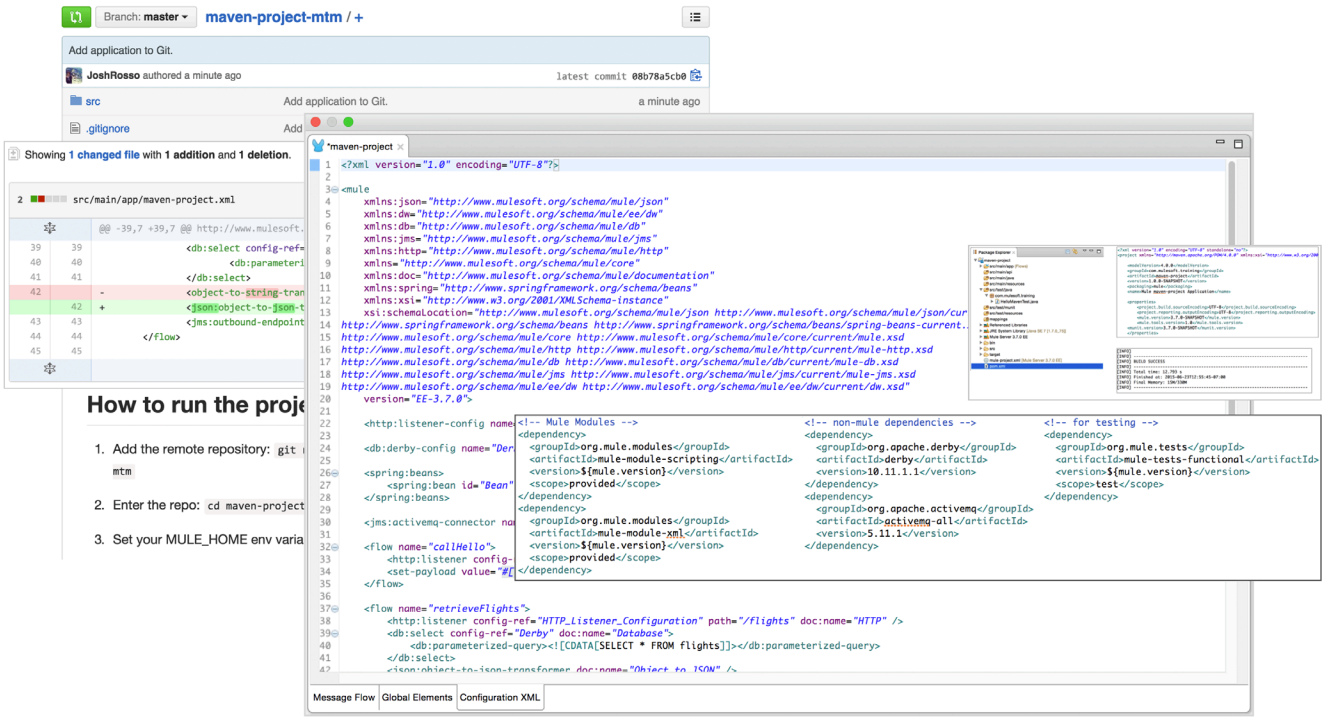
Anypoint Platform Development: Advanced

Student Manual

Mule runtime 3.8
May 20, 2016

Table of Contents

Module 1: Managing Mule Projects with Maven



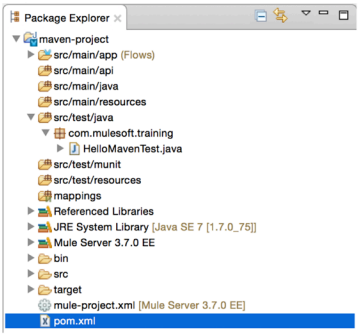
Objectives:

- Manage project dependencies.
- Maintain larger Mule projects.
- Configure a project to support multiple developers.
- Control a multi-phase build lifecycle.

Walkthrough 1-1: Create a Maven-based Mule project

In this walkthrough, you create, explore, and run a Maven-based Mule project. You will:

- Create a new Maven-based Mule project.
- Examine the pom.xml.
- Add a settings.xml.
- Run the project as a Maven-based Mule project.




```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/200

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mulesoft.training</groupId>
    <artifactId>maven-project</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>mule</packaging>
    <name>Mule maven-project Application</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <mule.version>3.7.0-SNAPSHOT</mule.version>
        <mule.tools.version>1.0</mule.tools.version>
    <munit.version>3.7.0-SNAPSHOT</munit.version>
    </properties>
```

```
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 12.793 s
[INFO] Finished at: 2015-06-23T12:55:45-07:00
[INFO] Final Memory: 15M/330M
[INFO] ------------------------------------------------------------------------
```

Make sure Maven is installed and set up

1. From a command-line interface, run mvn –v and make sure you see Maven version and installation directory.

```
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T04:57:37-
07:00)
Maven home: /Users/jeanette.stallons/dev/apache-maven-3.3.3
Java version: 1.8.0_45, vendor: Oracle Corporation
Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.11", arch: "x86_64", family: "mac"
```

Check if you have a local Maven repository

2. In your computer's file explorer, navigate to the default location of the local Maven repository: ${user_home}/.m2/.
   - Linux/Mac: ~/.m2
   - Windows: C:\Users\{yourUser}\.m2

   Note: You may need to enable viewing of hidden files and folders.

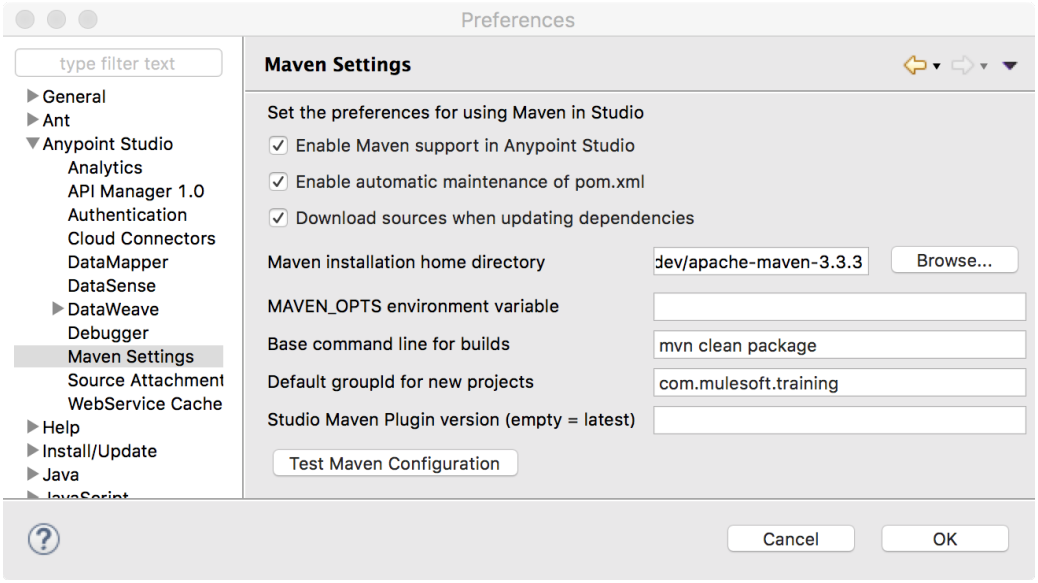
   Note: If this folder does not exist, do not create it. It will be created automatically.

   Note: This location will now be referred to as $M2.

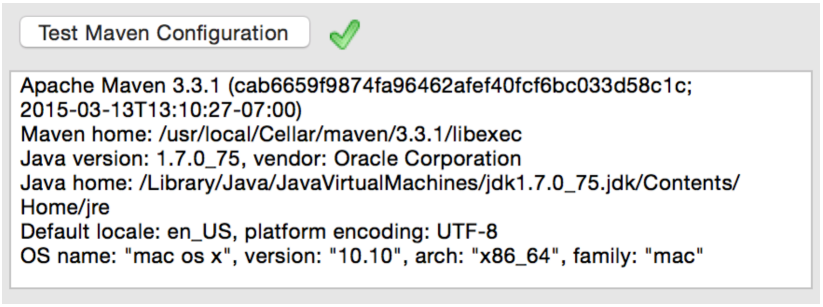
3. If the folder exists, drill-down and look for the repository subfolder: ${user_home}/.m2/repository.
4. Leave this window open.

Configure Anypoint Studio to use Maven

5. Open Anypoint Studio.
6. Open Anypoint Studio preferences.
7. Expand Anypoint Studio and click Maven Settings.
8. Check the three checkboxes for Maven preferences.
9. If the Maven installation home directory is not set, browse to the location of your Maven installation.
10. Set a default groupId for new projects field to com.mulesoft.training.



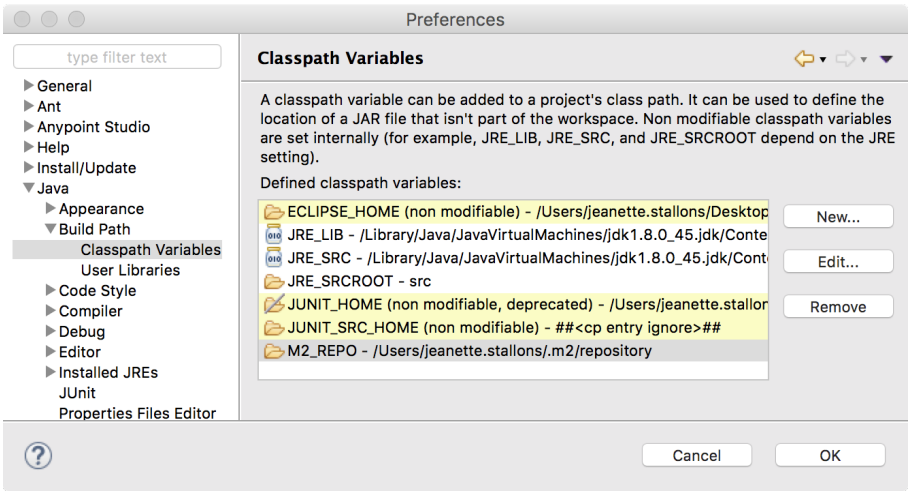
11. Click Test Maven Configuration and verify the test passes.



12. In the left-side navigation, expand Java.
13. Under Build Path, select Classpath Variables.
14. Verify that the JRE_LIB and JRE_SRC variables are pointing to a JDK and not a JRE.

15. Verify that there is an M2_REPO variable and that it is set to ${user_home}/.m2/repository.

    Note: If M2_REPO does not exist, click New and create a variable called M2_REPO that points to your repository location: ${user_home}/.m2/repository.

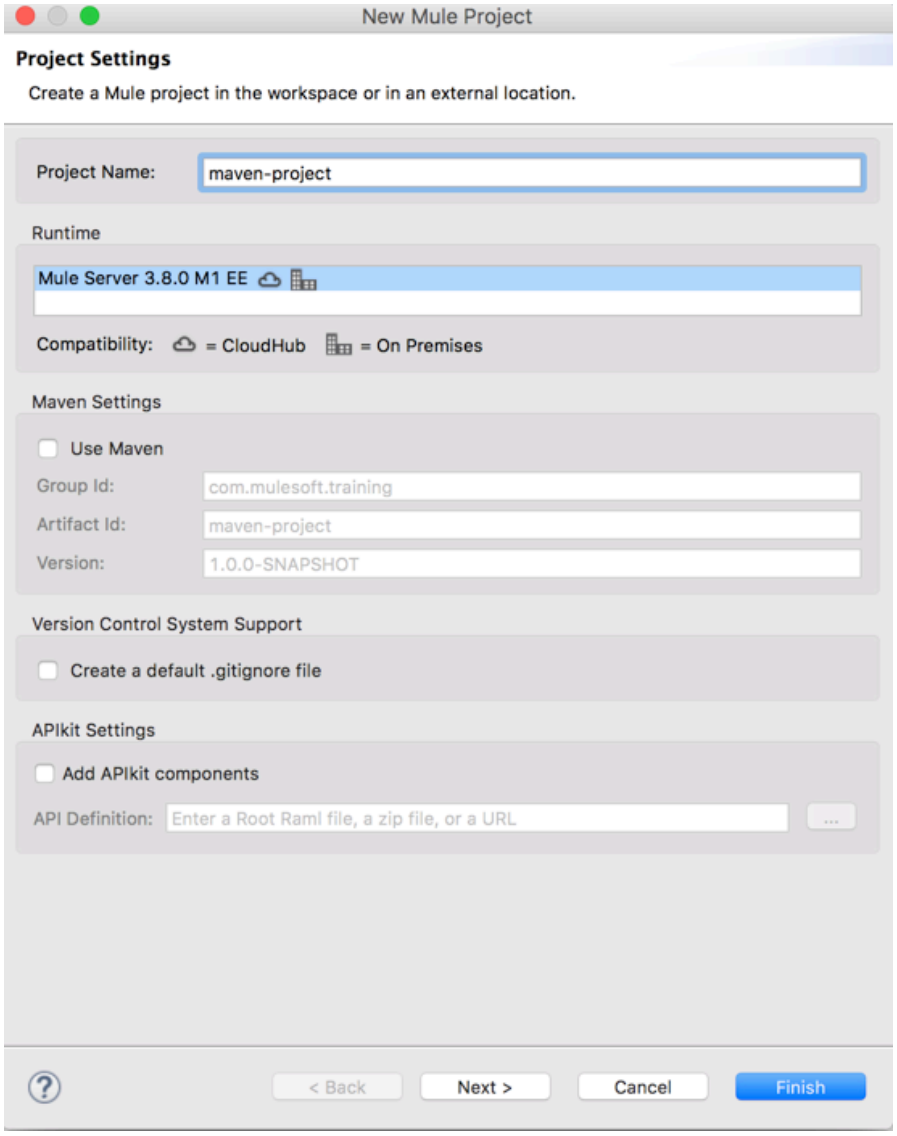


16. Click OK.

Create a Maven-based Mule project

17. Create a new Mule project named maven-project.
18. Check Use Maven.

19. Set group id to com.mulesoft.training.
20. Verify the artifact id is maven-project.



21. Click Finish.
22. Watch the console; you should see many dependencies being downloaded.

Check your local Maven repository

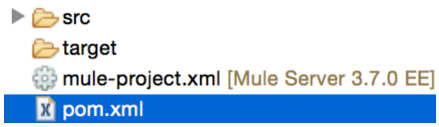
23. Return to $M2/repository in the system explorer.

    Note: Even if this folder did not exist before, it should exist now.

24. Examine the contents of the repository.
25. Leave this window open.

Explore the pom.xml

26. Return to Anypoint Studio.
27. From the Package Explorer, open pom.xml.



28. In the editor, click the Source tab.
29. Examine the following elements in the POM:
    - groupId
    - artifactId
    - version
    - packaging
    - properties
    - dependencies
30. In the dependencies element, locate mule-core-ee.

```
<!-- Xml configuration -->
<dependency>
  <groupId>com.mulesoft.muleesb</groupId>
  <artifactId>mule-core-ee</artifactId>
  <version>${mule.version}</version>
  <scope>provided</scope>
</dependency>
```

Note: mule-core-ee is an example of an enterprise library. Developers must have a MuleSoft license / subscription to access it.

See if the project build succeeds

31. Check the console to make sure the build succeeds.

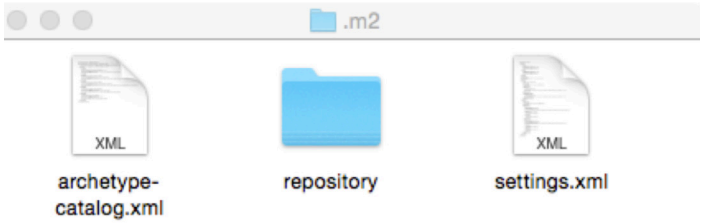
    Note: This may take a few minutes.

Add a settings.xml

32. Return to $M2 in the system explorer.
33. Verify whether a settings.xml preexists in $M2 or $M2/conf.
34. If settings.xml does not exist, open a web browser and navigate to http://mu.mulesoft-training.com/maven/settings.

    Note: This URL is also located in the course snippets.txt file in the student files.

35. Log in with the following credentials; a settings.xml file should be automatically downloaded.
    - username: muletraining.nexus
    - password: MuleTraining
36. Move the downloaded settings.xml file to $M2.



37. Open settings.xml in a text editor.
38. Examine the settings tags.

```
<servers>
   <server>
     <id>MuleRepository</id>
     <username>muletraining.nexus</username>
     <password>MuleTraining</password>
   </server>
 </servers>
```

Note: These are the credentials used when accessing the repository with Id MuleRepository.

39. Locate the repository with an Id of MuleRepository.

```
 <repository>
   <id>MuleRepository</id>
  <name>MuleRepository</name>
  <url>https://repository.mulesoft.org/nexus-ee/content/repositories/releases-
ee/</url>
  <layout>default</layout>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
 </repository>
```

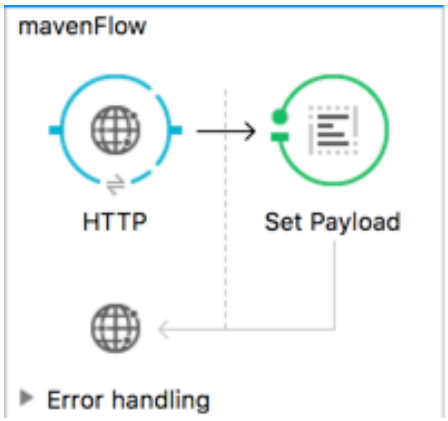
Note: This secured nexus repository is used for dependencies not found in the others by default.

40. Return to Anypoint Studio.

Create a hello world flow

41. Open maven-project.xml.
42. Add a flow.
43. Rename the flow mavenFlow.

44. Add an http:listener to the beginning of the flow.
45. Configure the listener for localhost:8081/hello.
46. Add a set-payload transformer.



47. Set the value attribute to Hello World.

```
#['Hello World']
```

48. Verify the configuration matches the code shown here:

```
<http:listener-config host="localhost" name="httpList" port="8081" />

<flow name="mavenFlow">
  <http:listener path="orders" config-ref="httpList" />
  <set-payload value="#['Hello World']"/>
</flow>
```

Run the project

49. Run the project.
50. In the console, verify the application deploys.

```
**********************************************************************************
*- - + APPLICATION + - -         *       - - + DOMAIN + - -       * - - + STATUS + - -
- *
**********************************************************************************
* maven-project                  * default                        * DEPLOYED
**********************************************************************************
```

51. In the right-corner of the console, click the Display Selected Console button.
52. In the console, verify the build is successful.

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:39 min
[INFO] Finished at: 2015-11-13T14:10:30-08:00
[INFO] Final Memory: 13M/131M
[INFO] ------------------------------------------------------------------------
```

Walkthrough 1-2: Control a multi-phase build lifecycle

In this walkthrough, you run Maven lifecycle commands for the project from the command-line. You will:

- Determine the Maven commands executed when a project is run in Anypoint Studio.
- Run lifecycle commands for a project from a command-line interface.
- Add a test to a project.
- Build a project that fails and then passes a test.
- Package the project with and without running tests.

Determine the Maven commands executed when a project is run

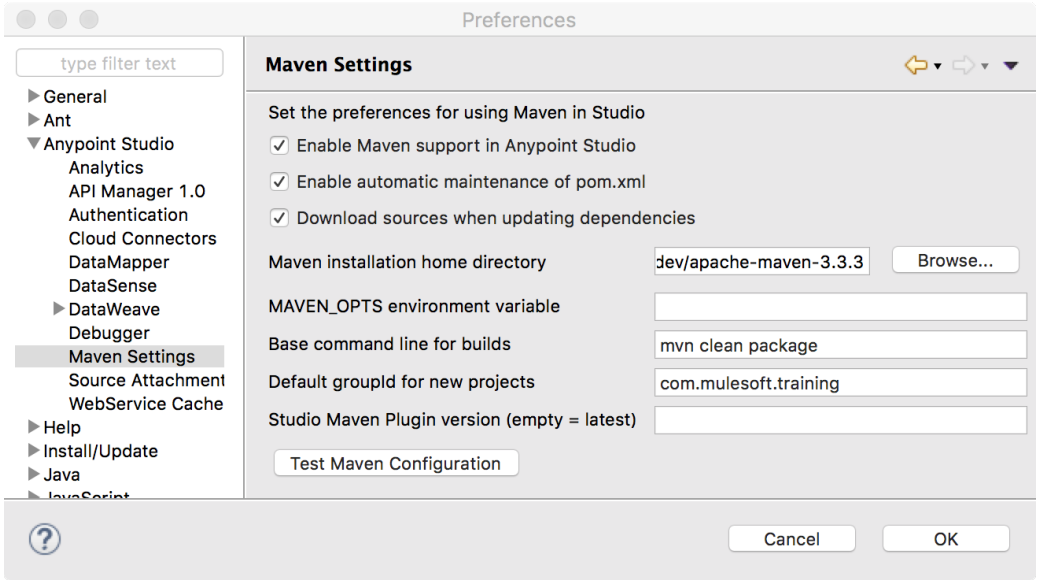
1. In the maven-project console, locate the mvn commands executed when the application was run.

   ```
   [14:33:58] Running: mvn clean package
   ```

   Note: This will create an application deployable to be used by Anypoint Studio's embedded runtime.

Locate the base command specified in Maven Settings

2. Open Anypoint Studio preferences.
3. Expand Anypoint Studio and click Maven Settings.
4. Locate the base command specified for builds.
5. Click OK.



Run lifecycle commands for the project from the command-line

6. From a command-line interface, navigate to the project's location.

   Note: You can get the project's location by right-clicking it in the Package Explorer, selecting Properties, and copying the value for the location.

   Note: This location will be referred to as $PROJECT_HOME.

7. Run ls on Mac/Linux or dir on Windows and examine the output.

```
drwxr-xr-x  5 mule  staff   170 Jun 23 11:47 bin
drwxr-xr-x  2 mule  staff    68 Jun 23 11:47 mappings
-rw-r--r--  1 mule  staff   267 Jun 23 11:47 mule-project.xml
-rw-r--r--  1 mule  staff  7698 Jun 23 12:55 pom.xml
drwxr-xr-x  4 mule  staff   136 Jun 23 11:47 src
drwxr-xr-x  7 mule  staff   238 Jun 23 15:35 target
```

8. Run ls target or dir target to look at the contents of the target folder.

```
drwxr-xr-x  6 mule  staff   204 Nov 13 15:09 classes
-rw-r--r--  1 mule  staff  1613 Nov 13 15:09 maven-project-1.0.0-SNAPSHOT.zip
drwxr-xr-x  3 mule  staff   102 Nov 13 15:09 maven-status
drwxr-xr-x  6 mule  staff   204 Nov 13 15:09 test-classes
```

9. Run mvn clean.

   Note: In order for mvn commands to function, you need Maven and Java in your PATH. Maven also requires JAVA_HOME to point to the root of your JDK.

10. Run ls or dir again; you should no longer see the target directory.

    Note: mvn clean will remove the target directory before calling other phases. It's often combined with other commands.

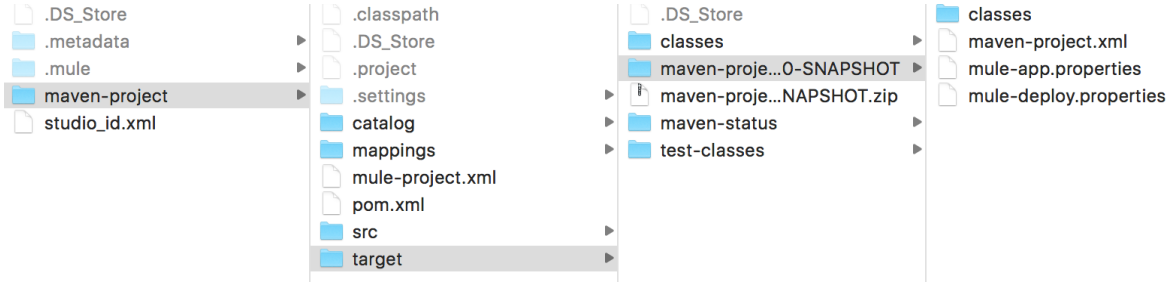
11. Run mvn test.
12. Run ls or dir again; you should see a target directory again.
13. Run ls target or dir target to look at the contents of the target folder again; you should not see the deployable ZIP.

```
drwxr-xr-x  6 mule  staff   204 Nov 13 15:09 classes
drwxr-xr-x  3 mule  staff   102 Nov 13 15:09 maven-status
drwxr-xr-x  6 mule  staff   204 Nov 13 15:09 test-classes
```

14. Run mvn clean.
15. Run ls or dir again.
16. Run mvn package.
17. Run ls or dir again.
18. Run ls target or dir target again; you should see the deployable ZIP again.

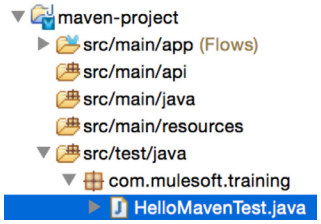
    Note: This archive can be deployed to a Mule runtime in the cloud or on-prem.

19. Unpack maven-project-1.0.0-SNAPSHOT.zip and examine its contents.



Add a test to the project

20. In your system explorer, navigate to the studentFiles for the course.
21. Copy the com folder located in /tests and add it to your project's src/test/java folder in Anypoint Studio.



Examine the test code

22. Open src/test/java/com/mulesoft/training/HelloMavenTest.java.
23. Make sure the name of the config file matches the name of your XML file.

```
@Override
protected String getConfigFile() {
        return "maven-project.xml";
}
```

24. Make sure the name of the flow in the test method (mavenFlow) matches the name of your flow.

```
@Test
public void mavenFlowReturnsHelloMaven() throws Exception {
        runFlowAndExpect("mavenFlow", "Hello Maven");
}
```

25. Notice that the test expects the flow to return a payload of Hello Maven.

```
@Test
public void mavenFlowReturnsHelloMaven() throws Exception {
        runFlowAndExpect("mavenFlow", "Hello Maven");
}
```

Run the project

26. Run the project and verify you receive a build failure.

```
Results : Failed tests:
 HelloMavenTest.mavenFlowReturnsHelloMaven:10
 >FunctionalTestCase.runFlowAndExpect:356 expected:
 <Hello [Maven]> but was:<Hello[World]>
 Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.616 s
[INFO] Finished at: 2015-06-22T12:57:34-07:00
[INFO] Final Memory: 33M/315M
[INFO] ------------------------------------------------------------------------
```

Note: Maven is executing mvn package. Part of the package lifecycle is mvn test. Unless the –DskipTests parameter is used, Maven will always run unit tests before packaging the application.

Package the project with and without running tests externally

27. Return to $PROJECT_HOME in the command-line interface.
28. Run mvn clean package.
29. Verify the build fails.
30. Run ls target or dir target; you should not see a deployable ZIP.
31. Run mvn clean package –DskipTests.
32. Verify the build succeeds.

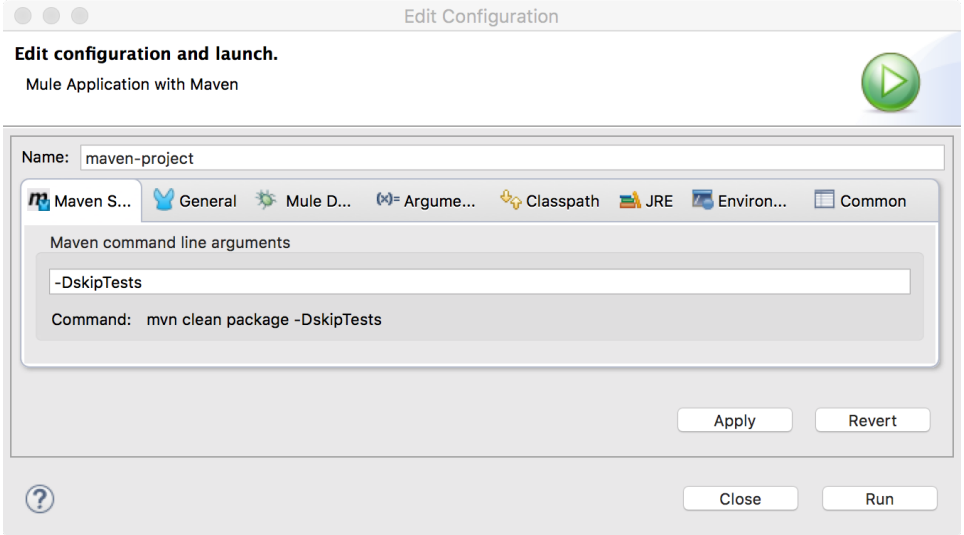
```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.616 s
[INFO] Finished at: 2015-06-22T12:57:34-07:00
[INFO] Final Memory: 33M/315M
[INFO] ------------------------------------------------------------------------
```

33. Run ls target or dir target again; you should now see a deployable ZIP.

Run the project without running tests in Anypoint Studio

34. Return to Anypoint Studio.
35. Right-click the project and select Run As > Mule Application with Maven (configure).

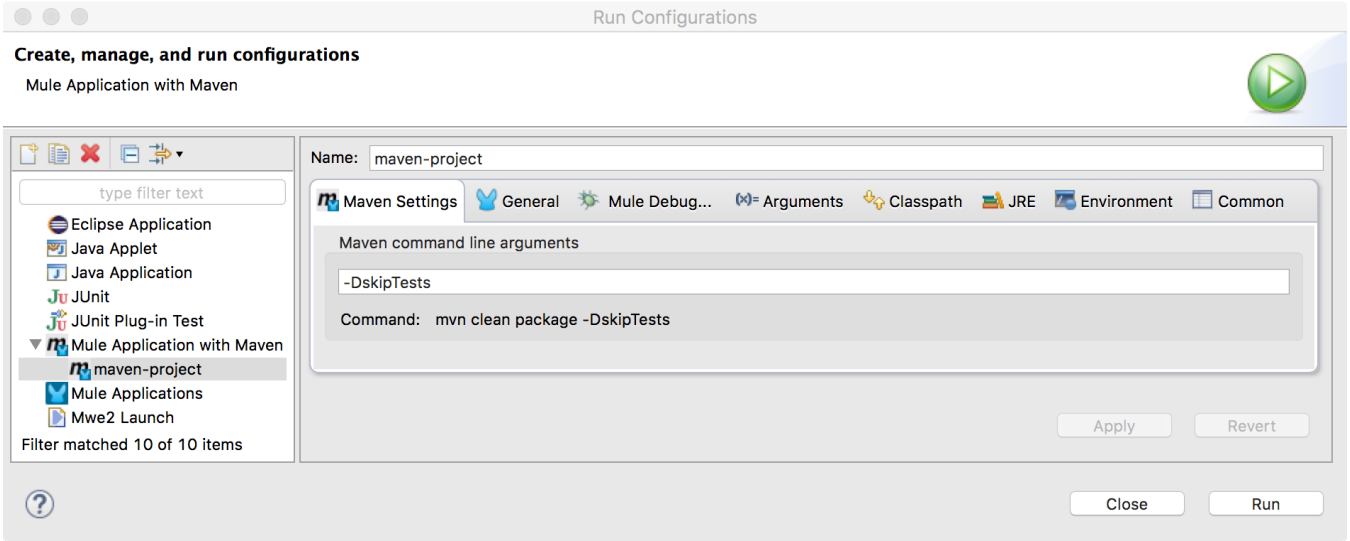
36. In the Edit Configuration dialog box, set the Maven command-line arguments to –DskipTests.



37. Click Run.
38. Verify the application deploys.
39. Make a request to http://localhost:8081/hello and verify you receive a response.

Run the project with tests again

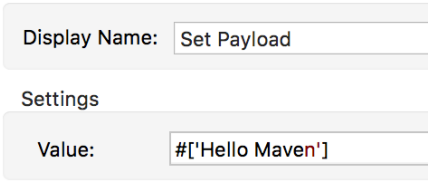
40. Stop the project.
41. Select Run > Run Configurations.
42. In the Run Configurations dialog box, locate the Maven Settings tab.
43. Remove the –DskipTests command-line argument.



44. Click Run.
45. Verify the build fails and the application is not deployed.

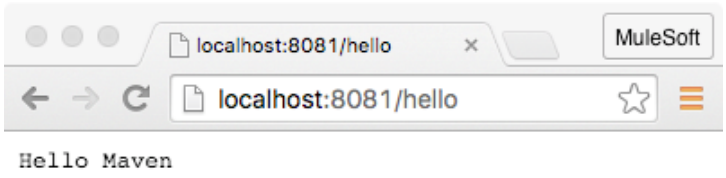
Fix the code so it passes the test

46. Return to maven-project.xml.
47. Change the Set Payload processor to return Hello Maven.



Run the project

48. Run the project again and verify it builds and starts successfully.
49. Make a request to http://localhost:8081/hello and verify you receive a response.

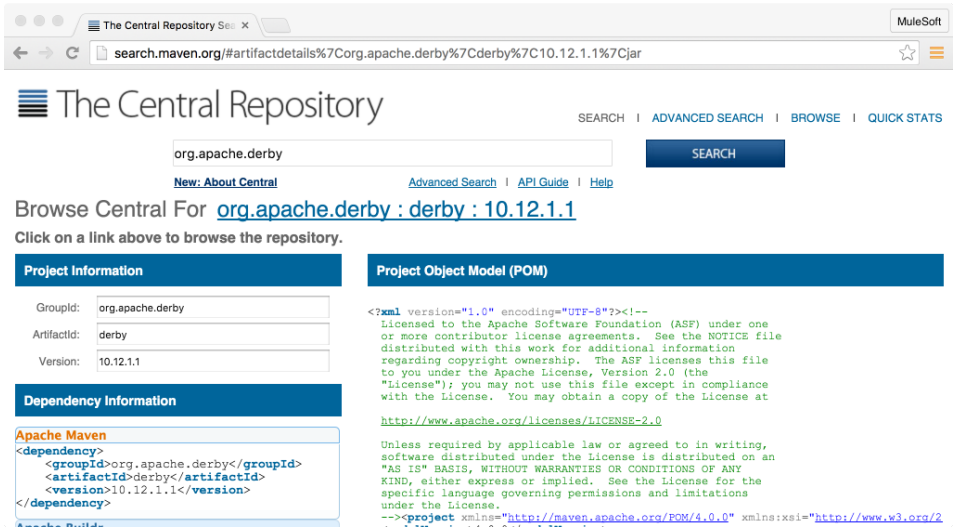


50. Stop the project.

Walkthrough 1-3: Add dependencies to a Maven-based Mule project

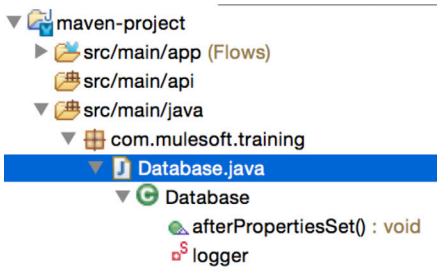
In this walkthrough, you add Derby and ActiveMQ dependencies to a Maven-based Mule project. You will:

- Get dependency information from The Central Repository.
- Add dependencies to the POM.
- Create and use Maven properties.
- Download dependencies from The Central Repository.



Create an in-memory database

1. From studentFiles/java, drag the com directory into your project's src/main/java folder.

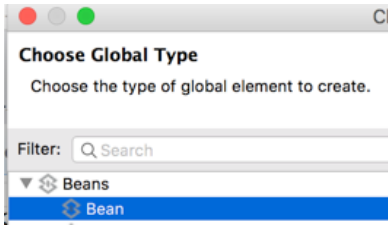


2. Open Database.java and examine its contents to understand its purpose.
3. Copy the value of the dbURL string: jdbc:derby:memory:muleEmbeddedDB;create=true.

   Note: You will use this value shortly.

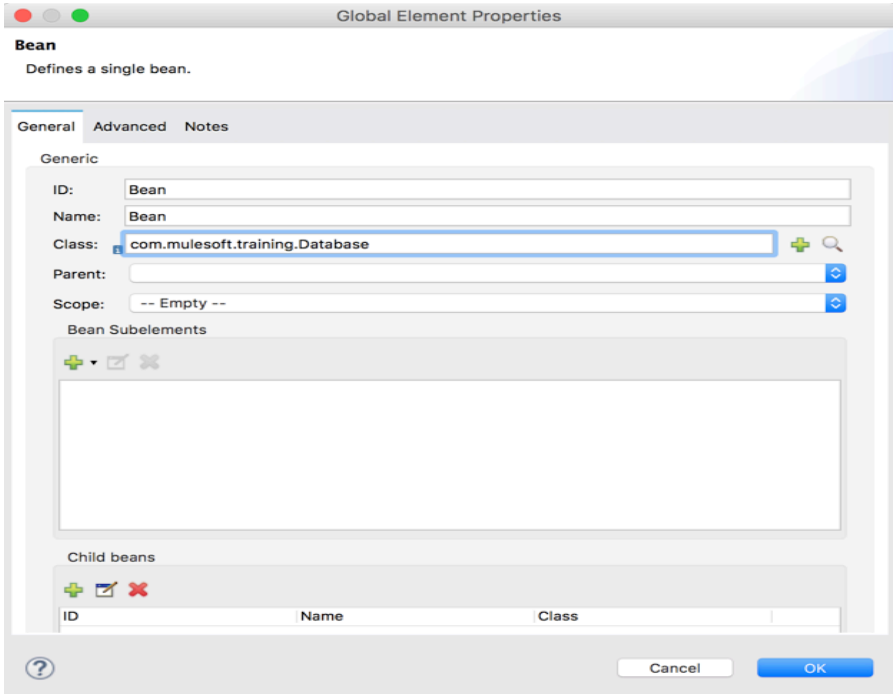
4. In maven-project.xml, switch to the Global Elements tab.

5. Create a new Bean global element.



Note: This bean is used at startup to generate an in-memory database.

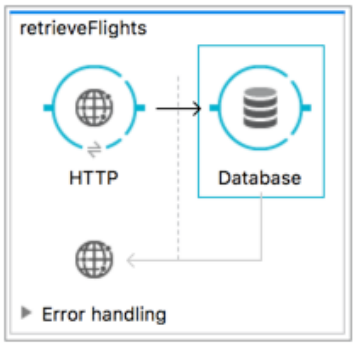
6. Set the bean's class to com.mulesoft.training.Database.



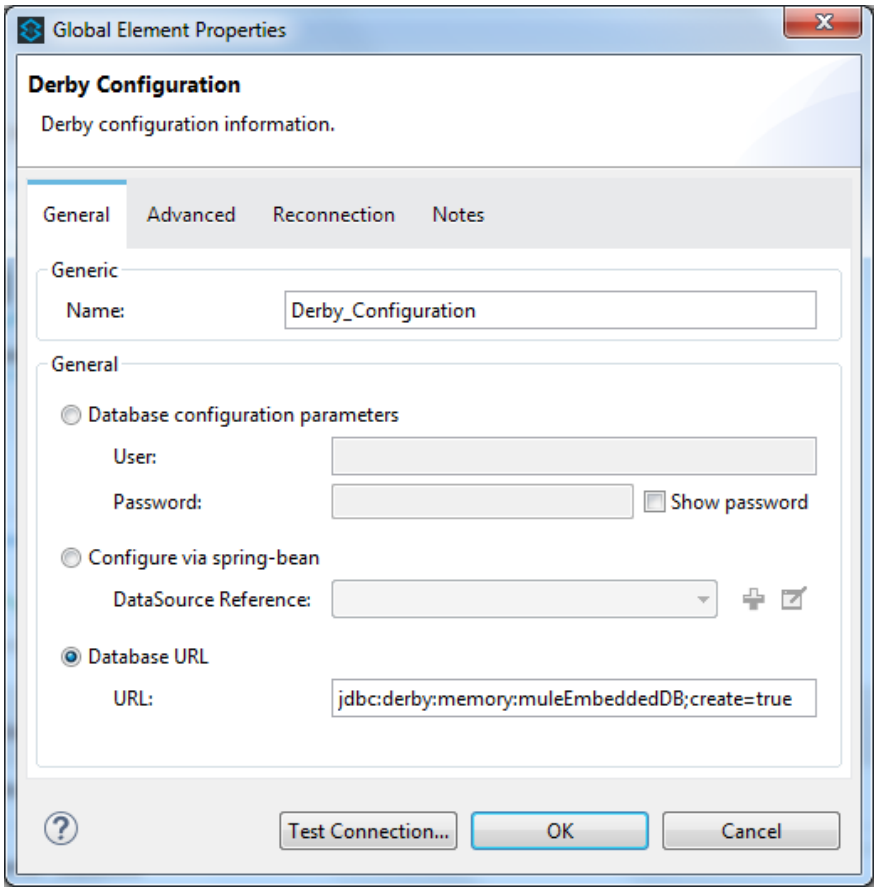
Create a flow to retrieve data from the database

7. Create a flow named retrieveFlights that receives requests at http://localhost:8081/flights.

8. Add a Database element to the flow.



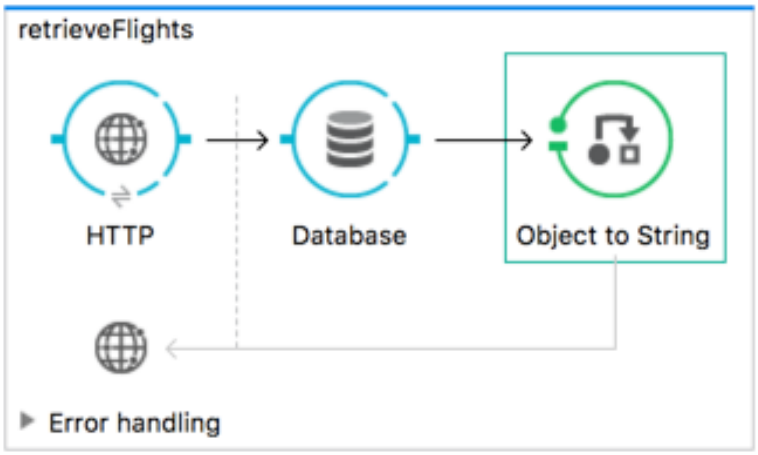
9. Create a new database connector configuration.
10. Select Derby Configuration.
11. Set the database URL to the value you copied from Database.java earlier:
    jdbc:derby:memory:muleEmbeddedDB;create=true



12. Configure the Database connector with a select operation to query for all FLIGHTS.

```
SELECT * FROM FLIGHTS
```

13. After the Database endpoint, add an object-to-string-transformer.



Run the project

14. Run the project and examine the console.
15. Find the line describing why the project failed.

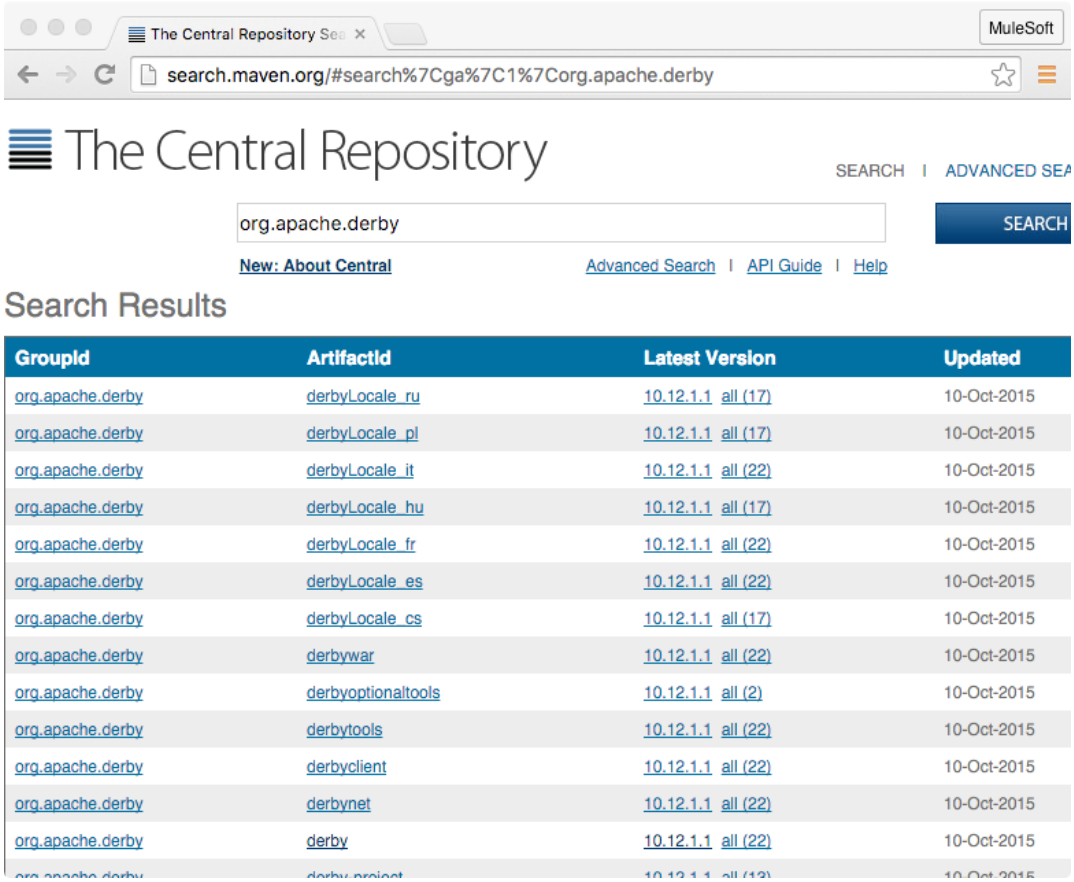
```
org.mule.config.spring.MuleArtifactContext: Exception encountered during context
initialization - cancelling refresh attempt
org.springframework.beans.factory.BeanCreationException: Error creating bean with
name 'com.mulesoft.training.Database#0' defined in URL
[file:/Users/josh/AnypointStudio/advancedSpace/maven-project/target/test-
classes/maven-project.xml]:  Invocation of init method failed; nested exception is
java.lang.ClassNotFoundException: org.apache.derby.jdbc.EmbeddedDriver
```

Note: The project is missing the derby.jdbc.EmbeddedDriver class.

Get Derby dependency information

16. In a web browser, go to http://search.maven.org.
17. Search for org.apache.derby.

18. Locate the line containing an ArtifactId of derby.



19. Click the latest version number to the right.
20. Locate the Dependency Information section.
21. Copy the dependency XML.

```
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derby</artifactId>
  <version>10.12.1.1</version>
</dependency>
```

22. Return to the Anypoint Studio.
23. Return to the project's pom.xml.
24. Locate the comment <!-- for testing -->.
25. Above that line, add the comment <!-- non-mule dependencies -->.

26. Under <!-- non-mule dependencies -->, paste the dependency XML.

```
<!-- non-mule dependencies -->
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derby</artifactId>
  <version>10.12.1.1</version>
</dependency>
```

Create and use a Maven property

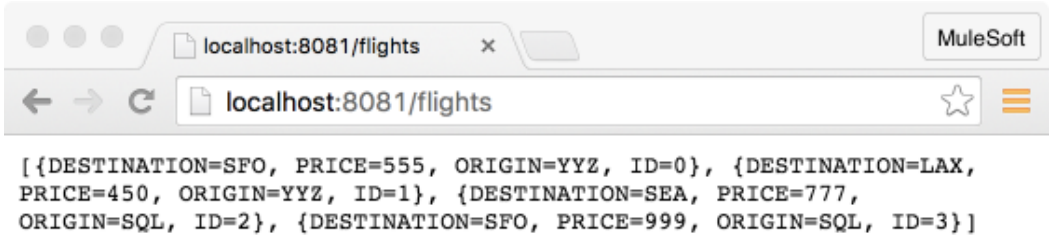
27. Cut the version value from inside the version tags.
28. Set the version value to the property ${derby.version}.
29. Locate the properties tag near the top of the POM.
30. Add the derby version to the properties tag.

```
<properties>
  <!-- more properties here -->
  <derby.version>10.12.1.1</derby.version>
</properties>
```

Note: Extracting versions to properties has no functional impact.

Run the project

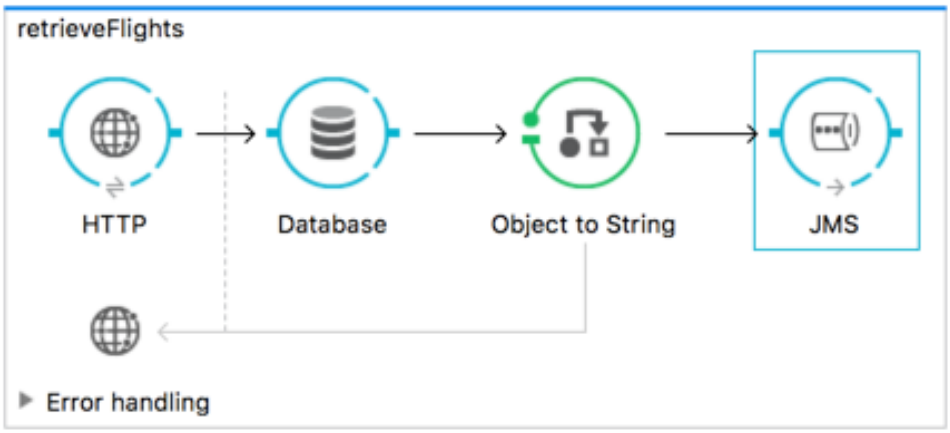
31. Run the project and examine the console; the project should now build successfully.
32. Navigate to http://localhost:8081/flights; you should see the flights returned.



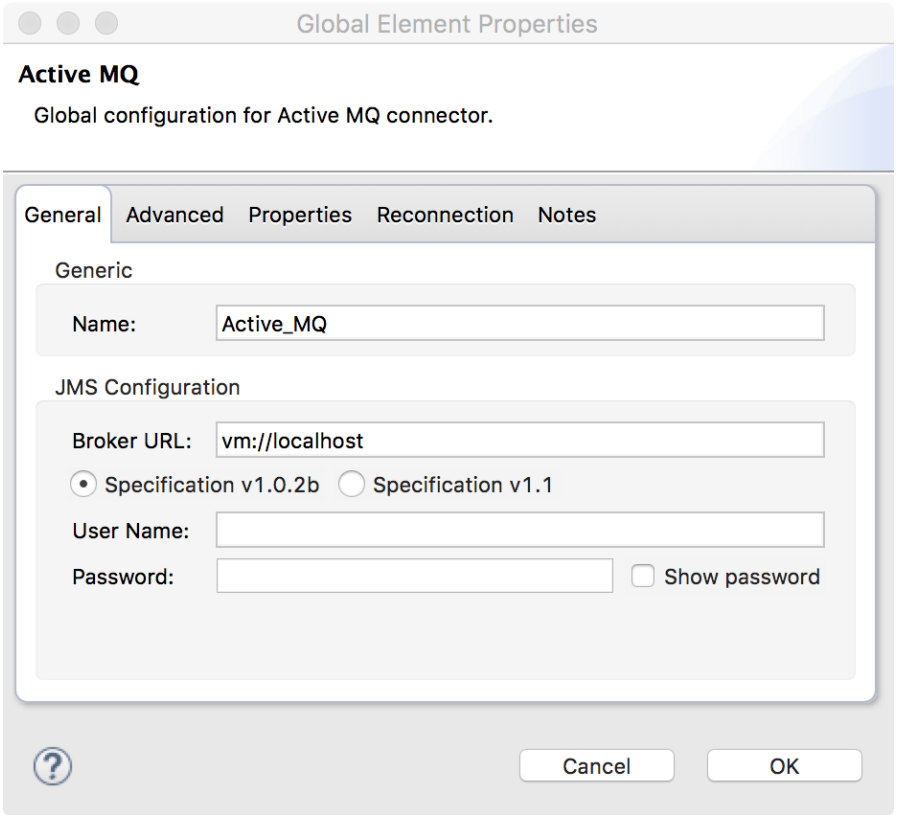
33. Stop the project.

Publish the payload to an ActiveMQ queue

34. After the object-to-string-transformer, add a JMS outbound-endpoint to the flow.



35. Select the JMS outbound-endpoint.
36. Configure the queue to flights.
37. Create a new connector configuration.
38. Select Active MQ.
39. Set the Broker URL value to vm://localhost.



Note: This will create an embedded broker using ActiveMQ libraries.

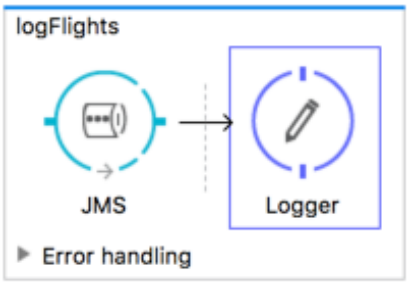
40. Verify your flow is configured as shown here:

```
<flow name="retrieveFlights">
   <http:listener path="flights" config-ref="HTTP_Listener_Configuration" />
   <db:select config-ref="Derby_Configuration">
     <db:parameterized-query><![CDATA[SELECT * FROM FLIGHTS]]>
     </db:parameterized-query>
   </db:select>
   <object-to-string-transformer />
   <jms:outbound-endpoint queue="flights" connector-ref="Active_MQ"/>
 </flow>
```

Create a flow to consume messages from the queue

41. Create a new flow named logFlights.
42. Add a JMS inbound-endpoint listening on the queue flights.
43. Add a logger that displays the payload.
44. Verify your flow is configured as shown here:

```
<flow name="logFlights">
  <jms:inbound-endpoint queue="flights" connector-ref="Active_MQ"/>
  <logger message="#[message.payload]" level="INFO"/>
</flow>
```



Run the project

45. Run the project.
46. Examine the console and locate the root cause of failure.

```
Root Exception was: org.apache.activemq.ActiveMQConnectionFactory. Type: class
java.lang.ClassNotFoundException Tests run: 1, Failures: 0, Errors: 1, Skipped: 0,
Time elapsed: 4.541 sec <<< FAILURE! - in com.mulesoft.training.HelloMavenTest
mavenFlowReturnsHelloMaven(com.mulesoft.training.HelloMavenTest)  Time elapsed: 3.936
sec  <<< ERROR! org.mule.retry.RetryPolicyExhaustedException:
org.apache.activemq.ActiveMQConnectionFactory (java.lang.ClassNotFoundException)
```

Note: You are missing the ActiveMQ library.

Add the ActiveMQ dependency

47. Get the latest activemq-client dependency information from The Central Repository.
48. Add the dependency to the POM.
49. Create and use a Maven property for the ActiveMQ version.
50. Repeat these steps to also add activemq-broker and activemq-kahadb-store dependencies.

Run the project

51. Run the project and verify it deploys successfully.
52. Make a request to http://localhost:8081/flights and verify the console displays the flights.

```
org.mule.api.processor.LoggerMessageProcessor: [{DESTINATION=SFO, PRICE=555,
 ORIGIN=YYZ, ID=0}, {DESTINATION=LAX, PRICE=450, ORIGIN=YYZ, ID=1}, {DESTINATION=SEA,
 PRICE=777, ORIGIN=SQL, ID=2}, {DESTINATION=SFO, PRICE=999, ORIGIN=SQL, ID=3}]
```

53. Stop the project.

Walkthrough 1-4: Change dependency scopes

In this walkthrough, you get familiar with dependency scopes. You will:

- Verify what dependencies are packaged with the application.
- Configure a dependency to be part of the application package.
- Configure a dependency to be excluded from the application package.

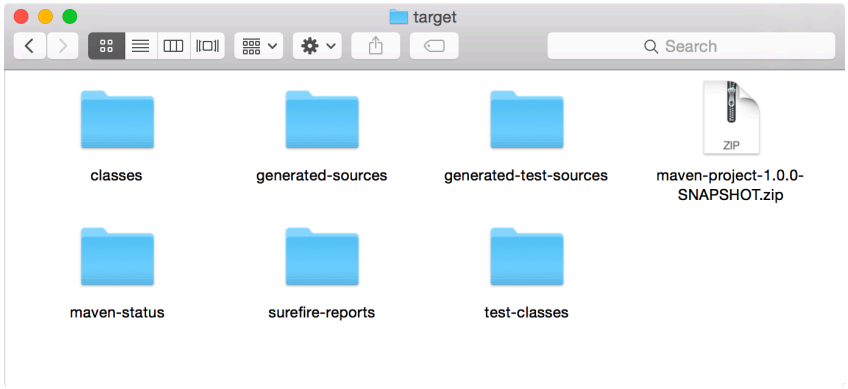
```
<!-- Mule Modules -->
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-scripting</artifactId>
  <version>${mule.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-xml</artifactId>
  <version>${mule.version}</version>
  <scope>provided</scope>
</dependency>
```

```
<!-- non-mule dependencies -->
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derby</artifactId>
  <version>10.11.1.1</version>
</dependency>
<dependency>
  <groupId>org.apache.activemq</groupId>
  <artifactId>activemq-all</artifactId>
  <version>5.11.1</version>
</dependency>
```

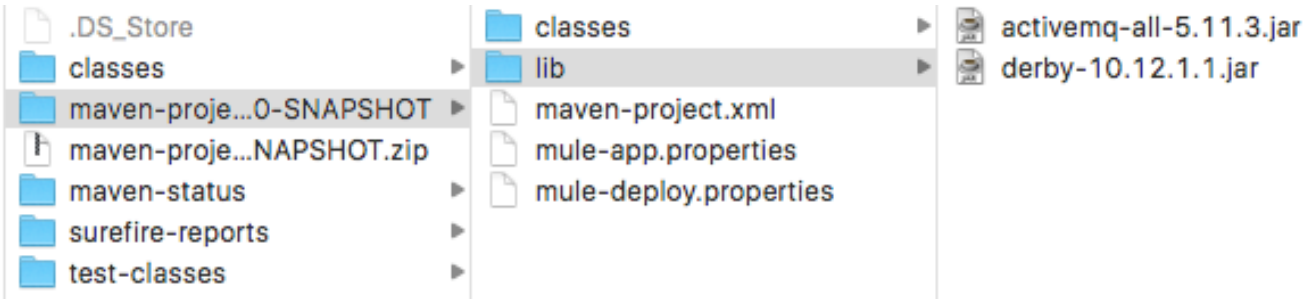
```
<!-- for testing -->
<dependency>
  <groupId>org.mule.tests</groupId>
  <artifactId>mule-tests-functional</artifactId>
  <version>${mule.version}</version>
  <scope>test</scope>
</dependency>
```

Confirm compile scope dependencies are packaged with the application

1. Return to $PROJECT_HOME in the command-line interface.
2. Run mvn clean package.
3. After the build succeeds, open $PROJECT_HOME/target in your system explorer.



4. Unpack maven-project-1.0.0-SNAPSHOT.zip.
5. Open the lib directory and examine the contents; you should see the two external libraries that had a scope of compile.



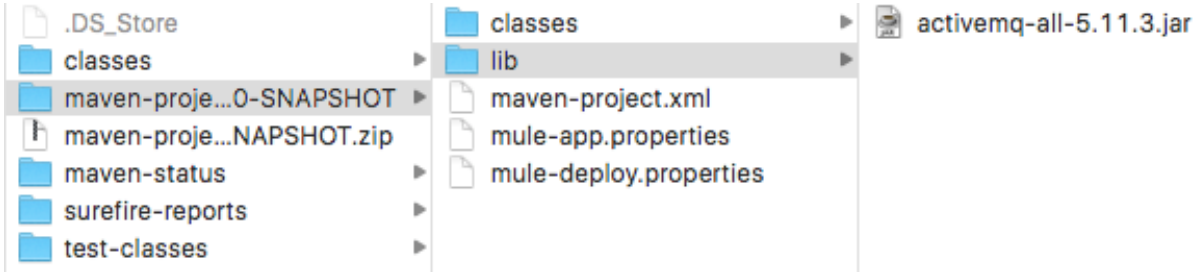
Set a dependency scope to provided

6. Return to the pom.xml in Anypoint Studio.
7. Locate the derby dependency.
8. Below version, add scope tags.
9. Set the value of scope to provided.

```
<dependency>
   <groupId> org.apache.derby</groupId>
   <artifactId>derby</artifactId>
   <version>${derby.version}</version>
   <scope>provided</scope>
 </dependency>
```

Confirm provided scope dependencies are not packaged with the application

10. Return to $PROJECT_HOME in the command-line interface.
11. Run mvn clean package.
12. In the system explorer, unpack $PROJECT_HOME/maven-project-1.0.0-SNAPSHOT.zip.
13. Open the lib directory and examine the contents; it should no longer include the derby JAR.



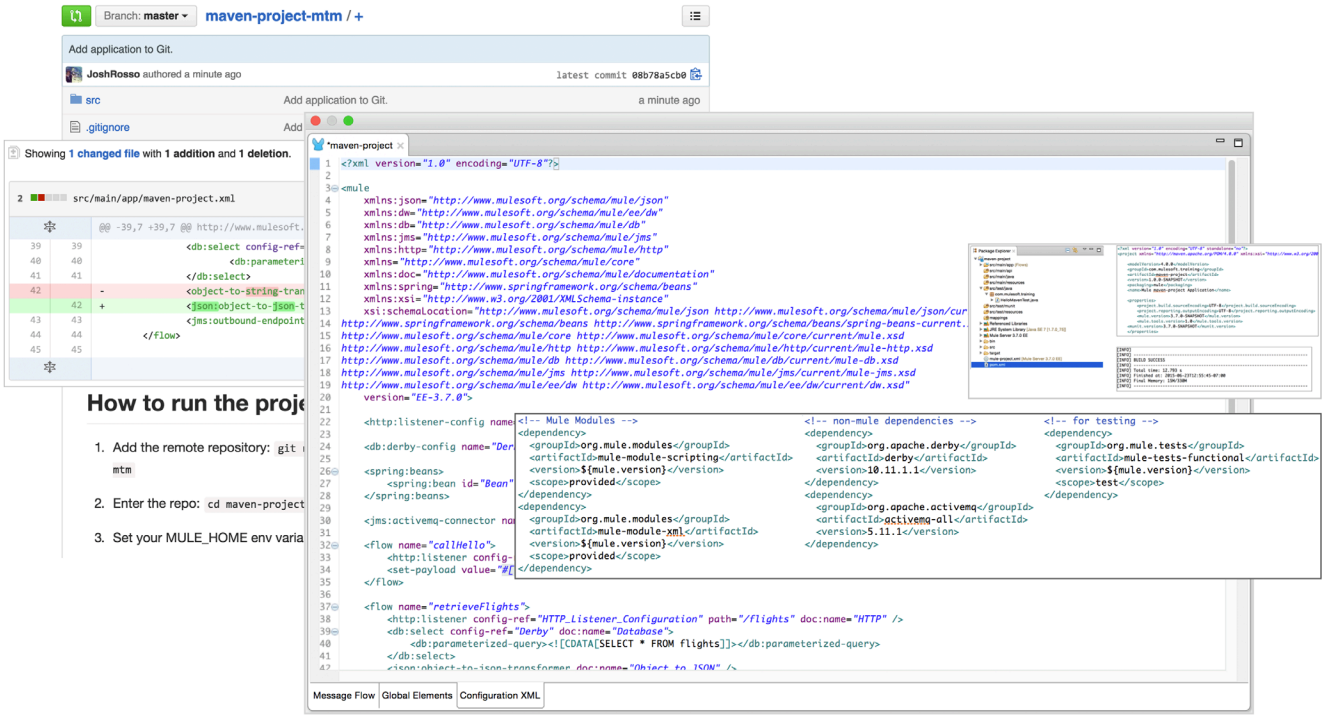
Note: For on-prem Mule runtimes, it is now your responsibility to provide the derby JAR at the server level. For the cloud, you cannot do this; you cannot add external libraries to CloudHub.

Note: If you try to run your project with Anypoint Studio again, the build will succeed but the application will fail to deploy. You will get a ClassNotFoundException: org.apache.derby.jdbc.EmbeddedDriver again.

Set the dependency scope back to compile

14. Return to the pom.xml in Anypoint Studio.
15. Inside the derby dependency, remove the scope tags.

Module 2: Managing Mule Code



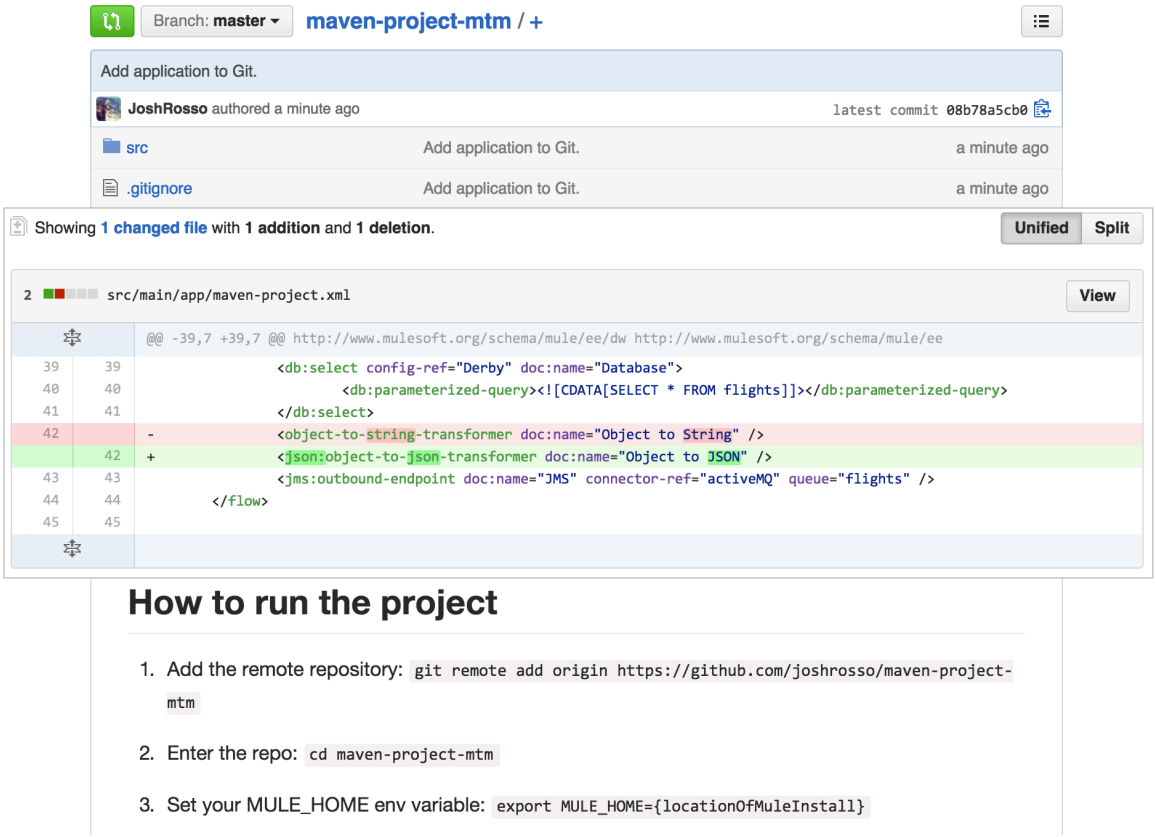
Objectives:

- Maintain larger Mule projects.
- Configure a project to support multiple developers.
- Use a source code management tool (Git).
- Format Mule code.

Walkthrough 2-1: Commit code to source control (Git)

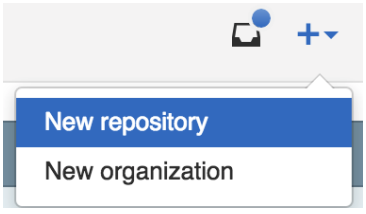
In this walkthrough, you will:

- Initialize a directory as a Git repository.
- Assign a remote repository for code pushes.
- Specify what files should be ignored.
- Stage and commit code in the local repository.
- Push code to the remote repository.

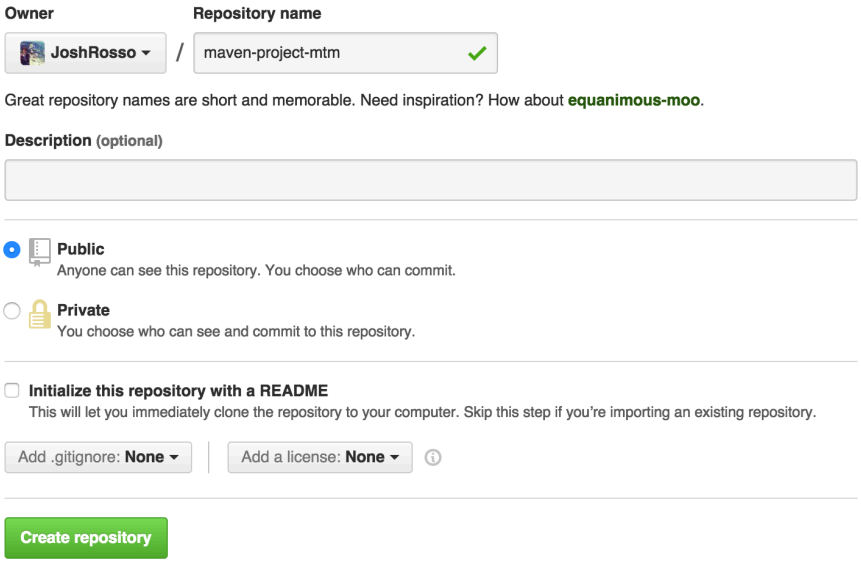


Create a GitHub repository

1. Log in to github.com.
2. In the top right, click the plus sign.
3. Select New Repository.



4. Name the repository maven-project.
5. Click Create repository.



6. Copy the code to push an existing repository from the command-line.

```
git remote add origin https://github.com/{yourGitHubUsername/maven-project}.git
```

Set up the local repository

53. Return to $PROJECT_HOME in the command-line interface.
54. Run ls –la or dir /a to display all files and directories including hidden files and folders.
7. Run git init.
55. Display all files again; you should now see a .git file.

```
drwxr-xr-x  15 mule  staff   510 Nov 14 02:45 .
drwxr-xr-x   7 mule  staff   238 Nov 13 15:30 ..
-rw-r--r--@  1 mule  staff  8196 Nov 13 17:14 .DS_Store
-rw-r--r--   1 mule  staff  2769 Nov 14 02:22 .classpath
drwxr-xr-x  10 mule  staff   340 Nov 14 02:45 .git
-rw-r--r--   1 mule  staff   592 Nov 14 02:22 .project
drwxr-xr-x   4 mule  staff   136 Nov 13 17:02 .settings
drwxr-xr-x   3 mule  staff   102 Nov 13 17:00 activemq-data
drwxr-xr-x   4 mule  staff   136 Nov 13 16:19 catalog
-rw-r--r--   1 mule  staff   804 Nov 13 18:08 derby.log
drwxr-xr-x   2 mule  staff    68 Nov 13 14:07 mappings
-rw-r--r--   1 mule  staff   267 Nov 13 14:07 mule-project.xml
-rw-r--r--   1 mule  staff  6116 Nov 14 02:21 pom.xml
drwxr-xr-x   4 mule  staff   136 Nov 13 14:07 src
drwxr-xr-x   8 mule  staff   272 Nov 14 02:22 target
```

8. Add the remote repo with the command copied earlier.

```
git remote add origin https://github.com/{yourGitHubUsername}/maven-project.git
```

9. Run git remote show origin to display the origin now associated with this project.

```
* remote origin
  Fetch URL: https://github.com/jeanettestallons/maven-project.git
  Push  URL: https://github.com/jeanettestallons/maven-project.git
  HEAD branch: (unknown)
```

10. Return to the course snippets.txt file and copy the code for README.md.
11. Create a new file called README.md in $PROJECT_HOME that contains the copied text.

    Note: Be sure to replace {yourGitHubUsername} with your username.

```
# maven-project

 This is my Maven project from MuleSoft's development class

 ## How to run the project

1. Add the remote repository: `git remote add origin
https://github.com/{yourGitHubUsername}/maven-project.git`

2. Enter the repo: `cd maven-project`

3. (Optional) Set your MULE_HOME env variable: `export
MULE_HOME={locationOfMuleInstall}`

4. Package and deploy: `mvn install`
```

Specify untracked files to ignore

12. Run git status in the command-line interface.

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)

  .classpath
  .project
  .settings/
  README.md
  activemq-data/
  catalog/
  derby.log
  mule-project.xml
  pom.xml
  src/
  target/
```

    Note: Many of these files should not be saved in your remote repository.

13. In your system explorer, navigate to the studentFiles for the course.
14. Locate the .gitignore file in the resources folder.
15. Make a copy of the file in $PROJECT_HOME.

16. Open .gitignore file in a text editor.

```
#-------------#
 # Maven Files #
 #-------------#
 target

 #---------------------------#
 # Anypoint Studio / Eclipse #
 #---------------------------#
 .studio
 .classpath
 .project
 .settings
 mule-project.xml
 .mule
 catalog

 #-----------#
 # Mac Files #
 #-----------#
 .DS_Store

 #------------------#
 # Derby / ActiveMQ #
 #------------------#
 activemq-data
 derby.log
```

17. Run git status in the command-line interface.

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)

  .gitignore
  README.md
  pom.xml
  src/
```

Stage the files (add them to the index)

18. Run git add –A.

    Note: This will stage all non-ignored files to be committed.

19. Run git status.

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
```

```
 new file:   .gitignore
 new file:   README.md
 new file:   pom.xml
 new file:   src/main/app/maven-project.xml
 new file:   src/main/app/mule-app.properties
 new file:   src/main/app/mule-deploy.properties
 new file:   src/main/java/com/mulesoft/training/Database.java
 new file:   src/main/resources/log4j2.xml
 new file:   src/test/java/com/mulesoft/training/HelloMavenTest.java
 new file:   src/test/resources/log4j2-test.xml
```

Commit code to the local repository

20. Run git commit -m "Initial commit".

```
[master (root-commit) 1c12f7d] initial commit
 10 files changed, 425 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 pom.xml
 create mode 100644 src/main/app/maven-project.xml
 create mode 100644 src/main/app/mule-app.properties
 create mode 100644 src/main/app/mule-deploy.properties
 create mode 100644 src/main/java/com/mulesoft/training/Database.java
 create mode 100644 src/main/resources/log4j2.xml
 create mode 100644 src/test/java/com/mulesoft/training/HelloMavenTest.java
 create mode 100644 src/test/resources/log4j2-test.xml
```

21. Return to your maven-project in GitHub at https://github.com/{yourGitHubUsername}/maven-project.
22. Refresh the page; you should not see any code yet.
23. Run git status in the command-line interface again.

```
On branch master
nothing to commit, working directory clean
```

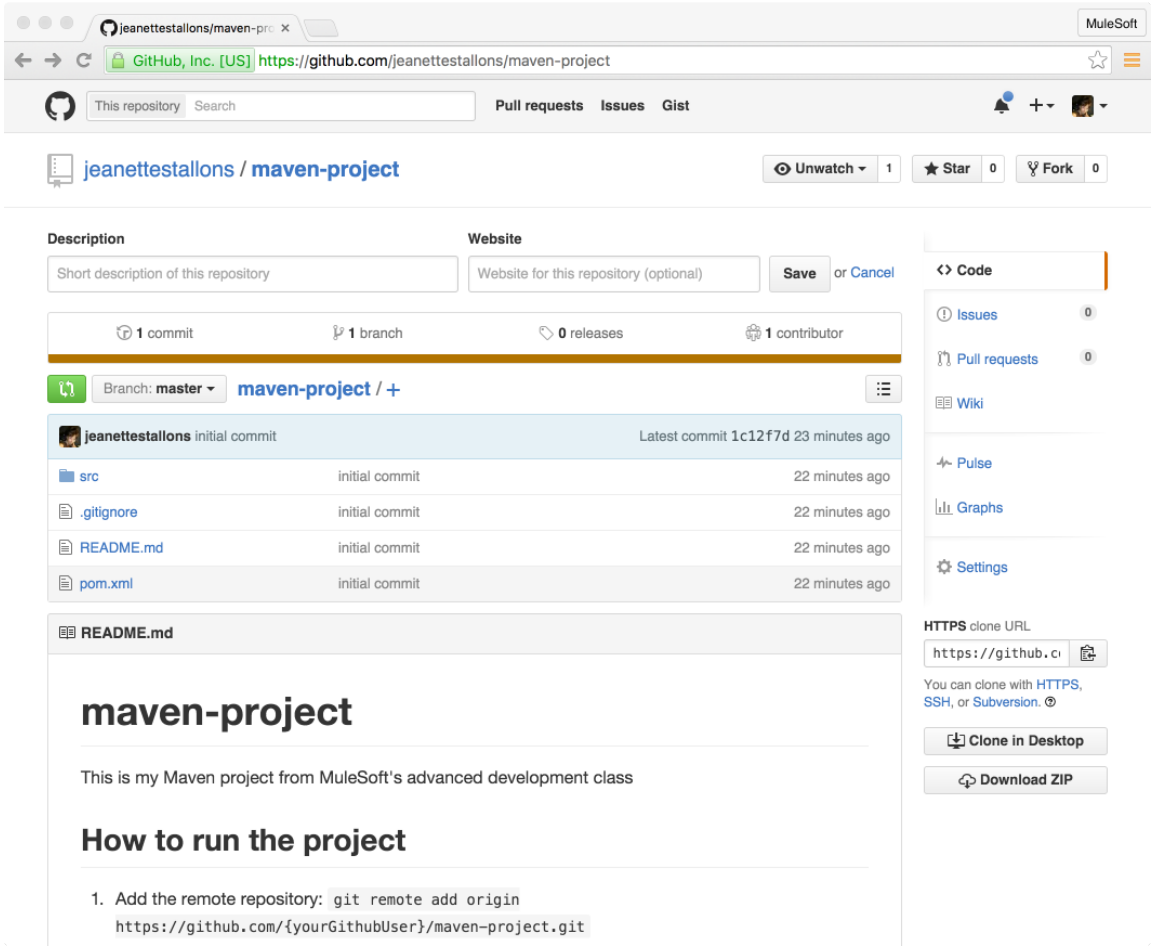
Push code to the remote repository

24. Run git push origin master.

```
Counting objects: 26, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (26/26), 5.50 KiB | 0 bytes/s, done.
Total 26 (delta 1), reused 0 (delta 0)
To https://github.com/yourUserName/maven-project.git
 * [new branch]      master -> master
```

25. Return to your maven-project in GitHub at https://github.com/{yourGitHubUsername}/maven-project.

26. Refresh the page; you should now see your project code.

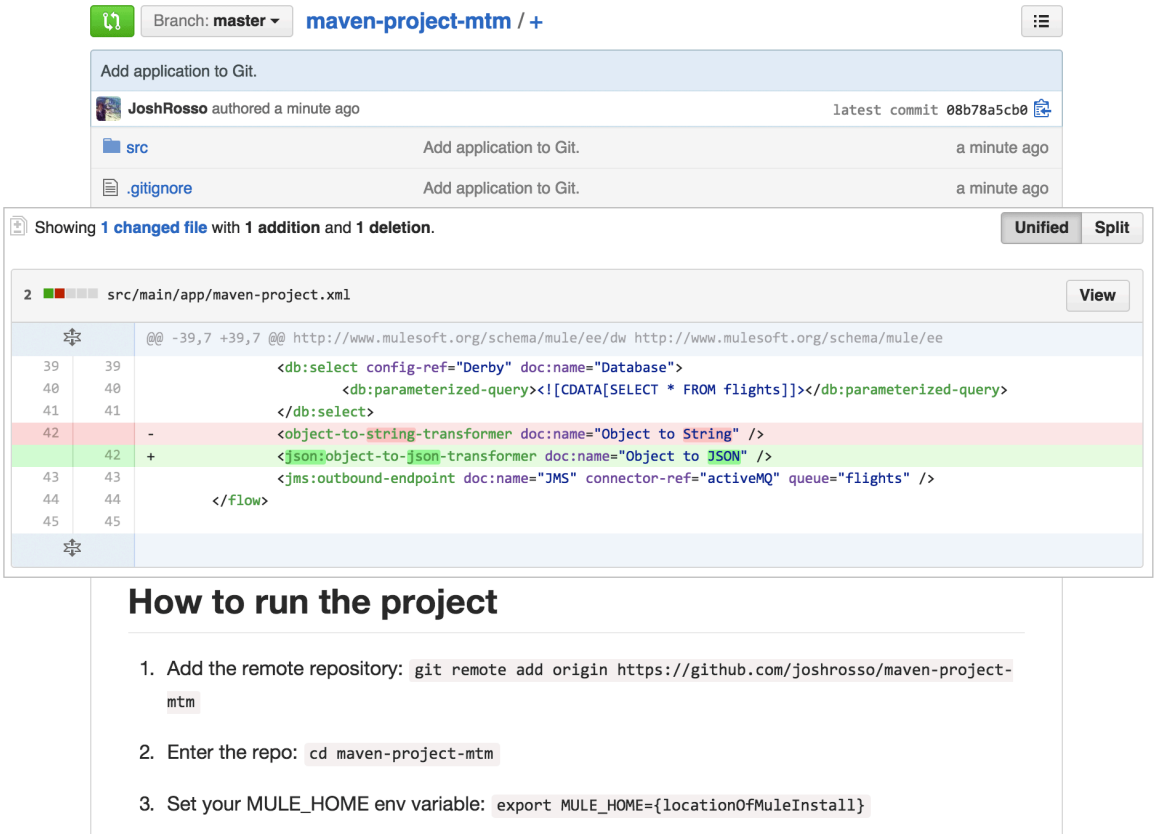


27. Browse the src directory.

Walkthrough 2-2: Modify code and commit changes

In this walkthrough, you make changes to Mule code and commit those changes to source control. You will:

- Add a code feature.
- Add, commit, and push the code to the remote repository.
- Review the code changes on GitHub.
- Modify the code format and explore the effect on this process.



Add a feature to the source code

1. Return to maven-project.xml in Anypoint Studio.

2. Replace the object-to-string-transformer with an object-to-json-transformer.

3. Save the XML file.
4. Return to pom.xml.
5. Locate the new mule-module-json artifact.

Commit the changes

6. Return to $PROJECT_HOME in the command-line interface.
7. Run git status.

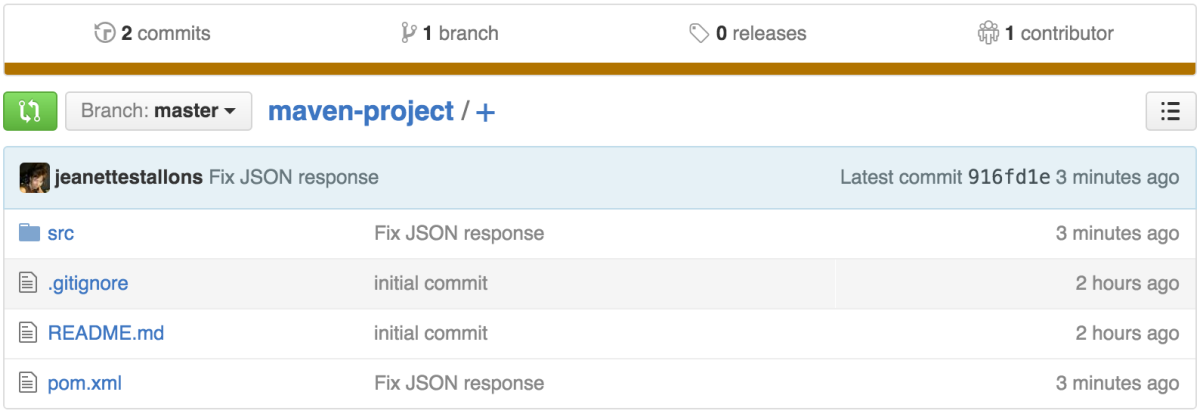
```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   pom.xml
  modified:   src/main/app/maven-project.xml
```

8. Run git add -A.
9. Run git status.
10. Commit the code by running git commit -m "Fix JSON response".
11. Run git push origin master.

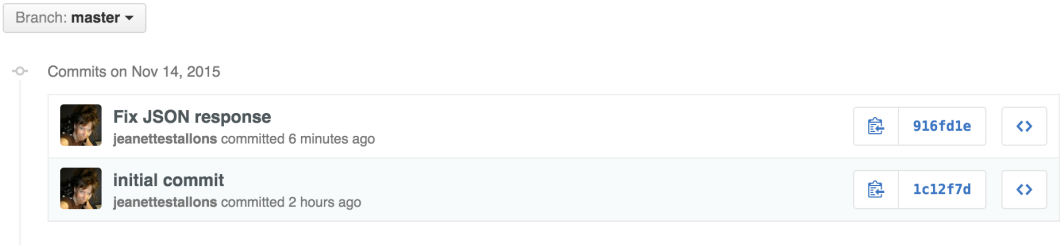
Review the changes

12. Return to the GitHub project page and if necessary, refresh the page.
13. Verify that the latest commit message is displayed next to the src folder and the pom.xml file.

14. Locate where the number of commits is listed at the top of the project.

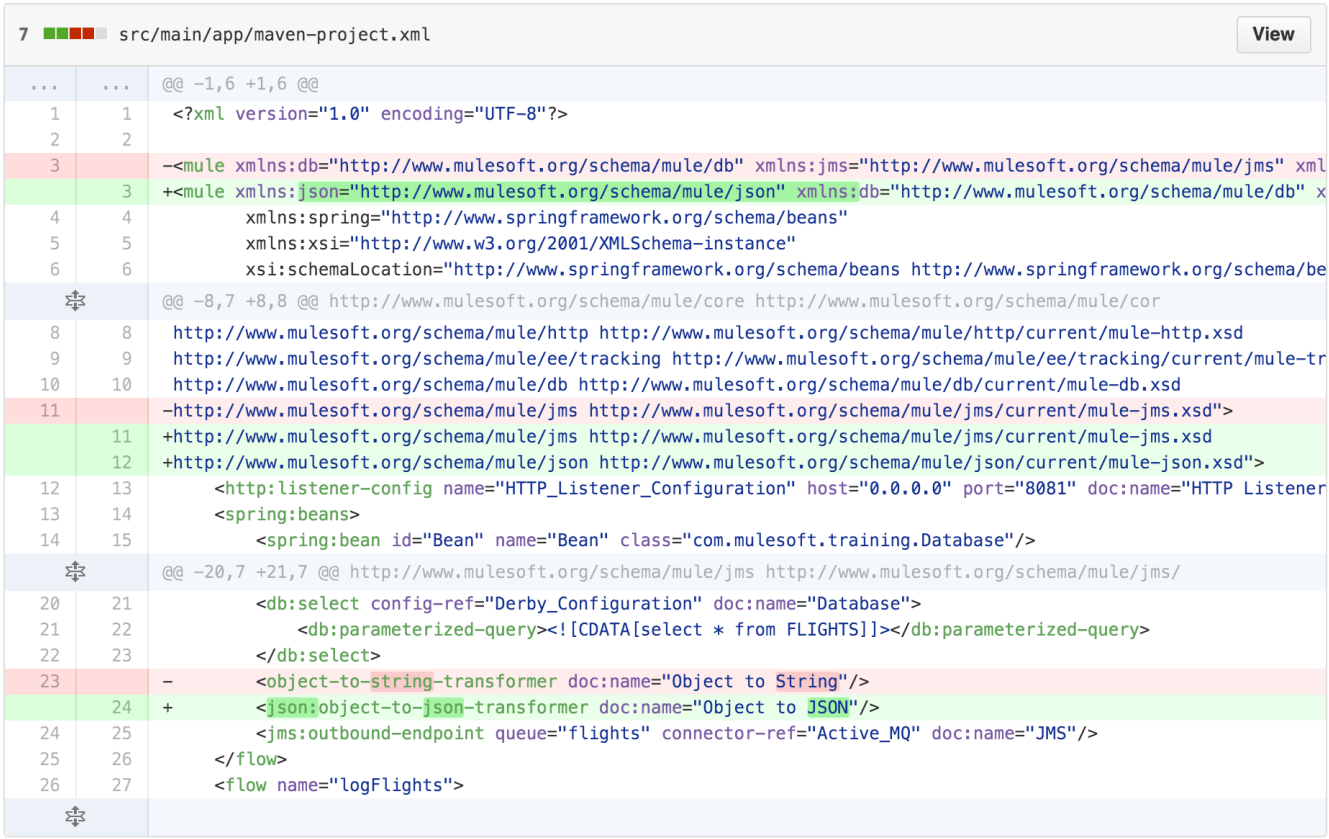


15. Click the 2 commits link.
16. Review the list of commits.



17. Click the Fix JSON response link.

18. Review the changes in the two files.



Add white space to the source code

19. Return to maven-project.xml in Anypoint Studio.
20. View the XML code.
21. Add a line break between the last global element and the first flow.

```
<db:derby-config name="Derby_Configuration"
url="jdbc:derby:memory:muleEmbeddedDB;create=true" doc:name="Derby Configuration"/>
    <jms:activemq-connector name="Active_MQ" brokerURL="vm://localhost"
validateConnections="true" doc:name="Active MQ"/>

    <flow name="retrieveFlights">
        <http:listener config-ref="HTTP_Listener_Configuration" path="/flights"
allowedMethods="GET" doc:name="HTTP"/>
```

22. Save the file.

Commit the changes

23. Return to $PROJECT_HOME in the command-line interface.

24. Run git status.

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   src/main/app/maven-project.xml
```

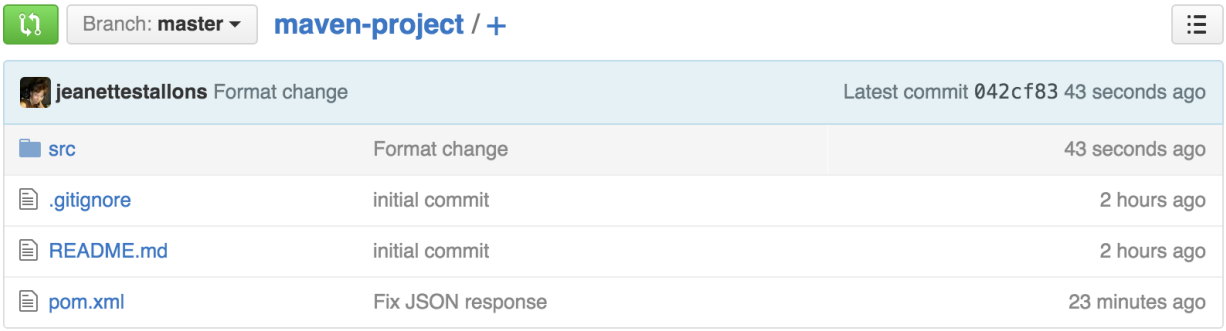
25. Add and commit your code with a message of Format change.

```
git add -A && git commit -m "Format change"
```

26. Run git push origin master.

Review the changes

27. Return to the home page for you GitHub project.
28. Verify that the latest commit message is displayed next to the src folder.
29. Locate where the number of commits is listed at the top of the project.



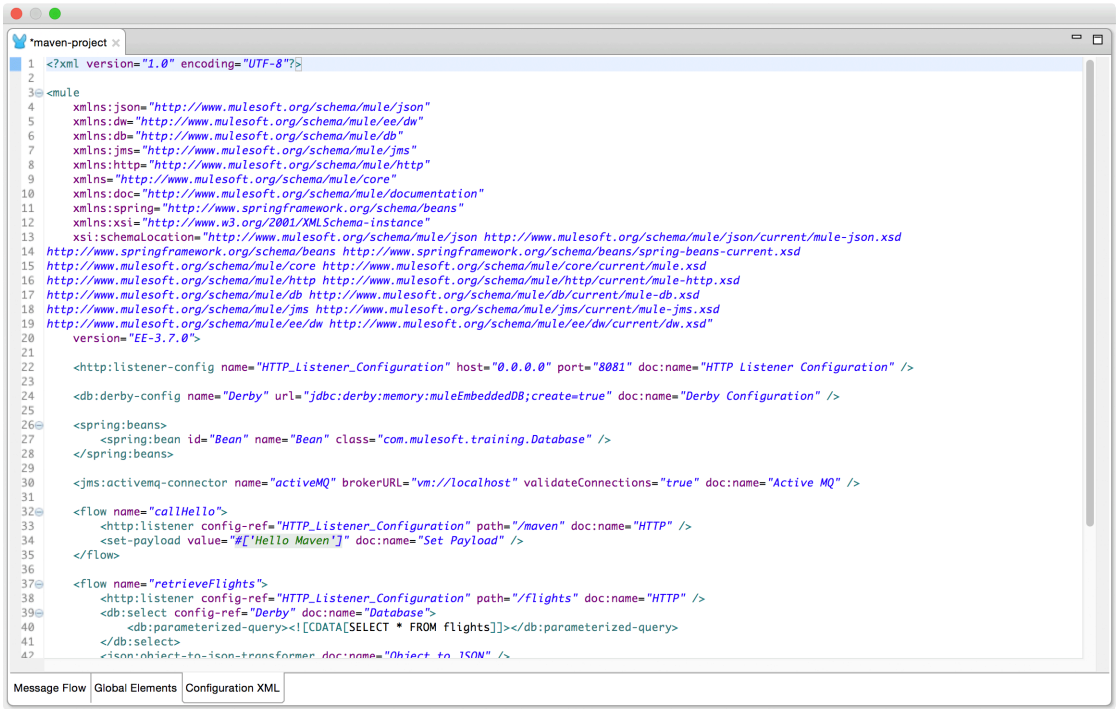
30. Click the 3 commits link.
31. Click the Format Change link.
32. Review the change in the file.

Note: Because all formatting changes will appear as diffs, this makes it more difficult for developers to quickly identify the code's purpose or changes when doing code review and checking diffs.

Walkthrough 2-3: Format Mule code

In this walkthrough, you format your code consistently so that it will be easier to identify the code's purpose or changes in source control. You will:

- Analyze code for formatting opportunities.
- Run Eclipse's XML formatter against Mule code.
- Prepare code to be pushed to source control.
- Push the code to source control.



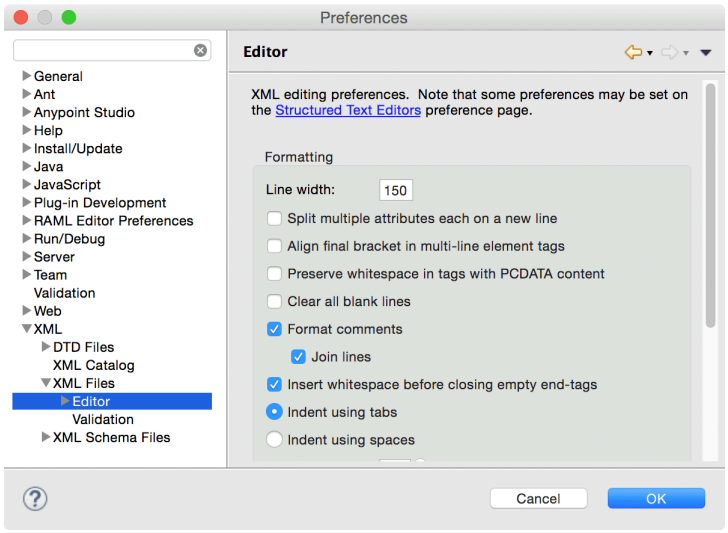
Review the source code

1. Return to the XML for maven-project.xml in Anypoint Studio.
2. Review the code formatting.
3. Open pom.xml and review the code formatting.

Configure Eclipse's XML formatting options

4. Open Anypoint Studio Preferences.
5. In the Preferences dialog box, expand XML.
6. Expand XML Files.
7. Select Editor.

8. Set the Line width to 150.



9. Click Apply.
10. Click OK.

Format the project's XML

11. Return to pom.xml.
12. From the main menu bar, select Source > Format.
13. Review the code changes.
14. Return to maven-project.xml.
15. Press Cmd+Shift+F or Ctrl+Shift+F.
16. Locate the global elements.
17. Add a line break after each global element.

```
<http:listener-config name="HTTP_Listener_Config" host="0.0.0.0" port="8081" />

<spring:beans>
  <spring:bean id="Bean" name="Bean" class="com.mulesoft.training.Database"/>
</spring:beans>

<db:derby-config name="Derby" url="jdbc:derby:memory:muleEmbeddedDB;create=true"/>

<jms:activemq-connector name="activeMQ" brokerURL="vm://localhost"
    validateConnections="true"/>
```

18. At the end of each flow, add a line break.

```
<flow name="mavenHello">
  <http:listener config-ref="HTTP_Listener_Configuration" path="/hello"/>
  <set-payload value="#['Hello Maven']"/>
</flow>

<flow name="retrieveFlights">
 <http:listener config-ref="HTTP_Listener_Configuration" path="/flights"/>
 <db:select config-ref="Derby">
 <db:parameterized-query><![CDATA[SELECT * FROM flights]]></db:parameterized-query>
 </db:select>
 <object-to-json-transformer/>
 <jms:outbound-endpoint connector-ref="Active_MQ" queue="flights"/>
</flow>

<flow name="logFlights">
  <jms:inbound-endpoint queue="flights" connector-ref="Active_MQ"/>
  <logger message="#[message.payload]" level="INFO"/>
</flow>
```

19. Locate the XML namespaces at the top of the file.
20. Before each xmlns, add a line break.
21. Move the default namespace to the top.

```
<mule
xmlns="http://www.mulesoft.org/schema/mule/core
xmlns:json="http://www.mulesoft.org/schema/mule/json"
xmlns:db="http://www.mulesoft.org/schema/mule/db"
xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
xmlns:http="http://www.mulesoft.org/schema/mule/http"
xmlns:tracking="http://www.mulesoft.org/schema/mule/ee/tracking"     "
xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
xmlns:spring="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/json
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-current.xsd
```

Note: This lets you easily identify the modules/libraries your project is using.

22. Locate the xmlns:tracking namespace.
23. Examine the rest of the XML to see if this module is used; you should not see any references.
24. Remove the xmlns:tracking namespace and its respective xsi:schemaLocation shown below.

```
xmlns:tracking=http://www.mulesoft.org/schema/mule/ee/tracking

xsi:schemaLocation="…
http://www.mulesoft.org/schema/mule/ee/tracking
http://www.mulesoft.org/schema/mule/ee/tracking/current/mule-tracking-ee.xsd
```

25. Save all the files.

Commit the changes

26. Return to $PROJECT_HOME in the command-line interface.
27. Run git status.
28. Add and commit your code with a message of XML formatted.

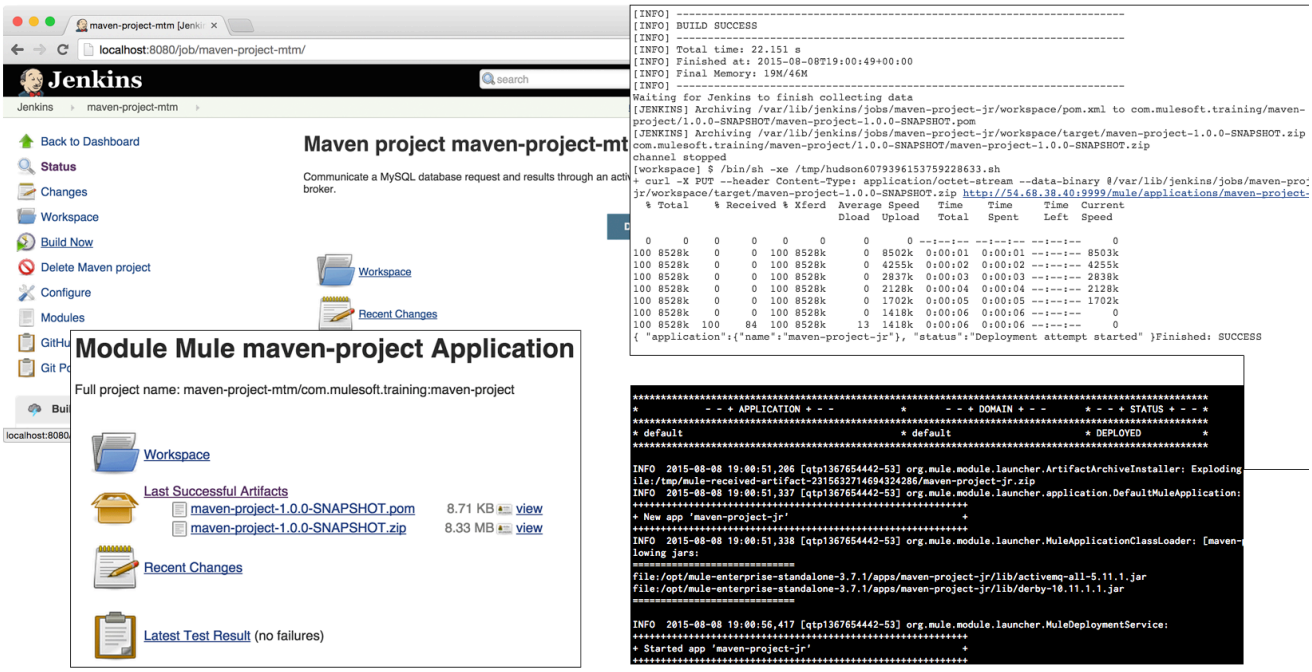
```
git add –A && git commit -m "XML formatted"
```

29. Run git push origin master.

Review the changes

30. Return to the home page for you GitHub project.
31. Verify that the latest commit message is displayed next to the src folder.
32. Locate where the number of commits is listed at the top of the project.
33. Click the 4 commits link.
34. Click the XML formatted link.
35. Review the changes in the file.

Module 3: Achieving Continuous Integration



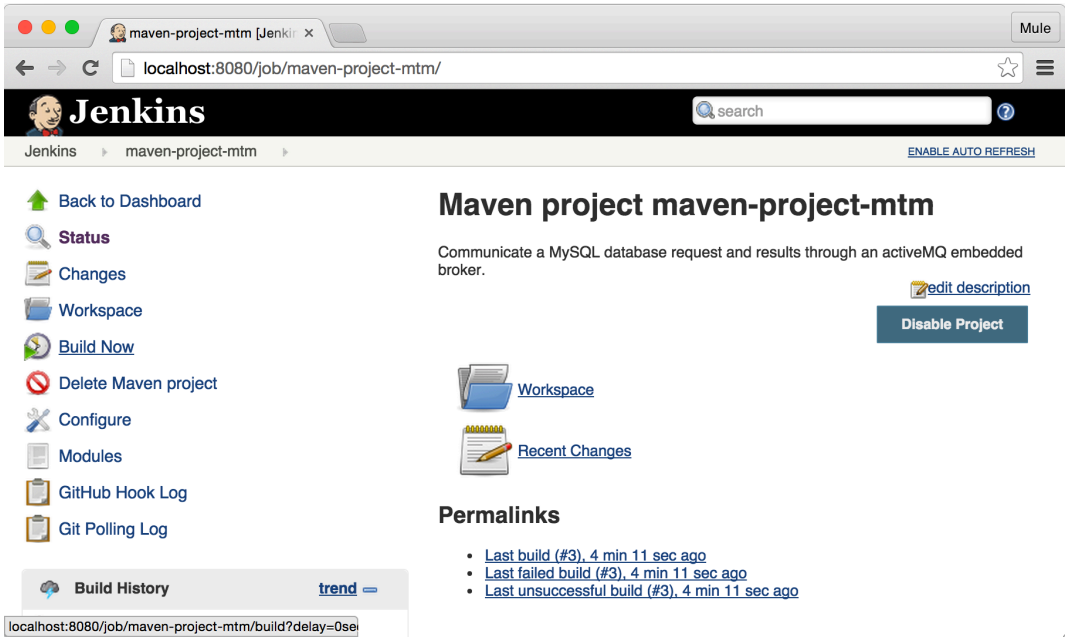
Objectives:

- Listen for new commits on your project's SCM.
- Pull project changes into a centralized server.
- Create build jobs on a centralized server.
- Run unit and integration tests during the build of a project.
- Access results of each build.

Walkthrough 3-1: Create a CI job (Jenkins)

In this walkthrough, you create a continuously integrated Mule project. You will:

- Create a new CI job on a Jenkins server.
- Point the CI server to your SCM (Git).
- Automatically build and generate artifacts for the project.



Log in to Jenkins

1. In a web browser, navigate to the Jenkins URL at http://adv.mulesoft-training.com.

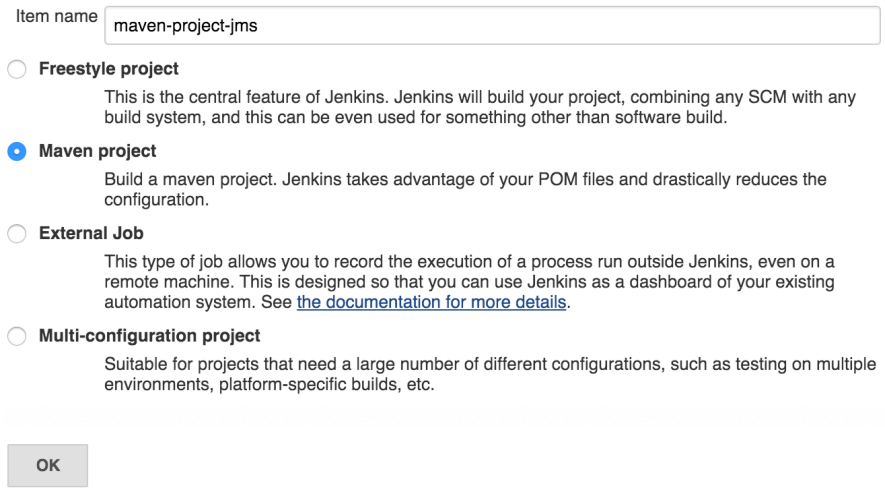
   Note: This URL is also located in the course snippets.txt file.

2. Log in to Jenkins using the credentials located in the course snippets.txt file.

Create a new Jenkins build item

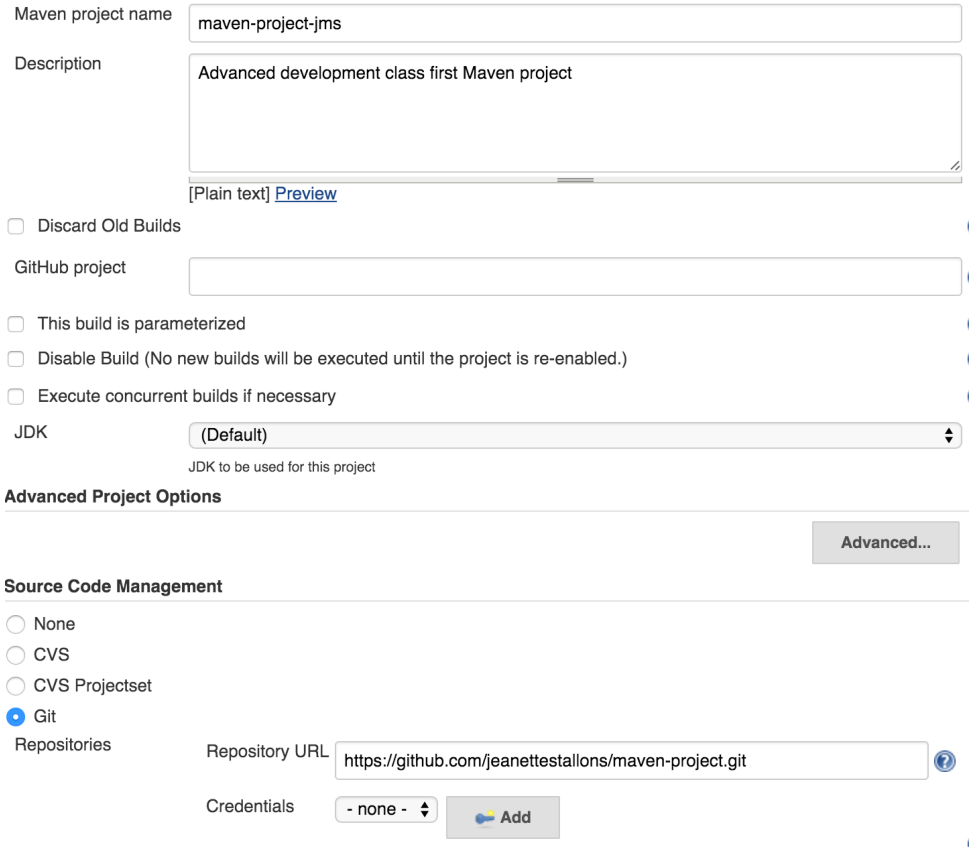
3. In Jenkins, click New Item in the left-side navigation.
4. Set the Item name to maven-project-{yourInitials}.

5. Select the Maven project radio button.



6. Click OK.
7. Add a description based on the project's functionality.
8. Under Source Control Management, select Git.
9. Set the Repository URL to your GitHub repository.

```
https://github.com/{yourGitHubUsername}/maven-project.git
```



10. Locate the branch specifier and examine its default value.

    Note: This setting ensures that only changes to the master branch are used for this job.

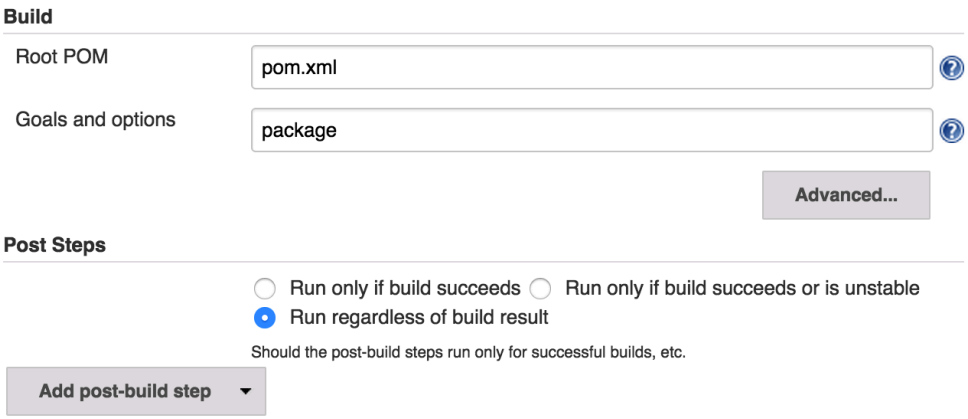
11. Under Build Triggers, check Poll SCM.
12. Set the Schedule to an expression to build every 2 minutes.

```
H/2 * * * *
```

Note: You can also select Build when a change is pushed to GitHub, but it will not be functional. In order for this to work, you must also grant GitHub access to your Jenkins instance and then add webhooks to GitHub.

13. Under Build, set the following goals and options.

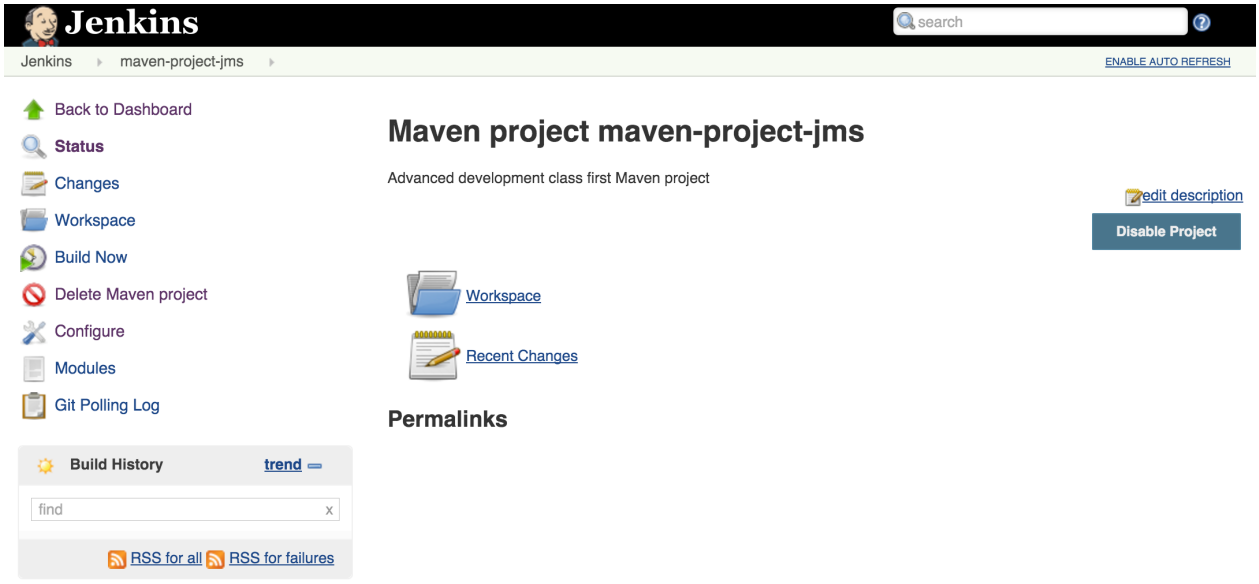
```
package
```



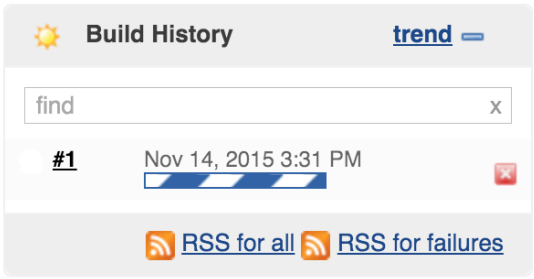
14. Click Save.

Run the first build

15. On the project page, locate the build history section.

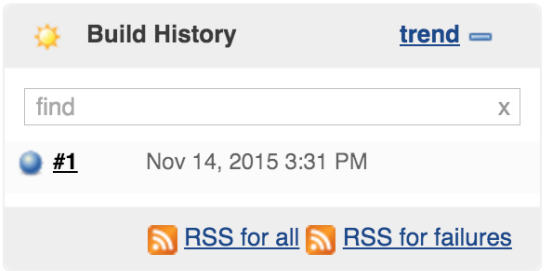


16. Wait until Jenkins runs the first build; you should see a build listed in the build history.



Note: If you are impatient, click the Build Now link in the left-side navigation.

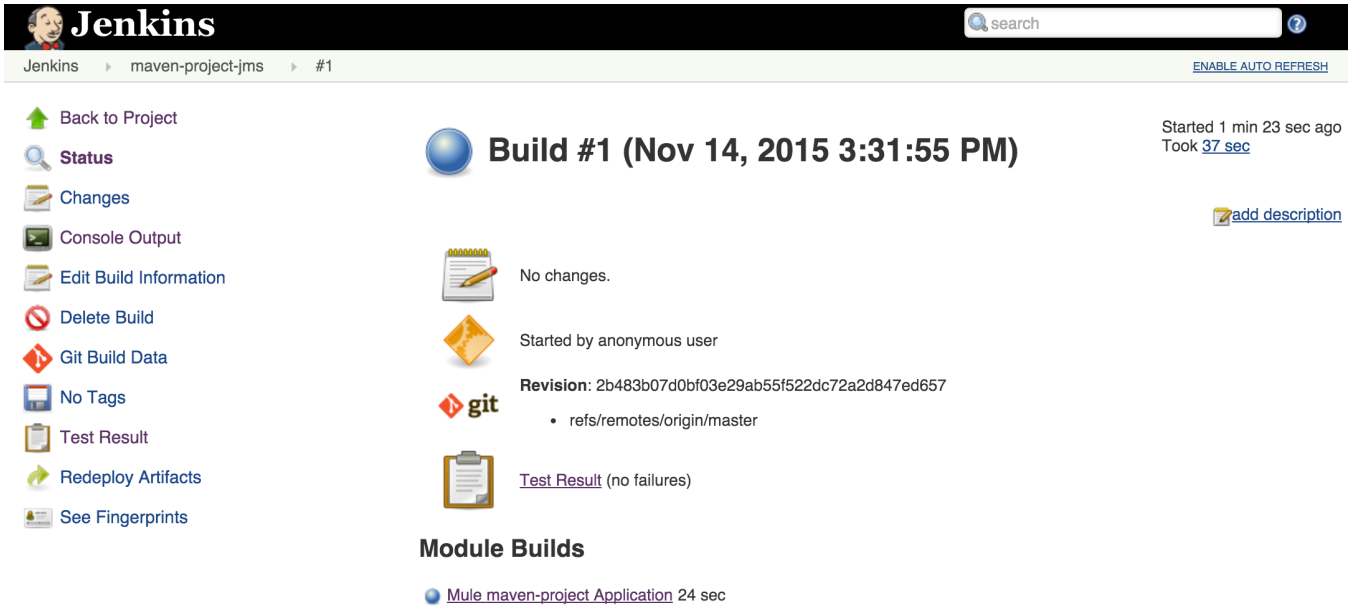
17. Wait until the build finishes; you should see a blue circle next to it in the build history.



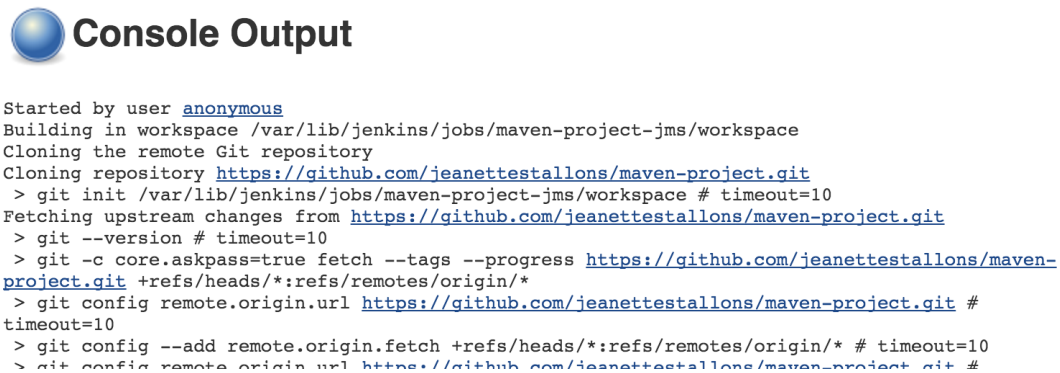
Examine the build results

18. In the build history, click build #1.

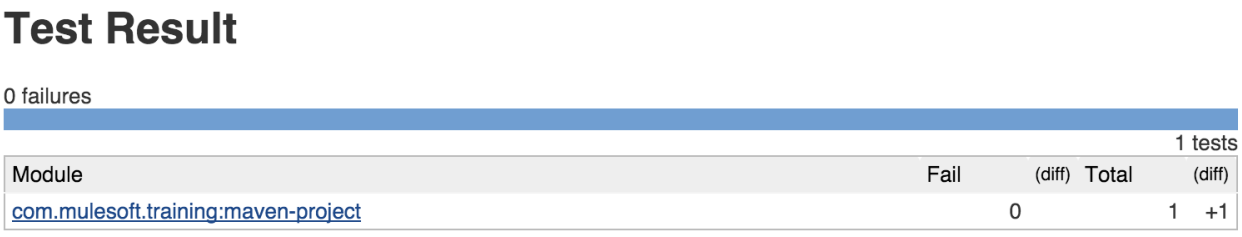
19. Examine the build information.



20. In the left-side navigation, click Console Output.
21. Examine the console output.

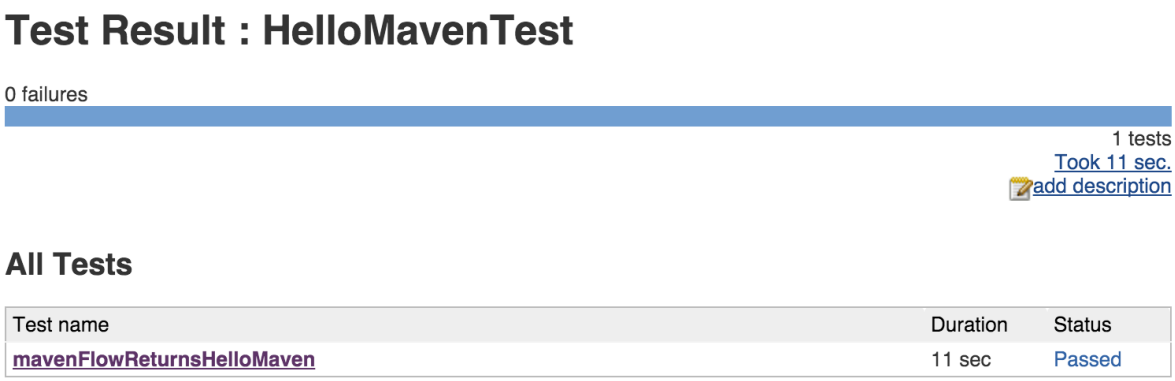


22. In the left-side navigation, click Test Result.
23. Examine the test results; you should get one test and zero failures.



24. Click com.mulesoft.training:maven-project.
25. Click com.mulesoft.training.

26. Click HelloMavenTest.
27. Examine the results for the mavenFlowReturns HelloMaven test.

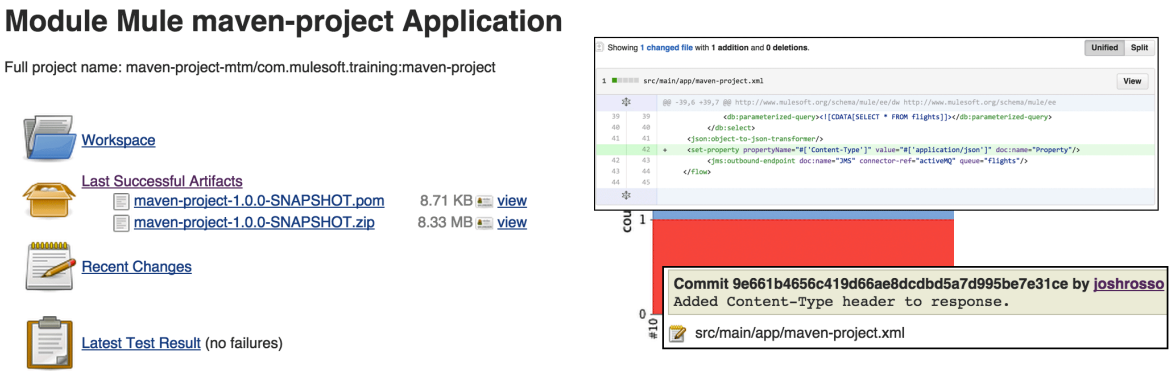


28. In the left-side navigation, click Back to Project.

Walkthrough 3-2: Trigger a CI build by modifying source code

In this walkthrough, you will:

- Modify the source code by adding a failing test.
- Commit and push the code to SCM.
- Examine the build results for this new source code on the CI server.



Modify source code by adding a failing test

1. Return to Anypoint Studio.
2. Open src/test/java/com/mulesoft/training/HelloMavenTest.java.
3. Open the course snippets.txt file and copy the retrieveFlightsAddsAppropriateHeader() test method.

```
@Test
 public void retrieveFlightsAddsAppropriateHeader() throws Exception {
   MuleEvent event = runFlow("retrieveFlights");
   String contentType = event.getMessage().getOutboundProperty("Content-Type");
   assertEquals("application/json", contentType);
 }
```

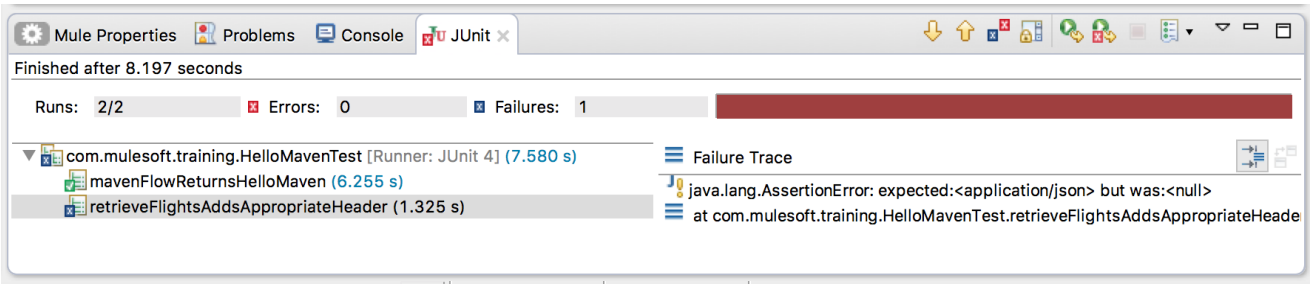
Note: This test method verifies that the flow sets a Content-Type outbound property to application/json.

4. Return to HelloMavenTest.java and paste the code inside the class definition.
5. Import org.mule.api.MuleEvent.

Run the test

6. Right-click in the file and select Run As > JUnit Test.

7. Verify the retrieveFlightsAddsAppropriateHeader test fails.



Build the project

8. Return to $PROJECT_HOME in a command-line interface.
9. Run git status.
10. Run mvn clean package and verify that the build fails.

```
Failed tests:
  HelloMavenTest.retrieveFlightsAddsAppropriateHeader:25 expected:<application/json>
but was:<null>

Tests run: 2, Failures: 1, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.990 s
[INFO] Finished at: 2015-11-14T08:04:16-08:00
[INFO] Final Memory: 42M/376M
[INFO] ------------------------------------------------------------------------
```

Commit the changes

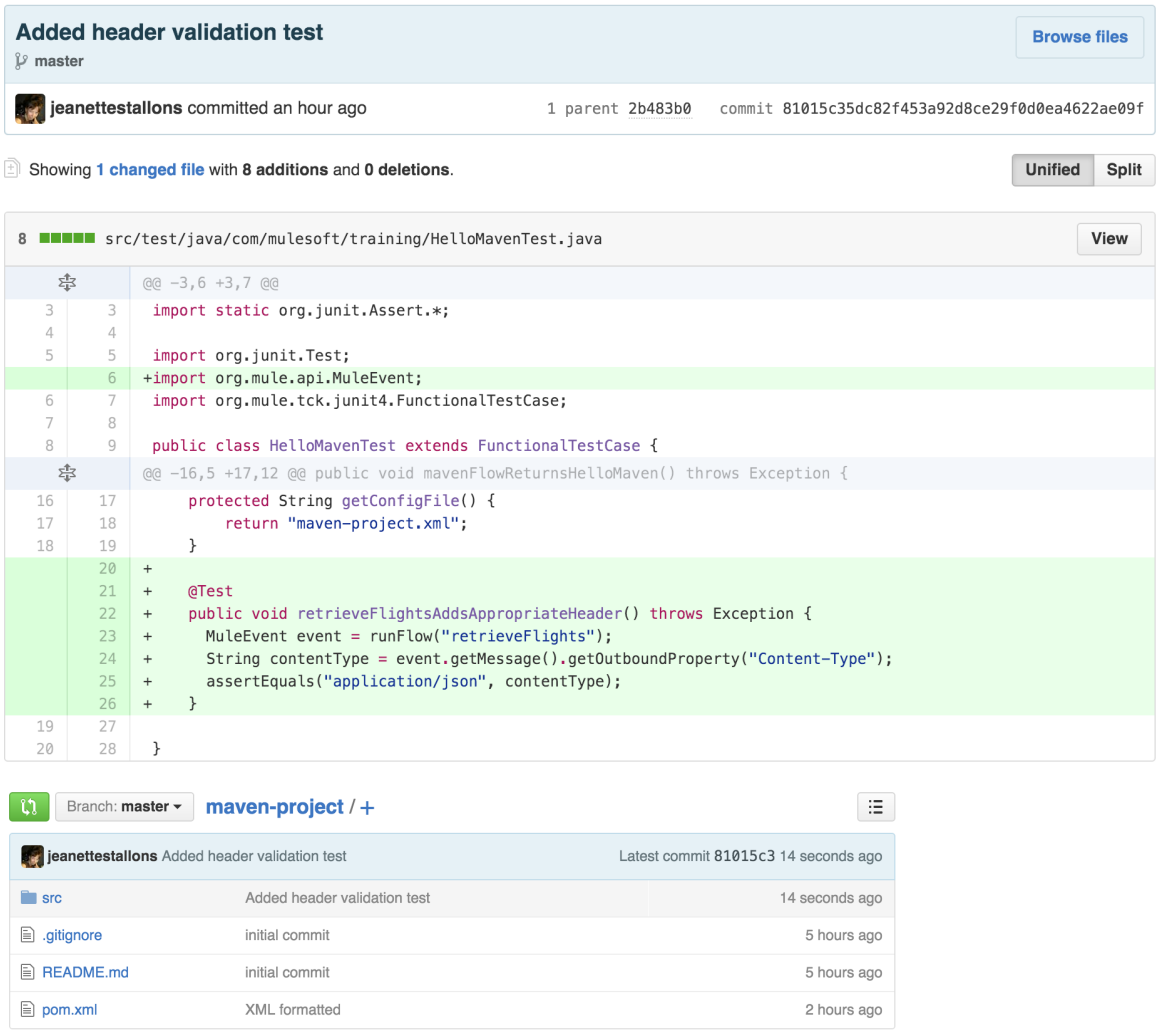
11. Add, commit, and push your changes.

```
git add -A && git commit -m "Added header validation test" && git push origin master
```

Note: In general, you would not commit failing code to a remote repository (and it is not recommended). You are going to commit it in this exercise, however, so you can see what happens on a CI server when a build of code from a SCM fails.

12. In a browser, return to your maven-project GitHub repository.
13. Navigate to the list of changes for the Added header validation test commit.

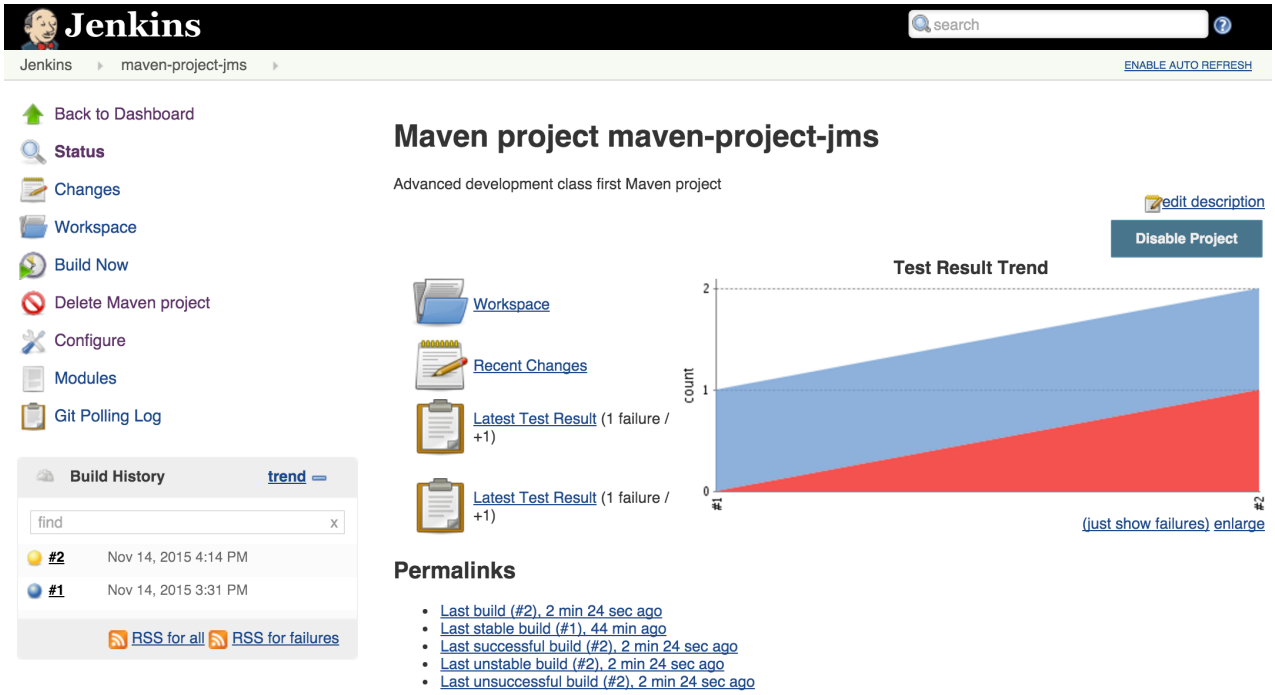
14. Locate the commit ID.



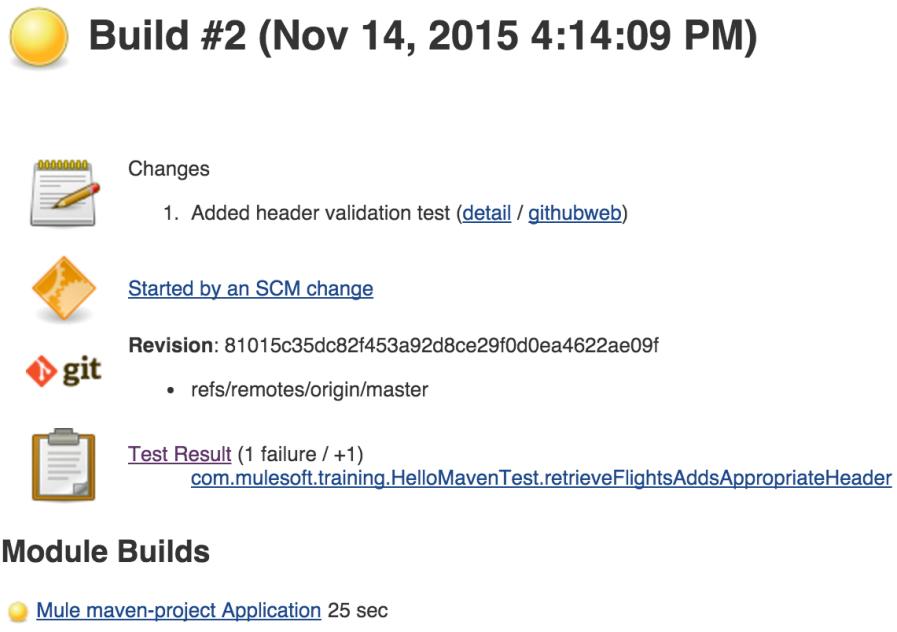
Examine the build results on Jenkins

15. Return to Jenkins.
16. Go to the maven-project-{yourInitials} project page; you should see that a second build was created because there were changes to the source code.

17. Examine the project information showing that the second build failed.



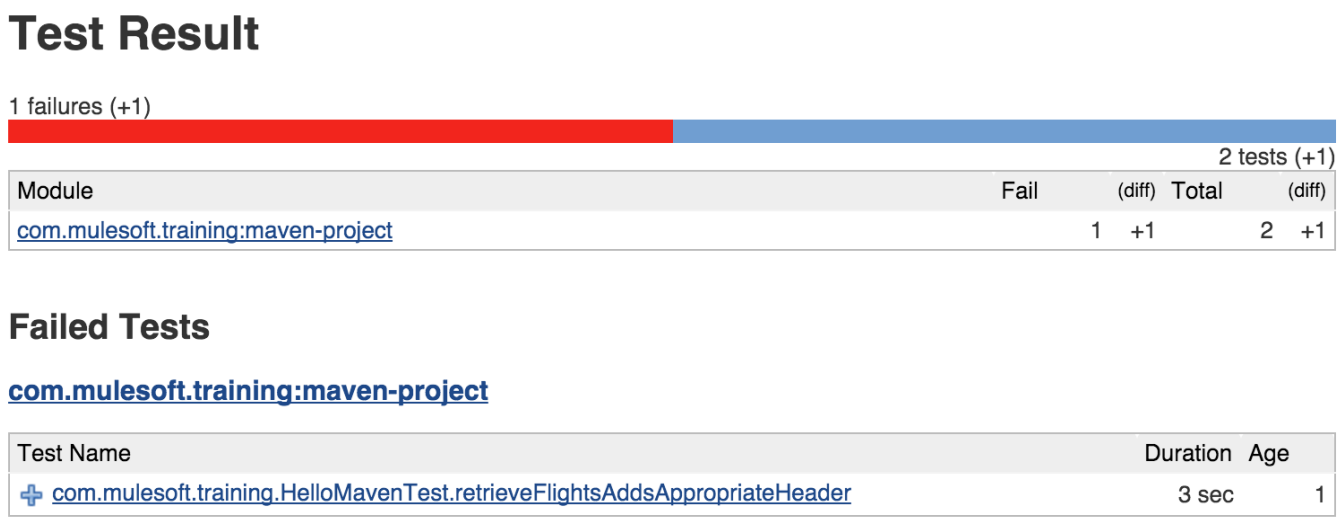
18. In the build history, click build #2.
19. Examine the build information.



20. In the left-side navigation, click Console Output.
21. Examine the console output and locate the cause of the failure.

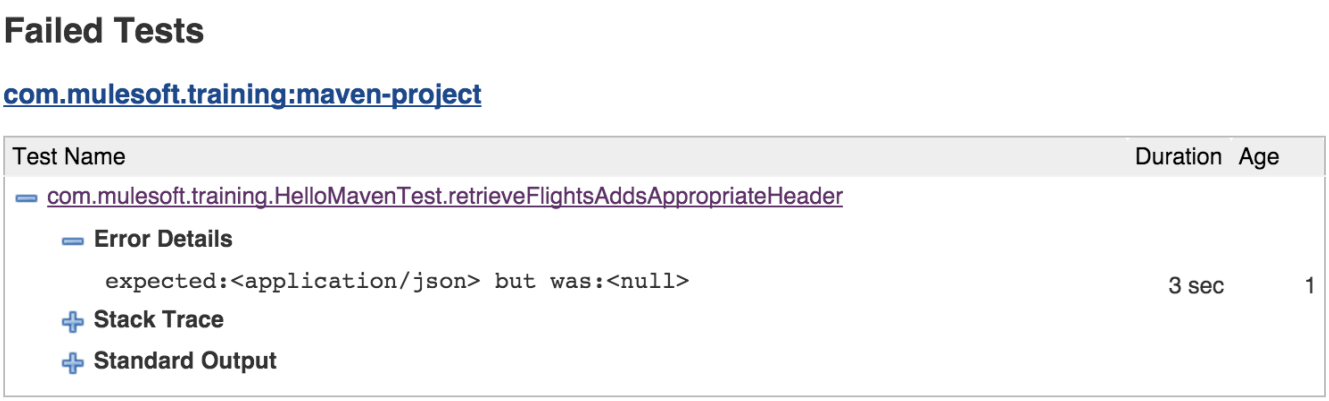
```
Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 14.343 sec
<<< FAILURE! - in com.mulesoft.training.HelloMavenTest
retrieveFlightsAddsAppropriateHeader(com.mulesoft.training.HelloMavenTest)  Time
elapsed: 3.008 sec  <<< FAILURE!
java.lang.AssertionError: expected:<application/json> but was:<null>
  at
com.mulesoft.training.HelloMavenTest.retrieveFlightsAddsAppropriateHeader(HelloMavenT
est.java:25)


Results :

Failed tests:
  HelloMavenTest.retrieveFlightsAddsAppropriateHeader:25 expected:<application/json>
but was:<null>

Tests run: 2, Failures: 1, Errors: 0, Skipped: 0

[ERROR] There are test failures.
```

22. In the left-side navigation, click Test Result.
23. Examine the test results; you should two tests and one failure.

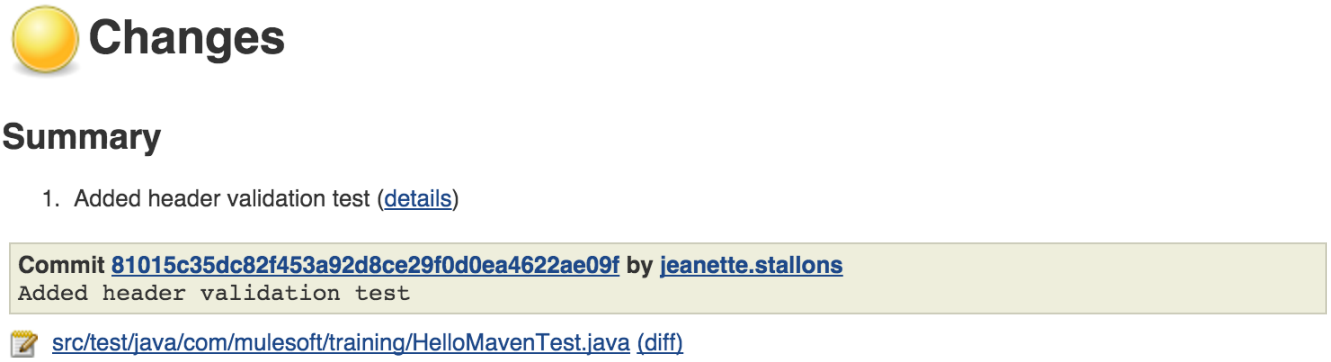


24. Under Failed Tests, click the plus sign next to retrieveFlightsAddsAppropriateHeader.

25. Examine the test details.



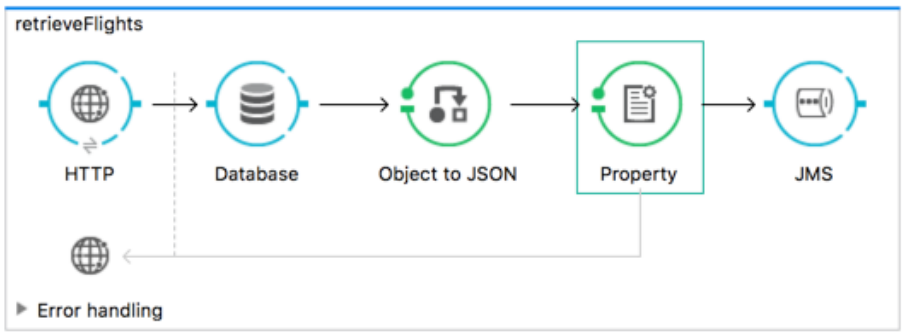
26. Expand Stack Trace.
27. Expand Standard Output.
28. In the left-side navigation, click Changes.
29. Locate the commit ID for the latest SCM changes.



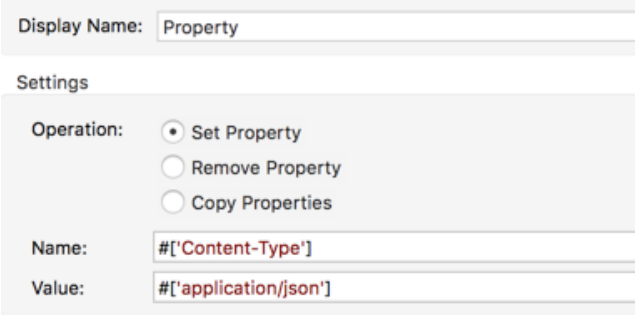
30. In the left-side navigation, click Back to Project.

Modify the source code so it passes the test

31. Return to maven-project.xml in Anypoint Studio.
32. In the retrieveFlights flow, add a set-property transformer after the json:object-to-json transformer.

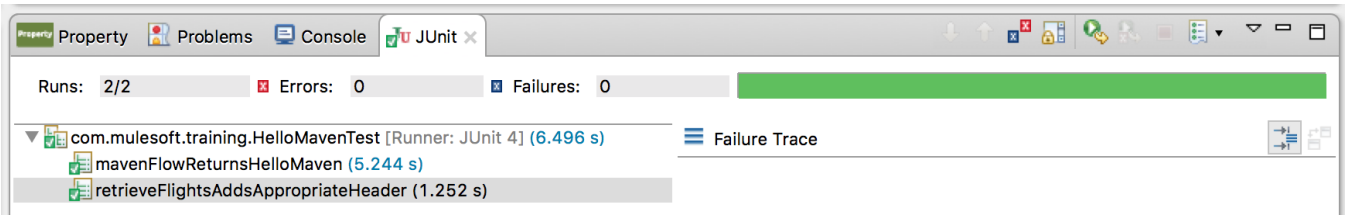


33. Set the name to Content-Type and the value to application/json.



Run the tests

34. Right-click the project in the Package Explorer and select Run As > JUnit Test.
35. Verify the retrieveFlightsAddsAppropriateHeader test now passes.



Note: If the retrieveFlightsAddsAppropriateHeader test still passes, try running the project and then testing again. At times, the workspace can get out of sync with the cached test.

Build the project

36. Return to $PROJECT_HOME in a command-line interface.

    Run mvn clean package and verify that the build succeeds.

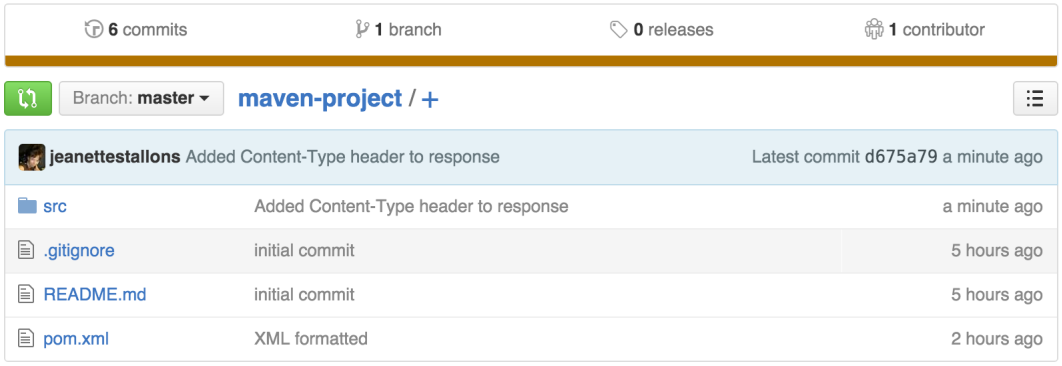
```
Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- mule-app-maven-plugin:1.1:mule (default-mule) @ maven-project ---
[INFO] Copying classes directly
[INFO] Adding <org.apache.activemq:activemq-all:jar:5.11.3> as a lib
[INFO] Copying mappings
[INFO] Building zip: /Users/mule/AnypointStudio/Documents/workspace/maven-
project/target/maven-project-1.0.0-SNAPSHOT.zip
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.936 s
[INFO] Finished at: 2015-11-14T08:44:43-08:00
[INFO] Final Memory: 42M/376M
[INFO] ------------------------------------------------------------------------
```

Commit the changes

37. Run git status.
38. Add, commit, and push your changes.

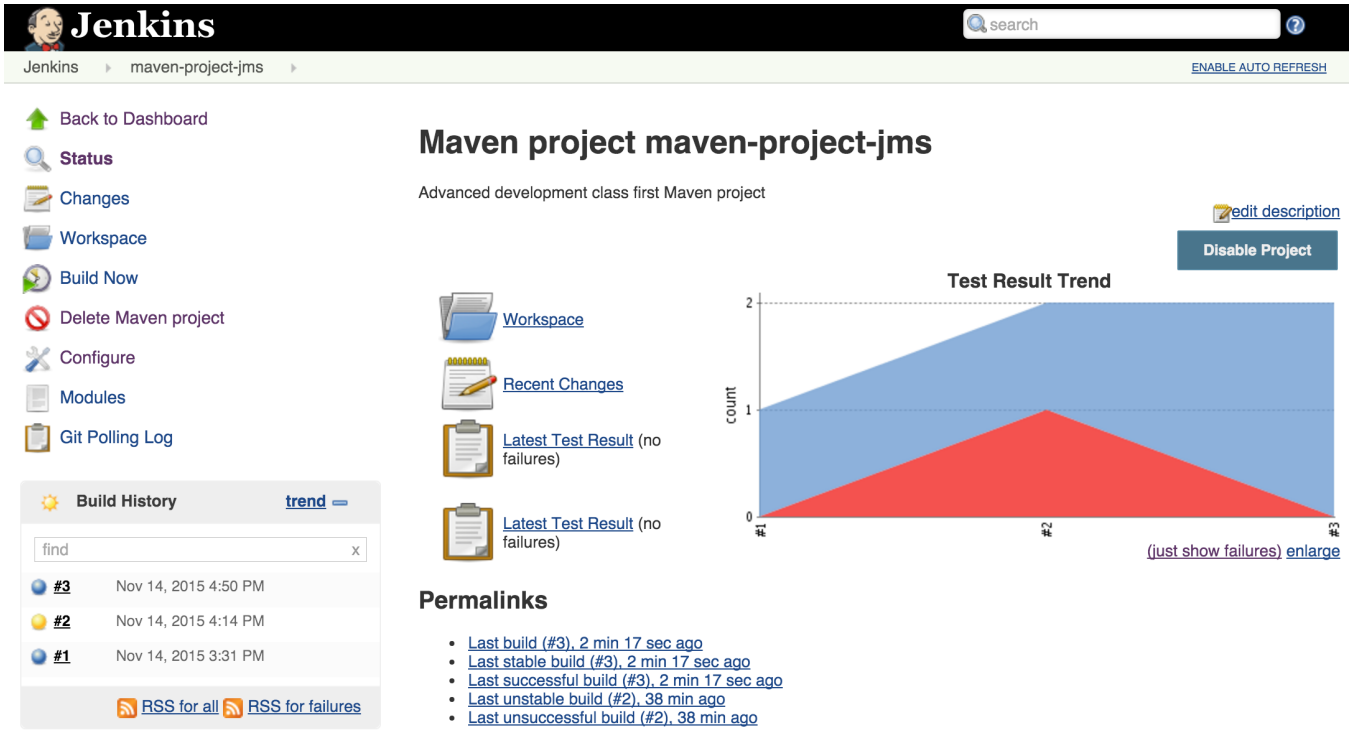
```
git add –A && git commit -m "Added Content-Type header to response " && git push origin master
```

39. In a web browser, return to your maven-project GitHub repository and verify the commit.

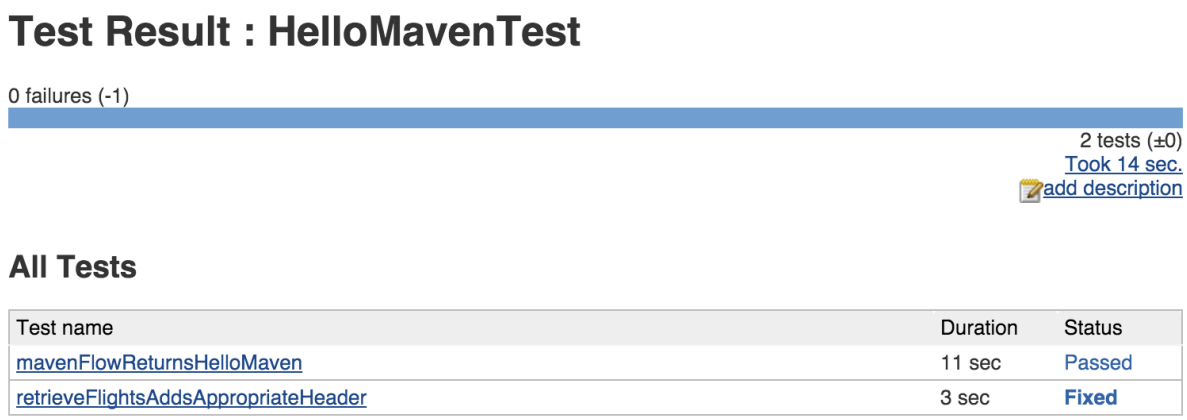


Validate the CI build

40. Return to Jenkins.
41. Go to the maven-project-{yourInitials} project page; you should see that a third build was created that passed all tests.

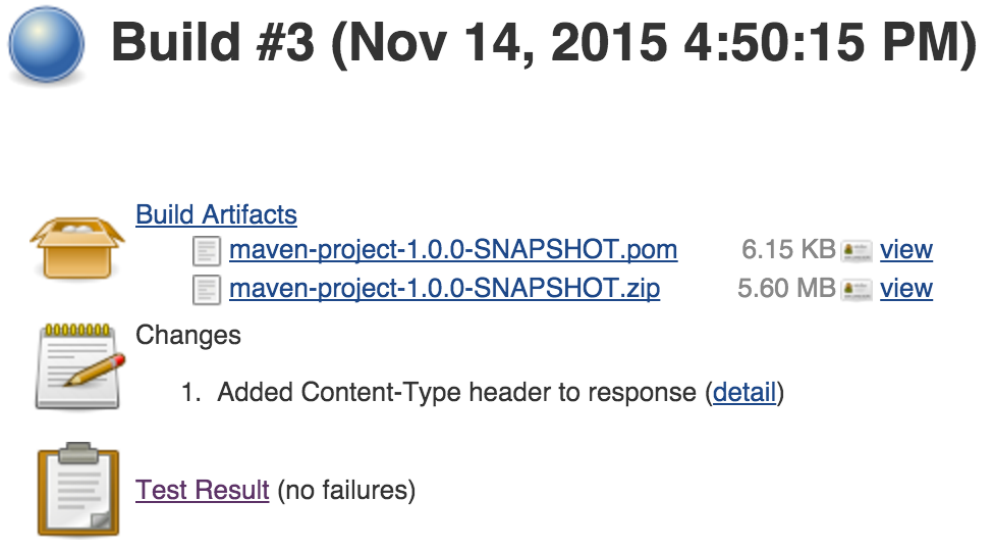


42. In the build history, click build #3.
43. In the left-side navigation, click Test Result.
44. Drill-down and examine the test details.



Get a deployable archive from the CI server

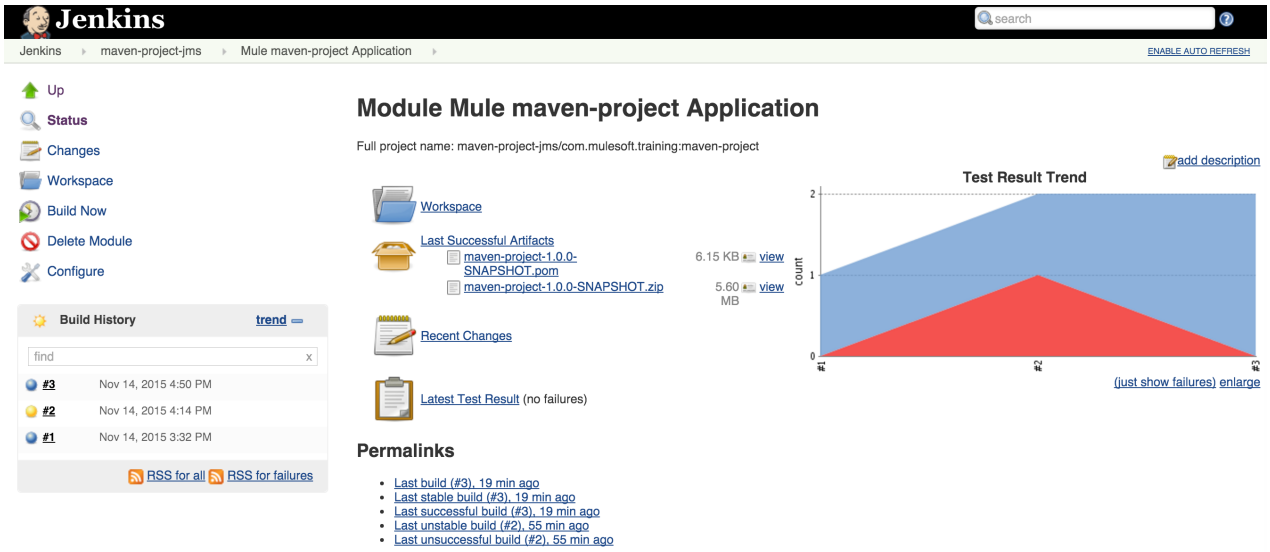
45. In the left-side navigation, click Status.
46. Examine the available build artifacts.



47. In the left-side navigation, click Back to Project.

48. Notice that the latest successful artifacts are also listed here.

Note: This ZIP is a Mule deployable archive that can be deployed to a server.



Walkthrough 3-3: Automate application deployment

In this walkthrough, you will:

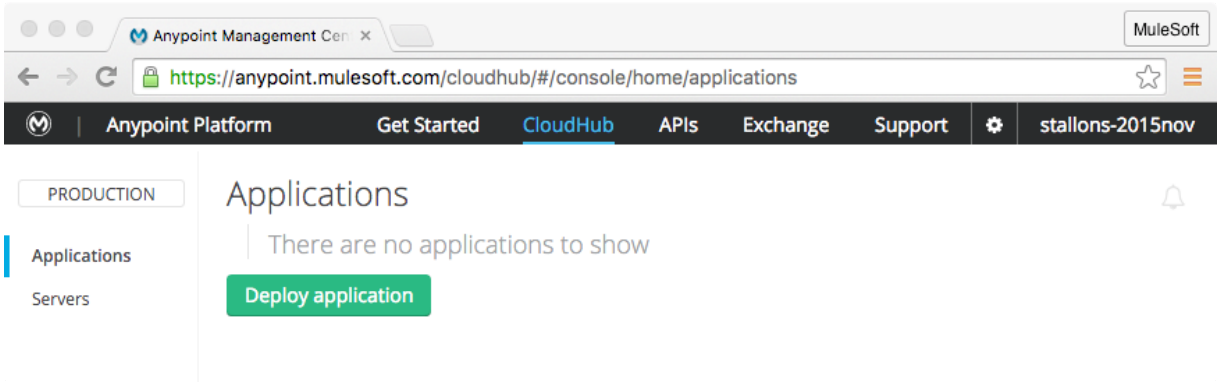
- Configure a Mule project to work with the mule-maven-plugin.
- Configure a CI server to deploy a Mule application if the build process succeeds.

Check for available application name on Anypoint Platform

1. In a web browser, navigate to http://anypoint.mulesoft.com.
2. Log in to Anypoint Platform.

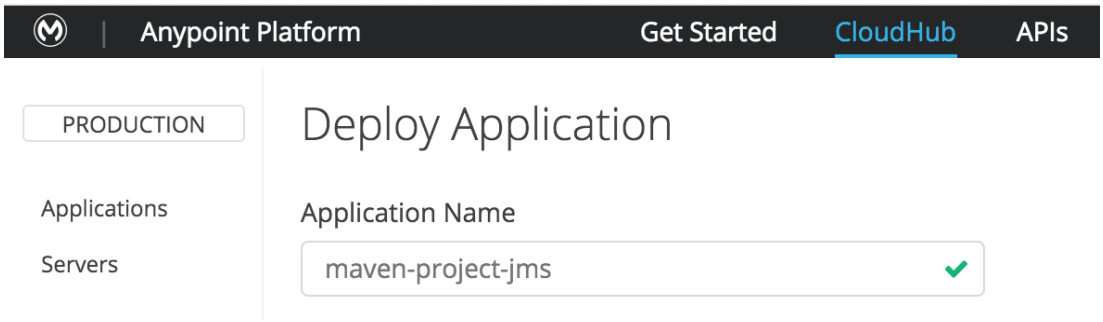
   Note: If you do not have an Anypoint Platform account, create a 30-day trial account now.

3. Click CloudHub.
4. Click Deploy application.



5. For the application name, enter maven-project-{yourInitials}.

6. See if a green check appears next to the name indicating that it is available.



7. If the name is not available, enter different values until you find one that is.
8. Click CloudHub to cancel the deployment and return to the application list.

Add deployment settings to the POM

9. Return to pom.xml in Anypoint Studio.
10. Locate the artifactId and change the value to maven-project-{yourInitials} or whatever application name you found available on CloudHub.

```
<modelVersion>4.0.0</modelVersion>
 <groupId>com.mulesoft.training</groupId>
 <artifactId>maven-project-jms</artifactId>
 <version>1.0.0-SNAPSHOT</version>
 <packaging>mule</packaging>
 <name>Mule maven-project-jms Application</name>
```

Note: This artifactId will be used as the name of the application when deployed.

11. Review the plugins section.
12. Return to the course snippets.txt file and copy the plugin code.
13. Return to pom.xml and paste it into the <plugins> element.

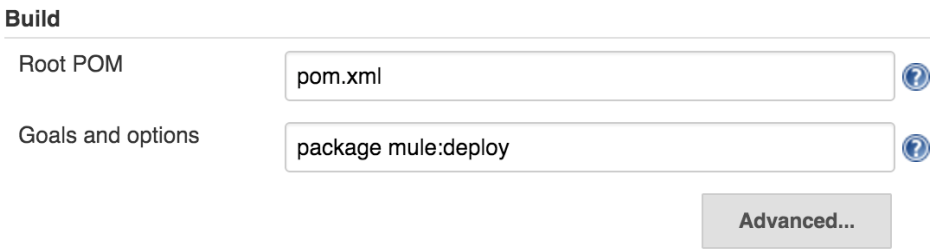
```
<plugin>
     <groupId>org.mule.tools.maven</groupId>
     <artifactId>mule-maven-plugin</artifactId>
     <version>2.0</version>
     <configuration>
       <deploymentType>cloudhub</deploymentType>
       <muleVersion>3.7.2</muleVersion>
       <username>yourUsername</username>
       <password>yourPassword</password>
       <workerType>Micro</workerType>
       <redeploy>true</redeploy>
       <environment>Production</environment>
     </configuration>
     <executions>
       <execution>
         <id>deploy</id>
         <phase>deploy</phase>
```

```
        <goals>
          <goal>deploy</goal>
        </goals>
      </execution>
    </executions>
</plugin>
```

14. Replace yourUsername and yourPassword with your Anypoint Platform credentials.

Configure CI server to trigger deployment

15. Return to Jenkins.
16. Go to the maven-project-{yourInitials} project page.
17. From the left-side navigation, click Configure.
18. In the Build section, locate Goals and options.
19. Add mule:deploy after package.



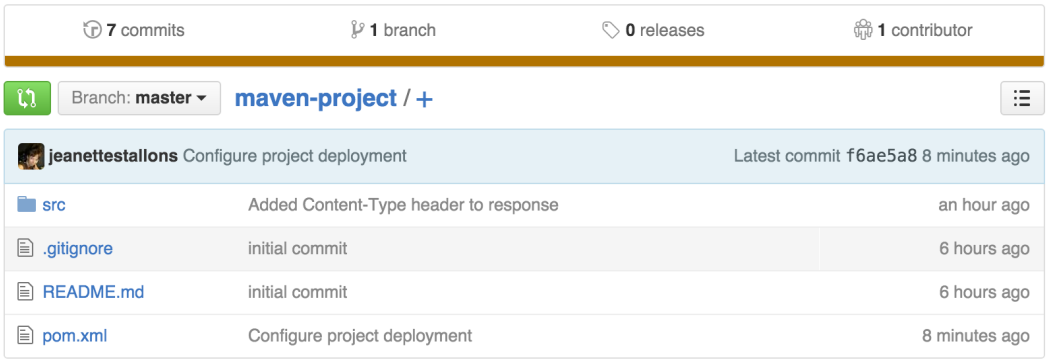
20. Click Save.

Commit the code

21. Return to $PROJECT_HOME in a command-line interface.
22. Run git status.
23. Add, commit, and push the changes.
24. In a browser, return to your maven-project GitHub repository and verify the commit.

```
git add -A && git commit -m "Configure project deployment" &&
 git push origin master
```

25. In a browser, return to your maven-project GitHub repository and verify the commit.



Trigger the build

26. Return to Jenkins.
27. Go to the maven-project-{yourInitials} project page.
28. Wait for the project to build from SCM changes or click the Build Now link.
29. In the build history, open the most recent build.
30. In the left-side navigation, click Console Output.
31. Examine the build logs and verify the application deployed.

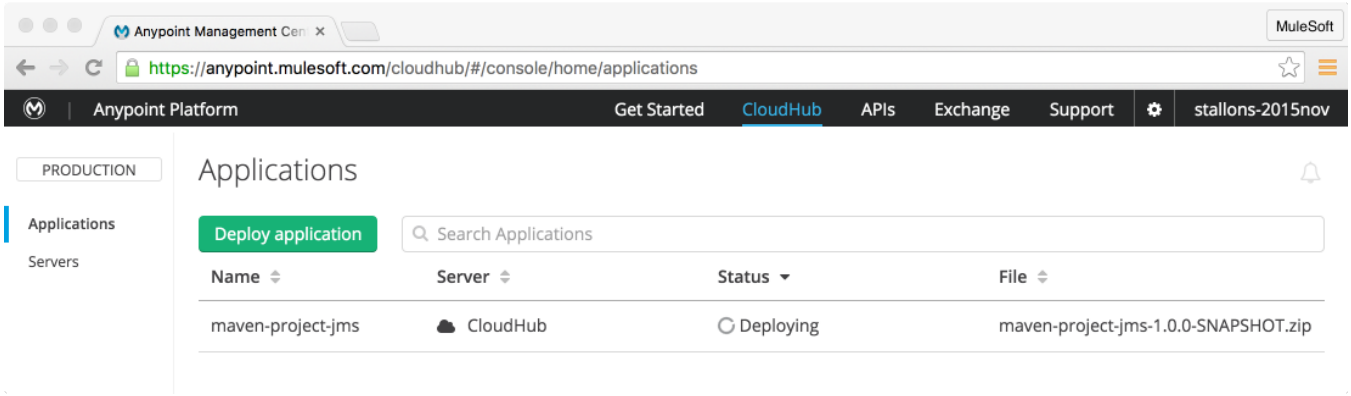
    Note: You can locate this easily by searching the file for deploy.

```
[INFO] --- mule-maven-plugin:2.0:deploy (default-cli) @ maven-project-jms ---
[INFO] No application configured. Using project artifact:
/var/lib/jenkins/jobs/maven-project-jms/workspace/target/maven-project-jms-1.0.0-
SNAPSHOT.zip
[INFO] Deploying application maven-project-jms to Cloudhub
[INFO] Creating application maven-project-jms
[INFO] Uploading application contents maven-project-jms
[INFO] Starting application maven-project-jms
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 32.900 s
[INFO] Finished at: 2015-11-14T17:58:59+00:00
[INFO] Final Memory: 25M/81M
[INFO] ------------------------------------------------------------------------
```

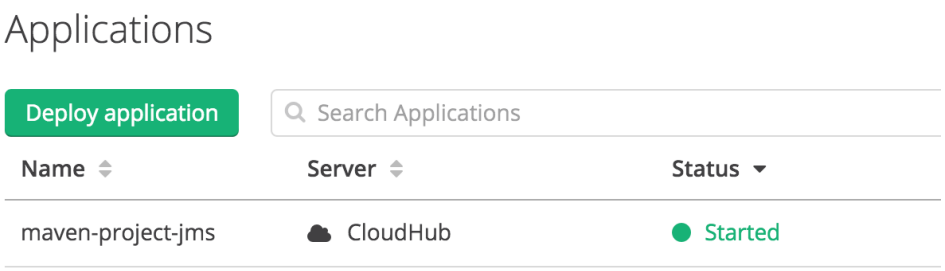
Locate the application on CloudHub

32. Return to Anypoint Platform.

33. Click CloudHub and look at the list of applications; you should see your application.



34. Wait for the application to successfully deploy.



35. Make a request to call the endpoint of your application.

```
http://maven-project-{yourInitials}.cloudhub.io/flights
```

36. Verify you receive a response back.

Module 4: Driving Development with MUnit

Objectives:

- Create a system that ensures quality, fosters agility, and presents acceptance of your work.
- Mock resources external to the unit of work you're testing.
- Identify the source of project issues without the overhead of debugging.

Walkthrough 4-1: Create acceptance criteria

In this walkthrough, you set the foundation for tests you'll be writing in subsequent walkthroughs. You will:

- Examine the use case and needs of the project.
- Determine what acceptance looks like for the application at hand.

Use case

Office Mule is in need of some serious order processing. They wish to accomplish this with Mule. On a high-level, they need this system to:

- Retrieve hourly orders from a legacy soap service.
    - The service is old, we don't know who developed it, and sometimes it sends invalid XML.
- Insert all international and domestic orders into their respective database.

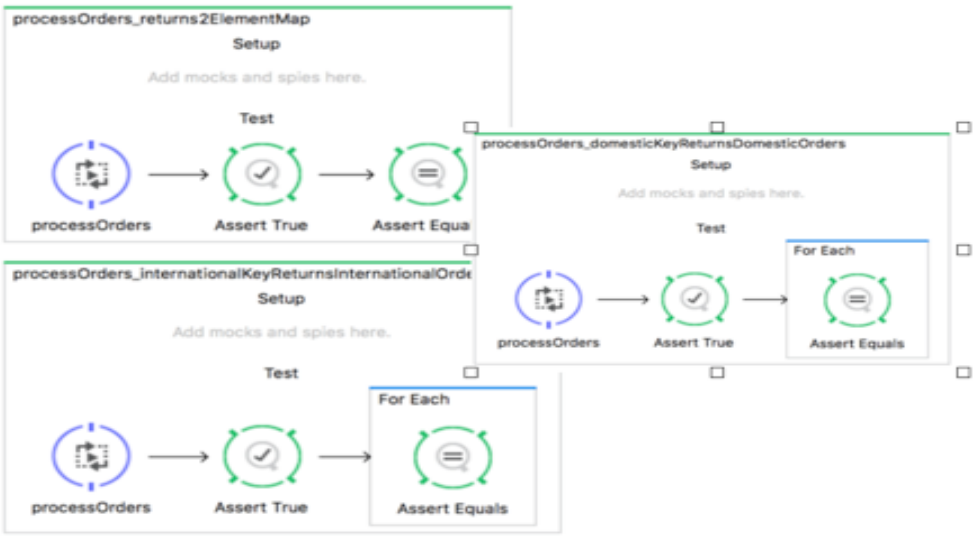
Goals

In groups, create a list of acceptance criteria for this use case. Do not concern yourself with architecture or design at this point.

Walkthrough 4-2: Create, fail, then pass tests

In this walkthrough, you will:

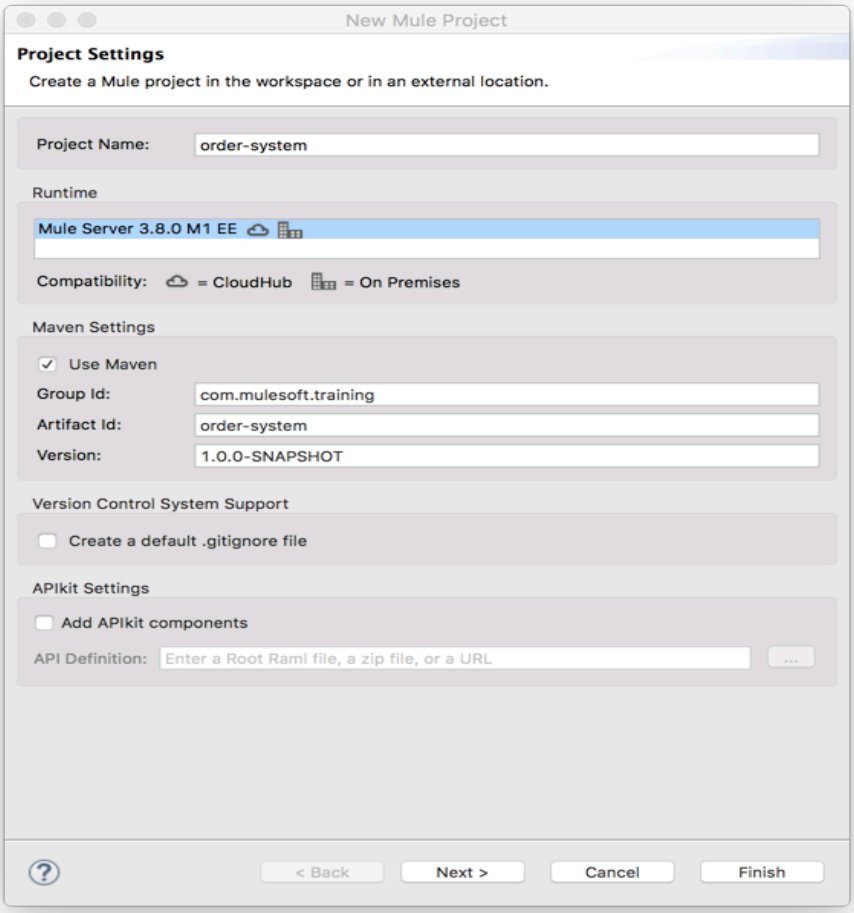
- Write tests that represent functions of your use case.
- Verify the tests fail.
- Implement Mule flows that cause the tests to pass.



Create a new Maven project

1. Create a new mule project named order-system.
2. Select a runtime at or greater than 3.8.0.
3. Check Use Maven.
4. Set the Group Id to com.mulesoft.training.

5. Verify the Artifact Id is order-system.



6. Click Finish.

Create your first test

7. Create a new flow.
8. Add a logger to the flow.
9. Rename the flow processOrders.

```
<flow name="processOrders">
  <logger/>
</flow>
```

Note: Think of this as your interface. You aren't concerned about implementation details until you establish what the goal of this flow is.

10. Right-click processOrders flow.

11. Select Munit > Create new order-system.xml Suite.
12. From the Package Explorer, expand src/test/munit.
13. Right-click on order-system-test-suite.xml.
14. Select Refactor > Rename and rename the test suite order-process_tests.xml.
15. In src/test/munit, open order-process_tests.xml.
16. Select the order-system-test-suite-processOrdersTest.
17. Set the name to processOrders_returns2ElementMap.
18. Set the Scenario description to “Calling processOrders returns two element map”.

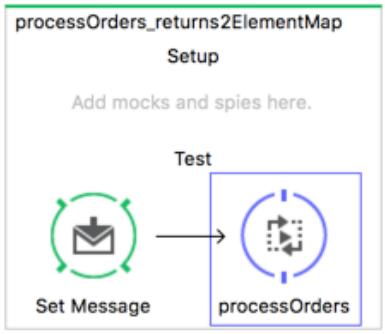
```
<munit:test name="processOrders_returns2ElementMap">
  <flow-ref name="processOrders"/>
</munit:test>
```

19. From the palette, drag in a Set Message processor before the flow-ref.
20. Configure the Payload with this expression:

```
#[[{'orderID':444, 'location':'international','price':14.01}, {'orderID':444,
'location':'international','price':14.01}, {'orderID':555,
'location':'domestic','price':14.01}]]
```

This expression can also be copied from your course snippets.txt file.

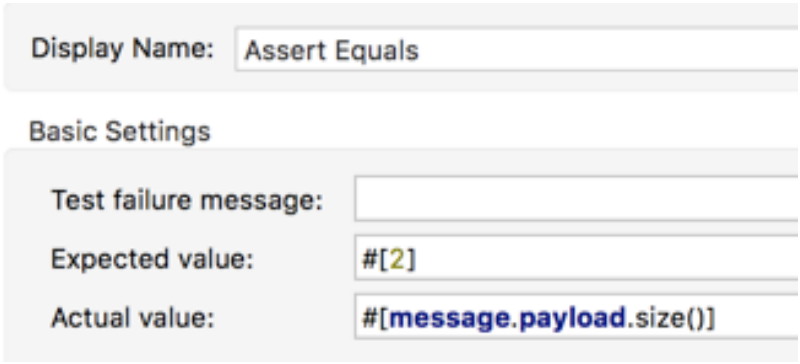
21. Rename the flow-ref to processOrders.



Note: Ensure when using flow-refs it's obvious what they're calling.

22. From the palette, drag in Assert True after the flow-ref.
23. Configure the condition to #[message.payload is java.util.Map].

24. From the palette, drag in Assert Equals after Assert True.
25. Configure Expected expression to #[2].
26. Configure the actualValue to #[message.payload.size()].



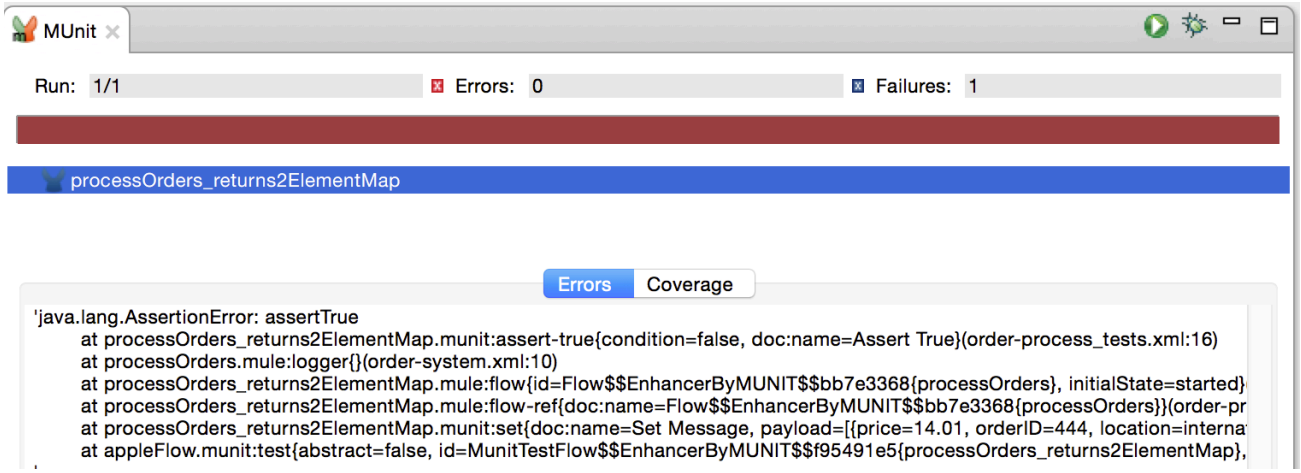
Note: Granularity of test cases can be a controversial topic. Some believe that a test should strive to only contain one assertion. As a best practice, consider the descriptiveness of your test. If it fails, will you intuitively know what failed and why?

Fail the test

27. Right-click order-process_tests.xml
28. Select Run As > MUnit.

Note: Triggering the test phase of the maven lifecycle would also be adequate.

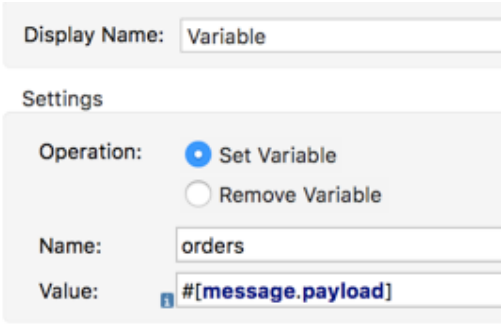
29. Verify the test shows red.



Implement a passing solution

30. Open order-system.xml.
31. Remove the logger from the processOrders flow.

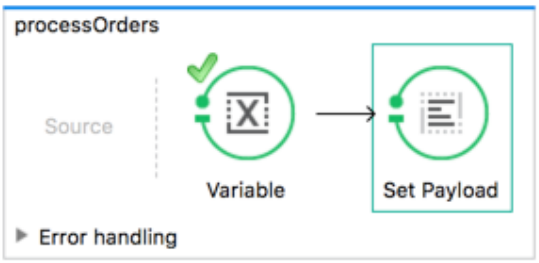
32. Add a set-variable transformer to processOrders.
33. Set the variableName to orders and value to #[message.payload].



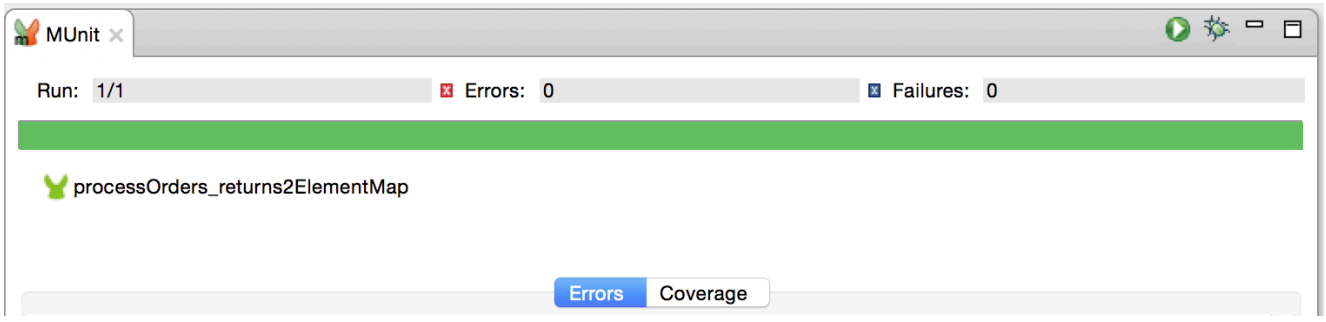
34. After set-variable, add a set-payload to processOrders.
35. Set the payload value to a new Map instance with keys international and domestic the value of each being a List of Maps.

```
#[{'international': java.util.Map<String,String>[], 'domestic':
java.util.Map<String,String>[]}]
```

Note: This script is available in the course snippets.txt file.



36. Run the munit tests again; verify they now pass.



Note: You've implemented a new Map that satisfies your data structure need. Your structure is empty and your test didn't account for the data inside.

Test for correct data

37. Return to src/test/munit/order-process_tests.xml.

38. Duplicate the processOrders_returns2ElementMap.
39. Select the duplicate test.
40. Change the duplicate's name to processOrders_internationalKeyReturnsInternationalOrders.
41. Change the Scenario description to:

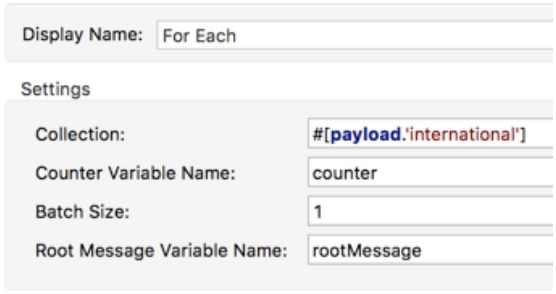
```
The value of the map key international contains only international orders
```

42. Replace the Set Message payload value to the following.

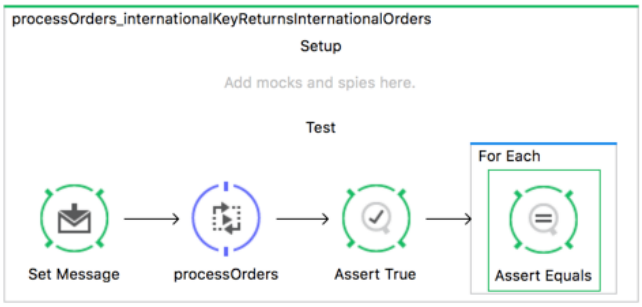
```
#[[{'orderID':444, 'location':'international','price':14.01}, {'orderID':444,
'location':'international','price':14.01}]]
```

Note: This script is available in the course snippets.txt file.

43. Add a foreach scope between the True and Equals assertions.
44. Set the collection value to #[payload.'international'].

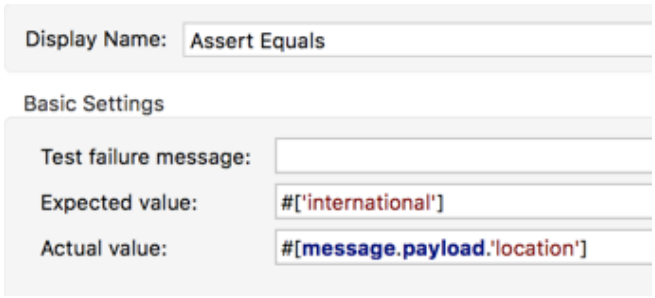


45. Drag Assert Equals into the For Each scope.



46. Select Assert Equals.
47. Set Expected expression to #['international'].

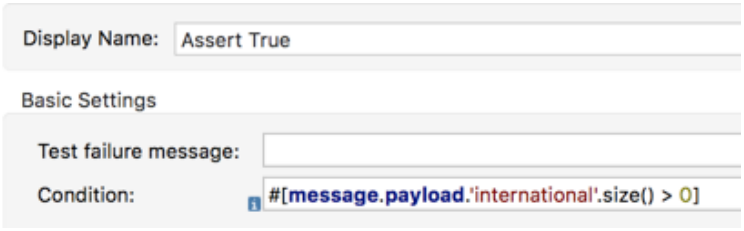
48. Set the actualValue to #[message.payload.'location'].



Note: Using foreach, you're able to loop through every item in the international key. MUnit doesn't require us to learn a whole new framework / syntax. You can use the same tools available in your Mule applications to build tests. The benefit of this will become even more evident when you enter integration tests.

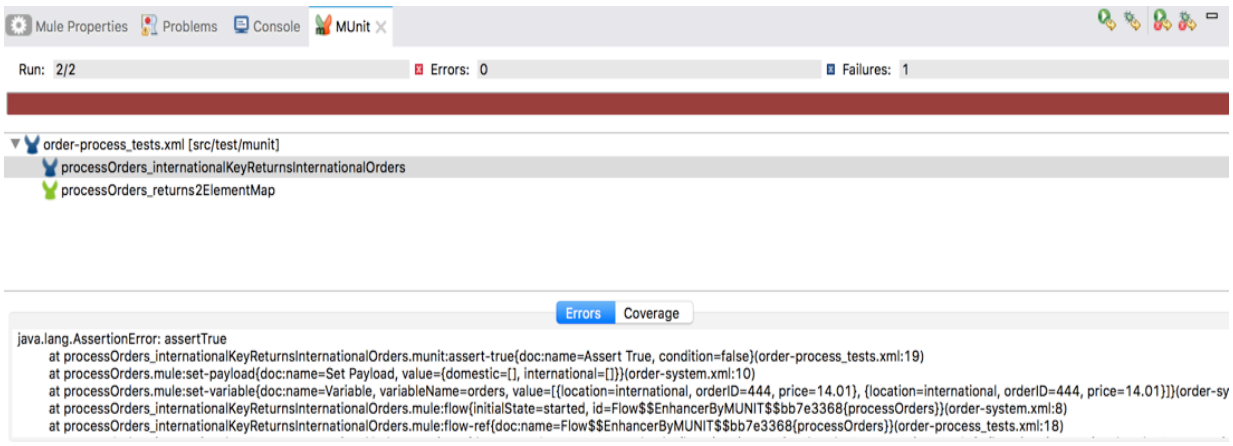
49. Select Assert True.
50. Set the Condition to check if the international key has contents:

```
#[message.payload.'international'.size() > 0]
```

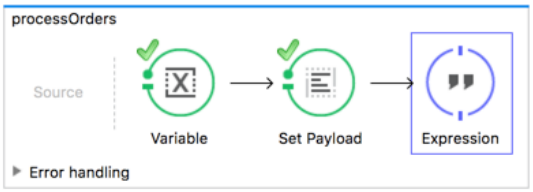


Note: This assertion is critical for the integrity of your Assert Equals. If data enters and nothing is written to the international key, foreach would not trigger and the test would pass.

51. Run the test; verify it fails.



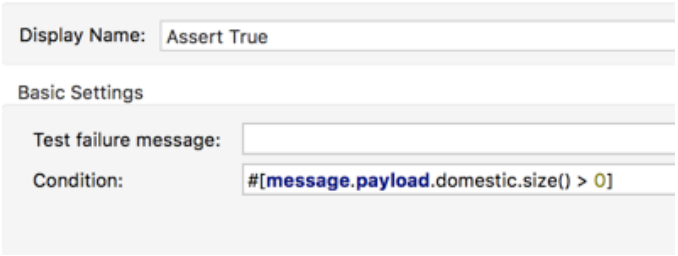
52. Open order-system.xml.
53. At the end of the processOrders flow, add an expression-component.



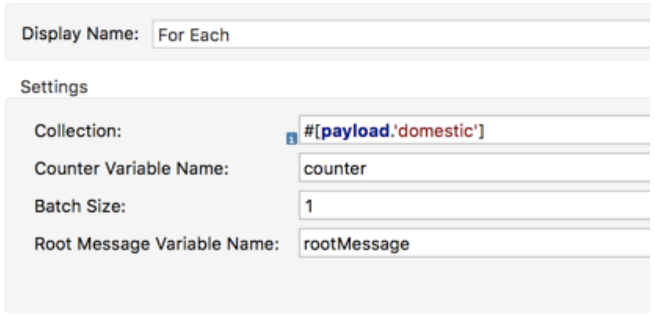
54. Write an expression that loops through flowVars.'orders' and adds each element to payload.international.

55. Run tests again; verify they pass.

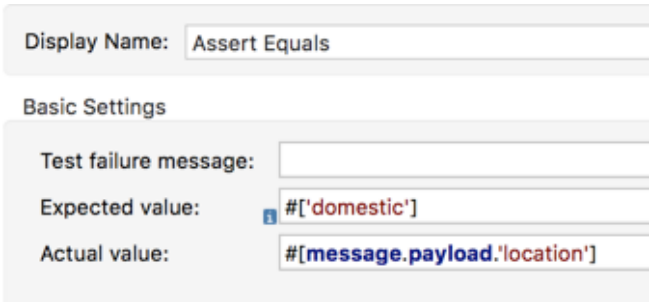
56. Return to src/test/munit/order-process_test.xml.
57. Duplicate processOrders_internationalKeyReturnsInternationalOrders and rename it to processOrders_internationalKeyReturnsDomesticOrders.
58. Configure the Assert-true for domesticOrders.



59. Configure the For-each collection to #[payload.'domestic']



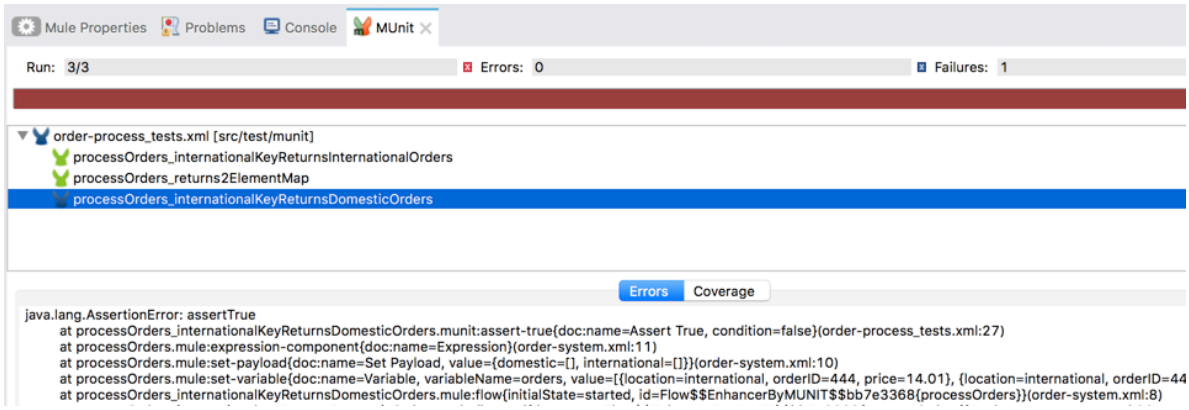
60. Configure the Assert-equals for domesticOrders.



61. Replace the Set Message element with the Payload.

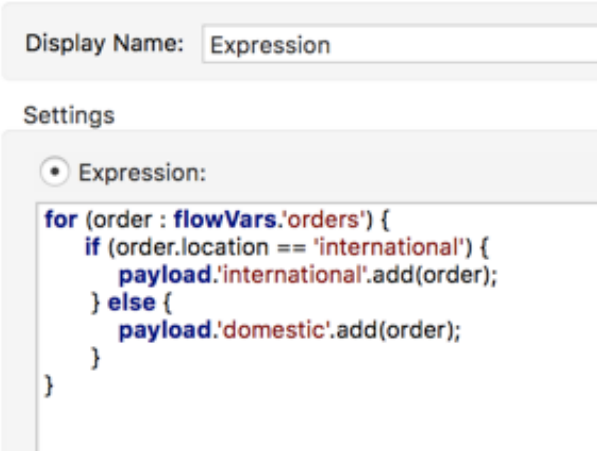
```
#[[{'orderID':444, 'location':'international','price':14.01},
    {'orderID':444, 'location':'international','price':14.01},
    {'orderID':555, 'location':'domestic','price':14.01}]]
```

62. Run tests again; verify the new test fails.

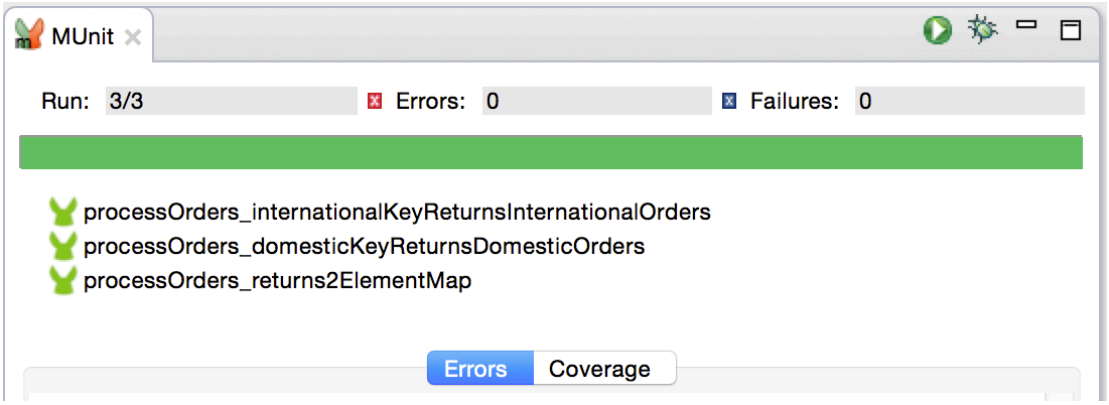


63. Open order-system.xml.

64. In the Expression component, add logic to separate domestic from international when adding to the data structure.



65. Run tests; verify they all pass.



Note: Looking at the internationalOrders test, you should now be able to use the same Mule message mock data.

66. Return to the processOrders_internationalKeyReturnsInternationalOrders test.
67. Replace the Set Message payload with that of the other tests.

```
<munit:set payload="#[[
    {'orderID':444, 'location':'international','price':14.01},
    {'orderID':444, 'location':'international','price':14.01},
    {'orderID':555, 'location':'domestic','price':14.01}
]]"/>
```

68. Save the project.
69. Verify the tests still pass.

Walkthrough 4-3: Refactor the test

In this walkthrough, you will:

- Identify a refactoring opportunity.
- Configure additional test scaffolding.

Create a before:test

1. Run all tests and verify they pass.

   Note: Running tests right before refactoring provides you peace of mind that if a test fails after refactoring, you should be able to roll back with a few undos.

2. Open src/test/munit/order-process_tests.
3. From the palette, drag a Before Test into the canvas, rename to “before.tests”.
4. Move the Set Message from one of the existing tests into the Before Tests.

   Note: Because you want your payload to be reset when a different test triggers, before-test is more appropriate than before-suite.

5. Delete any remaining Set Messages.
6. Run the tests; verify they pass.
7. Consider the following:
   - Are there other elements that could be moved into before:test?
   - What considerations would you make before moving them?

Walkthrough 4-4: Refactor the application

In this walkthrough, you will:

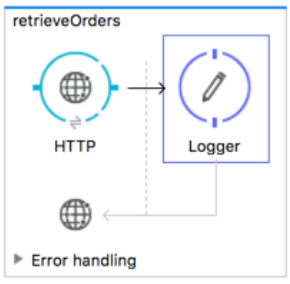
- Pass an existing test.
- Refactor part of a mule application through extraction.
- Verify the tests still pass.

Add a retrieveOrders flow

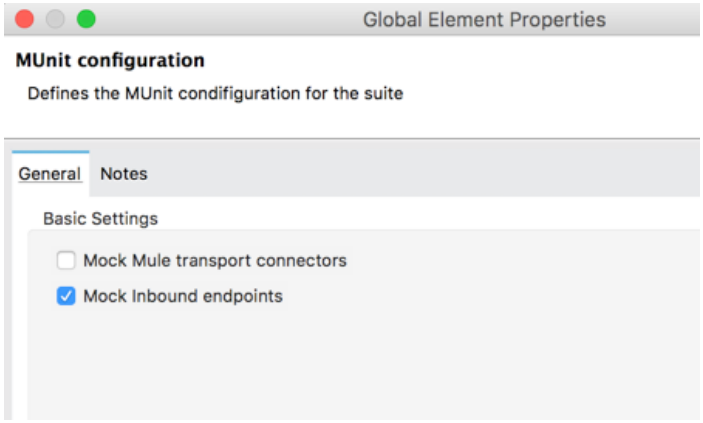
1. Open order-system.xml.
2. Create a flow named retrieveOrders.
3. Expose the flow on an http:listener available on port 8086.

```
<http:listener-config name="http8086" host="0.0.0.0" port="8086"/>
```

4. Add a logger to retrieve orders.



5. Right click retrieveOrders.
6. Create a new MUnit test suite named order-retrieve_tests.xml.
7. From Global Elements, edit the Munit configuration and deselect then option “Mock Mule transport connectors.



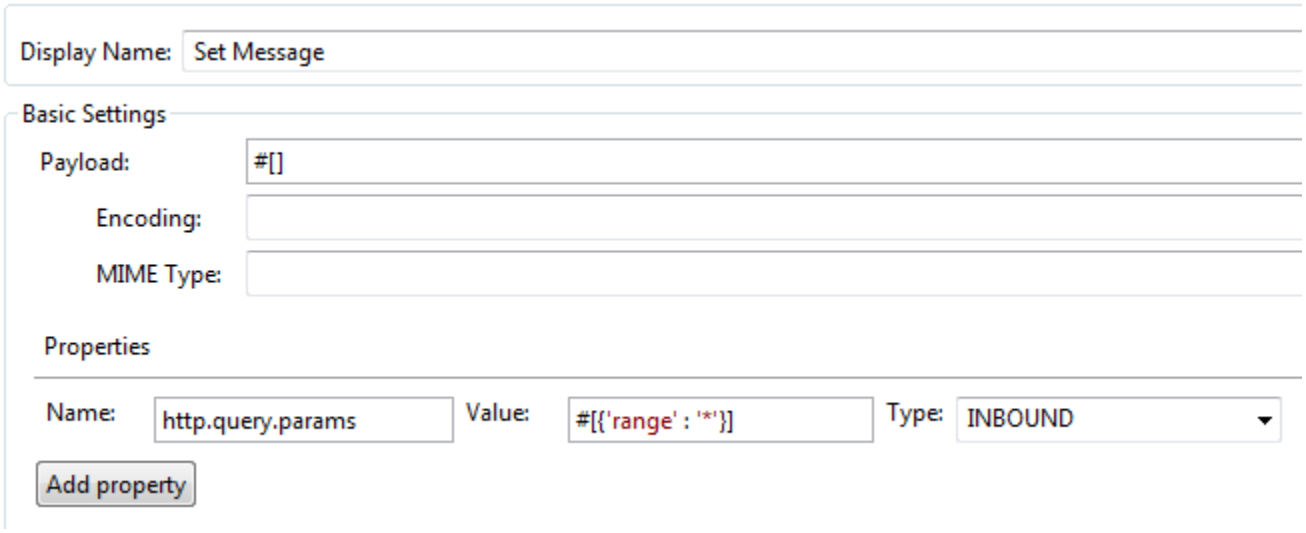
   Note: This raises an issue with your unit tests, in that they will go outbound to receive data. You'll fix this behavior later.

8. Return to the message flow and rename the flow reference to retrieveOrders.
9. Rename the test to retrieveOrders_returnsList.
10. Set the description to:

```
Calling retrieveOrders returns a java list of orders.
```

11. At the beginning of the test, add a munit:set (Set Message).
12. Keep the payload empty.
13. Click Add Property.
14. Set the property name to http.query.params.
15. Set the value to `#[{'range' : '*'}]`.

16. From the drop-down menu, select INBOUND property.

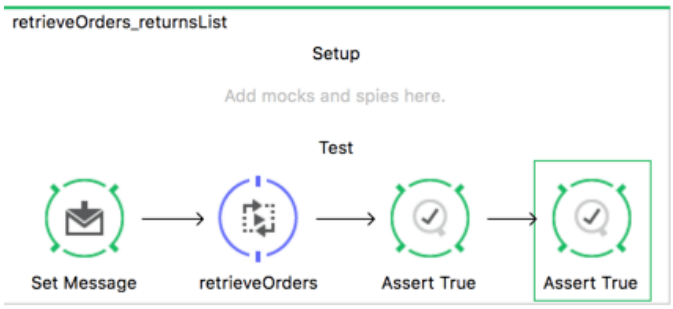


17. At the end of the test, add an assert-true.
18. Set the condition to ensure the payload is a List.

```
#[message.payload is java.util.List]
```

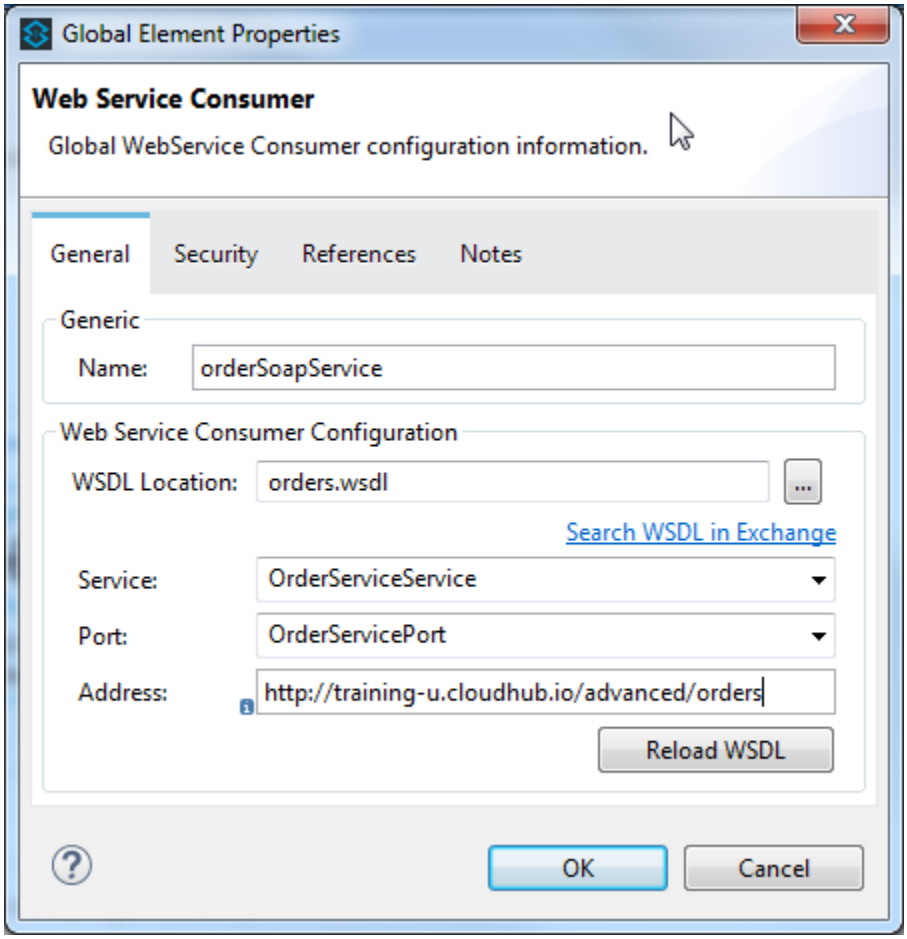
19. Add another assert-true to the end of the test.
20. Set the condition to ensure the list contains elements.

```
#[message.payload.size() > 0]
```



21. Run the test; verify it fails.
22. Open order-system.xml.
23. Delete the logger from retrieveOrders.
24. Add a ws:consumer to retrieveOrders.
25. Drag studentFiles/resources/orders.wsdl into src/main/resources.
26. Select ws:consumer.

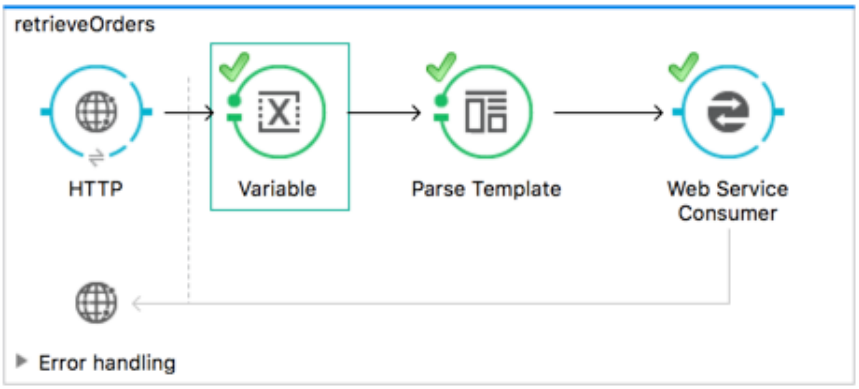
27. Create a global Web Service Consumer configuration.
28. Set the name orderSoapService.
29. Set the WSDL location to orders.wsdl.
30. If the service and port are not automatically populated after the WSDL is parsed, select them from the drop-down menus manually to the values shown below.



31. Click OK.
32. Set Operation to provisionOrder.
33. In the processOrders flow, before the ws:consumer, add a parse-template transformer.
34. Drag the studentFiles/resources/orderRequest.xml into src/main/resources.
35. Open orderRequest.xml; examine its contents.
36. Return to order-system.xml.
37. Select the Parse Template transformer; set its location to orderRequest.xml.
38. Add a set-variable transformer before the parse-template transformer.

39. Name the variable dateRange and give it a value of #[message.inboundProperties.'http.query.params'.range].

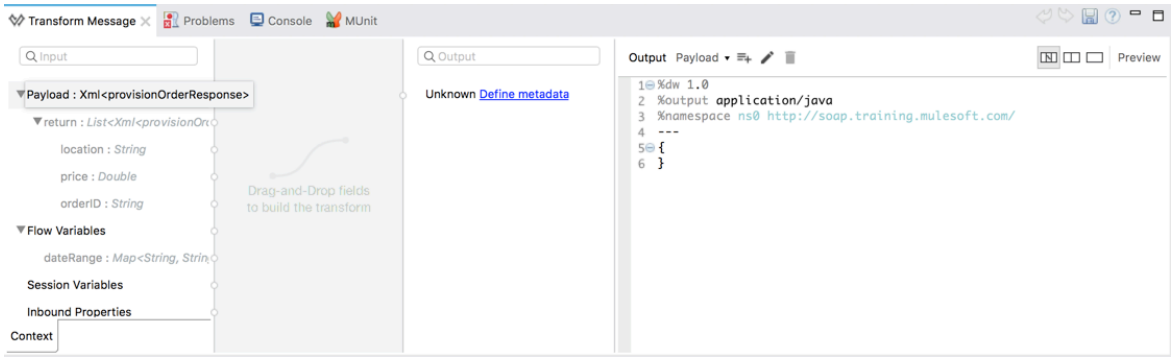
    Note: You will retrieve your dateRange from the http header.




```
<flow name="retrieveOrders">
  <http:listener path="/" config-ref="http8086" />
  <set-variable variableName="dateRange"
      value="#[message.inboundProperties.'http.query.params'.range]"/>
  <parse-template location="orderRequest.xml" doc:name="Parse Template"/>
  <ws:consumer config-ref="orderSoapService" operation="provisionOrder"/>
</flow>
```

Transform the response

40. After ws:consumer, drag in a dw:transformer.
41. From the left of DataWeave, expand the return node.
42. Examine its contents.



43. Select the Payload : Xml<provisionOrderResponse>.
44. Right-click and select Edit Sample Data.
45. Construct sample data with one international order and one domestic order.

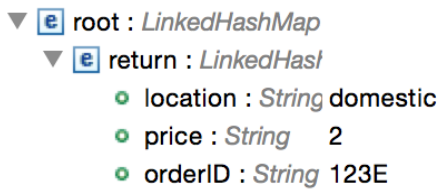
```
<?xml version='1.0' encoding='UTF-8'?>
 <ns0:provisionOrderResponse xmlns:ns0="http://soap.training.mulesoft.com/">
```

```
  <return>
    <location>international</location>
    <price>2</price>
    <orderID>123E</orderID>
  </return>
  <return>
    <location>domestic</location>
    <price>2</price>
    <orderID>123E</orderID>
    </return>
</ns0:provisionOrderResponse>
```

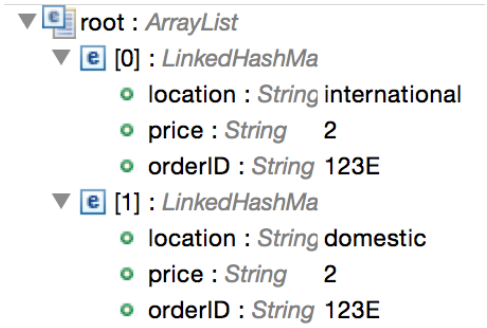
46. From the output panel, click the Preview tab.
47. In the script editor, remove the curl braces.
48. Add the payload and access the order response with its respectively namespace.

```
payload.ns0#provisionOrderResponse
```

49. Examine the change to the Output preview.



50. Add .*return at the end of the line.
51. Save and review the output.



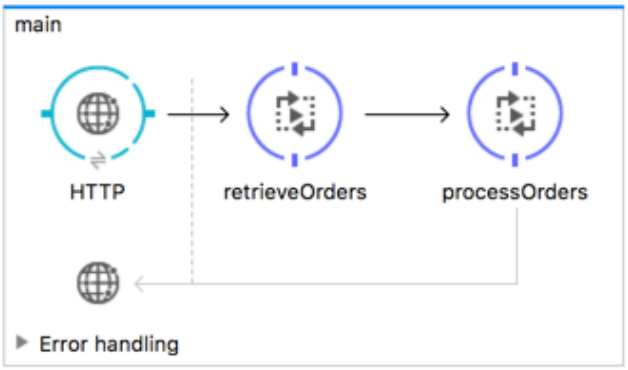
Note: This should match the structure you expected in your unit tests.

52. Run the test, verify it passes.

Refactor the app

53. Return to order-system.xml

54. Rename the flow containing the http:listener main.
55. In main, extract all the message processors into a subflow.
56. Name the subflow retrieveOrders.
57. Rename the flow reference retrieveOrders.
58. At the end of the main flow, add another flow reference pointed to processOrders.

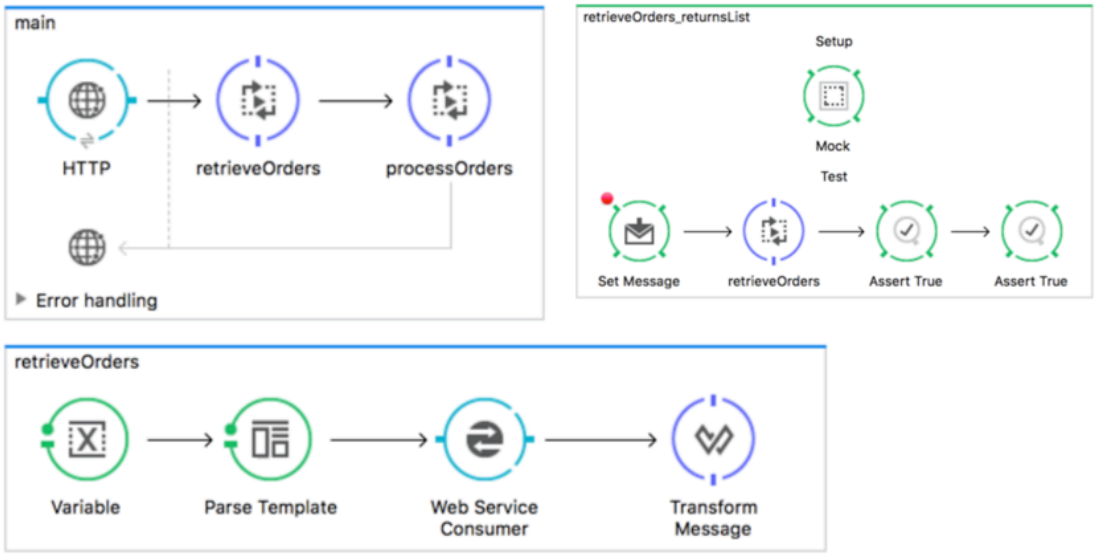


59. Run the tests again and verify they pass.

Walkthrough 4-5: Mock an external endpoint

In this walkthrough, you will:

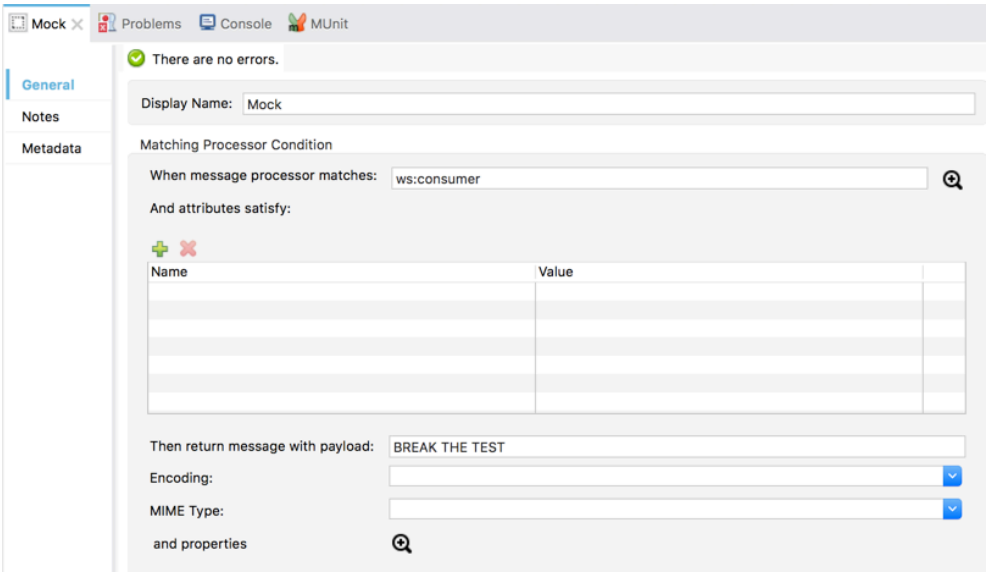
- Replace an external call with a mock.
- Define a mock payload response.
- Define a mock property response.



Mock Web Service Consumer

1. Return to Anypoint Studio and open order-retrieve_tests.xml
2. Inside the munit:test, add a mock:when element.
3. Set mock:when for a messageProcessor that matches ws:consumer.

4. Return a message with a payload of BREAK THE TEST.



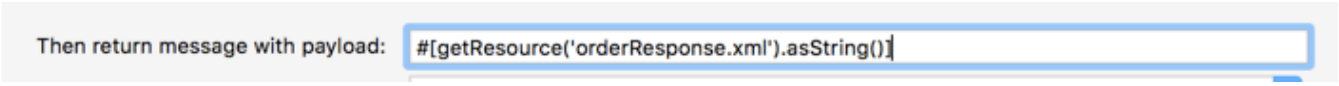
5. Run the MUnit tests again, verify it fails.
6. Locate the area in the logs; examine the reason for failure.

```
Type mismatch found :name, :string   required :name, :object
(com.mulesoft.weave.mule.exception.WeaveExecutionException). Message payload is of
type: String org.mule.api.MessagingException: Exception while executing:
payload.ns0#provisionOrderResponse.*return
```

Note: DataWeave threw an exception during your test due to not getting appropriate data.

7. From studentFiles/resources drag the orderResponse.xml into src/test/resources.
8. Select the mock:when element.
9. Change the return message with a payload value equal to the following:

   #[getResource('orderResponse.xml').asString()]



8. Run the MUnit tests and examine the source of failure.
9. Debug the application.

   Note: You may need to add -DskipTests to the Maven run configuration with Anypoint Studio.

10. Send a request into http://localhost:8086?range=all.
11. Place a breakpoint on DataWeave.
12. Examine the message's inboundProperties that it reaches DataWeave.

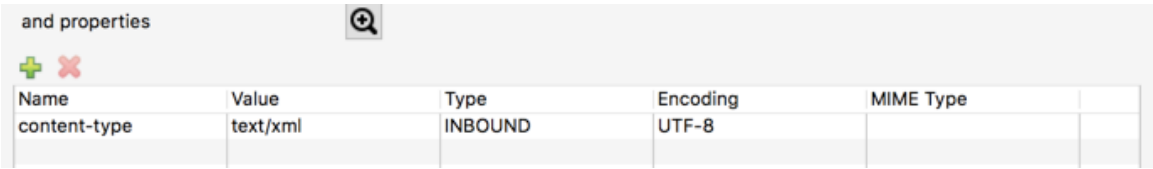
13. Locate the content-type header.

    Note: Your mock is not realistic. When Web Service Consumer returns it places a content-type header onto the message. You need to be sure your mocked service also returns a content-type with value text/xml; charset=UTF-8.

14. Stop the debugger.

Add a mocked inboundProperty resource

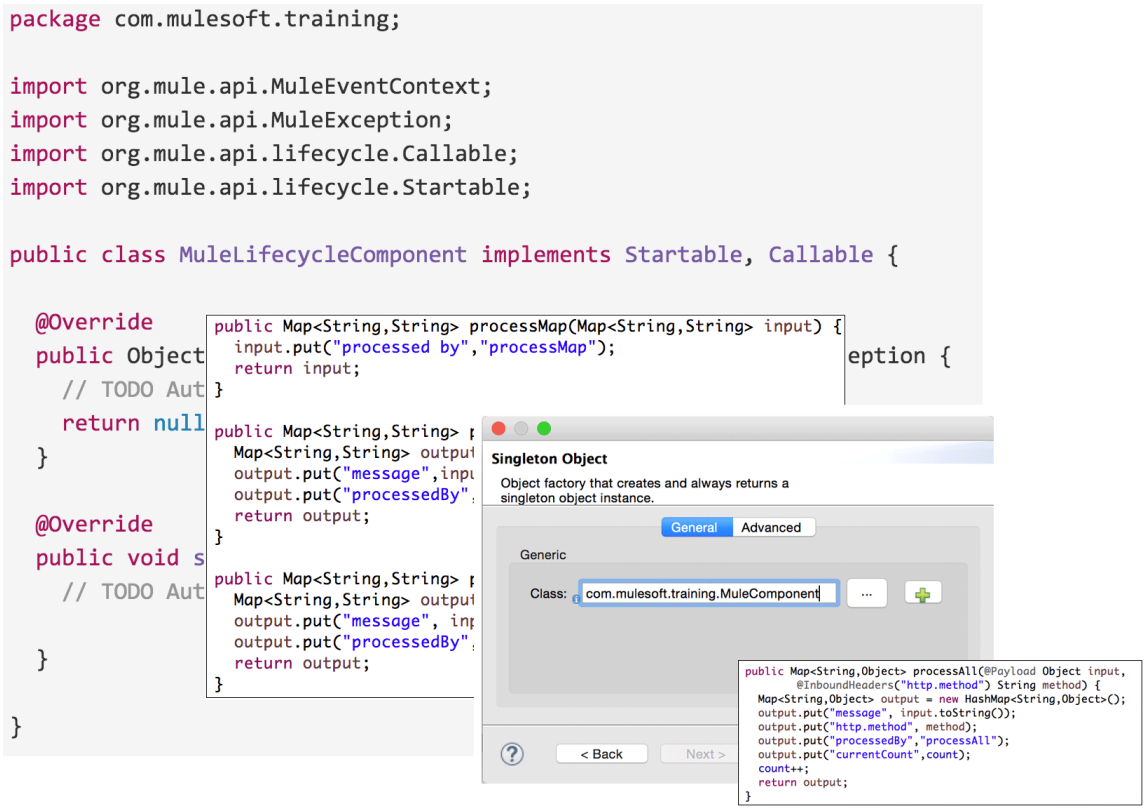
15. Select the mock:when element.
16. Click the expand properties section.
17. Click Add to add new property.
18. Set the Property Name to content-type.
19. Set the value to text/xml; Encoding=UTF-8.
20. Set Type to INBOUND.



21. Save the test.
22. Run the tests again; verify they succeed.

    Note: Feel free to attempt this tests with your internet disabled to ensure they're truly portable and not relying on external resources.

Module 5: Developing Custom Elements



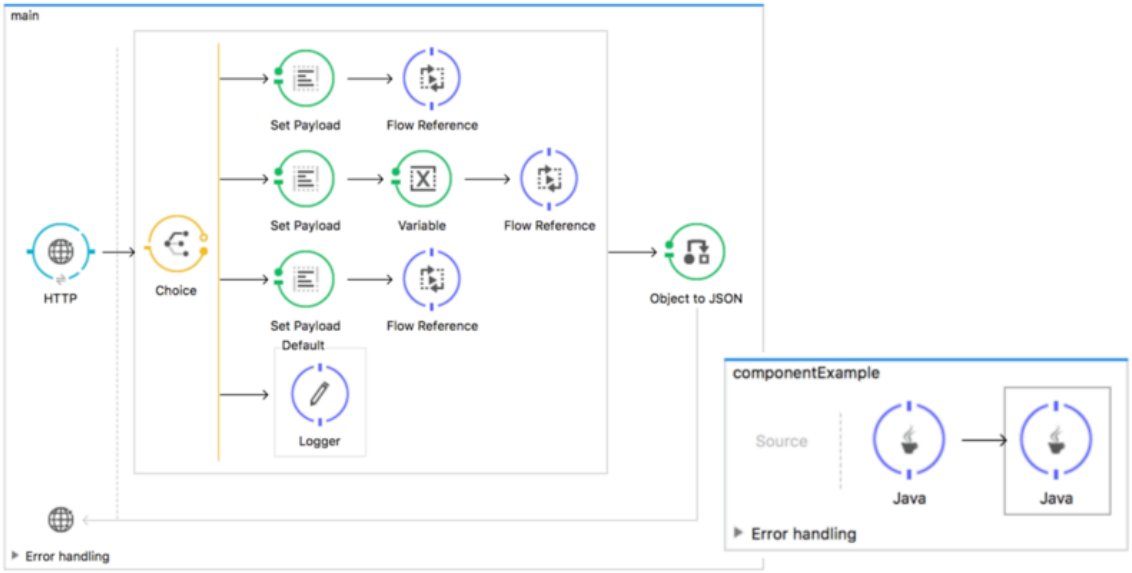
Objectives:

- Write custom Java components.
- Specify that method in code Mule should invoke.
- Examine custom transformer model.
- Examine custom filter model.

Walkthrough 5-1: Examine a component's behavior

In this walkthrough, you will:

- Configure a component inside a flow.
- Call the component.
- Examine how the entry point is resolved.



Import the project

1. Return to Anypoint Studio.
2. From the File menu, select Import.
3. Browse to the POM file inside of studentFiles/apps/custom-elements.
4. Click Finish.
5. If not already, open the mule-config.xml.
6. Select the choice router.
7. Examine the routes that choice contains.

When	Route Message to
#[message.inboundProperties.'http.query.params'.type == 'map']	Set Payload
#[message.inboundProperties.'http.query.params'.type == 'array']	Set Payload
#[message.inboundProperties.'http.query.params'.type == 'string']	Set Payload
Default	Logger

```
<choice>
  <when expression="#[message.inboundProperties.'http.query.params'.type == 'map']">
    <set-payload value="#[{'message' : 'i am a map'}]" />
    <flow-ref name="componentExample"/>
  </when>
  <when expression="#[message.inboundProperties.'http.query.params'.type ==
'array']">
    <set-payload value="#[['i am an array node']]" />
    <flow-ref name="componentExample"/>
  </when>
  <when expression="#[message.inboundProperties.'http.query.params'.type ==
      'string']">
    <set-payload value="i am a string" />
    <flow-ref name="componentExample"/>
  </when>
  <otherwise>
    <logger message="Could not locate requested path"/>
  </otherwise>
</choice>
```

Note: Based on routing, the applications examines a query parameter named type.

8. In the componentExample flow, add a Java component.
9. Set the class to com.mulesoft.training.MuleComponent.
10. Examine the rest of the application.
11. Start the project to verify it compiles and starts up.

Request the endpoint

12. Call the endpoint http://localhost:8081/component?type=map.
13. Verify an exception returns.

14. Return to Anypoint Studio.
15. Open the console.

16. Locate the root exception.

```
 Root Exception stack trace: org.mule.model.resolvers.EntryPointNotFoundException:
Failed to find entry point for component, the following resolvers tried but failed: [
ReflectionEntryPointResolver: Found too many possible methods on object
"com.mulesoft.training.MuleComponent" that accept parameters "{}", Methods matched
are "[public java.util.Map com.mulesoft.training.MuleComponent.processArray(), public
java.util.Map com.mulesoft.training.MuleComponent.processString2(), public
java.util.Map com.mulesoft.training.MuleComponent.processString(), public
java.util.Map com.mulesoft.training.MuleComponent.processMap()]"
CallableEntryPointResolver: Object "com.mulesoft.training.MuleComponent@1d64d521"
does not implement required interface "interface org.mule.api.lifecycle.Callable"
AnnotatedEntryPointResolver: Component: com.mulesoft.training.MuleComponent@1d64d521
doesn't have any annotated methods, skipping. MethodHeaderPropertyEntryPointResolver:
The required property "method" is not set on the event ]         at
org.mule.model.resolvers.DefaultEntryPointResolverSet.invoke(DefaultEntryPointResolve
rSet.java:49)         at
org.mule.component.DefaultComponentLifecycleAdapter.invoke(DefaultComponentLifecycleA
dapter.java:339)         at
org.mule.component.AbstractJavaComponent.invokeComponentInstance(AbstractJavaComponen
t.java:82)     + 3 more (set debug level logging or '-Dmule.verbose.exceptions=true'
for everything)
********************************************************************************
```

17. Locate and examine the 4 resolution errors individually.

```
 ReflectionEntryPointResolver: Found too many possible methods on object
 "com.mulesoft.training.MuleComponent" that accept parameters"
```

```
 CallableEntryPointResolver: Object "com.mulesoft.training.MuleComponent@1d64d521"
does not
 implement required interface "interface org.mule.api.lifecycle.Callable"
```

```
 AnnotatedEntryPointResolver: Component: com.mulesoft.training.MuleComponent@1d64d521
 doesn't have any annotated methods, skipping.
```

```
 MethodHeaderPropertyEntryPointResolver: The required property "method" is not set on
 the Event
```

18. Open the src/main/java/MuleComponent.java.

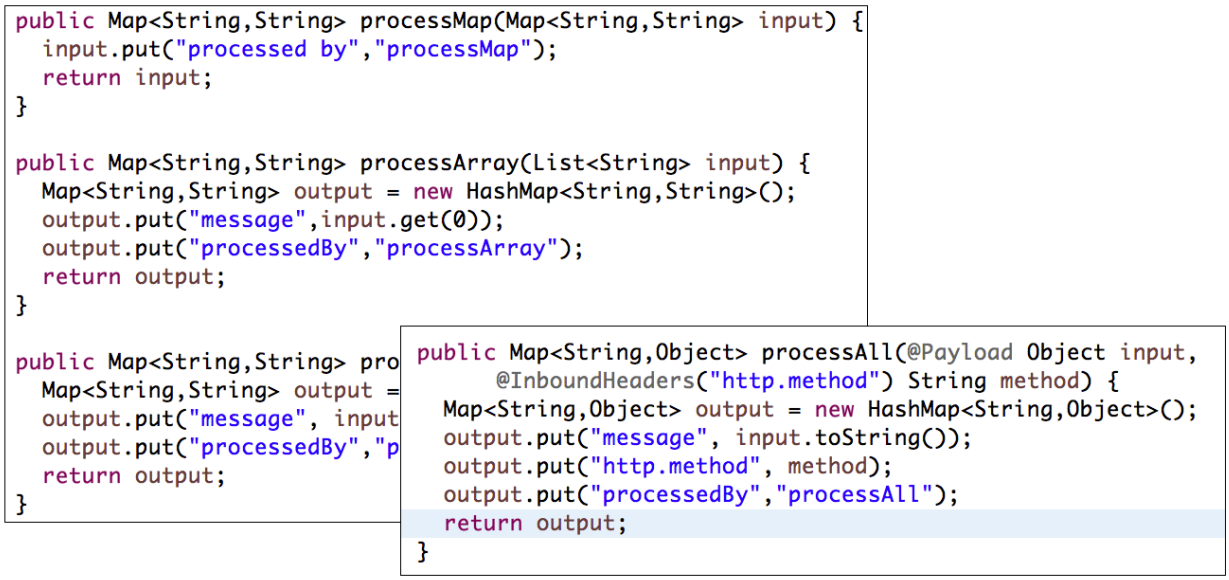
```
public class MuleComponent {
  public Map<String,String> processMap() {
    // processMap implementation
    return null;
  }

  public Map<String,String> processArray() {
    // processArray implementation
    return null;
  }

  public Map<String,String> processString() {
    // processString implementation
    return null;
  }
}
```

19. Examine its methods to answer the following:
    a. What is the root cause of the reflection exception?
    b. Why could a method not be resolved via annotations?
20. Stop the project.

Walkthrough 5-2: Resolve the entry point

In this walkthrough, you will:

- Resolve a method based on reflection.
- Resolve a method with annotations.
- Resolve a method based on a property.



Specify arguments

1. In the custom-element project, open src/main/java/MuleComponent.java.
2. Locate the processMap method.
3. Add a parameter named input of type Map<String,String>.
4. In processMap, put a new key named processedBy and value processMap.
5. Return the input.

```
public Map<String,String> processMap(Map<String,String> input) {
  input.put("processed by","processMap");
  return input;
}
```

Format the output

6. Open mule-config.xml.
7. At the end of the main flow, add an object-to-json-transformer.
8. Save and run the project.

Call the endpoint

9. Call the endpoint http://localhost:8081/component?type=map.
10. Verify the response is correct.

```
{
  "processed by" : "processMap",
  "message" : "i am a map"
}
```

11. Call the endpoint http://localhost:8081/component?type=array.
12. Verify the exception from the previous walkthrough is received.
13. Stop the application.

    Note: Stopping and restarting the application is required; because you're editing Java, you need to retrigger Mule's class loader.

Configure the other methods for reflection

14. Open src/main/java/MuleComponent.java.
15. Locate the processArray method.
16. Add a parameter named input of type List<String>.

    Note: Do Java package imports as needed.

17. In processArray, instantiate a HashMap named output.
18. Put the following two items into output.
    - 'message' : input.get(0)
    - 'processedBy' : 'processArray'
19. Return output.

```
public Map<String,String> processArray(List<String> input) {
  Map<String,String> output = new HashMap<String,String>();
  output.put("message",input.get(0));
  output.put("processedBy","processArray");
  return output;
}
```

20. Locate the processString method.

21. Implement the processString method following the pattern of processArray.

```
public Map<String,String> processString(String input) {
  Map<String,String> output = new HashMap<String,String>();
  output.put("message", input);
  output.put("processedBy","processString");
  return output;
}
```

21. Save the project.
22. Run the application.

Call all endpoints

23. Call all endpoints and verify appropriate results.

```
curl http://localhost:8081/component?type=map

{"processed by":"processMap","message":"i am a map"}
```

```
curl http://localhost:8081/component?type=array
{"message":"i am an array node","processedBy":"processArray"}
```

```
curl http://localhost:8081/component?type=string
{"message":"i am a string","processedBy":"processString"}
```

Resolve by annotations

24. Return to Anypoint Studio.
25. Open src/main/java/MuleComponent.java.
26. Duplicate the processString method.
27. Rename the method processAll.
28. Replace the input parameter's type with Object.
29. Annotate the parameter with @Payload from the org.mule.api.annotations.param package.

```
public Map<String,String> processAll(@Payload Object input) {
```

Note: You may wish to import org.mule.api.annotations.param..*

30. Add a second parameter named method of type String.

31. Annotate method with @InboundHeaders("http.method").

```
public Map<String,String> processAll(@Payload Object input,
   @InboundHeaders("http.method") String method) {
```

32. Update the first map put's value to input.toString().

```
output.put("message", input.toString());
```

33. Add another map put to include the value of http.method.

```
output.put("http.method", method);
```

34. Update the last map put to have a value of processAll.
35. Verify the method still returns ouptut.

```
public Map<String,String> processAll(@Payload Object input,
    @InboundHeaders("http.method") String method) {
  Map<String,String> output = new HashMap<String,String>();
  output.put("message", input.toString());
  output.put("http.method", method);
  output.put("processedBy","processAll");
  return output;
}
```

36. Save and run the application.

Call the endpoints

37. Call all endpoints and examine the results.

```
curl http://localhost:8081/component?type=map

{
  "message" : "{message=i am a map}",
  "http.method":"GET",
  "processedBy":"processAll"
}
```

```
curl http://localhost:8081/component?type=array

{
  "message" : "[i am an array node]",
```

```
  "http.method":"GET",
  "processedBy":"processAll"
}
```

```
curl http://localhost:8081/component?type=string

{
  "message" : "i am a string",
  "http.method" : "GET",
  "processedBy" : "processAll"
}
```

Note: Due to the annotations, all requests resolve to the processAll() method.

38. Stop the application.

Set property resolution

39. Return to Anypoint Studio.
40. Open mule-config.xml.
41. In the choice router, locate the array route.
42. After set-payload, add a set-variable transformer.
43. Set the variableName to “method” and the value to “processArray”.
44. Save and start the application.

Call the endpoints

45. Call all endpoints and examine the results.

```
curl http://localhost:8081/component?type=map

{
  "message" : "{message=i am a map}",
  "http.method":"GET",
  "processedBy":"processAll"
}
```

```
curl http://localhost:8081/component?type=array

{
  "message" : "i am an array node",
 "processedBy":"processArray"
}
```

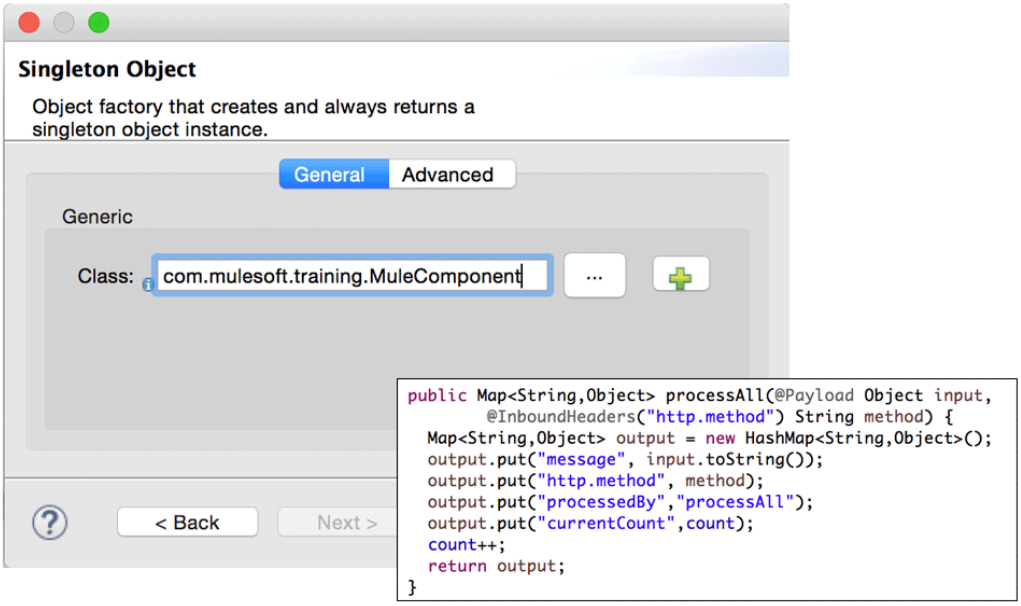
```
curl http://localhost:8081/component?type=string

{
  "message" : "i am a string",
  "http.method" : "GET",
  "processedBy" : "processAll"
}
```

Note: The array route is now triggering the processArray method. All other routes still call processAll.

Walkthrough 5-3: Set component scope

In this walkthrough, you will:

- Set a component's scope to singleton.
- Analyze the difference between singletons and prototypes.



Modify the MuleComponent

1. Open Anypoint Studio.
2. In the custom-elements project, open src/test/java/MuleComponent.java.
3. Add a integer data member named count.
4. Create a default constructor that sets count to 1.

```
public class MuleComponent {
  int count;

  public MuleComponent() {
    count = 1;
  }

  // methods
}
```

5. In processAll(), add another key / value pair of currentCount and count.
6. Before the return, increment count.
7. Replace processAll() return type with Map<String,Object>.

8. Replace output's type with Map<String,Object>.

```
public Map<String,Object> processAll(@Payload Object input,
    @InboundHeaders("http.method") String method) {
  Map<String,String> output = new HashMap<String,String>();
  output.put("message", input.toString());
  output.put("http.method", method);
  output.put("processedBy","processAll");
  output.put("currentCount",count);
  count++;
  return output;
}
```

Test the endpoint

9. Start the Mule application.
10. Open a web browser.
11. Request the endpoint http://localhost:8081/component?type=string.
12. Examine the currentCount value in each invocation.

```
curl http://localhost:8081/component?type=string
{
  "message" : "i am a string",
  "currentCount" : 1,
  "http.method" : "GET",
  "processedBy" : "processAll"
}
```

13. Stop the project.

Set the component to a singleton

14. Return to Anypoint Studio.
15. Open mule-config.xml.
16. In componentExample, locate the MuleComponent.
17. Remove the class specified and click the green plus sign.
18. Select a singleton.
19. Specify the class, com.mulesoft.training.MuleComponent.

```
<component>
  <singleton-object class="com.mulesoft.training.MuleComponent" />
</component>
```

20. Save and start the project.

Test the endpoint

21. Start the Mule application.
22. Open a web browser.
23. Request the endpoint http://localhost:8081/component?type=string.
24. Examine the currentCount value in each invocation.

```
curl http://localhost:8081/component?type=string

 {
   "message" : "i am a string",
   "currentCount" : 1,
   "http.method" : "GET",
   "processedBy" : "processAll"
 }
```

Note: While its not common to have stateful components such as these be set to singleton, this demonstrates the reuse of your initially instantiated class MuleComponent.

Walkthrough 5-4: Implement the component lifecycle

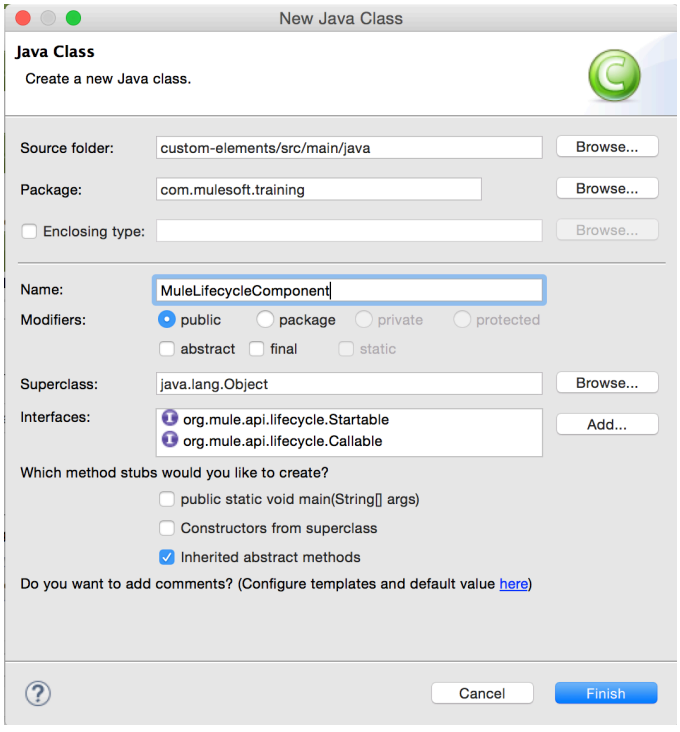
In this walkthrough, you will:

- Implement the Startable interface.
- Implement the Callable interface.
- Analyze the impact this has on singletons and prototypes.

```
import org.mule.api.MuleEventContext;
import org.mule.api.MuleException;
import org.mule.api.lifecycle.Callable;
import org.mule.api.lifecycle.Startable;

public class MuleLifecycleComponent implements Startable, Callable {
```

Create a new component

1. Return to Anypoint Studio.
2. Open the custom-elements project.
3. From src/main/java, create a new Class.
4. Set the following attributes on the class:
   - Package: com.mulesoft.training
   - Name: MuleLifecycleComponent
   - Interfaces: Startable and Callable (org.mule.api.lifecycle)



5. Examine the newly created class.

```
package com.mulesoft.training;

import org.mule.api.MuleEventContext;
import org.mule.api.MuleException;
import org.mule.api.lifecycle.Callable;
import org.mule.api.lifecycle.Startable;

public class MuleLifecycleComponent implements Startable, Callable {

  @Override
  public Object onCall(MuleEventContext eventContext) throws Exception {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public void start() throws MuleException {
    // TODO Auto-generated method stub
  }
}
```

6. At the top of the class, add the following import:

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

7. Add the following data member to the class:

```
private static final Logger logger =
    LogManager.getLogger("com.mulesoft.training.Logger");
```

8. Locate the start method.
9. In start(), add a info("Component initiated startup") call from the logger.

```
logger.info("Component initiated startup");
```

10. Locate the onCall() method.
11. In onCall(), add a info("Component fired") call from the logger.
12. Return the Message using eventContext.getMessage().

```
@Override
public Object onCall(MuleEventContext eventContext) throws Exception {
    // TODO Auto-generated method stub
    logger.info("************ Component executed **************");
    return eventContext.getMessage();
}
```

Reference the component

13. Open mule-config.xml.
14. Locate the componentExample flow.
15. Add a component to the end of the flow.
16. Set the class to com.mulesoft.training.MuleLifecycleComponent.

    Note: Recall this will act a prototype.

Test the application

17. Save and start the application.
18. Examine the logs after startup.
19. Was the start method invoked?
20. Call the endpoint http://localhost:8081/component?type=string multiple times.
21. Examine the logs.

```
INFO com.mulesoft.training.Logger: Component initiated startup
INFO com.mulesoft.training.Logger: Component fired
INFO com.mulesoft.training.Logger: Component initiated startup
INFO com.mulesoft.training.Logger: Component fired
```

22. Stop the application.

Change the scope to singleton

23. In the componentExample flow, return to the MuleLifecycleComponent.
24. Remove the class.
25. Click the green plus button.
26. Create a singleton pointed to the class com.mulesoft.training.MuleLifecycleComponent.
27. Save and start the application.

28. Examine the logs, locate the log from the start method.

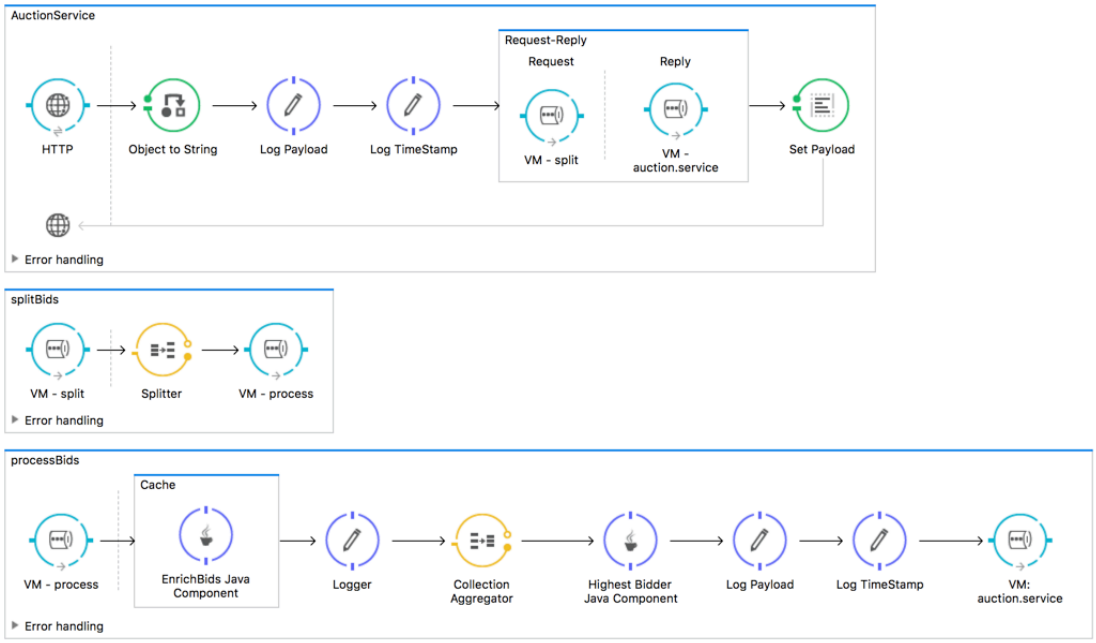
```
INFO com.mulesoft.training.Logger: Component initiated startup
INFO org.mule.DefaultMuleContext:
**********************************************************************
* Application: custom-elements-complete-1.0-SNAPSHOT                 *
* OS encoding: /, Mule encoding: UTF-8                               *
* Agents Running:                                                    *
*   JMX Agent                                                        *
*   Batch module default engine                                      *
*   DevKit Extension Information                                     *
*   Wrapper Manager                                                  *
**********************************************************************
```

Note: The singleton was created at startup; once the server starts, the Startable phase triggers.

29. Call the endpoint http://localhost:8081/component?type=string multiple times.
30. Examine the logs.

```
INFO com.mulesoft.training.Logger: Component fired
INFO com.mulesoft.training.Logger: Component fired
```

Module 6: Implementing Design Patterns



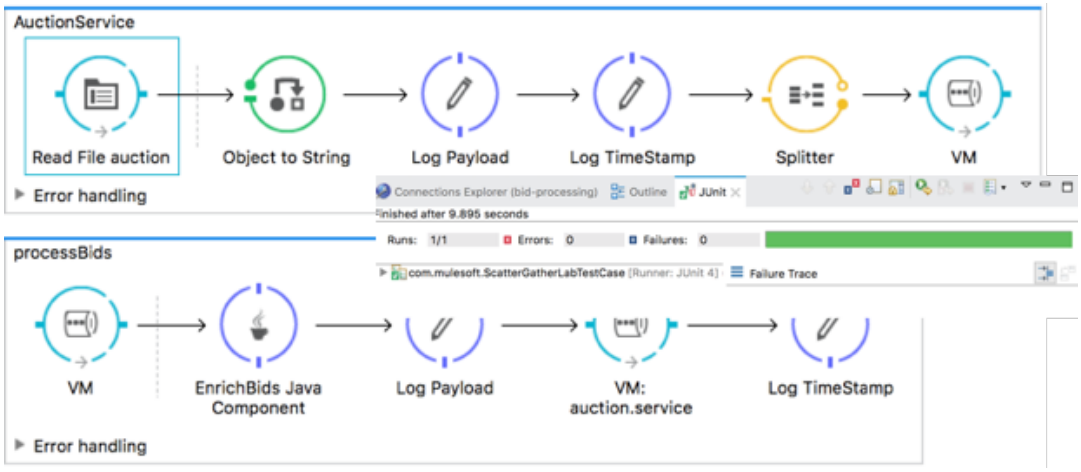
Objectives:

- Enrich data in Mule.
- Split larger payloads into smaller pieces of parallel processing / concurrent design.
- Work with asynchronous patterns.
- Create blocking processes that wait for the completion of asynchronous patterns.

Walkthrough 6-1: Split a Mule message

In this walkthrough, you will:

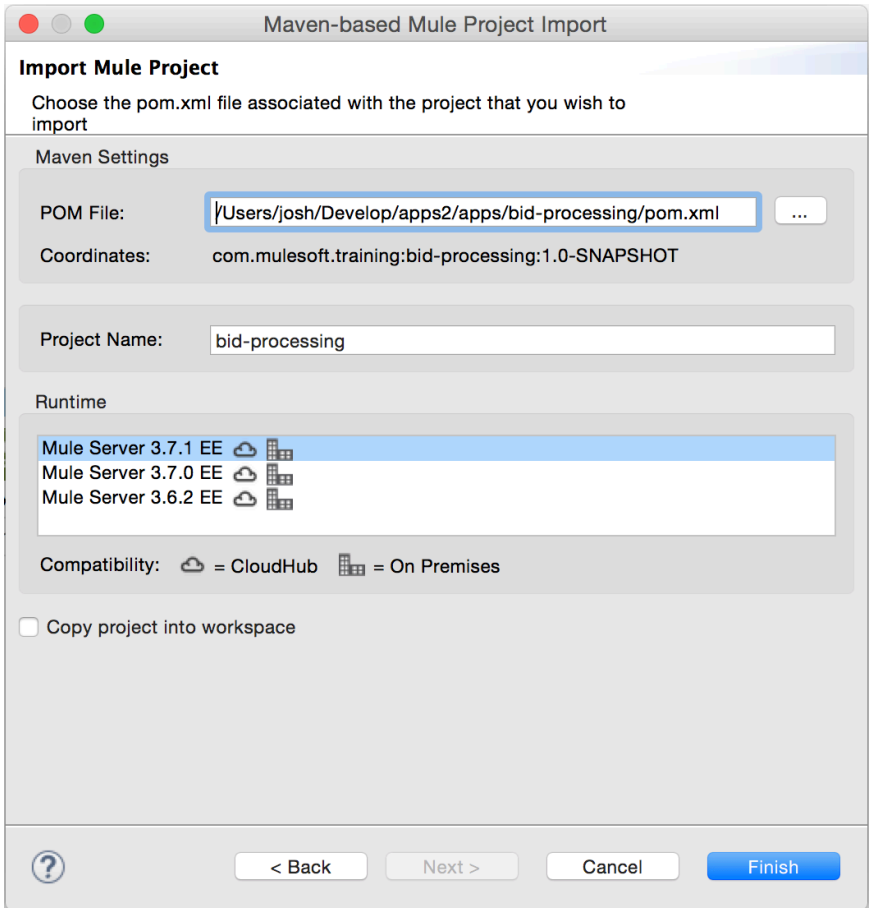
- Examine a linear process that processes bids sequentially.
- Identify a bottleneck in the application.
- Split the original message into smaller messages.
- Process each smaller message individually.



Import the bid system project

1. Return to Anypoint Studio.
2. From the File menu, select Import.
3. Expand Anypoint Studio and select Maven-based Mule Project from pom.xml.
4. Click Next.
5. Browse to studentFiles/apps/bid-processing.
6. Select the pom.xml.

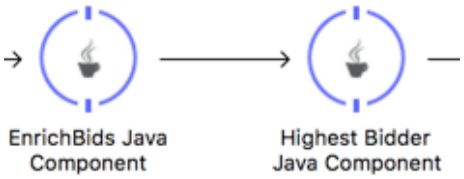
7. Click Open.



8. Select Copy project into workspace.
9. Click Finish.

Examine the components

10. Select the EnrichBids Java component and determine its class name.
11. Select the HighestBidder Java component and determine its class name.



12. Expand src/main/java/com/mulesoft.
13. Open EnricherAuctionComponent.java.

14. Locate Thread.sleep().

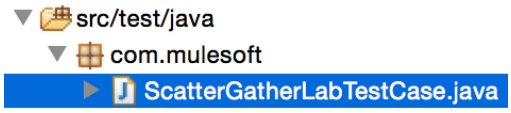
    Note: This line of code simulates an external lookup.

15. In src/main/java/com/mulesoft, open HighBidderAuctionComponent.
16. Examine the for-each loop's purpose.

    Note: Assuming the payload is iterable, this component will locate the highest object based on bid.

Examine the test

17. In the Package Explorer, expand src/test/java/com/mulesoft.



18. Open the ScatterGatherLabTestCase.java.
19. Examine doSetUp(); it creates a new CSV file that is consumed by the flow.
20. Examine testAuctionService(); it listens to the VM queue to which the winning bidder is sent.

Test the project

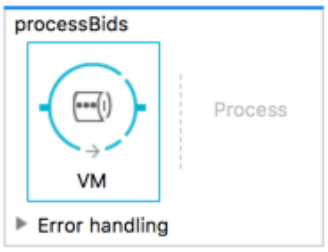
21. Right-click the bid-processing project.
22. Select Run As > JUnit Test.
23. Verify the test passes; record the resulting time.

24. Open mule-config.xml.
25. Examine the project and identify what could enhance its throughput.

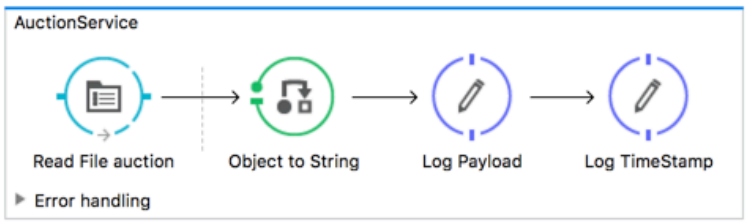
Split the message

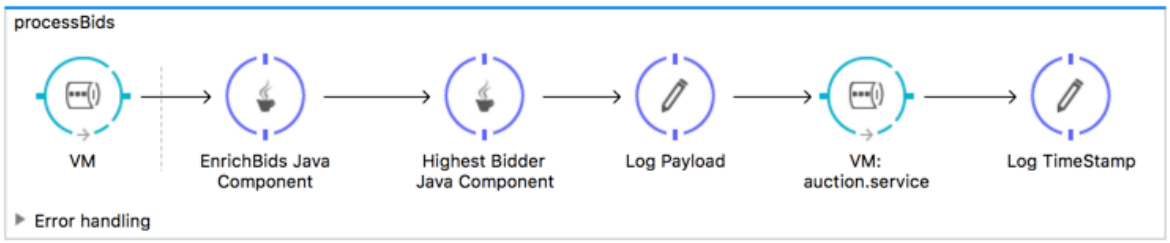
26. Create a new flow named processBids.
27. Add a vm:inbound-endpoint as the message source.

28. Set the queue path to process.



29. Move the following components to processBids and ensure they retain their order.
    - Enrich Bids Java Component
    - Highest Bidder Java Component
    - Log Payload (2nd instance)
    - Log Timestamp (2nd instance)
    - VM: auction.service





30. At the end of the AuctionService flow, add a splitter.
31. Select the splitter.
32. Set the expression to a Groovy script that reads each line of the generated CSV.

```
#[groovy:payload.readLines()]
```

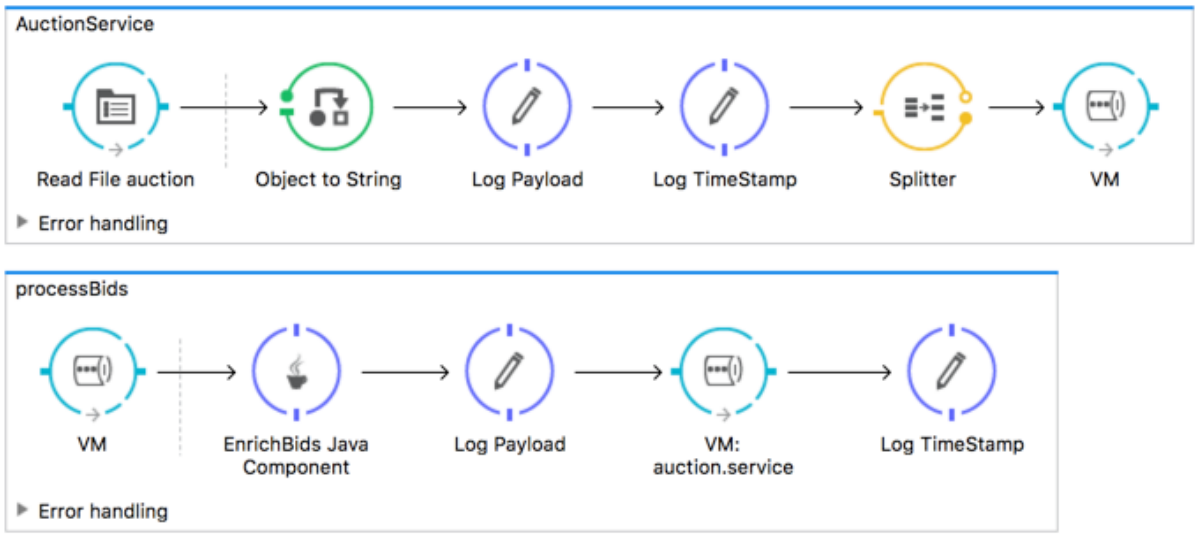
33. From the bottom of the canvas, click the Configuration XML tab.
34. Locate the component referencing the com.mulesoft.HighestBidderAuctionComponent class.
35. Comment out this element.

```
<!-- <component class="com.mulesoft.HighestBidderAuctionComponent" doc:name="Highest
Bidder Java Component"/> -->
```

Note: Commenting out the component ensures your unit test passes. Because you're splitting the message, you no longer have a list of bids to iterate through. You'll revisit this component later.

36. At the end of the AuctionService flow, add a vm:outbound-endpoint.
37. Configure the endpoint to send to the process queue.

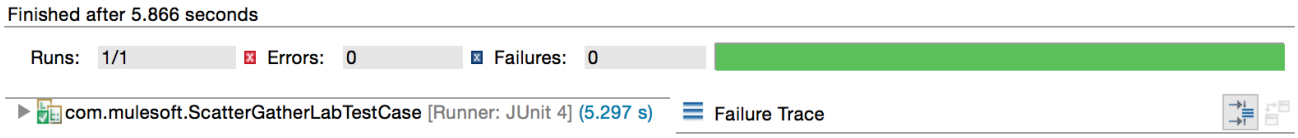
```
<vm:outbound-endpoint exchange-pattern="one-way" path="process" />
```



38. Save the project.

Test the new design

39. Right-click the splitter-aggregator-example project.
40. Select Run As and click JUnit test.
41. Verify the test passes and record the resulting time.



42. Open the console.
43. Locate the InterruptionException.

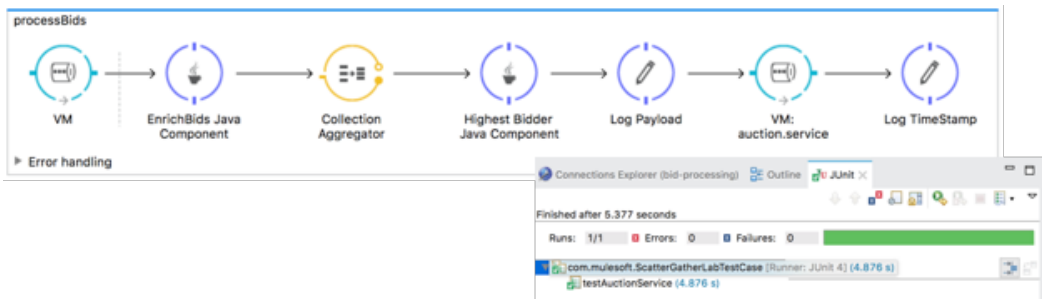
```
Root Exception stack trace:   java.lang.InterruptedException: sleep interrupted   at
java.lang.Thread.sleep(Native Method)
```

44. Examine the flows and determine what caused this exception.
45. Consider what has impacted performance; is your application currently operating in parallel?

Walkthrough 6-2: Aggregate multiple messages

In this walkthrough you will:

- Retrieve related messages.
- Aggregate those messages together.
- Process the collection result.

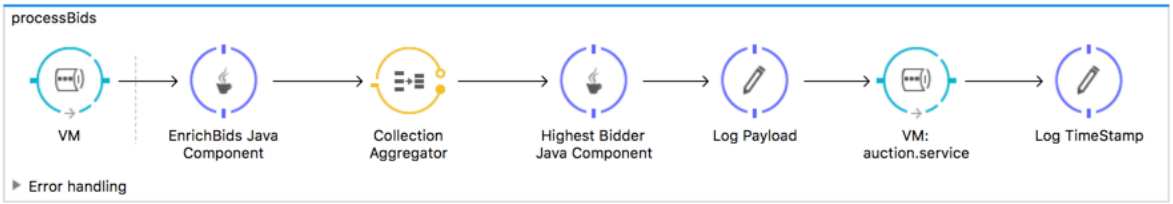


Add aggregation to processBids

1. In the bid-processing project, open mule-config.xml.
2. From the bottom of the canvas, click Configuration XML.
3. Locate the component that is commented out and uncomment it.

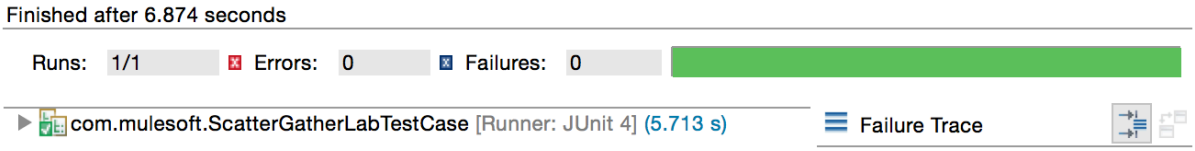
   Note: This component was commented out in the splitter exercise.

4. Add a collection-aggregator before the Highest Bidder Java Component.



Test the project

5. Right-click the bid-processing project.
6. Select Run As > JUnit Test.
7. Verify the test passes and record the resulting time.



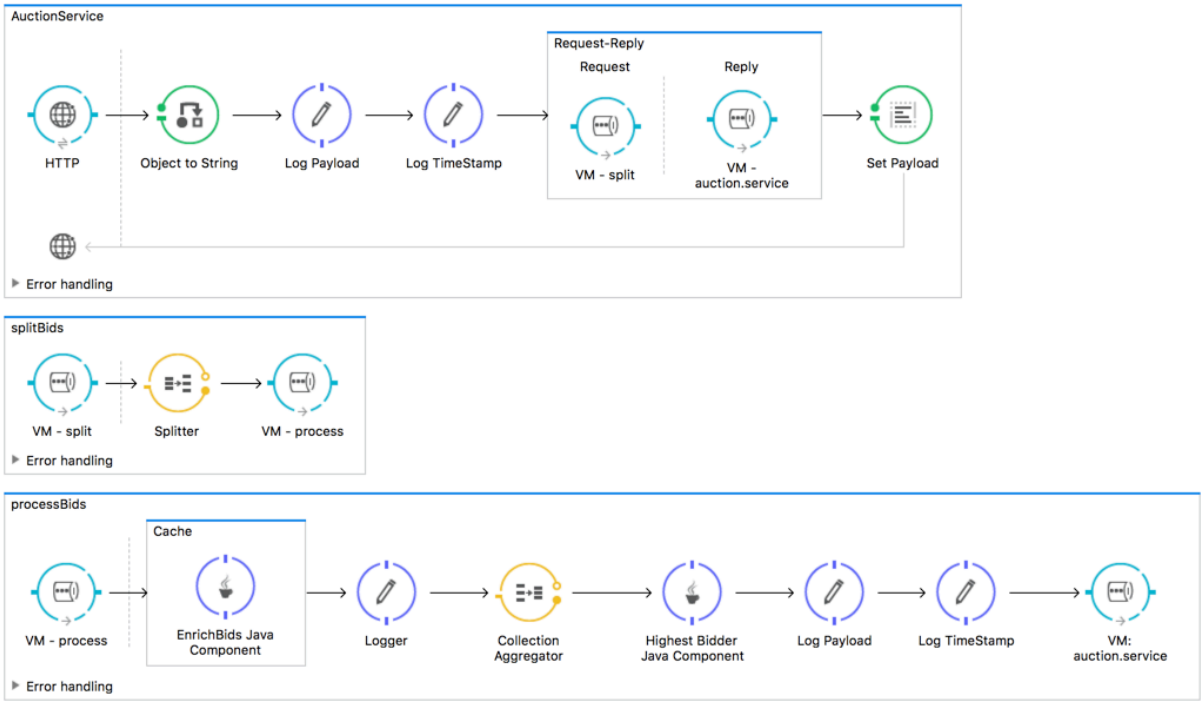
8. Open the console tab and examine the contents.

9. Consider the following:
    - Does the application still experience InterruptedExceptions?
    - Examine the application and determine what has changed.
    - Have these changes had a direct impact on the performance?

Walkthrough 6-3: Create a scalable parallel process

In this walkthrough, you will:

- Accept bid requests through HTTP rather than file.
- Send the http request on to asynchronous, one-way, processes.
- Call back to the original flow when processing is complete.
- Respond to the client with the winning bidder.



Expose the service on HTTP

1. In the bid-processing project, open the mule-config.xml.
2. At the start of AuctionService, remove the file:inbound-endpoint.
3. Add an http:listener to the beginning of the flow.
4. Create a new http:listener-config named HTTPListener8081; expose the listener on 8081.

```
<http:listener-config name="HTTPListener8081" host="0.0.0.0" port="8081" />
```

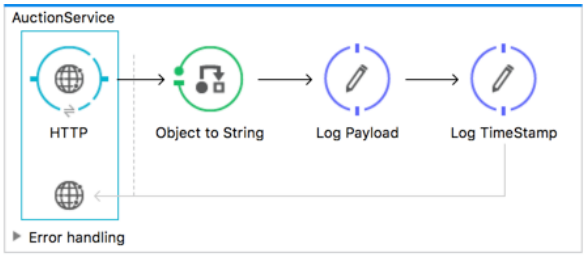
5. Configure the http:listener to be exposed on the path auction.

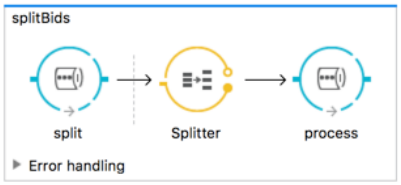
```
<http:listener config-ref="HTTPListener8081" path="/auction" />
```

Create a dedicated-splitter flow

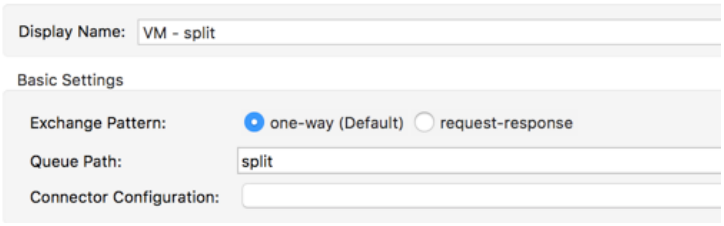
6. Create a new flow named splitBids.

7. Add the splitter and vm:outbound-endpoint from AuctionService into splitBids.



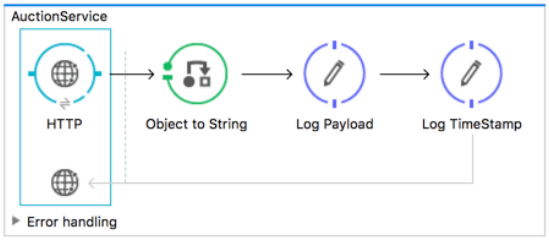


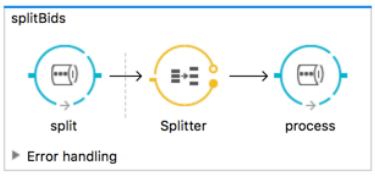
8. At the beginning of splitBids, add a vm:inbound-endpoint.
9. Set the vm:inbound-endpoint to consume off the queue split.

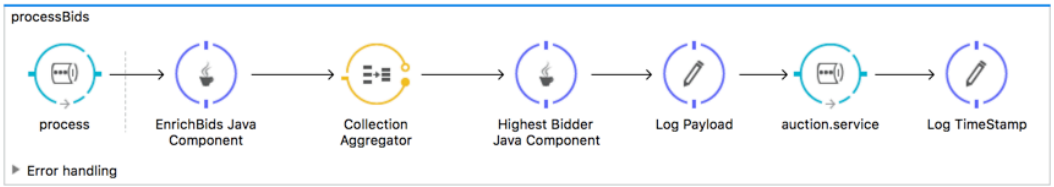


Add request-reply behavior to AuctionService

10. Set the doc:name of every VM endpoint to that of its path.

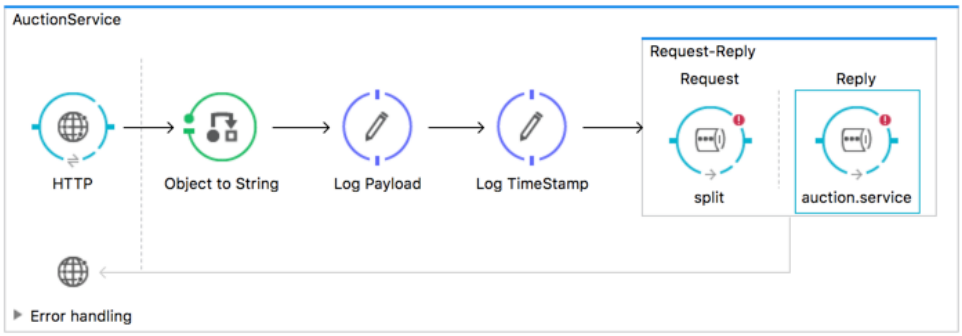






11. At the end of AuctionService, add a request-reply scope.

12. Add a vm:outbound-endpoint to the Request section of the request-reply scope.
13. Rename the endpoint split.
14. Add a vm:inbound-endpoint to the Reply section of the request-reply scope.
15. Rename the endpoint auction.service.



16. Set the path (queue) of the request endpoint to split.
17. Set the path (queue) of the response endpoint to auction.service.

```
<request-reply>
   <vm:outbound-endpoint exchange-pattern="one-way" path="split"/>
   <vm:inbound-endpoint exchange-pattern="one-way" path="auction.service" />
 </request-reply>
```

Note: doc:name is omitted for brevity.

18. At the end of AuctionService, add a set-payload transformer.
19. Set the value to send a json response back to the client containing the highest bidder (payload).

```
#['{ "winner" : "' + payload + '" }']
```

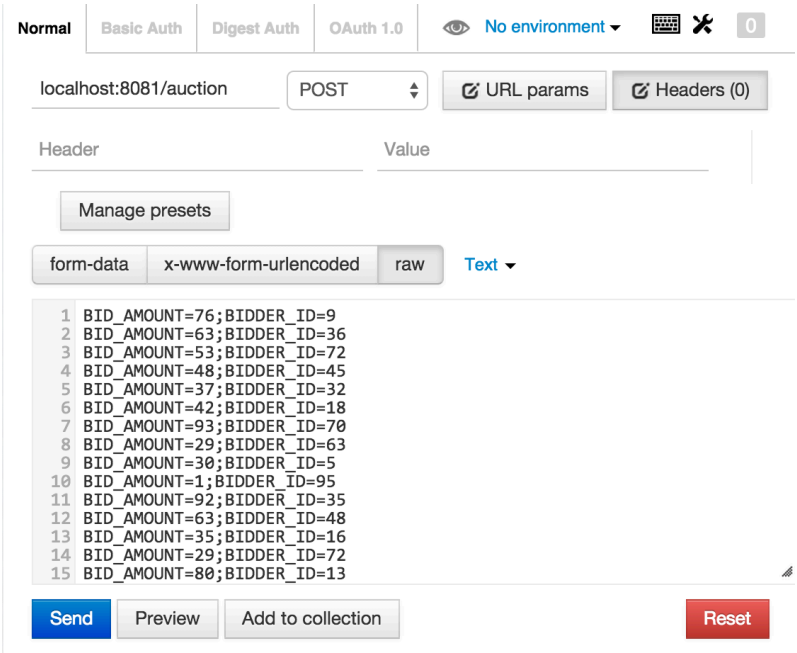
20. Right-click the bid-processing project.
21. Select Run As > Mule Application with Maven (Configure).
22. In the Maven command-line arguments, add a switch to skip tests.

```
-DskipTests
```

Note: Without this, the test would fail and block the build.

23. Click Run and verify the application starts.
24. Open a utility capable of making a rest API request.

25. POST the contents of studentFiles/resources/bids.txt to the Mule endpoint.
    - Example with Postman:



    - Example with curl:

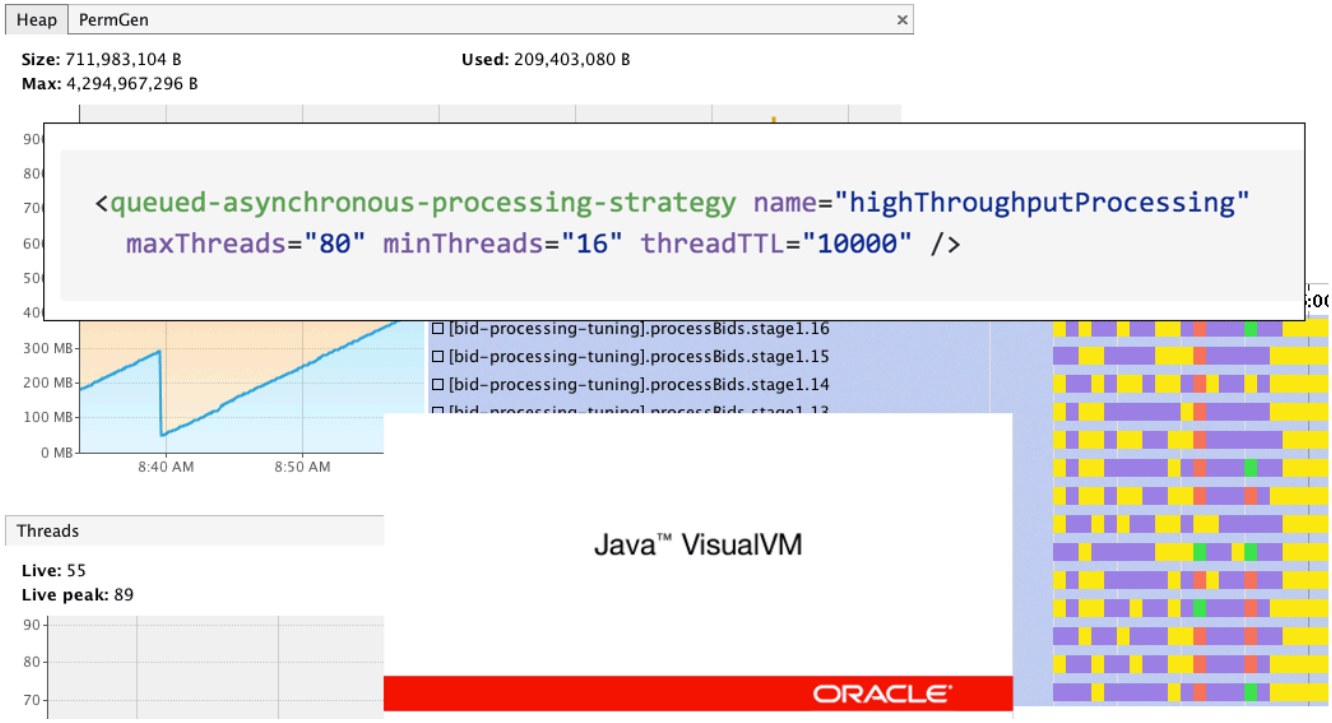
```
curl -X POST \
-d @$MY_FILE_LOCATION/bids.txt \
http://localhost:8081/auction
```

26. Verify the response contains the winning bidder.

```
 {"winner" :
BID_AMOUNT=98;BIDDER_ID=19;BIDDER_NAME=Adlfiu;BIDDER_SURNAME=Gfolzl;BIDDER_EMAIL=Adlf
iu.Gfolzl@mulesoft.com }
```

27. Send a few more requests through to verify functionality.

Module 7: Tuning Application Performance



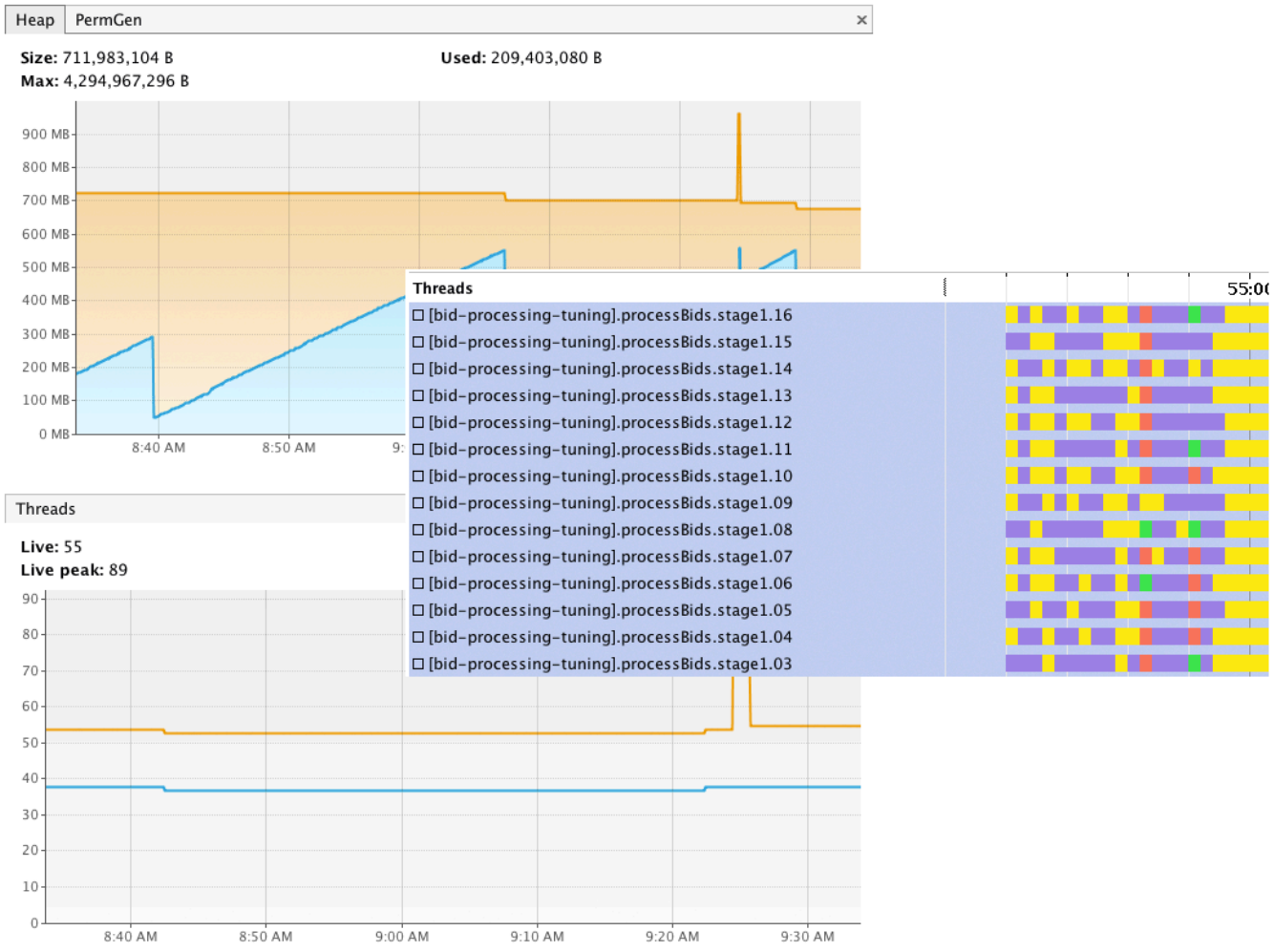
Objectives:

- Examine behaviors of Mule applications through a profiler.
- Alter the threading profile of a Mule application.
- Examine higher throughput through the alteration of profiles.

Walkthrough 7-1: Analyze Mule behavior

In this walkthrough, you will:

- Hook a VM profiler into a running Mule instance.
- Run load through the Mule project.
- Identify the behavior of threads and memory during processing.

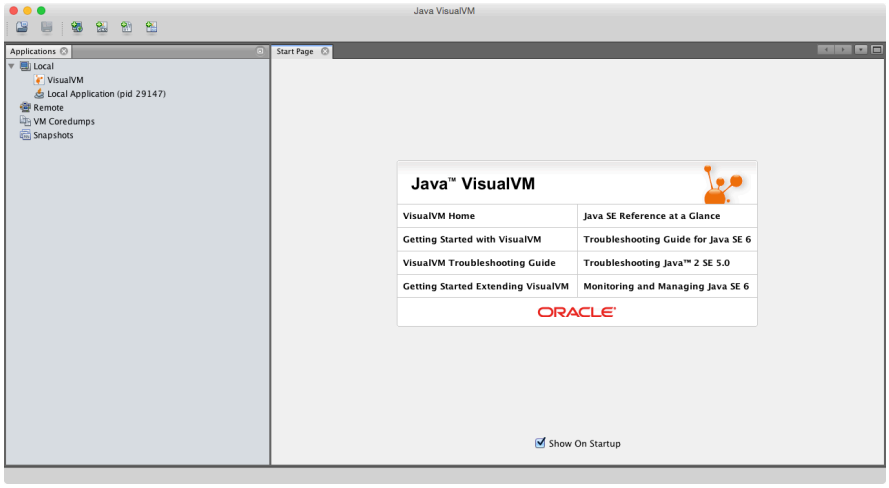


Setup the VM profiler

1. Return to Anypoint Studio.
2. In the console, verify no application is running.
3. Open a command-line interface.

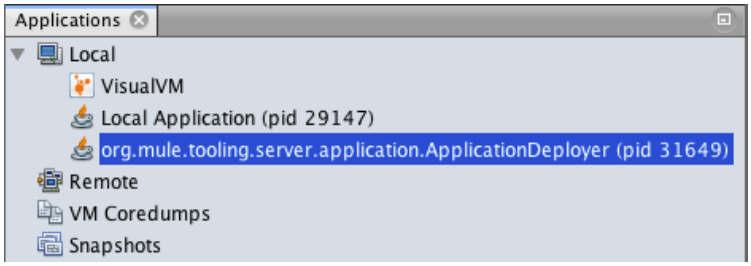
4. Run jvisualvm.

   Note: If the command is not found, the JDK is not part of your path. You can also launch the profiler from $JAVA_HOME/bin.



Examine the VM behavior

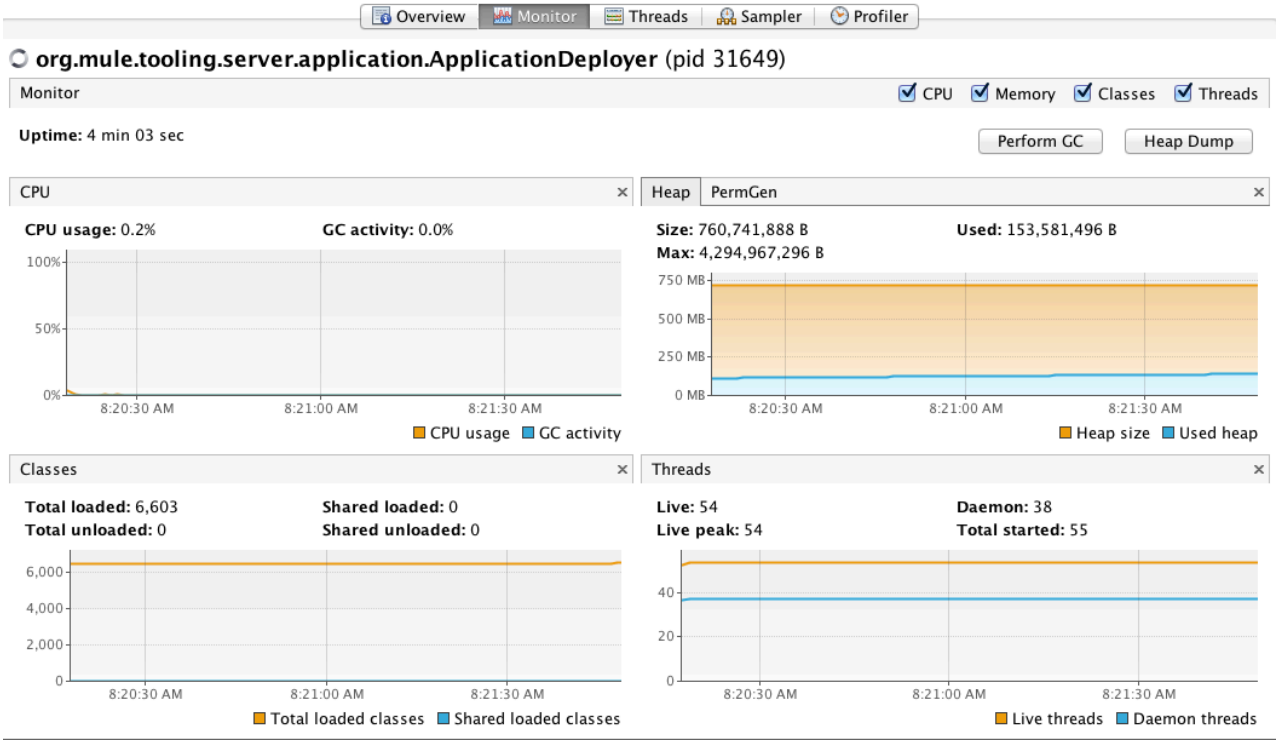
5. Return to Anypoint Studio.
6. Start the bid-processing application.
7. Return to jvisualvm.
8. In the left navigation, double-click the org.mule.tooling.server process.



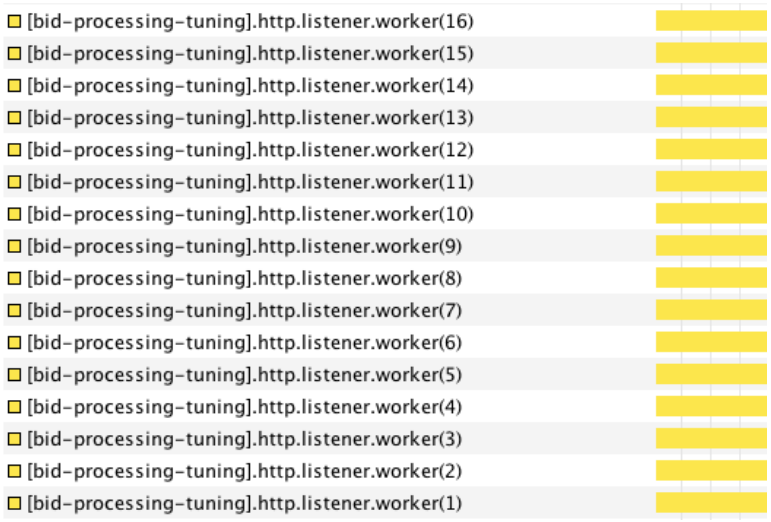
   Note: Your pid will vary.

9. Select the Monitor tab.

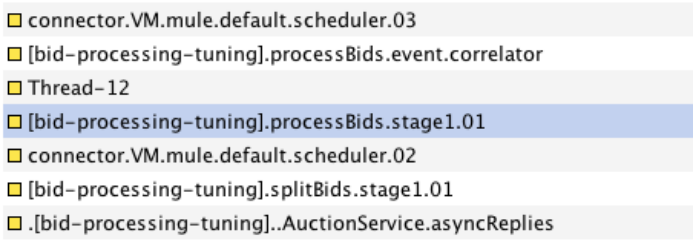
10. Review the current CPU, Heap, and Thread usage.



11. Select the Threads tab.
12. Expand the Threads column to reveal the full thread names.
13. Locate the http.listener.worker threads.
14. Count the quantity of threads available by default.



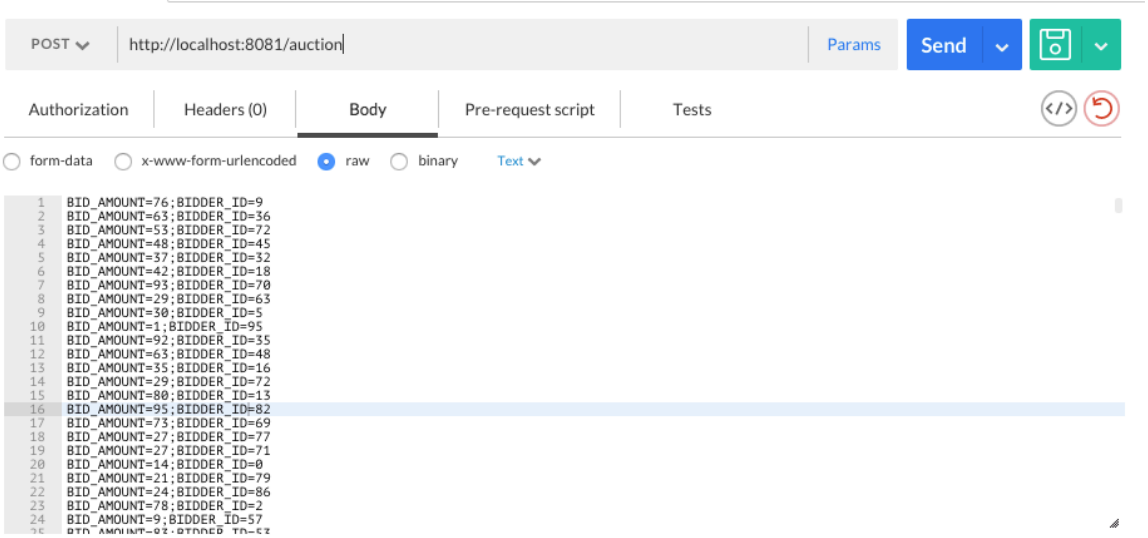
15. Locate any processBids threads and determine the number of threads available by default.



16. Open Postman or another tool capable of making a REST API request.

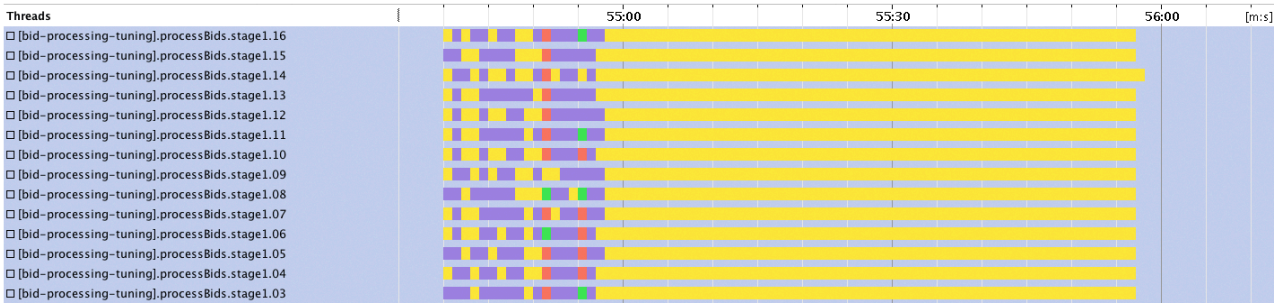
    Note: Prepare to quickly switch back to jvisualvm from this tool.

17. Create a request with the following attributes:
    - url: http://localhost:8081/auction
    - method: POST
    - body: Add the contents of studentFiles/resources/bids_large2.txt



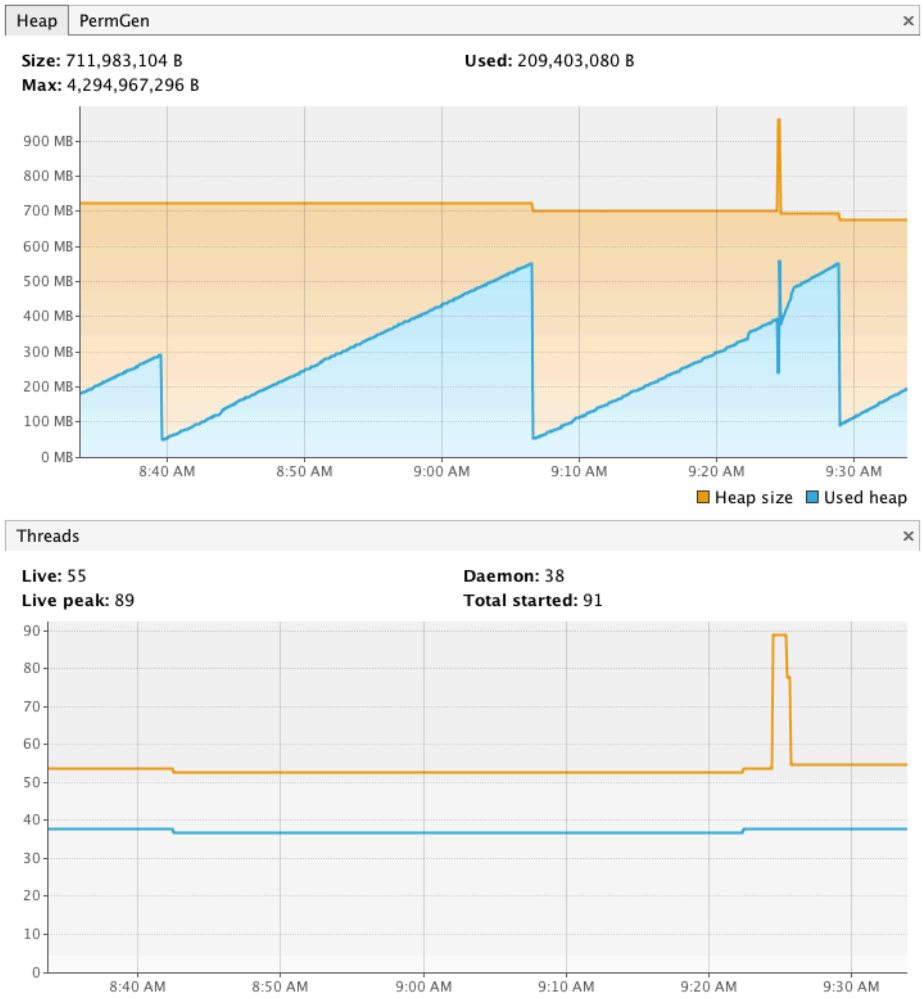
18. Click Send.
19. Return to jvisualvm.
20. Locate the processBids threads.
21. Determine the number of threads created.

22. Watch the processBids threads until they are deconstructed.



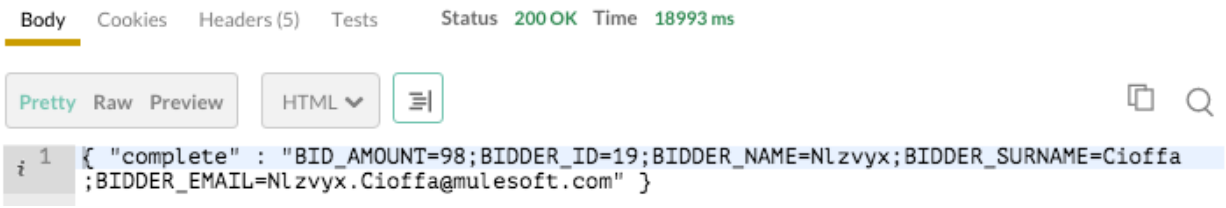
Note: The application ramped up to a total of 16 threads. After 60 seconds of waiting, each thread was deconstructed.

23. Locate the processBids thread that remains alive.
24. Select the Monitor tab.
25. Examine the impact the load had on the threads and heap.



26. Close the org.mule.tooling tab.

27. Return to Anypoint Studio.
28. From the console, stop the Mule runtime.
29. Return to your REST API tool.
30. Examine the response time.

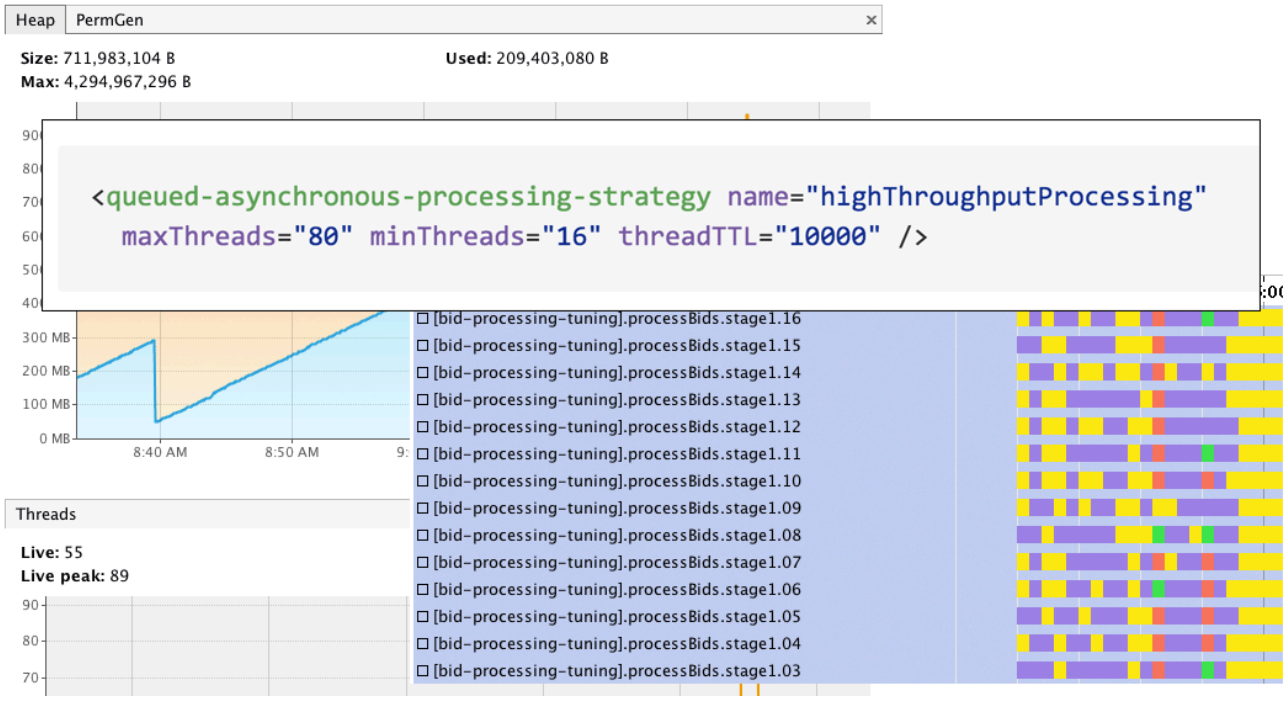


Note: Remember or jot down this time. You will compare it to the response in the next walkthrough.

Walkthrough 7-2: Alter the threading profile

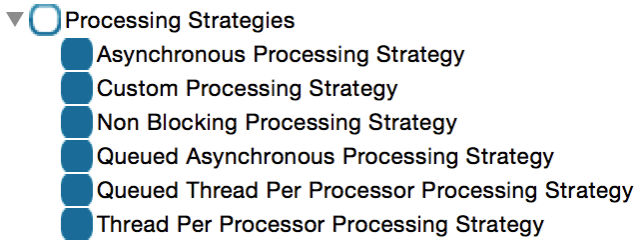
In this walkthrough you will:

- Define a custom threading profile.
- Reference a threading profile from a flow.
- Analyze the profile's impact on the application.



Create a processing strategy

1. Open Anypoint Studio and return to the bid-processing application.
2. In mule-config.xml, open Global Elements.
3. Click Create.
4. Expand Processing Strategies.



5. Choose Queued Asynchronous Processing Strategy.
6. Name the strategy highThroughputProcessing

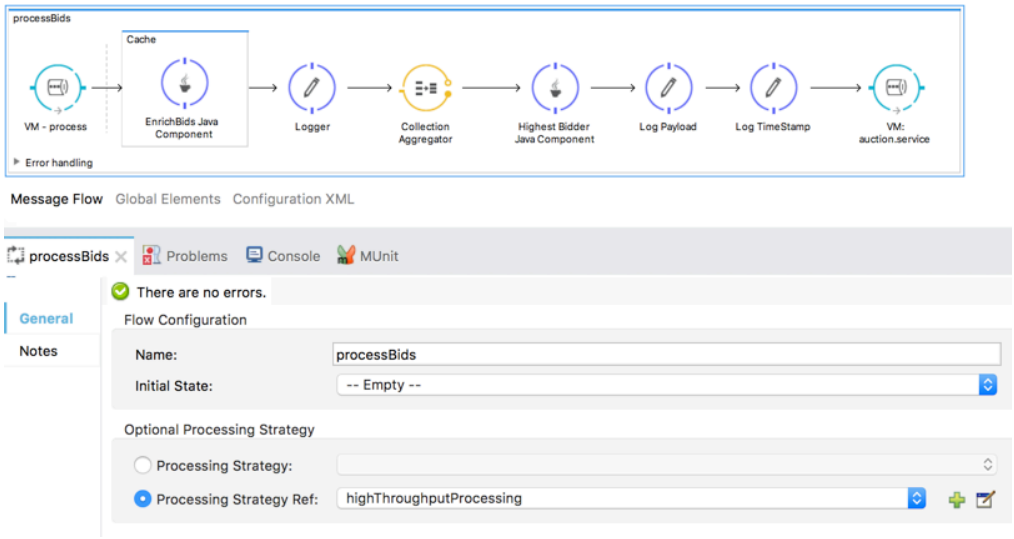
7. Set the following threading settings:
    - Max Threads: 80
    - Min Threads: 16
    - Thread TTL: 10000

```
<queued-asynchronous-processing-strategy name="highThroughputProcessing"
    maxThreads="80" minThreads="16" threadTTL="10000" />
```

8. Click OK.

Reference the strategy

9. Select the processBids flow.
10. Select Processing Strategy Ref.
11. From the drop-down list, select highThroughputProcessing.



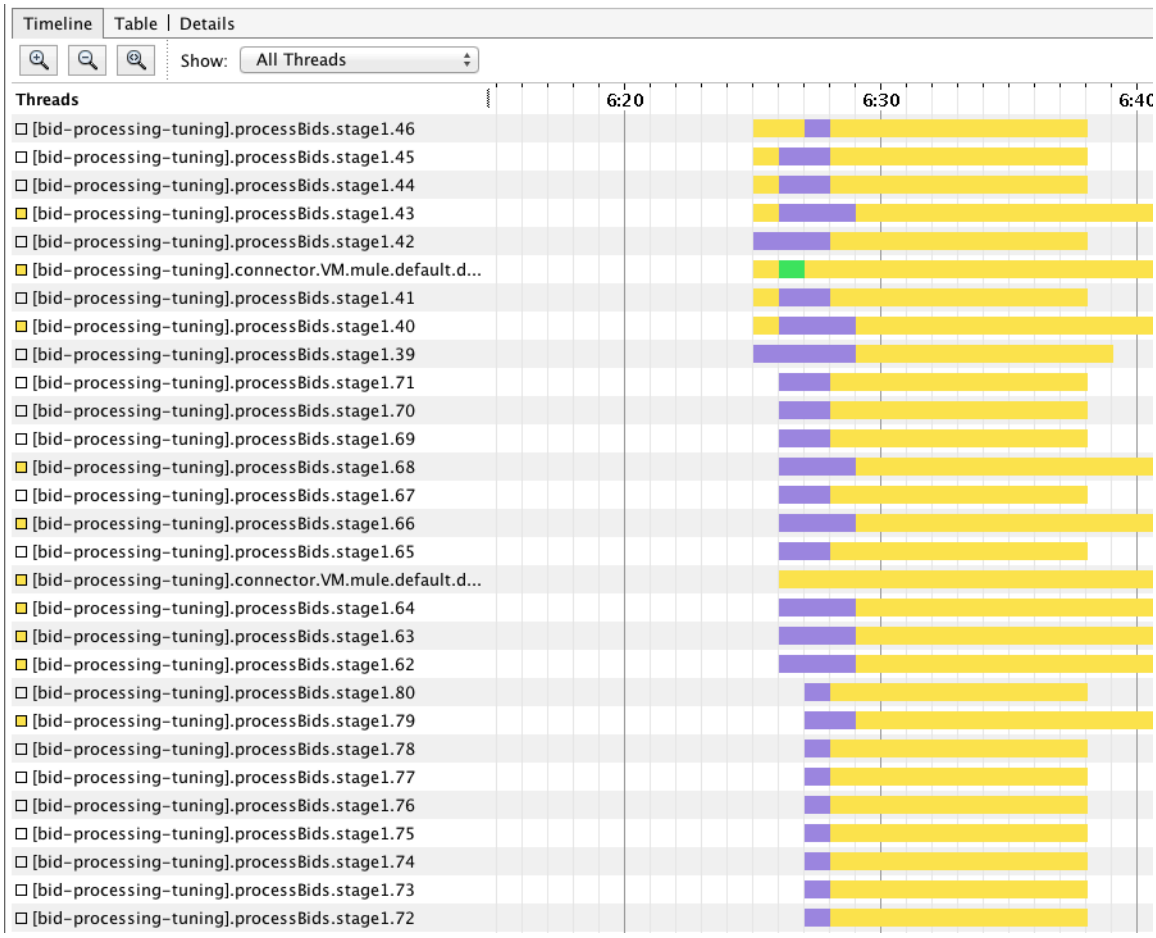
12. Save and run the application.

Profile the results

13. Open jvisualvm.
14. Double-click the new org.mule.tooling process on the left.
15. Select the Threads tab.
16. Locate the processBids thread.
17. Open your REST API tool.
18. Send another POST request through with the following configuration:
    - url: http://localhost:8081/auction

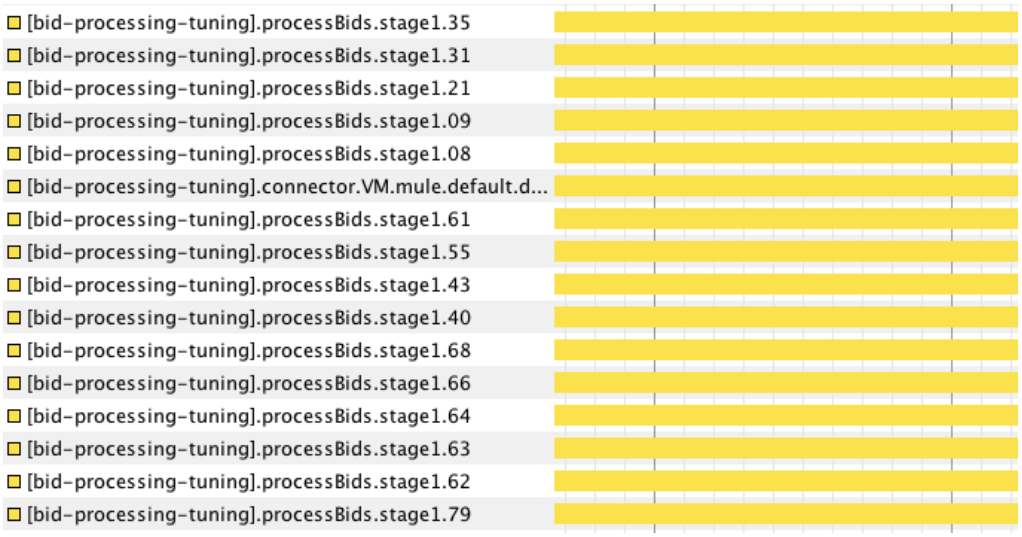
- method: POST
- body: Add the contents of studentFiles/resources/bids_large.txt

19. Return to jvisualvm.
20. Locate the newly created processBids threads.

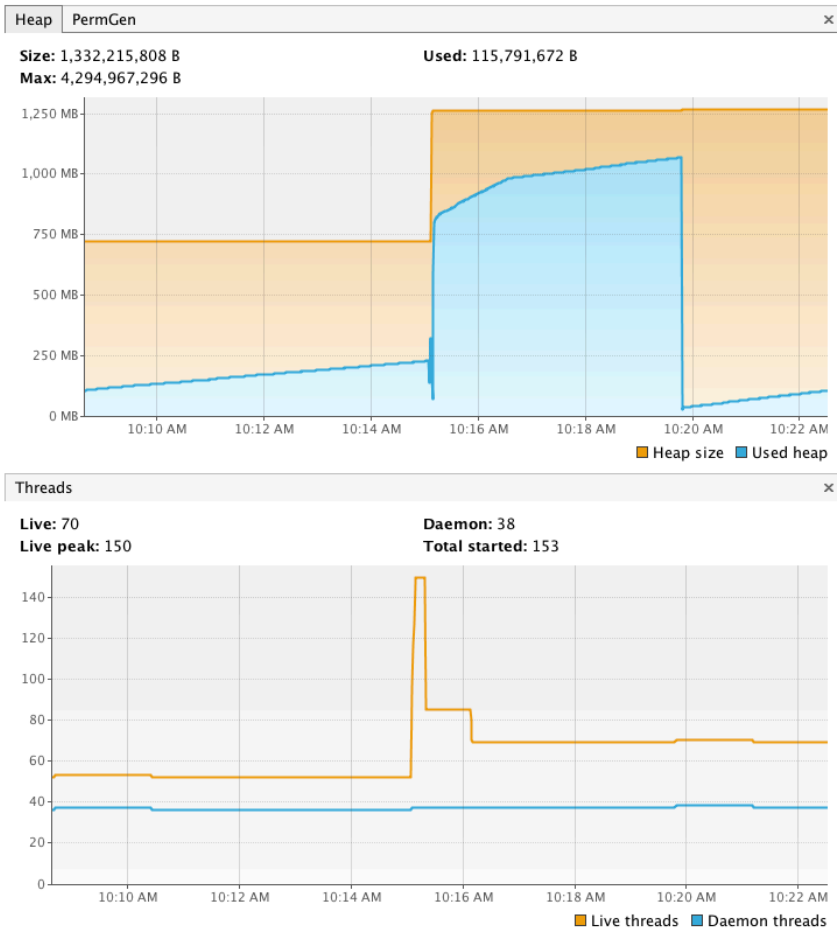


21. Examine the new threads as they process and are eventually deconstructed.

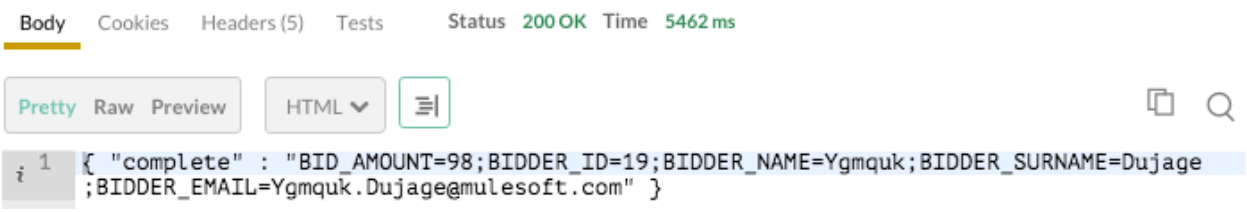
22. From the Show drop-down list, select Live Threads Only.



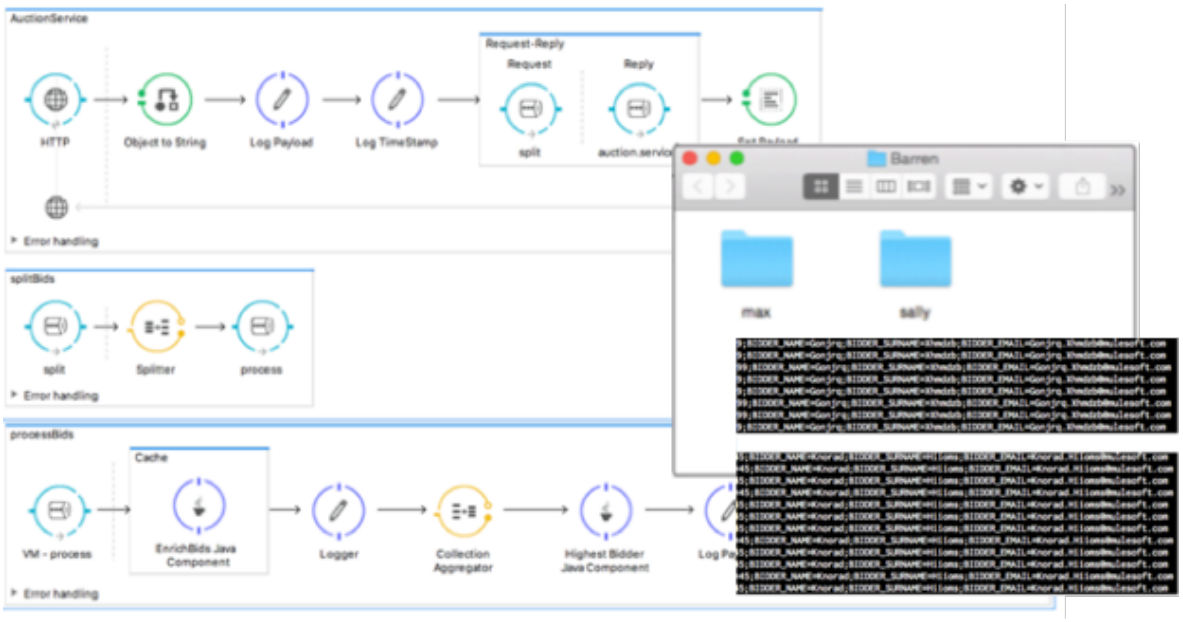
23. Determine how many processBids threads remain.
24. Select the Monitor tab.
25. Examine the heap and thread impact.



26. Return to your REST API tool and examine the response time.



Module 8: Working with State



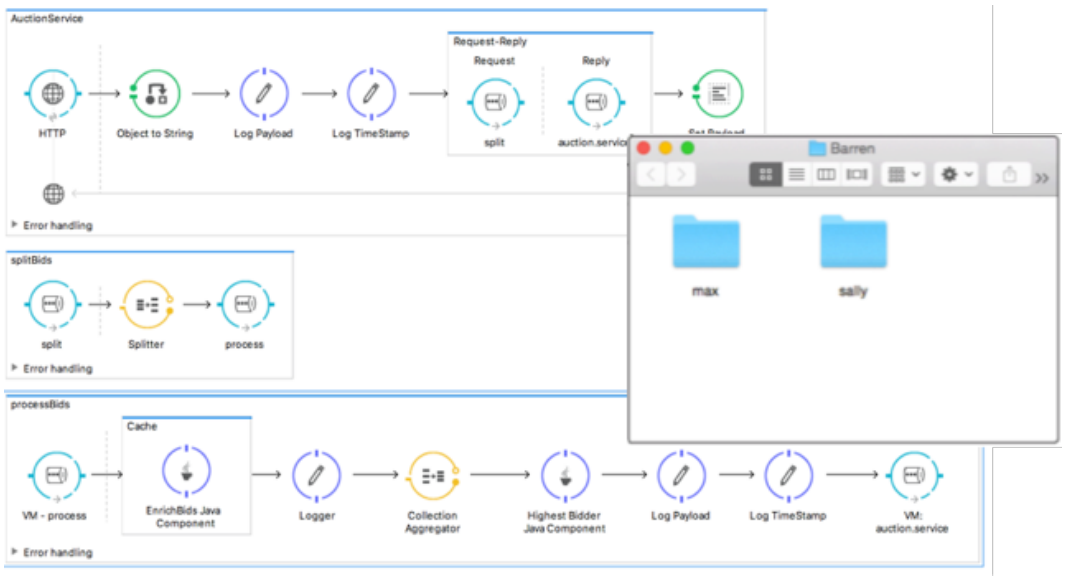
Objectives:

- Make design choices based on your Mule environment.
- Speak to the impact clustering has on an application.
- Implement caching on an external call.
- Configure a custom object store.

Walkthrough 8-1: Run the application in a cluster

In this walkthrough, you will:

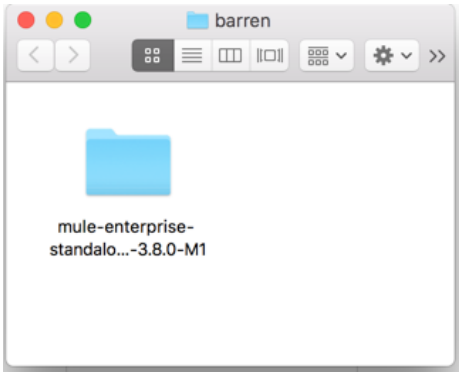
- Run two Mules in a cluster.
- Deploy the bid-processing application to the cluster.
- Examine the processing of bids.



Startup the servers

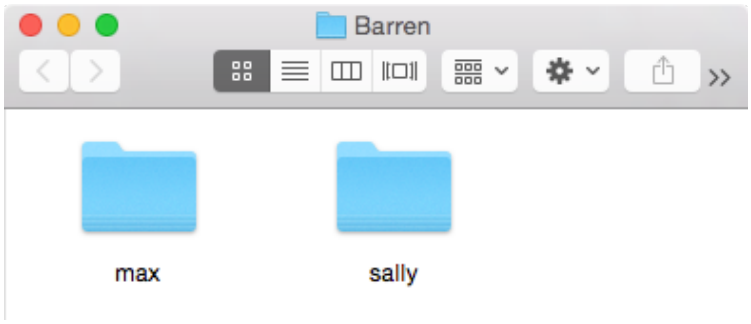
1. Create a folder on your computer named Barren.
2. Extract mule-ee-distribution-standalone-{version} inside of Barren.

   Note: The mule-ee-distribution-standalone download is listed in the setup section of this manual.



3. Duplicate the mule-enterprise-standalone folder

4. Rename the two folders:
    - max
    - sally



5. Start max and sally by running $MULE_HOME/bin/mule in each.

```
-rw-r--r--@  1       staff   518B Aug  2 15:55 LICENSE.txt
-rw-r--r--@  1       staff    27K Aug  2 15:55 MIGRATION.txt
-rw-r--r--@  1       staff   3.7K Aug  2 14:53 README.txt
drwxr-xr-x@  3       staff   102B Aug  2 14:53 apps
drwxr-xr-x@ 13       staff   442B Aug  4 11:23 bin
drwxr-xr-x@ 10       staff   340B Aug  2 15:55 conf
drwxr-xr-x@  4       staff   136B Aug  4 11:23 docs
drwxr-xr-x@  3       staff   102B Aug  2 14:53 domains
drwxr-xr-x@  3       staff   102B Aug  2 15:55 examples
drwxr-xr-x@ 11       staff   374B Aug  2 15:55 lib
drwxr-xr-x@  3       staff   102B Aug  4 11:23 logs
drwxr-xr-x@  4       staff   136B Aug  2 15:59 plugins
drwxr-xr-x@  3       staff   102B Aug  2 15:55 src
drwxr-xr-x@  3       staff   102B Aug  2 15:59 tools
mule$ ./bin/mule                                        !10062
MULE_HOME is set to /Users/Documents/Barren/max
Running in console (foreground) mode by default, use Ctrl-C to exit
...
MULE_HOME is set to /Users/Documents/Barren/max
Running Mule Enterprise Edition...
--> Wrapper Started as Console
Java Service Wrapper Standard Edition 32-bit 3.5.26
  Copyright (C) 1999-2014 Tanuki Software, Ltd. All Rights Reserved.
    http://wrapper.tanukisoftware.com
  Licensed to mulesoft.com for Mule ESB Enterprise Edition

Launching a JVM...
```

```
-rw-r--r--@  1       staff   518B Aug  2 15:55 LICENSE.txt
-rw-r--r--@  1       staff    27K Aug  2 15:55 MIGRATION.txt
-rw-r--r--@  1       staff   3.7K Aug  2 14:53 README.txt
drwxr-xr-x@  3       staff   102B Aug  2 14:53 apps
drwxr-xr-x@ 13       staff   442B Aug  4 11:23 bin
drwxr-xr-x@ 10       staff   340B Aug  2 15:55 conf
drwxr-xr-x@  4       staff   136B Aug  4 11:23 docs
drwxr-xr-x@  3       staff   102B Aug  2 14:53 domains
drwxr-xr-x@  3       staff   102B Aug  2 15:55 examples
drwxr-xr-x@ 11       staff   374B Aug  2 15:55 lib
drwxr-xr-x@  3       staff   102B Aug  4 11:23 logs
drwxr-xr-x@  4       staff   136B Aug  2 15:59 plugins
drwxr-xr-x@  3       staff   102B Aug  2 15:55 src
drwxr-xr-x@  3       staff   102B Aug  2 15:59 tools
mule$ ./bin/mule
MULE_HOME is set to /Users/Documents/Barren/sally
Running in console (foreground) mode by default, use Ctrl-C to exit...
MULE_HOME is set to /Users/Documents/Barren/sally
Running Mule Enterprise Edition...
--> Wrapper Started as Console
Java Service Wrapper Standard Edition 32-bit 3.5.26
  Copyright (C) 1999-2014 Tanuki Software, Ltd. All Rights Reserved.
    http://wrapper.tanukisoftware.com
  Licensed to mulesoft.com for Mule ESB Enterprise Edition

Launching a JVM...
```

6. Type Ctrl+C in the Mule terminal windows.

    Note: This has ensured the .mule directory was created.

Configure the cluster

7. Open the .mule directory in sally and max.
8. From the studentFiles/resources directory, copy mule-cluster.properties into max and sally's .mule directory of both Mules.
9. Edit one of the Mule's mule-cluster.properties.

10. Set the clusterNodeID to 1.

```
# Mule cluster properties
mule.clusterSize=2
mule.clusterSchema=partitioned-sync2backup
mule.clusterId=11223
mule.cluster.multicastenabled=true
mule.clusterPartitioningMode=OPTIMIZE_PERFORMANCE
mule.clusterNodeId=1
```

Start the cluster

11. In a command-line interface, return to max and sally's home directory.
12. Start both Mules by triggering $MULE_HOME/bin/mule and specifying a http.port.

```
$MAX_HOME/bin/mule -M-Dhttp.port=8081

$SALLY_HOME/bin/mule -M-Dhttp.port=8082
```

Note: This prevents port conflicts and lets you call either Mule.

13. In the logs, locate the two Member arrays.

```
Members [2] {
  Member [172.16.12.224]:5702 this
  Member [172.16.12.224]:5701
}
```

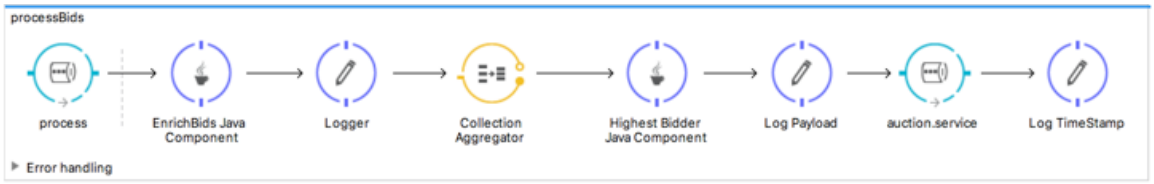
```
Members [2] {
  Member [172.16.12.224]:5701 this
  Member [172.16.12.224]:5702
}
```

Note: This implies the Mules are in a cluster.

Deploy the application to a cluster

14. Open Anypoint Studio.
15. In the bid-processing application, open mule-app.properties.
16. If the property http.port exists, delete it.
17. Open mule-config.xml.
18. Click the Global Elements tab.
19. Open http:listener-config.

20. If not already, set the port value to ${http.port}.
21. Click OK and return to Message Flow.
22. In the processBids flow after the EnrichBids component, drop a new logger.



23. Configure the logger to log the Node's ID and bidder's details.

```
<logger message="#[mule.nodeId + ' Processed: ' + message.payload]" level="INFO"/>
```

Note: nodeId displays that cluster node is doing the processing.

24. Save the application.
25. Open the project in a command-line interface.
26. Package the application using Maven and skip all tests.

```
mvn package
```

Note: If test case(s) are still attached to the project, include the -DskipTests argument.

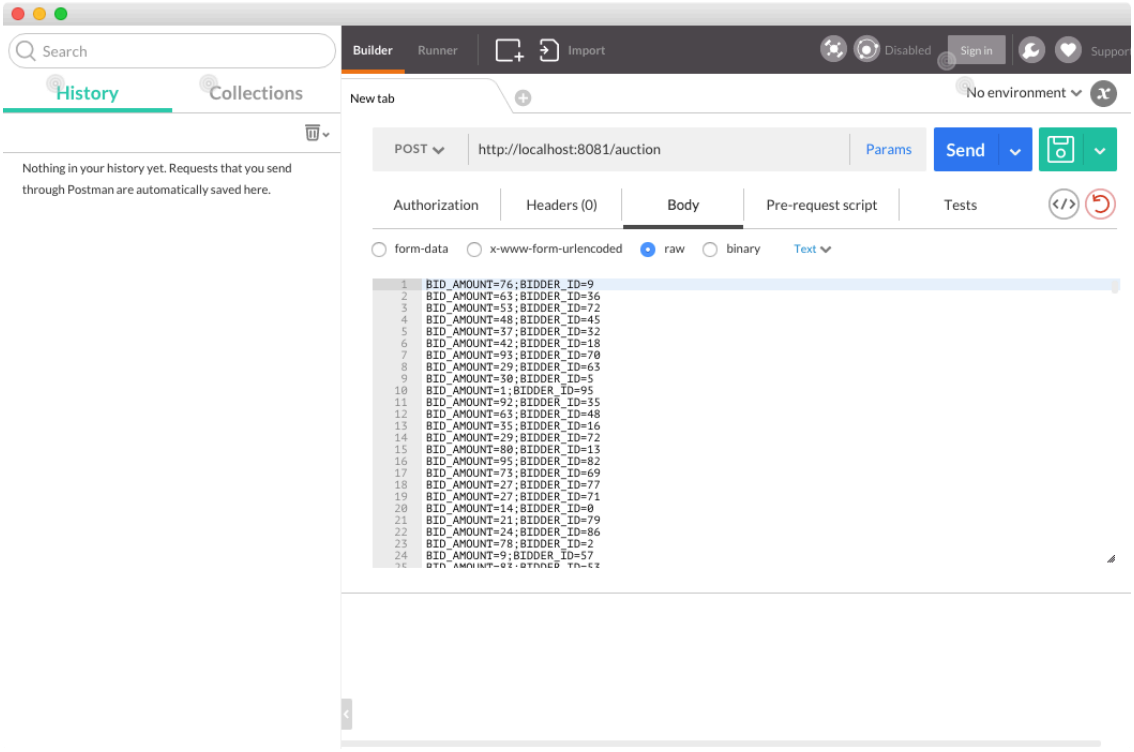
27. From the target folder, copy the deployable into max and sally's $MULE_HOME/apps directory.
28. Verify the application deploy successfully on both Mules.

```
* bid-processing-distributed-1.0.0-SNAPSHOT     * default     * DEPLOYED
```

Make a request

28. Open a tool capable of making REST API requests.

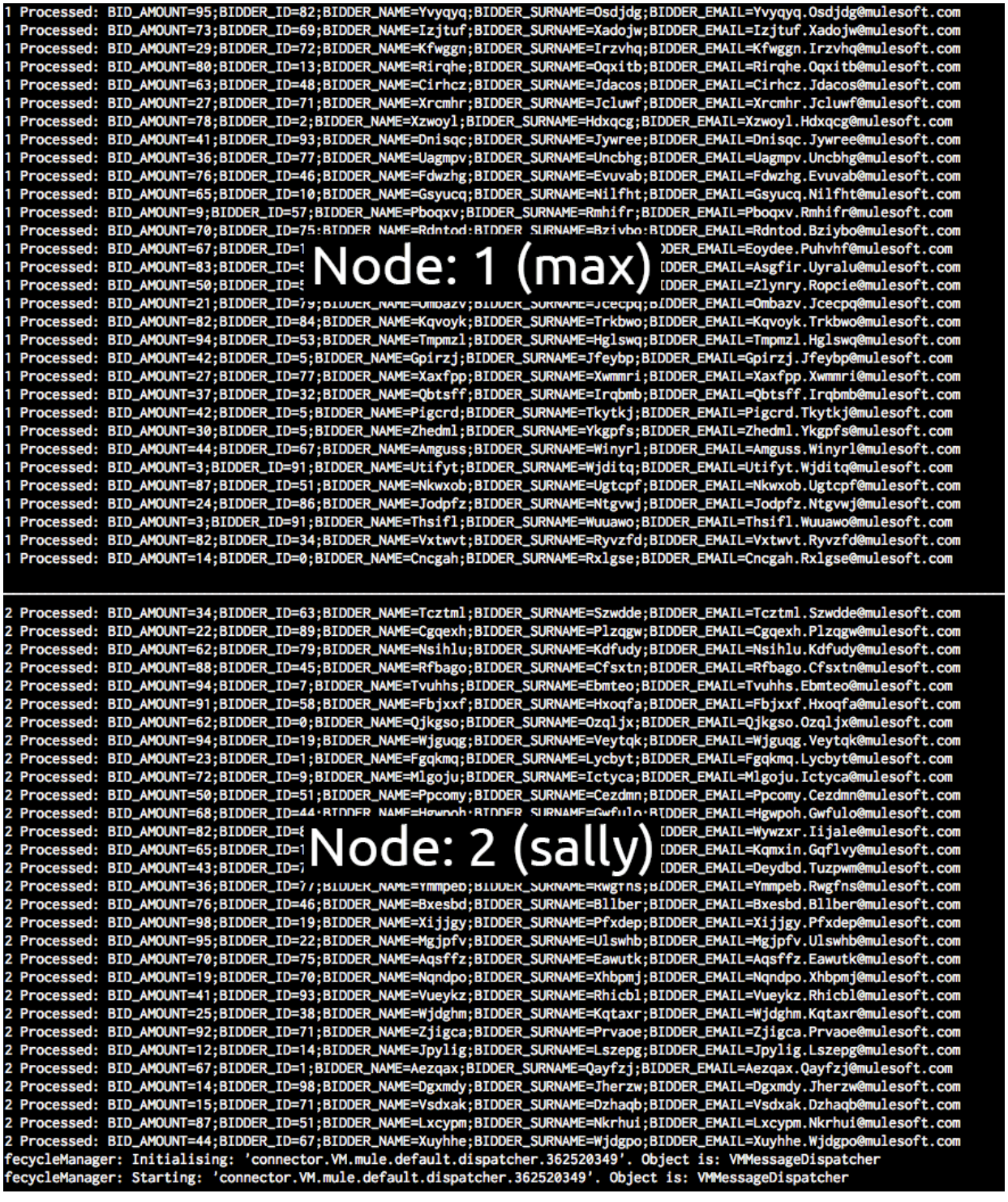
29. Create a request with the following attributes:
    - url: http://localhost:8081/auction
    - method: POST
    - body: add the contents of studentFiles/resources/bids_large.txt



Examine the logs

30. Open the logs directory of max and sally.
31. Open the mule-app-bid-processing logs for both Mules.

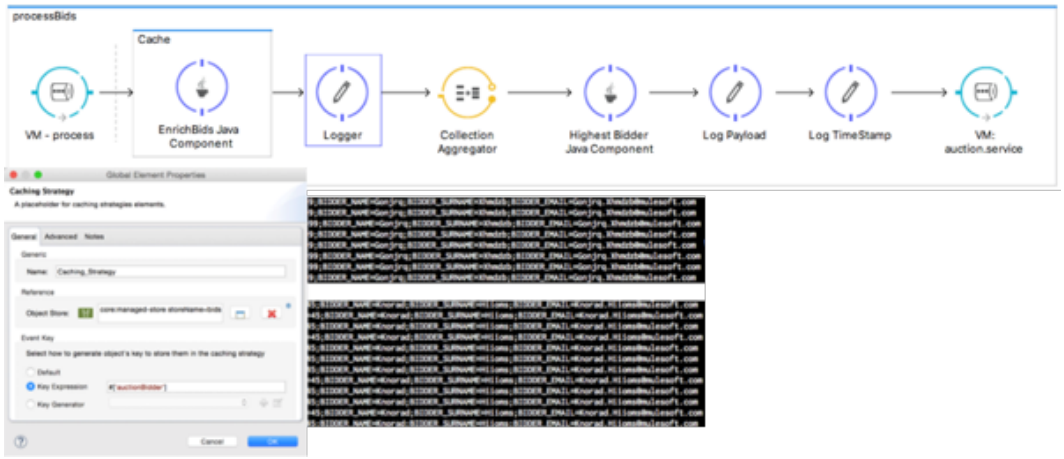
32. Examine the distribution of processing between nodes.



Walkthrough 8-2: Cache a processor's response

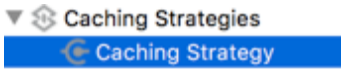
In this walkthrough you will:

- Configure a cache for the use of cluster nodes.
- Ensure the cache is shared amongst nodes.
- Expire the cache after 5 seconds.

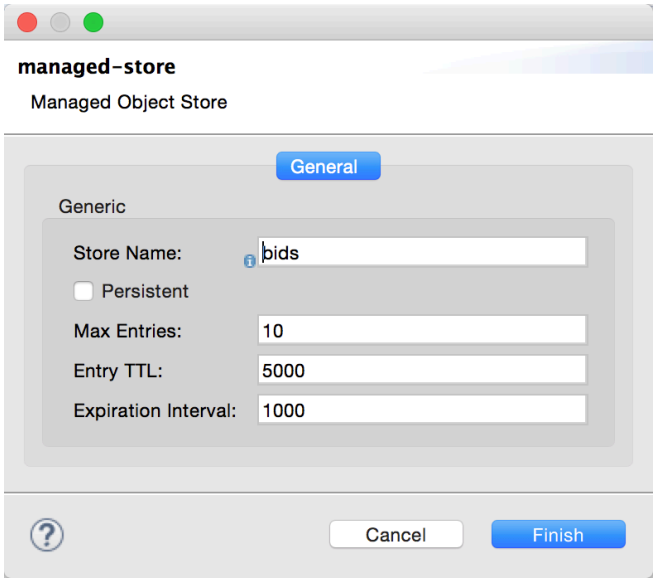


Create the cache

1. Return to Anypoint Studio.
2. In the bid-processing application, open mule-config.xml.
3. Open Global Elements.
4. Click Create.
5. Select Caching Strategies > Caching Strategy.



6. In Object Store, click create.
7. Select managed-store.
8. Set the following attributes:
   - Store Name: bids
   - Max Entries: 10
   - Entry TTL: 5000
   - Expiration Interval: 1000



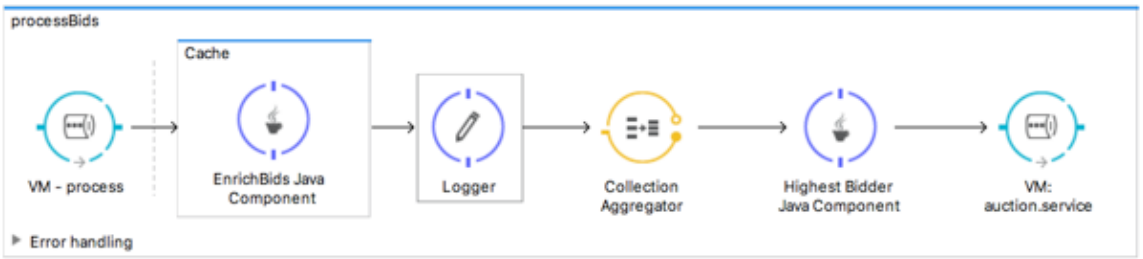
9. Click Finish.
10. In the Event Key, select Key Expression.
11. Set the expression to a static key named auctionBidder.

```
#['auctionBidder']
```

12. Click OK.

Cache the enrichment processor

13. Select Message Flow.
14. In the processBids flow, add an ee:cache scope.
15. Place the EnrichAuctionComponent inside the ee:cache scope.
16. Select the ee:cache scope.
17. Set the caching strategy to Caching_Strategy.



18. Save the application.

Deploy the application

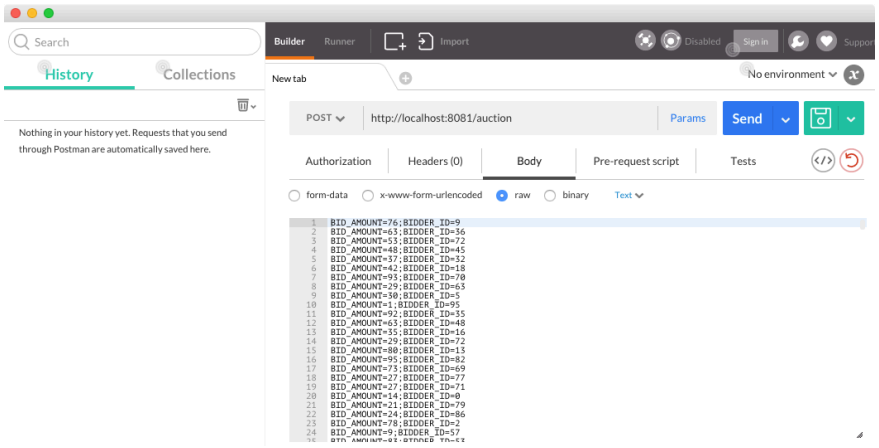
19. In a command-line interface, open the bid-processing project.
20. Package the application and skip tests.

```
mvn package -DskipTests
```

21. If max and sally aren't still running from the previous walkthrough, start them.
22. From the bid-processing/target directory, copy the deployable to max and sally's $MULE_HOME/apps directory.
23. Verify the application deploy's successfully.

Make a request

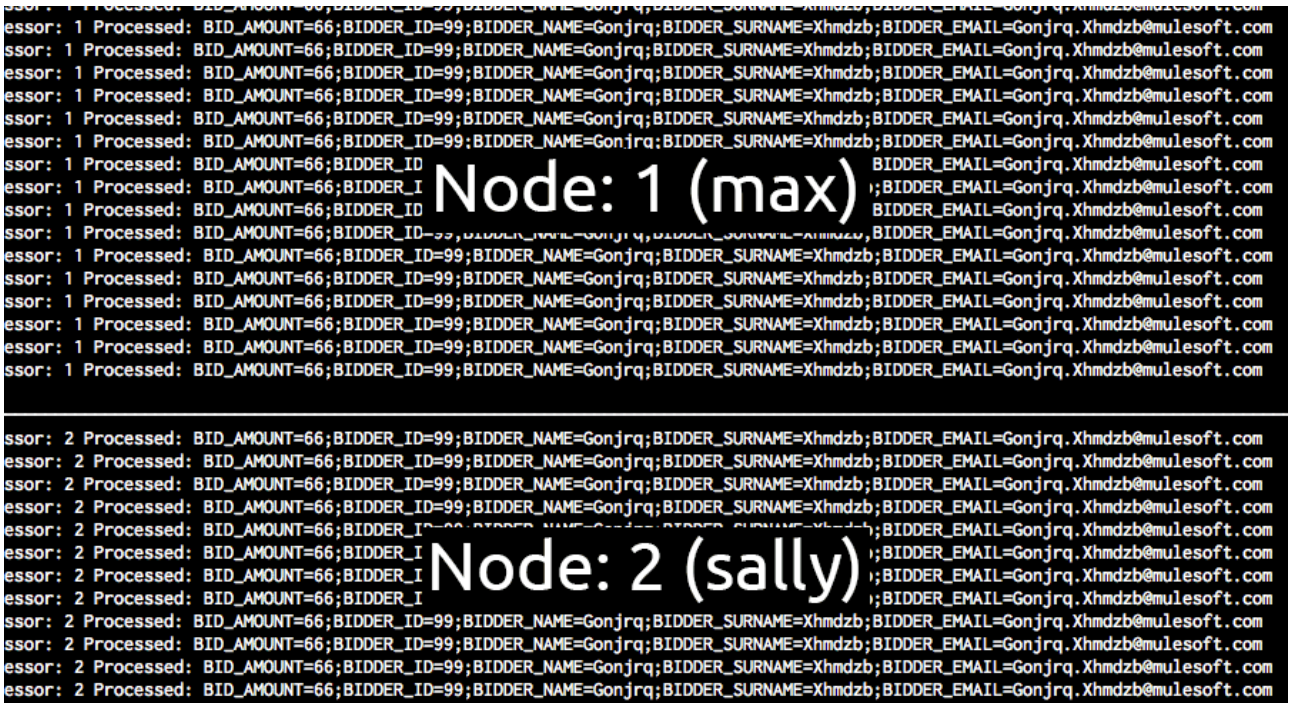
24. Open your REST API tool.
25. Send another POST request through with the following configuration:
    - url: http://localhost:8081/auction
    - method: POST
    - body: add the contents of studentFiles/resources/bids_large2.txt



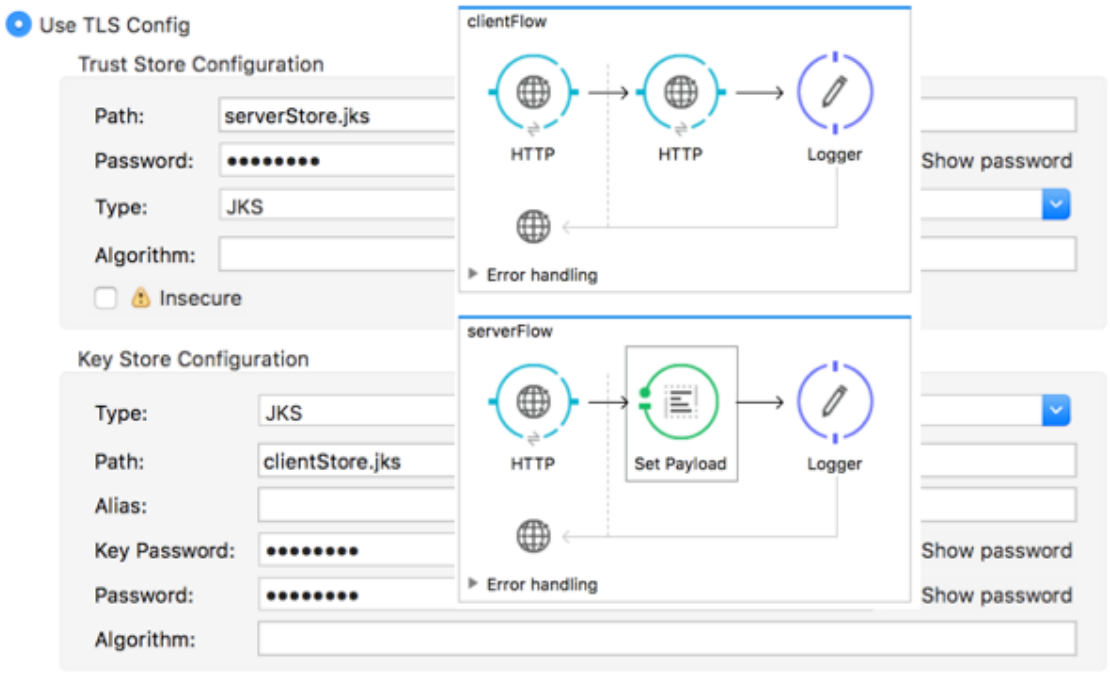
Examine the logs

26. Open the logs directory of max and sally.
27. Open the mule-app-bid-processing logs for both Mules.

28. Examine the cached result (duplicate emails) in the logs.



Module 9: Securing Communication with SSL



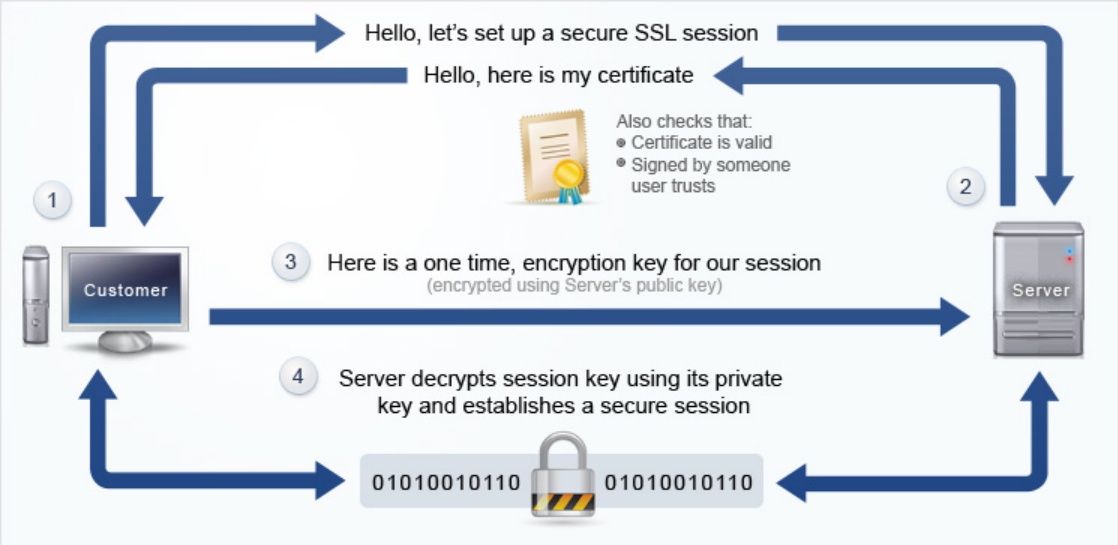
Objectives:

- Discern two-way and one-way SSL.
- Generate Java keystores to facilitate encrypted communication.
- Implement one-way SSL from the server and client perspective.
- Implement two-way SSL from the server and client perspective.

Walkthrough 9-1: Add SSL to Mule applications

In this walkthrough, you will:

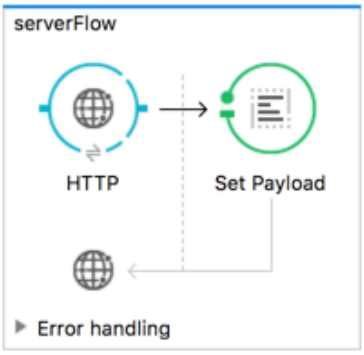
- Generate a keystore containing a server public / private key.
- Apply the keystore to a server's keystore.
- Verify the keystore exists.
- Create a client flow that accepts the server's key into its truststore.



Create a server flow

1. Return to Anypoint Studio.
2. Create a new Mule project with the following attributes:
    - Name: ssl-one-way
    - Use Maven: yes
    - Groupid: com.mulesoft.training
3. Create a new flow named serverFlow.
4. Add an http:listener to the start of the flow.

5. Add a set-payload transformer at the end of the flow.



6. Select the set-payload transformer.
7. Set the value attribute to #['Request received'].
8. Select the http:listener.
9. Create a http:listener-configuration named serverListener.
10. Expose the listener on port ${http.server.port}.
11. Click OK.
12. Set the path to /server.

```
 <http:listener-config name="serverListener" host="0.0.0.0"
port="${http.server.port}"/>
   <flow name="serverFlow">
   <http:listener config-ref="serverListener" path="/server" doc:name="HTTP"/>
   <set-payload value="#['Request received']" doc:name="Set Payload"/>
 </flow>
```

Test the flow

13. Run the application; verify it successfully deploys.
14. Call the endpoint from a browser.
15. Verify you receive a 200 response.

```
yourUser$ curl -i http://localhost:8082/server

HTTP/1.1 200
Content-Length: 16
Date: Mon, 17 Aug 2015 19:40:34 GMT

Request received
```

16. Stop the application.

Generate the server key store

17. In a command-line interface, navigate to the root of this project.
18. Change into the src/main/resources directory.
19. Create a server keystore by running the following command.

```
 keytool -genkey -keyalg RSA -alias selfsigned -keystore serverStore.jks -storepass
password -validity 360 -keysize 2048
```

Note: This command can be found in the course snippets.txt file.

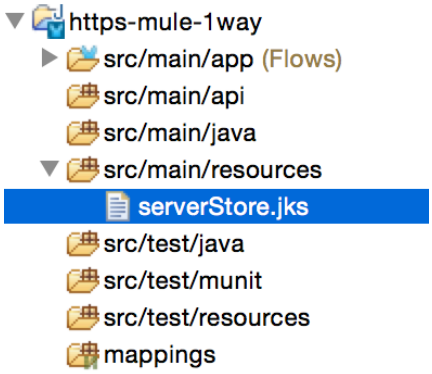
20. Enter localhost for first and last name.
21. Enter any information subsequent prompts.
22. Verify your selections and enter yes.

```
Is CN=localhost, OU=Tech, O=MuleSoft, L=San Francisco, ST=California, C=US
correct? [no]: yes
```

23. Set the key selfsigned to have the same password as the keystore.

Secure the server

24. Return to Anypoint Studio.
25. Right-click the project's src/main/resources directory, select Refresh.
26. Verify the new serverKeystore.jks is part of the folder.



27. Open the serverListener's (global element) configuration.
28. Set the Protocol to HTTPS.
29. Click the TLS/SSL tab.
30. Choose Use TLS Config.

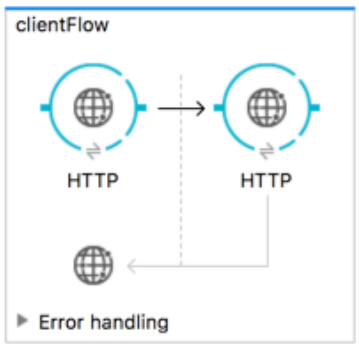
31. In the Key Store Configuration, add:
    - Type: JKS
    - Path: serverStore.jks
    - Key Password: password
    - Password: password

32. Click OK.

```
<http:listener-config name="serverListener" protocol="HTTPS" host="0.0.0.0"
    port="${http.server.port">
  <tls:context>
    <tls:key-store type="jks" path="serverStore.jks" alias="selfsigned"
        keyPassword="password" password="password"/>
  </tls:context>
</http:listener-config>
```

Create a client flow

33. Add a new flow named clientFlow.
34. Add an http:listener to the start of the flow.
35. Add an http:requestor to the end of the flow.



36. Select the http:listener.
37. Create a http:listener-configuration named HTTPUnsecureListener.

38. Expose the listener on port ${http.client.port}.
39. Click OK.
40. Set the path to /client.
41. Select the http:requestor.
42. Configure the requestor to call the serverFlow's endpoint.

```
<http:listener-config name="HTTPUnsecureListener" host="0.0.0.0"
    port="${http.client.port"/>

<http:request-config name="HTTPSecureRequest" host="localhost"
    port="${http.server.port}"/>

<flow name="clientFlow">
  <http:listener config-ref="ClientListener" path="/client"/>
  <http:request config-ref="HTTPSecureRequest" path="/server" method="GET"/>
</flow>
```

Add default port settings

43. In src/main/app, open mule-app.properties.
44. Add the http client property with a value of 8081.

```
http.client.port=8081
```

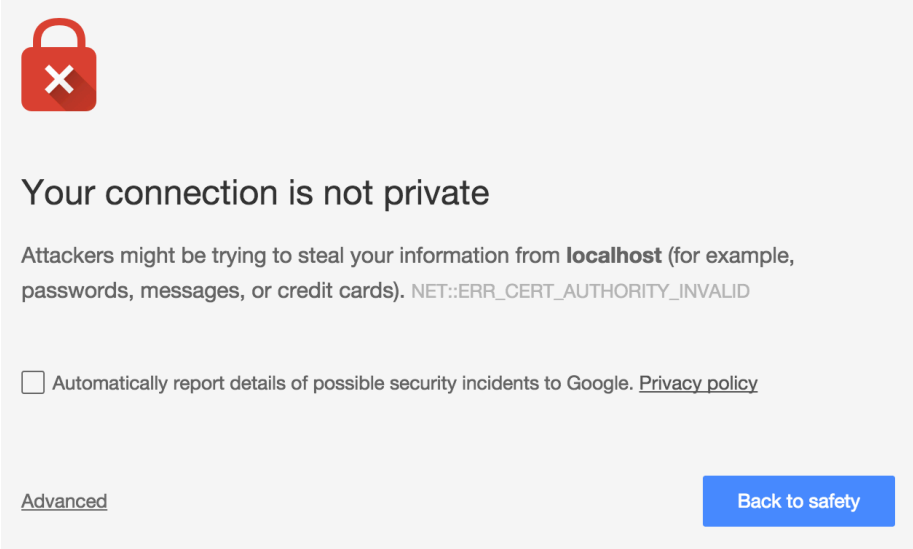
45. Add the http server property with a value of 8082.

```
http.server.port=8082
```

Test the application

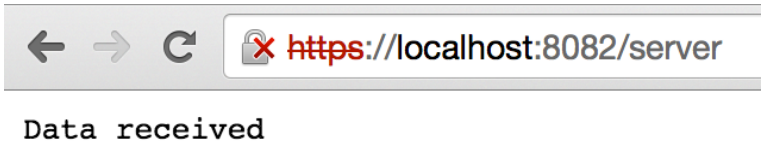
46. Run the application.
47. From a browser, hit the server flow directly at https://localhost:8082/server.

48. Examine the prompt in your browser.



Note: Results may vary depending on your browser.

49. If your browser has an allow access option, click it.



Note: In Chrome this will be Advanced > Proceed to localhost (unsafe).

50. Call the client flow at http://localhost:8081/client; examine the output.

```
Error sending HTTP request. Message payload is of type: NullPayload
```

51. Check the console logs.

Note: The client did not trust the server's certificate.

Trust the server

52. Return to Anypoint Studio.
53. In the Global Elements, open the configuration for HTTPSSecureRequest.
54. Select the TLS/SSL tab.
55. Click Use TLS Config.

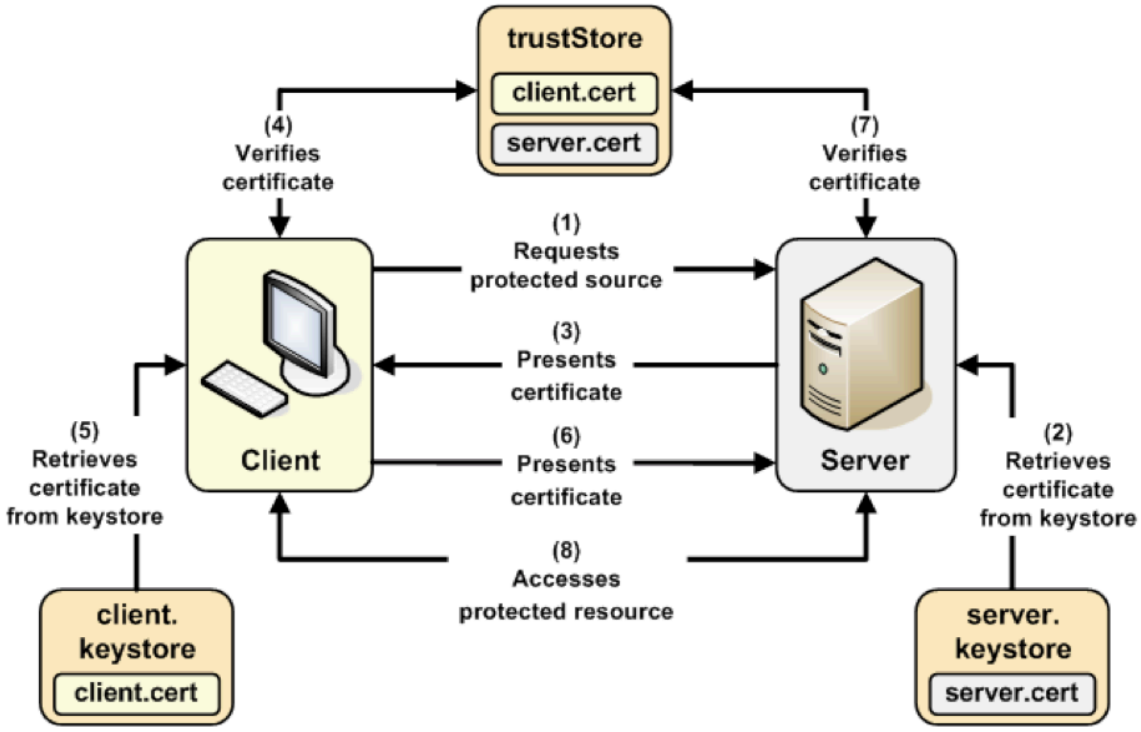
56. In Trust Store Configuration, add the following:
    - Path: serverStore.jks
    - Password: password
    - Type: JKS
57. Click OK.
58. Save the application.
59. Call the client flow again at http://localhost:8081/client; verify a response is returned.

```
Data received
```

Walkthrough 9-2: Add two-way SSL to Mule applications

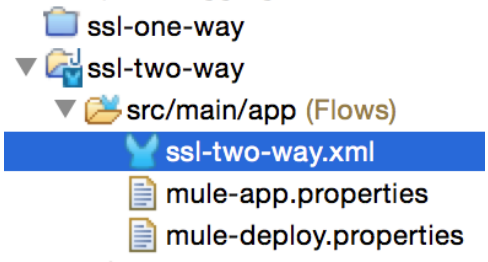
In this walkthrough, you will:

- Generate a keystore containing a server public / private key.
- Apply the keystore to a client's keystore.
- Apply the keystore to a server's keystore.
- Verify the server is only accessible through the Mule client.



Duplicate the existing project

1. Duplicate the existing ssl-one-way project.
2. Refactor > Rename the duplicate to ssl-two-way.
3. Rename the src/main/app/ssl-one-way.xml to ssl-two-way.xml.
4. Right click and close the ssl-one-way project.



Generate the client key store

5. Open a command-line interface and navigate to the root of ssl-two-way.
6. Change into the src/main/resources directory.
7. Create a client keystore by running the following command.

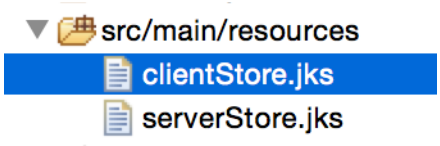
```
 keytool -genkey -keyalg RSA -alias selfsigned -keystore clientStore.jks -storepass
password -validity 360 -keysize 2048
```

Note: This command is also available in the course snippets.txt file.

8. Enter localhost for first and last name.
9. Enter any information subsequent prompts.
10. Verify your selections and enter yes.

```
  Is CN=localhost, OU=Tech, O=MuleSoft, L=San Francisco, ST=California, C=US
  correct? [no]: yes
```

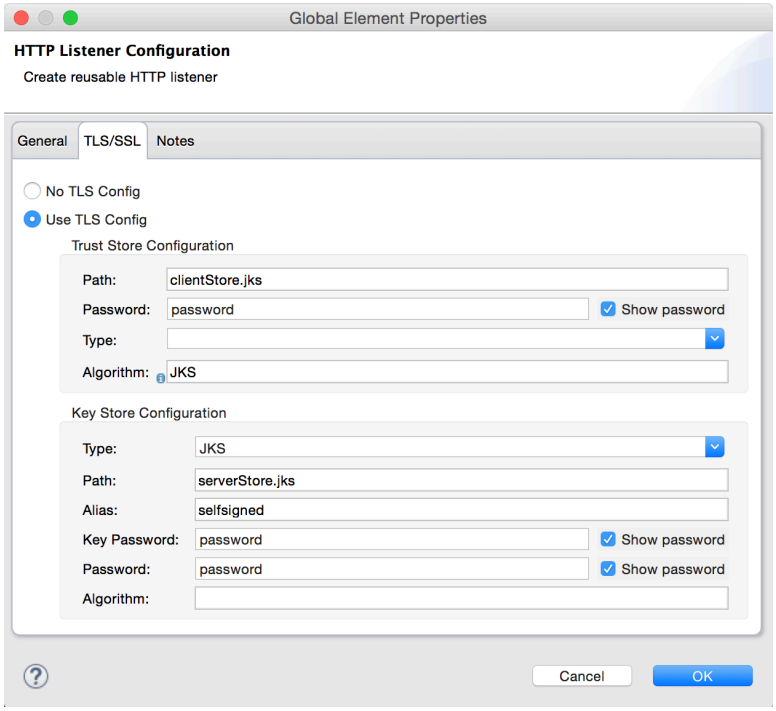
11. Return to Anypoint Studio.
12. Refresh the src/main/resoures directory and verify the clientStore.jks is present.



Verify the client identity from the server

13. In the serverFlow, select the http:listener.
14. Edit the connector-configuration.
15. Select the TLS/SSL tab.

16. In Trust Store Configuration, add:
    - Type: JKS
    - Path: clientStore.jks
    - Password: password
    - Key Password: password

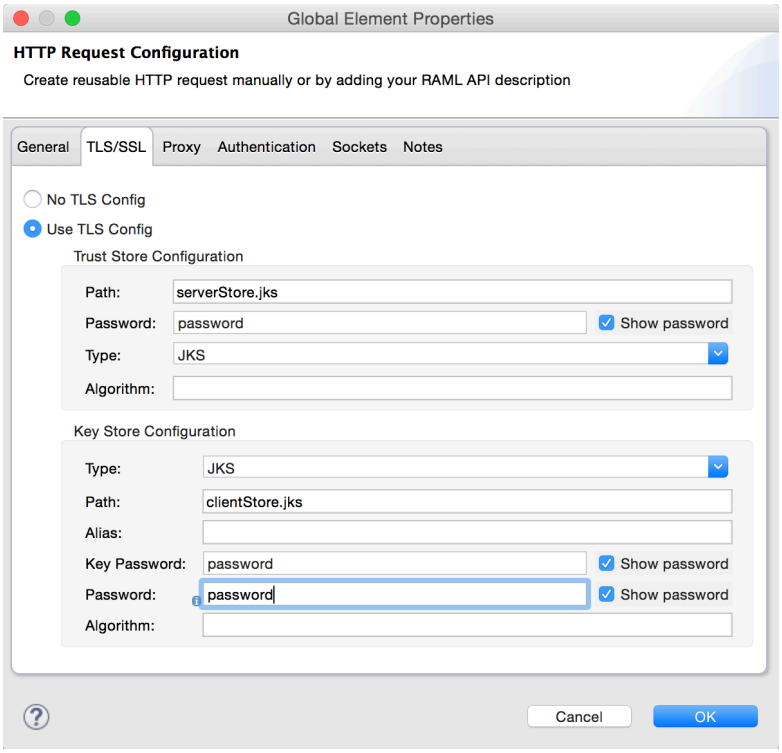


17. Click OK.
18. Save the application.
19. Run the project.
20. Open a web browser.
21. Call the server endpoint directly.
22. Verify the browser is unable to access the serverFlow.
23. Call the clientFlow endpoint.
24. Verify the client is also unable to access the serverFlow.

Add a key store to the client

25. Return to Anypoint Studio.
26. In the clientFlow, select the http:requestor.
27. Edit the http:requestor's connector configuration.
28. Select the TLS/SSL tab.

29. In Client Store Configuration, add:
    - Path: clientStore.jks
    - Password: password
    - Type: JKS



30. Click OK.
31. Save the application.

Test the server endpoint

32. Run the project.
33. Call the server endpoint directly.
34. Verify the browser is unable to access the serverFlow.
35. Call the clientFlow endpoint.
36. Verify the client is able to access the serverFlow.