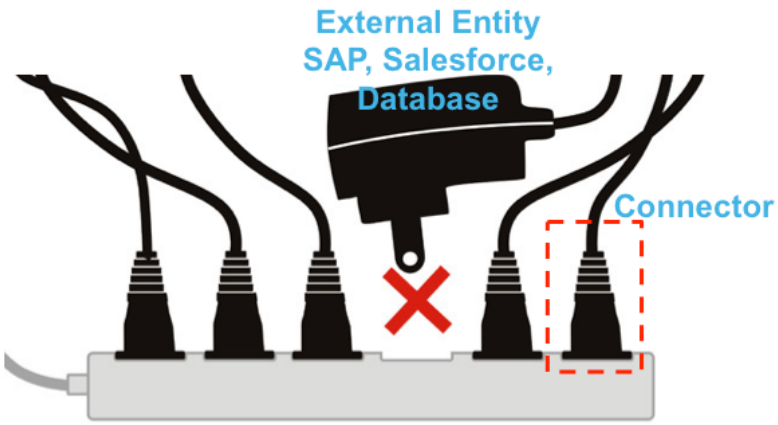
Anypoint Platform Development: Custom Connectors

Student Manual

Mule runtime 3.8
May 20, 2016

Table of Contents

Module 1: Introducing Anypoint Connectors and DevKit



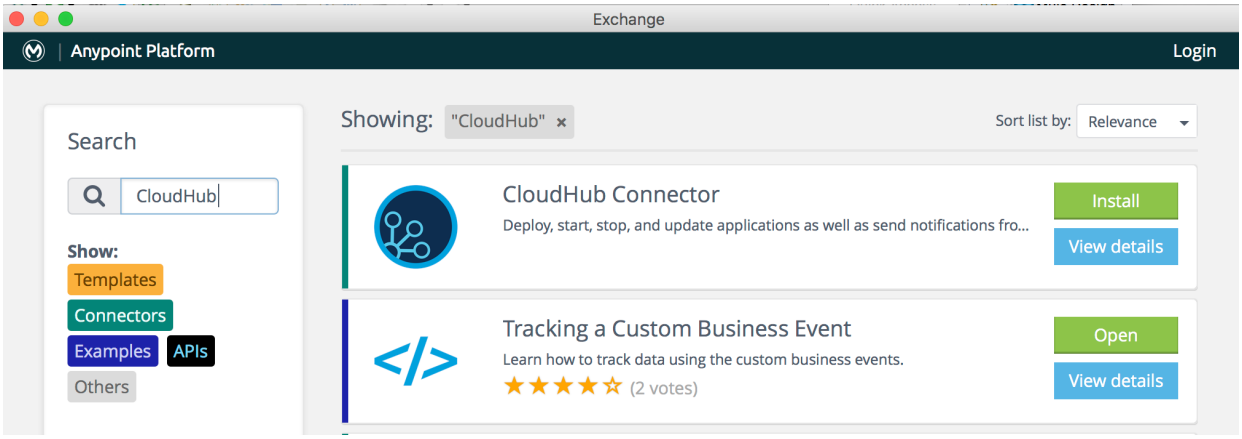
Objectives:

- Identify product components and solution architectures using Anypoint Connectors.
- Locate connectors outside of those included in Anypoint Studio.
- Learn how to use Anypoint DevKit to assist in custom connector creation.

Walkthrough 1-1 Explore the Anypoint Connector ecosystem

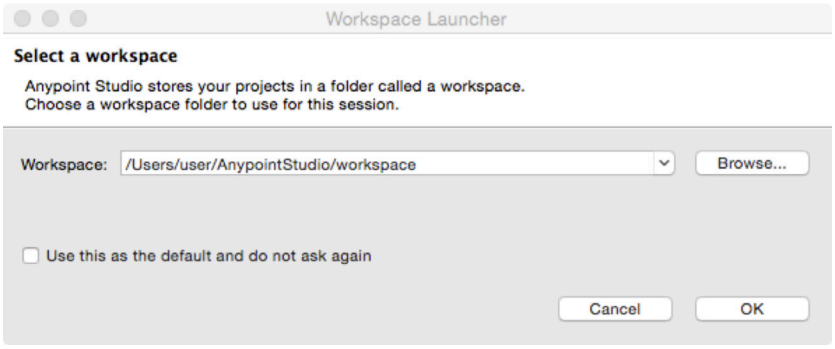
In this walkthrough, you get familiar with Anypoint connectors. You will:

- Log in and use the Anypoint Exchange.
- Download and install a connector into Anypoint Studio.
- Locate a connector's source code on GitHub.



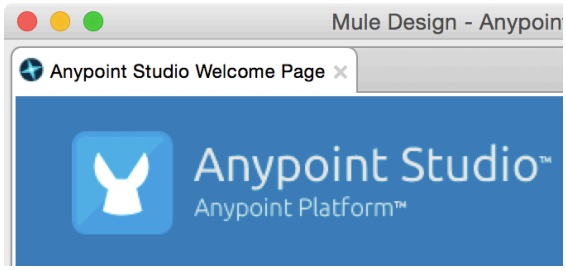
Install the CloudHub connector from the Anypoint Exchange

1. Launch Anypoint Studio.
2. In the Workspace Launcher dialog, examine the location of the default workspace; change the workspace location if desired.



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

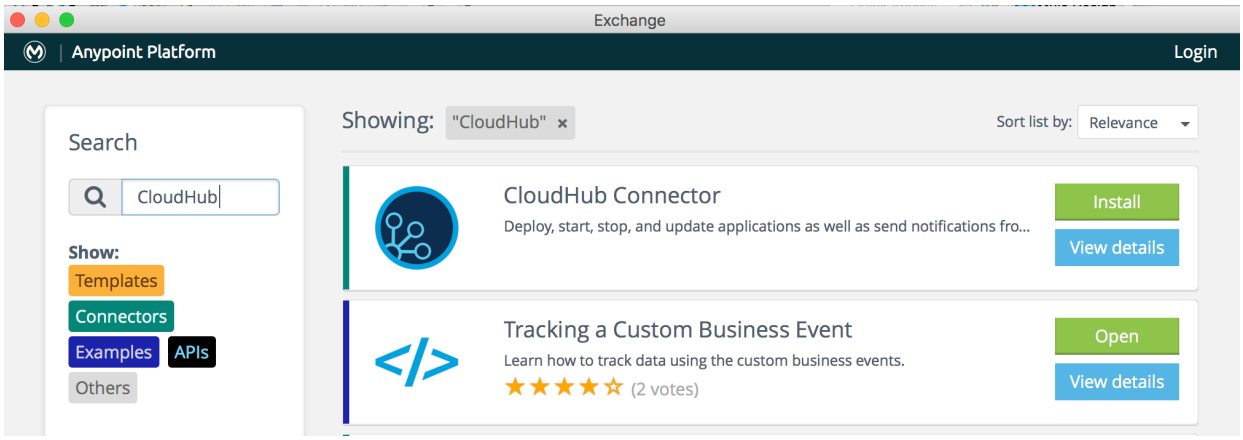
4. If greeted with a Welcome Page, click the X on the tab to close it.



5. Click the Open Exchange icon in the top left corner of Anypoint Studio.
6. In the Exchange window that opens, click Connectors and scroll through the available connectors.

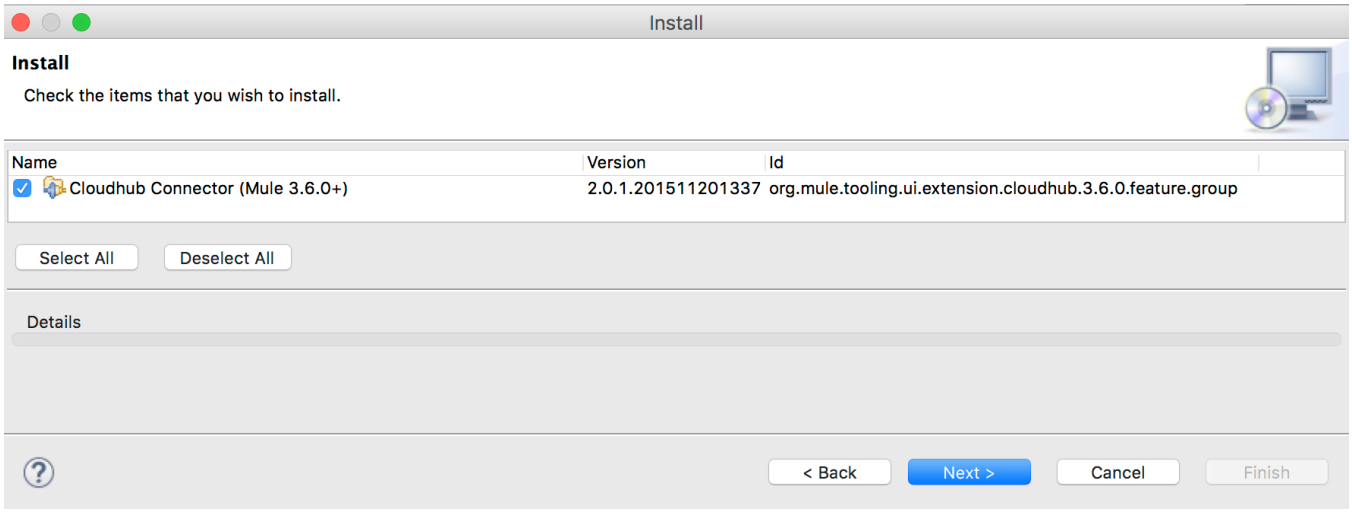
   Note: Notice that many of these connectors have an Install button. This Install button is a symbolic link to the connector's update site. You'll learn more about how Anypoint Connectors use Eclipse update sites soon.

7. Type CloudHub in the Search box and locate the CloudHub Connector.



8. Click View details and explore the CloudHub Connector's Exchange page.
9. Click Install.
10. Complete the registration information.

11. In the Install dialog box, examine the metadata available for the CloudHub connector.



12. Step through the installer, accept the terms, and click Finish.
13. If you see a Security Warning about installing unsigned content, click OK.
14. Wait for the software to be installed and allow Anypoint Studio to restart.

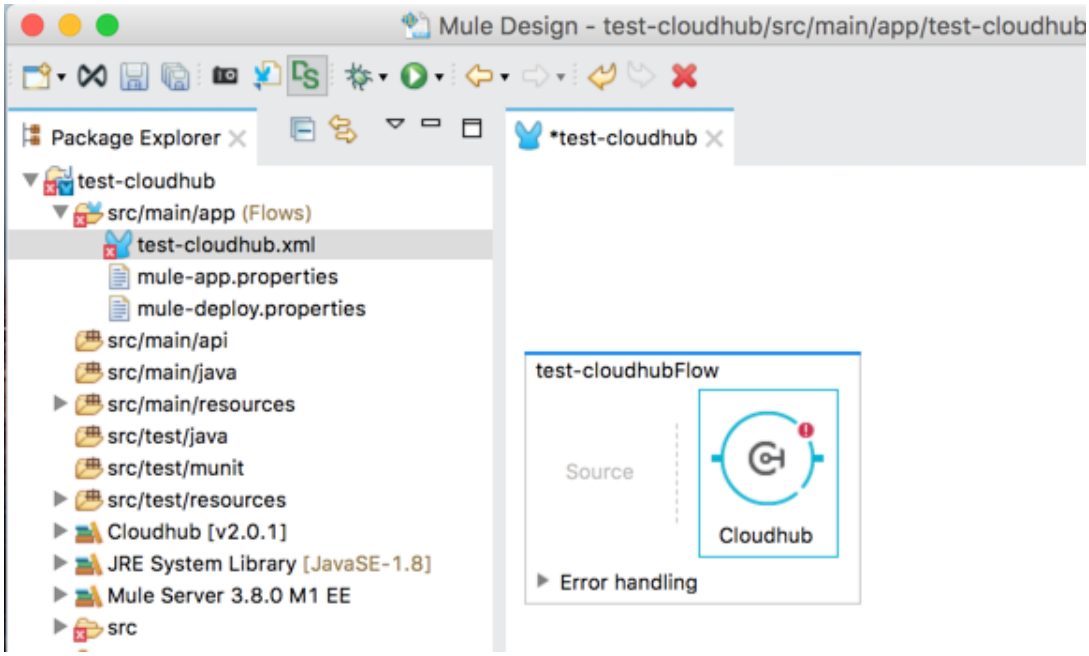
Use the CloudHub connector

15. Create a new Mule project named test-cloudhub.
16. Use a Mule Server 3.8.x EE runtime

    Note: If you do not see the 3.8.x EE runtime, select Help > Install new software.

17. Drag a Flow scope from the palette to the canvas.
18. Locate the new CloudHub connector in the palette.

19. Drag the CloudHub connector into the flow's process section.

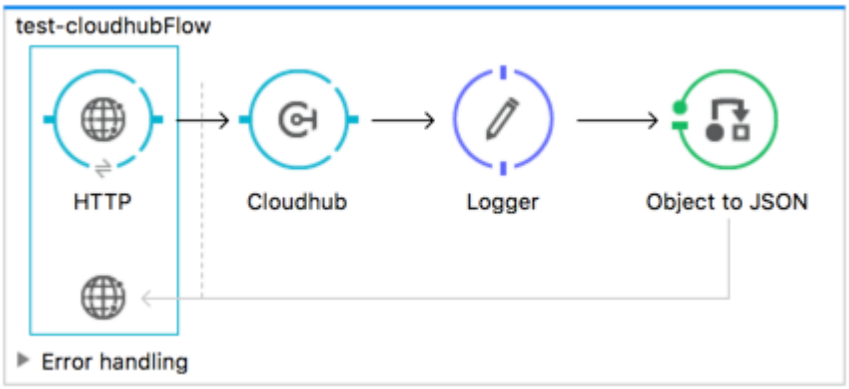


20. Double-click the CloudHub connector endpoint.
21. In the Properties view, click the Add button next to connector configuration.
22. Select Cloudhub: Basic Auth Authentication.
23. Type in your username and password.
24. Click the Test Connection button and verify you can connect to this CloudHub account.

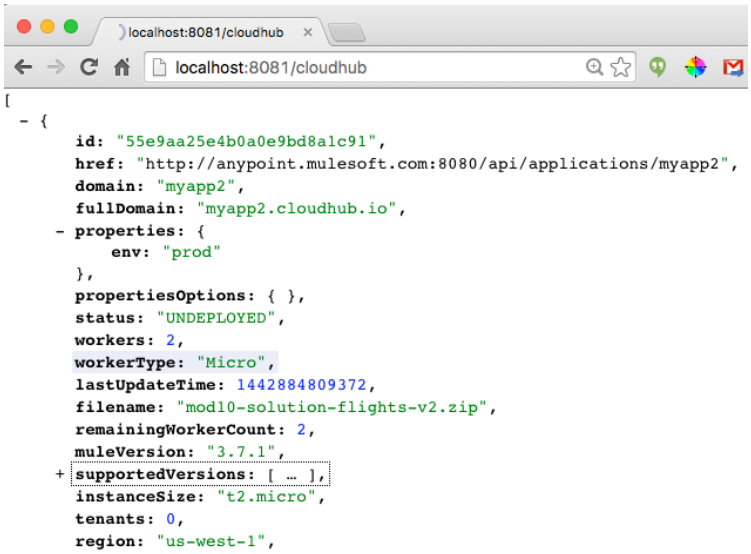
    Note: If you get a Forbidden error, make sure your default environment is set in your Anypoint Platform account, otherwise click the Add button again and select Cloudhub_Inhered_Token_Authentication. This will inherit credentials from the account that deploys the Mule application into CloudHub.

25. Select one of the operations, such as List applications.
26. Add a Logger component and an Object to JSON transformer after the CloudHub connector.
27. In the Properties view of the Logger component, set the Message value to #[message.payload].

28. Add an HTTP Connector and set its base path /cloudhub.



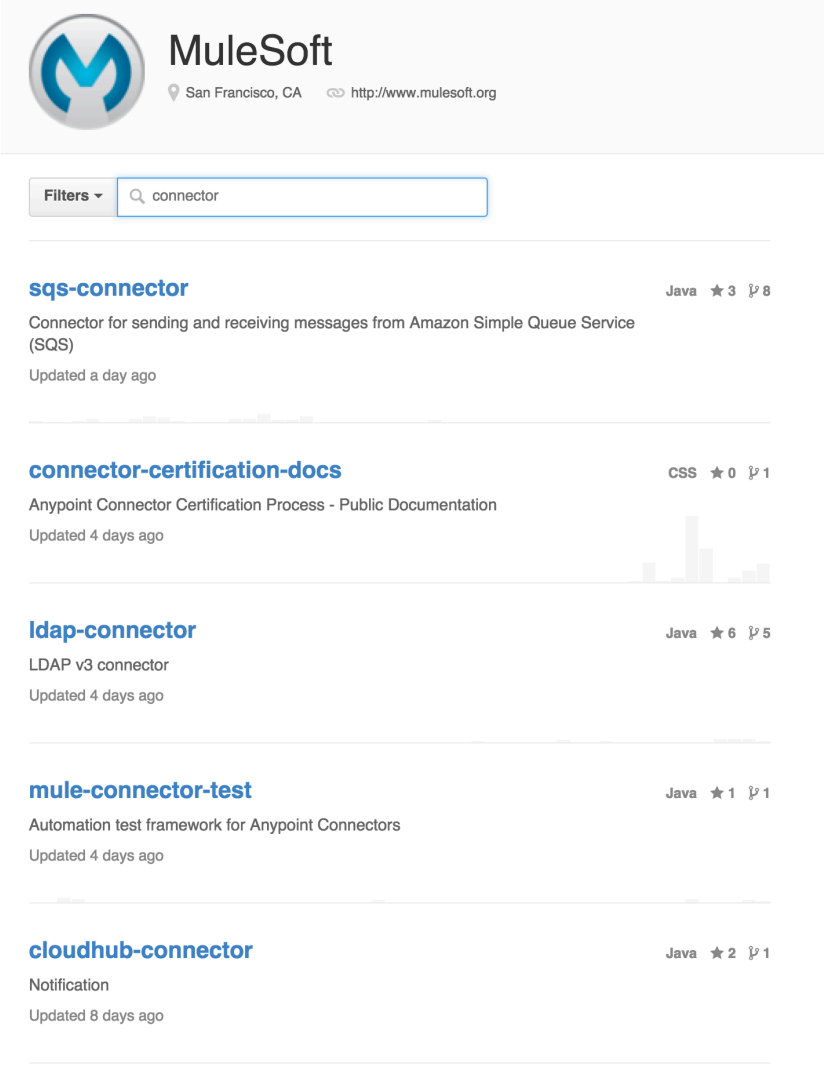
29. Add a breakpoint to the CloudHub connector.
30. Debug and test your Mule project.
31. Verify you see the expected response from the CloudHub connector (such as an array of all deployed applications).



Locate the CloudHub connector's source code on GitHub

32. In a web browser, navigate to http://github.com/mulesoft.

33. In the Filters box, type connector and examine the results.



Note: Filtering by connector is a quick and easy way to see all available MuleSoft connectors that are open source.

34. Search for cloudhub connector.
35. Click mulesoft/cloudhub-connector.

Note: If you're comfortable with Git, feel free to clone this connector to your computer for further exploration. From your preferred directory, simply run git clone https://github.com/mulesoft/cloudhub-connector.

36. Open src > main > java/org/mule/modules/cloudhub > CloudHubConnector.java.

Note: This is the primary class of the connector.

Scan through the connector code

37. Notice the Java file is part of the org.mule.modules.cloudhub package.
38. Notice this file integrates with the com.mulesoft.cloudhub.client Java library.

    Note: This is the Java SDK that actually connects to a CloudHub account and performs operations.

39. Locate the @Connector annotation that specifies which class in the Java file is the main Connector class.
40. Locate the @Config annotation that creates a private variable called config.
41. Search for the private CloudHubConnectionImpl client() method.

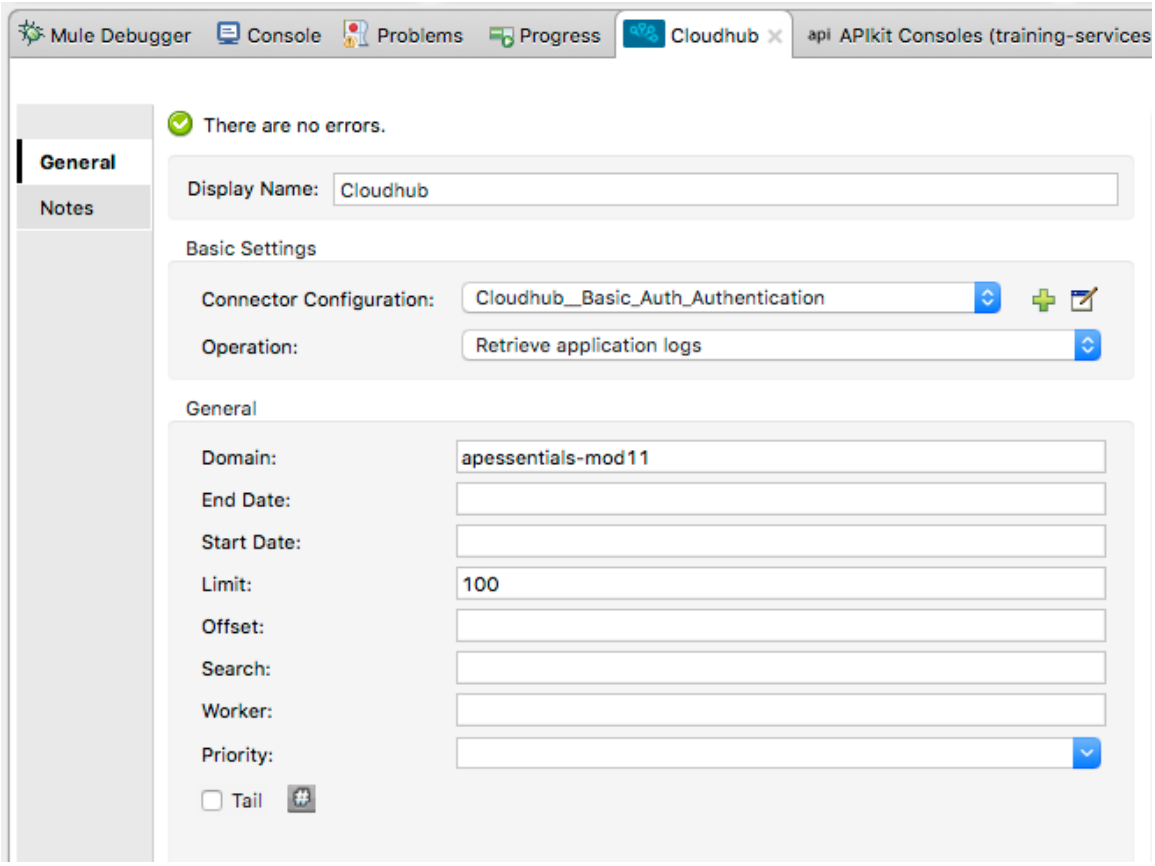
    Note: This method uses the config object to connect to a CloudHub account, using whatever connection configuration is created in the Mule project (either basic authentication, or inherited authentication within the same CloudHub account in which the project is deployed.

42. Look through the other operation implementations, such as deleteApplication().
43. Notice how the client() method is used to carry out CloudHub Java SDK operations.

Match a CloudHub operation with the Anypoint Studio GUI

44. Return to Anypoint Studio.
45. Select the CloudHub connector.
46. In the Properties view, select different operations and notice how some operations have different numbers and types of parameters.
47. Select the Retrieve application logs operation.

48. Review the properties listed in the general section.

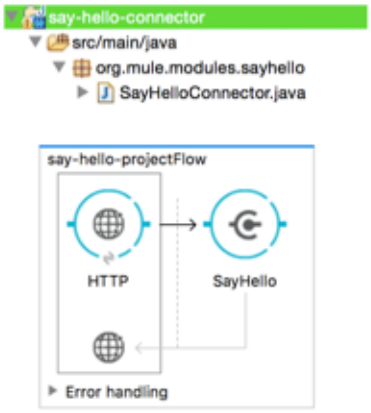


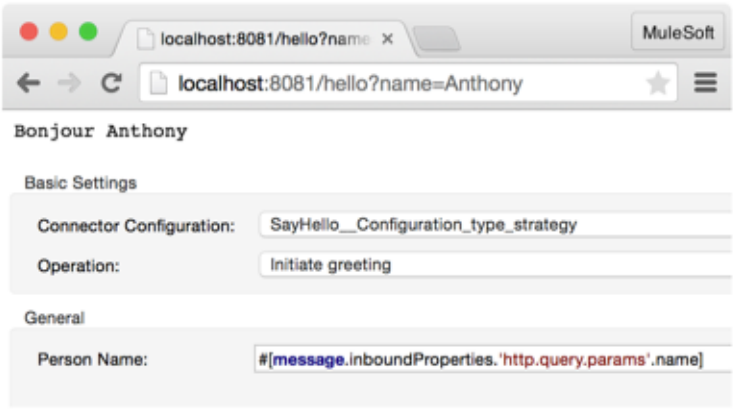
49. Select the Priority drop-down list and look at the enumerated values.
50. Return to the GitHub project and open the org.mule.modules.cloudhub.utils.LogPriority.java file; notice that LogPriority is an enum of log level values.
51. Search in the CloudHubConnector Java class file for the retrieveApplicationLogs() method.
52. Notice that method is annotated with the @Processor annotation.
53. Notice how the method parameters correspond with the configuration parameters in the Studio Cloudhub configuration view.
54. In particular, notice the priority parameter is of type LogPriority (which appears in the Studio configuration GUI as a drop down list of the enum values).

    Note: This shows you how the Mule Connector DevKit helps you quickly create graphical configuration Anypoint Studio configuration views for your custom connectors.

55. View the config/BasicAuthConfig.java file to see how authentication is handled with CloudHub.

Module 2: Creating Your First Connector Project





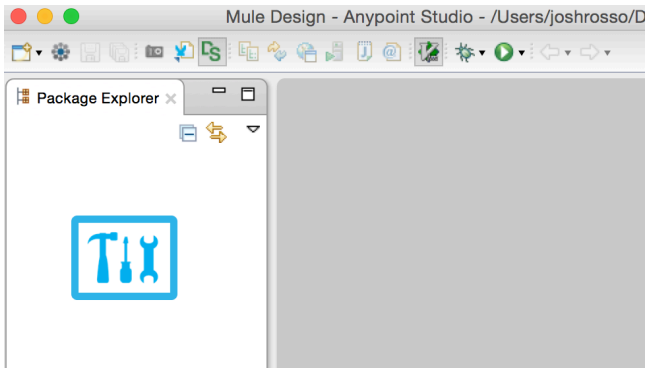
Objectives:

- Set up a development environment capable of building Anypoint Connectors.
- Configure Anypoint Studio to use Maven.
- Create an Anypoint Connector project.
- Develop a basic connector.
- Create an operation for the connector.
- Install a newly developed connector into Anypoint Studio.

Walkthrough 2-1: Set up the environment for building connectors

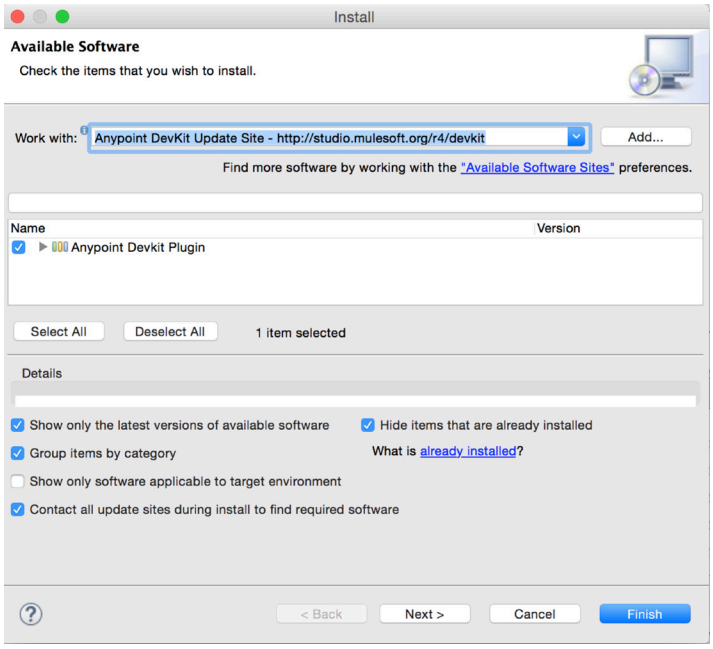
In this walkthrough, you prepare for connector development. You will:

- Download and install the Anypoint DevKit plugin.
- Verify the Anypoint DevKit plugin was successfully installed.
- Configure Anypoint Studio to use Maven.

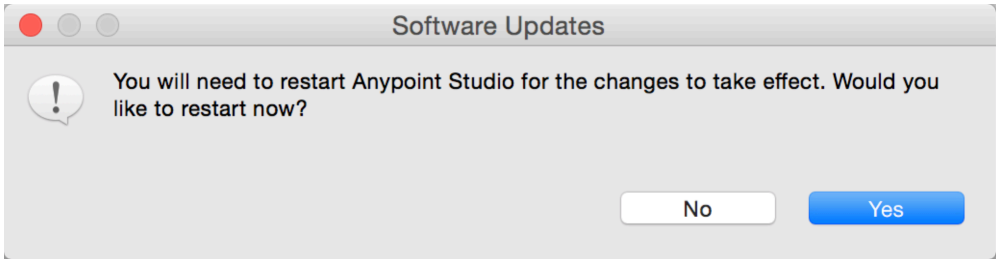


Install the Anypoint DevKit plugin

1. Return to Anypoint Studio.
2. Select Help > Install New Software.
3. In the Work with drop-down menu, select Anypoint DevKit Update Site.
4. Select Anypoint DevKit Plugin.
5. Click Next.

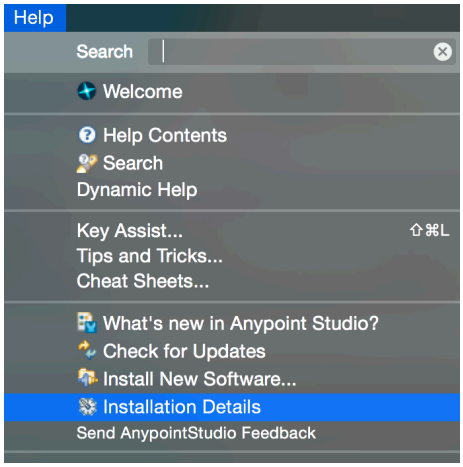


6. Click Finish and wait for the software to install.
7. In the Software Updates dialog box, click Yes to restart Anypoint Studio.

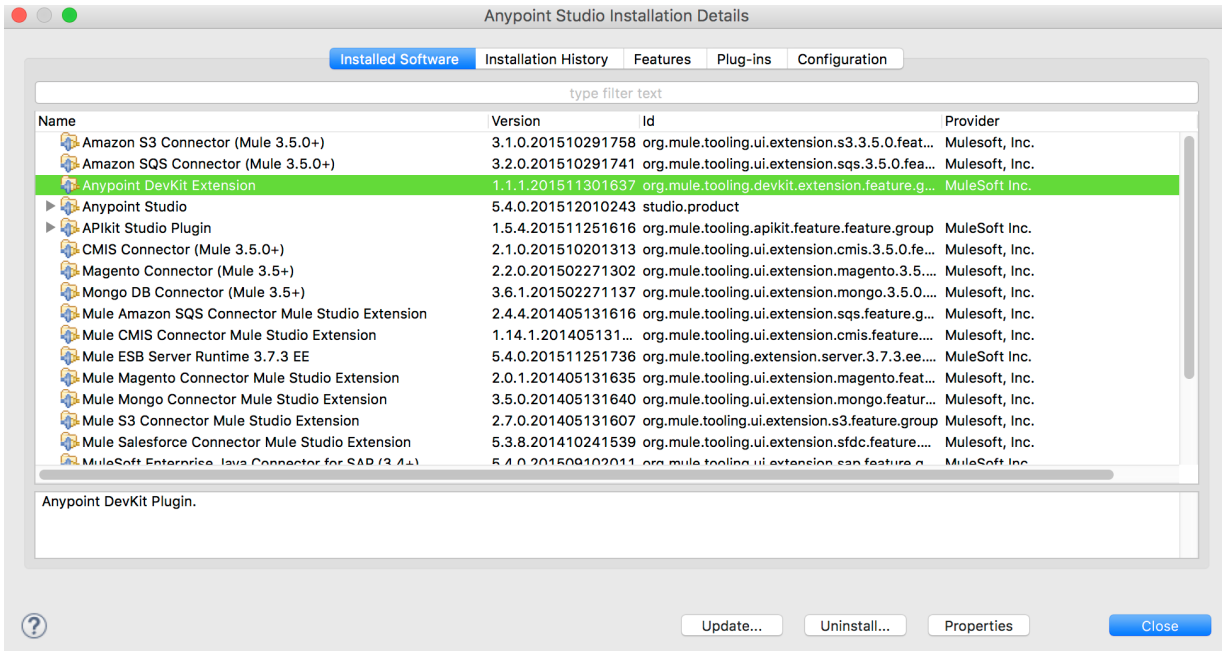


Verify the DevKit installation

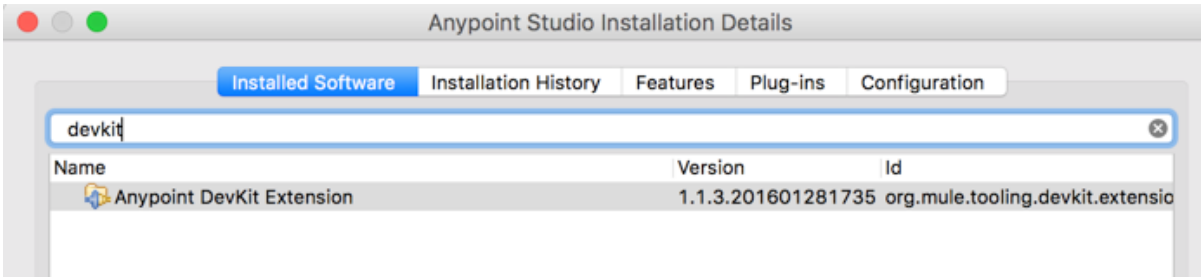
8. Select Help > Installation Details.



9. Search for DevKit.



10. Verify the Anypoint DevKit Extension is present.

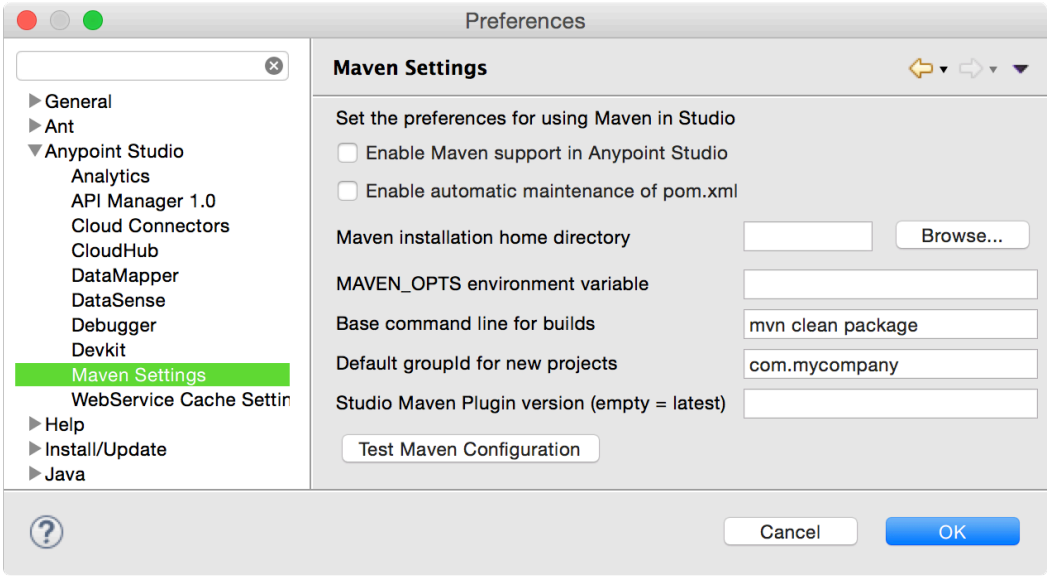


Note: Your version of DevKit may differ from the one shown in the screenshot above.

11. Click Close.

Verify the Maven installation

12. On Mac, select Anypoint Studio > Preferences or on Windows, select Window > Preferences.
13. In the Preferences dialog box, expand Anypoint Studio then select Maven Settings.

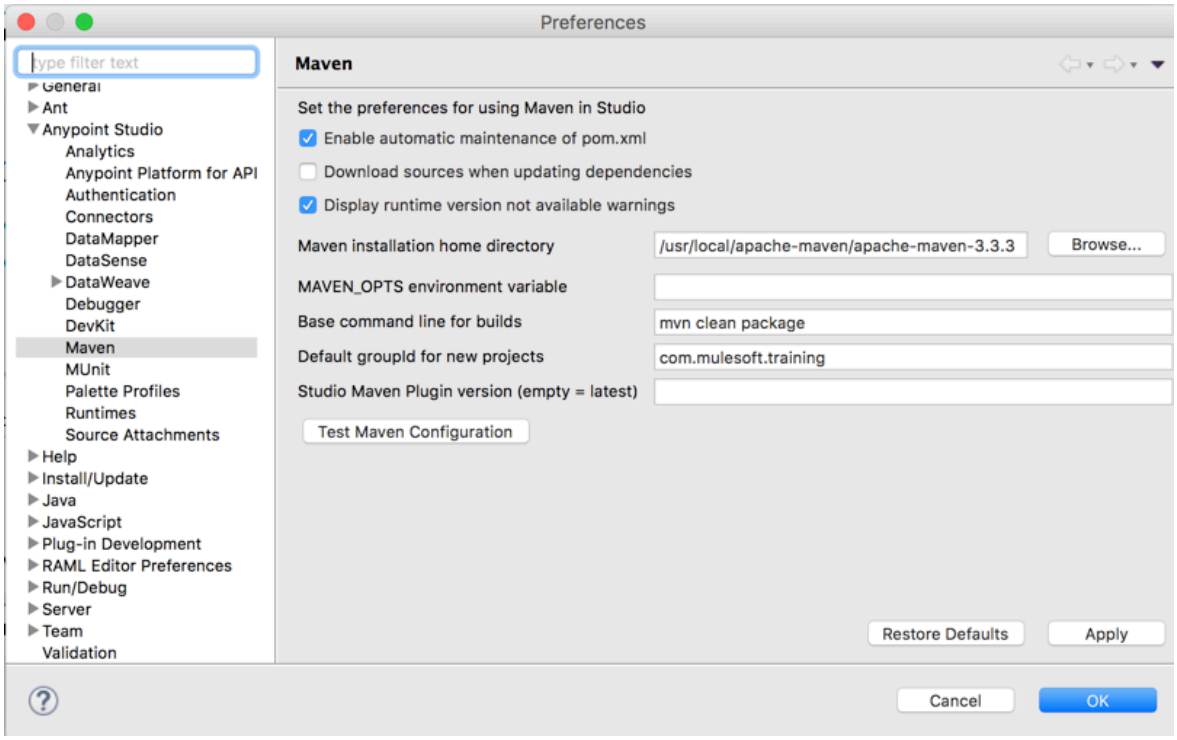


14. Set the preferences as follows:

- Check Enable Maven support in Anypoint Studio.
- Check Enable automatic maintenance of pom.xml.
- Set Maven installation home directory to your apache-maven installation folder (e.g. apache-maven-3.3.3).
- Set Default groupId for new projects to com.mulesoft.training.

Note: You don't have to use com.mulesoft.training, but all examples in this class will use that groupId.

15. Click Test Maven Configuration.
16. Verify the test completes successfully.

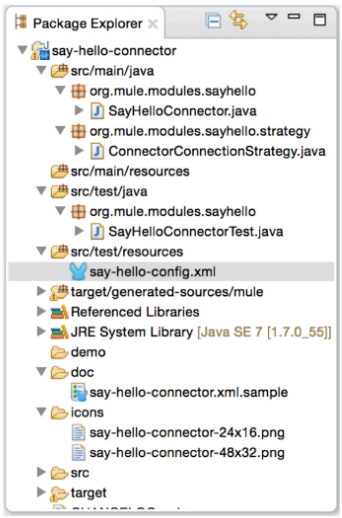


17. Click OK.

Walkthrough 2-2: Create an Anypoint Connector project

In this walkthrough, you create your first Anypoint Connector project. You will:

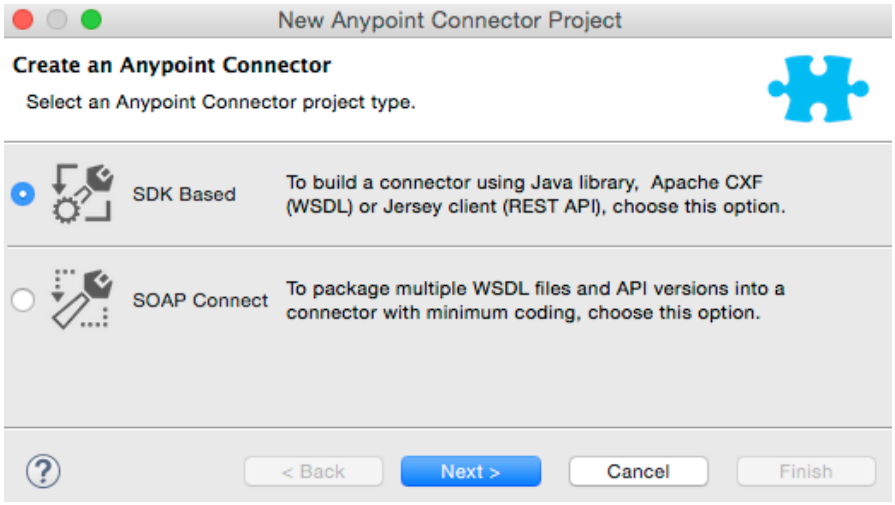
- Create an Anypoint Connector project in Anypoint Studio.
- Examine the project structure.
- Modify the default operation.




```
@Configurable
@Default("Hello")
private String greetingMessage;


@Processor
public String initiateGreeting(String personName) {
    String message = this.greetingMessage + " " + personName;
    return message;
}
```

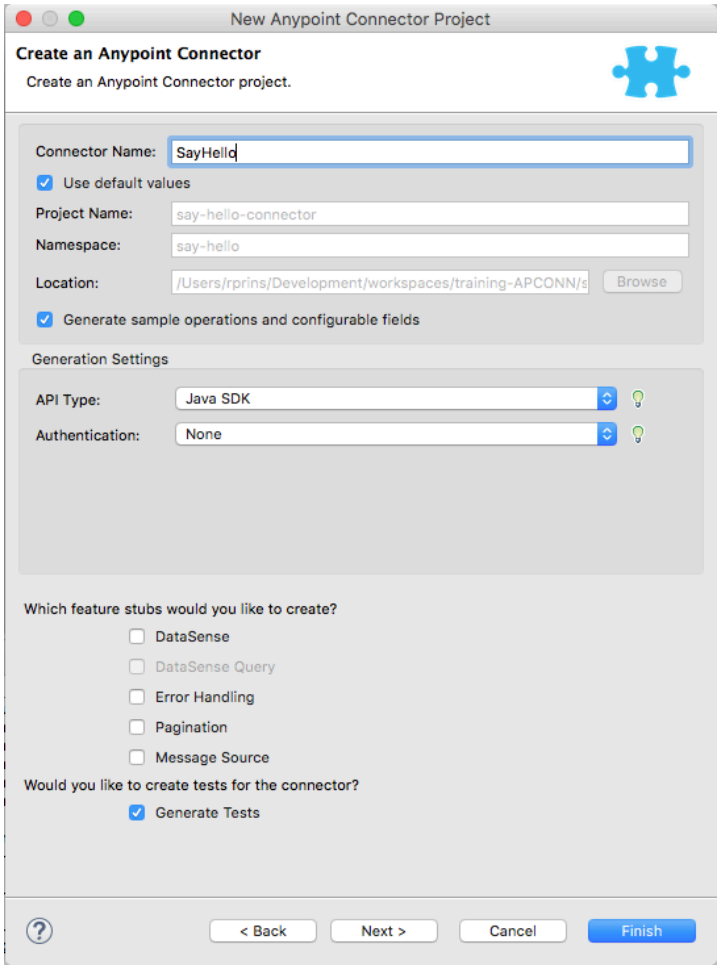
Create the project

1. Select File > New > Anypoint Connector Project.
2. Select the default type SDK Based and click Next.

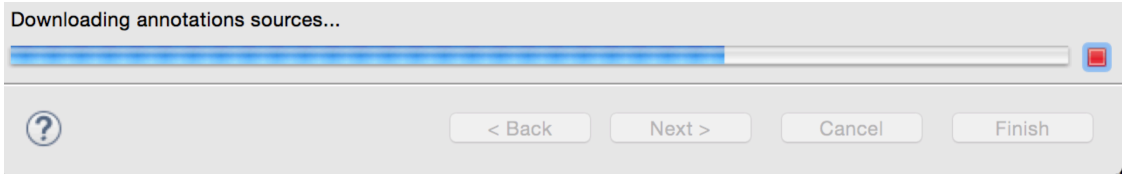


3. In the New Anypoint Connector Project dialog box, set the connector name to SayHello, and keep the default values.

4. Set the API Type to Java SDK.
5. Set Authentication to None.
6. Do not select any features stubs.
7. Check the Generate Tests checkbox.



8. Click Next and leave the default values checked.
9. Click Next and type in the new label:
    - Label: SayHello
10. Click Finish.
11. Wait for Maven to gather all necessary dependencies.



Note: This could take some time if this is your first connector project.

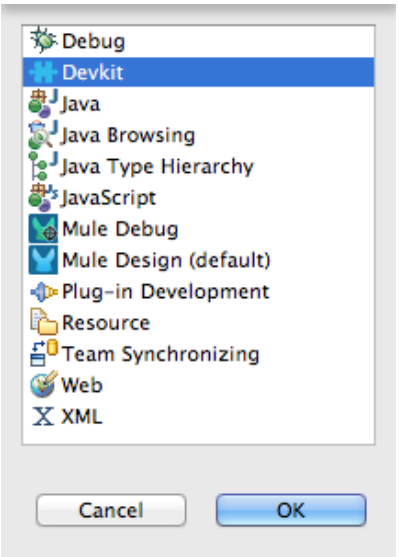
12. Once downloading is complete, check the console output; the build should have succeeded.

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16.772 s
[INFO] Finished at: 2015-01-31T23:04:40-08:00
[INFO] Final Memory: 60M/677M
[INFO] ------------------------------------------------------------------------
```

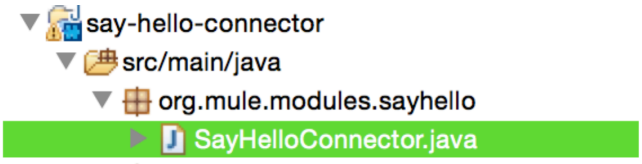
Note: You might see warnings about including the javadoc sources. If you want to remove these warnings, return to Anypoint Studio > Preferences, then Anypoint Studio > Maven Settings. Then you can add to MAVEN_OPTS environment variables: -DdownloadSources=true

Explore the project structure

13. In the upper-right side change the perspective from Mule Design to DevKit.



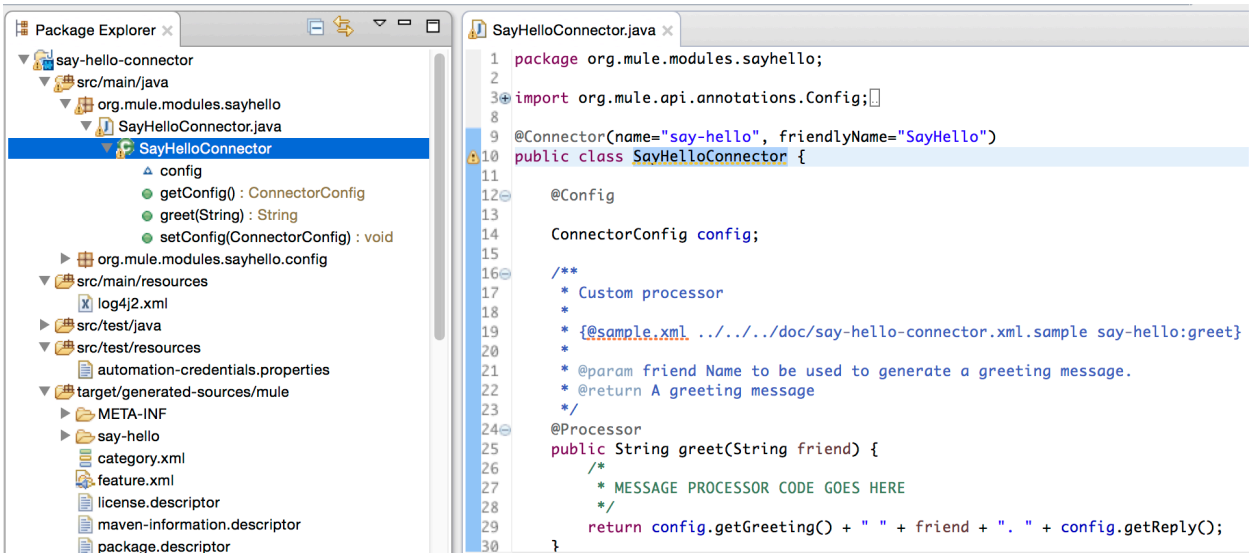
14. In the Package Explorer, expand the say-hello-connector project.
15. Expand the src/main/java/ package.
16. Expand the org.mule.modules.sayhello package.
17. Double-click SayHelloConnector.java to open the file.



Note: This file may have opened automatically.

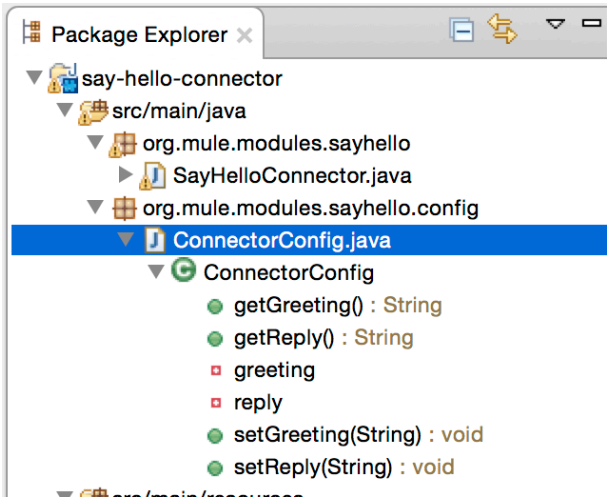
18. Scan through this connector file.

19. Explore the annotations used in this class including @Connector, @Config, and @Processor.



Note: This file serves as the primary class for the connector. Many of the items you want to expose to integration developers will be included here.

20. In the Package Explorer, expand the org.mule.modules.sayhello.config package.

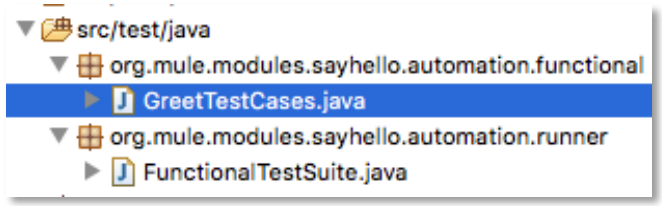


21. Open ConnectorConfig.java.

SayHelloConnector.java | ConnectorConfig.java

```
1  package org.mule.modules.sayhello.config;
2
3  import org.mule.api.annotations.components.Configuration;
6
7  @Configuration(friendlyName = "Configuration")
8  public class ConnectorConfig {
9
10     /**
11      * Greeting message
12      */
13     @Configurable
14     @Default("Hello")
15     private String greeting;
16
17     /**
18      * Reply message
19      */
20     @Configurable
21     @Default("How are you?")
22     private String reply;
23
24     /**
25      * Set greeting message
26      *
27      * @param greeting the greeting message
28      */
```

22. Expand the src/test/java package; you should see several classes generated.



23. Expand the org.mule.modules.sayHello.automation.functional and org.mule.modules.sayHello.automation.runner packages.
24. Open GreetTestCases.java and FunctionalTestSuite.java and examine the code.

    Note: If you have ever written FunctionalTestCases for Mule applications, this file will look very familiar. *You will eventually use this class to test your connector in the context of a Mule application.*

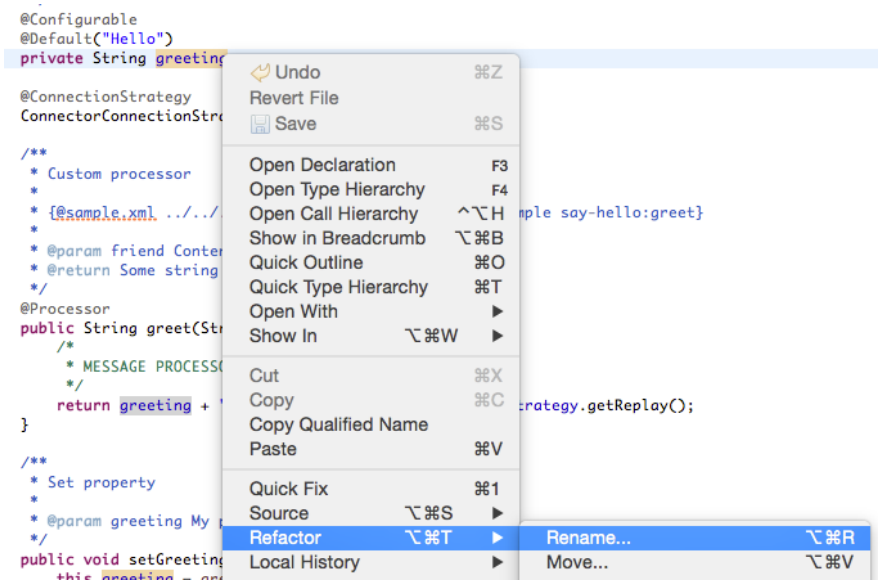
25. Open automation-credentials.properties located in src/test/resources and notice the property that is set.

Modify the connector

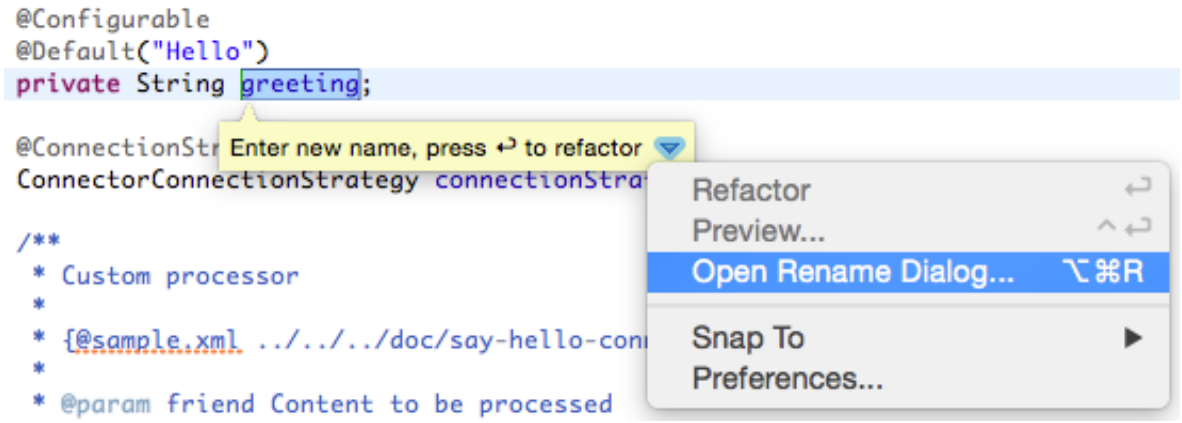
26. Return to ConnectorConfig.java.
27. Locate and select the field greeting.

```
/**
 * Configurable
 */
@Configurable
@Default("Hello")
private String greeting;
```

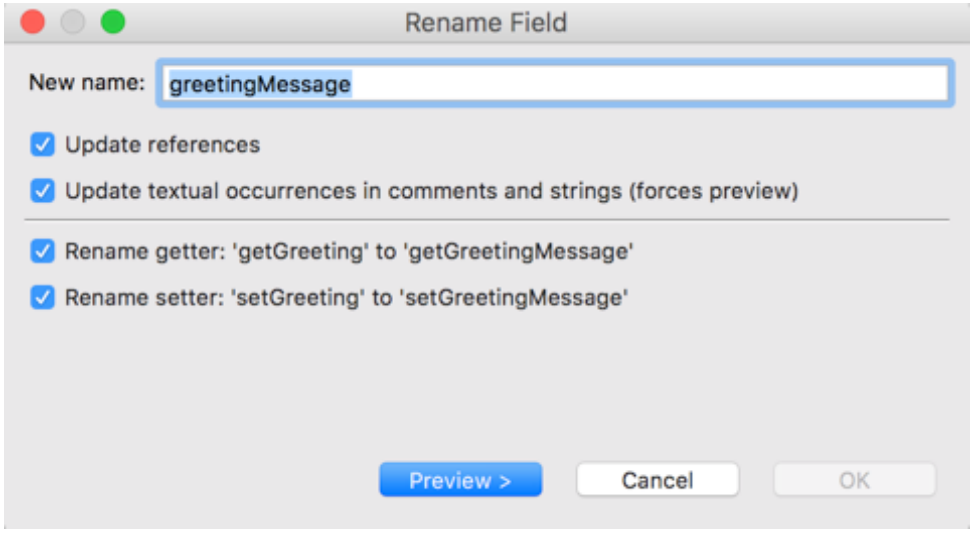
28. Right-click it and select Refactor > Rename.



29. In the pop-up that appears, click the menu button and select Open Rename Dialog.



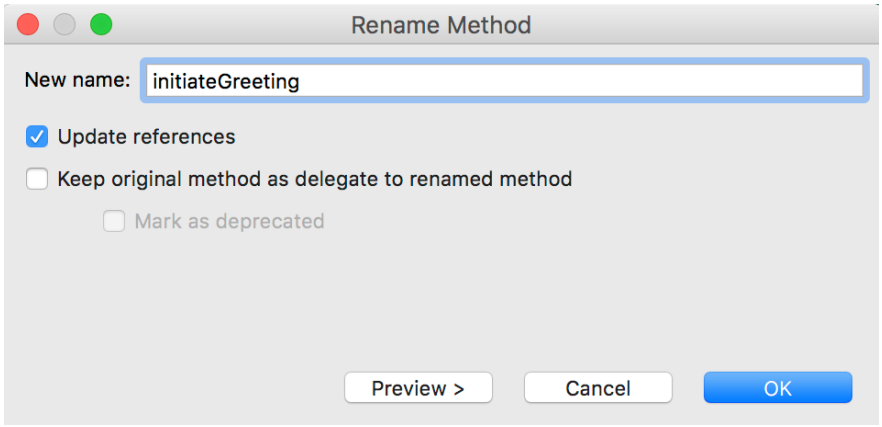
30. In the Rename Field dialog box, set New name to greetingMessage.
31. Click the down arrow button to open the Rename Dialog box.
32. Check all the options to Rename getter and Rename setter as shown below.



33. Click Preview and OK and save your changes.
34. Clean the project to remove any problems in the related SayHelloConnector.java file.
35. Open SayHelloConnector and locate the greet() method.

```
@Connector(name="say-hello", friendlyName="SayHello")
public class SayHelloConnector {

    @Config

    ConnectorConfig config;

    /**
     * Custom processor
     *
     * {@sample.xml ../../../doc/say-hello-connector.xml.sample say-hello:greet}
     *
     * @param friend Name to be used to generate a greeting message.
     * @return A greeting message
     */
    @Processor
    public String greet(String friend) {
        /*
         * MESSAGE PROCESSOR CODE GOES HERE
         */
        return config.getGreetingMessage() + " " + friend + ". " + config.getReply();
    }
```

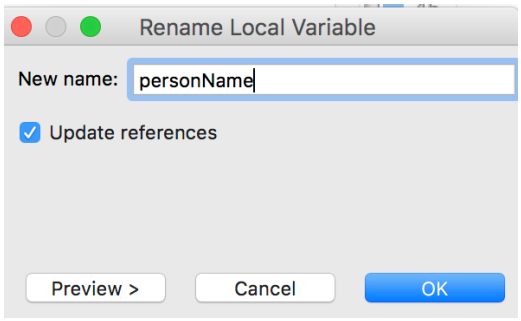
36. Highlight the method name greet and right-click to refactor greet to initiateGreeting.



37. In the upper menu, select Project > Clean.
38. Verify the clean operation removes any problems and view the code change.

```
@Processor
public String initiateGreeting(String friend) {
    /*
     * MESSAGE PROCESSOR CODE GOES HERE
     */
    return config.getGreetingMessage() + " " + friend + ". " + config.getReply();
}
```

39. Highlight the friend parameter and right-click to refactor it to personName.

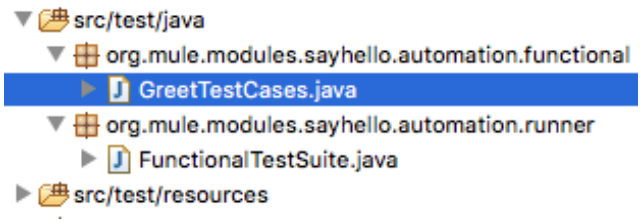


40. Clean the project again.
41. In the refactored initiateGreeting() method's implementation, delete the comment block.

```
@Processor
public String initiateGreeting(String personName) {
    return config.getGreetingMessage() + " " + personName + ". " + config.getReply();
}
```

42. Save the file.

43. Open org.mule.modules.sayhello.automation.functional.



44. Open GreetTestCases.java and locate the verify() method.

45. Verify the verify() method was updated with the initiateGreeting() method call.

```
@Test
public void verify() {
    java.lang.String expected = null;
    java.lang.String friend = null;
    assertEquals(getConnector().initiateGreeting(friend), expected);
}
```

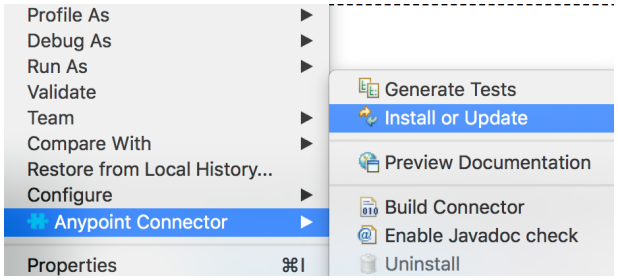
Walkthrough 2-3: Install and use a connector

In this walkthrough, you package a connector and use it in a Mule application. You will:

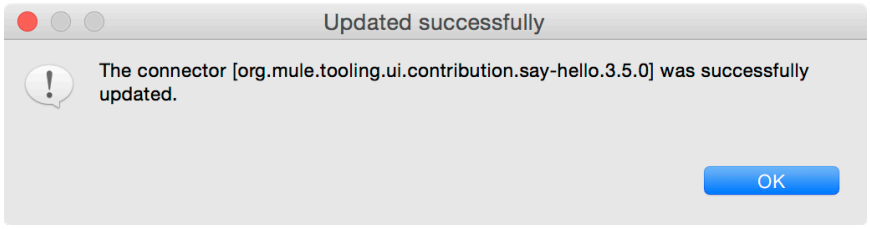
- Generate an update site for a connector.
- Use the update site to install the connector into Anypoint Studio.
- Use the connector in a Mule application.
- Test the output of the connector's operation.

Generate an update site

1. Right-click the say-hello-connector project and select Anypoint Connector > Install or Update.



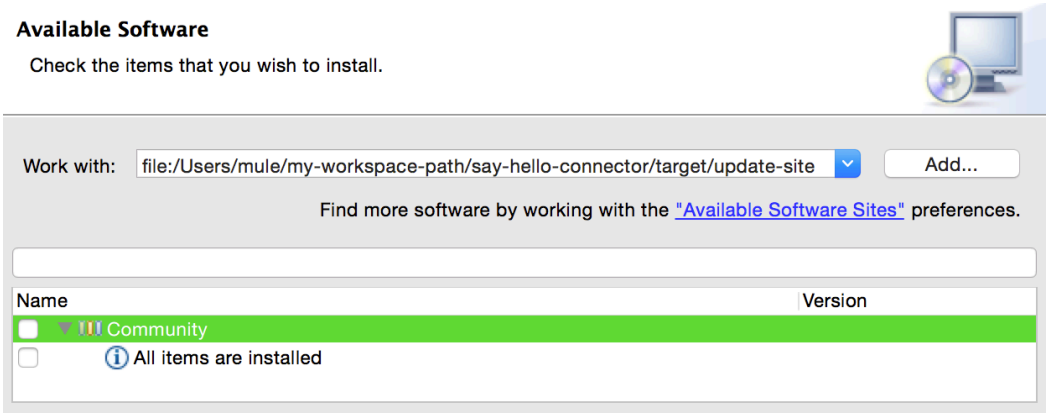
2. View the Console messages and wait (a few minutes) for the connector to install or update.



3. If you get a Security Warning dialog box, click OK.
4. In the Software Updates dialog box, if you are asked, click Yes to restart Anypoint Studio.
5. Select Project > Clean to rebuild the project.

View the local update site

6. Right-click the say-hello-connector project and select Properties.
7. Highlight the location path and copy it to the clipboard (type Ctrl+C).
8. Select Help > Install New Software.
9. In the Install dialog box, click the Add button and navigate to the location inside your workspace project (paste from the clipboard), then navigate inside the target/update-site folder.

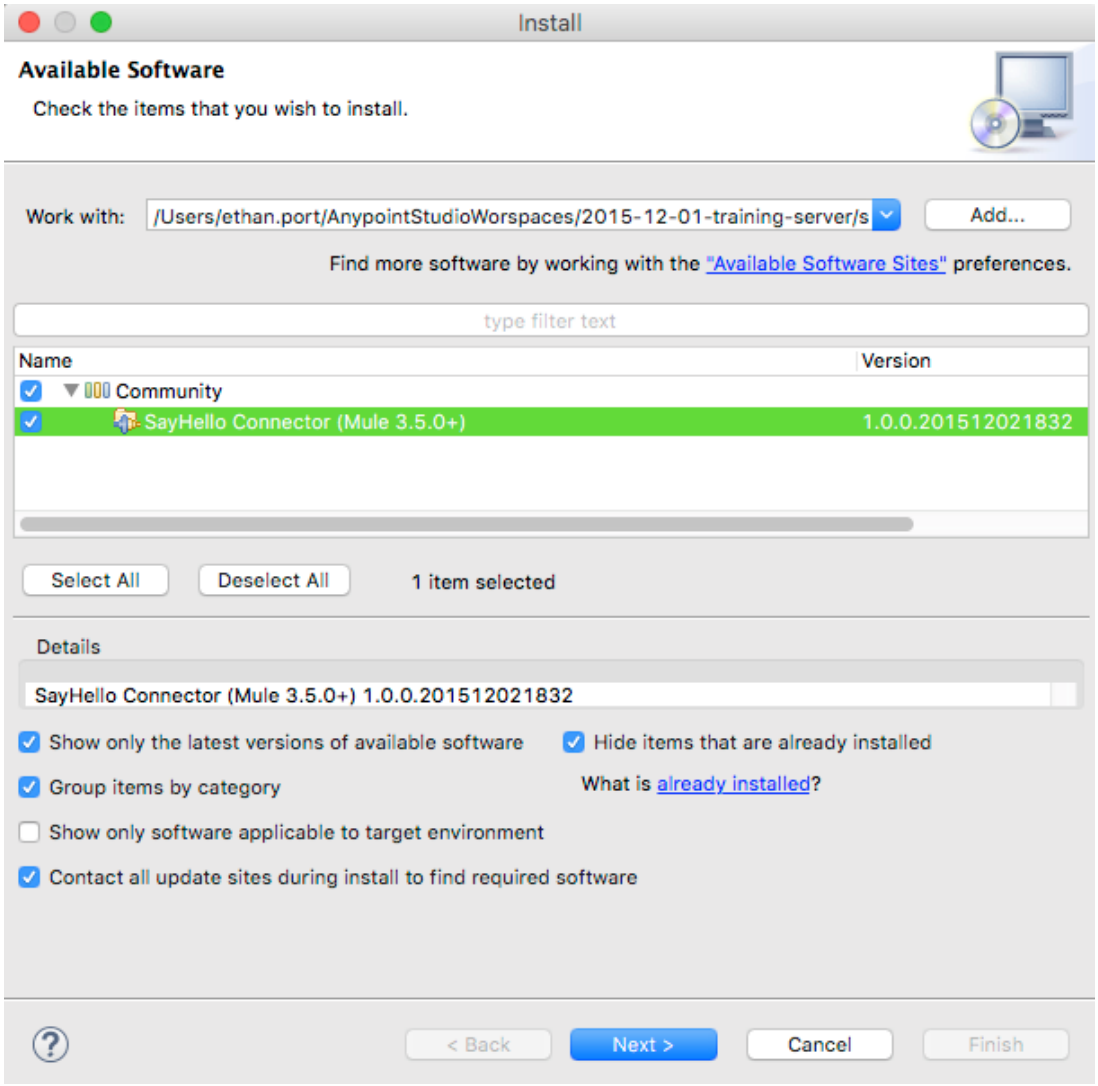


Note: The path you see in the update site name is the local location of your connector's update site.

Install the connector

10. Select the SayHello Connector and click Next twice.

11. Accept the license agreement and agree to the security warning.



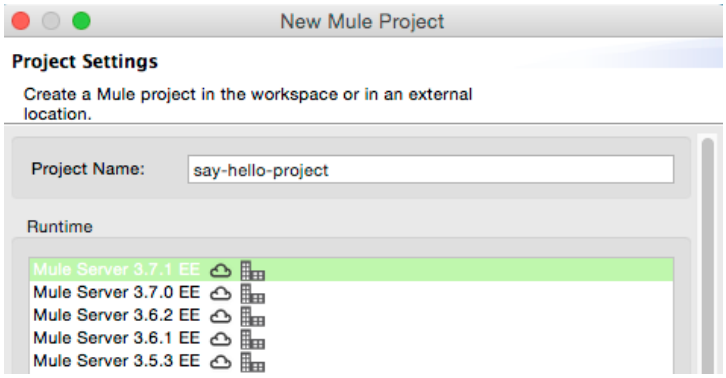
12. Restart Anypoint Studio.

    Note: Other users can install your connector using a local or online update site. Eventually, you would host this update site on an HTTP server so that it's accessible to all your developers.

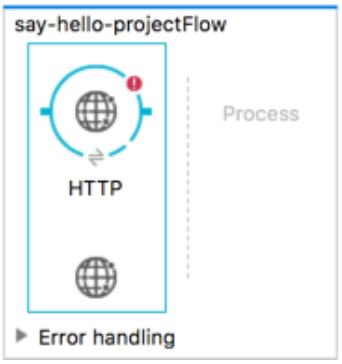
Add the connector to a Mule project

13. Select File > New > Mule Project.
14. In the New Mule Project dialog box, set the name to say-hello-project.

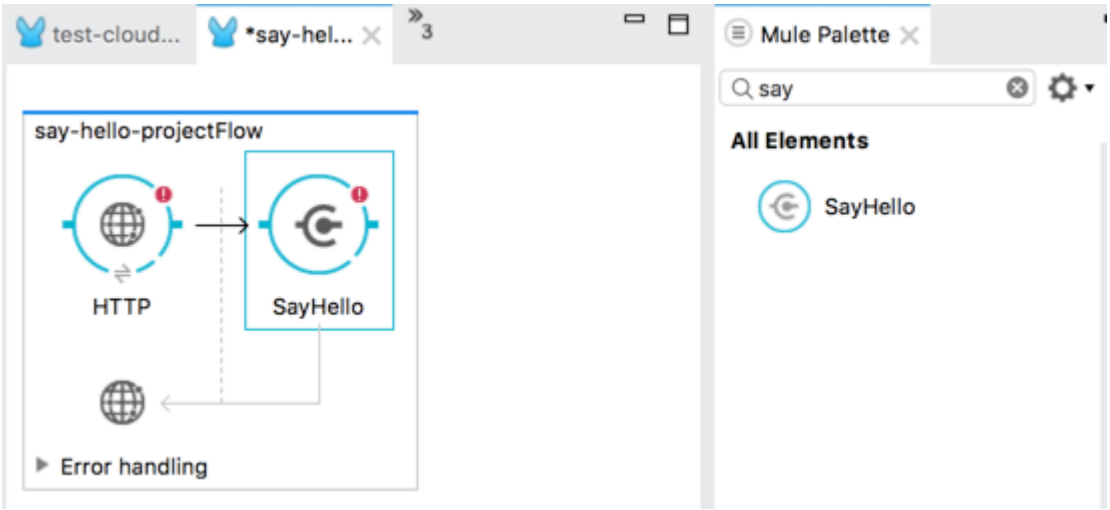
15. Select a runtime of 3.8.0 EE or greater and select Next.



16. Select a JavaSE-1.8 runtime and click Next.
17. Click Finish; the say-hello-project.xml configuration file should open.
18. Drag an HTTP connector into the canvas.



19. Drag the SayHello connector into the flow's process section.



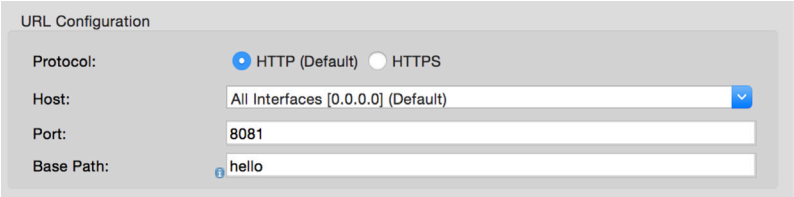
20. In the Package Explorer, examine the available libraries.

21. Expand the SayHello library and examine the imported jars.

Note: When you drag the SayHello connector into a Mule project, the required jars are automatically imported and added to the build path. If you were using Maven, these dependencies would be automatically added to the pom.xml.

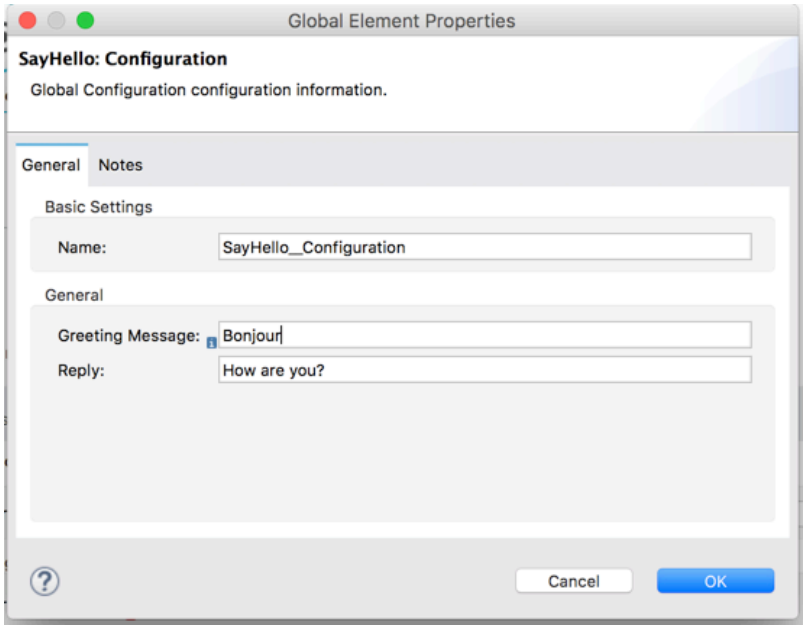
Configure the connectors

22. Double-click the HTTP connector.
23. In the Properties view, click the plus button for Connector Configuration.
24. In the Global Element Properties dialog box, set the base path to hello.



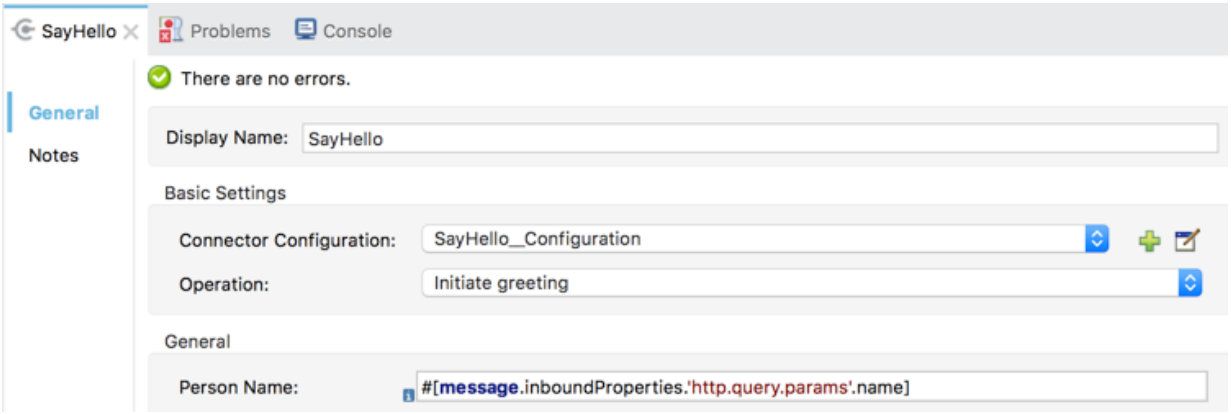
25. Double-click the SayHello connector endpoint.
26. In the Properties view, click the plus button for Connector Configuration.
27. In the Global Element Properties dialog box, change the Greeting Message to Bonjour.

Note: The default greeting message is Hello because this is the default annotation on the greetingMessage field in the connector.



28. Click OK.
29. In the Properties view, set the operation to Initiate greeting.
30. Set the value of the Person Name attribute to the value of a query parameter called name.

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



31. Ensure the configuration XML appears as shown here.

```
<http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081"
basePath="hello" doc:name="HTTP Listener Configuration"/>

<say-hello:config-type name="SayHello__Configuration" greetingMessage="Bonjour"
doc:name="SayHello: Configuration"/>
<flow name="say-hello-projectFlow">
       <http:listener config-ref="HTTP_Listener_Configuration" path="/"
       doc:name="HTTP"/>
       <say-hello:initiate-greeting config-ref="SayHello__Configuration"
       personName="#[message.inboundProperties.'http.query.params'.name]"
       doc:name="SayHello"/>
</flow>
```

32. Save the say-hello-project.

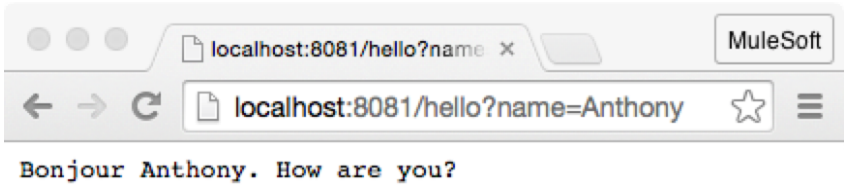
Run and test the connector

33. Right-click the say-hello-project and select Debug As > Mule Application.
34. Look at the console and verify the application deploys.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Mule is up and kicking (every 5000ms)                                +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
INFO  2015-02-01 11:25:44,326 [main] org.mule.module.launcher.StartupSummaryDeploymentListener:
**********************************************************************
*              - - + DOMAIN + - -               * - - + STATUS + - - *
**********************************************************************
* default                                       * DEPLOYED           *
**********************************************************************

********************************************************************************************************
*            - - + APPLICATION + - -            *       - - + DOMAIN + - -       * - - + STATUS + - - *
********************************************************************************************************
* say-hello-project                             * default                        * DEPLOYED           *
********************************************************************************************************
```

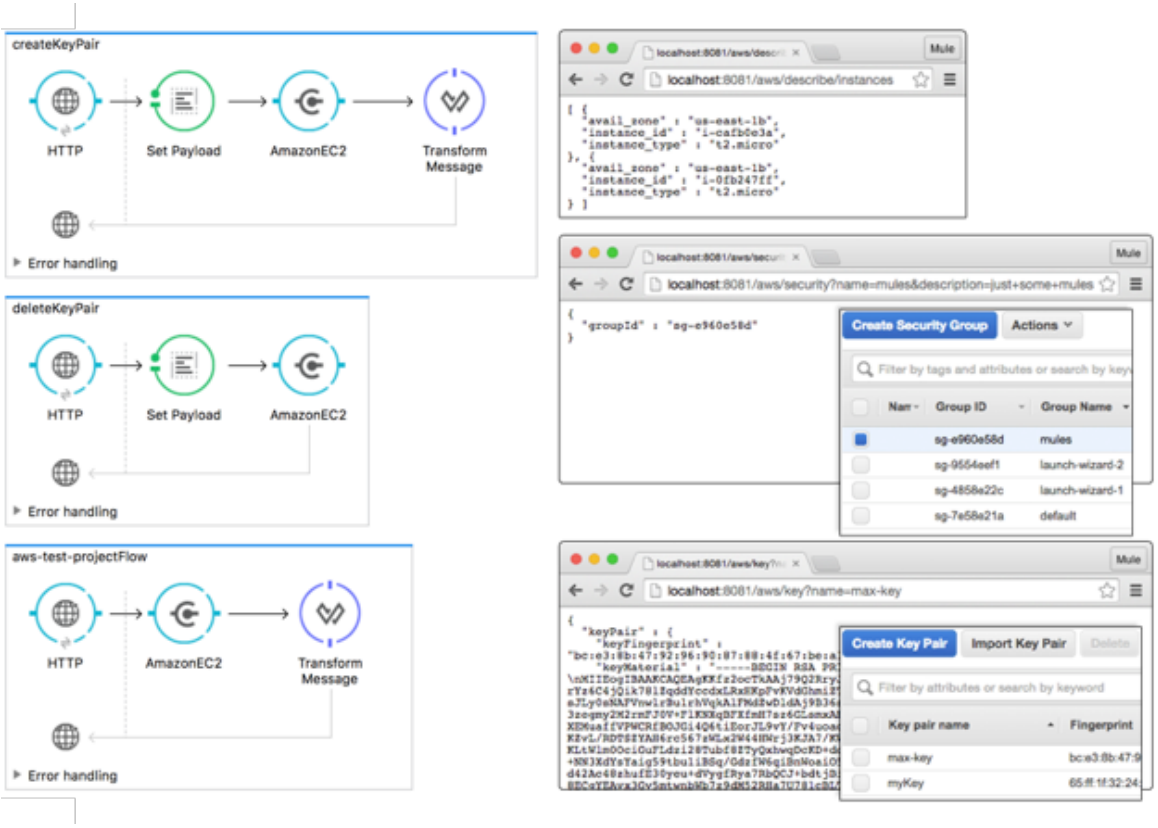
35. In a web browser, make a request to http://localhost:8081/hello?name=Anthony.

36. Verify the response.



37. Make another request with a different value for the name query parameter.
38. Stop the debugger.

Module 3: Developing an Anypoint Connector



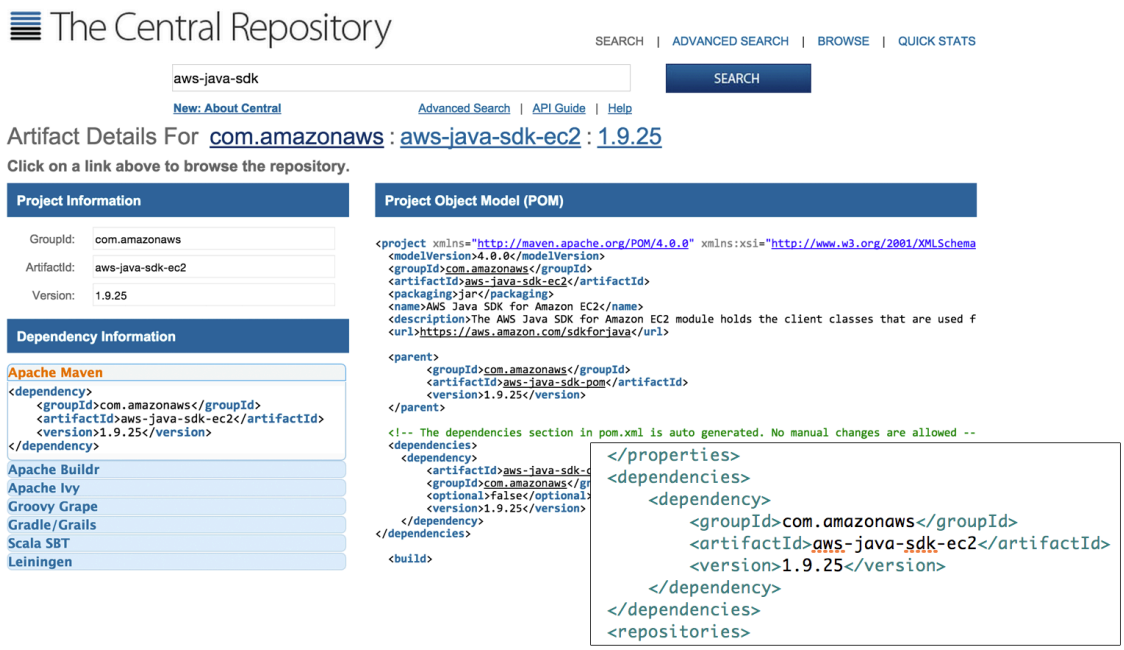
Objectives:

- Create a connector to integrate with an online service (Amazon EC2).
- Use a 3rd party SDK within a connector's code.
- Create multiple operations to support varying integration needs.
- Implement DataSense in a connector's operation.
- Install a connector into Anypoint Studio and test it from the perspective of an integration developer.

Walkthrough 3-1: Import an external dependency

In this walkthrough, you work with data from Amazon Web Services EC2 instances and leverage their library to create interaction between a connector and the backend resources. You will:

- Introduce dependencies to your pom.xml.
- Import the Amazon SDK dependency into your project using Maven.



Create an Amazon EC2 Account

1. If you do not already have an Amazon EC2 account, go to http://aws.amazon.com and register a new account.

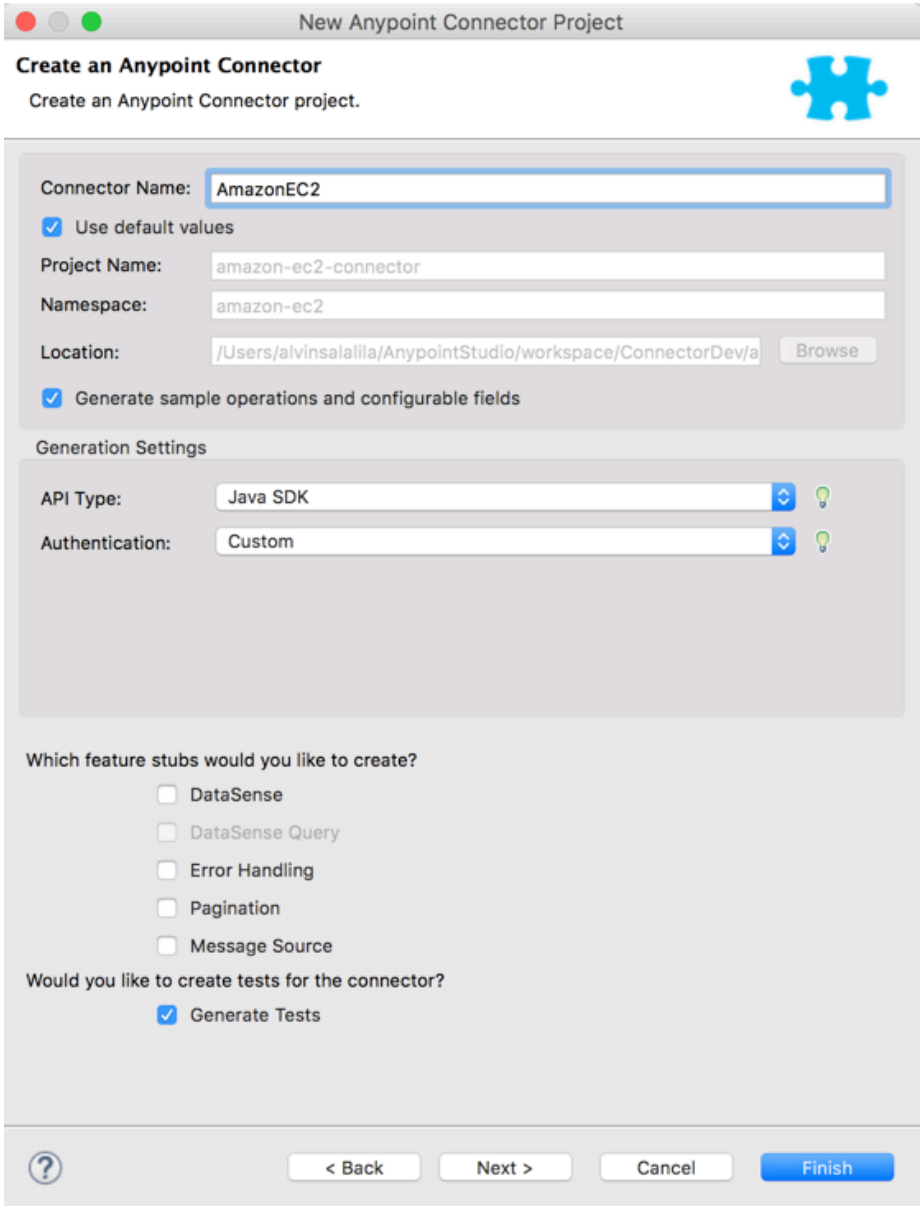
   Note: Although you can create a free Amazon EC2 account, you will need to use a credit card to sign up for the Amazon EC2 account. If you do not wish to do this, you can use the instructors trial account, but you will not be able to login to the Console in the later walkthrough steps.

2. Click Sign into the Console.
3. Enter the email address you want to use and your account password.
4. Click Sign in using our secure server.
5. Follow the steps to create a new free Amazon EC2 account.
6. Under your name choose Security Credentials and expand Access Keys (Access Key ID and Secret Access Key).
7. Click Create New Access Key.

8. Copy the Access Key ID and Secret Access Key into a local file (or into your credentials.txt file in your studentFiles folder).

Create a new connector project

9. Create a new Anypoint Connector project.
10. In the New Project dialog box, set the connector name to AmazonEC2.
11. Make sure Use default values and a type of Java SDK are selected.
12. For authentication, select Custom.



13. Check the Generate Tests checkbox.
14. Click Next twice.

15. Change the Connector Label from Connector to AmazonEC2.
16. Click Finish.
17. In the Console view, verify the Mule Connector project builds successfully.
18. In the Package Explorer notice your Connector project is named amazon-ec2-connector.
19. Expand the amazon-ec2-connector project.
20. Open org.mule.modules.amazonec2.config.ConnectorConfig.java and modify the @ConnectionManagement annotation to include the configElementName:

```
@ConnectionManagement(
   configElementName = "config-type",
   friendlyName = "Configuration"
)
public class ConnectorConfig {
```

Locate where to specify the external SDK (library) dependency

21. Open the project's pom.xml then select the Source tab.
22. Add a new line between the properties and repositories elements.
23. Add a dependencies element between the properties and repositories elements.

```
</properties>
<dependencies>

</dependencies>
<repositories>
```

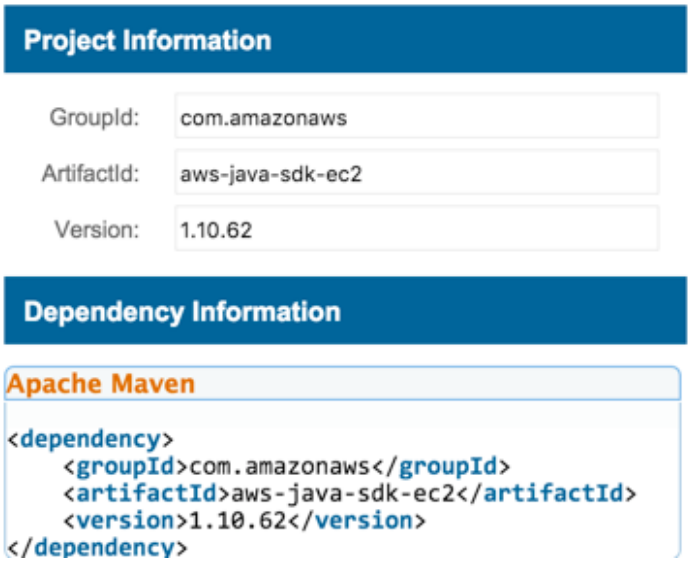
Note: You will add the amazon-sdk as a dependency of the connector project shortly. First, you need to locate the amazon-sdk.

Locate the Amazon SDK

24. In a web browser, navigate to http://search.maven.org/.
25. Search for aws-java-sdk-ec2.

26. Click the all link and search for version 1.10.62.

27. Locate the aws-java-sdk-ec2 ArtifactId for the com.amazonaws group.
28. Copy the dependency content for the 1.10.62 version dependency information from the Apache Maven section.




```
<dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-ec2</artifactId>
      <version>1.10.62</version>
</dependency>
```

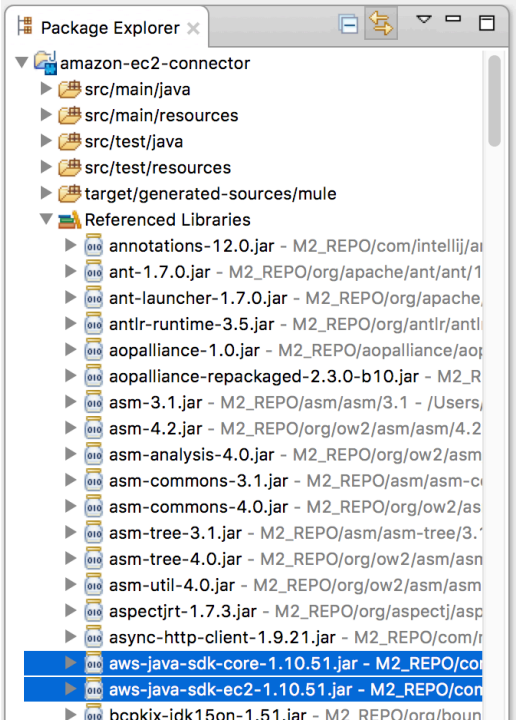
Add the SDK dependency

29. Return to pom.xml in Anypoint Studio.
30. Paste the dependency within the dependencies tags.

31. Save the amazon-ec2-connector project and wait for Anypoint Studio to retrieve any new dependencies.
32. Examine the console and verify your build is successful.

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 10.461 s
[INFO] Finished at: 2015-02-04T15:36:21-08:00
```

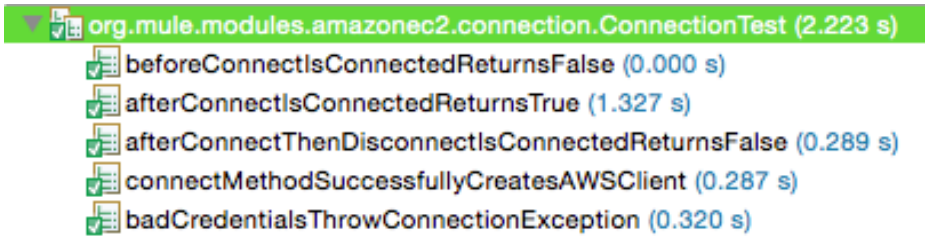
33. If you see any errors reported in the Problems tab, try running Project > Clean and see if the errors are resolved.
34. In the Package Explorer, expand the Referenced Libraries and locate the aws-java-sdk JAR files.



Walkthrough 3-2: Define and test a connection configuration

In this walkthrough, you configure the connector authentication. You will:

- Configure authentication logic used by your connector.
- Use connection tests to verify if connection is being established correctly.



Note: These tests are optional, but you do need to configure the connection information.

Import the connection tests

1. In the Package Explorer, expand src/test/java.
2. On your computer, navigate to the student files folder for the course and locate the tests/wt3-2/org directory.
3. Drag and drop the studentFiles/tests/wt3-2/org folder to the src/test/java folder of your AmazonEC2Connector project.

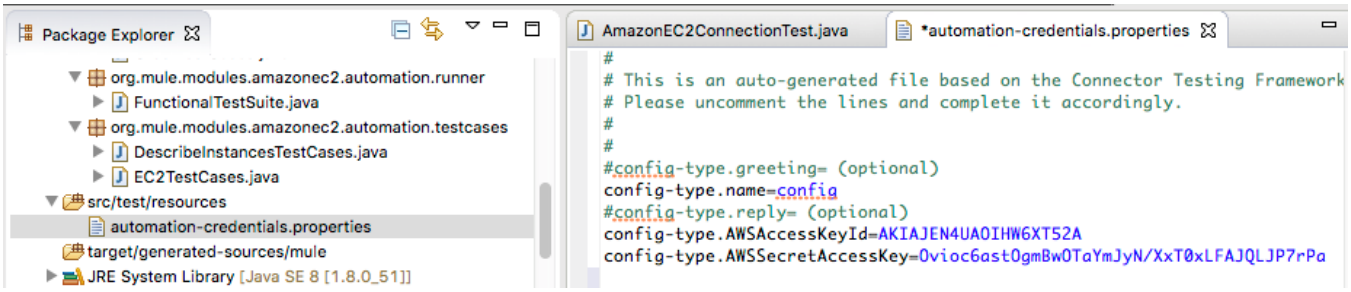
   Note: This adds several tests to your project. There should not be any Problems reported.
4. Some tests will produce Errors because they are used in later walkthroughs; either delete the test classes for now, or comment out the @Test methods in the tests that fail to compile.
5. Examine the tests; you will use these to build out and validate your AmazonEC2ConnectionTest.
6. Inside the org.mule.modules.amazonec2.automation.runner package, open the FunctionalTestSuite.java and notice the @RunWIth and @Suite annotations, with the EC2TestCases.class listed.
7. Notice that the AmazonEC2Connector.class is called in the initialiseSuite() method.

Add credentials

8. Return to the student files folder for the course, ConnDev3.7_studentFiles_{date}, and open credentials.txt.
9. Open src/test/resources/automation-credentials.properties.
10. Copy the AWSAccessKeyId and AWSSecretAccessKey lines from credentials.text into automation-credentials.properties.
11. Modify the entries to config.AWSAccessKeyId and config.AWSSecretAccessKey.

    Note: Do not put quotes around the values.

    ```
    config.AWSAccessKeyId=AKIAJEN4UAOIHW6XT52A
    config.AWSSecretAccessKey=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
    ```



12. Copy the setUp() method implementation from org.mule.amazonec2.automation.connection.AmazonEC2ConnectionTest.java into EC2TestCases.java.

    ```
    @Before
    public void setUp() {
       this.awsConnConfig = new ConnectorConfig();
    }
    ```

13. Copy the instance variables from AmazonEC2ConnectionTest.java into EC2TestCases.java.

    ```
    ConnectorConfig awsConnConfig;
    final static String awsKeyId = "AKIAJEN4UAOIHW6XT52A";
    final static String awsKeySecret = "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";
    ```

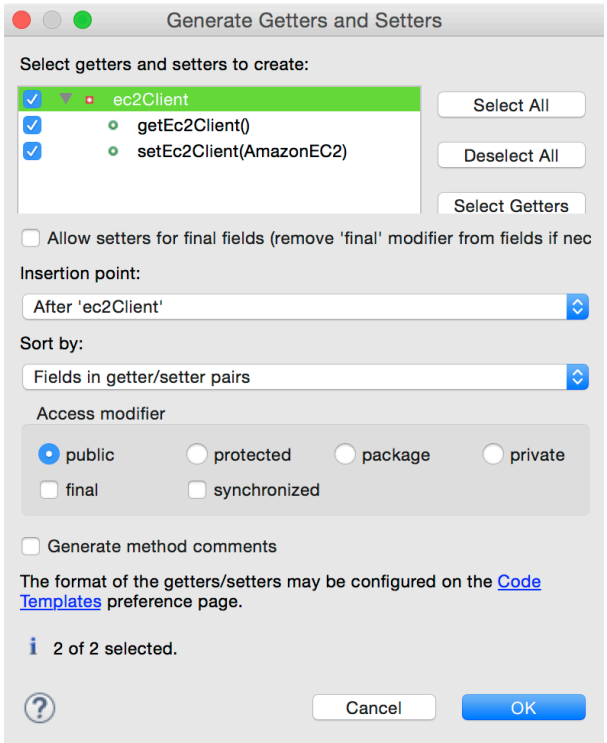
14. Copy and paste the AWSAccessKeyId and AWSAccessKeySecret values into the appropriate properties in AmazonEC2ConnectionTest.java.

    Note: These credentials are used for testing. Later when you add an AmazonEC2 Connector to a Mule flow, you will configure the credentials in the AmazonEC2 Connector's global configuration element.

Add a connection instance variable to the connector configuration

15. In the Package Explorer, expand src/main/java and org.mule.modules.amazonec2.config.
16. Open ConnectorConfig.java.
17. Add a private data member of type AmazonEC2 named ec2Client; be sure to import the appropriate class from the com.amazonaws.services.ec2 package.

18. Right-click ec2client and select Source > Generate Getters and Setters.
19. In the Generate Getters and Setters dialog box, check ec2Client.



20. Set the Insertion point to just after ‘getReply()’ and click OK.
21. Change the signature of the connect() method to accept parameters AWSAccessKeyId and AWSSecretAccessKey instead of username and password.
22. Add a line of code to the method to display a console message.

```
/**
 * Connect
 *
 * @param AWSAccessKeyId
 * @param AWSSecretAccessKey
 * @throws ConnectionException
 */
@Connect
@TestConnectivity
public void connect(@ConnectionKey String AWSAccessKeyId,
      @Password String AWSSecretAccessKey) throws ConnectionException {
   System.out.println("Connected!");
}
```

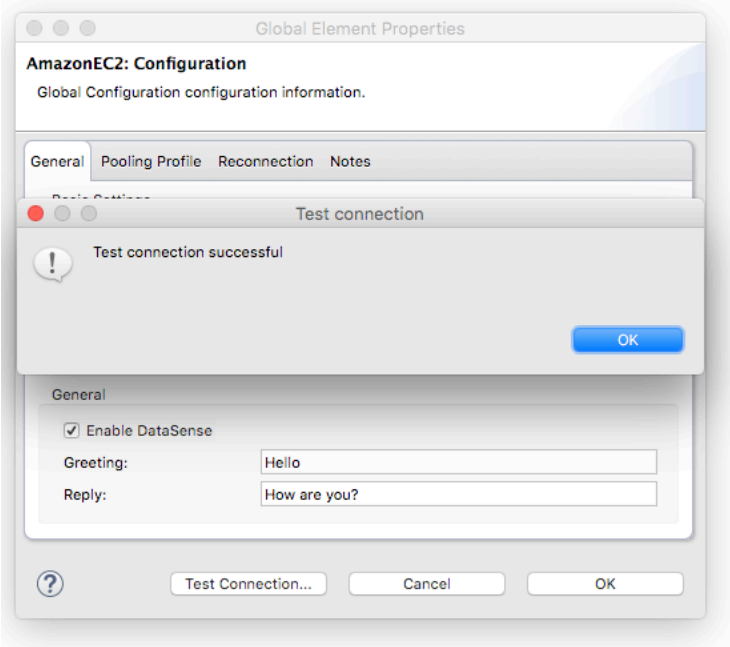
23. Save your changes to ConnectorConfig.java and AmazonEC2ConnectionTest.java in the amazon-ec2-connector project.
24. Right-click AmazonEC2ConnectionTest.java and choose Run As → JUnit Test.
25. Examine the test results and verify that some tests succeeded but the other tests failed.

    Note: We will continue to work on the tests. All the tests passing is critical to the basic functionality.

Walkthrough 3-3: Define connection configuration and credentials

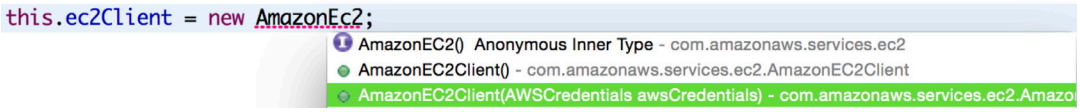
In this walkthrough, you configure authentication logic used by your connector. You will:

- Implement connection logic that creates a client session with AWS.
- Implement methods that end a client connection and validate whether any pre-existing connections exist.
- Update the connector into Anypoint Studio.



Implement connect()

1. Open ConnectorConfig.java.
2. Locate the connect() method.
3. Set `this.ec2Client = new AmazonEC2Client();`



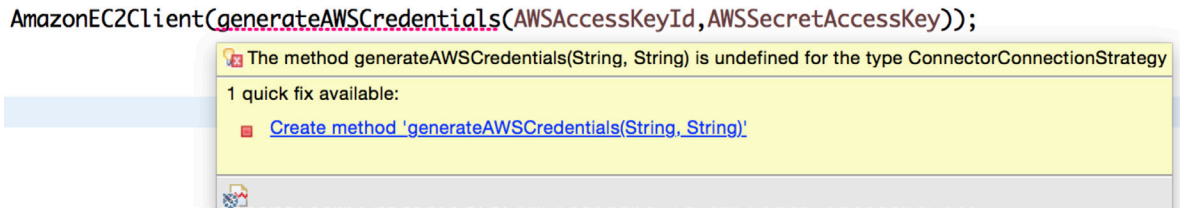
4. Insert the argument of generateAWSCredentails(AWSAccessKeyId, AWSSecretAccessKey).

```
@Connect
@TestConnectivity
public void connect(
        @ConnectionKey
        String AWSAccessKeyId,
        @Password
        String AWSSecretAccessKey)
                throws ConnectionException {
    this.ec2Client = new AmazonEC2Client(
            generateAWSCredentials(AWSAccessKeyId, AWSSecretAccessKey)
            );
}
```

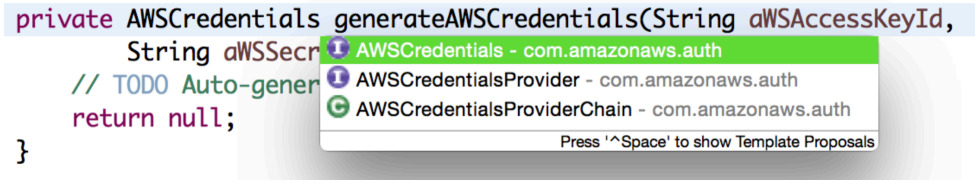
Note: generateAWSCredentials() is a method that you'll implement to use the AWS SDK's BasicAWSCredentials class. This will satisfy the need to provide the AmazonEC2Client with the AWSCredentials.

Create and implement generateAWSCredentials()

5. Hover over generateAWSCredentials to trigger tooltips.



6. Click Create method 'generateAWSCredentials(String,String)'.
7. Move the newly generated method to the bottom of the class.
8. Change generateAWSCredential's return type to AWSCredentials; make sure to import the com.amazonaws.auth package.



9. Add the implementation below into generateAWSCredentials().

```
if(aWSAccessKeyId == null || aWSSecretAccessKey == null) {
  System.out.println("Credentials are null");
  return null;
}
return new BasicAWSCredentials(aWSAccessKeyId, aWSSecretAccessKey);
```

```
private AWSCredentials generateAWSCredentials(String aWSAccessKeyId, String aWSSecretAccessKey) {
    if(aWSAccessKeyId == null || aWSSecretAccessKey == null) {
        System.out.println("Credentials are null");
        return null;
    }
    return new BasicAWSCredentials(aWSAccessKeyId, aWSSecretAccessKey);
}
```

10. Locate isConnected().
11. Update the implementation to return true when client is not null.

```
/**
 * Are we connected
 */
@ValidateConnection
public boolean isConnected() {
    return ec2Client != null;
}
```

Note: When the connect() method is successfully called, you will be pointing the ec2Client variable to an instance of AmazonEC2Client. Thus, validating the connection corresponds to testing if the ec2Client instance references an AmazonEC2Client instance (is not null).

12. Locate disconnect().
13. Update the implementation to ensure ec2Client no longer references an instance of AmazonEC2Client.

```
/**
 * Disconnect
 */
@Disconnect
public void disconnect() {
    this.ec2Client = null;
}
```

14. Locate the connect() method.

15. Add the following code:

```
if (isConnected()) System.out.println("Connected!");
```

```
/**
 * Connect
 *
 * @param AWSAccessKeyId
 * @param AWSSecretAccessKey
 * @throws ConnectionException
 */
@Connect
@TestConnectivity
public void connect(@ConnectionKey String AWSAccessKeyId,
      @Password String AWSSecretAccessKey) throws ConnectionException {
   this.ec2Client = new AmazonEC2Client(generateAWSCredentials(AWSAccessKeyId, AWSSecretAccessKey));
   if (isConnected()) System.out.println("Connected!");
}
```

16. Run AmazonEC2ConnectionTest again and examine the results.

Note: At this point, credentials are being passed, but your connect() method doesn't make an external call. Thus, if bad credentials are provided, you really aren't aware until you call an operation in your flow. Let's change that.

Add an external call to connect() and rerun tests

17. In ConnectorConfig.java, locate the connect() method.
18. Append the following code to the bottom of the method body.

```
this.ec2Client.describeAvailabilityZones();
```

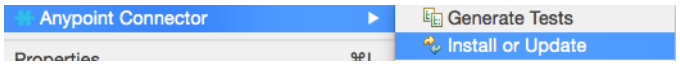
```
@Connect
@TestConnectivity
public void connect(
        @ConnectionKey
         String AWSAccessKeyId,
        @Password
        String AWSSecretAccessKey)
                throws ConnectionException {
    this.ec2Client = new AmazonEC2Client(
            generateAWSCredentials(AWSAccessKeyId, AWSSecretAccessKey));
    this.ec2Client.describeAvailabilityZones();
}
```

Note: This introduces a call which requires authentication into your connect contract. If your credentials are invalid, an AmazonServiceException should be thrown from the client. Thrown exceptions signify connection failure.

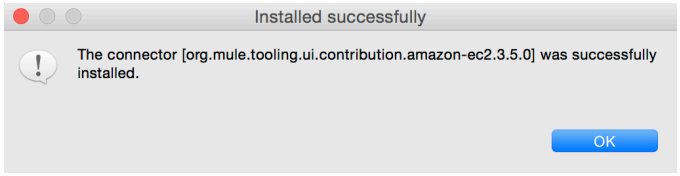
19. Rerun tests a final time and examine the results; all tests should now pass.

Install or update the connector in Anypoint Studio

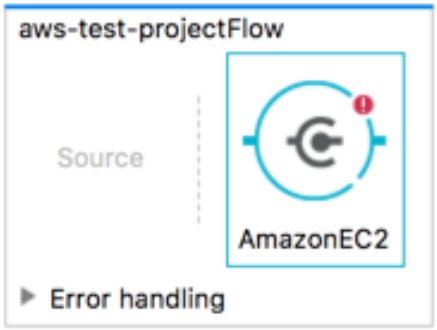
20. Right-click the amazon-ec2-connector project.
21. Select Anypoint Connector > Install or Update.



22. Verify you see a pop-up message that the amazon-ec2 connector was updated.

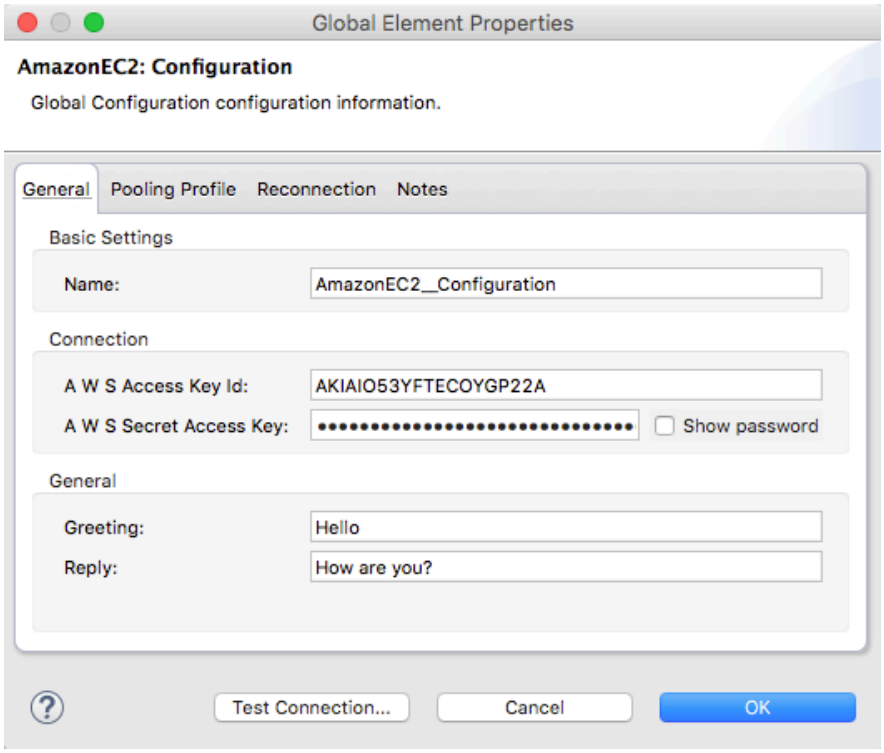


23. Install the new connector into Anypoint Studio.
24. If you see a dialog to restart Anypoint Studio, restart studio and open the same workspace.
25. Create a new Mule Project named aws-test-project.
26. Drag in a flow.
27. Drag in your AmazonEC2 Connector.



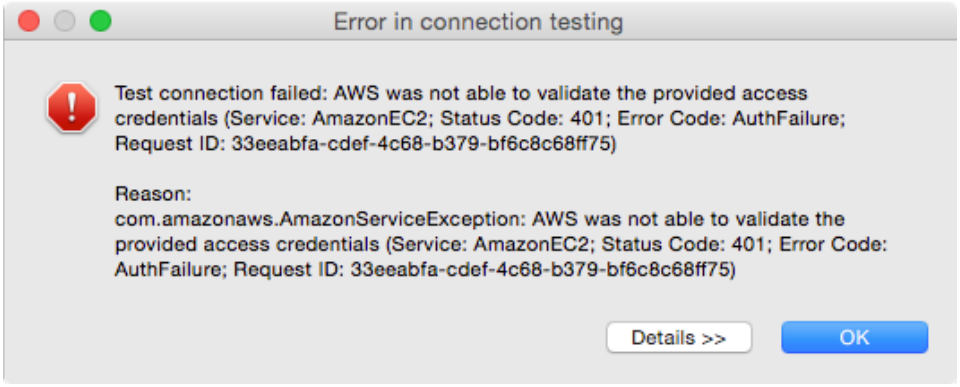
28. Select the AmazonEC2 endpoint.
29. Create a new Connector Configuration.

30. Insert the AWSAccessKeyId and AWSSecretAccessKey from your studentFiles.



Note: The 'AWS' has unwanted spaces in the label. You will fix this later with an additional annotation in your code.

31. Click Test Connection.
32. Verify the test connection was successful.
33. Click OK.
34. Change a character in the A W S Secret Access Key to make it invalid.
35. Click Test Connection.

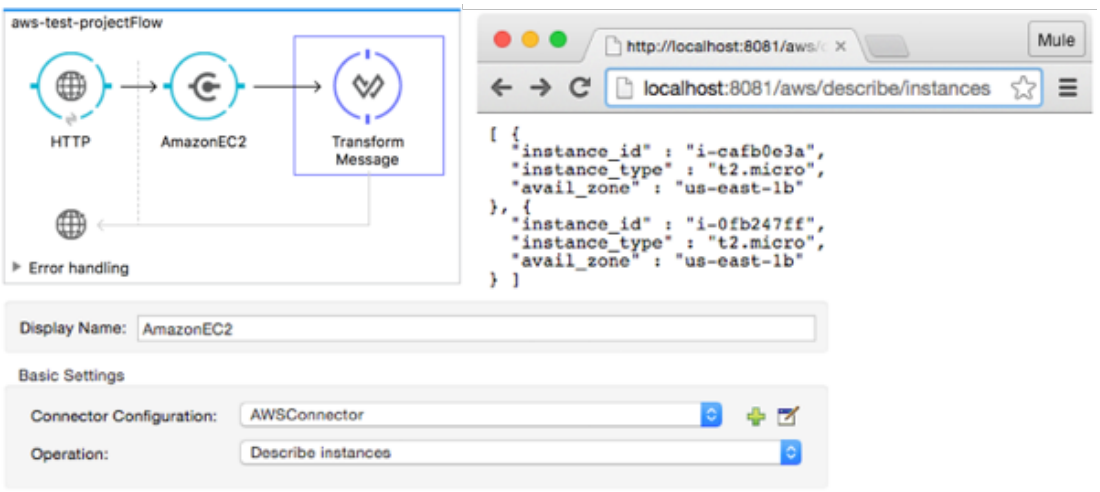


36. Verify that the user receives a message that AWS was not able to authenticate.
37. Change the A W S Secret Access Key back.
38. Click OK.

Walkthrough 3-4: Add an operation

In this walkthrough, you implement your first operation. You will:

- Reference the API client from the @Connector class.
- Implement an @Processor method.
- Transform the POJO connector result to JSON.



Reference the EC2 client connection

1. Expand src/main/java.
2. Under the org.mule.modules.amazonec2 package, edit AmazonEC2Connector.java.
3. Add a private field named client of type AmazonEC2.

```
@Connector(name="amazon-ec2", friendlyName="AmazonEC2")
public class AmazonEC2Connector
{

    private AmazonEC2 client;
```

4. Locate the setConfig() method.
5. Set the client to an instance of AmazonEC2 within the setConfig method.

```
public void setConfig(ConnectorConfig config) {
    this.config = config;
    this.client = config.getEc2Client();
}
```

Note: You now have reference to our client directly from the @Connector class. While this isn't a requirement that affects functionality, it saves you from needing to call config.getEc2Client() in every operation.

Develop an @Processor annotated method

6. Locate the default @Processor method greet().

```
@Processor
public String greet(String friend) {
    /*
     * MESSAGE PROCESSOR CODE GOES HERE
     */
    return config.getGreeting() + " " + friend + ". " + config.getReply();
}
```

7. Change the method name to describeInstances.
8. Remove all parameters from the method.
9. Return client.describeInstances() within the method's implementation.
10. Hover over client.describeInstances().

```
@Processor
public String describeInstances() {
    return client.describeInstances();
}
```

11. Click Change method return type to DescribeInstancesResult.

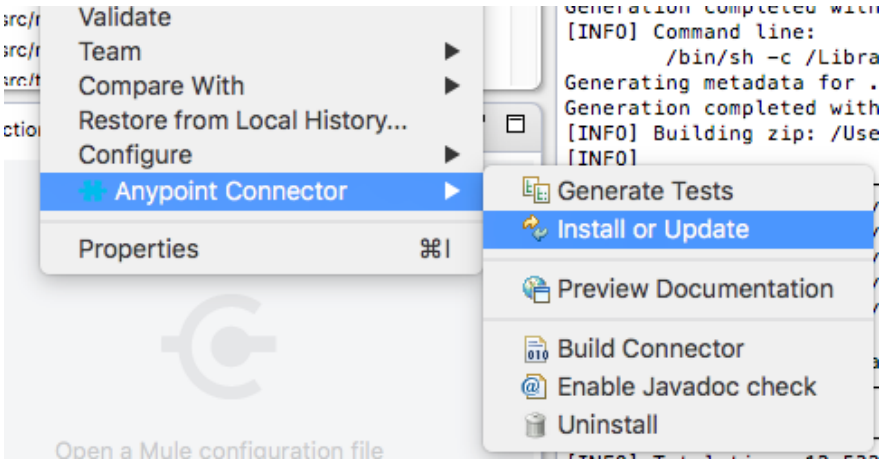
    Note: Make sure Anypoint Studio also imports com.amazonaws.services.ec2.model.DescribeInstancesResult.

12. Remove the private properties greeting and reply, including their annotations and the corresponding getter and setter methods.

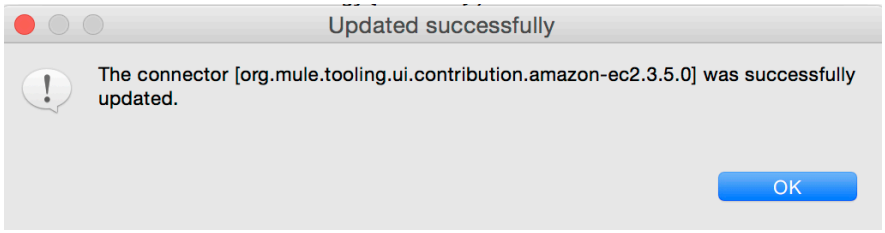
Test and update the connector

13. Fix any errors that appears in the test classes by changing the assertions into assertNotNull(getConnector().describeInstances()).
14. Run the JUnit Tests and examine the results; there should be only 1 failed test at this point.
15. Right-click the amazon-ec2-connector project.

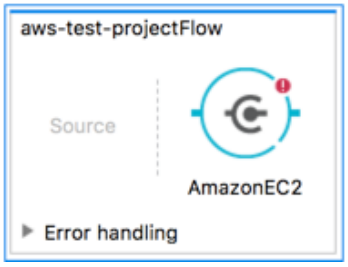
16. Select Anypoint Connector > Install or Update.



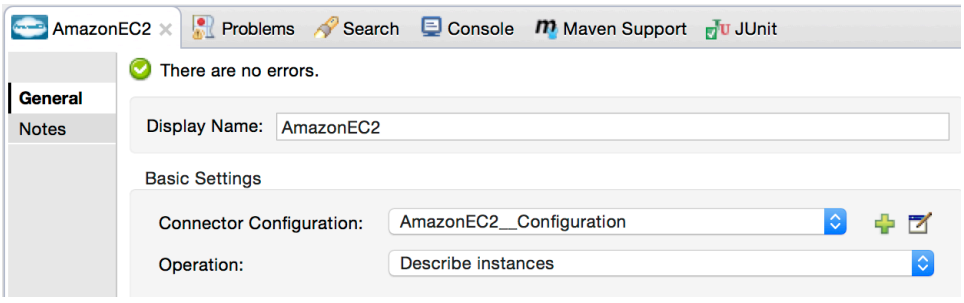
17. Update the connector and verify you see a success dialog message.



18. Open the aws-test-project.
19. Select the AmazonEC2 endpoint from within the aws-test-projectFlow.

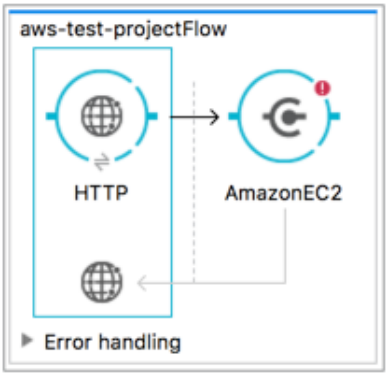


20. Select Describe Instances from the Operation drop-down menu.

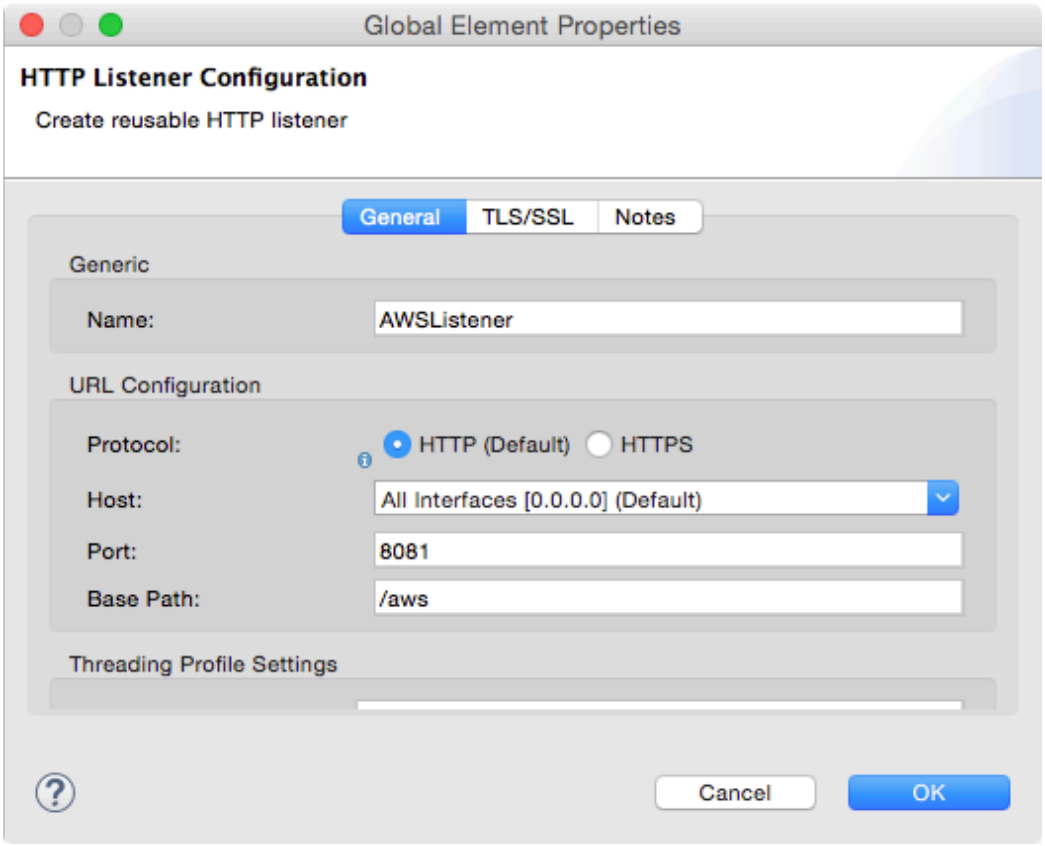


Note: Recall this is our @Processor annotated method named describeInstances(). The method name is propagated into Anypoint Studio's UI as an operation.

21. Drop an HTTP connector endpoint into the flow's source.

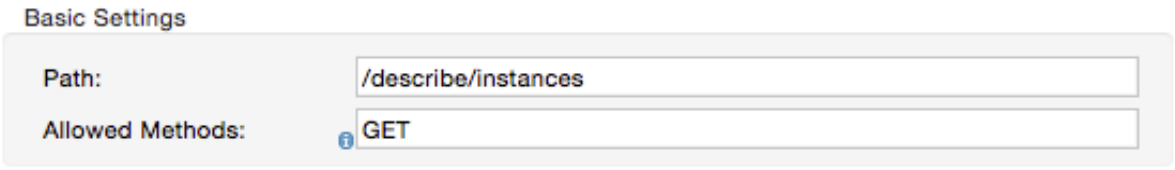


22. In the Properties view, click the Add button to create a new Connector Configuration.
23. In the Global Elements dialog box, set name to AWSListener.
24. Set Base Path to /aws.



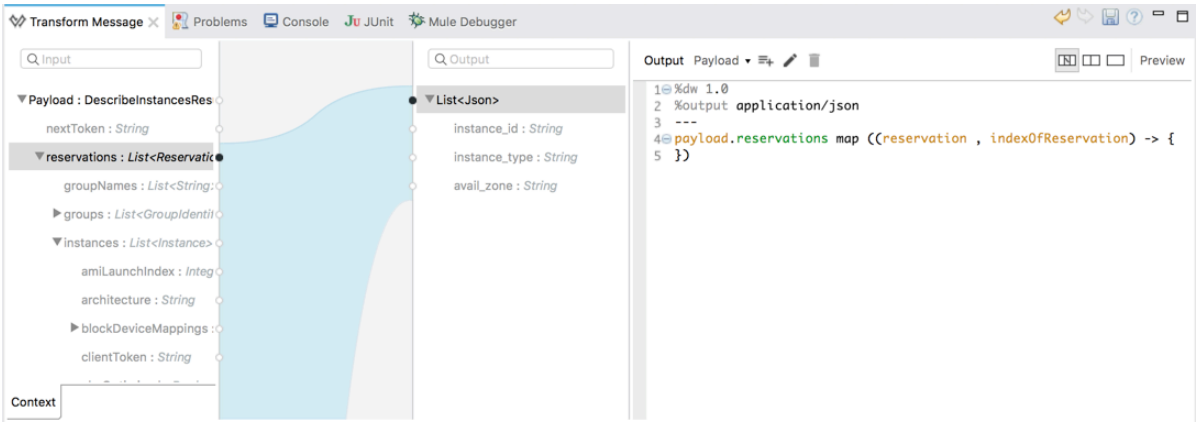
25. Click OK.
26. Set the connector endpoint's path to /describe/instances.

27. Set allowed methods to GET.

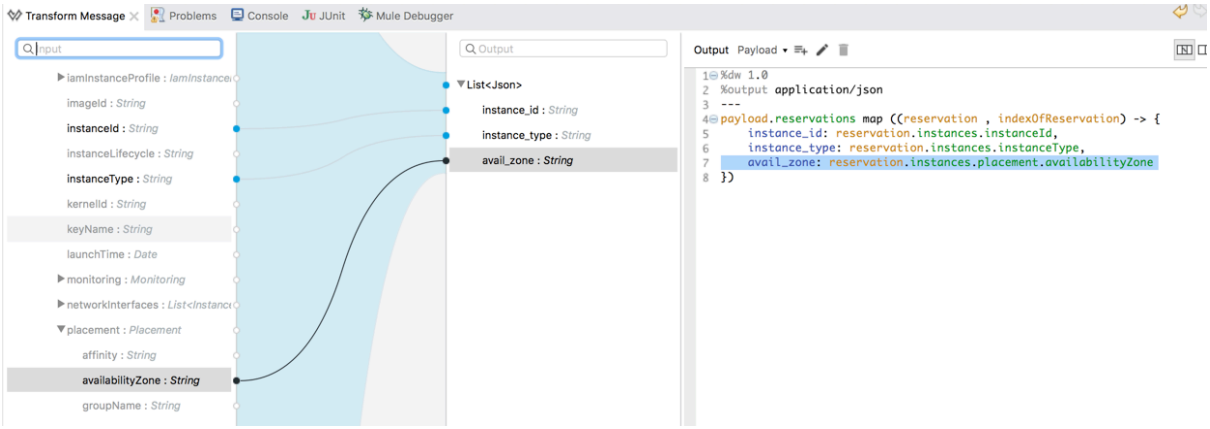


28. From your studentFiles folder, copy or drag file aws-instances-example.json to src/main/resources.
29. Drag a Transform Message component to the right of the AmazonEC2 connector endpoint.
30. In the output section, click Define metadata.

31. In the Select metadata type dialog box, select Create new type.
32. Configure the type as follows:
    - Type: JSON
    - Type Id: json-aws-instances
    - Example: aws-instances-example.json
33. Drag reservations : List<Reservation> from the input section to List<Json> in the output section.



34. Map instanceId to instance_id, instanceType to instance_type and availabilityZone to avail_zone from input to output.



35. Save the project.
36. Run aws-test-project.
37. Make a request to http://localhost:8081/aws/describe/instances.

Walkthrough 3-5: Implement a DataSense-enabled operation

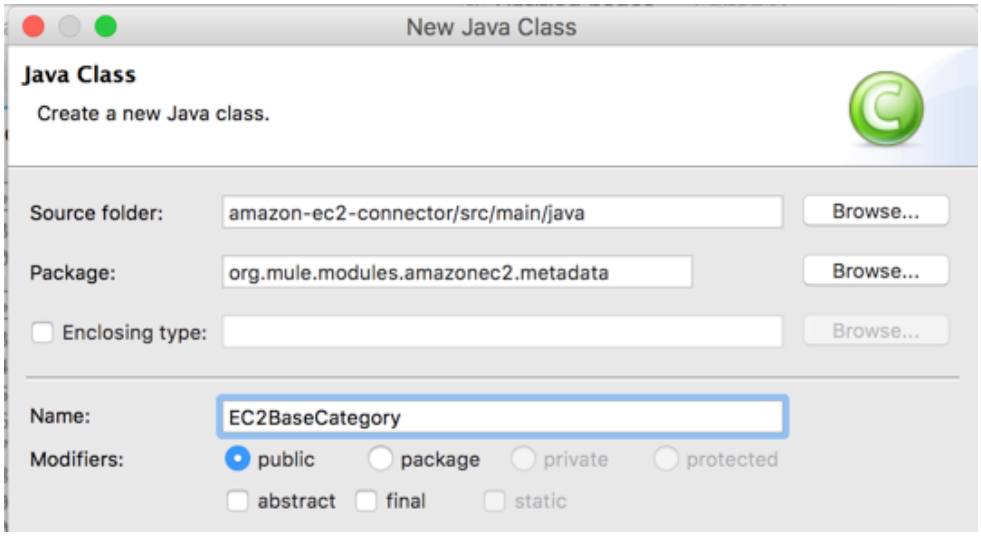
In this walkthrough, you implement another operation in the connector – this time, using DataSense. You will:

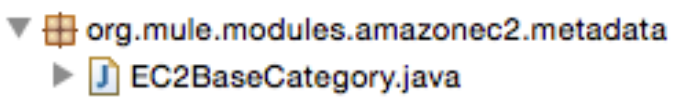
- Create a second @Processor method into the @Connector class.
- Configure metadata retrieval.
- Configure metadata awareness.
- Hook a processor into metadata retrieval.


```
@MetaDataKeyRetriever
public List<MetaDataKey> getEntities() throws Exception

@MetaDataRetriever
public MetaData describeEntity(MetaDataKey entityKey)

@MetaDataOutputRetriever
public MetaData describeEntityOutput(MetaDataKey entityKey) throws Exception
```

Configure metadata retrieval

1. Return to the amazon-ec2-connector project.
2. Expand src/main/java.
3. Create a new package named org.mule.modules.amazonec2.metadata.
4. Create a new class named EC2BaseCategory in the org.mule.modules.amazonec2.metadata package.





5. Annotate the new class with @MetaDataCategory from the org.mule.api.annotations.components package.
6. Add a private field named connector of type AmazonEC2Connector
7. Annotate the connector field with @Inject from the javax.inject package.

```
@MetaDataCategory
public class EC2BaseCategory {

    @Inject
    private AmazonEC2Connector connector;
```

8. Below the connector field, add the @MetaDataKeyRetriever from the org.mule.api.annotations package.
9. Create a public method named getEntities.
10. Specify a return type of List<MetaDataKey>.

    Note: MetaDataKey is from the org.mule.common.metadata package.

11. Declare the method to throw type Exception.

```
@MetaDataKeyRetriever
public List<MetaDataKey> getEntities() throws Exception {

}
```

12. Create a new variable named entities of type List<MetaDataKey>; where List is of type java.util.List.
13. Set entities equal to a new instance of ArrayList<MetaDataKey>.

```
@MetaDataKeyRetriever
public List<MetaDataKey> getEntities() throws Exception {
    List<MetaDataKey> entities = new ArrayList<>();
}
```

    Note: If your project is set to JDK 1.6 compliance you'll need to specify the type within ArrayList<>().

14. Add to entities a new instance of DefaultMetaDataKey with the arguments "KeyPair_id","Key Pair".
15. Add to entities a new instance of DefaultMetaDataKey with the arguments "SecurityGroup_id","Security Group".

16. Return entities from the method.

```
@MetaDataKeyRetriever
public List<MetaDataKey> getEntities() throws Exception {
    List<MetaDataKey> entities = new ArrayList<>();
    entities.add(new DefaultMetaDataKey("KeyPair_id","Key Pair"));
    entities.add(new DefaultMetaDataKey("SecurityGroup_id","Security Group"));
    return entities;
}
```

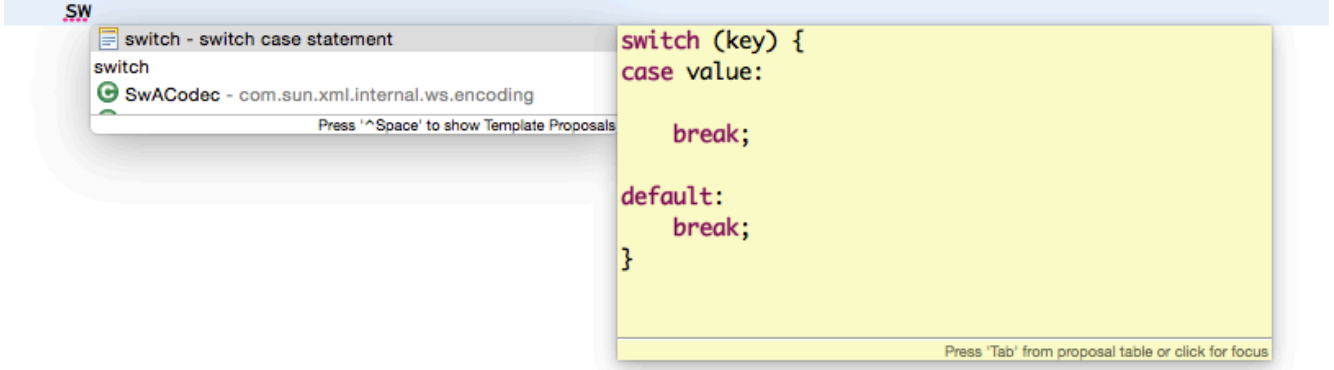
Note: This method will be used by Anypoint Studio to obtain which entities are available for use on methods using @MetaDataKeyParam.

17. Add the @MetaDataRetriever annotation under the getEntites method.
18. Create a public method named describeEntity which returns type MetaData from the org.Mule.common.metadata package.
19. Add the parameter entityKey of type MetaDataKey to the method declaration.

```
@MetaDataRetriever
public MetaData describeEntity(MetaDataKey entityKey)   {
}
```

20. Add a variable named model of type MetaDataModel.
21. Generate a switch statement using Eclipse's code templates.

```
@MetaDataRetriever
public MetaData describeEntity(MetaDataKey entityKey)   {
    MetaDataModel model;
    sw
```



Note: Template proposals are typically accessible by triggering Ctrl+Space.

22. Add to the switch entityKey.getId().

23. Add the following cases below.

```
@MetaDataRetriever
public MetaData describeEntity(MetaDataKey entityKey)    {
    MetaDataModel model;
    switch (entityKey.getId()) {
    case "KeyPair_id":
        model = new DefaultMetaDataBuilder().
        createPojo(CreateKeyPairRequest.class).build();
        break;
    case "SecurityGroup_id":
        model = new DefaultMetaDataBuilder().
        createPojo(CreateSecurityGroupRequest.class).build();
        break;
```

Note: The switch statement ensures the correct MetaData model is built based on which entityKey was retrieved for a given operation.

24. Set the default case to throw a descriptive runtime exception such as the one below:

```
        default:
            throw new RuntimeException(
                    String.format("This entity %s is not supported",
                            entityKey.getId()));
        }
```

25. Return a new instance of DefaultMetaData with the model variable as the argument.

```
@MetaDataRetriever
public MetaData describeEntity(MetaDataKey entityKey)   {
    MetaDataModel model;
    switch (entityKey.getId()) {
    case "KeyPair_id":
        model = new DefaultMetaDataBuilder().
        createPojo(CreateKeyPairRequest.class).build();
        break;
    case "SecurityGroup_id":
        model = new DefaultMetaDataBuilder().
        createPojo(CreateSecurityGroupRequest.class).build();
        break;
    default:
        throw new RuntimeException(
                String.format("This entity %s is not supported",
                        entityKey.getId()));
    }
    return new DefaultMetaData(model);
}
```

Note: When an entityKey is looked up, we're now able to construct the metadata model for the input. What about the output? It's safe to assume our connector operations won't receive the same metadata as the respond with. Let's look into specifying the output.

26. Below the @MetaDataRetriever method add a MetaDataOutputRetriever annotation.
27. Create a method named describeEntityOutput which returns MetaData.
28. Add a parameter named entityKey of type MetaDataKey.
29. Specify that the method throws type Exception.
30. Add a variable model of type MetaDataModel to the method's implementation.

```
@MetaDataOutputRetriever
public MetaData describeEntityOutput(MetaDataKey entityKey) throws Exception {
```

31. Copy and paste the @MetaDataRetriever's method implementation into the new @MetaDataOutputRetriever's method implementation.

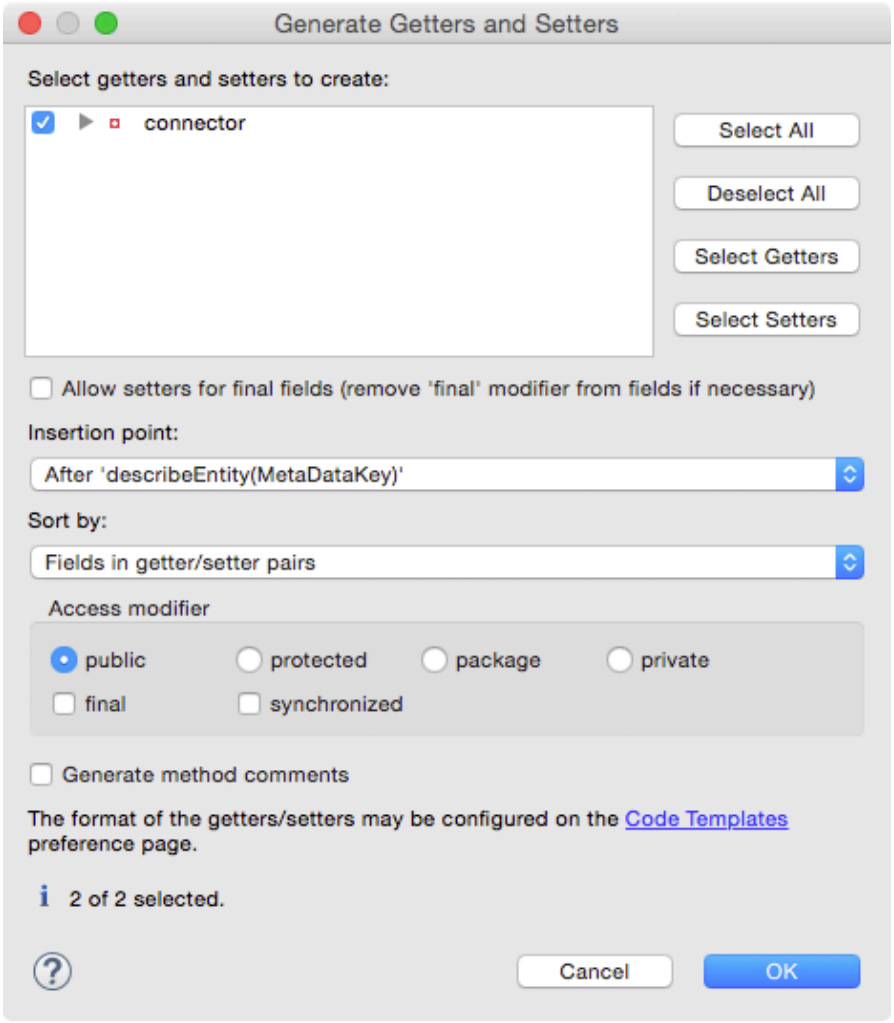
32. Replace the Create{aws-resource}**Request**.class arguments with Create{aws-resource}**Result**.class.

```
@MetaDataOutputRetriever
public MetaData describeEntityOutput(MetaDataKey entityKey) throws Exception{
    MetaDataModel model;
    switch (entityKey.getId()) {
    case "KeyPair_id":
        model = new DefaultMetaDataBuilder().
        createPojo(CreateKeyPairResult.class).build();
        break;
    case "SecurityGroup_id":
        model = new DefaultMetaDataBuilder().
        createPojo(CreateSecurityGroupResult.class).build();
        break;
    default:
        throw new RuntimeException(
                String.format("This entity %s is not supported", entityKey.getId()));
    }
    return new DefaultMetaData(model);
}
```

Note: You'll soon see when createKeyPair or createSecurityGroup is triggered from the connector, this method will allow Anypoint Studio to be aware of the endpoint's return type.

33. Click Source > Generate Getters and Setters from the menu bar.

34. Check the connector checkbox.



35. Click OK.
36. Save EC2BaseCategory.java.

Write a metadata-aware operation

37. Expand org.Mule.modules.amazonec2.
38. Open AmazonEC2Connector.java.

39. Add @MetaDataScope(EC2BaseCategory.class) above the @Connector annotation.

```
@MetaDataScope(EC2BaseCategory.class)
@Connector(name="amazon-ec2", friendlyName="AmazonEC2")
public class AmazonEC2Connector
{
```

Note: Connectors can pull from multiple metadata categories. You can tell a particular processor which metadata category to introspect from by adding the @MetaDataScope annotation above the operation's @Processor annotation.

40. Locate the describeInstances() method.
41. Add two line breaks after describeInstances() method.
42. Add an @Processor annotation.
43. Create a public method named create which returns type Object.
44. Add the parameter entityType of type String with the annotation @MetaDataKeyParam.
45. Add the argument affects=MetaDataKeyParamAffectsType.BOTH to the @MetaDataKeyParam annotation.

```
@Processor
@MetaDataScope(EC2BaseCategory.class)
public Object create(
        @MetaDataKeyParam(
                affects=MetaDataKeyParamAffectsType.BOTH
                ) String entityType,
```

Note: The affects argument will ensure that the MetaDataKey lookup will provide studio with metadata for both input and output of the connector endpoint.

46. Add a second parameter named requestData of type Object with the annotation @Default("#[payload]"); make sure to import the org.mule.api.annotations.param.Default annotation.

```
@Processor
@MetaDataScope(EC2BaseCategory.class)
public Object create(
        @MetaDataKeyParam(
                affects=MetaDataKeyParamAffectsType.BOTH
                ) String entityType,
        @Default("#[payload]") Object requestData){
```

47. Within the method's implementation, use Eclipse's code templates to generate a switch statement (type swit then press Ctrl+Space).
48. Set the switch condition to entityType.
49. Create a case for "KeyPair_id" and "SecurityGroup_id".
50. Have the default case break.

```
switch (entityType) {
case "KeyPair_id":
case "SecurityGroup_id":
default:
    break;
}
```

51. Add into the "KeyPair_id" case a return of createKeyPair((CreateKeyPairRequest) requestData).
52. Add into the "SecurityGroup_id" case a return of createSecurityGroup((CreateSecurityGroupRequest) requestData).
53. If a return statement is not triggered, ensure the method throws a RuntimeException.

```
@Processor
@MetaDataScope(EC2BaseCategory.class)
public Object create(
        @MetaDataKeyParam(
                affects=MetaDataKeyParamAffectsType.BOTH
                ) String entityType,
        @Default("#[payload]") Object requestData){

    switch (entityType) {
    case "KeyPair_id":
        return createKeyPair((CreateKeyPairRequest) requestData);
    case "SecurityGroup_id":
        return createSecurityGroup((CreateSecurityGroupRequest) requestData);
    default:
        break;
    }

    throw new RuntimeException("Entity not recognized");

}
```

Note: Based on the user's choice, we'll cast to the appropriate request type expected by the amazon client and call a method that we can implement the logic for.

54. Hover over createKeyPair.

55. Click Create method ‘createKeyPair(CreateKeyPairRequest)'.
56. Repeat for createSecurityGroup.
57. Set the return type of createSecurityGroup to CreateSecurityGroupResult.
58. Set the return type of createKeyPair to CreateKeyPairResult.

```
private CreateSecurityGroupResult createSecurityGroup(CreateSecurityGroupRequest requestData) {
    // TODO Auto-generated method stub
    return null;
}

private CreateKeyPairResult createKeyPair(CreateKeyPairRequest requestData) {
    // TODO Auto-generated method stub
    return null;
}
```

Note: Keeping the return type as a generic object would work fine as well. Since we know exactly what type will return (if an exception isn’t thrown) we’ll update the method return type.

59. Return client.createSecurityGroup(requestData) inside of createSecurityGroup.
60. Return client.createKeyPair(requestData) inside of createKeyPair.

```
private CreateSecurityGroupResult createSecurityGroup(
        CreateSecurityGroupRequest requestData) {
    // TODO Auto-generated method stub
    return client.createSecurityGroup(requestData);
}


private  CreateKeyPairResult createKeyPair(
        CreateKeyPairRequest requestData) {
    // TODO Auto-generated method stub
    return client.createKeyPair(requestData);
}
```

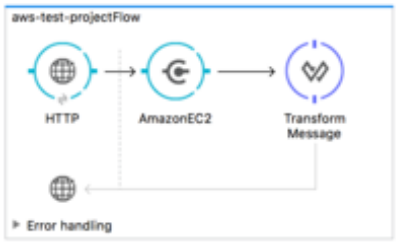
Note: Keep in mind that while these two methods are simply returning client calls, you have full control over implementing more complex logic that can aid in your developer’s use of these operations.

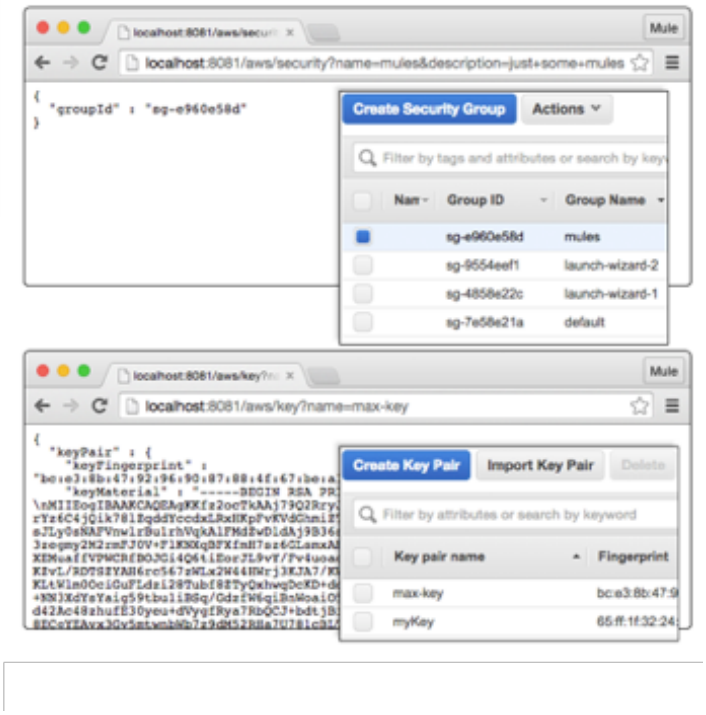
61. Save the project.

Walkthrough 3-6: Test DataSense

In this walkthrough, you test all of the functionality added to the connector so far. You will:

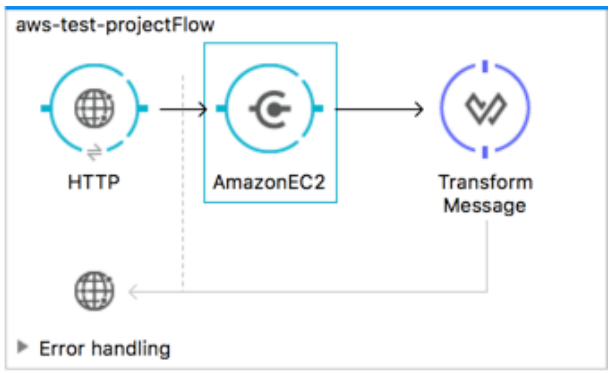
- Examine Anypoint Studio's use of DataSense with your connector endpoint.
- Test the creation of a security group.
- Test the creation of a key pair.





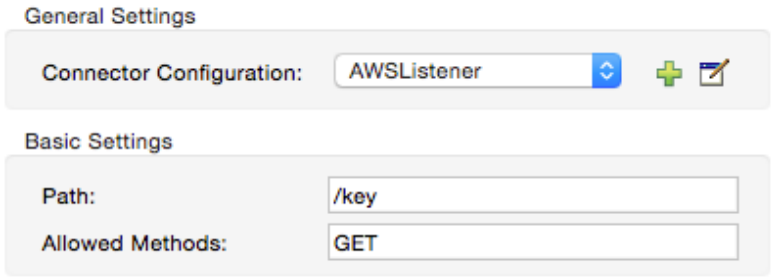
Update Anypoint Studio

1. Right-click the amazon-ec2-connector project.
2. Select Anypoint Connector > Install or Update.
3. Update the connector.
4. Open the aws-test-project.

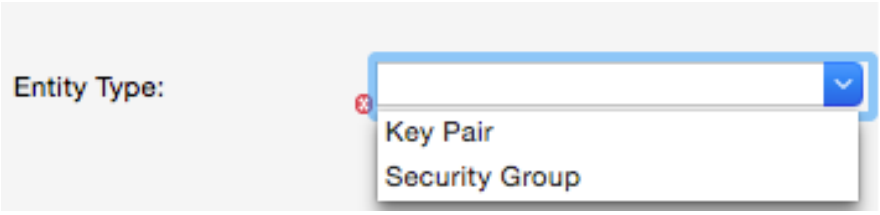


Use the Create Key Pair operation

5. Create a new flow and name it createKeyPair.
6. Add an HTTP connector endpoint to the source.
7. Set the connector configuration to the preexisting http:listener.
8. Set the path to key and Allowed Methods to GET.



9. Drag the AmazonEC2 connector into the flow.
10. Set the Connector Configuration to the pre-existing Connector.
11. Set the Operation to Create.
12. Drop-down Entity Type.



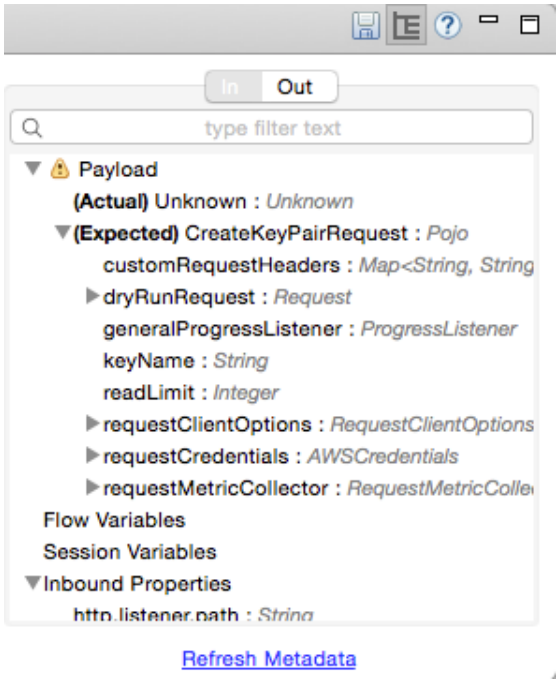
Note: Your two entities are now available from our @MetaDataKeyRetriever method.

13. Select Key Pair.

Note: Input/output structures are now being retrieved based on the configuration of our @MetaDataRetriever and our @MetaDataOutputRetriever.

14. Select the AmazonEC2 connector endpoint again.

15. In the MetaData explorer in the bottom right of studio expand the (Expected) line.

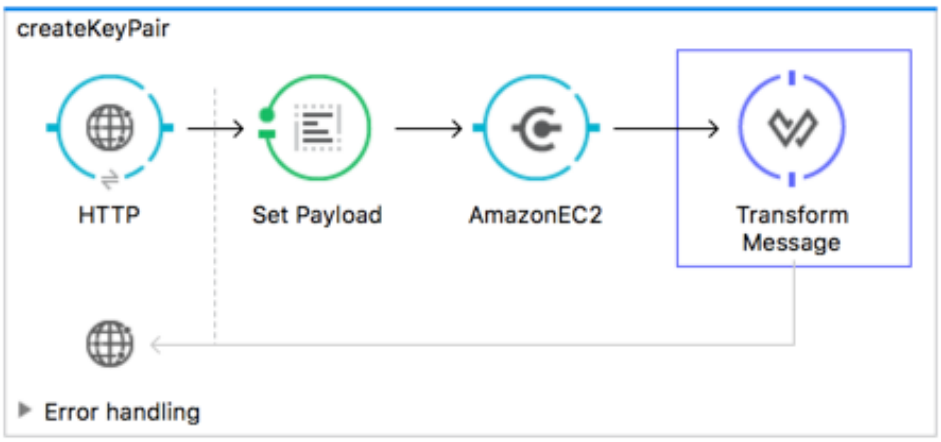


Note: Metadata explorer can be accessed by clicking the metadata icon in the bottom right of Anypoint Studio's default perspective.

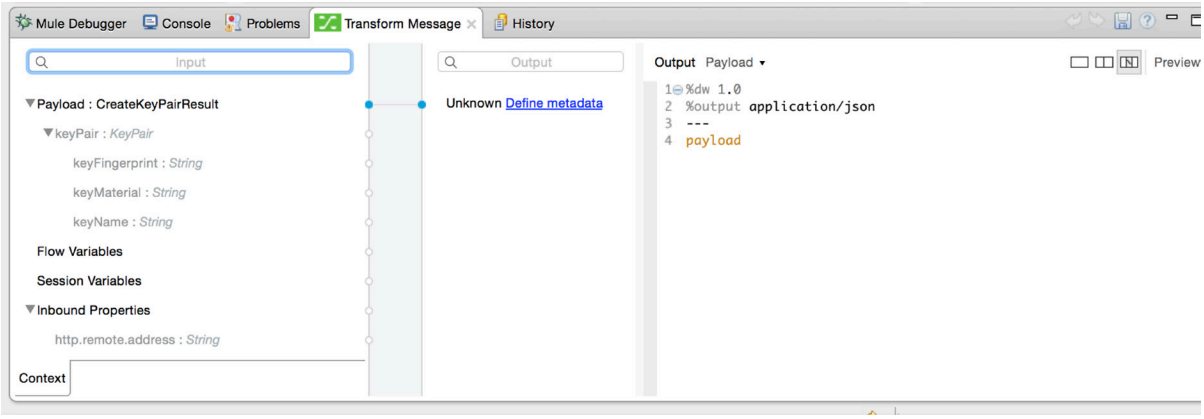
16. Click the Out tab and examine the expected output.
17. Add a Set Payload transformer before the connector endpoint.
18. Set the value to the following:

```
#[new
com.amazonaws.services.ec2.model.CreateKeyPairRequest(message.inboundP
roperties.'http.query.params'.keyName)]
```

19. Add a Transform Message component after the connector endpoint.



20. In the script area, set the output directive to application/json and the script to payload.



21. Save and run the project.
22. Make a request to the HTTP endpoint including a keyName query parameter).
23. Examine the result.

Note: You've just generated a new key within AWS!

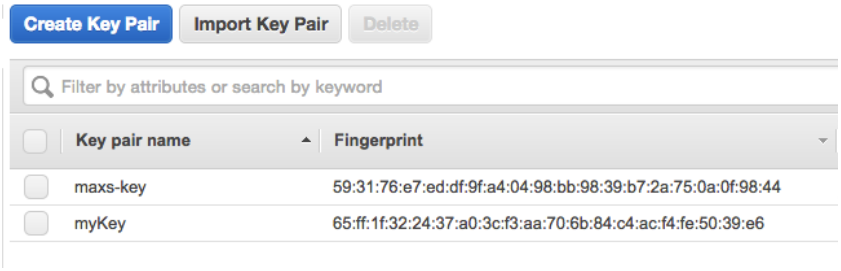
Note: If you get an error indicating the key already exists, try a different key name in the query parameter (for example use your full name).

24. Log in to the AWS console (http://aws.amazon.com/console).

Note: You'll find credentials for the AWS console in your credentials.txt file in the student files.

25. Ensure the origin is US East.
26. Navigate to Services > EC2.
27. Click Key Pairs in the left navigation bar.

28. Examine the available keys.



Note: The generated key is now part of the AWS account.

Walkthrough 3-7: Add a delete operation

In this walkthrough, you modify the metadata code from the create operation to implement a delete operation. You will:

- Reuse Anypoint Studio's use of DataSense with your connector endpoint.
- Test the deletion of a security group.
- Test the deletion of a key pair.

Add a metadata class to handle the delete operation

1. In the Package Explorer, copy and paste EC2BaseCategory.java to EC2DeleteBaseCategory.java.
2. Open EC2DeleteBaseCategory.java in the Anypoint Studio editor.
3. Modify the describeEntity() method to use DeleteKeyPairRequest.class and DeleteSecurityGroupRequest.class.

```
@MetaDataRetriever
public MetaData describeEntity(MetaDataKey entityKey)   {
    MetaDataModel model;
    switch (entityKey.getId()) {
    case "KeyPair_id":
        model = new DefaultMetaDataBuilder().
        createPojo(DeleteKeyPairRequest.class).build();
        break;
    case "SecurityGroup_id":
        model = new DefaultMetaDataBuilder().
        createPojo(DeleteSecurityGroupRequest.class).build();
        break;
    default:
        throw new RuntimeException(
                String.format("This entity %s is not supported",
                        entityKey.getId()));
    }
    return new DefaultMetaData(model);
}
```

Add a delete method

4. Locate the create() method in AmazonEC2Connector.java
5. Copy and paste the create() method, including all the annotations.
6. Change the @MetaDataScope annotation above the delete method so it uses the new EC2DeleteBaseCategory.class.

   Note: The class is using a @MetaDataScope(EC2BaseCategory.class) but this @Processor is using a different class.

```
@Processor
@MetaDataScope(EC2DeleteBaseCategory.class)
public Object delete(
        @MetaDataKeyParam(
                affects=MetaDataKeyParamAffectsType.BOTH
                ) String entityType,
        @Default("#[payload]") Object requestData){
```

7. Modify the delete() method's switch statement to call a new deleteKeyPair() with a DeleteKeyPairRequest, or to call a new deleteSecurityGroup() method with a DeleteSecurityGroupRequest.

```
@Processor
@MetaDataScope(EC2DeleteBaseCategory.class)
public Object delete(
        @MetaDataKeyParam(
                affects=MetaDataKeyParamAffectsType.BOTH
                ) String entityType,
        @Default("#[payload]") Object requestData){

    System.out.println("\n\n***************DELETE KEY called******************");
    switch (entityType) {
    case "KeyPair_id":
        return deleteKeyPair((DeleteKeyPairRequest) requestData);
    case "SecurityGroup_id":
        return deleteSecurityGroup((DeleteSecurityGroupRequest) requestData);
    default:
        break;
    }
    throw new RuntimeException("Entity not recognized");
}
```

8. Implement the deleteKeyPair() method to delete the key pair using the Amazon EC2 client connection.

```
private Object deleteKeyPair(DeleteKeyPairRequest requestData) {
    client.deleteKeyPair(requestData);
    return "Key Pair deleted: "+requestData.getKeyName();
}
```

9. Implement the deleteSecurityGroup() method to delete the security group using the Amazon EC2 client connection.

```
private Object deleteSecurityGroup(DeleteSecurityGroupRequest requestData) {
    // TODO Auto-generated method stub
    client.deleteSecurityGroup(requestData);

    return "Security Group deleted: "+requestData.getGroupId();
}
```

10. Save your changes.
11. Install or update the Amazon EC2 connector.

Use the delete method in a Mule flow

12. In the aws-test-project project, open the aws-test-project.xml configuration file.
13. Copy and paste the createKeyPair flow to a new flow named deleteKeyPair.
14. Change the HTTP path to /delete.
15. Select the Amazon_EC2 connector and change the operation from create to delete.
16. Change the set payload transformer to the following:

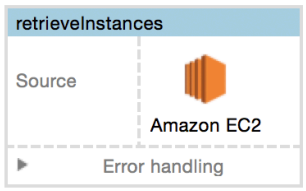
```
#[new com.amazonaws.services.ec2.model.DeleteKeyPairRequest(
message.inboundProperties.'http.query.params'.keyName)]
```

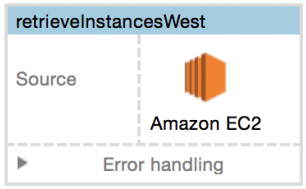
17. Delete the transform message component.
18. Save your changes.
19. Run the new flow in the Debugger.

Test the delete operation

20. Open a web browser and locate the URL you used to create the key.
21. Modify the URL from create to delete and submit the GET request; e.g. http;//localhost:8081/aws/delete?keyName=yourKeyName
22. Verify the delete completes successfully
23. Try to create the same key again and verify you do not get an exception; e.g. http;//localhost:8081/aws/create?keyName=max-test

Module 4: Finalizing a Connector





UpdateSite.zip
15.1 MB

Message Processors

<amazon-ec2:create>

Create a new aws resource based on the entityType argument.

XML Sample

```
<amazon-ec2:create config-ref="" entityType="" requestData="#[payload]"/>
```

Attributes

Name	Default Value	Description	Java Type	MIME Type	Encoding
config-ref		*Optional.* Specify which configuration to use.			
entityType		Determine what type of AWS resource will be created.	String	*/*	UTF-8
requestData	# [payload]	Request information used to instruct the creation of an AWS resource.	Object	*/*	

Returns

Return Type	Description
Object	Returns either a CreateSecurityGroupResult or CreatePairResult based on the entityType argument.

<amazon-ec2:describe-instances>

Retrieve all the Amazon EC2 instances in a given region.

XML Sample

```
<amazon-ec2:describe-instances config-ref="" />
```

Objectives:

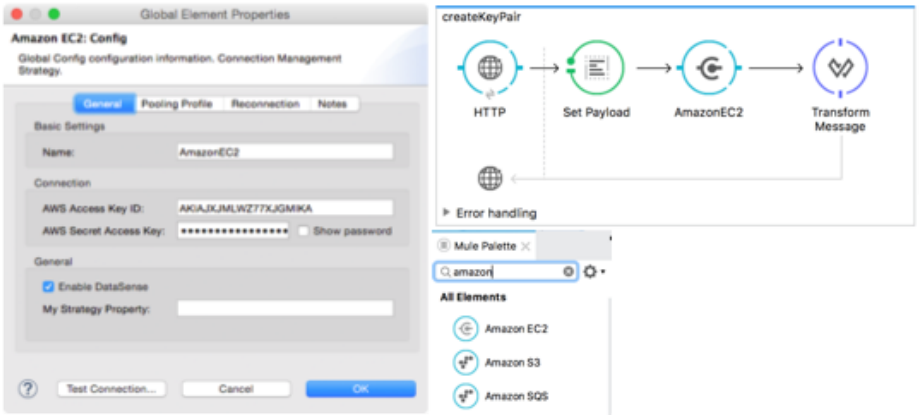
- Update a connector's user interface, ensuring it's clean and intuitive for developers.
- Set up functional tests which trigger a connector into the context of a Mule application.
- Create documentation for a connector.
- Generate an Eclipse update site so that developers can access a connector.

Walkthrough 4-1: Update the connector UI

In this walkthrough, you update the connector's user interface. You will:

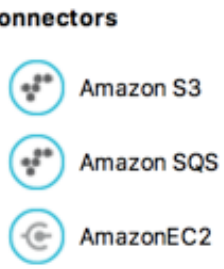
- Add a unique icon to the connector.
- Define the exact name of the connector and configuration details.
- Set configurable parameters.
- Repackage and update the connector in Anypoint Studio.

Note: There is a bug in Anypoint Studio 6.0.0 and customs icons are not being displayed.



Examine the existing connector

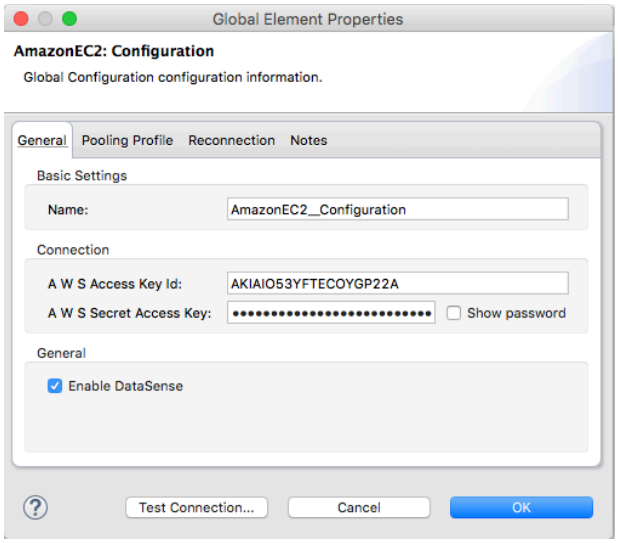
1. Return to the aws-test-project.
2. Search for Amazon in the palette and look at the names of the connectors.



Note: the name difference. For consistency, we'll look to add a space between Amazon and EC2 along with providing our connector a more appropriate icon.

3. Locate aws-test-project.xml and open the Global Elements tab.

4. Double-click the AmazonEC2: Configuration element.



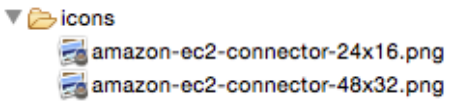
Note: Notice the Connection title of 'A W S {attribute-name}'.

Update the connector

5. Expand the amazon-ec2-connector project.
6. Open AmazonEC2Connector.java.
7. At the top of the connector change the friendlyName to Amazon EC2.

```
@MetaDataScope(EC2BaseCategory.class)
@Connector(name="amazon-ec2", friendlyName="Amazon EC2")
public class AmazonEC2Connector
{
```

8. Expand the icons directory within amazon-ec2-connector.
9. Replace the two PNGs with the PNGs inside of your student files.



Note: DevKit will use icons with the following syntax:

Small: {connectorName}-connector-24x16.png

Large: {connectorName}-connector-48x32.png

Note: There is a bug in Anypoint Studio 6.0.0 and customs icons are not being displayed.

Update the connection configuration

10. Open ConnectorConfig.java.

11. Change the @ConnectionManagement's friendlyName argument to Config.

```
@ConnectionManagement(configElementName = "config-type", friendlyName = "Config")
public class ConnectorConnectionStrategy
{
```

12. Locate the @Connect annotated method.
13. After @ConnectionKey, set a @FriendlyName annotation to AWS Access Key ID.
14. After @Password, set a @FriendlyName annotation to AWS Secret Access Key.

AmazonEC2Connector.java

```
/**
 * Connect
 *
 * @param AWSAccessKeyId a access key id
 * @param AWSSecretAccessKey a secret access key
 * @throws ConnectionException
 */
@Connect
@TestConnectivity
public void connect(
        @ConnectionKey
        @FriendlyName(value="AWS Access Key ID") String AWSAccessKeyId,
        @Password
        @FriendlyName(value="AWS Secret Access Key") String AWSSecretAccessKey)
                throws ConnectionException {
    this.ec2Client = new AmazonEC2Client(
            generateAWSCredentials(AWSAccessKeyId, AWSSecretAccessKey));
    this.ec2Client.describeAvailabilityZones();

}
```

15. Save the connector project.

Add a configurable parameter

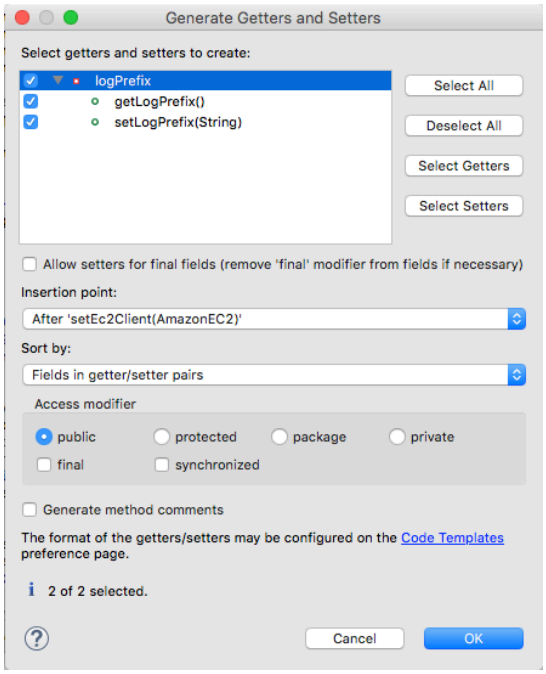
16. Create a new private member named logPrefix.
17. Add an annotation named @Configurable and a parameterized annotation named @Default("EC2-CONNECTOR").

```
@Configurable
@Default("EC2-CONNECTOR")
private String logPrefix;
```

Note: You may need to import the classes from package org.mule.api.annotations.

18. Right-click member logPrefix and select Source | Generate Getters and Setters.

19. Select After setEc2Client(AmazonEC2) for Insertion Point.



20. Save the connector project.
21. Rebuild the project to resolve any errors or problems.
22. Locate the @Connect annotated method and add the following line of code:

```
if (isConnected()) System.out.println(getLogPrefix() +" Connected!");
```

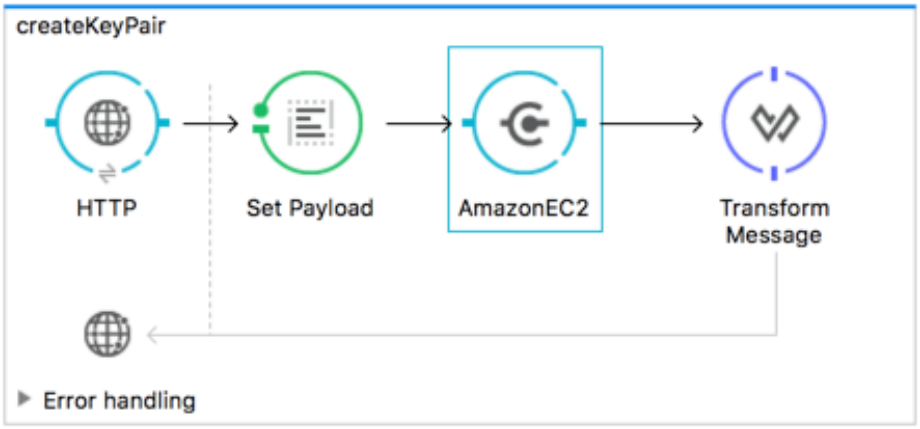
23. Open AmazonEC2Connector.java and locate the annotated create() method.
24. After each case statement, add a console output line as shown below:

```
@Processor
@MetaDataScope(EC2BaseCategory.class)
public Object create(@MetaDataKeyParam(affects=MetaDataKeyParamAffectsType.BOTH) String entityType,
                     @Default("#[payload]") Object requestData) {
    switch (entityType) {
    case "KeyPair_id":
        System.out.println(getConfig().getLogPrefix() +" Creating Key Pair.");
        return createKeyPair((CreateKeyPairRequest) requestData);
    case "SecurityGroup_id":
        System.out.println(getConfig().getLogPrefix() +" Creating Security Group.");
        return createSecurityGroup((CreateSecurityGroupRequest) requestData);
    default:
        break;
    }
    throw new RuntimeException("Entity not recognized");
}
```

25. Save the connector project and update the connector in Anypoint Studio.

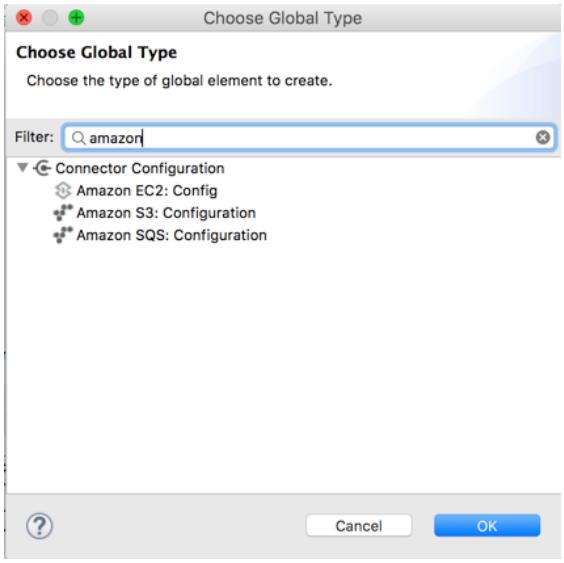
Review the updated connector in a Mule project

26. Open the aws-test-project.
27. Examine the new icon in the flows.

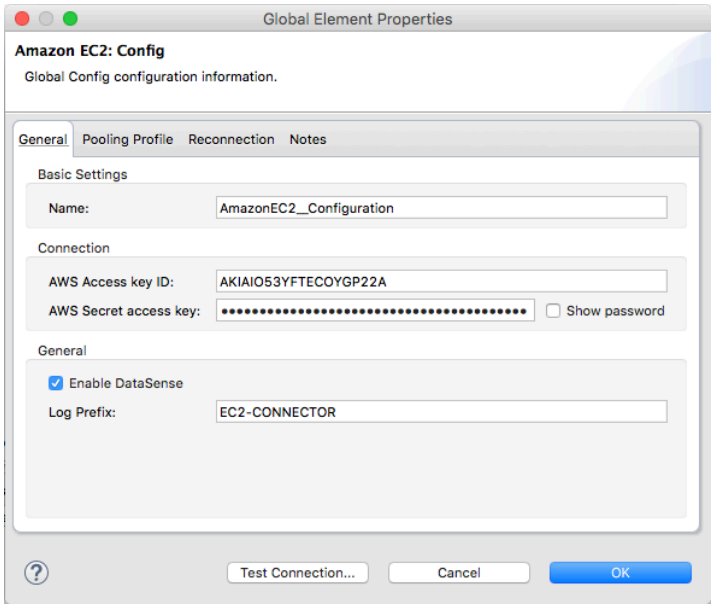


Note: New icons have been placed at each of our endpoints based on what was available to DevKit in our icons directory with the needed syntax.

28. Open the Global Elements tab.
29. Double-click the AmazonEC2: Configuration element.

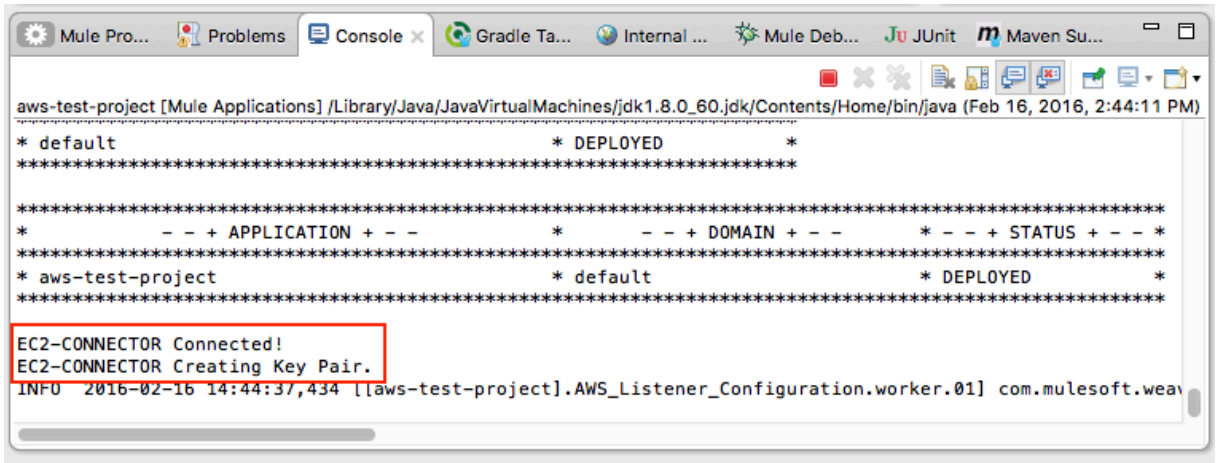


30. Double-click Amazon EC2: Config and examine the updated fields.



Note: This is an effect of the added @FriendlyName annotations in the @Connect method's parameters.

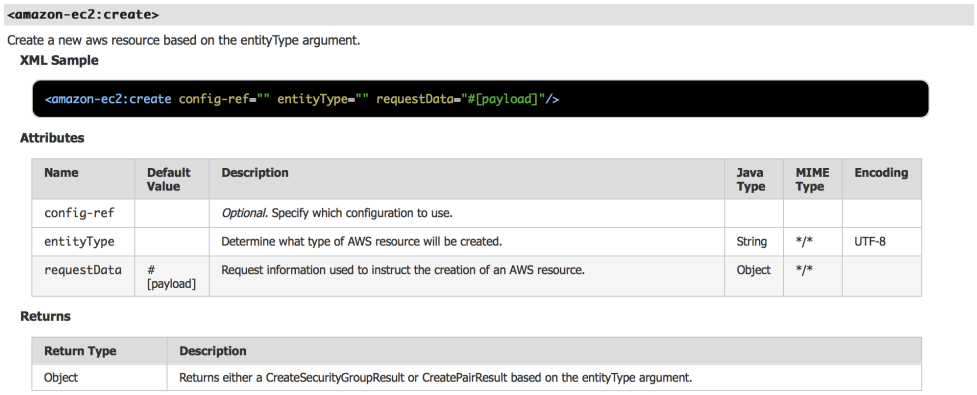
31. Click Cancel.
32. Run the project.
33. Make a request to http://localhost:8081/aws/key?keyName=WT41.
34. In Anypoint Studio, open the Console view and observe the output.



Walkthrough 4-2: Create documentation

In this walkthrough, you set up documentation to support the developers utilizing your connector. You will:

- Update documentation-based annotations.
- Implement tool-tips exposed by Anypoint Studio.
- Generate Java Docs.



<amazon-ec2:create>

Create a new aws resource based on the entityType argument.

XML Sample

```
<amazon-ec2:create config-ref="" entityType="" requestData="#[payload]"/>
```

Attributes

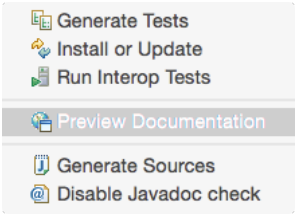
Name	Default Value	Description	Java Type	MIME Type	Encoding
config-ref		*Optional.* Specify which configuration to use.			
entityType		Determine what type of AWS resource will be created.	String	*/*	UTF-8
requestData	# [payload]	Request information used to instruct the creation of an AWS resource.	Object	*/*	

Returns

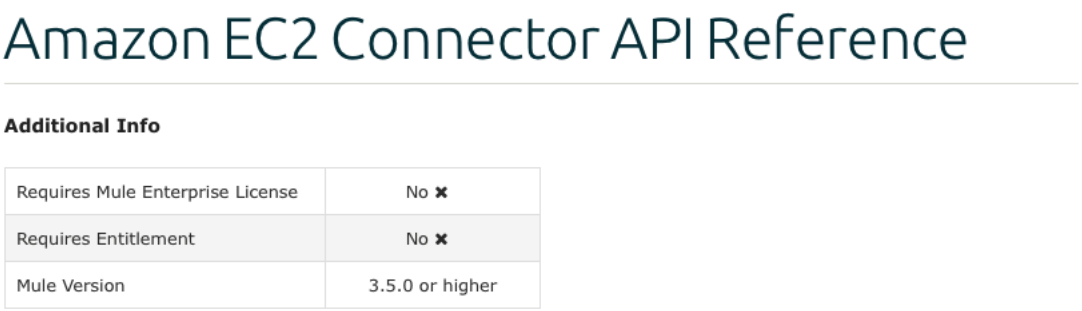
Return Type	Description
Object	Returns either a CreateSecurityGroupResult or CreatePairResult based on the entityType argument.

Generate default Java Docs

1. Right-click the amazon-ec2-connector project.
2. Select Anypoint Connector > Preview Documentation.



3. Explore the existing docs.



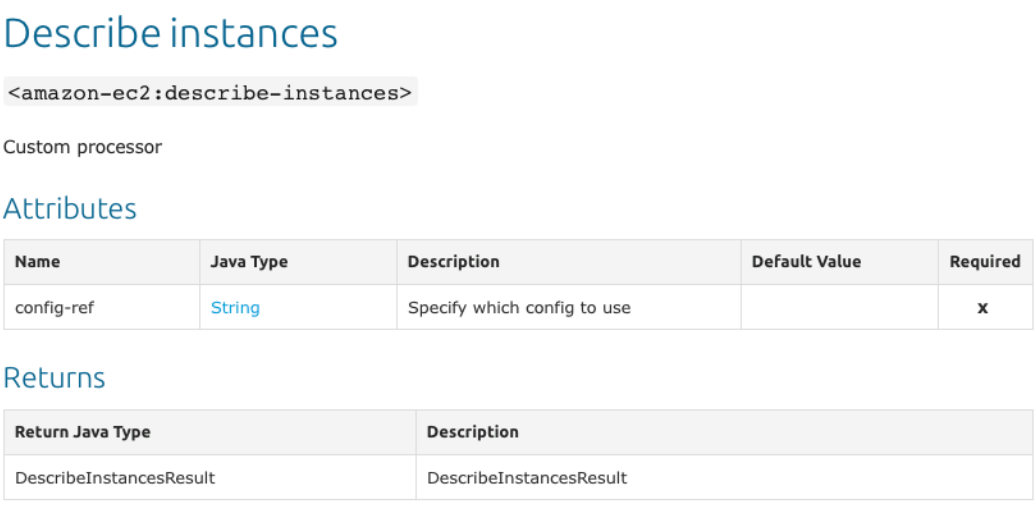
Amazon EC2 Connector API Reference

Additional Info

Requires Mule Enterprise License	No ✖
Requires Entitlement	No ✖
Mule Version	3.5.0 or higher

Note: After looking through the default documentation, it should be apparent that some things are missing and others are simply out of date.

4. Locate the amazon-ec2:describe-instances documentation.



Describe instances

`<amazon-ec2:describe-instances>`

Custom processor

Attributes

Name	Java Type	Description	Default Value	Required
config-ref	String	Specify which config to use		x

Returns

Return Java Type	Description
DescribeInstancesResult	DescribeInstancesResult

Note: You'll first look to update this processor's documentation.

Document a processor

5. Open AmazonEC2Connector.java.
6. Locate describeInstances().
7. Replace Custom Processor with a better descriptor.
8. Add a more descriptive @return value.
9. Update the sample.xml link to point to amazon-ec2:my-processor.

```
/**
 * Retrieve all the Amazon EC2 instances in a given region.
 *
 * {@sample.xml ../../../doc/amazon-ec2-connector.xml.sample amazon-ec2:describe-instances}
 *
 * @return Metadata for every instance in aws (started or stopped).
 */
@Processor
public DescribeInstancesResult describeInstances() {
    return client.describeInstances();
}
```

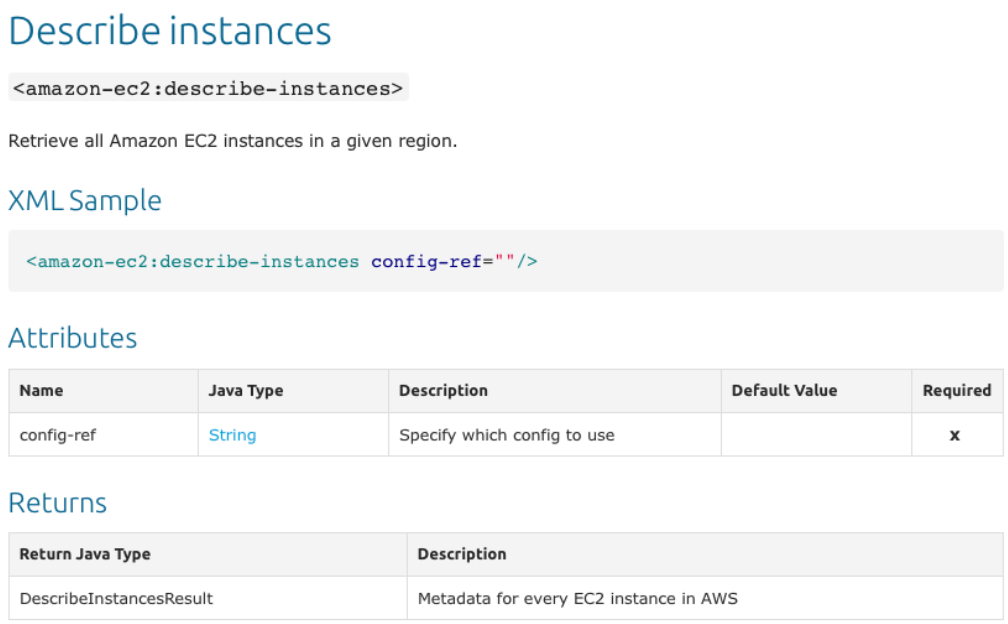
Note: With the @sample.xml link updated, you will need to update your file to correspond to describe-instances.

10. Expand the doc directory within the amazon-ec2-connector project.
11. Open amazon-ec2-connector.xml.sample.
12. Update amazon-ec2:greet to be amazon-ec2:describe-instances.

13. Delete the content attribute.

```
<!-- BEGIN_INCLUDE(amazon-ec2:describe-instances) -->
    <amazon-ec2:describe-instances config-ref="" />
<!-- END_INCLUDE(amazon-ec2:describe-instances) -->
```

14. Save the project.
15. Preview the documentation for the amazon-ec2-connector again.



Describe instances

`<amazon-ec2:describe-instances>`

Retrieve all Amazon EC2 instances in a given region.

XML Sample

```
<amazon-ec2:describe-instances config-ref=""/>
```

Attributes

Name	Java Type	Description	Default Value	Required
config-ref	String	Specify which config to use		x

Returns

Return Java Type	Description
DescribeInstancesResult	Metadata for every EC2 instance in AWS

Document remaining processors and the connector

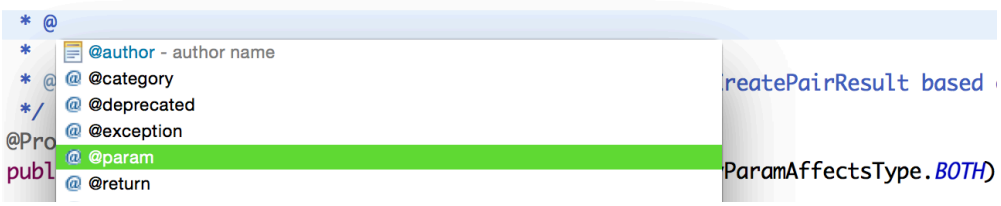
16. Open AmazonEC2Connector.java.
17. Locate the create() method.

18. Copy the documentation from above describeInstances to above create.

```
/**
 * Retrieve all the Amazon EC2 instances in a given region.
 *
 * {@sample.xml ../../../doc/amazon-ec2-connector.xml.sample amazon-ec2:describe-instances}
 *
 * @return Metadata for every instance in aws (started or stopped).
 */
@Processor
public DescribeInstancesResult describeInstances() {
    return client.describeInstances();
}

/**
 * Retrieve all the Amazon EC2 instances in a given region.
 *
 * {@sample.xml ../../../doc/amazon-ec2-connector.xml.sample amazon-ec2:describe-instances}
 *
 * @return Metadata for every instance in aws (started or stopped).
 */
@Processor
public Object create(@MetaDataKeyParam(affects=MetaDataKeyParamAffectsType.BOTH) String entityType,
```

19. Update the description Retrieve all the Amazon EC2 to Create a new aws resource based on the entityType argument.
20. Update amazon-ec2:describes instances within the @sample.xml to be amazon-ec2:create.
21. On a new comment line type @.
22. Use auto-complete (Ctrl+Space) to see which java docs annotations are available.

```
/**
 * Create a new aws resource based on the entityType argument.
 *
 * {@sample.xml ../../../doc/amazon-ec2-connector.xml.sample amazon-ec2:create}
 *
 * @
 *
 * @
 */
@Pro
publ
```



23. Chose @param.
24. After param, use autocomplete to select the entityType parameter.
25. Provide entityType a description such as, Determines what type of AWS resource will be created.
26. Repeat these steps for requestData.
27. Enter a description such as Request information used to instruct the creation of an AWS resource.

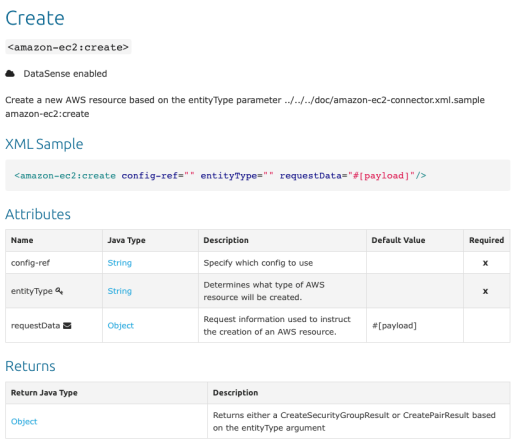
28. Update the return parameter to say Returns either a CreateSecurityGroupResult or CreatePairResult based on the entityType argument.

```
/**
 * Create a new AWS resource based on the entityType parameter
 *
 * {@sample.xml ../../../doc/amazon-ec2-connector.xml.sample amazon-ec2:create}
 *
 * @param entityType    Determines what type of AWS resource will be created.
 * @param requestData   Request information used to instruct the creation of an AWS resource.
 *
 * @return Returns either a CreateSecurityGroupResult or CreatePairResult based on the entityType argument
 */
@Processor
@MetaDataScope(EC2BaseCategory.class)
public Object create(
      @MetaDataKeyParam(affects = MetaDataKeyParamAffectsType.BOTH) String entityType,
      @Default("#[payload]") Object requestData) {
   switch (entityType) {
```

29. Save AmazonEC2Connector.java.
30. Open amazon-ec2-connector.xml.sample from the doc directory.
31. Duplicate all of the existing XML.
32. Update the second BEGIN_INCLUDE with an argument of amazon-ec2:create.
33. Do the same for END_INCLUDE.
34. Update the second XML snippet to be amazon-ec2:create.
35. Add the attribute entityType=”” to the xml element.
36. Add the attribute requestData=”#[payload]” to the XML element.

```
<!-- BEGIN_INCLUDE(amazon-ec2:describe-instances) -->
    <amazon-ec2:describe-instances config-ref="" />
<!-- END_INCLUDE(amazon-ec2:describe-instances) -->

<!-- BEGIN_INCLUDE(amazon-ec2:create) -->
    <amazon-ec2:create config-ref="" entityType="" requestData="#[payload]"/>
<!-- END_INCLUDE(amazon-ec2:create) -->
```

37. Save all files in the project.
38. Regenerate documentation and examine the results.



Create

`<amazon-ec2:create>`

DataSense enabled

Create a new AWS resource based on the entityType parameter ../../../doc/amazon-ec2-connector.xml.sample amazon-ec2:create

XML Sample

```
<amazon-ec2:create config-ref="" entityType="" requestData="#[payload]"/>
```

Attributes

Name	Java Type	Description	Default Value	Required
config-ref	String	Specify which config to use		x
entityType	String	Determines what type of AWS resource will be created.		x
requestData	Object	Request information used to instruct the creation of an AWS resource.	#[payload]	

Returns

Return Java Type	Description
Object	Returns either a CreateSecurityGroupResult or CreatePairResult based on the entityType argument

Walkthrough 4-3: Publish the Anypoint Connector

In this walkthrough, you take a look at what is created with building a connector project. You will:

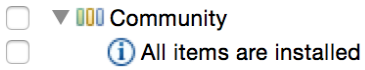
- Build the project from outside of Anypoint Studio, using Maven.
- Examine the various components that are created.
- View the generated update site.
- Discover how your connector can be added publicly or privately to the Anypoint Exchange.

Locate the existing update site in Anypoint Studio

1. Open Help.
2. Click Install New Software.
3. Open the Work with drop-down menu.
4. Locate the update site related to your amazon-ec2 connector.

   Note: Looking towards the end of the update sites' name should reveal which connector

5. Expand Community

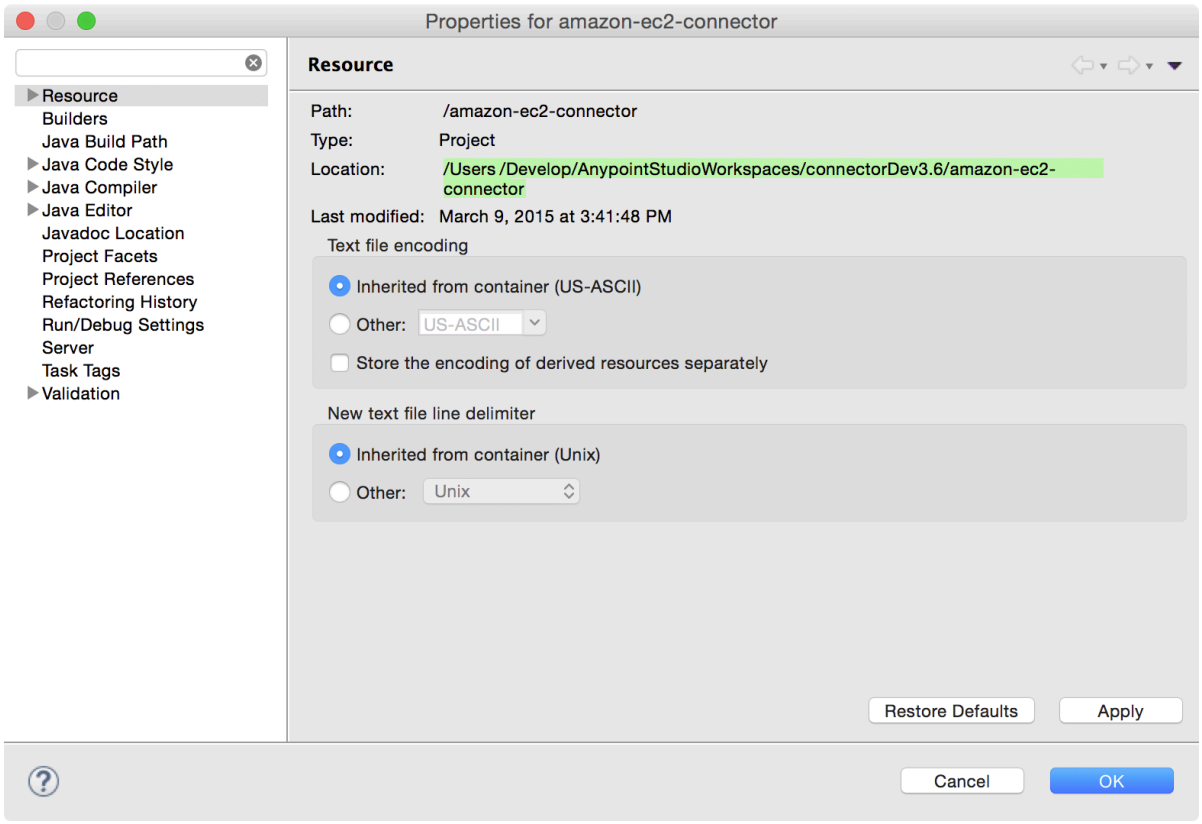


   Note: Currently Anypoint Studio is pointed at an update site on your local computer to determine whether new versions of the Amazon EC2 connector are available. While this works fine for your development, what if you wish to share the connector with a team in your company? Or, even more importantly, what if the project doesn't live in this specific directory forever? You will soon see how this can be remedied with the update site generated by Maven.

Navigate to the project within a command prompt

6. Right-click the amazon-ec2-connector within Anypoint Studio.
7. Click Properties.

8. Copy the location of the project to your clipboard.



9. Open terminal (Mac) or command prompt (Windows).
10. Change directories into the previously copied location.

```
➜  amazon-ec2-connector git:(master) ✗ cd /Users/Develop/AnypointStudioWorkspaces
/connectorDev3.6/amazon-ec2-connector
```

11. List the available directories by calling a ls (Mac) or dir (windows) and examine the results.

```
➜  amazon-ec2-connector git:(master) ✗ ls -l
total 128
-rw-r--r--  1 joshrosso  staff    745 Feb 18 14:17 CHANGELOG.md
-rw-r--r--  1 joshrosso  staff  27676 Feb 18 14:17 LICENSE.md
-rw-r--r--  1 joshrosso  staff    194 Feb 18 14:17 LICENSE_HEADER.txt
-rw-r--r--  1 joshrosso  staff   1088 Feb 18 14:17 README.md
drwxr-xr-x  2 joshrosso  staff     68 Feb 18 14:17 demo
-rw-r--r--  1 joshrosso  staff  18435 Mar  9 15:11 dependency-reduced-pom.xml
drwxr-xr-x  3 joshrosso  staff    102 Feb 18 14:17 doc
drwxr-xr-x  4 joshrosso  staff    136 Feb 25 18:09 icons
-rw-r--r--  1 joshrosso  staff   1599 Feb 23 11:01 pom.xml
drwxr-xr-x  4 joshrosso  staff    136 Feb 18 14:17 src
drwxr-xr-x  7 joshrosso  staff    238 Mar  9 15:42 target
➜  amazon-ec2-connector git:(master) ✗
```

Note: The target directory seen above contains all the artifacts that are created after the build process.

12. Run mvn clean.

    Note: This command removes the target directory from our project. This way we can build from scratch.

13. List directories to ensure the target directory is now absent.
14. Run mvn install.

    Note: Watch closely as this command runs. It will not only build the connector project and generate artifacts, it will also run all of your unit (using unit loosely) and functional Mule tests.

15. After completion of mvn install, list the directories again from your current command prompt location.