CONGA Overview

General concepts and usage

PRO!VISION
SOFTWARE CRAFTSMANSHIP

PVTRAIN-145

Technical Training – wcm.io

About CONGA

CONfiguration GenerAtor

Yet Another Configuration Generator?

- A lot of configuration generator tools already exist

- IT Automation Tools like Ansible, Puppet, Chef have their own concepts and tools for generating configuration files from templates

- But: None of them knows the **specialties of AEM, Sling and OSGi**

- It's very quite hard to generate AEM-specific configuration with them because the target is **not a simple text-based format**

- We want a tool that is **well-integrated with Maven** and typical Java CI/CD infrastructures

CONGA targets system configuration

- CONGA focuses on **System Configuration** that is usually defined at deployment time and is static at runtime.

    – It is not targeted to "runtime configuration" like site configuration, tenant configuration that can be changed at any time by authorized users

- CONGA is **<u>not</u> a deployment automation tool** – it focuses only on configuration generation.

    – can be integrated in an automated deployment process

    – or used for manual or simple script-based deployment

CONGA is flexible

- CONGA is **not limited to a specific type of application** or runtime environment, any system that relies on system configuration stored somewhere can be provisioned with this tool.

  – Typical target systems we had in mind when designing the tool are: AEM, AEM Dispatcher, Apache Tomcat and Apache HTTPd

- It **generates files** of any type, e.g.
  – Plain text files e.g. Properties, Scripts, Webserver configuration
  – JSON files
  – XML files
  – OSGi configuration snippets
  – Sling Provisioning Model
  – AEM Content Packages containing OSGi configurations

CONGA technology stack

- Implemented with **Java 8**
- **Maven Plugin** (standalone CLI available as well)
- **Handlebars** templating
- **YAML** files for role and environment definitions

- CONGA has a modular and plugin-based architecture
- Knowledge of new config formats can easily be added
- Generic formats like JSON and XML are supported out-of-the-box
- Plugins for Sling and AEM are provided

CONGA is Open Source

- **Apache 2.0 License**

- Sources on Github:
  https://github.com/wcm-io-devops/conga

- Documentation:
  http://devops.wcm.io/conga/

- First published in 2015

- Regular releases
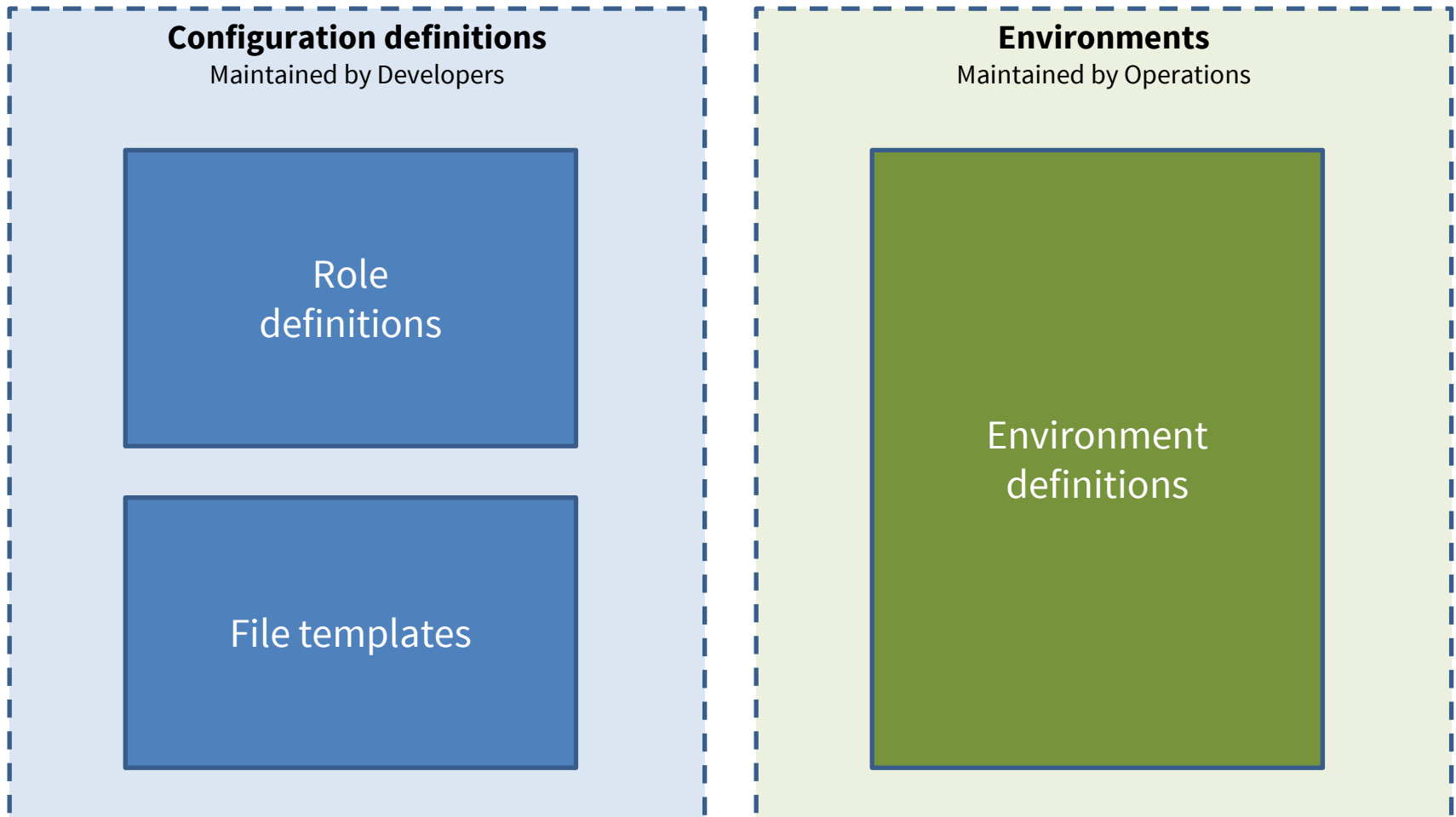
- Maintained by pro!vision

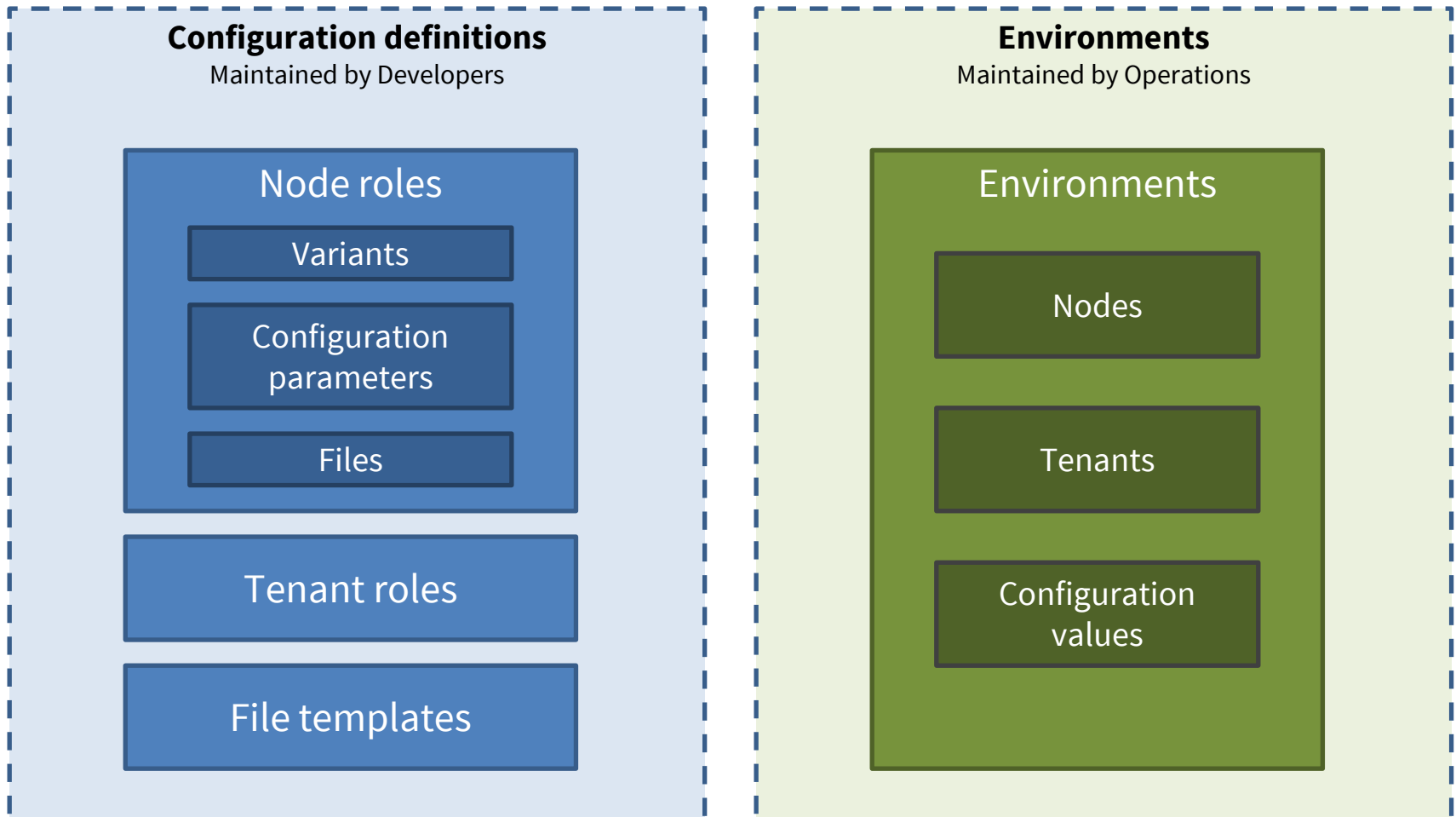# General Concepts

Terminology and concepts

Built for DevOps

- CONGA **separates the templates** for the generated files **from the actual configuration values** needed for each environment

- For each target environment only a **"high-level" parameter file** needs to be maintained

- CONGA **generates the complete configuration** from them

- Ideal for integrated DevOps teams, but it also provides a good level of separation of concerns if Dev and Op-Teams are separated organizationally

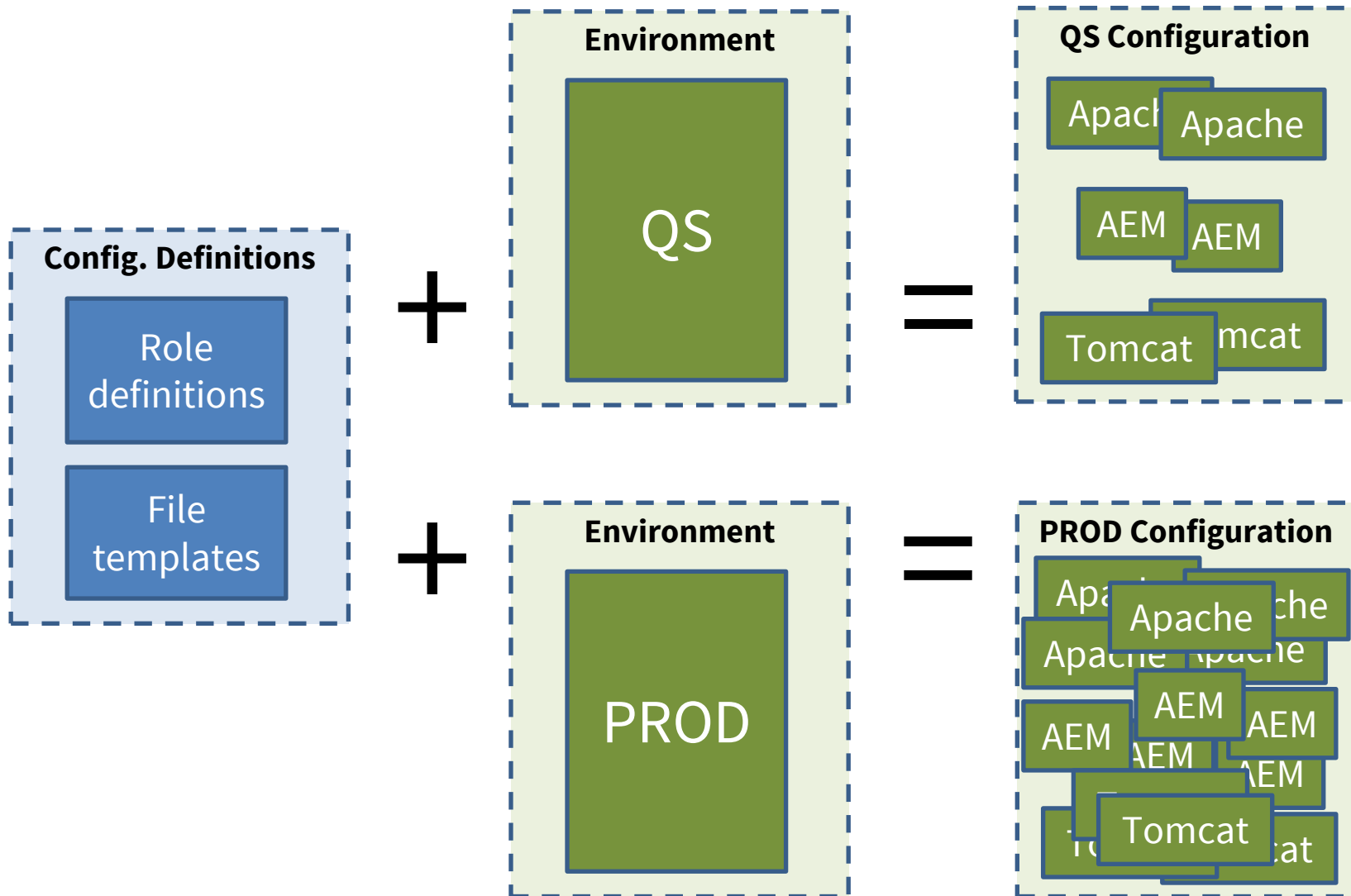# Configuration meta model

Configuration definitions
Maintained by Developers

Role
definitions

File templates

Environments
Maintained by Operations

Environment
definitions

Configuration definition model

Generated configuration example

Environments

- **Environment**: Environment for a specific project or group of projects with a selection of nodes that work together, e.g. "QS", "Prelive", "Prod" etc.

- **Node**: A system to deploy to, e.g. a physical machine, virtual machine, Docker container or any other deployment target.
  - For each node multiple roles can be assigned; for each role one variant
  - It is possible to assign a role multiple times to a node with different variants

- **Tenant**: List of tenants in the environment and their configuration
  - For each tenant multiple tenant roles can be assigned

- **Configuration value**: Configuration value for a configuration parameter in context of environments, nodes, roles and tenants.

Configuration definitions

- **Node role**: A set of functionality/application part that can be deployed to a node/machine, e.g. "AEM CMS", "AEM Dispatcher", "Tomcat Service Layer"
  - **Variant**: Variants of a role with same deployment artifacts but different configuration; e.g. "Author", "Publish", "Importer".
  - **Configuration parameter**: Definition of configuration parameters that can be set for each environment. The configuration parameter values are merged with the file templates when generating the configuration.
  - **File**: Defines file to be generated for Role/Variant based on a File Template

- **Tenant role**: Allows to define features required for a tenant, e.g. Tenant Website with or without additional applications

- **File template**: Script-based template the contains static configuration parts and placeholders for the configuration parameter values

Multitenancy

- Often a single infrastructure environment is used to host applications and websites for multiple tenants (e.g. for multiple markets or different brands)

- Most of this multi-tenancy aspects are managed outside the system configuration (e.g. in content hierarchy and content pages, context-aware configuration in repository)

- But in some occasions the system configuration is affected as well, e.g.
  - One vhost file for each tenant's website in the webserver configuration
  - Short URL Mapping in Dispatcher and AEM for each website

- To support this **tenants** may be defined in each environment, and it is possible to override some of the configuration parameters with tenant-specific values

- Using the "Tenant Multiply" plugin it is possible to generate multiple configuration files (one per tenant) based on a single file template.

- Tenants are independent from roles and role variants from the configuration definition. Tenant roles are specific to tenants and allow to express different characteristics of tenants e.g. with or without a specific feature-set.

File headers

- CONGA automatically adds a file header to each file/artifact it generates to notice that it is automatically generated.

- Additionally the header contains information which environment, role, variant and versions were used to generate this file.

- CONGA detects the file format automatically and applies the appropriate comment syntax.

Example:

```
# *********************************************************************
#
#    This file is AUTO-GENERATED by CONGA. Please do no change it manually.
#
#    Version 1-SNAPSHOT
#    Environment: prod
#    Role: tomcat-services
#    Variant: importer
#    Template: setenv.sh.hbs
#
#    Dependencies:
#    io.wcm.devops.conga/io.wcm.devops.conga.example.definitions/1-SNAPSHOT
#
# *********************************************************************
```

How to run CONGA

CONGA tooling

Run CONGA

- You have two alternatives to run CONGA
  - **Via Maven (recommended)**
  - Directly from the command line (via CLI)

- When executing via Maven you have more features like
  - Building maven artifacts with CONGA definitions
  - Use Maven versioning for configuration management
  - Use Maven repositories to distribute configuration definitions

Run CONGA via Maven

The CONGA plugin hooks into the Maven lifecycle:

- **`mvn validate`**
  - Validate definition files for syntax errors

- **`mvn generate-resources`**
  - Generate configurations

- **`mvn package`**
  - Package definitions or generated configurations as ZIP file

- **`mvn install`**
  - Install definitions or generated configurations in local Maven repository

Configure CONGA in your pom.xml

- Define the CONGA Maven Plugin

```xml
<build>
    <plugins>
      <plugin>
        <groupId>io.wcm.devops.conga</groupId>
        <artifactId>conga-maven-plugin</artifactId>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
```

- Create a dedicated Maven module for your configuration definitions
  - With `<packaging>config-definition</packaging>`
  - This module is usually part of your application's Maven POM hierarchy

- And a separate Maven project for your environments
  - With `<packaging>config</packaging>`
  - This is usually not part of your application

Maven project folder structure

Folder conventions used by CONGA:

Configuration definitions

- `src/main/roles`
  - Role definitions with available configuration parameters
- `src/main/templates`
  - Handlebars templates for generating files

Environments

- `src/main/environments`
  - Environment files with nodes, role references and configuration parameters
- `target`
  - Configuration is generated to the target folder

Exercise

Execute exercise

PVTRAIN-148-01 See CONGA in action

- Execute CONGA via Maven
- Have a first look at the role definitions, templates and environments
- Inspect the generated configuration

YAML syntax

Short introduction of YAML syntax

About YAML syntax

- CONGA uses YAML 1.1 syntax for role and environment definition metadata.

- Here is a good introduction of YAML syntax basics:
  http://docs.ansible.com/ansible/YAMLSyntax.html

- Normally you do not need to quote strings, even if they contain special chars like spaces.
  - If you want to quote them, use single quotes `'` for a 1:1 representation of the string, or double quotes `"` if you want to interpret control chars like `\n`.

- Never use tabs in YAML files.
  - Configure your text editor to always insert spaces instead of tabs (not only for YAML files)
  - Use default tab width of 2 characters

CONGA Environments

Configure nodes, tenants and parameters

CONGA Environment

A CONGA environment consists of

- **List of nodes** (target machines)

- **List of roles** for each node (= what is installed on each node)

- Role-specific global configuration (optional)

- Global configuration (optional)

- Tenant definitions (optional)

An environment is described using a YAML file.

Full documentation in JavaDocs:
http://devops.wcm.io/conga/generator/apidocs/io/wcm/devops/conga/model/environment/Environment.html

Define nodes

- Defines two nodes with one role each without further configuration
- Node name is either a symbolic name, or a real host name
- Role and variant names are defined in the configuration definition

```
# Defines an environment
nodes:

# Example node with 1 role
- node: services-2
  roles:
  - role: tomcat-services
    variant: services

# Example node with 1 role
- node: webserver
  roles:
  - role: webserver
```

Generate configuration for role **tomcat-services**
But only files assigned to variant **services**

Define nodes

- Defines one nodes with two roles and config parameters
- Configuration parameters can be referenced via `${}`

```
# Example node with 2 roles
- node: services-1
  # Config for all roles in this node
  config:
    jvm.heapspace.max: 2048m
    topologyConnectorPath: /specialConnector
  roles:
  - role: tomcat-services
    # Variants allow to pick a specific sub-configuration of a role
    variant: importer
    # Config only for this role
    config:
      topologyConnectors:
      # Merge with list defined already for this parameter
      - _merge_
      - http://host3${topologyConnectorPath}
  - role: tomcat-backendconnector
```

Global configuration

- Configuration parameters can be defined globally for all nodes and roles

```
# Global configuration
config:
  # It is possible to use a shortcut definition for nested maps.
  #   jvm.heapspace.max: 4096m
  # is equivalent to
  #   jvm:
  #     heapspace:
  #       max: 4096m
  jvm.heapspace.max: 4096m
  # Configuration entries can be used as variables for other entries
  topologyConnectorPath: /connector
  topologyConnectors:
  - http://host1${topologyConnectorPath}
  - http://host2${topologyConnectorPath}
```

Role-specific global configuration

- Configuration parameters can be defined globally for roles (on any node)

```
# Role-specific global configuration
roleConfig:
- role: role1
  config:
    var1: v1
```

Tenant definitions

- You can specify a list of tenants used for the configuration generation
- Example: One vhost file for each tenant in the httpd configuration

```
tenants:

# Tenant with two tenant roles (can be used for filtering file multiply)
- tenant: tenant1
  roles:
  - website
  - application
  config:
    domain: mysite.de
    website.hostname: www.${domain}

# Tenant with one tenant role
- tenant: tenant2
  roles:
  - website
  config:
    domain: mysite.fr
    website.hostname: www.${domain}
```

Flexible config map definition

- For nested maps a short notation is supported by using a "." notation.
- Both examples express the same configuration

```
config:
  param1: value1
  group1:
    param11: 5
    param12: true
  list1:
  - listValue1
  - listValue2
```

```
config:
  param1: value1
  group1.param11: 5
  group1.param12: true
  list1:
  - listValue1
  - listValue2
```

Configuration parameter inheritance

- Configuration parameter maps are inherited to "deeper levels" within the YAML structure, and the maps are merged on each level.

- The configuration parameters on the "deeper levels" overwrite the parameters from the higher level - inheritance order:
    1. Global configuration parameters from role definition
    2. Configuration from role variant definition
    3. Global configuration from environment
    4. Node configuration from environment
    5. Global role configuration from environment
    6. Role configuration from node
    7. Variant configuration from node
    8. Configuration from multiply plugins, e.g. the tenant-specific configuration

- Special support for list parameters: If you insert the keyword **`_merge_`** as list item on either of the list values, they are merged and the special keyword entry is removed.

Default context properties

- A set of default context properties are defined automatically by CONGA and merged with the parameter maps. Examples:

Property	Description
version	Environment version
nodeRole	Current node role name
nodeRoleVariant	Current node role variant name
environment	Environment name
node	Current node name
tenant	Current tenant name. This is only set if the tenant multiple plugin is used.

- The full list can be found at
  http://devops.wcm.io/conga/yaml-definitions.html

Exercise

Execute exercise

PVTRAIN-148-02 Configure CONGA environments

- Create a new environment
- Define multiple nodes
- Change configuration parameters

CONGA Roles

Define roles and templates

CONGA Role

A CONGA role definition consists of

- List of variants supported by the role (optional)

- Directory where the template files are stored

- List of files to be generated

- Definition of configuration parameters with default values

A role is described using a YAML file.

Full documentation in JavaDocs:
http://devops.wcm.io/conga/generator/apidocs/io/wcm/devops/conga/model/role/Role.html
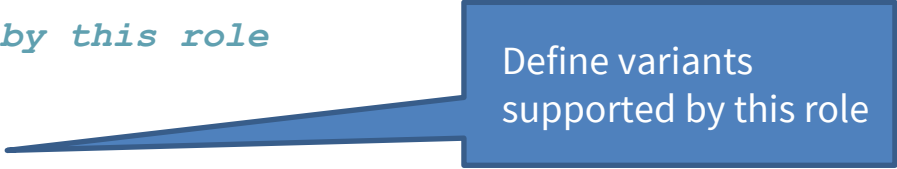
Define variants and template root path

- A role can define variants that are supported
- Switching between different variants in the environment may
  - activate/deactivate individual files the role
  - or affect conditions in the templates that generate the files
- The template path is always relative to the **`src/main/templates`** folder

```
# Variants supported by this role
variants:
- variant: services
- variant: importer


# Relative path to lookup the template files
templateDir: tomcat
```

Define variants supported by this role

Configuration parameters with default values

- Defines all configuration parameters (with default values) used by the role
- This acts also as "parameter documentation" for users of the role
  - Optional parameters should be documented as well (commented out)

```
# Defines configuration parameters and default values
config:

  tomcat:
    path: /path/to/tomcat

  jvm:
    path: /path/to/java
    heapspace:
      min: 512m
      max: 2048m
    permgenspace:
      max: 256m

  topologyConnectors:
    - http://localhost:8080/libs/sling/topology/connector

  # Optional - enable debug mode
  #debug: true
```

"Flexible config map definition" rules and support for variable placeholders apply here as well

39

File generation

- Mandatory parameters for each generated file:
  - `file`: Destination file name
  - `dir`: Relative path for destination directory
  - `template`: Handlebar template name

- Optional parameters:
  - `variants`: Generate the file only for a given list of role variants
  - `charset`: Define file encoding (default: UTF-8)
  - `condition`: Condition whether the file should be generated
    - The condition is a single variable placeholder
    - The condition is true if the resulting string of the variable is not empty and does not match "false".
  - `lineEndings`: Define line endings – unix (default), windows or macos

Plugins for file generation

- Optionally you can control which plugins should be applied:
  - **`validators`**: Validates the syntax of the generated file.
    - If not set, a plugin that accepts the file extension is chosen automatically.
  - **`validatorOptions`**: Options for the validator plugins
  - **`postProcessors`**: Post-process files (e.g. transform in different format)
  - **`postProcessorOptions`**: Options for post processor plugins
  - **`fileHeader`**: Adds a file header to the generated file
    - If not set, a plugin that accepts the file extension is chosen automatically.
  - **`escapingStrategy`**: Rules for escaping the inserted values
    - If not set, a plugin that accepts the file extension is chosen automatically.
  - **`multiply`**: Generate multiple files instead of one single file
  - **`multiplyOptions`**: Options for multiply plugins

Generate single file

- Generating single files

```
# Define a single file to be generated for all role variants
- file: setenv.sh
  dir: bin
  template: setenv.sh.hbs
  # Default charset is UTF-8 unless specified otherwise
  charset: ISO-8859-1

# Define a single file to be generated for role variant 'services'
- file: ROOT.xml
  variants:
  - services
  dir: conf/Catalina/localhost
  template: ROOT.xml.hbs
  # Allows to define special validators.
  # If missing the best-match validator is picked automatically.
  validators:
  - xml
```

Generate file only for this list of variants

Generate multiple files

- Generate multiple files with the sample template
- E.g. one file for each tenant (different multiply plugins may exist)

```
# Define a file to be generated per tenant
- file: "${tenant}_vhost.conf"
  dir: vhosts
  template: tenant_vhost.conf.hbs
  # Multiply file for each tenant that has the given roles
  multiply: tenant
  multiplyOptions:
    roles:
    - website
```

Download/Copy files

- As an alternative to generating the files it is possible to download and copy files into the target directory.
- They are not generated via handlebars, but may be post-processed as well.

```
# Copy file from classpath
- file: mysample.txt
  dir: download
  url: classpath:/sample.txt
  modelOptions:
    customOption1: value1
    customOption2: 123

# Download file from maven repository, use artifact filename.
# Derive version from maven project.
- url: mvn:x.y.myapp/x.y.myapp.complete-package//zip
  dir: packages
```

Sources for download/copy files

- The following URL prefixes are supported out of the box:
  - `file:` – Absolute filesystem path
  - `classpath:` – Classpath resource reference
  - `http://` or `https://` – External URL
  - `mvn:` – Maven Artifact coordinates
    - only supported when CONGA runs inside Maven)
    - Maven Coordinates Syntax 1 (Maven-style):
      `groupId:artifactId[:packaging][:classifier]:version`
    - Maven Coordinates Syntax 2 (SlingStart-style):
      `groupId/artifactId/version[/type][/classifier]`
    - classifier and type are optional.
    - If the version is empty in the role file it is resolved from the Maven project.
- If no prefix is specified the URL is interpreted as relative path in the local filesystem.

Handlebars quickstart

Template language for CONGA

handlebars

Handlebars provides the power necessary to let you build **semantic templates** effectively with no frustration.

Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates.

Handlebars templates look like regular HTML, with embedded handlebars expressions.

```html
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

A handlebars expression is a `{{`, some contents, followed by a `}}`

http://handlebarsjs.com/   http://jknack.github.io/handlebars.java/

About handlebars

- CONGA uses **handlebars** (Java) as template engine for file generation

- All handlebar language features can be used

- CONGA adds some additional expressions

- You can add your own expressions as well (via CONGA plugins)

Handlebars basics: Variable references

To insert a variable from configuration parameter maps with escaping (escaping strategy depending on file type):

`{{group1.param1}}`

To insert a variable without escaping (you have to take care of generating a valid file yourself):

`{{{group1.param1}}}`

Handlebars basics: Conditions

To conditionally generate a block:

```
{{#if group1.flag1}}
  condition met block...
{{/if}}
```

Optionally you can define an else block:

```
{{#if group1.flag1}}
  condition met block...
{{else}}
  condition not met block...
{{/if}}
```

Handlebars basics: For each loop

To loop about a list of values:

```
{{#each group1.list}}
  {{this.param1}}
{{/each}}
```

If you want to add a separator between each item but not after the last:

```
{{#each group1.list}}
  "{{this.param1}}"{{#unless @last}},{{/unless}}
{{/each}}
```

To insert the list index from the for each loop:

```
{{#each group1.list}}
  "prop{{@index}}": "{{this.param1}}",
{{/each}}
```

Handlebars basics: Whitespace handling

You can control whitespace handling around handlebar expressions by inserting ~ at the beginning or end of the handlebars expression. On the side of this expression all whitespaces are removed up to the next handlebars expression or non-white space content.

Example: Remove all whitespaces inside the expression:

```
{{#if group1.flag1 ~}}
  conditional block...
{{~/if}}
```

Example: Remove all whitespaces around the expression:

```
{{~#if group1.flag1}}
  conditional block...
{{/if ~}}
```

Handlebars basics: Partials and blocks

If you want to modularize your templates and reused a shared set of content or expressions in multiple templates you can use partials and blocks.

Example of a file with shared content/expressions using blocks:

```
{{#block "serverName"}}

  ServerName {{group1.serverName}}

{{/block}}


{{#block "documentRoot"}}

  DocumentRoot "{{group1.rootPath}}"

{{/block}}
```

Handlebars basics: Partials and blocks

You can include this file in another and overwrite parts from the shared file by overwriting single blocks with a partial:

```
... main template start


{{#partial "serverName"}}

  ServerName {{group1.otherServerName}}

  ServerAlias {{group1.aliasName}}
{{/partial}}


{{> role1/mypartialtemplate.conf.hbs}}


... main template end
```

Handlebars basics: Comments

To include a comment that is stripped from the generated file:

```
{{!-- my comment --}}
```

CONGA-specific handlebar expressions:

- **regexQuote** – To insert a variable expression and applying regex quoting

- **join** – To join a list of values with a separator character

- **ifEquals** – Conditional if statement with separate argument

- **defaultIfEmpty** – Inserts default value if expression not set

- **eachIf** – Conditional for each loop

- **eachIfEquals** – Conditional for each loop with separate argument.

The CONGA-specific expressions are documented here:
http://devops.wcm.io/conga/handlebars-quickstart.html

CONGA Extensibility model

Plugin Architecture

CONGA Extensibility model

- The CONGA architecture is very modular

- Most functionality provided by CONGA itself is implemented by plugins shipped with CONGA

- More functionality is provided by Plugins e.g. for Sling and AEM

- You can easily write your own CONGA plugins to add support for new file formats or other special features

Conga SPI

CONGA allows to provider custom plugins that are applied on generated files:

- **File Header Plugin**: Adds a file header to each generated file. *)

- **Validator Plugin**: Validate file syntax after generation. *)

- **Handlebars Escaping Strategy Plugin**: How to escape special characters in the generated file. *)

- **Post Processor Plugin**: Plugin that operates on a generated file, e.g. to convert it to a binary file.

*) These plugins detect files with certain extensions, and are executed automatically on them.

Conga SPI

Other plugins:

- **Multiply Plugin**: Generate multiple files with a single template.

- **Handlebars Helper Plugin**: Define your own handlebar expressions.

- **Node Model Export Plugin**: Export "model data" for IT automation tools.

- **URL File Plugin**: Define new sources to download/copy files from.

For a list of all built-in plugins see:
http://devops.wcm.io/conga/extensibility.html
http://devops.wcm.io/conga/plugins/sling/extensions.html
http://devops.wcm.io/conga/plugins/aem/extensions.html

Advanced Maven Topics

Use CONGA and Maven effectively

Configure the CONGA Maven plugin

You can configure:

- the paths to look up roles, templates and environments

- whether to generate configuration for all or only for a single environment

- whether to create a single ZIP file for all environments, or one for each

- whether to export model data (model.yaml) per node or not

Full documentation of the CONGA Maven plugin:
http://devops.wcm.io/conga/tooling/conga-maven-plugin/plugin-info.html

Combine multiple configuration definitions

- You are not limited to use only one single artifact which contains the CONGA configuration definitions (roles, templates)

- You can reference multiple of them in your environment POM and use and mix the roles as required
  - You will see examples of this in the next training "PVTRAIN-146 AEM Configuration with CONGA"

- You can also overlay templates files from referenced artifacts with modified versions from your own
  - In this case the dependency order in the Maven projects controls which file is loaded from the classpath if multiple exists with the same name
  - You should only overlay files this way if the file from the original role is designed for this, e.g. by using Handlebars partials and blocks

Deploy CONGA Maven artifacts

- Usually only the CONGA configuration definition artifacts are deployed to a central maven repository.

- The environments are kept in a source code management repository as well, but neither the environment definitions nor the generated configuration should be uploaded to a maven repository because they may contain sensitive data (e.g. passwords).

- The configuration definition is released and versioned together with the application. Thus it is possible to rollback to a previous version of the application together with the matching configuration definition, but still using the latest environment parameter values.

- The version of application (and configuration definition) that should be deployed is configured in the POM of the environment definition.

Exercise

Execute exercise

PVTRAIN-148-03 Define CONGA Roles and Templates

- Update roles and introduce variants
- Change templates
- Create new parameters
- Define tenants