Template Application Tutorial

Introduction

In this tutorial we will show you how to generate an ONOS template application using the `onos-create-app` tool. This tool relies on the Maven archetypes to generate a base ONOS application as well as several different variants that add CLI command, REST API, and a few means of extending the GUI.

Select ONOS version

Because the tool uses Maven archetypes, we need to first indicate what version of ONOS API we would like to build our application against. To do this, simply export the `ONOS_POM_VERSION` environment variable to the desired ONOS version as follows:

```
$ export ONOS_POM_VERSION=2.0.0
```

If the above environment variable is not explicitly set, Maven will look for the most recent version of the archetypes available in your `~/.m2/repository` (and then on Maven central) and this may not be what you intended so it's always best to set this explicitly.

Generate a new base ONOS application project

Let's now generate a skeletal ONOS application project which will be fully compilable and ready to be deployed. When creating the base project, we have to specify the Maven groupId, artifactId and version. These are necessary for locating the application in the Maven coordinate space. Additionally, we will also specify Java package name where the generated code will be located. Let's run the following command:

```
$ onos-create-app app org.foo foo-app 1.0-SNAPSHOT org.foo.app
```

Alternatively, you could invoke `mvn archetype:generate` command directly, but we recommend that you use the `onos-create-app` instead. Also, for a real application, you would want to use your own groupId, artifactId, etc., but for the purpose of this tutorial, it is recommended that you use the suggested values.

After this you should see the following output:

```
[INFO] Scanning for projects...
[INFO]
[INFO]
------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO]
------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-
sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-
sources @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @
standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] Archetype [org.onosproject:onos-bundle-archetype:2.1.0-SNAPSHOT]
found in catalog local
[INFO] Using property: groupId = org.foo
[INFO] Using property: artifactId = foo-app
[INFO] Using property: version = 1.0-SNAPSHOT
[INFO] Using property: package = org.foo.app
Confirm properties configuration:
groupId: org.foo
artifactId: foo-app
version: 1.0-SNAPSHOT
package: org.foo.app
 Y: :
```

Confirm the properties by typing "y" and you will be presented with the following output:

```
 Y: : y
[INFO]
----------------------------------------------------------------------------
-
[INFO] Using following parameters for creating project from Archetype:
onos-bundle-archetype:2.0.0
[INFO]
----------------------------------------------------------------------------
-
[INFO] Parameter: groupId, Value: org.foo
[INFO] Parameter: artifactId, Value: foo-app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: org.foo.app
[INFO] Parameter: packageInPathFormat, Value: org/foo/app
[INFO] Parameter: package, Value: org.foo.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: groupId, Value: org.foo
[INFO] Parameter: artifactId, Value: foo-app
[INFO] Project created from Archetype in dir: /Users/tom/foobar/foo-app
[INFO]
------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
------------------------------------------------------------------------
[INFO] Total time: 24.439 s
[INFO] Finished at: 2019-03-11T17:35:21-07:00
[INFO] Final Memory: 14M/219M
[INFO]
------------------------------------------------------------------------
```

This has now generated a new project for you. Note that the `onos-bundle-archetype` version `2.0.0` was used for this. This is because we explicitly set the `ONOS_POM_VERSION` to `2.0.0`.

Let's move on to building it and loading it into ONOS.

Compile, install and activate your new ONOS application

Now we are ready to build the ONOS app. We can do that by changing into the root directory of the generated project and invoking Maven as follows:

```
$ cd foo-app
$ mvn clean install
```

When the build is complete, both the OSGi bundle and the application archive have been installed in your local maven repository. To install the application into running ONOS instance (or cluster), you can use the `onos-app` tool, which uses ONOS REST API within, to upload the `.oar` file as shown in the following example. If you need help running ONOS please refer to this page. Note that by using the exclamation mark with the `install` parameter, the application will be activated immediately after being installed.

```
$ onos-app localhost install! target/foo-app-1.0-SNAPSHOT.oar
```

Now, from the ONOS console, you should be able to see the application has been installed and activated,

```
onos> apps -s -a
...
   29 org.foo.app                       1.0.SNAPSHOT ONOS OSGi bundle
archetype
```

You can now use the generated code for this test app as a framework for adding your own custom code. If you edit the file `src/main/java/org/foo/app/AppComponent.java`, you can see how the application is created, and add your own code to the application.

Generate various project overlays

ONOS applications can hook into the ONOS CLI, REST API and GUI. When generating your application, you can use overlays to generate the classes needed to give your application access to these services.

Command Line Overlay

To allow your application to add commands to the ONOS CLI, overlay the CLI interface like this:

Creating a CLI Overlay

```
$ onos-create-app cli org.foo foo-app 1.0-SNAPSHOT org.foo.app
```

Now, as before, we need to build and install our application. Since we installed it once already, we will use the `reinstall!` command to deploy it:

Rebuild and reinstall the app

```
$ mvn clean install
$ onos-app localhost reinstall! target/foo-app-1.0-SNAPSHOT.oar
```

Using the ONOS command line, we now have access to the 'sample' command, which was defined by our overlay:

```
onos> sample
Hello World
```

You can use the CLI overlay to add your own commands to the CLI for your app. Edit the file `src/main/java/org/foo/app/AppCommand.java` to see how the sample command is implemented.

REST API Overlay

You can even create a REST interface for your application by overlaying a REST API project facet. Run the following to generate the project overlay and to rebuild and reinstall the application:

Creating a REST interface

```
$ onos-create-app rest org.foo foo-app 1.0-SNAPSHOT org.foo.app
$ mvn clean install
$ onos-app localhost reinstall! target/foo-app-1.0-SNAPSHOT.oar
```

Note that ONOS automatically generates SwaggerUI documentation for the URIs exposed in your RESTful application. After activating the application, you can visit the SwaggerUI at http://localhost:8181/v1/docs/ . To view the documentation for the newly generated app REST API, you will need to select **Sample app REST API** from the SwaggerUI menu bar.

User Interface Overlays

To allow your application to add to the ONOS web UI, you can use one of several different project overlays. Navigate in the shell to the directory in which you have already created your app and run one of the following commands:

... generate a "*Custom View*" overlay, like this:

Creating a UI Overlay – Custom View

```
$ onos-create-app ui org.foo foo-app 1.0-SNAPSHOT org.foo.app
```

... or the "*Tabular View*" overlay, like this:

Creating a UI Overlay – Tabular View

```
$ onos-create-app uitab org.foo foo-app 1.0-SNAPSHOT org.foo.app
```

... or the "*Topology Overlay*" overlay, like this:

Creating a UI Overlay – Tabular View

```
$ onos-create-app uitopo org.foo foo-app 1.0-SNAPSHOT org.foo.app
```

Now, as before, we need to build and install our application. Since we installed it once already, we will use the reinstall command to deploy it:

Rebuild and reinstall the app

```
$ mvn clean install
$ onos-app localhost reinstall! org.foo.app target/foo-app-1.0-SNAPSHOT.oar
```

Your new forms of application that extends the GUI will now be active. To view the new application page that was just created, point your browser to the ONOS GUI at http://localhost:8181/onos/ui/. In the upper left corner of the home page, click the navigation button to activate the drop down navigation menu. At the bottom of the list, you will see an entry for "*Sample Custom*" or "*Sample Table*", depending on which overlay you used. If you select it, you will navigate to the page that was installed by your test application.

You can now use the generated web UI to add your own UI elements.

See the Web UI tutorials for more detailed information.

Publish the artifacts to local repository

Note that, if you wish to build an application against the current ONOS master, i.e. an unreleased version, you will need to first build ONOS yourself and then publish the resulting artifacts in the Maven repository under `~/.m2/repository` by running the following command:

```
$ onos-publish -l                # publish ONOS libraries only
```

Also, you will have to build the current version of the ONOS application archetypes as follows:

```
$ cd $ONOS_ROOT/tools/package/archetypes
$ mvn clean install                # yes, the archetypes are still built
via Maven
```

Happy coding

Learn more about the Application Subsystem.

Here is a post on Sample PortStatistics Application tutorial based on template application tutorial .

Return To : Tutorials and Walkthroughs