MuleSoft®

What's new with Mule 4?

Contents

Introduction: Meet Mule 4

Mule achieved amazing popularity in the 7 years since Mule 3.0 was released. Today, Mule is used by over 1,100 enterprise customers located in more than 60 countries across every major industry, and has a community of over 175K developers. With these metrics in mind, the strong reputation of Mule 3 was top of mind as Mule 4 was being developed.

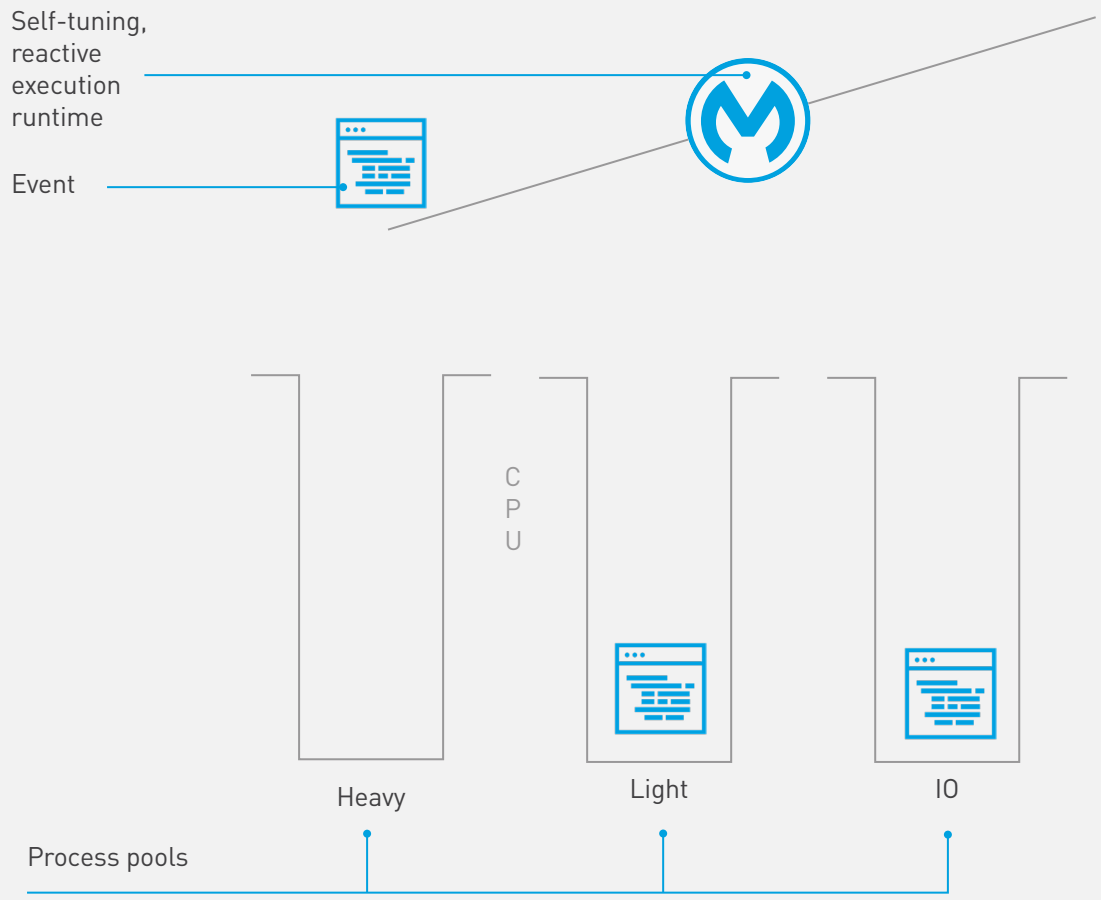
Mule 4 is a major evolution of the core runtime engine behind Anypoint Platform, and delivers innovation while keeping true to Mule fundamentals. Mule 4 enhances application development velocity with up to 50% fewer steps and concepts to learn, including: a simplified language for connectivity, a new error handling framework, consistent methods for accessing data, automatic tuning, and smoother upgrade processes.

Mule 4 is built with the fundamentals of Mule 3 in mind. The development evaluated ease of use, task simplification, runtime optimization and tuning, and long-term maintenance costs to ensure Mule 4 can power application networks by:

- Improving speed of delivery and accelerating developer on-ramp through a simplified language, error handling, and event and message model.
- Managing streams and larger-than-memory payloads transparently.
- Generating connectors automatically from RAML specs with REST connect.
- Creating powerful custom connectors and components with Mule SDK.
- Providing a guided development experience for integration novices with flow designer.
- Simplifying upgrades with classloader isolation.

Mule 4 is that next generation runtime engine, and we are excited by all that it can do.

Self-tuning engine



Mule 4 ships with a new reactive, non-blocking execution engine. This engine makes it possible to achieve optimal performance without having to do manual tuning steps, such as declaring exchange patterns, processing strategies or threading configuration. This is a task-oriented execution model that allows you to take advantage of non-blocking IO calls and avoid performance problems due to incorrect processing strategy configurations.

The Message processor can now inform the runtime if an operation is a CPU intensive, CPU light, or IO. Then the runtime self-tunes, or sizes the pools automatically, for different workloads dynamically. This removes the need for users to manage thread pools manually. As a result, Mule 4 removes complex tuning requirements to achieve optimum performance.

DataWeave

Mule 4 introduces DataWeave as the default expression language, replacing Mule Expression Language (MEL) with a scripting and transformation engine. In prior versions of Mule, there were a variety of evaluators with MEL to handle different inputs, like Groovy and JSON. While MEL handled these expressions and created a consistent experience to deal with these expressions, it did not handle transformations.

Extensive efforts were made by the DataWeave team on the language version 2.0. The team not only supported the runtime integration, but also worked to improve the language itself. This results in innovations like:

- *Reusing code broadly*: It is easy to now package and import scripts into others, enabling users to reuse and share code.
- *Access data without transformations*: Users no longer need to convert data into a set of objects to access it; instead, they can write expressions to access the data stored in CSV, JSON, XML or other forms directly.
- *Simplified syntax*: All operators are now functions, making DataWeave easier to learn.
- *Java interoperability*: Static methods can be executed through DataWeave.
- *New advanced capabilities*: Call Java functions directly, use multi-line comments, and define function types and variable types with type inference.

Transformers and DataMapper were the tools available for basic manipulation. But as the integration landscape grew, the amount of transformations and the complexity behind those transformations grew as well. DataWeave was introduced in 2015 to handle more complex transformations with high

performance; with a highly performing transformation engine and rich querying capabilities—DataWeave became a hit.

With DataWeave, the expressions are focused on the structure of our data, rather than its format. This is because a Java array is the same as a JSON one in DataWeave; in other words, users no longer need different expressions to handle them. Users can now query and evaluate any expression without first transforming it into a Java object.

Binary data can now be accessed anywhere with larger-than-memory, random, and repeatable access.

Mule 4 runtime gives DataWeave all data regarding the current execution, including payload, variables, expected output, and metadata. This enables DataWeave to know, for example, whether a String or a map is required, how to handle each variable, whether to coerce a type, and more. Simple one-line expressions are written as:

```
1  #[payload ++ variables.myStatus]
2  #[attributes.statusCode]
```

In this example, the payload, variables, and attributes keywords will be interpreted as their respective form.

Set an HTTP request header with DataWeave

```
1  #[output application/java --- payload ++ {host: 'httpbin.
2  org'}]
```

How does this work? DataWeave infers the output type, rather than relying on the user to explicitly declare the output format. While users can declare the output type, it is not necessary. A JSON payload now sets an HTTP request header, taking DataWeave's existing map of headers and adding it to the script.

Now, when a request is made, the backend will answer with the received headers that contain the values sent to the HTTP listener as a body and the added host.

Full integration

Improvements around routing and attribute one-liners is not the only improvement with DataWeave. Another simplification is for flows. The number of transform elements needed have been decreased by defining content "inline." For example, when building the content of a file *inside* the File connector's write component, there is no need to use a 'transform' component in order to get the payload beforehand. The user does not need *additional steps* to iterate the received JSON payload; the new file path is determined with the expression:

```
1   #[payload.name ++ '.' ++ dataType.mimeType.subType]
```

Also, users can add a desired "date" attribute within the write operation, exactly where it's needed, setting the content to:

```
1   #[payload ++ { date : now() }]
```

That last expression is a great example of the output type being inferred. Since the user knows the payload is JSON, there is no need to specify the output; the generated files will be in that format as well.

This works for all new connectors since they're supported by the new Mule SDK. In the example below, the HTTP body of a request is set with a script, where users could take advantage

of all of DataWeave's features—as in any transform component script:

```
#[
%dw 2.0

output application/json
---
payload ++ {location : 'LATAM', office : 'BA'}
]
```

Additionally, the listener response body can be configured with the expression:

```
#[payload.data]
```

This is because the backend server will return a JSON, where that attribute represents the payload sent to it. So, the data received by the listener will be modified to include some more attributes, and later forwarded back.

Compatibility with Mule Expression Language

DataWeave is the primary and default expression language for Mule 4. MEL is deprecate, however every expression can feature the `mel:` prefix to indicate that it should be evaluated with MEL.

For example, the HTTP listener below will answer whether a variable starts with "SUCCESS" or not, using MEL to configure the response body with the following expression:

```
#[mel:flowVars.receivedMessage.startsWith('SUCCESS').
toString()]
```

Error handling

Error handling in Mule 4 is different from Mule 3. Before, error handling was Java exception handling, without a way of communicating what kind of error each component threw. It was also a "trial and error" experience, where users would check source code frequently and force the error several times to see what happened.

Error handling with Mule 4 introduces a new experience

While Java Throwables are still available, the main error handling mechanism in Mule 4 is based on the Mule runtime engine's own errors: if something goes wrong with a component, Mule provides a clear error that is typed accordingly, with useful data on the problem. The problem can then be easily routed to a handler.

More importantly, each component declares the type of errors it may throw, so users are able to identify all potential errors at design time.

Mule errors

Execution failures are represented with Mule errors that have the following components:

- A description regarding the problem
- A type, used to characterize the problem
- A cause, the underlying Java Throwable that resulted in the failure
- An optional error message, which is used to include a proper Mule Message regarding the problem

For example, when an HTTP request fails with a 401 status code, a Mule Error provides the following information:

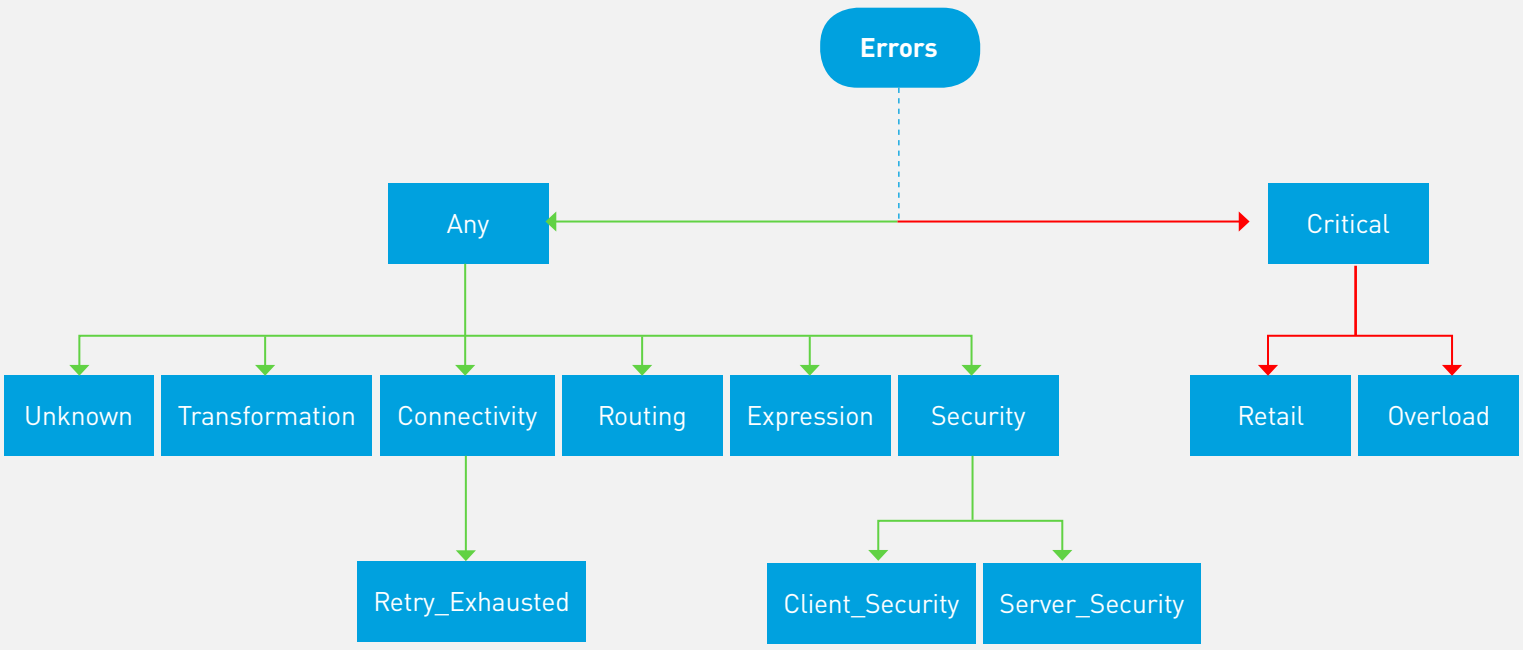
```
Description: HTTP GET on resource 'http://localhost:36682/
testPath' failed: unauthorized (401)

Type: HTTP:UNAUTHORIZED

Cause: a ResponseValidatorTypedException instance

Error Message: The entire HTTP 401 response received, including
the body and headers
```

Error types

In the example above, the error type is `HTTP:UNAUTHORIZED`, not just `UNAUTHORIZED`. This is because error types consist of a namespace and an identifier, allowing users to distinguish the types according to their domain, like `HTTP:NOT_FOUND` and `FILE:NOT_FOUND`. While connectors define their own namespace, core runtime errors have an implicit one: `MULE:EXPRESSION` and `EXPRESSION` are interpreted as one namespace.

Another important characteristic of error types is that they may have a parent type. For example, `HTTP:UNAUTHORIZED` has `MULE:CLIENT_SECURITY` as the parent, which, in turn, has `MULE:SECURITY` as the parent. This establishes error types as specifications of more global ones: an HTTP unauthorized error is a kind of client security error, which is a type of a more broad security issue.

These hierarchies mean routing can be more general, since, for example, a handler for `MULE:SECURITY` will catch HTTP unauthorized errors as well as OAuth errors. Below is a visual of can see what the core runtime's hierarchy looks like:



All errors are either `GENERAL` or `CRITICAL`, the latter being so severe that they cannot be handled. At the top of the general hierarchy is `ANY`, which allows matching all types under it. The `UNKNOWN` type is used when no clear reason for the failure is found.

When it comes to connectors, each connector defines its error type hierarchy considering the core runtime one. However, `CONNECTIVITY` and `RETRY_EXHAUSTED` types are always present since they are common to all connectors.

The most important thing about these error types is that they are part of each component's definition, so during the design process, users will be aware of any failures that could occur.

Error handler

Mule 4 also introduces the error handler component, that can any number of internal handlers that route an error to the first one matching it.

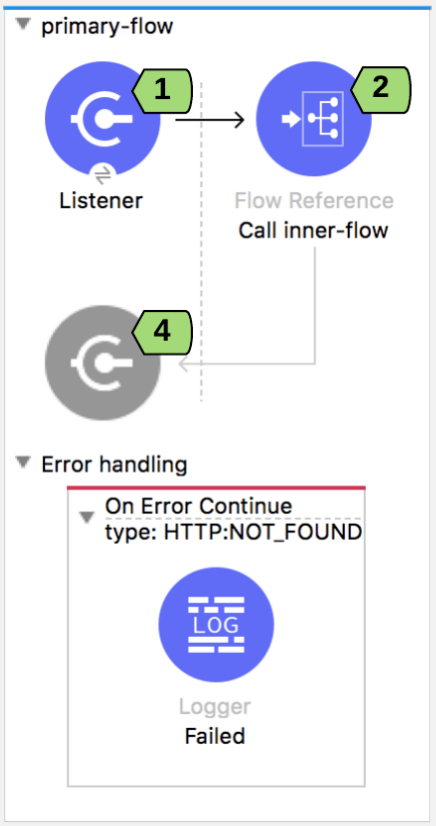
Such handlers are on-error-continue and on-error-propagate, which both allow conducting matching through an error type (or group of) or through an expression (*for advanced use cases*). These are similar but simpler to the choice, catch, and rollback exceptions strategies of Mule 3. If an error is raised within a flow, its error handler will be executed and the error will be routed to the matching handler. At this point, the error is

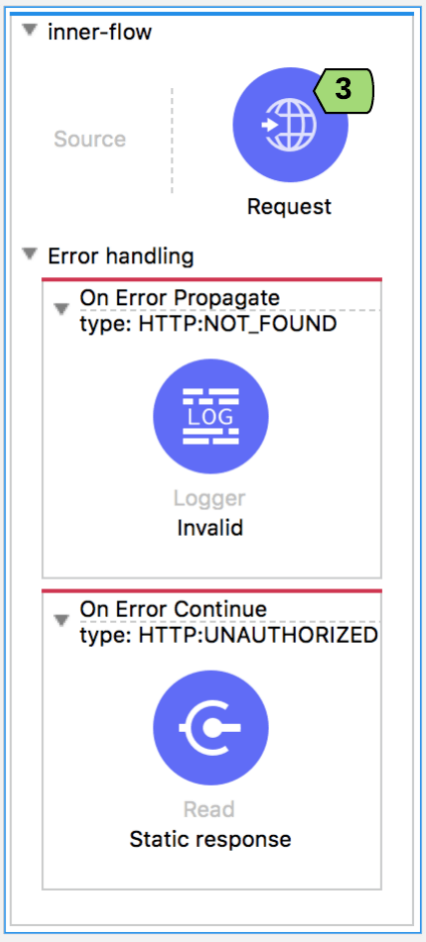
available for inspection, so the handlers could execute and act accordingly:

- An on-error-continue will execute and use the result of the execution, as the result of its owner—as if the owner had actually completed the execution successfully. Any transactions at this point would be committed as well.
- An on-error-propagate will rollback any transactions, execute, and use that result to rethrow the existing error—meaning its owner will be considered as "failing."

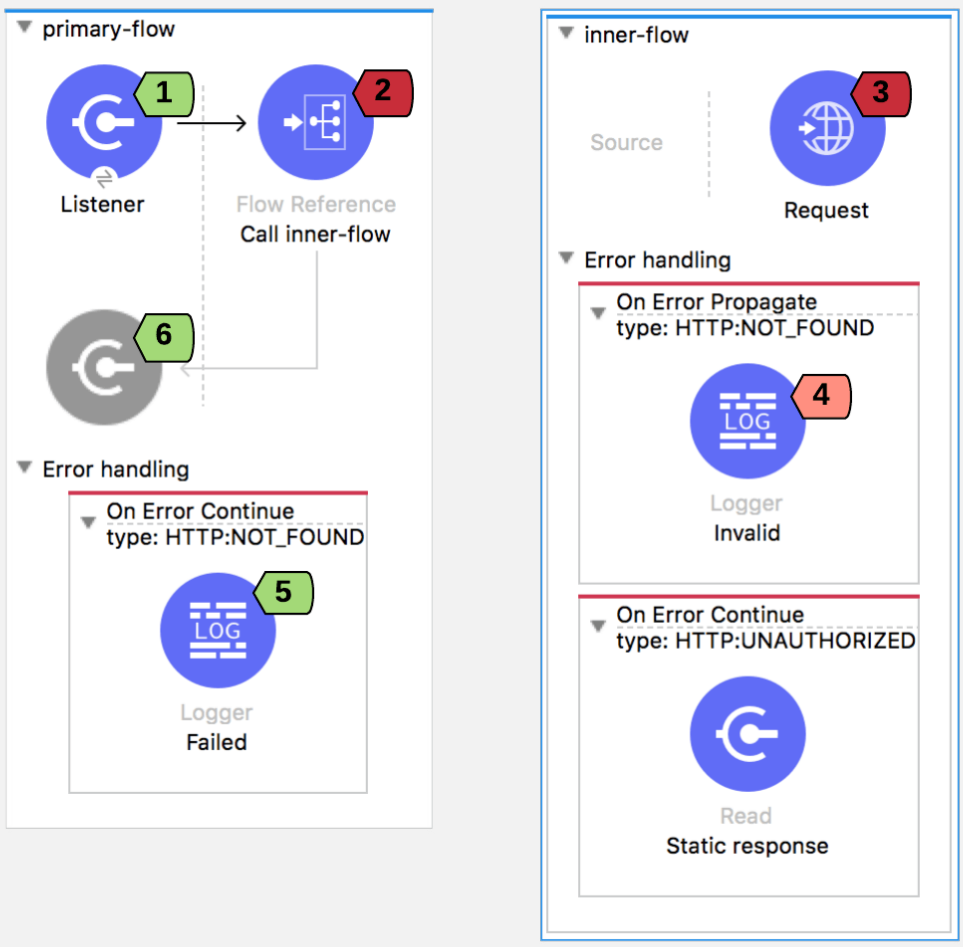
Consider a few examples to see how this works. First, consider the following application where an HTTP listener triggers a flow reference to another flow that performs an HTTP request.

If everything goes right when a message is received, (1) the reference is triggered (2) the request performed (3), then a successful response is expected (4).

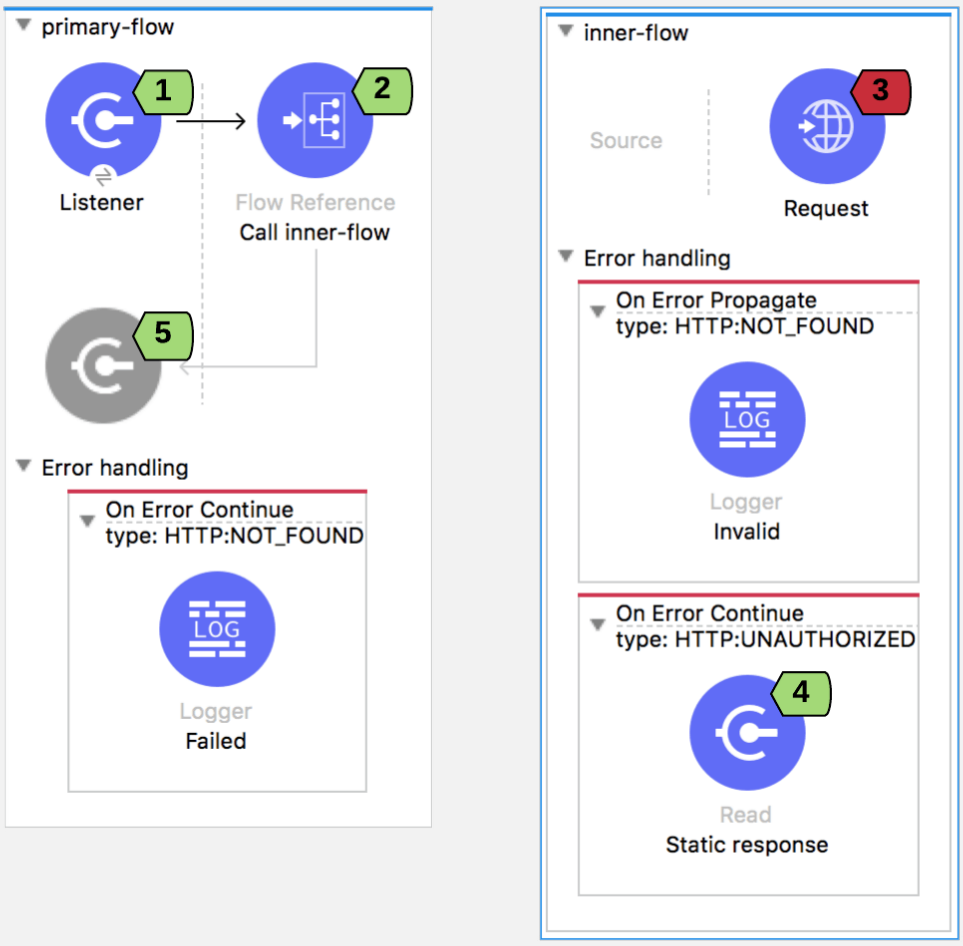




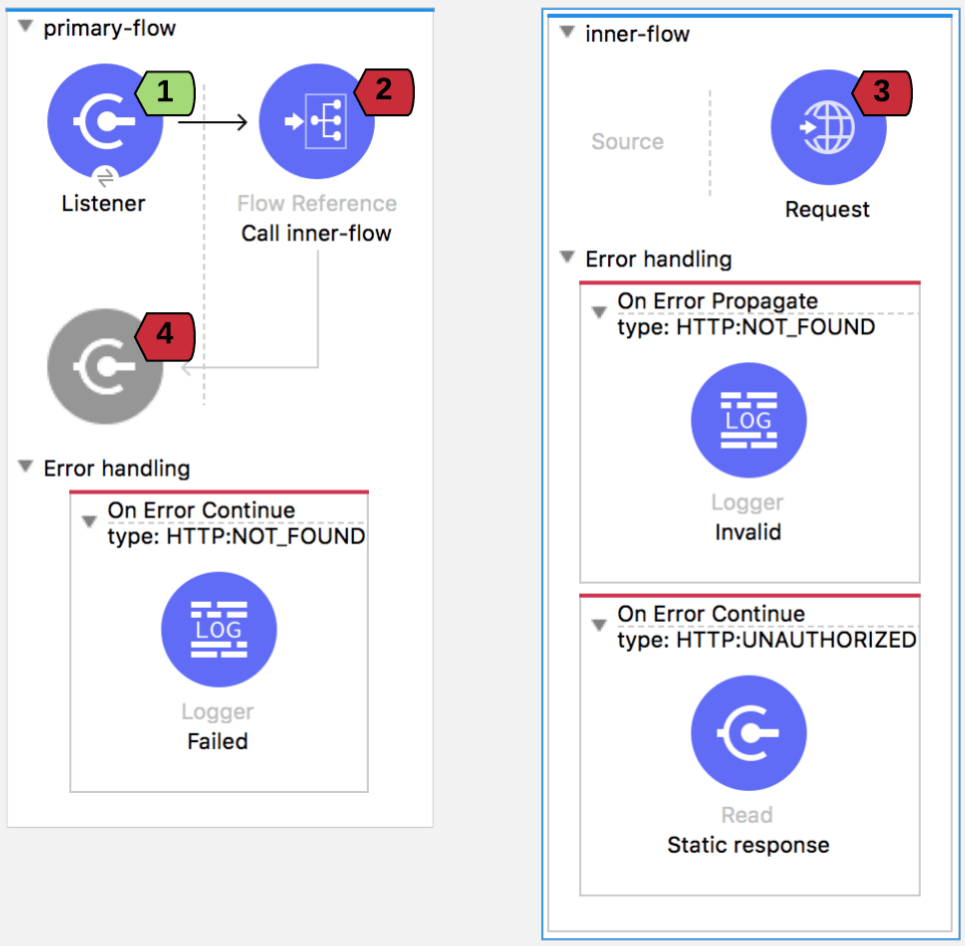
If the HTTP request fails with a not found error (3), because of the error handler setup of inner-flow, then the error will be propagated (4), and the flow reference will fail as well (2). However, since primary-flow is handling that with an on-error-continue, this will execute (5) and a successful response will be returned (6).



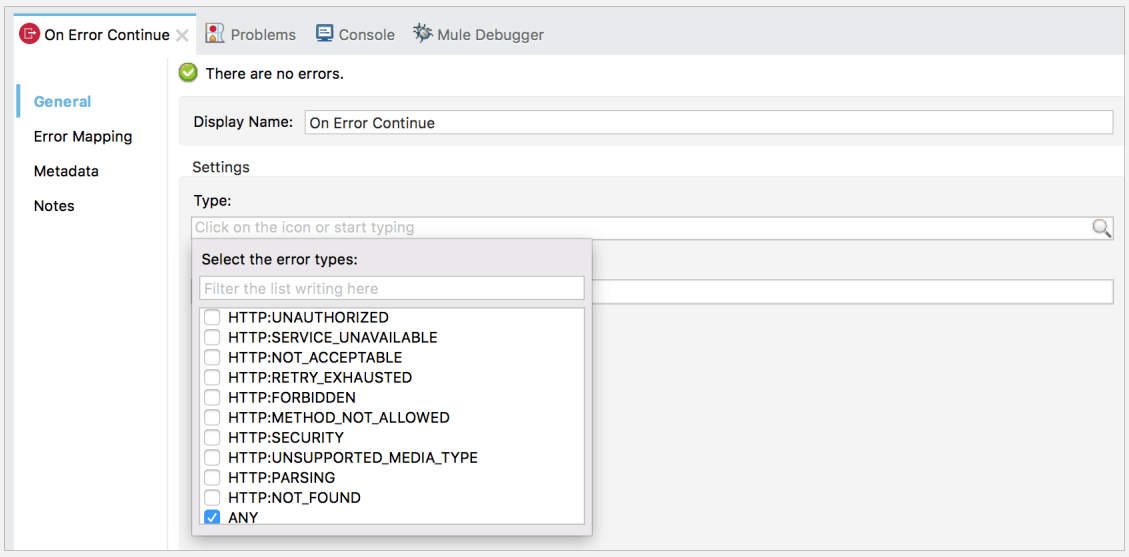
Should the request fail with an unauthorized error instead (3), then the inner-flow will handle it with an on-error-continue by retrieving static content from a file (4). Then, the flow reference will be successful as well (2) and a successful response will be returned (5).



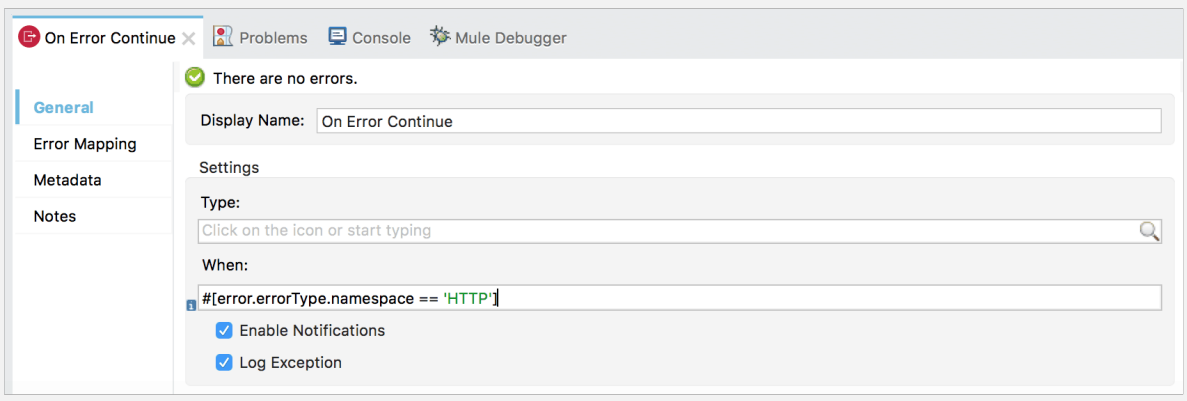
But what if another error occurred in the HTTP request? Although there are only handlers for “not found” and “unauthorized” errors in the flow, errors are still propagated by default. This means that if no handler matches the error that is raised, then it will be rethrown. For example, if the request fails with a “method not allowed” error (3), then it will be propagated causing the flow reference to fail (2), and that propagation will result in a “failure” response (4).



The scenario above could be avoided by making the last handler match `ANY`, instead of just `HTTP:UNAUTHORIZED`. Notice how, below, all the possible errors of an HTTP request are suggested:

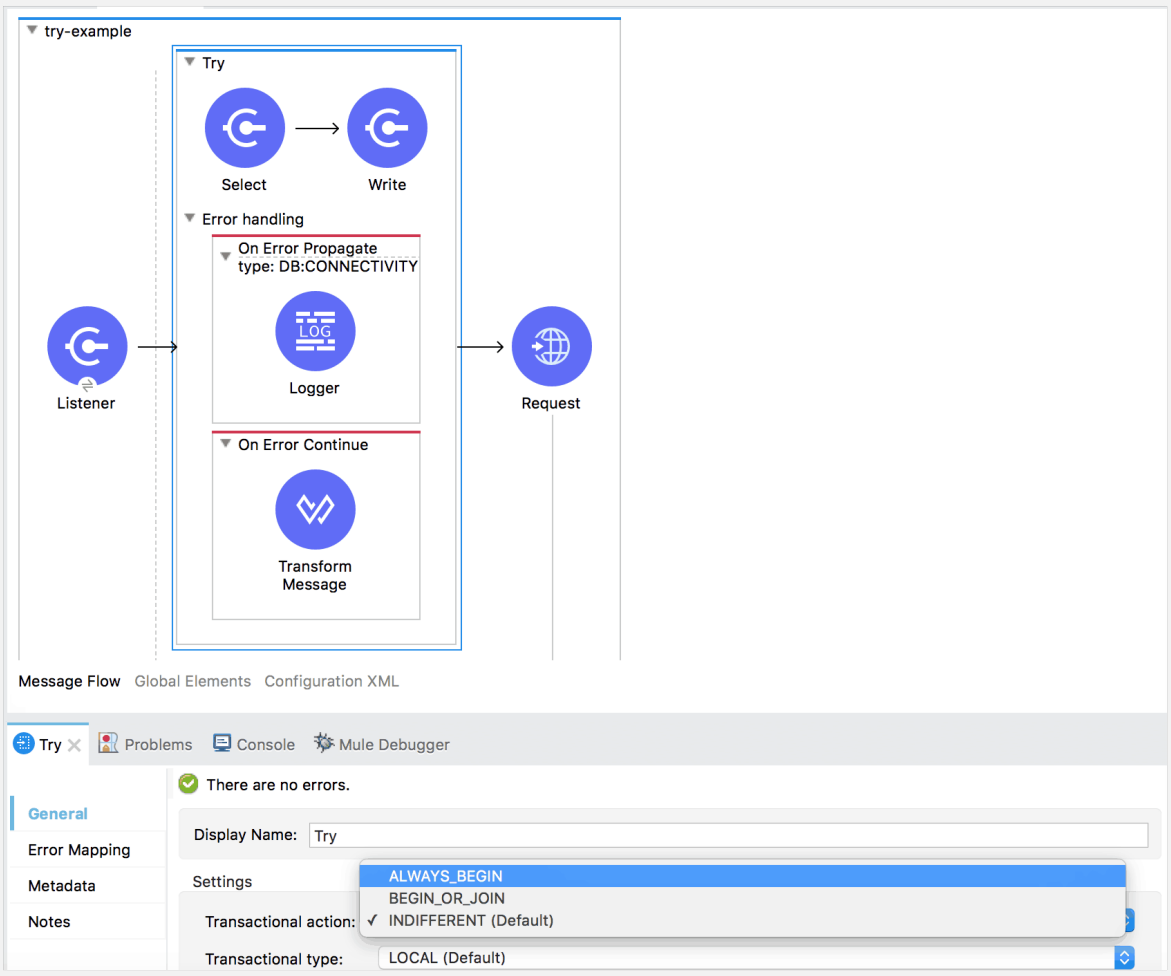


Errors can also be matched through expressions. For example, since the Mule error is available during error handling, it can be used to match all errors whose namespace is HTTP:



Try scope

Mule 4 provides more fine-grained error handling with the introduction of the “try” scope. Try scopes can be used within a flow to handle errors of inner components. The scope also supports transactions—replacing the old transactional scope.



The error handler behaves as explained earlier. In the example above, any database connection errors will be propagated, causing the try to fail and the flow’s error handler to execute. In this case, any other errors will be handled and the try scope will be considered successful which, in turn, means that the following processor, an HTTP request, will continue executing.

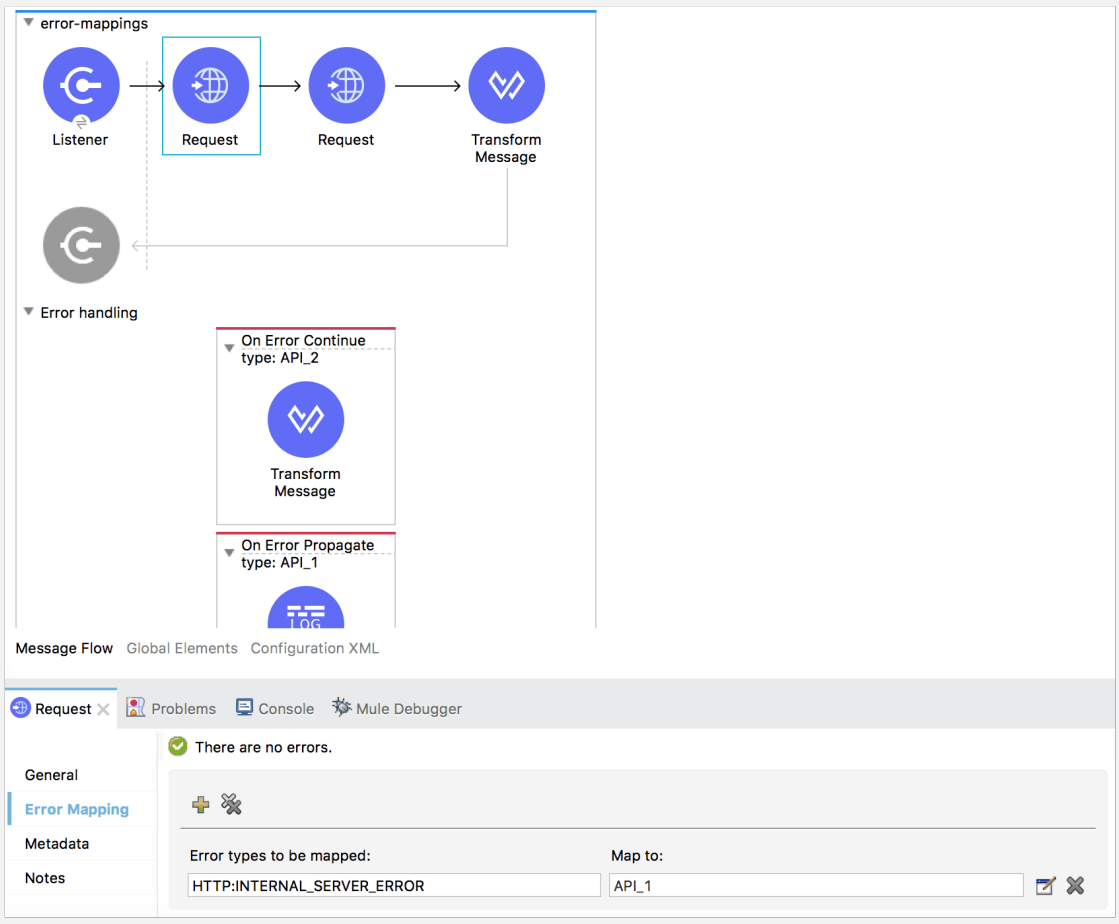
Error mappings

Error handling can also be mapped to custom errors. Error mappings are most useful when an app contains several equal components and there is a need to distinguish between each specific error.

By adding error mappings to each components, all specified types of errors will be mapped to another of the user's choosing. In doing so, the error handler can then digest accordingly. Consider if a flow is aggregating results from 2 APIs using an HTTP request component for each. The user may want to distinguish between the errors of API 1 and API 2, since by default their errors will be the same.

By mapping errors from the first request to a custom API_1 error and errors in the second request to API_2, users can route those errors to different handlers.

The example below shows the mapping of the `HTTP:INTERNAL_SERVER_ERROR`, where different handling policies can be applied in case an API goes down. It propagates the error in the first API and handles it in the second API.



Triggers and connectors

Anypoint Connectors now include new “triggers” for Mule flows. Rather than configuring logic to perform scheduled queries against target systems, the connector contains built-in logic to listen for data changes, such as when a new database row is created, a Salesforce lead is changed, or a file is created. Also, the File and FTP connectors have also been enhanced—providing users with the ability to append files, create directories, and perform other operations.

All connectors now follow the same model of operations and connector configurations because of the new Mule SDK. The File, FTP, JMS, and Email connectors have also been enhanced.

The features behind the File connector include:

- The ability to read files or fully list directories’ contents on demand; unlike the old transport, which only provided a polling inbound endpoint.
- Top level support for common File System operations such as copying, moving, renaming, deleting, creating directories, and more.
- Support for locking files on the file system level.
- Advanced file matching functionality.

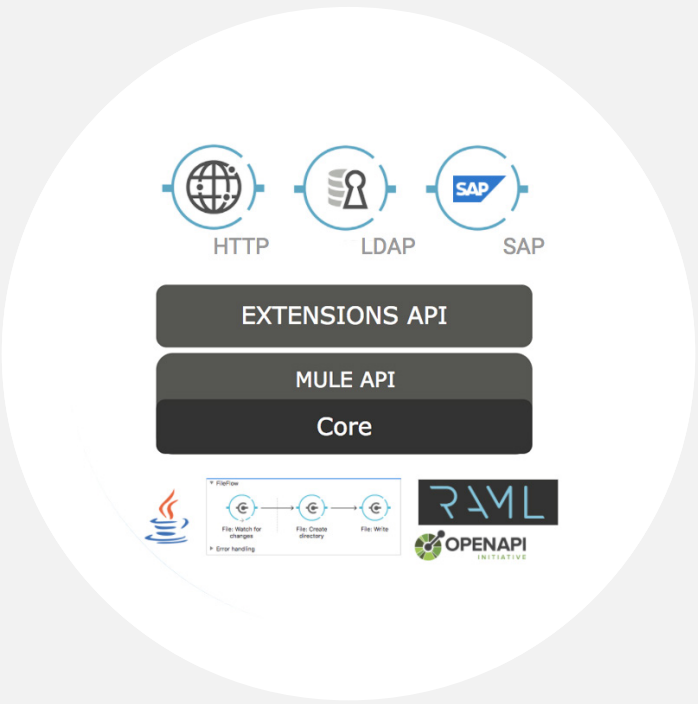
This connector was designed to be fully consistent with the FTP Connector. The same set of operations are available on both connectors in Anypoint Exchange and they behave *almost* the same.

Both the File and FTP connectors look exactly the same. Some of the main features of the FTP connector are:

- Read files or fully list directories contents on demand.
- Copy, move, rename, delete, and create directories quickly and consistently.
- Support for locking files.
- Advanced file matching functionality.

Mule SDK

Mule SDK is the successor to the Anypoint Connector DevKit that enables developers to easily extend Mule and create new Anypoint Connectors which can be shared in Anypoint Exchange. Mule SDK provides a simple annotation-based programming model for extending Mule and also offers enhanced forward compatibility with Mule versions, transactions support, and the ability to create routers.



Before, Anypoint Connector DevKit would take a Connector project and generate code around it, creating a wrapper that would then bridge the code to the Mule runtime internals. Now, the SDK relies on Mule APIs (*more precisely, the Extensions API*), and does not need to generate code between the connector and the runtime internals. This makes it easier to build connectors that are both backwards and forward compatible across version of the runtime. This also allows the runtime to inject cross-cutting functionality (such as streaming, media-type handling, reconnection, enrichment, etc.), across connector operations

Any artifact built with Mule SDK will be completely isolated from the runtime. This will allow users to use any dependency

or library without worrying about version conflicts with those shipped in the runtime. The SDK automatically enforces consistency and adds cross-cutting functionality – such as repeatable streaming, embeddable DataWeave scripts, automatic static DataSense, and mimeTypes support – across connector operations and into existing modules without the need to rebuild them. For a list of functionalities, read the SDK documentation.

Transactions, error handling, and more

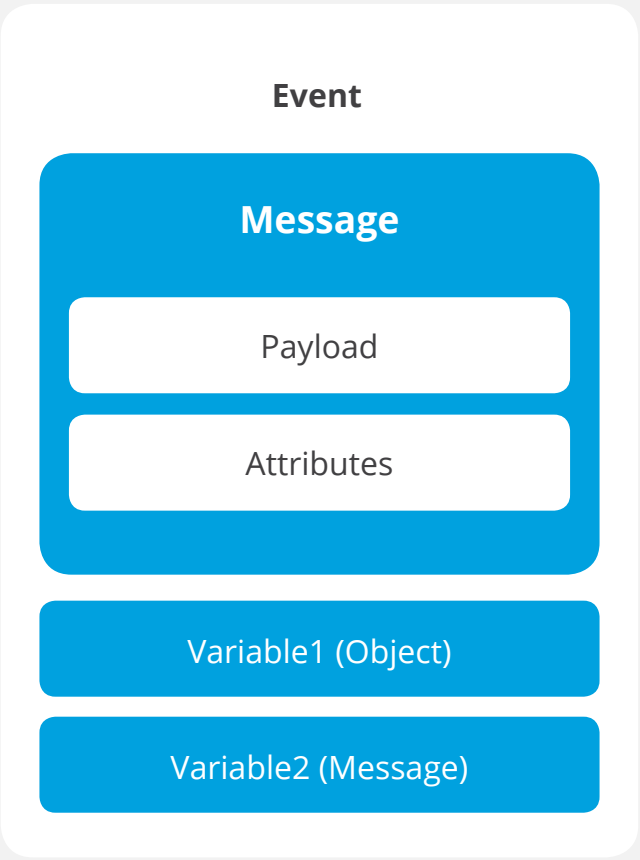
A transaction is a series of actions that work in an atomic way. For example, if a flow need to execute 5 steps, but the 4th fails, the first 3 steps are rollbacked. As a result, the system state does not change; instead, it prevents having an inconsistent state.

Transactions are not permanent actions until the entire sequence of events are complete. Users can now build both operations and message sources capable of participating in single resource or XA transactions. They can also develop non-blocking operations easily leveraging Mule 4's new reactive engine and use expressions on the module's config elements. This makes it super easy to develop modules which enable simultaneous connections to many different credentials.

Mule SDK also leverages the new error handling mechanisms, DataWeave 2.0, and DataSense. Each component in a module lists the full list of possible errors, making it easier for users to handle errors. Users can expose custom pieces of logic in their modules as DataWeave functions and by supporting dynamic DataSense, Mule SDK has a more powerful model to describe the dynamic types for every individual parameter and message sources.

Message model

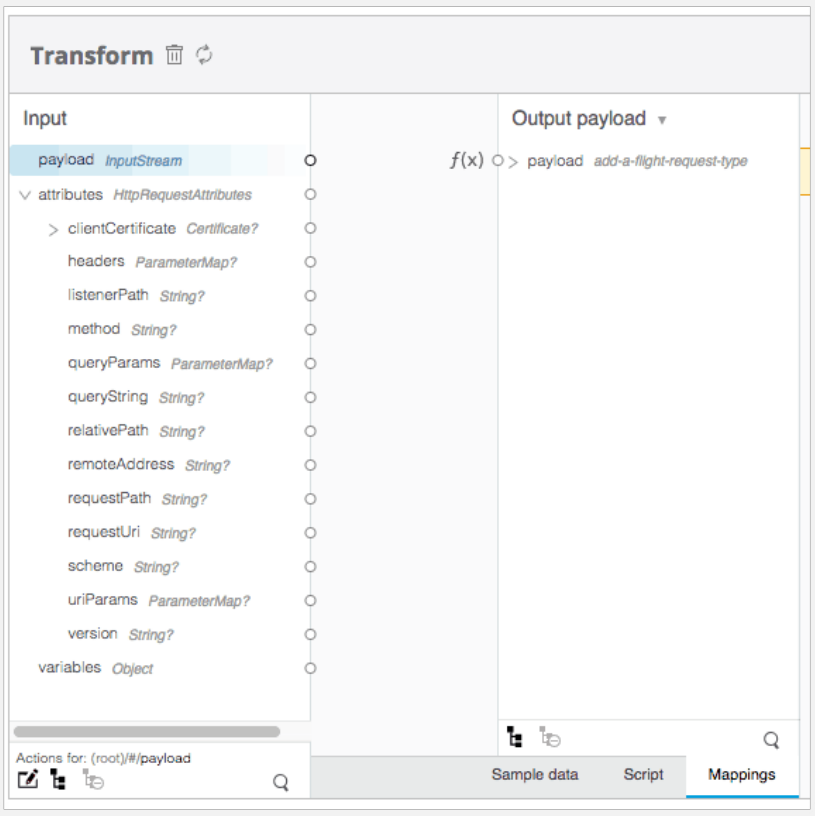
Mule 4 also introduces a simplified Message model which makes it easier to access content and metadata. Users no longer need to think about transport barriers because there is only one scope of properties. The Message model in Mule 4 and beyond looks like:



A Message in Mule 4 is made up of a Payload and Attributes. The new Attributes object replaces the inbound property scope in the Mule 3 Message model. This allows for easy differentiation of incoming HTTP headers and other metadata about the request provided by the HTTP listener. Also, as it is strongly typed, it is easy to discover and use the connector-populated metadata associated with each request, especially combined with DataSense support in Anypoint Studio.

When a Flow is triggered (e.g. a new HTTP request is received) the connector, in this case, the HTTP Listener, will create a Message that composes both the HTTP body and an instance of HttpAttributes. The HttpAttributes instance will contain both

the incoming HTTP headers and all other metadata. Because it is typed, it is easier to use and self-discover.



In Anypoint Studio and Design Center—flow designer, and via DataSense, the attributes populated by the previous connector source or operation can be seen and used. The screenshot above depicts this in the context of the DataWeave Transform component. While this example is HTTP-centric, when a different source connector (e.g. JMS) is used, the attributes object will be a specific instance to the connector and will, in turn, provide access to connector-specific metadata, (e.g JMS correlation ID)

Outbound properties

In Mule 4, outbound properties no longer exist. Instead, the headers or properties (e.g. HTTP headers or JMS properties) are now configured explicitly as part of the connector operation configuration. Users can user variables to store the value of headers and use it later in flows as well.

In Mule 3, it was possible to add outbound message properties anywhere in a flow. The next outbound endpoint or connector operation (e.g. an HTTP Requester) would then use the set of

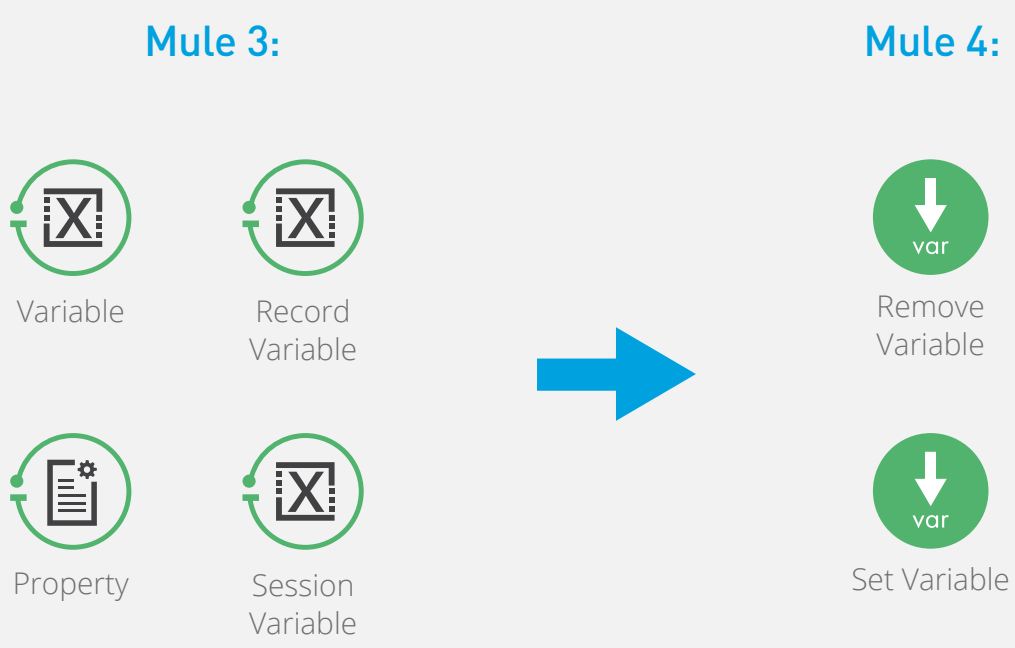
outbound properties present at that point in the flow and send them, along with the current payload, via the outbound end-point or connector (e.g. HTTP). While this approach seems very convenient, it resulted in a lot of issues where the behavior of flows and the outbound properties were not deterministic. Subsequently, the refactoring of a flow or the introduction of a new connector operation in an existing flow would significantly change the behavior of a flow. The improvements in the Mule 4 Message model avoid these issues with outbound properties in Mule 3.

To Mule 3 users: Impact on application design

The changes to Message properties will result in more deterministic flows. For Mule 3 users, the changes to Message properties will require a slight recalibration as to how users think about defining their integrations. When migrating existing applications to Mule 4 while using Mule 3 transports via the compatibility module, the Message properties will continue to work as they do in Mule 3. Only when using the new connectors in Mule 4 will you need to work with the new Attributes objects.

Variables

Flow variables remain largely untouched, with the major change being that they have been renamed to “variables” and that session variables have been removed.



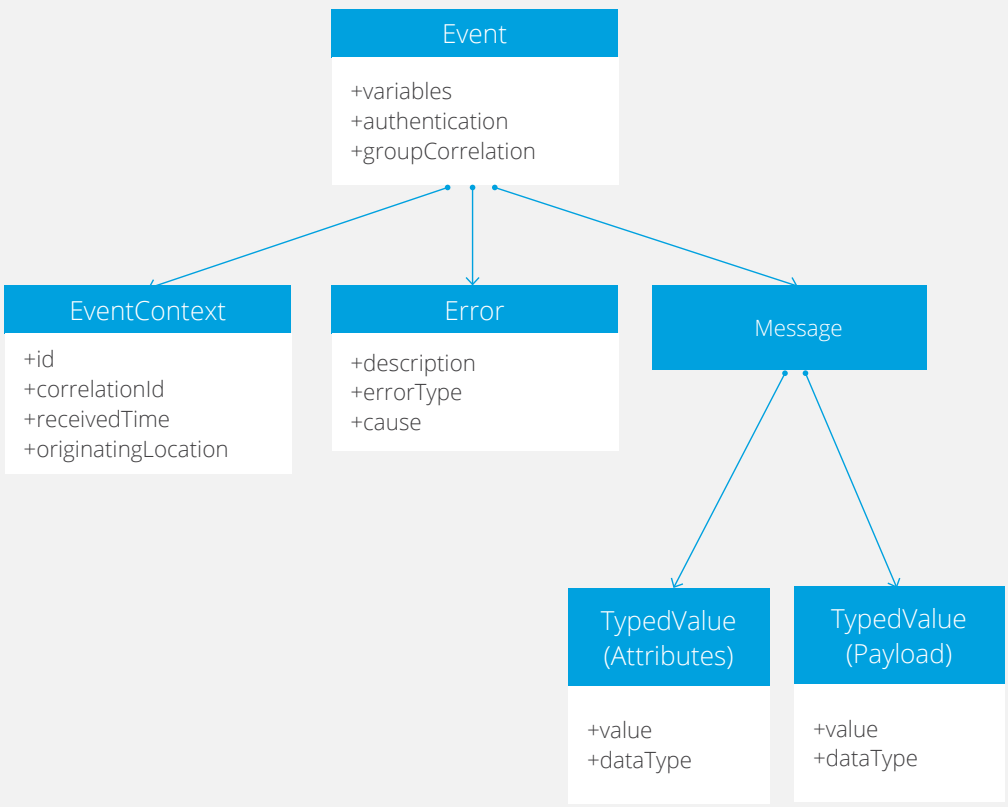
Mule 4 removes session variables for a couple of reasons. Firstly, session variables that were propagated between different flows didn't have anything to do with a stateful client session. Secondly, session variables were often seen as security concern because variables defined in a Flow were automatically being propagated via HTTP headers or JMS properties. Consequently, users were frequently disabling this functionality.

The more security conscious, and explicit, way of propagating values between different flows or systems is via the explicit use of HTTP headers or JMS properties.

Impact on application design

Other than the changes required in migrating to DataWeave expressions, there is nothing specific to take into account regarding variables. If you are migrating an application from Mule 3 that uses session variables, it is quite likely that you didn't ever need session variables, and flow variables will be sufficient. But, if you do need to propagate values between flows, then you should use set outbound and header/property as required in the source flow and then read this value from the attributes object in the destination flow.

The simplified updated model can be seen in the class diagram below:



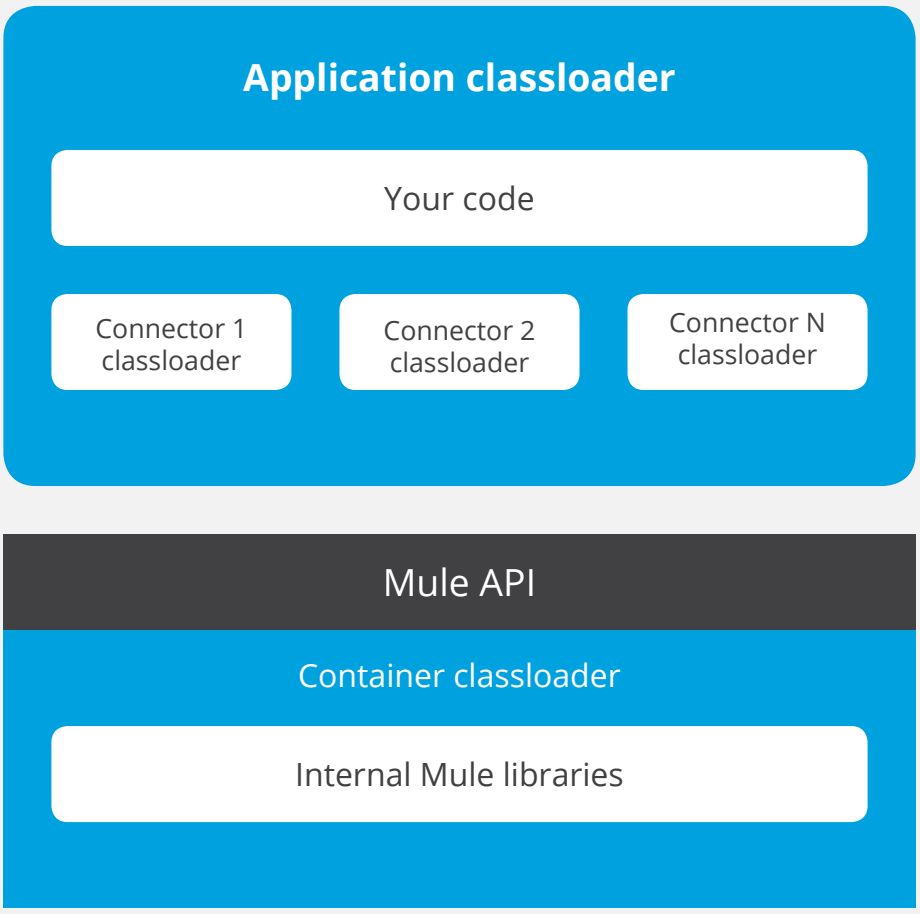
In summary:

- **Message:** The user message includes the payload, and an optional attributes instance. This is what users primarily work with and is what connector operations receive and return.
- **Event (*internal*):** The message, variables, authentication, and group correlation (which contains the sequence number and group size). This is internal to the runtime and updated during flow execution as the message and variables are mutated. The Event also has a reference to the current *EventContext*.
- **EventContext (*internal*):** Represents a single request or transaction with its correlationID and other details. The same *EventContext* instance is used throughout Flow execution.

Upgrade

The primary of objective when release Mule 4 was to make it easier for users to get new enhancements with little to no work on the user. This was difficult in prior versions of Mule because if users were dependent on Mule's internal Java libraries, then they were forced to upgrade as libraries often changed from release to release.

Classloader isolation



With new classloader isolation between Mule applications, the runtime, and Anypoint Connectors, any library changes that occur internally do not affect the application itself. There is now a well-defined Mule API, so users can be sure that they are using supported APIs. Connectors are distributed outside the runtime as well, making it possible to get connector enhancements and fixes without having to upgrade the runtime or vice versa.

Conclusion

Mule 4 is an incredible iteration of the Mule runtime. With all the improvements like the simplified development language of DataWeave and the ability to upgrade connectors and applications make it easy to create, collaborate, and connect.

Learn more

Blog: Mule 4

Read the Mule Runtime 4.1.1 Release Notes

Webinar: Mule 4 GA - Accelerate the speed of development

About MuleSoft

MuleSoft's mission is to help organizations change and innovate faster by making it easy to connect the world's applications, data and devices. With its API-led approach to connectivity, MuleSoft's market-leading Anypoint Platform™ is enabling over 1,000 organizations in more than 60 countries to build application networks. For more information, visit mulesoft.com.

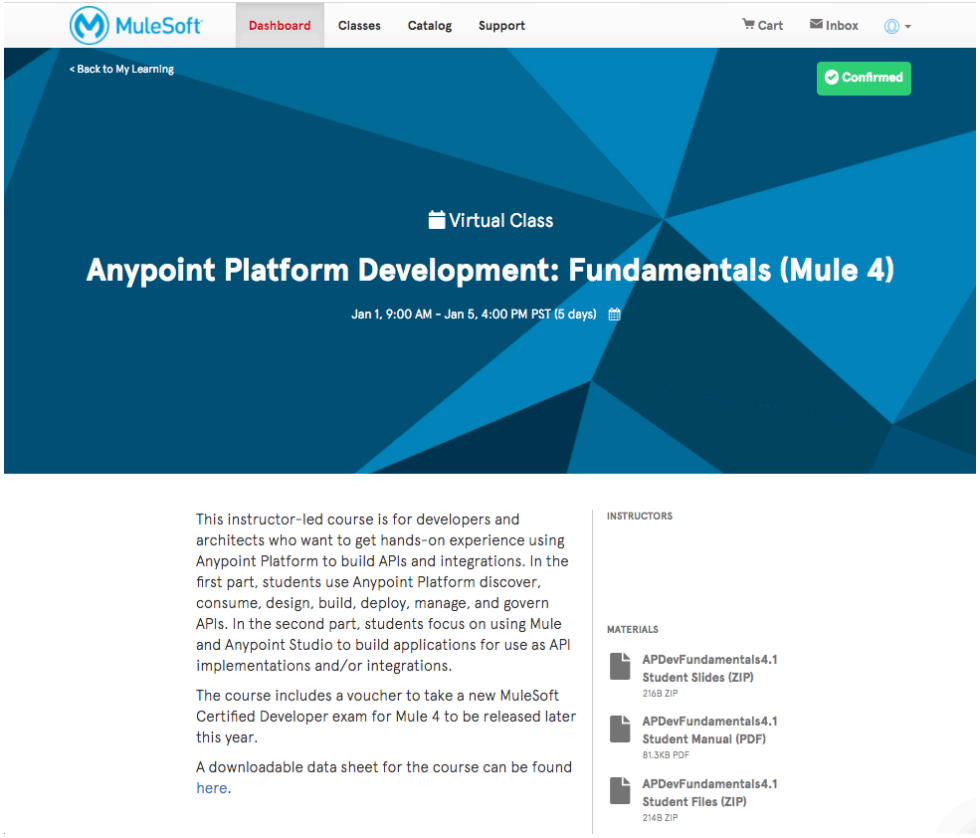
Anypoint Platform Development: Fundamentals

Student Manual

Mule runtime 4.1
April 29, 2018

Table of Contents

Introducing the Course



Objectives:

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

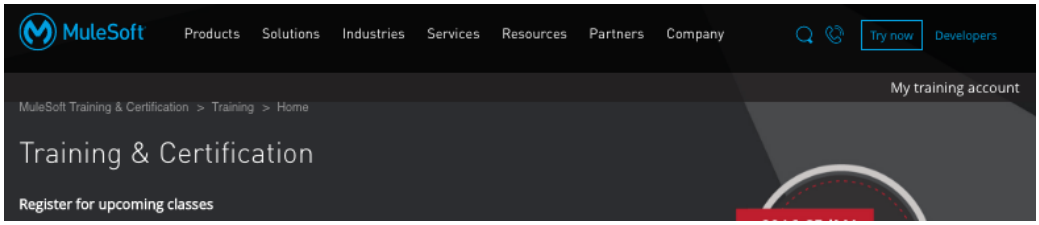
Walkthrough: Set up your computer for class

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

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

Download student files

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



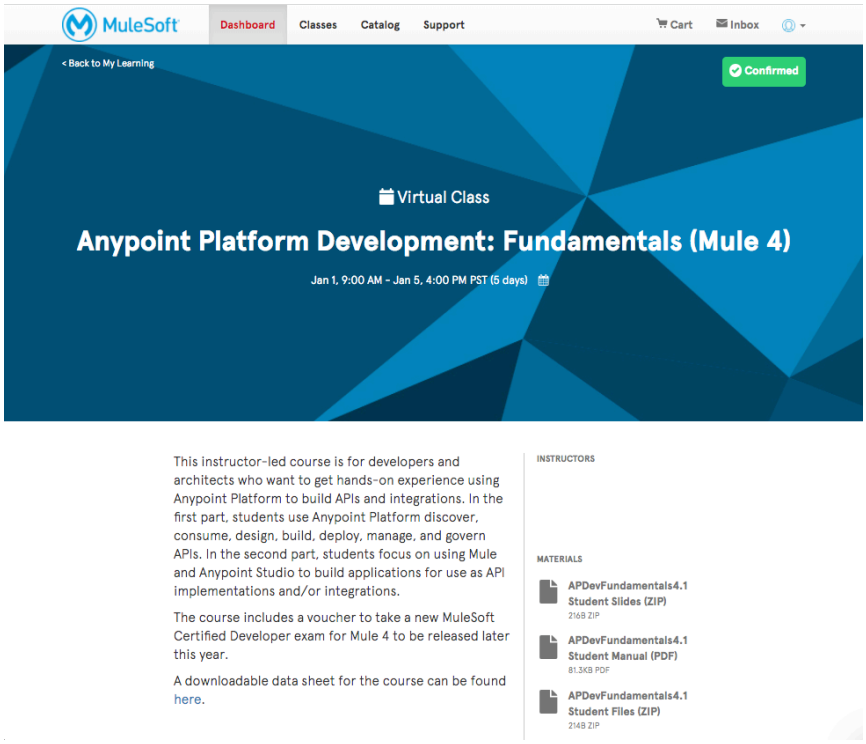
3. Log in to your MuleSoft training account using the email that was used to register you for class.

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

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

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

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



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

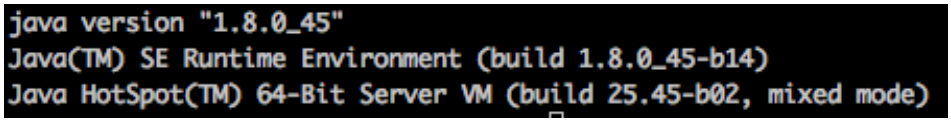
    Note: Keep this file open. You will copy and paste text from it during class.

Make sure you have JDK 1.8

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

    ```
    java -version
    ```

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



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

Note: JDK 1.9 is NOT supported.

Make sure you have Java in your PATH environment variable

15. In a command-line interface, type $PATH (Mac) or %PATH% (Windows) and press enter.
    - Mac: $PATH
    - Windows: %PATH%
16. After installing the correct JDK version, add or update an environment variable named JAVA_HOME that points to the installation location and then add JAVA_HOME/bin to your PATH environment variable.

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

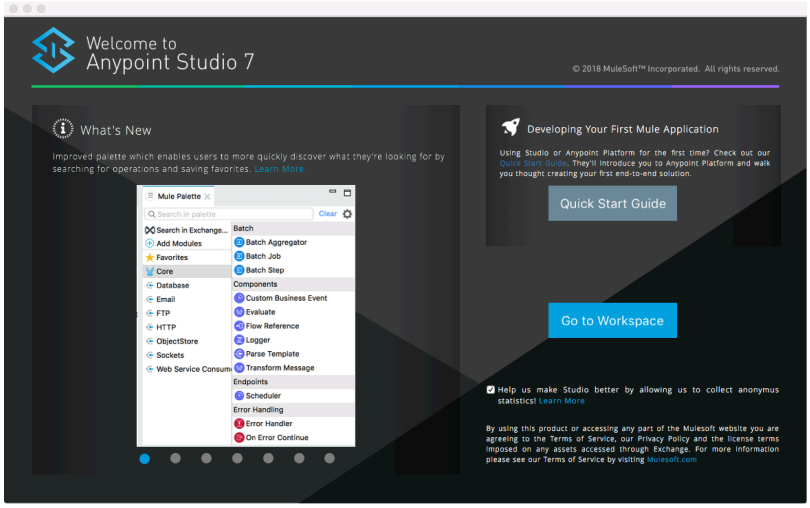
Start Anypoint Studio

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

    Note: If you do not have Anypoint Studio, you can download it from https://www.mulesoft.com/lp/dl/studio.
18. In the Workspace Launcher dialog box, look at the location of the default workspace; change the workspace location if you want.
19. Click OK to select the workspace; Anypoint Studio should open.

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

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



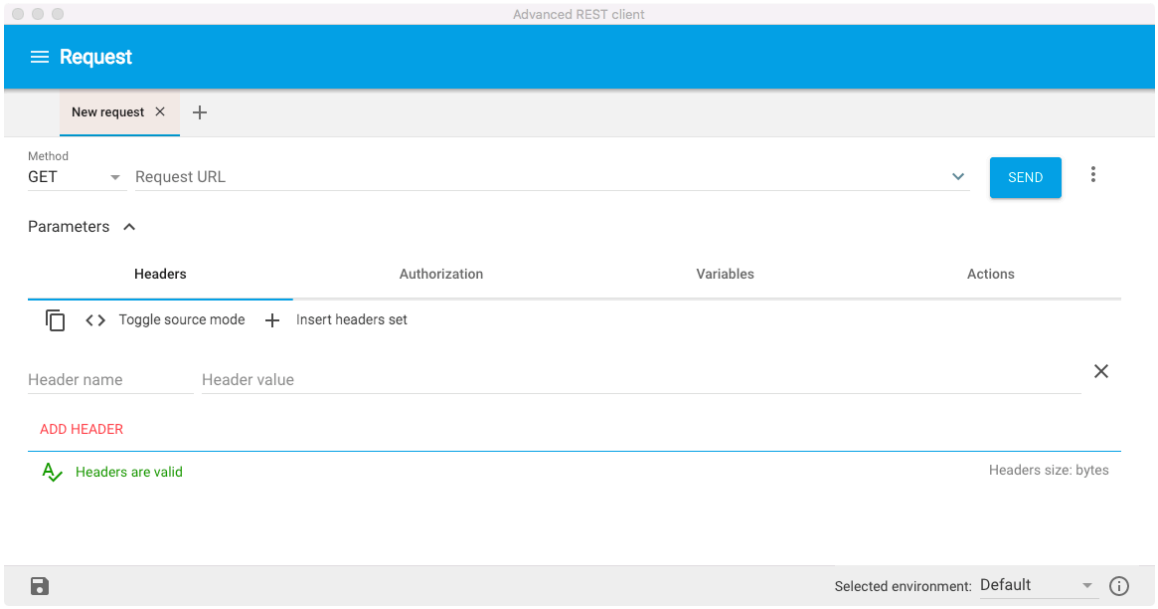
21. If you get an Updates Available pop-up in the lower-right corner of the application, click it and install the available updates.

Open Advanced REST Client

22. Open Advanced REST Client.

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

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



Make sure you have an active Anypoint Platform account

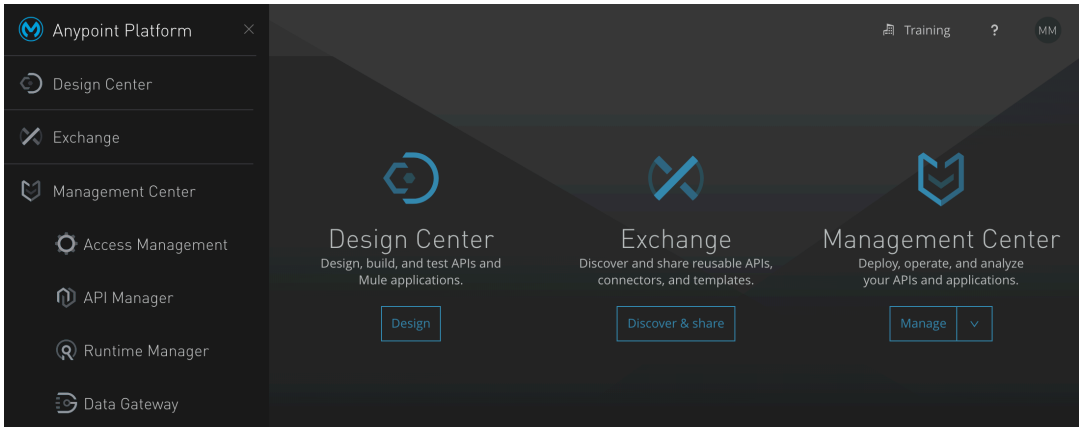
24. In a web browser, navigate to http://anypoint.mulesoft.com/ and log in.

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

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

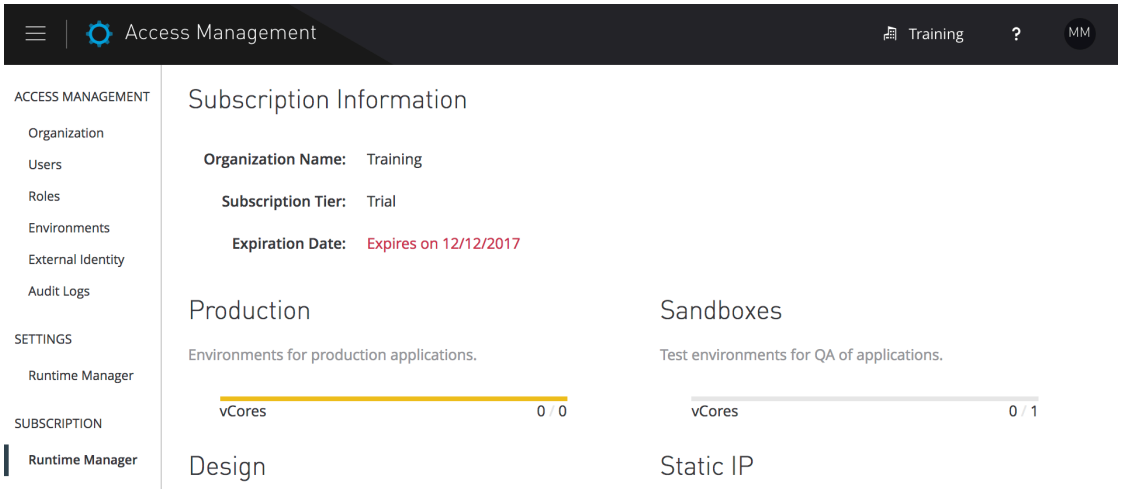
26. In the menu that appears, select Access Management.

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



27. In the left-side navigation, click the Runtime Manager link under Subscription.
28. Check your subscription level and if it is a trial account, make sure it is not expired.

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

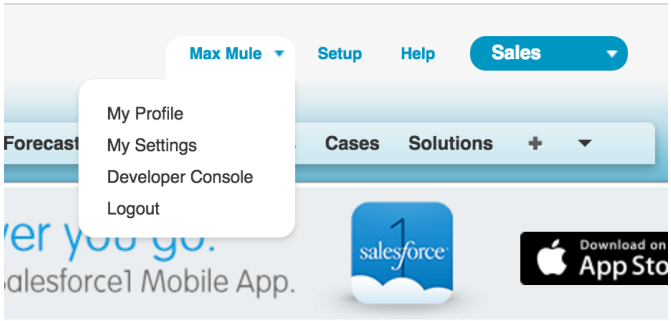


Make sure you have a Salesforce developer account and an API security token

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

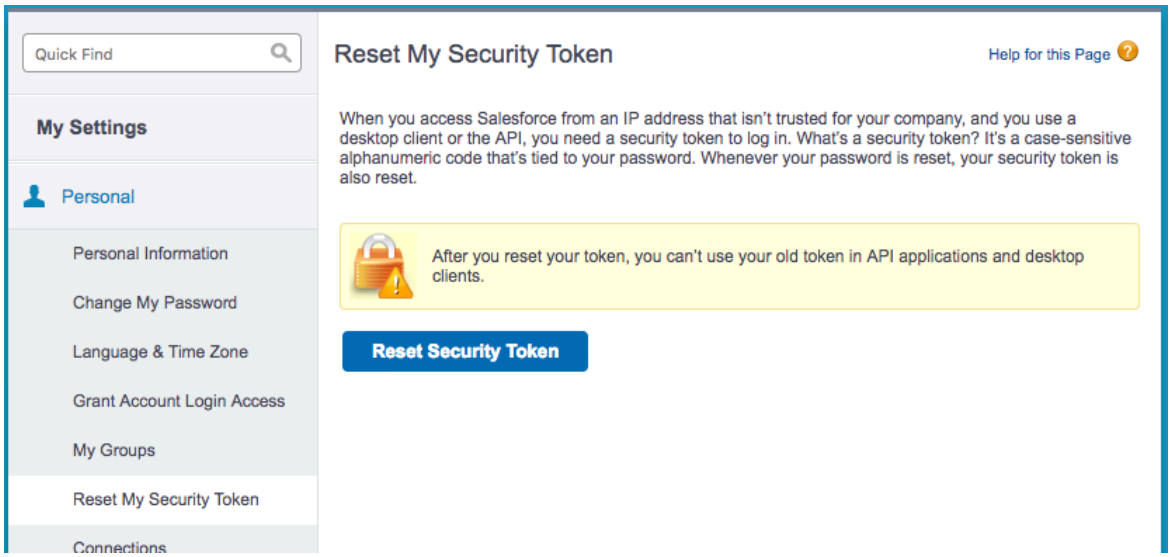
    Note: If you did not sign up for a free developer account yet, go to http://developer.salesforce.com/ and sign up for one now. You will want to use a free developer account and not your company account (if you already have one) for class. You will use the API to add new fictitious accounts to it and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

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

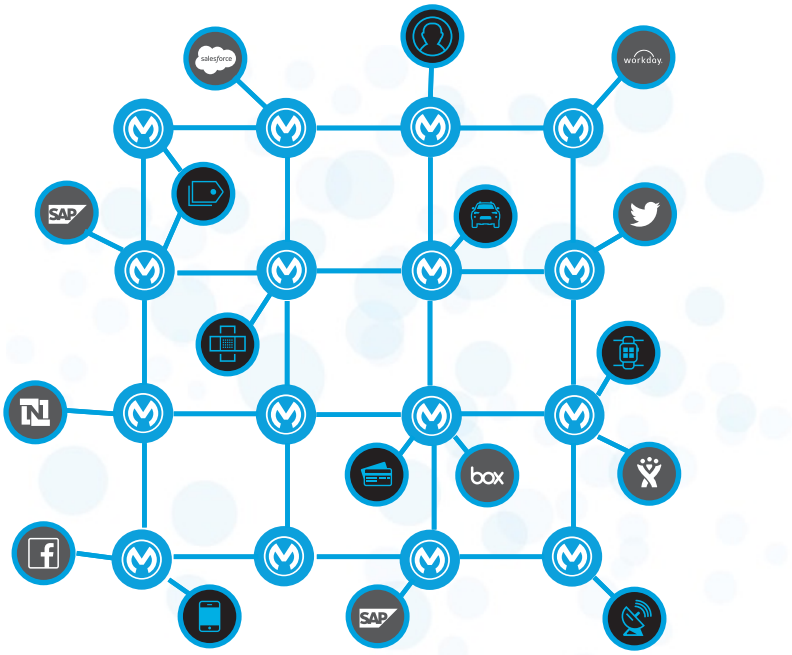


31. In the left-side navigation, select Personal > Reset My Security Token.
32. If you did not already request a security token, click the Reset Security Token button.

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



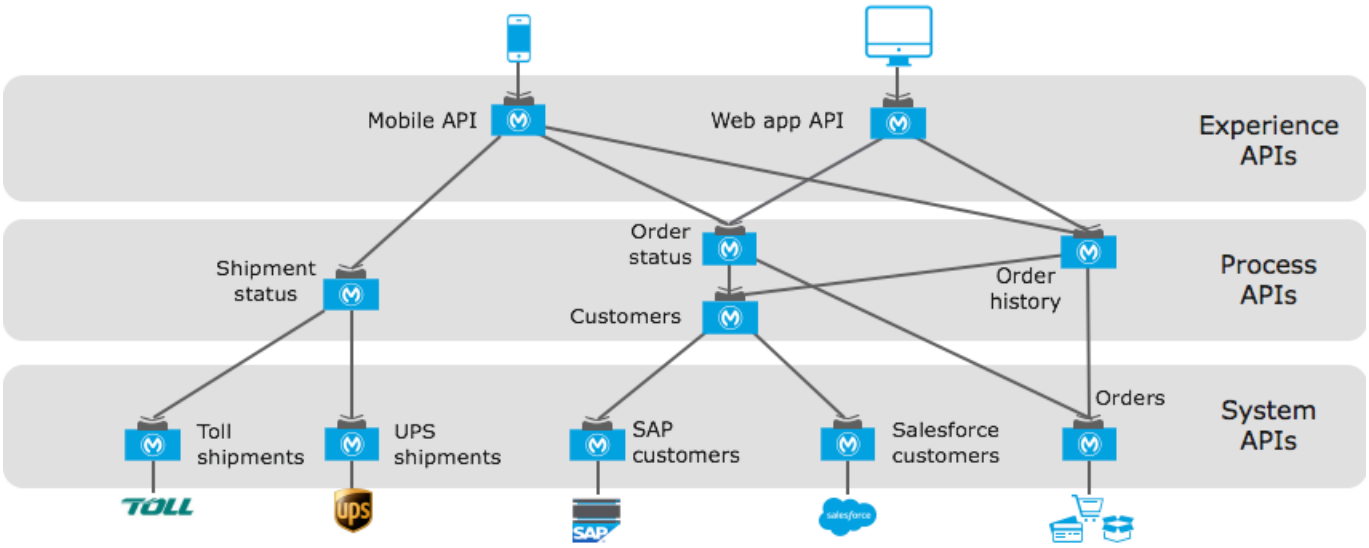
PART 1: Building Application Networks with Anypoint Platform



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

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

Module 1: Introducing Application Networks and API-Led Connectivity



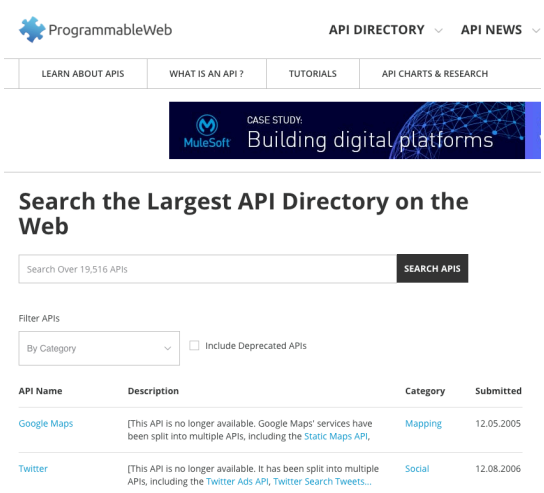
At the end of this module, you should be able to:

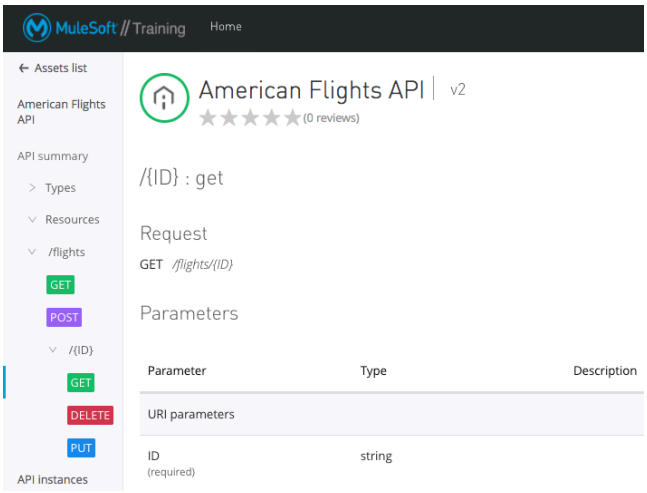
- Explain what an application network is and its benefits.
- Describe how to build an application network using API-led connectivity.
- Explain what web services and APIs are.
- Make calls to secure and unsecured APIs.

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

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

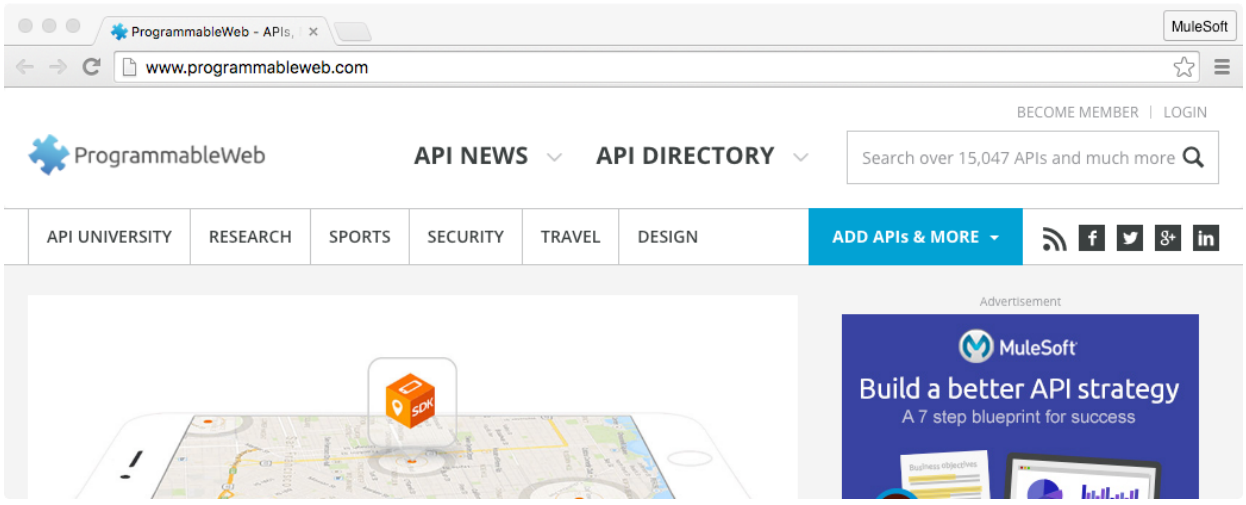
- Browse the ProgrammableWeb API directory.
- Explore the API reference for an API (like Twitter).
- Explore the API portal for an API to be used in the course.



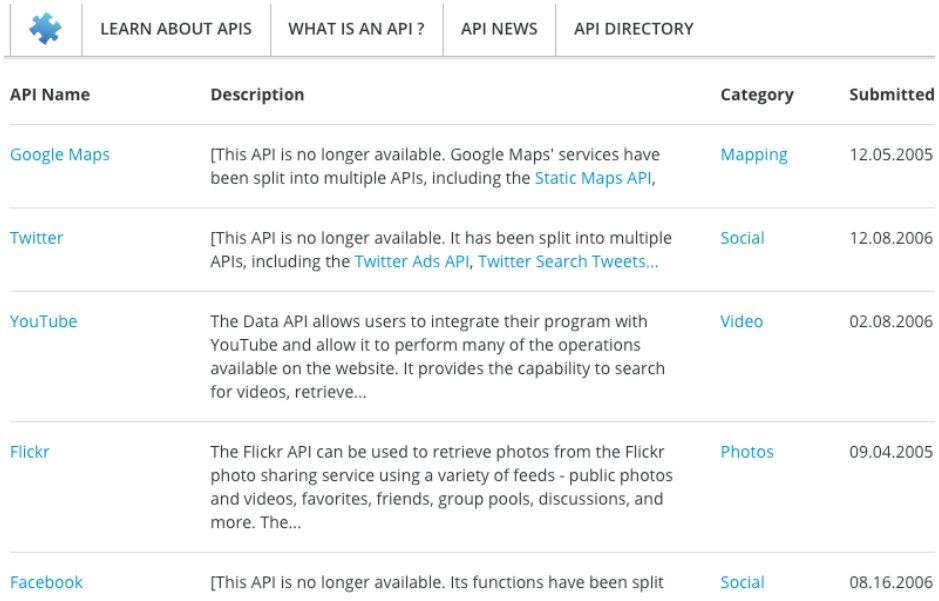


Browse the ProgrammableWeb API directory

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



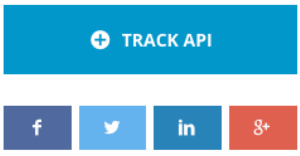
3. Browse the list of popular APIs.



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

Explore the API reference for the Twitter API

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



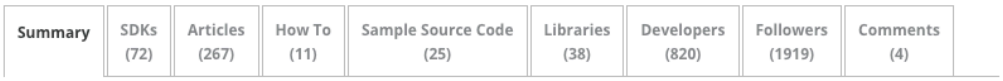
Twitter API

Social Blogging

[This API is no longer available. It has been split into multiple APIs, including the Twitter Ads API, Twitter Search Tweets API, and Twitter Direct Message API.

This profile is maintained for historical, research, and reference purposes only.]

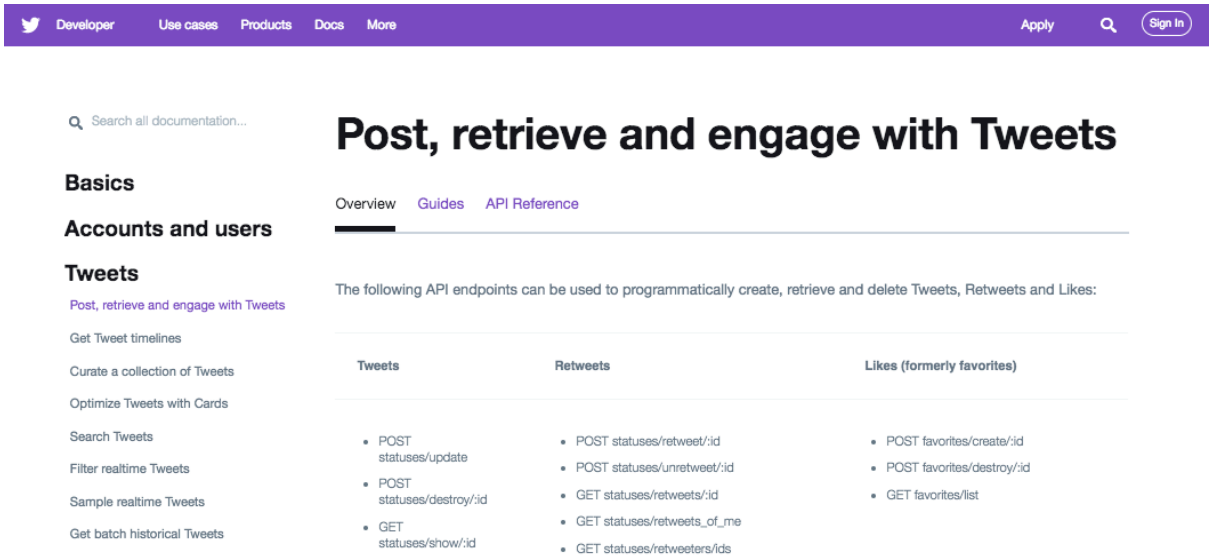
The Twitter micro-blogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with Twitter Search and trends data. The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



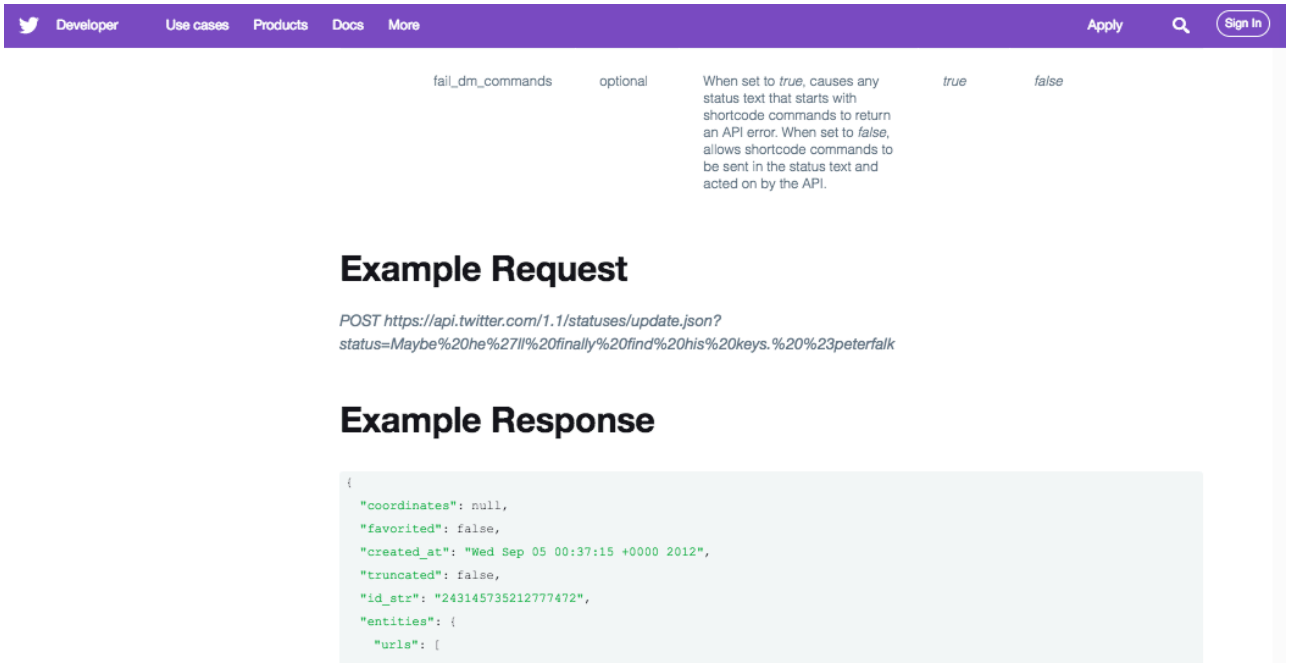
SPECS

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

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



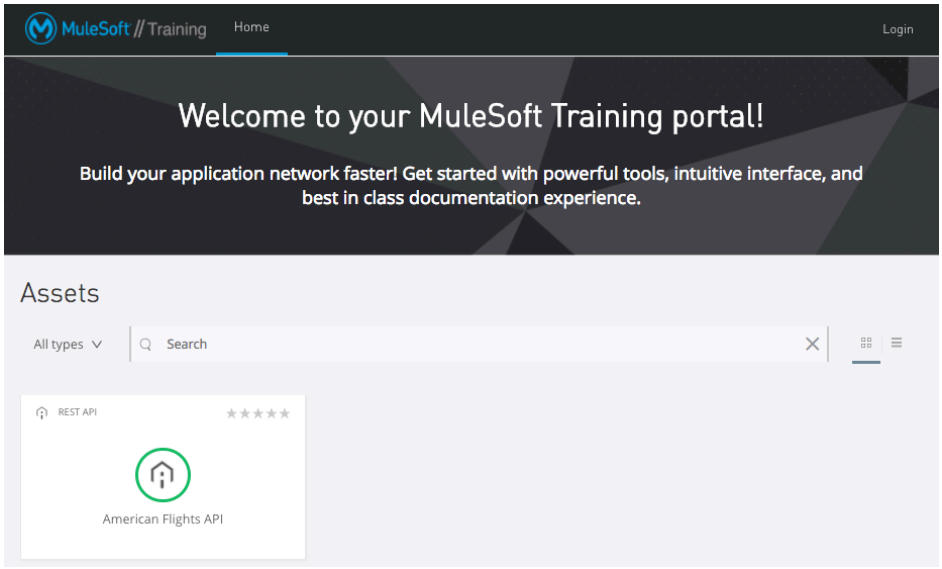
8. Browse the list of requests you can make to the API.
9. Select the API Reference tab beneath the title of the page.
10. Review the information for POST statuses/update including parameters, example request, and example response.



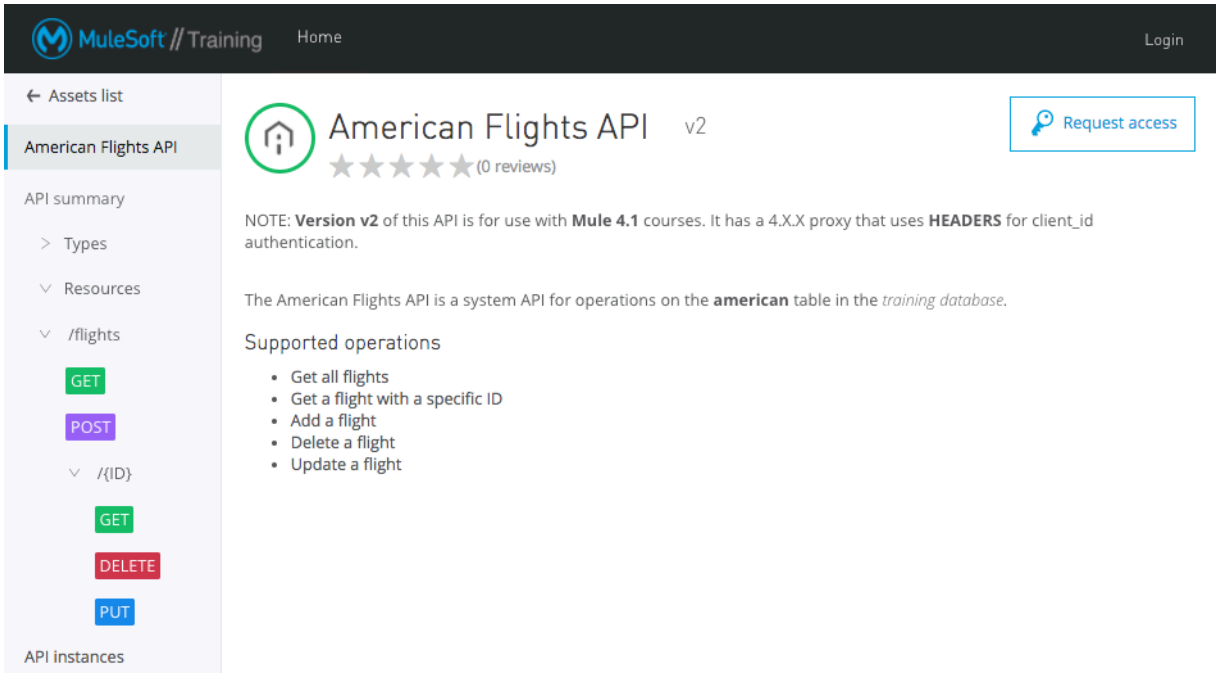
11. Close the browser tab.

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

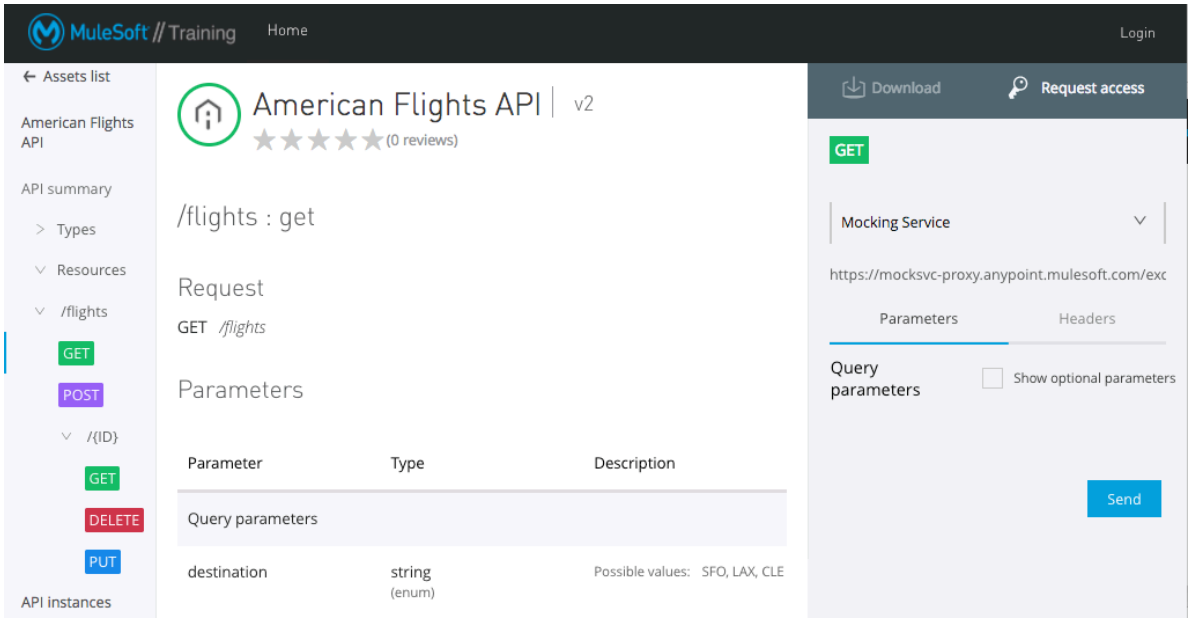
12. Return to the course snippets.txt file.
13. Copy the URL for the MuleSoft Training API portal.
14. Return to a browser window and navigate to that URL: https://anypoint.mulesoft.com/exchange/portals/muletraining/.



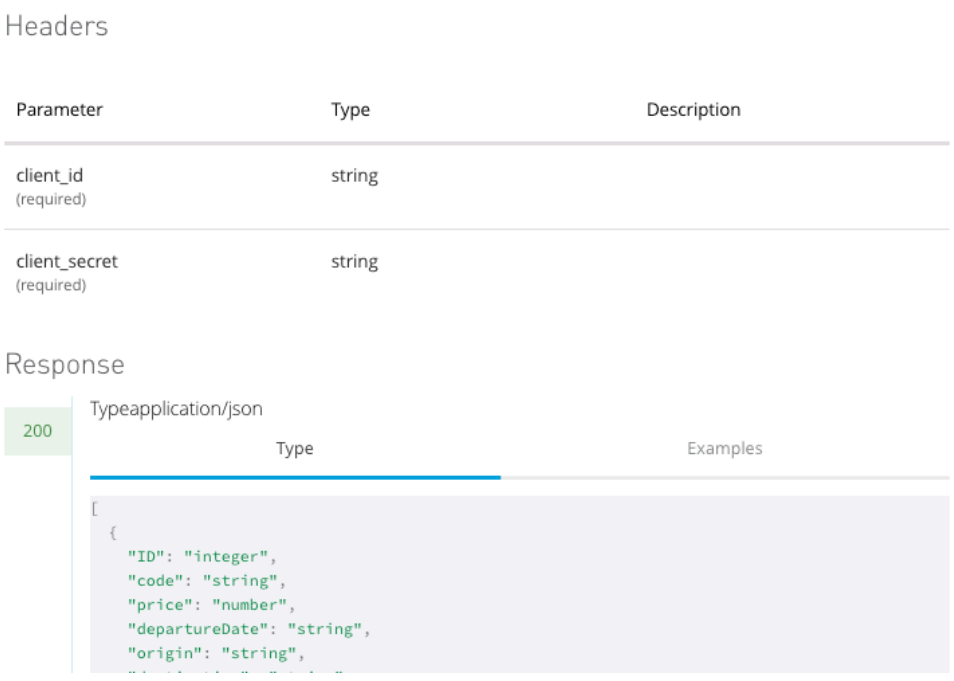
15. Click the American Flights API.
16. Browse the API Summary on the left-side and discover the resources that are available for the API.



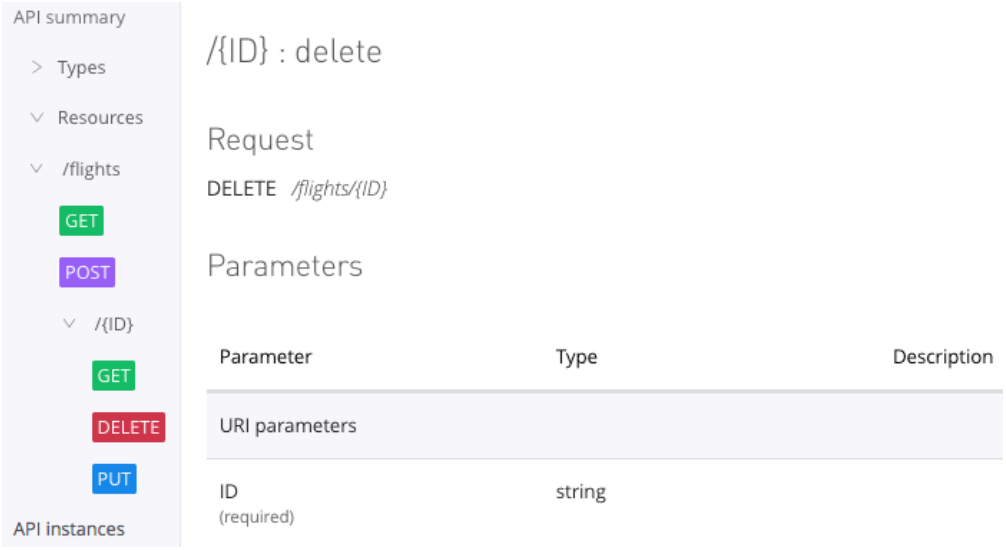
17. Select the GET method.
18. Review the information about the GET method; you should see there is an optional query parameter called destination.



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



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

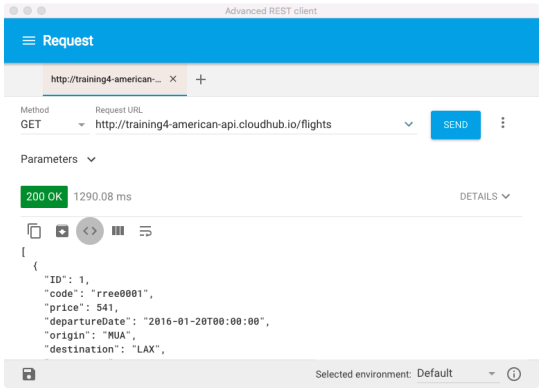


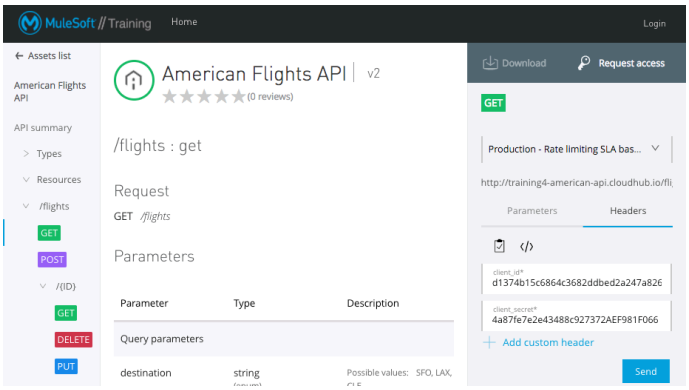
22. Review the information for the other resources.

Walkthrough 1-2: Make calls to an API

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

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



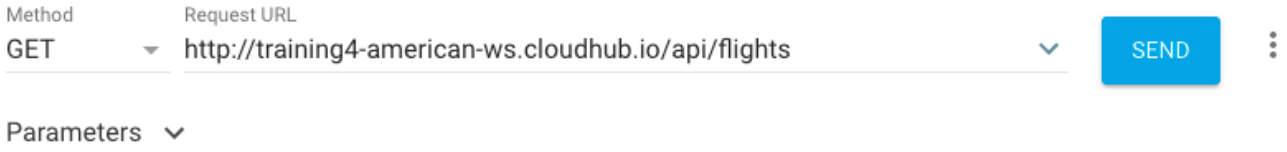


Use Advanced REST Client to make GET requests to retrieve data

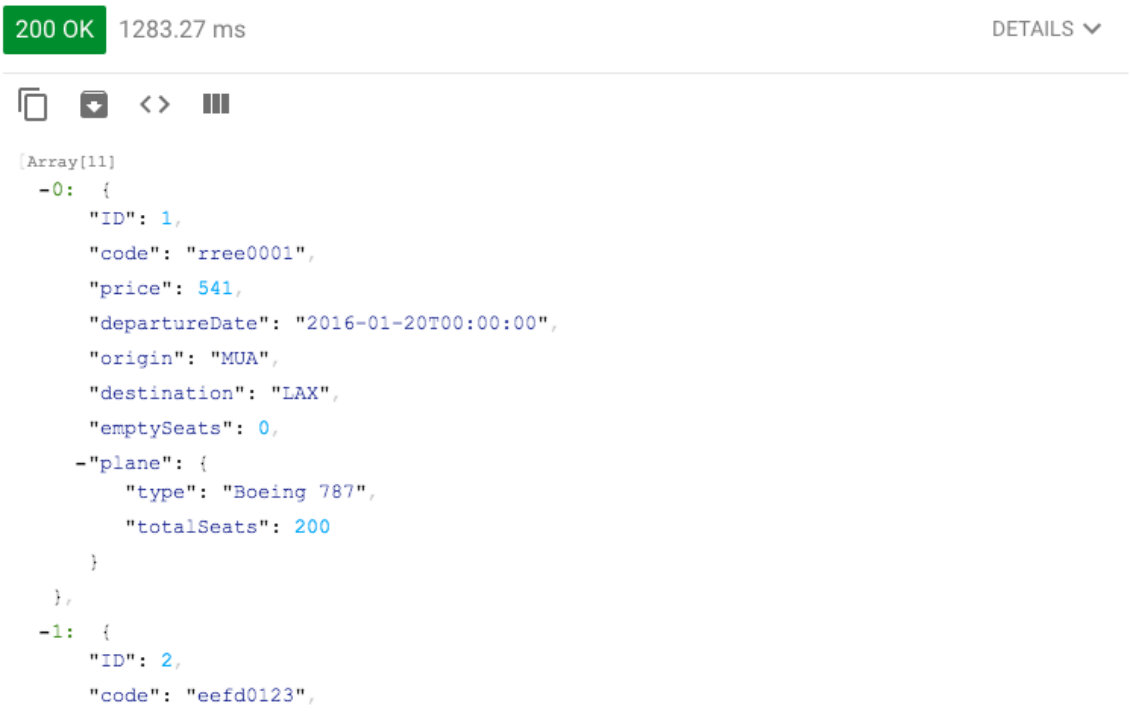
1. Return to or open Advanced REST Client.
2. Make sure the method is set to GET.
3. Return to the course snippets.txt file.
4. Copy the URL for the American Flights web service: http://training4-american-ws.cloudhub.io/api/flights.

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

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

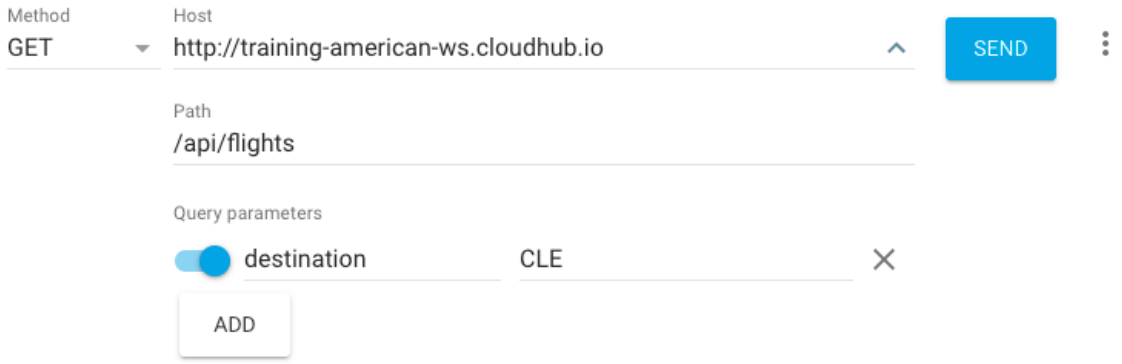


6. Click the Send button; you should get a response.
7. Locate the return HTTP status code of 200.
8. Review the response body containing flights to SFO, LAX, and CLE.

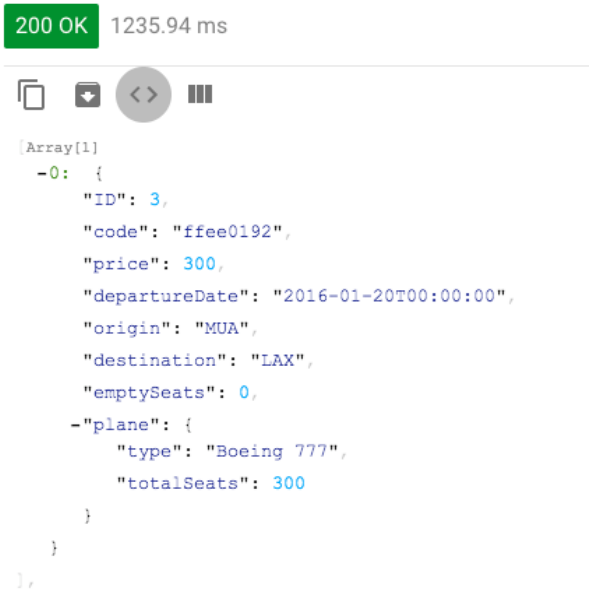


9. Click the Toggle raw response view button.

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



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

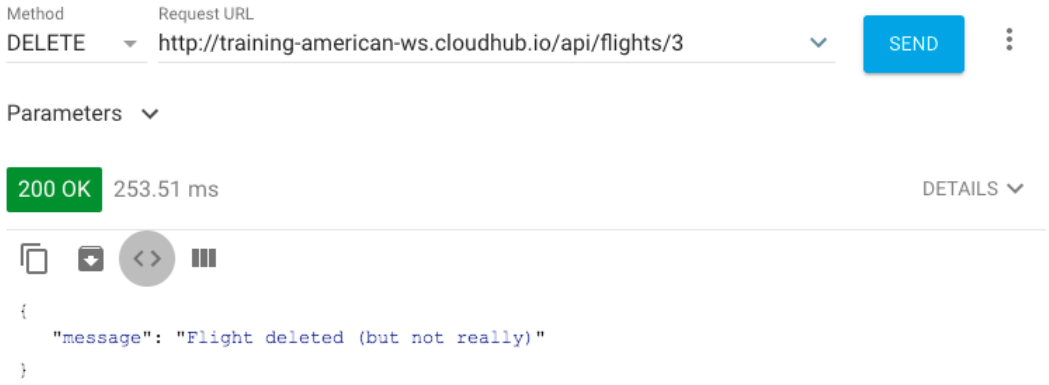


Make DELETE requests to delete data

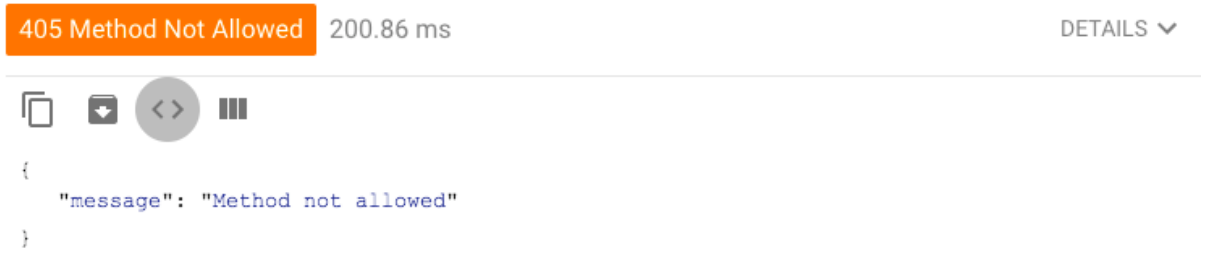
17. Change the method to DELETE.

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

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

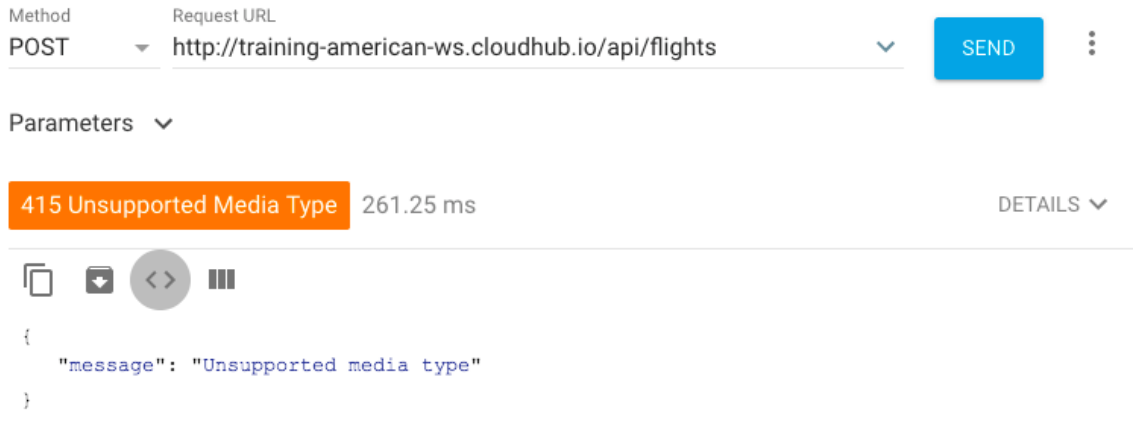


19. Remove the URI parameter from the request: http://training4-american-ws.cloudhub.io/api/flights.
20. Click the Send button; you should get a 405 response with a message of method not allowed.

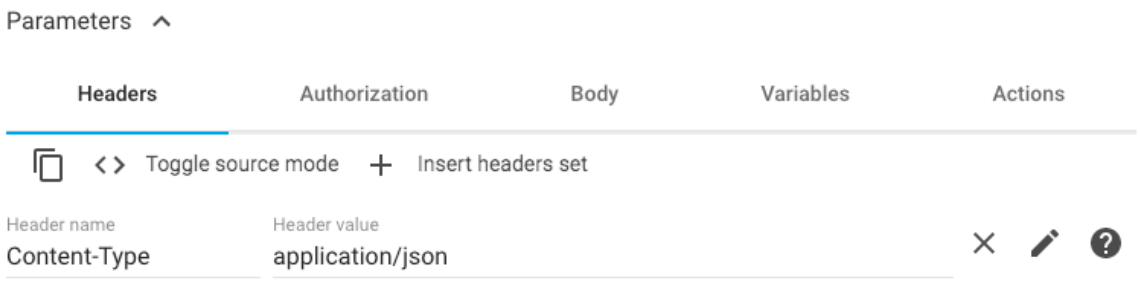


Make a POST request to add data

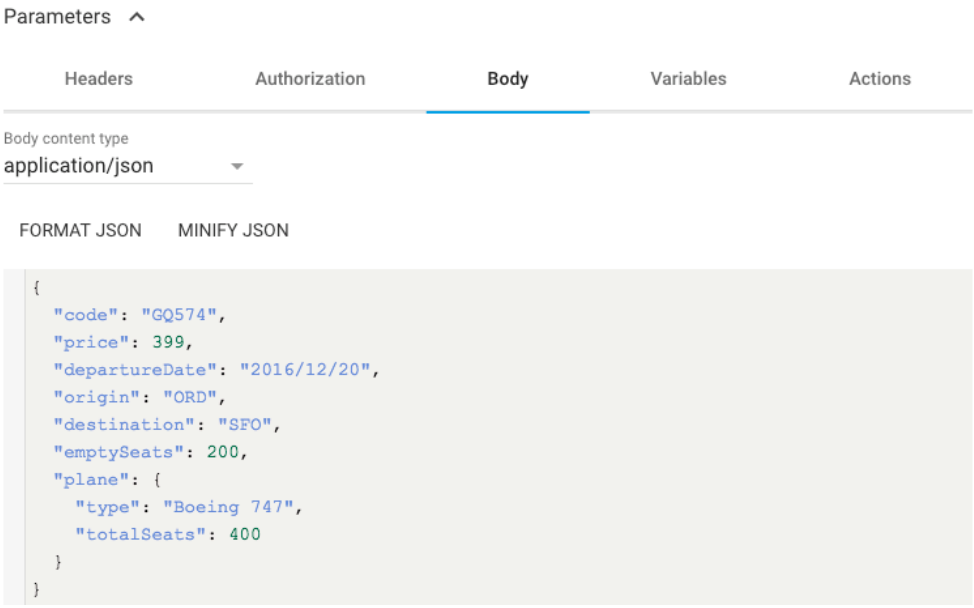
21. Change the method to POST.
22. Click the Send button; you should get a 415 response with a message of unsupported media type.



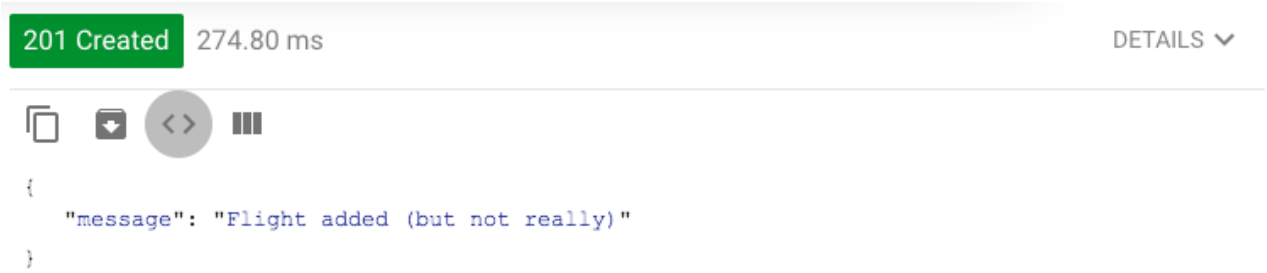
23. Click the arrow next to Parameters beneath the request URL.
24. Click in the Header name field, type C, and then select Content-Type.
25. In the Header value field, select application/json.



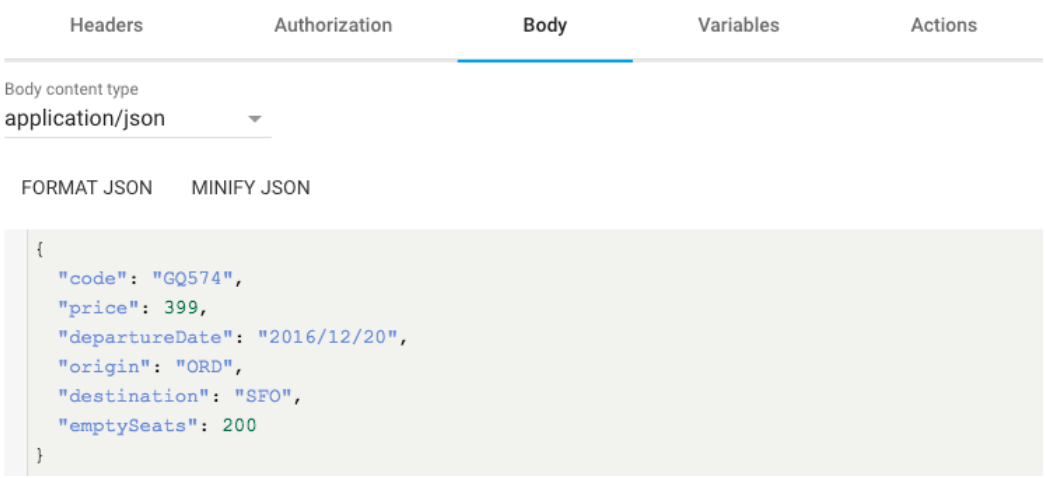
26. Select the Body tab.
27. Return to the course snippets.txt file and copy the value for American Flights API post body.
28. Return to Advanced REST Client and paste the code in the body text area.



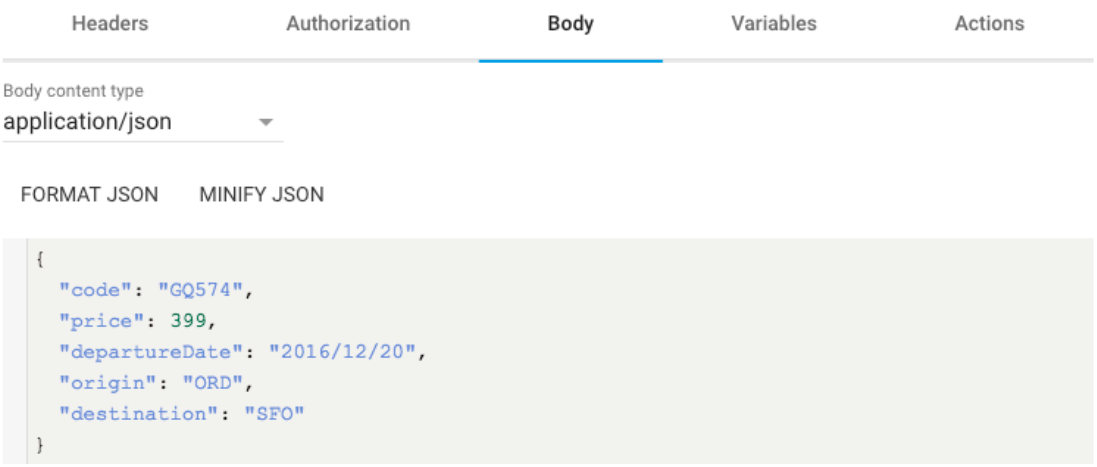
29. Click the Send button; you should see a 201 Created response with the message Flight added (but not really).



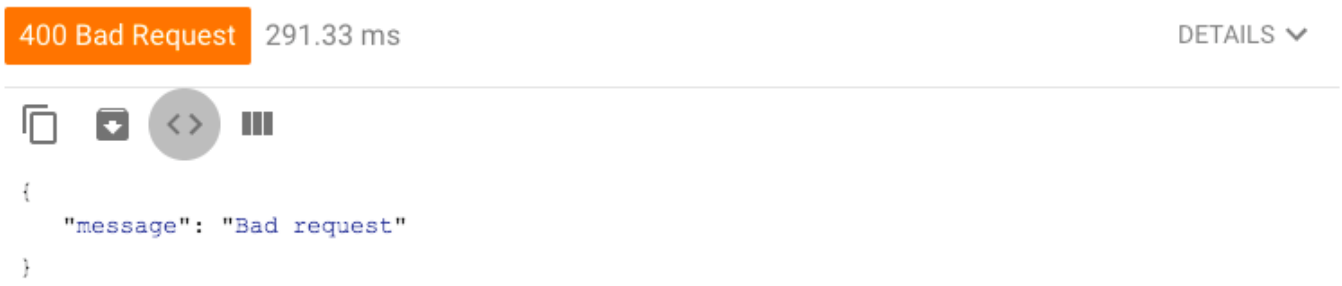
30. Return to the request body and remove the plane field and value from the request body.
31. Remove the comma after the emptySeats key/value pair.



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

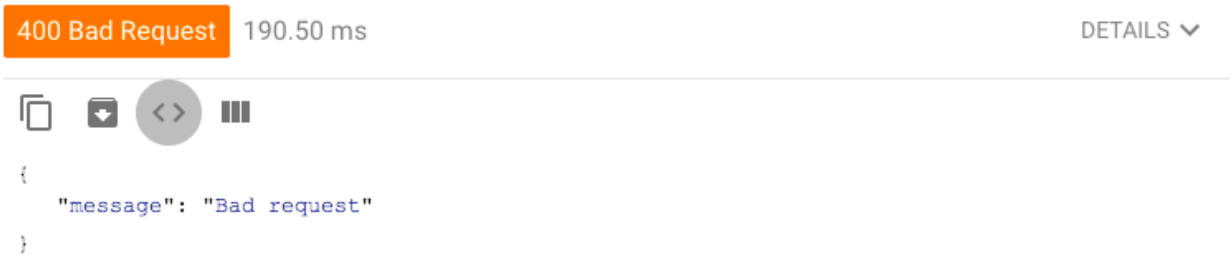


35. Send the request; you should see a 400 Bad Request response with the message Bad request.

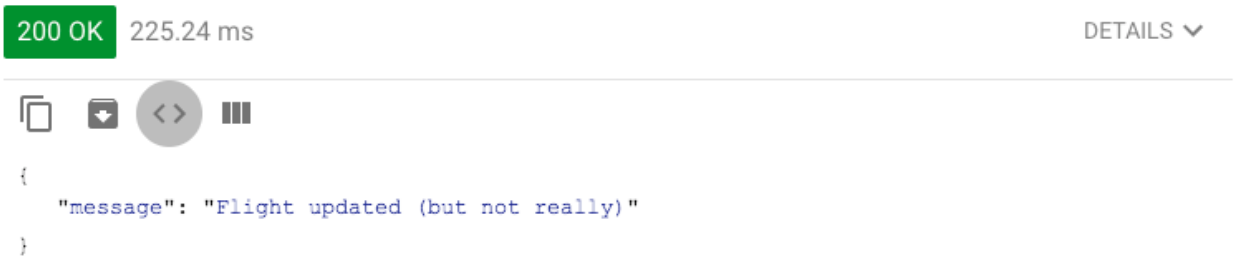


Make a PUT request to update data

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



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

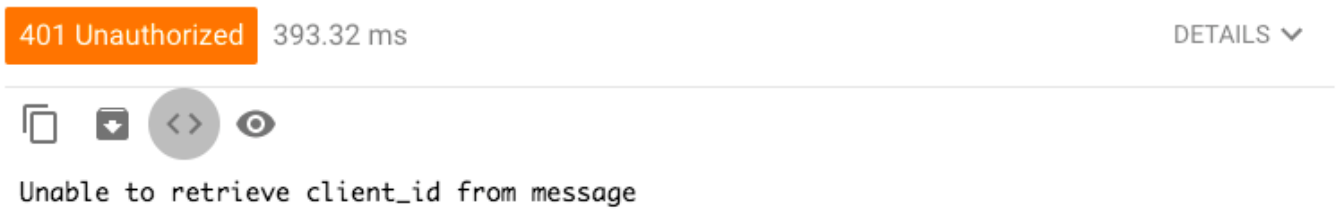


Make a request to a secured API

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

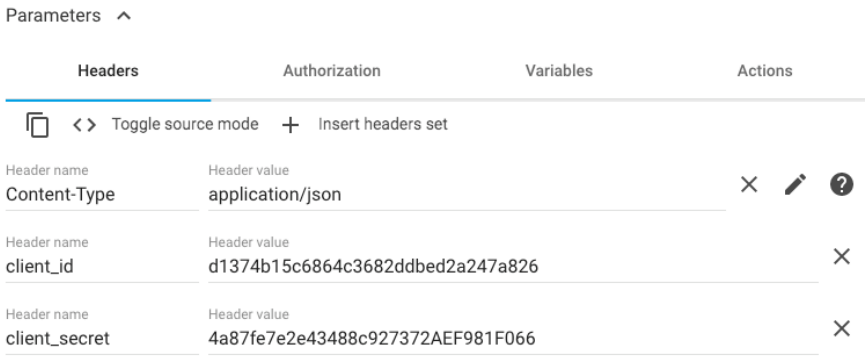
    Note: The -ws in the URL has been changed to -api and the /api removed.

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



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

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



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

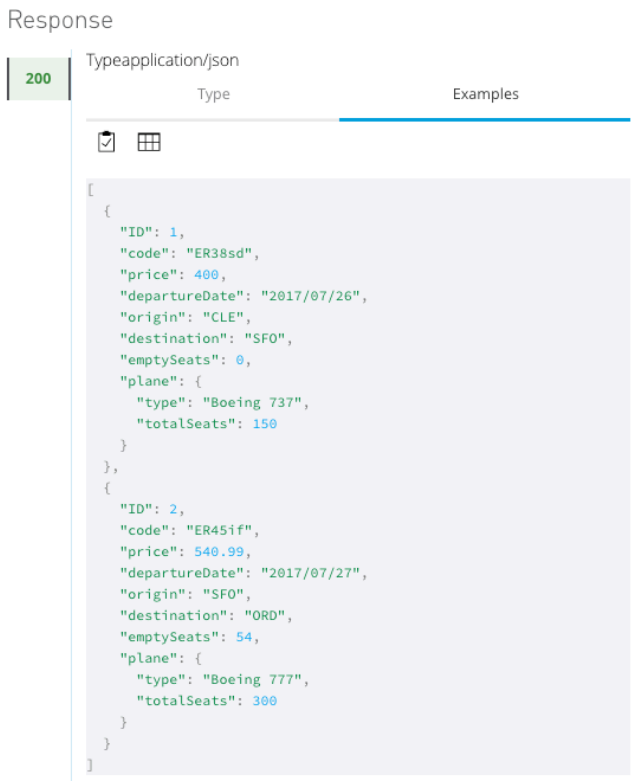
    Note: The API service level agreement (SLA) for the application with this client ID and secret has been set to allow three API calls per minute.

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

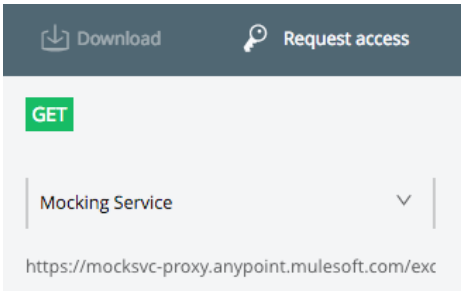
    Note: The API service level agreement (SLA) for the application with this client ID and secret has been set to allow three API calls per minute.

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

52. Return to the bowser window with the American Flights API portal at https://anypoint.mulesoft.com/exchange/portals/muletraining.
53. In the left-side navigation click the GET method for /flights.
54. Review the Headers and Response sections.
55. In the Response section, select Examples.

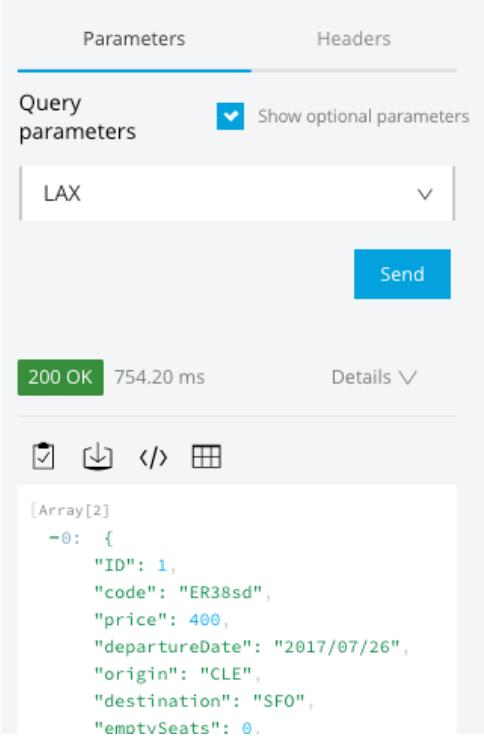


56. In the API console located on the right side of the page, make sure the Mocking Service endpoint is selected.
57. Look at the endpoint URL that is displayed.



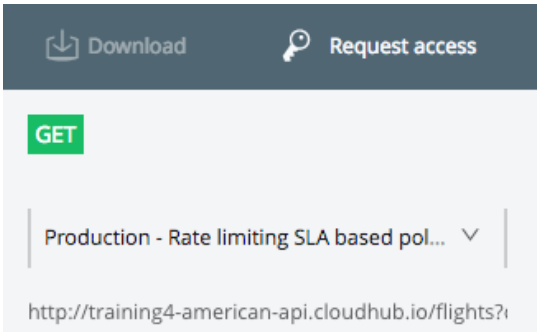
58. Click the Show optional parameters checkbox.
59. Select LAX in the destination drop-down menu.

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

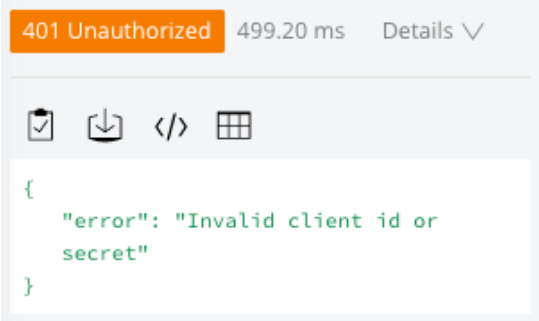


Make requests to the API using an API proxy endpoint

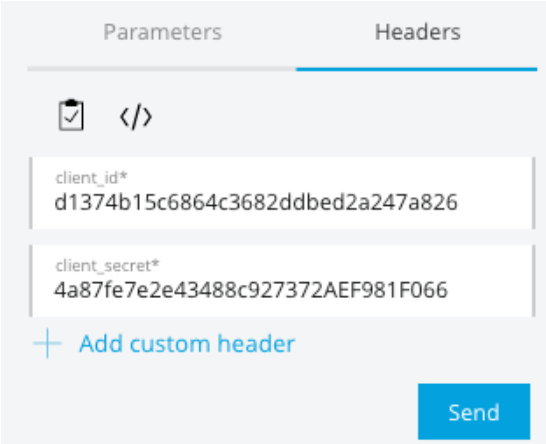
62. At the top of the API console, change the endpoint to Production – Rate limiting SLA based policy.
63. Look at the endpoint URL that is displayed.



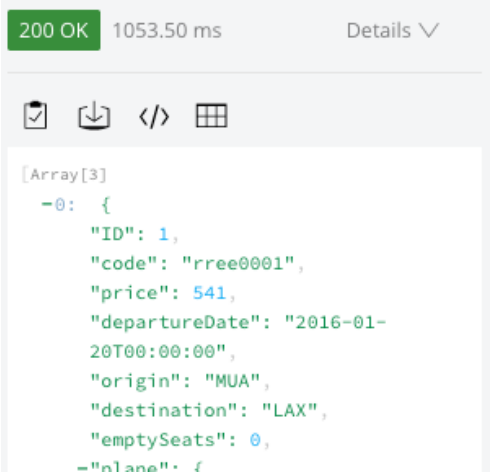
64. Click Send; you should get a 401 Unauthorized response.



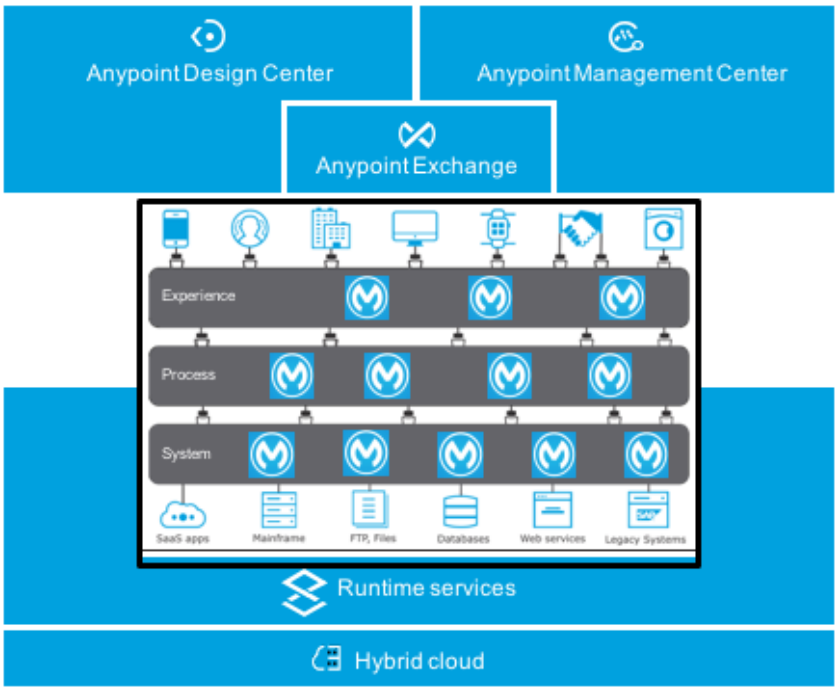
65. Select the Headers tab.
66. Copy and paste the client_id and cient_secret values from the course snippets.txt file.



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



Module 2: Introducing Anypoint Platform



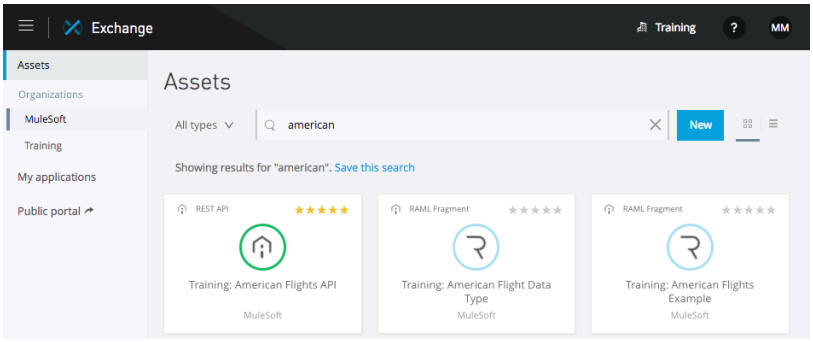
At the end of this module, you should be able to:

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

Walkthrough 2-1: Explore Anypoint Platform and Anypoint Exchange

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

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



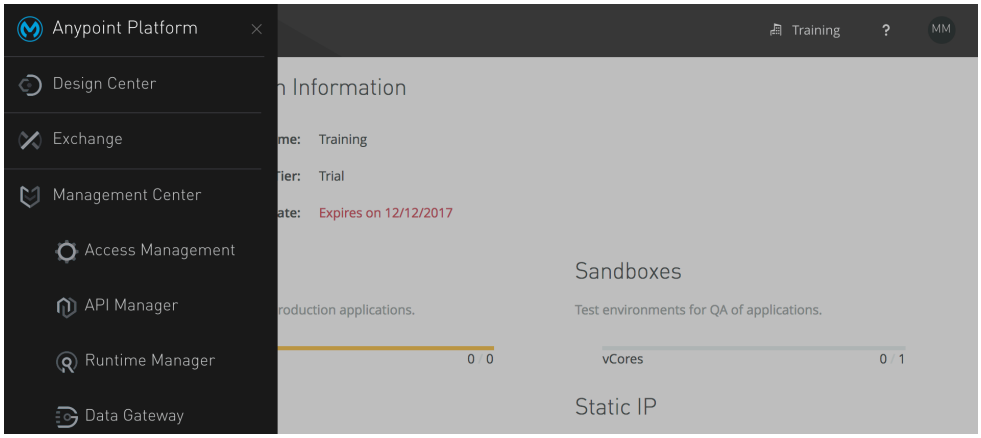
Return to Anypoint Platform

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

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

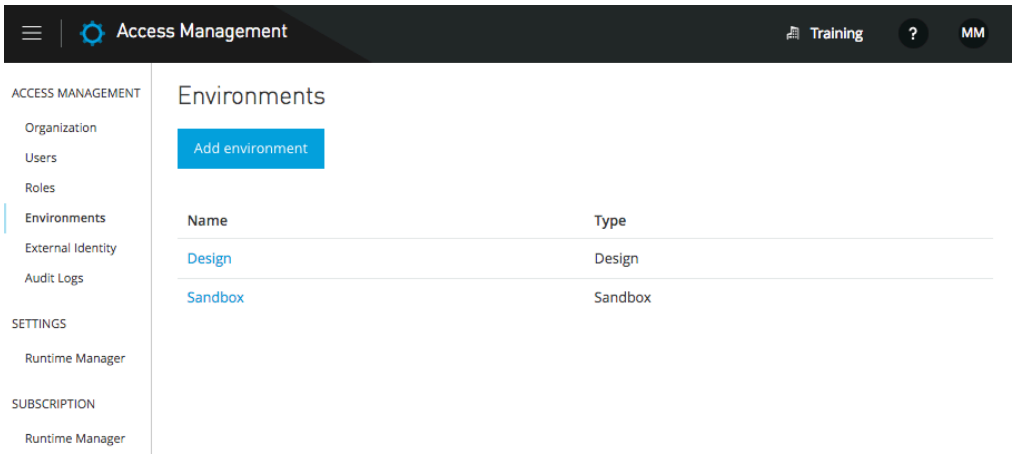
2. Click the menu button located in the upper-left in the main menu bar.
3. In the menu that appears, select Anypoint Platform; this will return you to the home page.

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

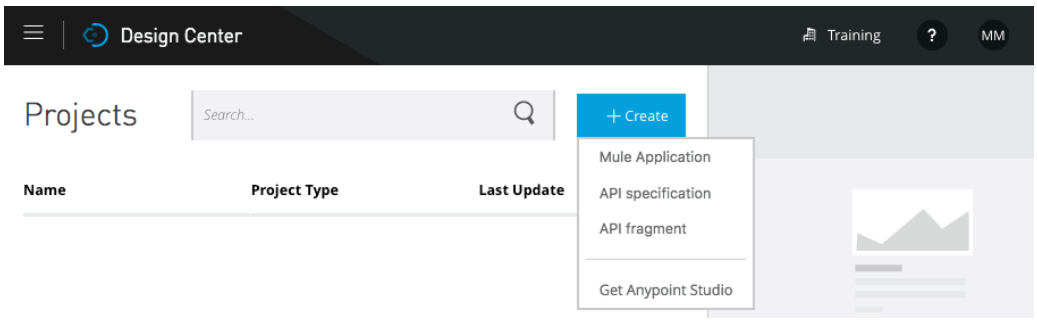


Explore Anypoint Platform

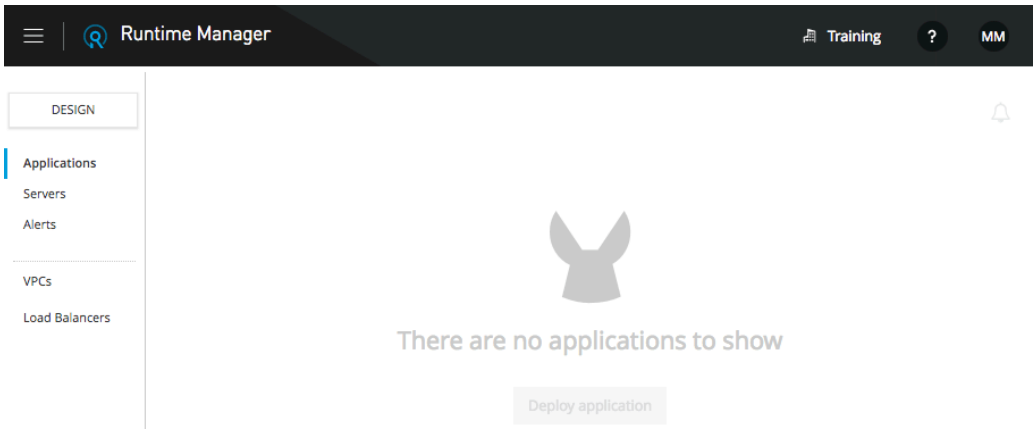
4. In the main menu, select Access Management.
5. In the left-side navigation, select Users.
6. In the left-side navigation, select Environments.



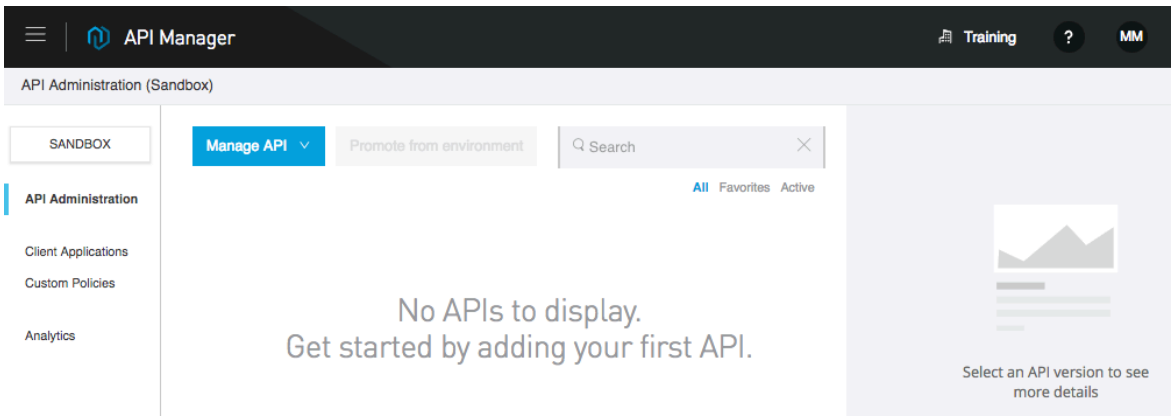
7. In the main menu, select Design Center.
8. Click the Create button and look at the options in the drop-down menu.



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

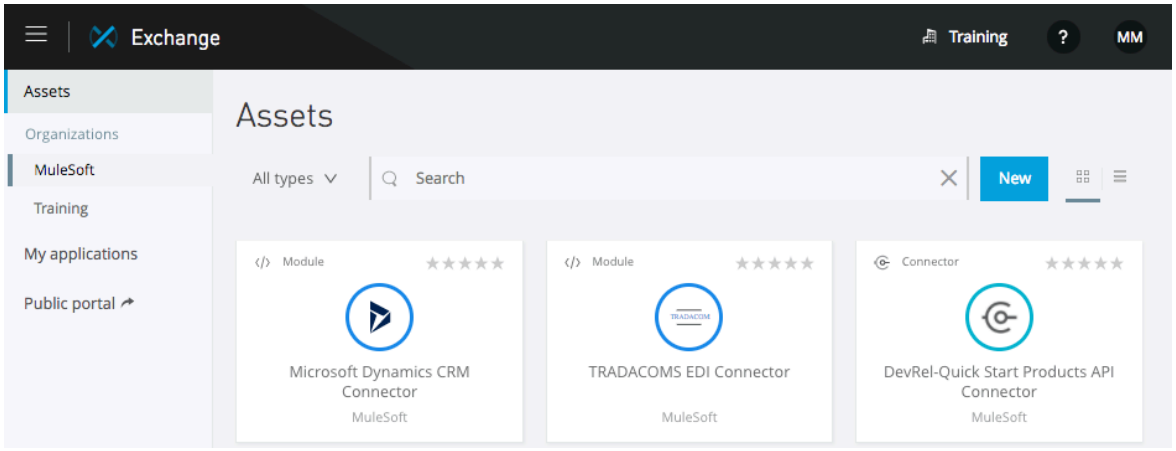


11. In the main menu, select API Manager.

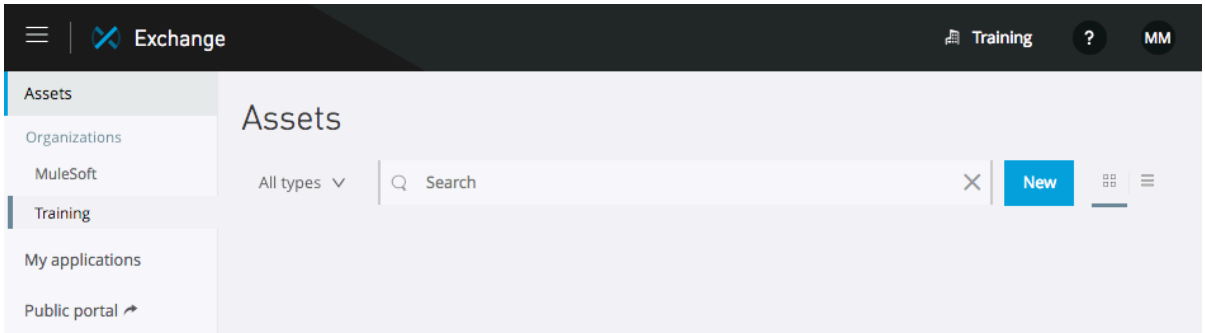


Explore Anypoint Exchange

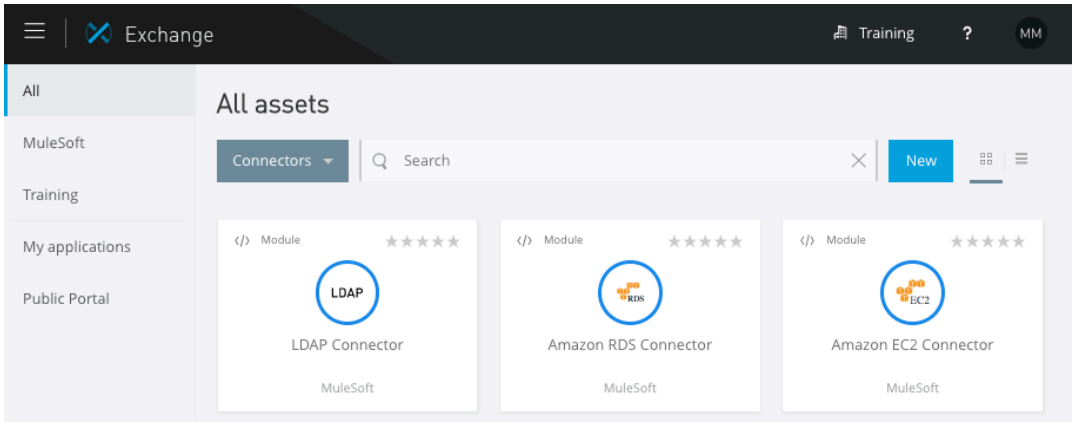
12. In the main menu, select Exchange.
13. In the left-side navigation, select MuleSoft; you should see all the content in the public Exchange.



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

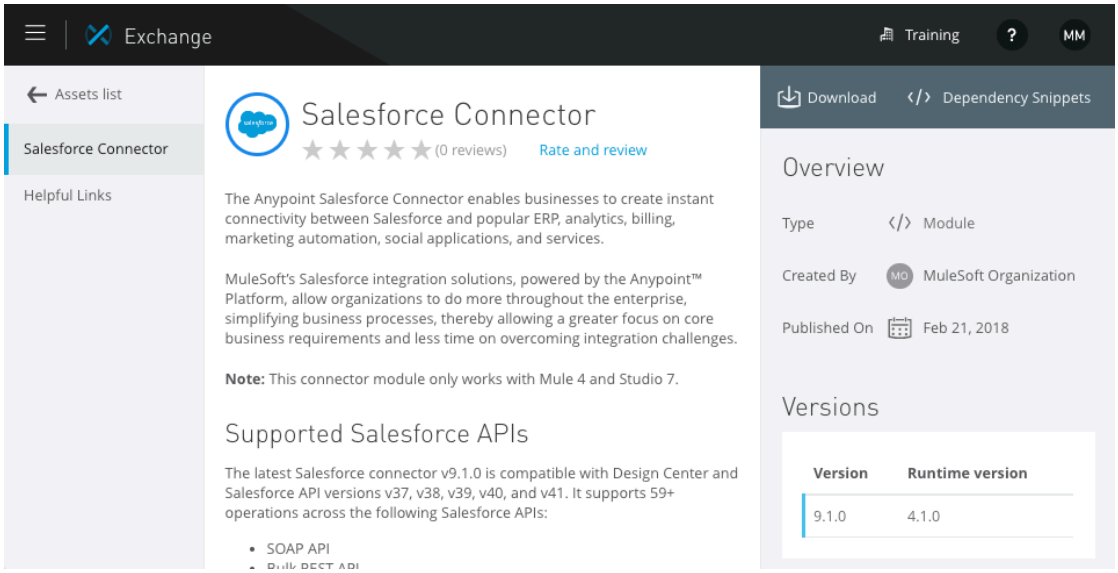


15. In the left-side navigation, select MuleSoft.
16. In the types menu, select Connectors.



17. Select one of the connectors and review its information.
18. In the left-side navigation, click the Assets list link.
19. Search for the Salesforce Connector and review its details.

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

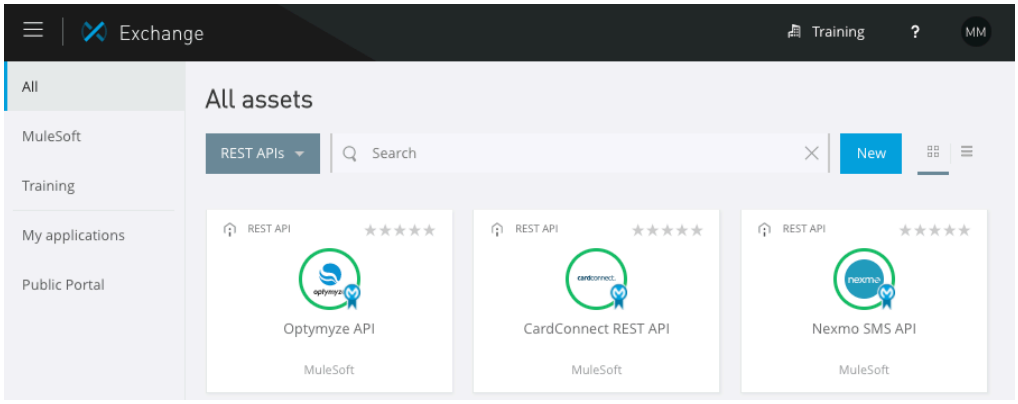


20. In the left-side navigation, click Assets list.
21. In the types menu, select Templates.
22. Remove salesforce from the search field and press Enter/Return.

Browse REST APIs in Anypoint Exchange

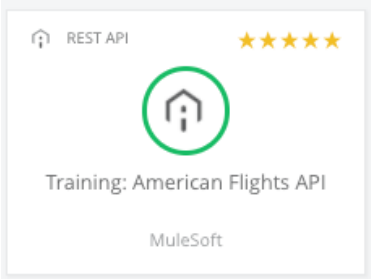
23. In the types menu, select REST APIs.

24. Browse the APIs.

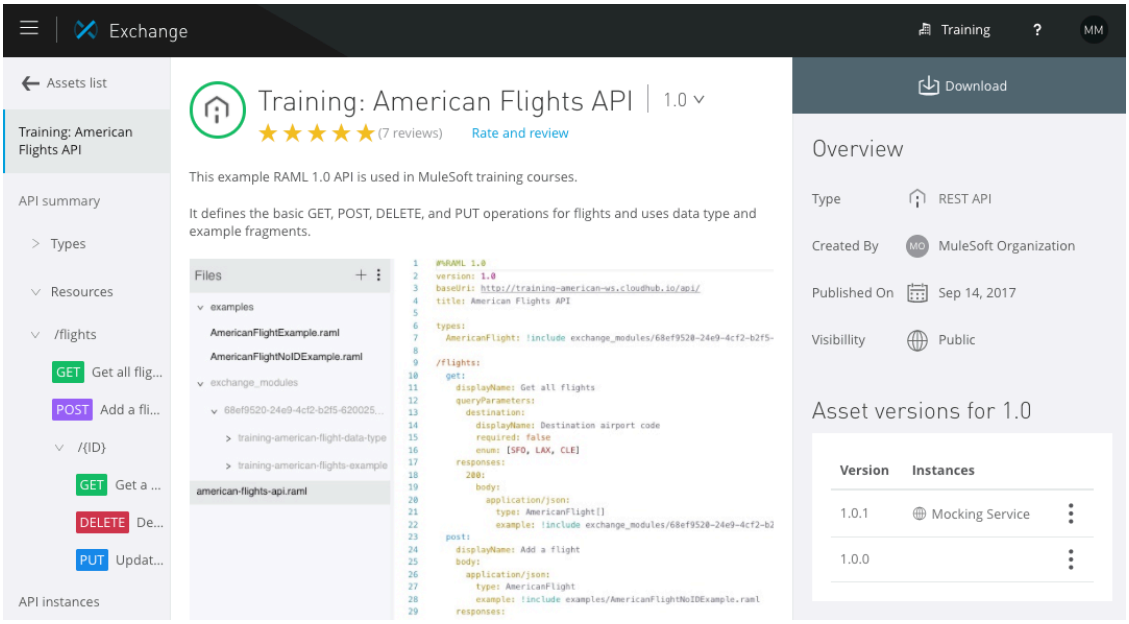


Discover and review the API portal for the Training: American Flights API

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

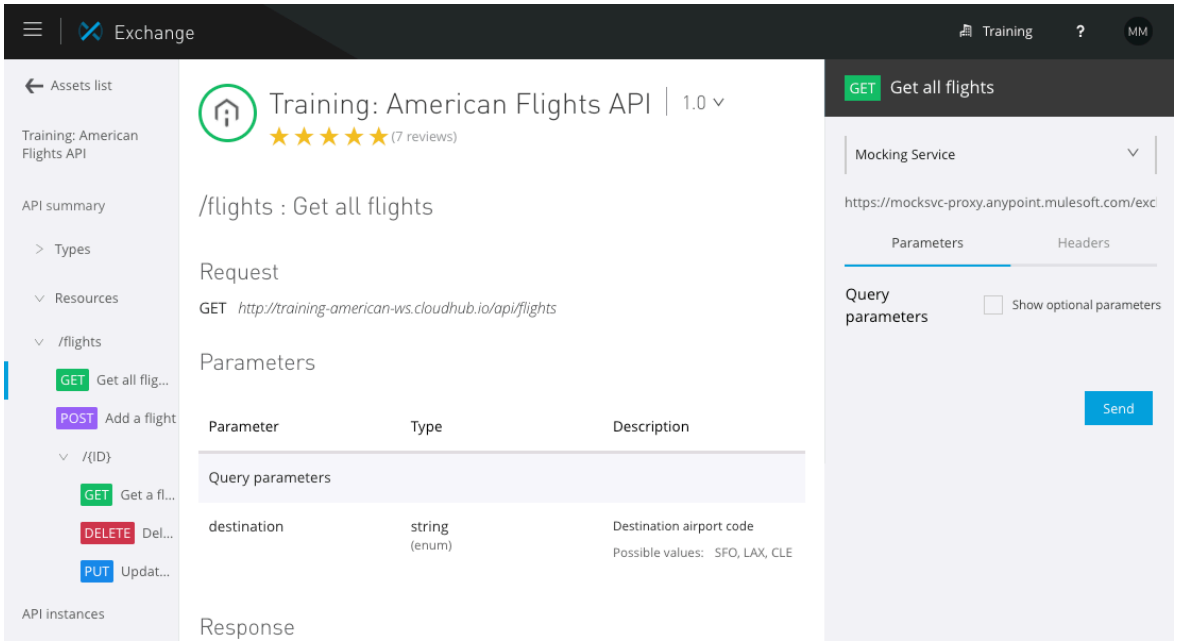


26. Review the API portal.



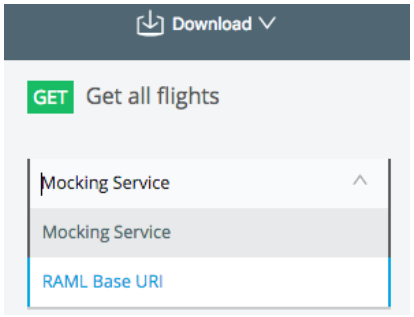
27. In the left-side navigation, expand and review the list of available resources.
28. Click the GET link for the /flights resource.

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



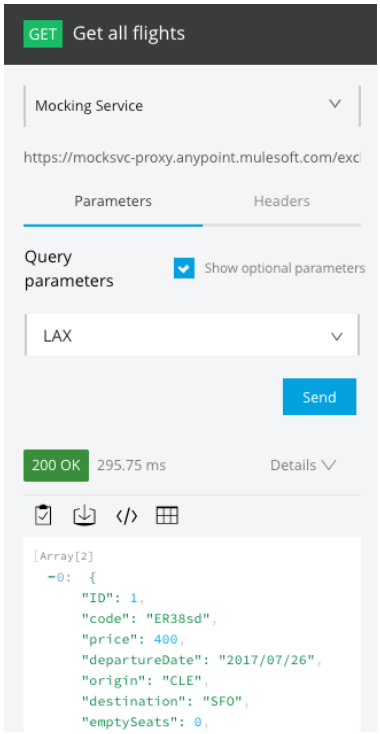
Use the API console to make calls to the Training: American Flights API

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

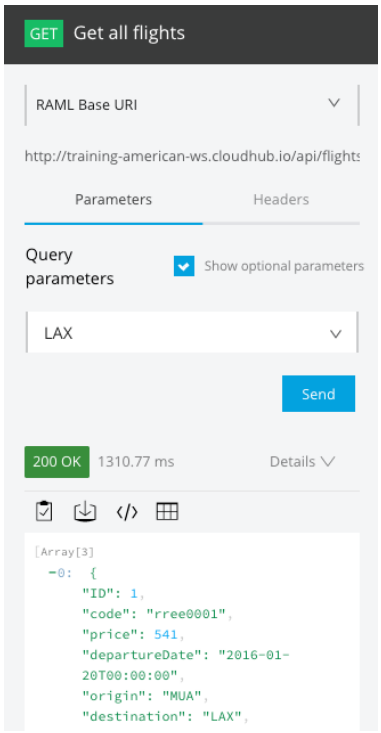


31. Select Mocking Service.

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



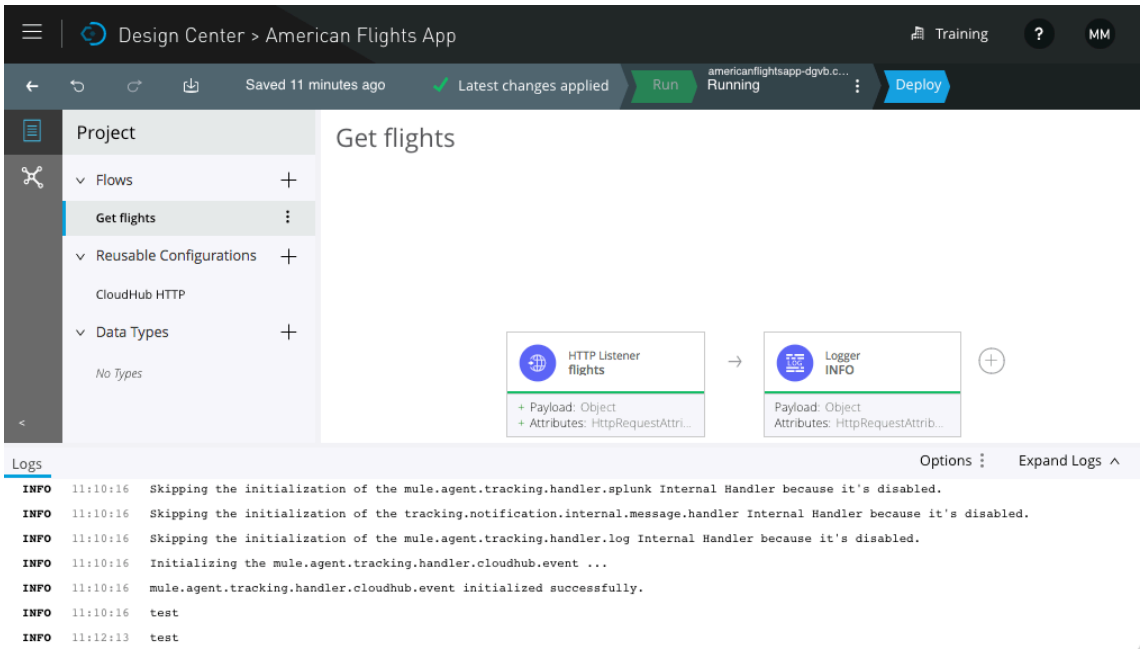
33. Change the API instance from Mocking Service to RAML Base URI.
34. Click Send again; you should get results from the actual API implementation for the destination you selected.



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

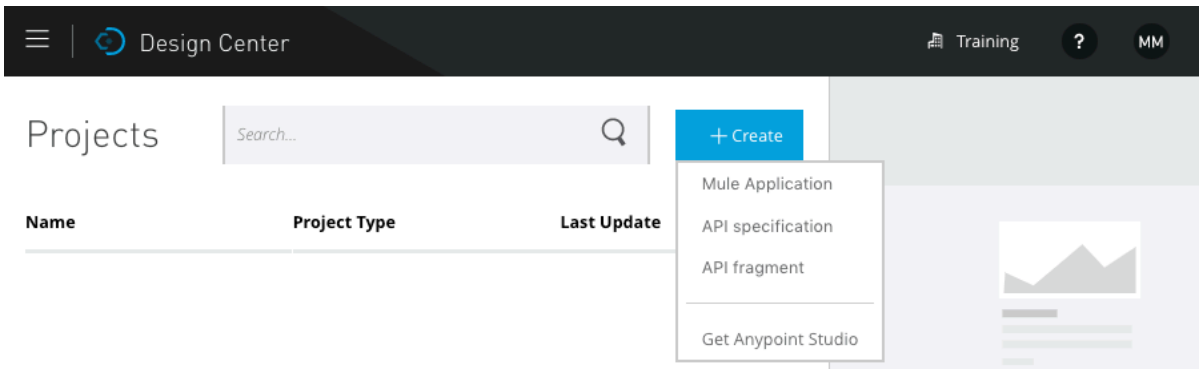
In this walkthrough, you build, run, and test a basic Mule application with flow designer. You will:

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

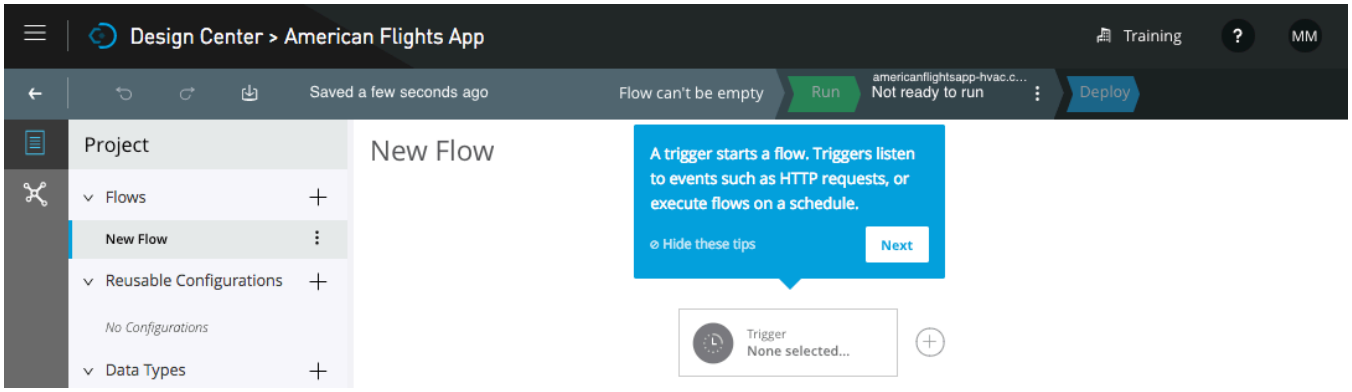


Create a Mule application project in Design Center

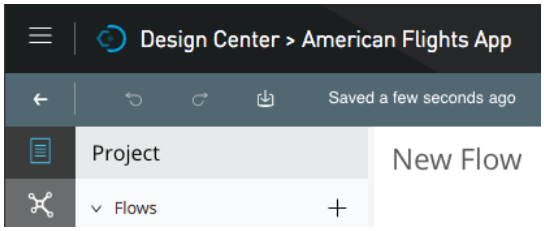
1. Return to Anypoint Platform.
2. In the main menu, select Design Center.
3. Click the Create button and select Mule Application.



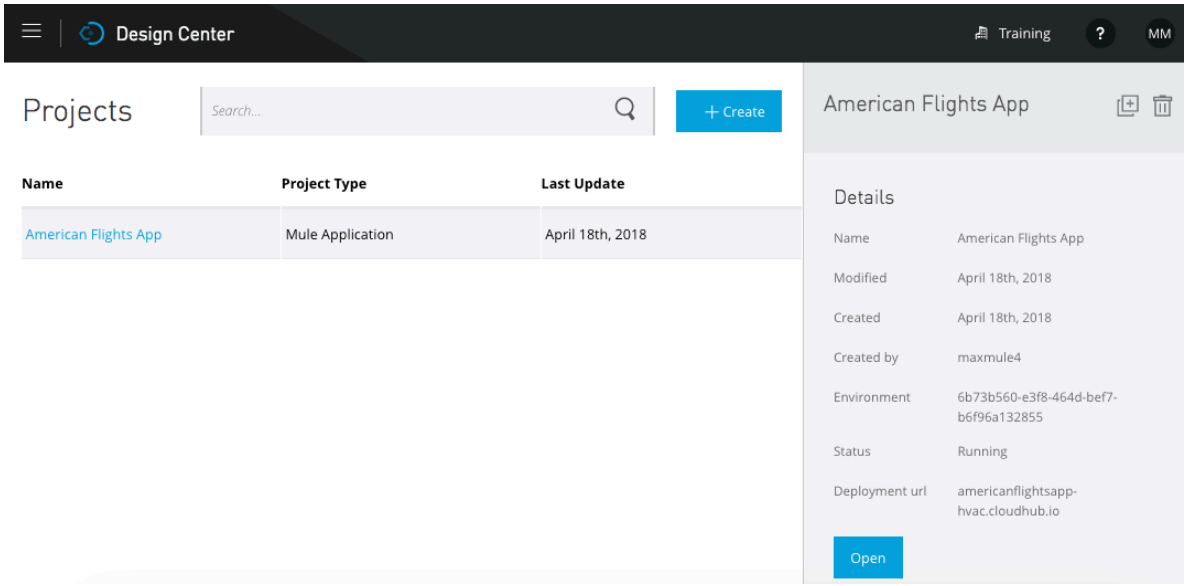
4. In the New Mule Application dialog box, set the project name to American Flights App.
5. Click Create; flow designer should open.



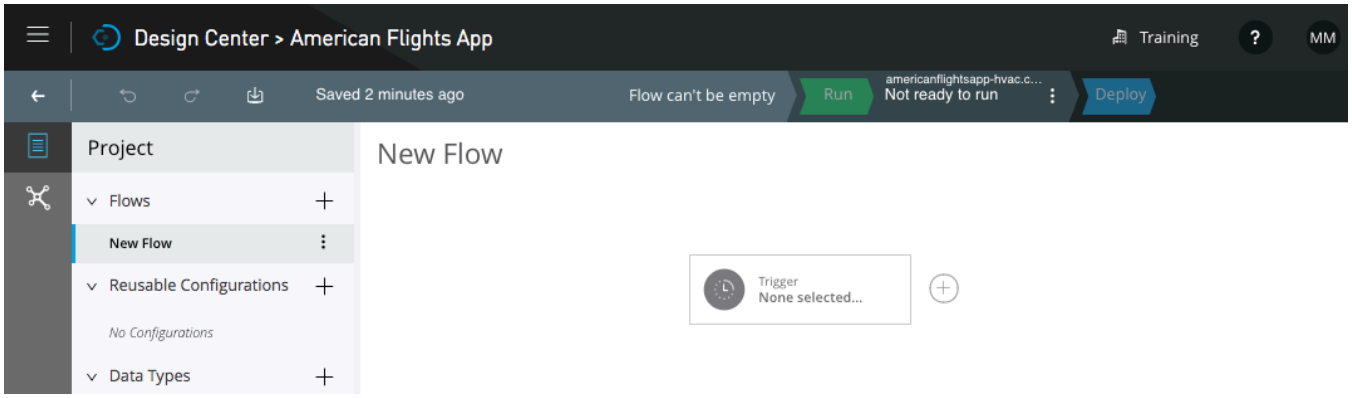
6. In the pop-up box in flow designer, click Hide these tips.
7. Click the arrow icon in the upper-left corner; you should return to the Design Center.



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

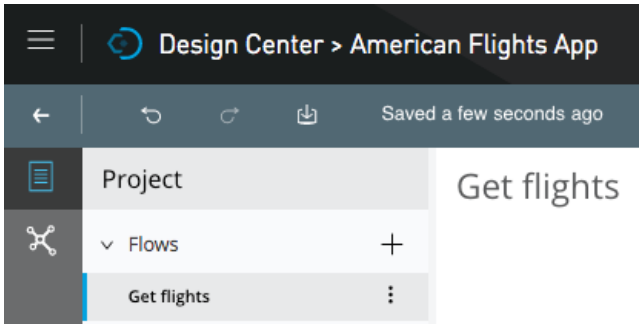


9. Click the Open button or click the American Flights App link in the project list; the project should open in flow designer.



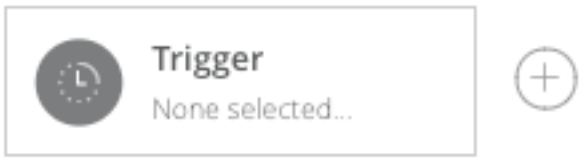
Rename the flow

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

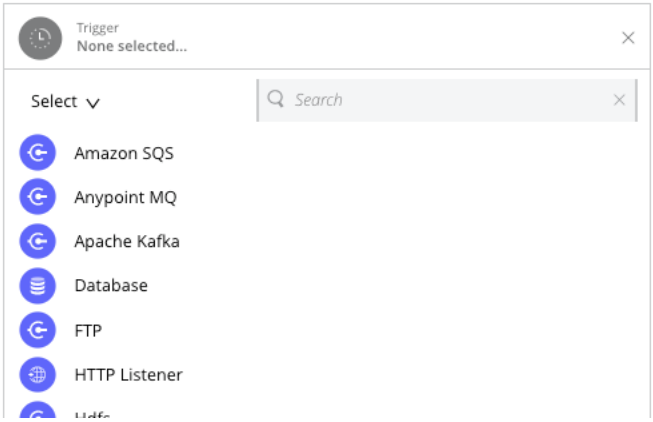


Create an HTTP trigger for a flow in the application

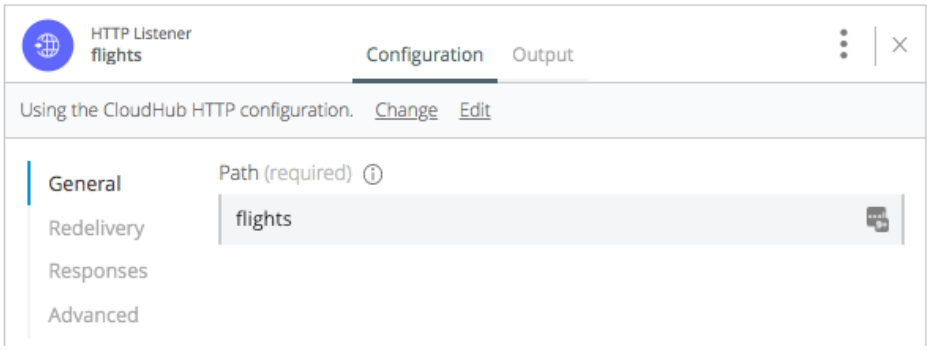
13. In flow designer, click the Trigger card.



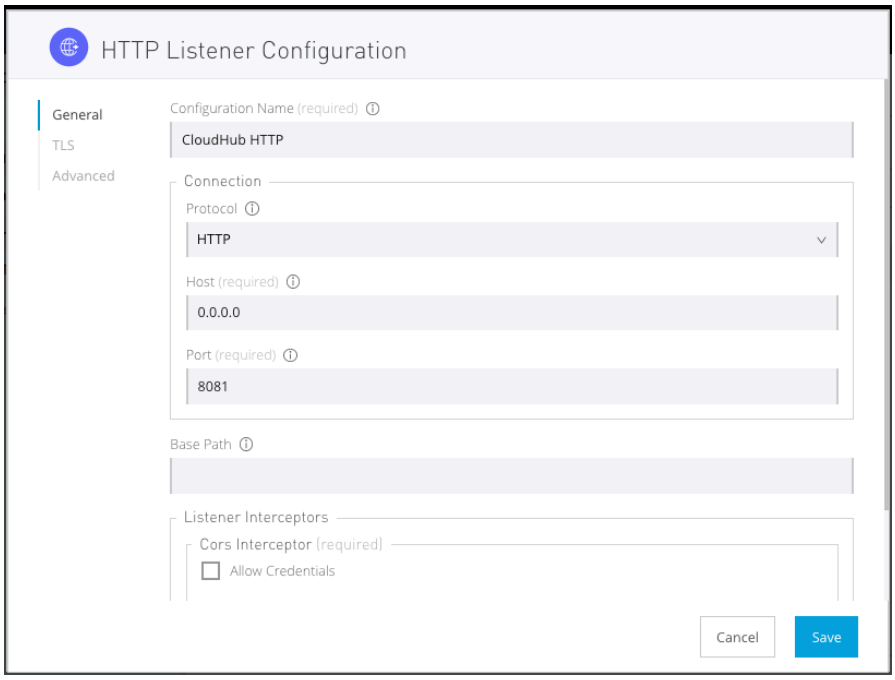
14. In the Trigger card, select HTTP Listener.



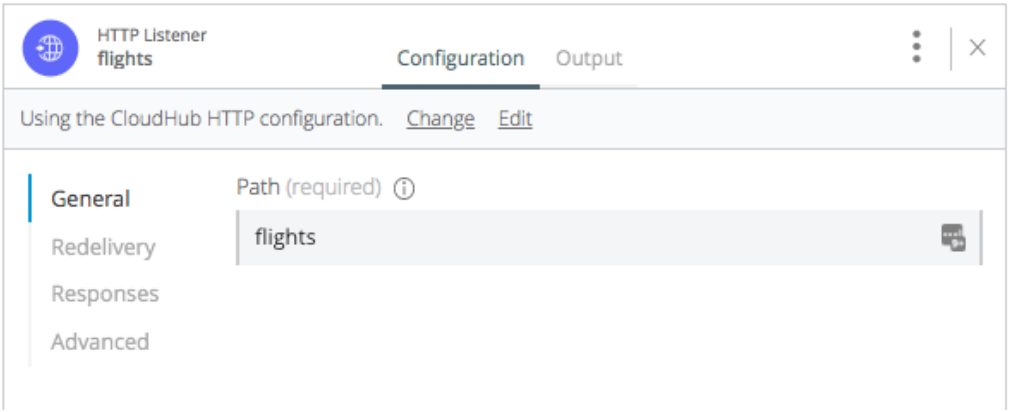
15. In the HTTP Listener dialog box, set the path to flights.



16. Click the Edit link for the CloudHub HTTP configuration.
17. In the HTTP Listener Configuration dialog box, review the information and click Cancel.

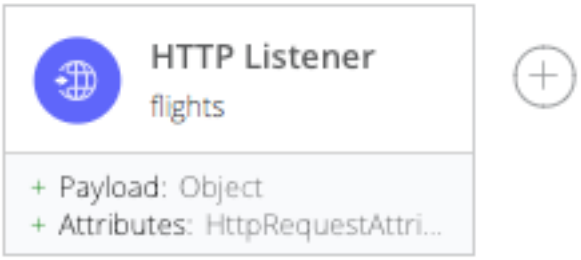


18. In the HTTP Listener dialog box, click the close button in the upper-right corner.

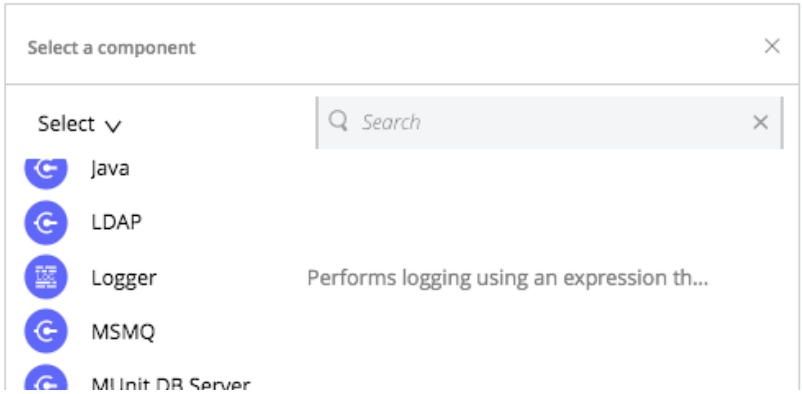


Add a Logger

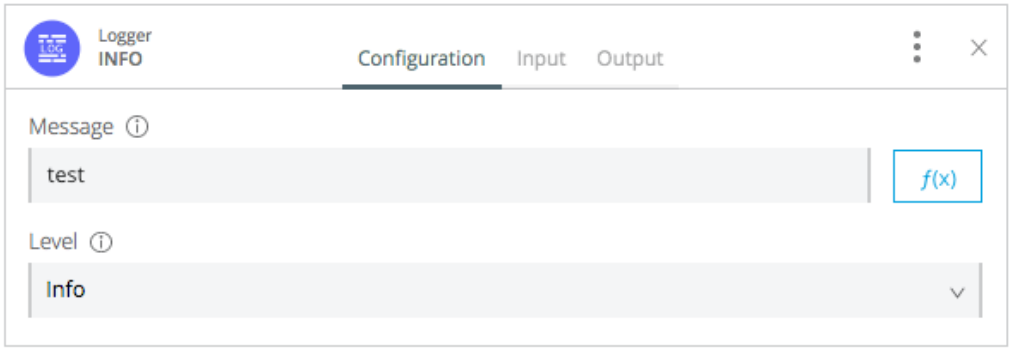
19. Click the add button next to the HTTP Listener card.



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



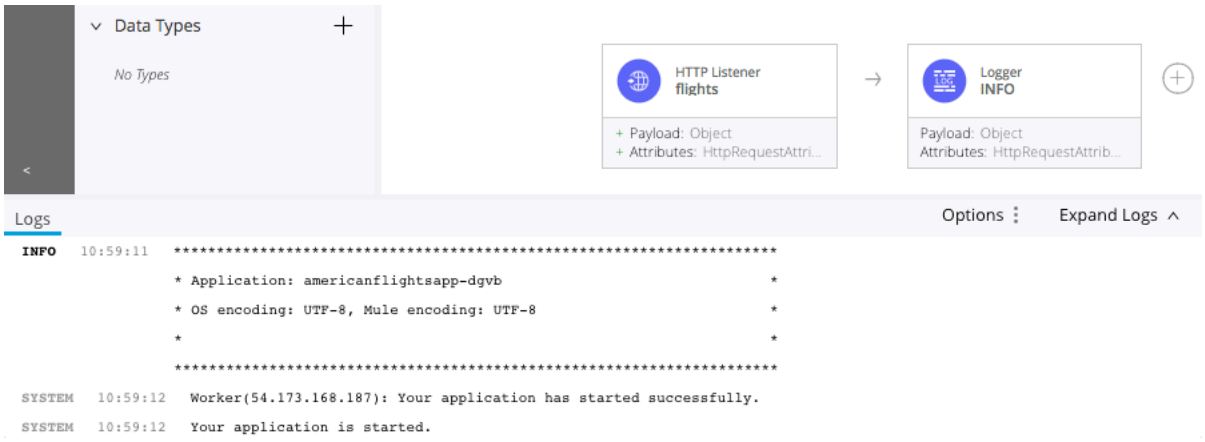
21. In the Logger dialog box, set the message to test.



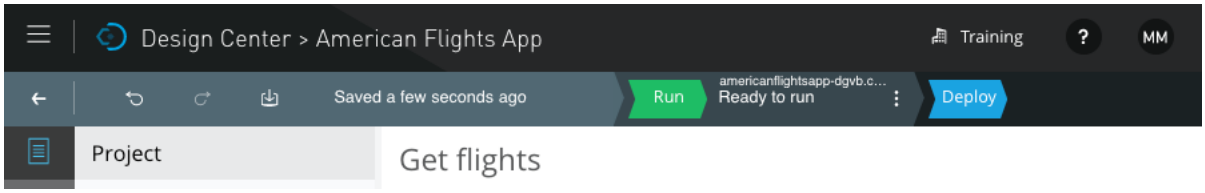
22. Close the card.
23. Notice that there are gray lines across the middle of both cards.

Deploy the application

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



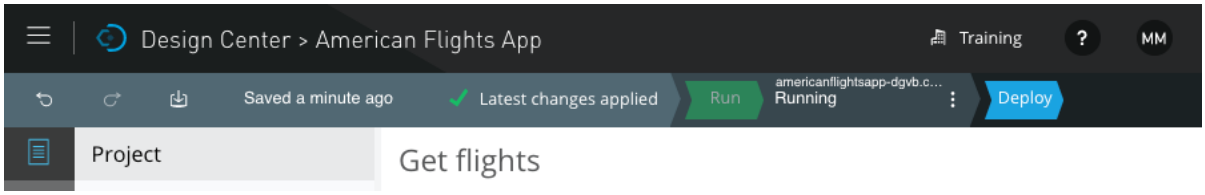
25. Locate the application status in the main menu bar; it should say Ready to run.



26. Look at the generated URL for the application.

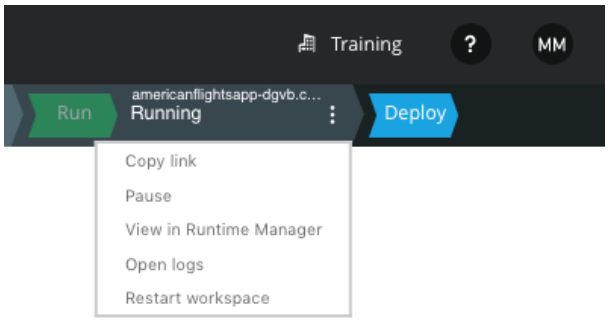
    Note: The application name is appended with a four-letter suffix to guarantee that it is unique across all applications on CloudHub.

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



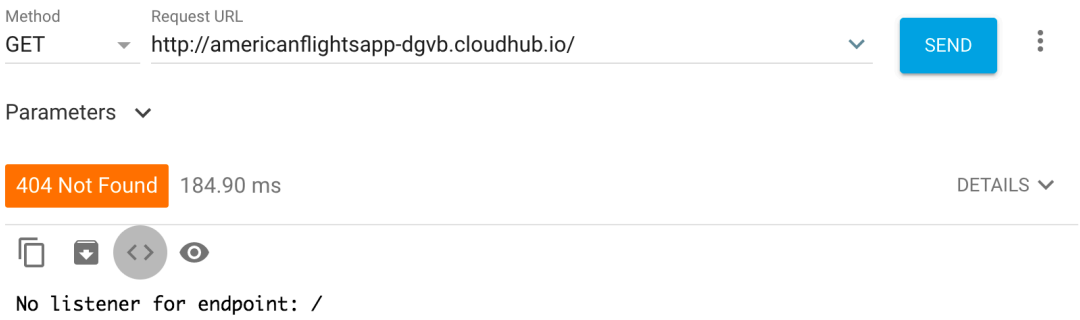
    Note: If your application fails to run, look at the messages in the Logs panel. If there is no message about incorrect syntax, try restarting the workspace by clicking the options menu in the application status area and selecting Restart workspace.

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

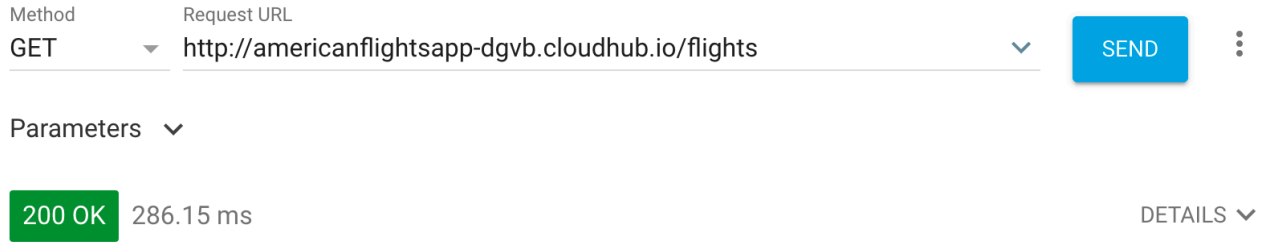


Test the application

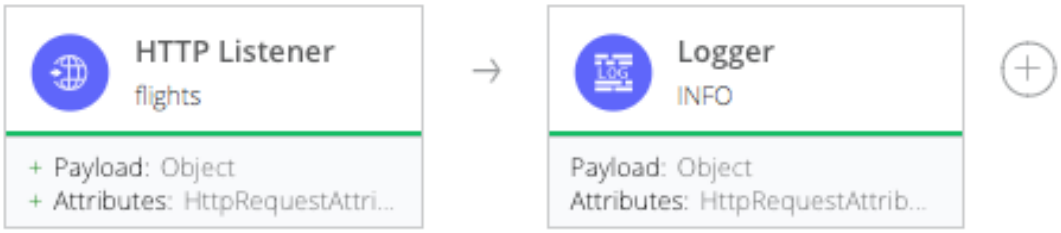
30. Return to Advanced REST Client, paste the copied link, and click Send; you should get a 404 Not Found status with a No listener for endpoint: / message.



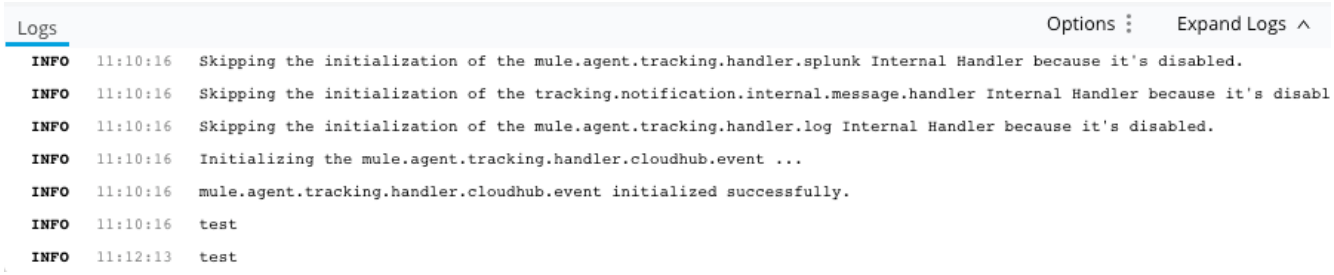
31. Add /flights to the path and click Send; you should get a 200 response with no body.



32. Click Send again to make a second request.
33. Return to flow designer.
34. Notice that there are now green lines across both cards.

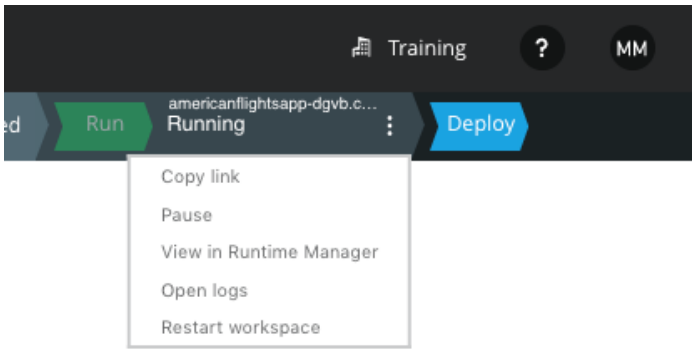


35. Look at the logs; you should see your Logger message displayed twice.

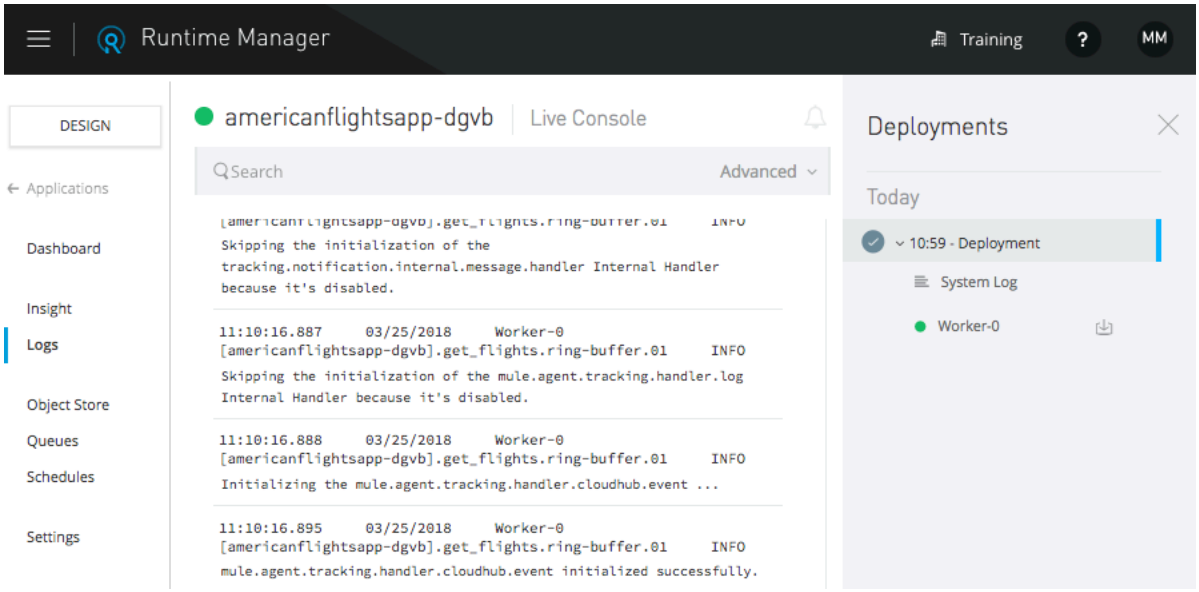


View the application in Runtime Manager

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

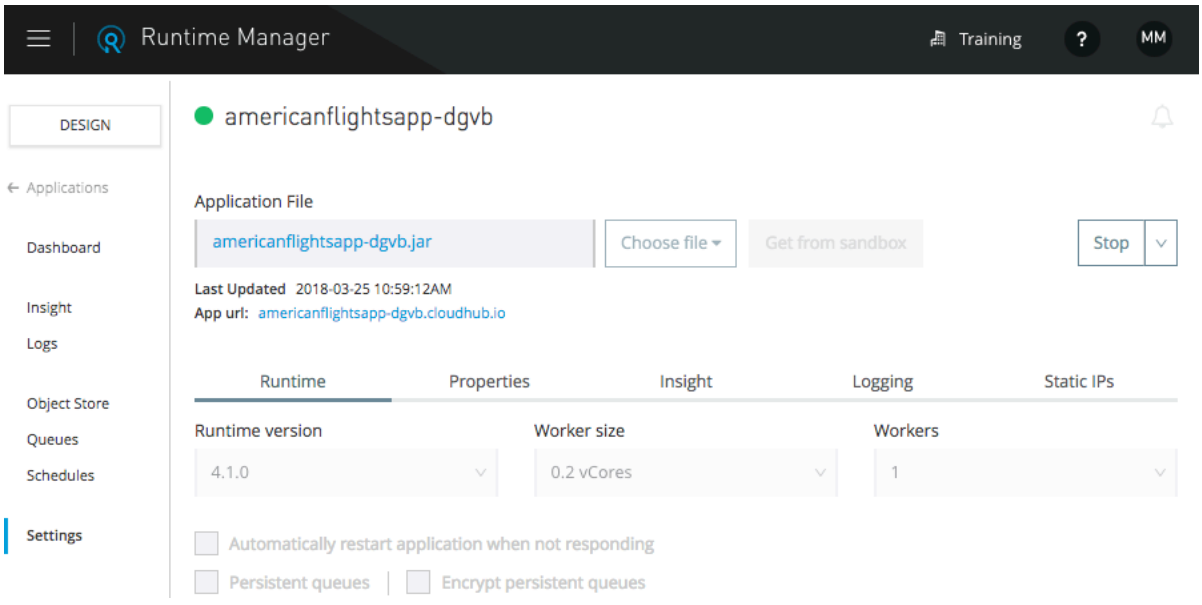


37. In the new browser tab that opens with Runtime Manager, review the application log file; you should see your test log messages.



38. In the left-side navigation, click Settings.
39. Review the settings page and locate the following information for the application:

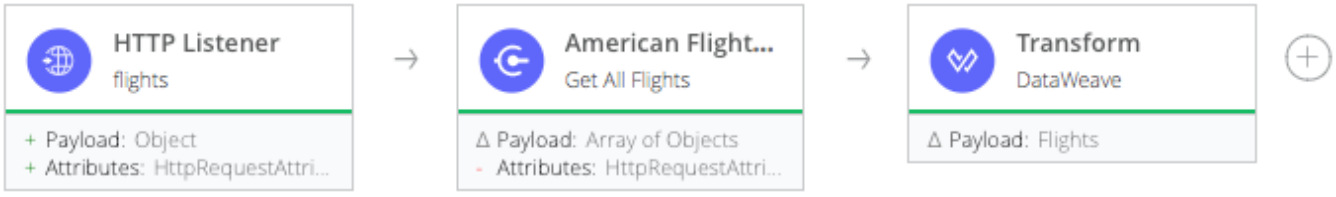
- To which environment it was deployed
- To what type of Mule runtime it was deployed
- To what size worker it was deployed



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

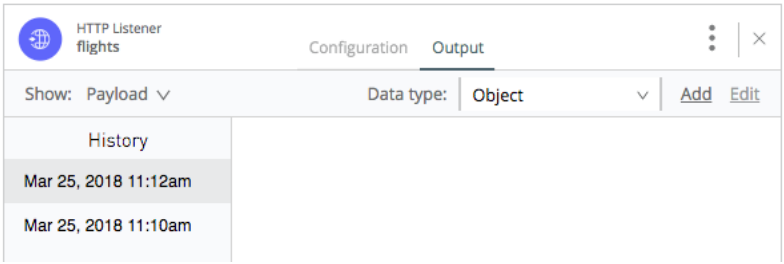
In this walkthrough, you build an integration application to consume an API from Anypoint Exchange. You will:

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



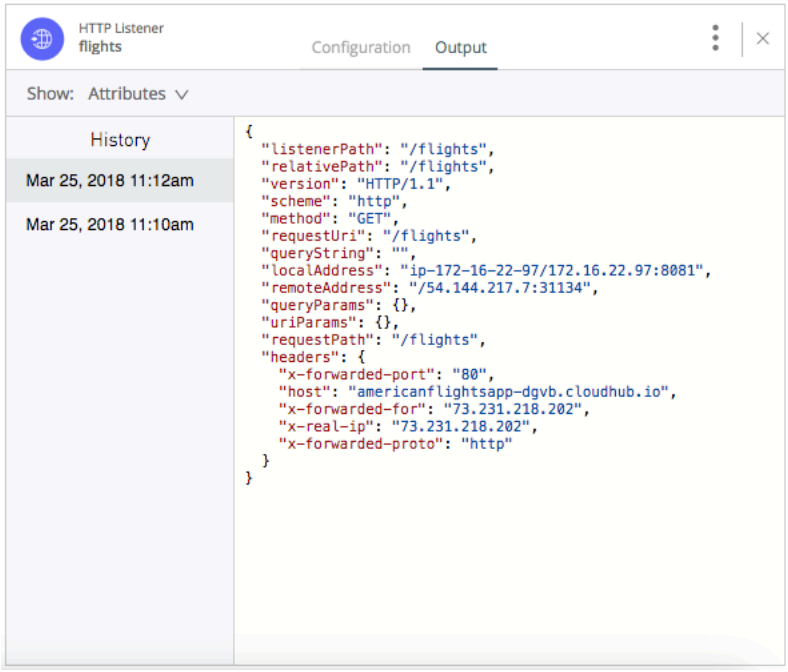
Review Mule event data for the calls to the application

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

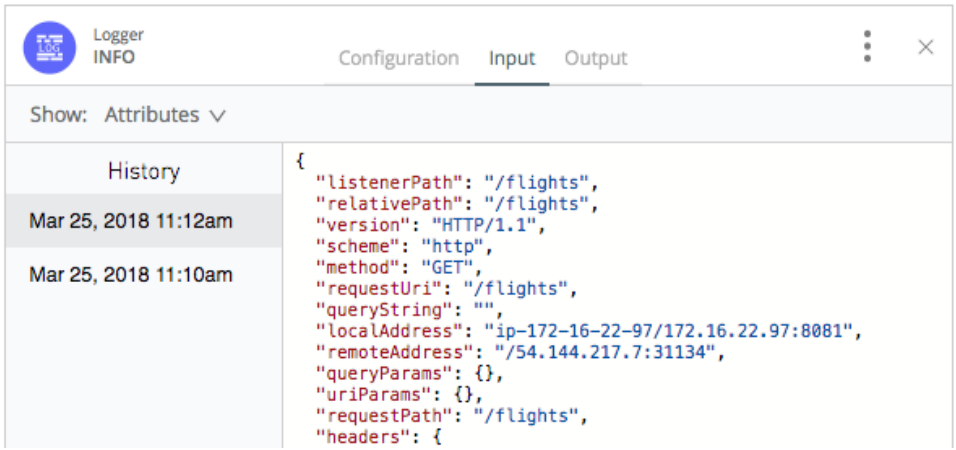


6. Locate your two calls to the application in the History panel; there should be no message payload for either call.
7. Change the Show drop-down menu to Attributes.

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



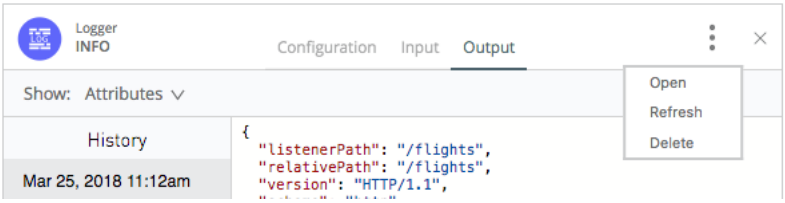
9. Close the card.
10. Open the Logger card.
11. Select the Input tab.
12. Review the payload and attributes values for the two calls.



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

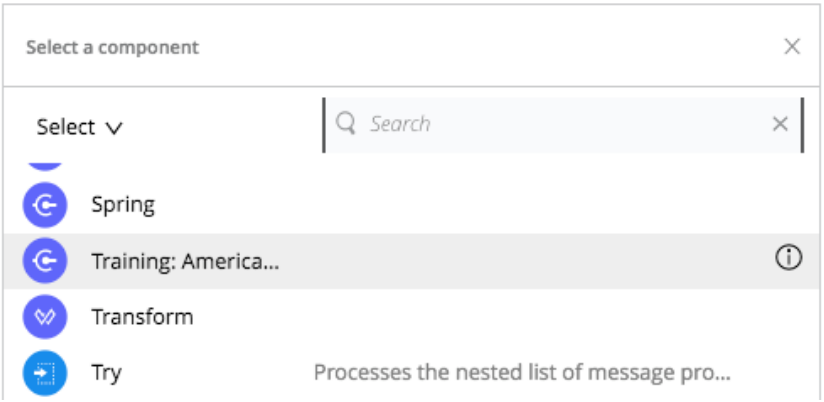
Delete a card

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

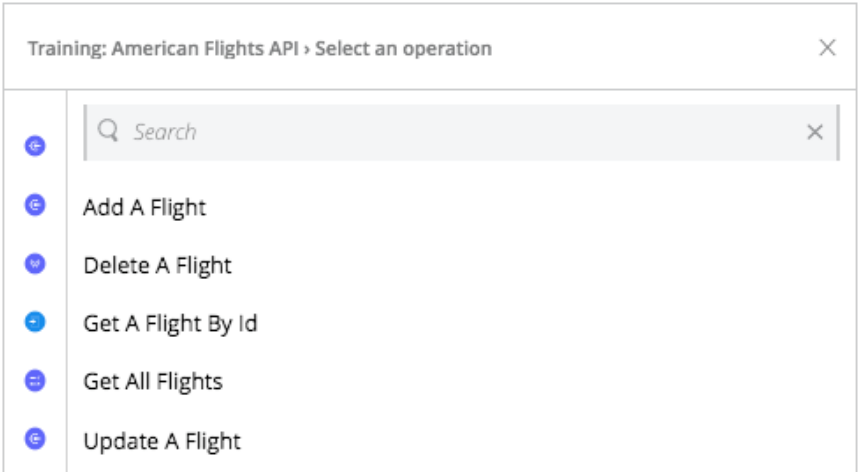


Use the American Flights API in Anypoint Exchange to get all flights

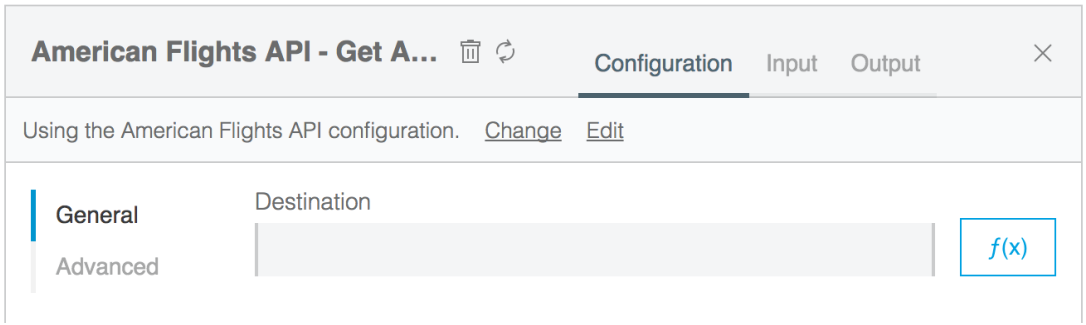
15. Click the Add button next to the HTTP Listener card.
16. In the Select a component dialog box, select the Training: American Flights API.



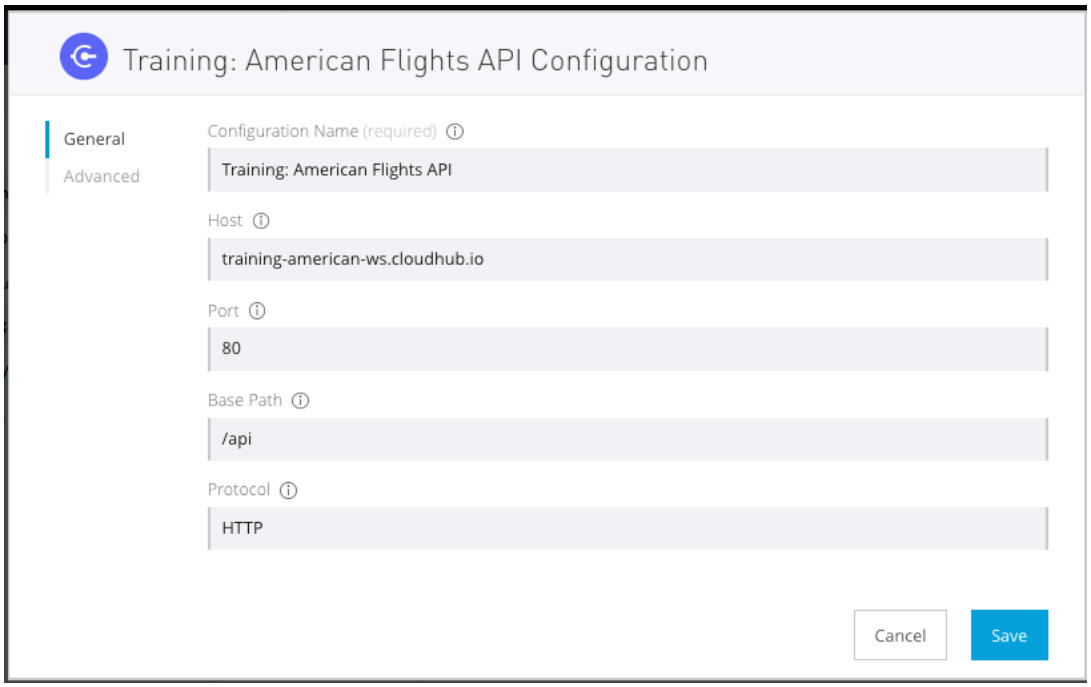
17. In the Training: American Flights API > Select an operation dialog box, select Get All Flights.



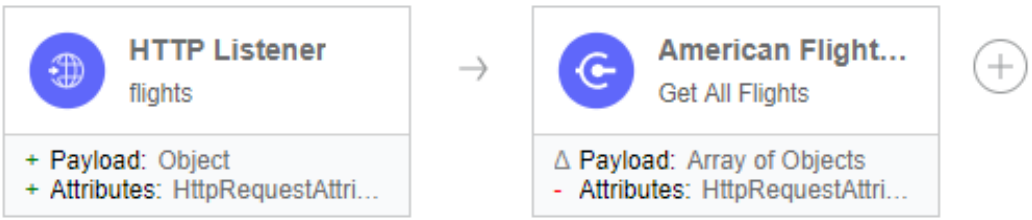
18. In the Get All Flights dialog box, click the Edit link for the Training: American Flights API configuration.



19. Review the information and click Cancel.



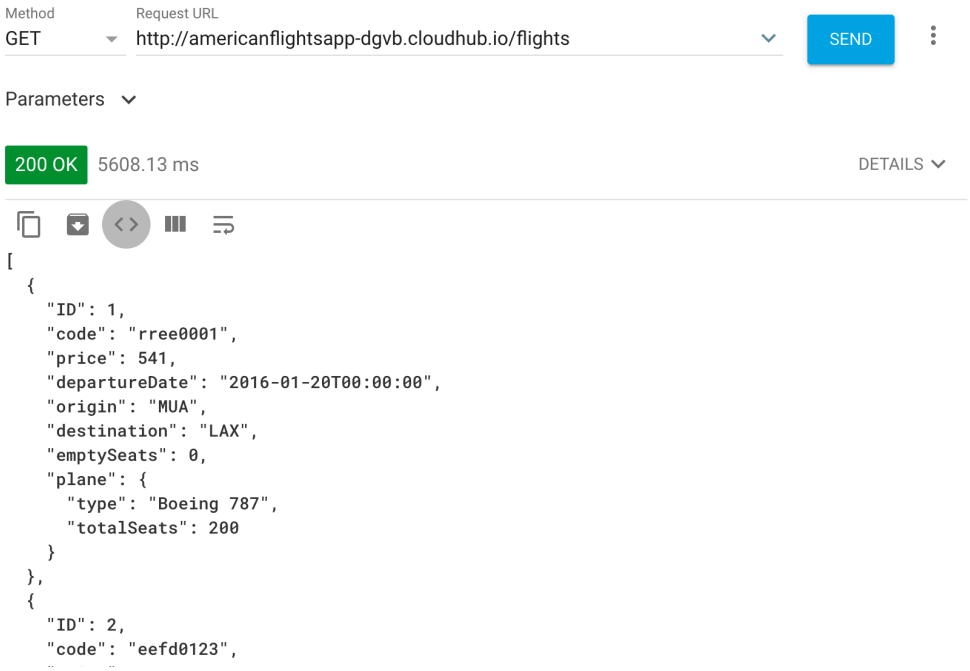
20. Close the American Flights API card.



21. Click the Run button in the main menu bar.
22. Wait until the application is running.

Test the application

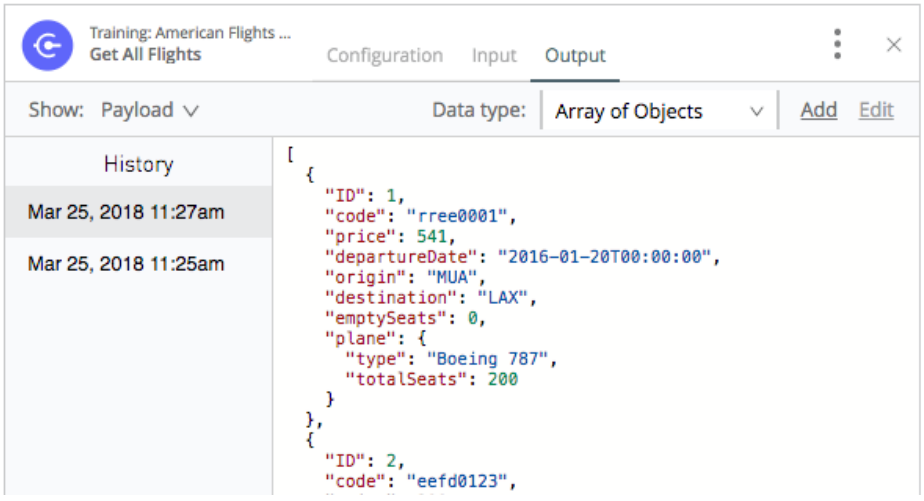
23. Return to Advanced REST Client and click Send; you should see flight data.



24. Click Send again to make a second request.

Review Mule event data

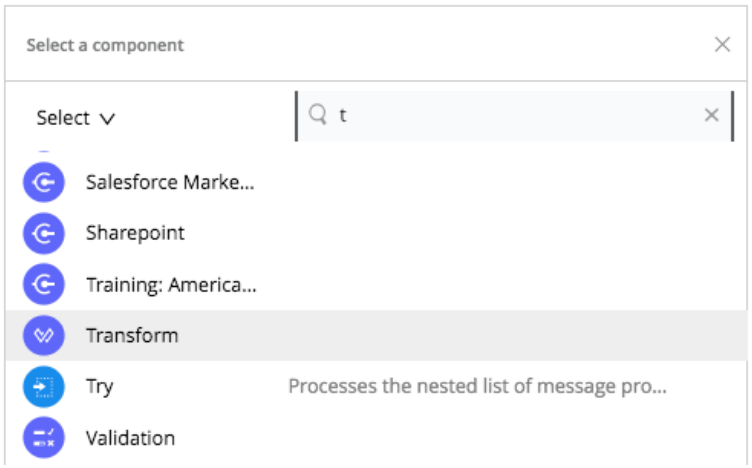
25. Return to flow designer and open the American Flights API card.
26. Select the Input tab and examine the Mule event data.
27. Select the Output tab; you should see payload data.



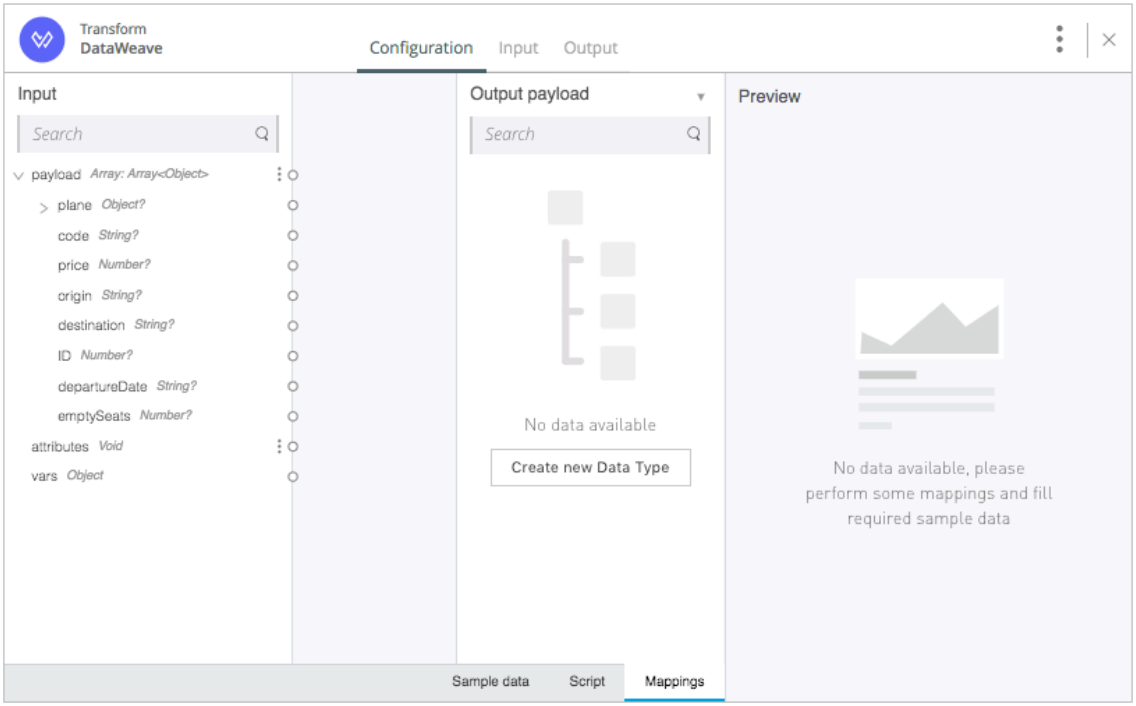
28. Close the card.

Add and configure a component to transform the data

29. Click the add button in the flow.
30. In the Select a component dialog box, select Transform.



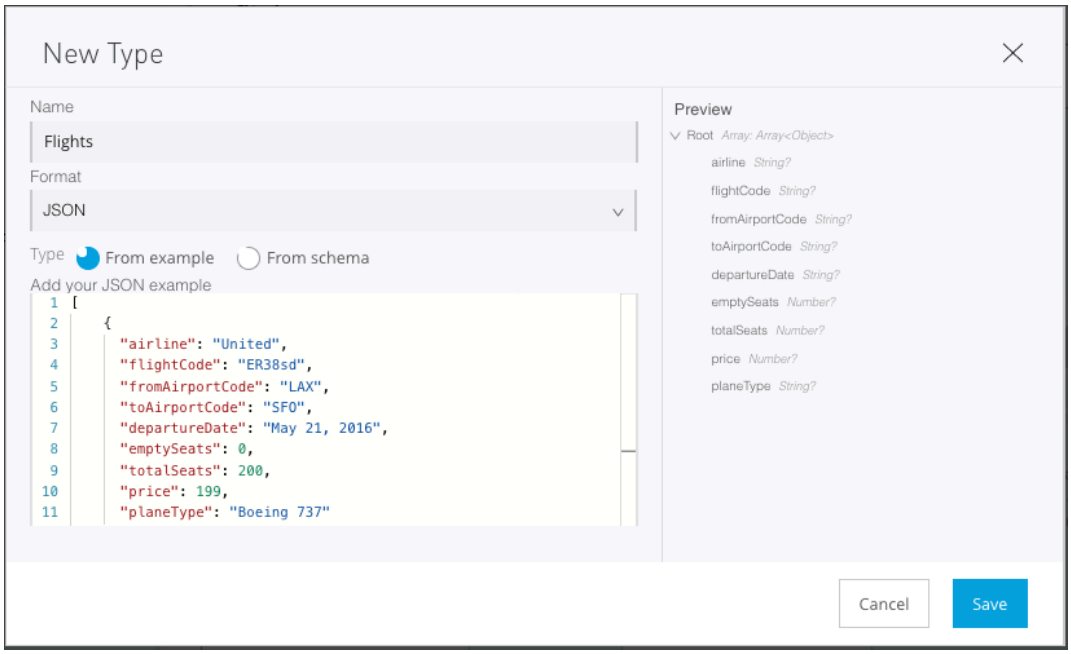
31. In the Transform card, look at the Mule event structure in the input section.
32. In the output section, click the Create new Data Type button.



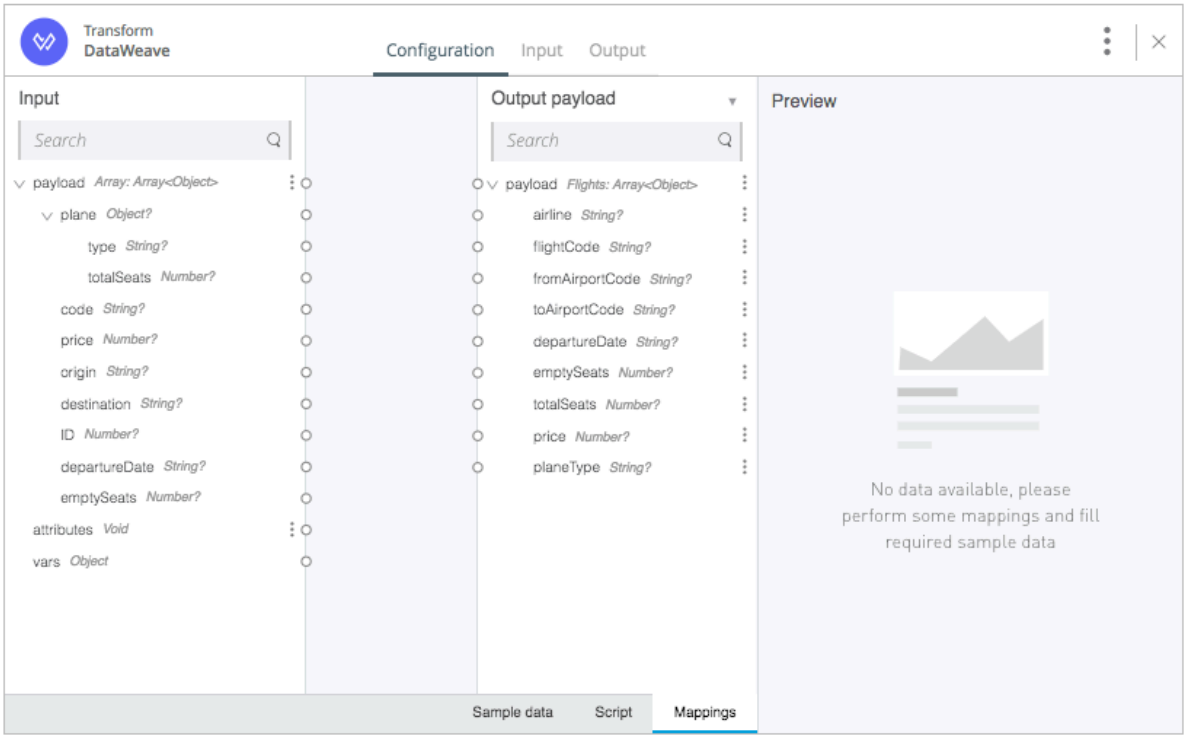
33. In the New Type dialog box, set the following values:

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

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



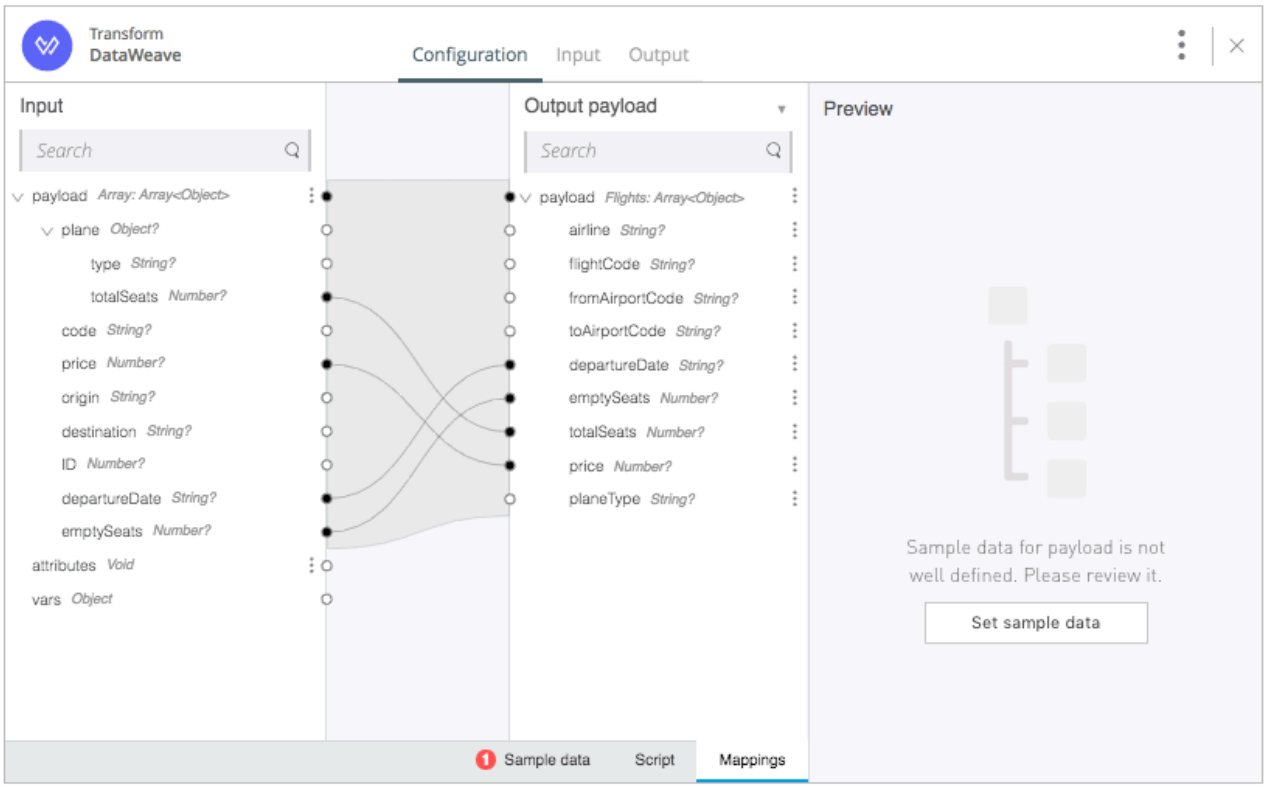
37. Click Save.
38. In the input section, expand the plane object.



Create the transformation

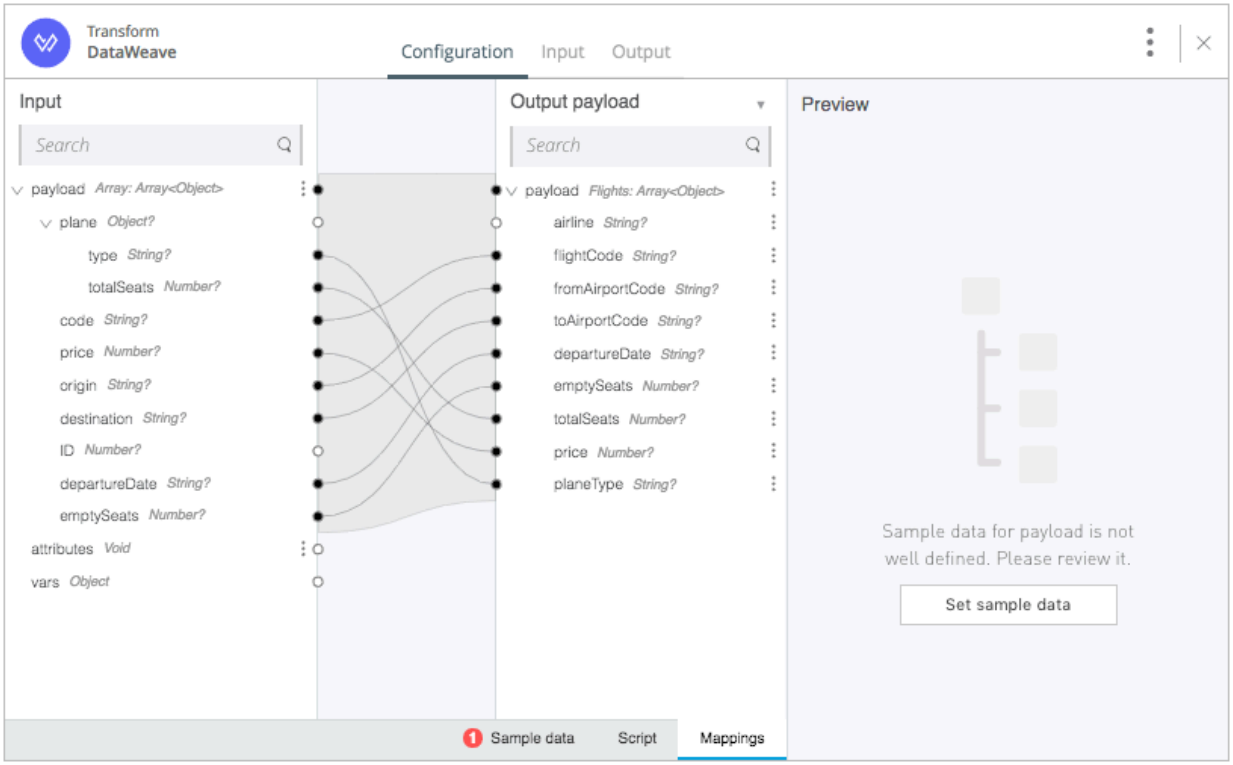
39. Map fields with the same names by dragging them from the input section and dropping them on the corresponding field in the output section.

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

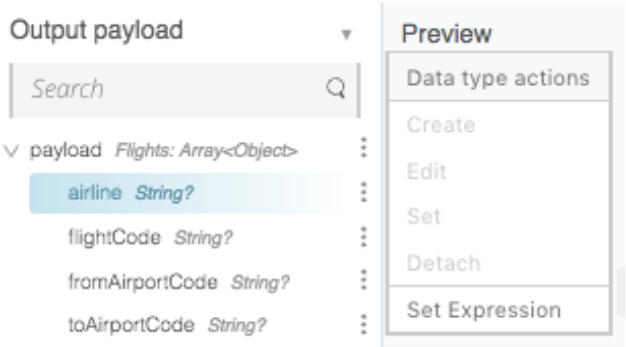


40. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

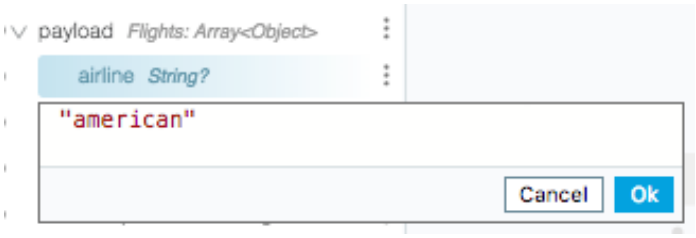
    - plane > type to planeType
    - code to flightCode
    - origin to fromAirport
    - destination to toAirport



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

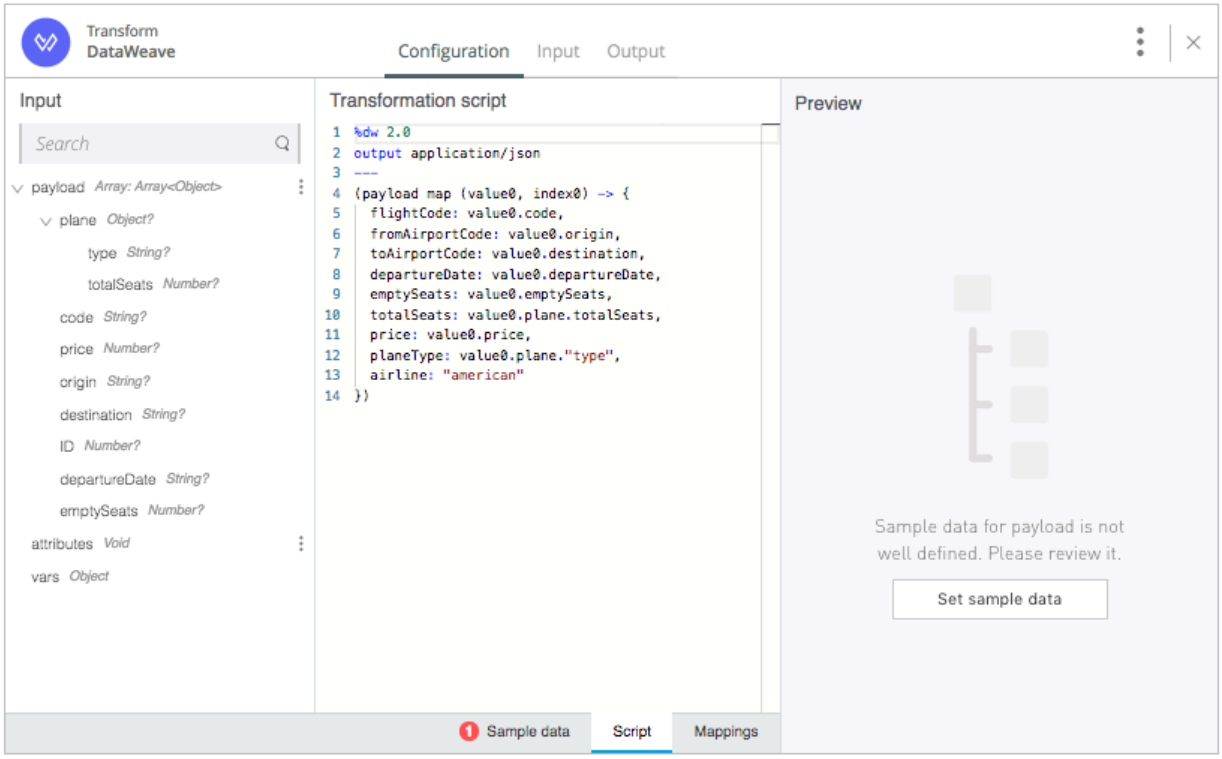


42. Change the value from null to "american" and click OK.



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

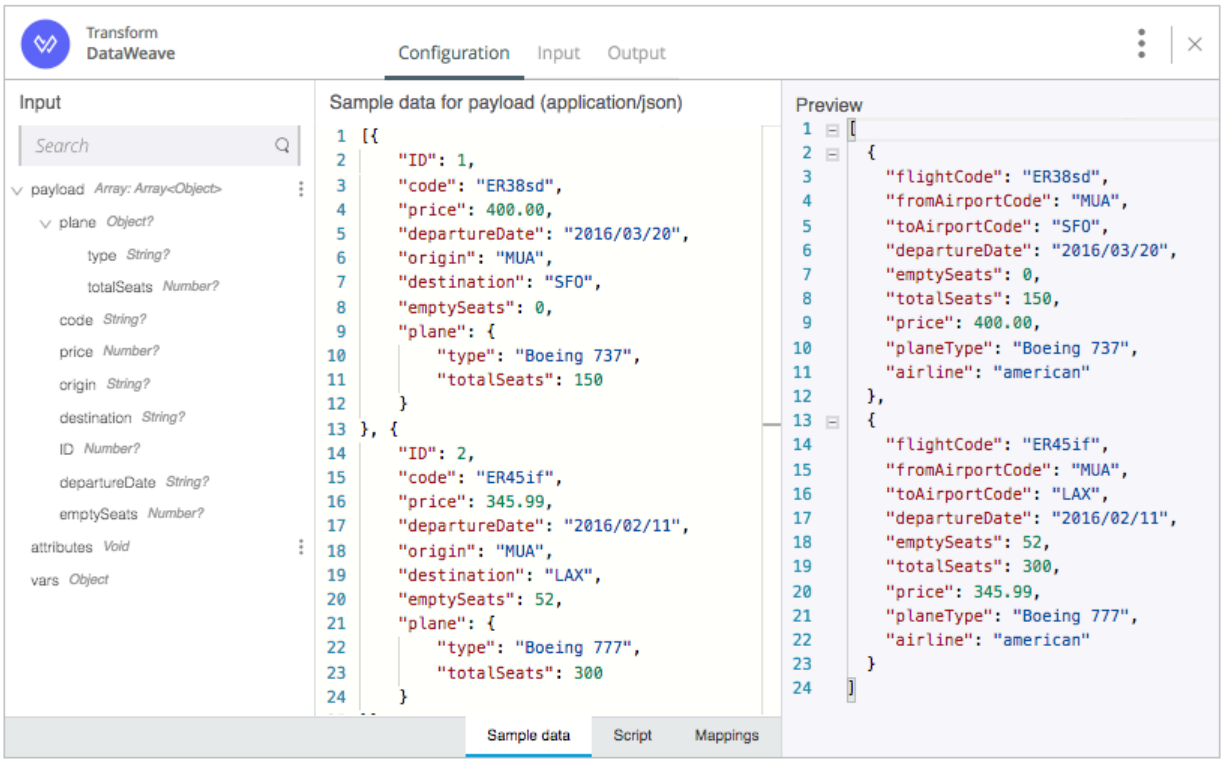
    Note: You learn to write DataWeave transformations later in the Development Fundamentals course.



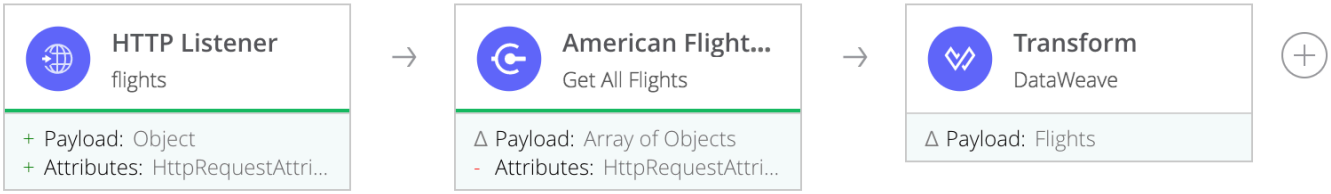
Add sample data

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

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

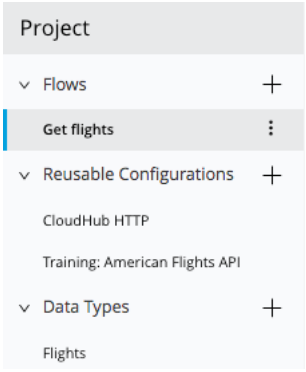


48. Look at the preview section, you should see a sample response for the transformation.
49. Close the card.



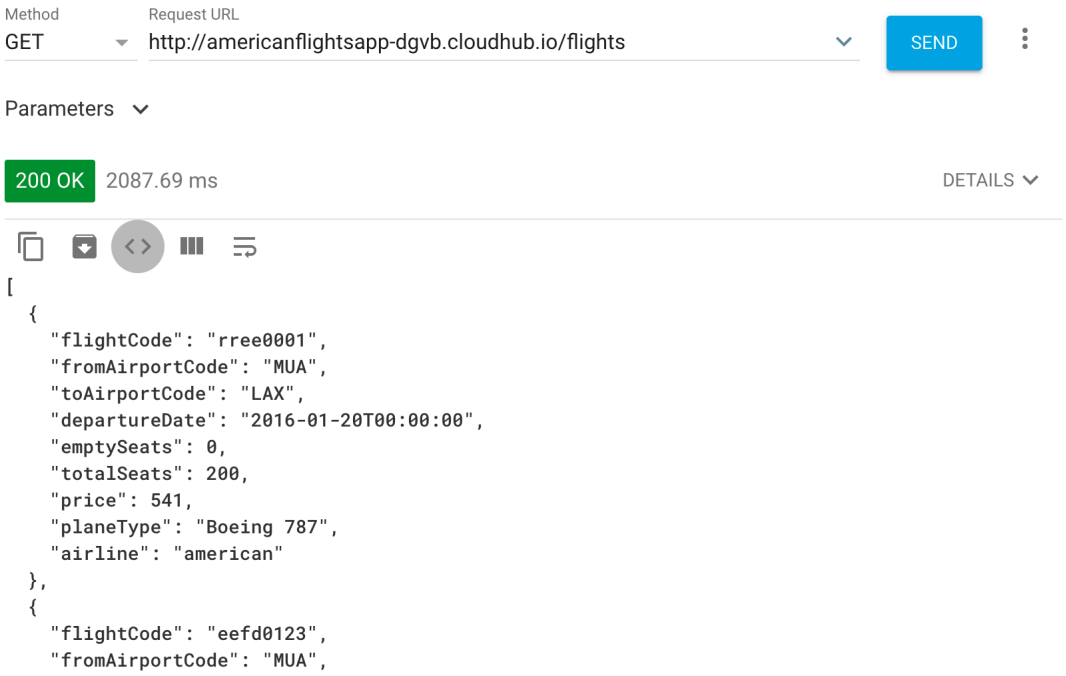
Locate the data type and configuration definitions

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



Test the application

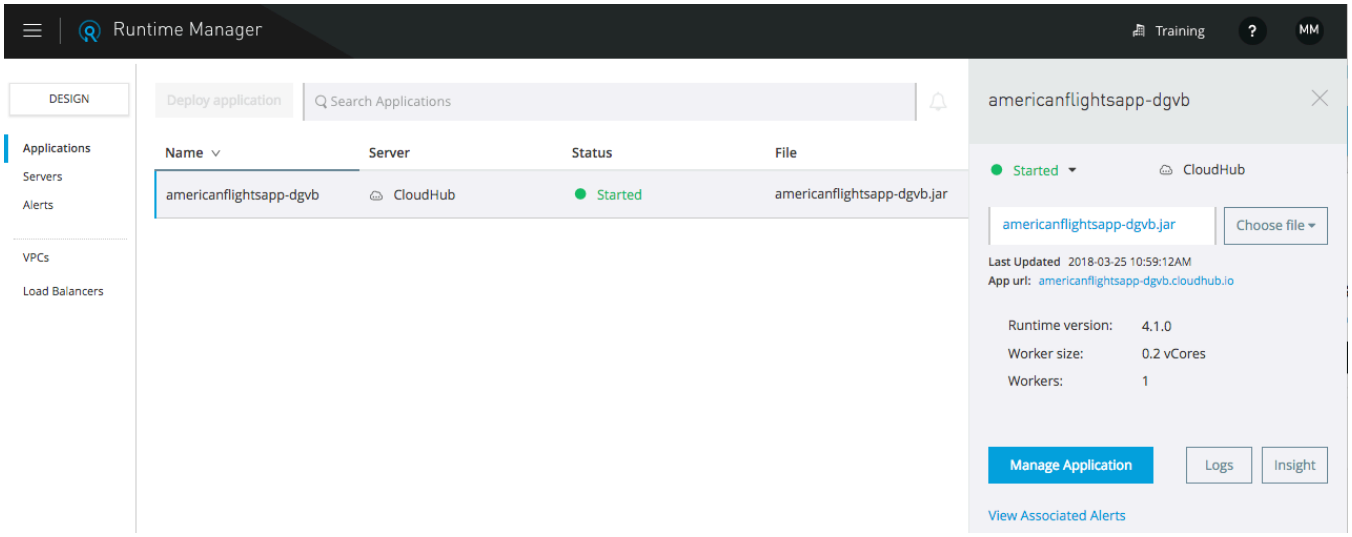
51. Run the project.
52. Return to Advanced REST Client and click Send to make another request to http://americanflightsapp-xxxx.cloudhub.io/flights; you should see all the flight data as JSON again but now with a different structure.



Stop the application

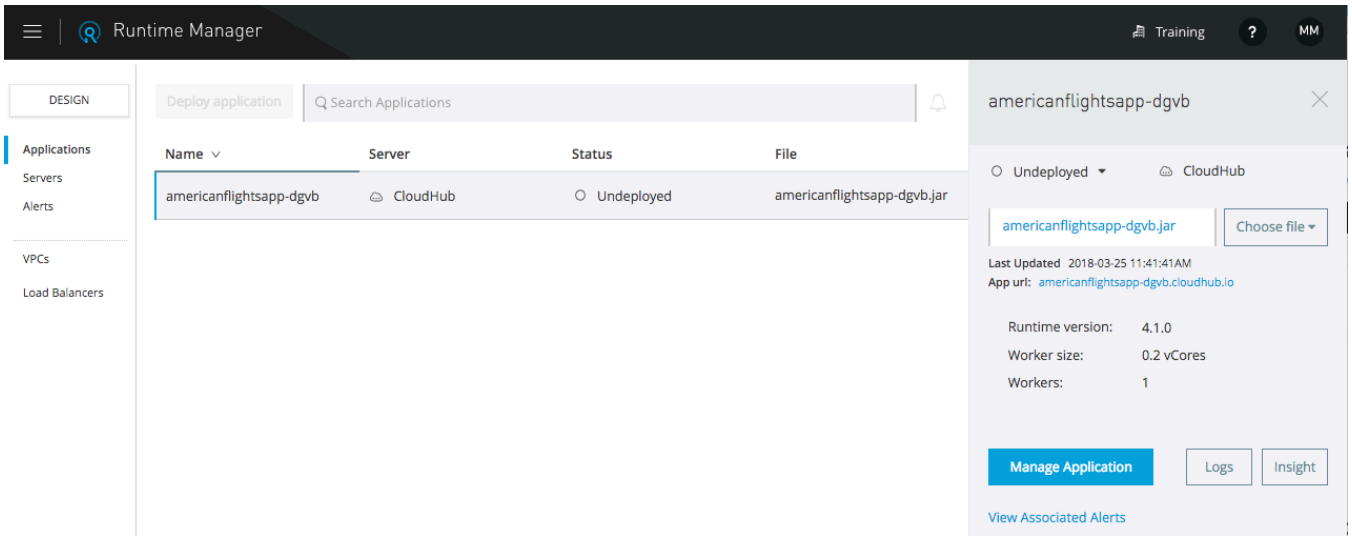
53. Return to Runtime Manager.
54. In the left-side navigation, click Applications.

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

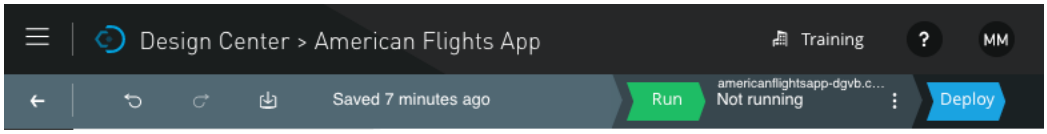


56. Click the drop-down menu button next to Started and select Stop; the status should change to Undeployed.

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

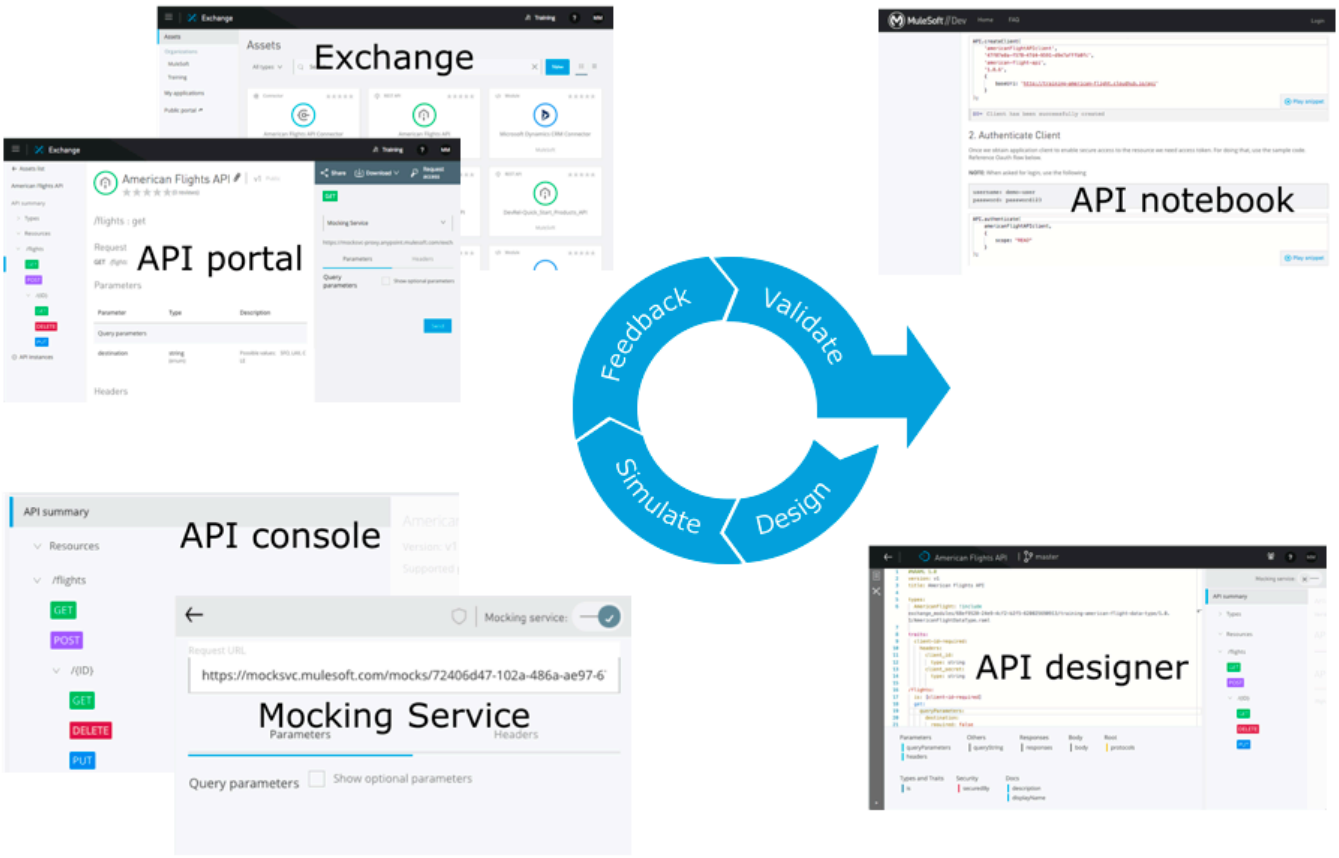


57. Close Runtime Manager.
58. Return to flow designer; you should see the application is not running.



59. Return to Design Center.

Module 3: Designing APIs



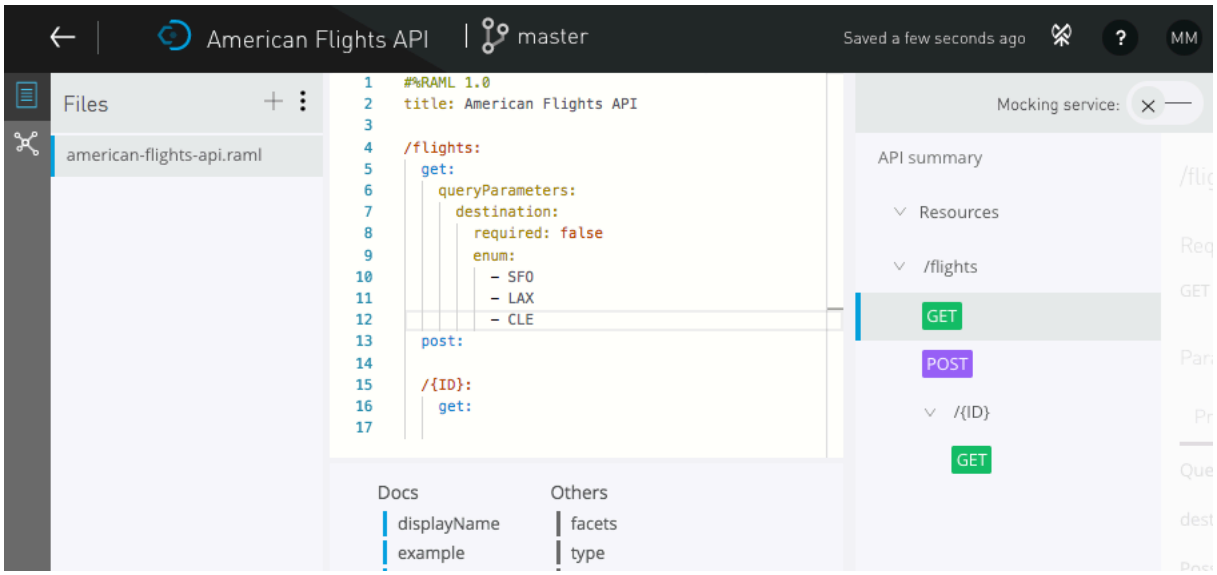
At the end of this module, you should be able to:

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

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

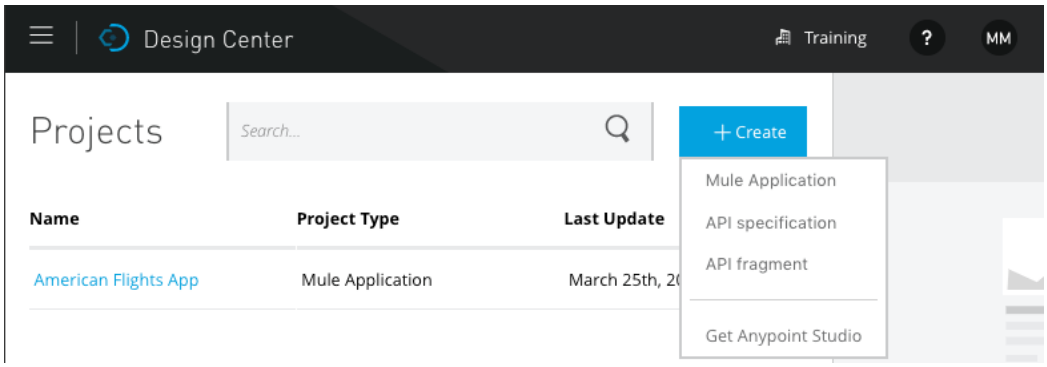
In this walkthrough, you create an API definition with RAML using API designer. You will:

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



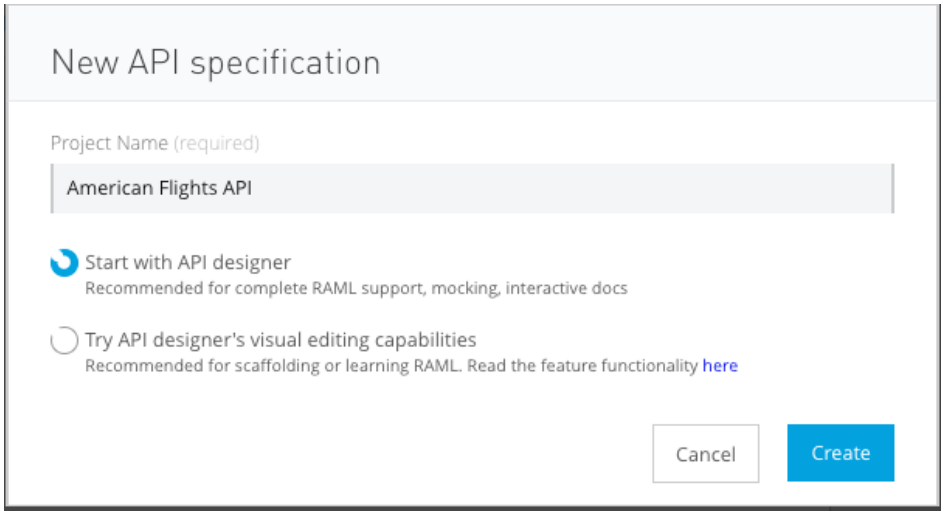
Create a new Design Center project

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

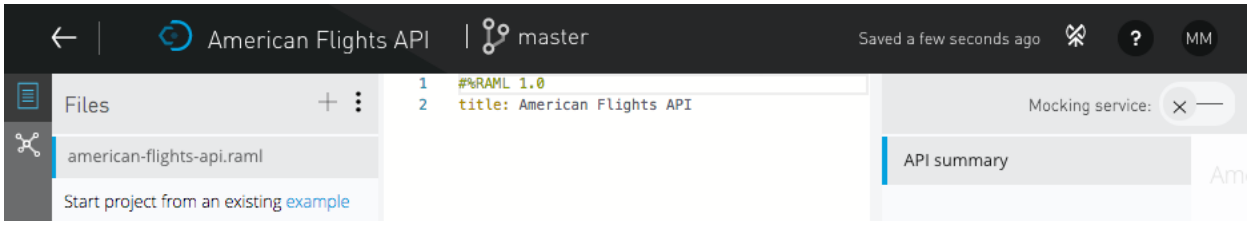


3. In the New API specification dialog box, set the project name to American Flights API.

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



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



Add a RAML resource

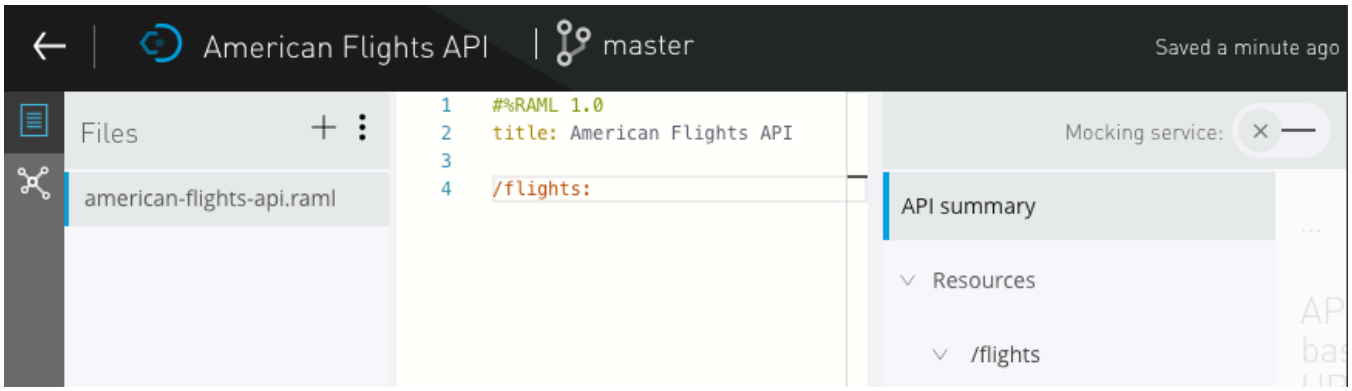
7. In the editor, place the cursor on a new line of code at the end of the file.
8. Add a resource called flights.

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

View the API console

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

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

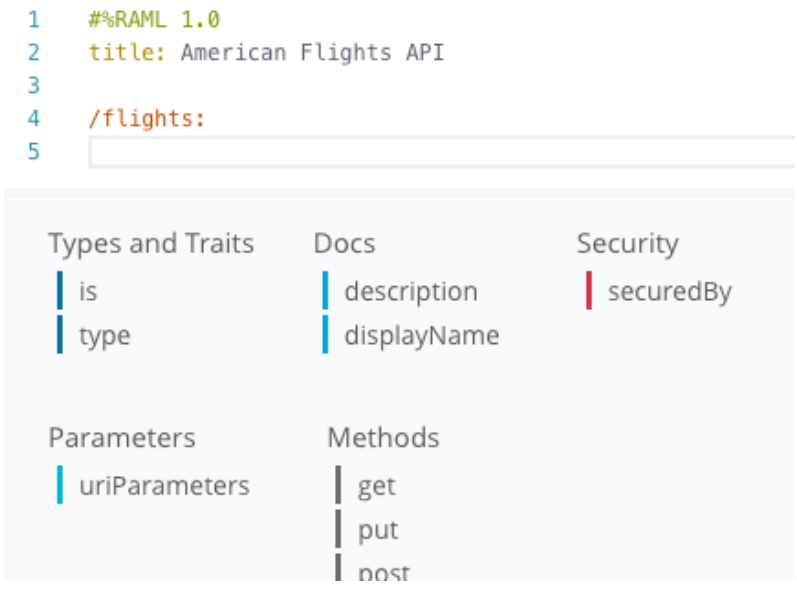


Add RAML methods

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

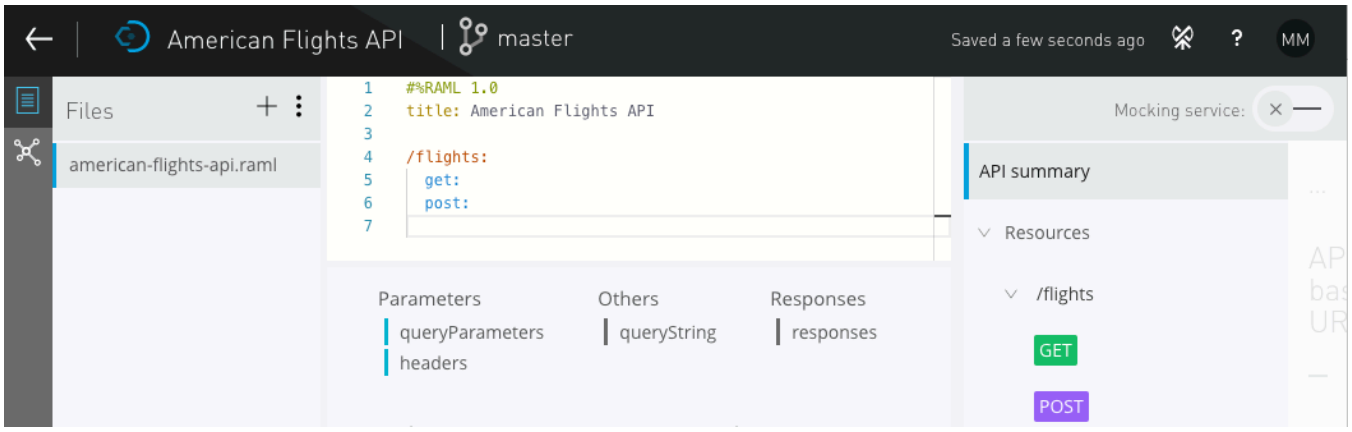
    Note: If you don't see the API designer shelf, it is either minimized or there is an error in your code. To check if it is minimized, go to the bottom of the web browser window and look for an arrow. If you see the arrow, click it to display the shelf.

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



12. Click the get method in the shelf.

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

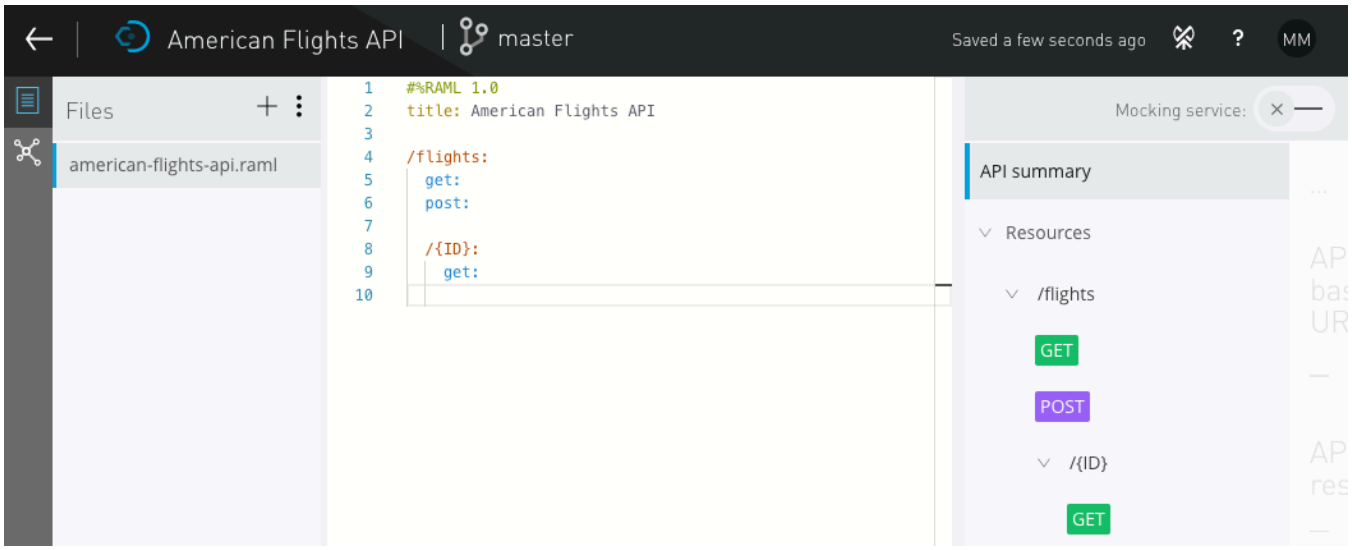


Add a nested RAML resource

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

```
/{ID}:
```

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



Add an optional query parameter

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

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

/flights:
  get:
    queryParameters:
      destination:
  post:

  /{ID}:
```

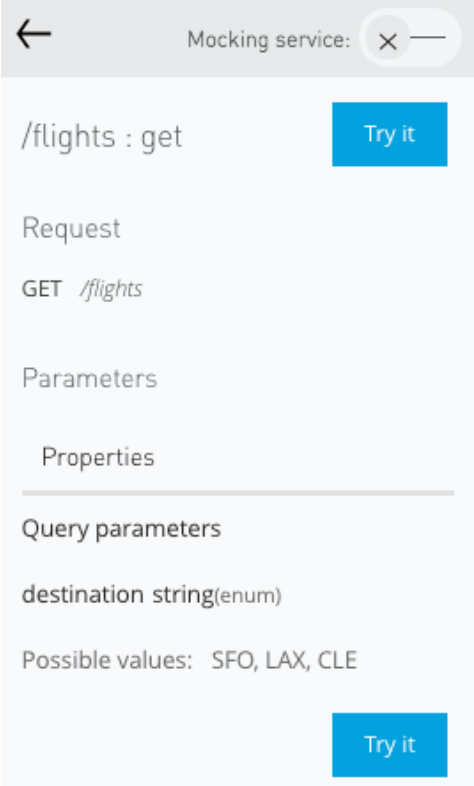
25. Indent under the destination query parameter and look at the possible parameters in the shelf.
26. In the shelf, click the required parameter.
27. In the shelf, click false.
28. Go to a new line of code; you should be at the same indent level as required.
29. In the shelf, click the enum parameter.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

```
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
```

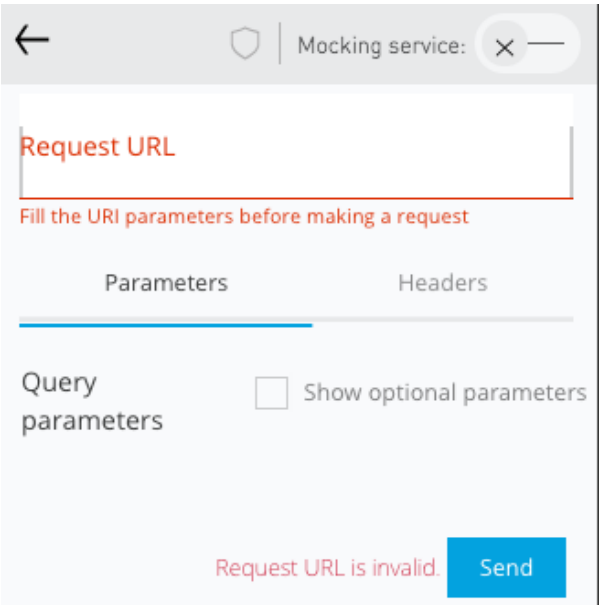
Try to call an API method using the API console

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

32. Review the information.



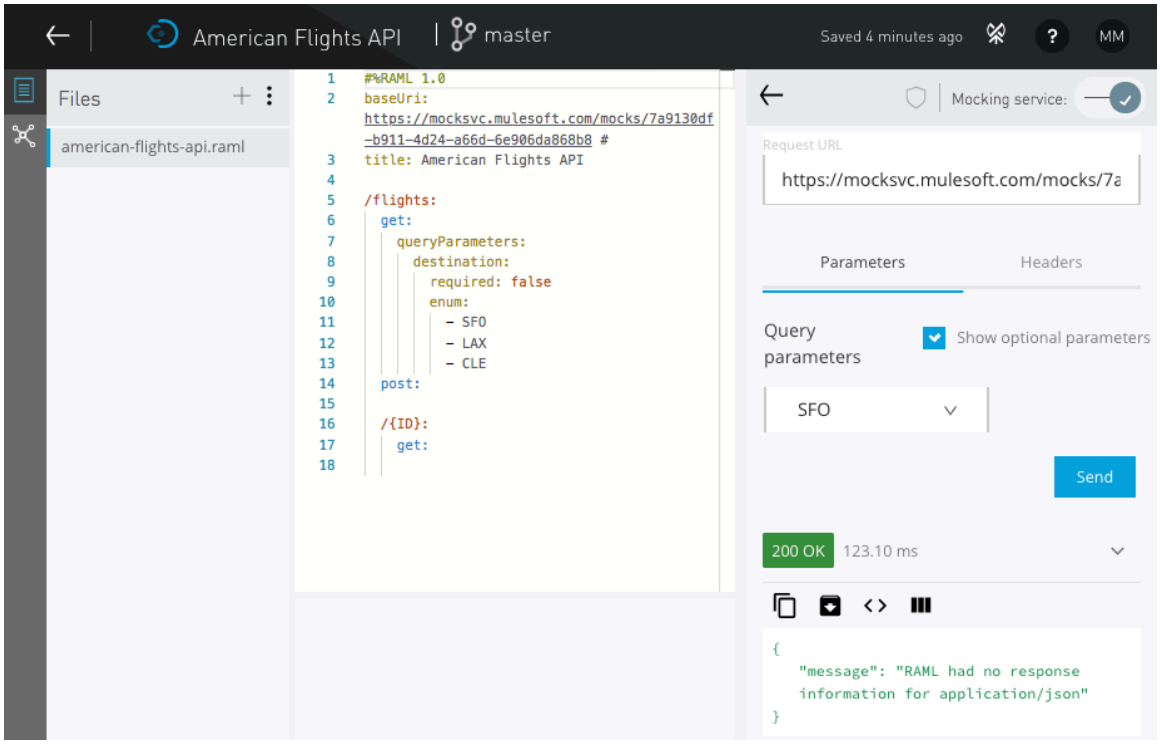
33. Click the Try it button; you should get a message that the Request URL is invalid; a URL to call to try the API needs to be added to the RAML definition.



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

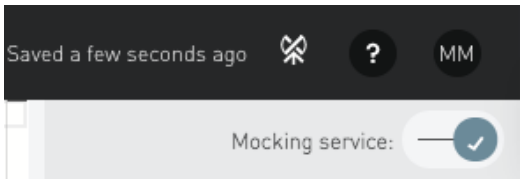
In this walkthrough, you test the API using the Anypoint Platform mocking service. You will:

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



Turn on the mocking service

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

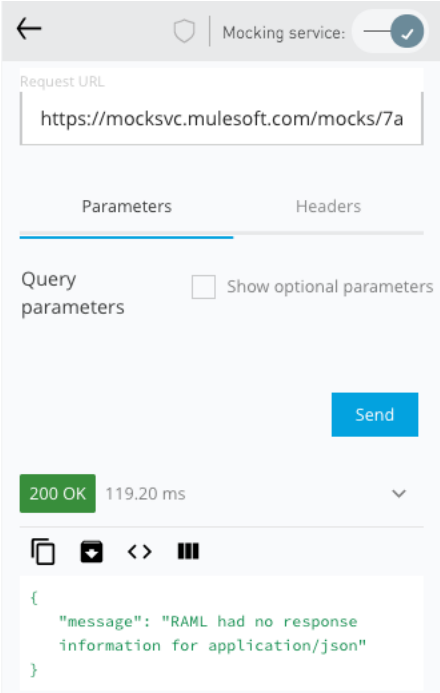


4. Look at the baseUri added to the RAML definition in the editor.

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

Test the /flights:get resource

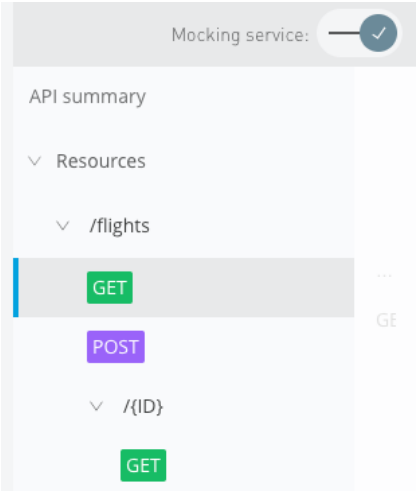
5. In API console, click the Send button for the flights GET method; you should get a 200 status code, a content-type of application/json, and a general RAML message placeholder.



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

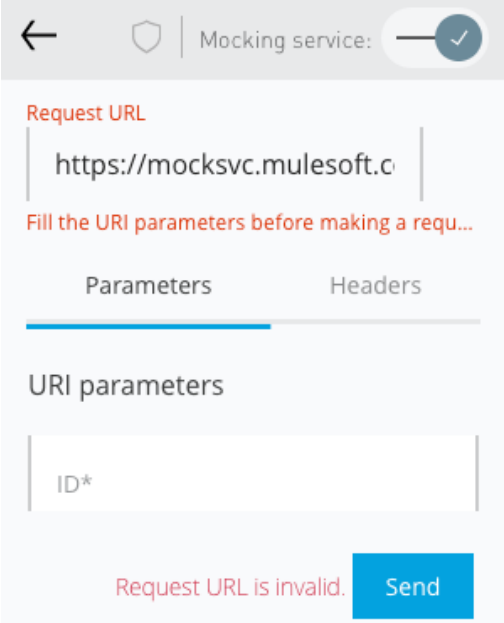
Test the /flights/{ID} resource

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

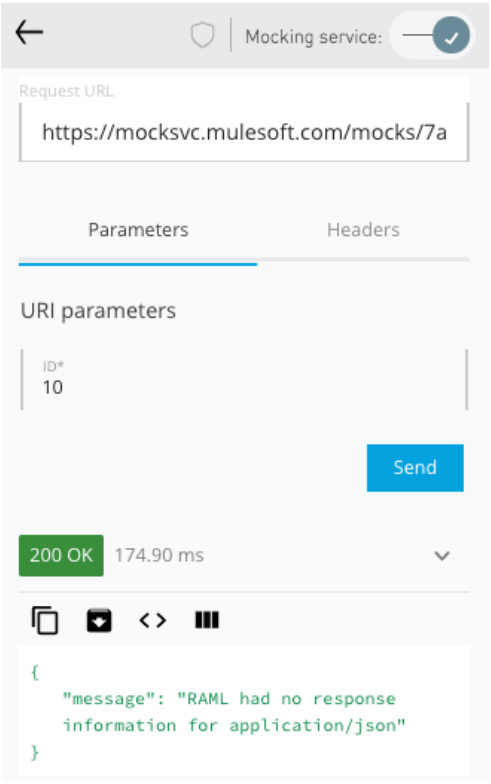


9. Click the GET method for the /{ID} nested resource.

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

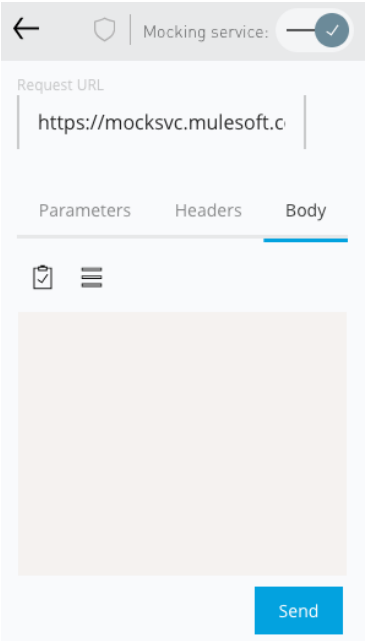


11. In the ID text box, enter a value of 10.
12. Click the Send button.
13. Look at the response; you should get the same default response with a 200 status code, a content-type of application/json, and the general RAML message placeholder.

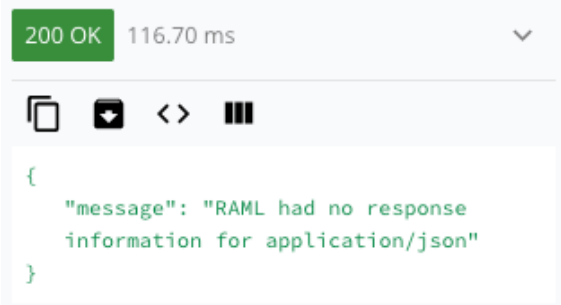


Test the /flights:post resource

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



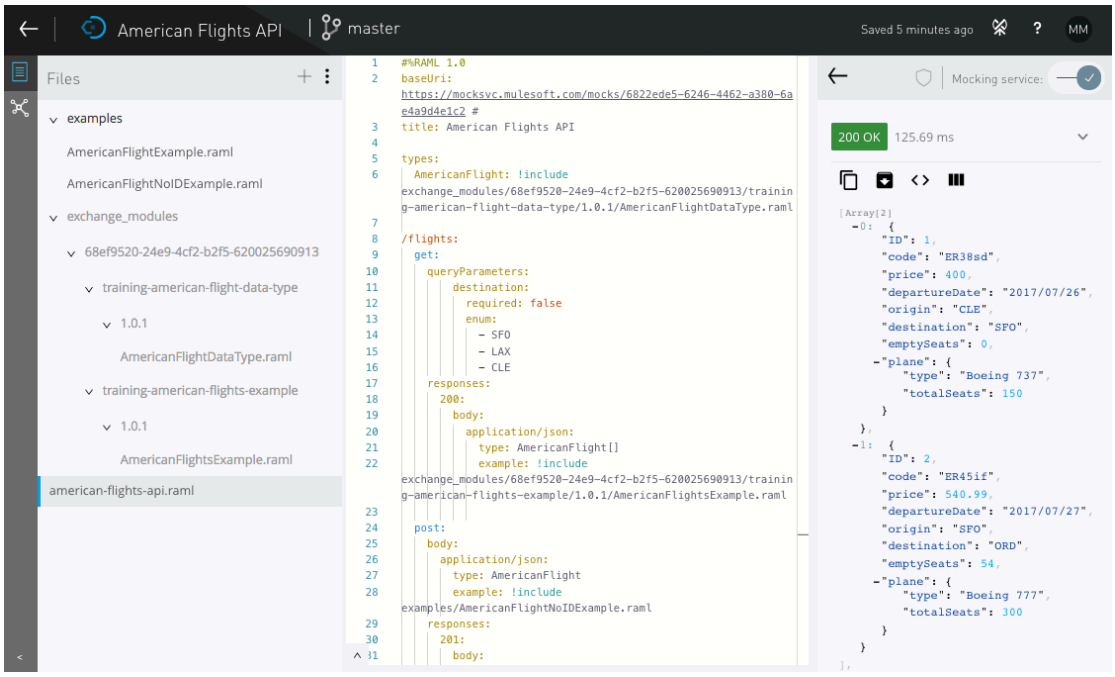
18. Click the Send button.
19. Look at the response; you should get the same generic 200 status code response.



Walkthrough 3-3: Add request and response details

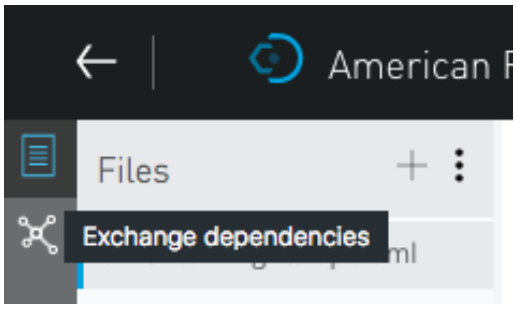
In this walkthrough, you add information about each of the methods to the API specification. You will:

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

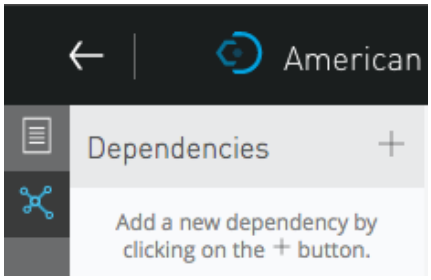


Add data type and example fragments from Exchange

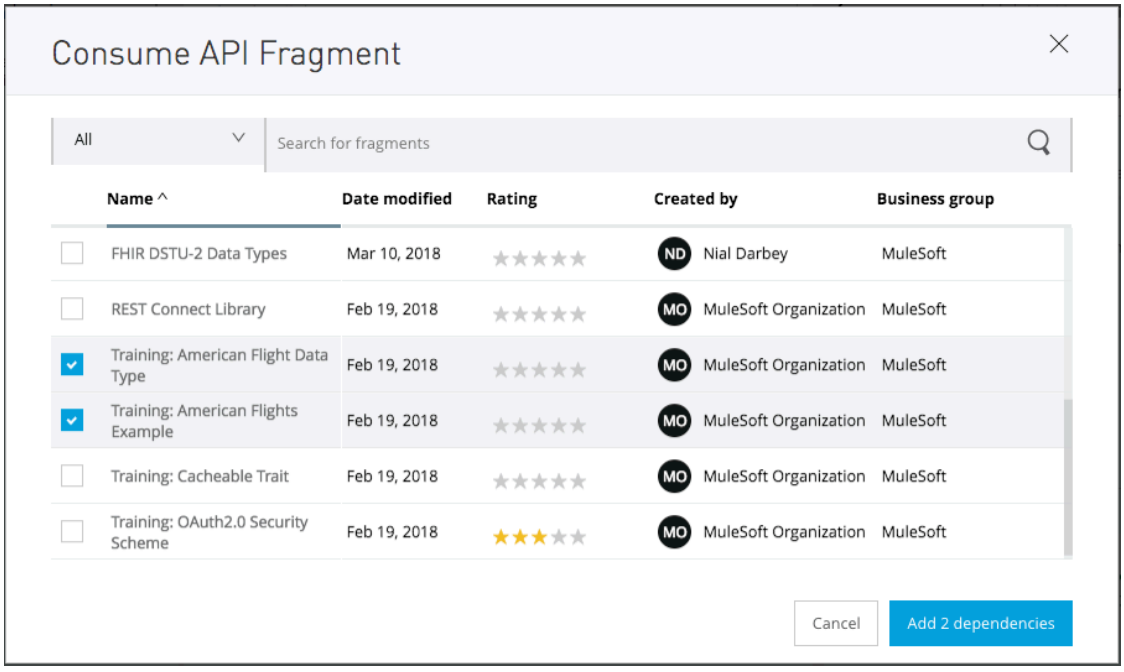
1. Return to API designer.
2. In the file browser, click the Exchange dependencies button.



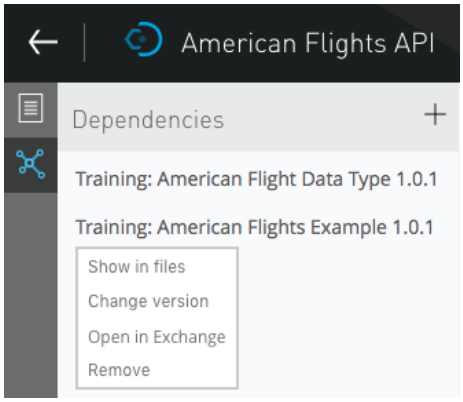
3. Click the Add button.



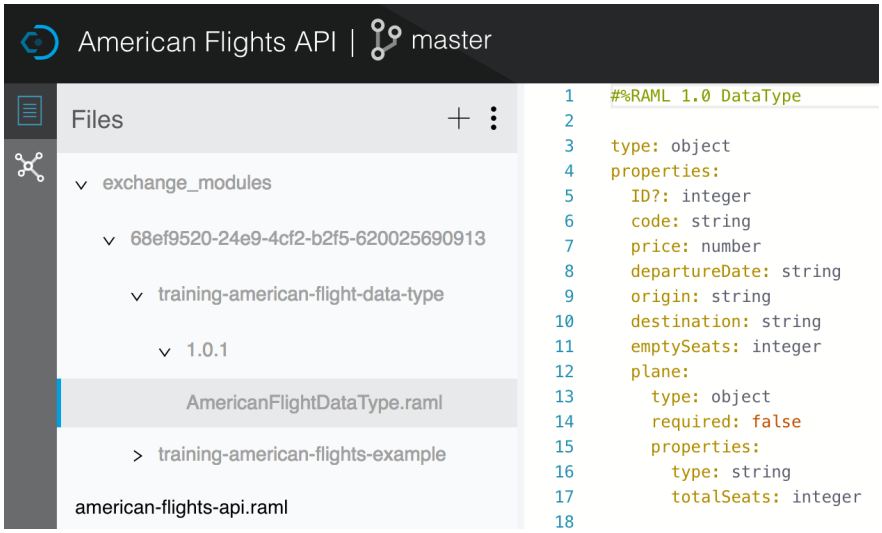
4. In the Consume API Fragment dialog box, select the Training: American Flight Data Type and the Training: American Flights Example.



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



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



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

Define an AmericanFlight data type for the API

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

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

```
#%RAML 1.0
baseUri: https://mocksvc.mulesoft.com/mocks/3220238a-17c6-4e31-b5fa-413e8129e12b #
title: American Flights API

types:
  AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/tr

/flights:
  get:
```

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

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

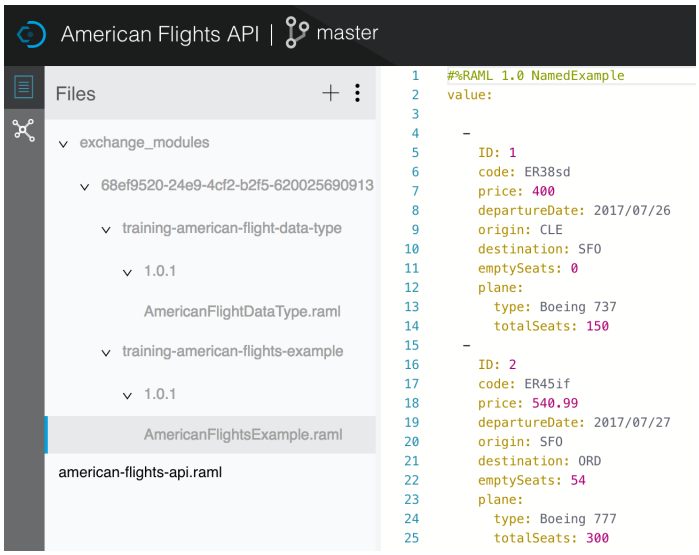
```
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
```

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

```
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight[]
```

Add an example response for the /flights:get method

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



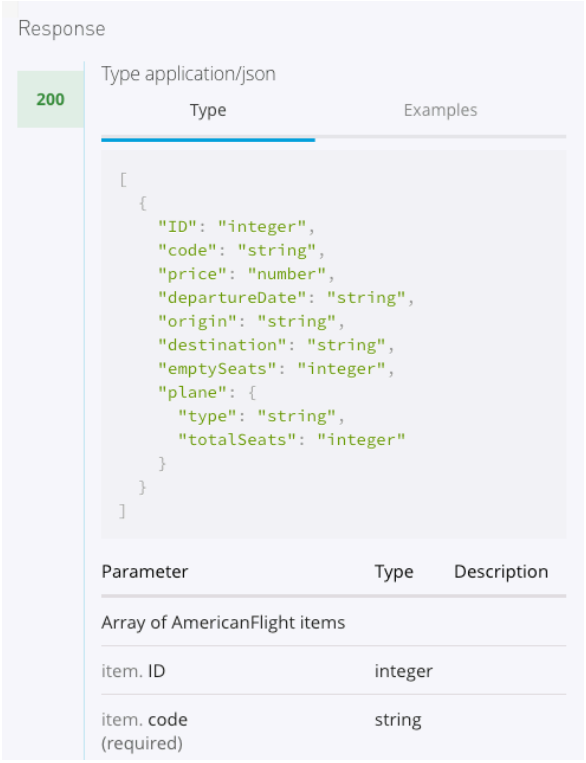
19. In the file browser, click the options menu next to AmericanFlightsExample.raml and select Copy path to clipboard.
20. Return to american-flights-api.raml.
21. In the editor, go to a new line after the type declaration in the /flights:get 200 response (at the same indentation as type).
22. In the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

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

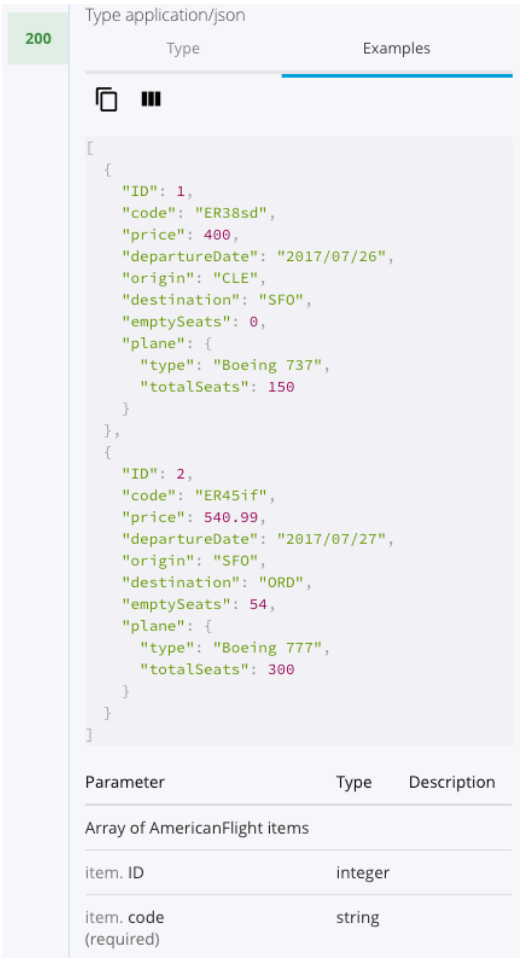
```
17      responses:
18        200:
19          body:
20            application/json:
21              type: AmericanFlight[]
22              example: !include exchange_modules/86f74f12-decb-452c
23    post:
```

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

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



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



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

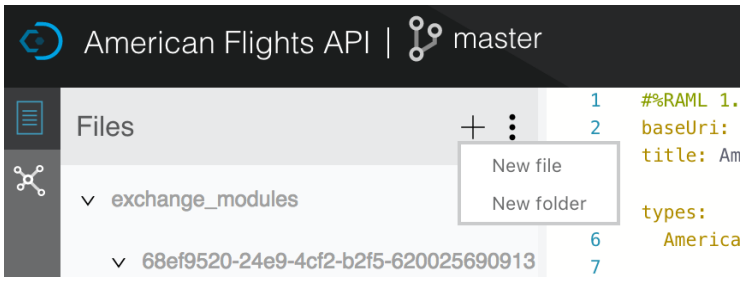
Specify the /{ID}:get method to return an AmericanFlight object

28. In the editor, indent under the /{ID} resource get method.
29. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

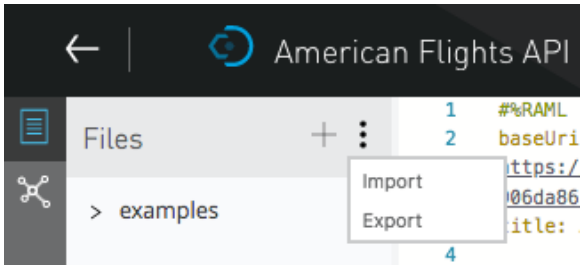
```
/{ID}:
  get:
    responses:
      200:
        body:
          application/json:
            type: AmericanFlight
```

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

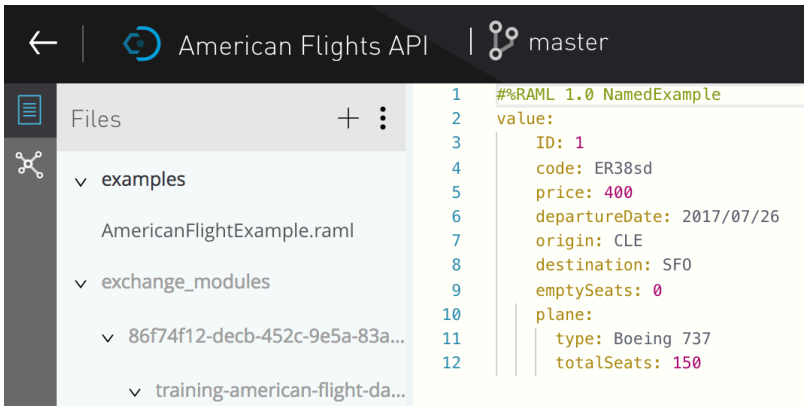
30. In the file browser, click the add button and select New folder.



31. In the Add new folder dialog box, set the name to examples and click Create.
32. In the file browser, click the menu button and select Import.



33. In the Import dialog box, leave File or ZIP selected and click the Choose file button.
34. Navigate to your student files and select the AmericanFlightExample.raml file in the examples folder.
35. In the Import dialog box, click Import; you should see the new file in API designer.
36. Review the code.
37. In the file browser, drag AmericanFlightExample.raml into the examples folder.



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

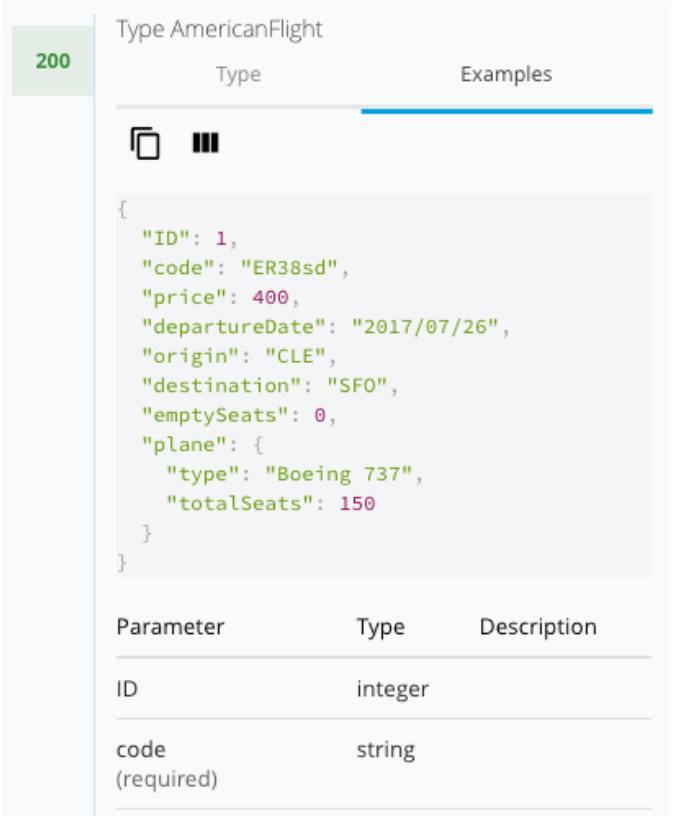
38. Return to american-flights-api.raml.
39. In the editor, go to a new line after the type declaration in {ID}:/get (at the same indentation).

40. In the shelf, click example.
41. Add an include statement and include the example in examples/AmericanFlightExample.raml.

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

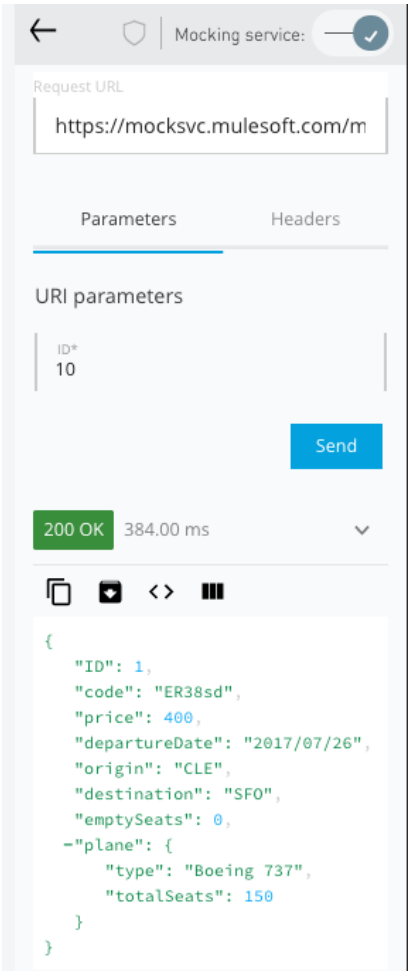
Review and test the /{ID}:get resource in API console

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



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

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



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

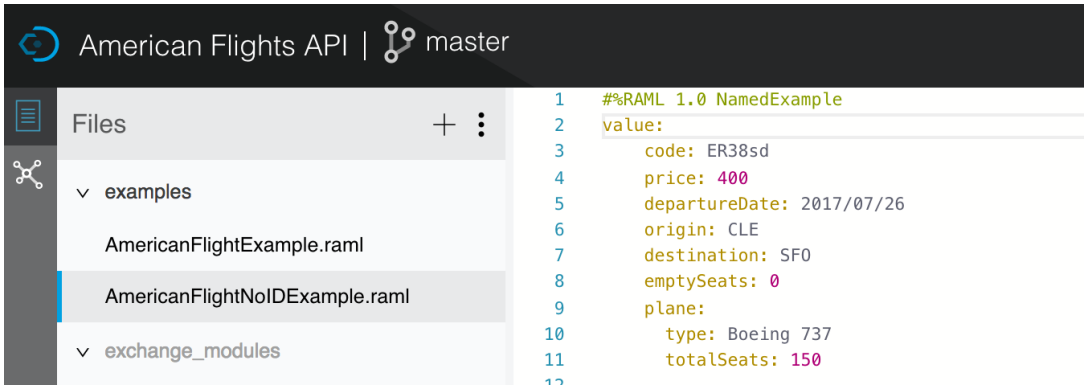
45. In the editor, indent under the /flights post method.
46. In the shelf, click body > application/json > type > AmericanFlight.

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

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

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

49. In the Add new file dialog box, set the following values:
    - Version: RAML 1.0
    - Type: Example
    - File name: AmericanFlightNoIDExample.raml
50. Click Create.
51. Delete any code in the new file and then paste the code you copied.
52. Delete the line of code containing the ID.



53. Return to american-flights-api.raml.
54. In the post method, go to a new line under type and add an example element.
55. Use an include statement to set the example to examples/AmericanFlightNoIDExample.raml.

```
post:
  body:
    application/json:
      type: AmericanFlight
      example: !include examples/AmericanFlightNoIDExample.raml
```

Specify an example response for the /flights:post method

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

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

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

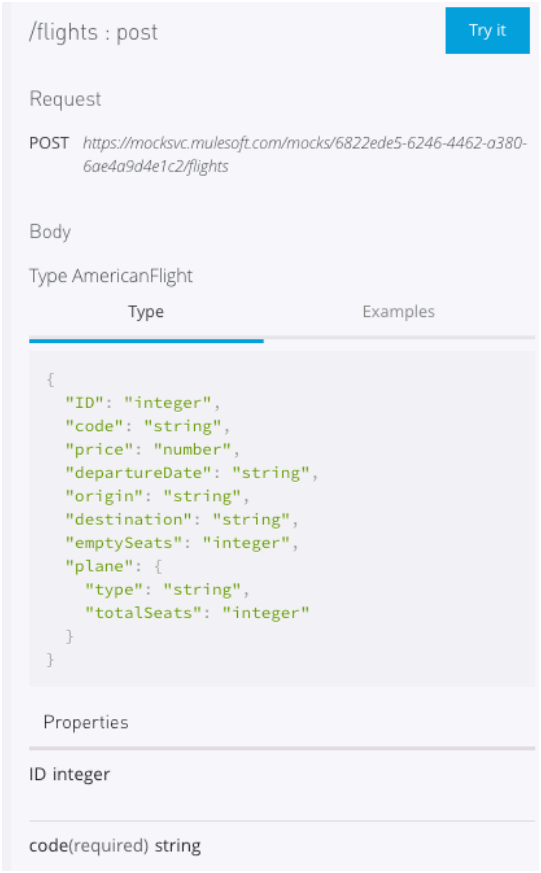
58. In the shelf, click example.
59. Indent under example and add a message property equal to the string: Flight added (but not really).

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

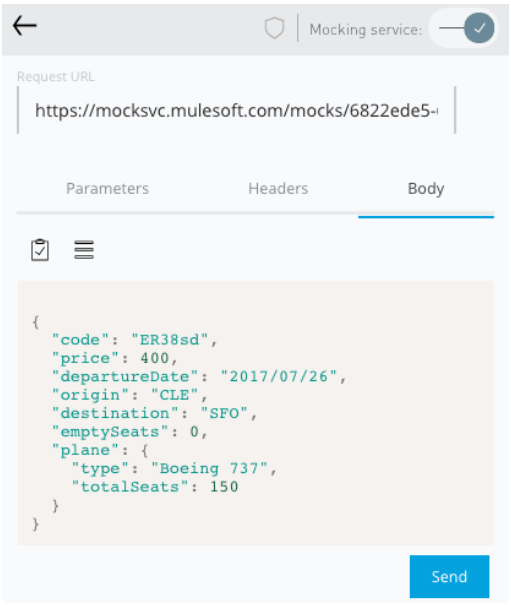
Review and test the /flights:post resource in API console

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

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

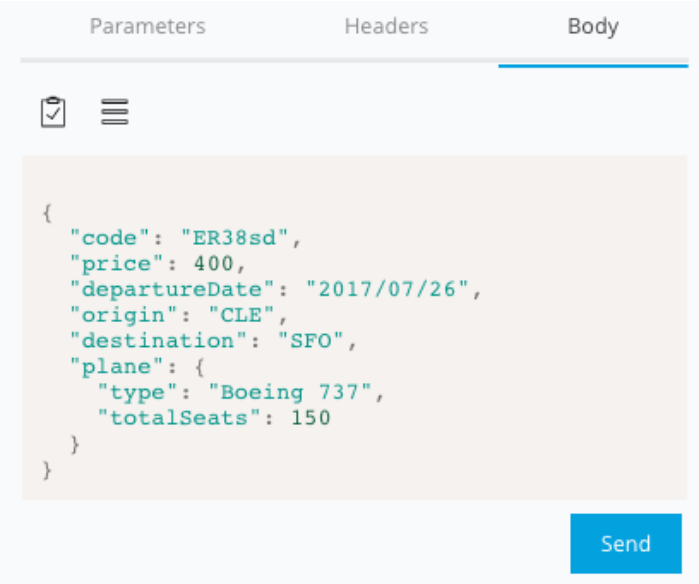


62. Click the Try it button and select the Body tab again; you should now see the example request body.

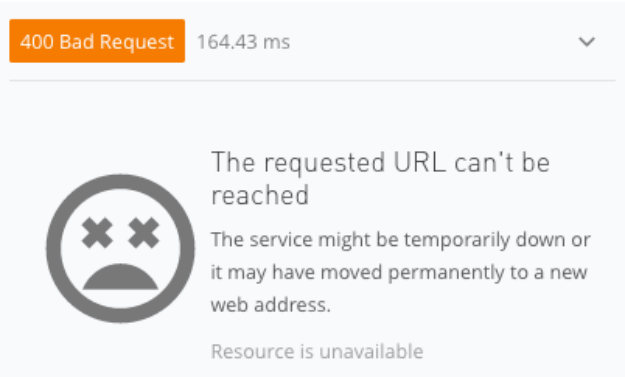


63. Click the Send button; you should now get a 201 response with the example message.

64. In the body, remove the emptySeats properties.



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

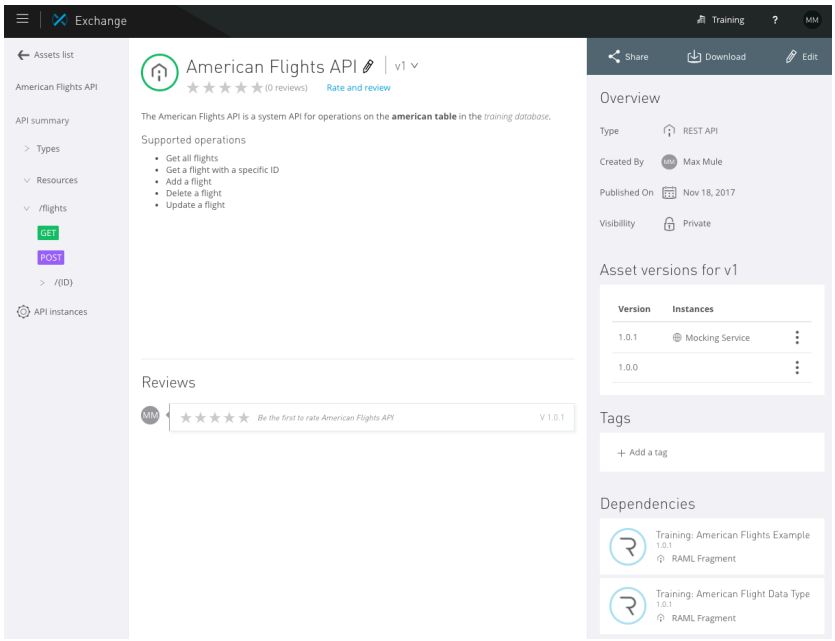


66. Add the emptySeats property back to the body.

Walkthrough 3-4: Add an API to Anypoint Exchange

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

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



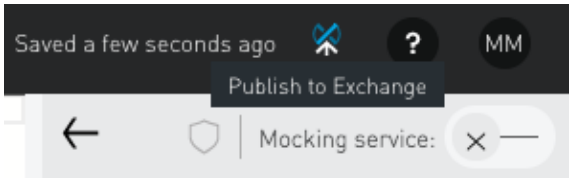
Remove the baseUri by turning off the mocking service

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

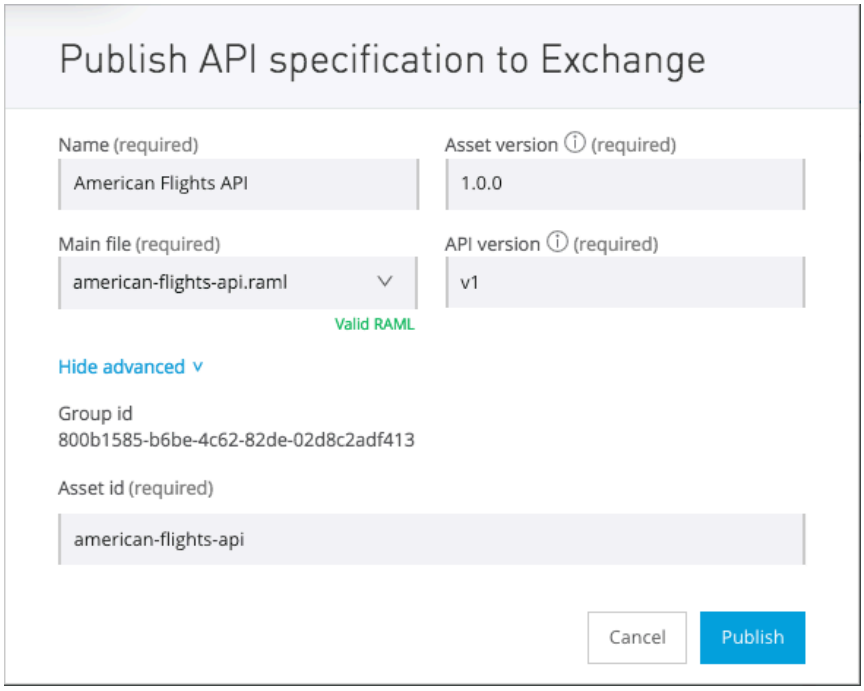
```
1    #%RAML 1.0
2    title: American Flights API
3
```

Publish the API to Anypoint Exchange from API designer

3. Click the Publish to Exchange button.



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



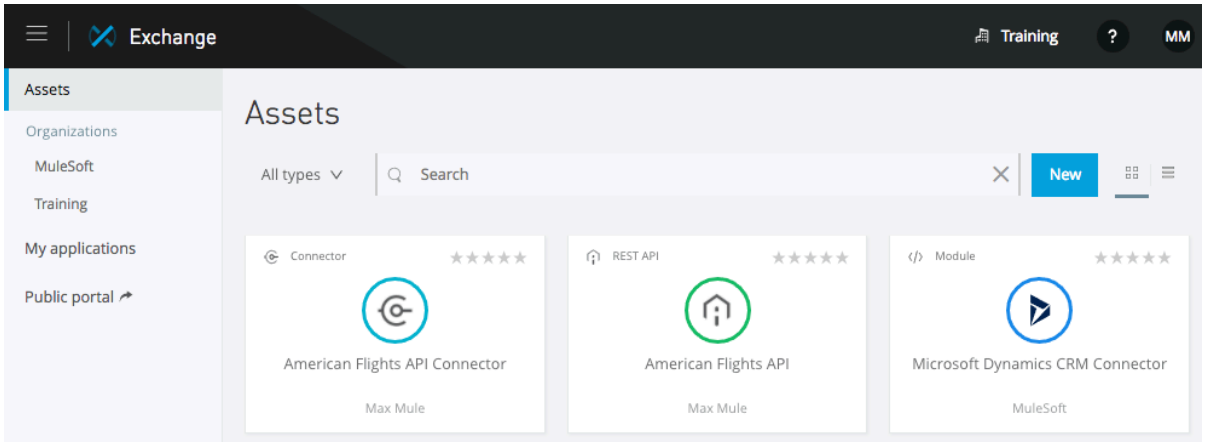
8. Click the Publish button.
9. In the Publish API specification to Exchange dialog box, click Done.
10. Look at the RAML file; you should see a version has been added.

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

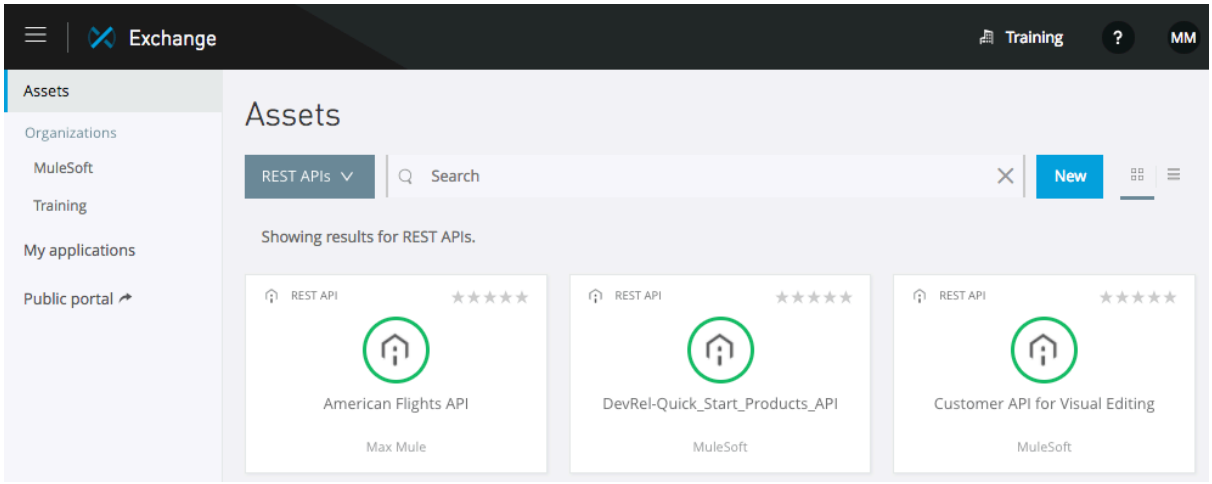
Locate your API in Anypoint Exchange

11. Return to Design Center.

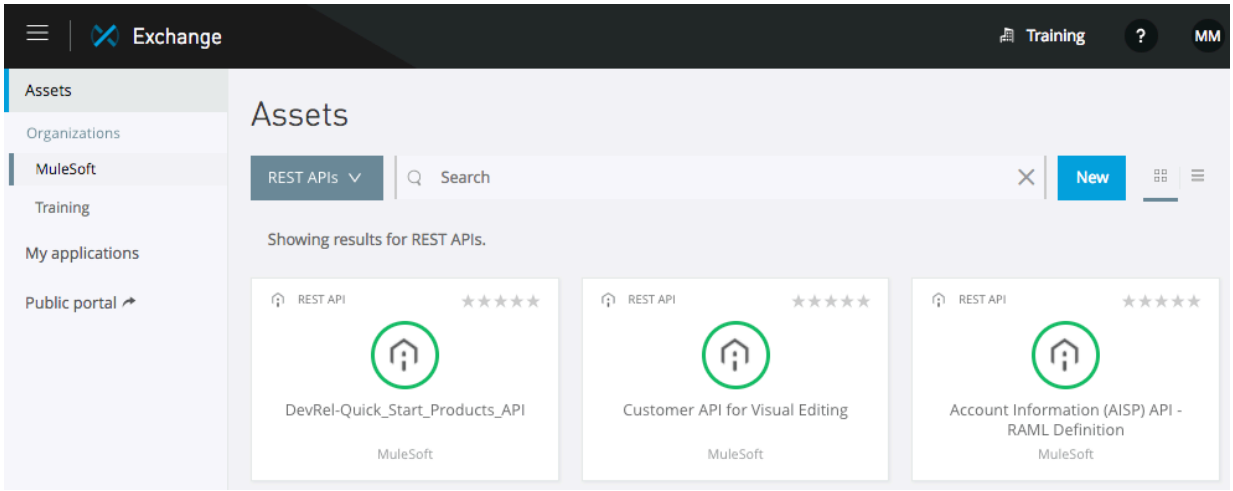
12. In the main menu, select Exchange; you should see your American Flights API.



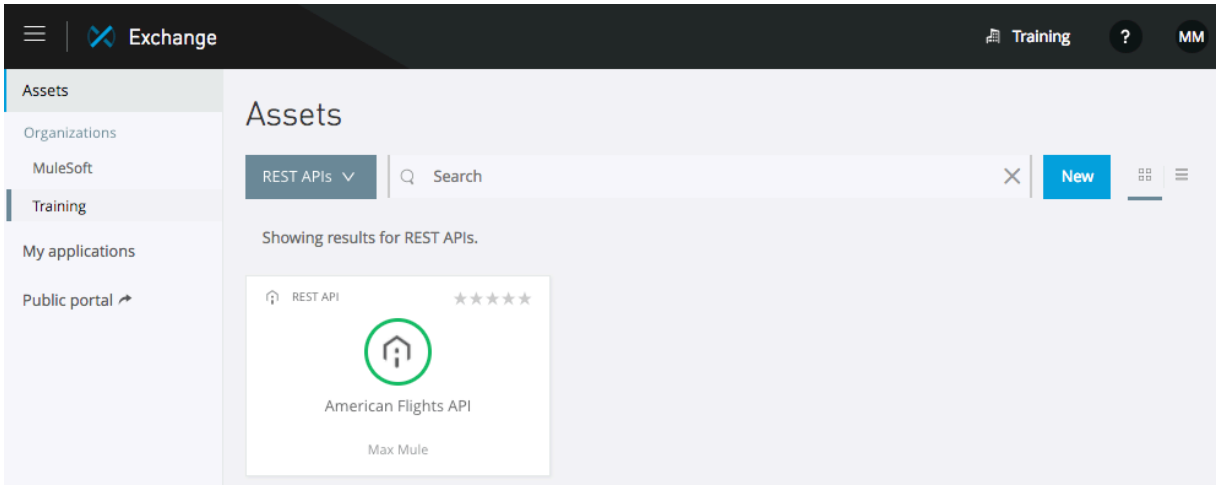
13. In the types drop-down menu, select REST APIs; you should still see your API.



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

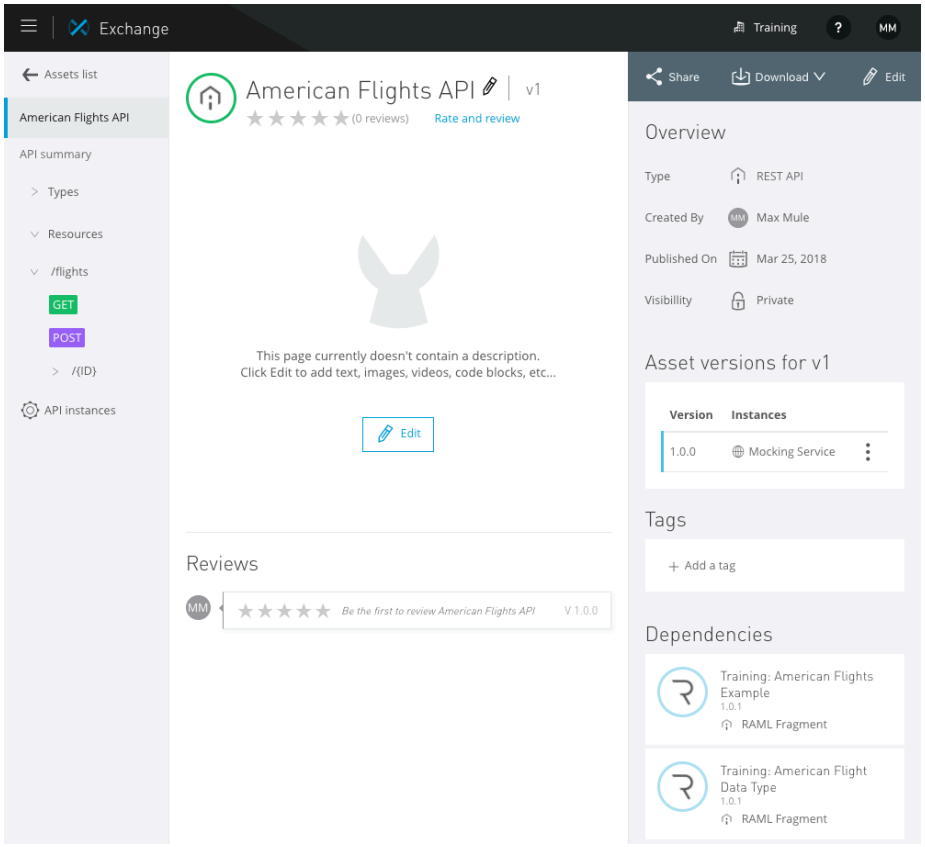


15. In the left-side navigation, select the name of your organization (Training in the screenshots); you should see your American Flights API in your private Exchange.

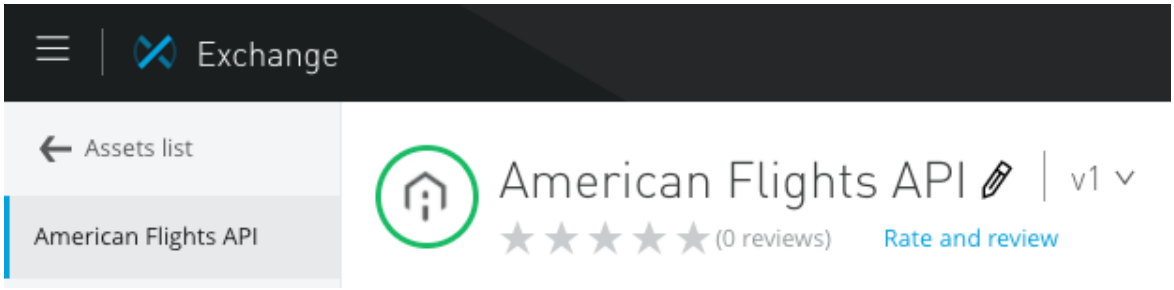


Review the auto-generated API portal

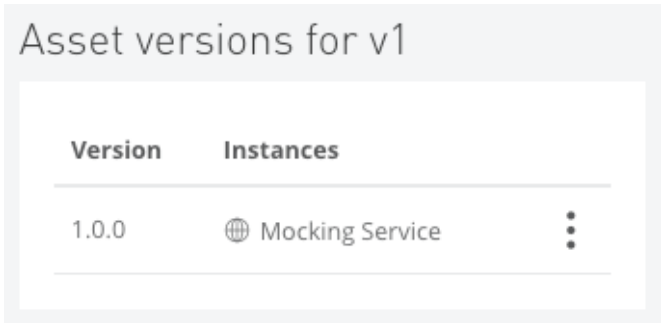
16. Click the American Flights API.
17. Review the page; you should see that as the creator of this API, you can edit, review, share, download, and add tags to this version.



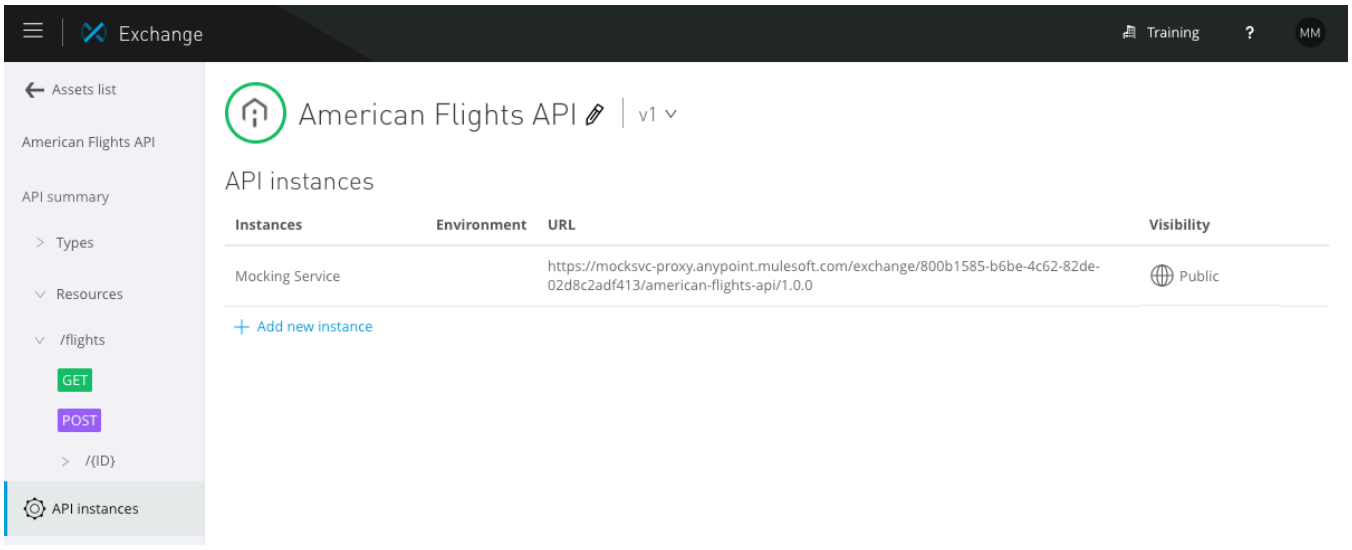
18. Locate the API dependencies in the lower-right corner.
19. Locate the API version (v1) next to the name of the API at the top of the page.



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

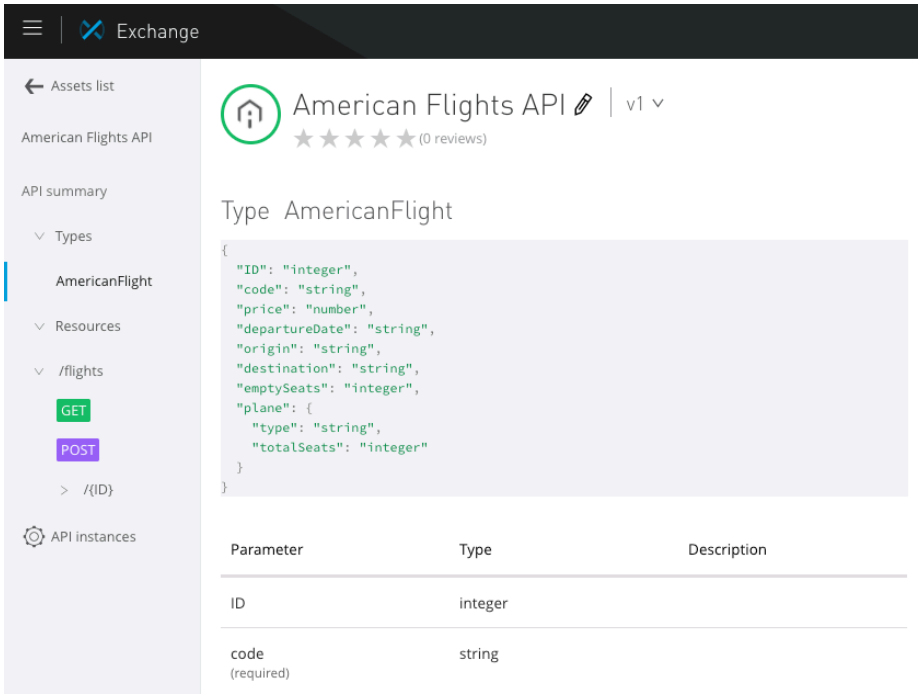


21. In the left-side navigation, select API instances; you should see information for the API instance generated from the API specification using the mocking service.



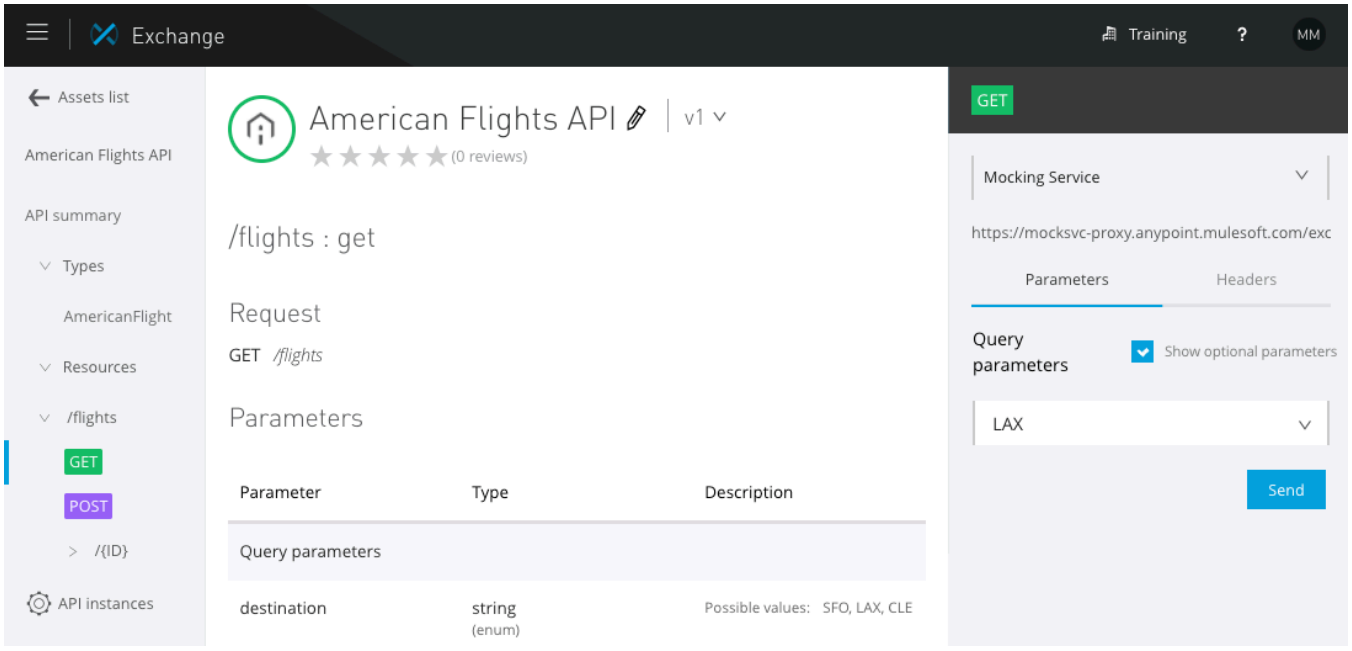
22. In the left-side navigation, expand Types.

23. Select AmericanFlight and review the information.

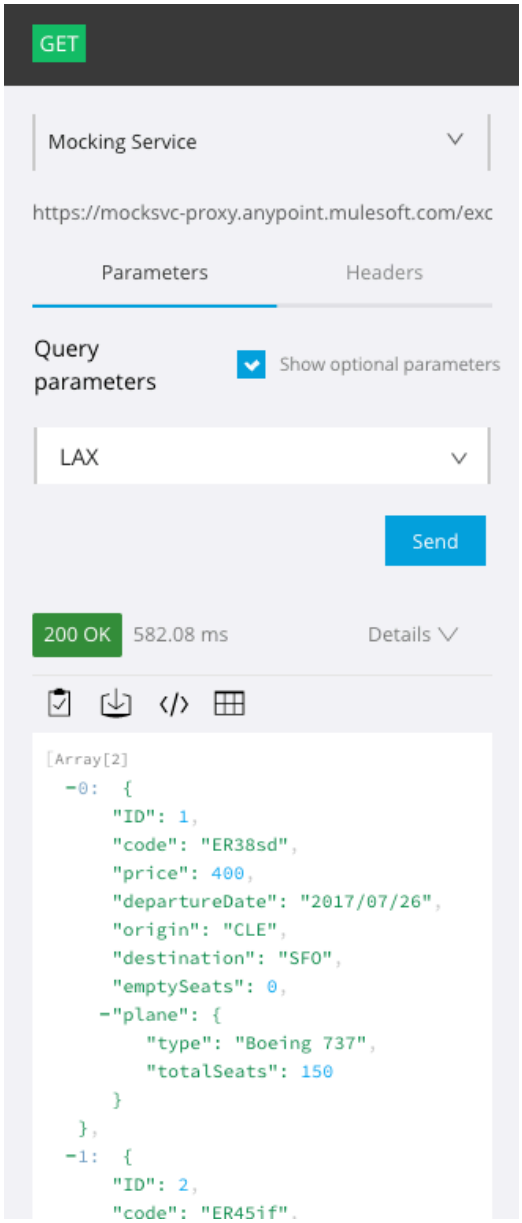


Test the API in its API portal in Exchange

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



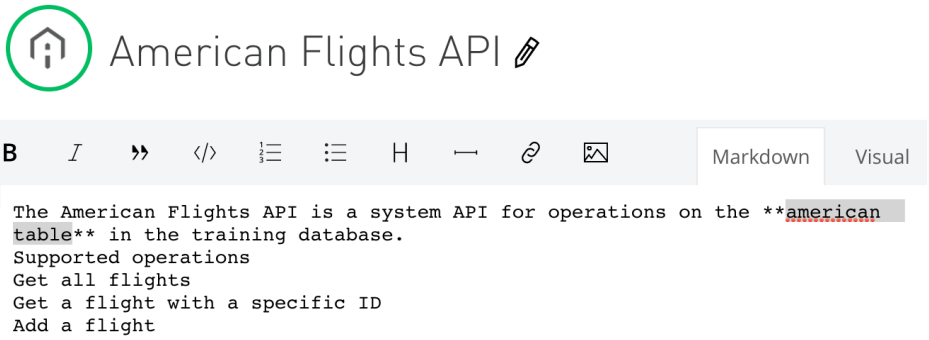
27. Click Send; you should get a 200 response and the example data displayed – just as you did in API console in API designer.



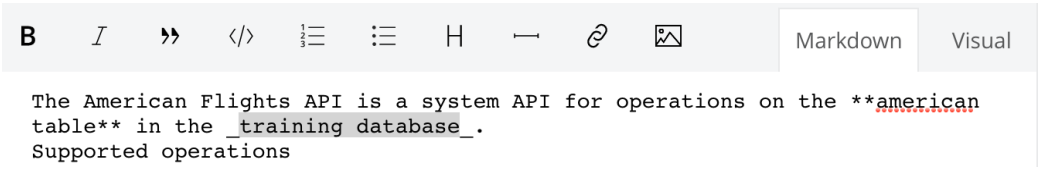
Add information about the API

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

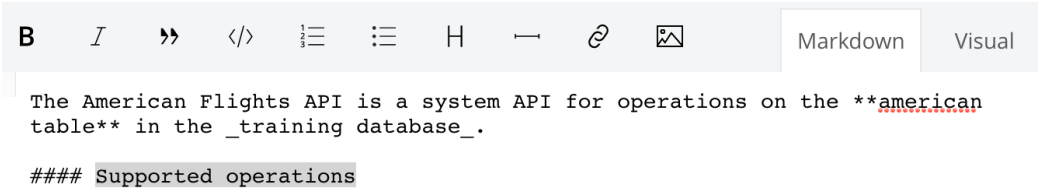
32. Select the words american table and click the strong button (the B).



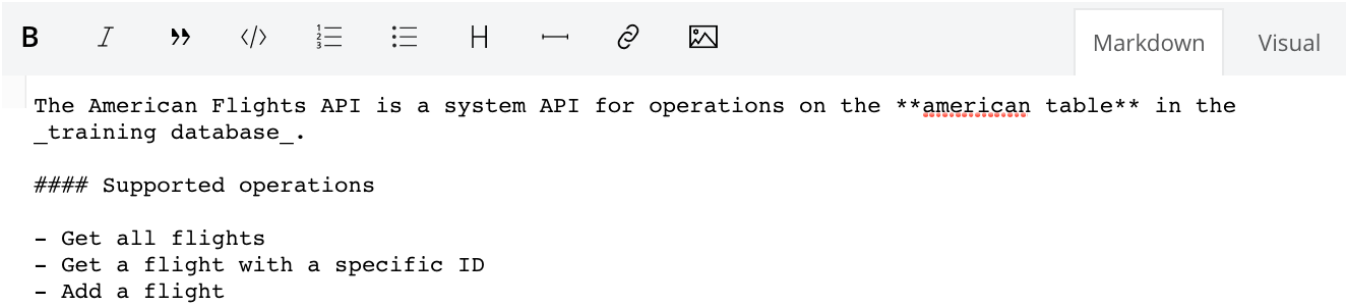
33. Select the words training database and click the emphasis button (the I).



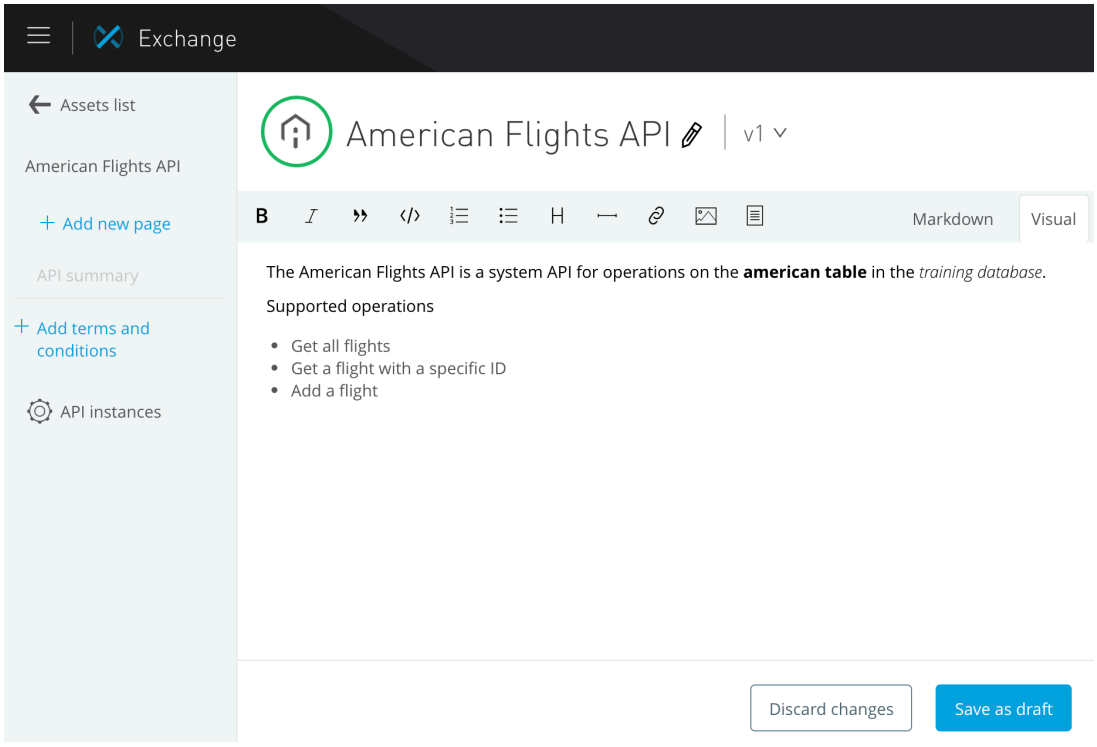
34. Select the words Supported operations and click the heading button four times (the H).



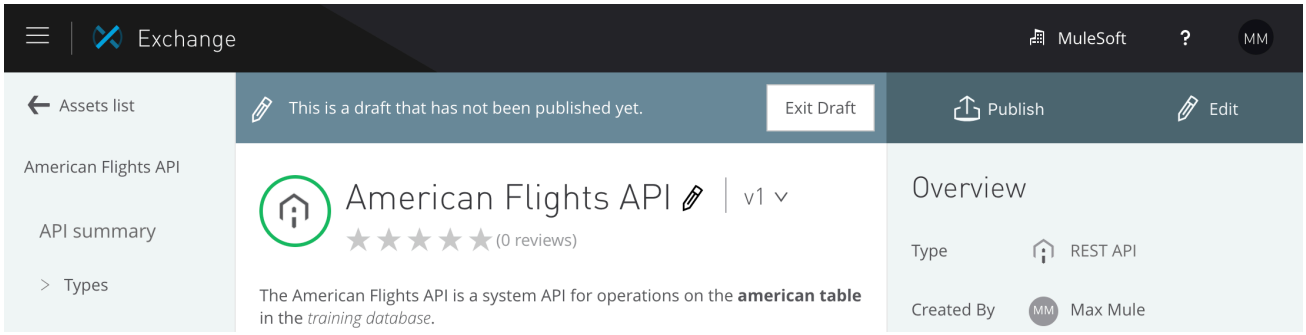
35. Select Get all flights and click the bulleted list button.
36. Repeat for the other operations.



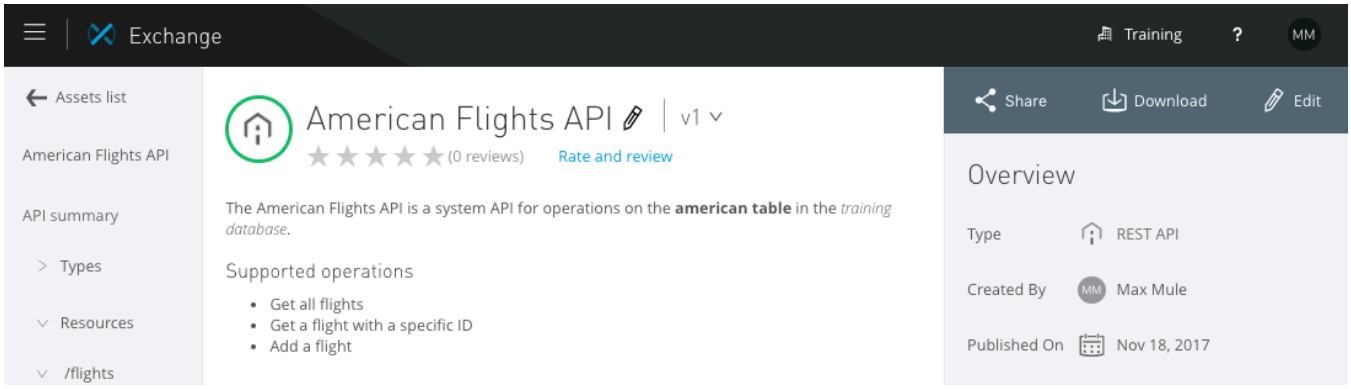
37. Select the Visual tab in the editor toolbar and view the rendered markdown.



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



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

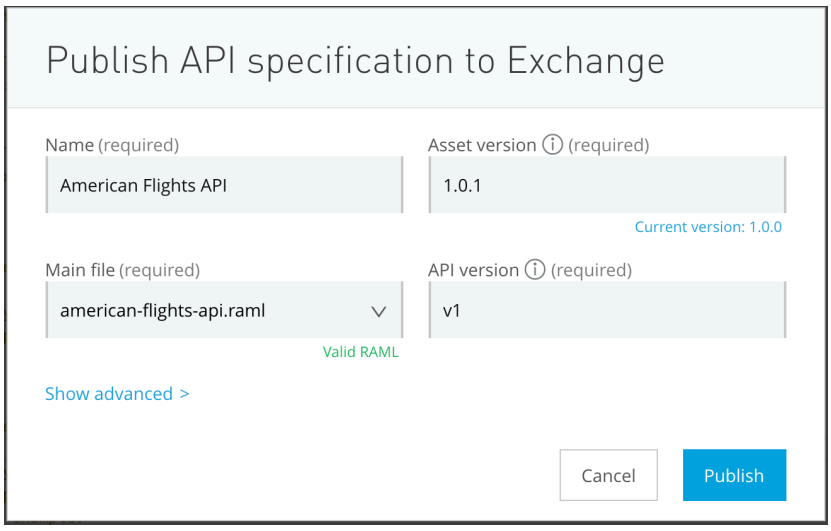


Modify the API and publish the new version to Exchange

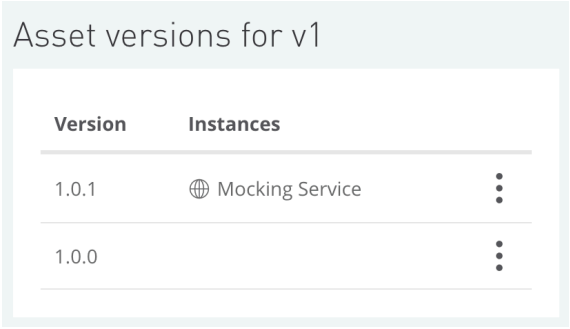
40. Return to your American Flights API in API designer.
41. Return to the course snippets.txt file and copy the American Flights API - /{ID} DELETE and PUT methods.
42. Return to american-flights-api.raml and paste the code after the {ID}/get method.
43. Fix the indentation.
44. Review the code.

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

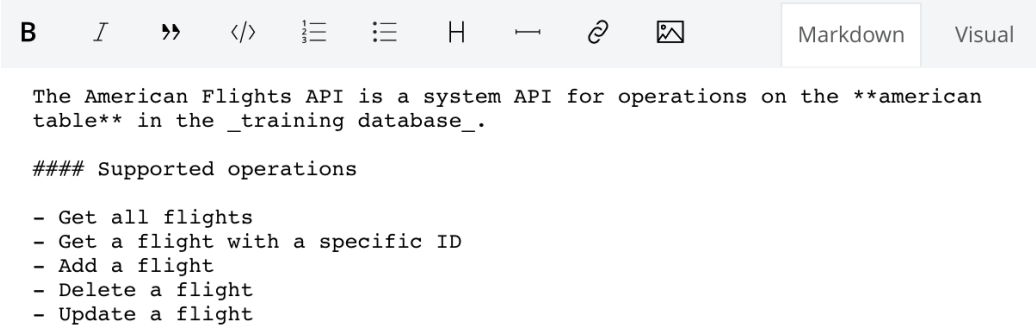
45. Click the Publish to Exchange button.
46. In the Publish API specification to Exchange dialog box, look at the asset version.



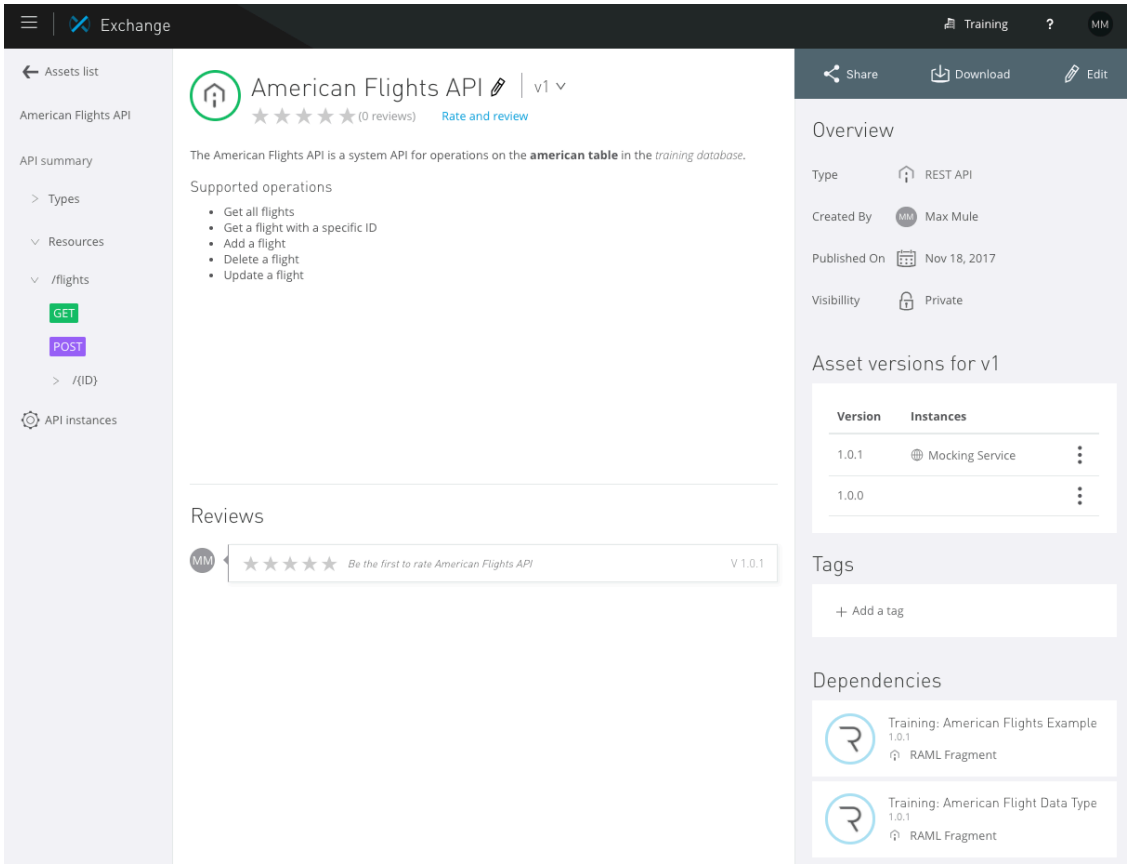
47. Click Publish.
48. In the Publish API specification to Exchange dialog box, click Exchange; Exchange should open in a new browser tab.
49. Locate the asset versions listed for the API; you should see both asset versions of the API listed with an associated API instance using the mocking service for the latest version.



50. Click the Edit button.
51. Add two new operations that delete a flight and update a flight.



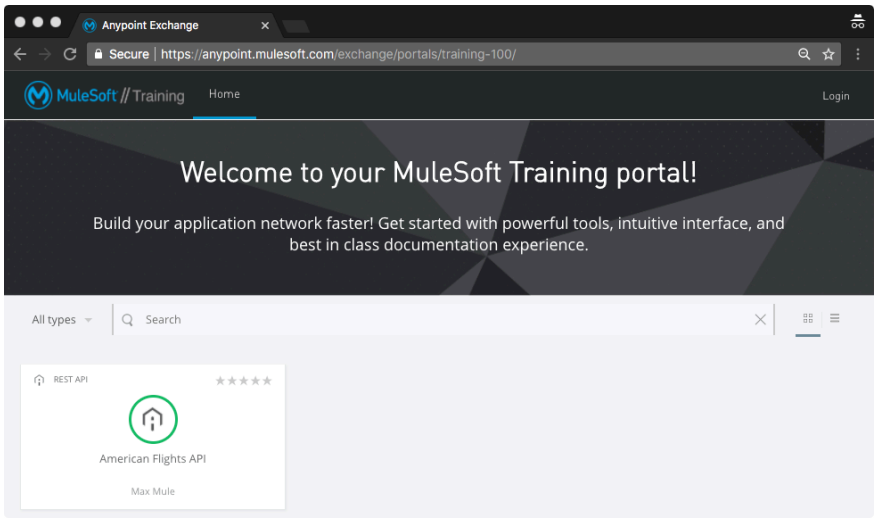
52. Click Save as draft and then Publish.



Walkthrough 3-5: Share an API

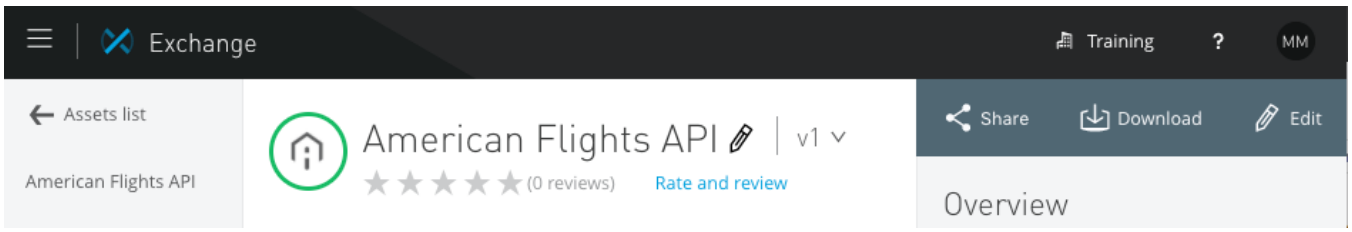
In this walkthrough, you share an API with both internal and external developers to locate, learn about, and try out the API. You will:

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

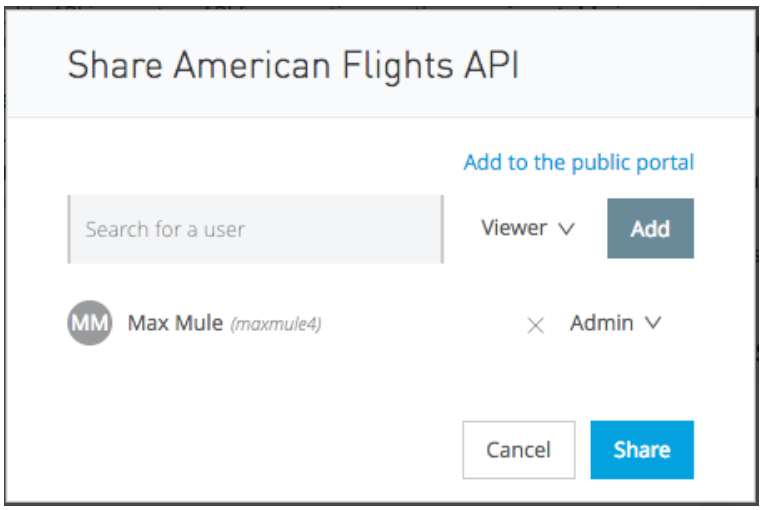


Share the API in the private Exchange with others

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



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

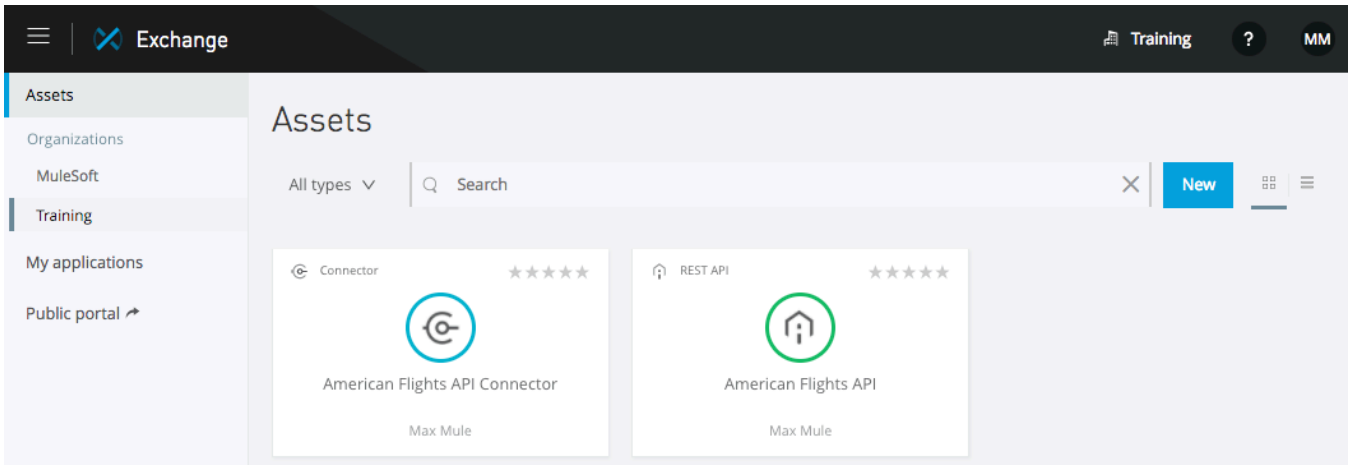


4. Open the Viewer drop-down menu located next to the user search field; you should see that you can add additional users to be of type Viewer, Contributor, or Admin.

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

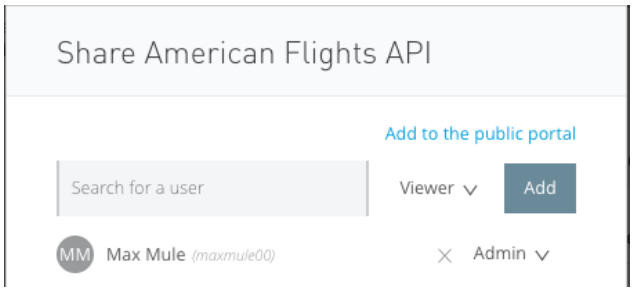
5. Click Cancel.
6. In the left-side navigation, click Assets list.
7. In the left-side navigation, select the name of your organization.
8. Click your American Flights API.

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

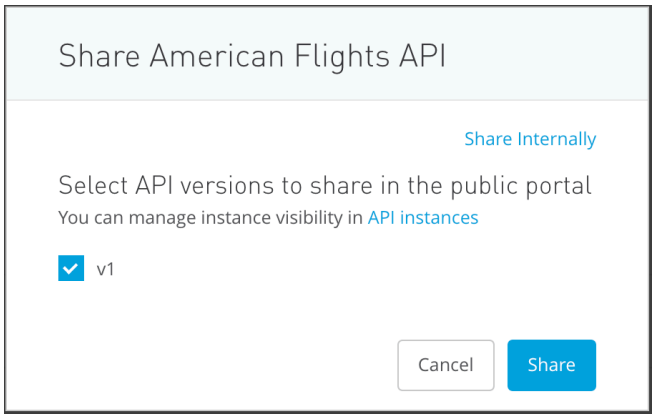


Create a public API portal

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

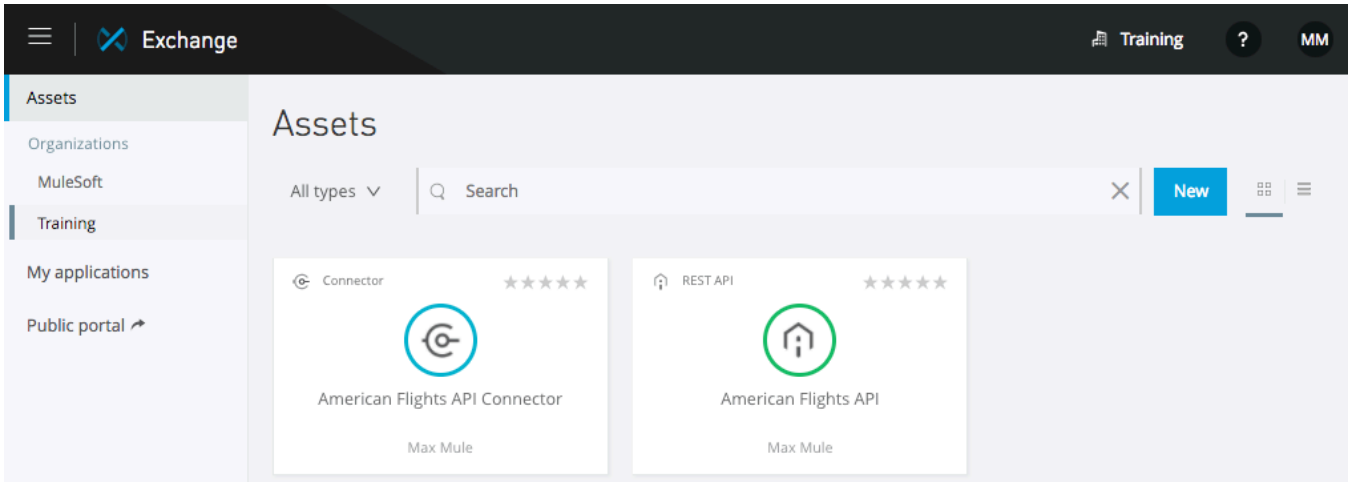


11. In the new dialog box, select to share v1 of the API.
12. Click Share.

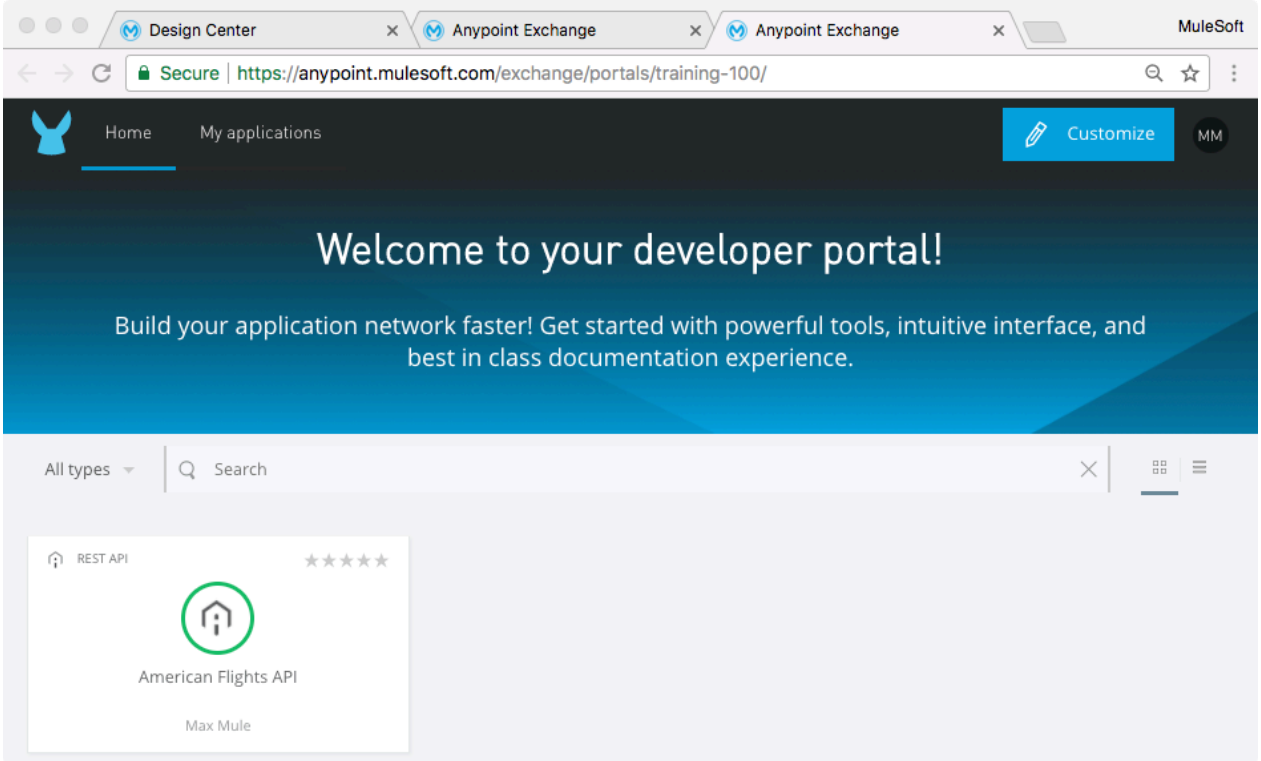


Explore the API in the public portal

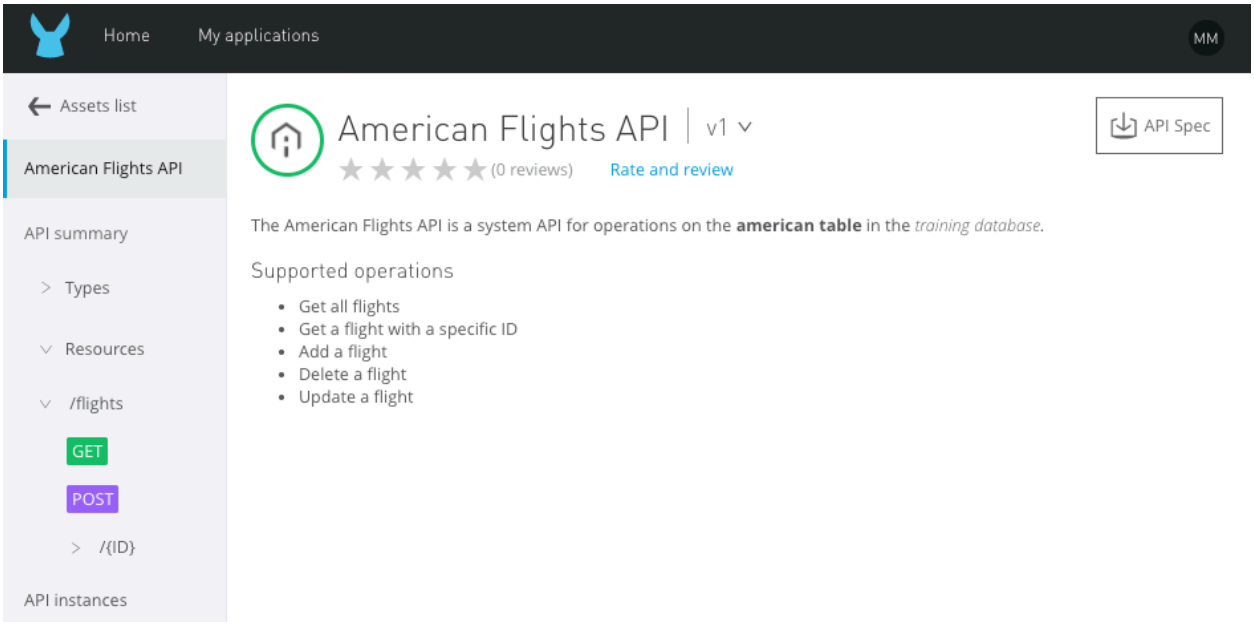
13. In the left-side navigation, click Assets list.
14. In the left-side navigation, select Public Portal.



15. Review the public portal that opens in a new browser tab.
16. Look at the URL for the public portal.

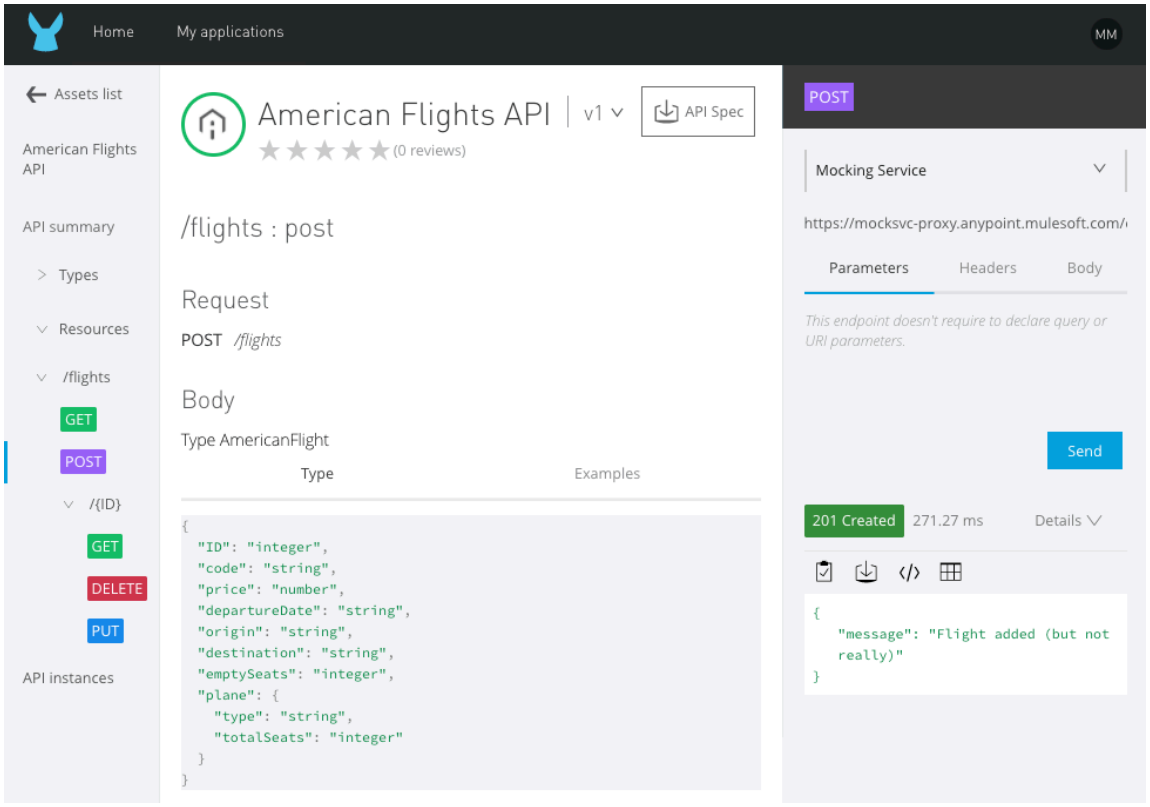


17. Click your American Flights API and review the auto-generated public API portal.



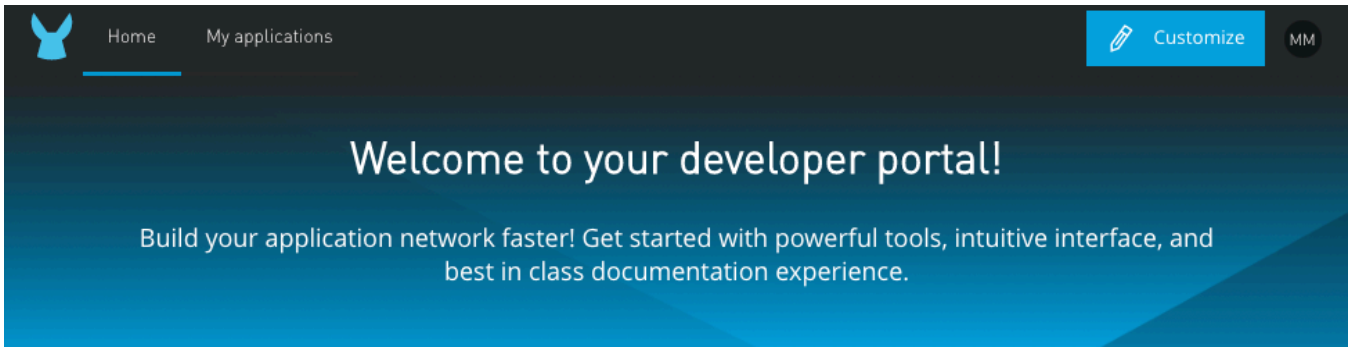
18. In the left-side navigation, click the POST method for the flights resource.
19. Review the body and response information.

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

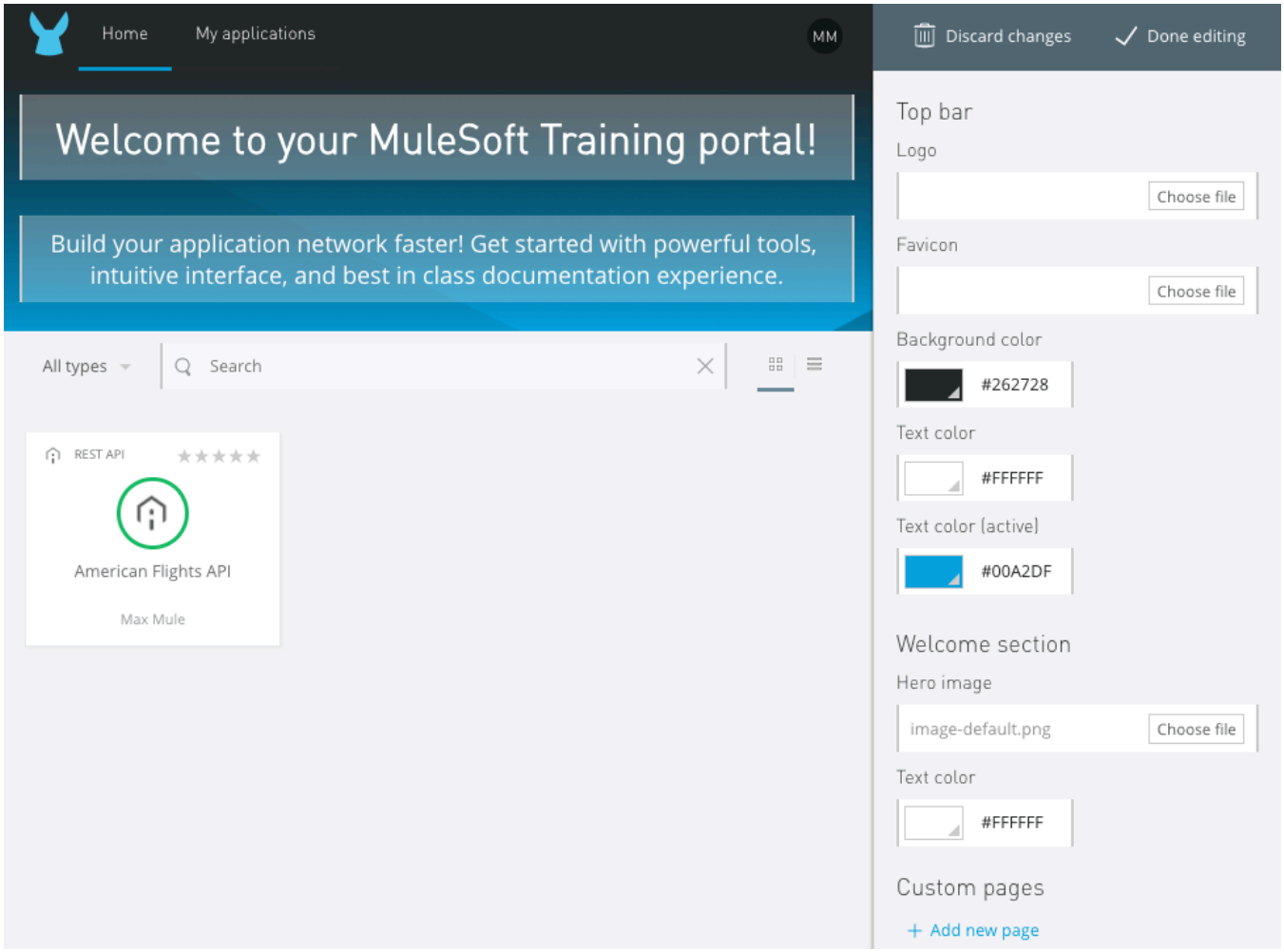


Customize the public portal

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

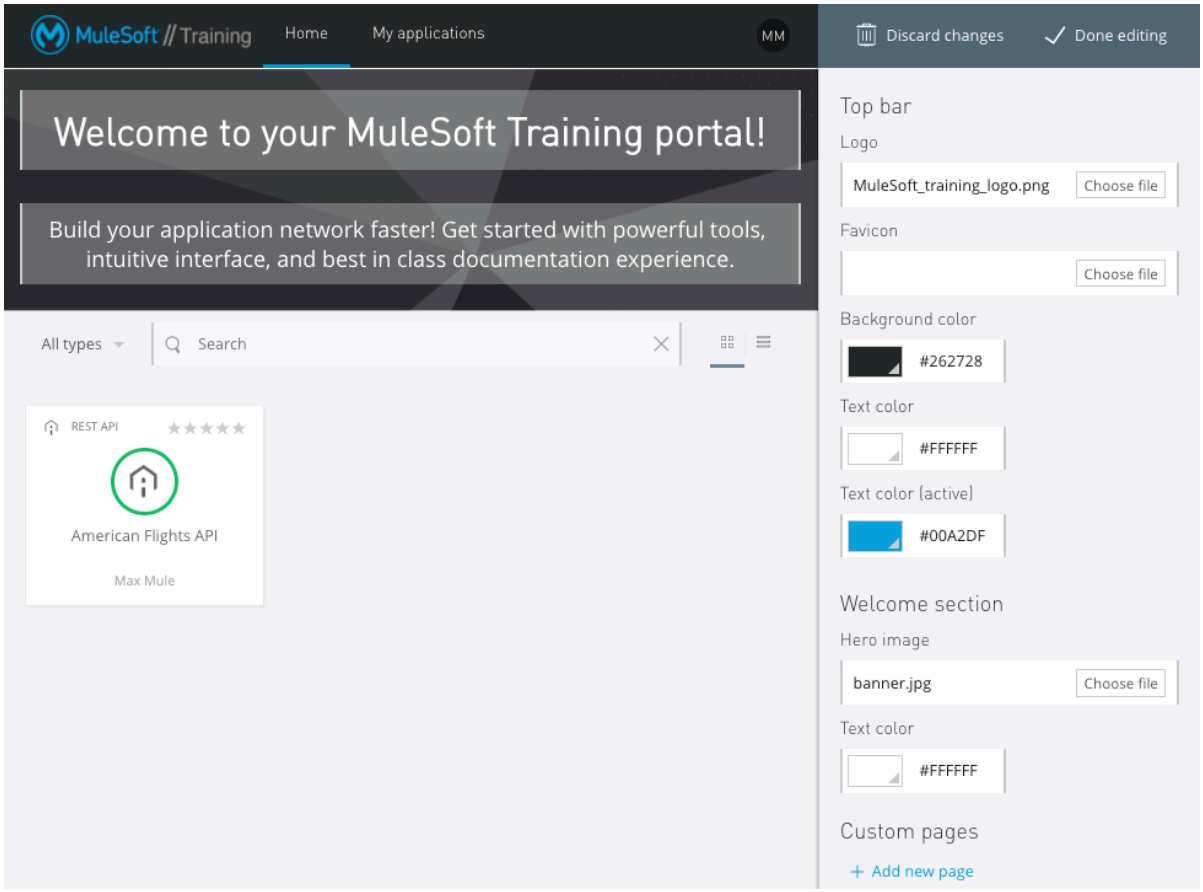


23. Change the text to Welcome to your MuleSoft Training portal!



24. In the logo field, click Choose file.
25. In the file browser dialog box, navigate to the student files and locate the MuleSoft_training_logo.png file in the resources folder and click Open.
26. Locate the new logo in the preview.
27. In the hero image field, click Choose file.
28. In the file browser dialog box, navigate to the student files and locate the banner.jpg file in the resources folder and click Open.

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

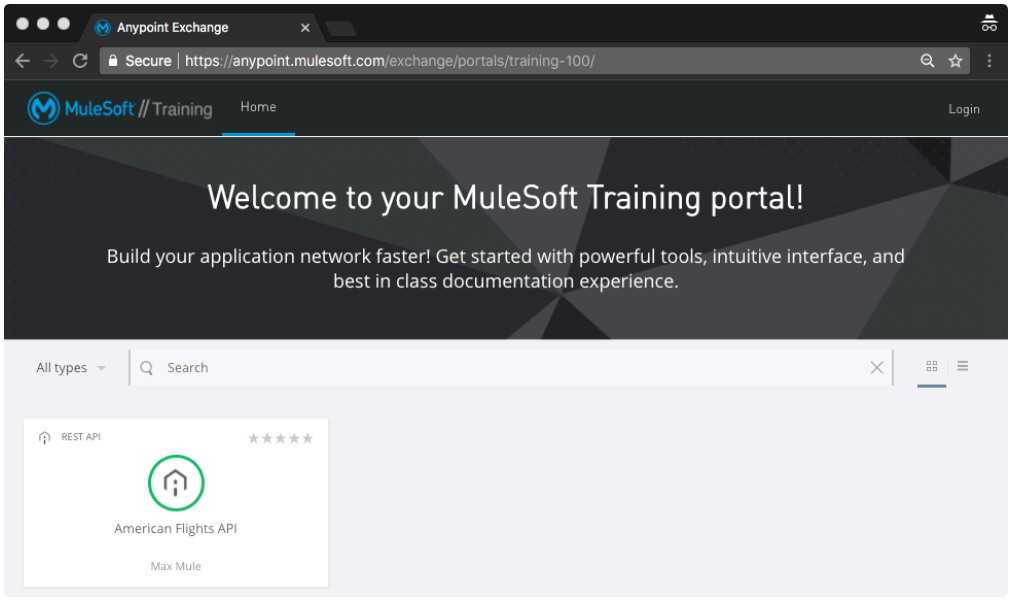


30. Change any colors that you want.
31. Click the Done editing button.
32. In the Publish changes dialog box, click Yes, publish; you should see your customized public portal.

Explore the public portal as an external developer

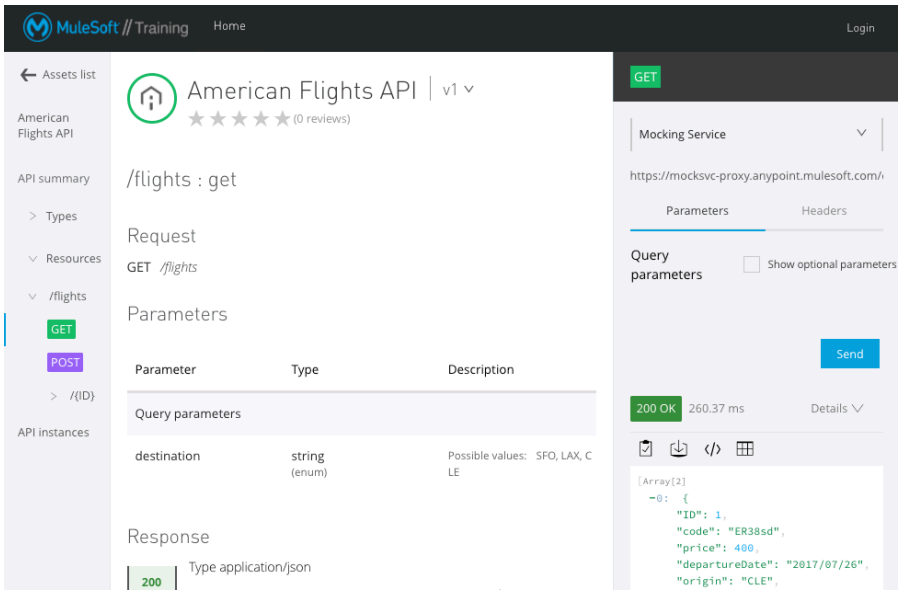
33. In the browser, copy the URL for the public portal.
34. Open a new private or incognito window in your browser.

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

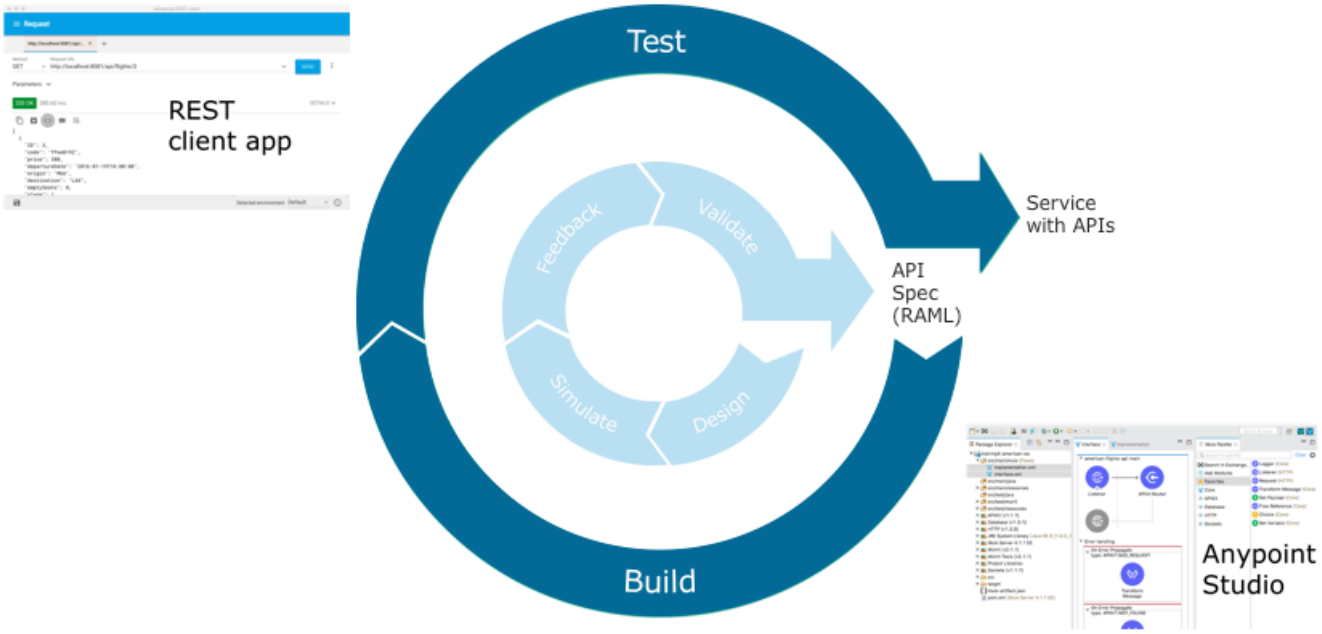


36. Click the American Flights API.
37. Explore the API portal.
38. Make a call to one of the resource methods.

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



Module 4: Building APIs



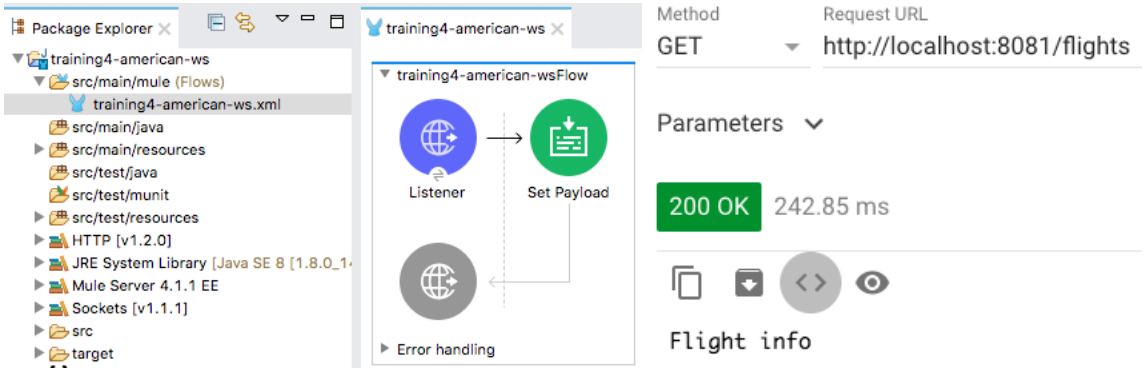
At the end of this module, you should be able to:

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

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

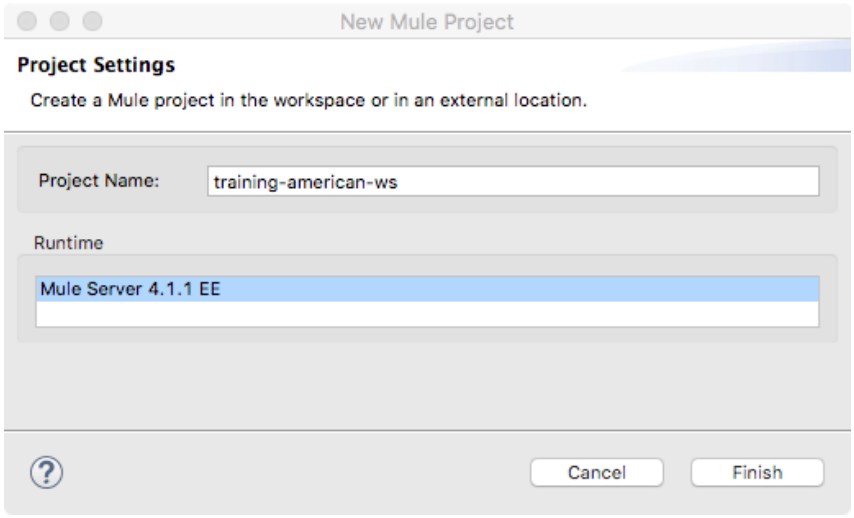
In this walkthrough, you build a Mule application. You will:

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



Create a Mule project

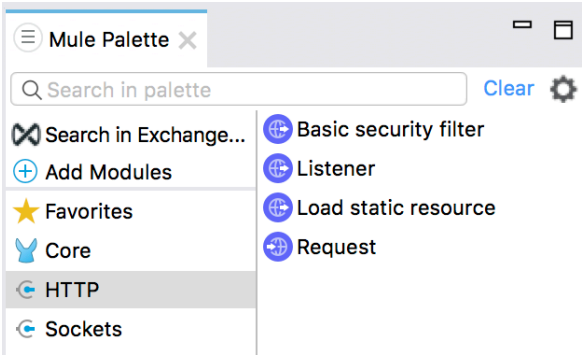
1. Open Anypoint Studio.
2. Select File > New > Mule Project.
3. In the New Mule Project dialog box, set the Project Name to training4-american-ws.
4. Ensure the Runtime is set to the latest version of Mule.



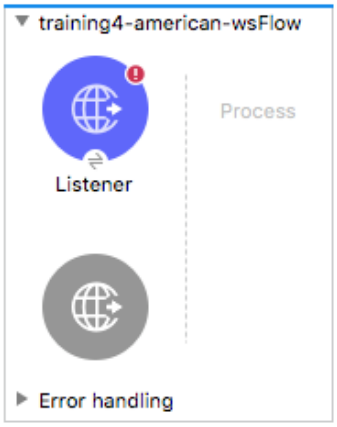
5. Click Finish.

Create an HTTP connector endpoint to receive requests

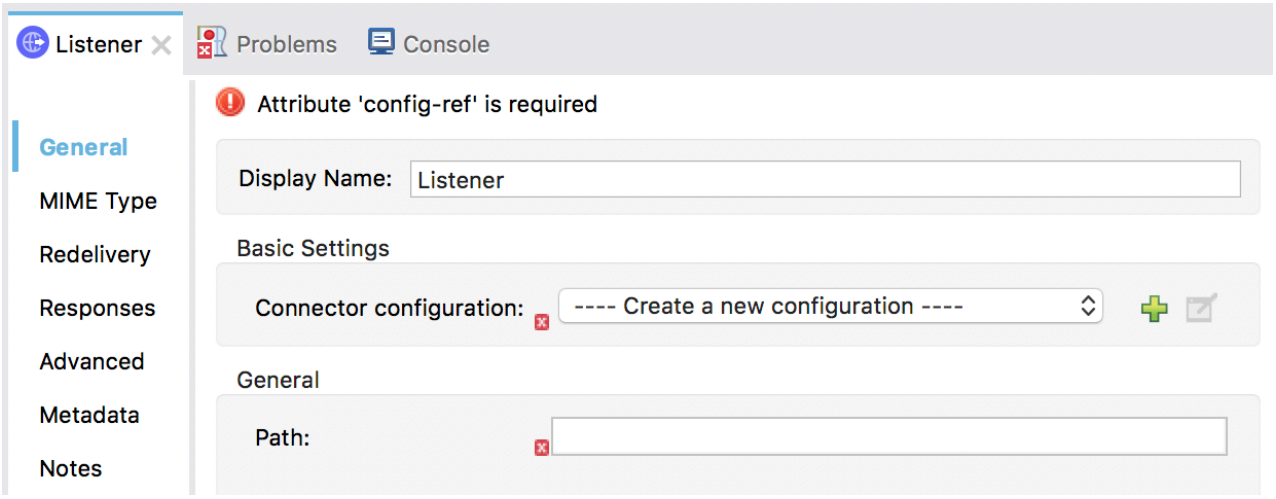
6. In the Mule Palette, select the HTTP module.



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

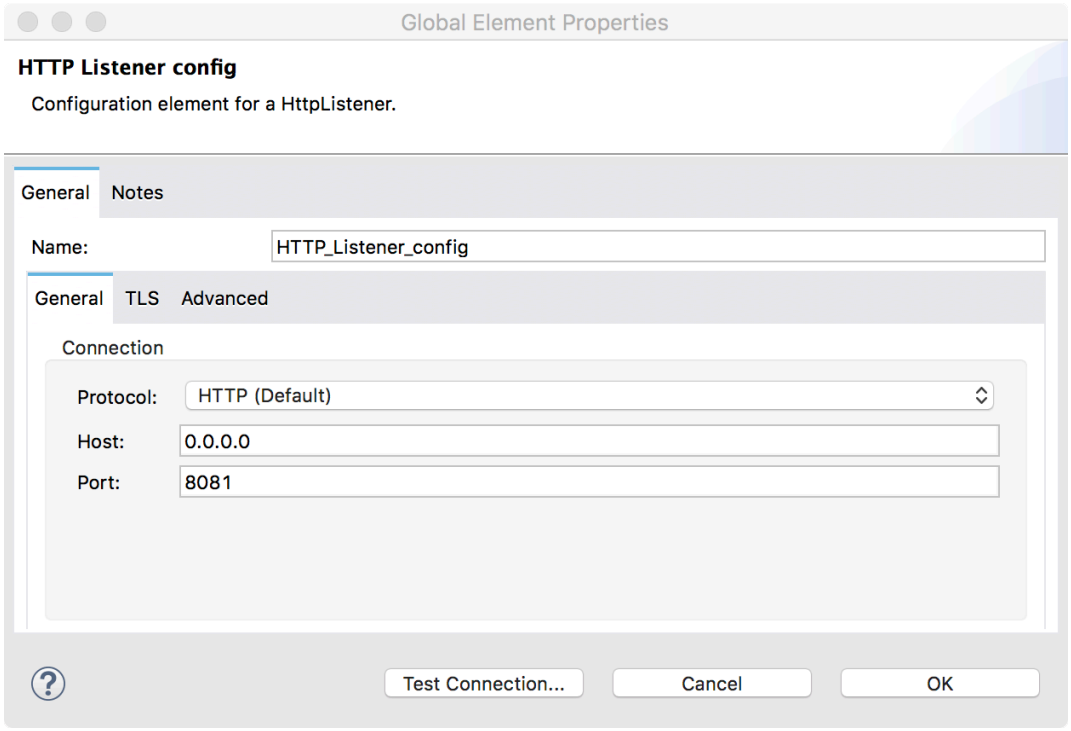


8. Double-click the HTTP Listener endpoint.
9. In the Listener properties view that opens at the bottom of the window, click the Add button next to connector configuration.

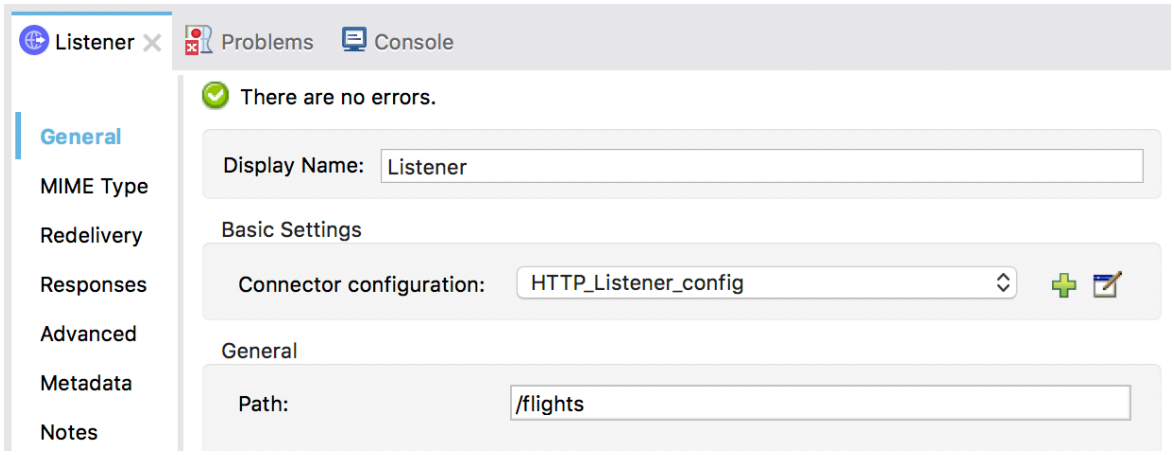


10. In the Global Element Properties dialog box, set the following default values:

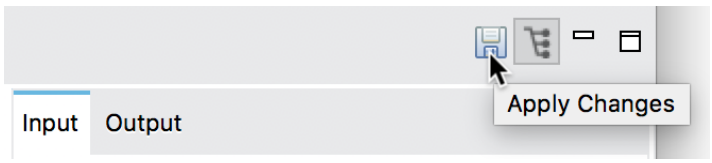
- Host: 0.0.0.0
- Port: 8081



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

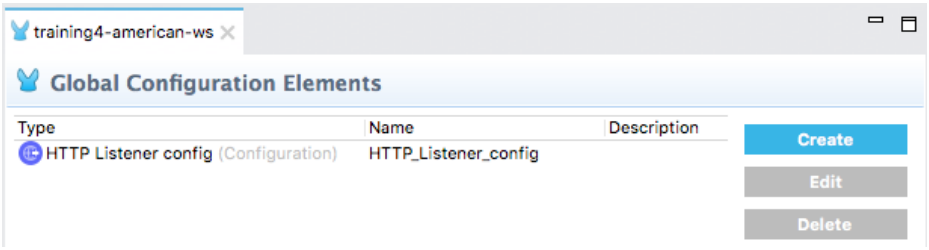


13. Click the Apply Changes button to save the file.



Review the HTTP Listener global element

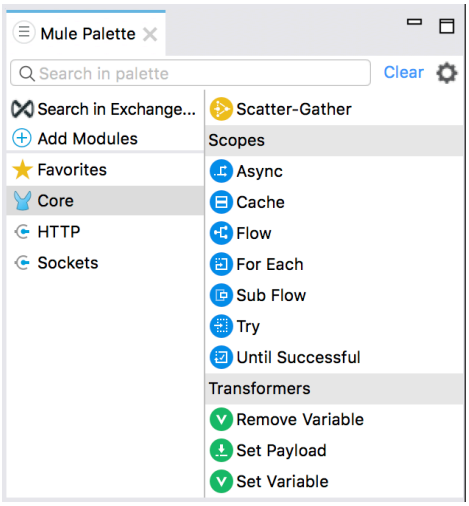
14. Select the Global Elements tab at the bottom of the canvas.
15. Double-click the HTTP Listener config.



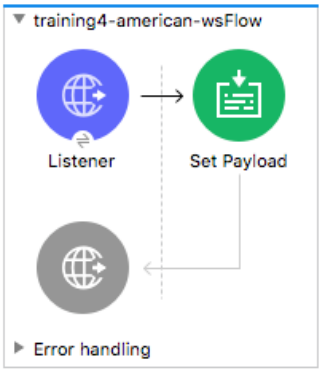
16. Review the information in the Global Element Properties dialog box and click Cancel.
17. Select the Message Flow tab to return to the canvas.

Display data

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

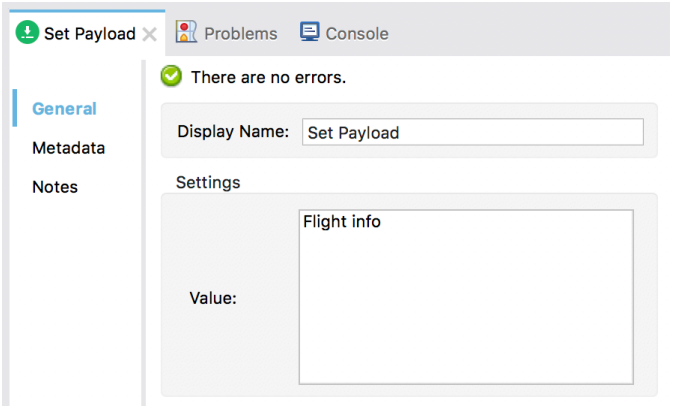


20. Drag the Set Payload transformer from the Mule Palette into the process section of the flow.

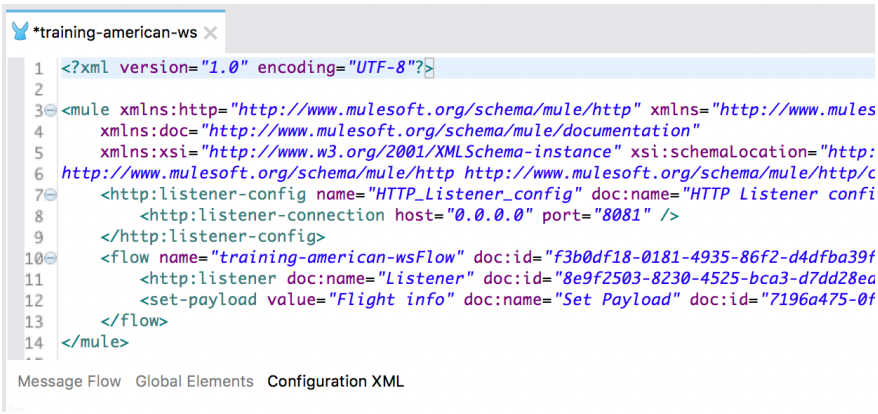


Configure the Set Payload transformer

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



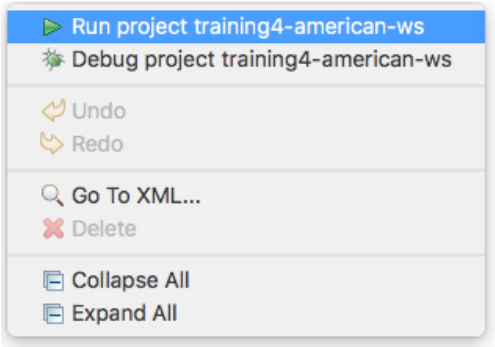
22. Select the Configuration XML tab at the bottom of the canvas and examine the corresponding XML.



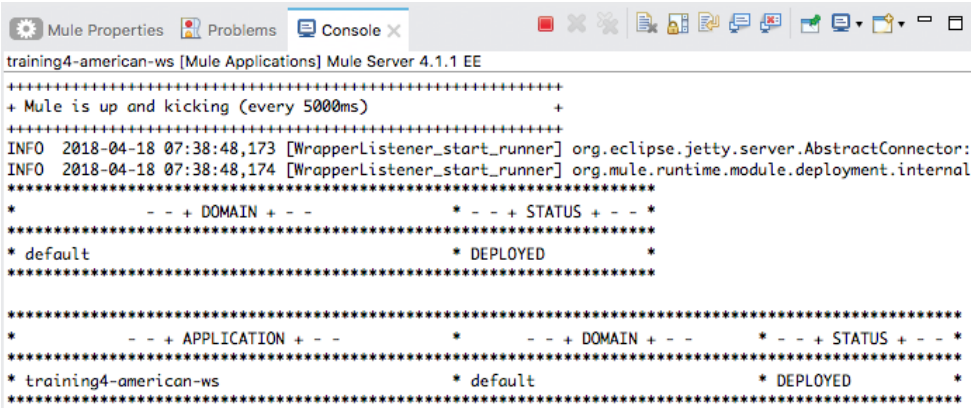
23. Select the Message Flow tab to return to the canvas.
24. Click the Save button or press Cmd+S or Ctrl+S.

Run the application

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

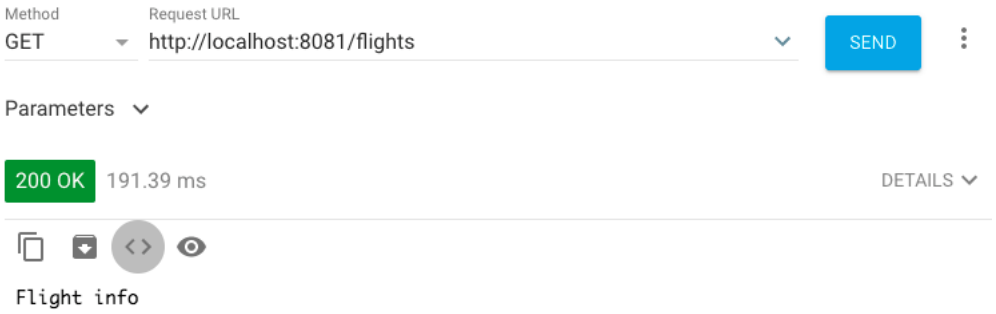


26. Watch the Console view; it should display information letting you know that both the Mule runtime and the training4-american-ws application started.

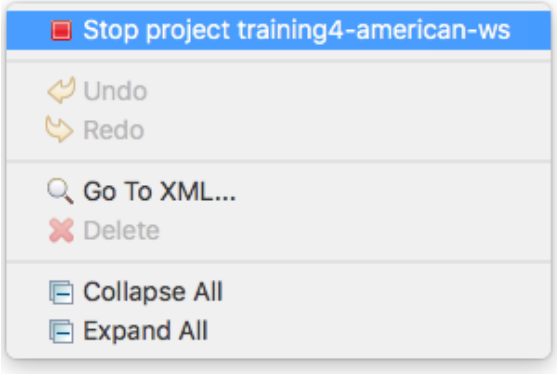


Test the application

27. Return to Advanced REST Client.
28. Make sure the method is set to GET and that no headers or body are set for the request.
29. Make a GET request to http://localhost:8081/flights; you should see Flight info displayed.



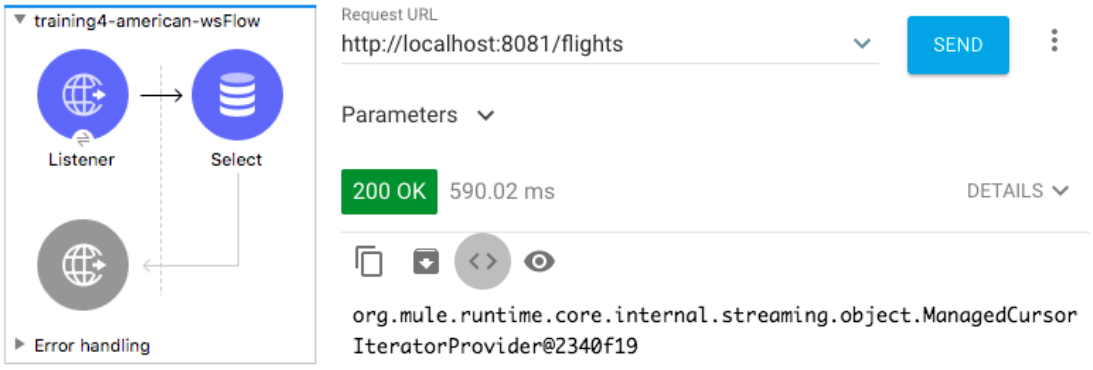
30. Return to Anypoint Studio.
31. Right-click in the canvas and select Stop project training4-american-ws.



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

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

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



Locate database information

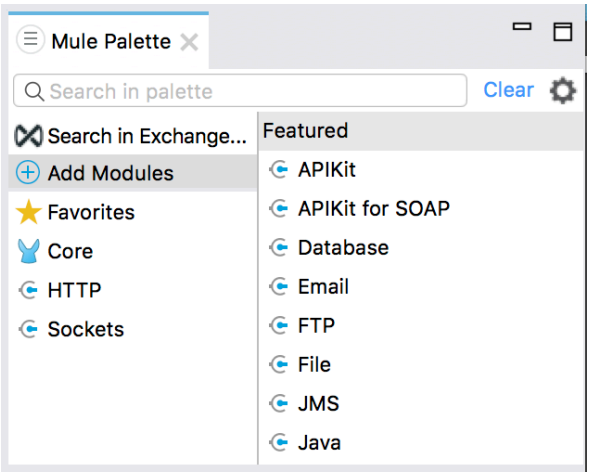
1. Return to the course snippets.txt file and locate the MySQL and Derby database information.

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

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

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

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

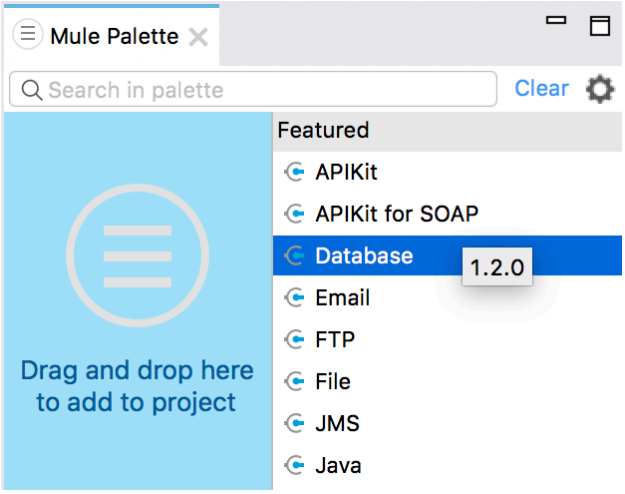
Note: The database information you see may be different than what is shown here; the values in the snippets file differ for instructor-led and self-study training classes.

Add a Database connector endpoint

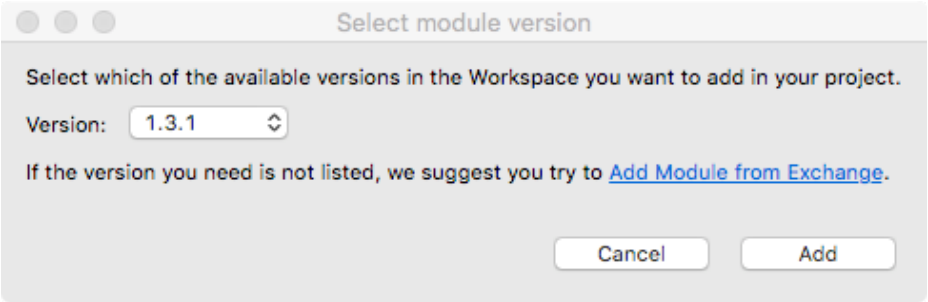
2. Return to Anypoint Studio.
3. Right-click the Set Payload message processor and select Delete.
4. In the Mule Palette, select Add Modules.



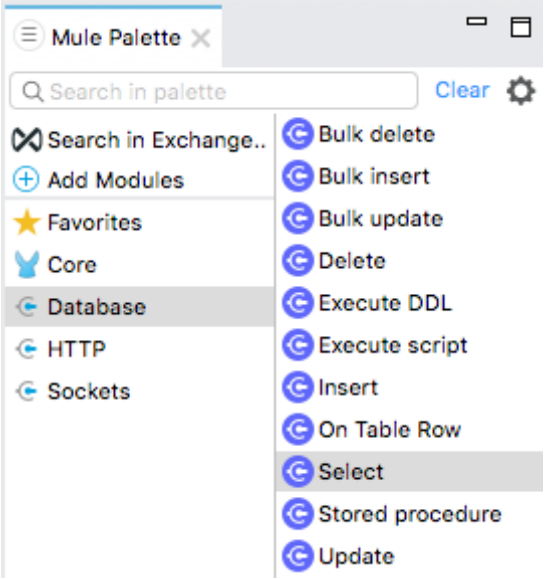
5. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.



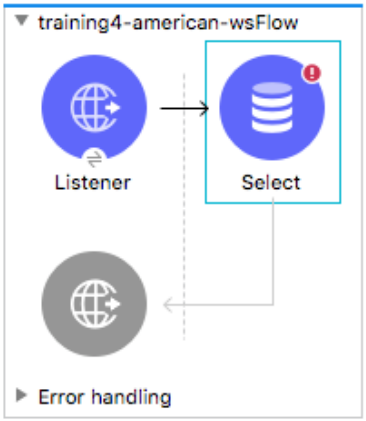
6. If you get a Select module version dialog box, select the latest version and click Add.



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

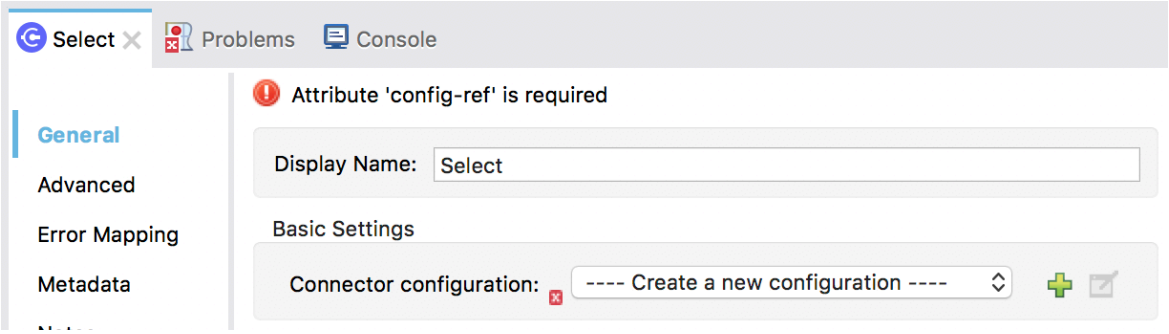


8. Drag and drop the Select operation in the process section of the flow.

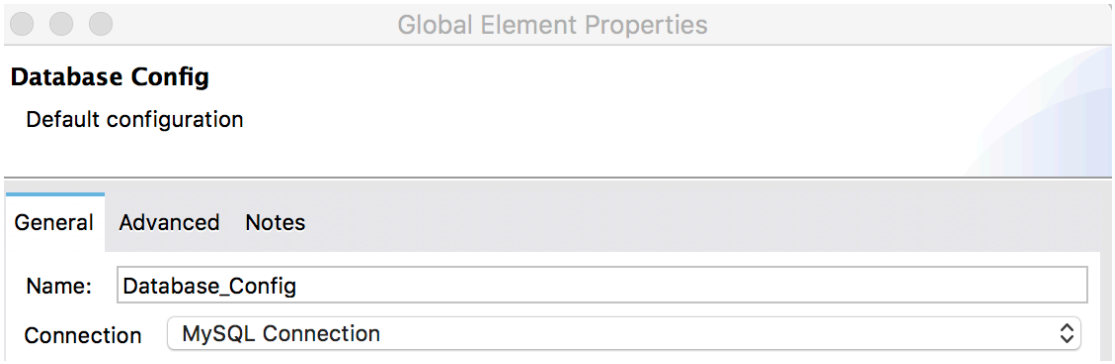


Option 1: Configure a MySQL Database connector (if you have access to port 3306)

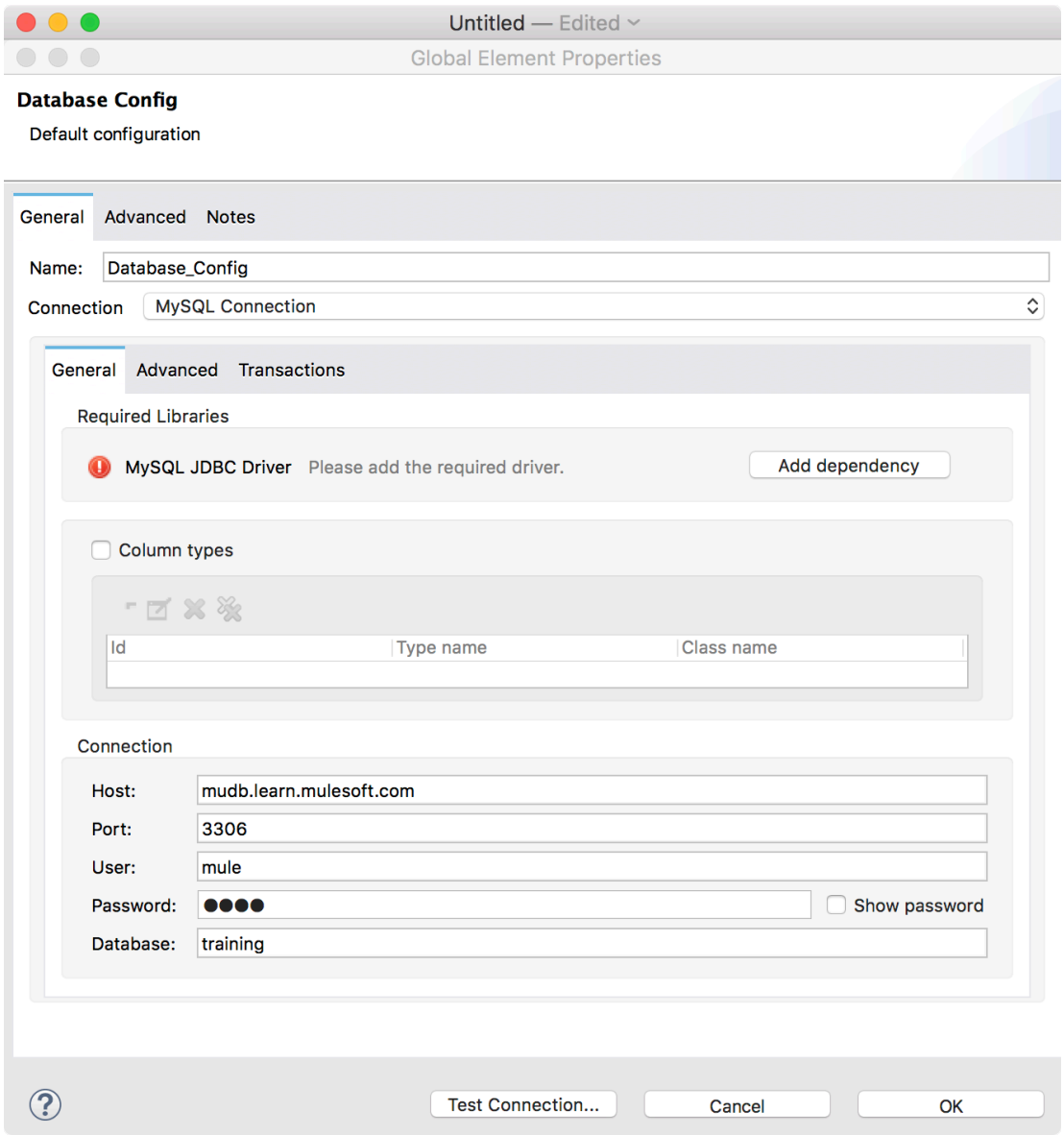
9. In the Select properties view, click the Add button next to connector configuration.



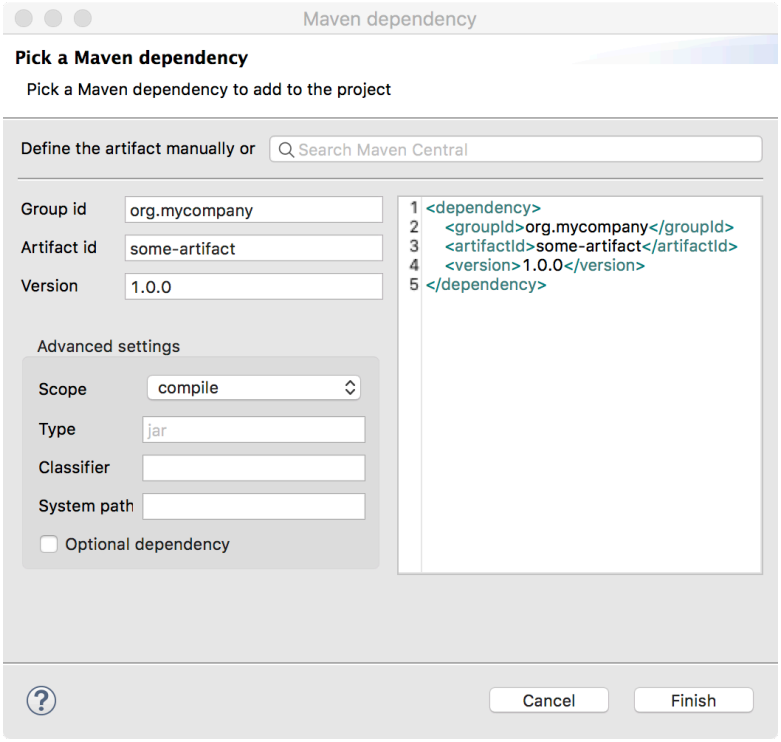
10. In the Global Element Properties dialog box, set the Connection to MySQL Connection.



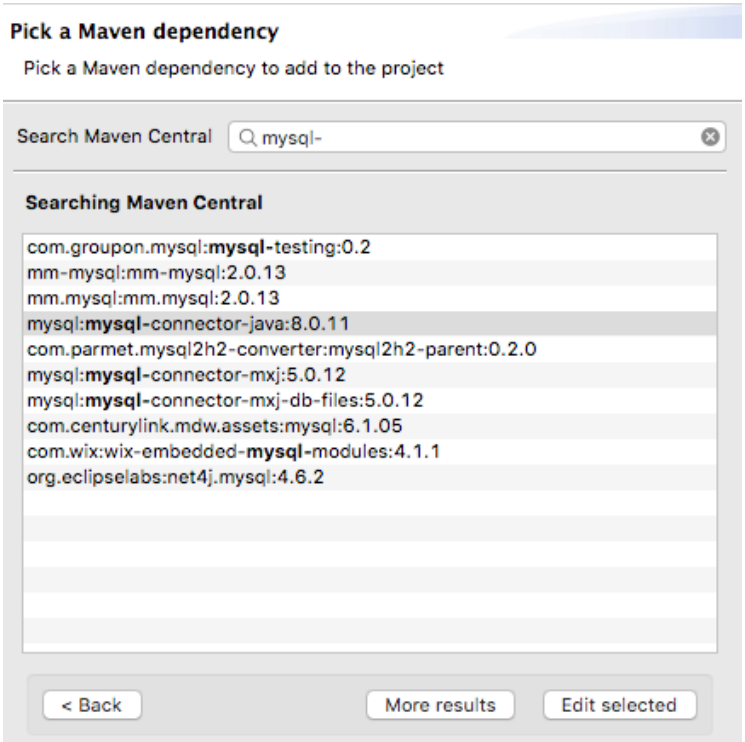
11. Set the host, port, user, password, and database values to the values listed in the course snippets.txt file.



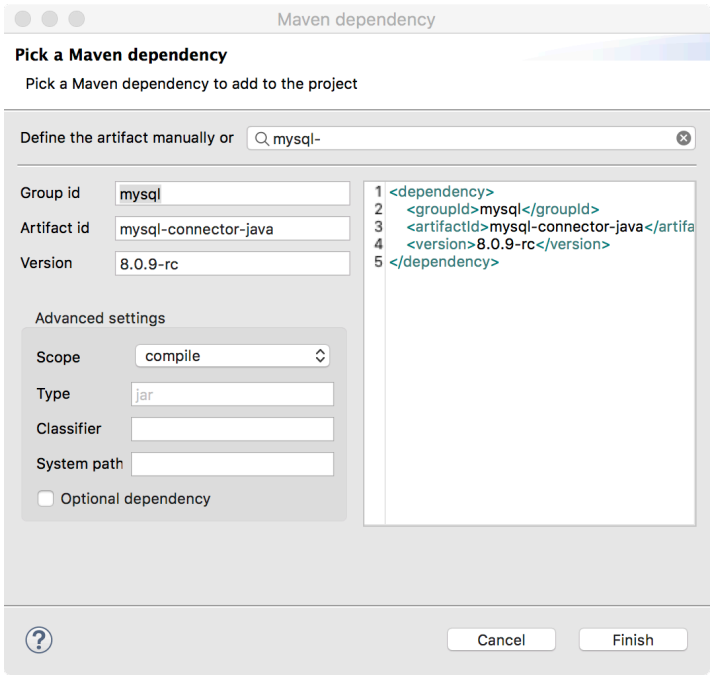
12. Click the Add dependency button next to MySQL JDBC Driver.
13. In the Maven dependency dialog box, locate the Search Maven Central text field.



14. Enter mysql- in the Search Maven Central text field.
15. Select mysql:mysql-connector-java in the results that are displayed.

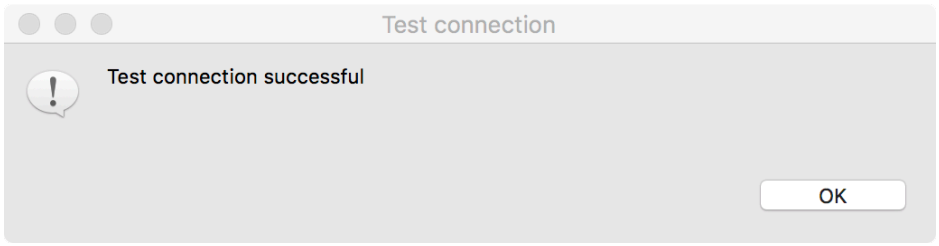


16. Click Edit selected.



17. Click Finish.
18. 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.

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

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

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

22. Run the mulesoft-training-services.jar file.

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

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

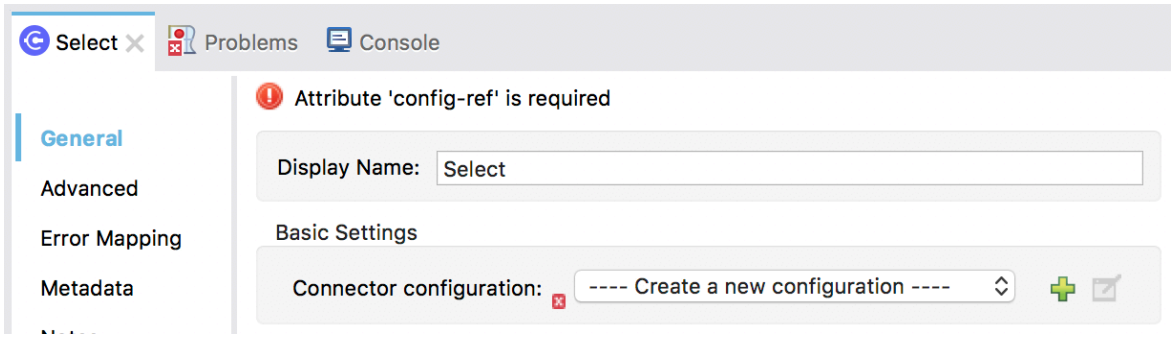
Note: The application uses ports 1527, 9090, 9091, and 61616. If any of these ports are already in use, you can change them when you start the application as shown in the following code.

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

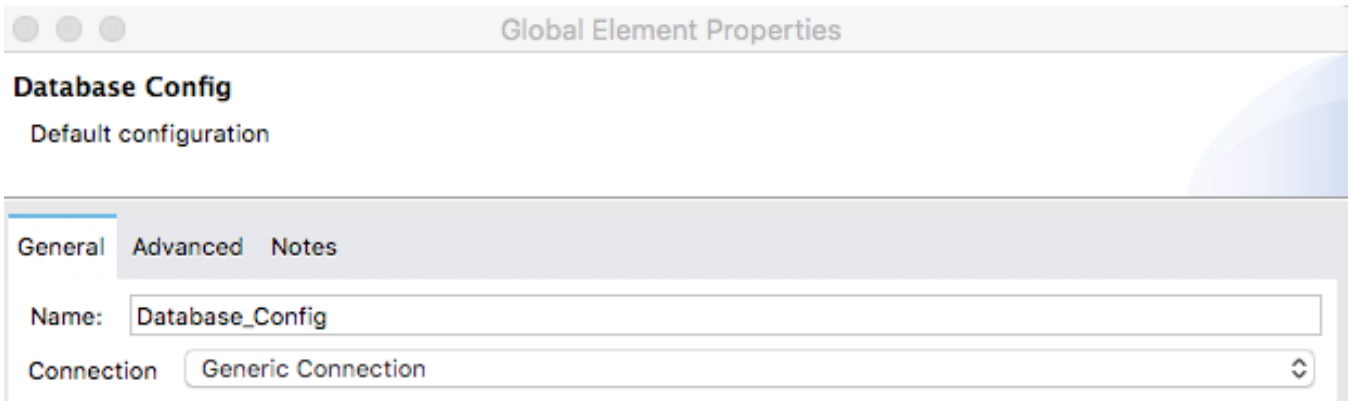
23. Look at the output and make sure all the services started.

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

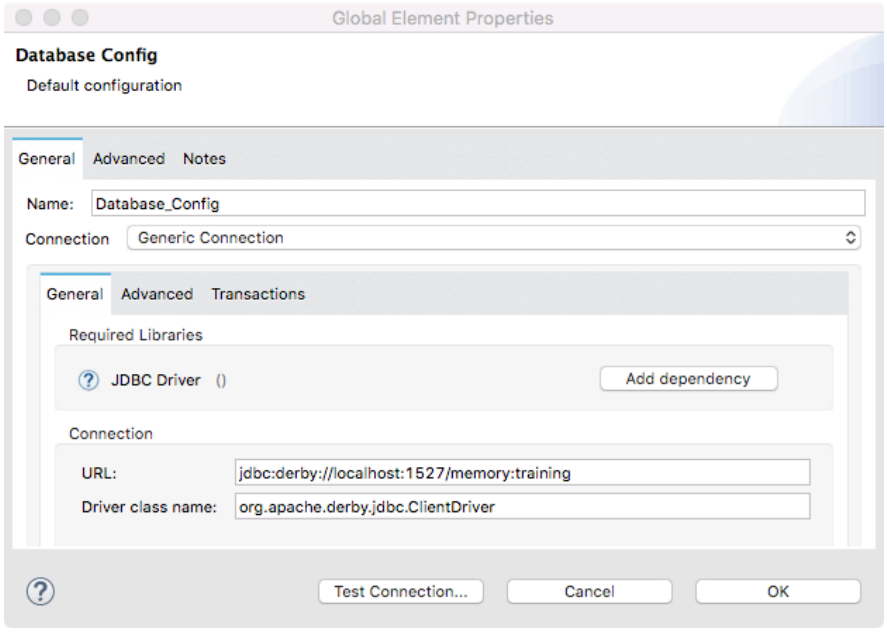
24. Return to Anypoint Studio.
25. In the Select properties view, click the Add button next to connector configuration.



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

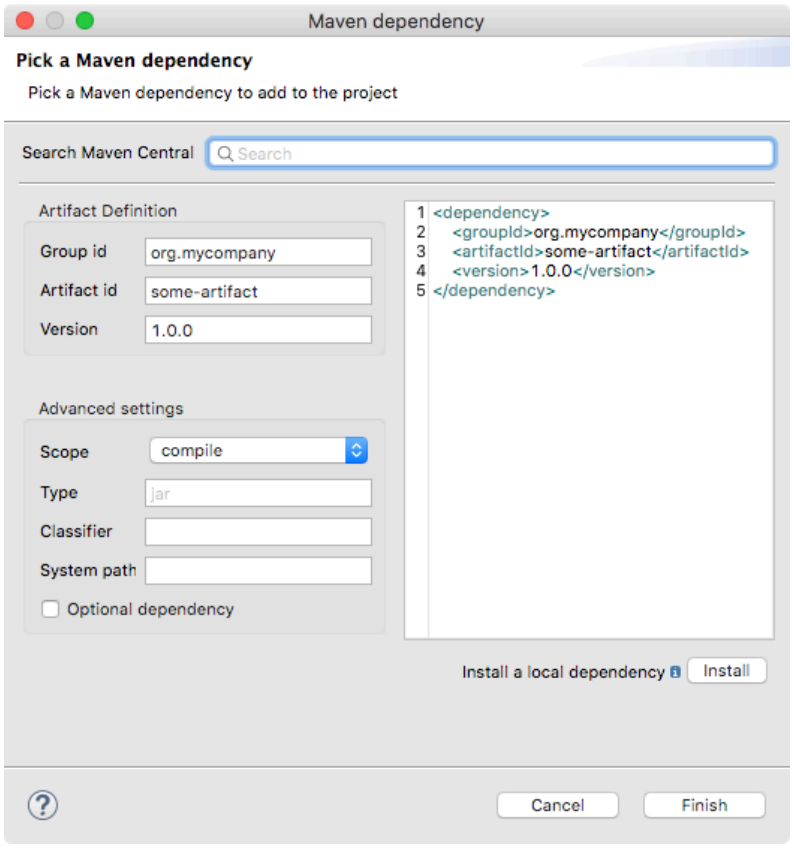


27. Set the URL and driver class name values to the values listed in the course snippets.txt file.

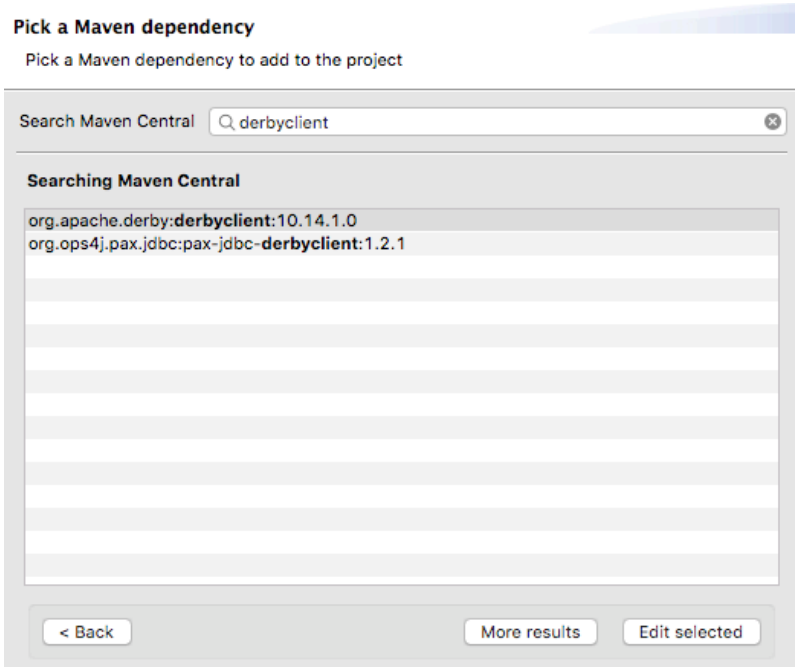


28. Click the Add dependency button next to JDBC Driver.

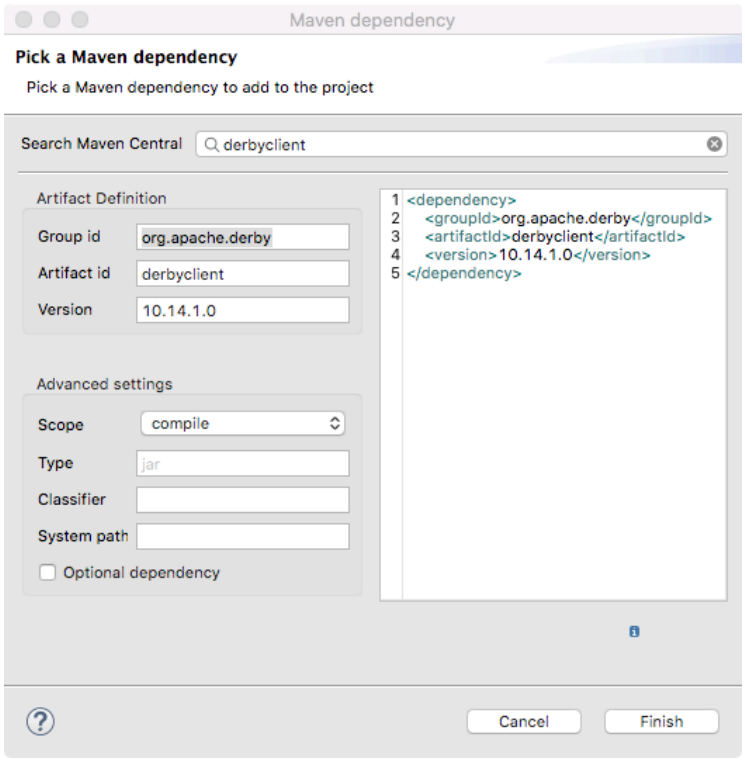
29. In the Maven dependency dialog box, locate the Search Maven Central text field.



30. Enter derbyclient in the Search Maven Central text field.
31. Select org.apache.derby:derbyclient in the results that are displayed.

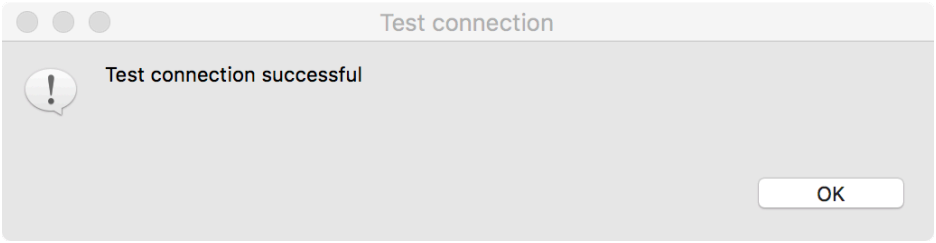


32. Click Edit selected.



33. Click Finish.
34. 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.

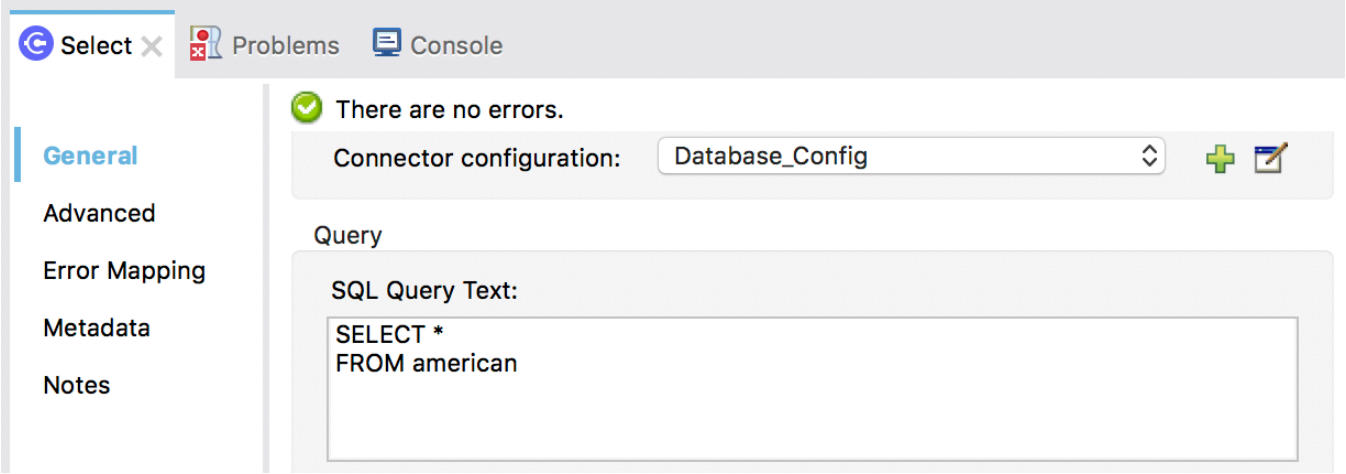


35. Click OK to close the dialog box.
36. Click OK to close the Global Element Properties dialog box.

Write a query to return all flights

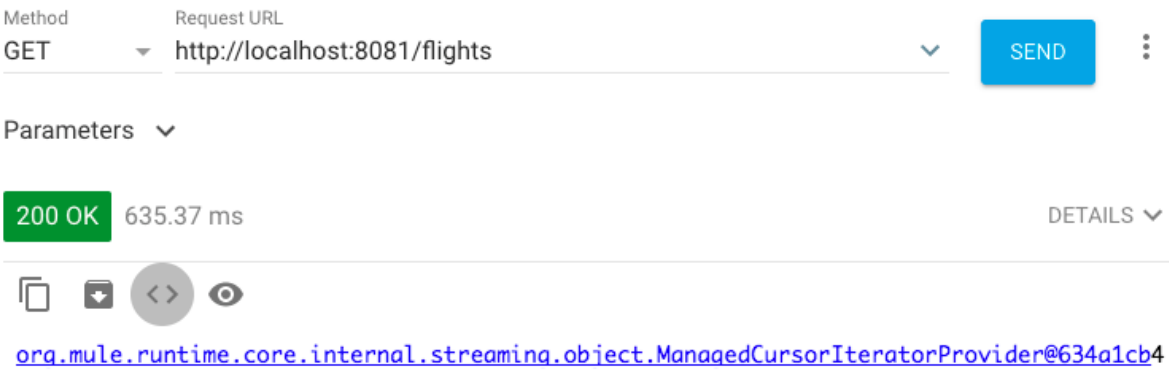
37. In the Select properties view, add a query to select all records from the american table.

```
SELECT *
FROM american
```



Test the application

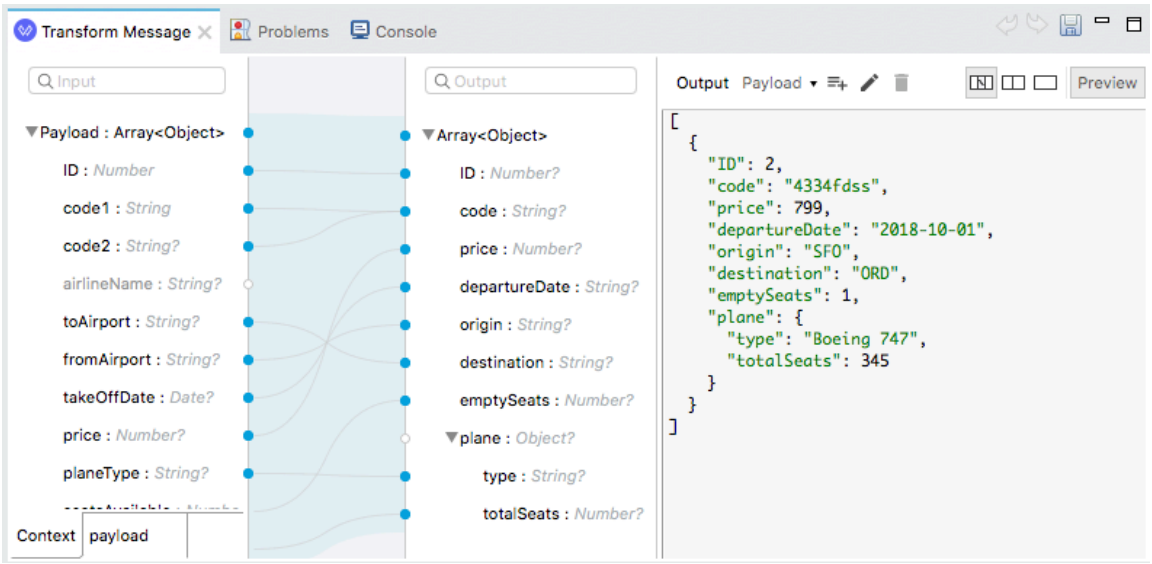
38. Run the project.
39. In the Save changes dialog box, select Yes.
40. Watch the console and wait for the application to start.
41. Once the application has started, return to Advanced REST Client.
42. In Advanced REST Client, make another request to http://localhost:8081/flights; you should get some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



Walkthrough 4-3: Transform data

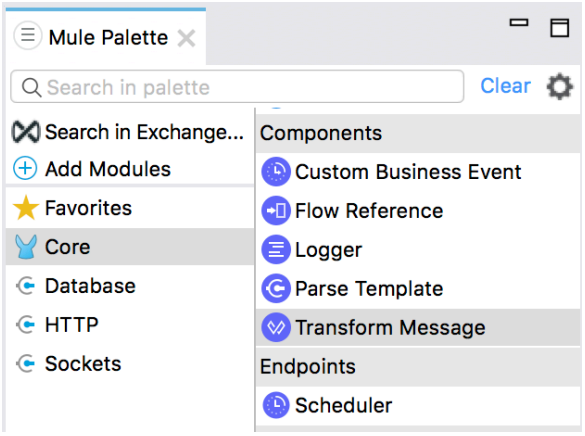
In this walkthrough, you transform and display the flight data into JSON. You will:

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

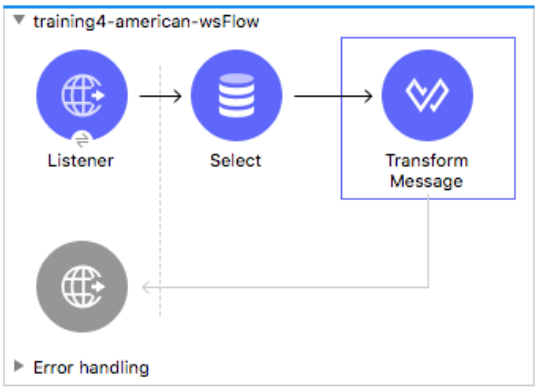


Add a Transform Message component

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



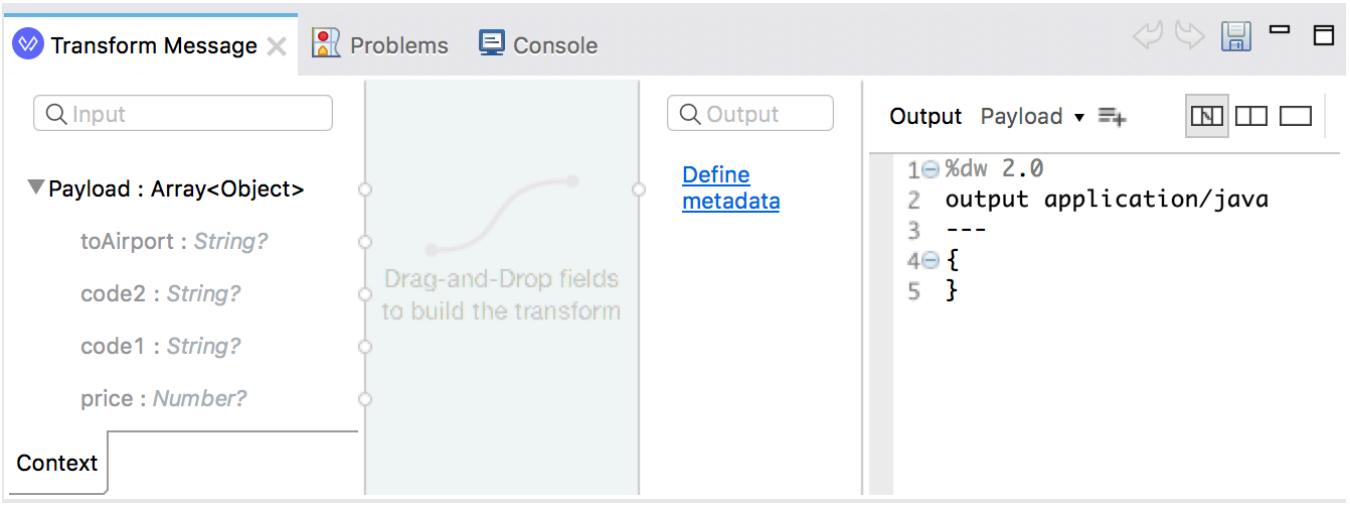
3. Drag the Transform Message and drop it after the Select processor.



Review metadata for the transformation input

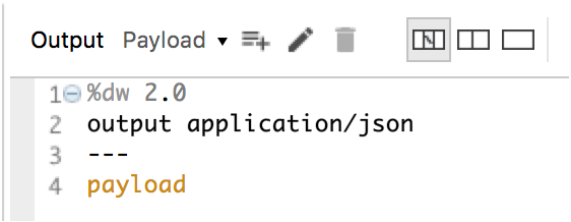
4. In the Transform Message properties view, look at the input section and review the payload metadata.

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



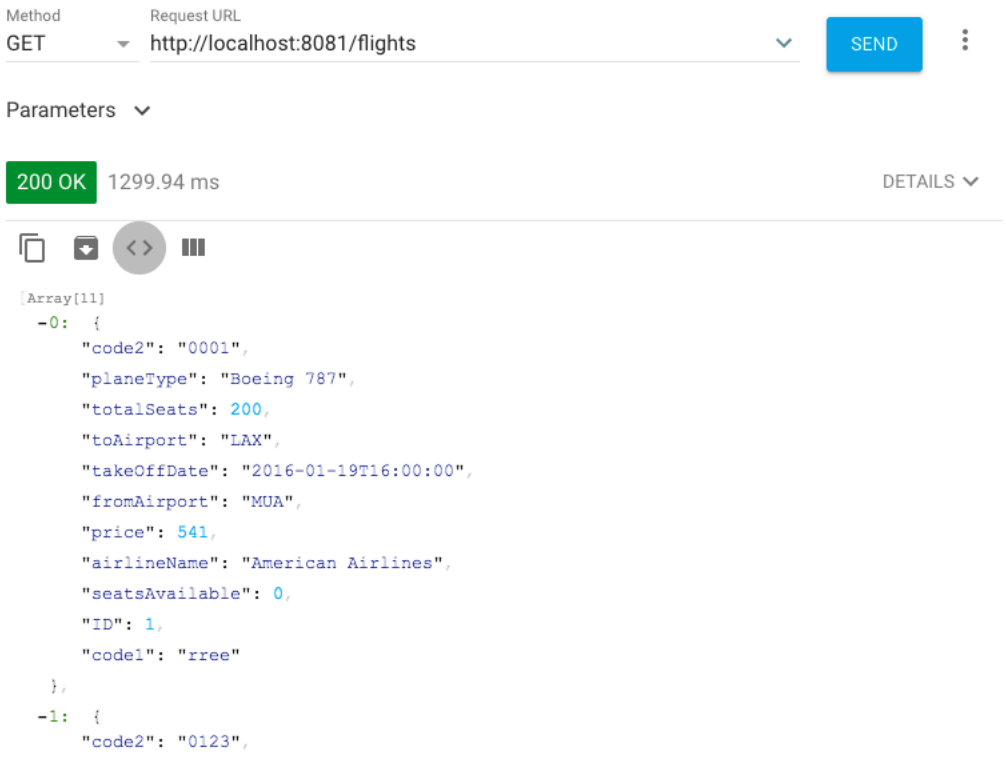
Return the payload as JSON

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



Test the application

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

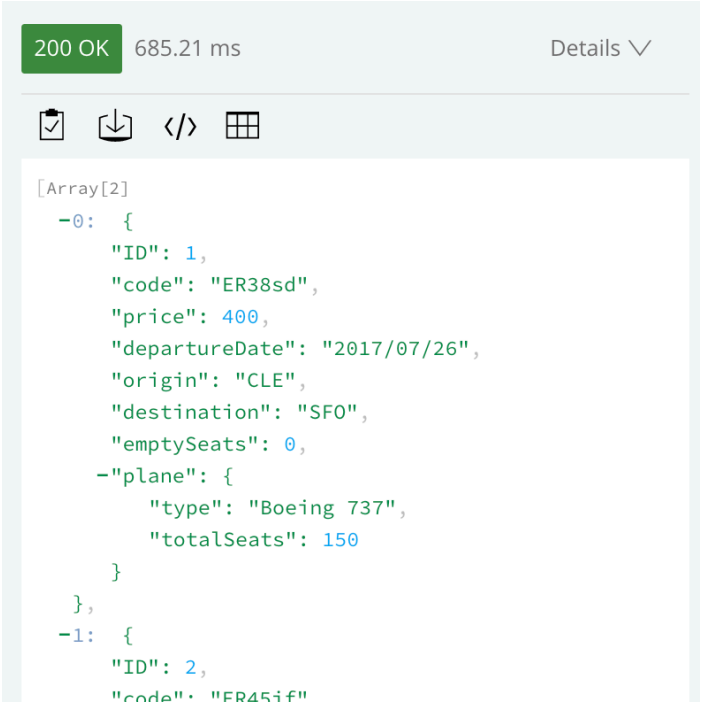


Note: If you are using the local Derby database, the properties will be uppercase instead.

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

8. Return to your American Flights API in Exchange.

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



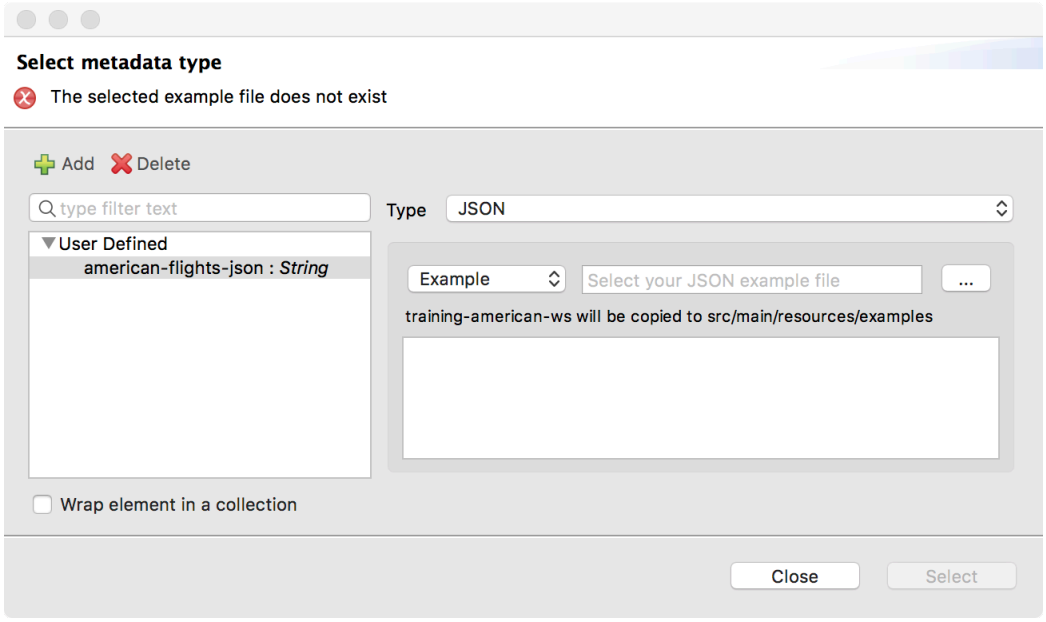
10. Notice that the structure of the JSON being returned by the Mule application does not match this JSON.

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

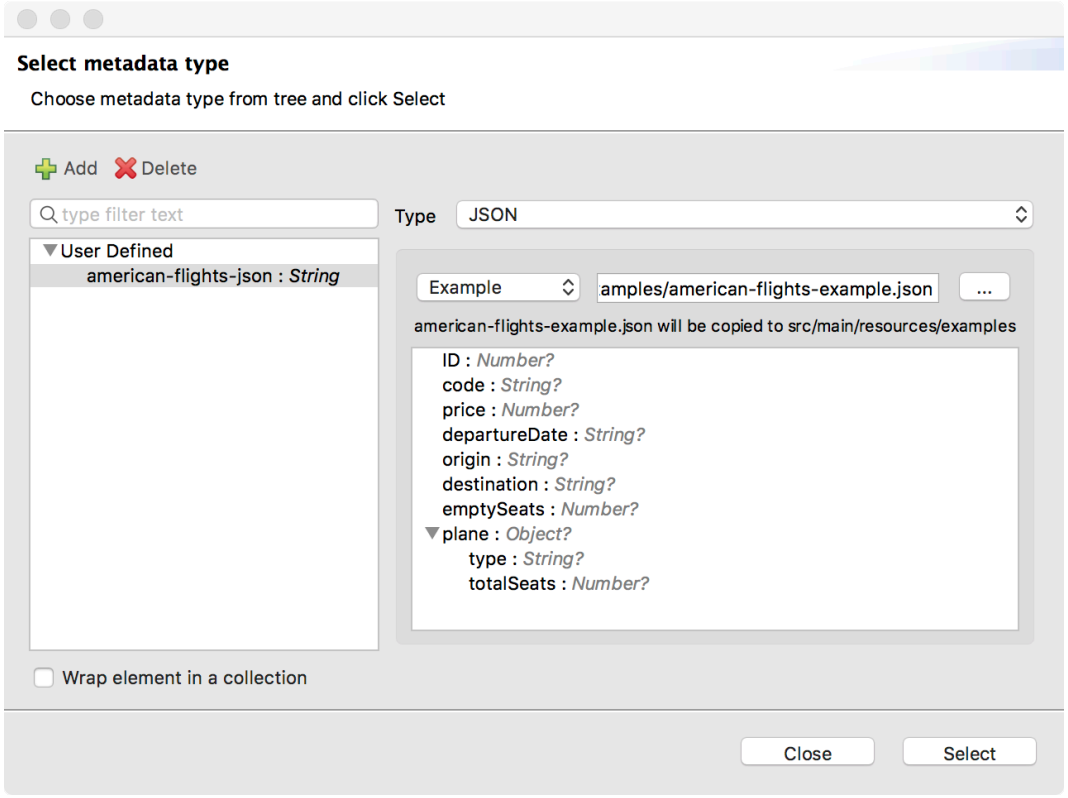
11. Return to Anypoint Studio.
12. In the Transform Message properties view, click the Define metadata link in the output section.

13. In the Select metadata type dialog box, click the Add button.
14. In the Create new type dialog box, set the type id to american_flights_json.
15. Click Create type.

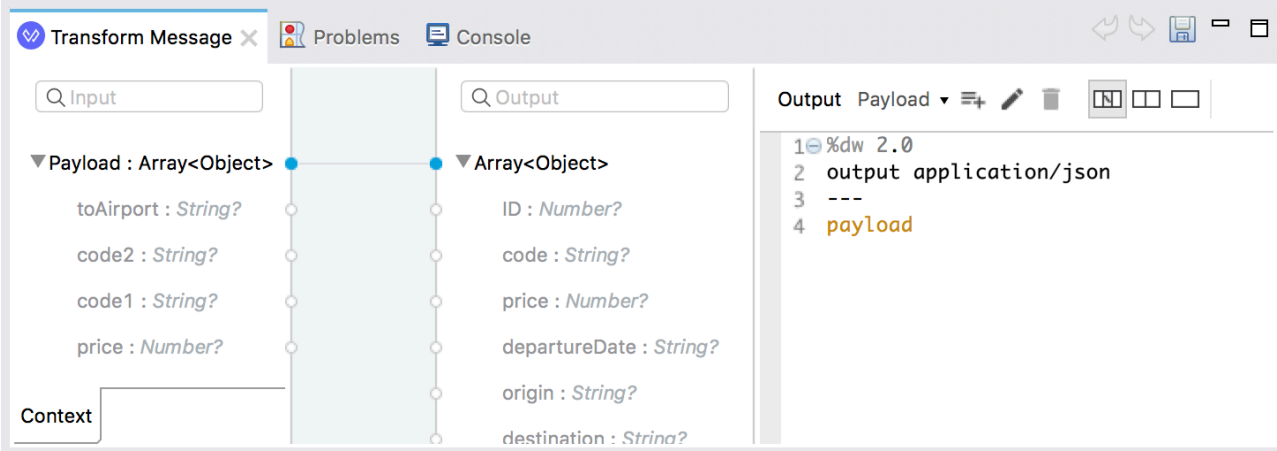
16. Back in the Set metadata type dialog box, set the type to JSON.
17. Change the Schema selection to Example.



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



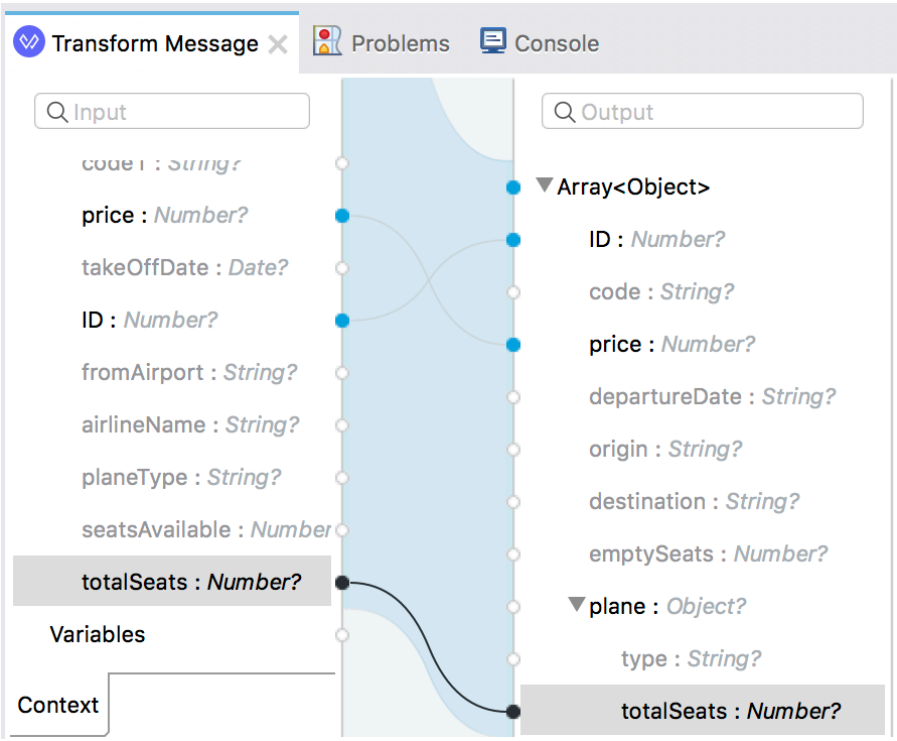
20. Click Select; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

21. 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

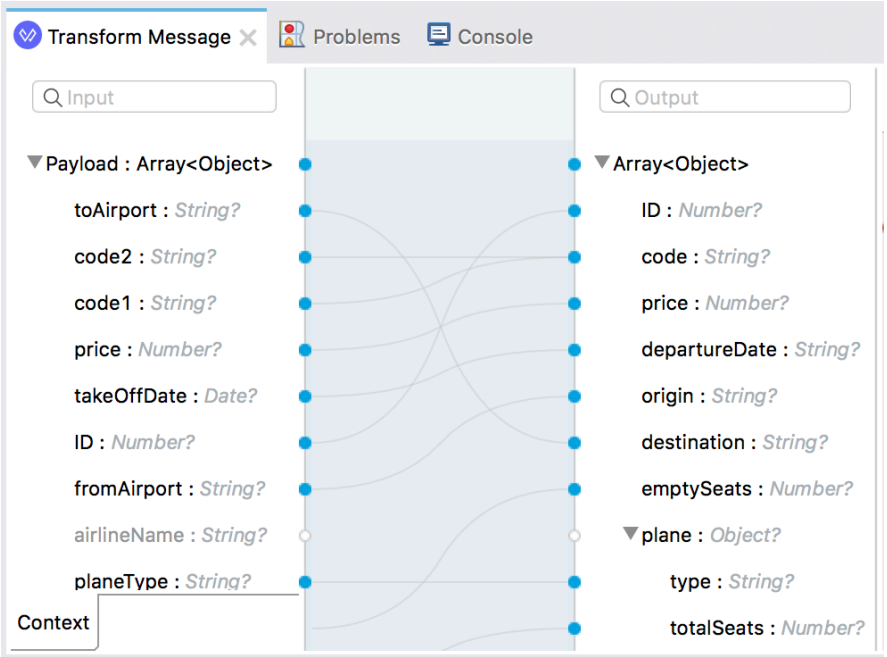


22. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

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

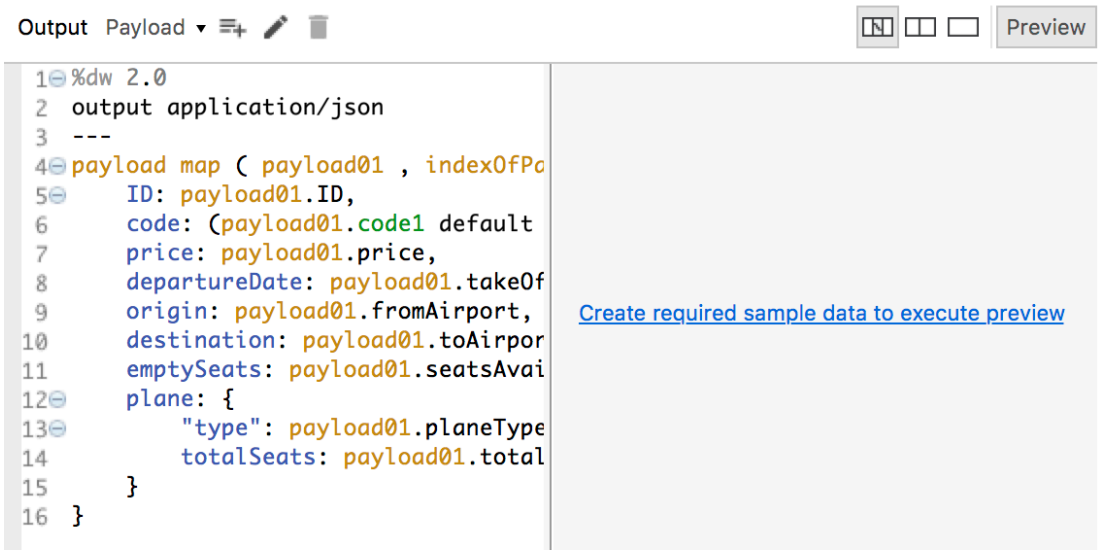
23. Concatenate two fields by dragging them from the input section and dropping them on the same field in the output section.

    - code1 to code
    - code2 to code

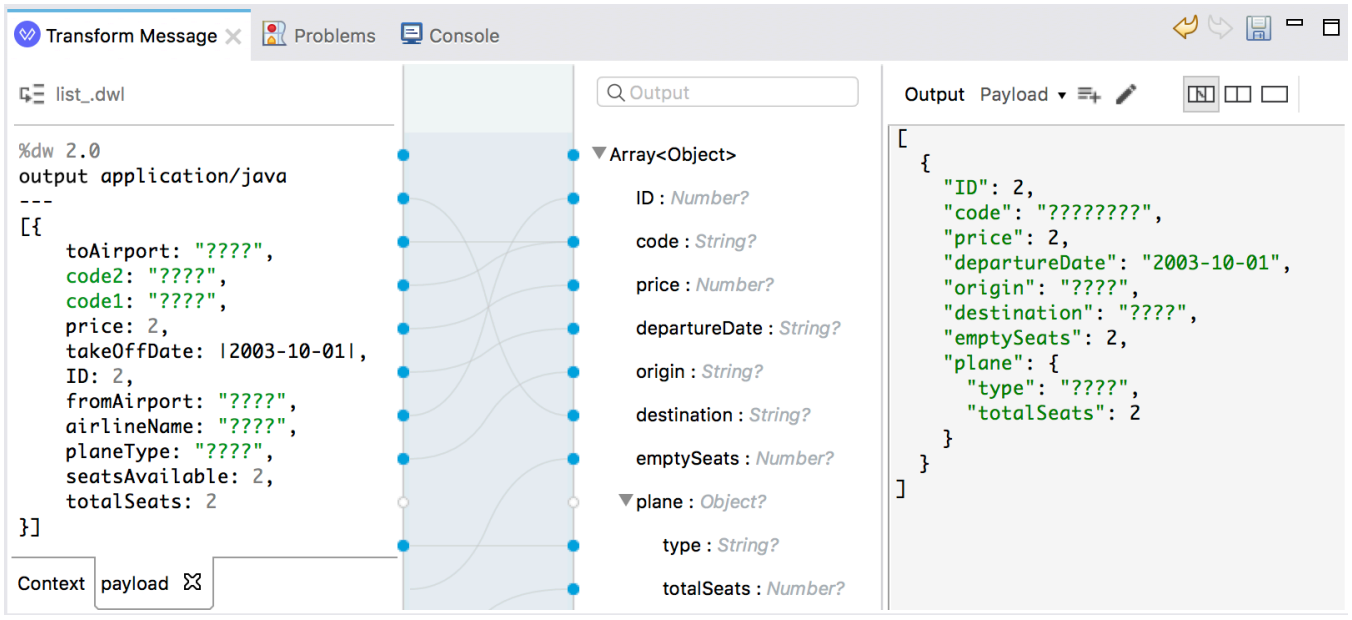


Add sample data (optional)

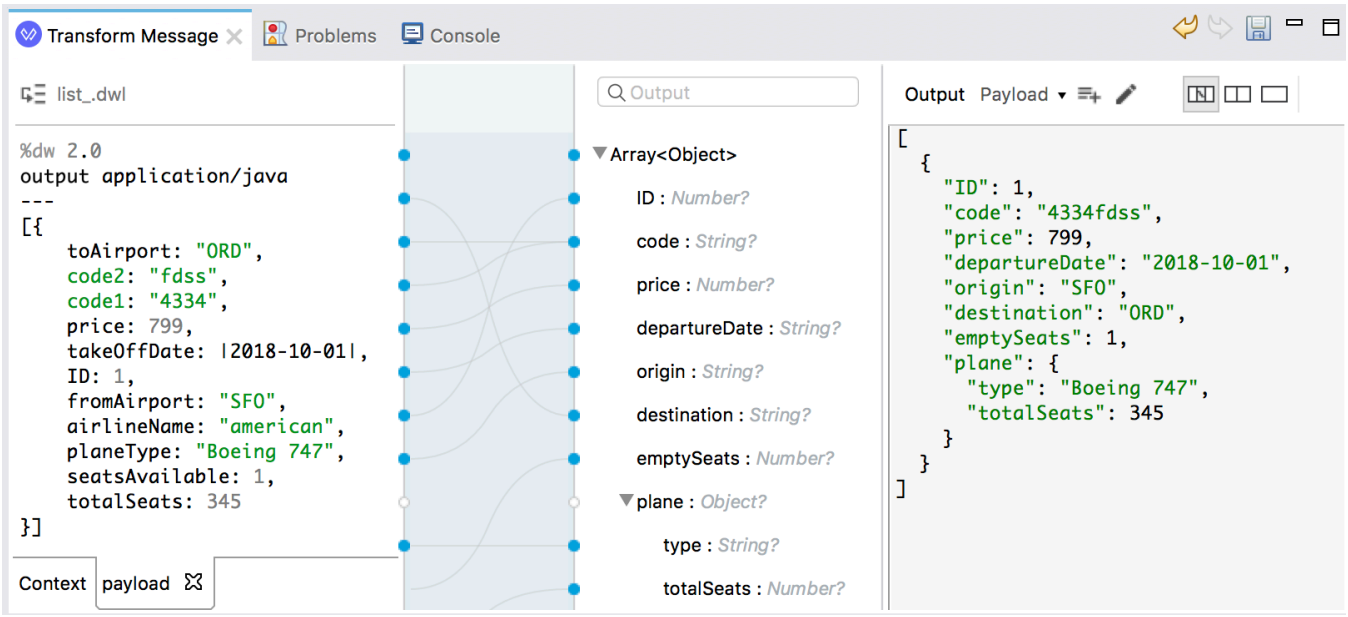
24. Click the Preview button in the output section.
25. In the preview section, click the Create required sample data to execute preview link.



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

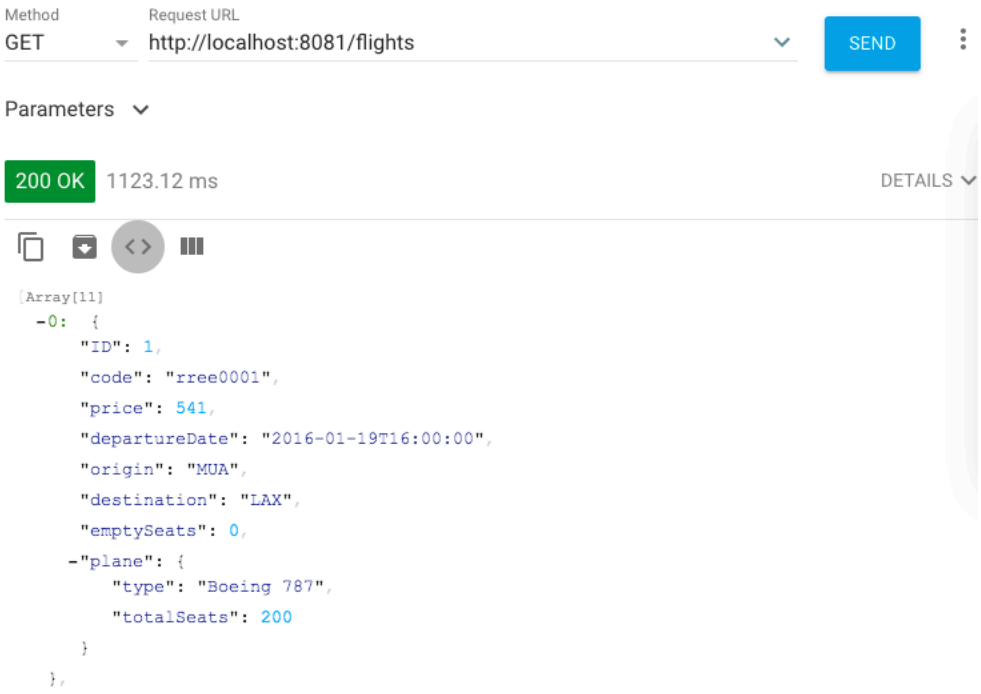


28. In the input section, replace all the ???? with sample values.
29. Look at the output section, you should see the sample values in the transformed data.



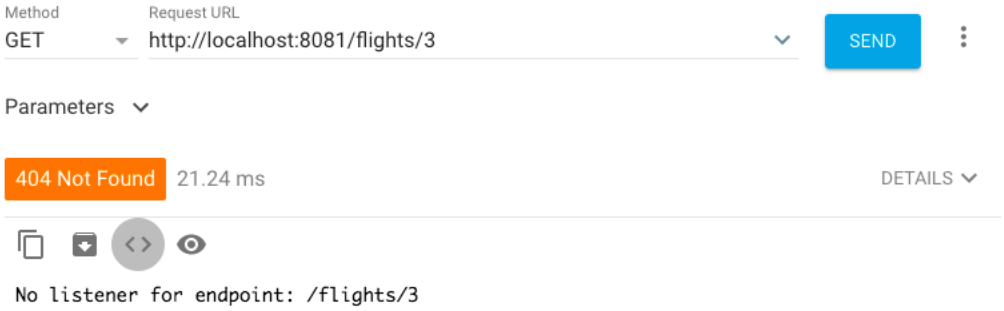
Test the application

30. Save the file to redeploy the project.
31. In Advanced REST Client, make another request to http://localhost:8081/flights; you should see all the flight data as JSON again but now with a different structure.



Try to retrieve information about a specific flight

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

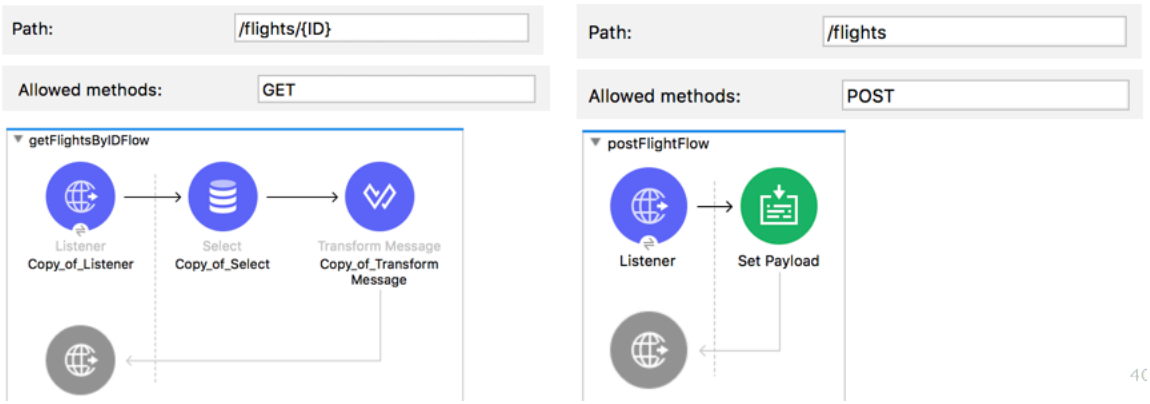


33. Return to Anypoint Studio.
34. Look at the console; you should get a no listener found for request (GET)/flights/3.

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

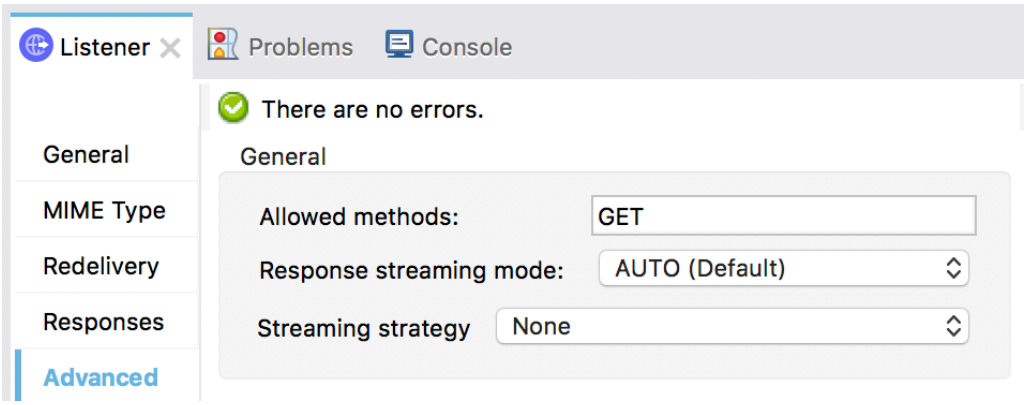
In this walkthrough, you continue to create a RESTful interface for the application. You will:

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



Restrict method calls to GET

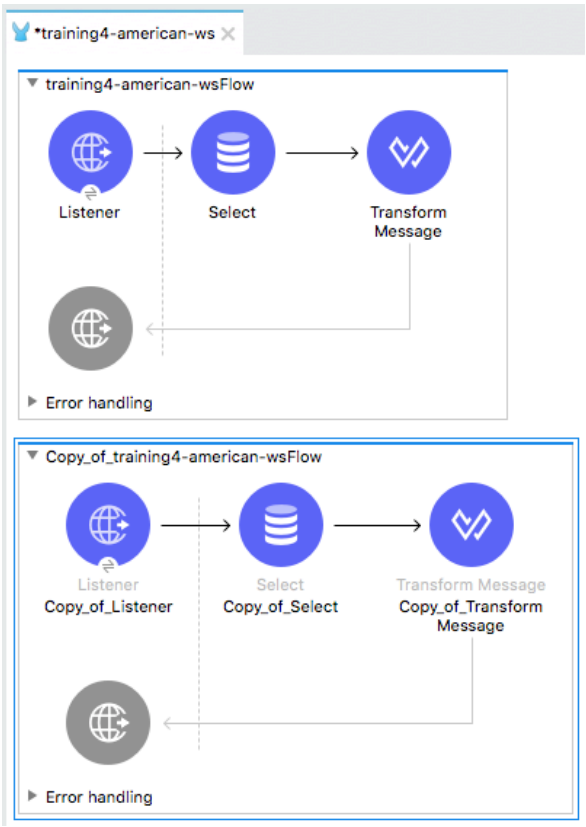
1. Return to Anypoint Studio.
2. Double-click the HTTP Listener in the flow.
3. In the left-side navigation of the Listener properties view, select Advanced.
4. Set the allowed methods to GET.



Make a copy of the existing flow

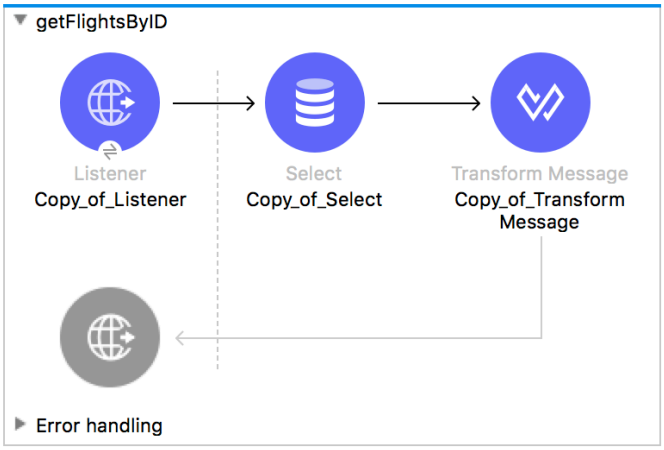
5. Click the flow in the canvas to select it.
6. From the main menu bar, select Edit > Copy.

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



Rename the flows

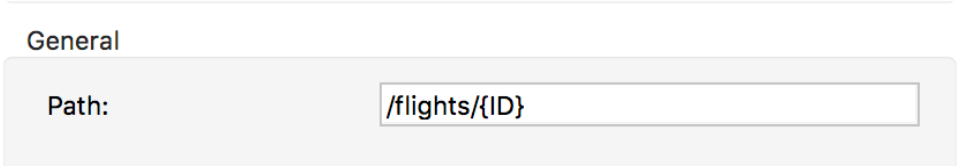
8. Double-click the first flow.
9. In the properties view, change its name to getFlights.
10. Change the name of the second flow to getFlightsByID.



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

Specify a URI parameter for the new HTTP Listener endpoint

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



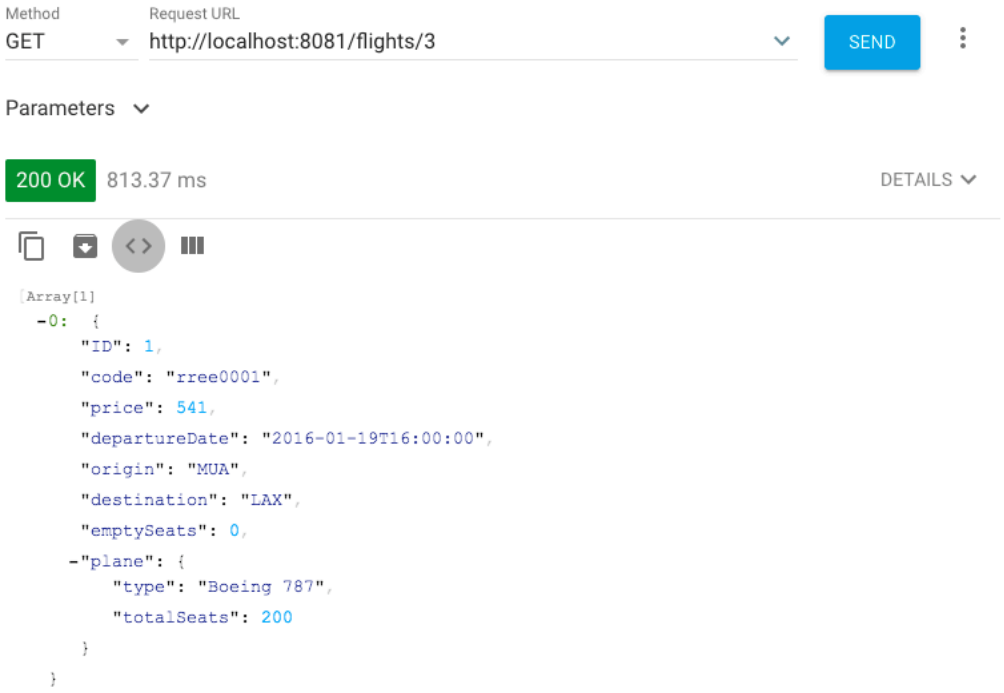
Modify the Database endpoint

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

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

Test the application

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



Modify the database query to use the URI parameter

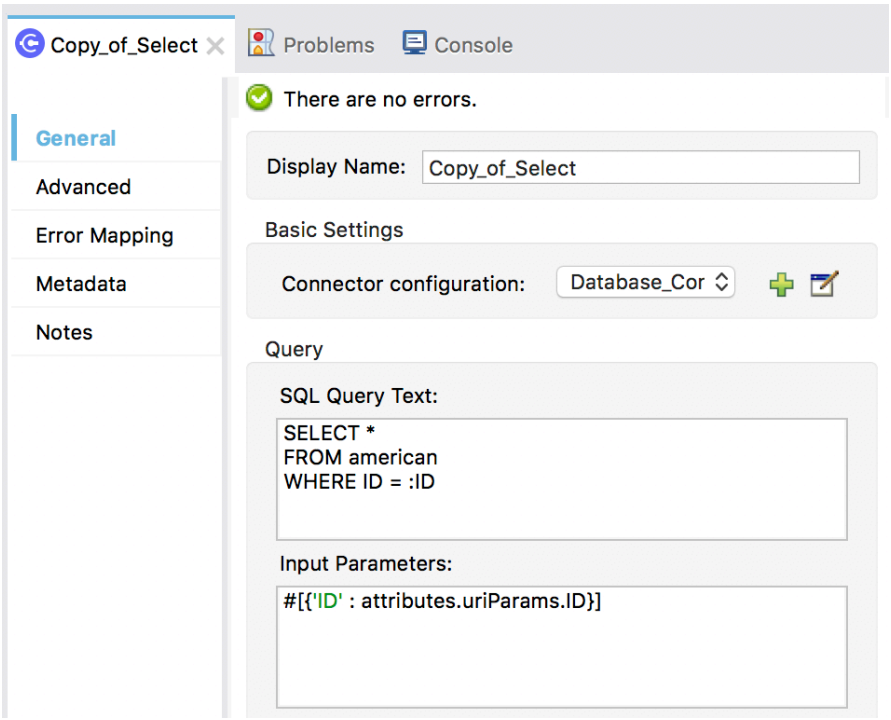
17. Return to the course snippets.txt file and copy the SQL input parameter expression.
18. Return to the getFlightsByID flow in Anypoint Studio.
19. In the Select properties view, locate the Query Input Parameters section and paste the expression you copied.

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

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

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

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



Test the application

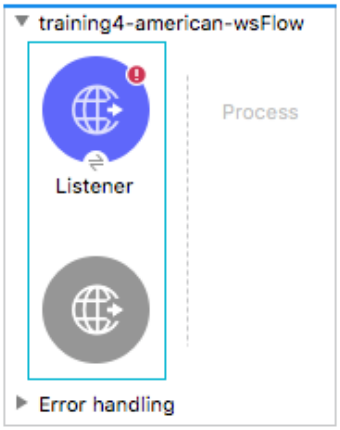
21. Save the file to redeploy the project.

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

23. Return to Anypoint Studio.

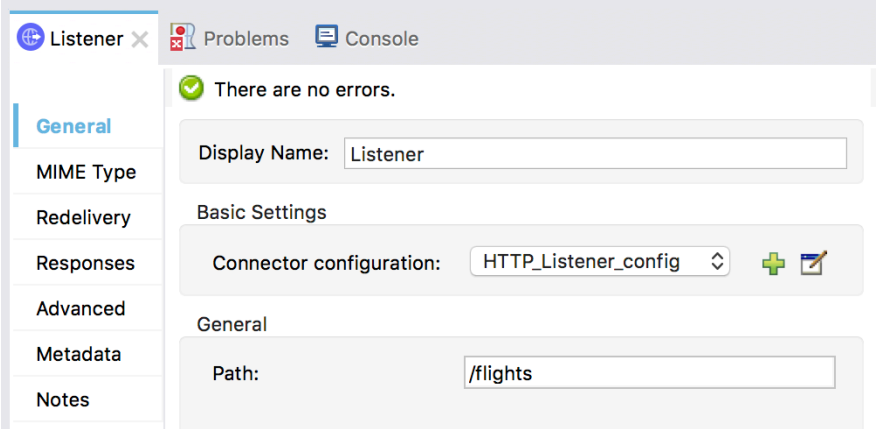
Make a new flow to handle post requests

24. In the Mule Palette, select HTTP.
25. Drag Listener from the Mule Palette and drop it in the canvas below the two existing flows.

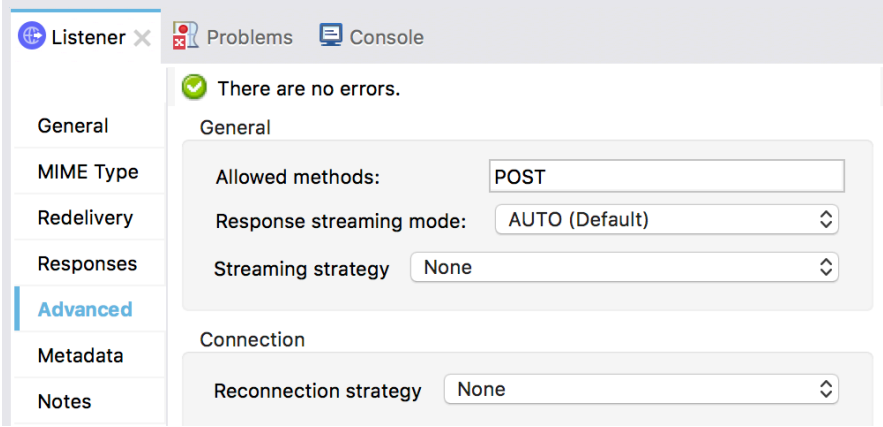


26. Change the name of the flow to postFlight.

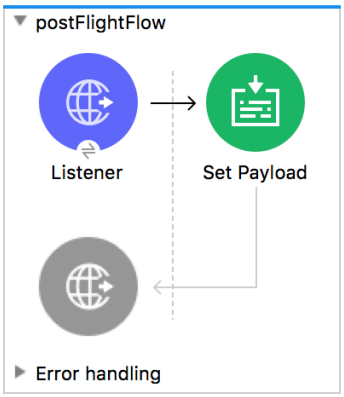
27. In the Listener properties view, set the connector configuration to the existing HTTP_Listener_config.
28. Set the path to /flights.



29. In the left-side navigation of the Listener properties view, select Advanced.
30. Set the allowed methods to POST.



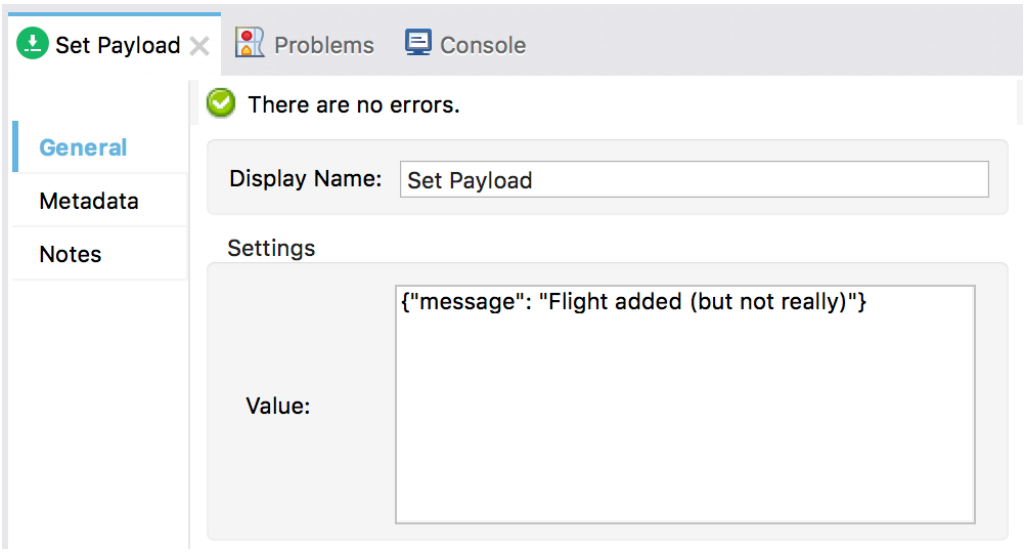
31. Drag the Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



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

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

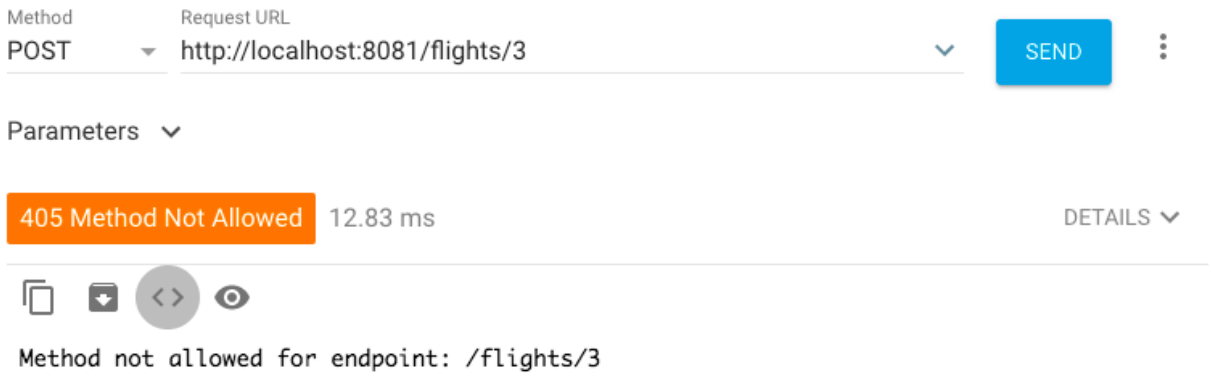
33. Return to Anypoint Studio and in the Set Payload properties view, set value to the value you copied.



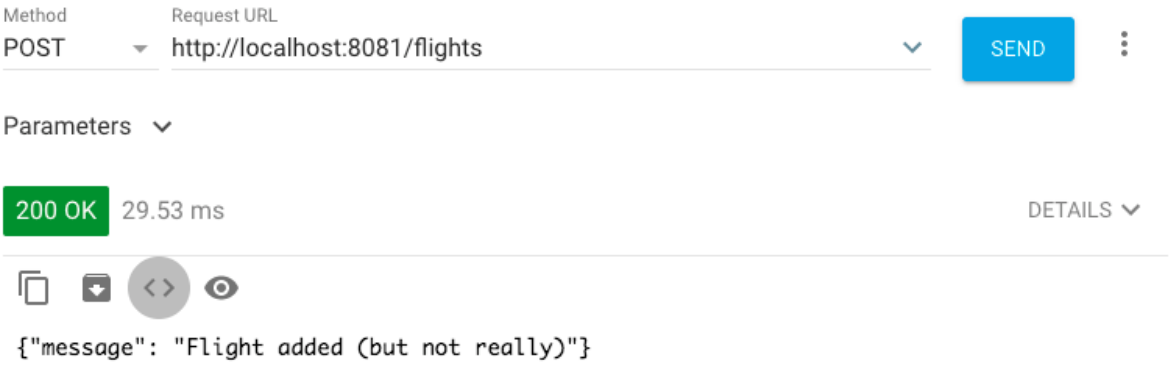
Note: This flow is just a stub. For it to really work and add data to the database, you would need to add logic to insert the request data to the database.

Test the application

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



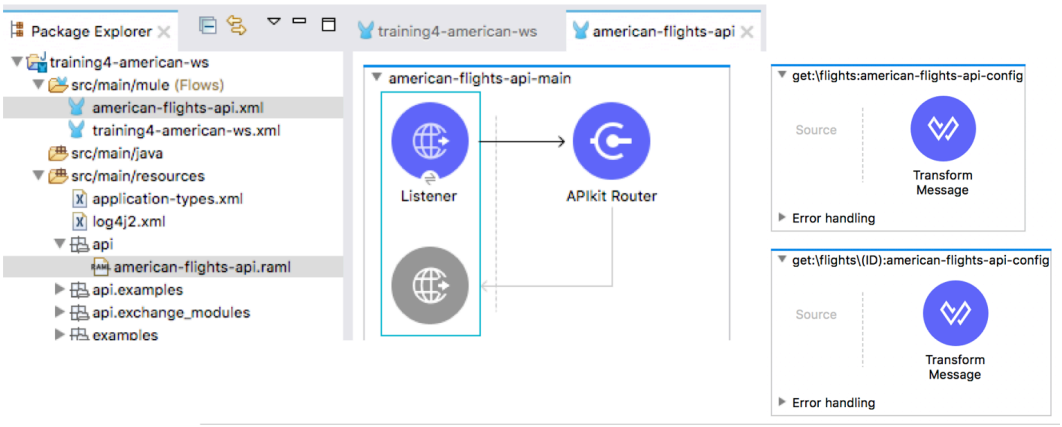
37. Remove the URI parameter from the request URL: http://localhost:8081/flights.
38. Send the request; you should now see the message the flight was added – even though you did not send any flight data to add.



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

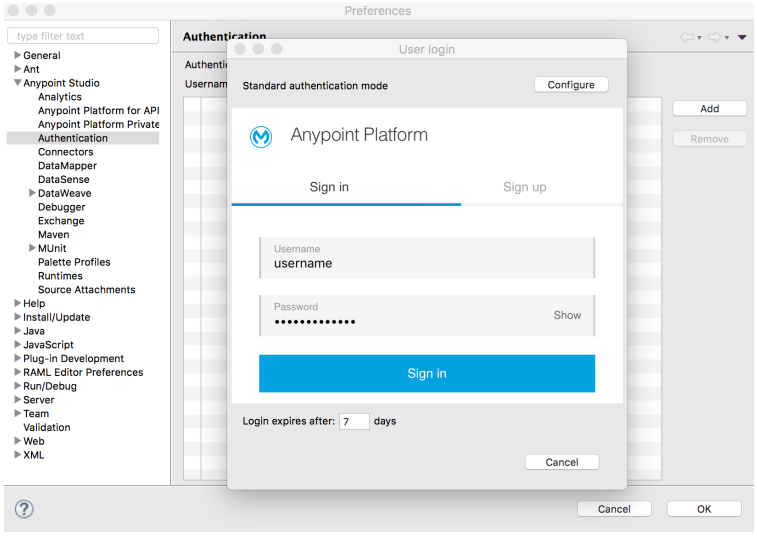
In this walkthrough, you generate a RESTful interface from the RAML file. You will:

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



Add Anypoint Platform credentials to Anypoint Studio

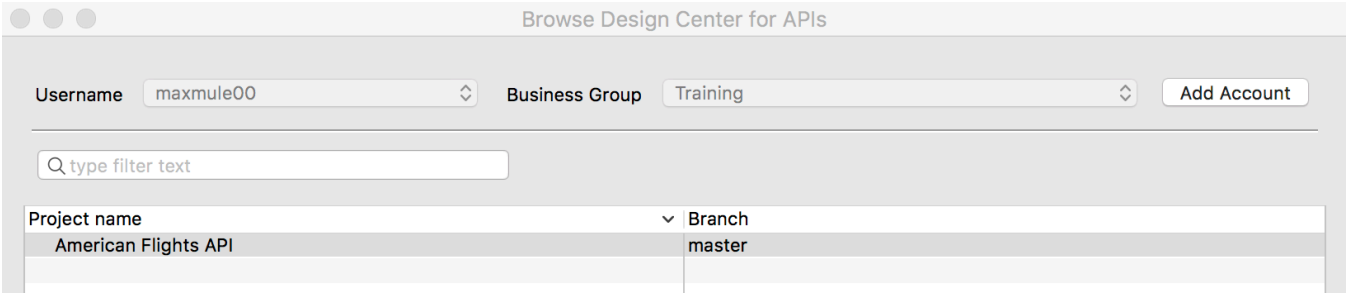
1. In Anypoint Studio, right-click training4-american-ws and select Anypoint Platform > Configure Credentials.
2. In the Authentication page of the Preferences dialog box, click the Add button.
3. In the Anypoint Platform Sign In dialog box, enter your username & password and click Sign In.



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

Add an API from Design Center to the Anypoint Studio project

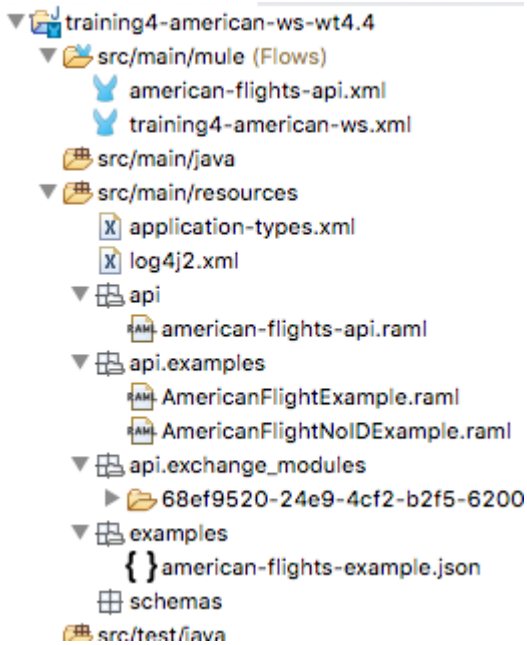
6. In the Package Explorer, locate the src/main/resources/api folder; it should not contain any files.
7. Right-click the folder (or anywhere in the project in the Package Explorer) and select Anypoint Platform > Import from Design Center.
8. In the Browse Design Center for APIs dialog box, select the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



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

Locate the API files added to the project

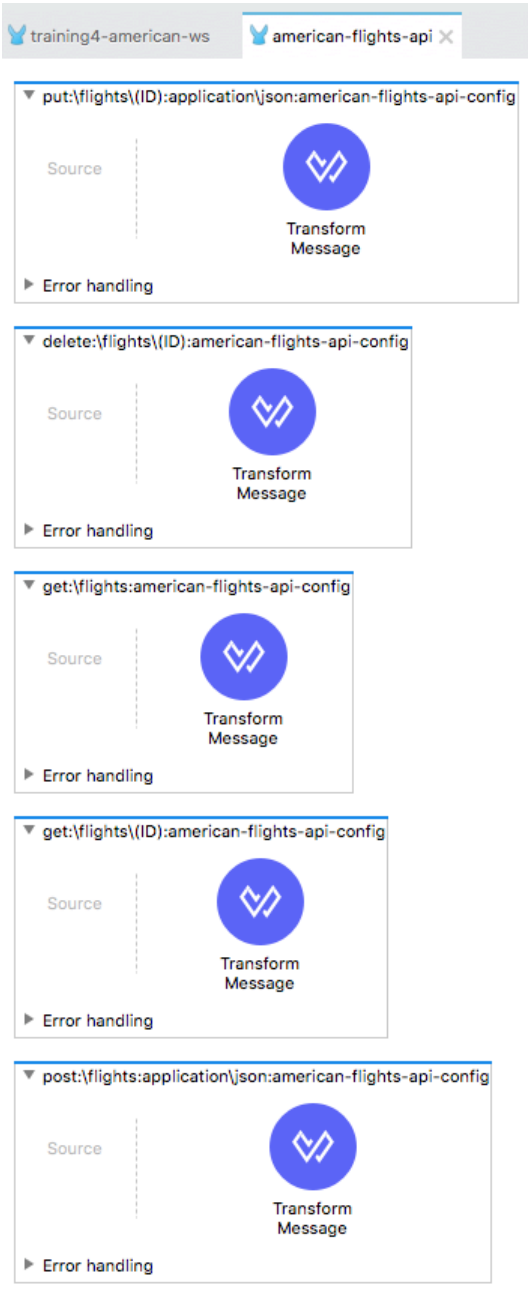
10. In the Package Explorer, locate and expand the src/main/resources folder; it should now contain api folders.
11. Expand the api folder; you should see the RAML file imported from Design Center.



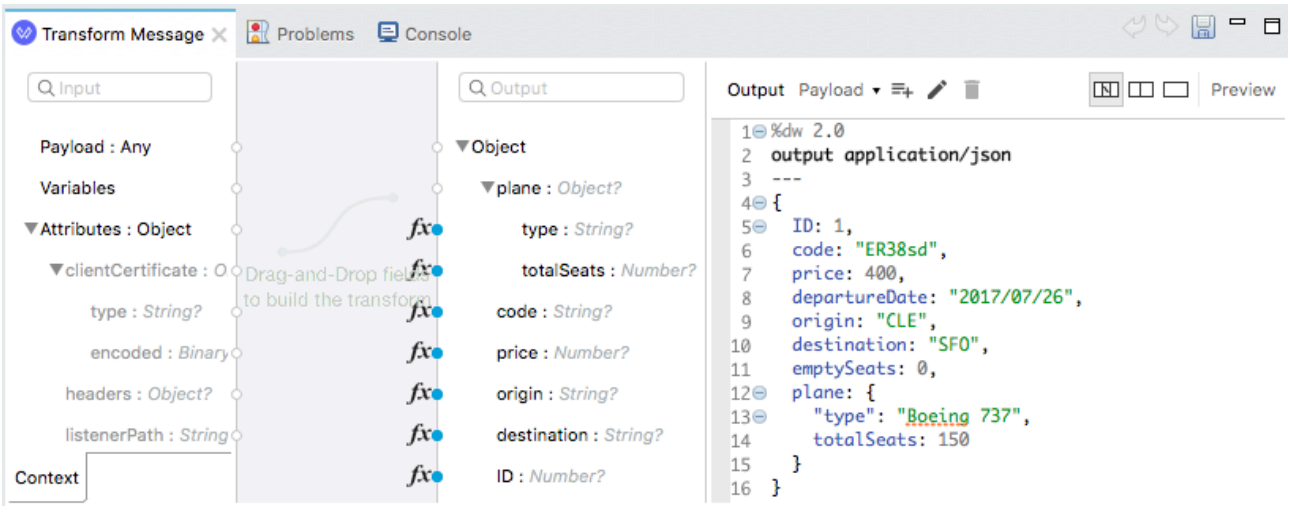
Examine the XML file created

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

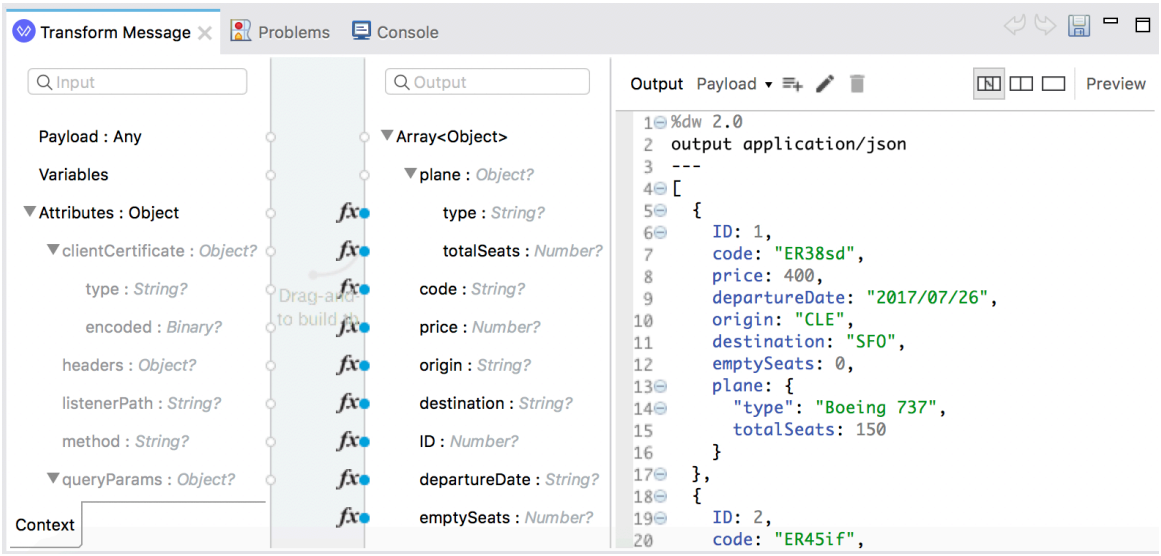
- get:/flights
- get:/flights/{ID}
- post:/flights
- delete:/flights/{ID}
- put:/flights/{ID}



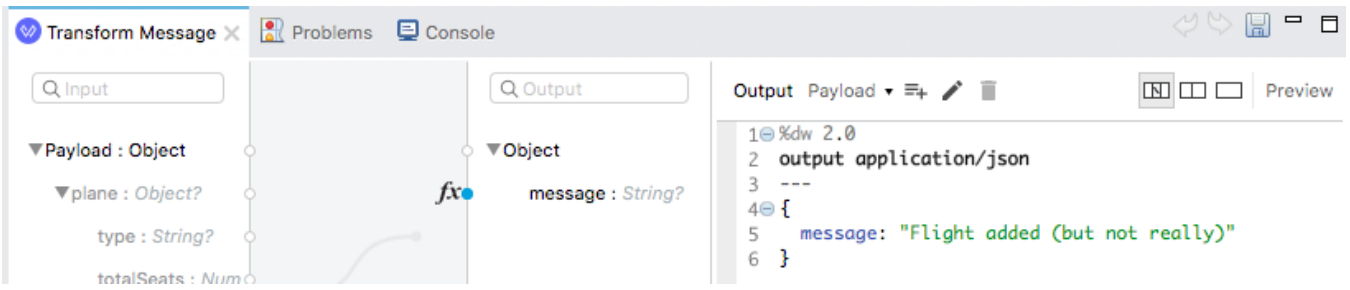
13. In the get:/flights/{ID} flow, double-click the Transform Message component and look at the value in the properties view.



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

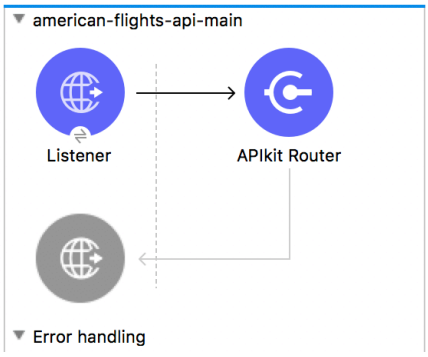


15. In the post:/flights flow, double-click the Set Payload transformer and look at the value in the Set Payload properties view.



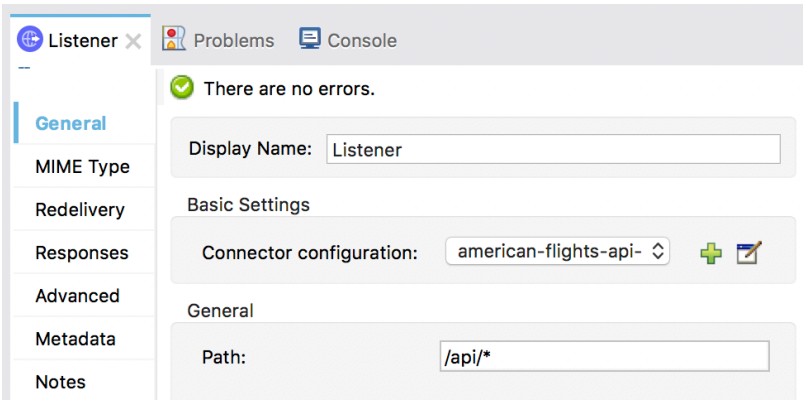
Examine the main flow and the new HTTP Listener endpoint

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



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

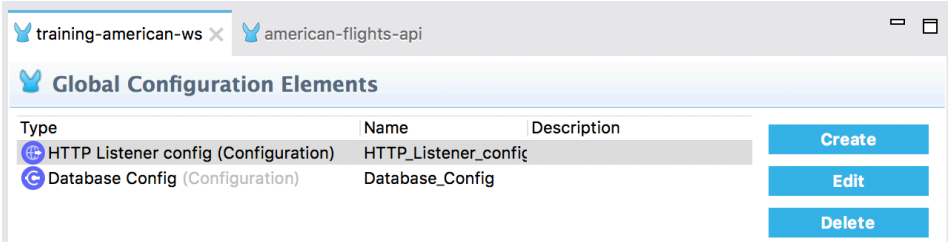
    *Note: The * is a wildcard allowing any characters to be entered after /api/.*



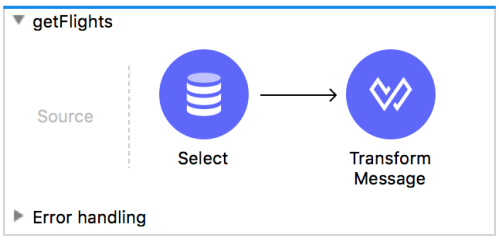
19. Click the Edit button for the connector configuration; you should see that the same port 8081 is used as the HTTP Listener you created previously.
20. Click OK.

Remove the other HTTP configuration and listeners

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



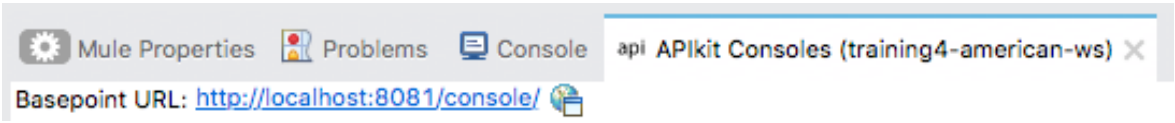
23. Return to the Message Flow view.
24. Right-click the HTTP Listener in getFlights and select Delete.



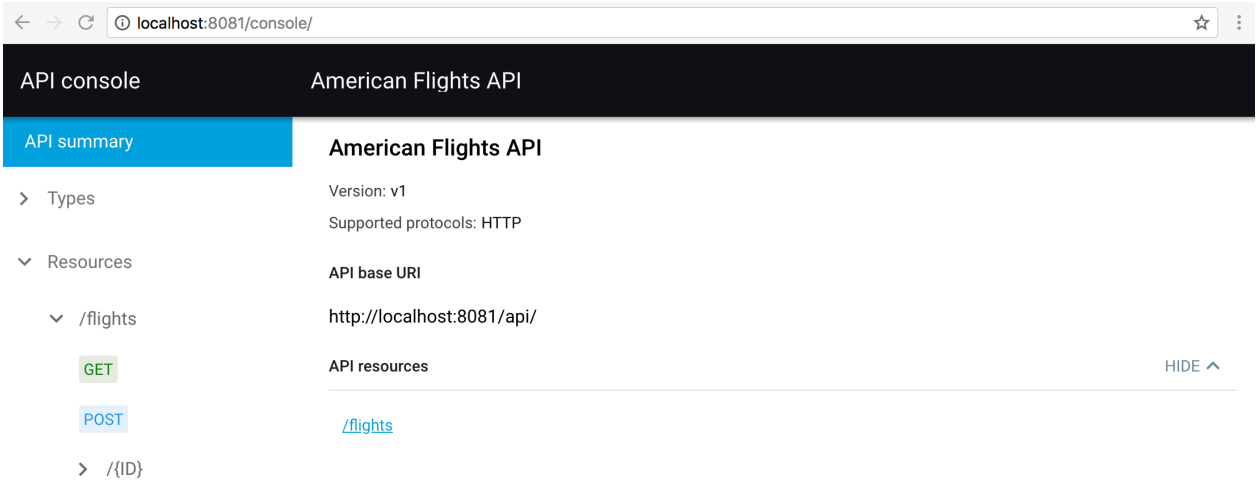
25. Delete the other two HTTP Listeners.

Test the web service using APIkit console

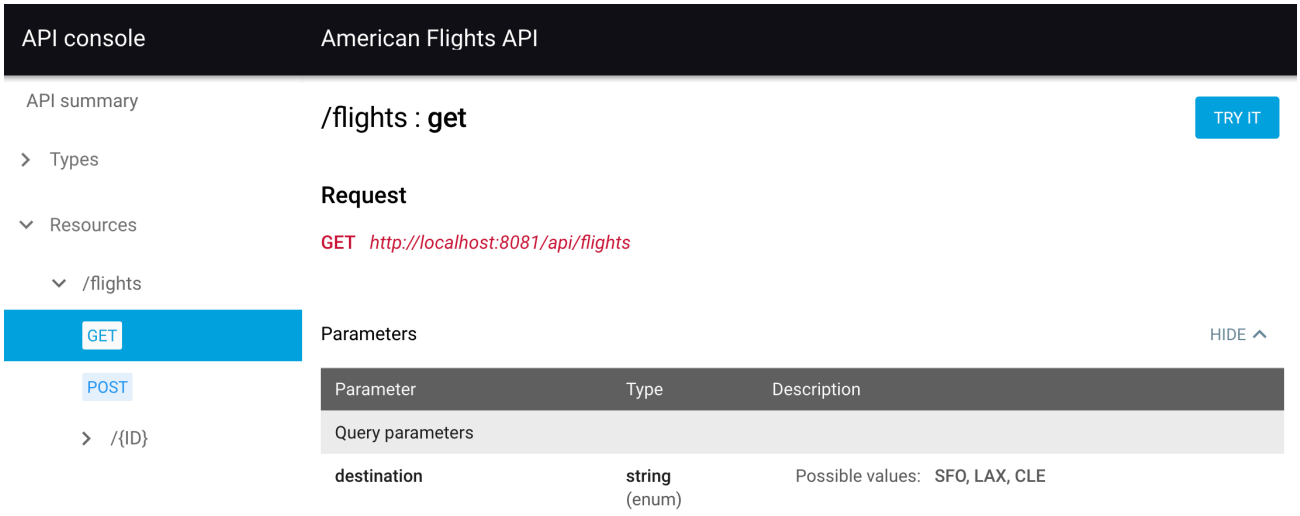
26. Save the files.
27. Look at the console; you should see the application was not redeplpoyed.
28. Stop the project.
29. Run the project and wait until Mule and the application restart.
30. Locate the new APIkit Consoles view that is created and opened in Anypoint Studio.



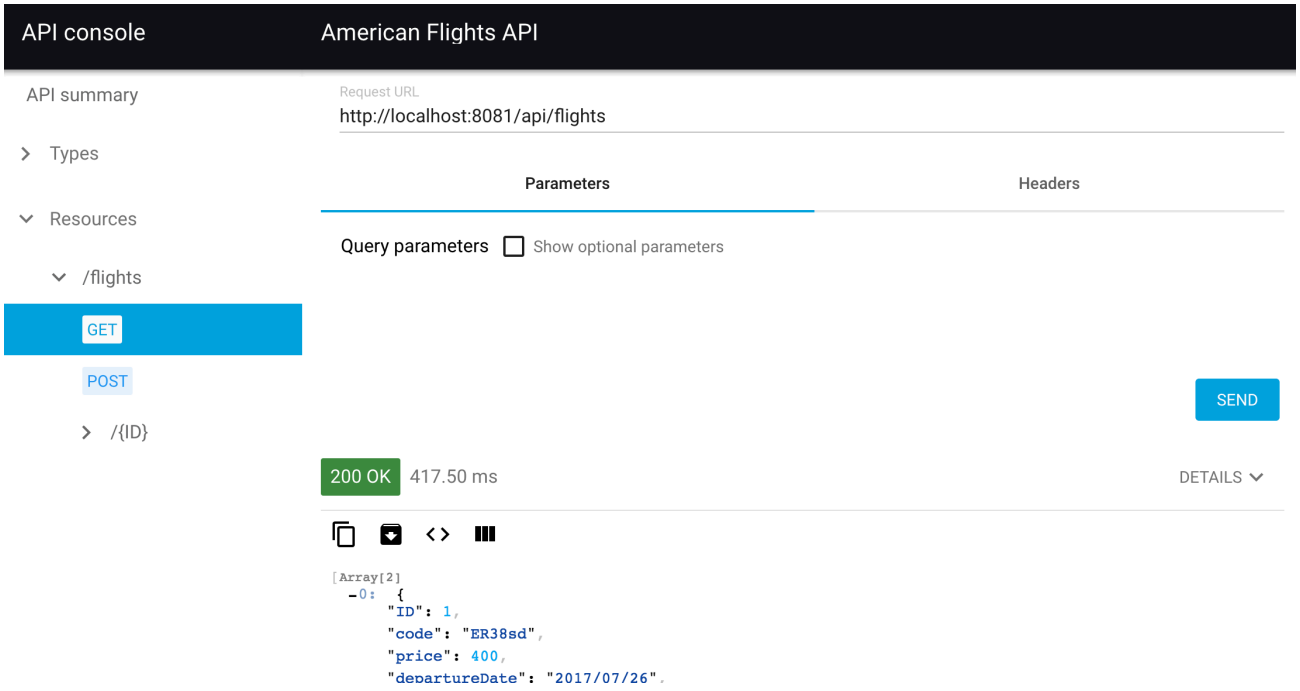
31. Click the console link; a browser window should be open with an API console.



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



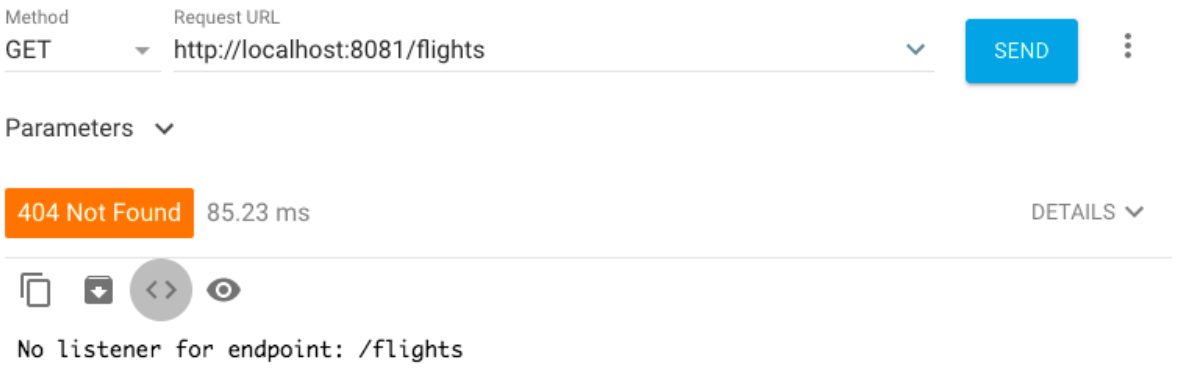
33. Click Send; you should get a 200 response with the example flight data – not all the flights.



34. Close the browser window.

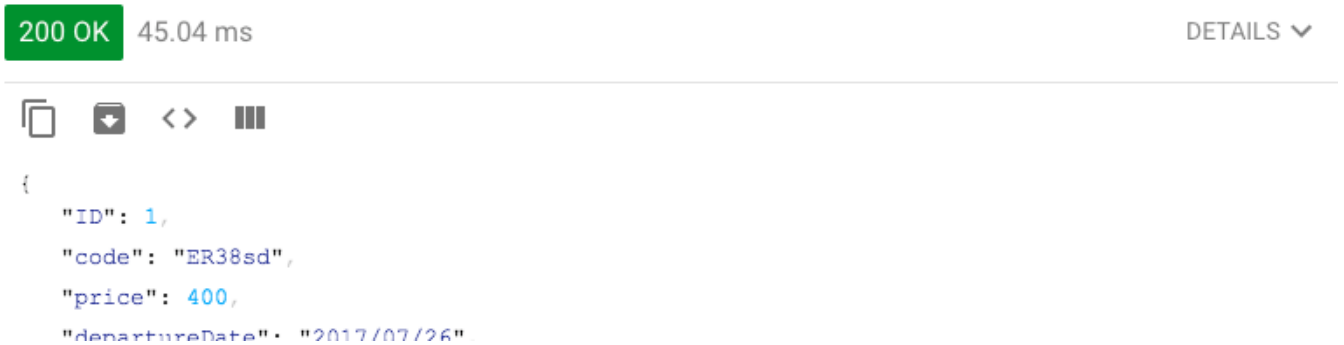
Test the web service using Advanced REST Client

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



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

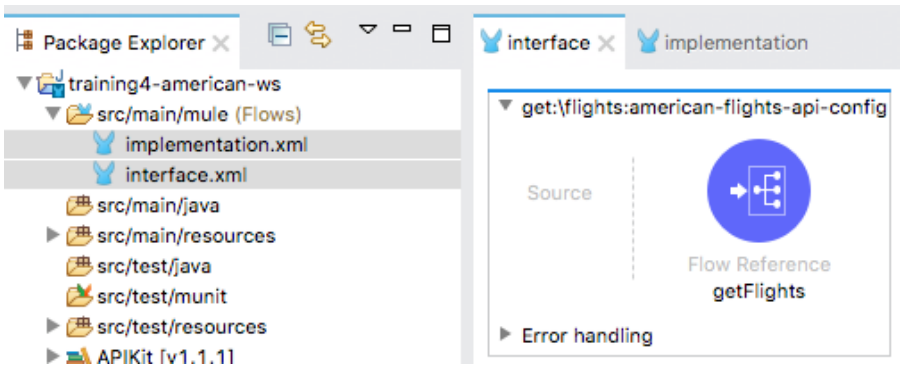
38. Make a request to http://localhost:8081/api/flights/3; you should see the example data returned for a flight with an ID of 1.



Walkthrough 4-6: Implement a RESTful web service

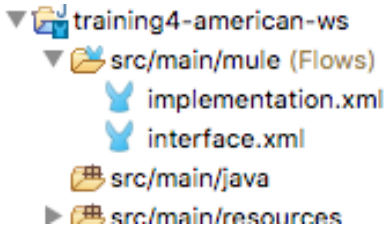
In this walkthrough, you wire the RESTful web service interface up to your back-end logic. You will:

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



Rename the configuration files

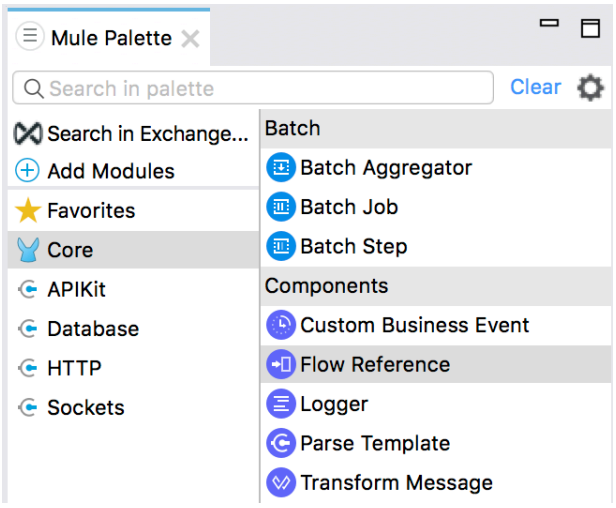
1. Return to Anypoint Studio.
2. Right-click american-flights-api.xml in the Package Explorer and select Refactor > Rename.
3. In the Rename Resource dialog box, set the new name to interface.xml and click OK.
4. Right-click training4-american-ws.xml and select Refactor > Rename.
5. In the Rename Resource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



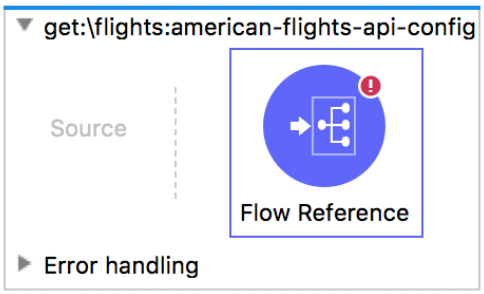
Use a Flow Reference in the /flights resource

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

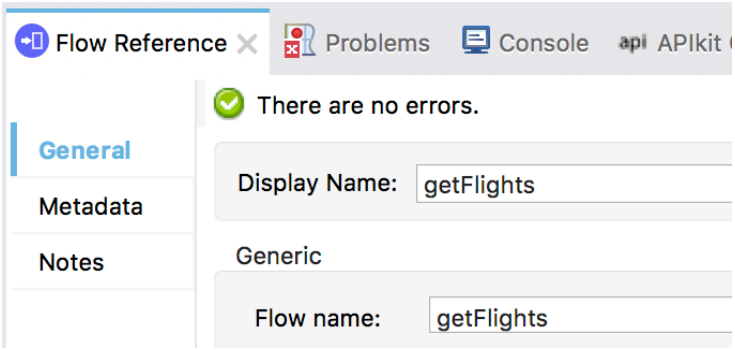
8. In the Mule Palette, select Core and locate the Components section in the right-side.



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



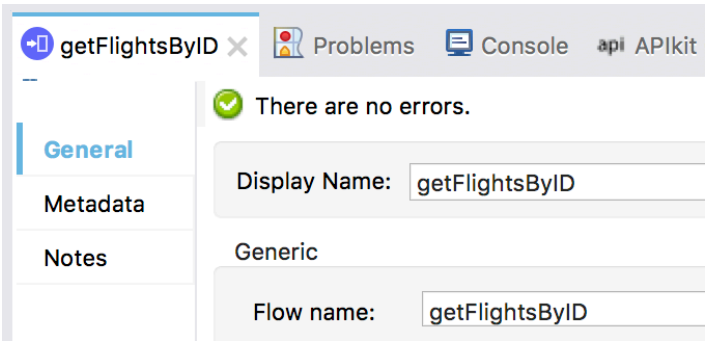
10. In the Flow Reference properties view, select getFlights for the flow name.



Use a Flow Reference in the /flights/{ID} resource

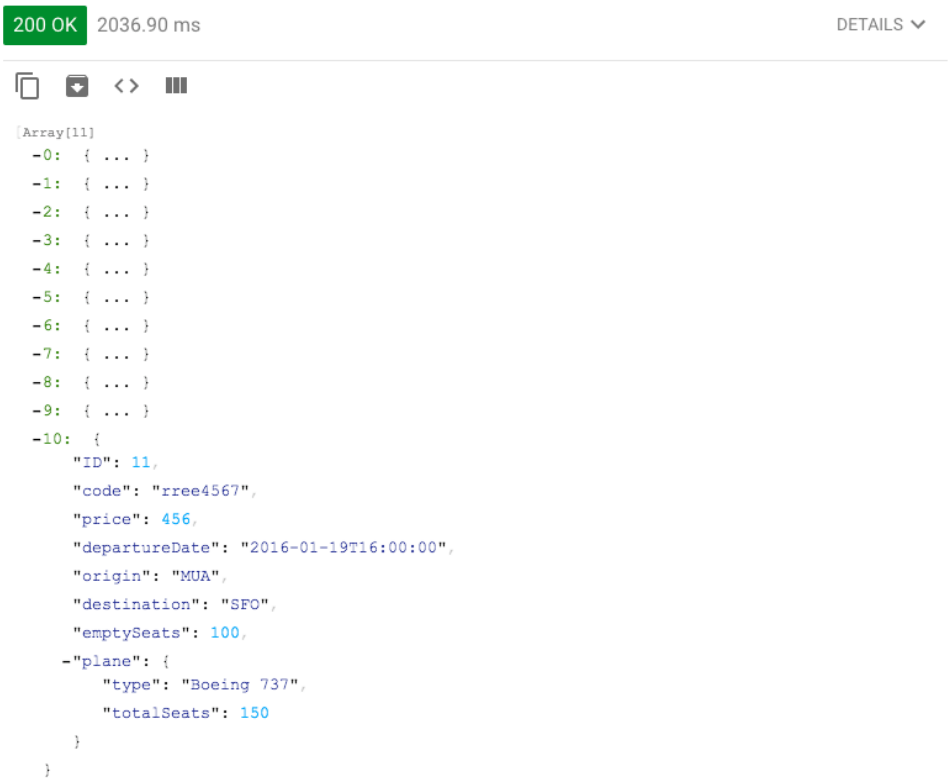
11. Delete the Transform Message component in the get:/flights/{ID} flow.
12. Drag a Flow Reference component from the Mule Palette and drop it into the flow.

13. In the Flow Reference properties view, select getFlightsByID for the flow name.



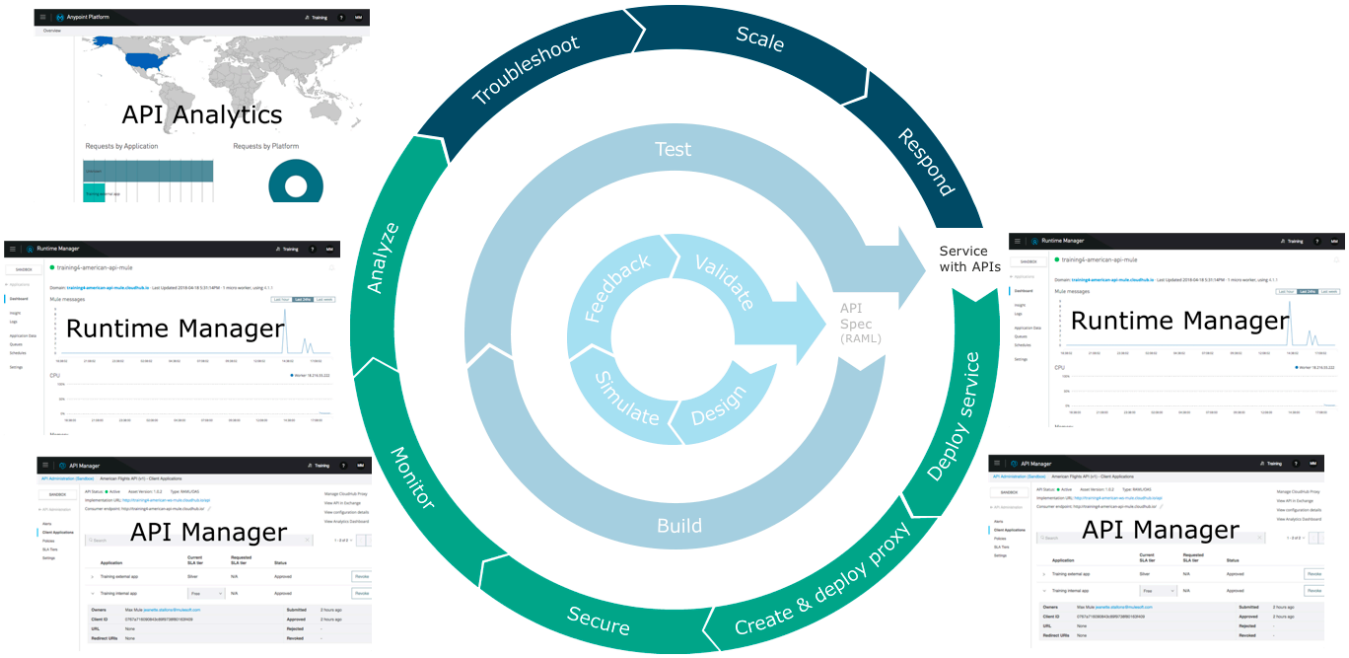
Test the web service using Advanced REST Client

14. Save the file to redeploy the project.
15. In Advanced REST Client, make a request to http://localhost:8081/api/flights; you should now get the data for all the flights from the database instead of the sample data.



16. Make a request to http://localhost:8081/api/flights/3; you should now get the data that flight from the database instead of the sample data.
17. Return to Anypoint Studio.
18. Stop the project.

Module 5: Deploying and Managing APIs



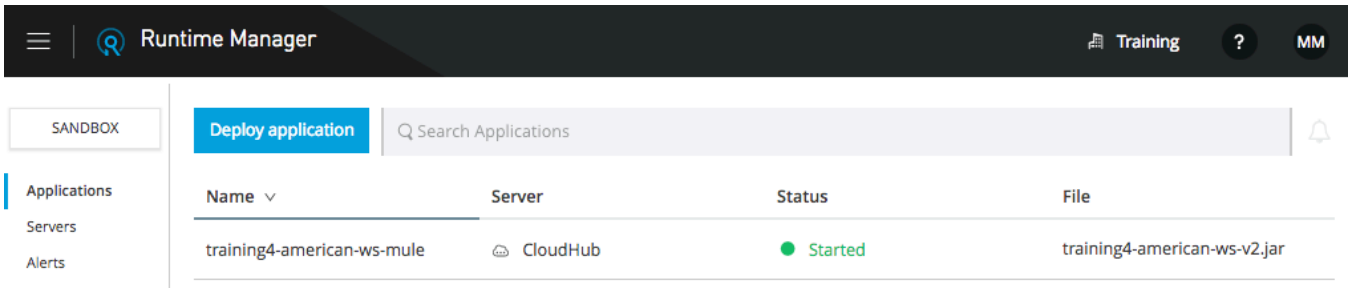
At the end of this module, you should be able to:

- Describe the options for deploying Mule applications.
- Deploy Mule applications to CloudHub.
- Use API Manager to create and deploy API proxies.
- Use API Manager to restrict access to API proxies.

Walkthrough 5-1: Deploy an application to CloudHub

In this walkthrough, you deploy and run your application on CloudHub. You will:

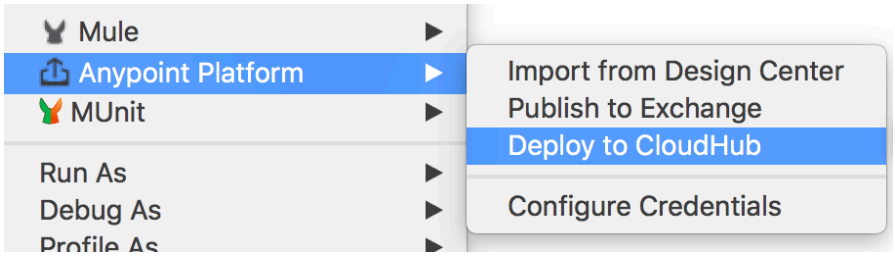
- Deploy an application from Anypoint Studio to CloudHub.
- Run the application on its new, hosted domain.
- Make calls to the web service.
- Update an API implementation deployed to CloudHub.



Note: If you do not have a working application at this point, import the training4-american-ws-wt4-6.jar solution into Anypoint Studio and work with that project.

Deploy the application to CloudHub

1. Return to Anypoint Studio.
2. In the Package Explorer, right-click the project and select Anypoint Platform > Deploy to CloudHub.



3. In the Choose Environment dialog box, select Sandbox.