PRO!VISION
SOFTWARE CRAFTSMANSHIP

AEM Configuration with CONGA

Configuration Management for AEM environments

PVTRAIN-146

Technical Training – wcm.io

Last Updated: September 2019

https://training.wcm.io/conga/

Challenges of AEM Configuration

System configuration for OSGi, Apache Sling and AEM

Apache Sling and OSGi system configuration

- **Sling Provisioning File Format *)**
  - A lot of Apache Sling tooling makes use of this, e.g. Slingstart Maven Plugin
  - Adobe also uses this format internally for building the AEM Quickstart JAR
  - Text-based format inspired by YAML, but with proprietary syntax
  - Can contain definitions for bundles and versions, configurations, run modes and special features like repository initialization
- **OSGi configurations** – Apache Felix Config Admin File Format
  - In Apache Sling OSGi configurations can be provided via filesystem folder or repository
  - Uses a configuration file format syntax defined by the Apache Felix project
  - Has some very special escaping rules which need to be respected
  - The syntax is also used inside the Sling Provisioning files for configurations

*) A switch to the new „Feature Model“ file format is currently on the way in Sling/AEM

AEM system configuration

- **AEM Content Package**
  - AEM Content Packages are ZIP files containing repository content in FileVault XML format
  - Content Packages are used to deploy configurations or content to AEM instances
  - Some configurations target OSGi services and use the Felix Config Admin file format, others are content structures with nodes and properties
  - Content Packages have additional metadata which may define filtering rules, handling of ACLs in content structures, requirements for restart etc.
- **Dispatcher ANY file format**
  - The Dispatcher webserver modules is configured via it's own "ANY" file format
  - This format has a very special syntax something between XML and JSON

CONGA Plugins and definitions

- Due to the modular architecture of CONGA it is easy to add support for managing these special file formats unique to Sling and AEM
- Two plugin artifacts are provided, each contains a set of technical plugins based on the CONGA extensibility interface (file headers, validators, escaping, post processors)
  - **CONGA Sling Plugin**
  - **CONGA AEM Plugin**
- Additionally a generic set of “AEM configuration definitions” is provided which implements best practices for configuring AEM environments
  - **CONGA AEM Definitions**
- Both plugins and definitions can be added to the CONGA build by simply adding them as dependencies in the POM to the CONGA Maven Plugin

CONGA Sling Plugin

Manage OSGi configuration for Sling and AEM applications

CONGA Sling plugin

Extends CONGA with:

- Manage OSGi configuration templates in Apache Sling Provisioning file format
- Generate OSGi configurations in Apache Felix Config Admin file format

Documentation:
https://devops.wcm.io/conga/plugins/sling/

OSGi configurations

- CONGA Sling plugin provides a **File Header** and **Escaping** plugin for OSGi configuration files using the Apache Felix Config Admin file format. They are automatically applied.
- File extension is **`.config`**
- The format supports configuration parameter of different data types.
- Single values and array values are supported.
- The first line of such a file might start with a comment line. Besides this inline comments are not allowed.
- Detailed description of the format:
  https://sling.apache.org/documentation/development/slingstart.html#default-configuration-format

OSGi configurations – example

`.config` file example:

```
# Single Line Comment
stringParam="Hello\ World"
stringArrayParam=["Hello","World"]
intParam=I"123"
intArrayParam=I["123","456"]
longParam=L"123456"
doubleParam=D"1.23"
booleanParam=B"true"
```

Supported data types:

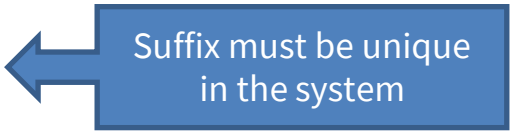
- 'I' : Integer
- 'L' : Long
- 'F' : Float
- 'D' : Double
- 'X' : Byte
- 'S' : Short
- 'C' : Character
- 'B' : Boolean

- The name of the config file is the OSGi PID
- Escaping with “\” required for: Quotes, double quotes, backslash, equals sign, space character

OSGi configurations – PIDs

- Within OSGi all singleton configurations have a unique PID
  - The PID is often equal to the class name of the service implementation, but may also be a custom one chosen by the implementor of the service
  - Example:
    `x.y.z.MyService`

- When factory configurations are used the factory PID is used with a sub name separated by "-"
  - Example:
    `x.y.z.MyServiceFactory-one`
    `x.y.z.MyServiceFactory-two`



- The PID is used a part of the file name e.g. **`x.y.z.MyService.config`**

Provisioning files

- CONGA Sling plugin provides a **File Header**, **Validator** and **Escaping** plugin for provisioning files. They are automatically applied.
- Additionally a **Post Processor** plugin is provided which generates a set of OSGi config files from a single provisioning files for all configurations contained.
- File extension is `.provisioning` or `.txt`
  - Sling defines only “txt” as file extension. To distinguish the files without doubt from plain text files it is recommended to use only the “provisioning” file extension within CONGA.
  - “txt” files are treated as provisioning files if they contain the string “[feature ” (heuristic)
- Detailed (although currently incomplete) description of the format: https://sling.apache.org/documentation/development/slingstart.html#model-files

Provisioning files – example

`.provisioning` file example:

```
[feature name=my-feature]

[configurations]
   # Comments are allowed
   x.y.z.MyService1
      stringParam="Hello\ World"
      stringArrayParam=["Hello","World"]

   x.y.z.MyService2
      intParam=I"123"

# Configuration applies only to certain run modes
[configurations runModes=author,runmode2]
   x.y.z.MyServiceFactory-one
      doubleParam=D"1.23"
      booleanParam=B"true"
```

Provisioning files – further notes

- The Sling provisioning file format supports much more features – CONGA uses only the configuration parameters from the `[configuration]` sections.
- The feature name is irrelevant for CONGA (but a name must be given to have a valid provisioning file).
- The configuration parameter key/value lists use the same syntax as the OSGi configurations of the Felix Config Admin file format.
- Within the configuration sections the PIDs or factory PID plus sub name are used to identify the configuration (same as the file names for .config files)
- When run modes are given the configuration is only applied to Sling/AEM instances running in all of the given run modes.

Provisioning files post processor

Example for applying the **`sling-provisioning-osgiconfig`** post processor within a CONGA role definition:

```
- file: sling-provisioning.provisioning
  dir: osgi-config
  template: sling-provisioning.provisioning.hbs
  # Transform provisioning file to single OSGi config files
  postProcessors:
  - sling-provisioning-osgiconfig
```

Template example:

```
[feature name=example]
[configurations]
  my.pid
    heapspaceMax="{{jvm.heapspace.max}}"
[configurations runModes=mode1]
  my.pid2
    stringProperty="{{var1}}"
    stringProperty2="{{var2}}"
```

CONGA AEM Plugin

Manage AEM content packages and AEM Dispatcher

CONGA AEM plugin

Extends CONGA with:

- Generate AEM content packages for OSGi configurations and from JSON content fragments
- Extract package properties from AEM content packages
- Manage ANY files for dispatcher configuration

And provides a **CONGA AEM Maven plugin** to deploy a bunch of AEM packages processed by CONGA to an AEM instance.

Documentation:
https://devops.wcm.io/conga/plugins/aem/

Generating AEM Content Packages for OSGi configs

- CONGA AEM plugin provides a **Post Processor** plugin for provisioning files that transforms the contained OSGi configurations to .config files and bundles them in an AEM content package that can be deployed to AEM.
- Usually the provisioning file was generated by a provisioning file template with placeholders. The generated AEM content package then contains the generated configuration for the environment.
- Run modes and factory configurations are supported as well.
- The metadata of the content package can be defined via post processor options (e.g. package group, name and filters).

Generating AEM Content Packages for OSGi configs

Example for applying the **`aem-contentpackage-osgiconfig`** post processor within a CONGA role definition:

```
- file: sling-provisioning.provisioning
  dir: packages
  template: sling-provisioning.provisioning.hbs
  # Transform OSGi configs from provisoning file to AEM content package
  postProcessors:
  - aem-contentpackage-osgiconfig
  postProcessorOptions:
    contentPackage:
      group: my-group
      name: config-sample
      description: The description of the sample package.
      version: "${version}"
      rootPath: /apps/sample/config
      filters:
      - filter: /apps/sample
```

AEM Content Package metadata

All post processors of the CONGA AEM plugin support these post processor options for defining the metadata of the content package:

Property	Description
`contentPackage.group`	Group name for content package
`contentPackage.name`	Package name for content package
`contentPackage.description`	Description for content package
`contentPackage.version`	Version for content package
`contentPackage.rootPath`	Root path for the content package
`contentPackage.filters`	Contains list with filter definitions, optionally with include/exclude rules. If not defined a simple filter rule is derived from the **`contentPackage.rootPath`** property.
`contentPackage.acHandling`	How to apply ACLs that are contained in the content package. Possible values: `ignore` (default), `overwrite`, `merge`, `merge_preserve`, `clear`.

Full list of supported package properties: https://devops.wcm.io/conga/plugins/aem/extensions.html

Generating AEM Content Packages from JSON

- CONGA AEM plugin provides a **Post Processor** plugin that transforms content structures from JSON files to AEM content packages.
- The JSON files use the same syntax which is produced by the Sling GET Servlet when calling a resource with .json file extension.
- The JSON files can be generated by a file templates thus can contain configuration parameters for the current environment.
- Use case examples:
  - Generate Sling Mapping Configuration
  - Create a package with system users and their ACLs on content paths
  - Create some root folders with special filter rules
- Via post processor options the metadata of the content package can be defined (e.g. package group, name and filters).

Generating AEM Content Packages from JSON

Example for applying the **`aem-contentpackage`** post processor within a CONGA role definition:

```
# AEM systems users with ACLs
- file: aem-systemusers.json
  dir: packages
  template: aem-systemusers.json.hbs
  # Transform JSON file to AEM content package
  postProcessors:
  - aem-contentpackage
  postProcessorOptions:
    contentPackage:
      name: aem-systemusers
      acHandling: merge
      rootPath: /
      filters:
      - filter: /content/rep:policy
      - filter: /home/users/system/sampleSystemUser
```

Generating AEM Content Packages from JSON

Example JSON for creating system users and ACLs:

```
{
  "jcr:primaryType": "rep:root",
  "content": {
    "jcr:primaryType": "sling:OrderedFolder",
    "rep:policy": {
      "jcr:primaryType": "rep:ACL",
      "allow-sampleSystemUser": {
        "jcr:primaryType": "rep:GrantACE",
        "rep:principalName": "sampleSystemUser",
        "rep:privileges": [ "rep:write", "crx:replicate" ]
      }
    }
  },
  "home": {
    "jcr:primaryType": "rep:AuthorizableFolder",
    "users": {
      "jcr:primaryType": "rep:AuthorizableFolder",
      "system": {
        "jcr:primaryType": "rep:AuthorizableFolder",
        "sampleSystemUser": {
          "jcr:primaryType": "rep:SystemUser",
          "jcr:uuid": "{{oakAuthorizableUuid "sampleSystemUser"}}",
          "rep:authorizableId": "sampleSystemUser",
          "rep:principalName": "sampleSystemUser"
        }
      }
    }
  }
}
```

Sets ACL for system user at `/content`

Creates system user at `/home/users/system/sampleSystemUser`

Adding binary files to AEM content packages

- It's also possible to add additional binary files to AEM content packages using the CONGA AEM plugin
- The files can be static ones from classpath, URL of maven artifact, or files generated by CONGA
- Documentation: https://devops.wcm.io/conga/plugins/aem/extensions.html
- Usage example for generating AC Tool files: https://wcm-io.atlassian.net/wiki/x/AQDYEQ

Extracting AEM content package metadata

- CONGA AEM plugin provides a **Post Processor** plugin **`aem-contentpackage-properties`** that is automatically applied to all ZIP files generated or copied/downloads by CONGA that are actually AEM content packages.
- The package properties of these content packages are extracted and stored in the CONGA model metadata.
- This has no effect on the generated configuration artifacts, but can be picked up by IT automation tools for further processing the content packages managed by CONGA.
  - Example: From this package metadata the Ansible AEM deployment knows if the instance needs to be restarted after package deployment.
  - See training **PVTRAIN-147 AEM Deployment with Ansible and CONGA** for details

AEM Dispatcher ANY files

- CONGA AEM plugin provides a **File Header**, **Validator** and **Escaping** plugin for ANY files. The are automatically applied.
- File extension is **`.any`**

Example ANY template:

```
# name of the dispatcher
/name "{{node}}"

# each farm configures a set of (loadbalanced) renders
/farms
 {
   # first farm entry (label is not important, just for your convenience)
   /website
     {
     /cache
       {
       # Cache configuration
       /rules
         {
         # List of cachable documents
         }
       /invalidate
         {
         # List of auto-invalidated documents
         }
       }
     /retryDelay "1"
     /numberOfRetries "5"
     /unavailablePenalty "1"
     /failover "1"
     }
 }
```

CONGA Maven AEM Plugin

- This is an AEM-specific CONGA plugin for Maven, not to be mixed up with the generic CONGA plugin for Maven which is used to generate the configuration.
- The **CONGA AEM Maven plugin** allows to deploy a bunch of AEM packages processed by CONGA to an AEM instance. It requires the CONGA configuration to be generated before, and a **`model.yaml`** needs to be located in each node's root folder (this is activated by default).

```
<plugin>
  <groupId>io.wcm.devops.conga.plugins</groupId>
  <artifactId>conga-aem-maven-plugin</artifactId>
  <configuration>
    <nodeDirectory>target/configuration/env1/node1</nodeDirectory>
    <serviceURL>http://localhost:4502/crx/packmgr/service</serviceURL>
    <userId>admin</userId>
    <password>admin</password>
  </configuration>
</plugin>
```

- Deploy all AEM packages processed by CONGA with:
  `mvn conga-aem:package-install`

Uses the same “resilience” package upload logic as the wcm.io Content Package Maven Plugin

CONGA AEM Custom Handlebars expressions

Custom AEM-specific Handlebars expressions (selection):

- **aemCryptoEncrypt** – Encrypts a password or other secret with the AEM crypto AES key.
- **oakPasswordHash** – Generates a password hash for an Oak JCR user from a plain text password.
- **oakAuthorizableUuid** – Generates a UUID for an authorizable node by deriving it from the authorizable Id.
- **webconsolePasswordHash** – Generates a password hash for the Apache Felix Webconsole (felix.webconsole.password)

The full list CONGA AEM-specific expressions is documented here:

https://devops.wcm.io/conga/plugins/aem/handlebars-helpers.html

Password Encryption in AEM

- AEM uses a **symmetric-key encryption** to protected passwords stored in OSGi configuration and repository.
  - This does not apply to the Oak repository passwords – they are stored and transported in packages only as hashes
- The encryption is based on a “crypto key” stored in the file system of each AEM instance (outside the repository)
- It is recommended that all AEM instances of one environment share the same crypto key.
  - The wcm.io DevOps Ansible tooling takes care of this

See also AEM Documentation:
https://helpx.adobe.com/experience-manager/6-5/sites/administering/using/security-checklist.html#MakeSureYouProperlyReplicateEncryptionKeysWhenNeeded

Password Encryption in AEM with CONGA

- CONGA AEM Plugin can generate a new crypto keys when new projects are set up
  - This is used by the Maven Archetype for AEM Configuration Management https://wcm.io/tooling/maven/archetypes/aem-confmgmt/
- CONGA AEM Plugin can encrypt passwords during the configuration generation using this key.
  - via Custom Handlebar expressions
  - Configuration files generated with CONGA should never contain clear text passwords
- CONGA AEM Plugin also provides a Command Line Interface (CLI) https://devops.wcm.io/conga/plugins/aem/crypto-cli.html

CONGA AEM Definitions

Predefined roles and templates for AEM best practices

CONGA AEM definitions

- A set of preconfigured CONGA roles and file templates for configuring an AEM environment using best practices
- Generates configurations for both AEM Author/Publisher and Webserver/Dispatcher
- Makes sure that configuration between AEM and dispatcher is always in sync (e.g. Sling Short URL mapping configuration)
- Usually mixed and extended with own project-specific roles

Documentation:
https://devops.wcm.io/conga/definitions/aem/

Role aem-cms

- Variants: **`aem-author`**, **`aem-publish`**

Features:

- Sling Mapping configuration for publish instance
- AEM replication configuration between author and publish
- AEM quickstart start script with JVM and AEM startup parameters
- Configure Sling Context-Aware Configuration OSGi overrides
- Enabled DavEx for CRX DE Lite
- Set Felix OSGi Management Console authentication
- Provide AEM Crypto keys

Role aem-dispatcher

Variants: **`aem-author`**, **`aem-publish`**, **`ssl`**

Features:

- Generates Apache HTTPd configuration files for Dispatcher webserver
- Generates Dispatcher configuration for author and publish instances
- Best practice default filter and caching rules, can be adapted to project needs via configuration
- Generates vHost for each tenant on publish
- SSL, HSTS and HTTP/2 Support
- Short URL configuration with Sling Mapping
- Enables CORS (optional)
- Configuration files use partials, can be overloaded and overwritten partially
- Supports Apache httpd 2.2 and 2.4

CONGA AEM definitions

For a detailed documentation of available parameters look into the role definitions and templates:

- Roles
  https://github.com/wcm-io-devops/conga-aem-definitions/tree/develop/conga-aem-definitions/src/main/roles
- Templates
  https://github.com/wcm-io-devops/conga-aem-definitions/tree/develop/conga-aem-definitions/src/main/templates
- Example environment using the roles and templates
  https://github.com/wcm-io-devops/conga-aem-definitions/tree/develop/example/src/main/environments

Bringing it together

Generate configuration for the whole AEM environment

Bringing it together

- For AEM projects you usually use everything together:
  - CONGA via CONGA Maven Plugin
  - CONGA AEM Sling and AEM plugins as plugin dependencies
  - CONGA AEM definitions as dependency
  - Add project-specific roles and templates
  - If really required: Overwrite some partials for webserver/dispatcher config
  - Define the project-/customer-specific environments
- Use this CONGA configuration for
  - Configuring local development AEM instances
  - Deploy to test and production systems via IT automation (e.g. Ansible)
  - Or just use CONGA to package all configuration artifacts in a ZIP file and send it to the operations team for further processing

Example POM

```
<project>
  <groupId>io.wcm.devops.conga.definitions</groupId>
  <artifactId>io.wcm.devops.conga.definitions.aem.example</artifactId>
  <packaging>config</packaging>

  <dependencies>
    <dependency>
      <groupId>io.wcm.devops.conga.definitions</groupId>
      <artifactId>io.wcm.devops.conga.definitions.aem</artifactId>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>io.wcm.devops.conga</groupId>
        <artifactId>conga-maven-plugin</artifactId>
        <extensions>true</extensions>
        <dependencies>
          <dependency>
            <groupId>io.wcm.devops.conga.plugins</groupId>
            <artifactId>io.wcm.devops.conga.plugins.sling</artifactId>
          </dependency>
          <dependency>
            <groupId>io.wcm.devops.conga.plugins</groupId>
            <artifactId>io.wcm.devops.conga.plugins.aem</artifactId>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>

</project>
```

This is already included in the `aem-global-parent` POM.

Typical Maven project structure

- Git project for application and configuration definitions
  - Published to Maven Artefact Manager, Releases with application

```
myproject
  |
  +-- bundles
  |
  +-- config-definition
  |
  +-- content-packages
  |
  +-- ...
```

Contains configuration definitions – **CONGA roles and templates**

Typically this also contains a **CONGA environment definition for development** (local AEM instance)

- Git project for configuration environments
  - Usually not published to Maven Artefact Manager

```
myproject-configuration-management
  |
  +-- configuration
```

Contains CONGA Environments for different stages, e.g. QS, Prelive, Prod

Using the wcm.io Maven Archetypes

- Typically, you do not have to setup all this manually, but you are using the wcm.io Maven Archetype to set up new projects – they come preconfigured with CONGA support

- **Maven Archetype for AEM**
  https://wcm.io/tooling/maven/archetypes/aem/
  - Sets up a new best-practice AEM project including CONGA configuration definitions based on the CONGA AEM Definitions

- **Maven Archetype for AEM Configuration Management**
  https://wcm.io/tooling/maven/archetypes/aem-confmgmt/
  - Sets up a corresponding “configuration management” project containing the environment definitions
  - Can also generate the required Ansible, Vagrant and AWS setup (via Terraform)

Exercise

Execute exercise

PVTRAIN-148-04 Configure AEM with CONGA

- Configure AEM OSGi configuration
- Deploy additional AEM packages
- Generate configuration content packages