DevOps for AEM as a Cloud Service

Configure AEM Cloud Service

Adobe

Student Workbook

Find your journey at learning.adobe.com >

Contents

10-14-2020

Introduction

Adobe Experience Manager (AEM) developers and administrators use OSGi(Open Service Gateway Initiative) configurations to manage AEM Service settings in conjunction with supported runmodes. They can create and customize OSGi configurations based on business requirements.

Objectives

After completing this module, you will be able to:

- Explain OSGi configurations
- Describe the most used OSGi configurations
- Create an OSGi configuration
- Explain the role of System Users
- Describe the Repository Initializer Service and its language

OSGi Configuration: An Overview

OSGi is a fundamental element in the technology stack of AEM. OSGi provides the standardized primitives that enable applications to be constructed from small, reusable and collaborative modules. These modules, commonly known as ' bundles', can be composed into an application and deployed. An OSGi application is typically made up of a set of bundles, configuration information and artifacts.

OSGi supports the modular deployment of the bundles that make up the application. You can stop, install, and start these bundles individually. Each bundle contains one or more OSGi components. The OSGi container handles the interdependencies between the bundles and their components automatically .

OSGi configurations are the definitions of the system parameters and settings contained in bundles. You can manage the configuration settings for OSGi components through the configuration files that are part of an AEM code project.

JSON Configuration Files [1]

The recommended best practice is to define OSGi configurations by using JSON configuration files. You should define the configurations in the AEM Project's code packages (ui.apps) as configuration files (.cfg,.json).

OSGi configurations target OSGi components through their Persistent Identity (PID), which by default takes the OSGi component's Java class name. The OSGi configurations are deployed as JSON files with the following file naming convention:

```
<PID><~unique qualifier for factory services>.cfg.json
```

For example, to provide the configuration for the Day CQ Root Mapping Servlet, as implemented by *com.day.cq.commons.servlets.RootMappingServlet* (a singleton service), you can define an OSGi configuration file at: **/apps/<my-app>/config/com.day.cq.commons.servlets.RootMappingServlet.cfg.json**. The contents of the configuration file will follow the **cfg.json** OSGi configuration format:

```
{

  "rootmapping.target" : "/sites.html"

}
```

OSGi configuration changes are applied immediately to the relevant OSGi component.

Note: The prior versions of AEM supported OSGi configuration files using different file formats such as .cfg., .config and as XML sling:OsgiConfig resource definitions. These formats are superseded by the cfg.json OSGi configuration format.

Common OSGi Configurations for AEM

Following are the common OSGi configurations:

- Day CQ WCM Undo
- Day CQ Root Mapping
- Maintenance Tasks
- Workflow Purge
- Ad-hoc Task Purge
- Project Purge

Day CQ WCM Undo

Persistent Identity: com.day.cq.wcm.undo.UndoConfig

AEM stores a history of actions that you perform and the sequence in which you performed them, so that you can undo multiple actions in the order you performed them and redo one or more actions if necessary. You can use these commands to restore the recent state of your web page as you make decisions about the content.

Day CQ Root Mapping

Persistent Identity: com.day.cq.commons.servlets.RootMappingServlet

The Day CQ Root Mapping defines where to direct a request to '/'.

Maintenance Tasks [2]

Maintenance Tasks are the processes that run on a schedule to optimize the repository. In AEM as a Cloud Service, customers need to do only minimal configuration of the Maintenance Task operational properties. Customers can focus their resources on application-level concerns, leaving the infrastructure operations to Adobe.

Caution: Adobe reserves the right to override a customer's maintenance task configuration settings to mitigate issues such as performance degradation.

Customers can schedule the Workflow Purge, Ad-hoc Task Purge and Project Purge Maintenance tasks to be executed during the daily, weekly, or monthly maintenance windows. These task scheduling configurations should be defined in the AEM Project's code packages (ui.apps) as configuration files (.cfg.json).

Workflow Purge

Persistent Identity: com.adobe.granite.workflow.purge.Scheduler

Minimizing the number of workflow instances increases the performance of the workflow engine. Adobe recommends that you regularly purge completed or running workflow instances from the repository. You can purge workflow instances according to their age, status, and model.

Because the service is a factory service, the name of the sling:OsgiConfig node requires an identifier suffix, for example: **com.adobe.granite.workflow.purge.Scheduler~myidentifier**

Ad-hoc Task Purge

Persistent Identity: com.adobe.granite.taskmanagement.impl.purge.TaskPurgeMaintenanceTask

Tasks represent the items of work to be done on content. You can use Tasks as a part of a workflow or assign them independent of the workflow, as a stand-alone tasks. Just like you need to purge workflow instances to free up repository space, you also need to purge unnecessary ad-hoc tasks on a regular basis.

Project Purge

Persistent Identity: com.adobe.cq.projects.purge.Scheduler

Projects help you group resources into one entity. Resources may include project and team information, assets, workflows, and other types of information. Unused project instances should be purged when no longer needed.

Because the service is a factory service, the name of the sling:OsgiConfig node requires an identifier suffix, for example: **com.adobe.cq.projects.purge.Scheduler~myidentifier**

Exercise 1: Use the OSGi Installer Configuration Printer

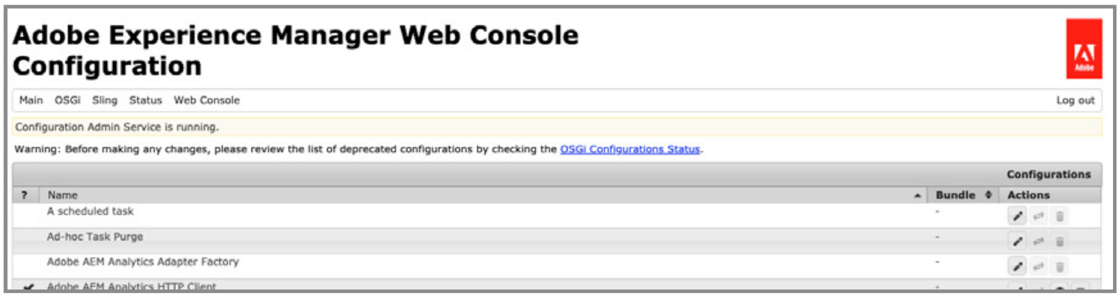
Scenario: Determine the JSON format for an existing node-based OSGi configuration.

Prerequisites:

- Local AEM author service
- AEM project created from AEM Maven Archetype

In this exercise, you will determine the correct JSON file format for an existing node-based OSGi configuration.

1. From the **Navigation**, navigate to **Tools** > **Operations** > **Web Console**, as shown:



Adobe Experience Manager Web Console Configuration

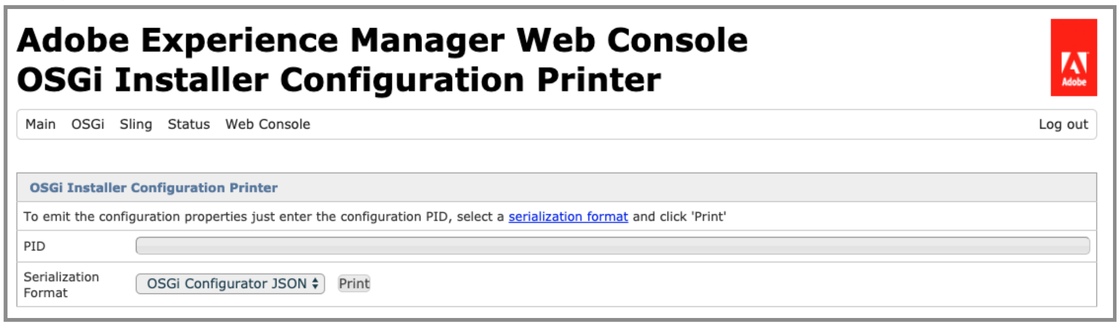
Main OSGi Sling Status Web Console — Log out

Configuration Admin Service is running.

Warning: Before making any changes, please review the list of deprecated configurations by checking the OSGi Configurations Status.

			Configurations
?	Name	Bundle	Actions
	A scheduled task	-	
	Ad-hoc Task Purge	-	
	Adobe AEM Analytics Adapter Factory	-	
✓	Adobe AEM Analytics HTTP Client	-	

2. Using the Web Console menus, select **OSGi** > **OSGi Installer Configuration Printer**, as shown:



Adobe Experience Manager Web Console OSGi Installer Configuration Printer

Main OSGi Sling Status Web Console — Log out

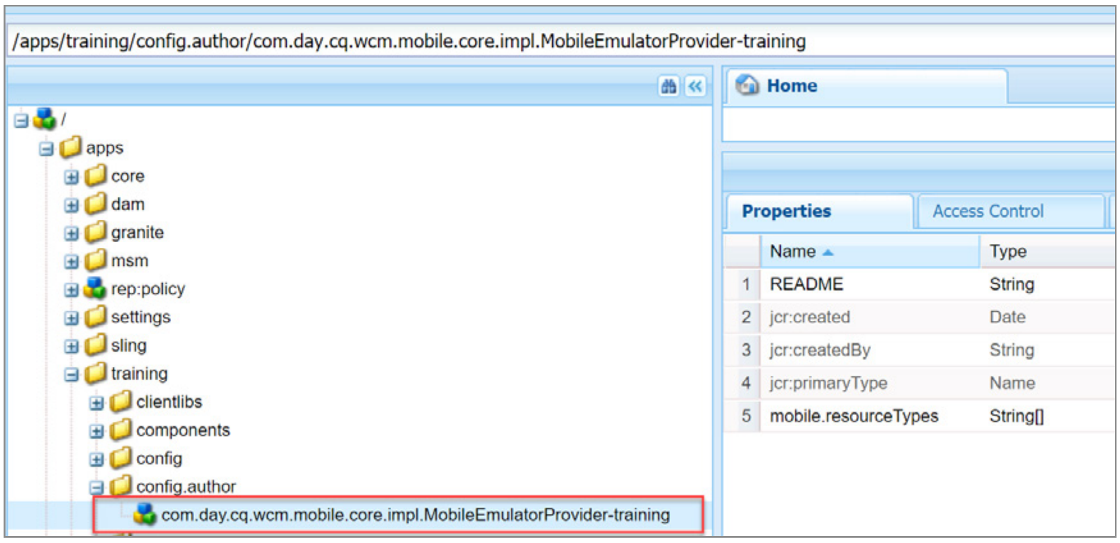
OSGi Installer Configuration Printer

To emit the configuration properties just enter the configuration PID, select a serialization format and click 'Print'

PID	
Serialization Format	OSGi Configurator JSON — Print

3. On another tab, open **CRXDE Lite** and navigate to **/apps/training/config.author**.

4. Select the **com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training** node, as shown:



/apps/training/config.author/com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training

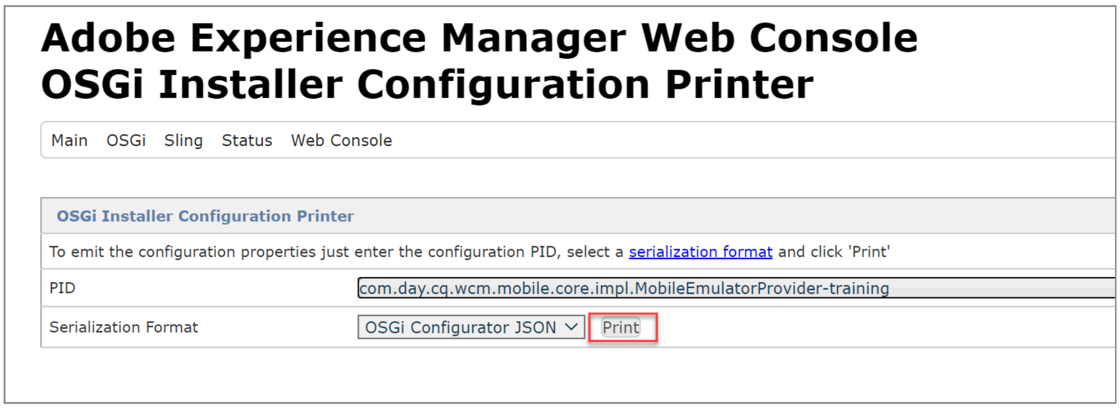
```
/
  apps
    core
    dam
    granite
    msm
    rep:policy
    settings
    sling
    training
      clientlibs
      components
      config
      config.author
        com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training
```

	Name	Type
1	README	String
2	jcr:created	Date
3	jcr:createdBy	String
4	jcr:primaryType	Name
5	mobile.resourceTypes	String[]

The property values are displayed.

5. Copy the **com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training** node name and paste it on the **PID field** on OSGi Configuration Printer tab.

6. Click **Print**, as shown:



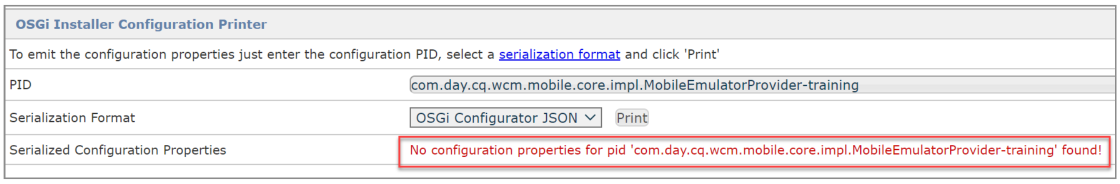
Adobe Experience Manager Web Console
OSGi Installer Configuration Printer

OSGi Installer Configuration Printer

To emit the configuration properties just enter the configuration PID, select a serialization format and click 'Print'

PID	com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training
Serialization Format	OSGi Configurator JSON Print

An error is displayed, as shown:



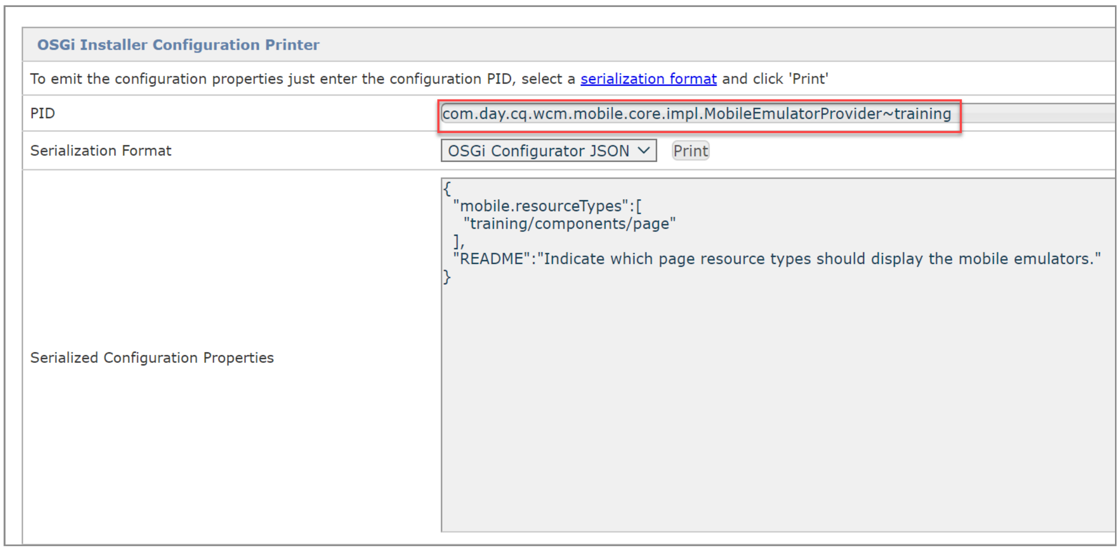
OSGi Installer Configuration Printer

To emit the configuration properties just enter the configuration PID, select a serialization format and click 'Print'

PID	com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training
Serialization Format	OSGi Configurator JSON Print
Serialized Configuration Properties	No configuration properties for pid 'com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-training' found!

The error is due to a mismatch between the naming convention of node-based OSGi factory configurations and the new naming convention of JSON-based OSGi factory configuration files. You need to change the PID to the unique configuration identifier that will be used for the JSON file.

7. Change the '-' (just before the training unique modifier) to '~' and click **Print**. The **Serialized Configuration Properties** is displayed, as shown:



OSGi Installer Configuration Printer

To emit the configuration properties just enter the configuration PID, select a serialization format and click 'Print'

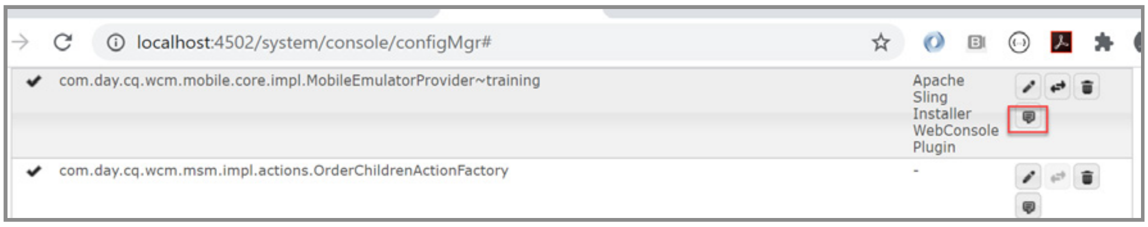
PID	com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider~training
Serialization Format	OSGi Configurator JSON Print

Serialized Configuration Properties

```
{
  "mobile.resourceTypes":[
    "training/components/page"
  ],
  "README":"Indicate which page resource types should display the mobile emulators."
}
```

8. Compare the OSGi configuration JSON format and the property values to those of the node-based configuration.

Tip: To validate the format of JSON configuration files, you can also use the Configuration Printer by navigating to **OSGi** > **Configuration** and selecting the **Show configuration as JSON** button, as shown:


localhost:4502/system/console/configMgr#

com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider~training Apache Sling Installer WebConsole Plugin

com.day.cq.wcm.msm.impl.actions.OrderChildrenActionFactory -

You now know how to use the OSGi Installer Configuration Printer to provide the correct JSON format for existing node-based OSGi configuration.

Exercise 2: Configure AEM with OSGi

Scenario: Modify the redirect target location, for requests made to '/', to /sites.html. For example, requests to the URL http://localhost:4502 should redirect to http://localhost:4502/sites.html.

Prerequisites:

- Local AEM author service
- AEM project created from AEM Maven Archetype
- AEM project imported into Eclipse
- Maven 'Run As" Profiles defined

  OSGi configuration files should be under change control. As a result, you should create the configuration files in your AEM project. Deploy the AEM project to AEM to test the newly defined configurations. In this exercise, you define the OSGi configuration files in JSON format, deploy the configurations to AEM and verify the configuration changes.

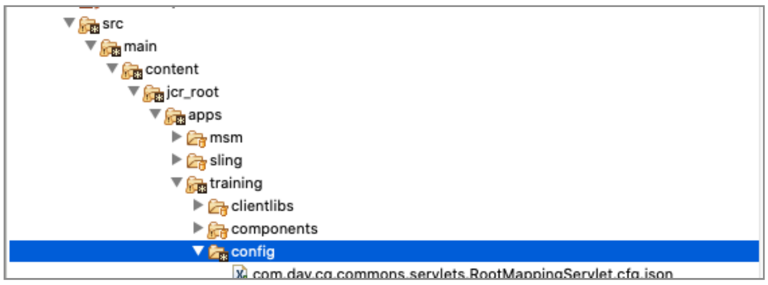
This exercise includes the following tasks:

1. Create an OSGi configuration file in JSON format

2. Install and verify the configuration

Task 1: Create an OSGi Configuration File in JSON format

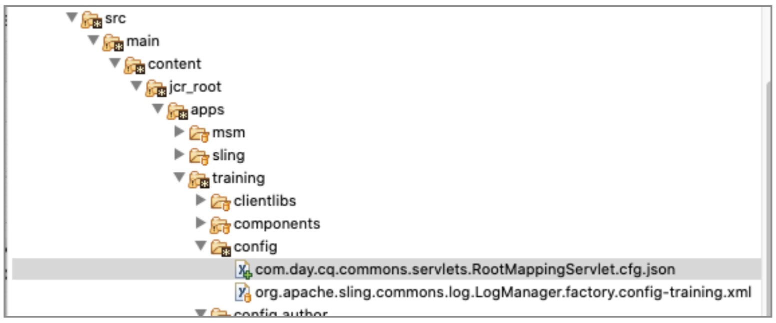
1. Using **Eclipse**, in Project Explorer, navigate to **ui.apps/src/main/content/jcr:root/apps/training**, as shown:



```
▼ src
  ▼ main
    ▼ content
      ▼ jcr_root
        ▼ apps
          ▶ msm
          ▶ sling
          ▼ training
            ▶ clientlibs
            ▶ components
            ▼ config
              com.day.cq.commons.servlets.RootMappingServlet.cfg.json
```

2. Right-click on the **config** folder and select **New > File**.

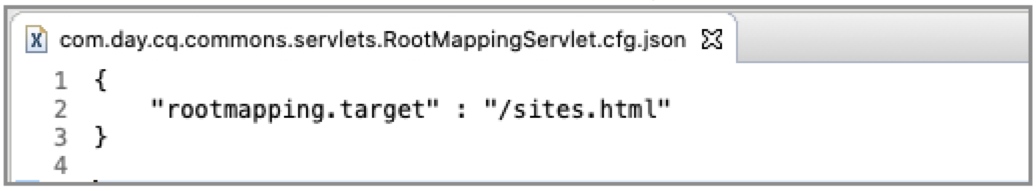
3. Enter **com.day.cq.commons.servlets.RootMappingServlet.cfg.json** as the file name and click **Finish**.



```
▼ src
  ▼ main
    ▼ content
      ▼ jcr_root
        ▼ apps
          ▶ msm
          ▶ sling
          ▼ training
            ▶ clientlibs
            ▶ components
            ▼ config
              com.day.cq.commons.servlets.RootMappingServlet.cfg.json
              org.apache.sling.commons.log.LogManager.factory.config-training.xml
            ▼ config.author
```

4. The file will open in the editor. Copy the contents of the file **com.day.cq.commons.servlets.RootMappingServlet.cfg.json** from **devops-training/training-files/OSGiConfig** and paste it in the newly created file.

5. Verify the file contents and click **Save** to save the changes in the Eclipse editor.



com.day.cq.commons.servlets.RootMappingServlet.cfg.json

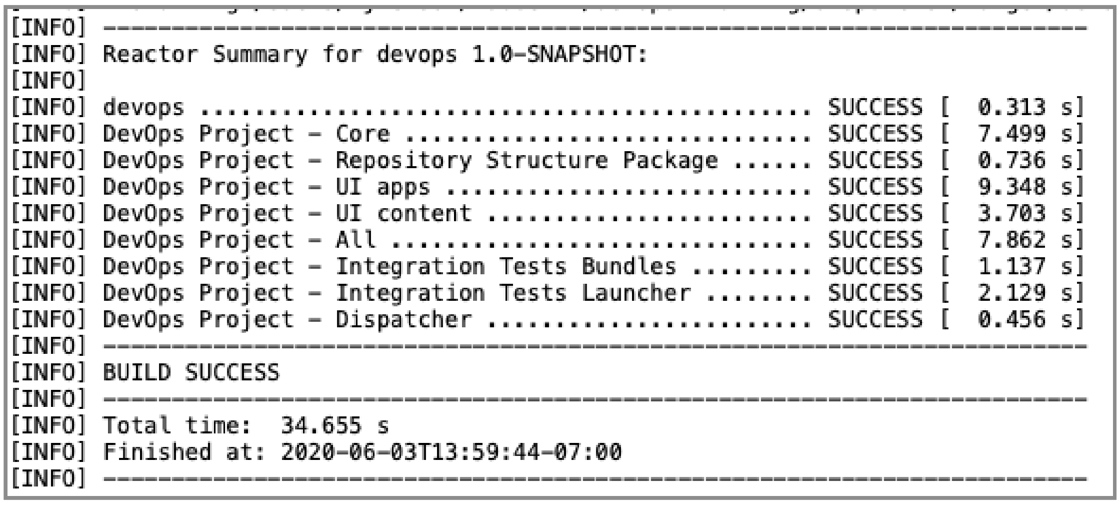
```
{
    "rootmapping.target" : "/sites.html"
}
```

Task 2: Install and Verify the configuration

1. If you do not already have a command line window open in your < AEM Project > directory, open one now.

2. Enter the following command to deploy your changes to the local AEM author instance:
   `$ mvn clean install -P adobe-public -P autoInstallSinglePackage`

3. Verify that the build completed successfully, as shown:

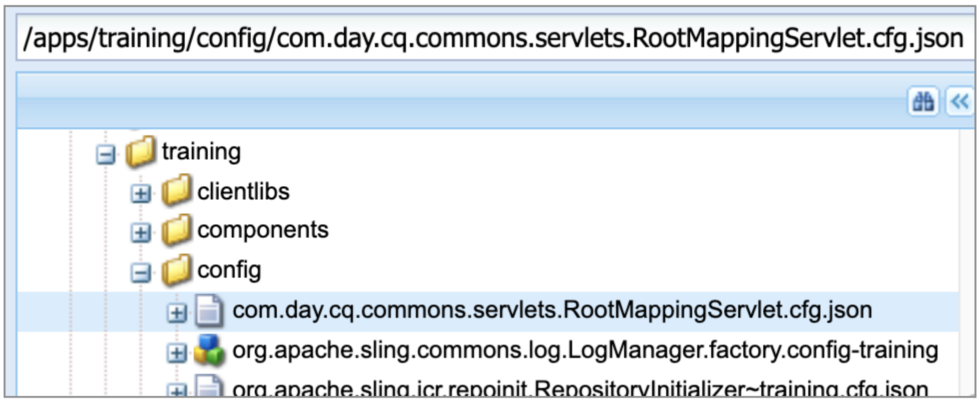


```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for devops 1.0-SNAPSHOT:
[INFO]
[INFO] devops ............................................. SUCCESS [  0.313 s]
[INFO] DevOps Project - Core .............................. SUCCESS [  7.499 s]
[INFO] DevOps Project - Repository Structure Package ...... SUCCESS [  0.736 s]
[INFO] DevOps Project - UI apps ........................... SUCCESS [  9.348 s]
[INFO] DevOps Project - UI content ........................ SUCCESS [  3.703 s]
[INFO] DevOps Project - All ............................... SUCCESS [  7.862 s]
[INFO] DevOps Project - Integration Tests Bundles ......... SUCCESS [  1.137 s]
[INFO] DevOps Project - Integration Tests Launcher ........ SUCCESS [  2.129 s]
[INFO] DevOps Project - Dispatcher ........................ SUCCESS [  0.456 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  34.655 s
[INFO] Finished at: 2020-06-03T13:59:44-07:00
[INFO] ------------------------------------------------------------------------
```

Note: Any changes you sync back to your maven project should be committed into source control:
git commit -a -m "added new files".

4. Open **CRXDE Lite** and navigate to **apps/training/config**.

5. Verify that the **com.day.cq.commons.servlets.RootMappingServlet.cfg.json** OSGi configuration file was deployed. Open the file and verify its contents, as shown:

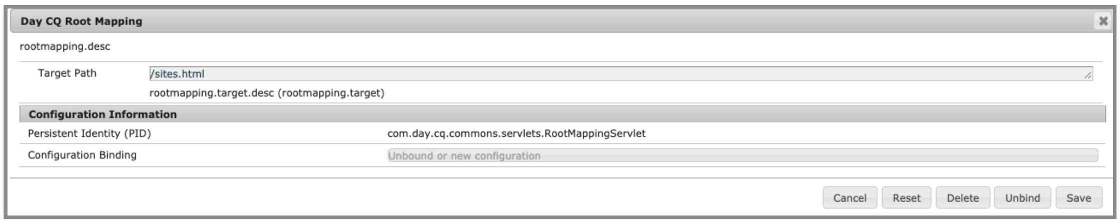


/apps/training/config/com.day.cq.commons.servlets.RootMappingServlet.cfg.json

training
clientlibs
components
config
com.day.cq.commons.servlets.RootMappingServlet.cfg.json
org.apache.sling.commons.log.LogManager.factory.config-training
org.apache.sling.jcr.repoinit.RepositoryInitializer~training.cfg.json

6. On another browser tab, from the **Navigation** page, navigate to **Tools** > **Operations** > **Web Console**.

7. Search for **Day CQ Root Mapping** and verify that the **Target Path** value is /sites.html.



Day CQ Root Mapping

rootmapping.desc

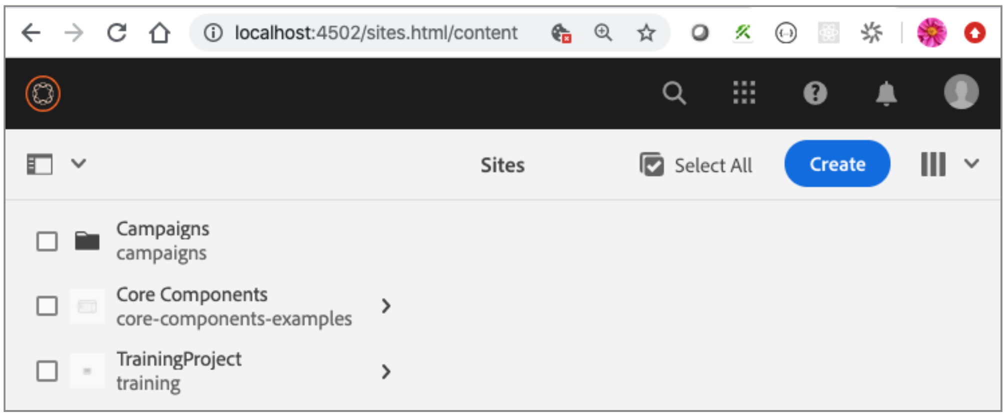
Target Path: /sites.html

rootmapping.target.desc (rootmapping.target)

Configuration Information

Persistent Identity (PID)	com.day.cq.commons.servlets.RootMappingServlet
Configuration Binding	Unbound or new configuration

8. On another browser tab, enter the AEM root path: **http://localhost:4502** and verify that the root path redirects to /sites.html, as shown:



localhost:4502/sites.html/content

You have defined an OSGi configuration file in the JSON format, deployed the configuration to AEM, and verified the results.

Securing Background Services 3

A service is a piece or collection of functionality. Examples of services include the Sling queuing system, Tenant Administration, Event Handlers, custom Workflow processes, and a Message Transfer System. Each service is identified by a unique service name. Because a service is implemented as a component in an OSGi bundle, services are named by the bundles providing them.

Adobe recommends running a service within the calling user's permission context when possible. However, background tasks are often not associated with a specific user. As a result, Apache Sling provides the following three parts mechanism for service authentication and authorization.

- Service user
- Service ID
- ServiceUserMapper

Service Users

A service user is a JCR user with no password set and a minimal set of privileges that are necessary to perform a specific task. Having no password set means that it will not be possible to log in with a service user, except as a known service.

Service ID

A service may be comprised of multiple parts, so each part of the service may be further identified by a subservice name. A subservice name is optional. The examples of subservice name are the names for the subsystems in a Message Transfer System, such as accepting messages, queueing messages, and delivering messages.
The combination of the Service Name and Subservice Name defines the Service ID. The Service ID is finally mapped to a Resource Resolver and/or JCR user ID for authentication.
Therefore, the actual service identification (service ID) is defined as:

```
service-id = service-name [ ":" subservice-name ]
```

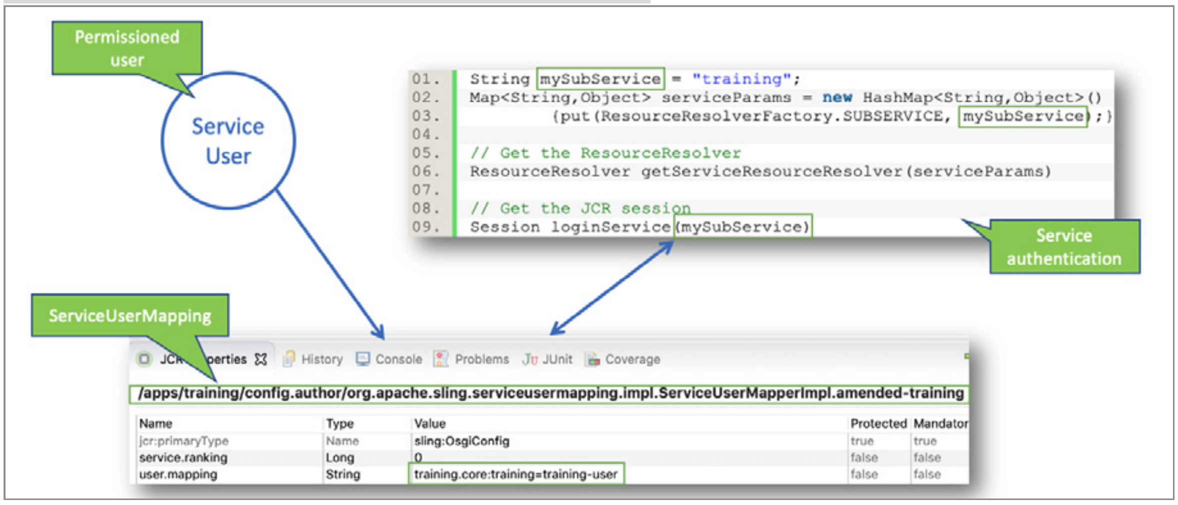
ServiceUserMapper

The ServiceUserMapper service enables you to map the Service IDs comprised of the Service Names defined by the providing bundles and optional Subservice Name to ResourceResolver and/or JCR Repository user IDs. This mapping is configurable such that system administrators are in full control of assigning users to services.

ResourceResolverFactory

The Sling ResourceResolverFactory provides a factory method to allow the service to authenticate to the repository, as shown:

```
ResourceResolver getServiceResourceResolver(Map<String, Object>
authenticationInfo) throws LoginException;
```



```
String mySubService = "training";
Map<String,Object> serviceParams = new HashMap<String,Object>()
        {put(ResourceResolverFactory.SUBSERVICE, mySubService);}

// Get the ResourceResolver
ResourceResolver getServiceResourceResolver(serviceParams)

// Get the JCR session
Session loginService(mySubService)
```

/apps/training/config.author/org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training

Name	Type	Value	Protected	Mandator
jcr:primaryType	Name	sling:OsgiConfig	true	true
service.ranking	Long	0	false	false
user.mapping	String	training.core:training=training-user	false	false

SlingRepositoryInitializer 4

The Apache Sling SlingRepositoryInitializer enables you to create resources at startup before the Sling Repository Service is registered as an OSGi service. As a result, the application logic can consider the existence of those resources for granted. Repoinit statements should be under change control and therefore are defined in OSGi configurations.

The *repoinit* Repository Initialization language:

- Creates paths, service users and Access Control Lists in a content repository
- Registers JCR namespaces and node types

An example of *repoinit* statements:

```
create service user training-user-3
set ACL on /content
  allow jcr:read for training-user-3
end
create path /content/example3.com(sling:Folder)"
create path /content/example3.com(sling:Folder mixin mix:referenceable,
mix:sharable)
```

Note: The repoinit statements are defined in an OSGi configuration (immutable). However, repoinit statements often refer to or modify mutable content.

Exercise 3: Define service user using repoinit

Scenario: A developer is implementing a background service that requires authentication and authorization to write to the repository. A service user is necessary to authenticate the service to the repository and to create a permission context within which the service will run. In addition, the service user needs to be granted the appropriate permissions.
Prerequisites:

- Local AEM author service
- AEM project created from AEM Maven Archetype
- AEM project imported into Eclipse
- Maven 'Run As" Profiles defined

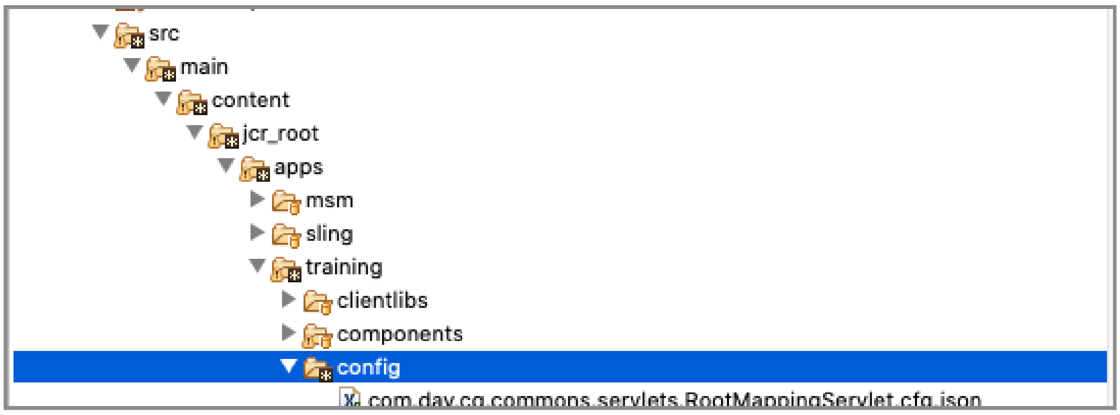
This exercise includes the following tasks:

1. Create an OSGi configuration file in JSON format

2. Install and verify the configuration

Task 1: Define the repoinit OSGi configuration file in JSON format

1. Using Eclipse, in Project Explorer, navigate to **ui.apps/src/main/content/jcr:root/apps/training**, as shown:

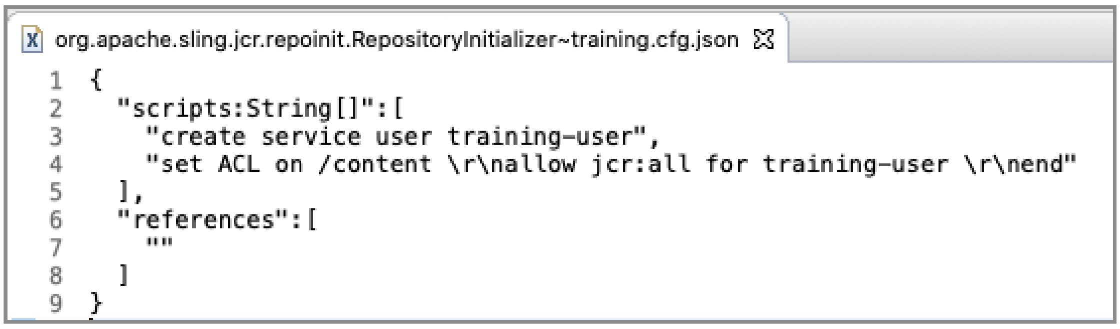


2. Right-click the **config** folder and select **New** > **File**.

3. Enter **org.apache.sling.jcr.repoinit.RepositoryInitializer~training.cfg.json** in the File Name field and click **Finish**. The RepositoryInitializer service is a factory service, so a unique identifier is added to the Persistent Identity of the service.

4. The file will open in the editor. Copy the contents of the file **org.apache.sling.jcr.repoinit.RepositoryInitializer~training.cfg.json** from **devops-training/training-files/OSGiConfig** and paste it in the newly created file.

5. Verify the file contents and click **Save** to save your changes in the Eclipse editor, as shown:



```
{
  "scripts:String[]":[
    "create service user training-user",
    "set ACL on /content \r\nallow jcr:all for training-user \r\nend"
  ],
  "references":[
    ""
  ]
}
```

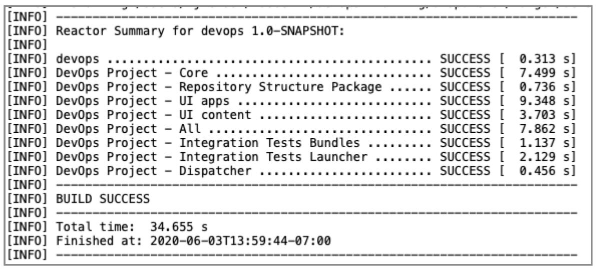
Task 2: Install and verify the configuration

1. If you do not already have a command line window open in your < AEM Project > directory, open one now.

2. Enter the following command to deploy your changes to the local AEM author instance:

```
$ mvn clean install -P adobe-public -P autoInstallSinglePackage
```

3. Verify that the build is completed successfully, as shown:

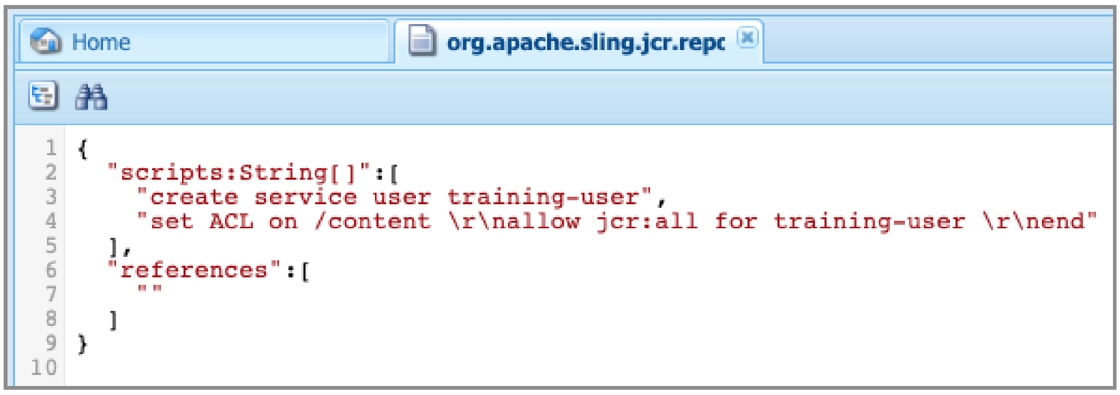


```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for devops 1.0-SNAPSHOT:
[INFO]
[INFO] devops ............................................ SUCCESS [  0.313 s]
[INFO] DevOps Project - Core ............................. SUCCESS [  7.499 s]
[INFO] DevOps Project - Repository Structure Package ...... SUCCESS [  0.736 s]
[INFO] DevOps Project - UI apps .......................... SUCCESS [  9.348 s]
[INFO] DevOps Project - UI content ....................... SUCCESS [  3.703 s]
[INFO] DevOps Project - All .............................. SUCCESS [  7.862 s]
[INFO] DevOps Project - Integration Tests Bundles ......... SUCCESS [  1.137 s]
[INFO] DevOps Project - Integration Tests Launcher ........ SUCCESS [  2.129 s]
[INFO] DevOps Project - Dispatcher ....................... SUCCESS [  0.456 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  34.655 s
[INFO] Finished at: 2020-06-03T13:59:44-07:00
[INFO] ------------------------------------------------------------------------
```

Note: Any changes you sync back to your maven project should be committed into source control:
git commit -a -m "added new files".

4. Open **CRXDE Lite** and navigate to **apps/training/config**.

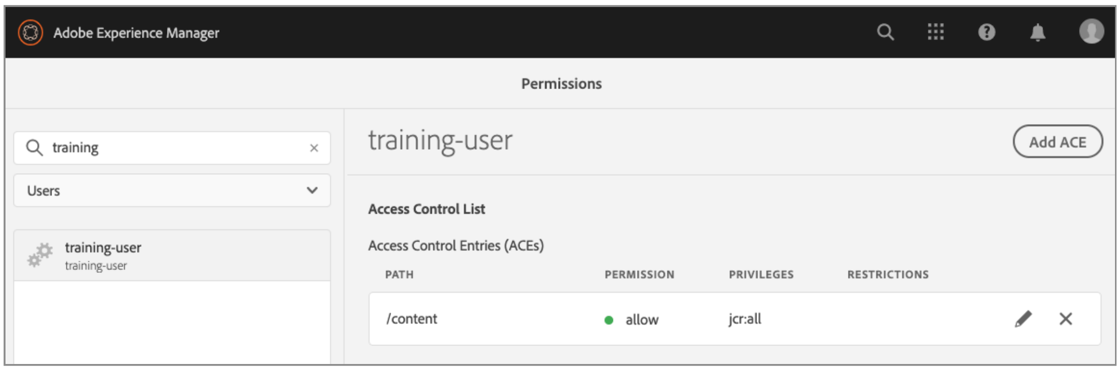
5. Verify that the **org.apache.sling.jcr.repoinit.RepositoryInitializer~training.cfg.json** OSGi configuration file is deployed. Open the file and verify the contents, as shown:



```
{
  "scripts:String[]":[
    "create service user training-user",
    "set ACL on /content \r\nallow jcr:all for training-user \r\nend"
  ],
  "references":[
    ""
  ]
}
```

6. On a browser tab, from the **Navigation** page, navigate to **Tools** > **Security** > **Permissions**.

7. Select Users from the drop-down menu and search for the **training-user** to verify that the user is created and an ACE was created, as shown:



training-user

Access Control List

Access Control Entries (ACEs)

PATH	PERMISSION	PRIVILEGES	RESTRICTIONS
/content	● allow	jcr:all	

You now know how to initialize configuration information for an AEM instance. You have defined a set of repoinit statements in an OSGi configuration, deployed the configuration to AEM and verified the results.

References

1. Configuring OSGi for AEM as a Cloud Service: https://docs.adobe.com/content/help/en/experience-manager-cloud-service/implementing/deploying/configuring-osgi.html ↩

2. Maintenance Tasks: https://docs.adobe.com/content/help/en/experience-manager-cloud-service/operations/maintenance.html ↩

3. Service Authentication: https://sling.apache.org/documentation/the-sling-engine/service-authentication.html ↩

4. Apache Sling Repository Initializer: https://sling.apache.org/documentation/bundles/repository-initialization.html ↩