081/flights?code=CLE&airline=united>.
 31. In the Mule Debugger, step through the application; you should see the Choice router pass the message to the United branch.



32. Step through the rest of the application; you should see the message passed to getUnitedFlightsFlow and then back to getFlightsFlow.



33. Return to Postman; you should see only United flights to CLE returned.

A screenshot of the Postman application interface. The top bar shows 'GET' selected, the URL 'localhost:8081/flights?code=CLE&airline=united', and various buttons like 'Params', 'Send', and 'Save'. Below the header, the 'Body' tab is active, showing JSON response data. The JSON structure is as follows:

```
2 *
3 {
4   "airline": "United",
5   "price": 845,
6   "departureDate": "2015/07/11",
7   "plane": "Boeing 727",
8   "origination": "MUA",
9   "code": "ER9fje",
10  "emptySeats": 32,
11  "destination": "CLE"
},
```

The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 38960 ms'.

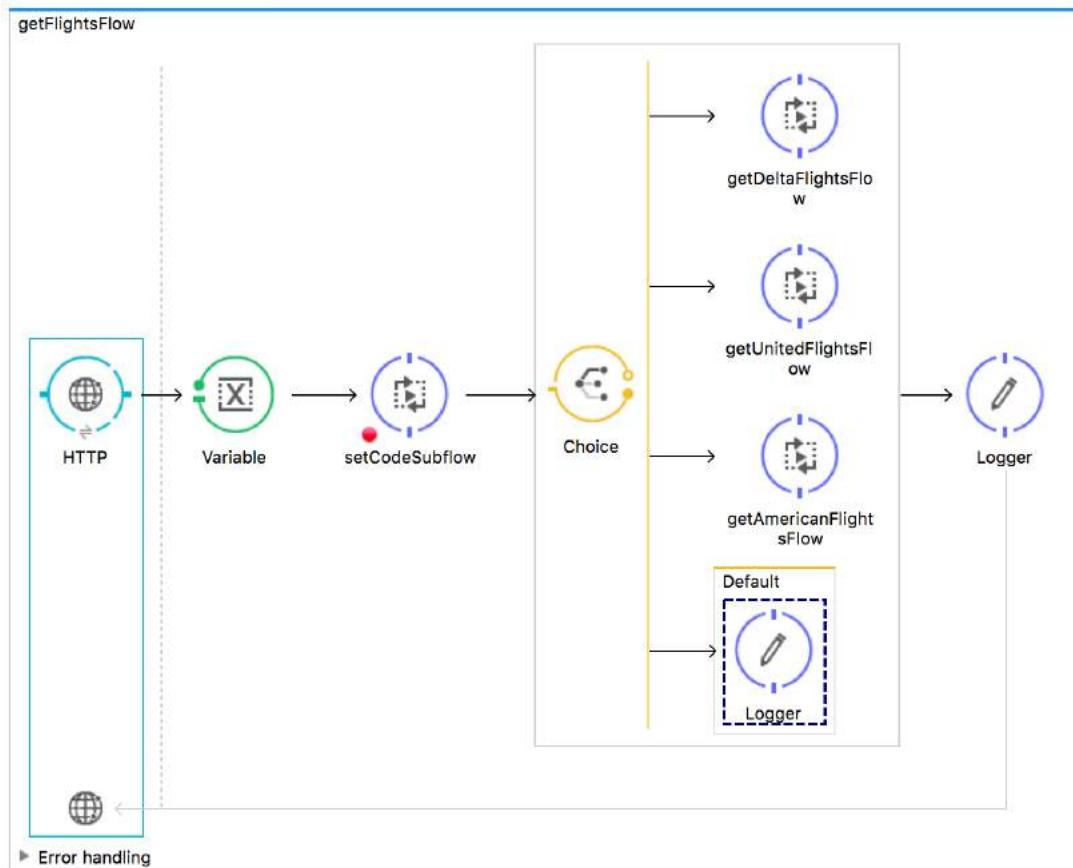
34. Change the airline to delta and the code to LAX.

35. Send the request.

36. Step through the application; the message should be routed to the Delta branch.

37. Return to Postman; you should see only Delta flights to LAX are returned.

38. In Postman, remove both parameters and make a request to <http://localhost:8081/flights>.
39. In the Mule Debugger, step through the application; you should see the Choice router pass the message to the default branch.



40. Click the Resume button.
41. Return to Postman; no flights are returned.

A screenshot of the Postman interface. The top bar shows "GET" selected, the URL "http://localhost:8081/flights", and a "Send" button. Below the bar, the "Body" tab is active, showing a single character "1" in a text area. The status bar indicates "Status: 200 OK" and "Time: 60431 ms".

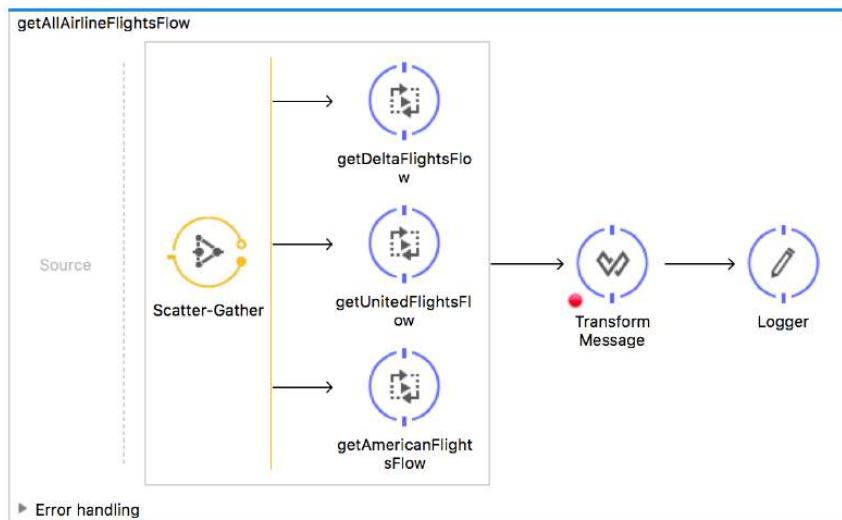
Note: If the next walkthrough, you will change this behavior so that if no airline is specified, flights for all airlines will be returned.

42. Return to Anypoint Studio and stop the project.

Walkthrough 9-2: Multicast a message

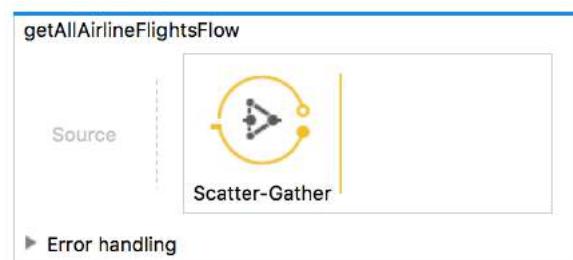
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.
- Use DataWeave to sort the flights by price and return them as JSON.
- (Optional) Modify the airline flows to use DataWeave to each return a Java collection of Flight objects.



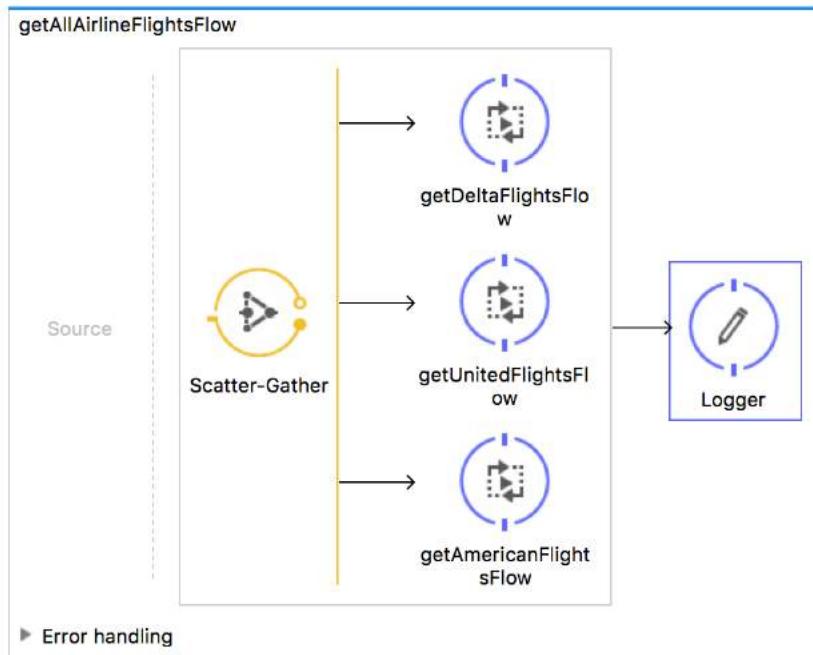
Create a flow to use a router to call all three airline services

1. Return to implementation.xml.
2. Drag a Scatter-Gather flow control element from the Mule Palette and drop it at the bottom of the canvas.
3. Change the name of the flow to getAllAirlineFlightsFlow.



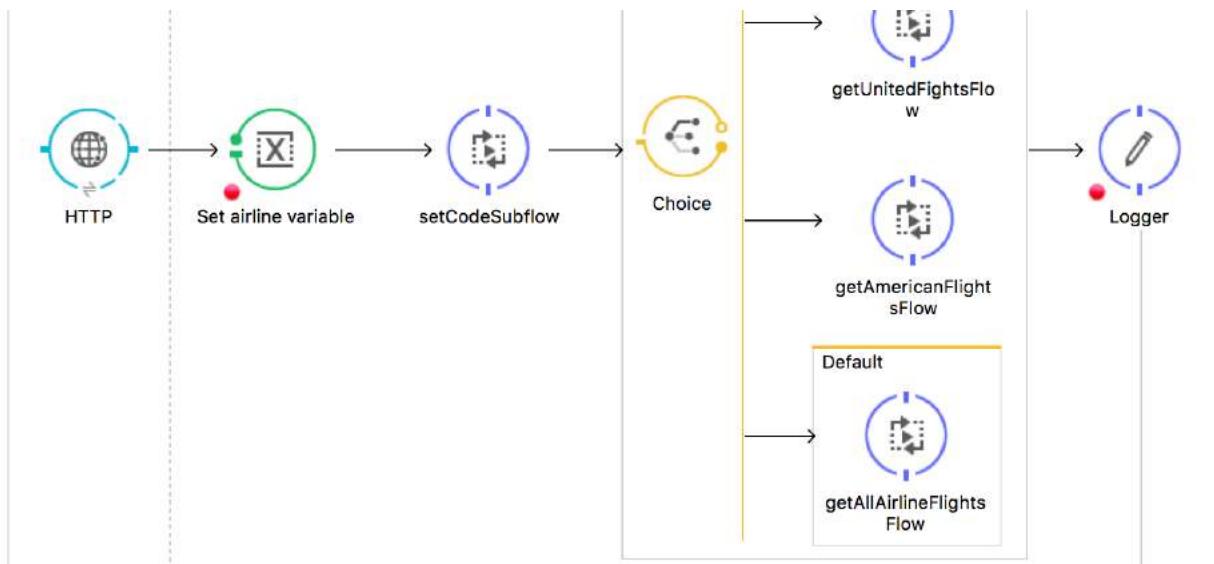
4. Drag three Flow Reference components into the Scatter-Gather router.
5. Using the Properties view, set the flow name of the first Flow Reference to getDeltaFlightsFlow.

6. Set the flow name of the second Flow Reference to getUnitedFlightsFlow.
7. Set the flow name of the third Flow Reference to getAmericanFlightsFlow.
8. Add a Logger after the router.



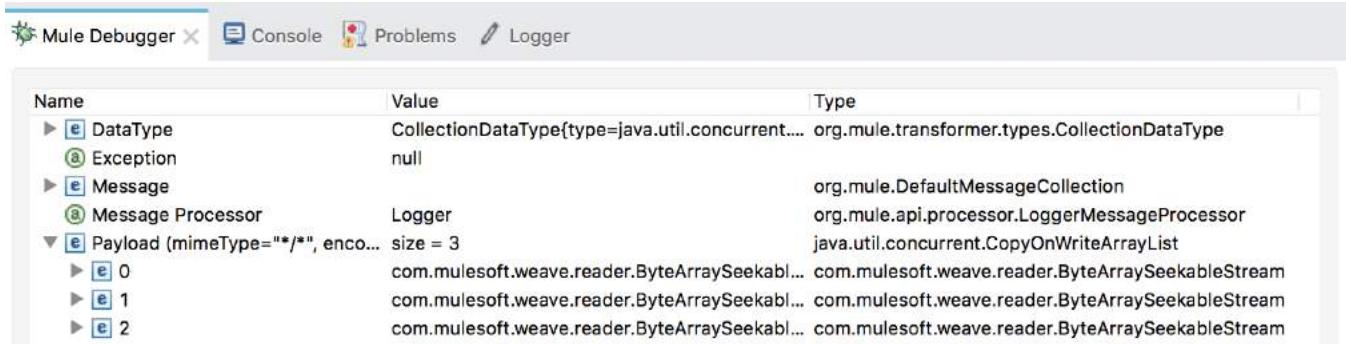
Call this flow if no airline is specified

9. Return to getFlightsFlow at the top of the canvas.
10. Delete the Logger in the default branch.
11. Add a Flow Reference to the default branch and set its flow name to getAllAirlineFlightsFlow.



Test the application

12. Debug the project.
13. In Postman, make the same request to <http://localhost/flights>.
14. In the Mule Debugger, step through the application to the default branch of the choice router, then into the Scatter-Gather, and then into each of the airline flows.
15. Stop at the Logger back in getFlightsFlow and look at the payload.



The screenshot shows the Mule Debugger interface with the 'Logger' tab selected. The payload is displayed in a table:

Name	Value	Type
» (DataType)	CollectionDataType{type=java.util.concurrent....	org.mule.transformer.types.CollectionDataType
» (Exception)	null	
» (Message)		org.mule.DefaultMessageCollection
» (Message Processor)	Logger	org.mule.api.processor.LoggerMessageProcessor
» (Payload (mimeType="*/*", enco...))	size = 3	java.util.concurrent.CopyOnWriteArrayList
» 0	com.mulesoft.weave.reader.ByteArraySeekabl...	com.mulesoft.weave.reader.ByteArraySeekableStream
» 1	com.mulesoft.weave.reader.ByteArraySeekabl...	com.mulesoft.weave.reader.ByteArraySeekableStream
» 2	com.mulesoft.weave.reader.ByteArraySeekabl...	com.mulesoft.weave.reader.ByteArraySeekableStream

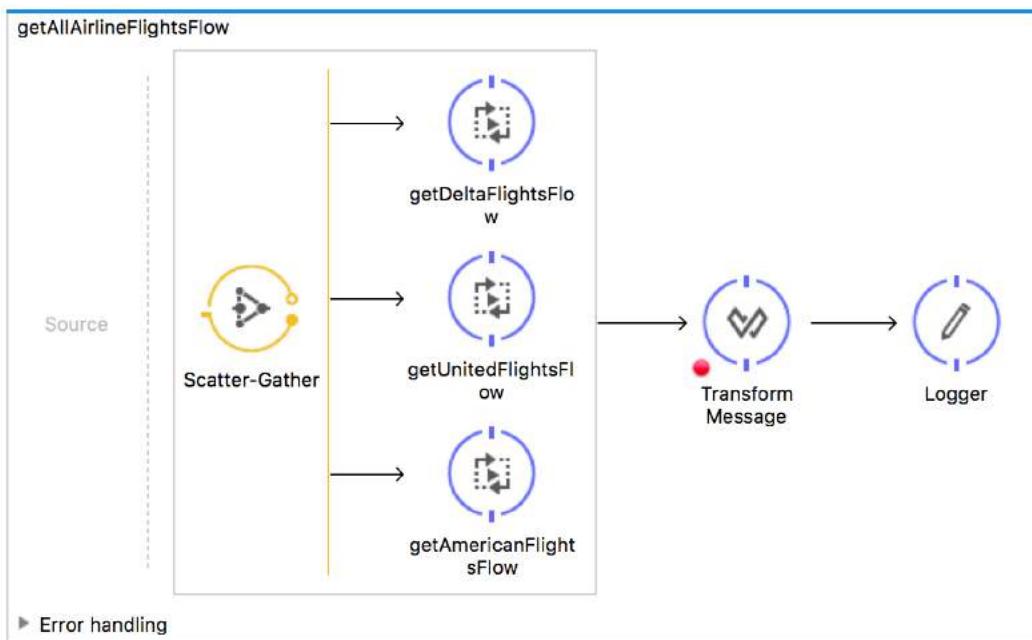
Note: You get three different DataWeave stream objects. You want to combine the three collections and then sort them by price and return them as JSON. Because the airline flight flows were written first and already return JSON, you can just flatten the collections into one and then order by price. More typically, however, you would have each of the airline flows return data of a common canonical format – in this case, a collection of Flight Java objects – and then flatten, order, and transform that data to the JSON specified by the API.

16. Click the Resume button.
17. Stop the project.

Flatten the combined results

18. Return to getAllAirlineFlightsFlow.

19. Add a Transform Message component before the Logger.



20. In the Properties view, set the DataWeave expression to flatten the payload.

`flatten payload`

```
1 %dw 1.0
2 %output application/java
3 ---
4 flatten payload
```

Note: You will learn about DataWeave operators in the later module Writing DataWeave expressions.

Test the application

21. Save the file to redeploy the application.

22. In Postman, make the same request to <http://localhost:8081/flights>.

23. In the Mule Debugger, step through the application to the Logger after the Choice router; you should see the payload is now one ArrayList of HashMaps.

```

▼ [E] Payload (mimeType="appli... size = 11          java.util.ArrayList
  ► [E] 0           {airline=Delta, flightCode=A1B2...  java.util.LinkedHashMap
  ► [E] 1           {airline=Delta, flightCode=A1BT...  java.util.LinkedHashMap
  ► [E] 10          {airline=American, flightCode=rr...  java.util.LinkedHashMap
  ► [E] 2           {airline=Delta, flightCode=A142...  java.util.LinkedHashMap
  ► [E] 3           {airline=United, flightCode=ER3...  java.util.LinkedHashMap
  ► [E] 4           {airline=United, flightCode=ER3...  java.util.LinkedHashMap
  ► [E] 5           {airline=United, flightCode=rr...  java.util.LinkedHashMap
  ► [E] 6           {airline=American, flightCode=rr...  java.util.LinkedHashMap
  ► [E] 7           {airline=American, flightCode=ee...  java.util.LinkedHashMap
  ► [E] 8           {airline=American, flightCode=ff...  java.util.LinkedHashMap
  ► [E] 9           {airline=American, flightCode=ee...  java.util.LinkedHashMap
    ► [E] 0           airline=American          java.util.LinkedHashMap$Entry
    ► [E] 1           flightCode=eefd3000       java.util.LinkedHashMap$Entry
    ► [E] 2           fromAirportCode=MUA        java.util.LinkedHashMap$Entry
    ► [E] 3           toAirportCode=SFO        java.util.LinkedHashMap$Entry
    ► [E] 4           departureDate=2016-02-01T00...  java.util.LinkedHashMap$Entry
    ► [E] 5           emptySeats=0            java.util.LinkedHashMap$Entry

```

24. Step through the rest of the application and stop the project.

25. In Postman, you should get a representation of Java objects returned.

The screenshot shows a Postman interface with a GET request to "localhost:8081/flights". The response status is 200 and the time taken is 97733 ms. The "Body" tab is selected, showing the response in "Pretty" format. The response content is a JSON representation of a Java object, specifically an ArrayList of HashMaps, which corresponds to the payload shown in the Mule Debugger. The JSON structure includes fields like airline, flightCode, fromAirportCode, toAirportCode, departureDate, and emptySeats.

```

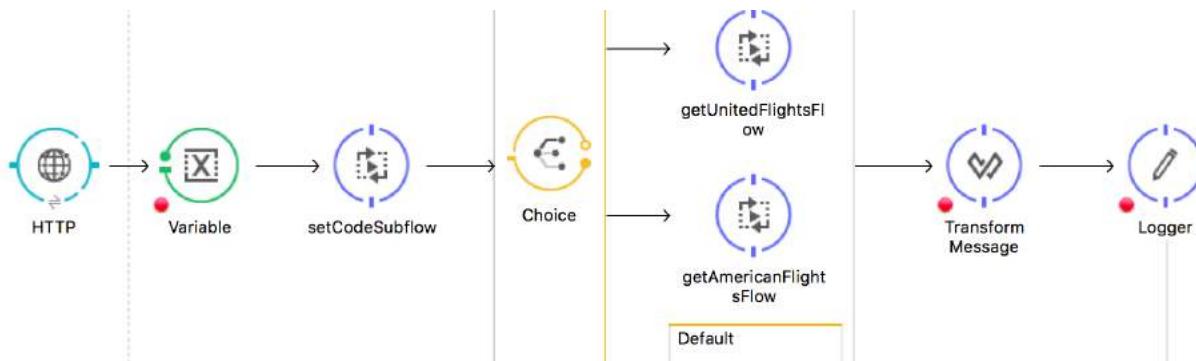
[{"id": 1, "airline": "Delta", "flightCode": "A1B2C3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 2, "airline": "Delta", "flightCode": "A1BT", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 10, "airline": "American", "flightCode": "rr", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 2, "airline": "Delta", "flightCode": "A142", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 3, "airline": "United", "flightCode": "ER3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 4, "airline": "United", "flightCode": "ER3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 5, "airline": "United", "flightCode": "rr", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 6, "airline": "American", "flightCode": "rr", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 7, "airline": "American", "flightCode": "ee", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 8, "airline": "American", "flightCode": "ff", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 9, "airline": "American", "flightCode": "ee", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}]

```

Return the data as JSON and sort by price

26. Return to getFlightsFlow.

27. Add a Transform Message component after the Choice router.



28. In the Transform Message properties view, change the output to application/json.

29. Change the transformation expression to order the payload by price.

```
Output Payload ▾ ⌂
```

```
1 %dw 1.0
2 %output application/json
3 ---
4 payload orderBy $.price
```

Note: You will learn about DataWeave operators in the later module Writing DataWeave expressions.

Test the application

30. Run the project.

31. In Postman, send the same request to <http://localhost:8081/flights>; you should get all the flights to SFO returned and they should be sorted by price.

The screenshot shows the Postman interface. At the top, there's a header bar with 'GET' selected, the URL 'localhost:8081/flights', and buttons for 'Params', 'Send', and 'Save'. Below the header, the 'Body' tab is active, showing a JSON response. The JSON is a list of flight objects, each with fields like airline, flightCode, fromAirportCode, toAirportCode, departureDate, emptySeats, totalSeats, price, and planeType. The flights are sorted by price, with Delta's flight being the most expensive at \$294.0 and American's flight being the least expensive at \$142. The JSON is displayed in a 'Pretty' format with line numbers on the left.

```
1+ [ ]
2+ {
3+   "airline": "American",
4+   "flightCode": "RREE1093",
5+   "fromAirportCode": "MUA",
6+   "toAirportCode": "SFO",
7+   "departureDate": "2016-02-11T00:00:00",
8+   "emptySeats": 1,
9+   "totalSeats": 150,
10+  "price": 142,
11+  "planeType": "Boeing 737"
12+ },
13+ {
14+   "airline": "Delta",
15+   "flightCode": "A14244",
16+   "fromAirportCode": "MUA",
17+   "toAirportCode": "SFO",
18+   "departureDate": "2015/02/12",
19+   "emptySeats": "10",
20+   "price": "294.0",
21+   "planeType": "Boeing 787"
```

32. Add an airline parameter equal to united and send the request:

<http://localhost:8081/flights?airline=united>; you should get united flights to SFO sorted by price.

33. Add a code parameter equal to PDF and send the request:

<http://localhost:8081/flights?airline=united&code=PDF>; you should get one united flight to PDF.

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

- Method: GET
- URL: localhost:8081/flights?airline=united&code=PDF
- Params: None
- Send and Save buttons
- Body tab selected, showing a JSON response:

```
1 [  
2 {  
3   "airline": "United",  
4   "flightCode": "ER95jF",  
5   "fromAirportCode": "MUA",  
6   "toAirportCode": "PDF",  
7   "departureDate": "2015/02/12",  
8   "emptySeats": 23,  
9   "price": 234,  
10  "planeType": "Boeing 787"  
11 }]  
12 ]
```
- Status: 200 Time: 249 ms

34. Change the airline to delta and send the request:

<http://localhost:8081/flights?airline=delta&code=PDF>; you should see the message that there are no flights to PDF – which is correct.

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

- Method: GET
- URL: localhost:8081/flights?airline=delta&code=PDF
- Params: None
- Send and Save buttons
- Body tab selected, showing an error message:

```
i 1 NO FLIGHTS to PDF  
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:  
3 payload orderBy $.price
```
- Status: 400 Time: 489 ms

35. Remove the airline parameter and send the request: <http://localhost:8081/flights?code=PDF>;

you should get the one United flight to PDF, but instead you get the message that there are no flights.

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

- Method: GET
- URL: localhost:8081/flights?code=PDF
- Params: None
- Send and Save buttons
- Body tab selected, showing an error message:

```
i 1 NO FLIGHTS to PDF  
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:  
3 payload orderBy $.price
```
- Status: 400 Time: 1409 ms

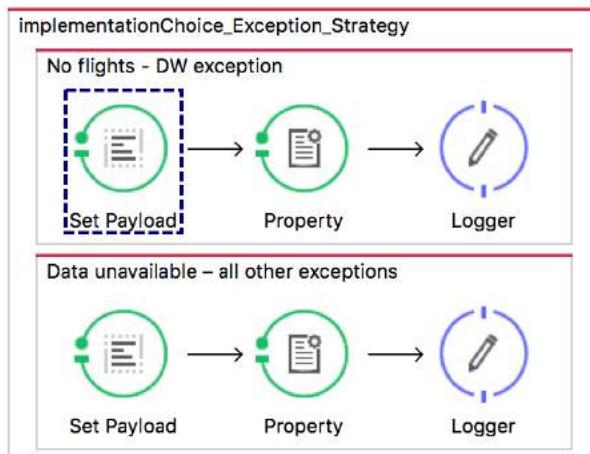
36. Return to Anypoint Studio and stop the project.

Debug the application

37. Debug the project.

38. In Postman, send the same request to <http://localhost:8081/flights?code=PDF>.

39. In the Mule Debugger, step through the application until an exception is thrown – and handled by the default global exception handler.



40. Continue to step through the application until you reach the Transform Message component in getAllAirlineFlightsFlow; you should see different types of objects returned.

The screenshot shows the Mule Debugger interface with the following components and states:

- Message Flow:** A sequence of components: Source → Scatter-Gather → getUnitedFlightsFlow → Transform Message → Logger.
- Global Elements:** Contains the 'getUnitedFlightsFlow' component.
- Configuration XML:** Contains the 'Transform Message' component.
- Mule Properties:** Shows the following table:

Name	Value	Type
(DataType)	CollectionDataType{type=java.util.List<com.mulesoft.weave.mule.W...	org.mule.transformer.types.Coll...
(Exception)	null	
(Message)		org.mule.DefaultMessageCollection
(Message Processor)	Transform Message	com.mulesoft.weave.mule.Weav...
(Payload (mimeType="*/*"))	size = 3 NO FLIGHTS to PDF com.mulesoft.weave.reader.Byte... com.mulesoft.weave.reader.Byte...	java.util.concurrent.CopyOnWrit...
(0)	NO FLIGHTS to PDF	java.lang.String
(1)	com.mulesoft.weave.reader.Byte...	com.mulesoft.weave.reader.Byt...
(2)	com.mulesoft.weave.reader.Byte...	com.mulesoft.weave.reader.Byt...

41. Step to the Transform Message component in getFlightsFlow; you should see the payload still contains the error message in addition to the valid flight results to PDF.

The screenshot shows the Mule Studio interface. At the top, there is a message flow diagram with components: setCodeSubflow, Choice, getUnitedFlightsFlow, Transform Message (highlighted with a dashed blue box), and Logger. Below the diagram are tabs for 'Message Flow', 'Global Elements', and 'Configuration XML'. The 'Mule Properties' tab is selected in the bottom navigation bar. The Mule Debugger panel displays the following table:

Name	Value	Type
(DataType)	CollectionDataType{type=java.util.ArrayList, org.mule.transformer.types.Collecti...	
(Exception)	null	
(Message)		org.mule.DefaultMuleMessage
(Message Processor)	Transform Message	com.mulesoft.weave.mule.WeaveMe...
(Payload (mimeType="application/pdf"))	size = 2 0: NO FLIGHTS to PDF 1: {airline=United, flightCode=ER95jf, f...}	java.util.ArrayList java.lang.String java.util.LinkedHashMap

42. Step to the end of the application; you should not get the United results to PDF.

The screenshot shows a Postman request configuration and a response preview. The request is a GET to 'localhost:8081/flights?code=PDF'. The response status is 400, with a time of 203856 ms. The body of the response is:

```
i 1 NO FLIGHTS to PDF
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload orderBy $.price
```

Note: You could write a custom aggregation strategy for the Scatter-Gather router with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages in the next walkthrough.

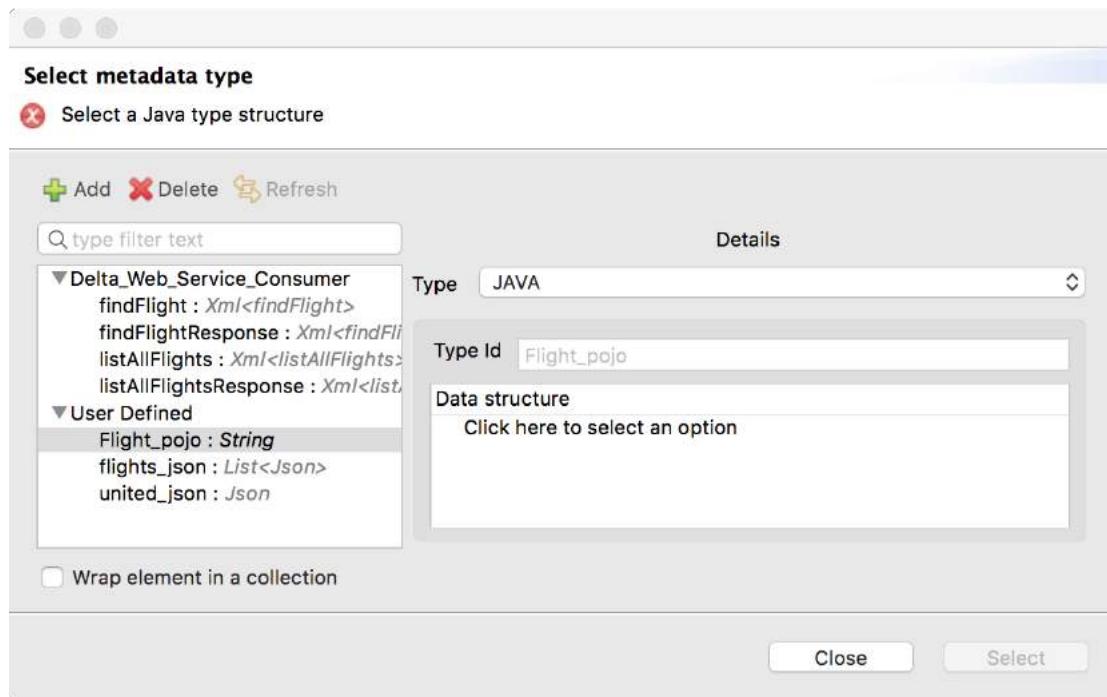
43. Return to Anypoint Studio and stop the project.

(Optional) Modify the airline flows to return Java Flight objects instead of JSON

Note: The rest of the walkthrough is optional. It contains steps to modify each of the airline flows to return data of a common canonical format – in this case, a collection of Flight Java objects.

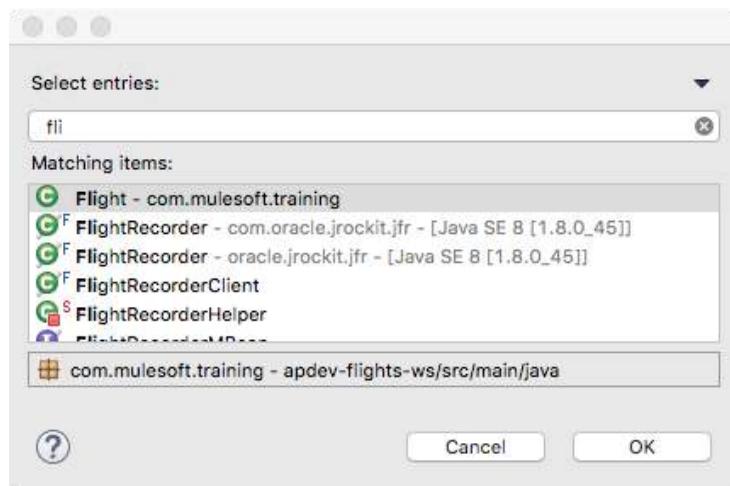
Change the United airline flows to return Java Flight objects instead of JSON

44. Return to getUnitedFlightsFlow in implementation.xml.
45. In the Transform Message properties view, right-click List <Json> in the output section and select Clear Metadata.
46. Click the Define metadata link.
47. In the Select metadata type dialog box, click the Add button.
48. In the Create new type dialog box, set the name to Flight_pojo and click Create type.
49. In the Select metadata type dialog box, change the type to JAVA.
50. In the Data structure section, click the Click here to select an option link.



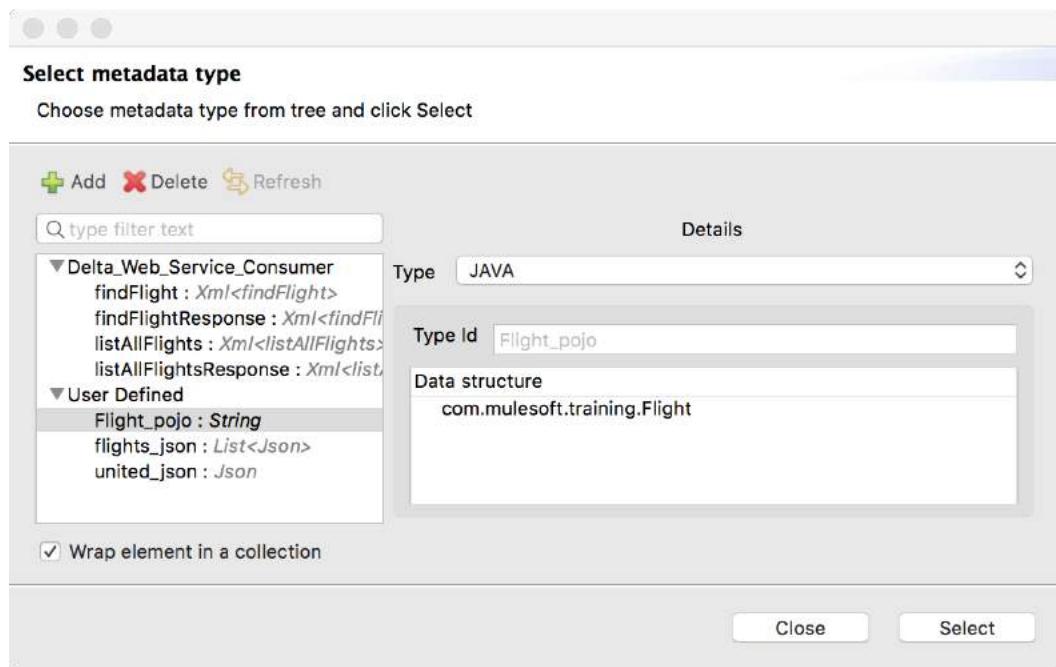
51. In the drop-down menu that appears, select Java object.

52. In the dialog box that opens, type fli and then in the list of classes that appears, select Flight – com.mulesoft.training.com.



53. Click OK.

54. In the Select metadata type dialog box, select Wrap element in a collection in the lower-left corner.



55. Click Select.

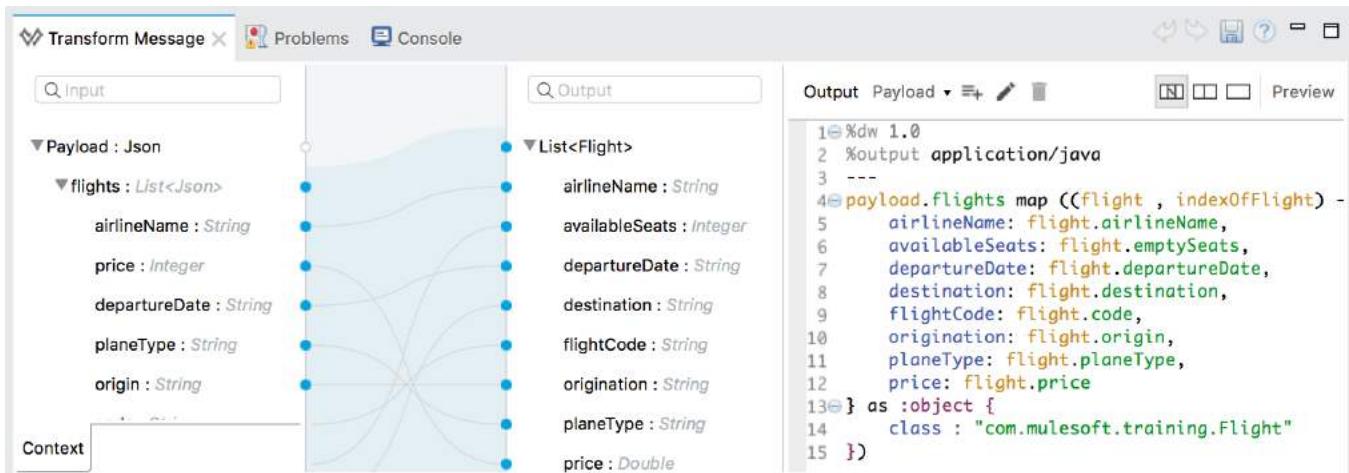
56. In the Transform Message properties view, replace the current transformation expression with an empty object.

```

Output Payload ▾ + 🖍️ 🗑️ Preview
1 %dw 1.0
2 %output application/java
3 ---
4 {}

```

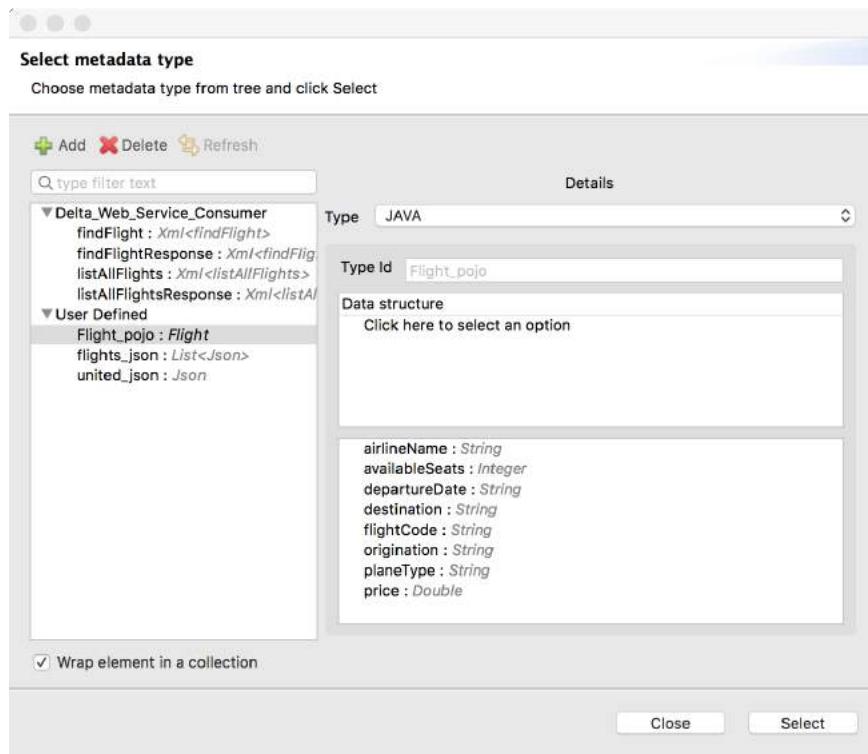
57. Use the graphical editor to map the fields appropriately.



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

58. Return to getAmericanFlightsFlow.
59. In the Transform Message properties view, right-click List <Json> in the output section and select Clear Metadata.
60. Click the Define metadata link.
61. In the Select metadata type dialog box, select Flight_pojo.

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



63. Click Select.

64. In the Transform Message properties view, replace the current transformation expression with an empty object.

65. Use the graphical editor to map the fields appropriately.

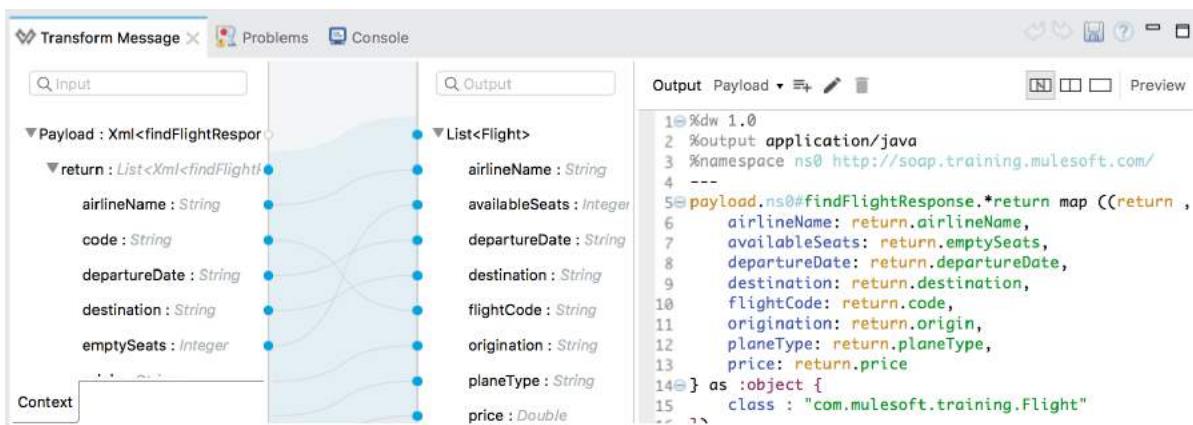
66. In the output section of the graphical editor, double-click airlineName; the field should be added to the DataWeave expression.

67. Set its value to "American".

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map ((payload01 , indexOfPayload01) -> {
5     airlineName: "American",
6     availableSeats: payload01.emptySeats,
7     departureDate: payload01.departureDate,
8     destination: payload01.destination,
9     flightCode: payload01.code,
10    origination: payload01.origin,
11    planeType: payload01.plane.type,
12    price: payload01.price
13} ) as :object {
14     class : "com.mulesoft.training.Flight"
15 })
```

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

68. Return to getDeltaFlightsFlow.
69. In the Transform Message properties view for the component after the Delta SOAP Request, right-click List <Json> in the output section and select Clear Metadata.
70. Click the Define metadata link.
71. In the Select metadata type dialog box, select Flight_pojo.
72. Select the Wrap element in a collection checkbox in the lower-left corner.
73. Click Select.
74. In the Transform Message properties view, replace the current transformation expression with an empty object.
75. Use the graphical editor to map the fields appropriately.



Test the application

76. Debug the project.
77. In Postman, remove the parameters and send a request to <http://localhost:8081/flights>.
78. In the Mule Debugger, step through the application to the Transform Message component in getAllAirlineFilghtsFlow; the payload should be a collection of three ArrayLists of Flight objects.

▼ [E] Payload (mimeType="*/*", encoding="UT... size = 3		java.util.concurrent.CopyOnWriteArrayList
► [E] 0	[com.mulesoft.training.Flight@24222585, com....	java.util.ArrayList
► [E] 1	[com.mulesoft.training.Flight@5979a9c0, com....	java.util.ArrayList
▼ [E] 2	[com.mulesoft.training.Flight@736e92d2, com....	java.util.ArrayList
► [E] 0	com.mulesoft.training.Flight@736e92d2	com.mulesoft.training.Flight
► [E] 1	com.mulesoft.training.Flight@45cf35b4	com.mulesoft.training.Flight
► [E] 2	com.mulesoft.training.Flight@37b9ada5	com.mulesoft.training.Flight
► [E] 3	com.mulesoft.training.Flight@4174ecb8	com.mulesoft.training.Flight
▼ [E] 4	com.mulesoft.training.Flight@468b0e46	com.mulesoft.training.Flight
⑧ airlineName	American	java.lang.String
⑧ availableSeats	100	java.lang.Integer
⑧ departureDate	2016-01-20T00:00:00	java.lang.String
⑧ destination	SFO	java.lang.String
⑧ flightCode	rree4567	java.lang.String
⑧ origination	MUA	java.lang.String
⑧ planeType	Boeing 737	java.lang.String
⑧ price	456.0	java.lang.Double

79. Step to the Logger; the payload should now be one ArrayList of Flight objects.

Payload (mimeType="application/json")		
size = 11		java.util.ArrayList
0	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
1	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
10	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
2	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
3	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
4	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
5	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
6	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
7	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
8	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
9	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
airlineName	American	java.lang.String
availableSeats	0	java.lang.Integer
departureDate	2016-02-01T00:00:00	java.lang.String

80. Step through to the end of the application.

Test the application

Note: You completed the rest of the walkthrough steps already before modifying the airline flows to return collections of Flight objects. You are repeating them as a lead in to the next walkthrough.

81. Run the project.

82. In Postman, send the same request to <http://localhost:8081/flights>; you should get all the flights to SFO returned and they should be sorted by price.

The screenshot shows the Postman interface with a GET request to 'localhost:8081/flights'. The response status is 200 and the time taken is 1810 ms. The response body is displayed in JSON format:

```
1 [ { 2   "flightCode": "rree1093", 3   "availableSeats": 1, 4   "destination": "SFO", 5   "planeType": "Boeing 737", 6   "origination": "MUA", 7   "price": 142, 8   "departureDate": "2016-02-11T00:00:00", 9   "airlineName": "American" 10 }, 11 { 12   "flightCode": "A14244", 13   "availableSeats": 10, 14   "destination": "SFO", 15   "planeType": "Boing 787", 16   "origination": "MUA", 17   "price": 294, 18   "departureDate": "2015/02/12" 19 }
```

83. Add an airline parameter equal to united and send the request:

<http://localhost:8081/flights?airline=united>; you should get united flights to SFO sorted by price.

84. Add a code parameter equal to PDF and send the request:

<http://localhost:8081/flights?airline=united&code=PDF>; you should get one united flight to PDF.

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

- Method: GET
- URL: localhost:8081/flights?airline=united&code=PDF
- Params: None
- Send and Save buttons
- Body tab selected, showing a JSON response:

```
1 {
2   "flightCode": "ER95jf",
3   "availableSeats": 23,
4   "destination": "PDF",
5   "planeType": "Boeing 787",
6   "origination": "MUA",
7   "price": 234,
8   "departureDate": "2015/02/12",
9   "airlineName": "United"
10 }
11 ]
12 ]
```
- Status: 200 Time: 277 ms

85. Change the airline to delta and send the request:

<http://localhost:8081/flights?airline=delta&code=PDF>; you should see the message that there are no flights to PDF – which is correct.

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

- Method: GET
- URL: localhost:8081/flights?airline=delta&code=PDF
- Params: None
- Send and Save buttons
- Body tab selected, showing an error message:

```
i 1 NO FLIGHTS to PDF
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload orderBy $.price
```
- Status: 400 Time: 363 ms

86. Remove the airline parameter and send the request: <http://localhost:8081/flights?code=PDF>; you should get the one United flight to PDF, but instead you get the message that there are no flights.

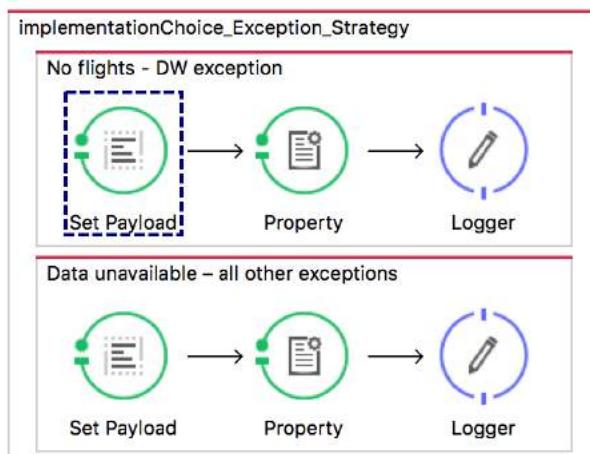
The screenshot shows a Postman request for `localhost:8081/flights?code=PDF`. The response status is 400, and the body contains the following stack trace:

```
i 1 NO FLIGHTS to PDF
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload orderBy $.price
```

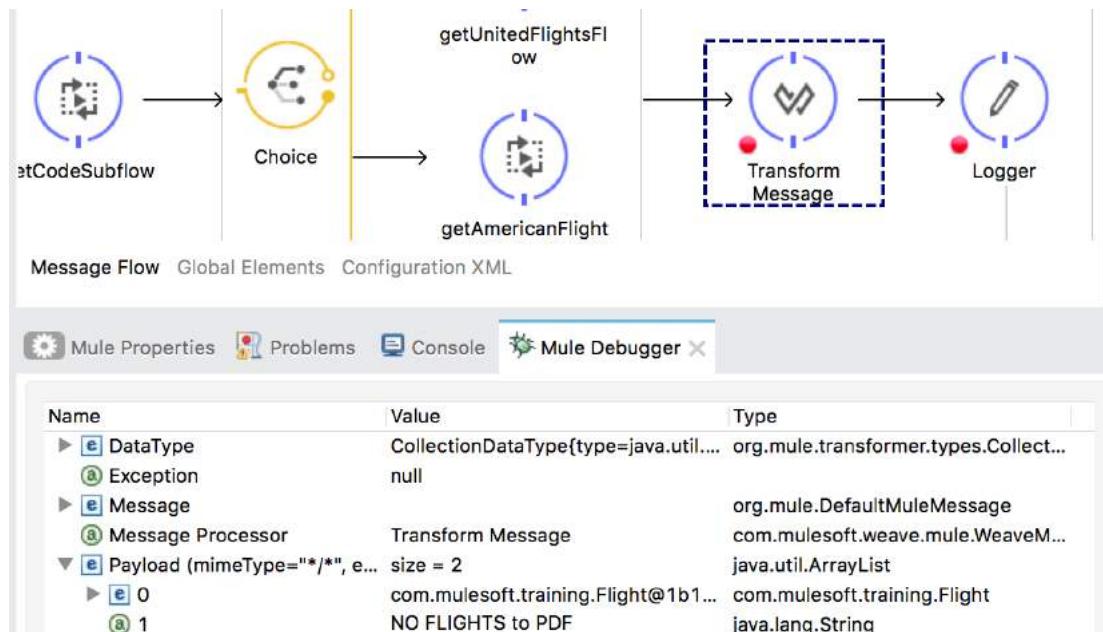
87. Return to Anypoint Studio and stop the project.

Debug the application

88. Debug the project.
89. In Postman, send the same request to <http://localhost:8081/flights?code=PDF>.
90. In the Mule Debugger, step through the application until an exception is thrown – and handled by the default global exception handler.



91. Continue to step through the application until you reach the Transform Message component in getFlightsFlow; you should see the payload still contains the error message in addition to the valid flight results to PDF.



92. Step to the end of the application; you should not get the United results to PDF.

GET localhost:8081/flights?code=PDF Params Send Save

Body Cookies Headers (3) Tests Status: 400 Time: 103759 ms

Pretty Raw Preview HTML

```
i 1 NO FLIGHTS to PDF
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload orderBy $.price
```

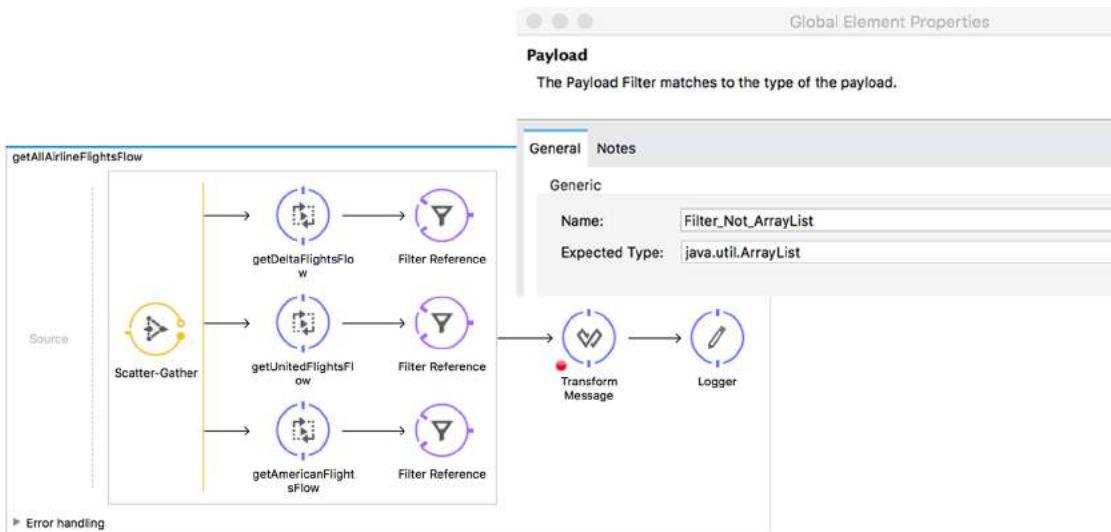
Note: You could write a custom aggregation strategy for the Scatter-Gather router with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages in the next walkthrough.

93. Return to Anypoint Studio and stop the project.

Walkthrough 9-3: Filter messages

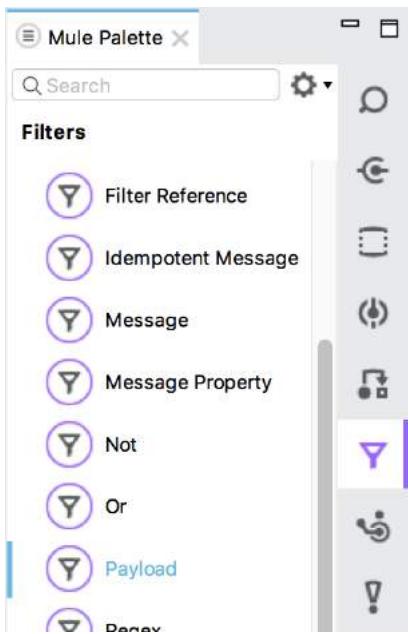
In this walkthrough, you filter the results in the multicast to ensure they are ArrayLists and not exception strings. You will:

- Use the Payload filter.
- Create and use a global filter.



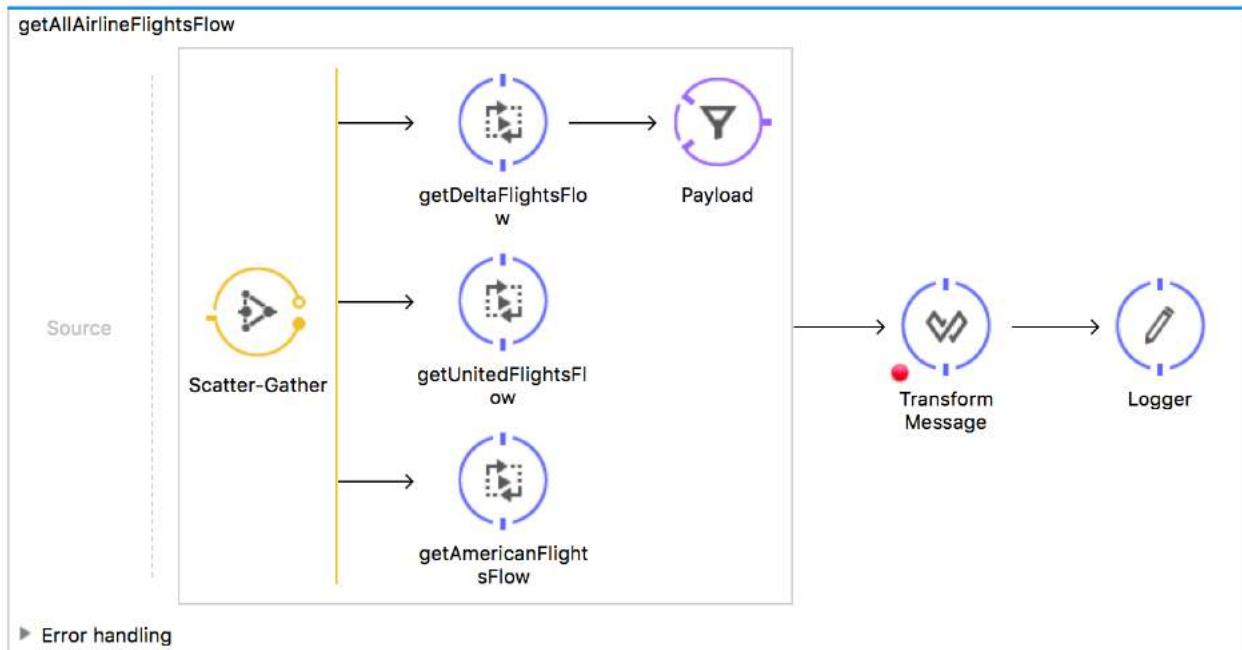
Browse the filter elements in the Mule Palette

1. In the Mule Palette, select the Filters tab.
2. View the available filters.

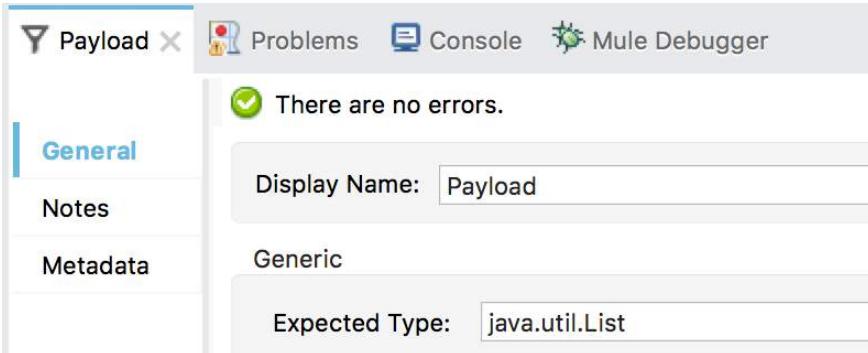


Add a filter

3. Return to getAAirlineFlightsFlow.
4. Drag out a Payload filter from the Mule Palette and drop it after the getDeltaFlightsFlow reference in the Scatter-Gather.



5. In the Payload properties view, set the expected type to java.util.ArrayList.



Test the application

6. Debug the project.
7. In Postman, make the same request to PDF and all airlines.

8. In the Mule Debugger, step to the Transform Message component after the scatter-gather; this time you should see the payload only contains the ArrayLists and not the error string.

Name	Value	Type
Payload (mimeType="/*",...	size = 2	java.util.concurrent.CopyOnWrite...
e 0	[com.mulesoft.training.Flight@ee...	java.util.ArrayList
e 0	com.mulesoft.training.Flight@ee...	com.mulesoft.training.Flight
e 1	[]	java.util.ArrayList

9. Step to the end of the application.

10. In Postman, view the response; you should see the United results to PDF.

```

1  [
2   {
3     "flightCode": "ER95jf",
4     "availableSeats": 23,
5     "destination": "PDF",
6     "planeType": "Boeing 787",
7     "origination": "MUA",
8     "price": 234,
9     "departureDate": "2015/02/12",
10    "airlineName": "United"
11  }
12 ]

```

11. Change the code parameter to FOO and send the request.

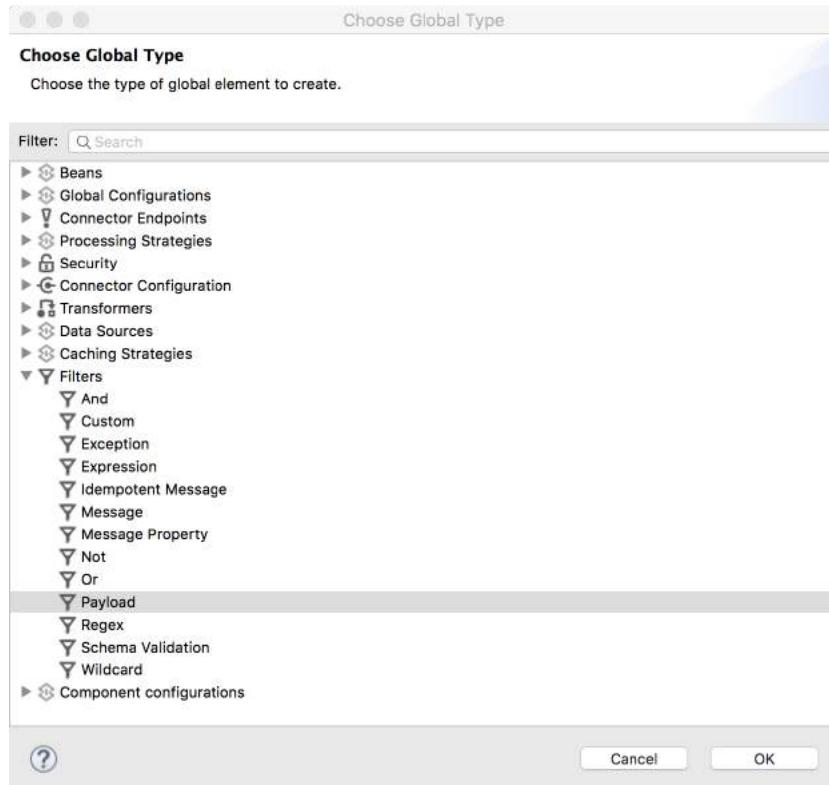
12. Click Resume until you step through to the end of the application.

13. In Postman, view the response; you should see an array with the error message.

The screenshot shows a Postman request for 'localhost:8081/flights?code=FOO'. The response status is 200, and the time taken is 8838 ms. The JSON body contains an error message: "NO FLIGHTS to FOO\ncom.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing: \npayload.flights map ((flight , indexOffFlight) -> {\n^\\nType mismatch for 'Value Selector' operator\n found :binary, :name\n required :datetime, :name or\n required :localdatetime, :name or\n required :object, :name or\n required :time, :name or\n required :array, :name or\n required :date, :name or\n required :localtime, :name or\n required :period,\n :name"}".

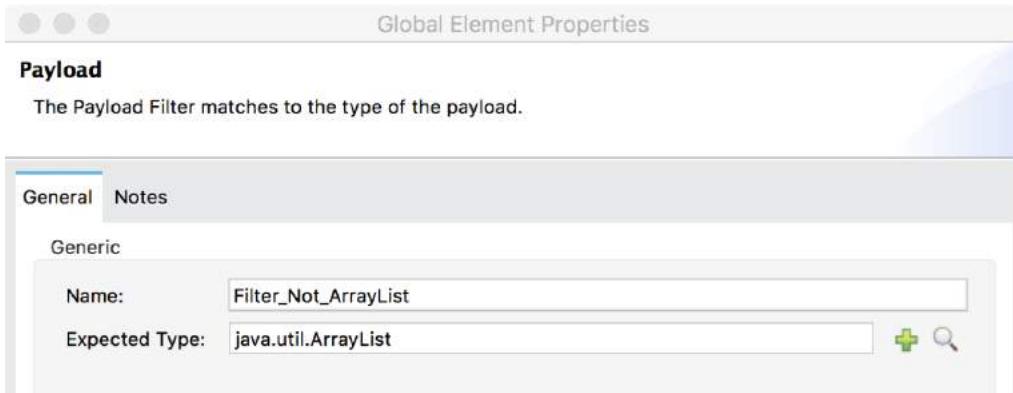
Create a global filter

14. Return to getAllAirlineFlightsFlow.
15. Delete the Payload filter.
16. Return to global.xml.
17. Switch to the Global Elements view and click Create.
18. In the Choose Global Type dialog box, select Filters > Payload and click OK.



19. In the Global Elements dialog box, set the name to Filter_Not_ArrayList.

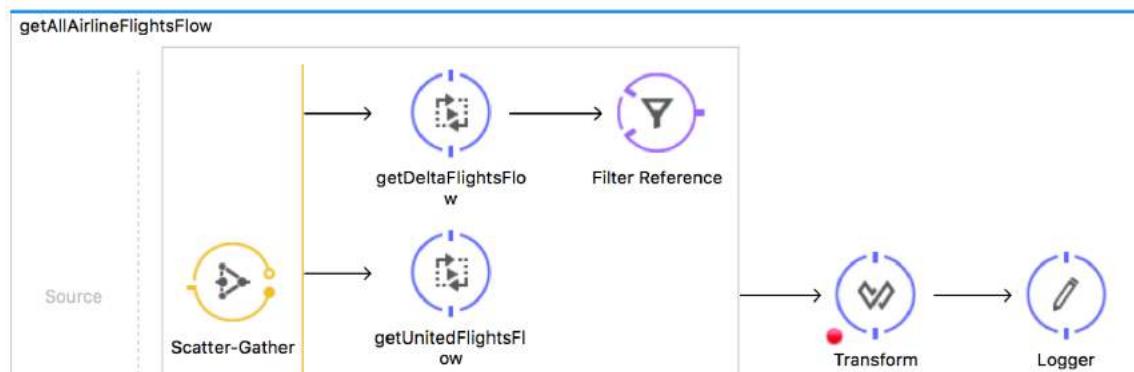
20. Set the expected type to java.util.ArrayList and click OK.



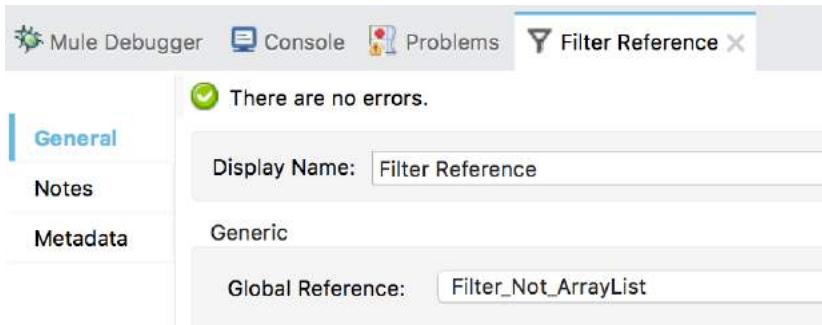
Use the global filter

21. Return to getAllAirlineFlightsFlow in implementation.xml.

22. Drag a Filter Reference from the Mule Palette and drop it after getDeltaFlightsFlow in the Scatter-Gather.



23. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.

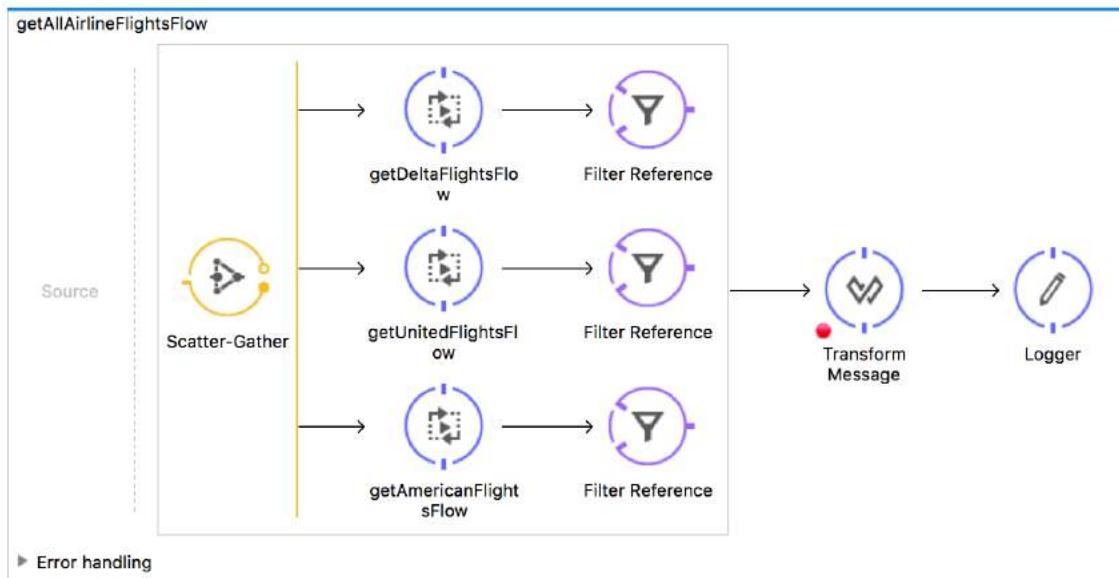


24. Add a Filter Reference after getUnitedFlightsFlow.

25. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.

26. Add a Filter Reference after getAmericanFlightsFlow.

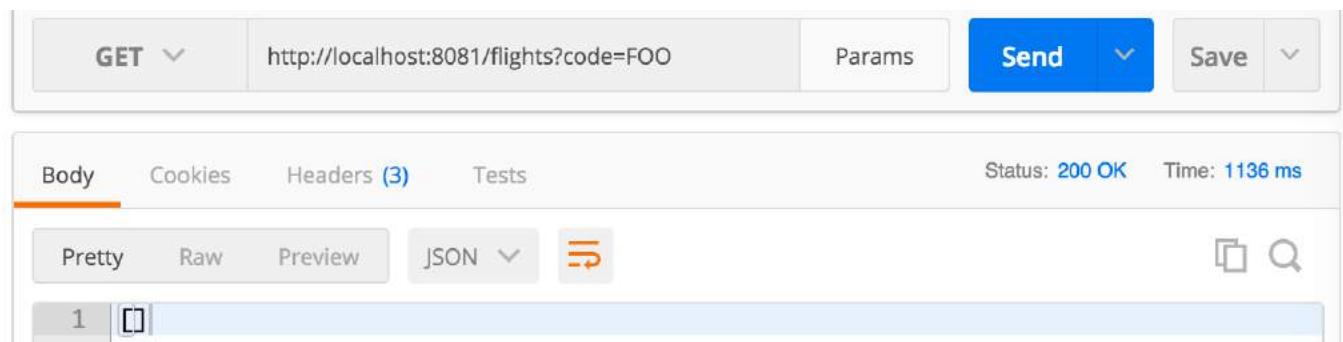
27. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.



Test the application

28. Run the project.

29. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>; you should now get no results and an empty array as a result.



A screenshot of the Postman application interface. At the top, there is a header with "GET" selected, the URL "http://localhost:8081/flights?code=FOO", and buttons for "Params", "Send", and "Save". Below the header, the "Body" tab is active, showing a JSON response with one item. The response body is an empty array: []. To the right of the response, status information is displayed: "Status: 200 OK" and "Time: 1136 ms". Other tabs like "Cookies", "Headers (3)", and "Tests" are visible but inactive. Below the tabs, there are buttons for "Pretty", "Raw", "Preview", and "JSON". On the far right, there are icons for "Copy" and "Search".

30. Return to Anypoint Studio and stop the project.

Walkthrough 9-4: Validate messages

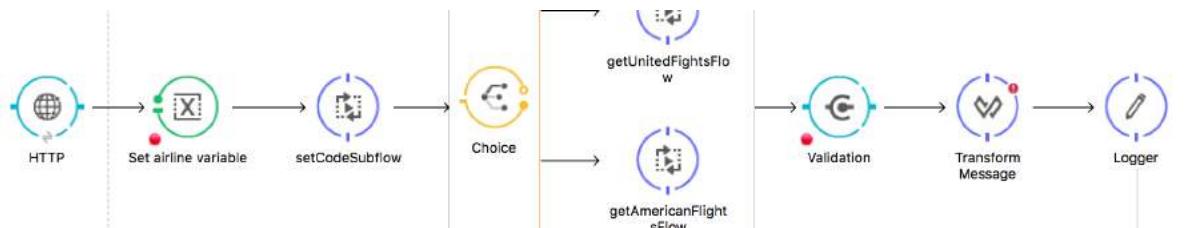
In this walkthrough, you use a validator before the final transformation to check if there are flights and to throw an exception if there are not. You will:

- Use the Validation component to throw an exception.
- Catch the ValidationException in the global exception strategy.



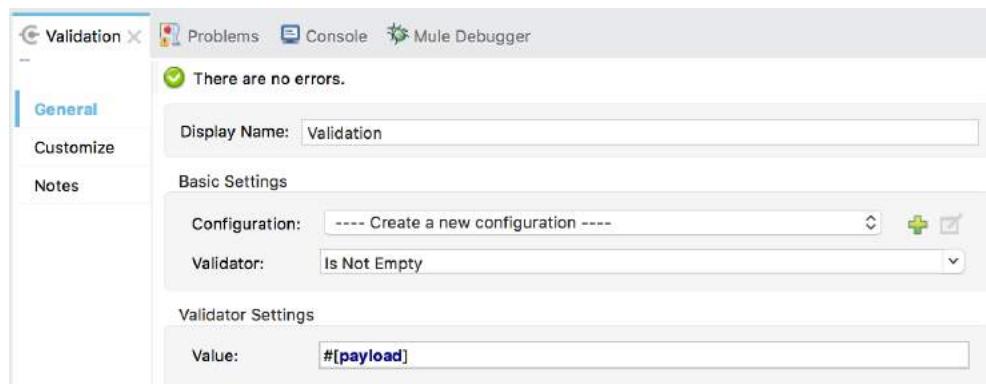
Add a validator

1. Return to getFlightsFlow.
2. Drag out a Validation component from the Mule Palette and drop it before the Transform Message component.

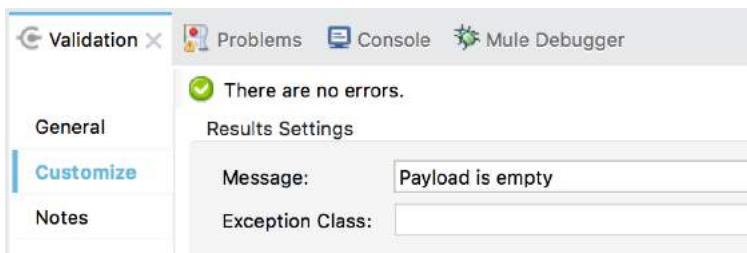


Configure the validator

3. In the Validation properties view, set the validator to Is Not Empty.
4. Set the value to #[payload].

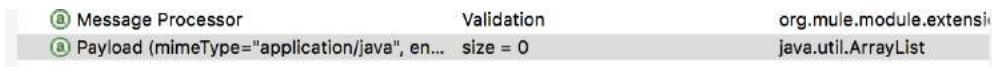


5. In the left-side navigation, click Customize.
6. Set the message to Payload is empty.



Test the application

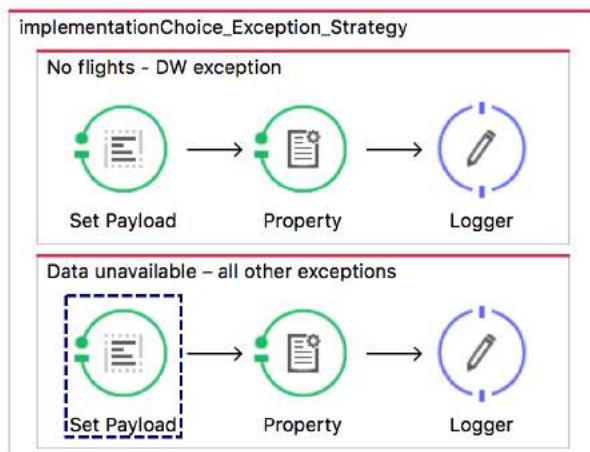
7. Debug the project.
8. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>.
9. In the Mule Debugger, step to the Validation component after the Choice router; you should see the payload has a size of zero.



10. Step again and then drill-down into the exceptionThrown element; you should see a ValidationException was thrown.
11. Locate the value you set for the custom message.

Name	Value	Type
exceptionThrown	org.mule.api.MessagingException	org.mule.api.MessagingException
cause	org.mule.extension.validation.api...	org.mule.extension.validation.api...
CAUSE_CAPTION	Caused by:	java.lang.String
causeRollback	false	java.lang.Boolean
detailMessage	org.mule.extension.validation.api...	java.lang.String
EMPTY_THROWABLE_A...	[Ljava.lang.Throwable;@475d2dbc	java.lang.Throwable[]
errorCode	-1	java.lang.Integer
event	MuleEvent: 0-27eceb21-1dd0-1...	org.mule.DefaultMuleEvent
EXCEPTION_MESSAGE...	*****	java.lang.String
EXCEPTION_MESSAGE...	-----	java.lang.String
failingMessageProcessor	org.mule.module.extension.intern...	org.mule.module.extension.inter...
handled	false	java.lang.Boolean
i18nMessage	Payload is empty (org.mule.exten...	org.mule.config.i18n.Message
info	{Payload Type=java.util.ArrayList,...}	java.util.HashMap
message	Payload is empty.	java.lang.String
muleMessage		org.mule.DefaultMuleMessage
Payload is empty.		

12. Note that a period has been added to the end of the message.
13. Step again; you should move into the Data unavailable catch exception strategy in global.xml.



14. Step to the end of the application.
15. In Postman, you should see the data unavailable message.

The screenshot shows a POSTMAN request configuration and its response. The request is a GET to `http://localhost:8081/flights?code=FOO`. The response is:

- Status: 500 OK
- Time: 196134 ms
- Body (Pretty): DATA IS UNAVAILABLE. TRY LATER.
org.mule.api.MessagingException: Payload is empty.

Modify the global exception handler

16. Return to global.xml.
17. Double-click the No flights catch exception strategy.
18. In the Properties view, modify the execute when expression to also check and see if the exception message is equal to the custom message set by the Validator.

```
# [exception.causeMatches('com.mulesoft.weave.*') ||  
exception.message=='Payload is empty.']}
```

19. Change the display name to No flights.

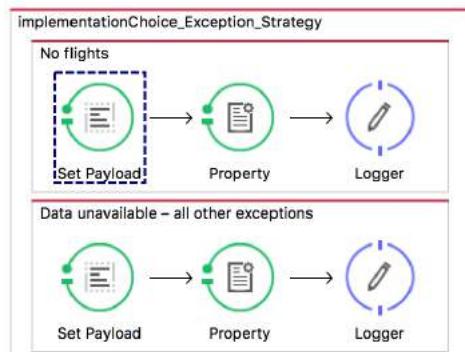
The screenshot shows the Mule Debugger interface with the tab 'No flights - DW exception' selected. A green checkmark indicates 'There are no errors'. The 'Display Name' is set to 'No flights'. Under 'Settings', there is an expression for 'Configure conditional execution using an expression': `#[exception.causeMatches('com.mulesoft.weave.*') || exception.message=='Payload is empty.']}`. The 'Enable Notifications' checkbox is checked.

Test the application

20. Debug the project.

21. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>.

22. In the Mule Debugger, step past the Validation component into the exception strategy; this time, you should move into the No flights catch exception strategy.



23. Step to the end of the application.

24. In Postman, you should see the no flights message.

The screenshot shows the Postman interface. The request method is 'GET', the URL is 'http://localhost:8081/flights?code=FOO', and the response status is '400 OK' with a time of '201905 ms'. The response body is displayed in 'Pretty' format and contains the message: 'NO FLIGHTS to FOO' and 'org.mule.api.MessagingException: Payload is empty.'

25. Return to Anypoint Studio.

26. Close the project.

Module 10: Writing DataWeave Transformations

The screenshot shows the Mule Studio interface for writing DataWeave transformations. On the left, the DataWeave code is displayed:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7@ flights: payload.ns0:listAllFlightsResponse.*return map {
8   destination: $.destination,
9   price: $.price as :number as :currency,
10  planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11  departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12  as :string {format: "MMM dd, yyyy"},
13  availableSeats: $.emptySeats as :number,
14  //totalSeats: getNumSeats($.planeType)
15  totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
16 }
```

On the right, the transformation results are shown in a table:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
destination	SFO
price	400.00
planeType	BOEING 737
departureDate	Oct 20, 2015
availableSeats	40
Unknown	
[1]	LinkedHashMap
destination	LAX
price	199.99
planeType	BOEING 737

Objectives:

- Write DataWeave expressions for basic XML, JSON, and Java transformations.
- Store DataWeave transformations in external files.
- Write DataWeave transformations for complex data structures with repeated elements.
- Coerce and format strings, numbers, and dates.
- Use DataWeave operators.
- Define and use custom data types.
- Call MEL functions and Mule flows from DataWeave transformations.

Walkthrough 10-1: Write your first DataWeave transformation

In this walkthrough, you work with JSON data for a flight posted to a flow. You will:

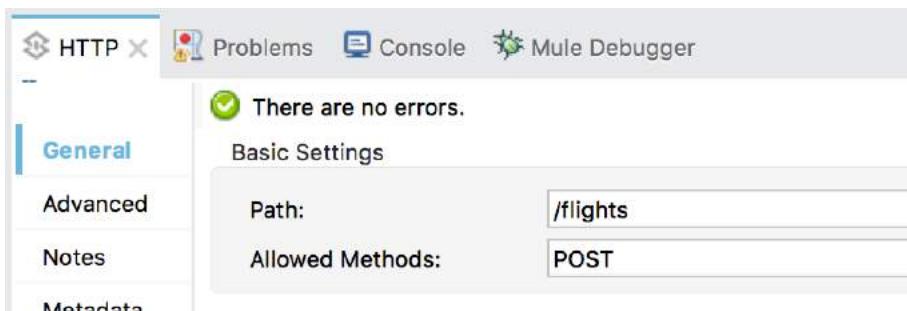
- Create a new flow that receives POST requests.
- Write a DataWeave expression to transform the data to Java, XML, or JSON.
- Add sample data and use live preview.
- Save the transformation in an external file.

The screenshot shows the Mule Studio interface. On the left is the project tree with files like 'flights-DEV.properties' and 'log4j2.xml'. The main area is a code editor with XML code for a flow named 'postFlightFlow'. The code includes an HTTP listener, a DataWeave transform message component, and a logger. To the right of the code editor is a preview window showing a DataWeave transformation script:

```
*implementation
1 %dw 1.0
2 %output application/xml
3 ---
4 payload
```

Create a new flow

1. Return to implementation.xml.
2. Drag an HTTP connector from the Mule Palette to the bottom of the canvas.
3. Change the name of the new flow to postFlightFlow.
4. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /flights and the allowed methods to POST.

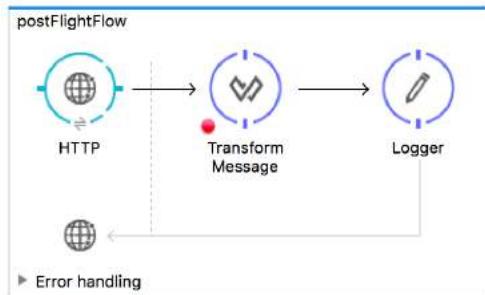


6. Add a Logger to the flow.

Add a DataWeave Transform Message component

7. Add a DataWeave Transform Message component before the Logger.

- Add a breakpoint to it.



- In the Transform Message properties view, locate the drop-down menu at the top of the view that sets the output type; it should be set to Payload.
- Beneath it, locate the directive that sets the output to application/java.
- Delete the existing DataWeave expression (the curly braces) and set it to payload.

Output Payload Preview

```

1 %dw 1.0
2 %output application/java
3 ---
4 payload
  
```

Post data to the flow

- Open api.raml in src/main/api.
- Locate the post method and copy the example for the post body.
- Debug the project.
- In Postman, change the method to POST and remove any query parameters.
- Add a request header called Content-Type and set it equal to application/json.
- For the request body, select raw and paste the value you copied from api.raml.
- Make the request to <http://localhost:8081/flights>.

```

{
  "airline": "Delta",
  "flightCode": "ER0945",
  "fromAirportCode": "PDX",
  "toAirportCode": "CLE",
  "departureDate": "June 1, 2016",
  "emptySeats": 24,
  "totalSeats": 350,
  "price": 450,
  "planeType": "Boeing 747"
}
  
```

19. In the Mule Debugger, you should see the payload has a mime-type of application/json and is of type BufferInputStream.

Name	Value	Type
» e DataType	SimpleDataType{type=or...}	org.mule.transformer.types.SimpleDataType
» a Exception	null	
» e Message		org.mule.DefaultMuleMessage
» a Message Processor	Transform Message	com.mulesoft.weave.mule.WeaveMessag...
» e Payload (mimeType="application/json",...	org.glassfish.grizzly.utils....	org.glassfish.grizzly.utils.BufferInputStream

20. Look at the inbound properties; you should see the content-type is set to application/json.

21. Step to the Logger; you should see the payload now has a mime-type of application/java and is a LinkedHashMap.

Name	Value	Type
» a message		org.mule.DefaultMuleMessage
» a Message Processor	Logger	org.mule.api.processor.LoggerMes...
» e Payload (mimeType="application/java",...	size = 9 airline=Delta flightCode=ER0945 fromAirportCode=PDX toAirportCode=CLE departureDate=June 1, 2016 emptySeats=24 totalSeats=350 price=450 planeType=Boeing 747	java.util.LinkedHashMap java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry java.util.LinkedHashMap\$Entry

22. Click the Resume button.

Change output data type to JSON

23. In the Transform Message properties view, change the output directive to application/json.

24. Debug the project.

25. In Postman, post the same request to <http://localhost:8081/flights>.

26. In the Mule Debugger, step to the Logger; you should see the payload has a mime-type of application/json and is a Weave ByteArraySeekableStream.

Name	Value	Type
» e DataType	SimpleDataType{type=or...}	org.mule.transformer.types.SimpleDataType
» a Exception	null	
» e Message		org.mule.DefaultMuleMessage
» a Message Processor	Transform Message	com.mulesoft.weave.mule.WeaveMessageProc...
» e Payload (mimeType="application/json",...	org.glassfish.grizzly.utils....	org.glassfish.grizzly.utils.BufferInputStream

27. Step to the end of the application.

Change output data type to XML

28. In the Transform Message properties view, change the output directive to application/xml.

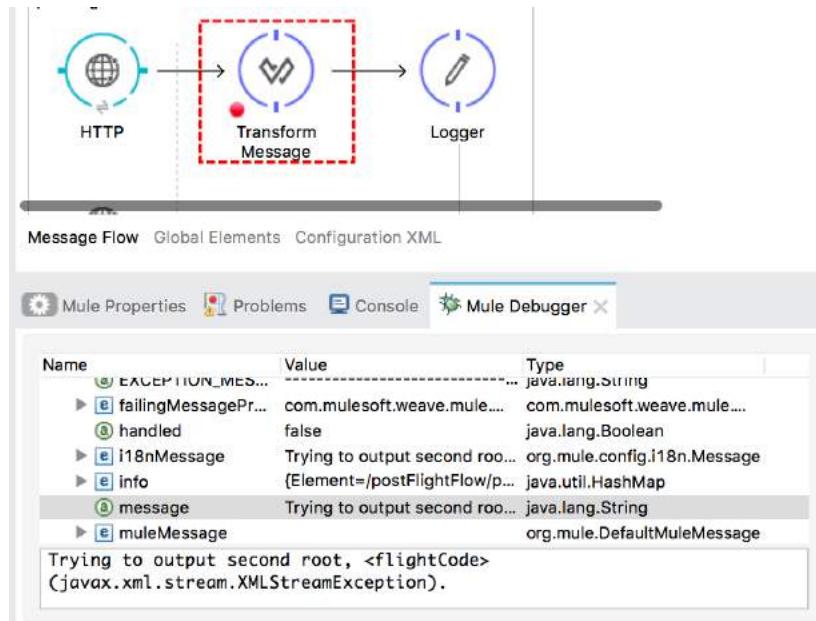
29. Debug the project.

30. In Postman, post the same request to <http://localhost:8081/flights>.

31. In the Mule Debugger, click Next processor; you should get an exception.

32. Drill-down into the exceptionThrown and locate the message; you should see there is a

DataWeave error when trying to output the second root.



33. Click Resume.

34. Stop the project.

Add input metadata and sample data

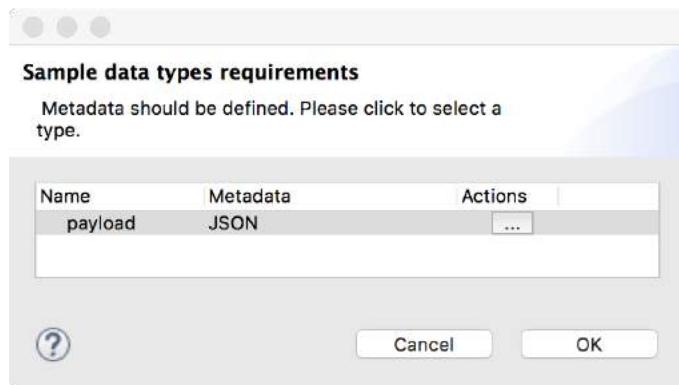
35. In the Transform Message properties view, change the output type from application/xml back to application/json.

36. Click the Preview button.

37. Click the Create required sample data to execute preview link.

38. In the Sample data types requirements dialog box, click the word Unknown under Metadata.

39. In the drop-down menu that appears, select JSON.



40. Click the button under Actions.

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

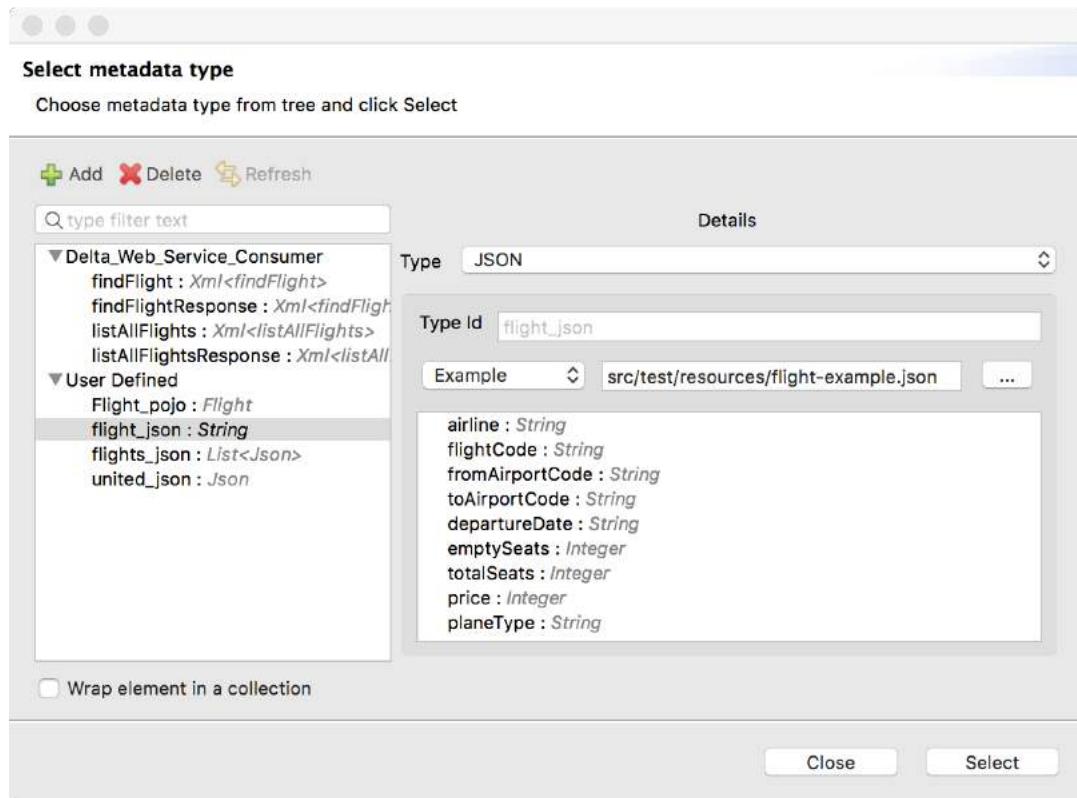
42. In the Create new type dialog box, set the type id to flight_json and click Create type.

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

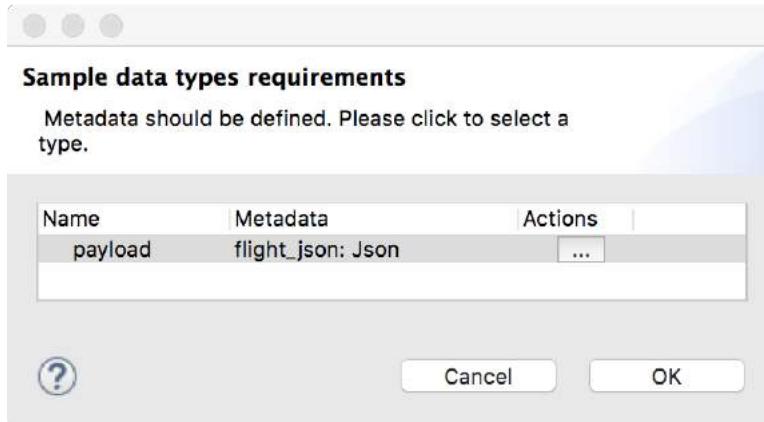
44. Select Example and browse to and select the flight-example.json file in src/main/resources.

Note: Be sure to select flight-example.son (singular) and not flights-example.json (plural).

45. In the Select metadata type dialog box, click Select.



46. In the Sample data types requirements dialog box, locate the new metadata type value; it should be flight_json: Json.
47. Click OK.



Preview sample data and sample output

48. In the Transform Message properties view, look at the input section; you should see a new tab called payload and it should contain the sample data.
49. Click the Show Source With Trees button to the left of the Preview button.
50. Look at the preview section; you should see the sample output there of type JSON.

```

flight-example.json
{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}
Context payload ✖

Output Payload ↕ ⚪ 🗑️ Preview
Unknown Define metadata
1 %dw 1.0
2 %output application/json
3 ---
4 payload
{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}
  
```

Change the output type

51. In the transform section, change the output type from application/json to application/java.
52. Look at the preview section; you should see the sample output there is now a LinkedHashMap object populated with the sample data.

flight-example.json

```
{ "airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737" }
```

Output Payload ▾ Preview

Unknown [Define metadata](#)

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload
```

Name	Value
root : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737

53. In the transform section, change the output type from application/java to application/xml.
54. Locate the warning icons indicating that there is a problem.

Output Payload ▾ 1 issue found

```
1 %dw 1.0
2 %output application/xml
3 ---
4 payload
```

55. 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>.
56. Click the icon above the code; a List of errors dialog box should open.

List of errors

Select an error to see details below

Name	Target
javax.xml.stream.XMLStreamException: Trying to output seco...	Payload

javax.xml.stream.XMLStreamException: Trying to output second root, <flightCode>

OK

57. In the List of errors dialog box, click OK.
58. Change the output from application/xml to application/java for now.

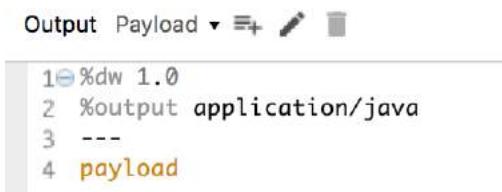
Note: You will learn how to successfully transform to XML in the next walkthrough.

Save the transformation in an external file

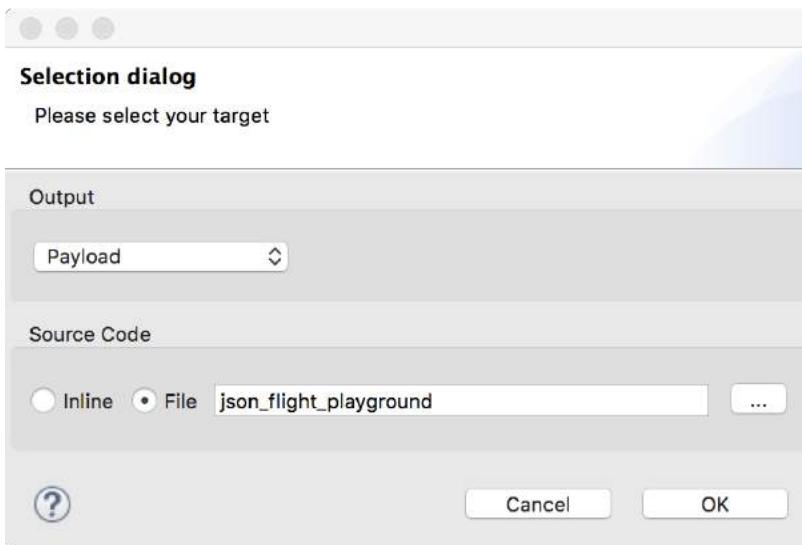
59. Switch the canvas to the Configuration XML view.
60. Locate and review the code for the DataWeave transformation.

```
148      </flow>
149      <flow name="postFlightFlow">
150          <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedMethods="GET,POST,PUT,DELETE">
151              <dw:transform-message metadata:id="5ded8a12-3d01-4c39-98f1-395a9ac25b31" doc:name="DataWeave">
152                  <dw:input-payload doc:sample="flight-example.json"/>
153                  <dw:set-payload><![CDATA[%dw 1.0
154 %output application/java
155 ---
156 payload]]></dw:set-payload>
157             <dw:transform-message>
158                 <logger level="INFO" doc:name="Logger"/>
159             </dw:transform-message>
160         </dw:transform-message>
161     </flow>
```

61. Switch back to the Message Flow view.
62. In the Transform Message properties view, click the Edit current target button (the pencil) above the code.



63. In the Selection dialog dialog box, change the source code selection form inline to file.
64. Set the file name to json_flight_playground.



65. Click OK.

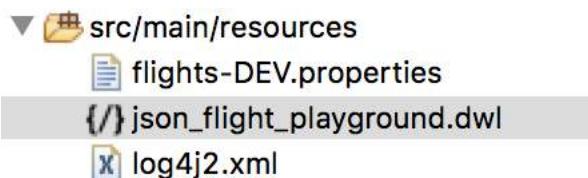
Locate and review the code

66. Switch to Configuration XML view.

67. Scroll down and look at the new code for the transformation in postFlightFlow.

```
148     </TLOW>
149     <flow name="postFlightFlow">
150         <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedMethods="GET,POST" />
151         <dw:transform-message metadata:id="5ded8a12-3d01-4c39-98f1-395a9ac25b31" doc:name="JSON-to-Flight">
152             <dw:input-payload doc:sample="flight-example.json"/>
153             <dw:set-payload resource="classpath:json_flight_playground.dwl"/>
154         </dw:transform-message>
155         <logger level="INFO" doc:name="Logger"/>
156     </flow>
```

68. In the Package Explorer, expand src/main/resources.



69. Open json_flight_playground.dwl.

70. Review and then close the file.

The screenshot shows the Eclipse IDE's code editor with the file 'json_flight_playground.dwl' open. The tab bar at the top shows the file name. The code in the editor is:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload
```

71. Return to Message Flow view in implementation.xml.

72. Save the file.

Walkthrough 10-2: Transform basic Java, JSON, and XML data structures

In this walkthrough, you continue to work with the JSON flight data posted to the flow. You will:

- Write expressions to transform the JSON payload to various Java structures.
- Create a second transformation to store a transformation output in a flow variable.
- Write expressions to transform the JSON payload to various XML structures.

The screenshot shows the 'Output' tab of the Transform Message properties view. The code is as follows:

```
1 %dw 1.0
2 %output application/xml
3 ---
4 data: {
5     hub: "MUA",
6     flight @{airline: payload.airline}: {
7         code: payload.toAirportCode
8     }
}
```

The preview pane shows the resulting XML output:

```
<?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 in postFlightFlow.
2. Type a period after payload and select price from the pop-up menu.

The screenshot shows the 'Output' tab of the Transform Message properties view. The code is as follows:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.
```

A context menu is open over the 'payload.' part of the code, showing options like 'airline : string', 'departureDate : string', etc. To the right, the preview pane shows a table of values:

Name	Value
root : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
departureDate : string	LAX
emptySeats : number	SFO
flightCode : string	May 21, 20
fromAirportCode : string	0
planeType : string	200
price : number	199
toAirportCode : string	Boeing 737
totalSeats : number	

3. Look at the preview; you should see the output is an integer.

The screenshot shows the 'Output' tab of the Transform Message properties view. The code is as follows:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.price
```

The preview pane shows a table with one row:

Name	Value
root : Integer	199

4. Change the DataWeave expression to data: payload.price.

Output Payload ▾  

1@%dw 1.0 2 %output application/java 3 --- 4 data: payload.price	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td> data : Integer</td> <td>199</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : Integer	199
Name	Value						
root : LinkedHashMap							
data : Integer	199						

   Preview

5. Change the DataWeave expression to data: payload.

Output Payload ▾  

1@%dw 1.0 2 %output application/java 3 --- 4 data: payload	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td> data : LinkedHashMap</td> <td></td> </tr> <tr> <td> airline : String</td> <td>United</td> </tr> <tr> <td> flightCode : String</td> <td>ER38sd</td> </tr> <tr> <td> fromAirportCode : String</td> <td>LAX</td> </tr> <tr> <td> toAirportCode : String</td> <td>SFO</td> </tr> <tr> <td> departureDate : String</td> <td>May 21, 2016</td> </tr> <tr> <td> emptySeats : Integer</td> <td>0</td> </tr> <tr> <td> totalSeats : Integer</td> <td>200</td> </tr> <tr> <td> price : Integer</td> <td>199</td> </tr> <tr> <td> planeType : String</td> <td>Boeing 737</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : LinkedHashMap		airline : String	United	flightCode : String	ER38sd	fromAirportCode : String	LAX	toAirportCode : String	SFO	departureDate : String	May 21, 2016	emptySeats : Integer	0	totalSeats : Integer	200	price : Integer	199	planeType : String	Boeing 737
Name	Value																								
root : LinkedHashMap																									
data : LinkedHashMap																									
airline : String	United																								
flightCode : String	ER38sd																								
fromAirportCode : String	LAX																								
toAirportCode : String	SFO																								
departureDate : String	May 21, 2016																								
emptySeats : Integer	0																								
totalSeats : Integer	200																								
price : Integer	199																								
planeType : String	Boeing 737																								

   Preview

6. Change the DataWeave expression to data: {}.

7. Add a field called hub and set it to "MUA".

Output Payload ▾  

1@%dw 1.0 2 %output application/java 3 --- 4@data: { 5 hub: "MUA" 6 }	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td> data : LinkedHashMap</td> <td></td> </tr> <tr> <td> hub : String</td> <td>MUA</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : LinkedHashMap		hub : String	MUA
Name	Value								
root : LinkedHashMap									
data : LinkedHashMap									
hub : String	MUA								

   Preview

8. Add a field called code and set it to the toAirportCode property of the payload.

9. Add a field called airline and set it to the airline property of the payload.

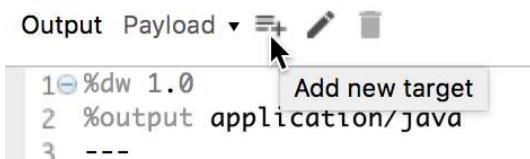
Output Payload ▾  

1@%dw 1.0 2 %output application/java 3 --- 4@data: { 5 hub: "MUA", 6 code: payload.toAirportCode, 7 airline: payload.airline 8 }	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td> data : LinkedHashMap</td> <td></td> </tr> <tr> <td> hub : String</td> <td>MUA</td> </tr> <tr> <td> code : String</td> <td>SFO</td> </tr> <tr> <td> airline : String</td> <td>United</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : LinkedHashMap		hub : String	MUA	code : String	SFO	airline : String	United
Name	Value												
root : LinkedHashMap													
data : LinkedHashMap													
hub : String	MUA												
code : String	SFO												
airline : String	United												

   Preview

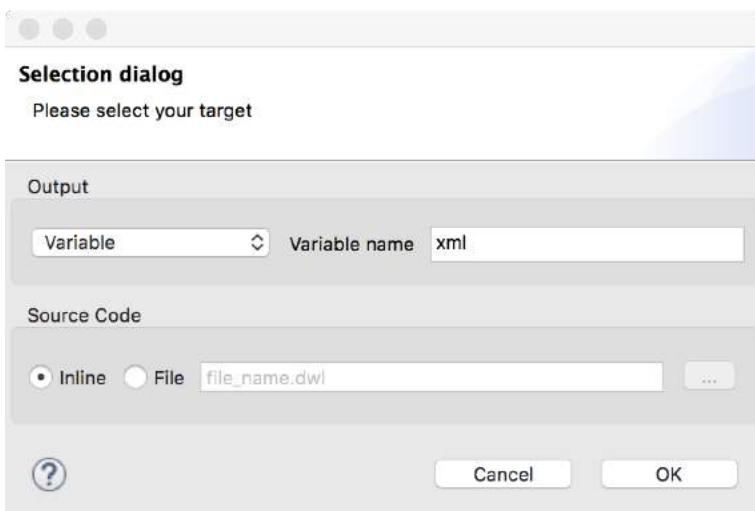
Create a second transformation with the same component

10. Click the Add new target button.



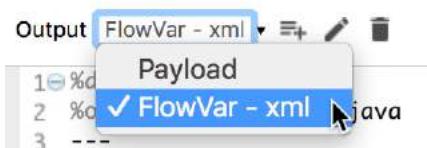
11. In the Selection dialog dialog box, leave the output set to variable.

12. Set the variable name to xml.



13. Click OK.

14. 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.



Write an expression to output data as XML

15. For the FlowVar – xml transformation, click the Preview button.
16. Set the DataWeave expression to payload.
17. Change the output type to application/xml; you should get an issue displayed.
18. Mouse over the warning icon and read the message; it should be the same one you saw in the last walkthrough.
19. Change the DataWeave expression to data: payload.

20. Look at the preview; you should now see XML.

Output FlowVar - xml ▾ Preview

```
1@%dw 1.0
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>
```

21. In the output section menu bar, click the drop-down arrow next to FlowVar – xml and select Payload.

22. Copy the DataWeave expression you just wrote.

23. Switch back to FlowVar – xml and replace the DataWeave expression with the one you just copied.

Output FlowVar - xml ▾ Preview

```
1@%dw 1.0
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>
```

24. Modify the expression so the code and airline properties are child elements of a new element called flight.

Output FlowVar - xml ▾ Preview

```
1@%dw 1.0
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>
```

25. Modify the expression so the airline is an attribute of the flight element.

The screenshot shows the Mule Studio interface. On the left, the XML configuration is displayed:

```
1 @%dw 1.0
2 %output application/xml
3 ---
4 @data: {
5     hub: "MUA",
6     flight @{airline: payload.airline}: {
7         code: payload.toAirportCode
8 }
```

On the right, the generated XML output is shown:

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
    <hub>MUA</hub>
    <flight airline="United">
        <code>SF0</code>
    </flight>
</data>
```

Debug the application

26. Debug the project.

27. In Postman, post the same request to <http://localhost:8081/flights>.

28. In the Mule Debugger, step to the Logger and click the Variables tab; you should now see a flow variable.

The screenshot shows the Mule Debugger interface with the Variables tab selected. On the left, the variable tree is shown:

- Message
- Message Processor
- Payload (mimeType: application/xml) size = 1
 - 0 data={hub=MUA, cod...}
 - key
 - value
 - 0 hub=MUA
 - 1 code=CLE
 - 2 airline=Delta
- size = 1

On the right, the variable details are shown:

Name	Value	Type
@ xml (mimeType: application/xml)	<?xml version='1.0' encoding='UTF-8'?><data><hub>MUA</hub><flight airline="Delta"><code>CLE</code></flight></data>	java.lang.String

29. Stop the project.

Walkthrough 10-3: Transform complex data structures with arrays

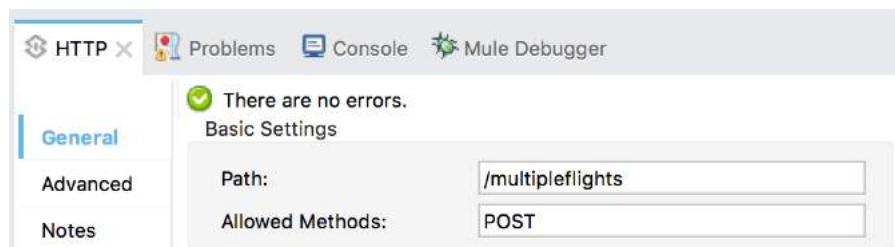
In this walkthrough, you work with JSON data for multiple flights posted to a flow. You will:

- Create a new flow that receives POST requests of a JSON array of objects.
- Transform a JSON array of objects to Java.

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
flightO : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
flight : Integer	1

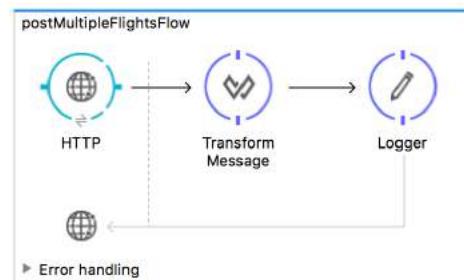
Create a new flow

1. Return to implementation.xml.
2. Drag out an HTTP connector to the bottom of the canvas.
3. Change the flow name to postMultipleFlightsFlow
4. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /multipleflights and set the allowed methods to POST.



Transform the payload to Java

6. Add a Transform Message component and a Logger to the flow.



- In the Transform Message properties view, set the DataWeave expression to payload.
- Click the Preview button.

Add input metadata and sample data

- In the preview section, click the Create required sample data to execute preview link.
- In the Sample data types requirements dialog box, set the metadata type to JSON.
- Click the Actions button.
- In the Select metadata dialog box, select flights_json and click Select.
- In the Sample data types requirements dialog box, click OK.
- Look at the preview section; the sample output should be an ArrayList of LinkedHashMaps.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. In the preview section, there is a JSON sample file named 'flights-example.json' containing flight information. Below it, a 'Context' table shows 'payload' with a value of 'flights-example.json'. The preview panel displays the DataWeave code and the resulting payload structure. The payload is an ArrayList of two LinkedHashMaps. The first LinkedHashMap has keys 'airline', 'flightCode', 'fromAirportCode', 'toAirportCode', 'departureDate', 'emptySeats', 'totalSeats', 'price', and 'planeType'. The second LinkedHashMap has the same keys but with different values (e.g., 'airline': 'Delta').

Name	Value
root : ArrayList	
[0] : LinkedHashMap	airline : String flightCode : String fromAirportCode : String toAirportCode : String departureDate : String emptySeats : Integer totalSeats : Integer price : Integer planeType : String
[1] : LinkedHashMap	airline : String flightCode : String fromAirportCode : String toAirportCode : String departureDate : String emptySeats : Integer totalSeats : Integer price : Integer planeType : String

Return values for the first object in the collection

- Change the DataWeave expression to payload[0]; in the preview section, you should see one LinkedHashMap object.
- Change the DataWeave expression to payload[0].price; in the preview section, you should get 199, the first price value.
- Change the DataWeave expression to payload[0].*price; in the preview section, you should see an ArrayList with one integer value of 400 – the first price value.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. The preview panel displays the DataWeave code and the resulting payload structure. The payload is an ArrayList of two LinkedHashMaps. The first LinkedHashMap has keys 'airline', 'flightCode', 'fromAirportCode', 'toAirportCode', 'departureDate', 'emptySeats', 'totalSeats', 'price', and 'planeType'. The second LinkedHashMap has the same keys but with different values (e.g., 'airline': 'Delta').

Name	Value
root : ArrayList	
[0] : LinkedHashMap	199

Return collections of values

18. Change the DataWeave expression to return a collection of all the prices; in the preview section, you should see two prices in an ArrayList.

```
payload.*price
```

Output Payload ▾ Preview

Name	Value
root : ArrayList	
[0] : Integer	199
[1] : Integer	450

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.*price
```

19. Change the DataWeave expression to payload.price; you should get the same result.

20. Change the DataWeave expression to return a collection of prices and available seats; in the preview section, you should see an ArrayList with two ArrayLists.

```
[payload.price, payload.emptySeats]
```

Output Payload ▾ Preview

Name	Value
root : ArrayList	
[0] : ArrayList	
[0] : Integer	199
[1] : Integer	450
[1] : ArrayList	
[0] : Integer	0
[1] : Integer	24

```
1 %dw 1.0
2 %output application/java
3 ---
4 [payload.price, payload.emptySeats]
5
```

Use the map operator to return object collections with different data structures

21. Change the DataWeave expression to payload map \$; in the preview section, you should see two ArrayList of two LinkedHashMaps again.

Output Payload ▾ Preview

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
airline : String	Delta
flightCode : String	ER0945
fromAirportCode : String	PDX
toAirportCode : String	CLE
departureDate : String	June 1, 2016
emptySeats : Integer	24
totalSeats : Integer	350
price : Integer	450
planeType : String	Boeing 747

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map $
5
```

22. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.

Output Payload ▾

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
[1] : LinkedHashMap	
flight : Integer	1

```

1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$
6 }
7

```

23. 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 ▾

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
destination : String	SFO
[1] : LinkedHashMap	
flight : Integer	1
destination : String	CLE

```

1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$,
6   destination: $.toAirportCode
7 }
8

```

24. Change the DataWeave expression to dynamically add each of the properties present on the payload objects.

Output Payload ▾

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
[0] : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
flight : Integer	1

```

1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$,
6   ($$): $
7 }
8

```

25. 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 and its preview pane. The DataWeave code is:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map {
5     flight: $$,
6     'flight$$': $
7 }
```

The preview pane displays the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
flight0 : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
flight : Integer	1

Note: If you want to test the application, change the output type to application/json and then in Postman, change the request URL to <http://localhost:8081/multipleflights> and replace the request body with JSON from the flights-example.json file, .

Change the expression to output XML

26. Change the DataWeave expression output type from application/java to application/xml; you should get an issue displayed.

27. Change the DataWeave expression to create a root node called flights; you should still get an issue.

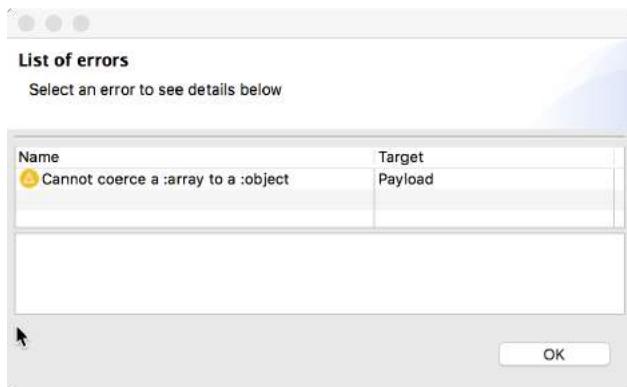
The screenshot shows the DataWeave editor and its preview pane. The DataWeave code is identical to the previous example:

```
1 %dw 1.0
2 %output application/xml
3 ---
4 flights: payload map {
5     flight: $$,
6     'flight$$': $
7 }
```

The preview pane shows the same structure as before, but there is a warning icon in the top bar indicating "1 issue found".

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
flight0 : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd

28. Look at the error message.



Walkthrough 10-4: 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.
- Transform XML with repeated elements to Java.

```
flights-example.xml
<ns2:listAllFlightsResponse
  xmlns:ns2="http://soap.training.mulesoft.com/"
  >
  <return
    airlineName="United"
    code="A1B2C3"
    departureDate="2015/10/20"
    destination="SFO"
    emptySeats="40"
    origin="MUA"
    planeType="Boeing 737"
    price="400.0"
  </return>
</listAllFlightsResponse>
```

```
Output Payload
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {$(payload.ns0:listAllFlightsResponse.*return map {
6   flight: {
7     dest: $.destination,
8     price: $.price
9   }
10 })
11 }
12
13
```

```
<?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>
  <flight>
    <dest>PDX</dest>
    <price>283.0</price>
  </flight>
</flights>
```

Transform the JSON array of objects to XML

- Return to the Transform Message properties view in postMultipleFlightsFlow.
- Change the expression to map each item in the input array to an XML element.

```
flights: {$(payload map {
  flight: $$,
  'flight$$': $
})}
```

- Look at the preview; the JSON should be transformed to XML successfully.

```
Output Payload
1 %dw 1.0
2 %output application/xml
3 ---
4 flights: {$(payload map {
5   flight: $$,
6   'flight$$': $
7 })
8 }
9
10
```

```
<?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>
```

4. Remove the flight property.
5. Modify the expression so the flights object has a single property called flight.

The screenshot shows the Mule Studio interface with the 'Output' tab selected. In the payload editor, there is a code editor containing Java-like pseudocode and an XML preview pane. The XML preview shows a flight object with properties like airline, flightCode, departureDate, etc.

```

1@%dw 1.0
2 %output application/xml
3 ---
4@ flights: {payload map {
5   flight: $}
6 }
7 }
8
9

```

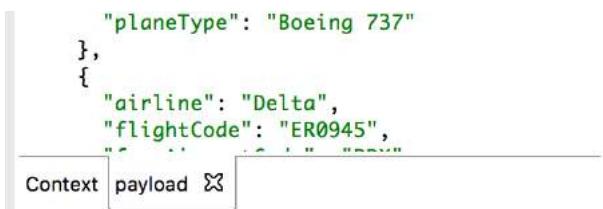
```

<?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>

```

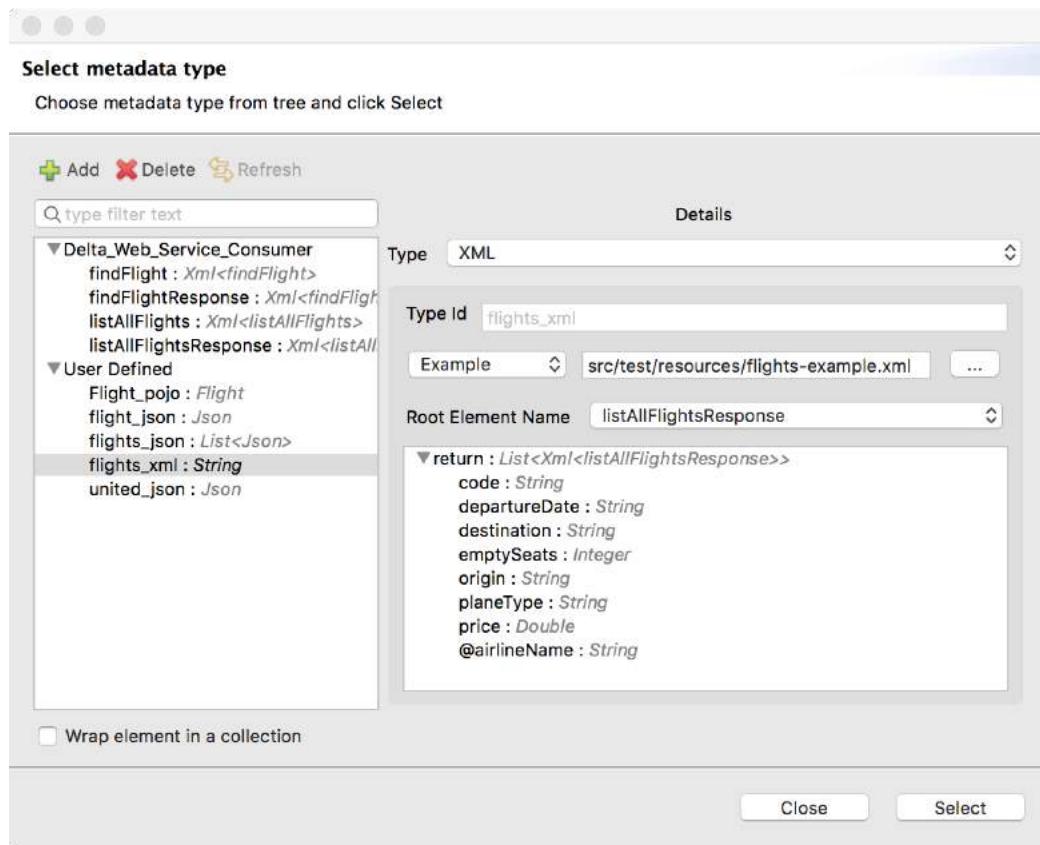
Change input metadata and sample data to XML

6. In the input section, click the x on the payload tab.



7. In the Close Sample Data dialog box, click Close tab and Keep file.
8. In the input section, right-click Payload: List<Json> and select Clear Metadata.
9. In the preview section, click the link to Create required sample data to execute preview.
10. In the Sample data types requirements dialog box, set the metadata type to XML.
11. Click the Actions button.
12. In the Select metadata type dialog box, click the Add button.
13. In the Create new type dialog box, set the type id to flights_xml and click Create type.
14. In the Select metadata dialog box, set the type to XML.
15. Select Example and browse to and select the flights-example.xml file in src/main/resources.

16. Click Select.



17. In the Sample data types requirements dialog box, click OK.

18. Look at the XML sample data in the input section.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. The 'flights-example.xml' file is open in the editor. The XML content is as follows:

```
<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><emptySeats>10</emptySeats>
    </return>

```

Below the editor, there are tabs for 'Context' and 'payload'.

Examine the sample output for the transformation

19. Review the transformation expression.

20. Look at the preview section; you should see all the flights are children of the flight element.

Output Payload ▾   Preview

```
1 %dw 1.0
2 %output application/xml
3 ---
4 @flights: {$(payload map {
5   flight: $}
6 )} 
7
8
9
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <return airlineName="United">
      <code>A1B2C3</code>
      <departureDate>2015/10/20</departureDate>
      <destination>SF0</destination>
      <emptySeats>40</emptySeats>
      <origin>MUA</origin>
      <planeType>Boing 737</planeType>
      <price>400.0</price>
    </return>
    <return airlineName="Delta">
      <code>A1B2C4</code>
```

Change the return structure of the XML

21. Change the DataWeave expression to loop over the listAllFlightsResponse element of the payload; you should see the flights are now correctly transformed.

Output Payload ▾   Preview

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 @flights: {$(payload.ns0#listAllFlightsResponse map {
6   flight: $}
7 )} 
8
9
10
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <code>A1B2C3</code>
    <departureDate>2015/10/20</departureDate>
    <destination>SF0</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>
```

22. Change the DataWeave expression to loop over the payload.listAllFlightsResponse.return element.

Output Payload ▾   Preview

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 @flights: {$(payload.ns0#listAllFlightsResponse.return map {
6   flight: $}
7 )} 
8
9
10
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>A1B2C3</flight>
  <flight>2015/10/20</flight>
  <flight>SF0</flight>
  <flight>40</flight>
  <flight>MUA</flight>
  <flight>Boing 737</flight>
  <flight>400.0</flight>
</flights>
```

23. Change the DataWeave expression use * to loop over the XML repeated return elements.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor contains the following code:

```
1@%dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: {${payload.ns0:listAllFlightsResponse.*}return map {
6   flight: $.
7 }
8 )}
9
10
```

On the right, the XML preview pane shows the resulting XML structure:

```
<?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>
```

24. Change the DataWeave expression to return flight elements with dest and price elements that map to the corresponding destination and price input values.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor contains the following code:

```
1@%dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: {${payload.ns0:listAllFlightsResponse.*}return map {
6   flight: {
7     dest: $.destination,
8     price: $.price
9   }
10 }
11 )}
12
```

On the right, the XML preview pane shows the resulting XML structure:

```
<?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>
  <flight>
    <dest>PDX</dest>
```

Note: If you want to test the application with Postman, 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 transformation to return Java

25. Change the output type from application/xml to application/java.

26. Look at the preview; you should see flights is a LinkedHashMap with one flight property.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor contains the same code as in the previous examples:

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: {${payload.ns0:listAllFlightsResponse.*}return map {
6   flight: {
7     dest: $.destination,
8     price: $.price
9   }
10 }
11 )}
```

On the right, the Java preview pane shows the resulting Java structure:

```
Name
  root : LinkedHashMap
    flights : LinkedHashMap
      flight : LinkedHashMap
        dest : String
        price : String
```

27. In the DataWeave expression, remove the {{ }} around the map expression.
28. Change the DataWeave expression to remove the flight property.
29. Look at the preview; you should see flights is an ArrayList with five LinkedHashMaps.

Output Payload   

    Preview

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.0
[1] : LinkedHashMap	
dest : String	LAX

```

1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price|
8 }

```

Walkthrough 10-5: Coerce and format strings, numbers, and dates

In this walkthrough, you continue to work with the XML flights data posted to the flow. You will:

- Explore the DataWeave documentation.
- Coerce data types.
- Format strings, numbers, and dates.

The screenshot shows the Anypoint Studio interface with the DataWeave editor open. The code pane contains the following DataWeave script:

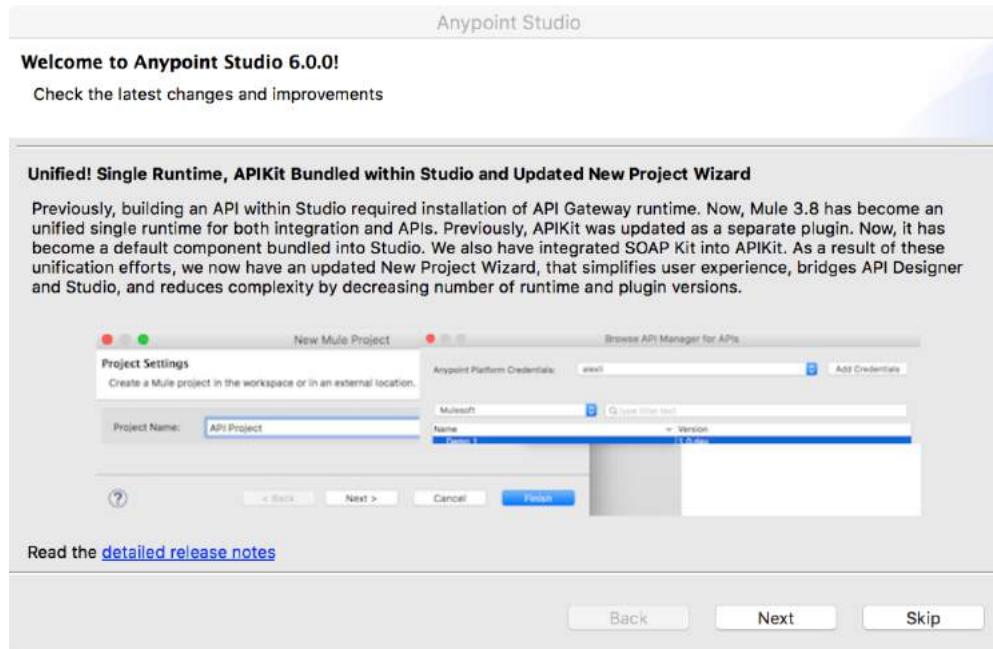
```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###.##"}, // Coerces price to a string with two decimal places
8     plane: upper $.planeType,
9     date: $.departureDate as :date {format: "yyyy/MM/dd"} // Coerces date to a string in YYYY/MM/DD format
10    as :string {format: "MMM dd, yyyy"} // Coerces date to a string in MMM dd, yyyy format
11 }
12
13
14
```

The preview pane on the right displays the resulting data structure as a table:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOING 737
date	Oct 20, 2015
[1]	LinkedHashMap
dest	LAX
price	199.99
plane	BOING 737
date	Oct 21, 2015

Browse DataWeave reference documentation

- In the main menu bar in Anypoint Studio, select Help > What's new in Anypoint Studio?
- In the Anypoint Studio dialog box, click the detailed release notes link; a new web browser window should open.



- In the left-side navigation, navigate to Mule User Guide > Developing > Transformers > DataWeave > DataWeave Operators.

Search the docs

Nav Edit on GitHub ★★★★★

Mule User Guide Version 3.8 (Latest) ▾

Map

Check the [Precedence Table](#) to see the order in which DataWeave expressions are compiled.

In this topic:

- Map
- Map Object
- Pluck
- Filter
- Remove
- AS (Type Coercion)**

- Locate the In this topic navigation on the right-side of the page.
- Scroll the right-side navigation.
- Click the AS (Type Coercion) link.
- Scroll the page and browse the examples and documentation.

Coerce to date

(:string', 'type)(:number', 'type) => :date'

Date types can be coerced from string or number.

Any format pattern accepted by [DateTimeFormatter](#) is allowed.

Transform

```
%dw 1.0
%output application/json
---
{
  a: 143628723 as :datetime,
  b: "2015-10-07 16:40:32.000" as :localdatetime {format: "yyyy-MM-dd HH:mm:s"}
}
```

Output

```
{
  "a": "2015-07-07T16:40:32Z",
  "b": "2015-10-07 16:40:32.000"
}
```

DATAWEAVE

JSON

In this topic:

- Remove by Matching Key and Value
- AND
- OR
- IS
- Concat
- Contains
- AS (Type Coercion)**
- Type Of
- Flatten
- Size Of
- Array Push

- In the right-side navigation, click the AS (Type Coercion) link again.

9. In the page, locate and click the type coercion table link.

The screenshot shows a web browser displaying the Mule User Guide at <https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave-operators#as-type-coercion>. The page title is "AS (Type Coercion)". A sidebar on the right lists various DataWeave operators, with "AS (Type Coercion)" highlighted. The main content area contains a brief description of the operator and two callout boxes with icons: one about DataWeave's default conversion behavior and another about checking the type coercion table.

10. Browse the type coercion table.

The screenshot shows a web browser displaying the Mule User Guide at <https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave-types#type-coercion-table>. The page title is "Type Coercion Table". It includes a note about the AS Operator and its properties, followed by a table showing source and target types and their associated properties.

Source	Target	Property
:object	:array	(1)
:range	:array	
:number	:binary	

In this topic:

- Array
- Object
- String
- Number
- Boolean
- Dates

Format a string

11. Return to Anypoint Studio.
12. In the Anypoint Studio dialog box, click the Skip button.

13. Navigate to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
14. Change the DataWeave expression to return objects that also have a property called plane equal to the value of the input planeType values.
15. Use the upper operator to return the value in uppercase.

```
plane: upper $.planeType
```

16. Look at the preview; you should see the uppercase plane fields.

Output Payload

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.0
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX

```
1%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price,
8   plane: upper $.planeType
9 }
10
```

Coerce a string to a number

17. In the preview, look at the data type of the prices; you should see that they are strings.
18. Change the DataWeave expression to use the as operator to return the prices as numbers.

```
price: $.price as :number,
```

19. Look at the preview; you should see the prices are now either Integer or Double objects.

Output Payload

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : Integer	400
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX
price : Double	199.99
plane : String	BOING 737

```
1%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number,
8   plane: upper $.planeType
9 }
10
11
12
```

Coerce a string to a specific type of number object

20. Change the DataWeave expression to use the class metadata key to coerce the prices to java.lang.Double objects.

```
price: $.price as :number {class:"java.lang.Double"},
```

21. Look at the preview; you should see the prices are now all Double objects.

The screenshot shows the DataWeave editor and its preview pane. The code is:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 @flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number {class:"java.lang.Double"}, // Coerced to Double
8   plane: upper $.planeType
9 }
10
11
12
```

The preview pane shows the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.0
plane	BOING 737
[1]	LinkedHashMap
dest	LAX
price	199.99

Format a number

22. Use the format schema property in the DataWeave expression to format the prices to zero decimal places.

23. Remove the coercion to a java.lang.Double.

```
price: $.price as :number as :string {format: "###"},
```

24. Look at the preview; the prices should now be formatted.

The screenshot shows the DataWeave editor and its preview pane. The code is identical to the previous one:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 @flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###"}, // Formatted to String
8   plane: upper $.planeType
9 }
10
11
```

The preview pane shows the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400
plane	BOING 737
[1]	LinkedHashMap
dest	LAX
price	200

25. Change the DataWeave expression to format the prices to two decimal places.

```
price: $.price as :number as :string {format: "##.##"},
```

26. Look at the preview; the prices should be formatted differently.

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX
price : String	199.99

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.##"},
8   plane: upper $.planeType
9 }
10
11
```

27. Change the DataWeave expression to format the prices to two minimal decimal places.

```
price: $.price as :number as :string {format: "###.00"},
```

28. Look at the preview; all the prices should be formatted to two decimal places.

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.00
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX
price : String	199.99

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"}, // Modified here
8   plane: upper $.planeType
9 }
10
11
12
```

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

29. Add a date field to the return object.

```
date: $.departureDate
```

30. 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 preview pane. The DataWeave code is:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"}, 
8   plane: upper $.planeType,
9   date: $.departureDate
10 }
```

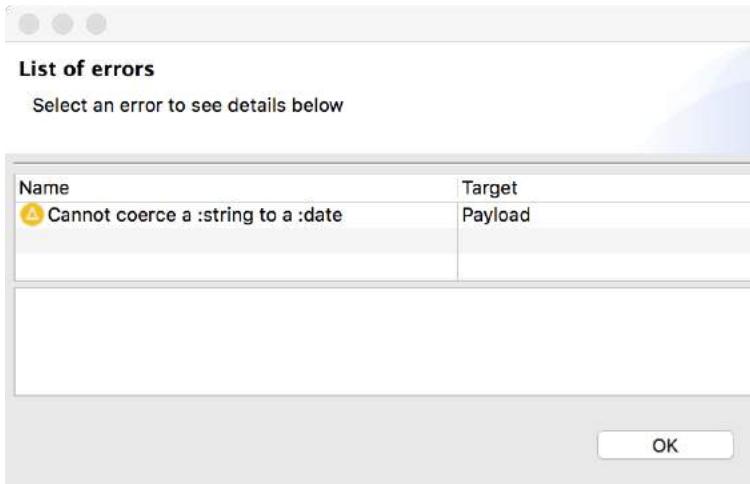
The preview pane shows the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOING 737
date	2015/10/20
[1]	LinkedHashMap

31. Change the DataWeave expression to use the as operator to convert departureDate to a date object; you should get an exception.

```
date: $.departureDate as :date
```

32. Look at the exception.



33. Look at the format of the dates in the preview.

34. Change the DataWeave expression to use the format schema property to specify the pattern of the input date strings.

```
date: $.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 DateFormat class to review documentation on pattern letters.

<https://docs.oracle.com/javase/8/docs/api/java/text/format/DateFormat.html>

35. Look at the preview; you should see date is now a Date object.

The screenshot shows the Mule Studio interface with the 'Preview' tab selected. On the left is the MEL expression:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"},
8   plane: upper $.planeType,
9   date: $.departureDate as :date {format: "yyyy/MM/dd"}
10 }
```

On the right is the preview pane showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOING 737
date	Tue Oct 20 00:00:00 PDT 2015
[1]	LinkedHashMap

Format a date

36. Use the as operator to convert the dates to strings, which can then be formatted.

```
date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
```

37. Look at the preview section; the date is again a String – but with a different format.

The screenshot shows the Mule Studio interface with the 'Preview' tab selected. On the left is the MEL expression:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"},
8   plane: upper $.planeType,
9   date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
10 }
```

On the right is the preview pane showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOING 737
date	2015-10-20
[1]	LinkedHashMap

38. Use the format schema property with any pattern letters and characters to format the date strings.

```
date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
{format: "MMM dd, yyyy"}
```

39. Look at the preview; the dates should now be formatted according to the pattern.

The screenshot shows the Mule Studio interface with the 'Preview' tab selected. On the left is the MEL expression:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"},
8   plane: upper $.planeType,
9@   date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
10  {format: "MMM dd, yyyy"}
11 }
12
13
14
```

On the right is the preview pane showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOING 737
date	Oct 20, 2015
[1]	LinkedHashMap
dest	LAX
price	199.99
plane	BOING 737
date	Oct 21, 2015

Walkthrough 10-6: Use DataWeave operators

In this walkthrough, you continue to work with the flights JSON posted to the flow. You will:

- Replace data values using pattern matching.
- Order data, filter data, and remove duplicate data.

The screenshot shows the DataWeave editor and its preview pane. The DataWeave code is as follows:

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"},
8   plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9@   date: $.departureDate as :date {format: "yyyy/MM/dd"}
10  as :string {format: "MMM dd, yyyy"},
11  seats: $.emptySeats as :number
12 } orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)
13
14
```

The preview pane shows the resulting flight data:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
[3]	LinkedHashMap
dest	SFO
price	400.00
plane	BOEING 737
date	Oct 20, 2015
seats	40

Replace data values

- Return to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
- Use the replace operator in the DataWeave expression to replace the string Boing with Boeing.
plane: upper \$.planeType replace /(BOING)/ with "BOEING",
- Look at the preview section; Boeing should not be spelled correctly.
- In the DataWeave expression, place parentheses around the replace operator expression.
- Look at the preview section; Boeing should now be spelled correctly.

The screenshot shows the DataWeave editor and its preview pane. The DataWeave code has been modified to include parentheses around the replace operator:

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"},
8   plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9@   date: $.departureDate as :date {format: "yyyy/MM/dd"}
10  as :string {format: "MMM dd, yyyy"}
11 }
```

The preview pane shows the resulting flight data, where the plane value is now correctly spelled "Boeing".

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	Boeing
date	Oct 20, 2015
[1]	LinkedHashMap
dest	LAX

Order data

- In the preview section, look at the flight prices; the flights should not be ordered by price.

7. Change the DataWeave expression to use the orderBy operator to order the objects by price.

```
payload.listAllFlightsResponse.*return map {
    ...
} orderBy $.price
```

8. Look at the preview; the flights should now be ordered by price.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###.00"}, 
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9@     date: $.departureDate as :date {format: "yyyy/MM/dd"}
10    as :string {format: "MMM dd, yyyy"}
11 } orderBy $.price
12
13
14
```

To the right is a preview table showing two flight records:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	LAX
price	199.99
plane	BOEING 737
date	Oct 21, 2015
[1]	LinkedHashMap
dest	PDX
price	283.00

9. Look at the PDX flights; they should not be ordered by date.

10. Change the DataWeave expression to first sort by date and then by price.

```
payload.listAllFlightsResponse.*return map {
    ...
} orderBy $.date orderBy $.price
```

11. 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 same code as the previous preview, but the preview table now shows the flights sorted by date:

Name	Value
[0]	Oct 20, 2015
[2]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 20, 2015
[3]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 21, 2015

Remove duplicate data

12. Use the distinctBy operator in the DataWeave expression to remove any duplicate objects.

```
payload.listAllFlightsResponse.*return map {  
    ...  
} orderBy $.date orderBy $.price distinctBy $
```

13. Look at the preview; you should now get only four flights instead of five.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left is the DataWeave editor with the following code:

```
1@%dw 1.0  
2 %output application/java  
3 %namespace ns0 http://soap.training.mulesoft.com/  
4 ---  
5@ flights: payload.ns0:listAllFlightsResponse.*return map {  
6     dest: $.destination,  
7     price: $.price as :number as :string {format: "###.00"},  
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),  
9@     date: $.departureDate as :date {format: "yyyy/MM/dd"}  
10    as :string {format: "MMM dd, yyyy"}  
11 } orderBy $.date orderBy $.price distinctBy $  
12  
13  
14
```

On the right is a preview table showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 20, 2015
[2]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 21, 2015
[3]	LinkedHashMap

14. Add an seats field that is equal to the emptySeats field and coerce it to a number.

```
seats: $.emptySeats as :number
```

15. Look at the preview section; you should get five flights again.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left is the DataWeave editor with the following code:

```
1@%dw 1.0  
2 %output application/java  
3 %namespace ns0 http://soap.training.mulesoft.com/  
4 ---  
5@ flights: payload.ns0:listAllFlightsResponse.*return map {  
6     dest: $.destination,  
7     price: $.price as :number as :string {format: "###.00"},  
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),  
9@     date: $.departureDate as :date {format: "yyyy/MM/dd"}  
10    as :string {format: "MMM dd, yyyy"},  
11    seats: $.emptySeats as :number  
12 } orderBy $.date orderBy $.price distinctBy $  
13  
14  
15
```

On the right is a preview table showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 20, 2015
seats	23
[3]	LinkedHashMap
[4]	LinkedHashMap

Filter data

16. In the preview section, look at the values of the seats properties; ou should see some are equal to zero.

17. Add a filter to the DataWeave expression that removes any objects that have availableSeats equal to 0.

```
payload.listAllFlightsResponse.*return map {  
    ...  
} orderBy $.departureDate orderBy $.price distinctBy $  
    filter ($.seats !=0)
```

18. Look at the preview; you should no longer get the flight that had no available seats.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left is the DataWeave editor with the following code:

```
1@ %dw 1.0  
2 %output application/java  
3 %namespace ns0 http://soap.training.mulesoft.com/  
4 ---  
5@ flights: payload.ns0:listAllFlightsResponse.*return map {  
6     dest: $.destination,  
7     price: $.price as :number as :string {format: "###.00"},  
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),  
9     date: $.departureDate as :date {format: "yyyy/MM/dd"},  
10    as :string {format: "MMM dd, yyyy"},  
11    seats: $.emptySeats as :number  
12 } orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)  
13  
14
```

On the right is a preview table showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
[3]	LinkedHashMap
dest	SFO
price	400.00
plane	BOEING 737
date	Oct 20, 2015
seats	40

Walkthrough 10-7: 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.

The screenshot shows the Mule Studio interface with a DataWeave editor and a preview pane. The DataWeave code defines a custom data type 'currency' and uses it in a transformation to map flight data to a POJO structure. The preview pane shows the resulting POJO object with five flight items, each containing details like airlineName, availableSeats, departureDate, destination, flightCode, origination, planeType, and price.

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.00"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7@flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12    as :string {format: "MMM dd, yyyy"},
13    availableSeats: $.emptySeats as :number
14 } as :flight
15
16
```

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	Flight
airlineName	String
availableSeats	Integer 40
departureDate	Date Oct 20, 2015
destination	String SFO
flightCode	String
origination	String
planeType	String BOEING 737
price	Double 400.0
[1]	Flight
[2]	Flight
[3]	Flight
[4]	Flight

Define a custom data type

1. Return to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
2. Select and cut the string formatting expression for the prices.
3. In the header section of the transform section, define a custom data type called currency.
4. Set it equal to the value you copied.
5. Remove the as operator.

```
%type currency = :string {format: "###.00"}
```

```
1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.00"}
5 ---
```

Use a custom data type

6. In the DataWeave expression, set the price to be of type currency.

```
price: $.price as :number as :currency,
```

7. Look at the preview; the prices should still be formatted as strings to two decimal places.

The screenshot shows the Mule Studio interface with a DataWeave script and a preview table.

```

1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 ---
6 flights: payload.ns0:listAllFlightsResponse.*return map {
7     dest: $.destination,
8     price: $.price as :number as :currency,
9     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
10    date: $.departureDate as :date {format: "yyyy/MM/dd"},
11    as :string {format: "MMM dd, yyyy"}, 
12    seats: $.emptySeats as :number
13 } orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)
  
```

Preview:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	LAX
price	199.99
plane	BOEING 737
date	Oct 21, 2015
seats	10
[1]	LinkedHashMap
dest	PDX
price	283.00

Transform objects to POJOs

8. Open the Flight.java class in the project's src/main/java folder and look at the names of the properties.

The screenshot shows the Java code editor with the Flight.java file open.

```

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    }
  
```

9. Return to postMultipleFlightsFlow in implementation.xml.
 10. In the header section of the transform section, 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 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
```

11. In the DataWeave expression, set the map objects to be of type flight.

```
payload.listAllFlightsResponse.*return map {  
    ...  
} orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)  
as :flight
```

12. Change the name of the dest key to destination.

```
destination: $.destination,
```

13. Change the other keys to match the names of the Flight class properties:

- plane to planeType
- date to departureDate
- seats to availableSeats

14. Change the operators to use the new property names.

```
payload.listAllFlightsResponse.*return map {  
    ...  
} orderBy $.departureDate orderBy $.price distinctBy $ filter  
($.availableSeats !=0) as :flight  
  
1@%dw 1.0  
2 %output application/java  
3 %namespace ns0 http://soap.training.mulesoft.com/  
4 %type currency = :string {format: "###.00"}  
5 %type flight = :object {class: "com.mulesoft.training.Flight"}  
6 ---  
7@ flights: payload.ns0#listAllFlightsResponse.*return map {  
8     destination: $.destination,  
9     price: $.price as :number as :currency,  
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),  
11@    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}  
12    as :string {format: "MMM dd, yyyy"},  
13    availableSeats: $.emptySeats as :number  
14@ } orderBy $.departureDate orderBy $.price distinctBy $  
15 filter ($.availableSeats !=0) as :flight
```

15. Look at the preview; you should now get an ArrayList of Flight objects.

16. If you get an exception instead; delete the operators after the map.

```
payload.listAllFlightsResponse.*return map {  
    ...  
} as :flight
```

Note: There is a bug in Mule 3.8.0 with custom objects.

17. Look at the preview; you should now get an ArrayList of Flight objects.

Output Payload ▾  

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7@ flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11@   departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12    as :string {format: "MMM dd, yyyy"},
13    availableSeats: $.emptySeats as :number
14 } as :flight
15
16
```

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	Flight
airlineName	String
availableSeats	Integer
departureDate	String
destination	SFO
flightCode	String
origination	String
planeType	String
price	Double
[1]	Flight
[2]	Flight
[3]	Flight
[4]	Flight

Walkthrough 10-8: Call MEL functions and other flows

In this walkthrough, you continue to work with the DataWeave transformation in getMultipleFlightsFlow. You will:

- Define a global MEL function in global.xml.
- Call a global MEL function in a DataWeave expression.
- Call a flow in a DataWeave expression.

```
//totalSeats: getNumSeats($.planeType)
totalSeats: lookup("getTotalSeatsFlow", {type: $.planeType})
```

The screenshot shows the Anypoint Studio interface. On the left, the 'global' configuration file is open, displaying XML code for a consumer configuration. On the right, the 'Expression' editor is open for the 'getTotalSeatsFlow' expression. The editor shows the expression code: `if (payload.type.contains('737')){ payload = 150; } else { payload = 300; }`. The 'General' tab is selected in the sidebar, and the status bar indicates 'There are no errors.'

Define a global MEL function in global.xml

1. Return to global.xml and switch to the Configuration XML view.
2. Locate the configuration element.
3. On a new line between the configuration tags, type <e and the press Ctrl+Space for autocompletion; an expression-language tag set should be added.

```
17    <ws:consumer-config name="Delta_Web_Service_Consumer" service="TicketServiceService"
18      <configuration defaultExceptionStrategy-ref="implementationChoice_Exception_Strateg
19        <http:config useTransportForUris="false"/>
20          <expression-language></expression-language>
21      </configuration>
22      <payload-type-filter expectedType="java.util.ArrayList" name="Filter_Not_ArrayList"
```

4. Place the expression-language tags on different lines and place the cursor between them.
5. Type <g, press Ctrl+Space, and select global-functions; a global-functions tag set should be added.

6. Place the global-functions tags on different lines and place the cursor between them.

```
18<configuration defaultExceptionStrategy-ref="implementationChoice_Exception">
19    <http:config useTransportForUris="false"/>
20    <expression-language>
21        <global-functions>
22
23        </global-functions>
24    </expression-language>
25 </configuration>
```

7. Use the def keyword to define a function called getNumSeats with an argument called type.

```
def getNumSeats(type){}
```

```
18<configuration defaultExceptionStrategy-ref="implementationChoice_Exception">
19    <http:config useTransportForUris="false"/>
20    <expression-language>
21        <global-functions>
22            def getNumSeats(type){
23
24        }
25        </global-functions>
26    </expression-language>
27 </configuration>
```

8. Inside the function, add an if/else block that checks to see if type contains the string 737.

```
if (type.contains('737')){}
```

9. If the expression is true, return 150 and if false, return 300.

```
if (type.contains('737')){
    return 150;
} else {
    return 300;
}
```

10. Save the file.

11. Run the project; the application should fail to deploy.

12. Locate the exception in the console.

```
ERROR 2016-06-07 19:53:03,383 [main] org.mule.module.launcher.application.DefaultMuleApplication: null
org.xml.sax.SAXParseException: cvc-complex-type.2.4.a: Invalid content was found starting with element
'expression-language'. One of '{"http://www.mulesoft.org/schema/mule/core":abstract-configuration-extension}' is expected.
at org.apache.xerces.util.ErrorHandlerWrapper.createSAXParseException(Unknown Source) ~[?:?]
at org.apache.xerces.util.ErrorHandlerWrapper.error(Unknown Source) ~[?:?]
```

13. Return to global.xml and delete the http:config tag inside the configuration tag set.

```
<http:config useTransportForUris="false"/>
```



```
17 <ws:consumer-config name="Delta_Web_Service_Consumer" service="T
18 <configuration defaultExceptionStrategy-ref="implementationChoice
19 <expression-language>
20 <global-functions>
21     def getNumSeats(type){
22         if (type.contains('737')){
23             return 150;
24         } else {
25             return 300;
26         }
27     }
28 </global-functions>
29 </expression-language>
30 </configuration>
31 <onload-type-filter expectedType="java.util.ArrayList" name="Fil
```

14. Run the project; the application should deploy.

15. Stop the project.

Call a global MEL function from DataWeave

16. Return to postMultipleFlightsFlow in implementation.xml.

17. In the Transform Message properties view, add a field called totalSeats inside the transformation function.

18. Set totalSeats equal to the return value from the getNumSeats() function.

19. Pass to getNumSeats(), the planeType.

```
totalSeats: getNumSeats($.planeType)
```

20. Look at the preview; you should see the Flight objects now have a totalSeats property equal to 150 or 300.

21. If you get an exception, remove the coercion to a Flight object.

Note: There is a bug in Mule 3.8.0 with custom objects.

22. Look at the preview; you should see the Flight objects now have a totalSeats property equal to 150 or 300.

```

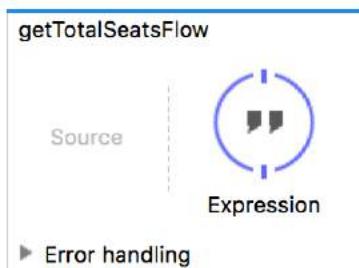
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7 @flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"},
12    as :string {format: "MM dd, yyyy"},
13    availableSeats: $.emptySeats as :number,
14    totalSeats: getNumSeats($.planeType)
15 }
16
17

```

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
destination	SFO
price	400.00
planeType	BOEING 737
departureDate	Oct 20, 2015
availableSeats	40
totalSeats	150
[1]	LinkedHashMap
[2]	LinkedHashMap
destination	PDX
price	283.00
planeType	BOEING 777
departureDate	Oct 21, 2015
availableSeats	30
totalSeats	300

Create a lookup flow

23. Return to implementation.xml.
 24. Drag an Expression component (not a filter or a transformer!) from the Mule Palette and drop it at the bottom of the canvas to create a new flow.
 25. Change the name of the flow to getTotalSeatsFlow.



26. Return to global.xml and copy the if/else block.
 27. Return to implementation.xml.
 28. In the Expression properties view, paste the if/else block you copied into the expression text area.
 29. In the expression, change type to payload.type.

```
if (payload.type.contains('737')){
```

30. Change the two return statements to set the payload with the values instead.

```
if (payload.type.contains('737')){  
    payload = 150;  
} else {  
    payload = 300;  
}
```

The screenshot shows the Mule Studio interface with the 'Expression' tab selected in the top navigation bar. Below the tabs are three buttons: 'Problems', 'Console', and 'Mule Debugger'. A green checkmark icon indicates 'There are no errors.' On the left, there's a sidebar with 'General' selected, followed by 'Notes' and 'Metadata'. In the main area, under 'Display Name:', the value 'Expression' is entered. Under 'Settings', the 'Expression:' radio button is selected, and the code block contains the provided DataWeave script.

Call a flow from a DataWeave expression

31. Return to the Transform Message properties view of the component in postMultipleFlightsFlow.
32. Comment out the existing totalSeats assignment by placing // in front of it.

```
//totalSeats: getNumSeats($.planeType)
```

33. Add another property called totalSeats inside the transformation function.
34. Set totalSeats equal to the return value from the DataWeave lookup() function.

```
totalSeats: lookup()
```

35. Pass to lookup() an argument that is equal to the name of the flow to call: "getTotalSeatsFlow".
36. Pass an object to lookup() as the second argument.
37. Give the object a field called plane (to match what the flow is expecting) and set it equal to the value of the planeType field.

```
totalSeats: lookup("getTotalSeatsFlow", {type: $.planeType})
```

38. Look at the preview; you should not see the totalSeats because its value can only be evaluating by calling the flow at runtime.

The screenshot shows the Mule Studio interface. On the left, there is a code editor window containing MEL (Mule Expression Language) code. The code is as follows:

```

1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object [class: "com.mulesoft.training.Flight"]
6 ---
7@ flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12    as :string {format: "MMM dd, yyyy"},
13    availableSeats: $.emptySeats as :number,
14    //totalSeats: getNumSeats($.planeType)
15    totalSeats: lookup("getTotalSeatsFlow", {type: $.planeType})
16 }

```

On the right, there is a preview pane showing the resulting data structure. The preview pane has two columns: 'Name' and 'Value'. The data is represented as a linked hash map:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
destination	SFO
price	400.00
planeType	BOEING 737
departureDate	Oct 20, 2015
availableSeats	40
Unknown	
[1]	LinkedHashMap
destination	LAX
price	199.99
planeType	BOEING 737

Test the application

39. Change the output type to application/json.
40. Run the project.
41. In Postman, make sure the method is set to POST and the request URL is set to localhost:8081/multipleflights.
42. Set a Content-Type header to application/xml.
43. Set the request body to the value contained in the flights-example.xml file.

The screenshot shows the Postman application interface. The request details are as follows:

- Method:** POST
- URL:** localhost:8081/multipleflights
- Headers:** (1)
- Body:** (selected)
- Content-Type:** XML (application/xml)

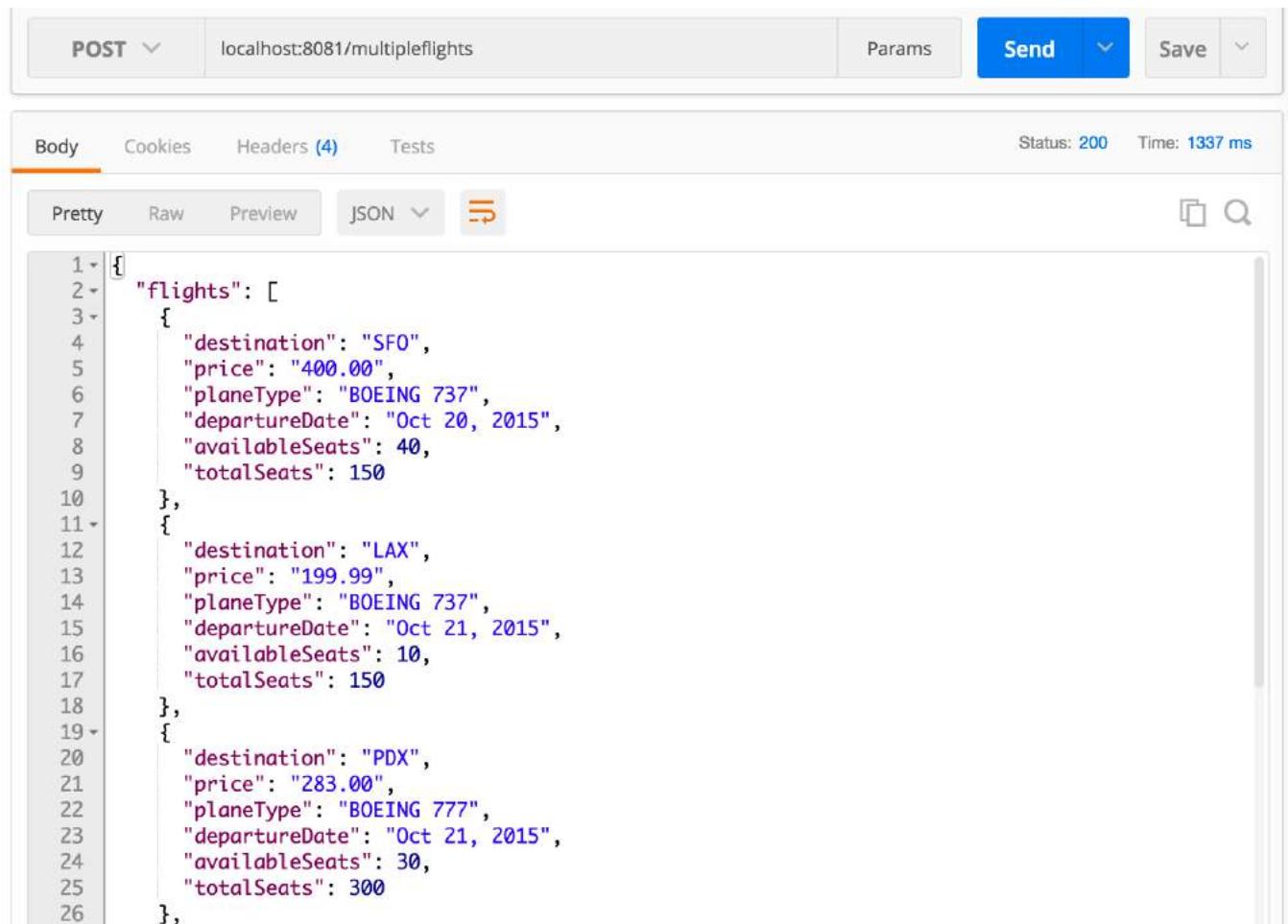
The XML body is defined as:

```

1<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
2    <return airlineName="United">
3        <code>A1B2C3</code><departureDate>2015/10/20</departureDate>
4        <destination>SFO</destination><emptySeats>40</emptySeats>
5        <origin>MUA</origin>
6        <planeType>Boing 737</planeType>
7        <price>400.0</price>
8    </return>
9    <return airlineName="Delta">
10       <code>A1B2C4</code>

```

44. Send the request; you should get JSON flight data returned and each flight should have a totalSeats property equal to 150 or 300.



```
1+ [
2+   "flights": [
3+     {
4+       "destination": "SFO",
5+       "price": "400.00",
6+       "planeType": "BOEING 737",
7+       "departureDate": "Oct 20, 2015",
8+       "availableSeats": 40,
9+       "totalSeats": 150
10+      },
11+      {
12+        "destination": "LAX",
13+        "price": "199.99",
14+        "planeType": "BOEING 737",
15+        "departureDate": "Oct 21, 2015",
16+        "availableSeats": 10,
17+        "totalSeats": 150
18+      },
19+      {
20+        "destination": "PDX",
21+        "price": "283.00",
22+        "planeType": "BOEING 777",
23+        "departureDate": "Oct 21, 2015",
24+        "availableSeats": 30,
25+        "totalSeats": 300
26+      }
]
```

45. Stop the project.

46. Close the project.

Module 11: Connecting to Additional Resources



Objectives:

- Connect to SaaS applications.
- Connect to files.
- Poll resources.
- Connect to JMS queues.
- Discover and install connectors not bundled with Anypoint Studio.

Walkthrough 11-1: Connect to a SaaS application (Salesforce)

In this walkthrough, you retrieve account records from Salesforce. You will:

- Browse Salesforce data on <http://salesforce.com>.
- Add and configure a Salesforce connector.
- Add a Salesforce endpoint to retrieve accounts for a specific postal code.
- Use the Query Builder to write a query.

The screenshot shows the MuleSoft Anypoint Studio interface. At the top, there's a navigation bar with links like Home, Chatter, Campaigns, Leads, Accounts, Contacts, Opportunities, Forecasts, Contracts, Orders, Cases, Solutions, and Help. Below the navigation bar is a banner for the Salesforce mobile app. The main workspace displays a Mule flow named 'getSFDCAccountsFlow'. The flow starts with an 'HTTP' component, followed by a 'Salesforce' connector, a 'Transform Message' component containing a JSON payload, and finally a 'Logger' component. The 'Transform Message' component's configuration shows a JSON response with fields such as 'BillingCountry', 'BillingCity', 'BillingStreet', 'BillingPostalCode', 'Id', 'Type', 'BillingState', and 'Name'. The JSON payload is as follows:

```
1. {
2.   "BillingCountry": "USA",
3.   "BillingCity": "Washington",
4.   "BillingStreet": "325 5. Lexington Ave",
5.   "BillingPostalCode": "27215",
6.   "Id": null,
7.   "Type": "Account",
8.   "BillingState": "NC",
9.   "Name": "Burlington Textiles Corp of America"
10. }
```

Note: To complete this walkthrough, you need a Salesforce Developer account. Instructions for creating a Salesforce developer account and getting an API access token are included in the first walkthrough at the beginning of this student manual.

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.

The screenshot shows the Salesforce login page. At the top, there's a navigation bar with links like Home, Chatter, Campaigns, Leads, Accounts, Contacts, Opportunities, Forecasts, and Contracts. The 'Accounts' link is highlighted in blue. Below the navigation bar is a banner for the Salesforce mobile app. The main content area shows a list of recent items, with 'Bens Books' being the most recent. There's also a 'Recent Accounts' section with a 'New' button.

- Look at the existing account data; a Salesforce Developer account is populated with some sample data.

Action	Account Name	Billing State/Prov...	Phone
Edit Del +	Burlington Textiles...	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	Add	Account Name	Top
Shipping Zip/Postal Code		Billing Zip/Postal Code	Up
Shipping Country		Billing Country	Down
Fax		Billing State/Province	Bottom
Website		Account Number	
Employees		Phone	
D&B Company		Type	
Data.com Key			
SIC Code			
SIC Description			
NAICS Code			
NAICS Description			
D-U-N-S Number			
Tradestyle			
Year Started			

- 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	

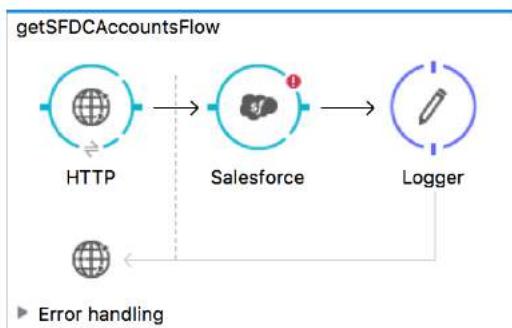
Review the starting flow in the examples project

13. Return to Anypoint Studio.
14. Open the apdev-examples project.
15. Open accounts.xml.
16. For the HTTP Listener endpoint in getSFDCAccountsFlow, see the path is set to /sfdc.

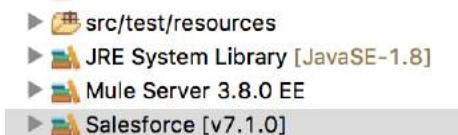


Add a Salesforce endpoint

17. Drag out a Salesforce connector and drop it before the Logger.



18. In the Package Explorer, locate the new Salesforce library that was added.

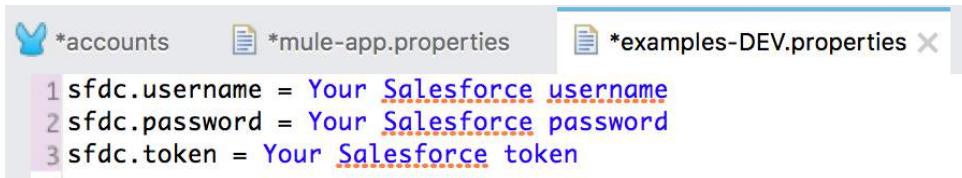


Configure the Salesforce connector

19. Open mule-app.properties in src/main/app.
20. Set an environment variable called env equal to DEV.



21. In the Package Explorer, right-click src/main/resources and select New > File.
22. In the new File dialog box, set the file name to examples-DEV.properties and click Finish.
23. In examples-DEV.properties, create properties for your Salesforce username, password, and security token.



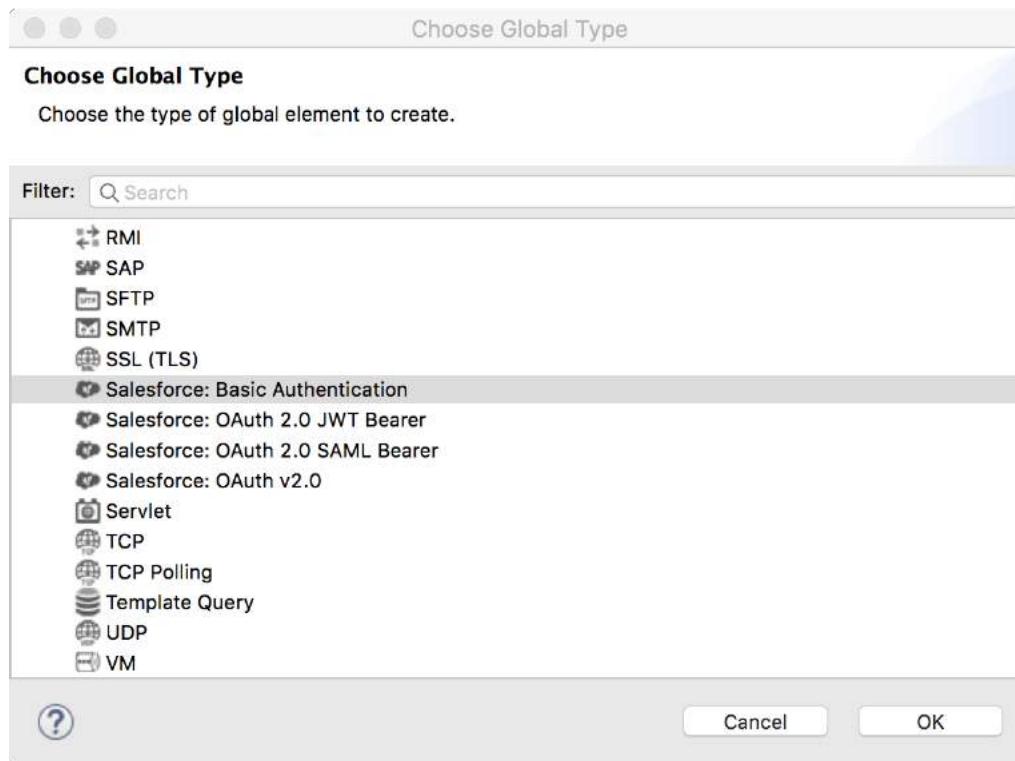
```

*accounts          *mule-app.properties      *examples-DEV.properties X
1 sfdc.username = Your Salesforce username
2 sfdc.password = Your Salesforce password
3 sfdc.token = Your Salesforce token

```

Note: Instructions for creating a Salesforce Developer account and getting a security token are included in the first walkthrough at the beginning of this student manual.

24. Save all the files.
25. Return to the Global Elements view of global.xml.
26. Click Create.
27. In the Choose Global Type dialog box, select Component configurations > Property Placeholder and click OK.
28. Set the location to examples-\${env}.properties and click OK.
29. Click Create.
30. In the Choose Global Type dialog box, select Connector Configuration > Salesforce: Basic Authentication and click OK.



31. In the Global Element Properties dialog box, set the following values and click OK.

- Username: \${sfdc.username}
- Password: \${sfdc.password}
- Security Token: \${sfdc.token}

32. Click the Validate configuration button; you will get a Test Connection dialog box letting you know if the connection succeeds or fails.



33. Click OK to close the Test connection dialog box.

34. If your test connection failed, fix it; do not proceed until it is working.

Note: If it failed, check to see if you have any extra whitespace in your entries.

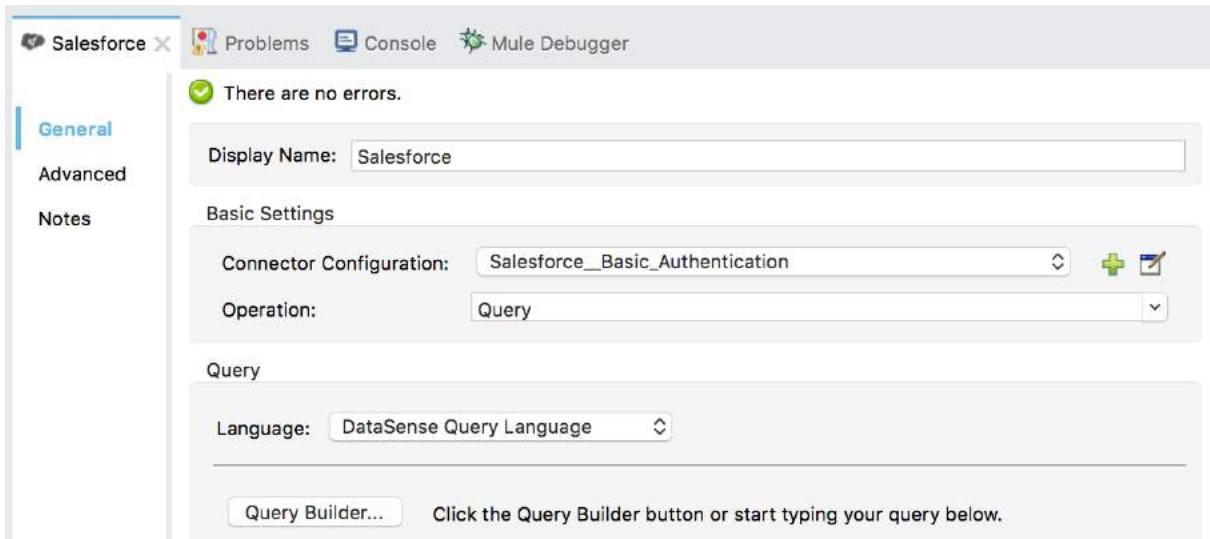
35. Click OK to close the Global Elements Properties dialog box.

Configure the Salesforce endpoint

36. Return to accounts.xml.

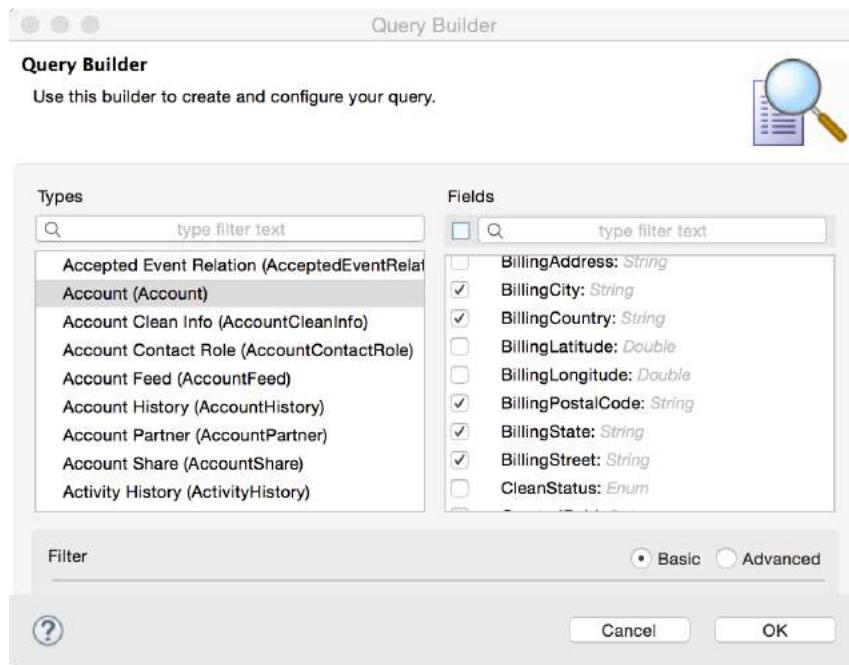
37. In the Salesforce properties view, set the connector configuration to the existing Salesforce__Basic_Authentication.

38. Set the operation to Query.



Write the query

39. Click the Query Builder button.
40. In the Query Builder dialog box, select a type of Account (Account).
41. Select fields BillingCity, BillingCountry, BillingPostalCode, BillingState, BillingStreet, and Name.



42. Click OK.
43. In the Salesforce properties view, examine the generated query.

The screenshot shows the 'Query Text:' field containing the following SQL-like query:
1 SELECT BillingCity,BillingCountry,BillingPostalCode,BillingState,BillingStreet,Name
FROM Account

Test the application

44. Run the apdev-examples project.
45. In Postman, make a request to <http://localhost:8081/sfdc>; you should see the type of Java object created.

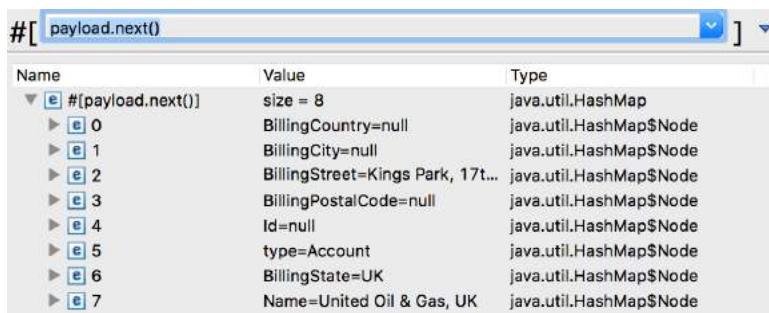
The screenshot shows a Postman request. The URL is 'localhost:8081/sfdc'. The response status is 200 and time is 1072 ms. The body of the response is a single line of text: 'org.mule.streaming.ConsumerIterator@73eccb2f'.

Debug the application

46. Return to accounts.xml.
47. Stop the project.
48. Add a breakpoint to the Logger.
49. Debug the project.
50. In Postman, make another request to <http://localhost:8081/sfdc>.
51. In the Mule Debugger, drill-down into the payload variable.
52. Click the Evaluate Mule Expression button.
53. Enter the following expression and press the Enter key.

```
##[payload.next()]
```

54. Expand the results; you should see the account data for the first account.



Name	Value	Type
#[payload.next()]	size = 8	java.util.HashMap
#e 0	BillingCountry=null	java.util.HashMap\$Node
#e 1	BillingCity=null	java.util.HashMap\$Node
#e 2	BillingStreet=Kings Park, 17t...	java.util.HashMap\$Node
#e 3	BillingPostalCode=null	java.util.HashMap\$Node
#e 4	Id=null	java.util.HashMap\$Node
#e 5	type=Account	java.util.HashMap\$Node
#e 6	BillingState=UK	java.util.HashMap\$Node
#e 7	Name=United Oil & Gas, UK	java.util.HashMap\$Node

55. Click anywhere outside the expression window to close it.
56. Stop the project.

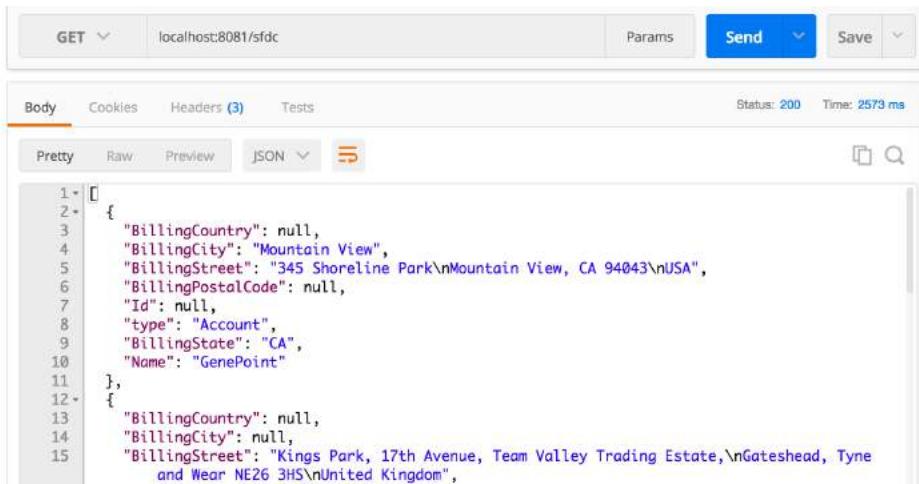
Display the return data

57. Add a Transform Message component before the Logger component.
58. In the Transform Message properties view, look at the input metadata.
59. In the output expression section, change the output type to application/json.
60. Set the DataWeave expression to payload.



Test the application

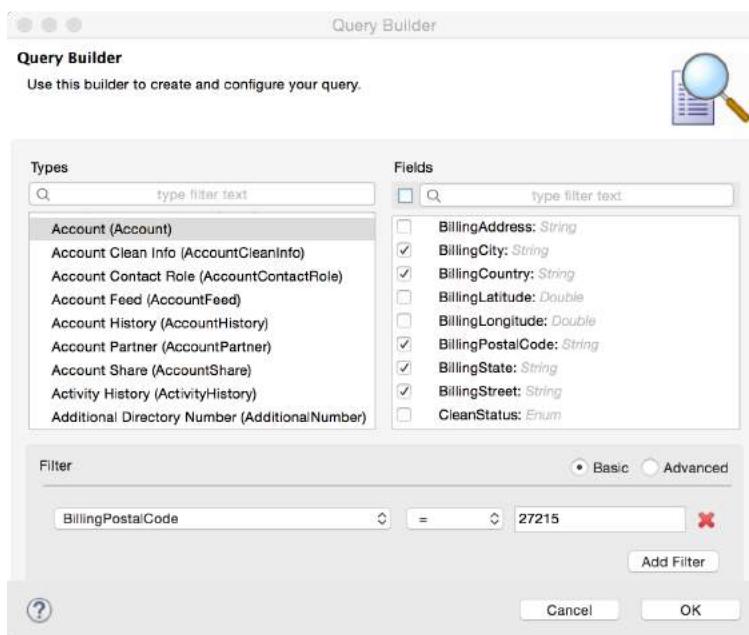
61. Run the project.
62. In Postman, make another request to <http://localhost:8081/sfdc>; you should see the Salesforce accounts displayed.



```
1. {
2.   "BillingCountry": null,
3.   "BillingCity": "Mountain View",
4.   "BillingStreet": "345 Shoreline Park\nMountain View, CA 94043\nUSA",
5.   "BillingPostalCode": null,
6.   "Id": null,
7.   "type": "Account",
8.   "BillingState": "CA",
9.   "Name": "GenePoint"
10. },
11. [
12.   {
13.     "BillingCountry": null,
14.     "BillingCity": null,
15.     "BillingStreet": "Kings Park, 17th Avenue, Team Valley Trading Estate,\nGateshead, Tyne and Wear NE26 3HS\nUnited Kingdom",
16.   }
17. ]
```

Modify the query

63. Return to accounts.xml.
64. In the Salesforce Properties view, click the Query Builder button.
65. In the Query Builder dialog box, click the Add Filter button.
66. Create a filter for BillingPostalCode equal to one of the postal codes (like 27215) in your Salesforce account data.



67. Click OK.

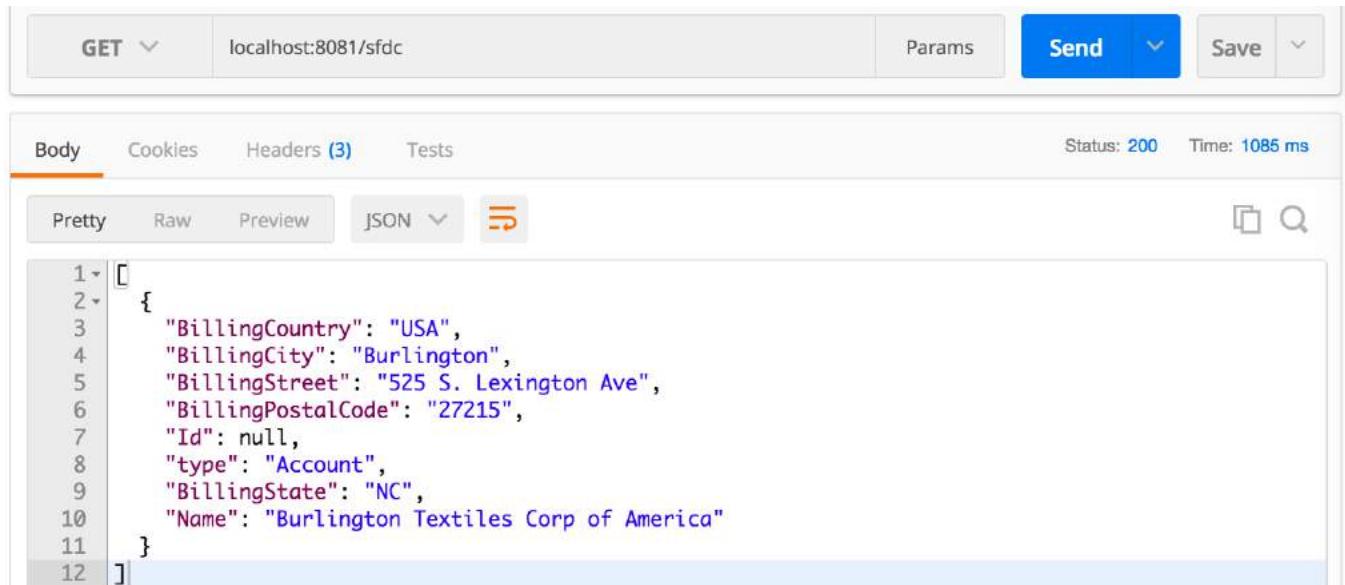
68. Examine the generated query.

```
Query Text:  
1 SELECT BillingCity,BillingCountry,BillingPostalCode,BillingState,BillingStreet,Name  
2 FROM Account  
3 WHERE BillingPostalCode = '27215'
```

Test the application

69. Save the file to redeploy the application.

70. In Postman, make another request to <http://localhost:8081/sfdc>; you should see only the accounts with the specified postal code displayed.



The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: localhost:8081/sfdc
- Params: None
- Send button: Blue button labeled "Send" with a dropdown arrow.
- Save button: Grey button labeled "Save" with a dropdown arrow.
- Body tab selected: Shows the JSON response.
- Cookies tab: None.
- Headers tab: (3) entries.
- Tests tab: None.
- Status: 200
- Time: 1085 ms
- Pretty tab: Selected.
- Raw tab: Unselected.
- Preview tab: Unselected.
- JSON tab: Selected.
- Copy icon: Copy to clipboard icon.
- Search icon: Search icon.
- Response Body (Pretty JSON):

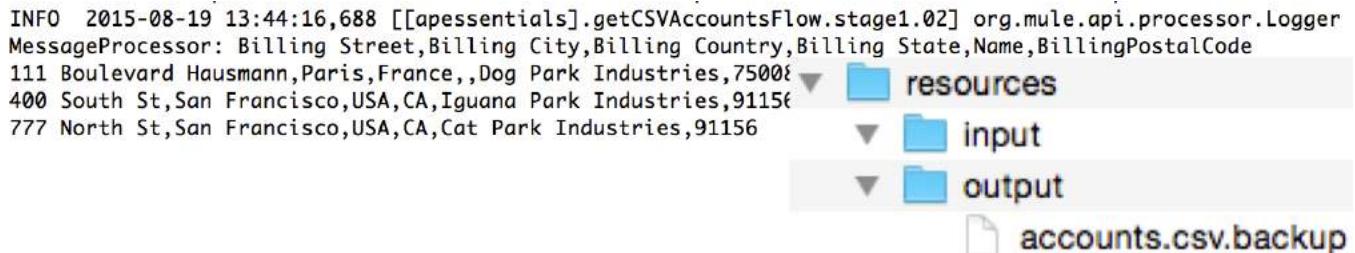
```
1 [  
2 {  
3   "BillingCountry": "USA",  
4   "BillingCity": "Burlington",  
5   "BillingStreet": "525 S. Lexington Ave",  
6   "BillingPostalCode": "27215",  
7   "Id": null,  
8   "type": "Account",  
9   "BillingState": "NC",  
10  "Name": "Burlington Textiles Corp of America"  
11 }]  
12 ]
```

71. Return to Anypoint Studio and stop the project.

Walkthrough 11-2: Connect to a file (CSV)

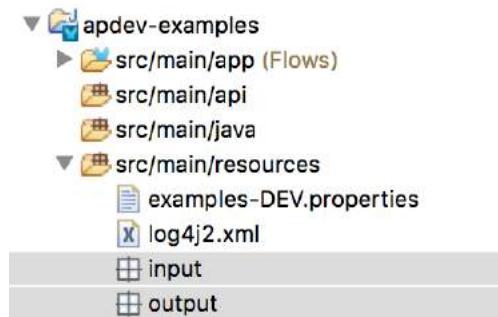
In this walkthrough, you load data from a local CSV file. You will:

- Add and configure a File endpoint to watch an input directory, read the contents of any added files, and then rename and move the files.
- Use DataWeave to convert a CSV file to a string.
- Add a CSV file to the input directory and watch it renamed and moved.
- Restrict the type of file read.
- Add metadata to a file endpoint.



Create new directories

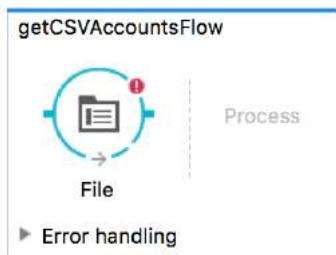
1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > Folder.
3. Set the folder name to input and click Finish.
4. Create a second folder called output.



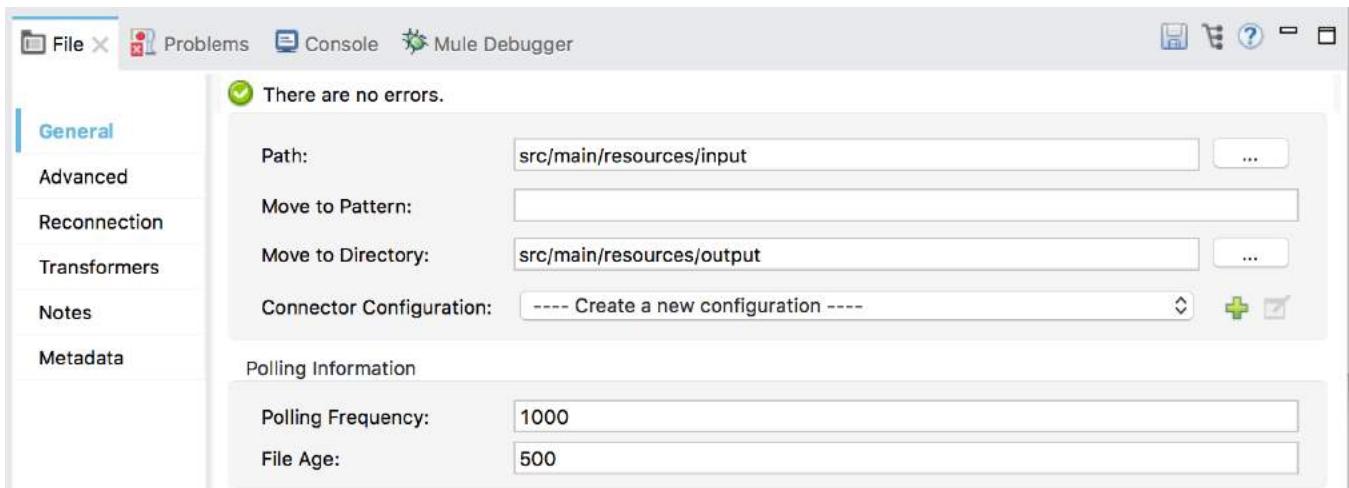
Add and configure a File endpoint

5. Return to accounts.xml.
6. Drag a File connector from the Mule Palette and drop it in the canvas to create a new flow.

7. Rename the flow to getCSVAccountsFlow.



8. In the File properties view, set the path to src/main/resources/input.
9. Set the move to directory to src/main/resources/output.
10. Look at the default polling information; the endpoint checks for incoming messages every 1000 milliseconds and sets a minimum of 500 milliseconds a file must wait before it is processed.



Display the file contents

11. Add a Transform Message component to the process section of the flow.

12. In the Transform Message properties view, look at the metadata for the input; there should be no metadata for the payload but there should be metadata for variables and properties.

Transform Message Problems

Input

Payload : Unknown [Define metadata](#)

Flow Variables

- moveToDirectory : String
- originalFilename : String

Session Variables

Inbound Properties

- timestamp : Long
- directory : String
- fileSize : Long
- originalFilename : String

Outbound Properties

- filename : String

13. Set the transformation expression to payload.

Transform Message Problems Console Mule Debugger

Input

Output

Output Payload [Define metadata](#)

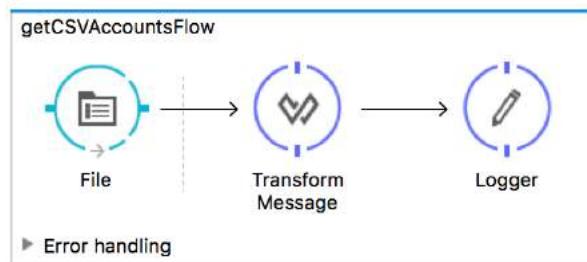
Flow Variables

- moveToDirectory : String

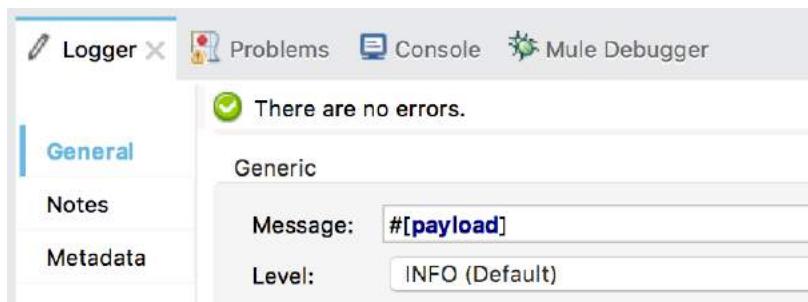
Output

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload
```

14. Add a Logger after the component.



15. In the Logger properties view, set the message to display the payload.

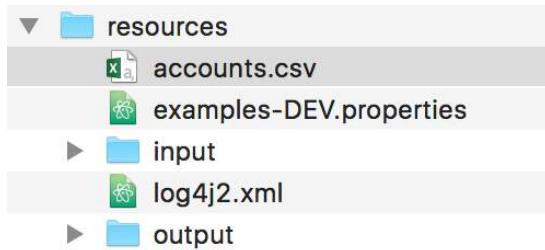


Run the application

16. Run the project.

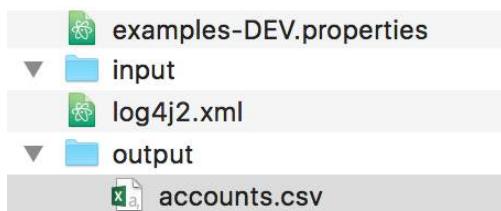
Add a CSV file to the input directory

17. Right-click src/main/resources in the Package Explorer and select Show In > System Explorer.
18. Open the resources folder.
19. In another file browser window, navigate to the student files for the course.
20. Locate the resources/accounts.csv file.
21. Copy this file and then paste it in the src/main/resources folder of the apdev-examples project.



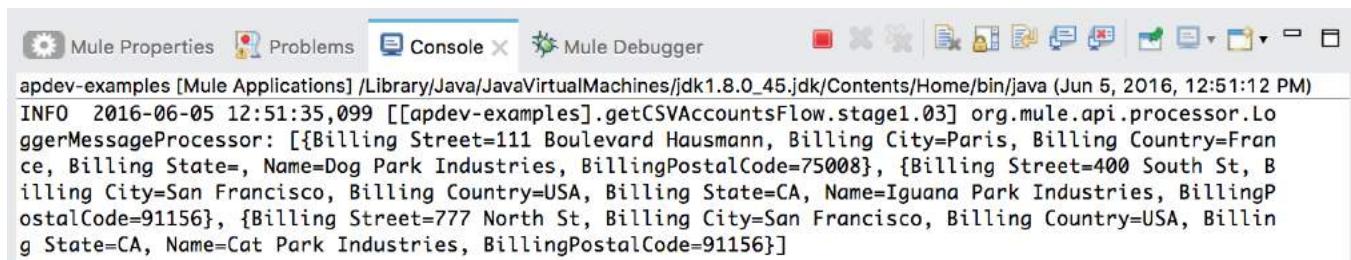
Test the application

22. Drag the accounts.csv file into the input folder; you should see it almost immediately moved to the output folder.



23. Drag it back into the input folder; it will be read again and then moved to the output folder.
24. Leave this folder open.

25. Return to Anypoint Studio and look at the console; you should see the file contents displayed.



```
apdev-examples [Mule Applications] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Jun 5, 2016, 12:51:12 PM)
INFO 2016-06-05 12:51:35,099 [[apdev-examples].getCSVAccountsFlow.stage1.03] org.mule.api.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}]
```

26. Stop the project.

Debug the application

27. In getCSVAccountsFlow, add a breakpoint to the Logger.

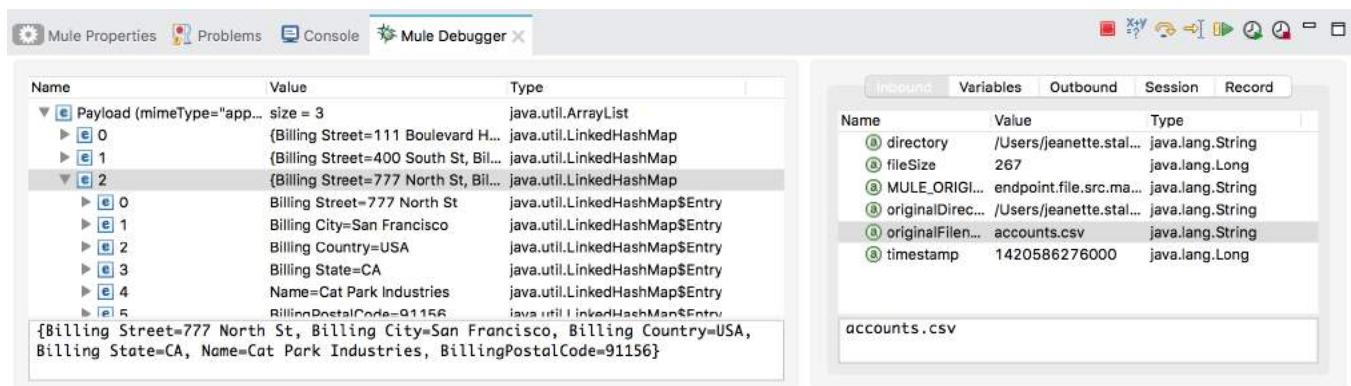
28. Debug the project.

29. Move the accounts.csv file from the output folder back to the input folder.

Note: You can do this in your computer's file browser or in Anypoint Studio. If you use Anypoint Studio, you will need to right-click the project and select Refresh to see the file.

30. In the Mule Debugger, drill-down and look at the payload.

31. Look at the inbound message properties and locate the original filename.



The screenshot shows the Mule Debugger interface with two main panes. The left pane displays the inbound payload as a Java.util.ArrayList of LinkedHashMaps. The right pane shows inbound message properties, including the original filename 'accounts.csv'.

Name	Value	Type
Payload (mimeType="app... size = 3		java.util.ArrayList
0	{Billing Street=111 Boulevard H...	java.util.LinkedHashMap
1	{Billing Street=400 South St, Bil...	java.util.LinkedHashMap
2	{Billing Street=777 North St, Bil...	java.util.LinkedHashMap
0	Billing Street=777 North 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=Cat Park Industries	java.util.LinkedHashMap\$Entry
5	BillingPostalCode=91156	java.util.LinkedHashMap\$Entry

Inbound	Variables	Outbound	Session	Record
directory	/Users/jeanette.stal...	java.lang.String		
fileSize	267	java.lang.Long		
MULE_ORIGI...	endpoint.file.src.ma...	java.lang.String		
originalDirec...	/Users/jeanette.stal...	java.lang.String		
originalFilen...	accounts.csv	java.lang.String		
timestamp	1420586276000	java.lang.Long		

32. Click the Resume button.

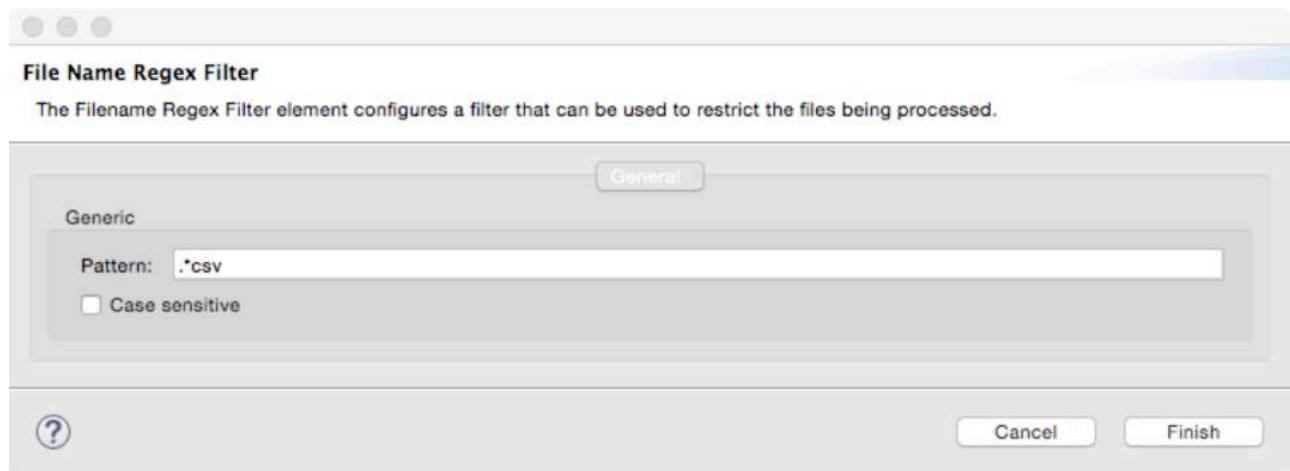
33. Stop the project.

Restrict the type of file read

34. In the File Properties view, click the Add button next to file name regex filter.



35. In the File Name Regex Filter dialog box, set the pattern to `.*csv` (not `*.csv`) and uncheck case sensitive.



36. Click Finish.

Rename the file

37. Set the move to pattern to append `.backup` to the original filename.

```
#*[message.inboundProperties.originalFilename].backup
```

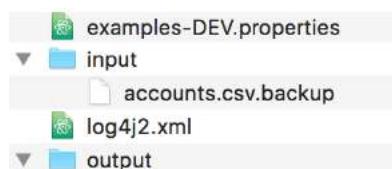
Test the application

38. Run the project.

39. 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.



40. Move the `accounts.csv.backup` file from the output folder back to the input folder; it should not be processed and moved to the output folder.



41. Return to Anypoint Studio and stop the project.

Modify the File endpoint to not rename the files

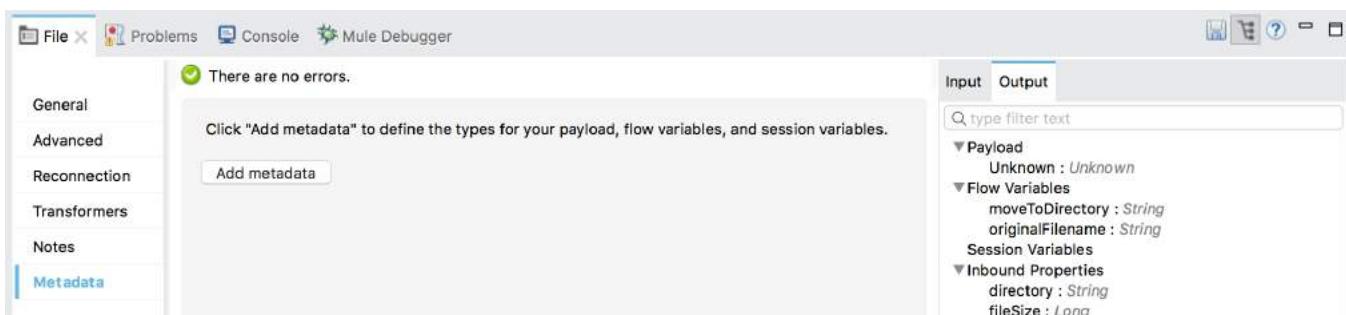
Note: In the last part of the walkthrough, you add metadata for the File endpoint. Although not needed here, this metadata will be used when the data is synchronized to Salesforce in the next module.

42. In the Package Explorer, right-click the src/main/resources/input folder and select Refresh.
43. Expand the src/main/resources/input folder.
44. Right-click accounts.csv.backup and select Refactor > Rename.
45. In the Rename Resource dialog box, set the new name to accounts.csv and click OK.
46. In the File Properties view, delete the move to pattern.

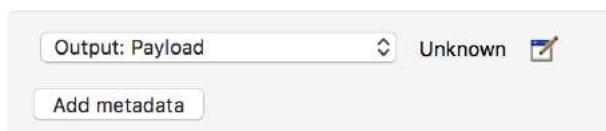
Note: This will make it easier to test the application because you won't have to keep renaming the file as you move it back to the input directory.

Add file endpoint metadata

47. In the File properties view, click the output tab in the Metadata explorer and review the metadata.
48. Click Metadata in the left-side navigation.
49. Click the Add metadata button.

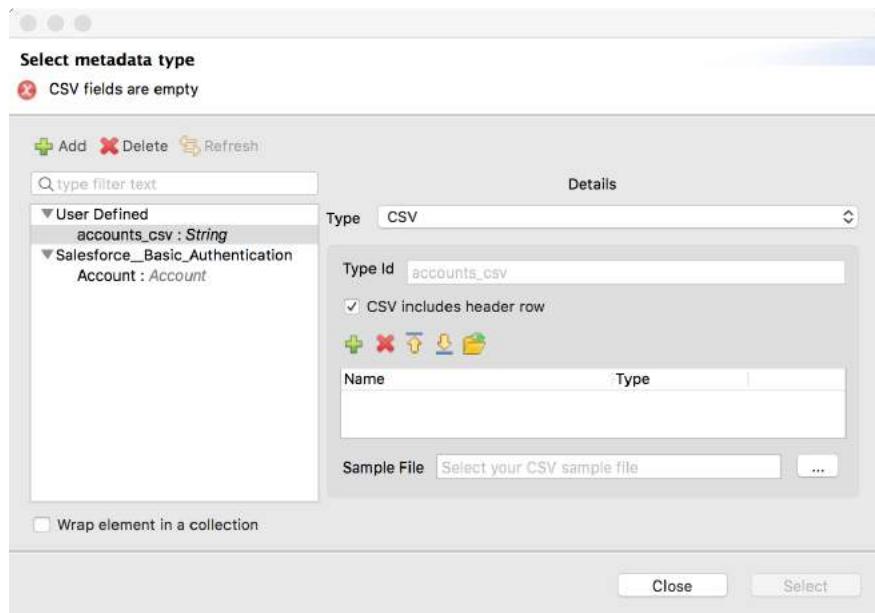


50. In the drop-down menu that appears, make sure Output: Payload is selected.
51. Click the Edit button next to the drop-down menu.



52. In the Select metadata type dialog box, click the Add button.
53. In the Create new type dialog box, set the name to accounts_csv and click Create type.
54. In the Select metadata dialog box, change the type to CSV.

55. Make sure CSV includes header row is checked.

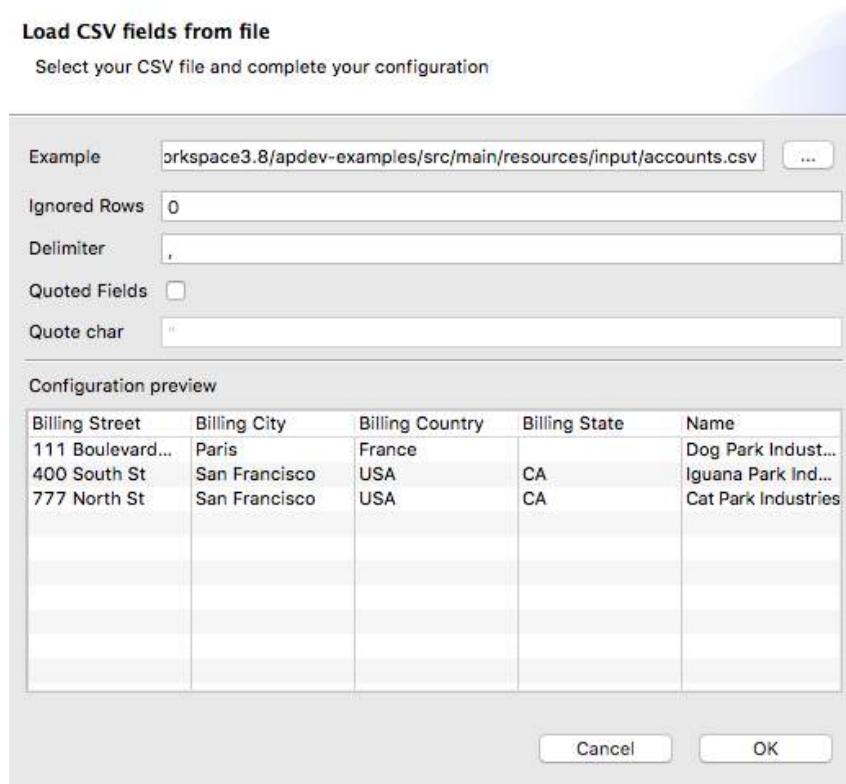


56. Click the Load from example button (the folder icon).

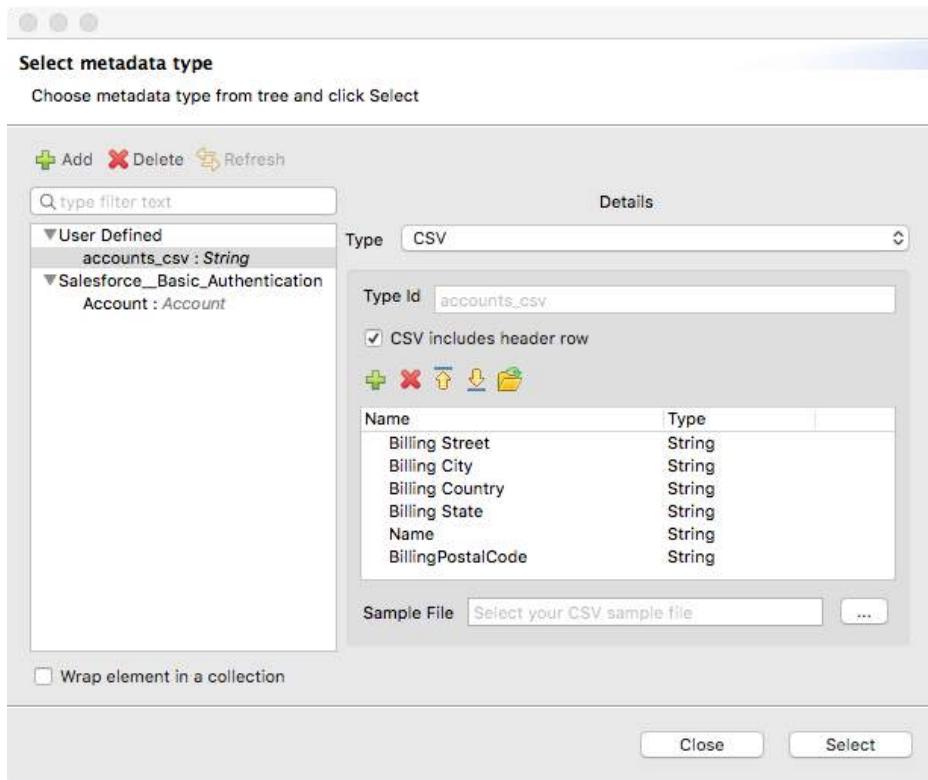
57. In the Load CSV fields from file dialog box, click the Browse button next to Example.

58. Browse to the project's src/main/resources/input folder, select accounts.csv and click Open.

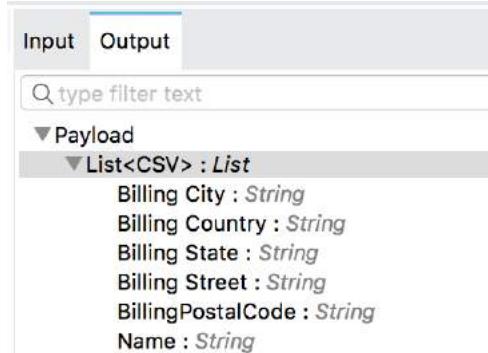
59. In the Load CSV fields from file dialog box, click OK.



60. In the Select metadata type dialog box, click Select.



61. In the File properties view, click the Output tab of the metadata section again; the payload should now have associated metadata and be shown to be a List <CSV>.



62. In the Transform Message properties view, look at the payload metadata in the input section; the payload should now have associated metadata.

63. Save the file.

Walkthrough 11-3: Poll a resource

In this walkthrough, you poll a database. You will:

- Use a form to add accounts for a specific postal code to an accounts table in a database.
- Create a flow with a Poll scope as the message source.
- Poll a database every 5 seconds for records with a specific postal code.
- Use a poll watermark to track the ID of the latest record retrieved.
- Use the watermark to only retrieve new records with that postal code from the database.

accountID	name	street	city	state	postal	country
4	Bevs Bees	12312 Blue Rd	Cleveland	Ohio	44147	United States
3	Marks Markers	39203 red st	San Francisco	California	94116	United States
2	Dans Databases	329329 Blue St	San Francisco	California	94116	United States
1	Webbers bagels	32423 Blue Rd	Avon	Ohio	44011	United States

pollDatabaseFlow

Poll

Database

Logger

Error handling

Parameterized query:

```
SELECT *
FROM accounts
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

Get familiar with the data in the database

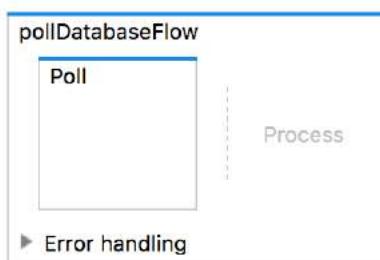
1. Return to the course snippets.txt file and copy the Account list URL for the MySQL database.
2. In a web browser, navigate to the URL you copied.
3. Look at the existing data and the name of the columns (which match the names of the database fields) and the postal values.

accountID	name	street	city	state	postal	country
4	Bevs Bees	12312 Blue Rd	Cleveland	Ohio	44147	United States
3	Marks Markers	39203 red st	San Francisco	California	94116	United States
2	Dans Databases	329329 Blue St	San Francisco	California	94116	United States
1	Webbers bagels	32423 Blue Rd	Avon	Ohio	44011	United States

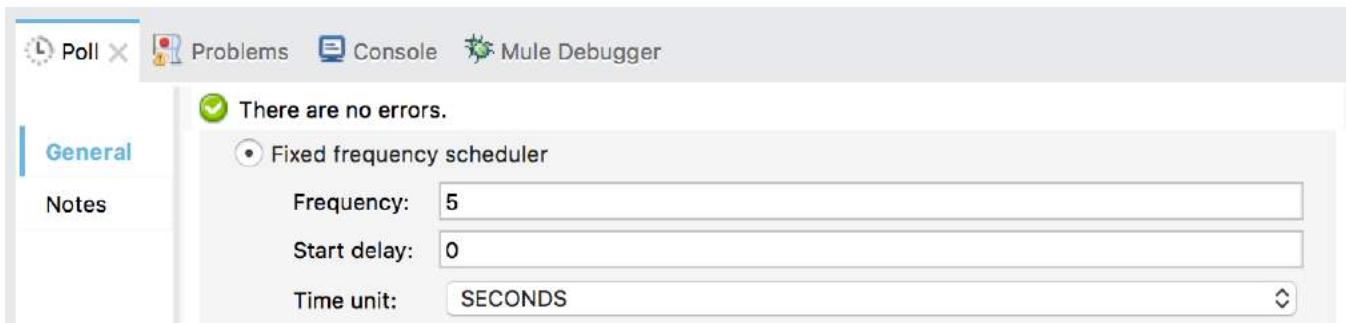
- Click the Create More Accounts button.
- Add a new record with a specific postal code; you will retrieve this record in this walkthrough and insert it into your Salesforce account in the next module.

Create a flow that polls a database

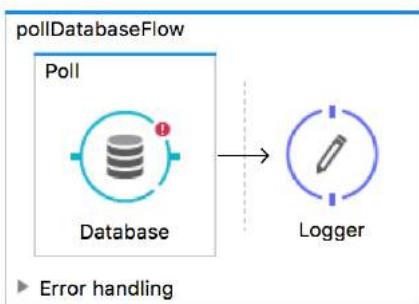
- Return to accounts.xml in Anypoint Studio.
- Drag a Poll scope from the Mule Palette and drop it at the top of the canvas.
- Change the name of the flow to pollDatabaseFlow.



- In the Poll properties view, select the fixed frequency scheduler.
- Set the frequency to 5 and the time unit to seconds.



- Drag a Database connector from the Mule Palette to the poll scope.
- Add a Logger to the process section of the flow.



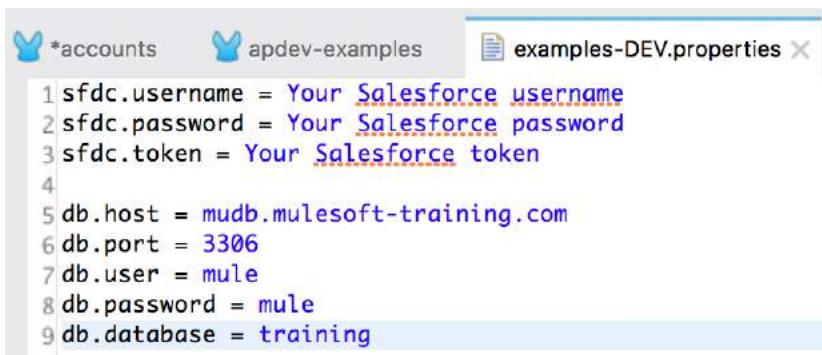
13. In the Logger properties view, set the message to display the payload.

Note: Instead of polling the database directly, you could use the poll scope to make periodic requests to an API that governs access to the database.

Configure the Database connector

14. Return to the course snippets.txt file and copy the database parameters (the five starting with db).

15. Return to examples-DEV.properties and paste the values.



```
sfdc.username = Your Salesforce username
sfdc.password = Your Salesforce password
sfdc.token = Your Salesforce token
db.host = mudb.mulesoft-training.com
db.port = 3306
db.user = mule
db.password = mule
db.database = training
```

16. Save the file.

17. Return to the Global Elements view of global.xml.

18. Click Create.

19. In the Choose Global Type dialog box, select Connector Configuration > MySQL Configuration and click OK.

20. In the Global Element Properties dialog box, set the following values and click OK.

- Host: \${db.host}
- Port: \${db.port}
- User: \${db.user}
- Password: \${db.password}
- Database: \${db.database}

21. Under Required dependencies, click the Add File button next to MySQL Driver.

22. Navigate to the student files folder, select the MySQL JAR file located in the resources folder, and click Open.

23. 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.

24. Click OK to close the dialog box.
25. Click OK to close the Global Element Properties dialog box.

Configure the Database endpoint

26. Return to accounts.xml.
27. In the Database properties view, set the connector configuration to the existing MySQL_Configuration.
28. Set the operation to Select.
29. Add a query to select the data for the postal code of the record you added to the accounts table.

```
SELECT *
FROM accounts
WHERE postal = '94108'
```

Test the application

30. Run the project.
31. Watch the console; you should see records displayed every 5 seconds.

```
INFO 2016-05-14 10:52:18,088 [[apdev-examples].pollDatabaseFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF, postal=94108}]
INFO 2016-05-14 10:52:27,845 [[apdev-examples].pollDatabaseFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF, postal=94108}]
INFO 2016-05-14 10:52:37,855 [[apdev-examples].pollDatabaseFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF, postal=94108}]
```

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.

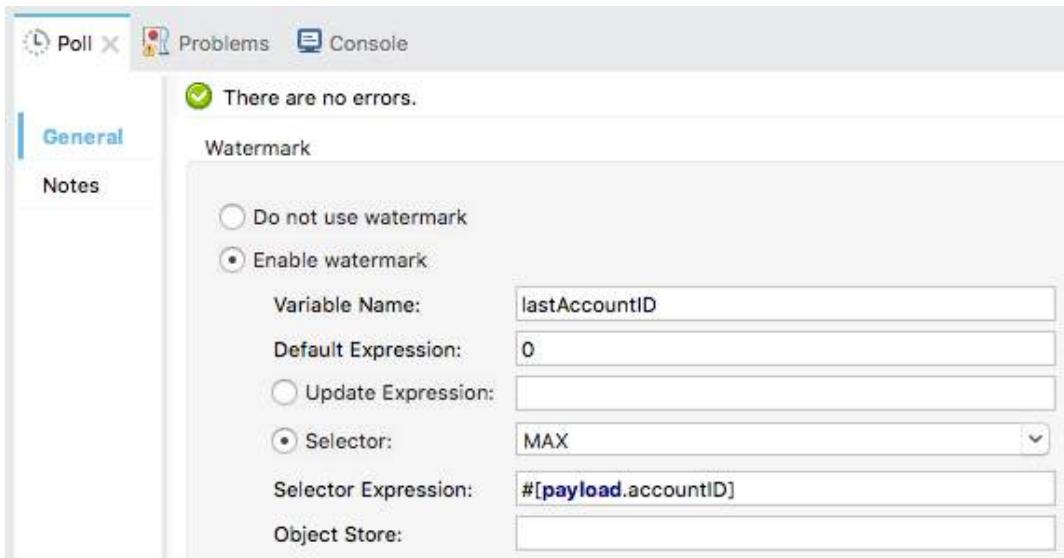
32. Stop the project.

Use a watermark to keep track of the last record retrieved

33. In the Poll properties view, select Enable watermark.

34. Set the watermark to store the max accountID returned by setting the following values.

- Variable name: lastAccountID
- Default expression: 0
- Selector: MAX
- Selector Expression: #[payload.accountID]



Run the application

35. Run the project; the application should deploy but then throw an exception.

36. Locate the exception in the console; you should see a message that watermarking requires synchronous polling.

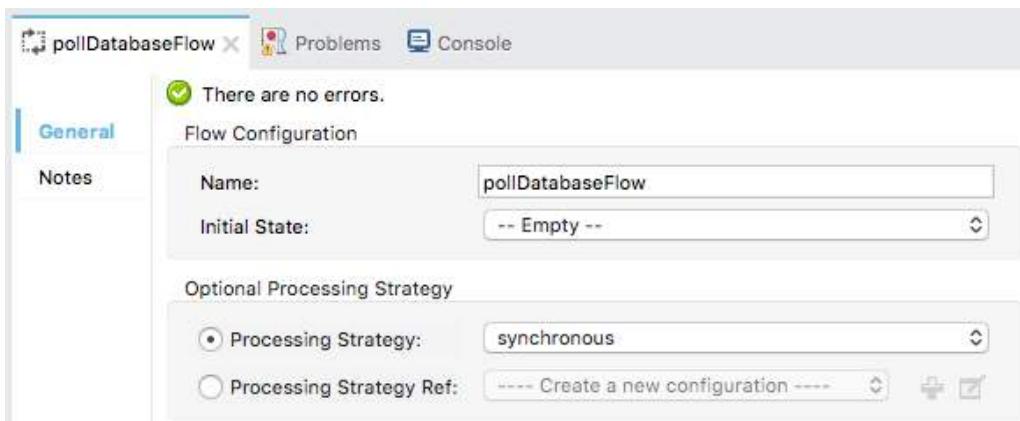
```
ERROR 2016-05-14 10:55:23,871 [pool-32-thread-1] org.mule.exception.DefaultSystemExceptionStrategy:  
*****  
Message : Watermarking requires synchronous polling  
-----  
Root Exception stack trace:  
org.mule.api.config.ConfigurationException: Watermarking requires synchronous polling  
    at org.mule.transport.polling.watermark.WatermarkPollingInterceptor.prepareRouting(WatermarkPollingInte  
rceptor.java:53)
```

37. Stop the project.

Make the flow synchronous

38. In the Properties view for pollDatabaseFlow, select processing strategy.

39. In the processing strategy drop-down menu, select synchronous.



Debug the application and examine the watermark value

40. Place a breakpoint on the Logger.
41. Debug the project.
42. Wait until the breakpoint is hit.
43. Locate your watermark variable in the Variables section of the Mule Debugger view; initially, you should see a default value of zero.

Variables		
Name	Value	Type
lastAccountID (mimeType=...)	0	java.lang.String
pollingFrequency (mimeType=...)	1000	java.lang.Long

44. Click the Resume button.
45. Look at the value of the watermark variable again; it should now be equal to the max accountID for training accounts records with the postal code you are using.

Variables		
Name	Value	Type
lastAccountID (mimeType=...)	2769	java.lang.Integer
pollingFrequency (mimeType=...)	1000	java.lang.Long

46. Resume the application multiple times; the same records should still be selected over and over again.
47. Stop the project.

Modify the database query to use the watermark

48. In the Database properties view, modify the query so it only returns records for your postal code and with accountID values greater than the watermark lastAccountID value.

```
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

Parameterized query:

```
SELECT *
FROM accounts
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

Test the application

49. Run the project.
50. Look at the console; you should see that no records are retrieved at all this time.

```
INFO 2016-05-14 11:06:06,312 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
INFO 2016-05-14 11:06:06,312 [pool-31-thread-1] org.mule.transport.polling.watermark.Watermark: Watermark value will not be updated since poll processor returned no results
INFO 2016-05-14 11:06:16,039 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
INFO 2016-05-14 11:06:16,039 [pool-31-thread-1] org.mule.transport.polling.watermark.Watermark: Watermark value will not be updated since poll processor returned no results
```

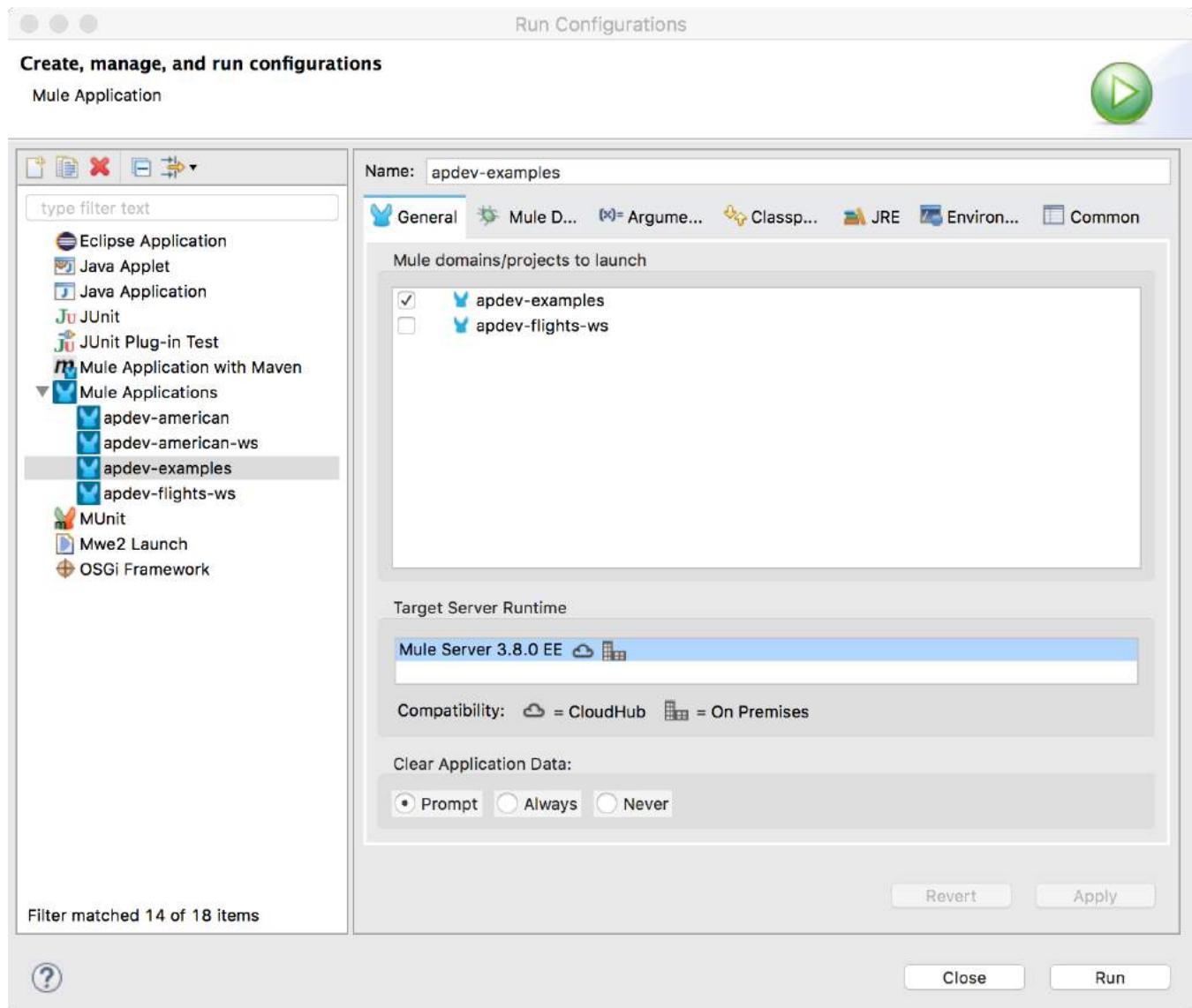
Note: By default, the watermark is stored in a persistent object store so its value is retained between different executions of the application.

51. Stop the project.

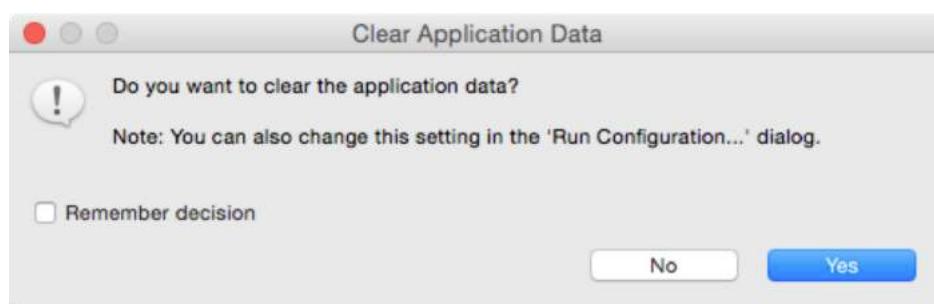
Clear application data

52. Select Run > Run Configurations.

53. Make sure your apdev-examples project is selected and then on the General tab, scroll down and change Clear Application Data from Never to Prompt.



54. Click Run; you should be prompted to clear the application data.
55. In the Clear Application Data dialog box, click Yes.



Test the application

56. Look at the console; you should see the latest matching records retrieved from the database again – but this time, only once.
57. Watch the console and see that all subsequent polling events retrieve no records.

```
INFO 2016-05-14 11:07:52,600 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: [{country=USA, accountID=4338, street=77 Geary St, state=CA, name=John Doe, city=SF, postal=94108}]\nINFO 2016-05-14 11:08:02,318 [pool-31-thread-1] org.mule.api.processor.LoggerMessageProcessor: []\nINFO 2016-05-14 11:08:02,318 [pool-31-thread-1] org.mule.transport.polling.watermark.Watermark: Watermark value will not be updated since poll processor returned no results
```

Add a new account with the same postal code

58. Return to the web browser displaying the account data.
59. Add another record with the same postal code.
60. Return to the console in Anypoint Studio.
61. Watch the console until you see your new record displayed on the next polling event.

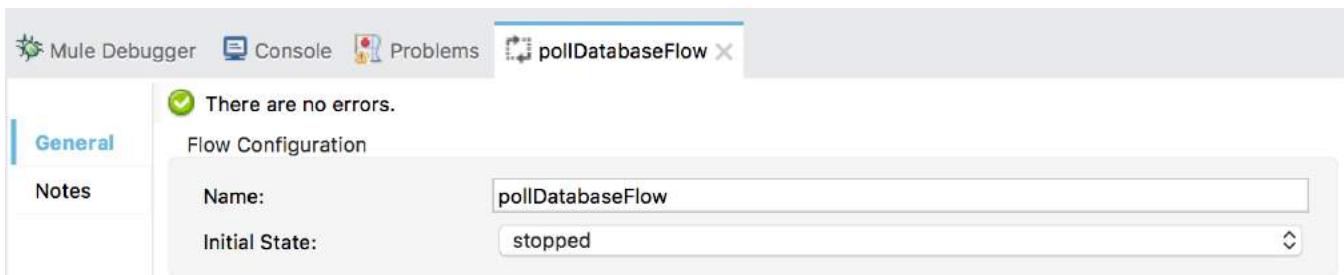
```
INFO 2016-06-05 14:13:10,810 [pool-27-thread-1] org.mule.transport.polling.watermark.Watermark: Watermark value will not be updated since poll processor returned no results\nINFO 2016-06-05 14:13:15,810 [pool-27-thread-1] org.mule.api.processor.LoggerMessageProcessor: [{country=United States, accountID=2770, street=77 Geary Street, state=CA, name=Molly Mule, city=San Francisco, postal=94108}]
```

INFO 2016-06-05 14:13:20,821 [pool-27-thread-1] org.mule.api.processor.LoggerMessageProcessor: []
INFO 2016-06-05 14:13:20,821 [pool-27-thread-1] org.mule.transport.polling.watermark.Watermark: Watermark value will not be updated since poll processor returned no results

62. Stop the project.

Stop the flow

63. In the Properties view for pollDatabaseFlow, set initial state to stopped.



64. Save the file.

Walkthrough 11-4: Connect to a JMS queue (ActiveMQ)

In this walkthrough, you read and write messages from a JMS topic. You will:

- Create a flow accessible at <http://localhost:8081/jms>.
- Add and configure an ActiveMQ connector.
- Use a JMS endpoint to retrieve messages from a JMS topic.
- Add messages to the topic using a web form.
- Use a JMS endpoint to send messages to a JMS topic.

Send a Message to a JMS Topic

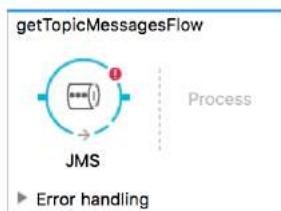
Name
Max
Message
Hello

INFO 2016-06-05 14:46:36,845 [[apdev-examples].getTopicMessagesFlow.st...
LoggerMessageProcessor: Hello
INFO 2016-06-05 14:47:11,557 [[apdev-examples].getTopicMessagesFlow.st...
LoggerMessageProcessor: Is it break time yet?

Send

Create a JMS inbound endpoint

1. Return to the apdev-examples project.
2. Create a new Mule configuration file called jms.xml.
3. Drag out a JMS connector from the Mule Palette and drop it in the canvas to create a new flow.
4. Give the flow a new name of getTopicMessagesFlow.



5. In the JMS properties view, leave the exchange pattern set to one-way.
6. Select topic and set it to apessentials.

'connector-ref' attribute is required

Display Name: JMS

Basic Settings

Exchange Pattern: one-way (Default) request-response

Queue:

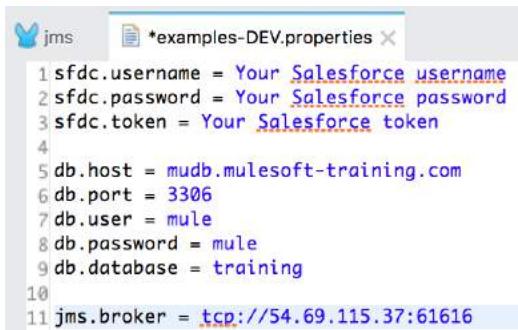
Topic: apessentials

Connector Configuration: [Create a new configuration](#) [+](#) [-](#)

Configure the JMS connector

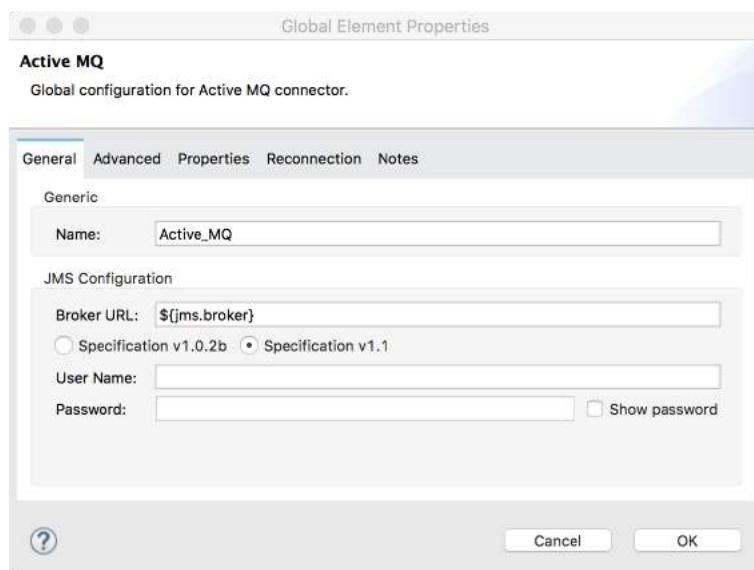
7. Return to the course snippets.txt file and copy the value for the ActiveMQ Broker URL.
8. Return to Anypoint Studio.
9. Return to examples-DEV.properties.
10. Create a new property called jms.broker and set it equal to the value you copied from the course snippets.txt file.

Note: If you do not have access to port 61616, set jms.broker equal to vm://localhost instead.



```
jms *examples-DEV.properties X
1 sfdc.username = Your Salesforce username
2 sfdc.password = Your Salesforce password
3 sfdc.token = Your Salesforce token
4
5 db.host = mudb.mulesoft-training.com
6 db.port = 3306
7 db.user = mule
8 db.password = mule
9 db.database = training
10
11 jms.broker = tcp://54.69.115.37:61616
```

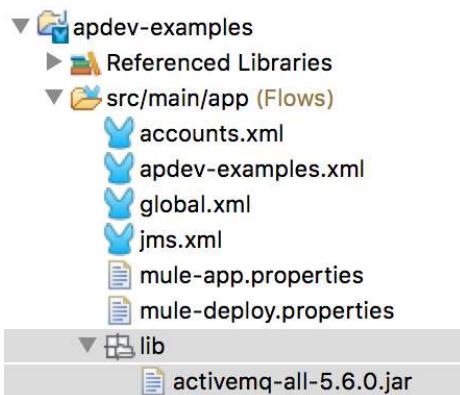
11. Save the file.
12. Return to the Global Elements view in global.xml.
13. Click Create.
14. In the Choose Global Type dialog box, select Connector Configuration > JMS > Active MQ and click OK.
15. In the Global Element Properties dialog box, change the broker URL to the property \${jms.broker}.
16. Set the Specification to v1.1.



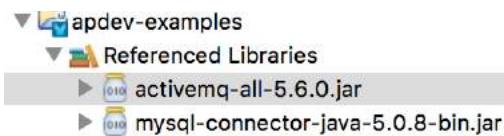
17. Click OK.
18. Return to jms.xml.
19. In the JMS properties view, set the connector configuration to the existing Active_MQ configuration.

Add the ActiveMQ library

20. In the Package Explorer, right-click apdev-examples and select New > Folder.
21. In the New Folder dialog box, set the folder name to lib and click Finish.
22. In your computer's file browser, locate the activemq-all.jar file located in the jars folder in the student files.
23. Copy and paste or drag the JAR file into the lib folder.



24. Right-click the JAR file in the Package Explorer and select Build Path > Add to Build Path; you should now see it under Referenced Libraries.



Note: Adding activemq-all.jar can create conflicts with other dependencies in projects, so it is recommended that you only add only the JAR files you need in relation to what you need ActiveMQ for. For more details, see the documentation at <https://docs.mulesoft.com/mule-user-guide/v/3.8/activemq-integration>.

Test the application and receive messages

25. Return to jms.xml.
26. Add a Logger to the flow and set its message to #[payload].
27. Run the project.

28. In the Clear Application Data dialog box, select Remember decision and click No.
29. Return to the course snippets.txt file and copy the value for the JMS Form URL.
30. In a web browser, navigate to the URL you copied.
31. In the form, enter your name and a message and click Send.

Send a Message to a JMS Topic

Name
Max

Message
Hello

Send

32. In the popup window with the response, click OK.
33. Return to the console in Anypoint Studio; you should see your message along with those from your classmates – but you don't see the names of the people who sent the messages.

```
INFO 2016-06-05 14:46:36,845 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Hello
INFO 2016-06-05 14:47:11,557 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Is it break time yet?
```

34. Stop the project.

Debug the application

35. Add a breakpoint to the Logger.
36. Debug the project.
37. Make another request to the JMS form URL and submit another message.
38. In the Mule Debugger view, look at the payload and inbound message properties.

Name	Value	Type
(DataType)	SimpleDataType{type=org....}	org.mule.transformer.types...
(Exception)	null	
(Message)		org.mule.DefaultMuleMessage
(Message Processor)	Logger	org.mule.api.processor.Log...
(Payload (mimeType...))	Is it break time yet?	java.lang.String

Name	Value	Type
MULE_ENDPOINT	jms://topic:apessentials	java.lang.String
MULE_MESSAGE_ID	ID:ip-172-16-188-2...	java.lang.String
MULE_ORIGIN	endpoint:jms.apessen...	java.lang.String
MULE_SESSION	rO0ABXNyACNvcmcu...	java.lang.String
name	Molly	java.lang.String

39. Step through the rest of the application.
40. Stop the project.

Display names with the messages

41. In the Logger properties view, change the message to use an expression to display the name and message.

```
##[message.inboundProperties.name + ":" + payload]
```

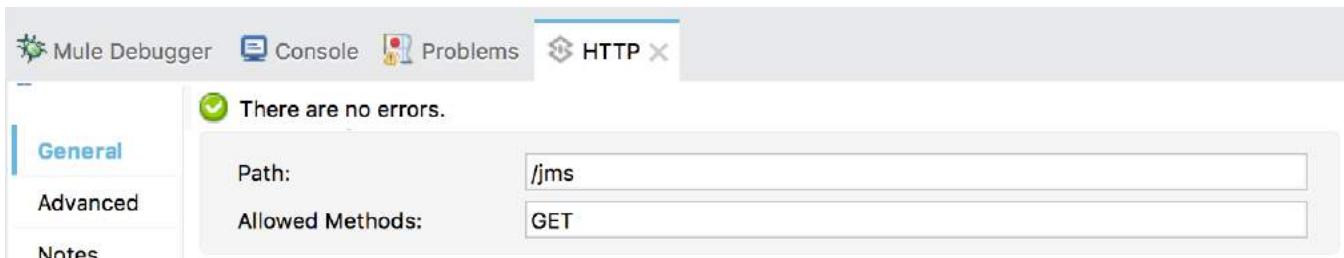
Test the application

42. Run the project.
43. Return to the message form and submit a new message.
44. Look at the console; you should now see names along with messages.

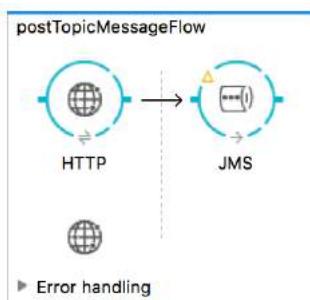
```
INFO 2016-06-05 14:50:26,999 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Max: Hello
INFO 2016-06-05 14:50:32,902 [[apdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Molly: Is it break time yet?
```

Create a JMS outbound-endpoint

45. Drag out an HTTP connector from the Mule Palette and drop it in the canvas.
46. Give the flow a new name of postTopicMessageFlow.
47. In the HTTP properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.
48. Set the path to `/jms` and the allowed methods to `GET`.



49. Drag out a JMS connector from the Mule Palette and drop it into the process section of the flow.



50. In the JMS Properties view, select topic and set it to apessentials.

51. Set the connector configuration to the existing Active_MQ.

The screenshot shows the Mule Studio interface with the JMS tab selected. On the left, there's a sidebar with tabs: General (selected), Advanced, Reconnection, Transformers, and Notes. The main panel has a green checkmark icon and the message "There are no errors." Below it, the Exchange Pattern is set to "one-way (Default)". Under "Topic:", there are two options: "Queue:" (unchecked) and "Topic:" (checked), with the value "apessentials" entered. The "Connector Configuration:" dropdown is set to "Active_MQ". There are also standard UI controls like a plus sign and a pencil icon.

Set a message

52. Add a Set Payload transformer between the HTTP and JMS connector endpoints.

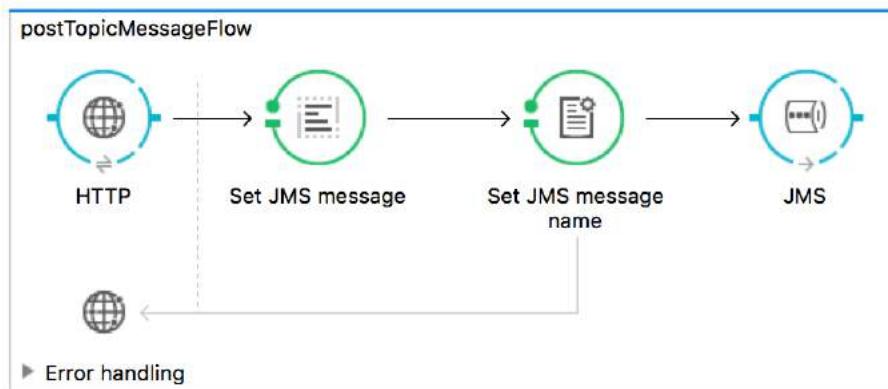
53. In the Set Payload Properties view, change the display name to Set JMS message.

54. Set the value to a message query parameter.

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

55. Add a Property transformer after the Set Payload transformer.

56. In the Properties view, change the display name to Set JMS message name.



57. Select Set Property and set the name to name and the value to your name.

Note: You can set this to a query parameter instead if you prefer.

The screenshot shows the Mule Studio interface with the "Set JMS message name" tab selected. On the left, there's a sidebar with tabs: General (selected), Notes, and ... (partial). The main panel has a green checkmark icon and the message "There are no errors." Below it, there are two rows of properties: "Name:" with the value "name" and "Value:" with the value "Max".

Test the application and post messages

58. Save the file to redeploy the project.
59. Make a request to <http://localhost:8081/jms?message=Hello>.
60. Look at the console; you should see your name and message displayed – along with those of your classmates.

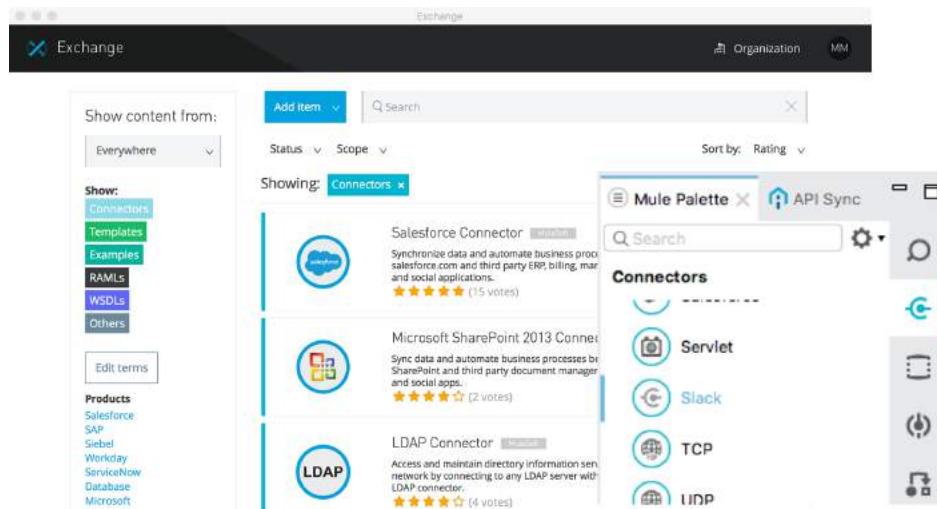
```
INFO 2016-06-05 14:58:05,084 [[cpdev-examples].getTopicMessagesFlow.stage1.02] org.mule.api.processor.  
LoggerMessageProcessor: Max: Hello
```

61. Stop the project.

Walkthrough 11-5: Find and install not-in-the-box connectors

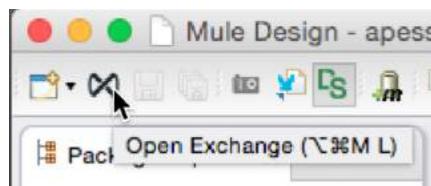
In this walkthrough, you learn how to add a new connector to Anypoint Studio. You will:

- Browse the Anypoint Exchange.
- Install a connector from the Exchange to Anypoint Studio.
- Locate the new connector in Anypoint Studio.
- Manage installed software.

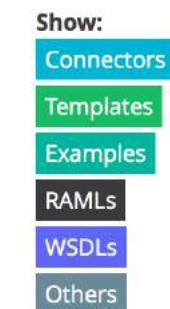


Browse the Anypoint Exchange from Anypoint Studio

1. In Anypoint Studio, click the Open Exchange button; the Anypoint Exchange should open in a new window.



2. In the Exchange, click the Connectors button.



3. Browse the connectors.

The screenshot shows the Exchange connector library interface. On the left, there's a sidebar with navigation links: 'Show content from:' (set to 'Everywhere'), 'Add item' (dropdown), 'Search' (input field), 'Status' (dropdown), 'Scope' (dropdown), 'Sort by: Rating' (dropdown), and a 'Show' dropdown set to 'Connectors'. Below these are categories: 'Connectors' (selected), 'Templates', 'Examples', 'RAMLs', 'WSDLs', and 'Others'. There's also an 'Edit terms' button. Under 'Products', there are links for 'Salesforce', 'SAP', 'Siebel', 'Workday', 'ServiceNow', 'Database', and 'Microsoft'. The main area displays a list of connectors:

- Salesforce Connector** [MuleSoft] Update
Sync data and automate business processes between salesforce.com and third party ERP, billing, marketing automation and social applications.
★★★★★ (15 votes)
- Microsoft SharePoint 2013 Connector** [MuleSoft] Install
Sync data and automate business processes between Microsoft SharePoint and third party document management, collaboration and social apps.
★★★★☆ (2 votes)
- LDAP Connector** [MuleSoft] Install
Access and maintain directory information services over an IP network by connecting to any LDAP server with the Anypoint LDAP connector.
★★★★☆ (4 votes)

4. Locate the Slack Connector (or any other connector you are interested in).

The screenshot shows the Exchange connector library interface, similar to the previous one but with different connector options. The sidebar includes 'Premium', 'Select', 'Standard', and 'Community' filters, and 'Category' filters for 'Collaboration', 'Enterprise IT', 'Finance', 'Human Resources', 'Marketing', 'Sales', and 'Business Process Administration'. The main area displays a list of connectors:

- Slack Connector** [MuleSoft] Install
Automate responses and notifications for channels and groups, integrate easily your business with the great team communication tool.
★★★★☆ (3 votes)
- SAP R/3 Business Suite Connector** [MuleSoft] View details
Sync data and automate business processes between SAP Business Suite and third party applications, either on premise or in the cloud.
★★★★☆ (3 votes)
- Workday Connector** [MuleSoft] Install
Sync data and enable connectivity between Workday and third party HR and ERP applications and services, on-premise and in the cloud.
★★★★☆ (3 votes)

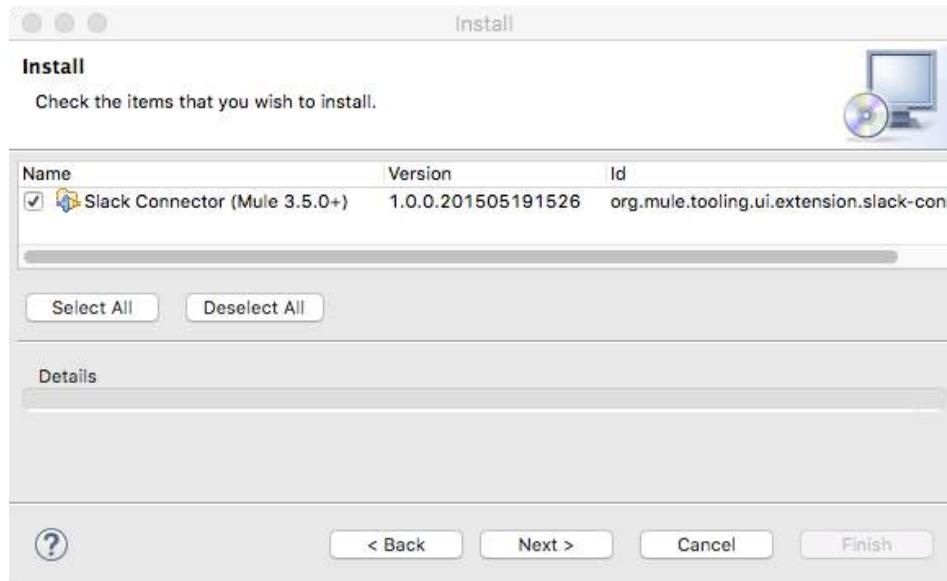
5. Click the View details button and browse the connector's documentation.

- Locate the connector level and the type of support; you should see this is a community connector and not supported by MuleSoft.

The screenshot shows the Slack Connector page on the MuleSoft Exchange. At the top right, there are links for 'Organization' and 'MM'. Below the header, there's a large 'Install' button. To its left is the connector's name, 'Slack Connector MuleSoft', and a brief description: 'Automate responses and notifications for channels and groups, integrate easily your business with the great team communication tool.' On the right side, there are three buttons: 'Share URL', 'Contact us', and 'Clone me!'. Below these buttons is a 'Rating' section showing 3 stars from 3 votes. A note says 'Please install this Connector to rate it.' Further down, there are sections for 'Level' (Community) and 'Support' (Developer-provided).

Install the connector

- Click the Install button.
- In the Install dialog box, enter your personal information and click Install.
- Wait until an Install dialog box appears in Anypoint Studio.



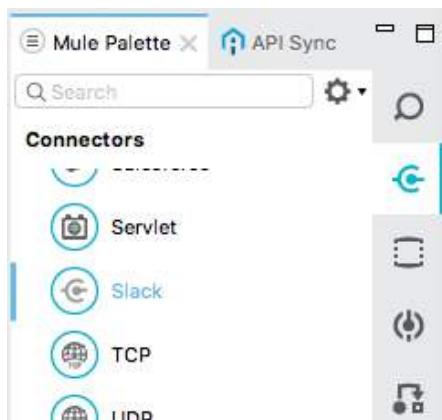
Note: If you do not see the connector listed, resize the Install dialog box.

Note: You can also install connector's directly from Anypoint Studio. From the main menu bar, select Help > Install New Software. In the Install dialog box, click the Work with drop-down button and select Anypoint Connectors Update Site. Drill-down into Community and select the Slack Connector (or some other connector).

10. Click Next.
11. On the Install Details page, click Next.
12. On the Review Licenses page, select the I accept the terms of the license agreement radio button.
13. Click Finish; the software will be installed and then you should get a message to restart Anypoint Studio.
14. In the Software Updates dialog box, click Yes to restart Anypoint Studio.

Locate the new connector in Anypoint Studio

15. After Anypoint Studio restarts, locate the new connector in the Connectors section of the Mule Palette.



Manage installed software

16. After Anypoint Studio restarts, locate the new connector in the Connectors section of the Mule Palette.
17. In the main menu, select Help > Installation Details.

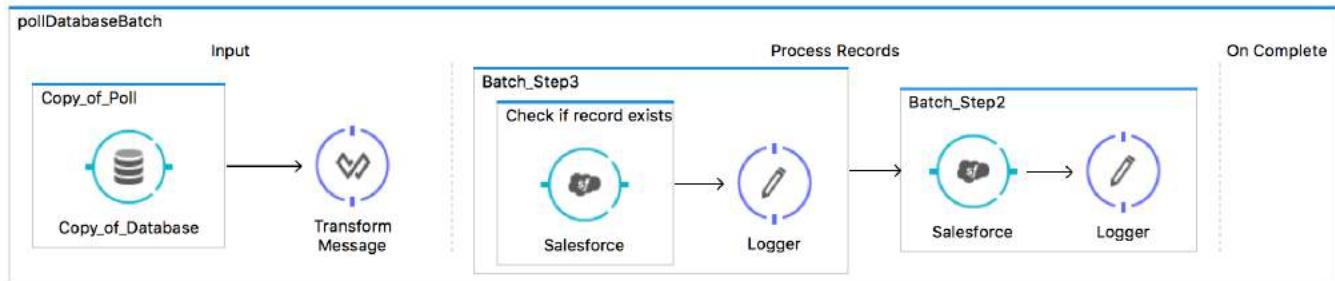
18. Select the Slack Connector; you should see Update, Uninstall, and Properties buttons for the connector.

The screenshot shows the 'Anypoint Studio Installation Details' window with the 'Installed Software' tab selected. The window lists various installed connectors and tools, each with its name, version, and ID. The 'Slack Connector (Mule 3.5.0+)' is highlighted in the list. At the bottom of the window, there are buttons for '?', 'Update...', 'Uninstall...', 'Properties', and 'Close'.

Name	Version	Id
Amazon S3 Connector (Mule 3.5.0+)	4.0.1.201601151645	org.mule.tooling.ui.extension.s3.3.5.0.fea...
Amazon SQS Connector (Mule 3.5.0+)	4.0.2.201605051332	org.mule.tooling.ui.extension.sqs.3.5.0.fea...
Anypoint APIkit SOAP Extension	1.0.0.201605131527	org.mule.tooling.soapkit.extension.feature....
Anypoint MQ Connector (3.7+)	1.0.1	org.mule.tooling.messaging.extension.feat...
▶ Anypoint Studio	6.0.0.201605161739	studio.product
▶ APIkit Studio Plugin	1.5.4.201605131414	org.mule.tooling.apikit.feature.feature.group
CMIS Connector (Mule 3.5.0+)	2.1.0.201510201313	org.mule.tooling.ui.extension.cmis.3.5.0.fe...
Google Contacts Connector (Mule 3.5+)	1.2.4.201403071703	org.mule.tooling.ui.extension.google-conta...
Magento Connector (Mule 3.5+)	2.2.0.201502271302	org.mule.tooling.ui.extension.magento.3.5....
Mongo DB Connector (Mule 3.5+)	3.6.1.201502271137	org.mule.tooling.ui.extension.mongo.3.5.0....
Mule Server Runtime 3.8.0 EE	6.0.0.201605131244	org.mule.tooling.extension.server.3.8.0.ee...
MuleSoft Enterprise Java Connector for SAP (3.5+)	6.0.0.201605091440	org.mule.tooling.ui.extension.sap.3.5.0.fea...
MUnit Anypoint Studio Plugin	1.2.0.201605131722	org.mule.tooling.munit.extension.feature.group
Munit Synchronize Module for Integration Testing	1.2.0.201605131722	org.mule.tooling.ui.synchronize.extension.f...
RAML API Editor	0.9.0.201605131301	org.raml.editor.feature.feature.group
Salesforce Connector (Mule 3.5+)	7.1.0.201604131352	org.mule.tooling.ui.extension.sfdc.3.5.0.fea...
Slack Connector (Mule 3.5.0+)	1.0.0.201505191526	org.mule.tooling.ui.extension.slack-connec...

19. Click Close.

Module 12: Processing Records



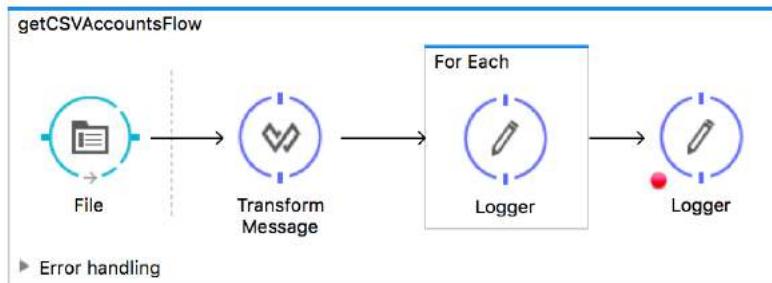
Objectives:

- Use the For Each scope to process items in a collection individually.
- Use the batch job element (EE) to process individual records.
- Trigger a batch job using a poll.
- Use a batch job to synchronize data from a legacy database to a SaaS application.

Walkthrough 12-1: Process items in a collection individually

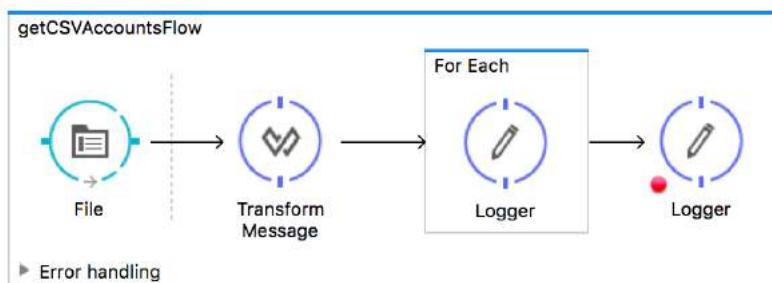
In this walkthrough, you split a collection and process each item in it individually. You will:

- Use the For Each scope element to process each item in a collection individually.
- Look at the thread used to process each record.



Add a for each scope element

1. Return to accounts.xml.
2. Review getCSVAccountsFlow.
3. Drag a For Each scope element form the Mule Palette and drop it after the Transform Message component.
4. Add a Logger to the for each scope.



Process each element

5. In the Logger properties view, set the message to #[payload].

Debug the application

6. Add a breakpoint to the Transform Message component.
7. Debug the project.
8. Drag accounts.csv from the src/main/resources/output folder to the src/main/resources/input folder; application execution should stop at the Transform Message component.

- In the Mule Debugger view, look at the payload value.
- Step to the For Each scope; the payload should be an ArrayList of LinkedHashMaps.

Name	Value	Type
Payload (mimeType...)	size = 3	java.util.ArrayList
0	{Billing Street=111 Boule...	java.util.LinkedHashMap
1	{Billing Street=400 South...	java.util.LinkedHashMap
2	{Billing Street=777 North...	java.util.LinkedHashMap
0	Billing Street=777 North St	java.util.LinkedHashMap\$...
1	Billing City=San Francisco	java.util.LinkedHashMap\$...

- Step into the for each; the payload should be a LinkedHashMap.

Message Flow Global Elements Configuration XML

Mule Properties Problems Console Mule Debugger

Name	Value	Type
Message Processor	Logger	org.mule.api.processor.Lo...
Payload (mimeType...)	size = 6	java.util.LinkedHashMap
0	Billing Street=111 Boulev...	java.util.LinkedHashMap\$...
1	Billing City=Paris	java.util.LinkedHashMap\$...
2	Billing Country=France	java.util.LinkedHashMap\$...

- Look at the console; you should see the data for the first record.

- Step through the application; you should see each record displayed.

```

loggerMessageProcessor: {Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}
INFO 2016-06-05 15:55:30,638 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}
INFO 2016-06-05 15:55:30,639 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Cat Park Industries, BillingPostalCode=91156}
INFO 2016-06-05 15:55:33,708 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: {Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}

```

- Step through to the Logger after the For Each scope; the payload should be an ArrayList again.
- Step to the end of the application.
- Stop the project.

Look at the processing threads

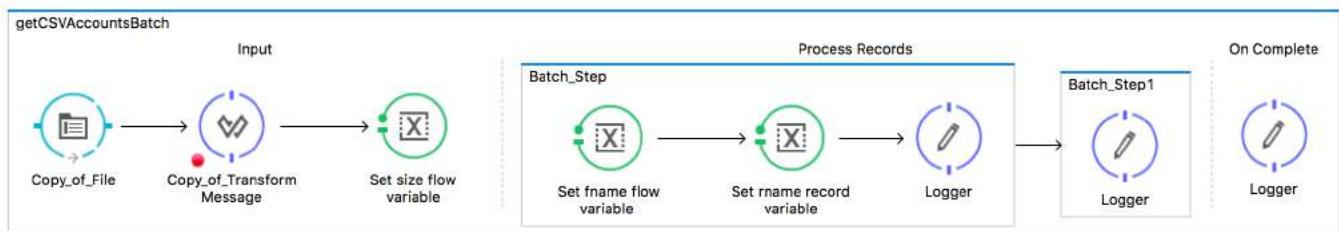
17. In the console, locate the thread number used to process each record in the collection; the same thread should be used for each: apdev-examples.getCSVAccountsFlow.stage1.02.

```
INFO 2016-06-05 15:55:30,637 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: {Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}
INFO 2016-06-05 15:55:30,638 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}
INFO 2016-06-05 15:55:30,639 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Cat Park Industries, BillingPostalCode=91156}
INFO 2016-06-05 15:55:33,708 [[apdev-examples].getCSVAccountsFlow.stage1.02] org.mule.api.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 12-2: Create a batch job for records in a file

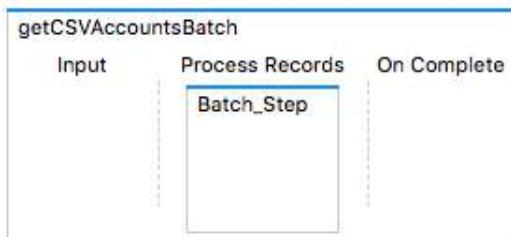
In this walkthrough, you create a batch job to process records in a CSV file. You will:

- Create a batch job.
- Explore flow and record variable persistence across batch steps and phases.
- In the input phase, check for CSV files every second and convert them to a collection of objects.
- In the process records phase, create two batch steps for setting and tracking variables.
- In the on complete phase, look at the number of records processed and failed.
- Look at the thread used to process each record in each step.



Create a batch job

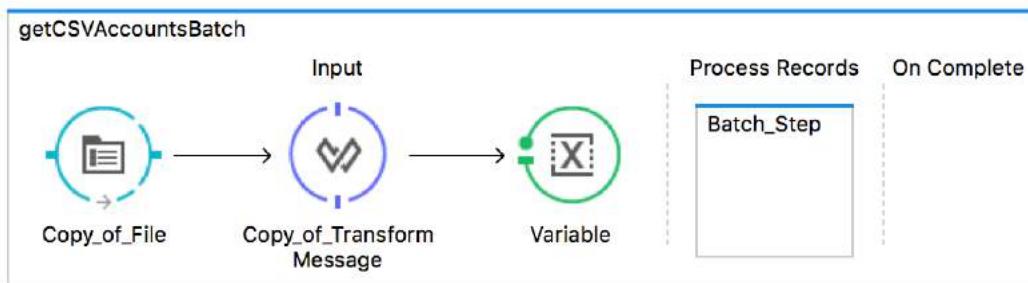
1. Return to accounts.xml.
2. Drag a Batch scope element from the Mule Palette and drop it above getCSVAccountsFlow.
3. Change the batch job name to getCSVAccountsBatch.



Add elements to the input phase

4. Select the File and Transform Message elements in getCSVAccountsFlow and select Edit > Copy.
5. Click the input section of the batch job and select Edit > Paste.

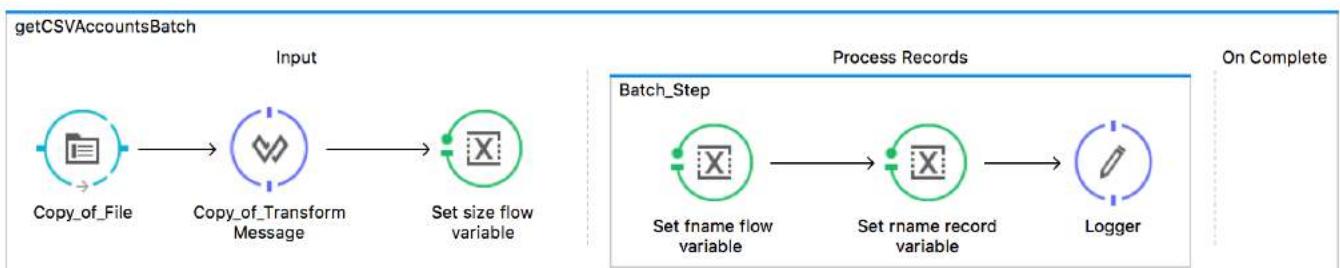
- Add a Variable transformer at the end of input phase.



- In the Variable properties view, change the display name to Set size flow variable and select Set Variable.
- Set the name to size and the value to #[payload.size()].

Add processors to a batch step in the process records phase

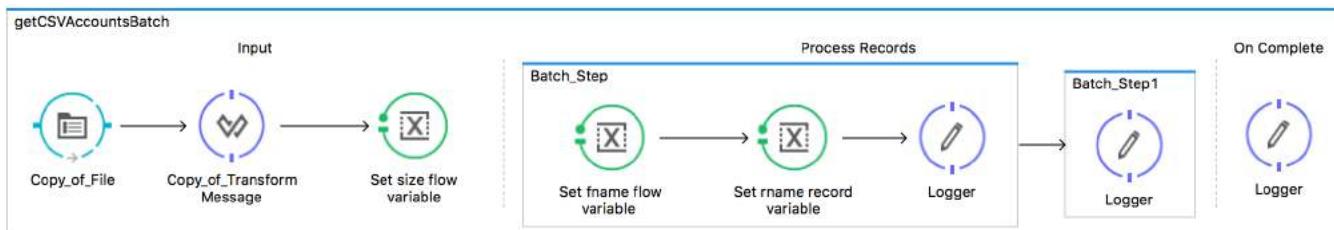
- Add a Variable transformer to the batch step in the process records phase.
- In the Variable properties view, change the display name to Set fname flow variable and select Set Variable.
- Set the name to fname and the value to #[payload.Name].
- Add a Record Variable transformer to the batch step.
- Change the display name to Set rname record variable.
- In the Record Variable properties view, select Set Record Variable.
- Set the name to rname and the value to #[payload.Name].
- Add a Logger component to the batch step.
- Set its message to #[recordVars.rname].



Create a second batch step

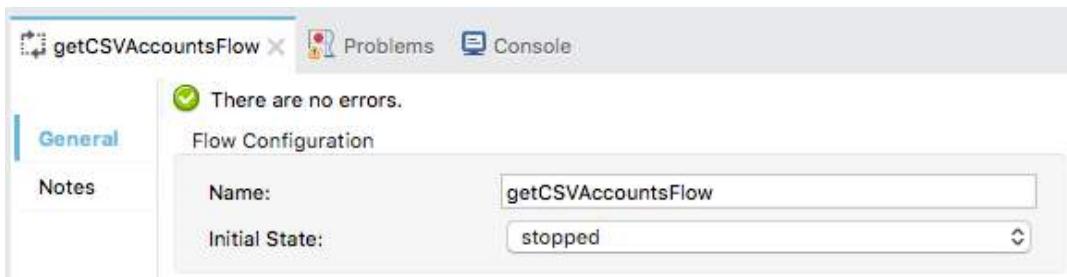
- Drag a Batch Step scope from the Mule Palette and drop it in the process records phase after the first batch step.

19. Add a Logger to the second batch step.
20. Set its message to #[recordVars.rname].
21. Add a Logger to the on complete phase.



Stop the other flow watching the same file location from being executed

22. Double-click getCSVAccountsFlow.
23. In the getCSVAccountsFlow properties view, set the initial state to stopped.



Debug the application

24. In getCSVAccountsBatch, add a breakpoint to the Transform Message component.
25. Debug the project.
26. After the application starts, drag the accounts.csv file from the project's src/main/resources/output folder to the src/main/resources/input folder.
27. In the Mule Debugger view, watch the payload value.
28. Step into the process records phase.
29. Click the Variables tab; you should see the size flow variable.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
③ fname	Dog Park Industries	java.lang.String		
③ moveToDirectory	src/main/resources...	java.lang.String		
③ originalDirectory	/Users/jeanette.sta...	java.lang.String		
③ originalFilename	accounts.csv	java.lang.String		
③ size	3	java.lang.Integer		

30. Step to the Logger in the first batch step.

31. Click the Record tab; you should see the rname flow variable.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
⑧ rname	Dog Park Industries	java.lang.String		

32. Step through the rest of the records in the first batch step and watch the payload, the fname flow variable, and the rname record variable.

33. Step into the second batch step and look at the payload, the flow variables, and the record variables; you should see the rname record variable and size flow variable but not the fname flow variable.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
⑧ batchJobInstan...	9478c300-500c-11e5-...	java.lang.String		
⑧ moveToDirectory	src/main/resources/ou...	java.lang.String		
⑧ originalDirectory	/Users/jeanette.stallon...	java.lang.String		
⑧ originalFilename	accounts.csv	java.lang.String		
⑧ size	3	java.lang.Integer		

34. Step through the rest of the records in the second batch step.

35. Step into the on complete phase; you should see the payload is an ImmutableBatchJobResult.

36. Locate the values for totalRecords, successfulRecords, and FailedRecords.

Mule Properties	Problems	Console	Mule Debugger
Name	Value	Type	
⑧ DataType	SimpleDataType{type=com.mulesoft.module.batch.ImmutableBatchJobResult}	org.mule.transformer.types.SimpleType	
⑧ Exception	null		
⑧ Message		org.mule.DefaultMuleMessage	
⑧ Message Processor	Logger	org.mule.api.processor.Logger	
⑧ Payload (mimeType="*/*", encoding="UTF-8")	com.mulesoft.module.batch.ImmutableBatchJobResult@2...	com.mulesoft.module.batch.ImmutableBatchJobResult	
⑧ batchJobInstanceId	547d5740-2b74-11e6-a3b1-985aeb897c14	java.lang.String	
⑧ elapsedTimeInMillis	135123	java.lang.Long	
⑧ failedOnCompletePhase	false	java.lang.Boolean	
⑧ failedOnInputPhase	false	java.lang.Boolean	
⑧ failedOnLoadingPhase	false	java.lang.Boolean	
⑧ failedRecords	0	java.lang.Long	
⑧ inputPhaseException	null	java.lang.Exception	
⑧ loadedRecords	3	java.lang.Long	
⑧ loadingPhaseException	null	java.lang.Exception	
⑧ onCompletePhaseException	null	java.lang.Exception	
⑧ processedRecords	3	java.lang.Long	
⑧ serialVersionUID	4323747859995526737	java.lang.Long	
⑧ stepResults	{Batch_Step=com.mulesoft.module.batch.ImmutableBatchJobResult@2...}	java.util.Collections.UnmodifiableMap	
⑧ successfulRecords	3	java.lang.Long	
⑧ totalRecords	3	java.lang.Long	

37. Step to the end of the application.

38. Stop the project.

Look at the processing threads

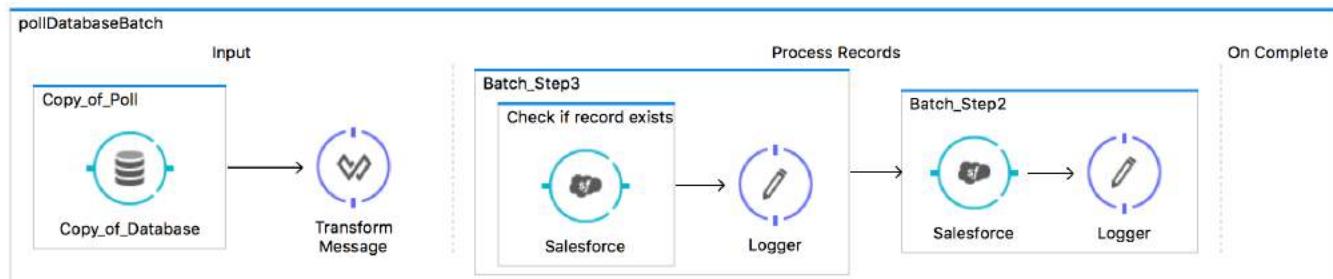
39. In the console, locate the thread number used to process each record in the collection in each step of the batch process.

```
INFO 2016-06-05 16:32:46,099 [[apdev-examples].connector.file.mule.default.receiver.01] com.mulesoft.module.batch.engine.queue.BatchQueueLoader: Finished loading phase for instance ce37a4e0-2b75-11e6-beca-985aeb897c14 of job getCSVAccountsBatch. 3 records were loaded
INFO 2016-06-05 16:32:46,100 [[apdev-examples].connector.file.mule.default.receiver.01] com.mulesoft.module.batch.engine.DefaultBatchEngine: Started execution of instance 'ce37a4e0-2b75-11e6-beca-985aeb897c14' of job 'getCSVAccountsBatch'
INFO 2016-06-05 16:32:46,150 [batch-job-getCSVAccountsBatch-work-manager.01] org.mule.api.processor.LoggerMessageProcessor: Dog Park Industries
INFO 2016-06-05 16:32:46,157 [batch-job-getCSVAccountsBatch-work-manager.01] org.mule.api.processor.LoggerMessageProcessor: Iguana Park Industries
INFO 2016-06-05 16:32:46,157 [batch-job-getCSVAccountsBatch-work-manager.01] org.mule.api.processor.LoggerMessageProcessor: Cat Park Industries
INFO 2016-06-05 16:32:46,171 [batch-job-getCSVAccountsBatch-work-manager.01] com.mulesoft.module.batch.DefaultBatchStep: Step Batch_Step finished processing all records for instance ce37a4e0-2b75-11e6-beca-985aeb897c14 of job getCSVAccountsBatch
INFO 2016-06-05 16:32:46,173 [batch-job-getCSVAccountsBatch-work-manager.02] org.mule.api.processor.LoggerMessageProcessor: Dog Park Industries
INFO 2016-06-05 16:32:46,173 [batch-job-getCSVAccountsBatch-work-manager.02] org.mule.api.processor.LoggerMessageProcessor: Iguana Park Industries
INFO 2016-06-05 16:32:46,174 [batch-job-getCSVAccountsBatch-work-manager.02] org.mule.api.processor.LoggerMessageProcessor: Cat Park Industries
INFO 2016-06-05 16:32:46,183 [batch-job-getCSVAccountsBatch-work-manager.02] com.mulesoft.module.batch.engine.DefaultBatchEngine: Starting execution of onComplete phase for instance ce37a4e0-2b75-11e6-beca-985aeb897c14 of job getCSVAccountsBatch
```

Walkthrough 12-3: Create a batch job to synchronize records from a database to Salesforce

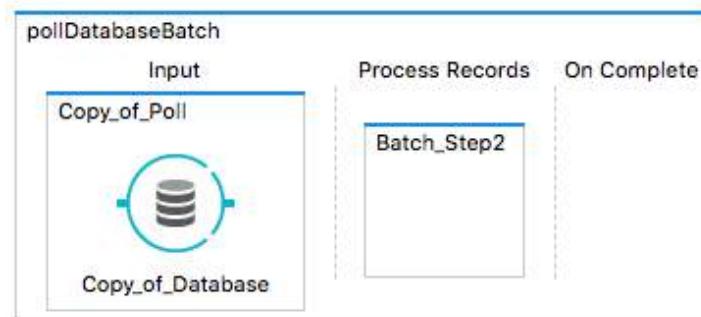
In this walkthrough, you create a batch job to retrieve records from a legacy database and insert them into Salesforce if they do not already exist. You will:

- Create a batch job that polls a database for records with a specific postal code.
- Use a Message Enricher scope to check if a record already exists in Salesforce (an account with the same Name) and stores the result in a record variable and retains the original payload.
- Add a second batch step with a filter that only allows new records (records that don't already exist in Salesforce) to be added to Salesforce.
- Use a batch commit scope to commit records in batches.



Create a batch job that polls a database

1. Return to accounts.xml.
2. Drag a Batch scope from the Mule Palette and drop it above pollDatabaseFlow.
3. Change the flow name to pollDatabaseBatch.
4. Copy the Poll scope in pollDatabaseFlow and paste it into the input phase of pollDatabaseBatch.



Log each record to the console in the process records phase

5. Add a Logger component to the batch step in the process records phase.

6. Set its message to display each record – the message payload in that phase – on a new line.

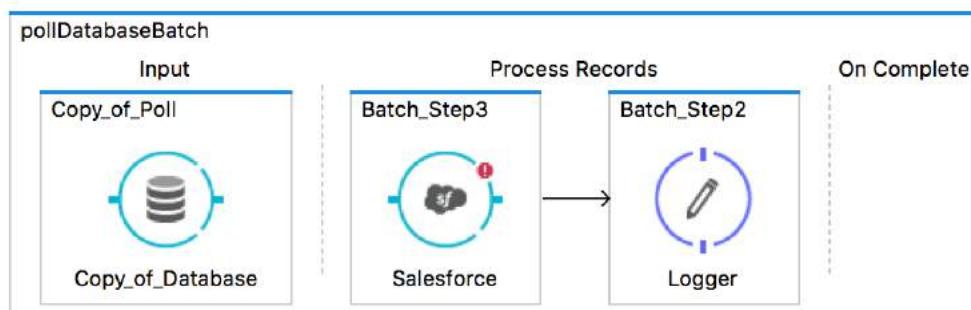
```
#['\n\nRECORD: ' + payload]
```

Test the application

7. Select Run > Run Configurations.
8. In the Run Configurations dialog box, set clear application data to prompt.
9. Click Run.
10. In the Clear Application Data dialog box, click Yes.
11. Watch the console; you should see the same behavior as for pollDatabaseFlow: records with the specific postal code are displayed once and then every 5 seconds no new records are returned.

Add logic to check if a record already exists in Salesforce

12. Drag a Batch Step scope from the Mule Palette and drop it at the beginning of the process records phase before the existing batch step.
13. Add a Salesforce endpoint to the new batch step.



14. Configure the Salesforce endpoint to use the existing Salesforce connector configuration.
15. Set the operation to Query.
16. Click the Query Builder button.
17. Set the type to Account (Account).
18. Select a field of Name; it does not matter what you select, you just want to see if any records are returned.
19. Click the Add Filter button.

20. Create a filter to check if the Name field in the record being processed already exists in the database.

Name = #[payload.Name]

Note: Right now, you can use payload.name or payload.Name because the payload is a caseSensitiveHashMap. Later this walkthrough, however, you will transform the Map to an Account object with a Name property and will have to refer to this as payload.Name.

The screenshot shows the 'Query Builder' window. In the 'Types' section, 'Account (Account)' is selected. In the 'Fields' section, 'Name' is checked. The 'Filter' section contains a condition: 'Name' is equal to '#[payload.Name]'. The 'Basic' radio button is selected. At the bottom, there are 'Cancel' and 'OK' buttons, with a question mark icon next to the 'Cancel' button.

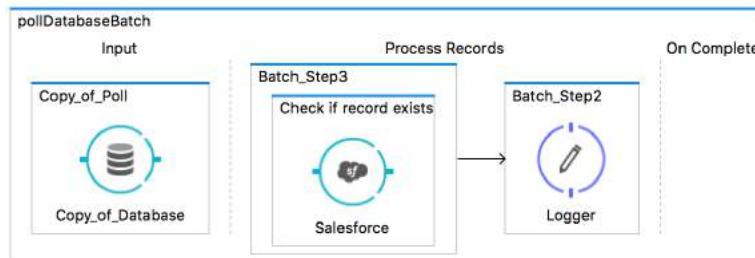
21. Click OK.

22. Review the generated query.

The screenshot shows the Mule Studio interface with the 'Salesforce' tab selected. In the top navigation bar, there are links for 'Problems', 'Console', and 'Mule Debugger'. On the left, a sidebar has tabs for 'General', 'Advanced', and 'Notes', with 'General' currently selected. The main area displays a query configuration. It shows 'Connector Configuration: Salesforce_Basic_Authentication' and 'Operation: Query'. The 'Language:' dropdown is set to 'DataSense Query Language'. Below this is a text input field containing the query: '1 SELECT Name FROM Account WHERE Name = '#[payload.Name]'

Place the logic in a message enricher

23. Add a Message Enricher scope element to the first batch step.
24. Move the Salesforce endpoint so it is inside the message enricher.
25. Change the message enricher display name to Check if record exists.



26. In the message enricher properties view, set the source to an expression that checks to see if any records were returned, i.e. if there is a payload.

```
##[payload.size() > 0]
```

27. Set the target to assign the result of the source expression to a record variable called exists.

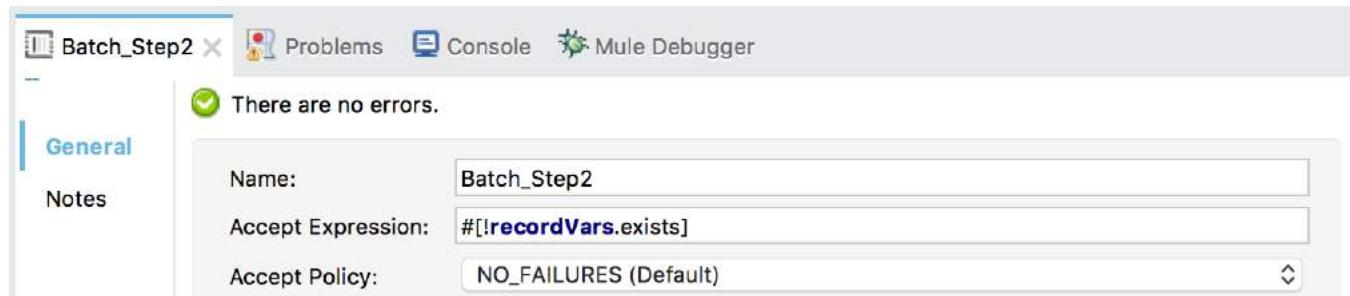
```
##[recordVars.exists]
```

The screenshot shows the properties view for the 'Check if record exists' message enricher. At the top, it says 'There are no errors.' The 'General' tab is selected. Under 'Display Name:', the value is 'Check if record exists'. Under 'Generic', there are two fields: 'Source:' with the expression '##[payload.size() > 0]' and 'Target:' with the expression '##[recordVars.exists]'

Set a filter for the insertion step

28. Navigate to the Properties view for the second batch step.
29. Set the accept expression so that only records that have a record variable exists set to false are processed.

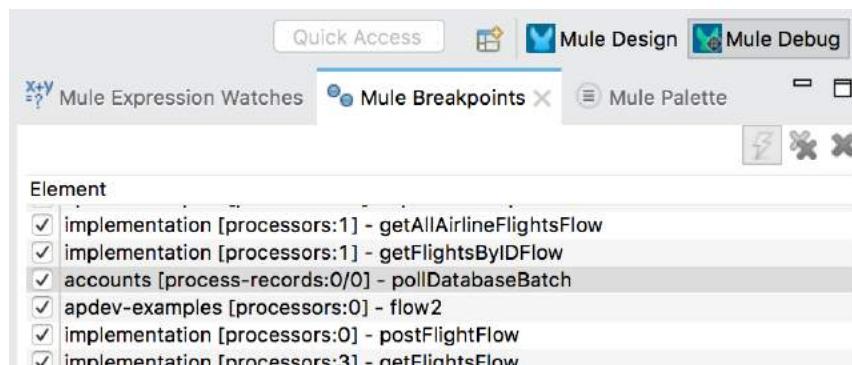
```
#![!recordVars.exists]
```



Test the application

30. Add a breakpoint to the message enricher.

Note: In Anypoint Studio 6.0.0, the breakpoint does not appear in the canvas after you add it. You can confirm it was added by looking in the Mule Breakpoints view. In the Mule Debug perspective, this view can be found next to the Mule Palette.



31. Add a Logger after the Message Enricher scope in the first batch step.



32. Debug the project and clear the application data.
33. Step through the application and watch the record variables; you should see exists set to false for records with names that don't exist in Salesforce and true for those that do.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
exists	false	java.lang.Boolean		

34. Stop the project.

Add an account to Salesforce manually with a name matching one of the database records

35. In a web browser, navigate to <http://login.salesforce.com/> and log in with your Salesforce Developer account.
36. Click the Accounts link in the main menu bar.
37. In the view drop-down menu, select All Accounts with Postal Code and click the Go button.
38. Click the New Account button.
39. Enter an account name (one that matches one of the accounts you added with your postal code to the MySQL database) and click Save.

 **New Account**

Account Edit		Save	Save & New	Cancel
Account Information				
Account Owner	Max Mule			
Account Name	Molly Mule <input data-bbox="665 1396 693 1431" type="button" value="..."/>			
Parent Account	<input data-bbox="665 1438 726 1474" type="text"/> <input data-bbox="693 1438 721 1474" type="button" value="..."/>			
Account Number	<input data-bbox="665 1480 726 1516" type="text"/>			
Account Site	<input data-bbox="665 1522 726 1558" type="text"/>			
Type	<input data-bbox="665 1564 742 1600" type="button" value="--None--"/>			
Industry	<input data-bbox="665 1607 742 1643" type="button" value="--None--"/>			
Annual Revenue	<input data-bbox="665 1649 726 1685" type="text"/>			

Test the application

40. Return to Anypoint Studio.
41. Debug the project and clear the application data.

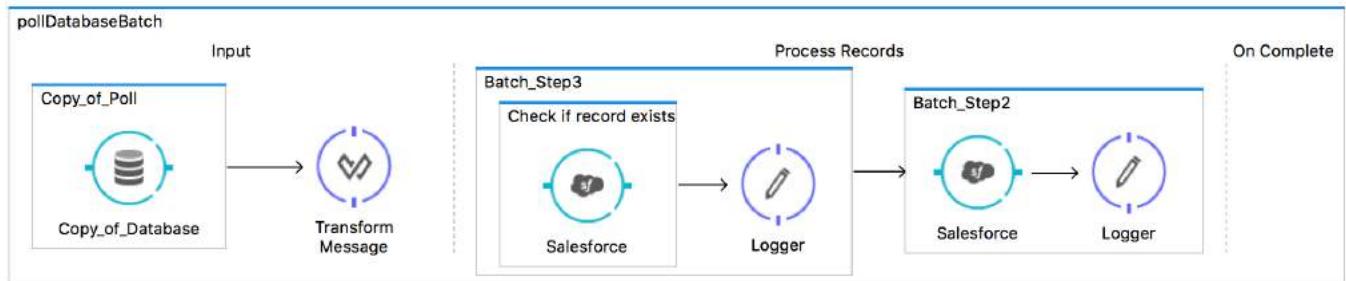
42. Step through the application and watch the record variables; you should now get one record that gets its exists record variable set to true.

Inbound	Variables	Outbound	Session	Record
	Name @ exists Value true Type java.lang.Boolean			

43. Stop the project.

Add an endpoint to add new account records to Salesforce

44. Add a Salesforce endpoint to the second batch step in the processing phase.



45. Configure the Salesforce endpoint to use the existing Salesforce connector configuration.

46. Set the operation to Create.

47. Set the sObject type to Account (Account).

48. Leave the sObject field mappings to from expression #[payload].

The screenshot shows the Mule Studio interface with a 'Salesforce' endpoint selected. The 'General' tab is active, displaying the following configuration:

- Display Name: Salesforce
- Connector Configuration: Salesforce_Basic_Authentication
- Operation: Create
- sObject Type: Account (Account)
- sObject Field Mappings:
 - sObjects: From Expression #[payload]
 - Create Objects manually (radio button is unselected)

49. Look at the input tab of the Metadata explorer; you should see a problem letting you know that payload is a Map in this step but the outbound Salesforce endpoint is expecting a List of Account objects.

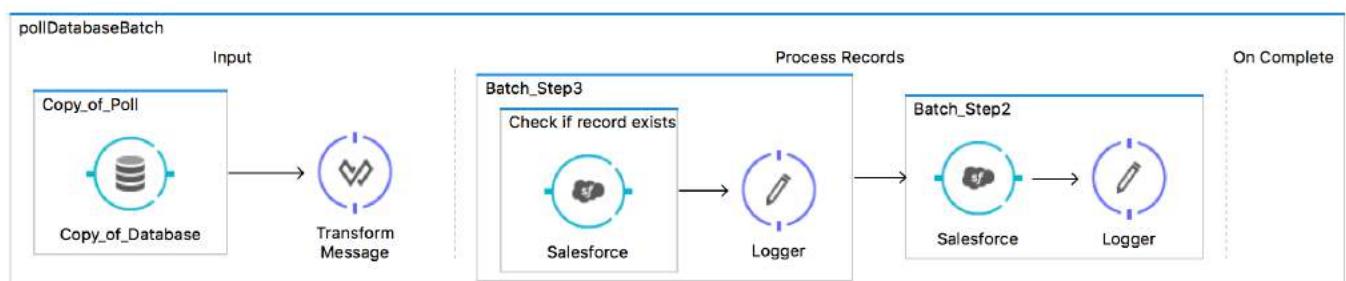
50. Drill-down in both values and look at the fields.

The screenshot shows the MuleSoft Metadata Explorer interface. The top navigation bar has tabs for 'Input' and 'Output'. Below the tabs is a search bar with placeholder text 'Q type filter text'. Under the 'Input' tab, there is a section titled '(Actual) List<Map> : List' which contains fields: accountID (Integer), city (String), country (String), name (String), postal (String), state (String), and street (String). Under the 'Output' tab, there is a section titled '(Expected) List<Account> : List' which contains fields: AccountNumber (String), AccountSource (Enum), Active_c (Enum), AnnualRevenue (String), BillingCity (String), BillingCountry (String), and BillingGeocodeAccuracy (Enum).

Transform the objects into the Account objects that the endpoint is expected

51. Add a Transform Message component after the poll scope in the input phase.

Note: You want to transform all the records at once in the input phase – not one by one in the processing phase.



52. In the Transform Message properties view, look at the input section and see the input has associated metadata.

53. Look at the output section; it has no metadata and no way to add any.



54. Copy the Salesforce endpoint in the second batch step and paste it after the Transform Message component in the batch input phase.

Note: You are temporarily adding this Salesforce endpoint after the database endpoint so DataSense will work to create the mappings. You could instead add the DataWeave transformation to the process records phase, but this would add additional overhead.



55. Look at the output section of the Transform Message properties view again; you should now see metadata for a List of Account objects.



56. Drag properties from the input section to the output section to make the following mappings:

- country to BillingCountry
- city to BillingCity
- street to BillingStreet
- name to Name
- state to BillingState
- postal to BillingPostalCode

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' tab displays a list of properties: accountID, country, city, street, name, state, postal, lastAccountID, and Session Variables. The 'Output' tab shows a list of fields: Id, IsDeleted, MasterRecordId, Name, Type, ParentId, BillingStreet, BillingCity, BillingState, BillingPostalCode, and BillingCountry. Graphical mappings are shown as lines connecting properties like 'country' to 'BillingCountry' and 'city' to 'BillingCity'. The 'Output' tab also contains a DataWeave expression:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map ((payload01 , indexOfPayload01) -> {
5   Name: payload01.name,
6   BillingStreet: payload01.street,
7   BillingCity: payload01.city,
8   BillingState: payload01.state,
9   BillingPostalCode: payload01.postal,
10  BillingCountry: payload01.country
11 })
```

57. Delete the Salesforce endpoint in the input phase.

58. Look at the mapping in the Transform Message properties view; the graphical mapping is gone but the DataWeave expression is still there.

This screenshot shows the same 'Transform Message' component after the graphical mapping has been removed. The 'Input' tab lists the same properties as before. The 'Output' tab shows the same fields, but the graphical connections between them are no longer present. The DataWeave expression remains the same:

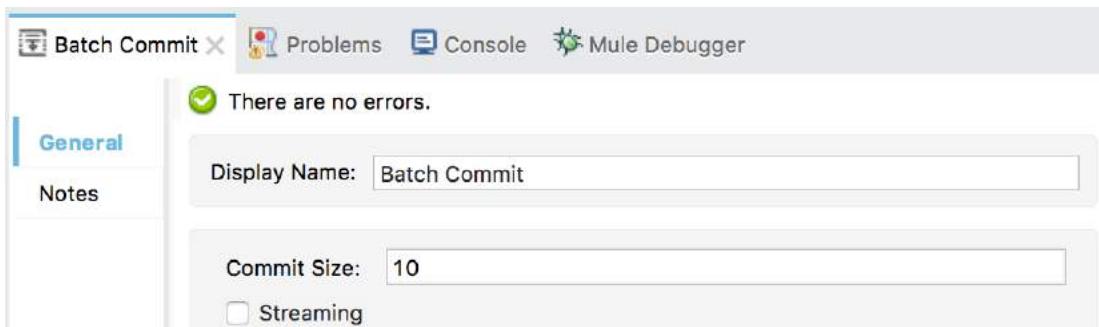
```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map ((payload01 , indexOfPayload01) -> {
5   Name: payload01.name,
6   BillingStreet: payload01.street,
7   BillingCity: payload01.city,
8   BillingState: payload01.state,
9   BillingPostalCode: payload01.postal,
10  BillingCountry: payload01.country
11 })
```

Create the List that the endpoint is expecting

59. Return to the Properties view for the Salesforce endpoint in the second batch step.
60. Look at the input tab of the Metadata explorer; you should still see a problem: the payload is a Map but the outbound Salesforce endpoint is expecting a List of Account objects.
61. Drag a Batch Commit scope from the Mule Palette and drop it in the second batch step.
62. Add the Salesforce endpoint to it.



63. In the Batch Commit properties view, set the commit size to 10.



Test the application

64. Run the project and clear the application data.
65. Check the console and make sure you see your new records processed.

66. Return to Salesforce in a web browser 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"/> Action	Account Name ↑	Billing Country	Billing Zip/Postal Co...	Billing State/Province
<input type="checkbox"/> Edit Del	Burlington Textiles Corp of America	USA	27215	NC
<input type="checkbox"/> Edit Del	Dickenson plc	USA	66045	KS
<input type="checkbox"/> Edit Del	Edge Communications			TX
<input type="checkbox"/> Edit Del	Express Logistics and Transport			OR
<input type="checkbox"/> Edit Del	GenePoint			CA
<input type="checkbox"/> Edit Del	Grand Hotels & Resorts Ltd			IL
<input type="checkbox"/> Edit Del	John Doe	USA	94108	CA
<input type="checkbox"/> Edit Del	Molly Mule			

67. Run the application again but do not clear the application data; no records should be processed.

68. Return to salesforce.com and locate your new record(s); they should have been inserted only once.

69. Return to Anypoint Studio and stop the project.