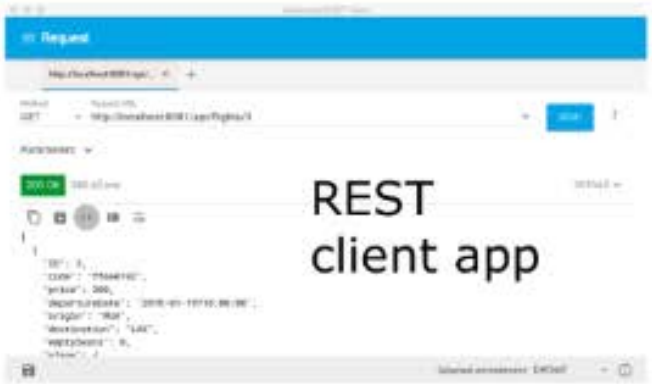
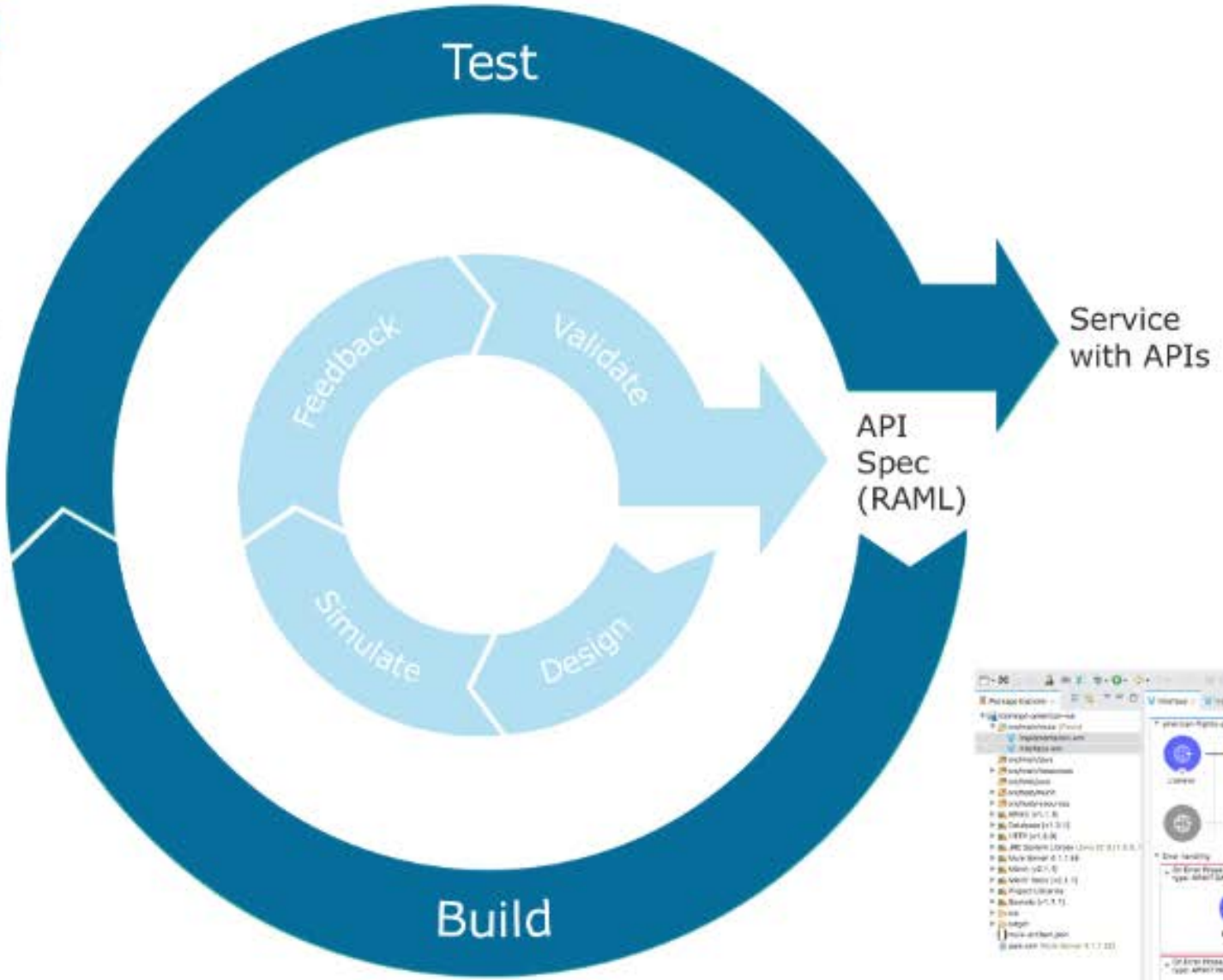
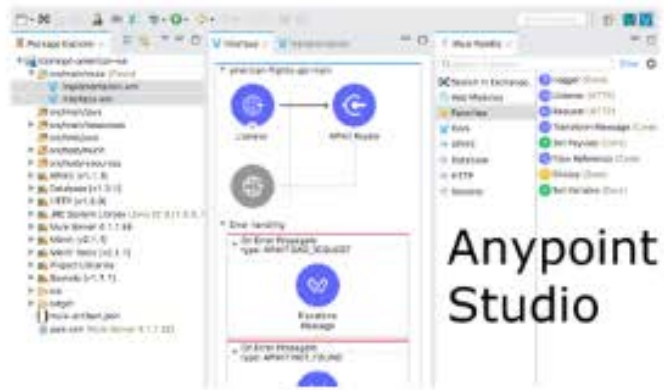
Module 4: Building APIs

Goal



REST
client app





Anypoint
Studio

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

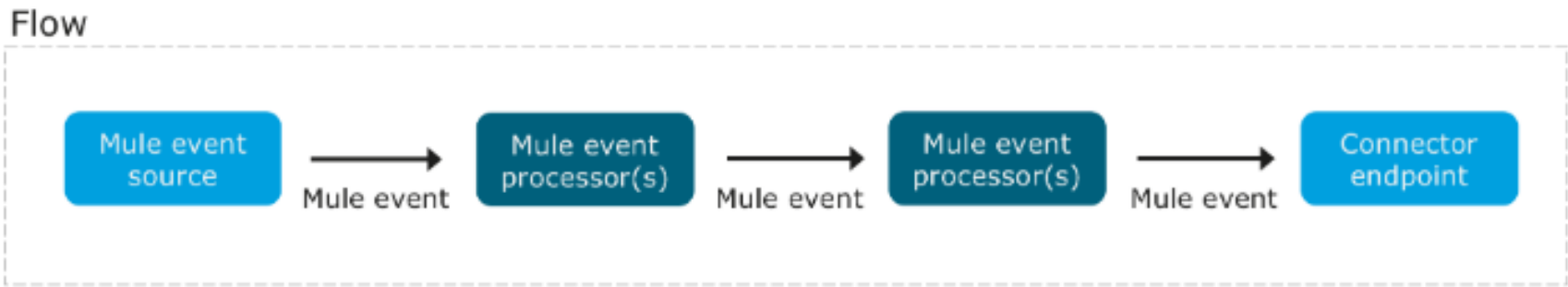
- Use Anypoint Studio to build, run, and test Mule applications
- Use a connector to connect to databases
- Use the graphical DataWeave editor to transform data
- Create RESTful interfaces for applications from RAML files
- Connect API interfaces to API implementations

Reviewing Mule 4 applications

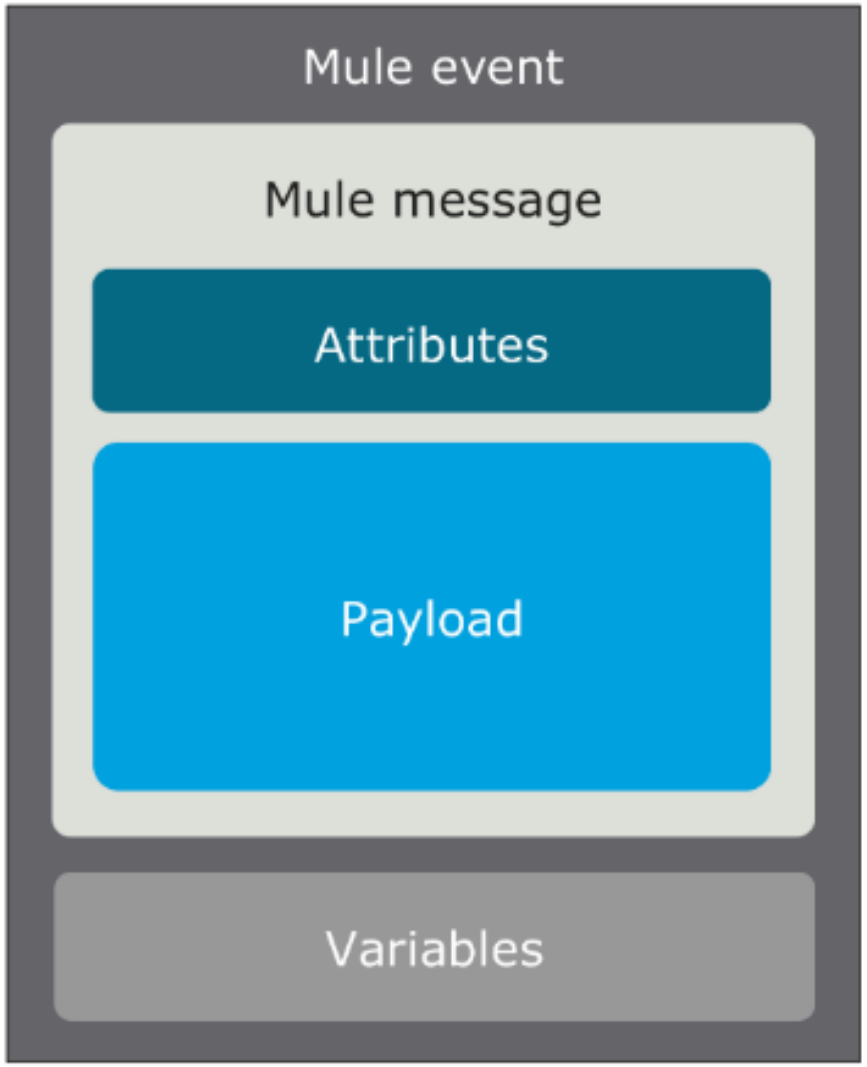
Review: Mule 4 applications and flows

- Mule applications receive events, process them, and route them to other endpoints

- Mule applications accept and process a Mule event through a series of Mule event processors plugged together in a flow with
  - A **Mule event source** that initiates the execution of the flow
  - **Mule event processors** that transform, filter, enrich, and process the event and its message



Review: Mule 4 event structure



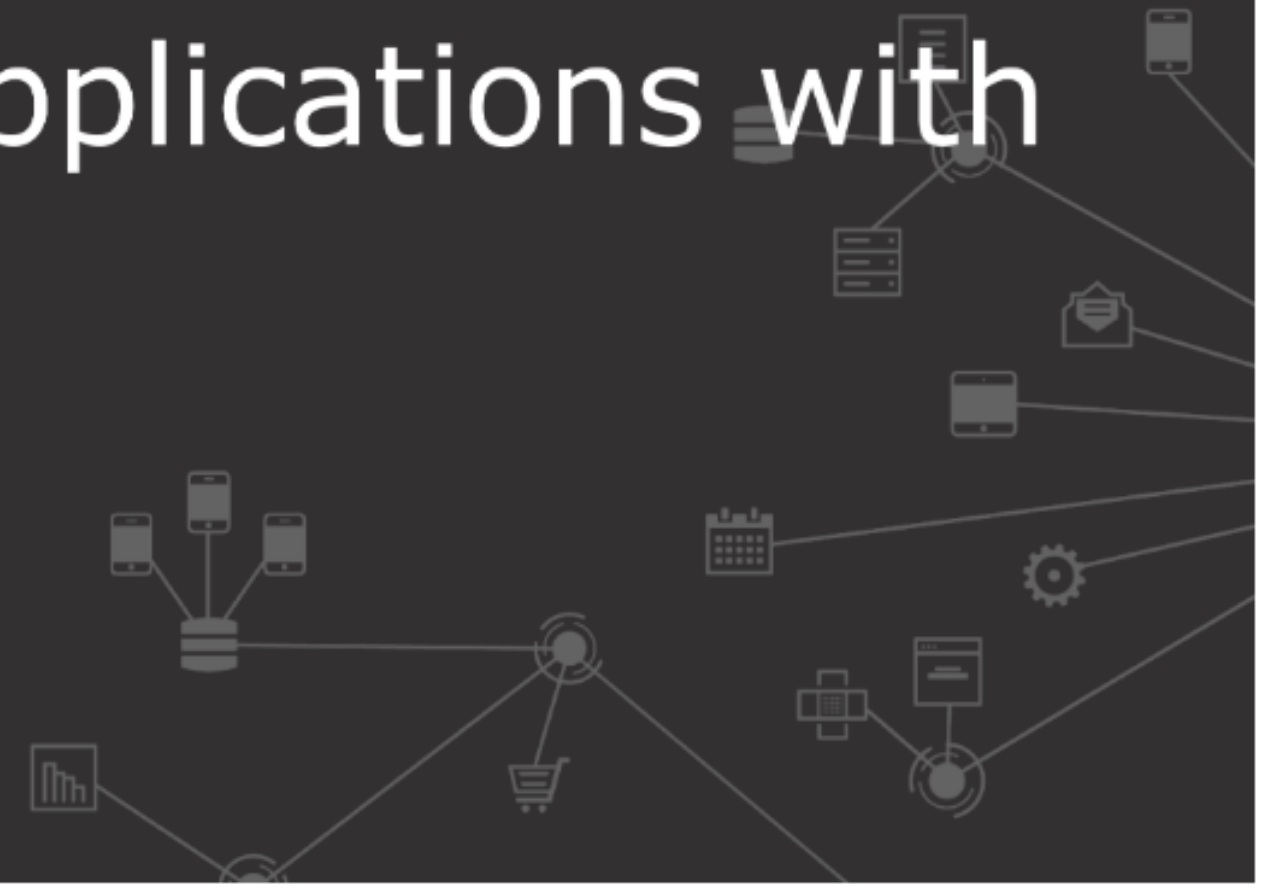
The data that passes through flows in the app

Metadata contained in the message header

The core info of the message - the data the app processes

Metadata for the Mule event - can be defined and referenced in the app processing the event

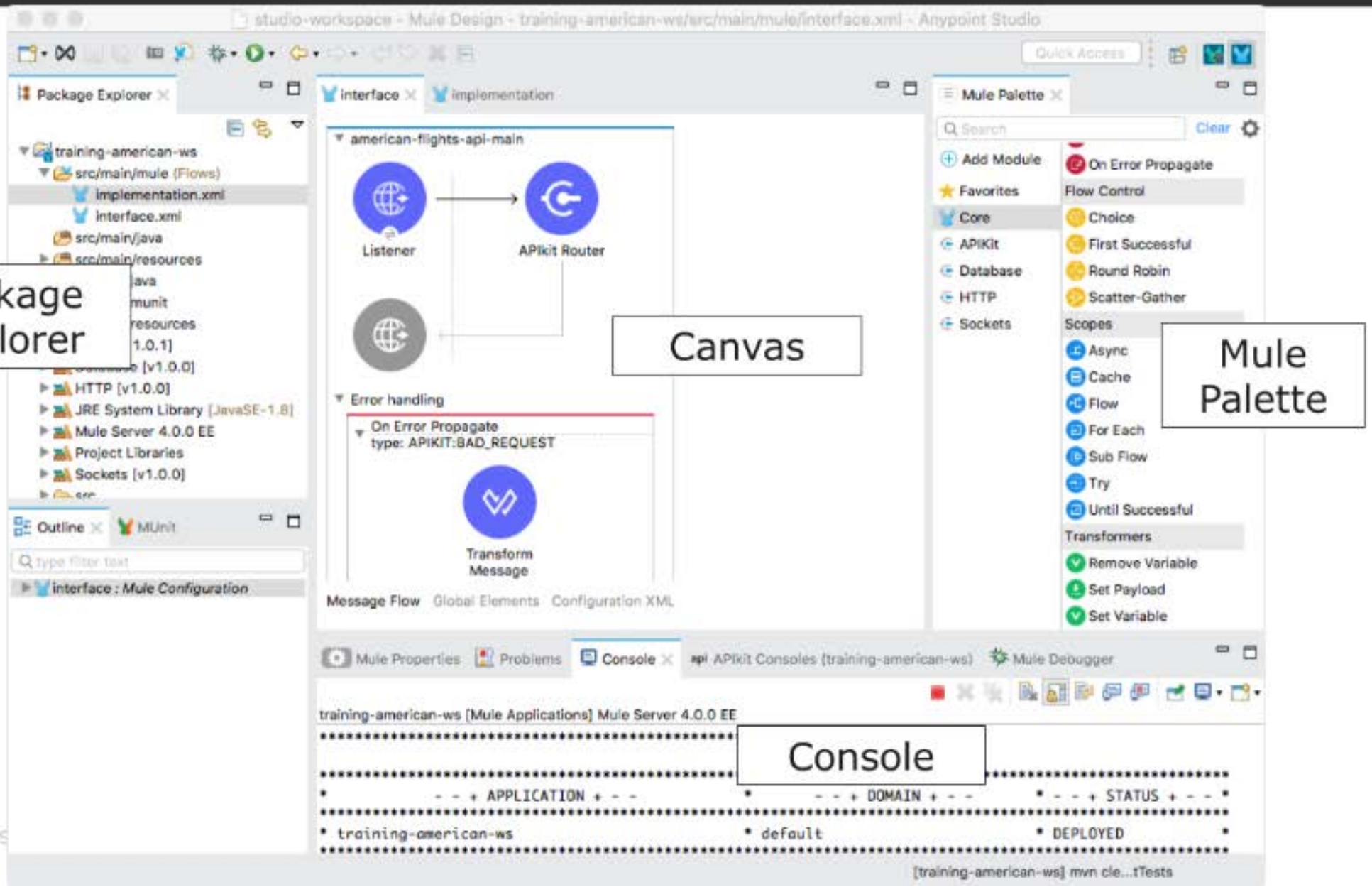
Creating Mule applications with Anypoint Studio



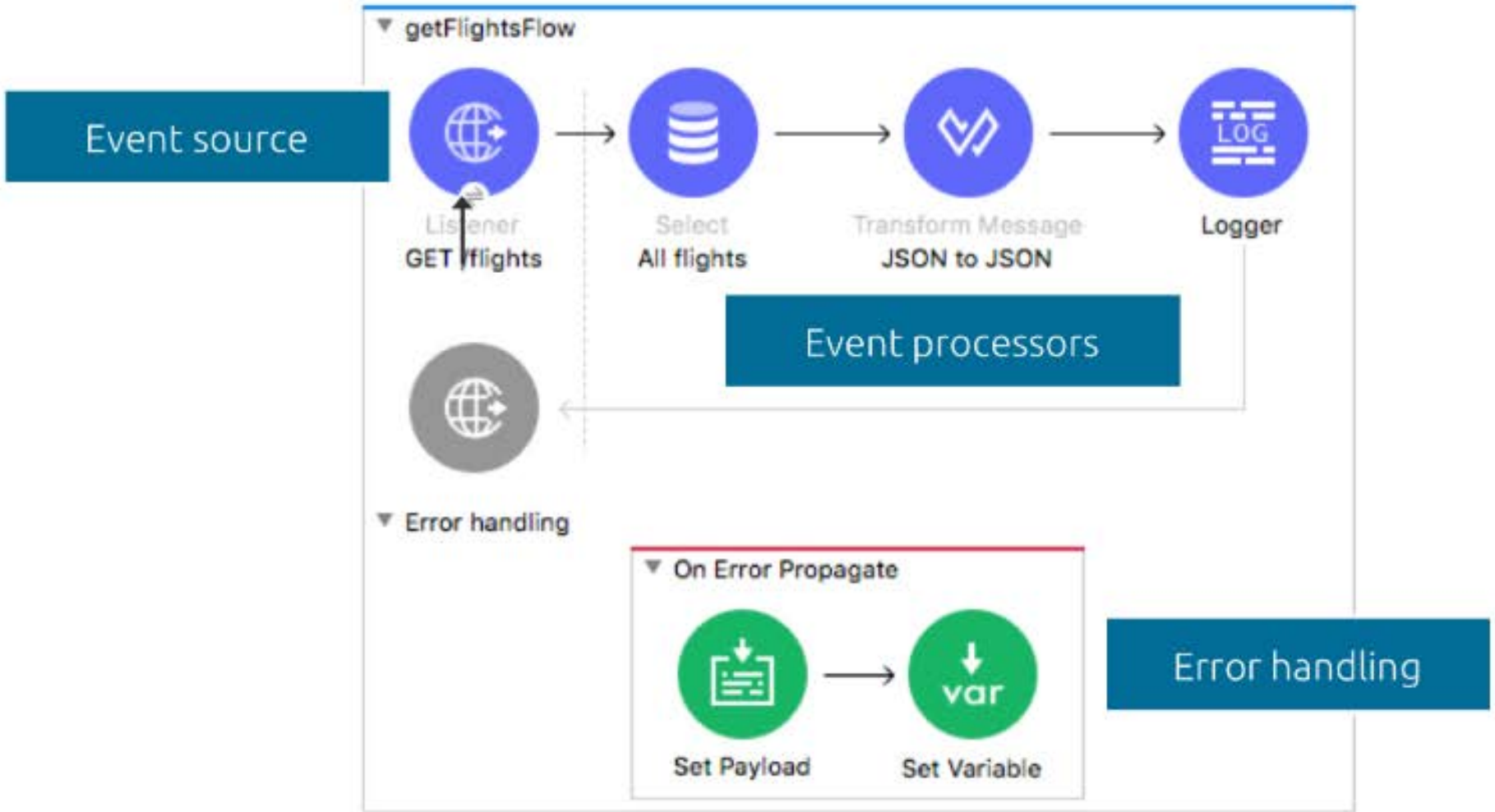
Introducing Anypoint Studio

- Based on Eclipse, a common Java integrated development environment
- Features include
  - Two-way editing between graphical and XML views
  - Pre-built tooling to connect to APIs (REST, SOAP), protocols (HTTP, FTP, SMTP, more), and popular services (Salesforce, Workday, more!)
  - A data transformation framework and language
  - An embedded Mule runtime to test applications without leaving it
  - Visual debugging
  - One-click deployment of applications to CloudHub
  - Templates for common integration patterns
  - Integration with Maven for continuous build processes

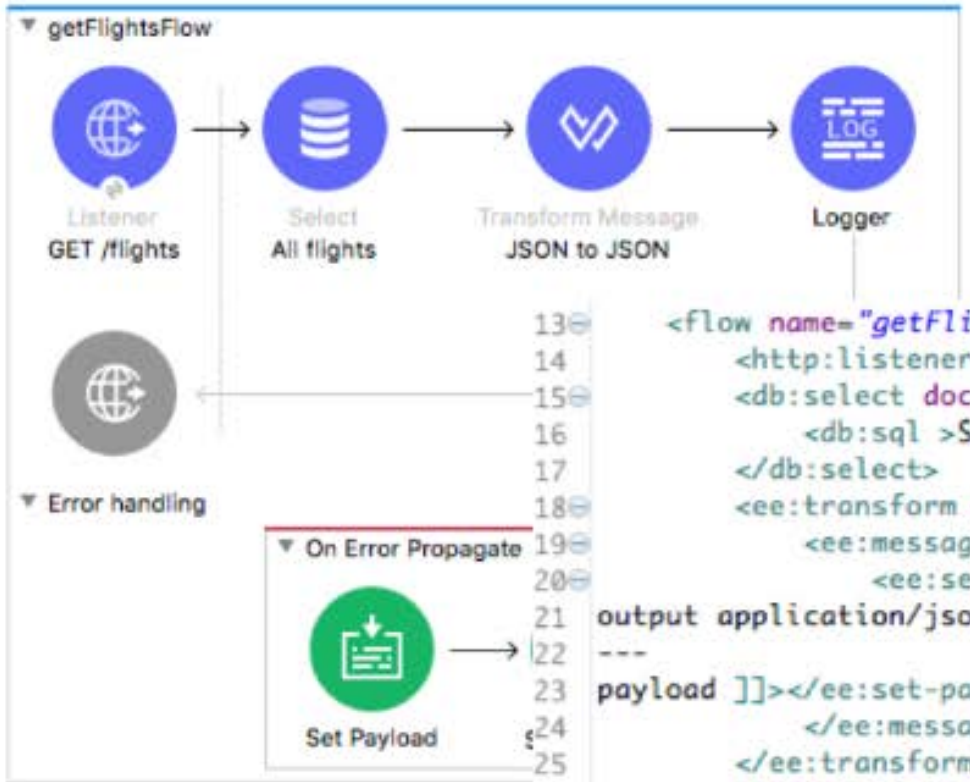
Anypoint Studio anatomy



Anatomy of a flow: Visual



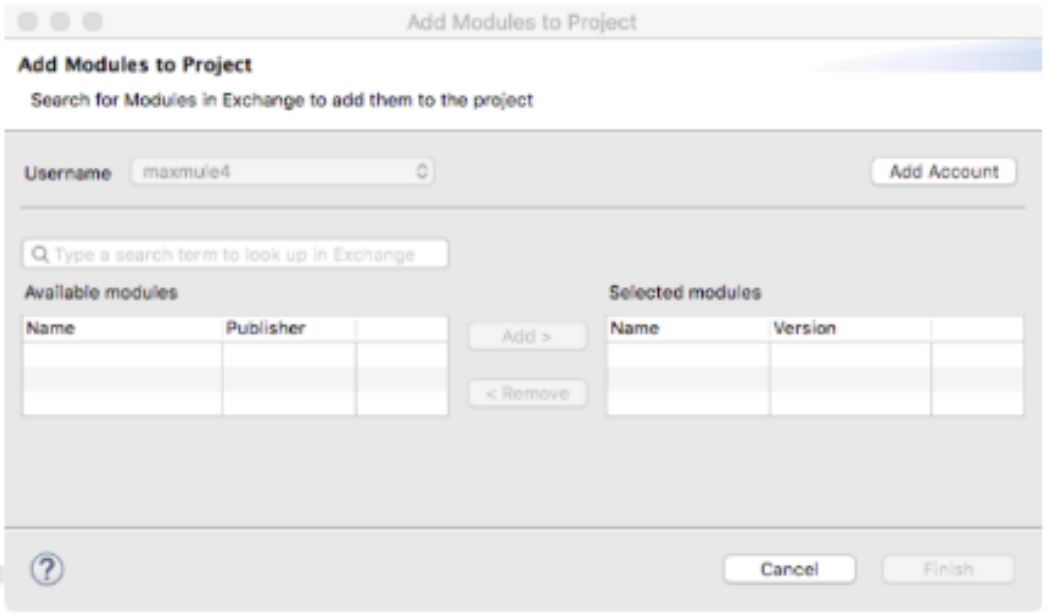
Anatomy of a flow: XML

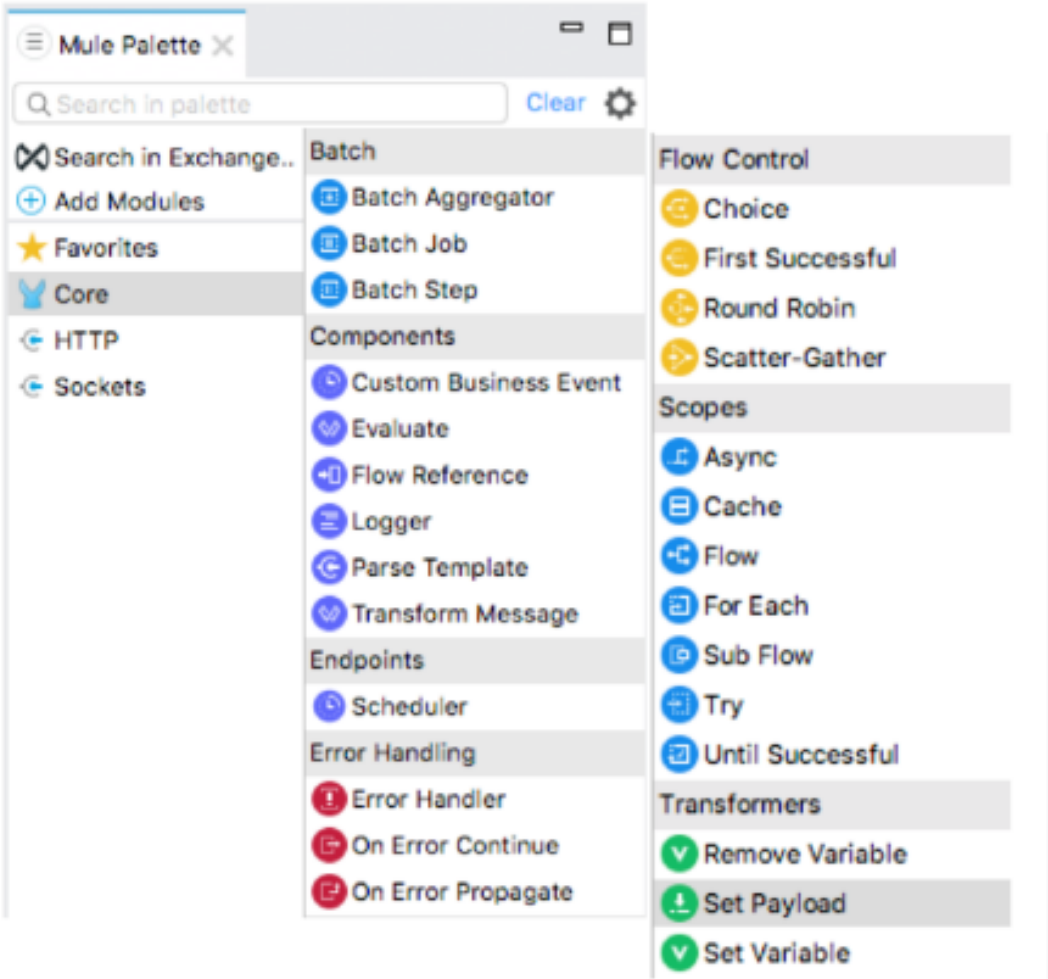



```
13    <flow name="getFlightsFlow" doc:id="fd1b7165-5456-476a-818f-4b25016b4a26" >
14        <http:listener doc:name="GET /flights" doc:id="97ab8f0f-2cb3-4809-ab61-
15        <db:select doc:name="All flights" doc:id="13978339-a263-4512-9b6b-114f1
16            <db:sql >SELECT * FROM american</db:sql>
17        </db:select>
18        <ee:transform doc:name="JSON to JSON" doc:id="8a319aca-486a-487b-840a-c
19            <ee:message >
20                <ee:set-payload ><![CDATA[%dw 2.0
21 output application/json
22 ---
23 payload ]]></ee:set-payload>
24            </ee:message>
25        </ee:transform>
26        <logger level="INFO" doc:name="Logger" doc:id="5b1628cf-ec96-4beb-81d6-
27        <error-handler >
28            <on-error-propagate enableNotifications="true" logException="true"
29                <set-payload value="#[payload]" doc:name="Set Payload" doc:id='
30                <set-variable value="test" doc:name="Set Variable" doc:id="f40e
31            </on-error-propagate>
32        </error-handler>
33    </flow>
```

Mule application building blocks

- Are separated into categories in the Core section of the Mule Palette
- By default, projects include HTTP and Sockets modules
- Can add additional modules



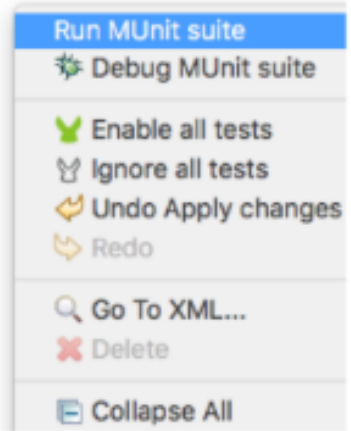


Running applications

- Anypoint Studio comes with an embedded Mule runtime to test applications without leaving it
- The console outputs application logs and information

Automating testing of applications

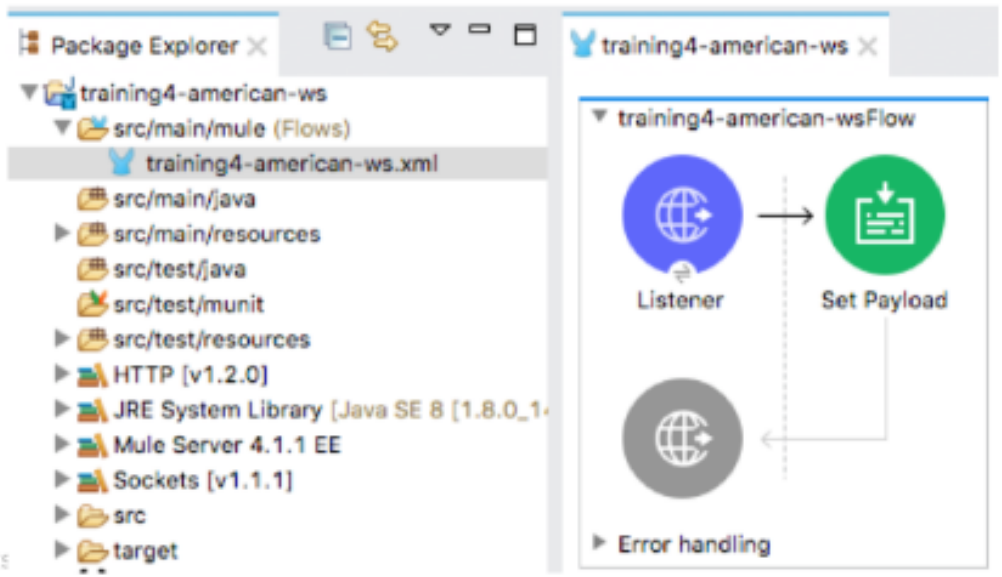
- You can automate testing of Mule applications using MUnit
- MUnit is a Mule app testing framework for building automated tests
- MUnit is fully integrated with Anypoint Studio
  - You can create, design, and run MUnit tests and suites of tests just like you do Mule applications

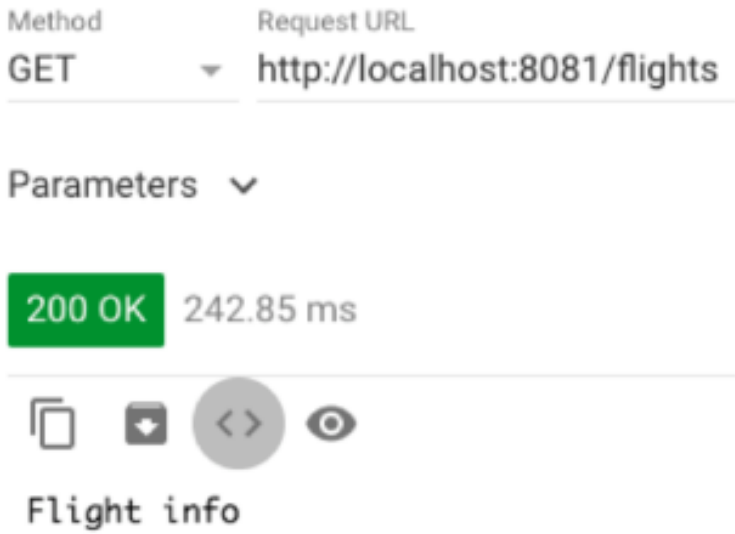


- MUnit is covered in *Anypoint Platform Development: Advanced*

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

- 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 ARC

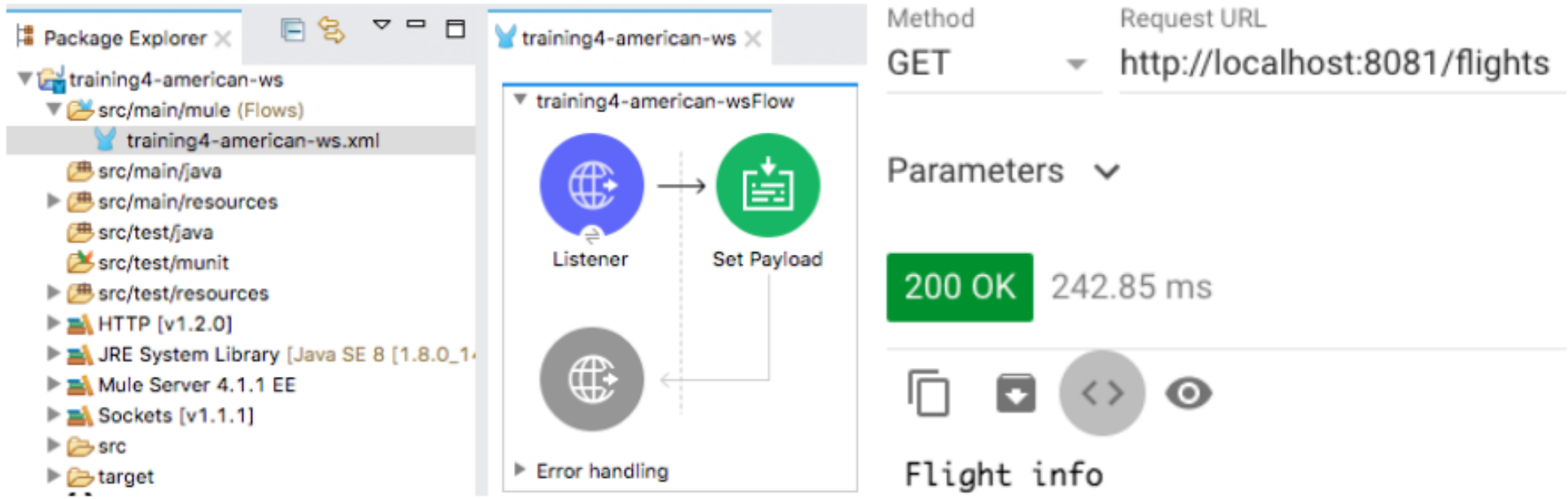




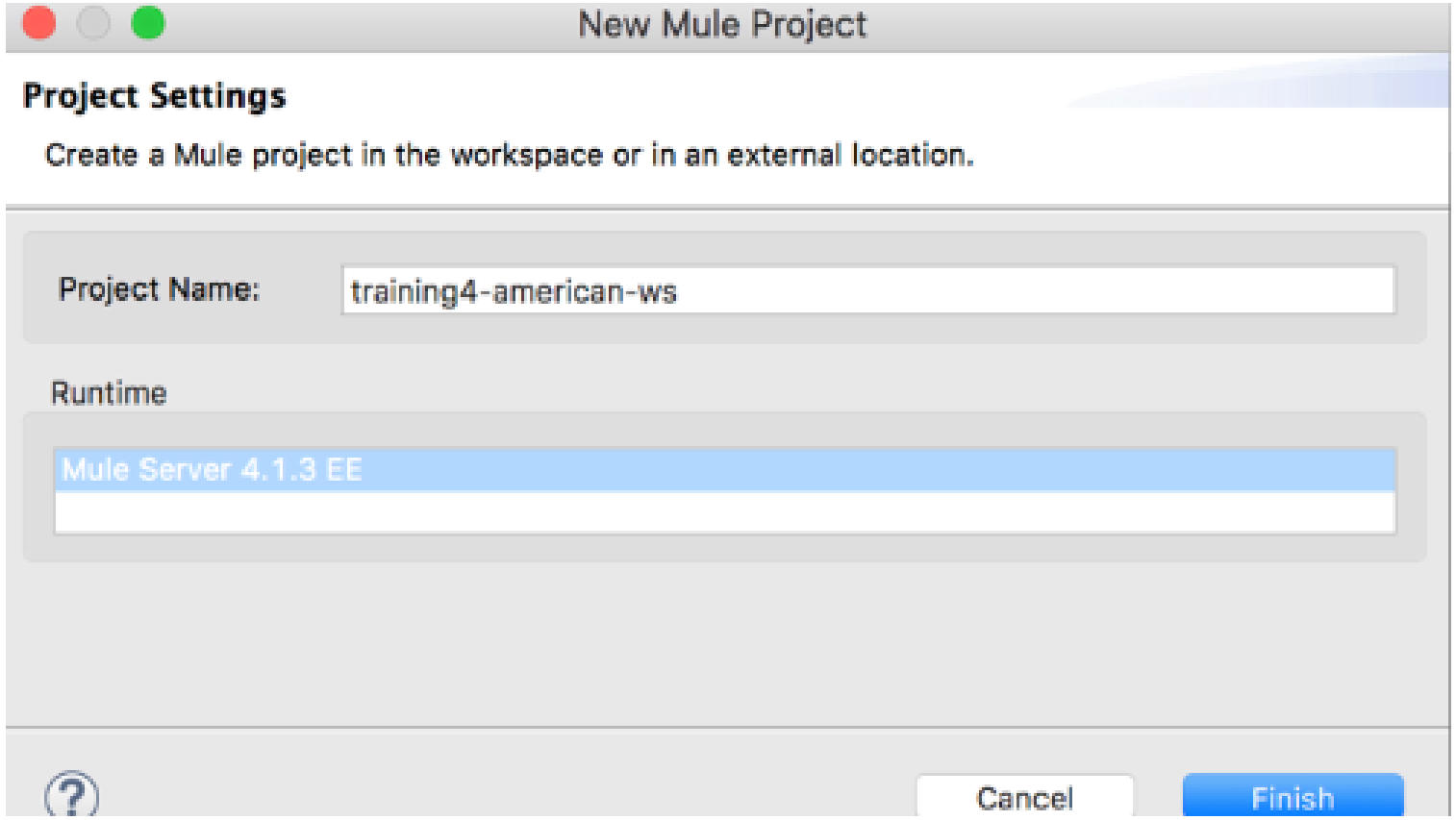
Walkthrough 4-1: Create a Mule application with Anypoint Studio

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.

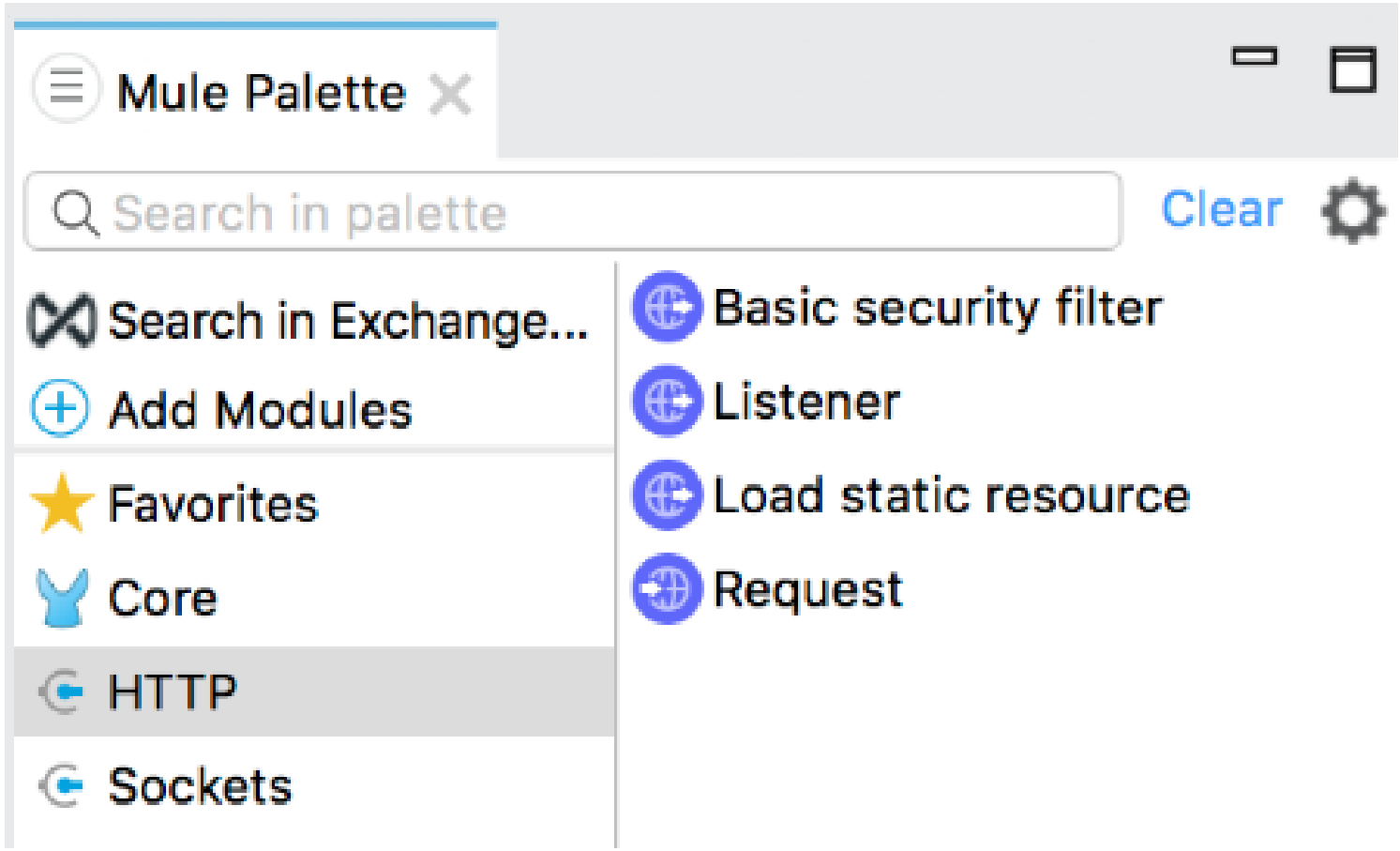


1. Open Anypoint Studio.
2. Select File > New > Mule Project.
3. In the New Mule Project dialog box, set the Project Name to training4-american-ws.
4. Ensure the Runtime is set to the latest version of Mule.

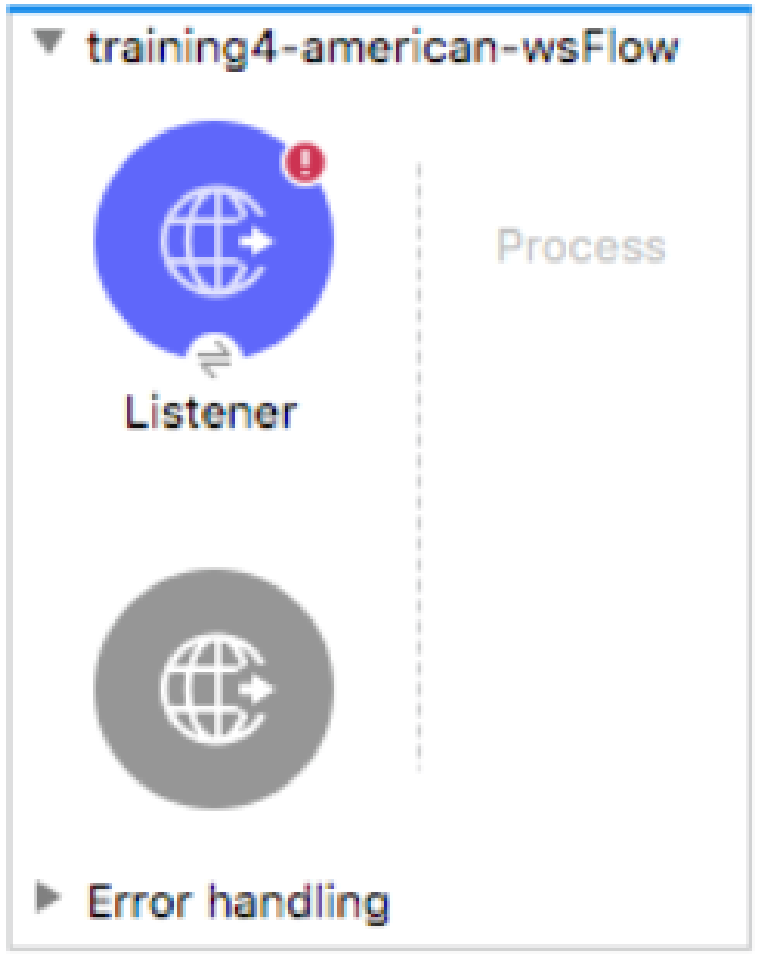


Create an HTTP connector endpoint to receive requests

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

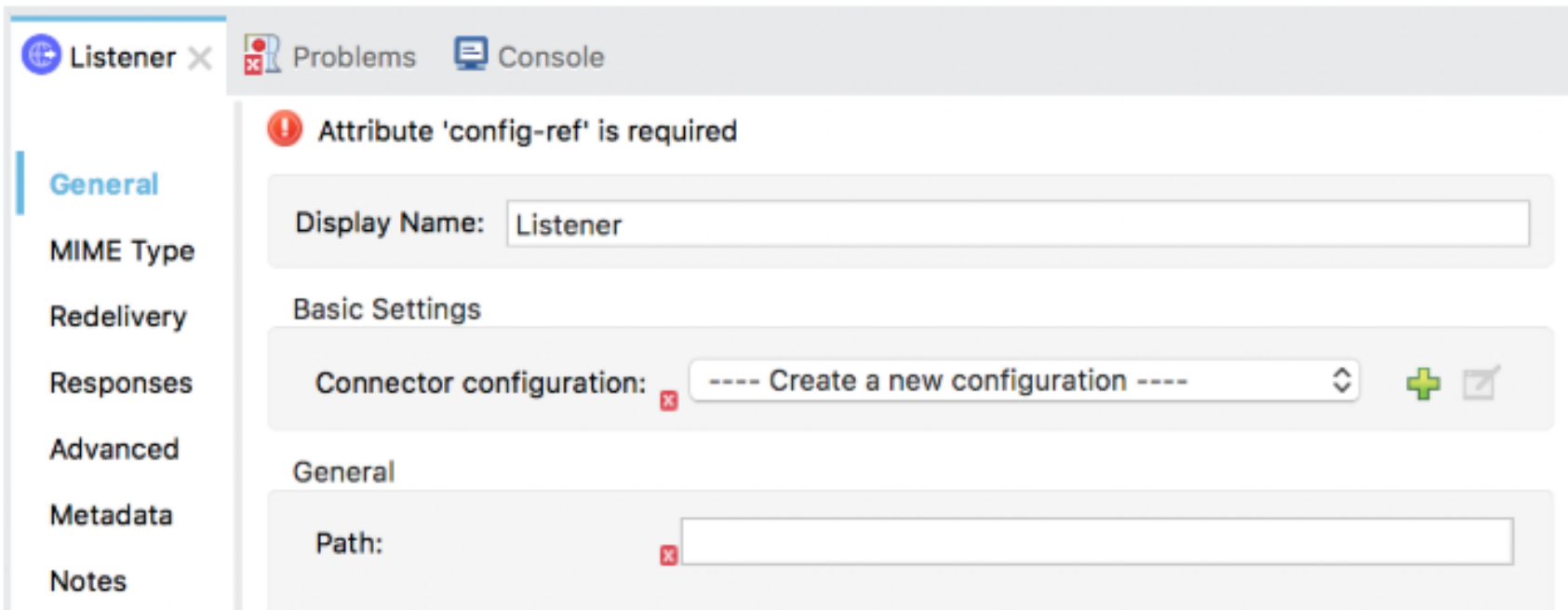


7. Drag the Listener operation from the Mule Palette to the canvas.



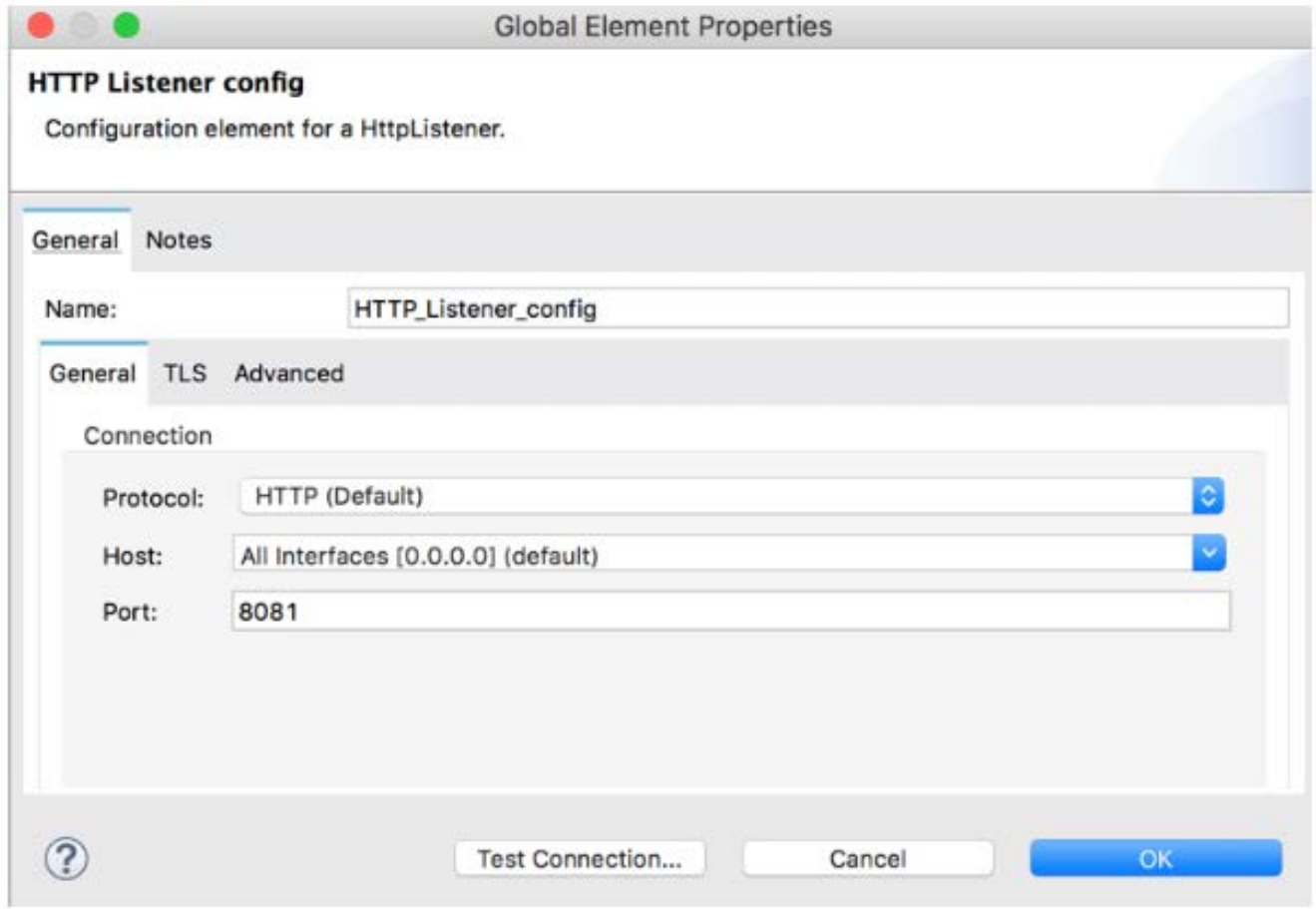
8. Double-click the HTTP Listener endpoint.

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



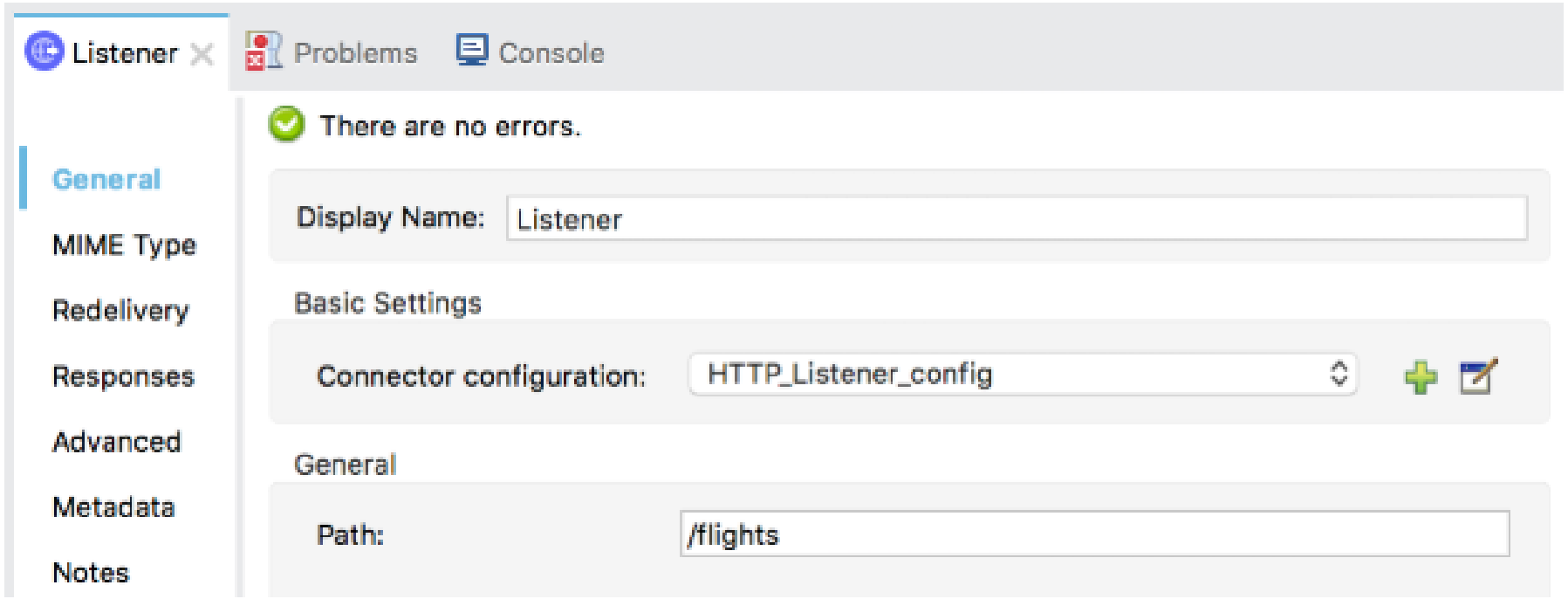
10. In the Global Element Properties dialog box, verify the following default values are present.

- Host: 0.0.0.0
- Port: 8081

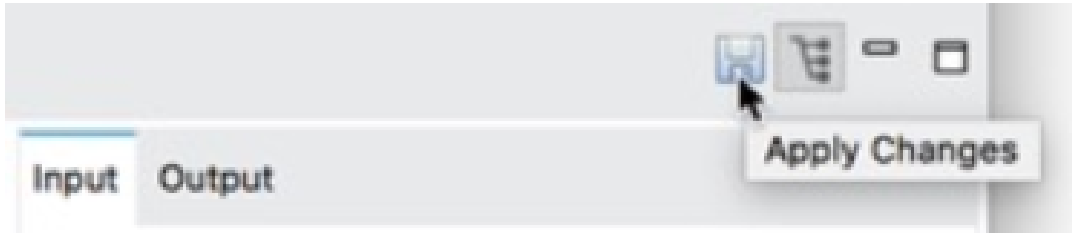


11. Click OK.

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

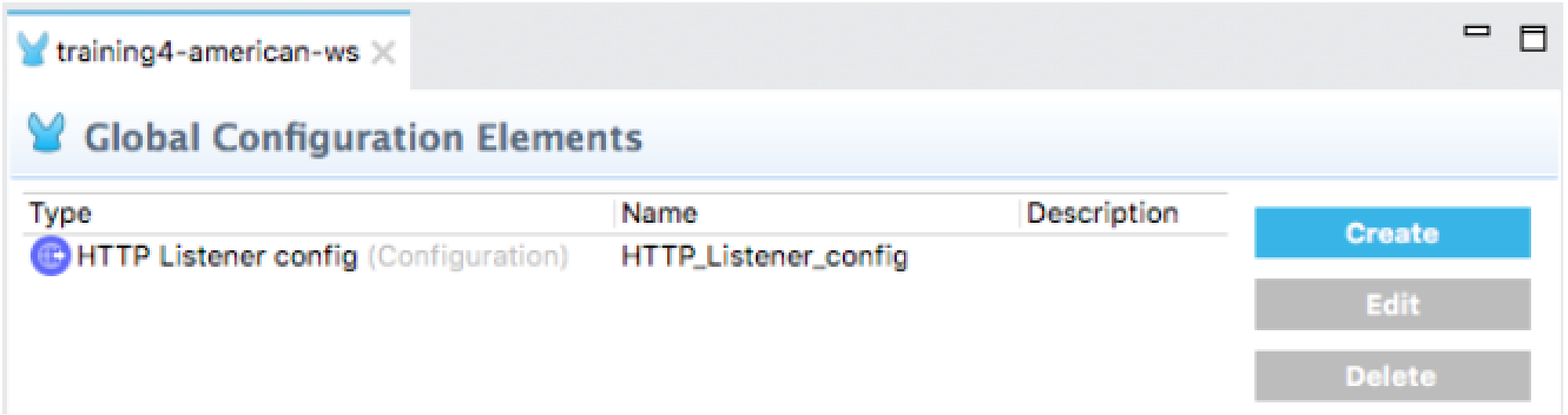


13. Click the Apply Changes button to save the file.



Review the HTTP Listener global element

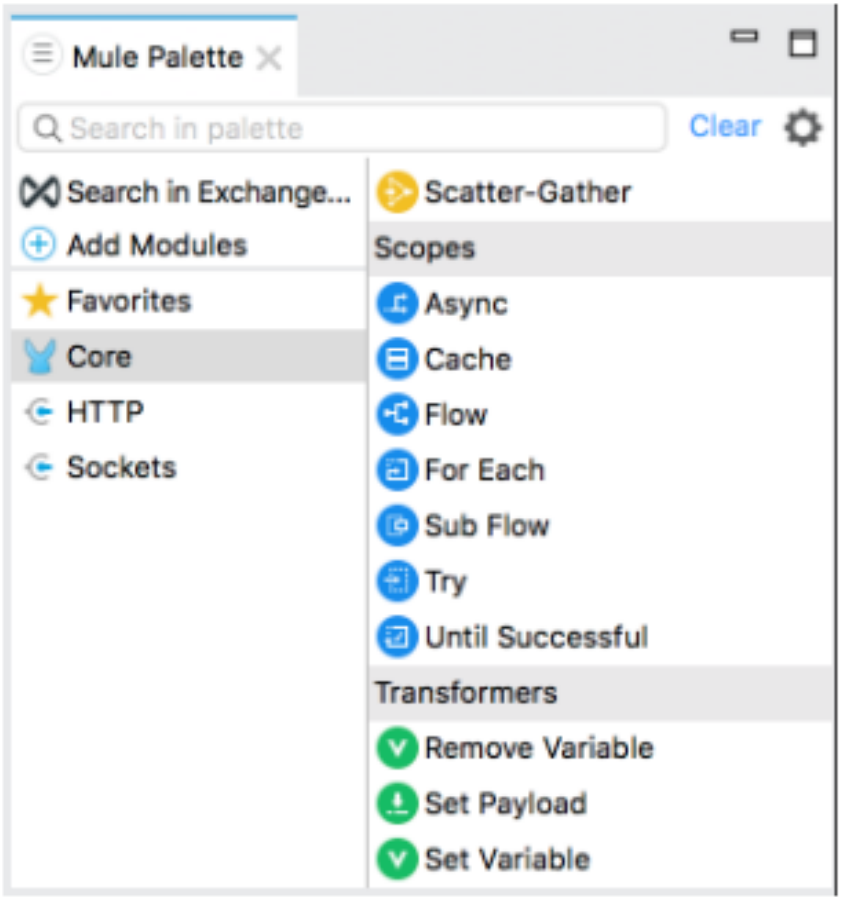
14. Select the Global Elements tab at the bottom of the canvas.
15. Double-click the HTTP Listener config.



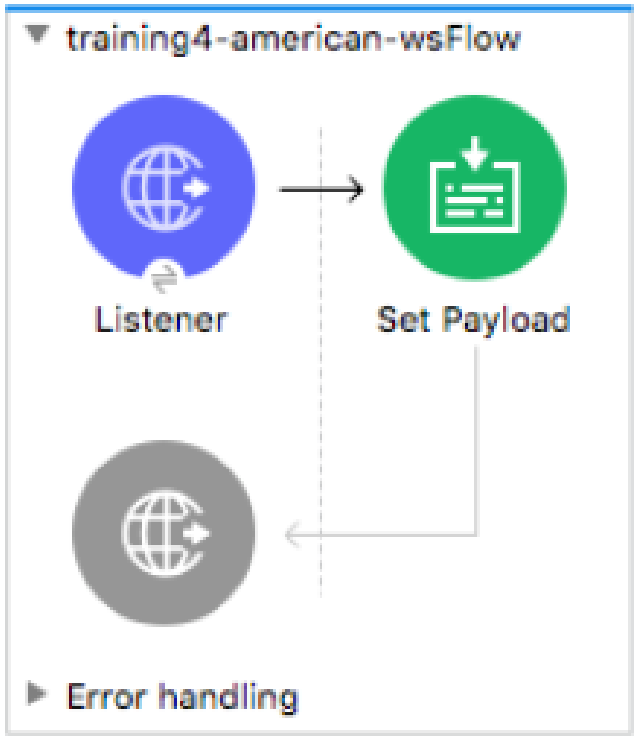
16. Review the information in the Global Element Properties dialog box and click Cancel.
17. Select the Message Flow tab to return to the canvas.

Display data

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



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

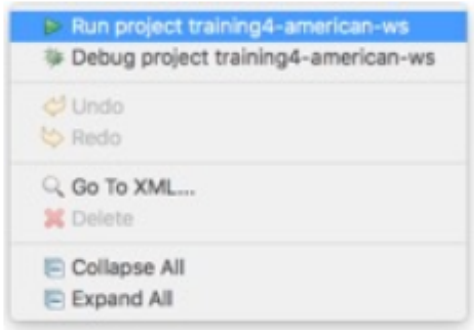


22. Select the Configuration XML tab at the bottom of the canvas and examine the corresponding XML.

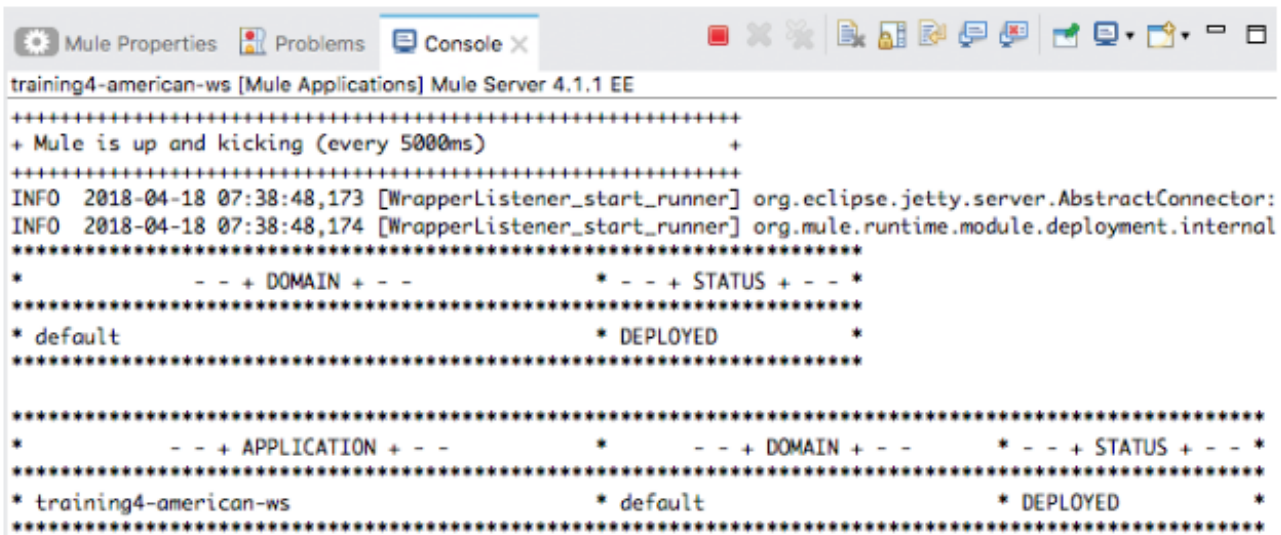
23. Select the Message Flow tab to return to the canvas.
24. Click the Save button or press Cmd+S or Ctrl+S.

Run the application

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

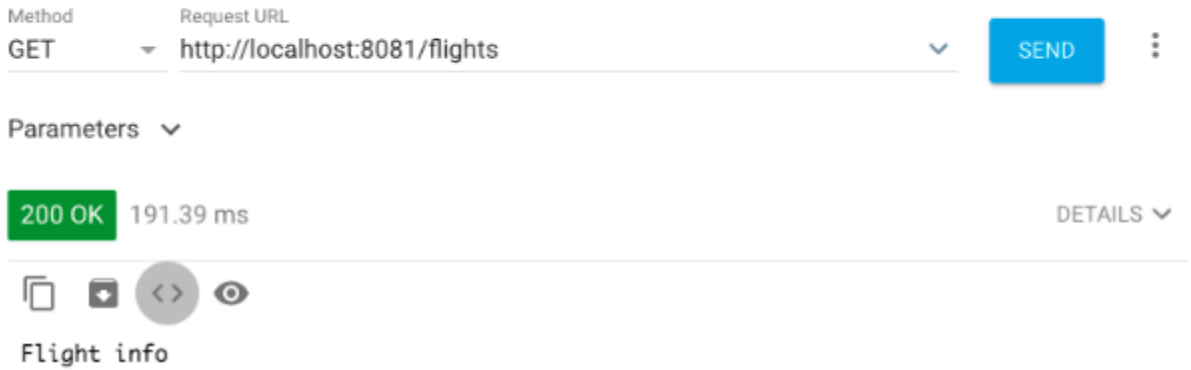


26. Watch the Console view; it should display information letting you know that both the Mule runtime and the training4-american-ws application started.

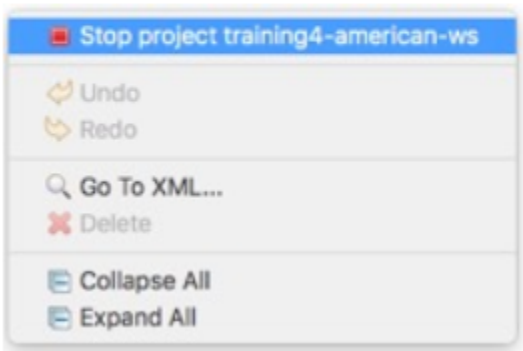


Test the application

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.

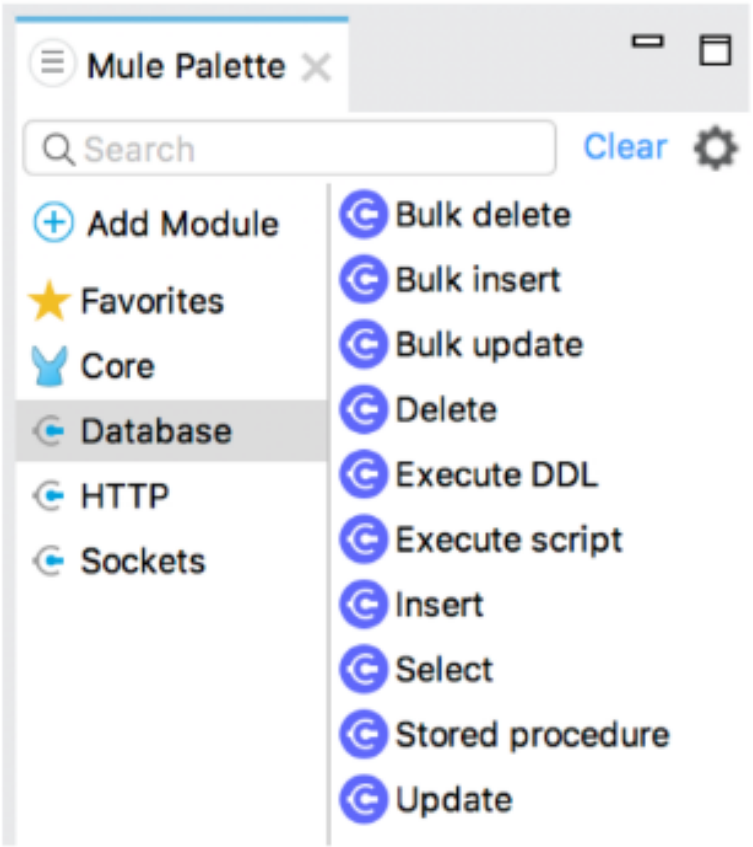


Connecting to data

The Database connector

- Can connect to almost any JDBC relational database
  - Any database engine for which you have a driver

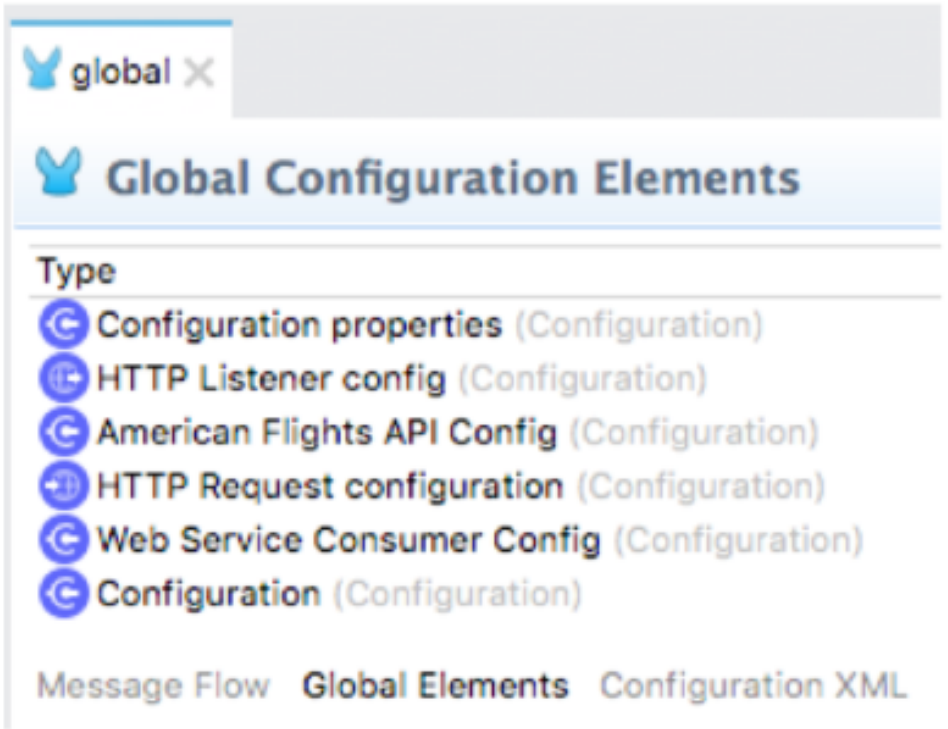
- To use
  - Add the Database module to your project
  - Add a database operation to a flow
  - Configure the connection to the database



Global configuration elements

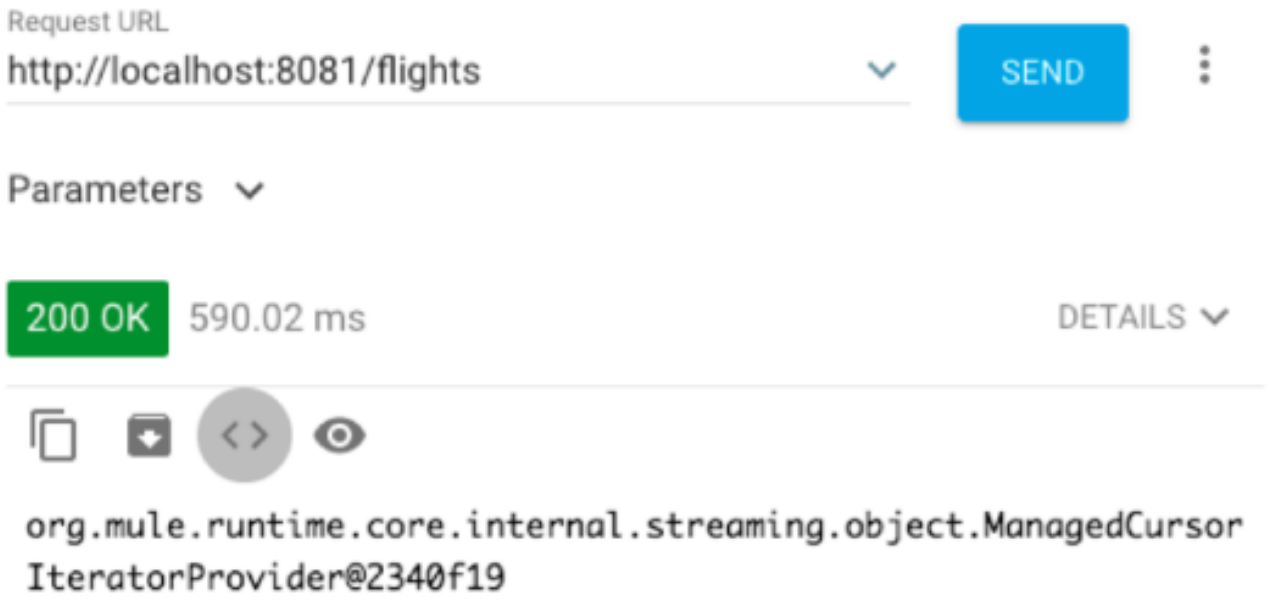
- For most operations, a lot of the configuration is encapsulated in a separate global element
  - A reusable configuration that can be used by many operations
  - Defines a connection to a network resource

- This is a connector configuration
  - Though it is sometimes referred to simply as the connector



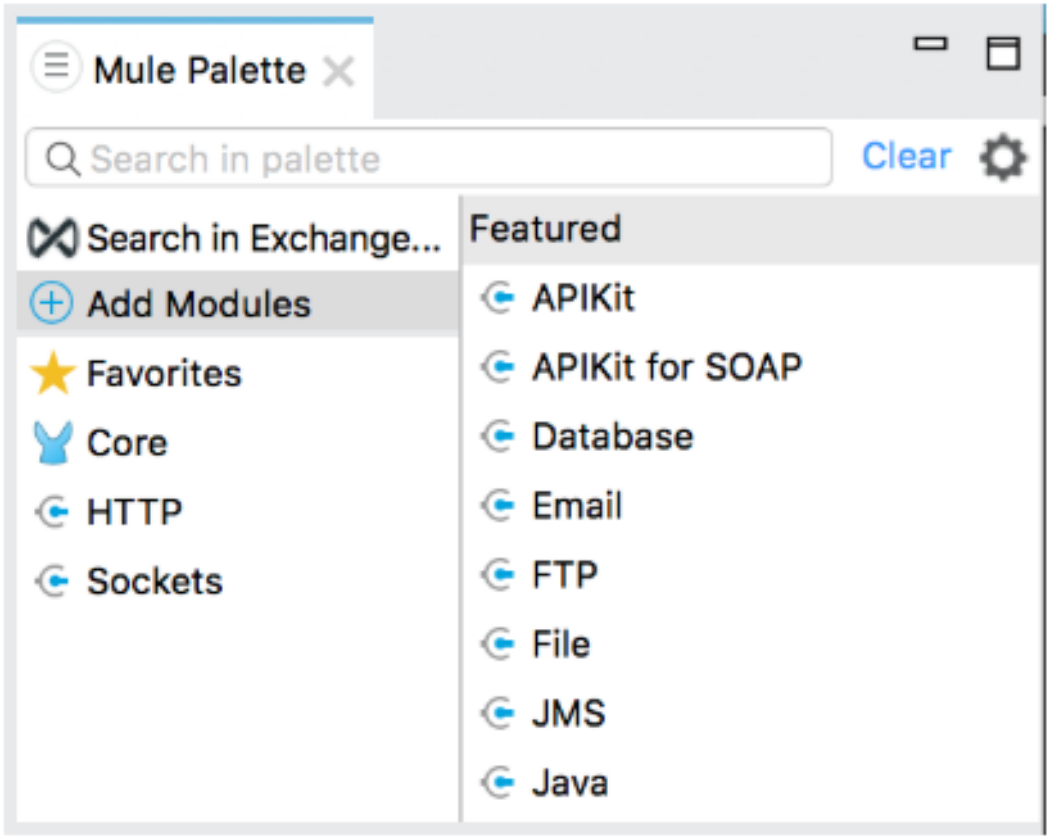
Walkthrough 4-2: Connect to data (MySQL database)

- 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

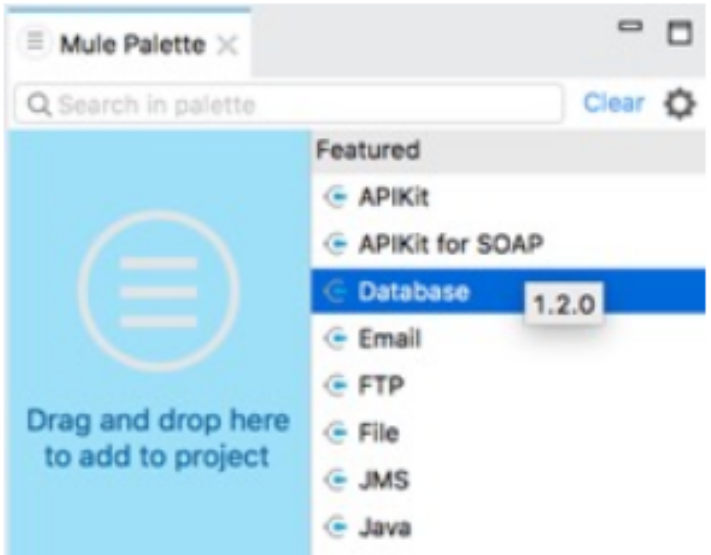


Add a Database connector endpoint

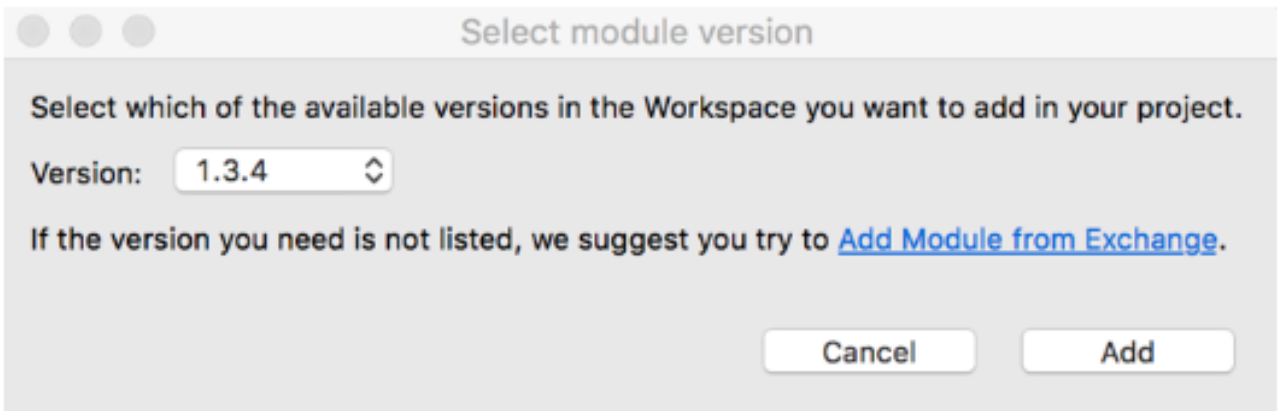
2. Return to Anypoint Studio.
3. Right-click the Set Payload message processor and select Delete.
4. In the Mule Palette, select Add Modules.



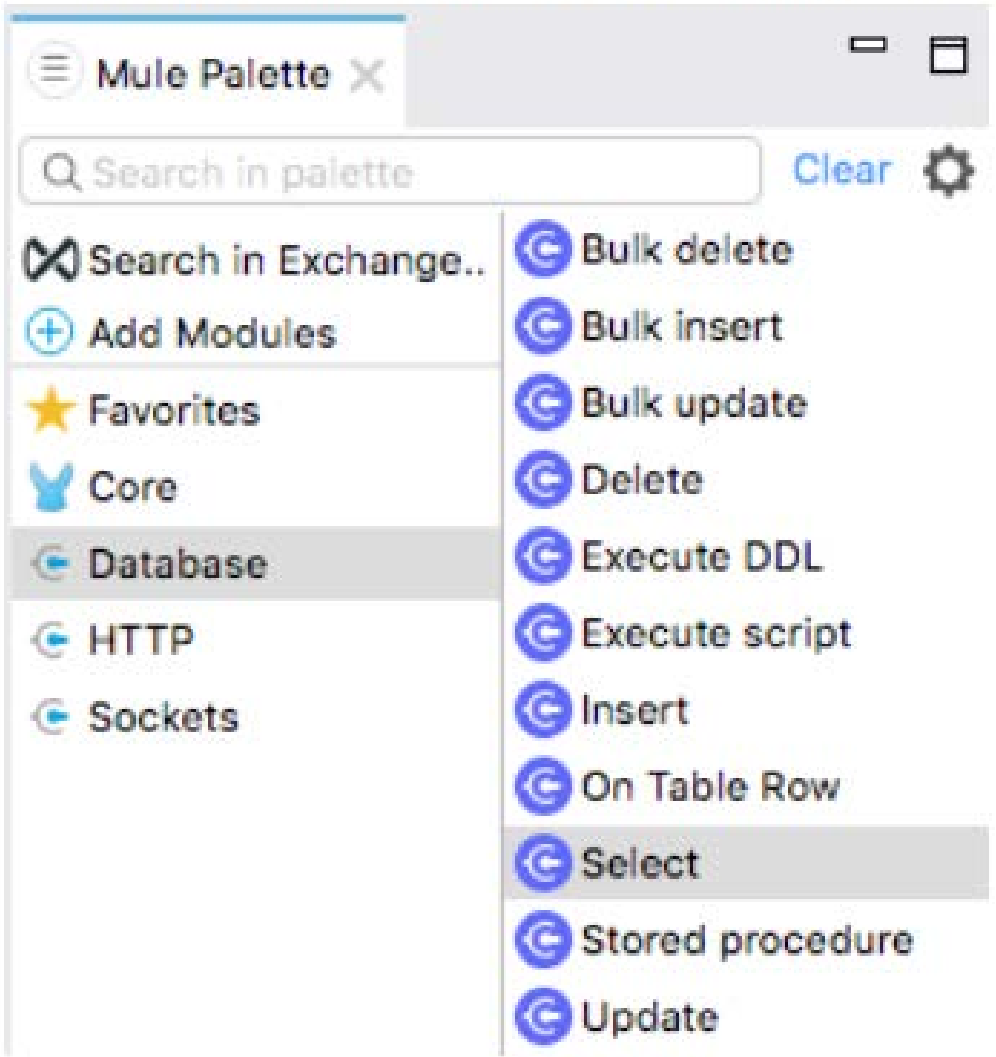
5. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.



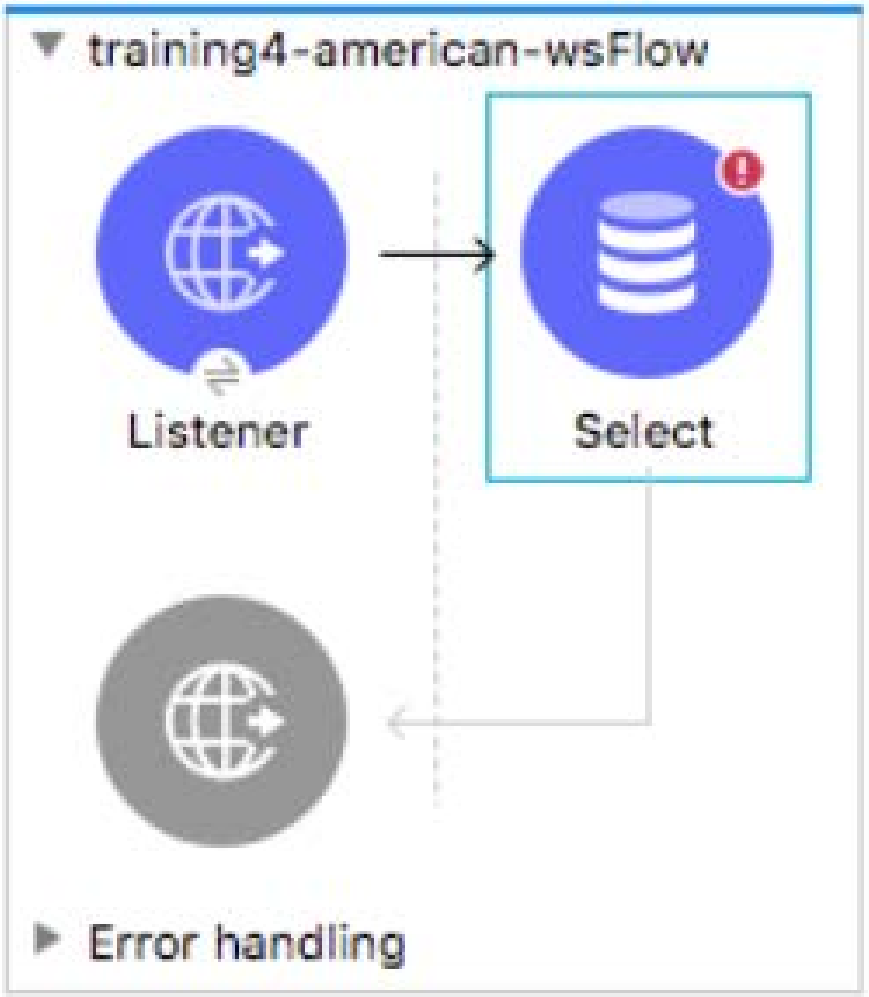
6. If you get a Select module version dialog box, select the latest version and click Add.



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

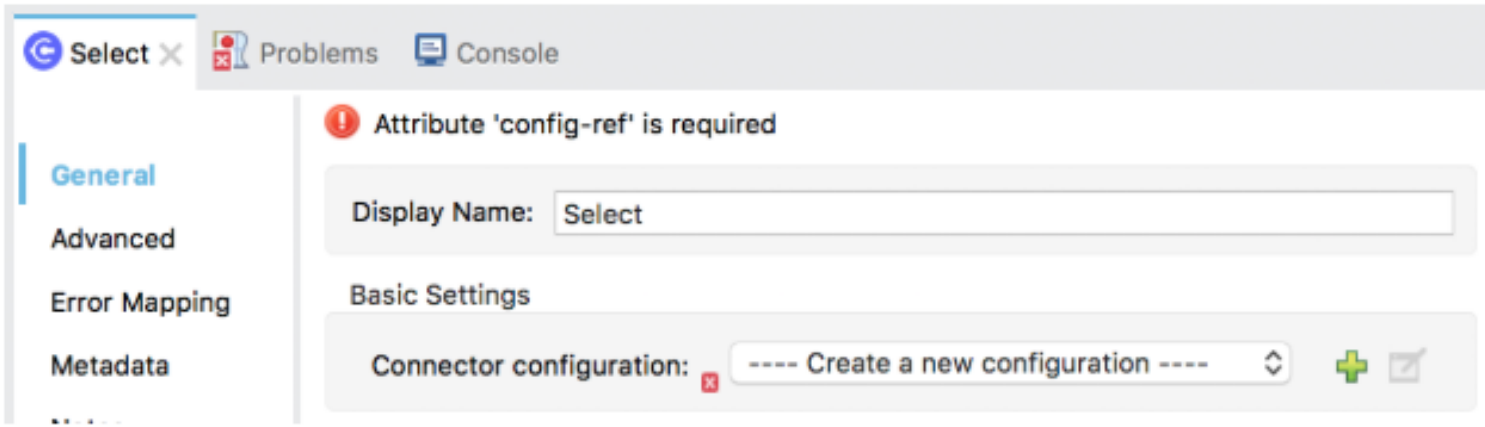


8. Drag and drop the Select operation in the process section of the flow.

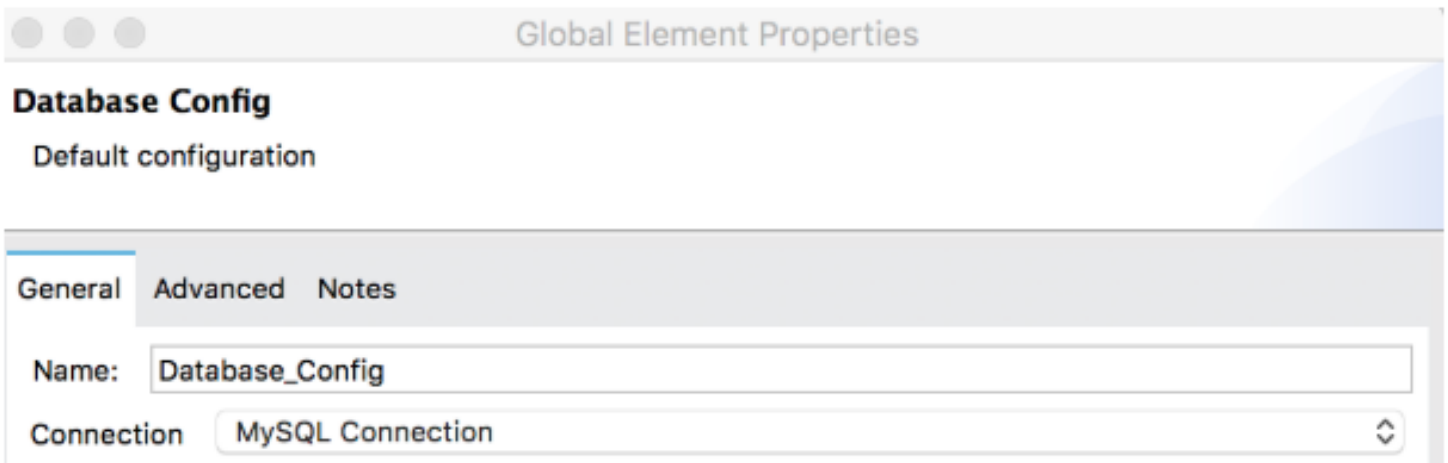


Option 1: Configure a MySQL Database connector (if you have access to port 3306)

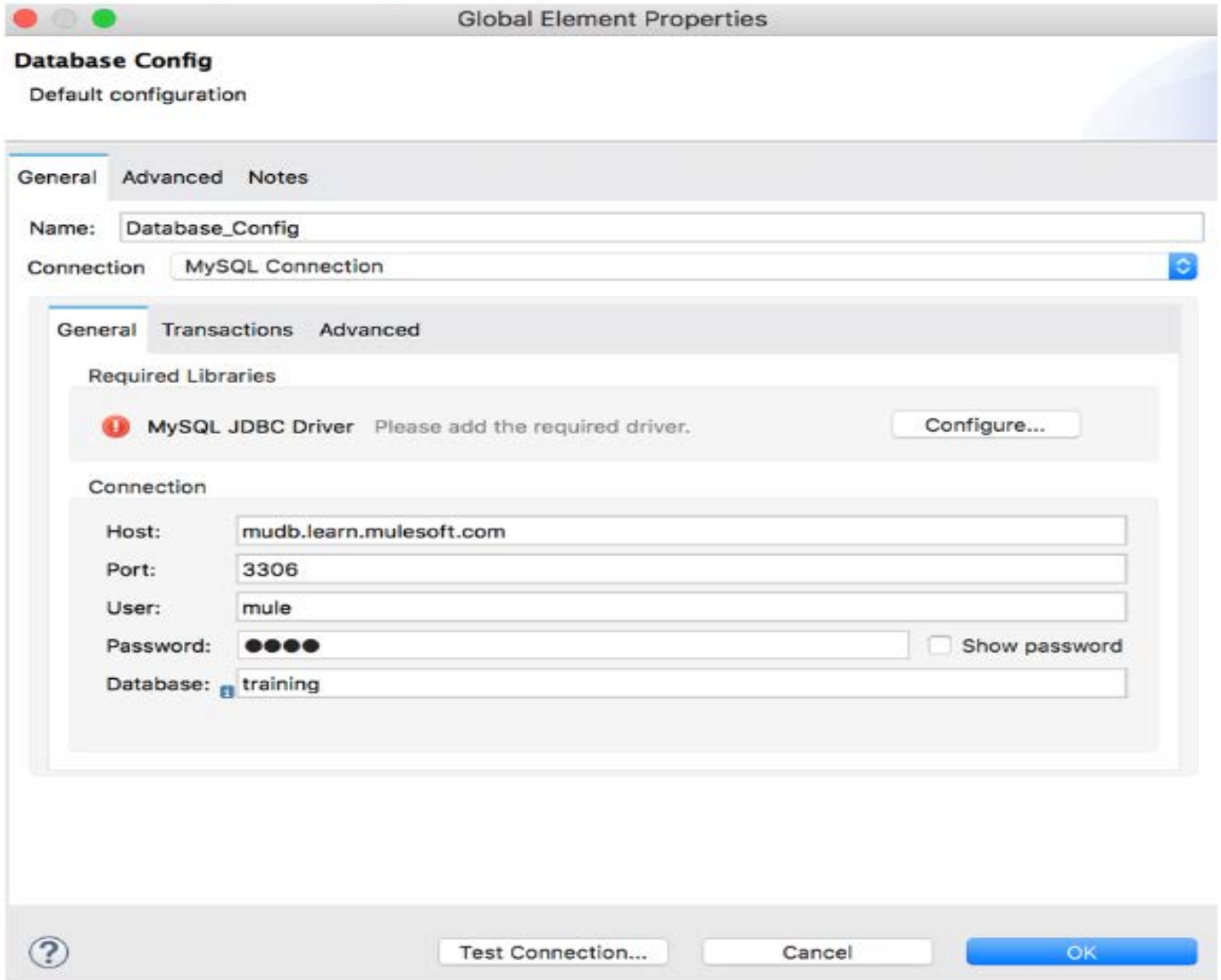
9. In the Select properties view, click the Add button next to connector configuration.



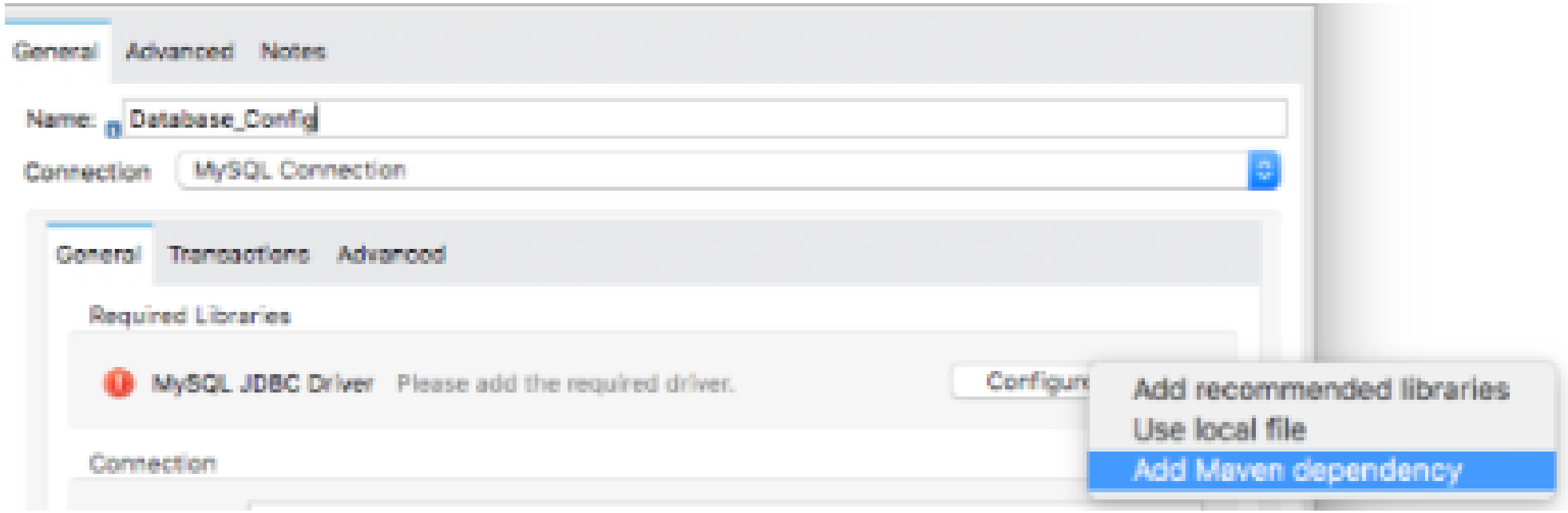
10. In the Global Element Properties dialog box, set the Connection to MySQL Connection.



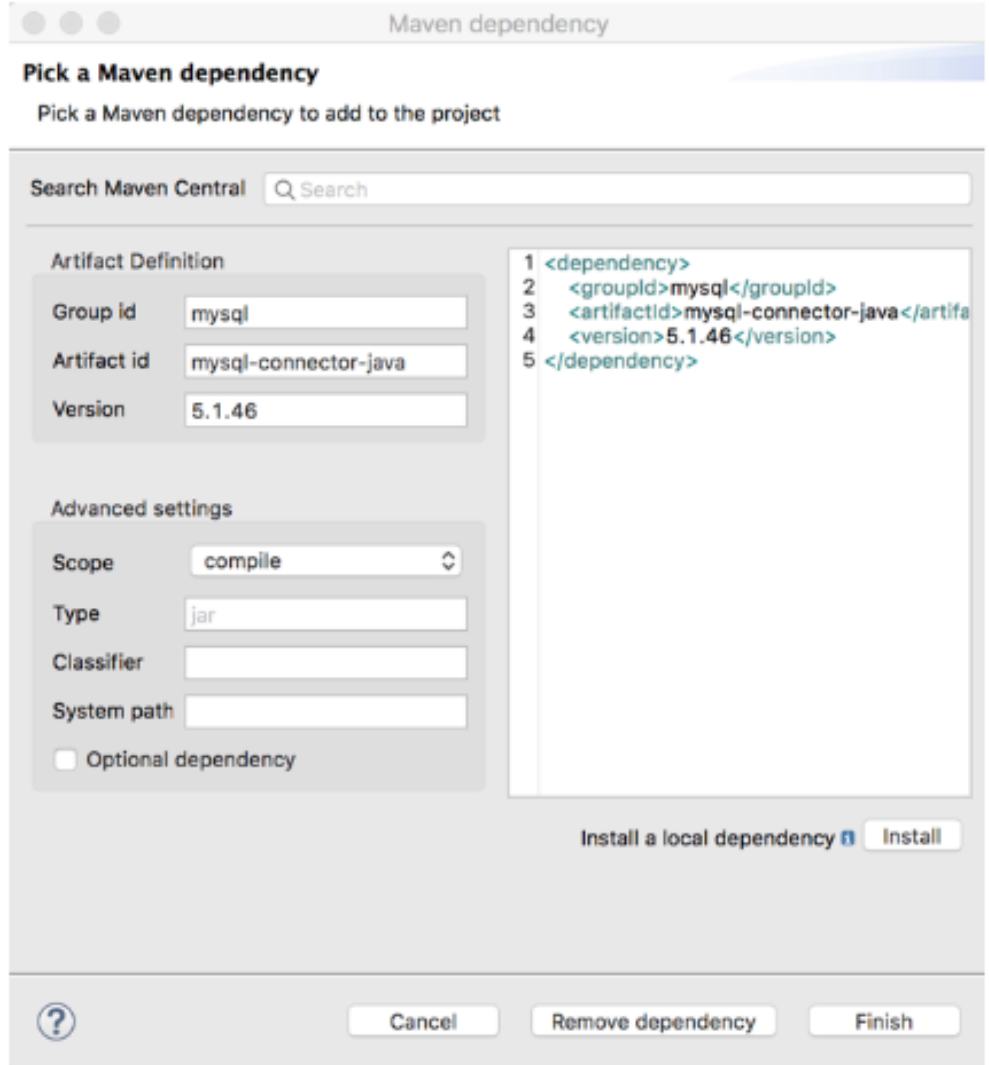
11. Set the host, port, user, password, and database values to the values listed in the course snippets.txt file.



12. Click the Configure button next to MySQL JDBC Driver.
13. In the configure drop-down list, select Add Maven dependency

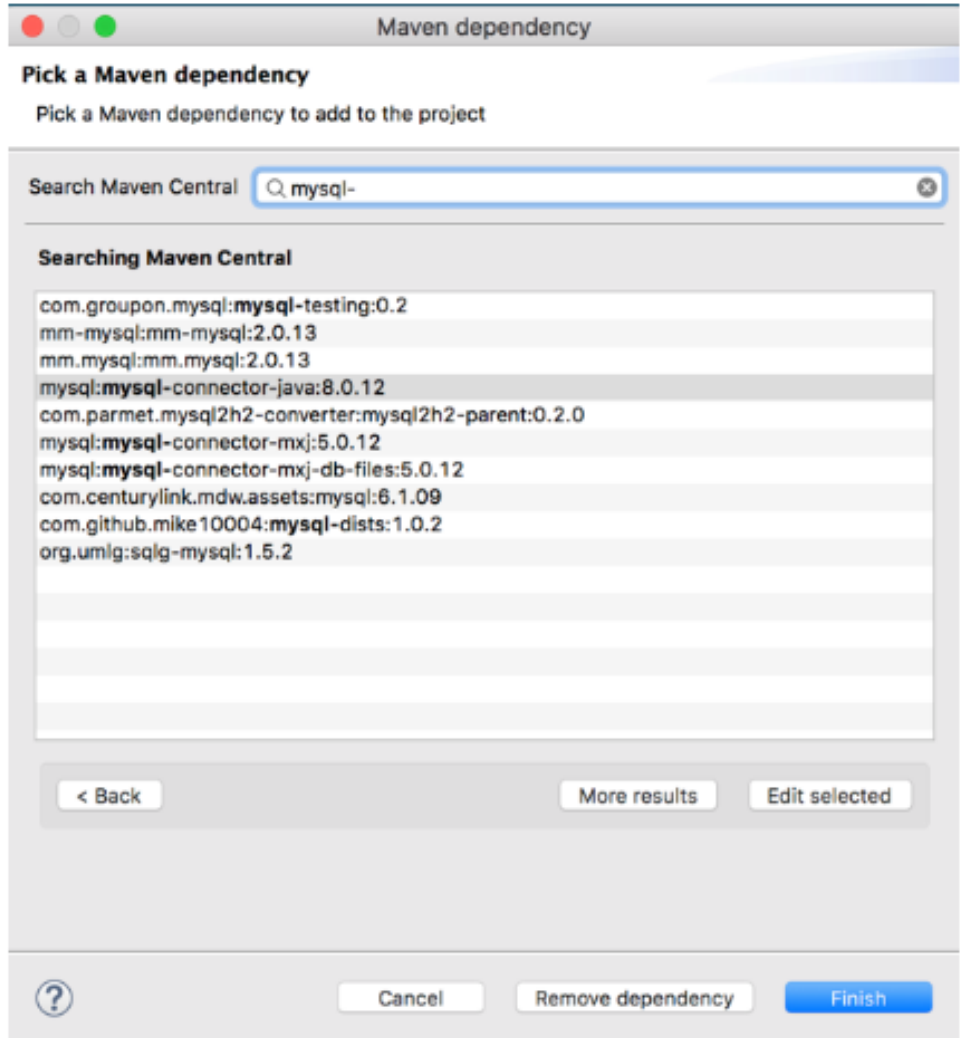


14. In the Maven dependency dialog box, locate the Search Maven Central text field.



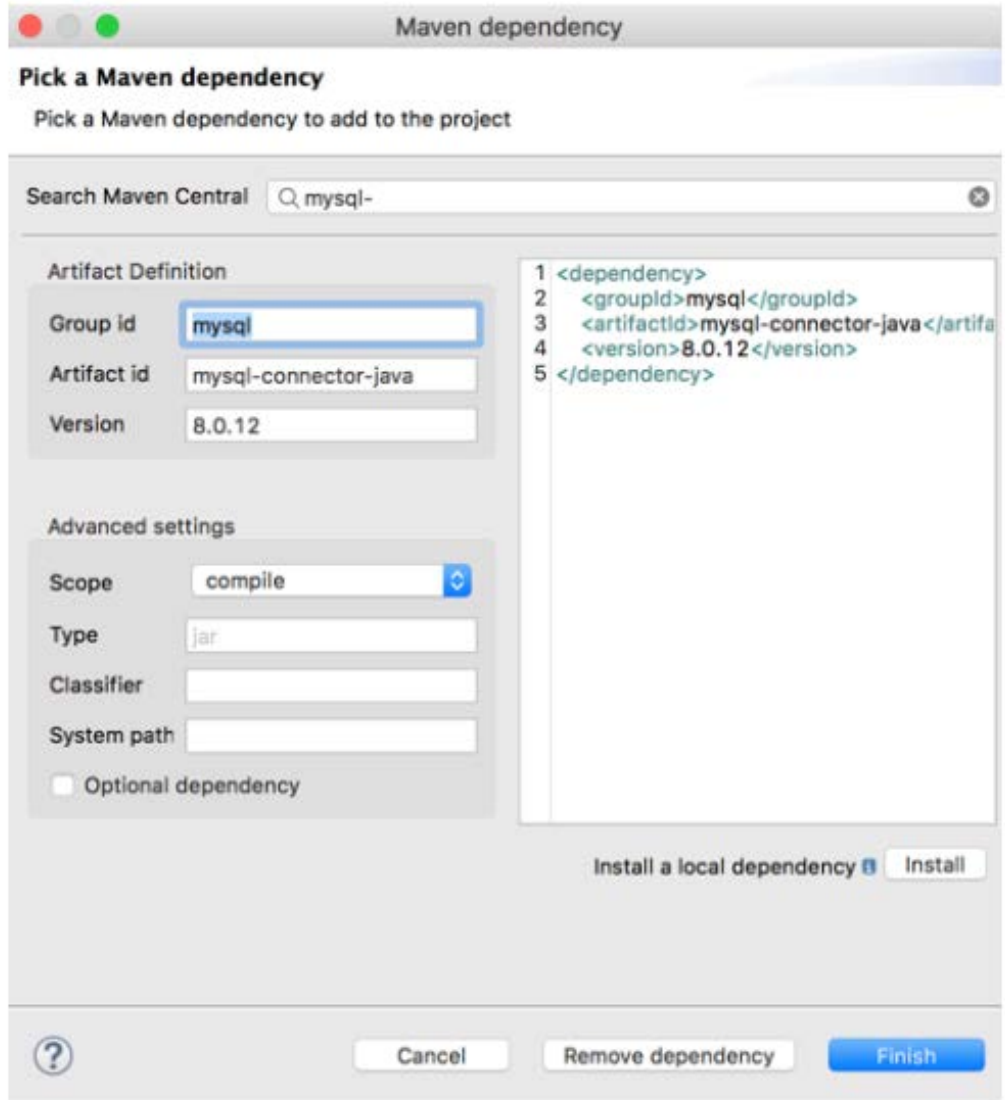
15. Enter mysql- in the Search Maven Central text field.

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



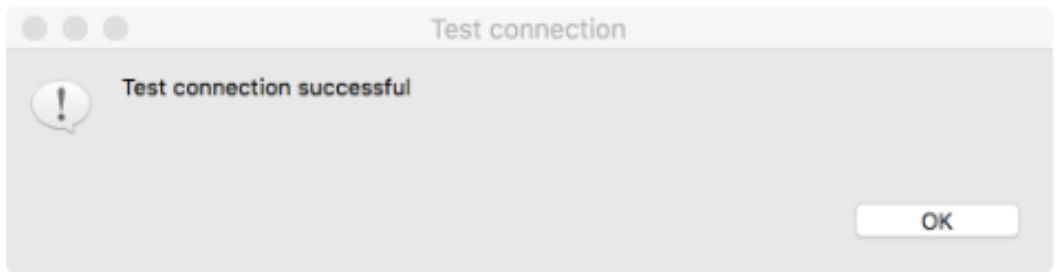
17. Click Edit selected.

18. Click Finish.



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

    Note: Make sure the connection succeeds before proceeding.



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

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

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

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

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

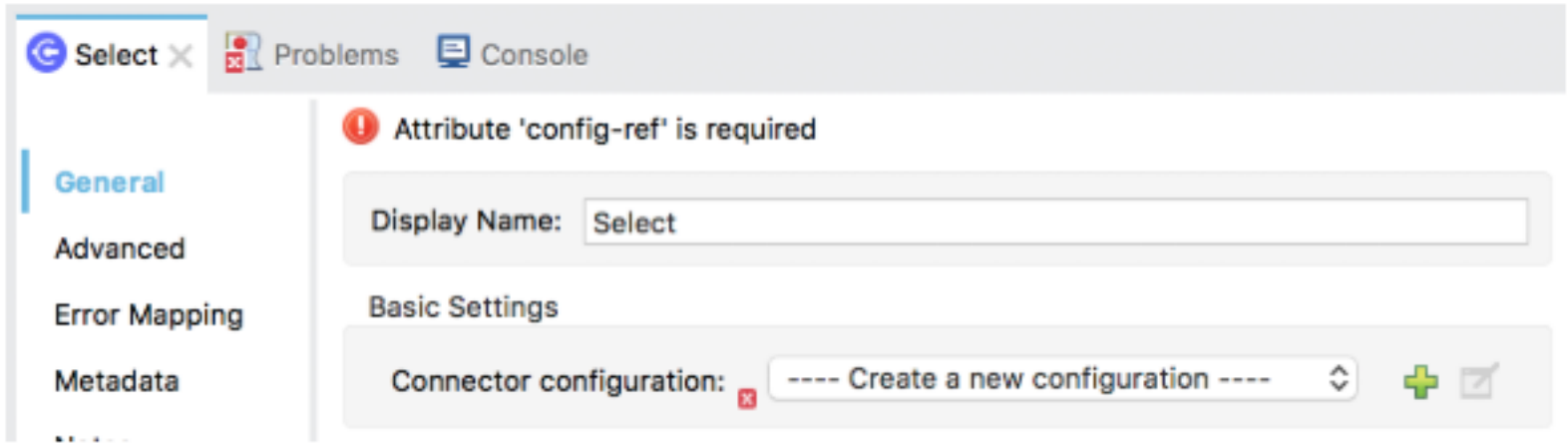
    Note: Replace X.X.X with the version of the JAR file, for example 1.6.2.

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

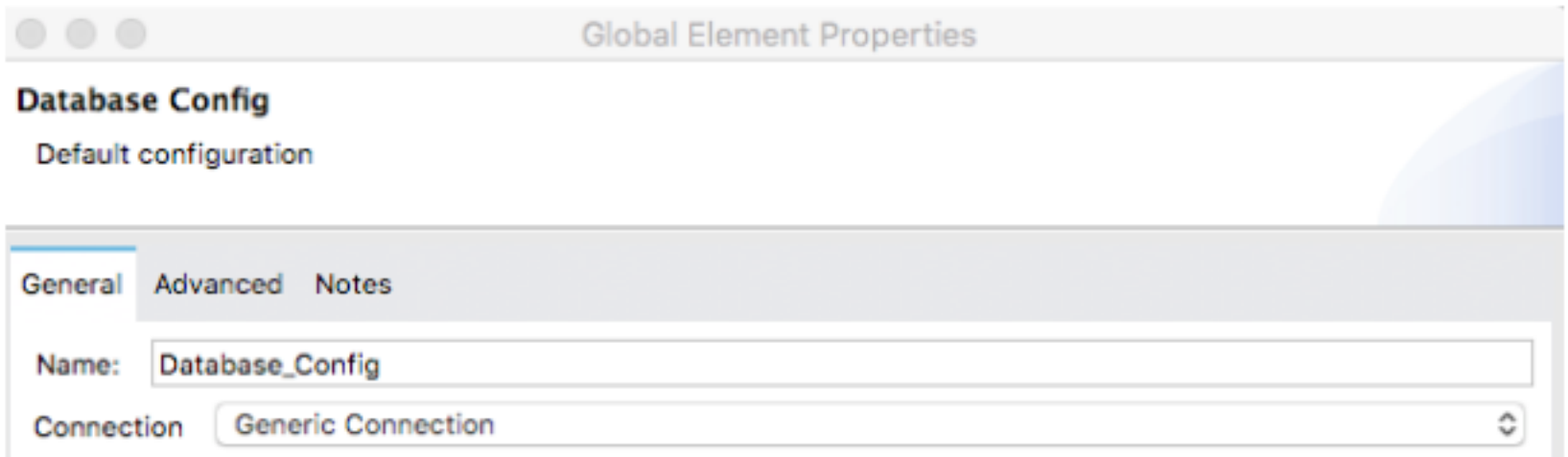
    ```
    java -jar mulesoft-training-services-X.X.X.jar --database.port=1530 --ws.port=9092 --
    spring.activemq.broker-url=tcp://localhost:61617 --server.port=9193
    ```

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

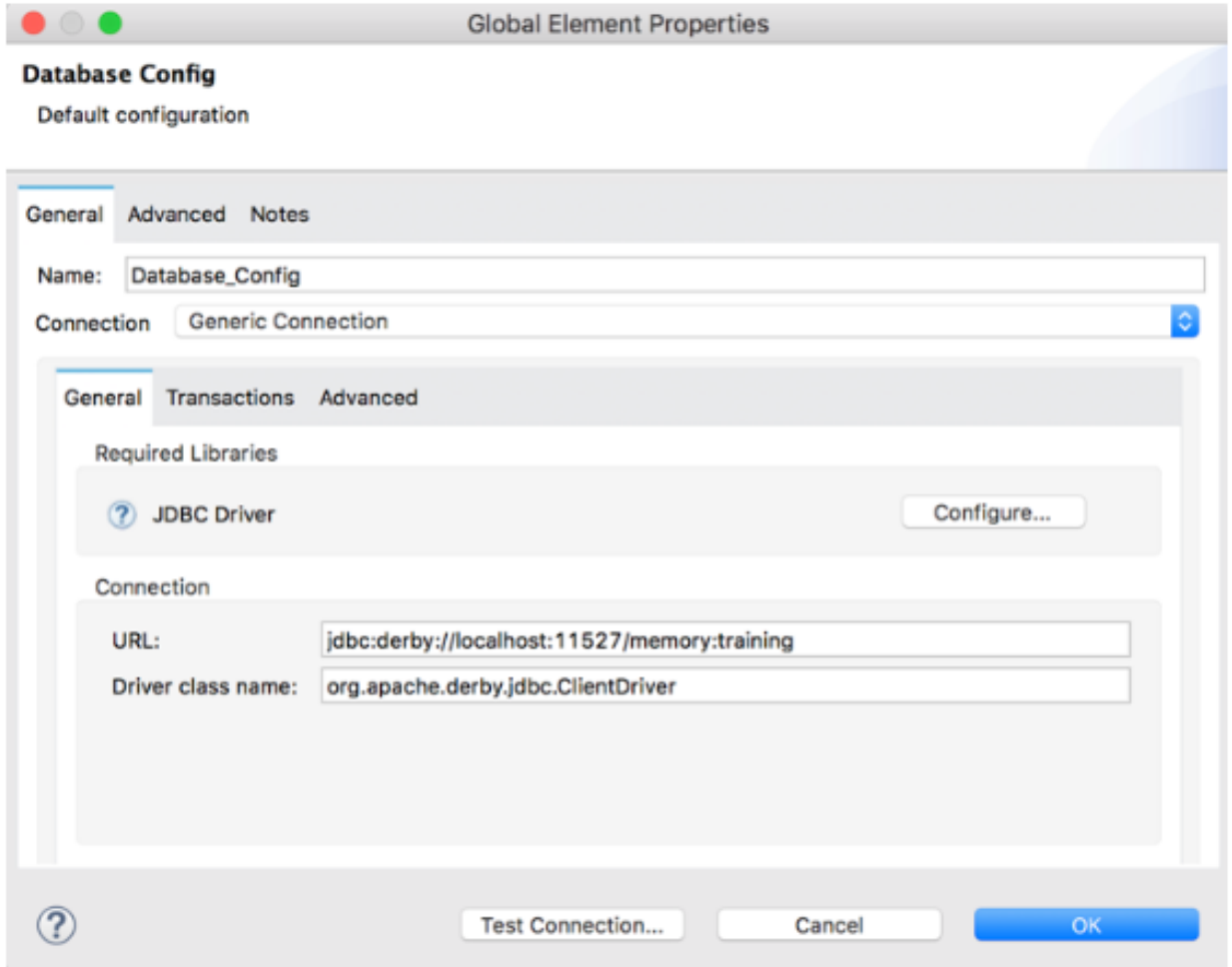
25. Return to Anypoint Studio.
26. In the Select properties view, click the Add button next to connector configuration.



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

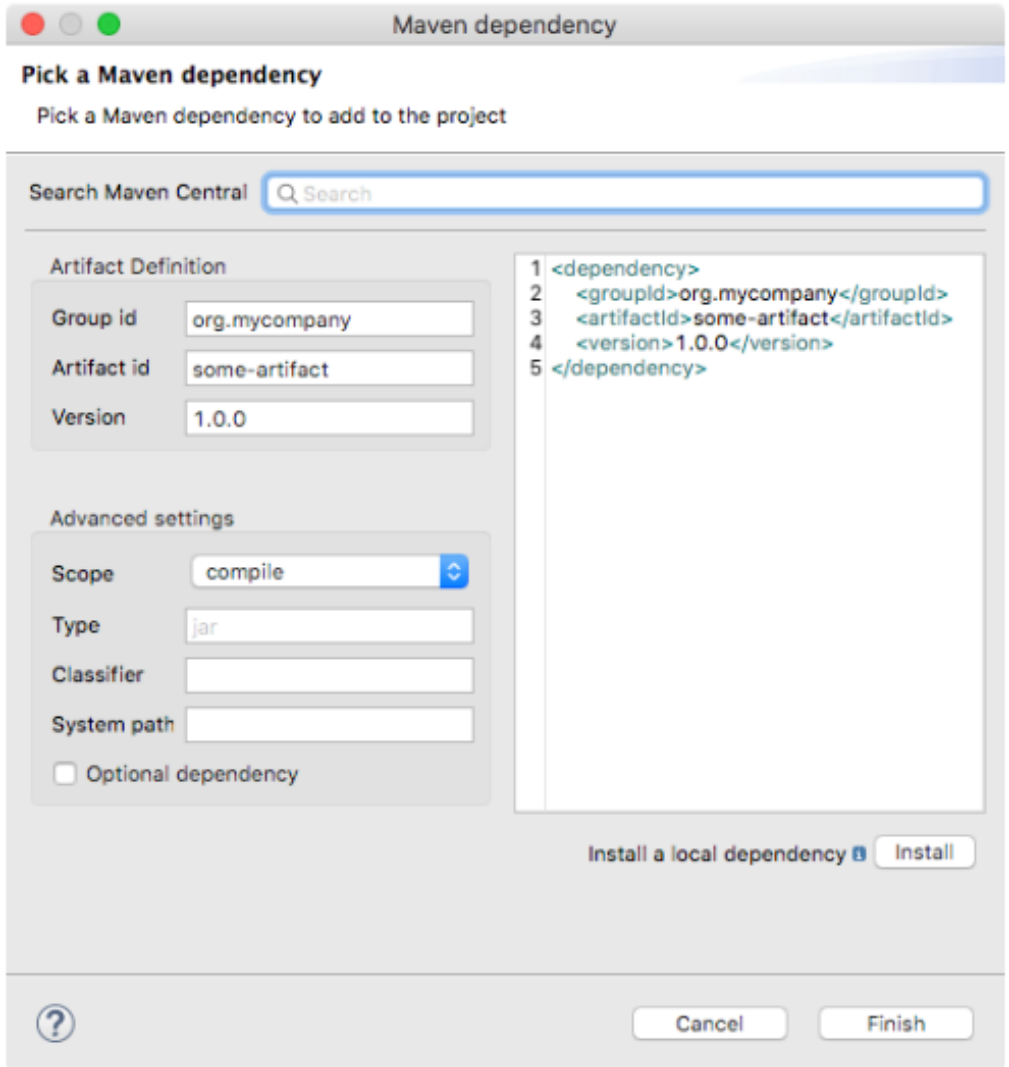


28. Set the URL and driver class name values to the values listed in the course snippets.txt file.



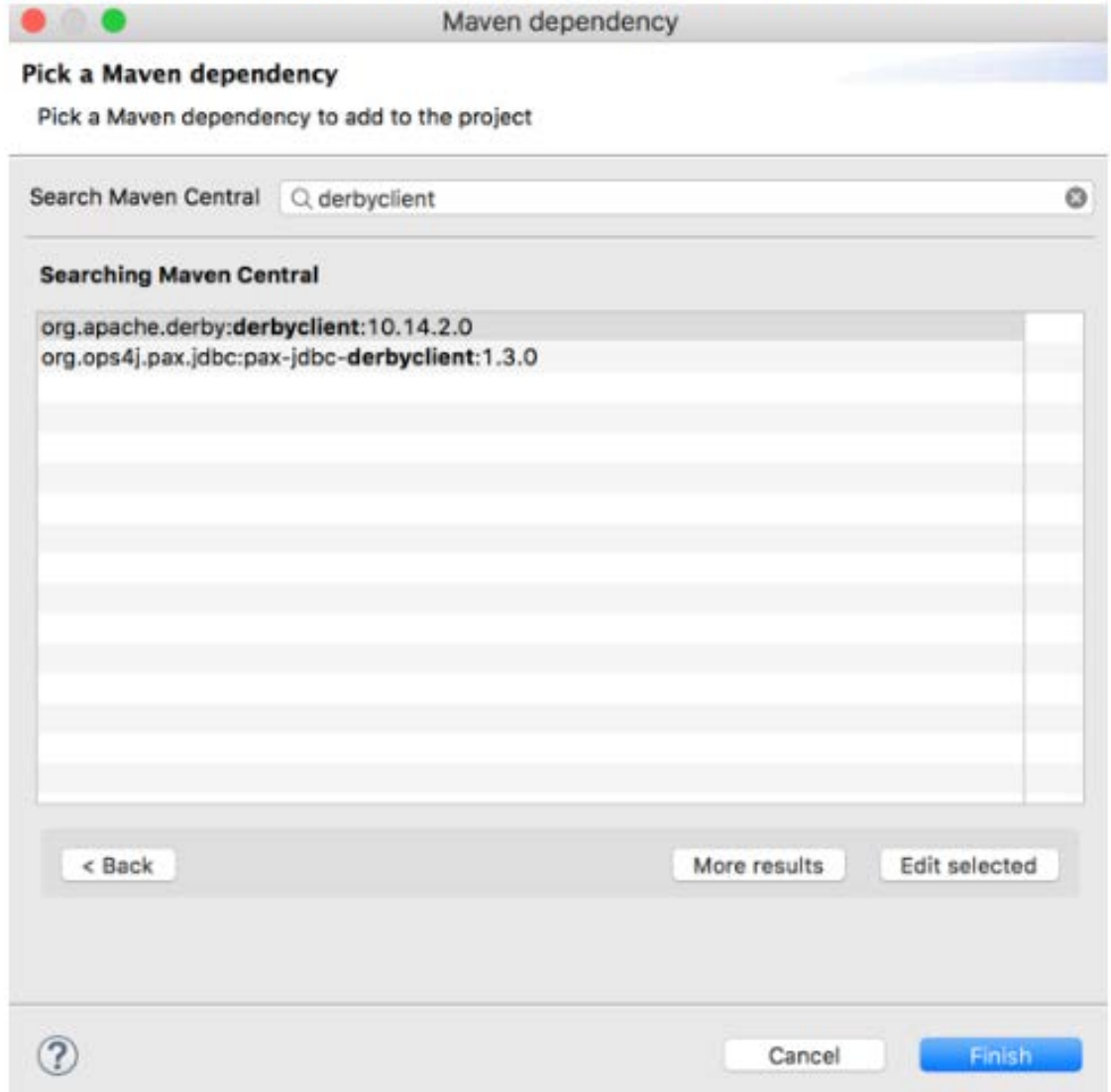
29. Click the Configure button next to JDBC Driver.
30. In the configure drop-down list, select Add Maven dependency.

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

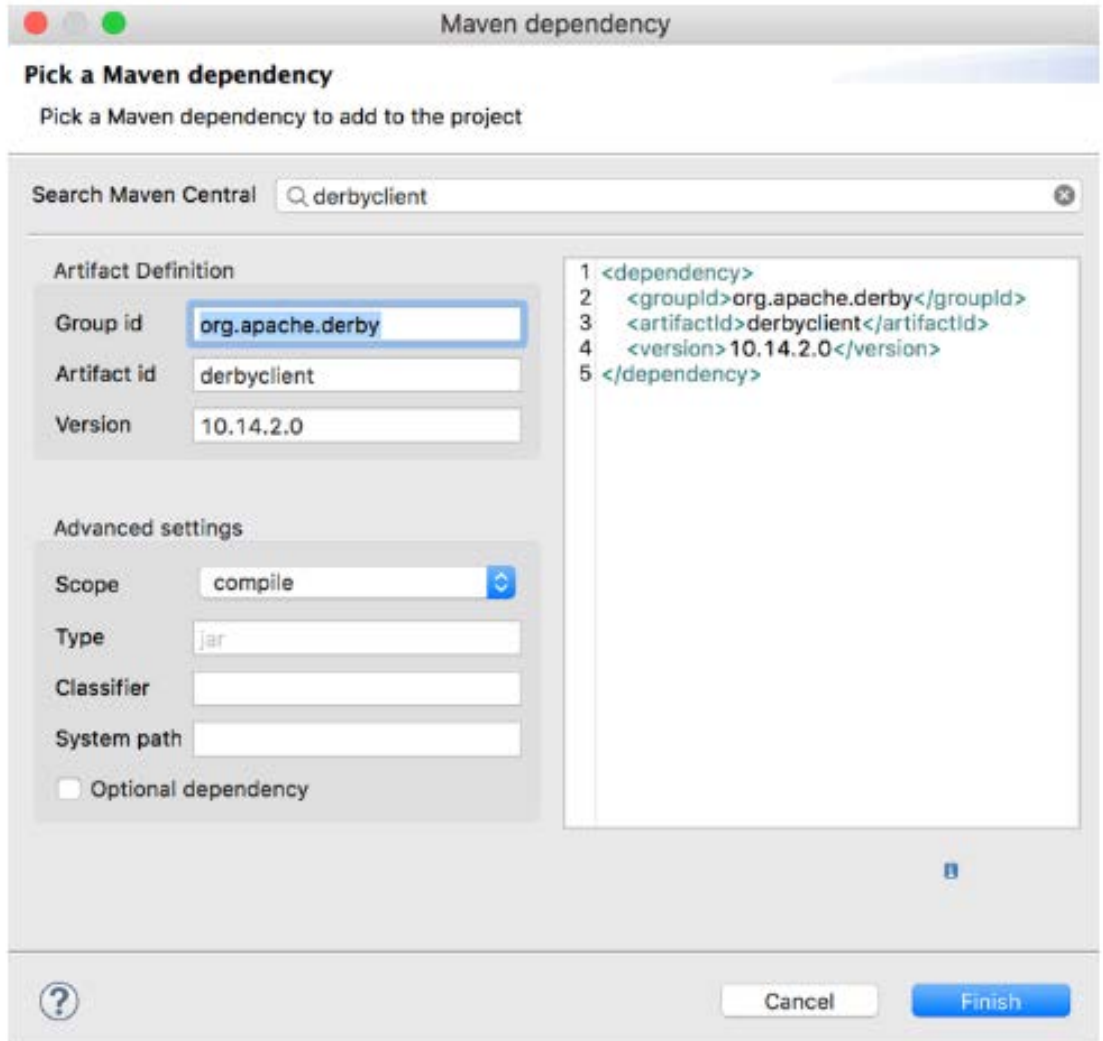


32. Enter derbyclient in the Search Maven Central text field.

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



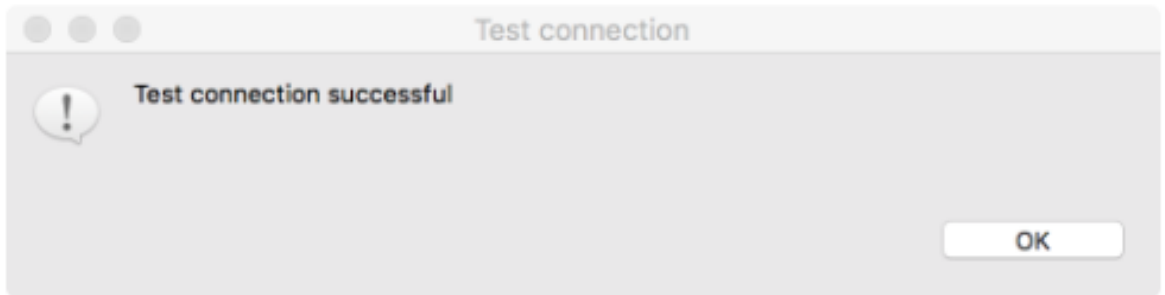
34. Click Edit selected.



35. Click Finish.

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

    Note: Make sure the connection succeeds before proceeding.



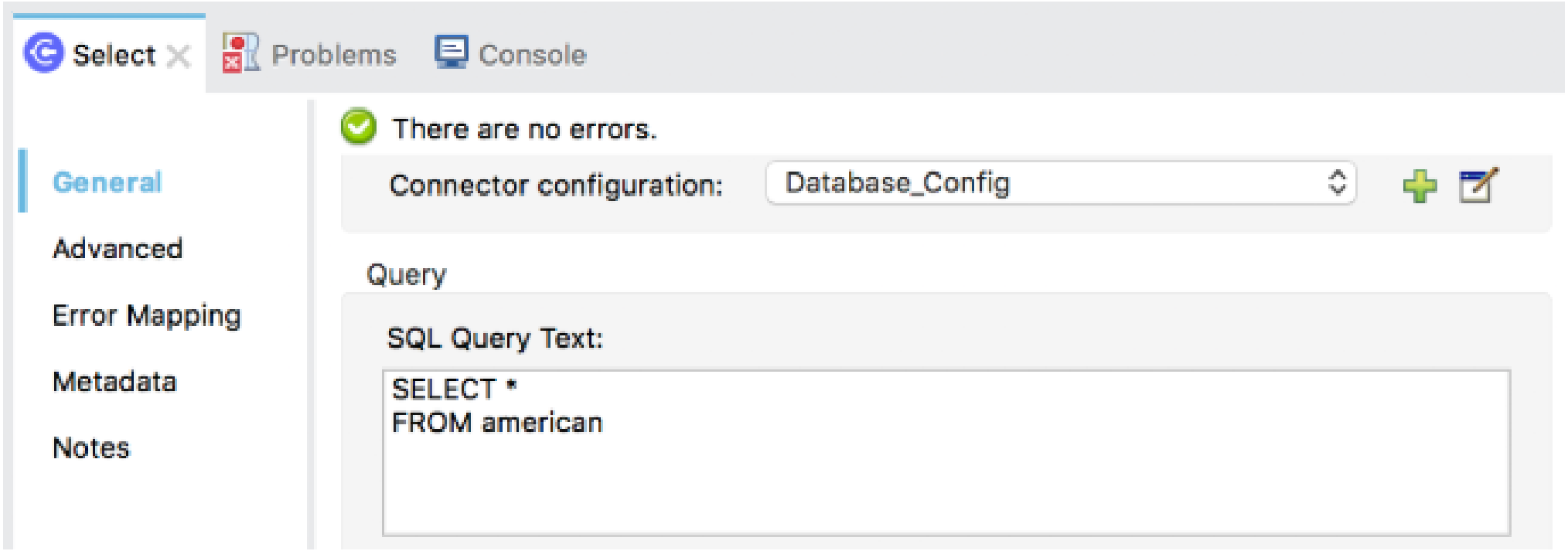
37. Click OK to close the dialog box.
38. Click OK to close the Global Element Properties dialog box.

Write a query to return all flights

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

    SELECT *

    FROM american



Test the application

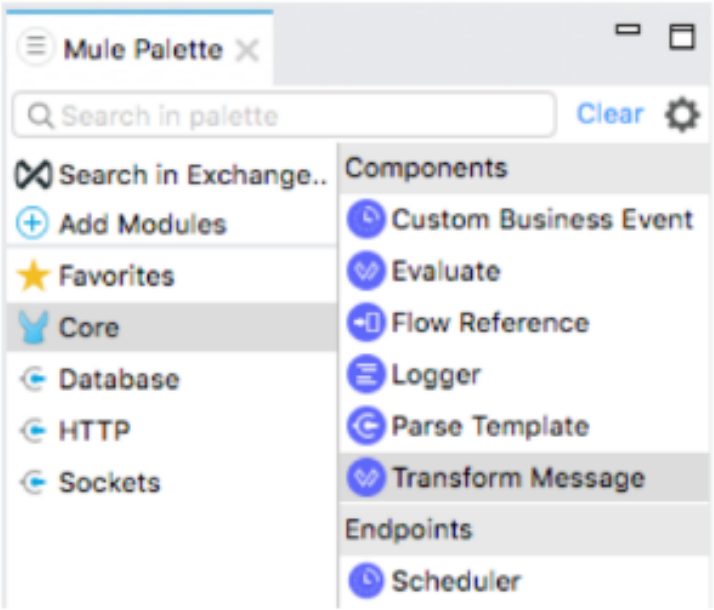
40. Run the project.
41. In the Save changes dialog box, select Yes.
42. Watch the console and wait for the application to start.
43. Once the application has started, return to Advanced REST Client.
44. In Advanced REST Client, make another request to http://localhost:8081/flights; you should get some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

Transforming data

Transforming data

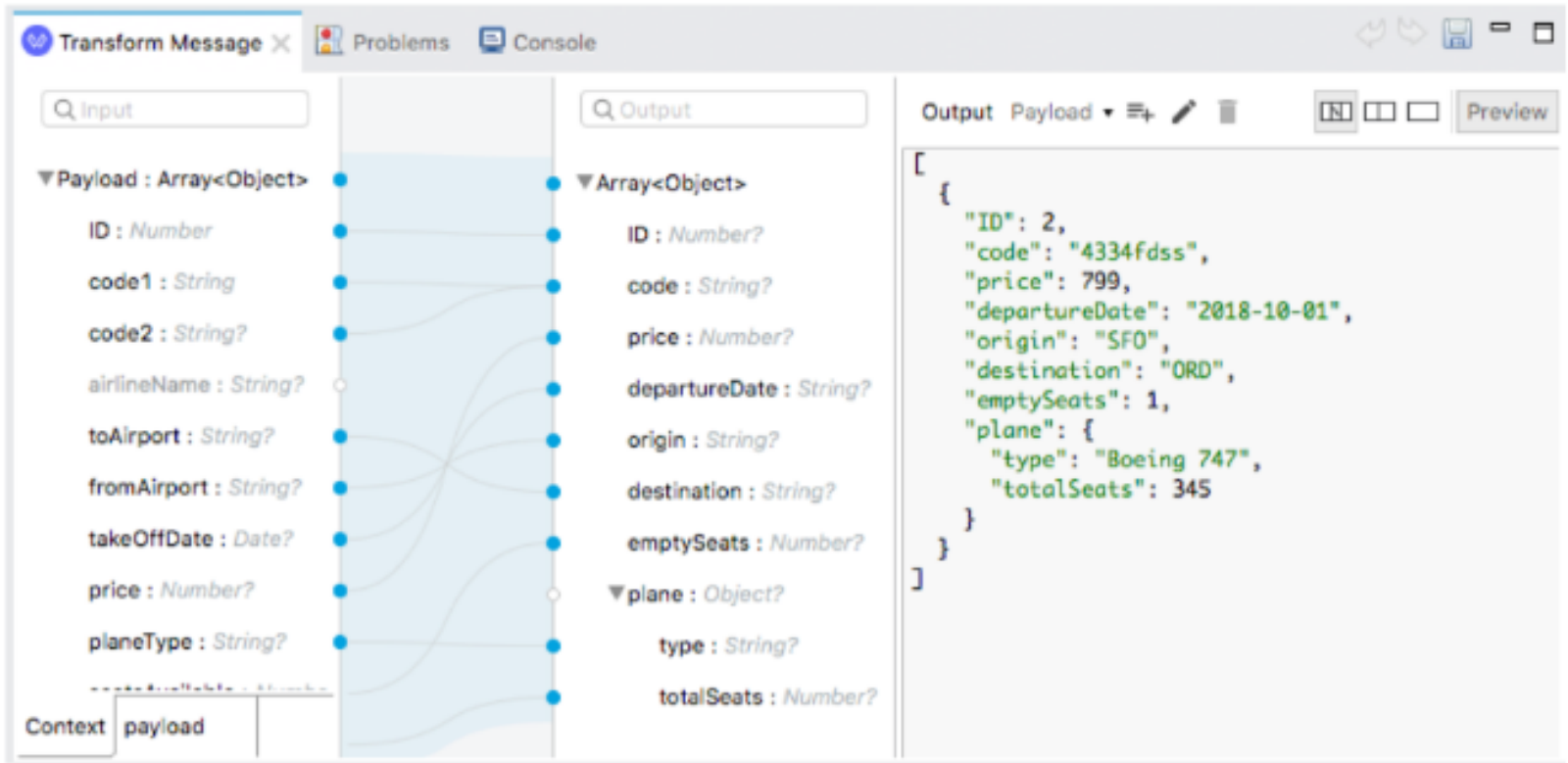
- **DataWeave 2.0** is the expression language for Mule to access, query, and transform **Mule 4** event data
  - DataWeave was introduced and used in Module 2

- In Studio, use **Transform Message** component for transformations
  - Graphical interface with payload-aware development



Walkthrough 4-3: Transform data

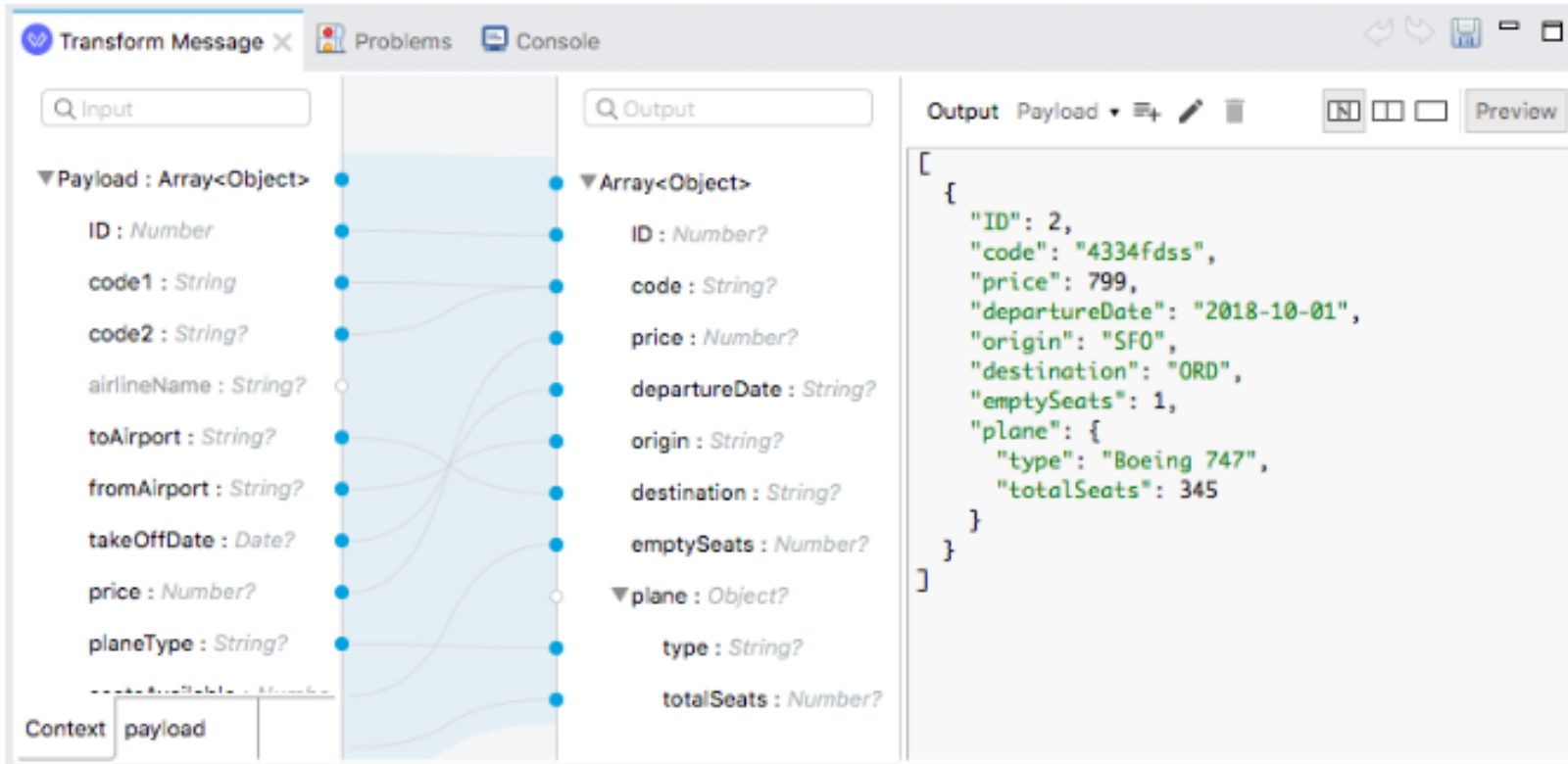
- Use the Transform Message component
- Use the DataWeave visual mapper to change the response to a different JSON structure



Walkthrough 4-3: Transform data

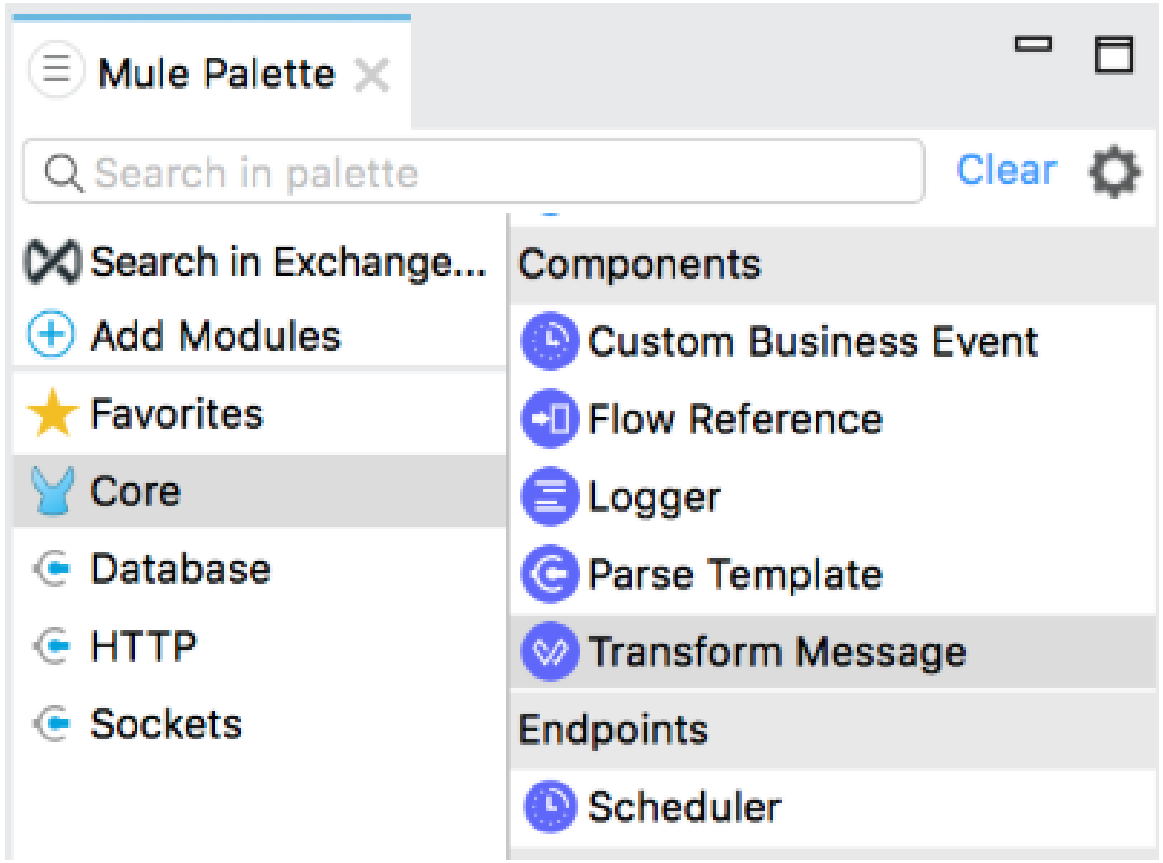
In this walkthrough, you transform and display the flight data into JSON. You will:

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

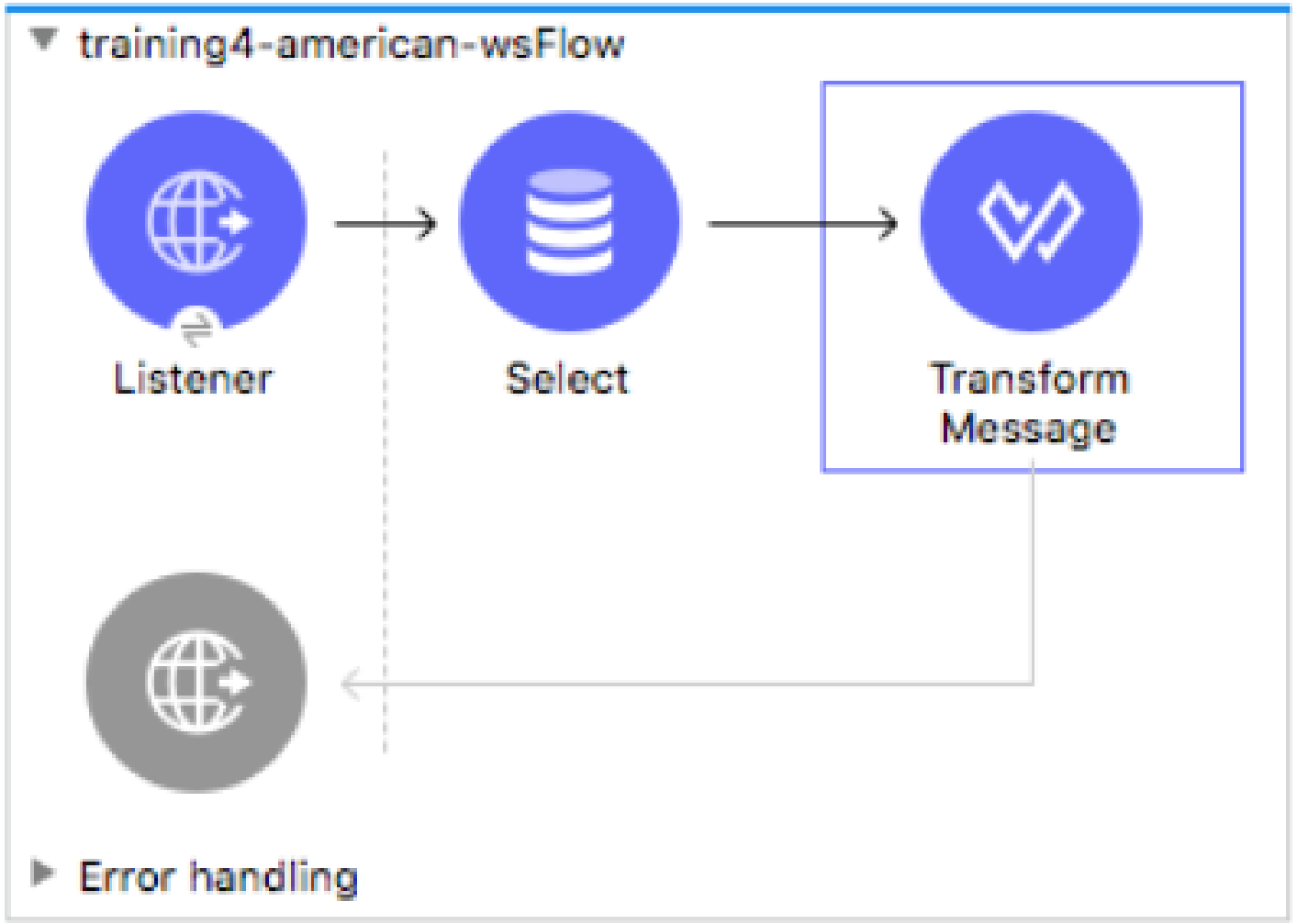


Add a Transform Message component

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



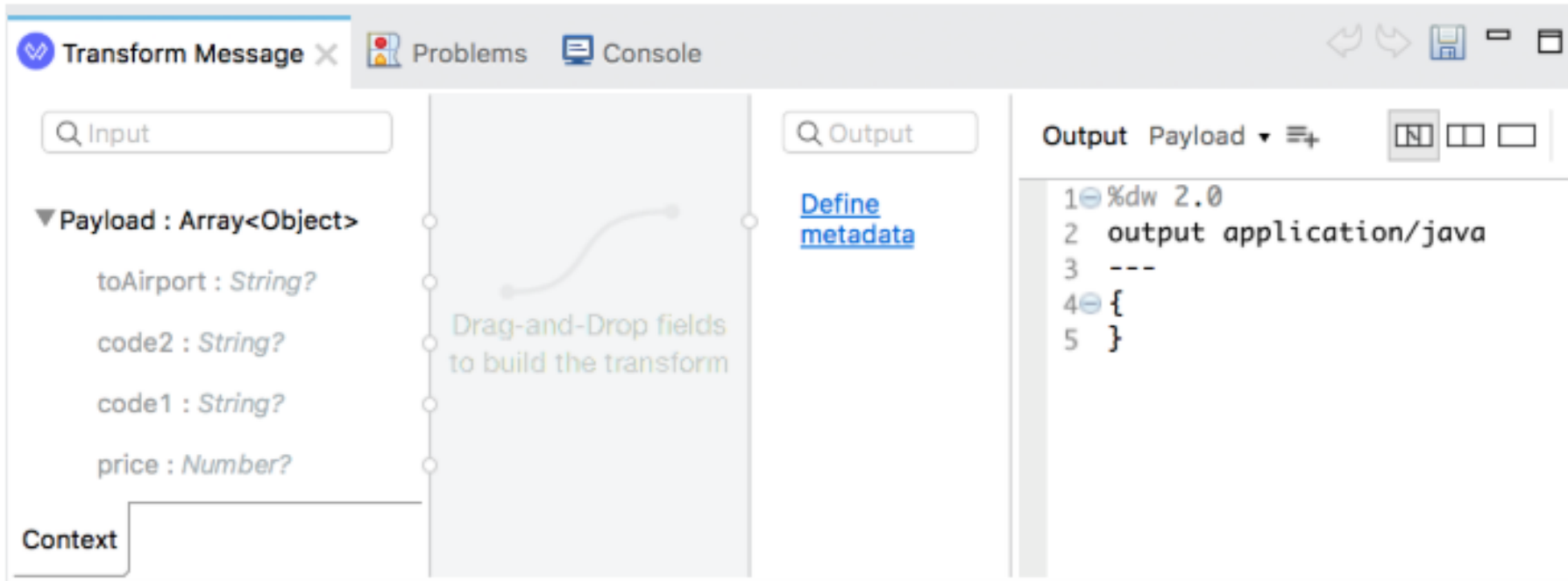
3. Drag the Transform Message and drop it after the Select processor.



Review metadata for the transformation input

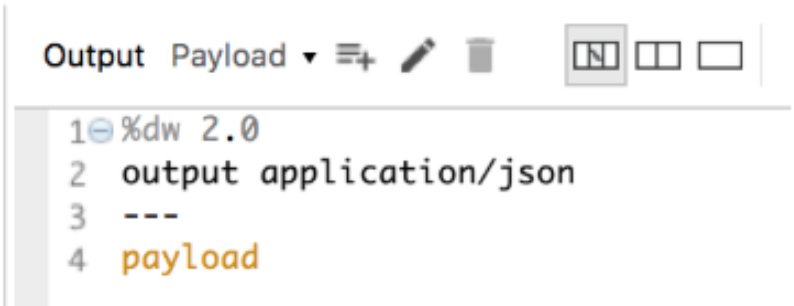
4. In the Transform Message properties view, look at the input section and review the payload metadata.

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



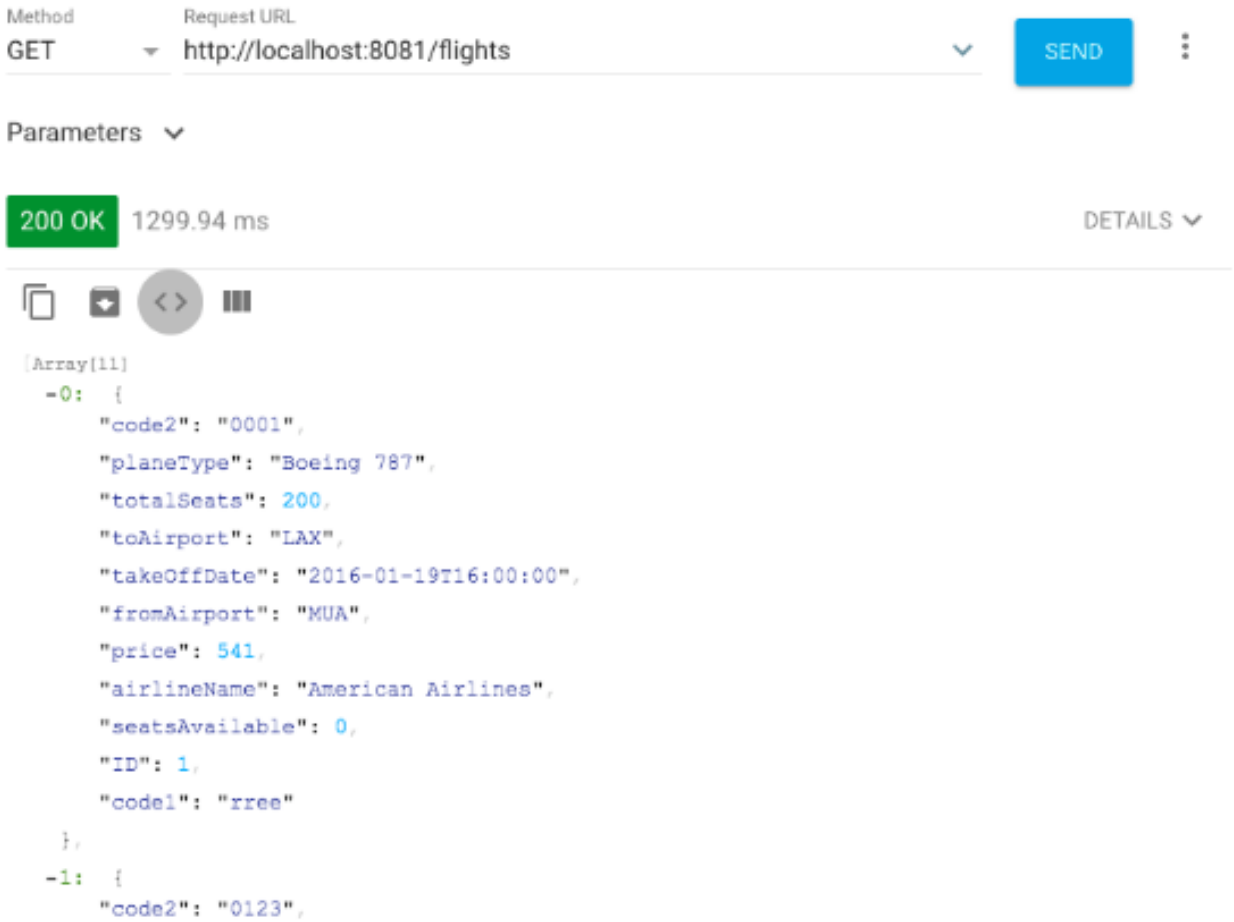
Return the payload as JSON

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



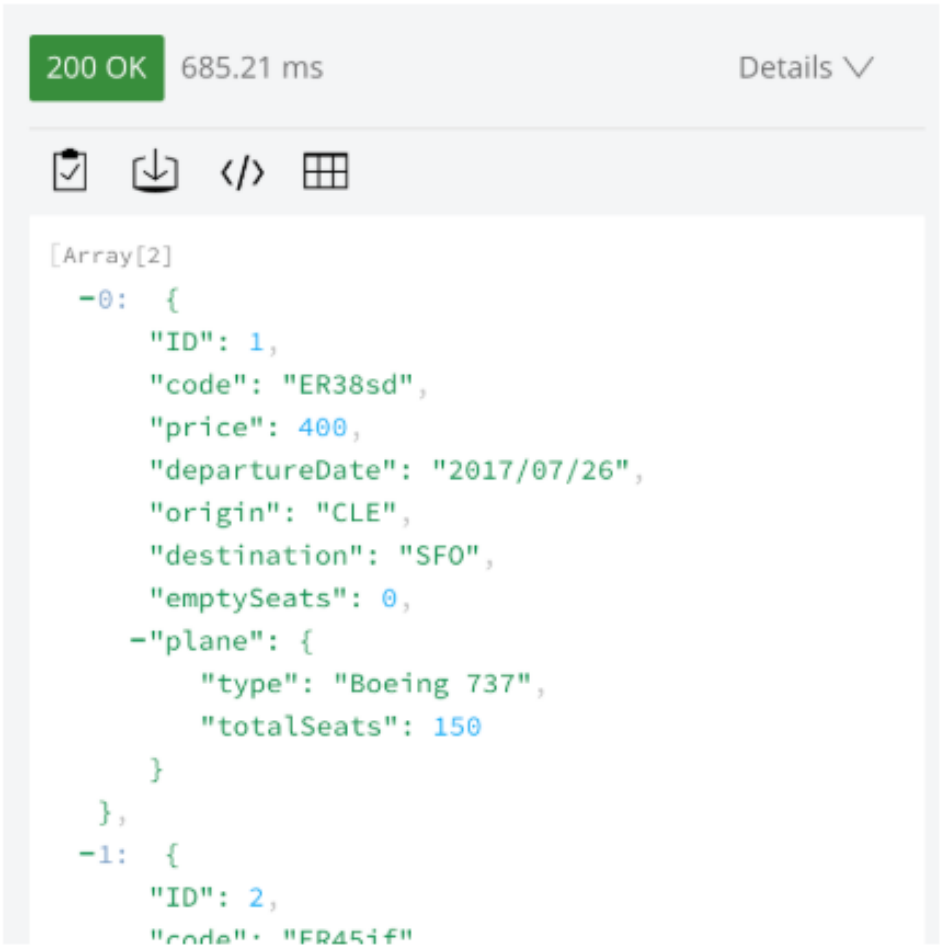
Test the application

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.



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

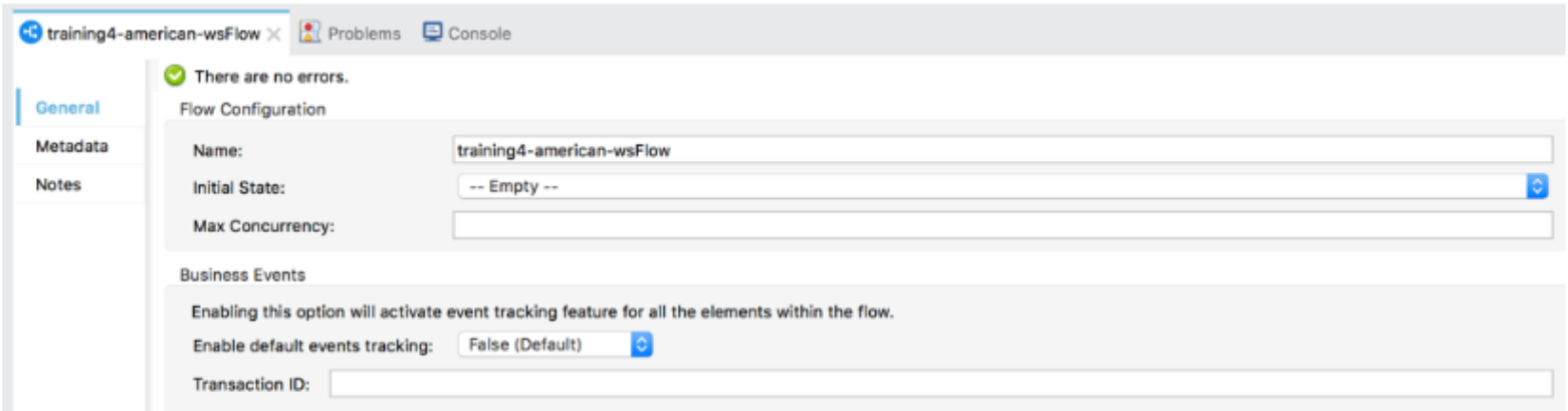
8. Return to your American Flights API in Exchange.
9. Look at the example data returned for the /flights GET method.



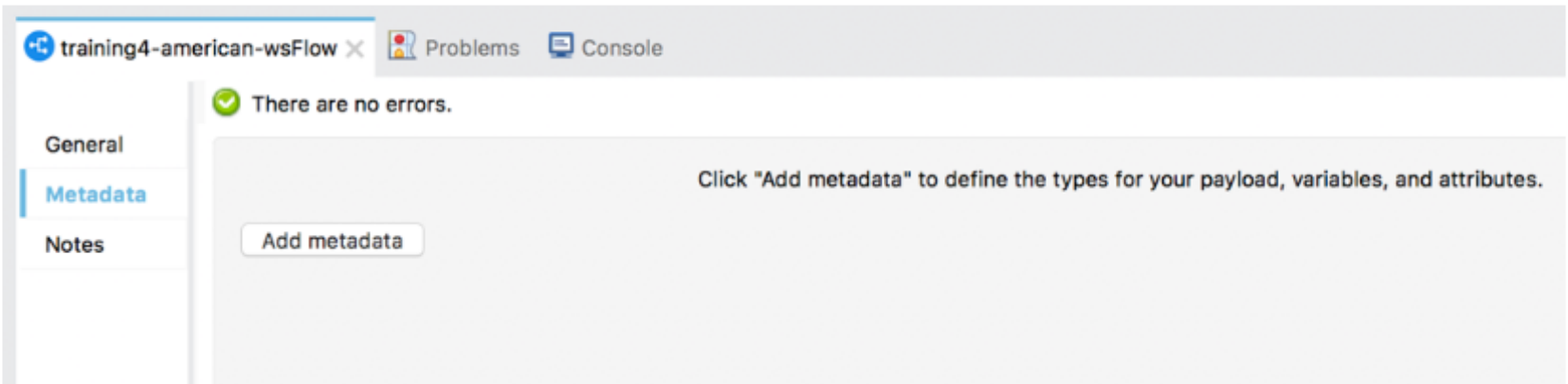
10. Notice that the structure of the JSON being returned by the Mule application does not match this JSON.

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

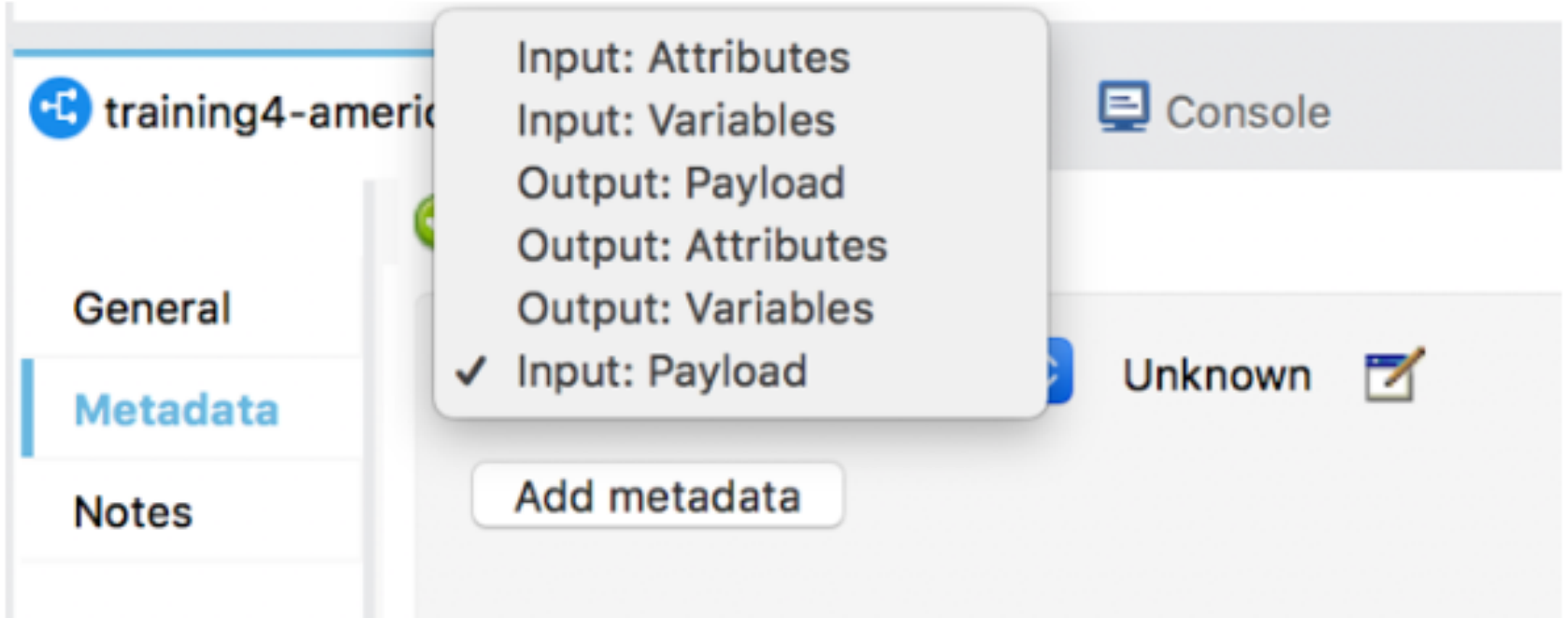
11. Return to Anypoint Studio.
12. In the canvas, click training4-american-wsFlow name.
13. In the training-american-wsFlow properties view, click the Metadata tab.



14. Click the Add metadata button.

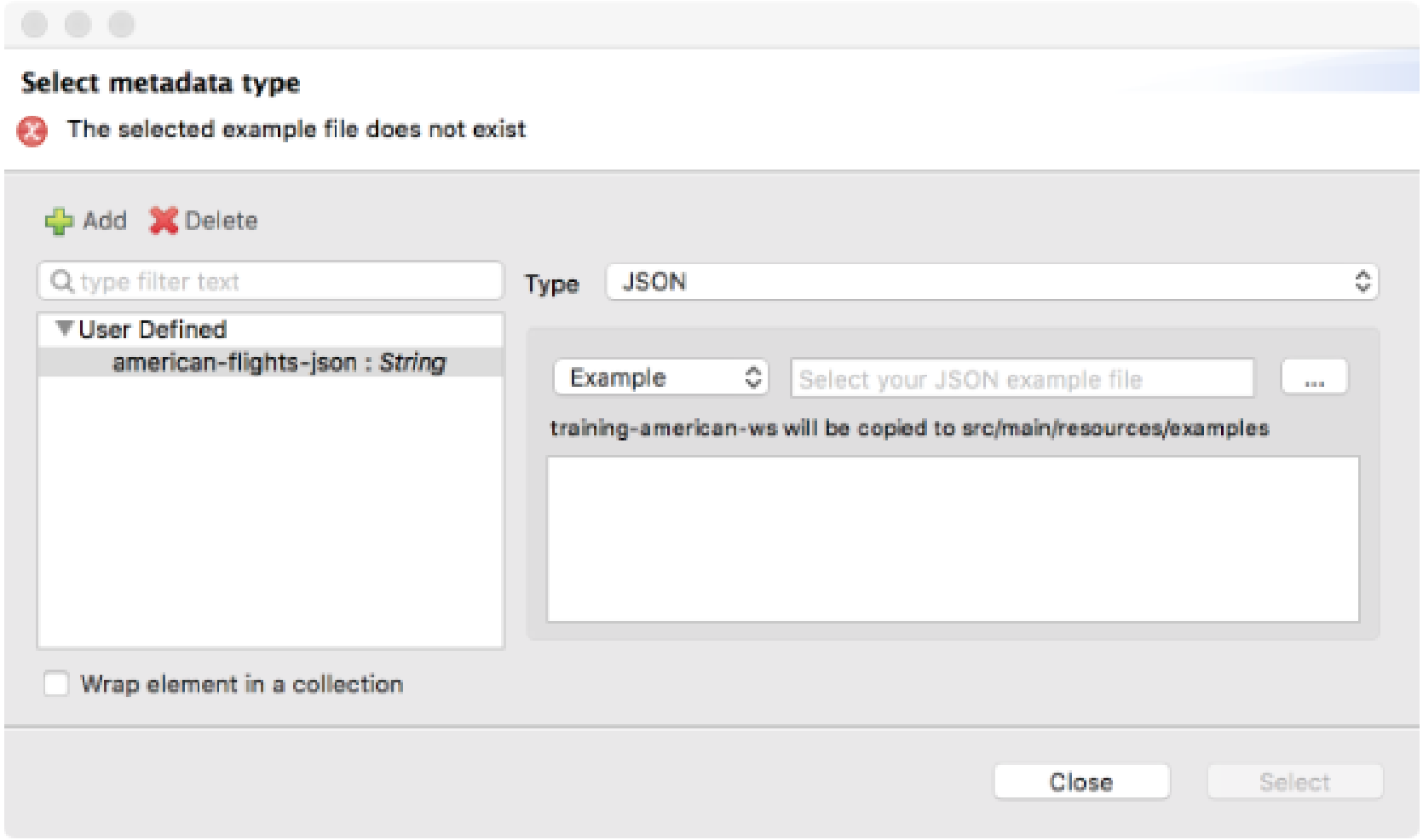


15. Change the drop-down menu to Output: Payload.



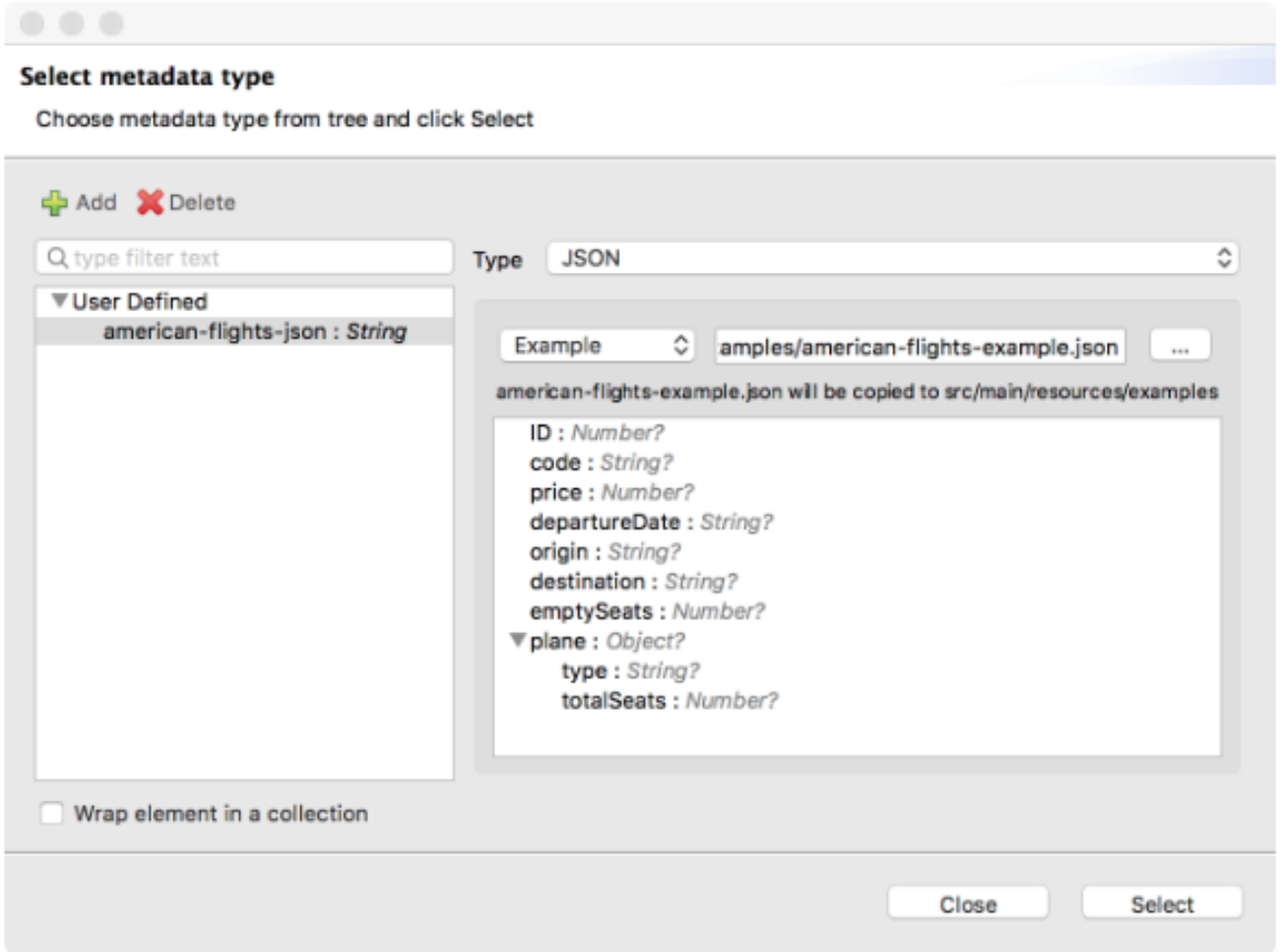
16. Click the Edit button.
17. In the Select metadata type dialog box, click the Add button.
18. In the Create new type dialog box, set the type id to american_flights_json.
19. Click Create type.
20. Back in the Set metadata type dialog box, set the type to JSON.

21. Change the Schema selection to Example.



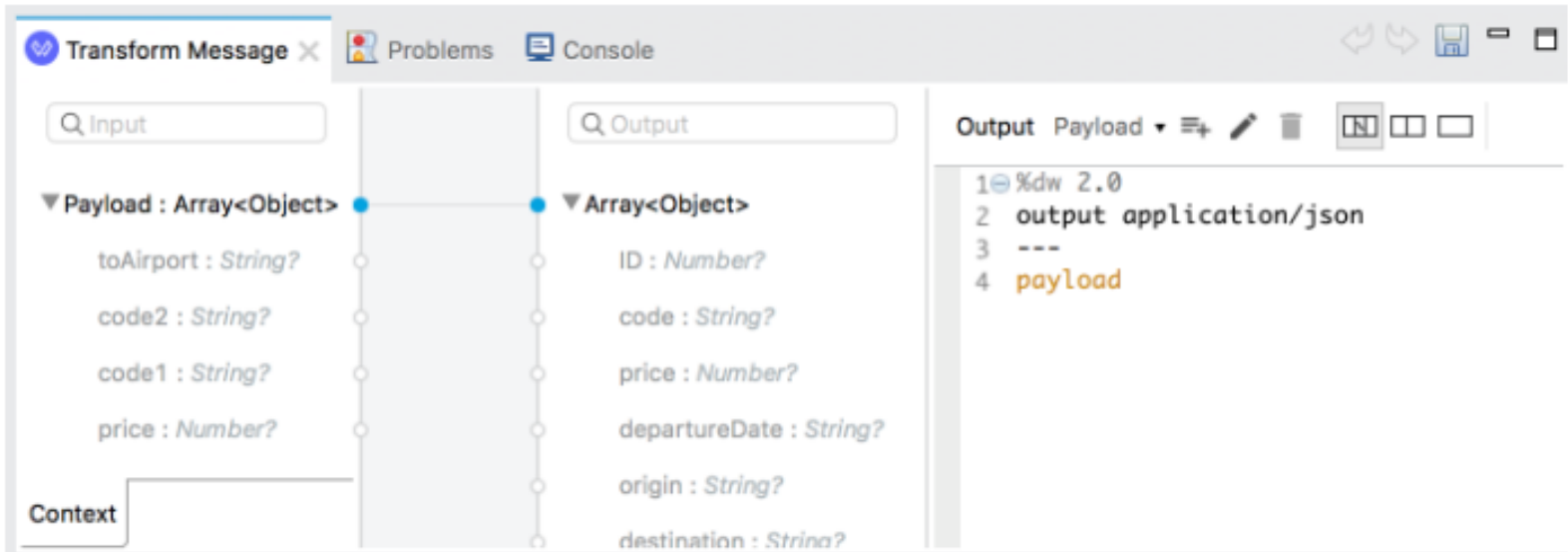
22. Click the browse button and navigate to the course student files.

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



24. Click Select.

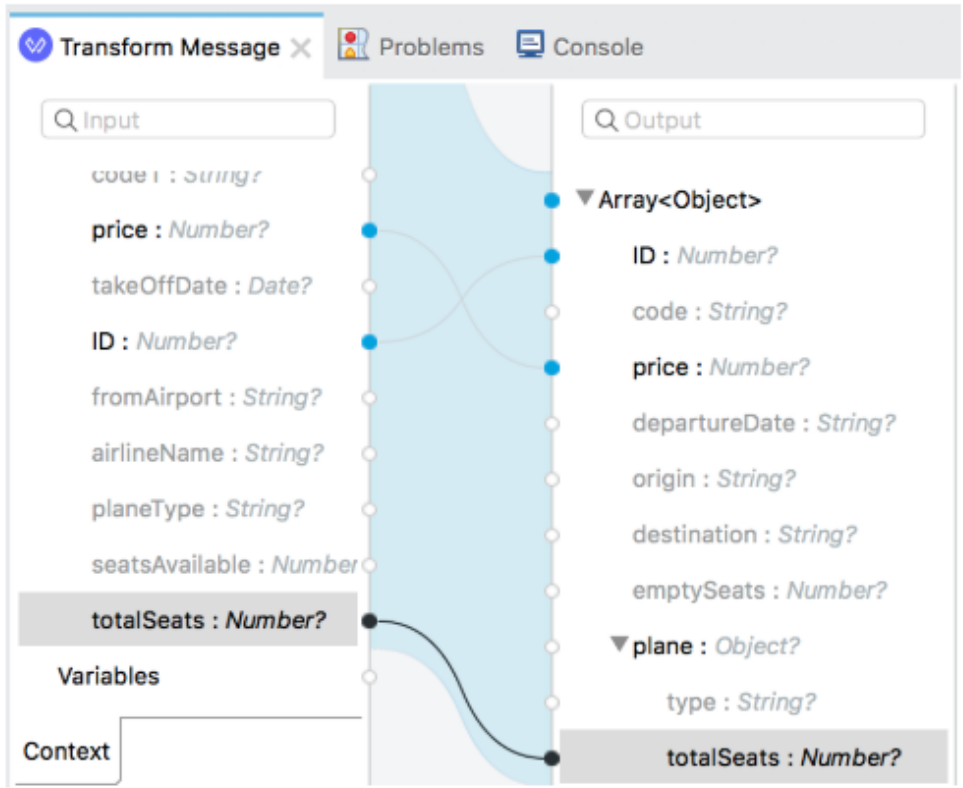
25. In the training4-american-wsFlow, click the Transform Message component; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

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

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

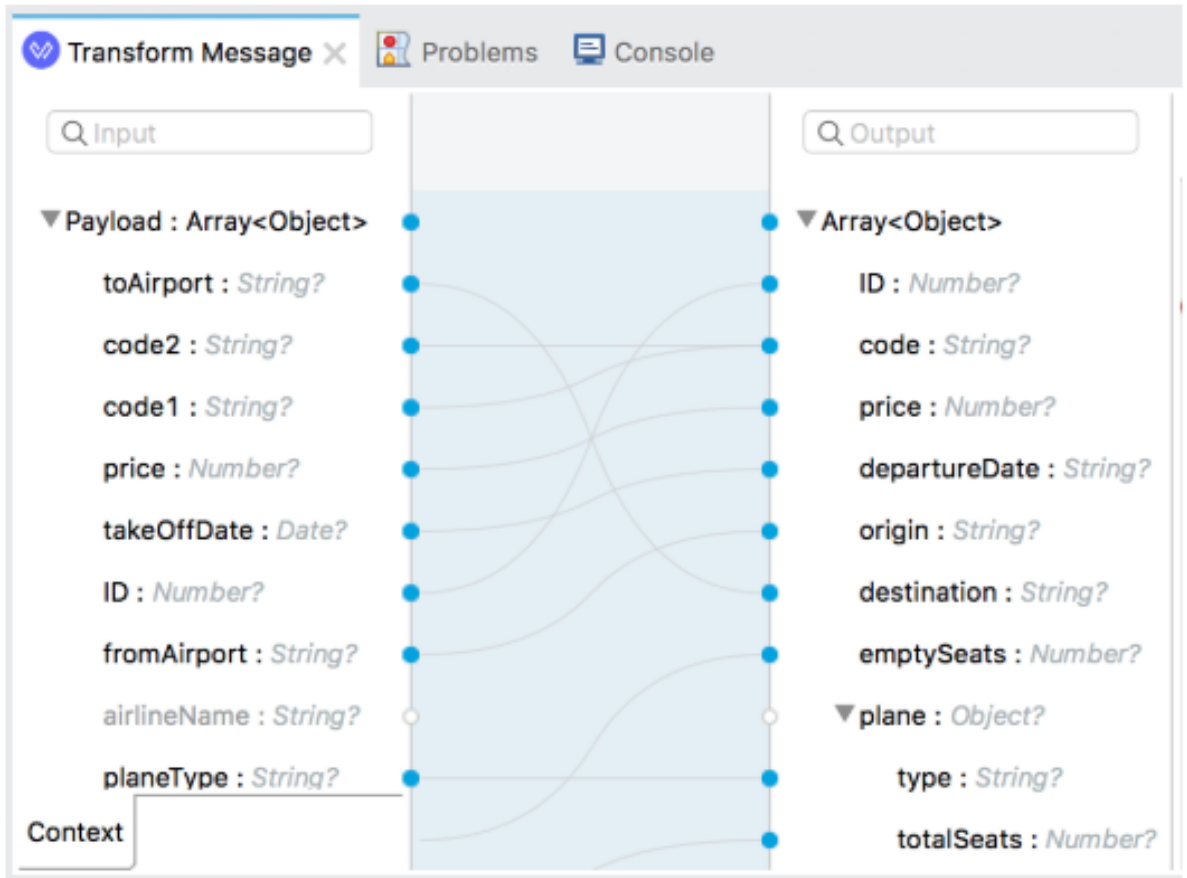


27. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

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

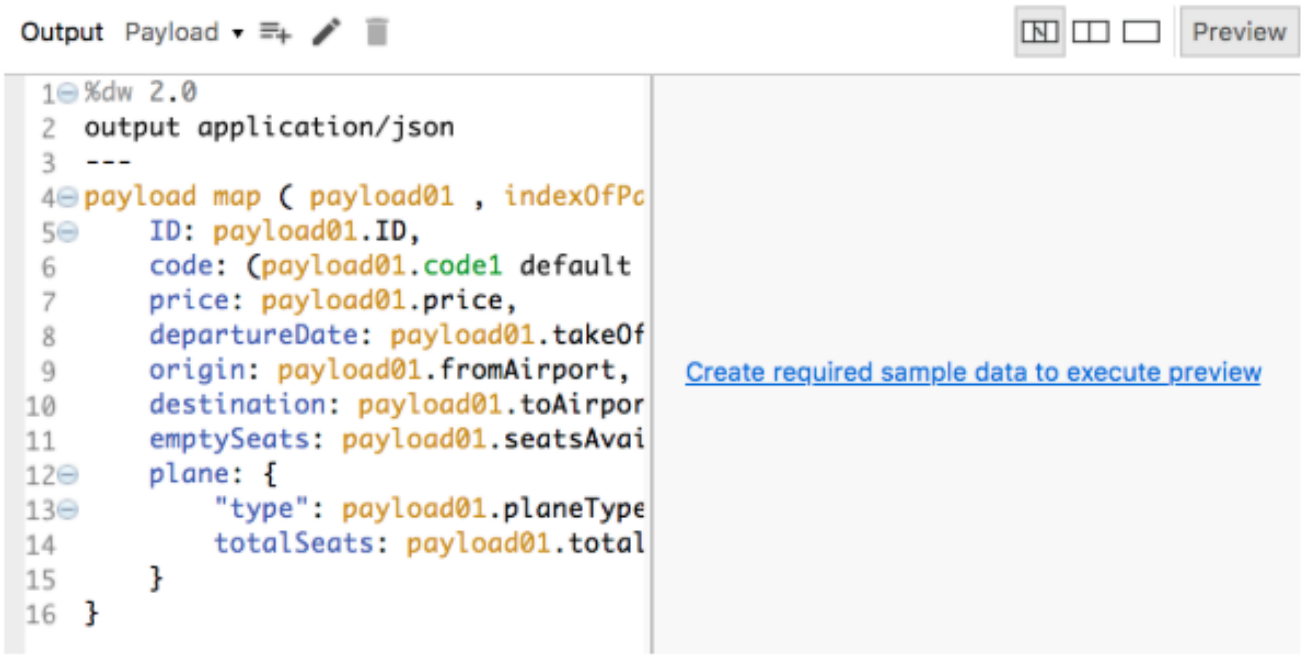
28. Concatenate two fields by dragging them from the input section and dropping them on the same field in the output section.

- code1 to code
- code2 to code



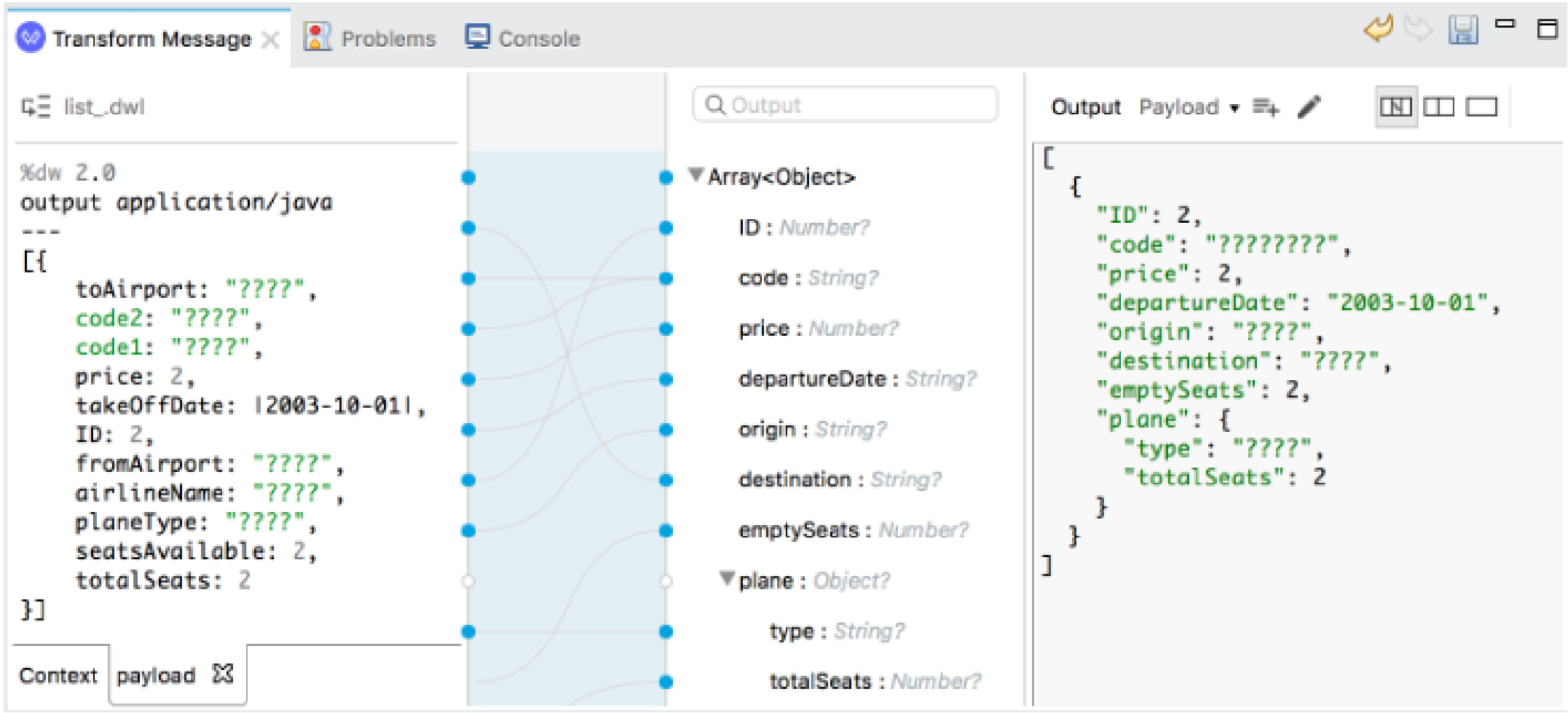
Add sample data (optional)

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



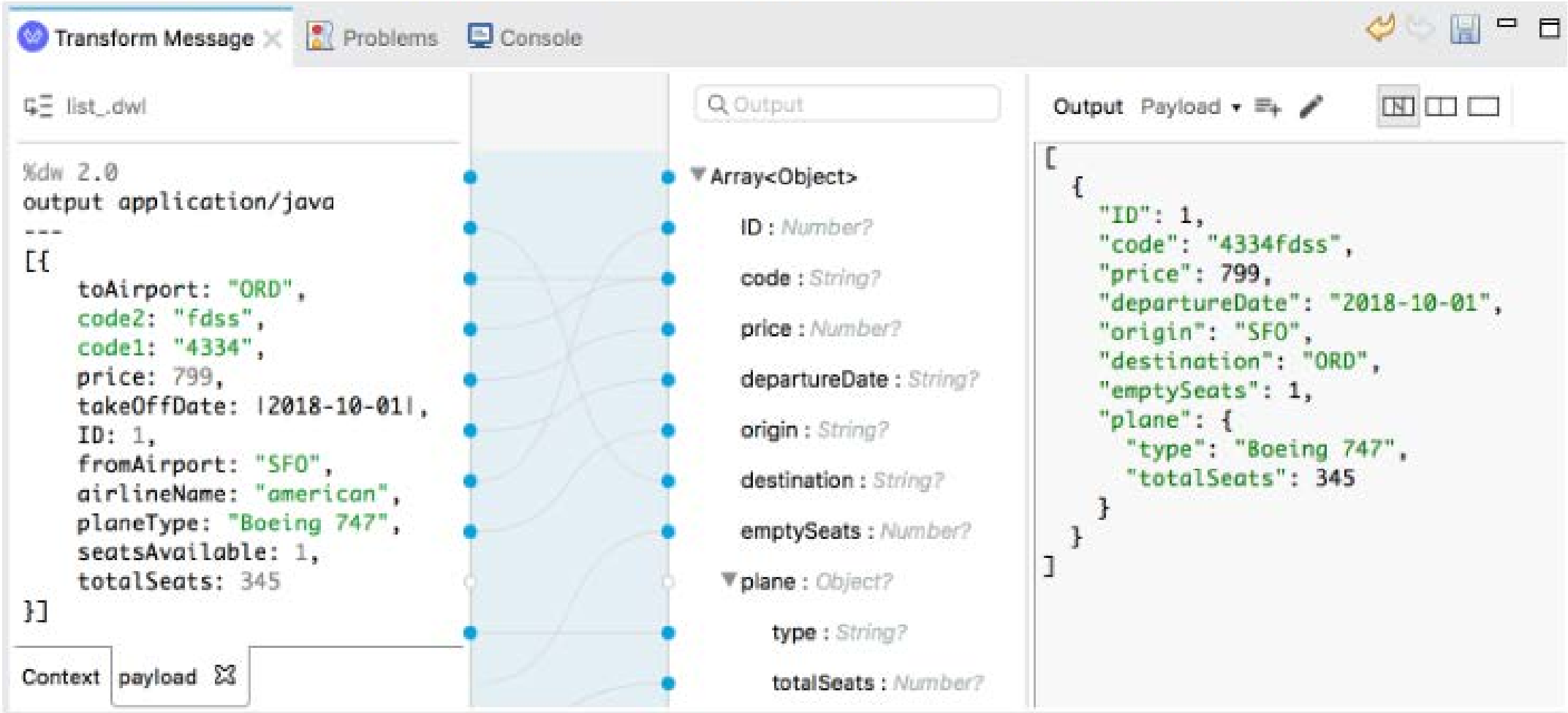
31. Look at the input section, you should see a new tab called payload with sample data generated from the input metadata.

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



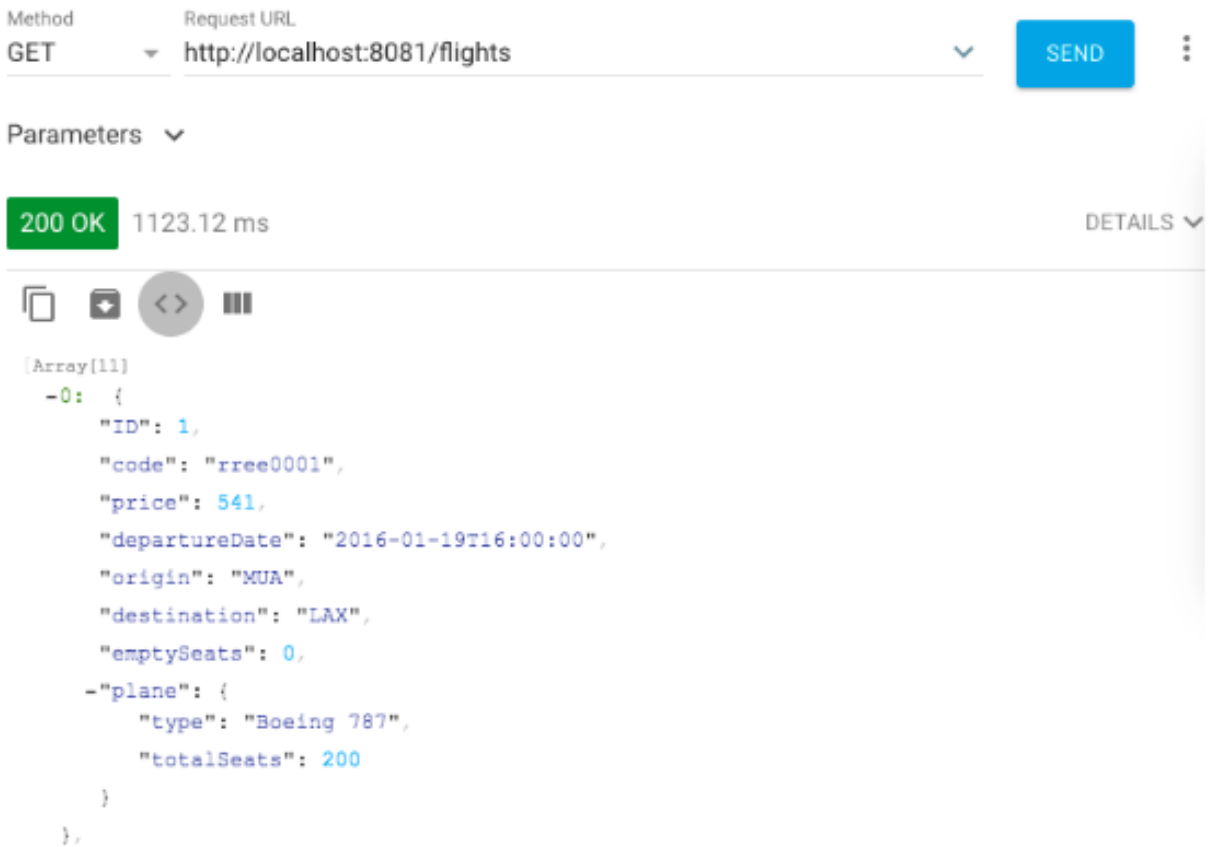
33. In the input section, replace all the ???? with sample values.

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



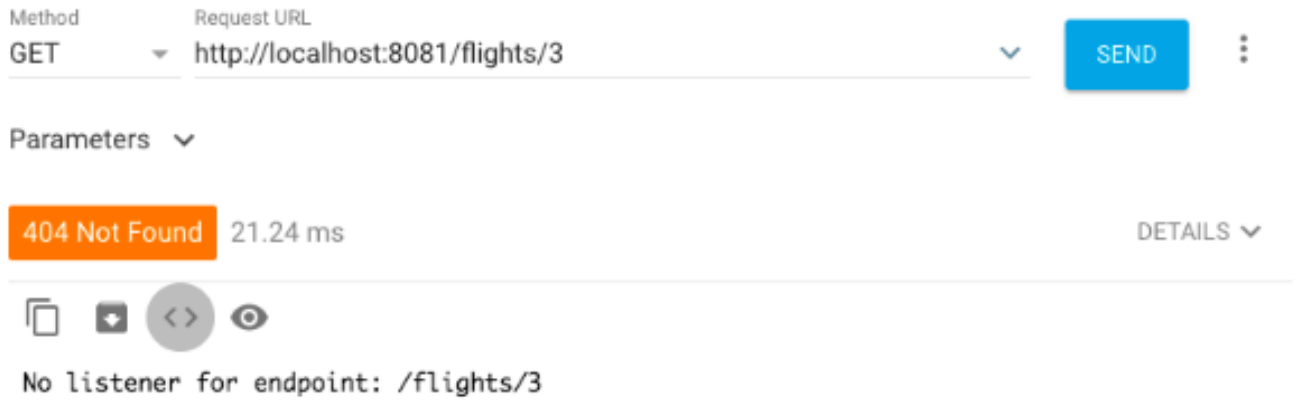
Test the application

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



Try to retrieve information about a specific flight

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



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

Creating RESTful interfaces manually for Mule applications

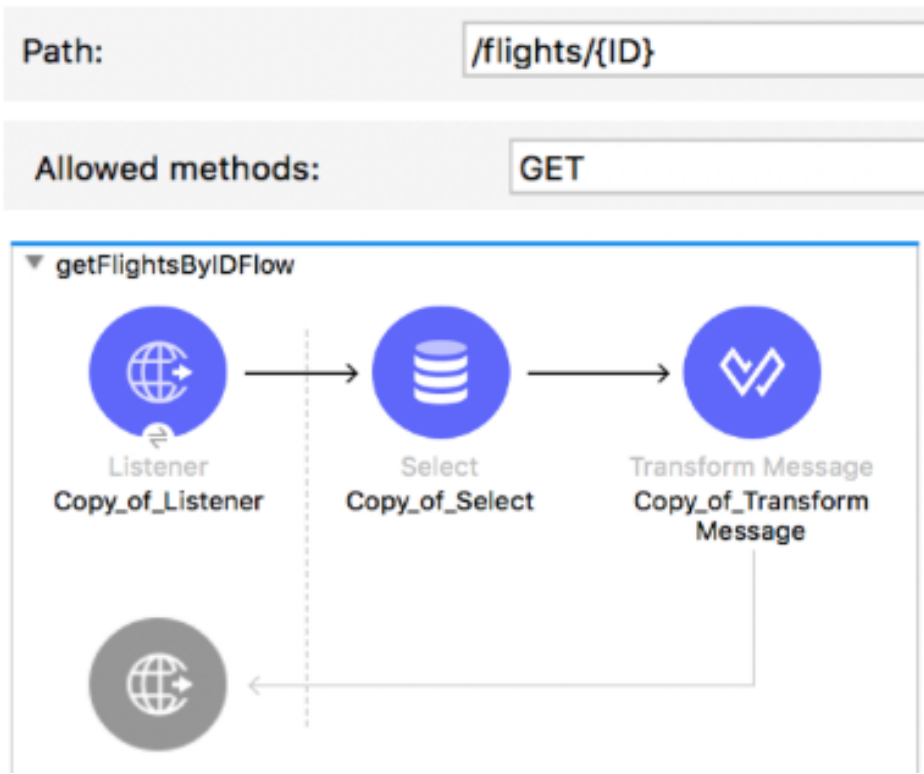
Creating RESTful interfaces

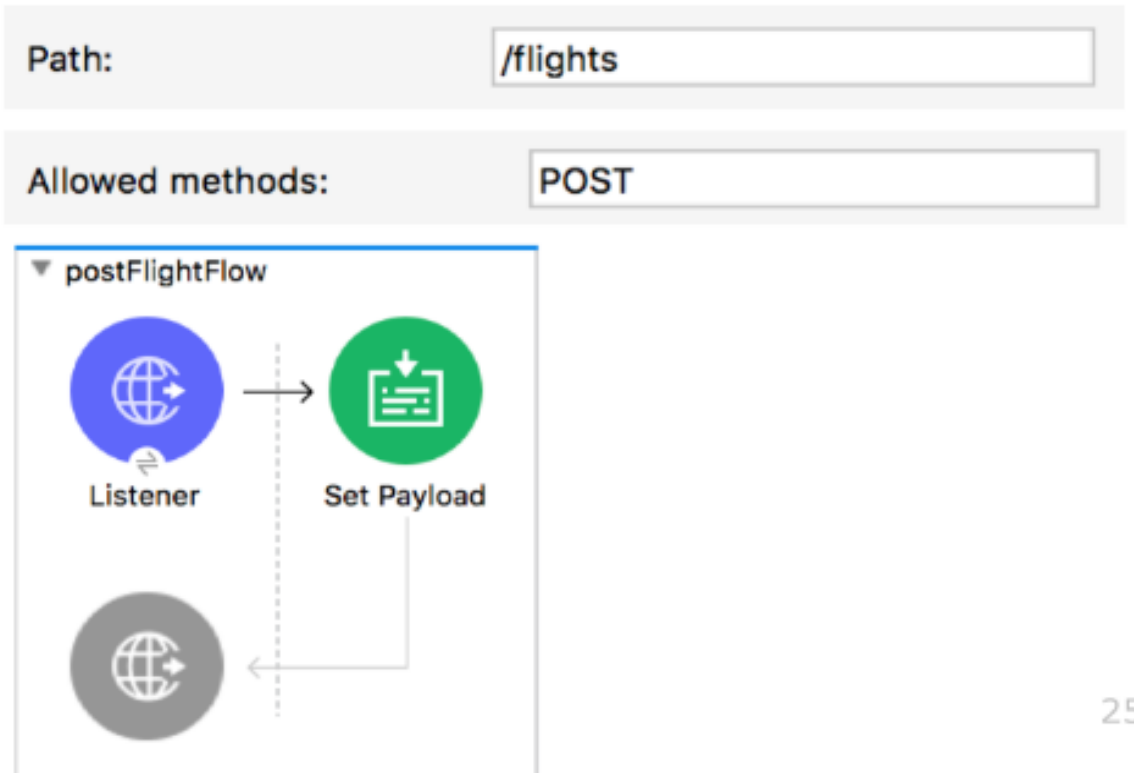
- A RESTful interface for an application will have listeners for each resource / method pairing defined by the API
  - GET: /flights
  - POST: /flights
  - GET: /flights/{ID}
  - DELETE: /flights/{ID}
  - PUT: /flights/{ID}

- You can create the interface manually or have it generated from the API definition
  - We will do both in the next two walkthroughs

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

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

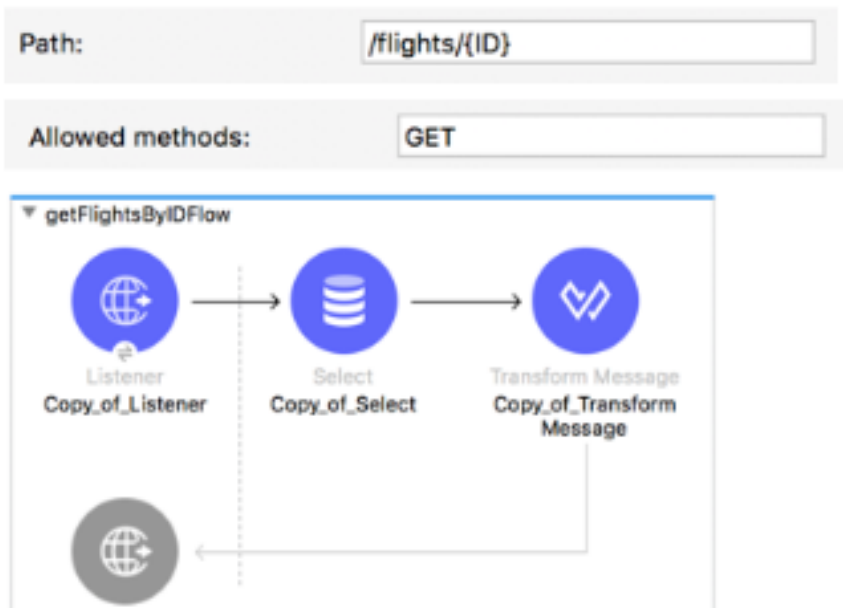


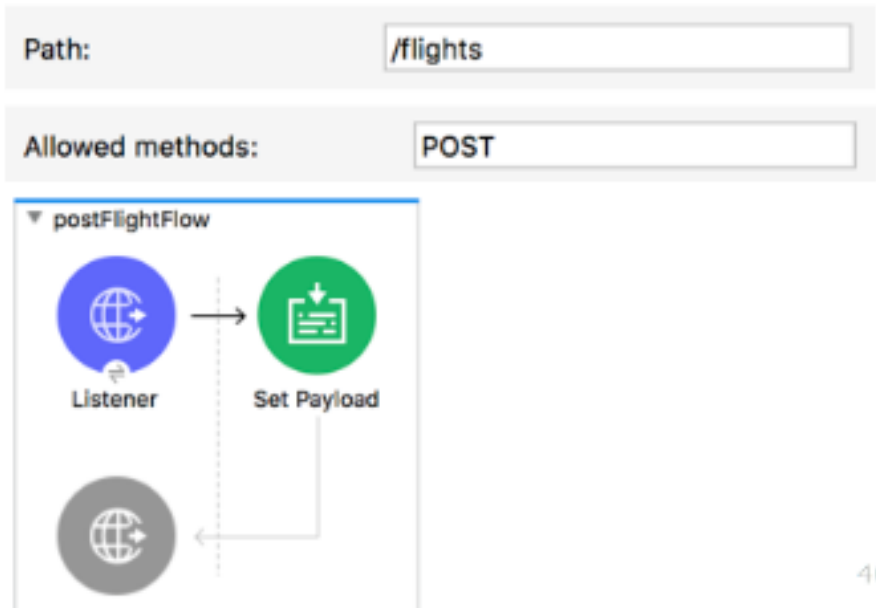


Walkthrough 4-4: Create a RESTful interface for a Mule application

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

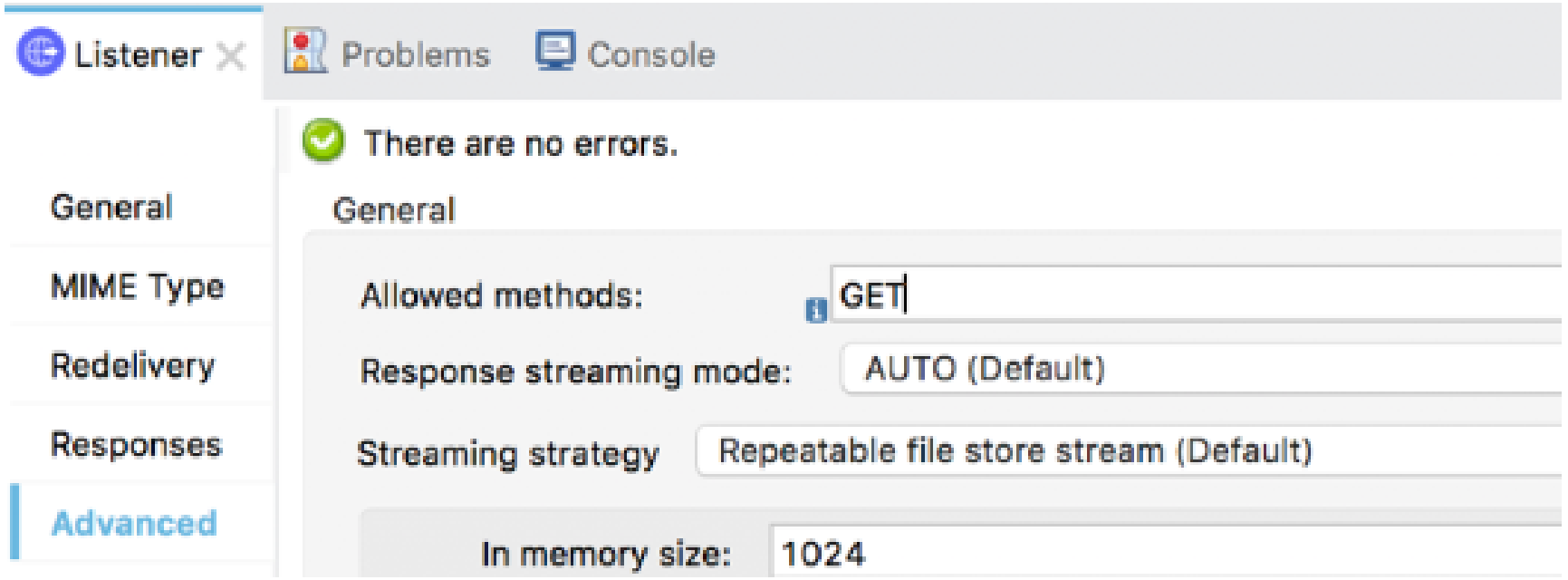
- Route based on path.
- Use a URI parameter in the path of a new HTTP Listener.
- Route based on HTTP method.





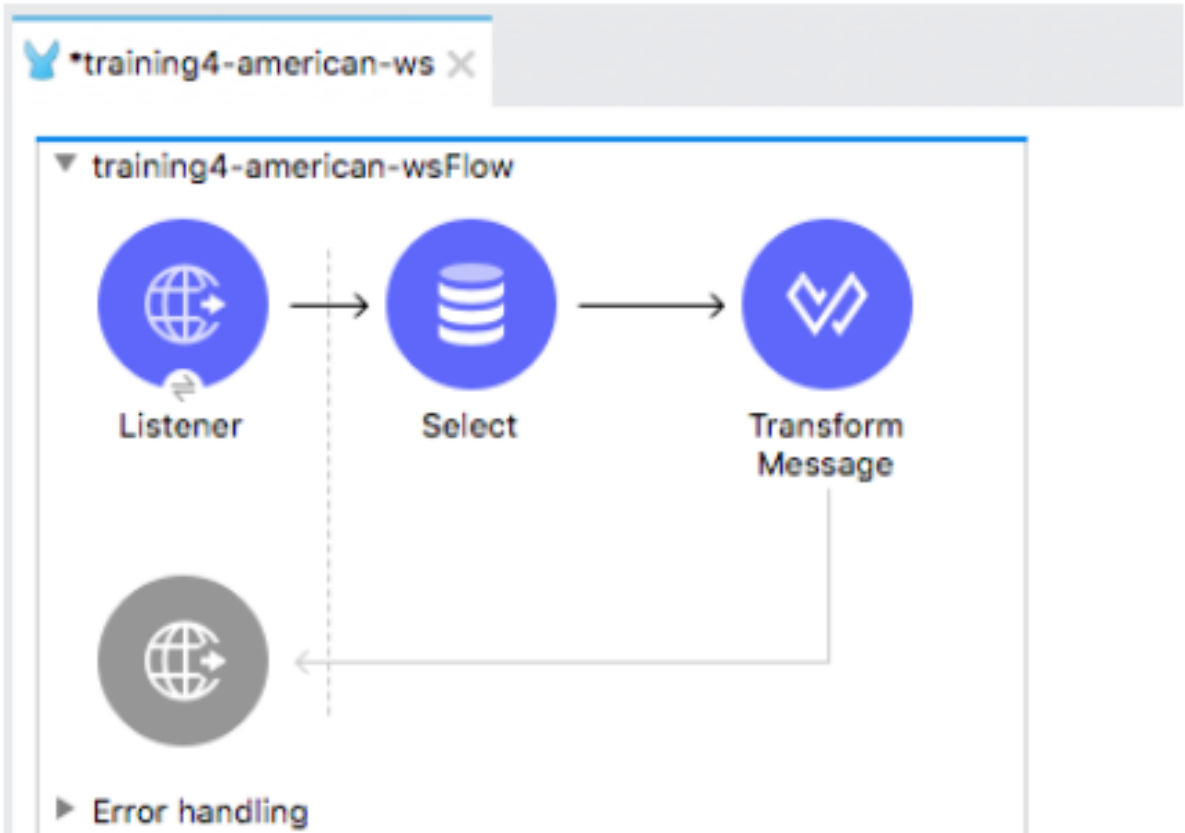
Restrict method calls to GET

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.



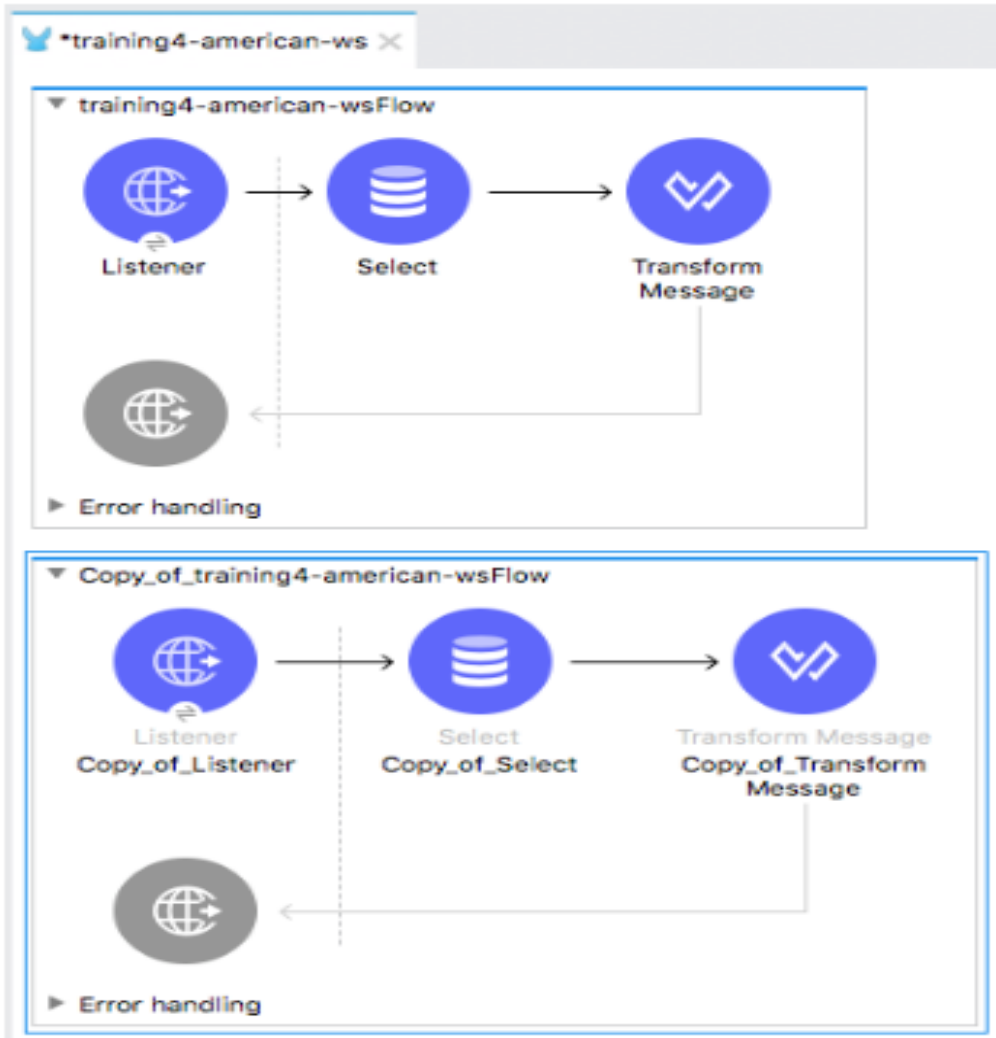
Make a copy of the existing flow

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.



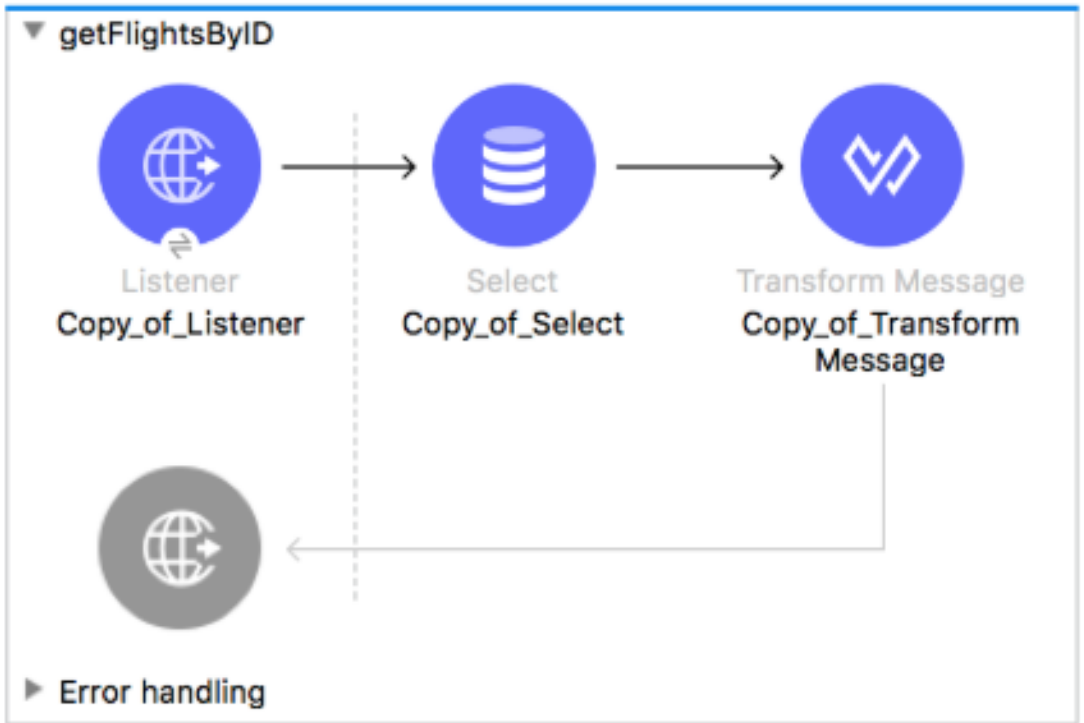
Make a copy of the existing flow

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

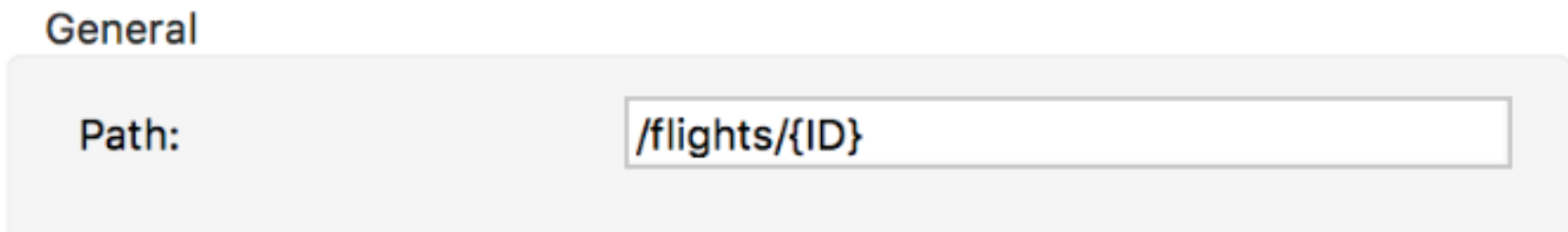
8. Double-click the first flow.
9. In the properties view, change its name to getFlights.
10. Change the name of the second flow to getFlightsByID.



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}]
    ```

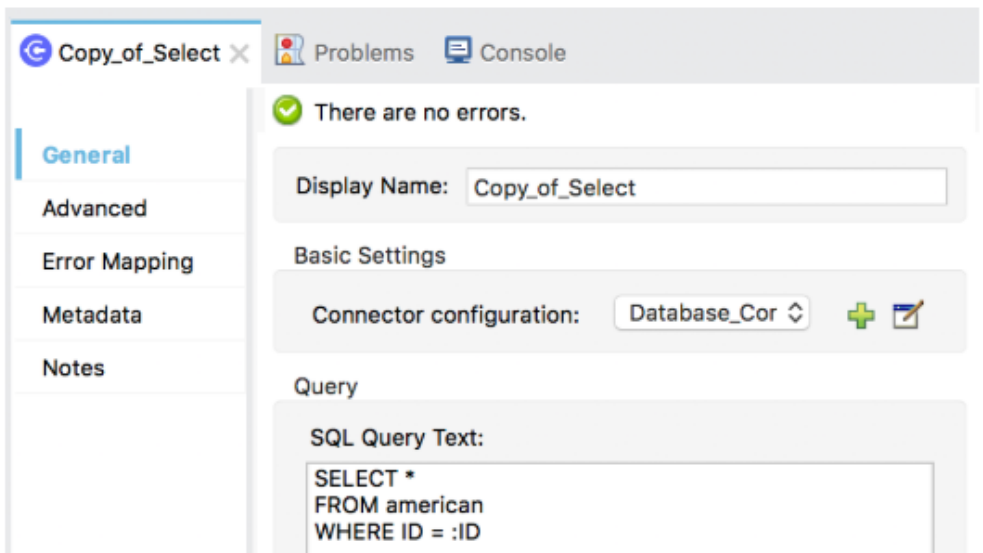
    Note: You learn to write expressions in a later module in the Development Fundamentals course.

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

    SELECT *

    FROM American

    WHERE ID = **:ID**



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}]
    ```

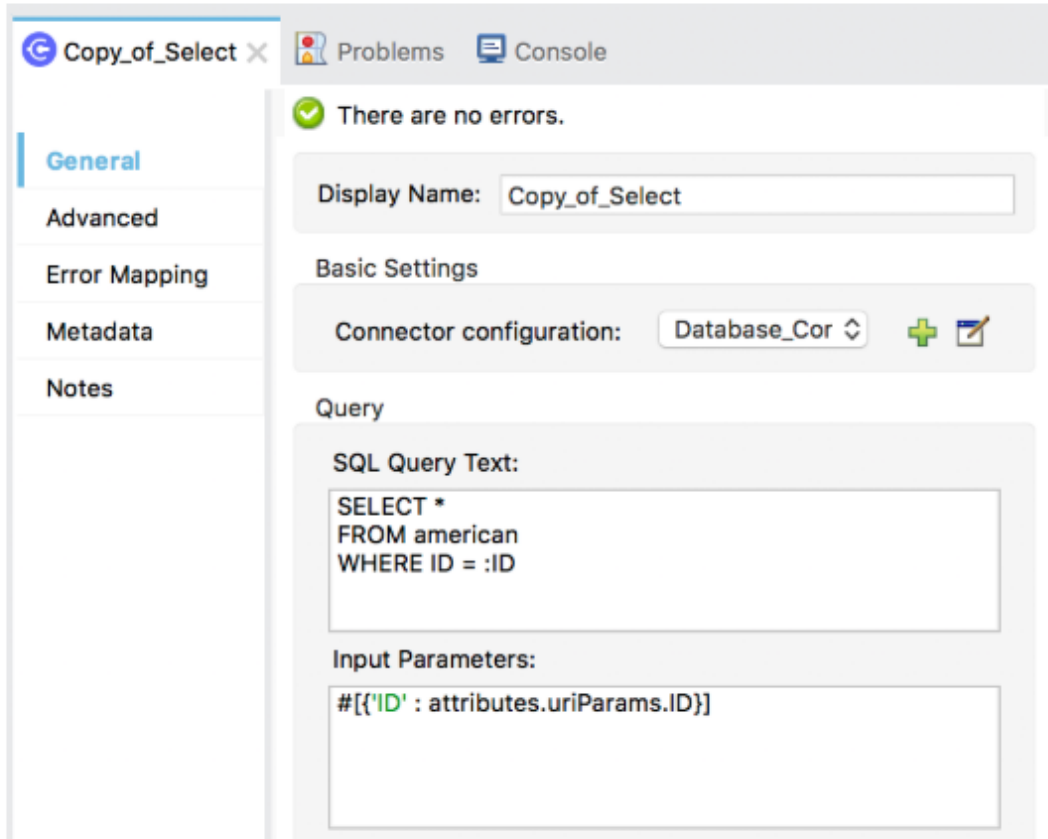
    Note: You learn to write expressions in a later module in the Development Fundamentals course.

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

SELECT *

FROM American

WHERE ID = **:ID**



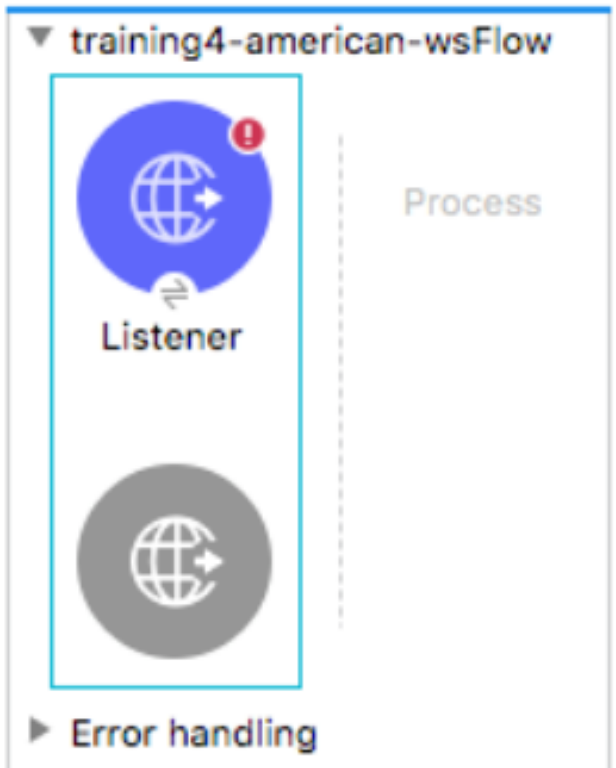
Test the application

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.

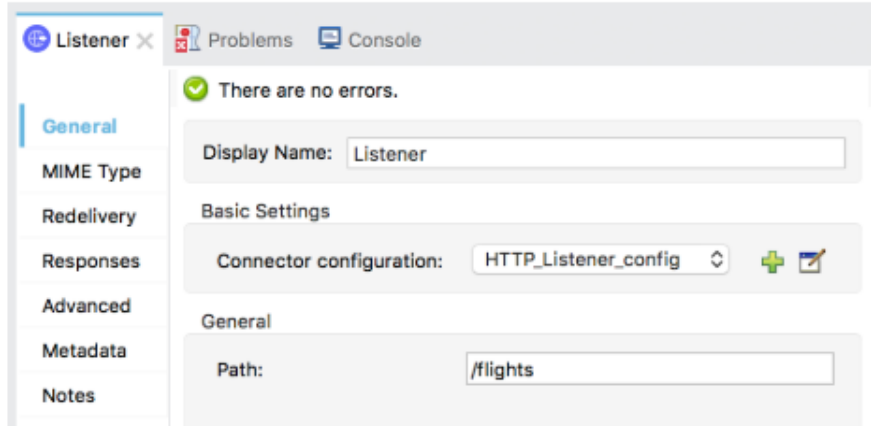
Make a new flow to handle post requests

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.

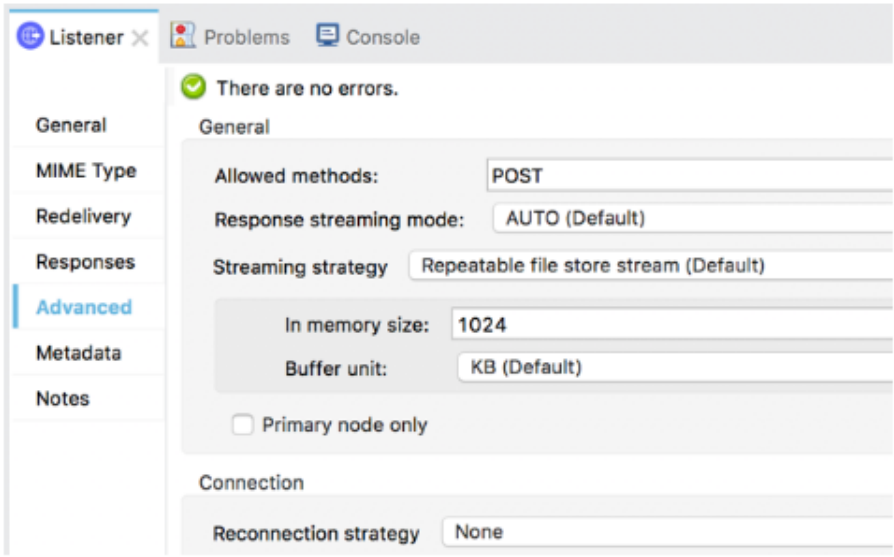


26. Change the name of the flow to postFlight.
27. In the Listener properties view, set the connector configuration to the existing HTTP_Listener_config.

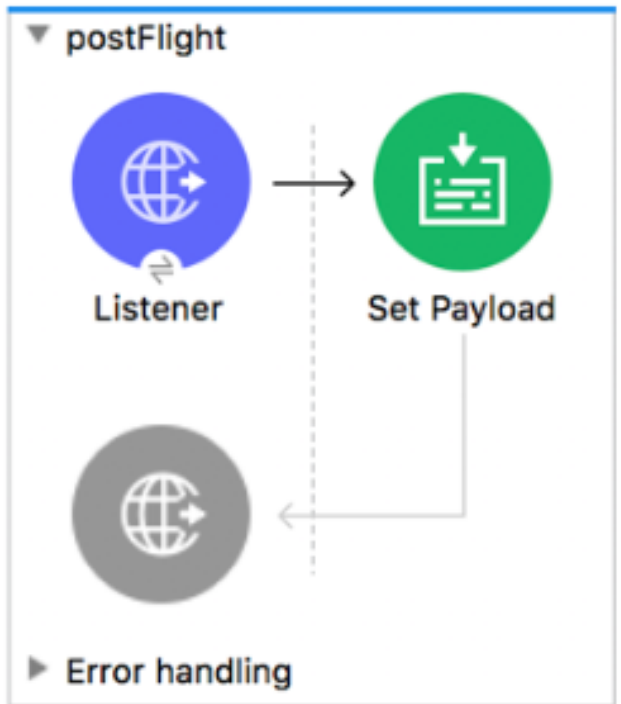
28. Set the path to /flights.



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



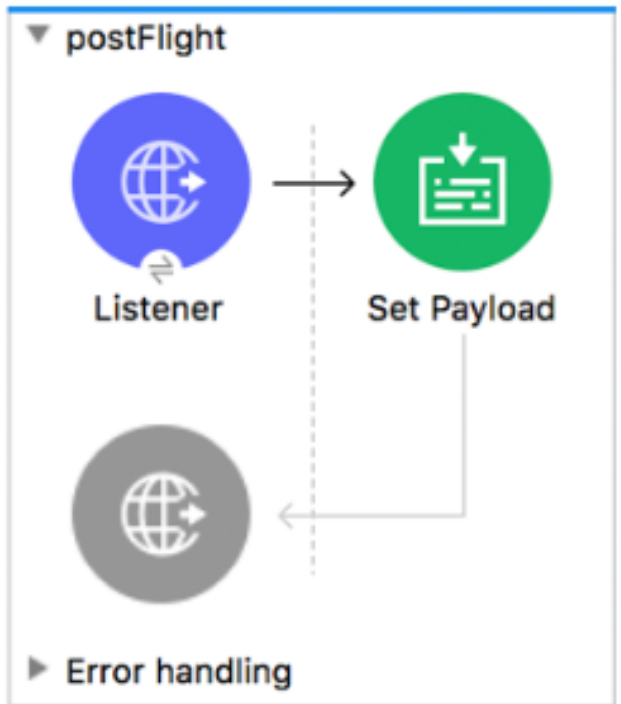
31. Drag the Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



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

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

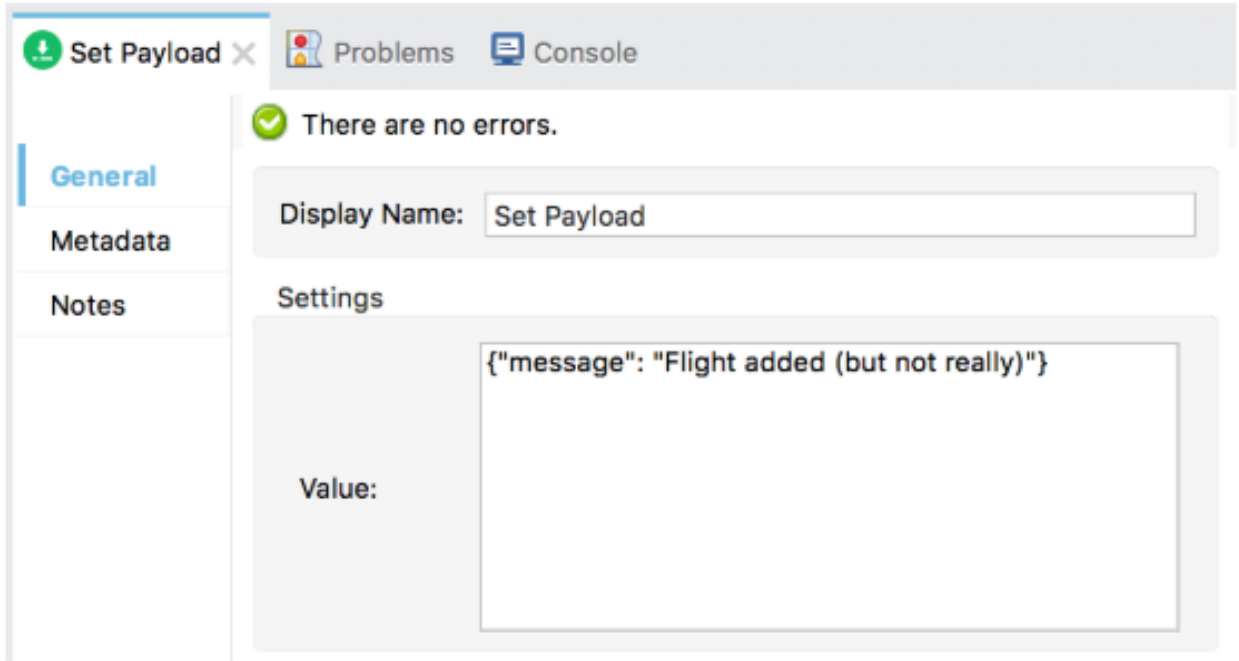
31. Drag the Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



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

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

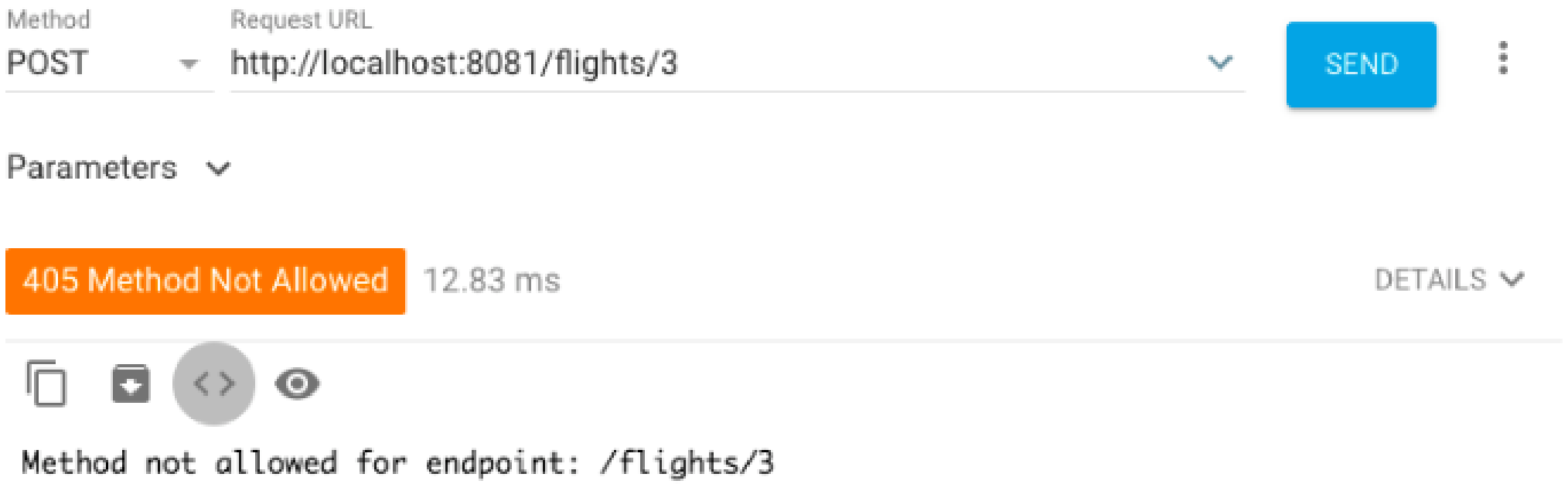
33. Return to Anypoint Studio and in the Set Payload properties view, set value to the value you copied.



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.



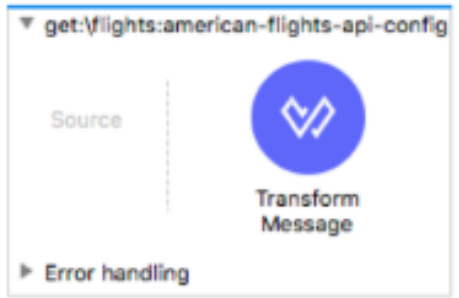
37. Remove the URI parameter from the request URL: http://localhost:8081/flights.

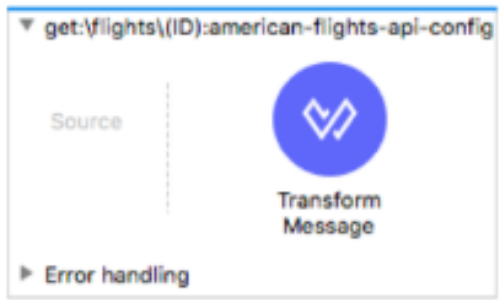
38. Send the request; you should now see the message the flight was added – even though you did not send any flight data to add.

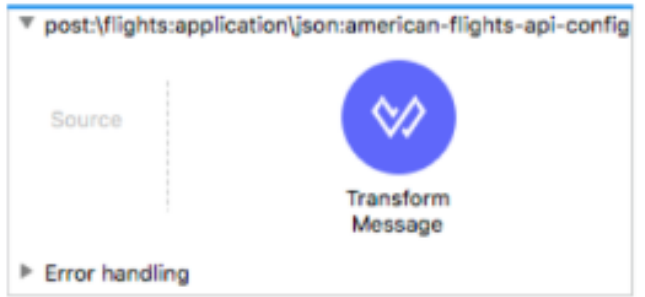
Generating RESTful interfaces automatically using APIkit

Creating RESTful interfaces automatically using APIkit

- **APIkit** is an open-source toolkit that includes an Anypoint Studio plugin
- The Anypoint Studio **APIkit plugin** can generate an interface automatically from a RAML API definition
  - For new or existing projects
- It generates a main routing flow and flows for each of the API resource / method pairs
- You add processors to the resource flows to hook up to your backend logic

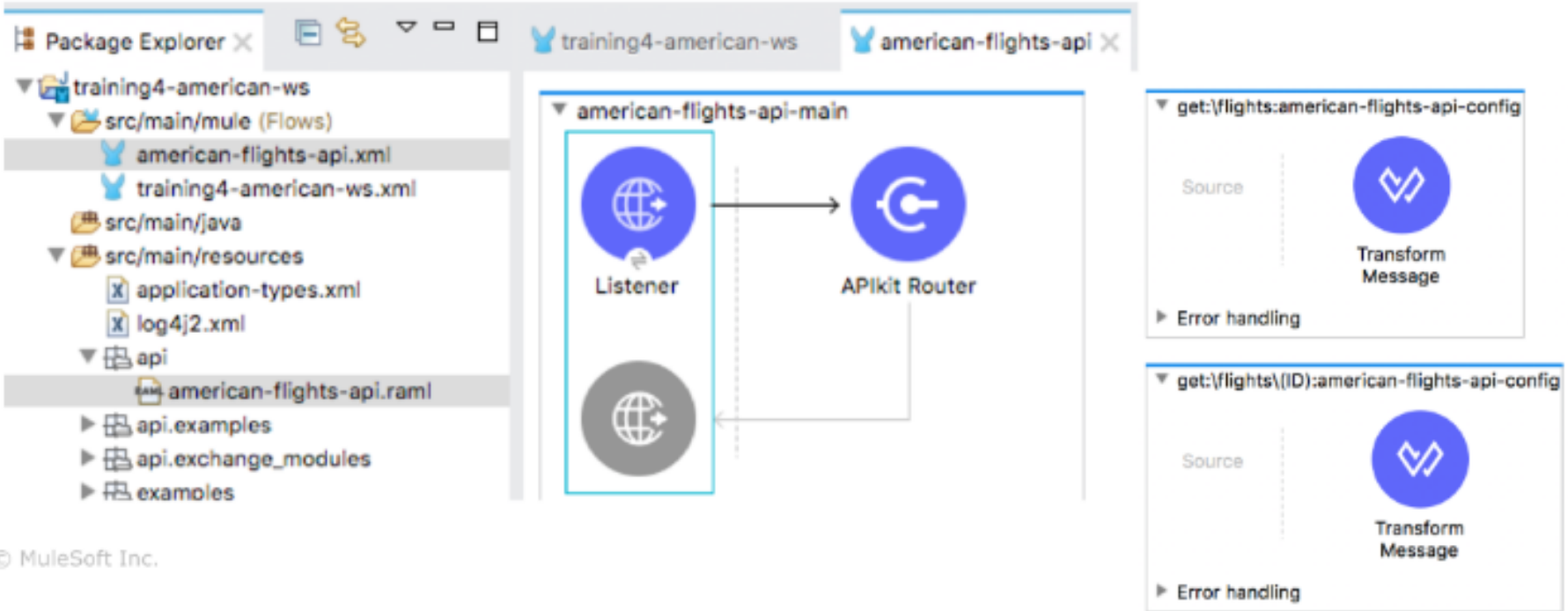






Walkthrough 4-5: Use Anypoint Studio to create a RESTful API interface from a RAML file

- 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



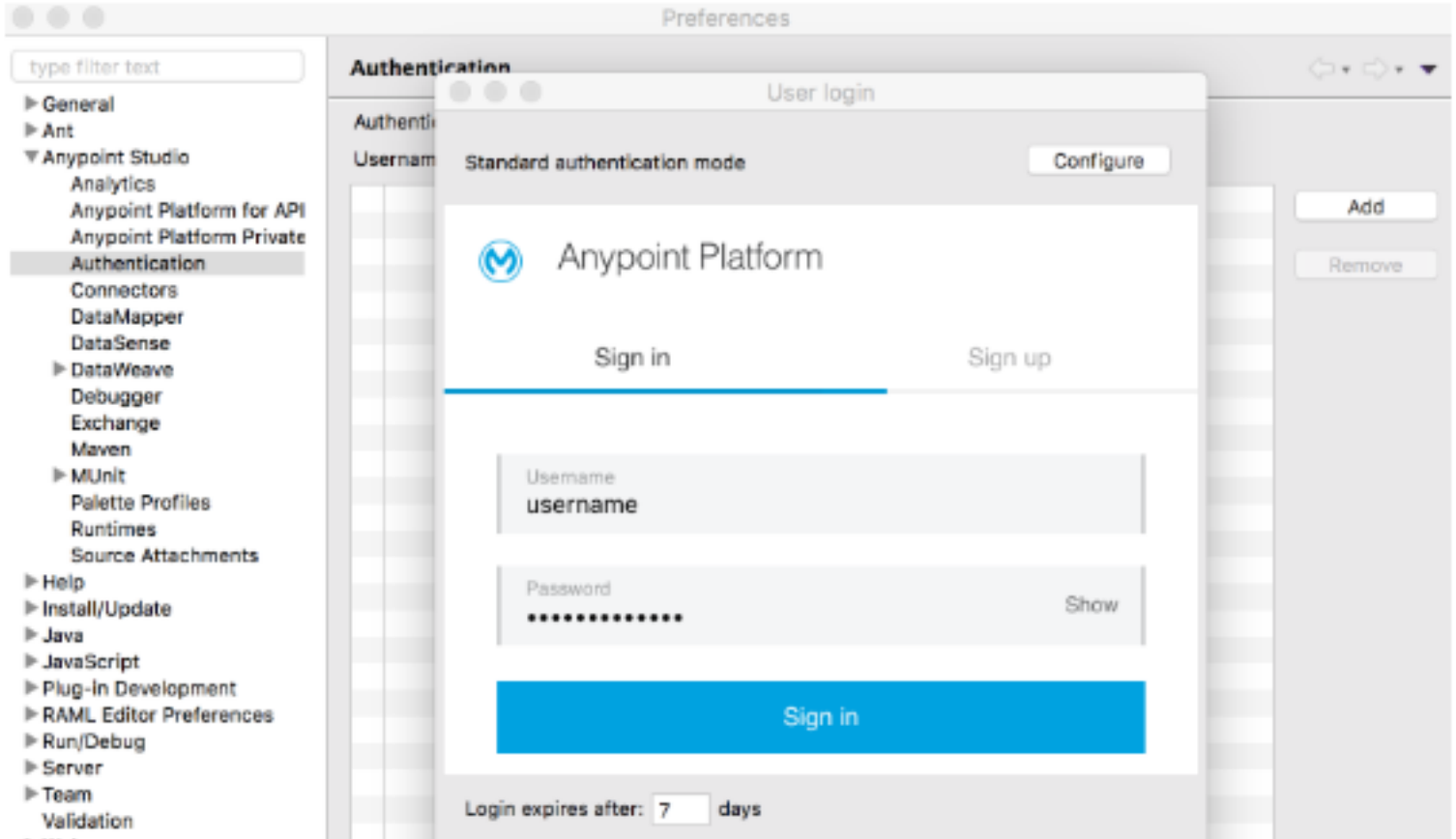
Walkthrough 4-5: Use Anypoint Studio to create a RESTful API interface from a RAML file

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.

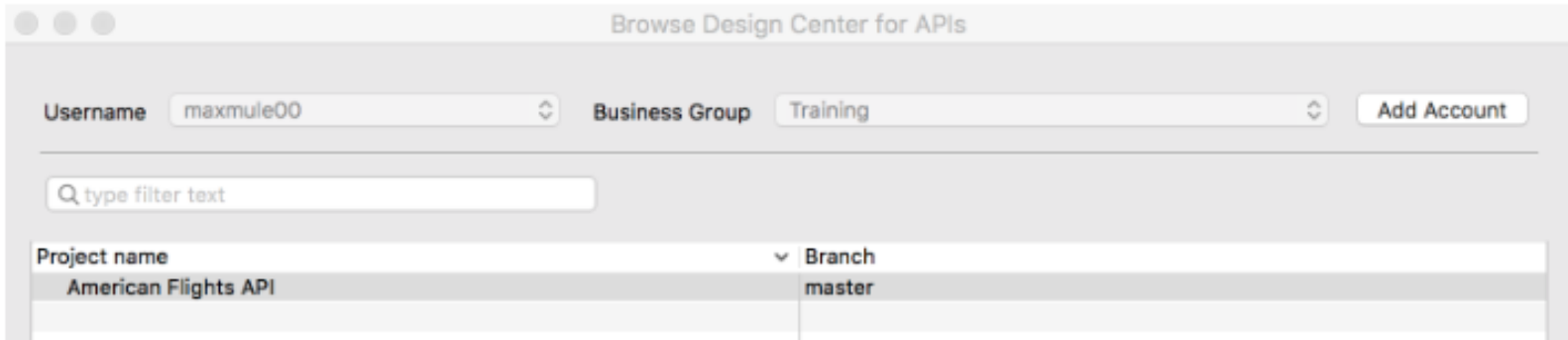
Add Anypoint Platform credentials to Anypoint Studio

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.



Add an API from Design Center to the Anypoint Studio project

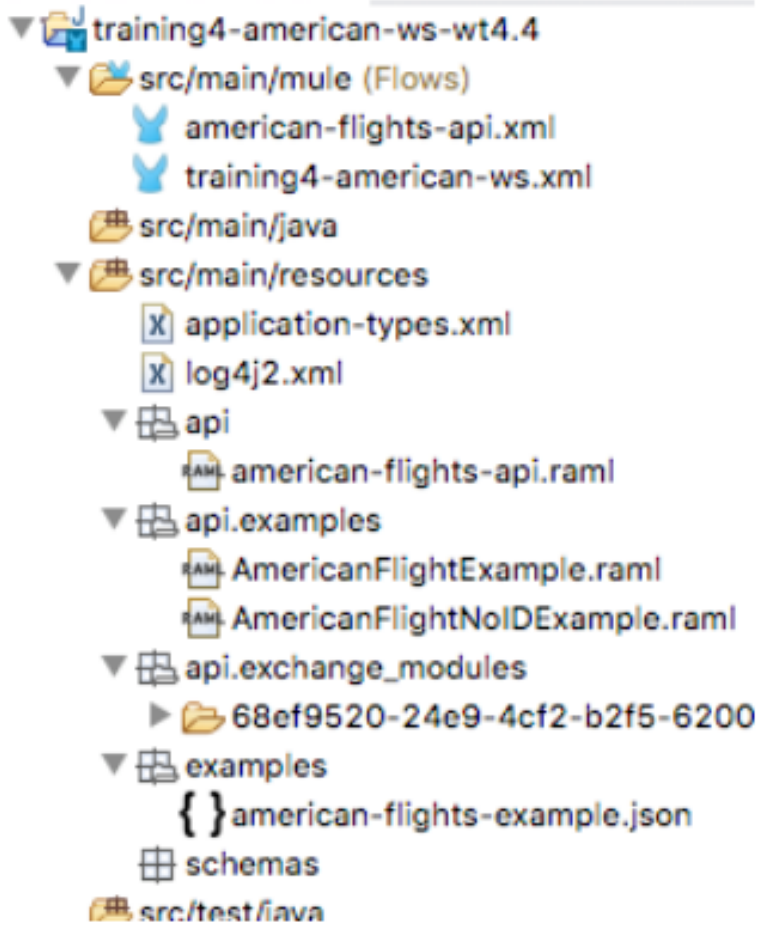
6. In the Package Explorer, locate the src/main/resources/api folder; it should not contain any files.
7. Right-click the folder (or anywhere in the project in the Package Explorer) and select Anypoint Platform > Import from Design Center.
8. In the Browse Design Center for APIs dialog box, select the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



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

Locate the API files added to the project

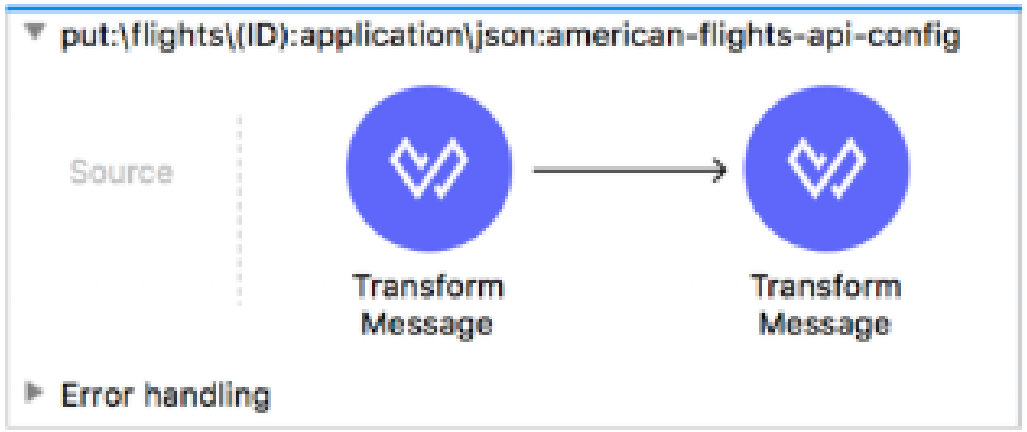
10. In the Package Explorer, locate and expand the src/main/resources folder; it should now contain api folders.
11. Expand the api folder; you should see the RAML file imported from Design Center.

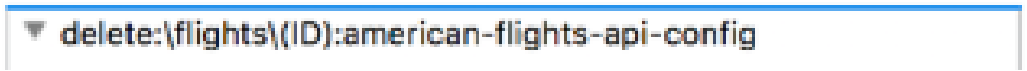


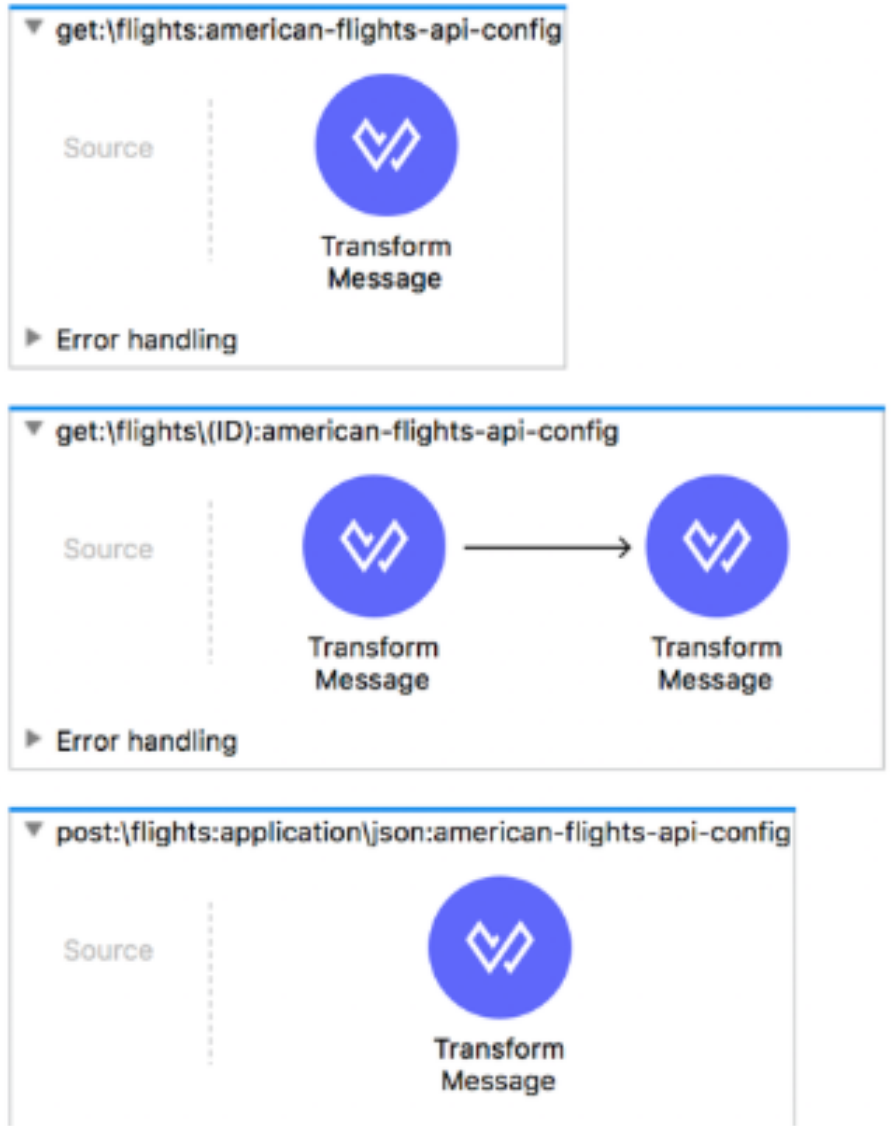
Examine the XML file created

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

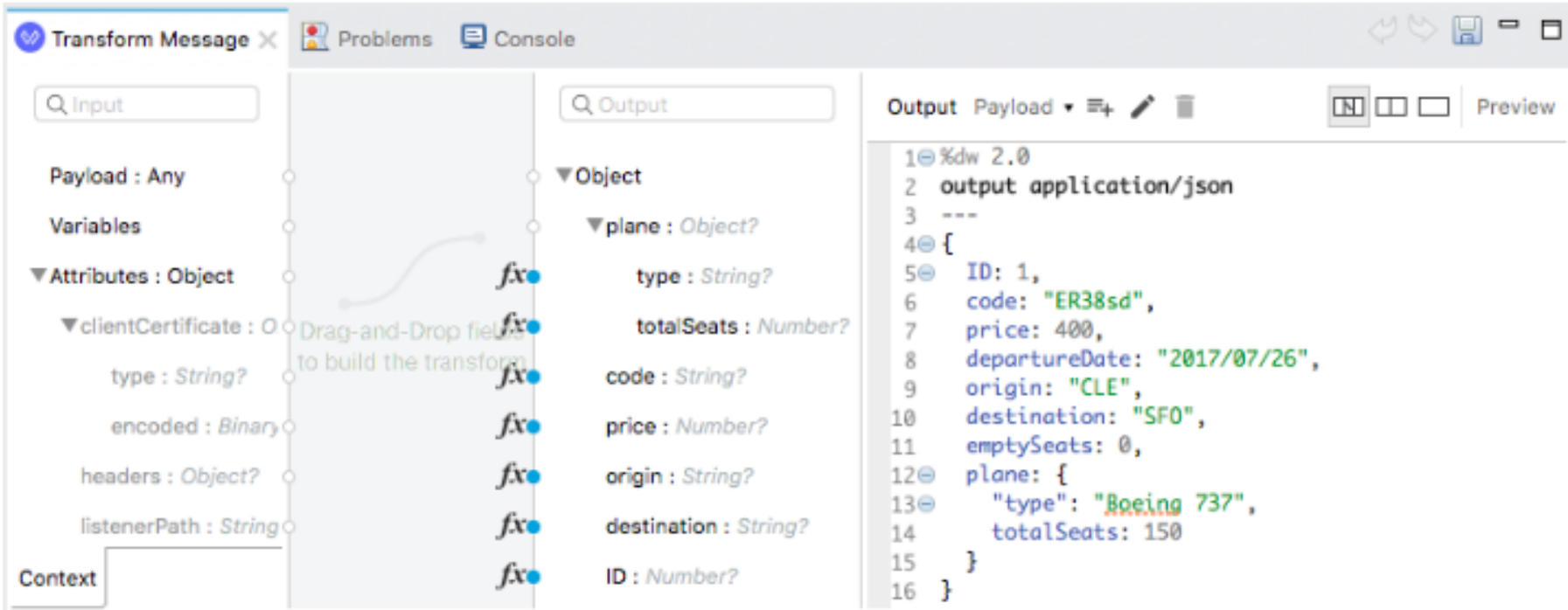
- get:/flights
- get:/flights/{ID}
- post:/flights
- delete:/flights/{ID}
- put:/flights/{ID}



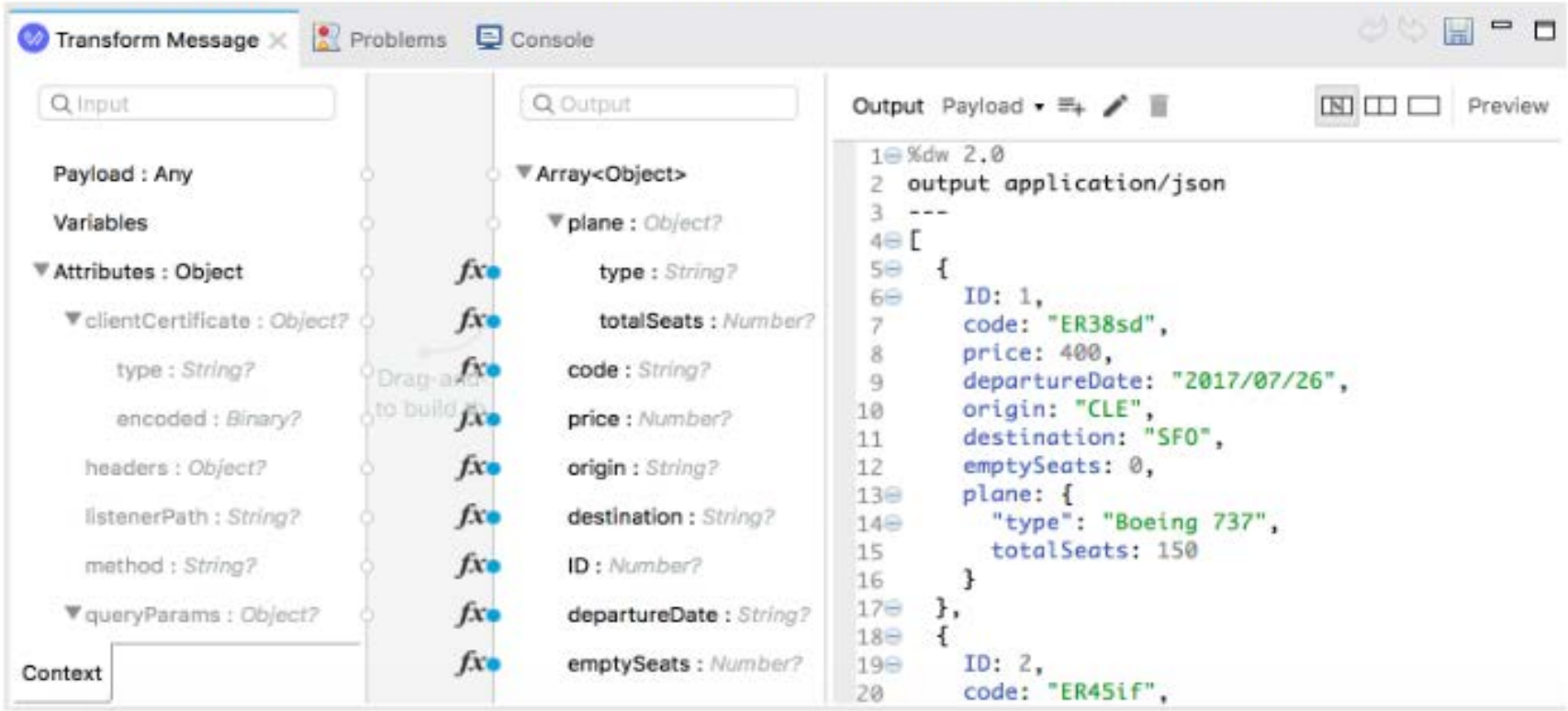



get:\flights:american-flights-api-config
Source
Transform Message
Error handling
get:\flights\(ID):american-flights-api-config
Source
Transform Message
Transform Message
Error handling
post:\flights:application\json:american-flights-api-config
Source
Transform Message

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



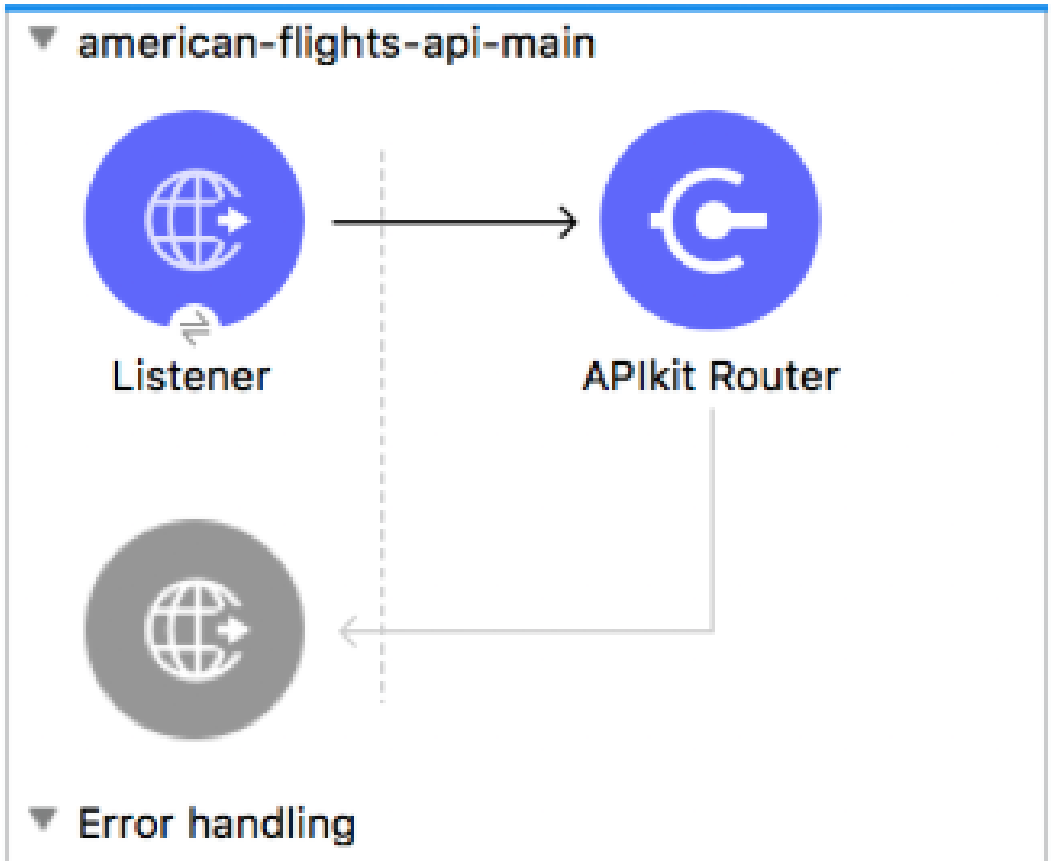
14. In the get:/flights flow, double-click the Transform Message component and look at the output JSON in the properties view.



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

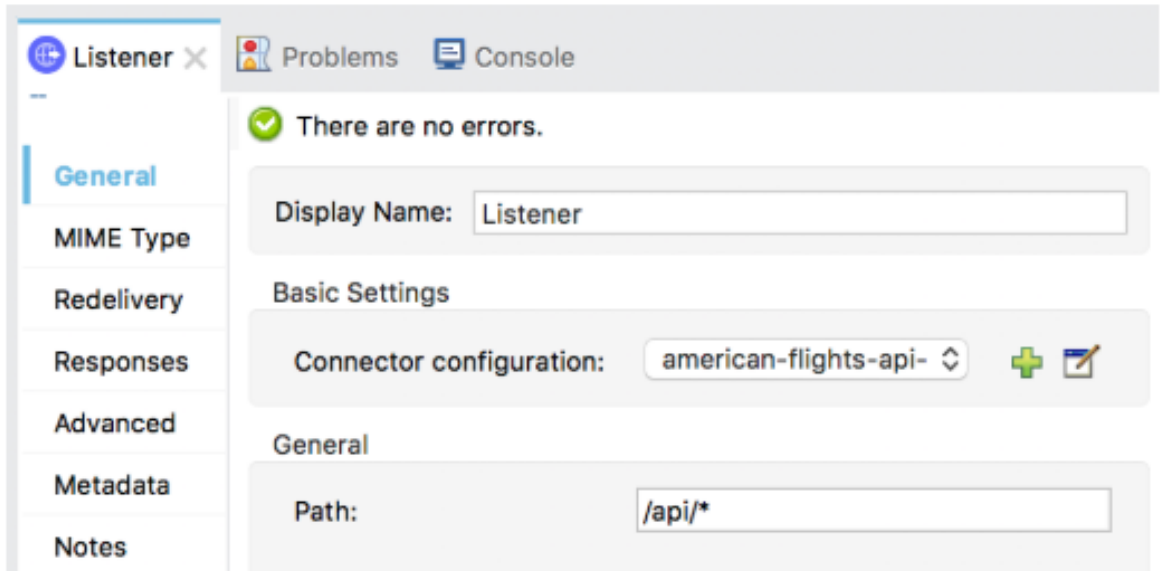
Examine the main flow and the new HTTP Listener endpoint

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



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

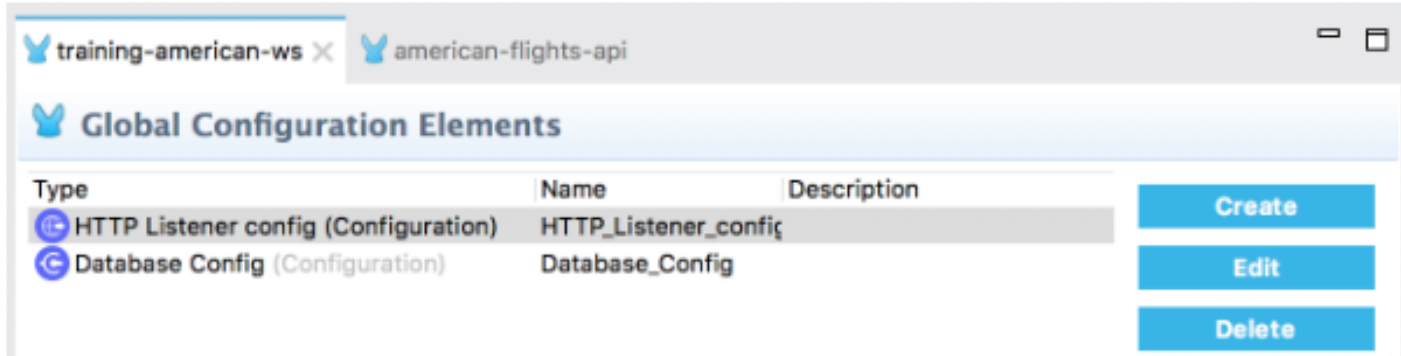
    *Note: The * is a wildcard allowing any characters to be entered after /api/.*



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.

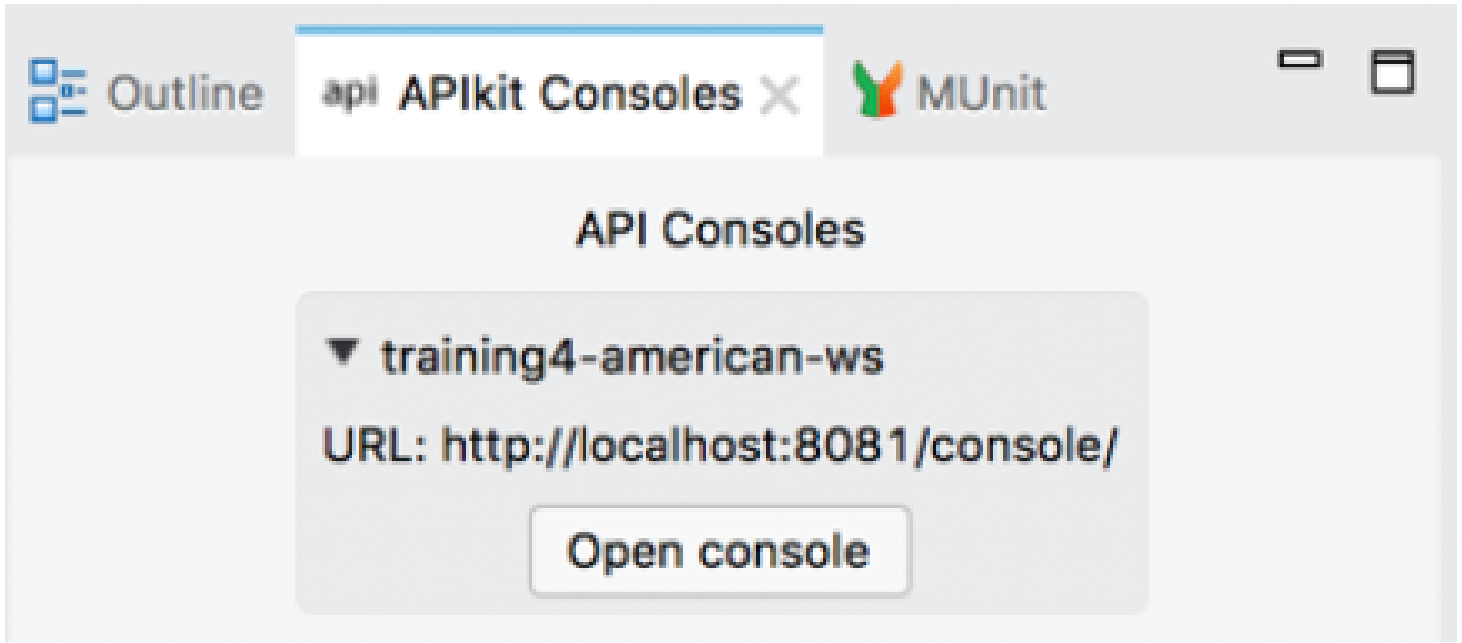


23. Return to the Message Flow view.
24. Right-click the HTTP Listener in getFlights and select Delete.

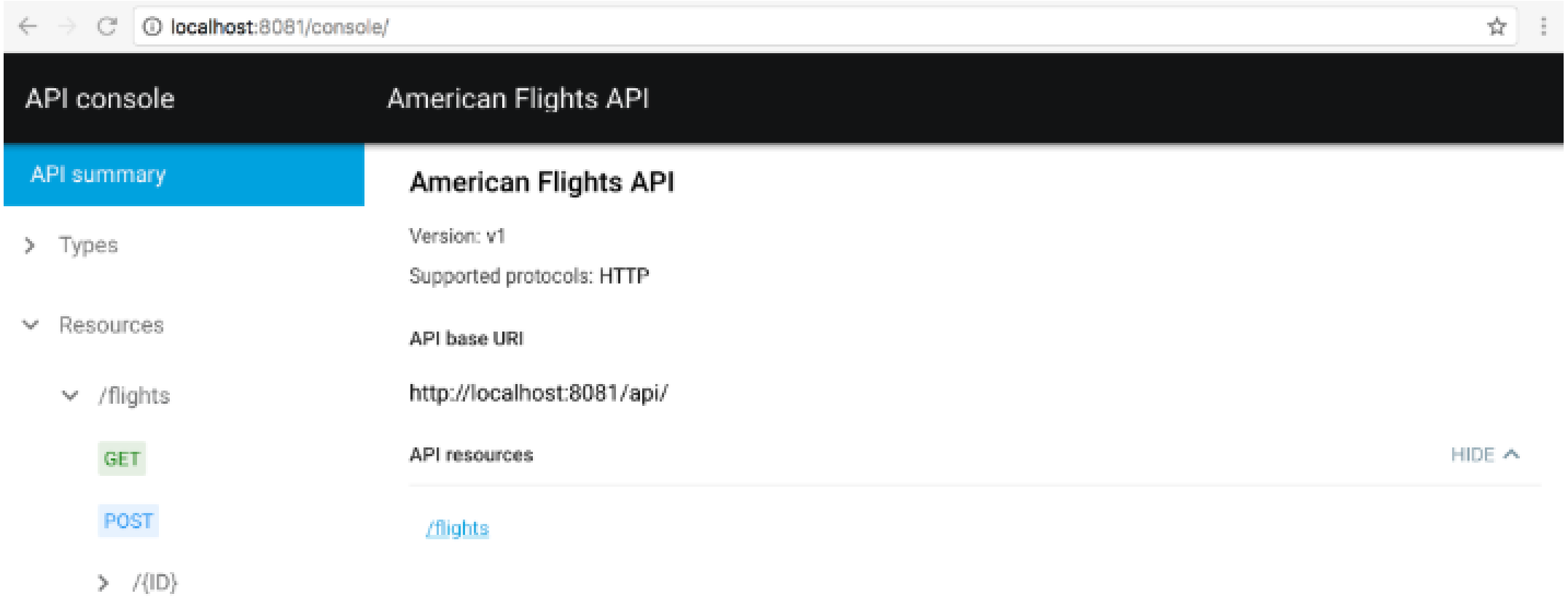
25. Delete the other two HTTP Listeners.

Test the web service using APIkit console

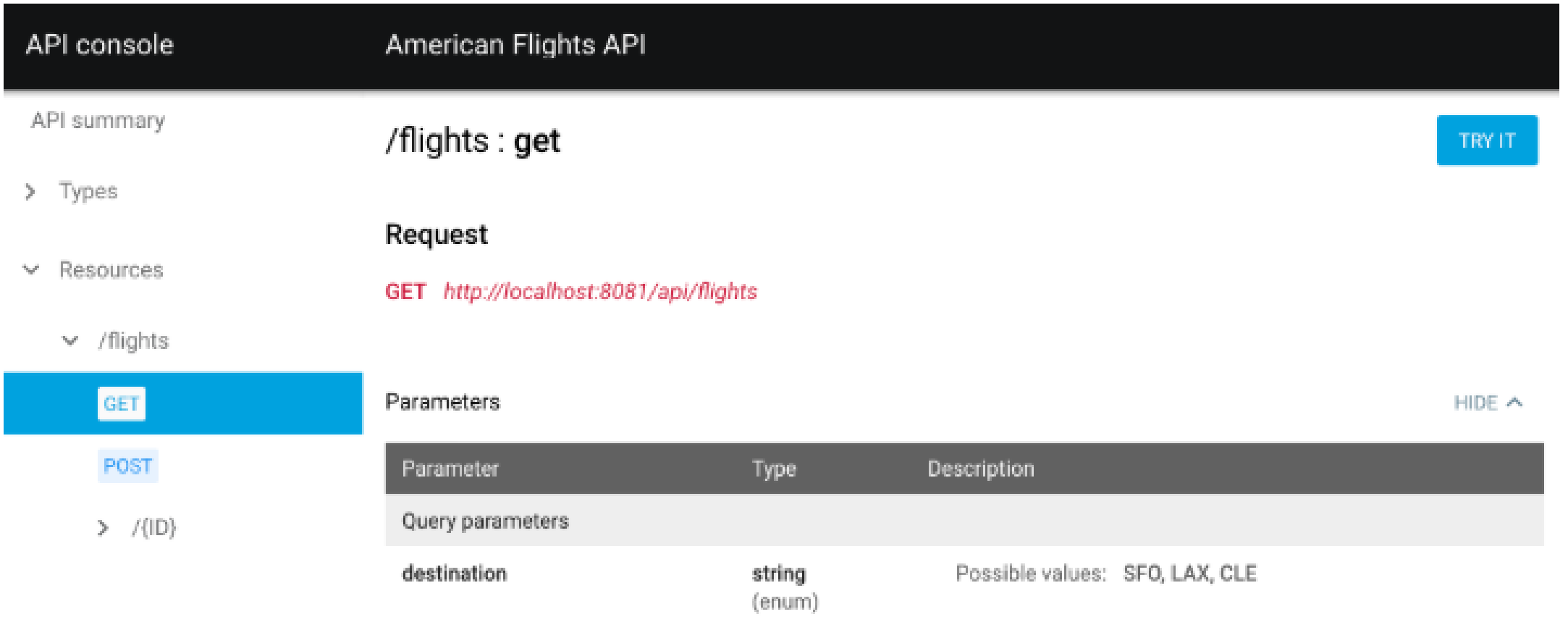
26. Save the files.
27. Look at the console; you should see the application was not redeplpoyed.
28. Stop the project.
29. Run the project and wait until Mule and the application restart.
30. Locate the new APIkit Consoles view that is created and opened in Anypoint Studio.



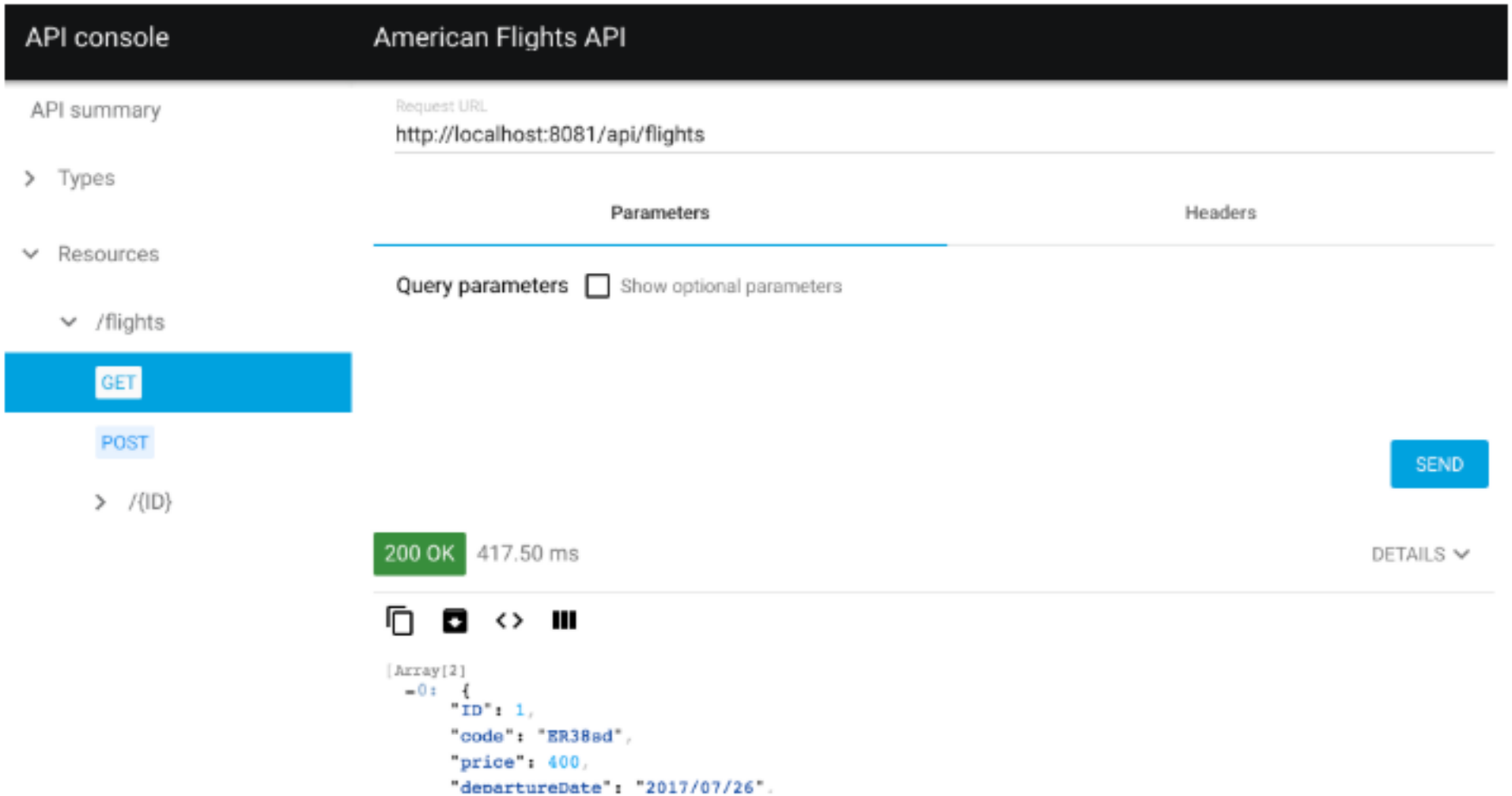
31. Click the Open console button; a browser window should be open with an API console.



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



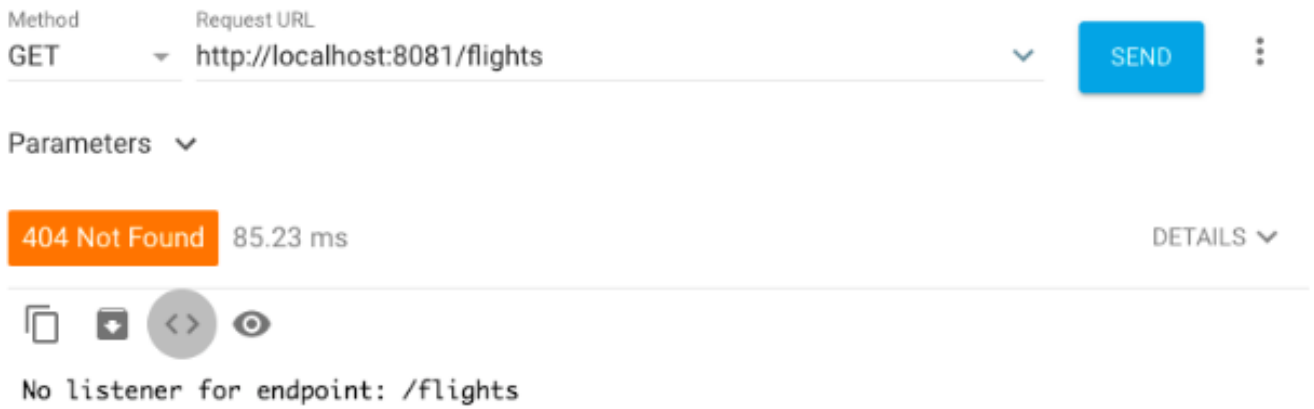
33. Click Send; you should get a 200 response with the example flight data – not all the flights.



34. Close the browser window.

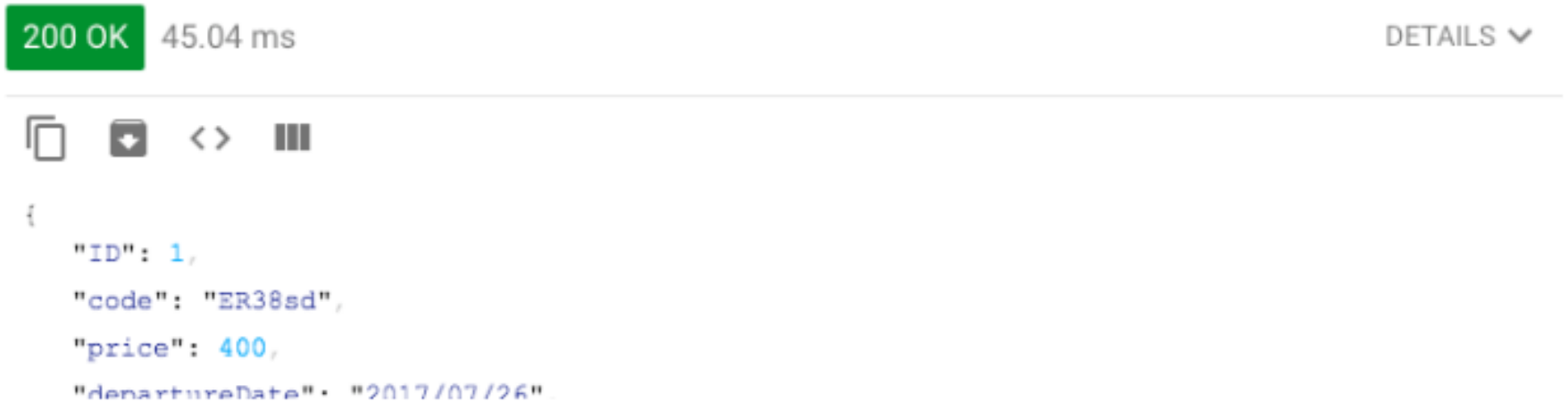
Test the web service using Advanced REST Client

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.

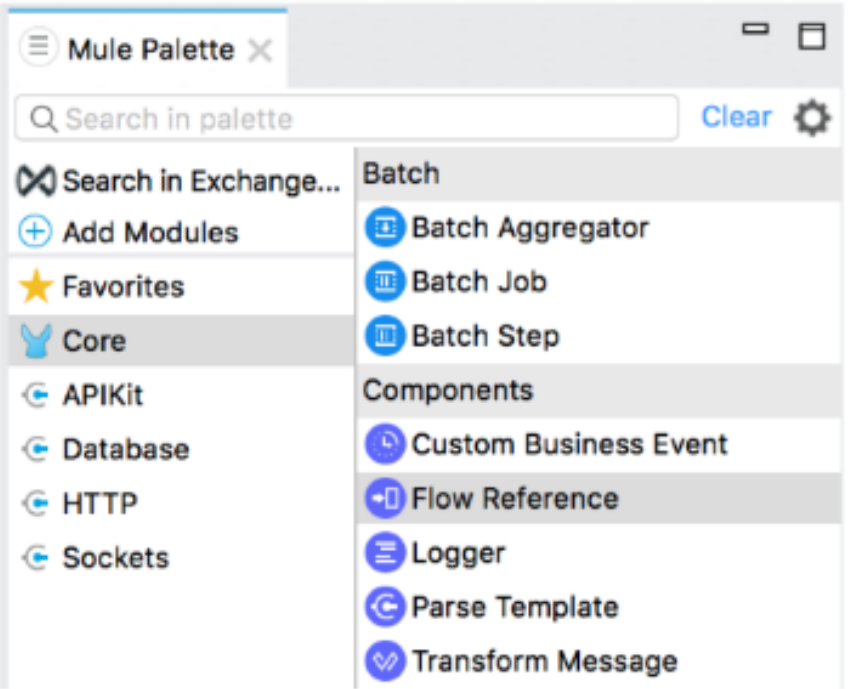
38. Make a request to http://localhost:8081/api/flights/3; you should see the example data returned for a flight with an ID of 1.



Connecting interfaces to implementations

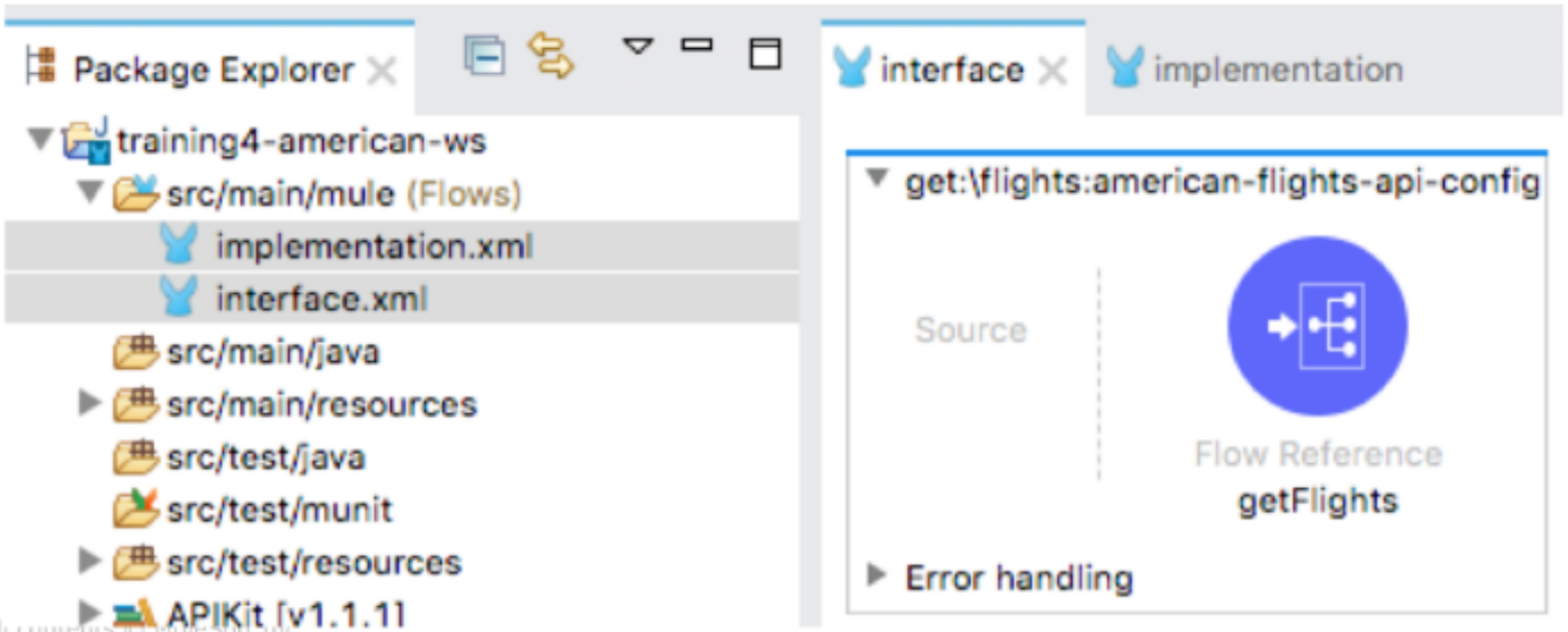
Passing messages to other flows

- Flows can be broken into multiple flows
  - Makes the graphical view more intuitive and the XML code easier to read
  - Promotes code reuse
- All flows are identified by name and can be called via **Flow Reference** components in other flows



Walkthrough 4-6: Implement a RESTful web service

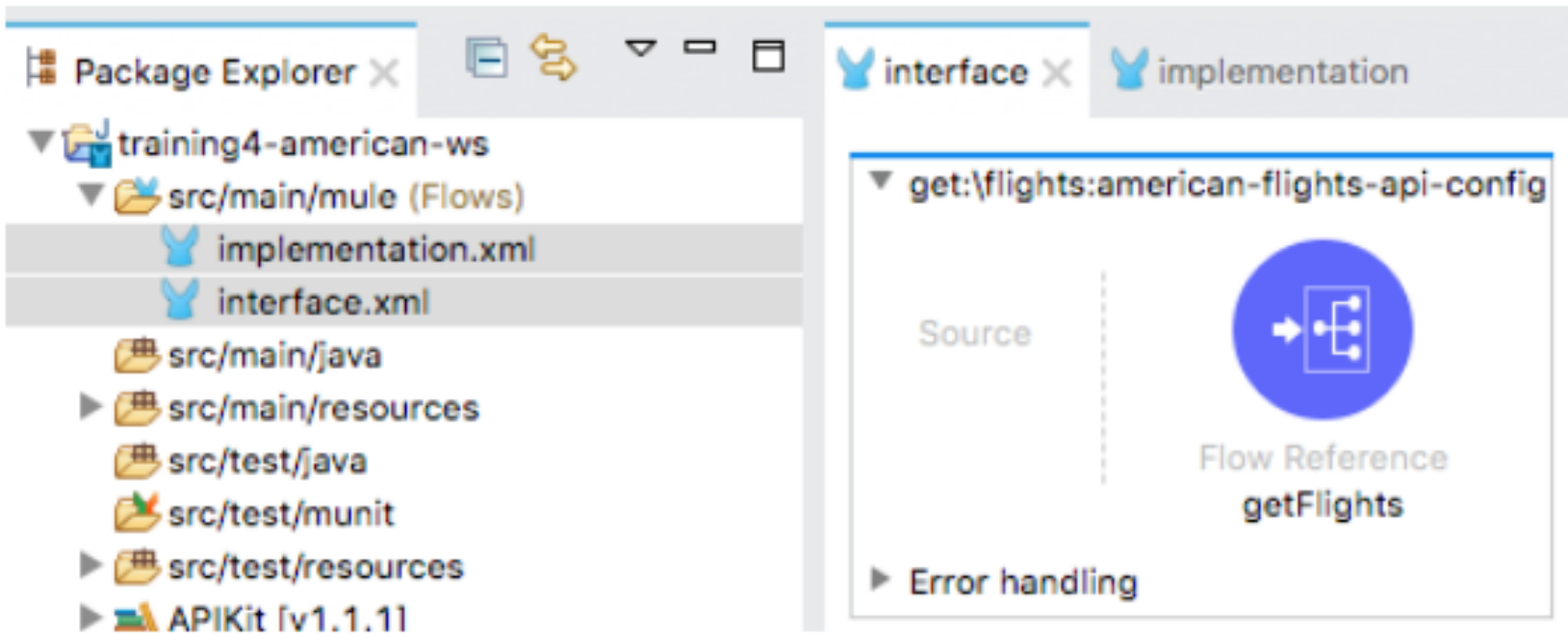
- Pass an event from one flow to another
- Call the backend flows
- Create new logic for the nested resource call
- Test the web service using APIkit console



Walkthrough 4-6: Implement a RESTful web service

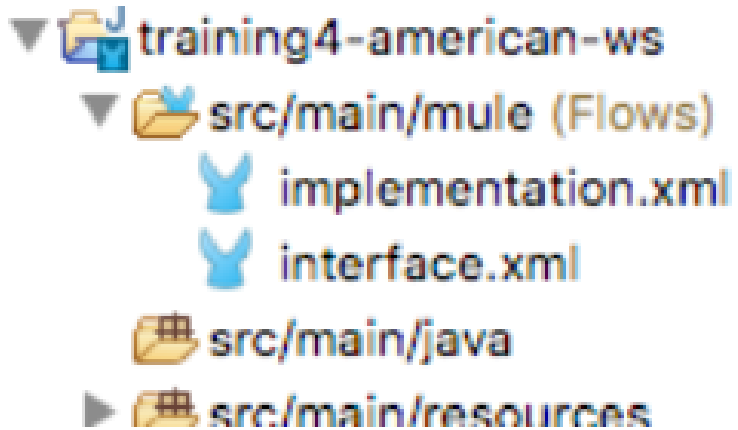
In this walkthrough, you wire the RESTful web service interface up to your back-end logic. You will:

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



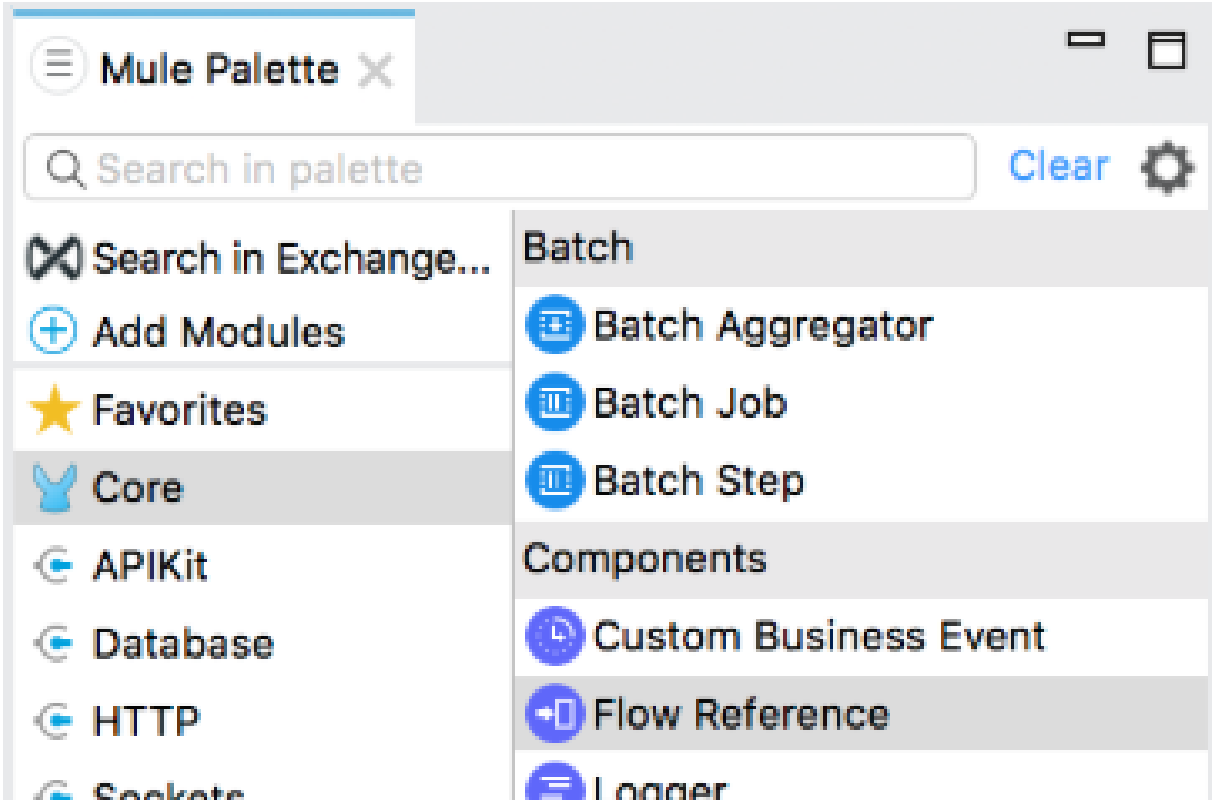
Rename the configuration files

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

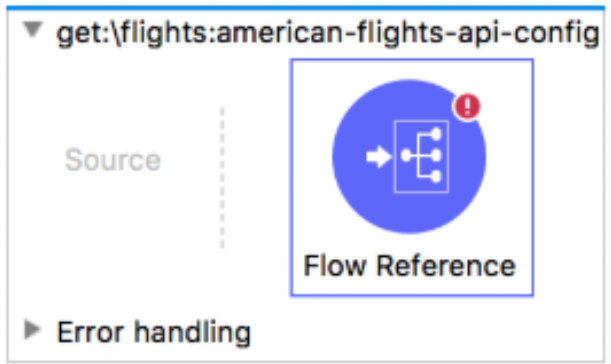


Use a Flow Reference in the /flights resource

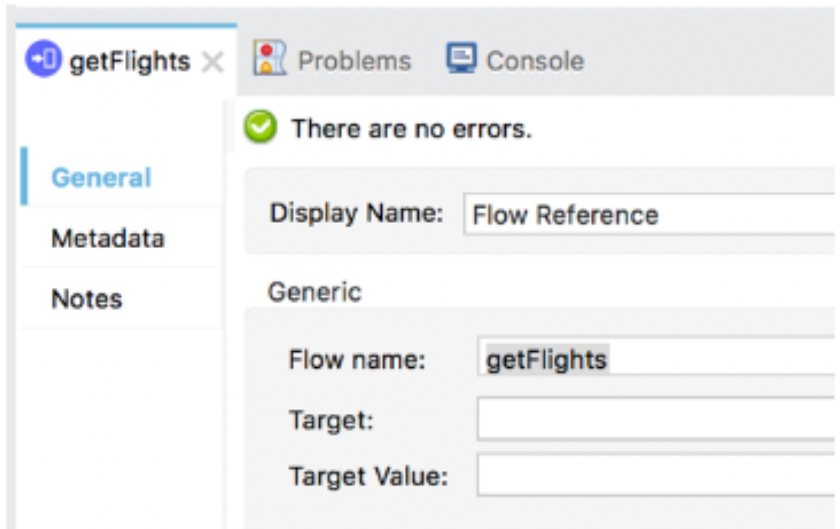
6. Open interface.xml.
7. Delete the Transform Message component in the get:/flights flow.
8. In the Mule Palette, select Core and locate the Components section in the right-side.



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



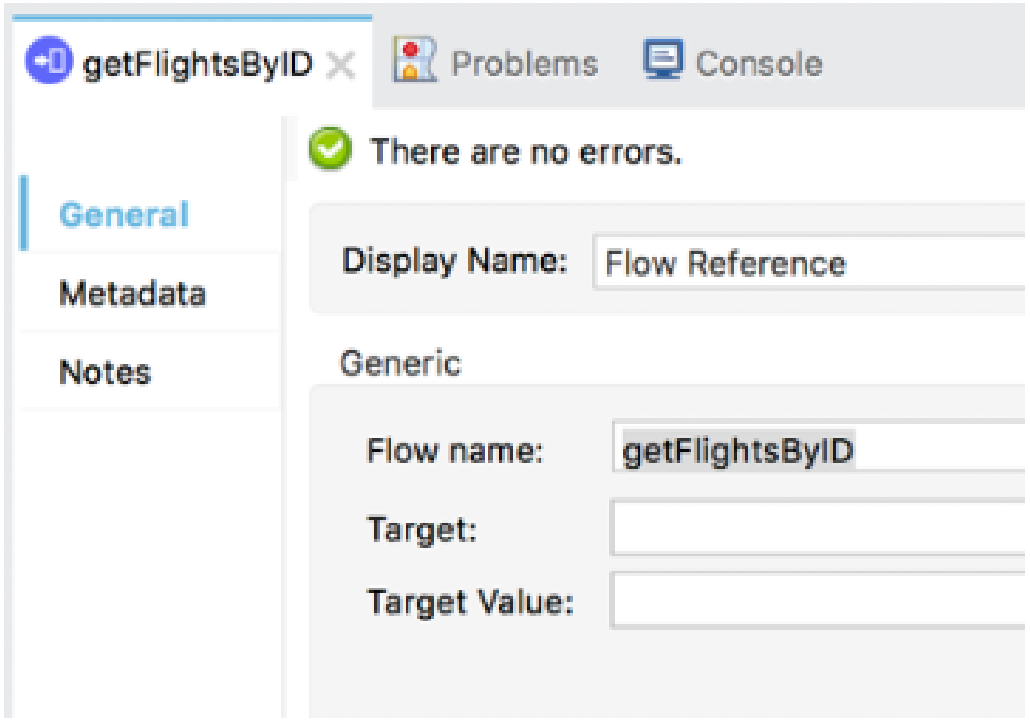
10. In the Flow Reference properties view, select getFlights for the flow name.



11. Change the display name to getFlights

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

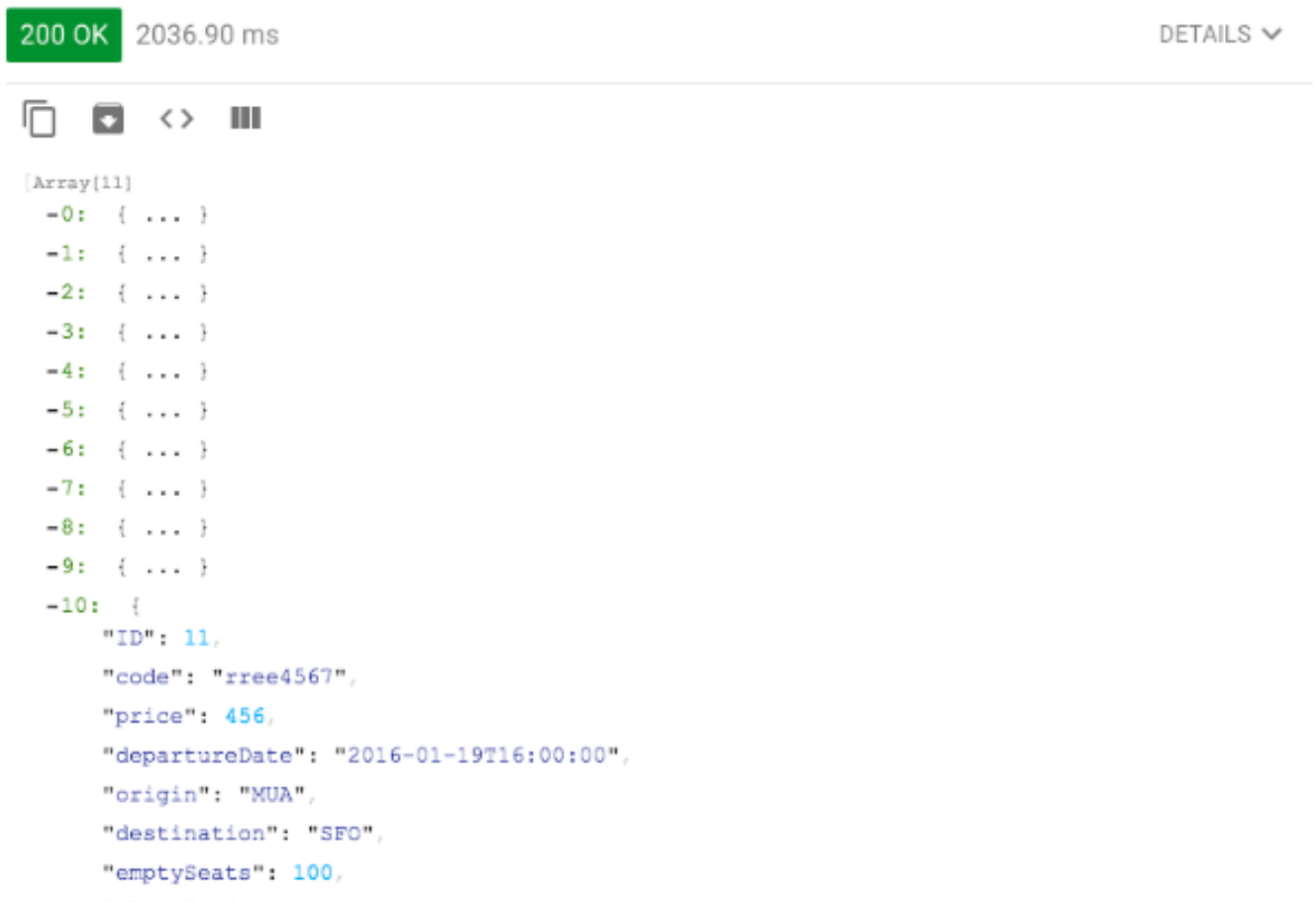
12. Delete both the Transform Message components in the get:/flights/{ID} flow.
13. Drag a Flow Reference component from the Mule Palette and drop it into the flow.
14. In the Flow Reference properties view, select getFlightsByID for the flow name.



15. Change the display name to getFlightsByID.

Test the web service using Advanced REST Client

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



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

Summary

Summary

- **Anypoint Studio** can be used to build Mule applications for integrations and API implementations
  - Two-way editing between graphical and XML views
  - An embedded Mule runtime for testing applications
- **Mule applications** accept and process events through a series of event processors plugged together in a flow
  - Use the **HTTP Listener** as an inbound endpoint to trigger a flow with an HTTP request
  - Use the **Set Payload** transformer to set the payload
  - Use the **Database** connector to connect to JDBC databases
  - Use DataWeave and the **Transform Message** component to transform messages from one data type and structure to another

Summary

- Create RESTful interfaces for applications
  - **Manually** by creating flows with listeners for each resource/method pairing
  - **Automatically** using Anypoint Studio and **APIkit**
- Connect web service interfaces to implementations using the **Flow Reference** component to pass messages to other flows

DIY Exercise 4-1: Implement a REST API using APIkit

Time estimate: 2 hours

Objectives

In this exercise, you implement a REST API that has a RAML specification. You will:

- Use APIkit to create implementation flows.
- Add logic to APIkit auto-generated flows.
- Enhance an API and regenerate the flows using APIkit.

Scenario

A RAML specification for an Accounts API has been published to Anypoint Exchange. You need to implement this API using Anypoint Studio. Account data should be retrieved from the flights_customers database and then transformed to match the Account data type defined in the Accounts API. After you finish implementing the API, the requirements change and you need to extend the existing RAML specification and modify the API implementation.

Use Anypoint Studio and APIkit to create the API implementation from the API

Create a new Anypoint Studio project that includes your API solution from exercise 3-1 or use /files/module03/accounts-mod03-api-solution.zip (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources).

Create a new API interface Mule configuration file by right-clicking the API RAML file and selecting Mule > Generate Flows from REST API. Leave the HTTP Listener's path as /api/*.

Debug the auto-generated Mule application and use the APIkit Console to make GET requests to http://localhost:8081/api/accounts. Be sure to set the required headers and query parameters.

Use a REST client such as Advanced Rest Client to make valid GET requests to http://localhost:8081/api/accounts and remember to set any required headers and query parameters.

Answer the following question

- How does the APIkit router set the HTTP Listener to accept any endpoint?

Implement GET /accounts to retrieve account data from the accounts database

Implement the functionality to return all customer data from the flights_customers database using the following MySQL database credentials:

- Host: mudb.learn.mulesoft.com
- Port: 3306
- User: mule
- Password: mule
- Database: training
- Table: flights_customers

Add additional logic to only return accounts that match a particular type (business or personal). Use an input parameter to safely pass the account type to the database SELECT statement.

Note: For GET /accounts, you DO NOT need to implement the optional query parameters for name and country. However, if you want to implement these query parameters, you can use the Choice router to implement different query scenarios (which is covered later in Module 9 in the Fundamentals course). You also do not need to implement anything for the Request-ID header.

Answer the following question

- What is the data type and format of the database results?

Map results from the database to Account objects

Transform the database results to match the Account schema specified in the accounts-api.raml file. Here is an example of the desired transformed output:

```
[
 {
  "id": "100"
  "firstName": "Alice",
  "lastName": "Green",
  "address": "77 Geary St., San Francisco"
  "postal": "94108"
  "country": "USA",
  "creationDate": "2018-10-01T23:57:59Z+0:00",
  "accountType": "business",
  "miles": 1000
 }
]
```

Hint: You can use the DataWeave splitBy and joinBy functions to split the name into a first name and a last name. Assume that any word after the first space in the name is the last name.

Enhance the API specification for new requirements

Even though you have already started implementing the API implementation, requirements suddenly change; your key stakehol specific accounts, with the ability to retrieve and modify one account based on the account id.

Modify the Accounts API RAML specification with the following new requirements:

- The API has a nested resource /accounts/{id}, where id is a URI parameter that should match the accountID column of t
- The /accounts/{id} resource has an optional type query parameter.
- The /accounts/{id} resource has a GET method that returns an Account object in JSON format.

Note: Don't duplicate the definition of the Account data type in /accounts/id, because the type query parameter is now optional.

- The /accounts/{id} resource has the following additional methods:
    - A PUT method that returns a JSON message:
      {"message": "account replaced (but not really)"}
    - A PATCH method that returns a JSON message:
      {"message": "account modified (but not really)"}
    - A DELETE method that returns a JSON message:
      {"message": "account deleted (but not really)"}
- Each method should have a description.
- Each method should have a custom 400 message error.

Update the API in Anypoint Exchange

Publish the new version of the API to Anypoint Exchange and update the public portal.

Regenerate the API implementation

Use APIkit to regenerate the implementation for your updated RAML file.

Note: There was a bug for certain versions of Anypoint Studio that caused duplicate flows to be created when an API was reimported from Design Center. If this happens to you, undo the step and then directly import the RAML files into Anypoint Studio and do not import them from Design Center.

Answer the following question

- What happens to the existing APIkit-generated private flows?

Implement GET /accounts/{id}

After you regenerate the flows, add functionality to implement the /accounts/{id} resource's GET method. In the flow, get all rows from the flights_accounts database where the accountID matches the supplied id URI parameter, then transform the result to an Accounts JSON object (as specified in the project's datatypes/account.raml file). Use an input parameter in the SQL to safely pass in the account ID to the SELECT query.

Answer the following question

- How does APIkit implement the id URI parameter?

Verify your solution

Load the solution /files/module04/accounts-mod04-api-implementation-solution.jar (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.