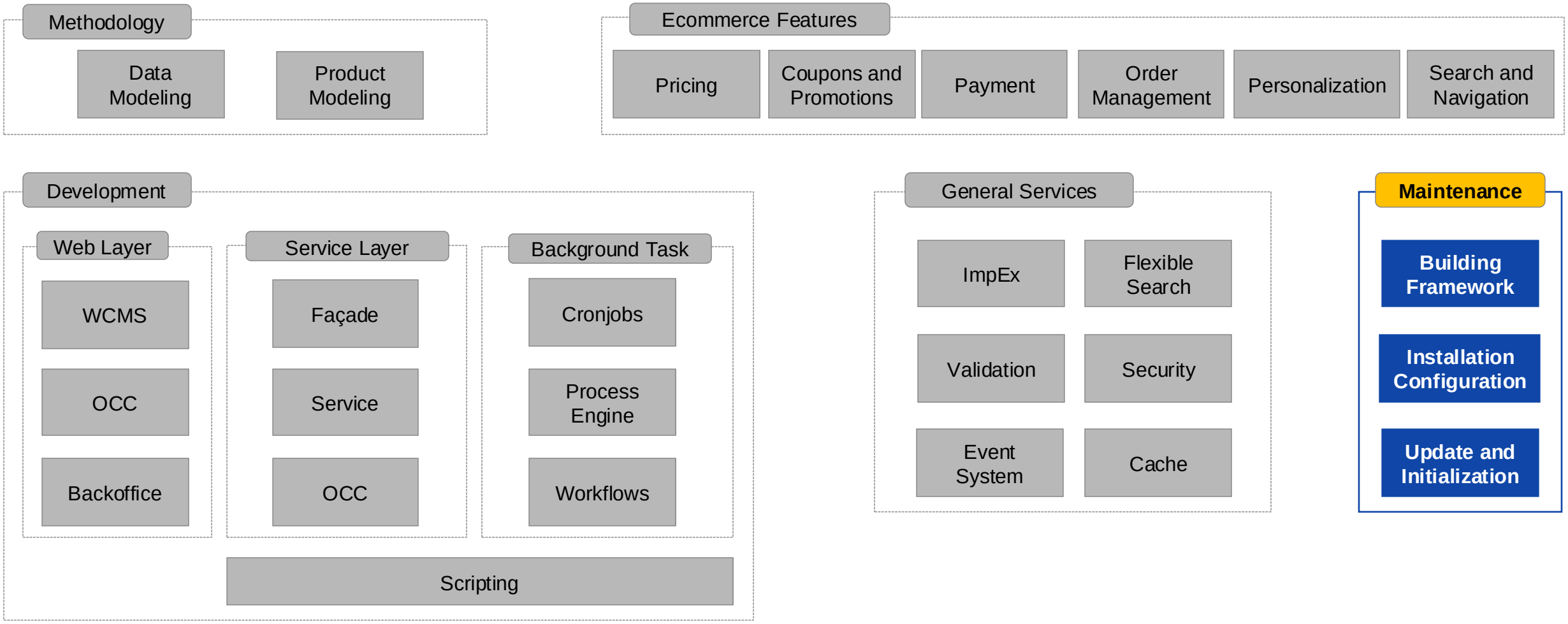
SAP Customer Experience

Installing SAP Commerce Cloud

What we will cover in this topic



We will learn about:

- Build Framework
- Extension Concept
- Basic Configuration
- SAP Commerce Cloud Server
- HAC, Initialization, and Update
- Recipes
- Spring in SAP Commerce Cloud
- Exercise How-to

The Context

SAP Commerce Cloud is a **highly flexible** and **modular** platform for delivering modern E-commerce experiences. Its rich and extensible library of functionality ensures the success of any commerce project.

Build Framework

Build Framework

- SAP Commerce Cloud has a build framework based on Apache Ant
  - Ant handles compilation and a number of automation tasks
    - An assortment of automated tasks (such as compilation) is called an Ant Target
  - Class executables compiled by the Eclipse IDE are not used by SAP Commerce Cloud

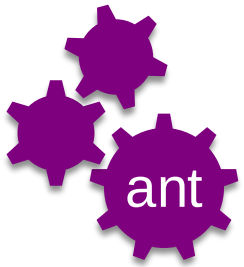
- Ant Targets are described in build files (`build.xml` by default) that contain script-like instructions about the tasks that should be automated

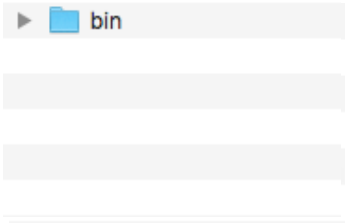
- In SAP Commerce Cloud, there is a build file in every extension, but we generally use the one in the platform extension to build the entire suite
  - Because of that, Ant is usually executed from the platform extension directory
  - It builds every extension listed in (or referenced by) `localextensions.xml`

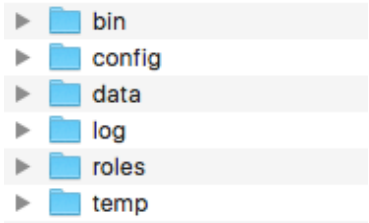
Building with ant

When you call `ant`, the build framework:

- generates and compiles **Model** classes (covered in the next module)
  - according to the definitions in the `*-items.xml` files
  - based on extensions dependency (hierarchy)
- updates configuration of the Commerce Server
- generates five new folders (only in the first run!)







Common Ant Targets in SAP Commerce Cloud

Ant Target	Description
all	Builds the application and configures the server *
clean	Deletes class files and autogenerated Java source files from platform and extensions
extgen	Generates an individual, standalone extension based on a template
initialize	Creates or resets type system, instance data, and localization definitions
updatesystem	Similar to initialize, but doesn't delete existing data
unittests	Execute all unit tests
-p	Shows a list of all Ant targets (many more than listed here)

* “all” is the default Ant Target. It will be executed if no target name is provided when invoking ant.

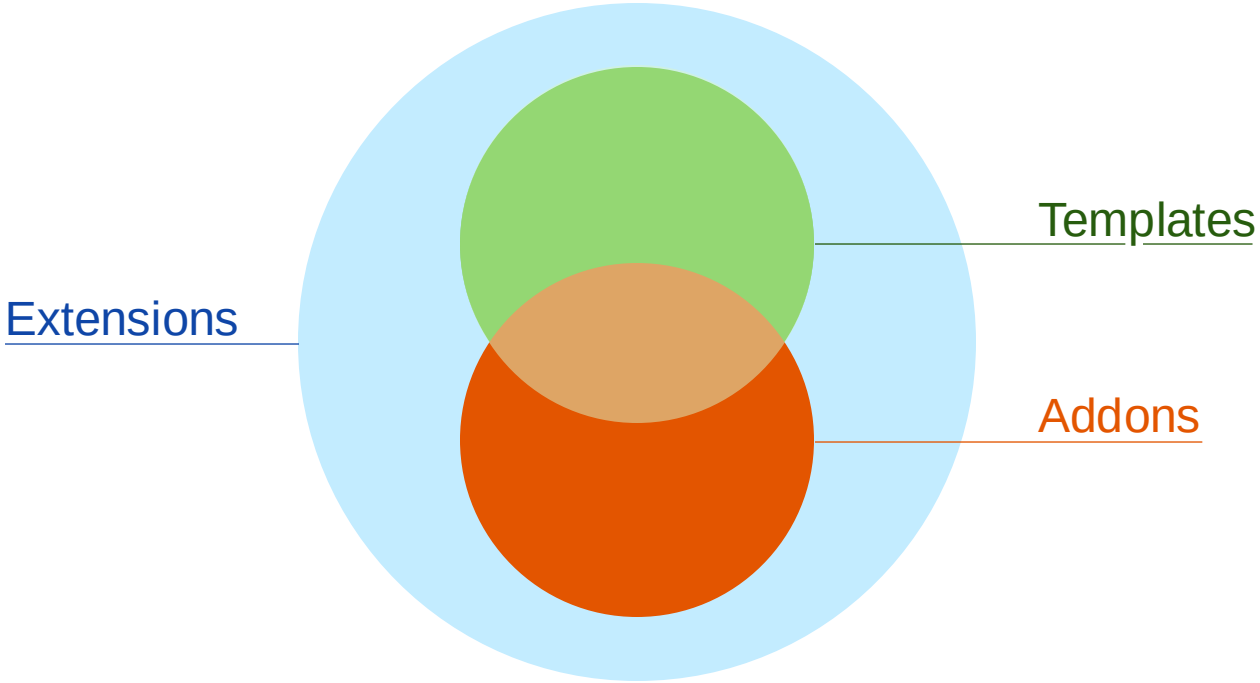
Extension Concept

Where to Begin with SAP Commerce Cloud

- Starting any new project involves creating one or more extensions.
  - Each extension will contribute a part of the greater whole, including modifying or adding to the data models, business logic, backoffice configuration, etc.
  - Your (project) code is separated from SAP Commerce (framework) code, making your code easier to reuse and to migrate to future versions.
  - SAP Commerce Cloud bundles a proprietary code generator.
- In general, a project can include:
  - A core extension that defines and implements business domain logic API (such as services).
  - A facade extension to orchestrate across business domains.
  - A commerce-driven RESTful web services extension (OCC extension) with REST Controller classes, other related classes and resources, etc.
  - A testing extension, containing the test cases and data.

Extension, Addon, and Template

- Extension: Packaging mechanism for SAP Commerce Cloud features
- Addon: Special extension to extend storefront/OCC functionality (deprecated since 2205)
- Template: Predefined extension/addon duplicated as a starting point when creating a new extension/addon



Creating a New Extension

- Most extensions use a template, such as

yempty	Empty extension with minimal configuration
ybackoffice	Structure for a Custom Backoffice Extension
yocc	Serves as a starting point for creating a new extensions for Commerce Web Services.

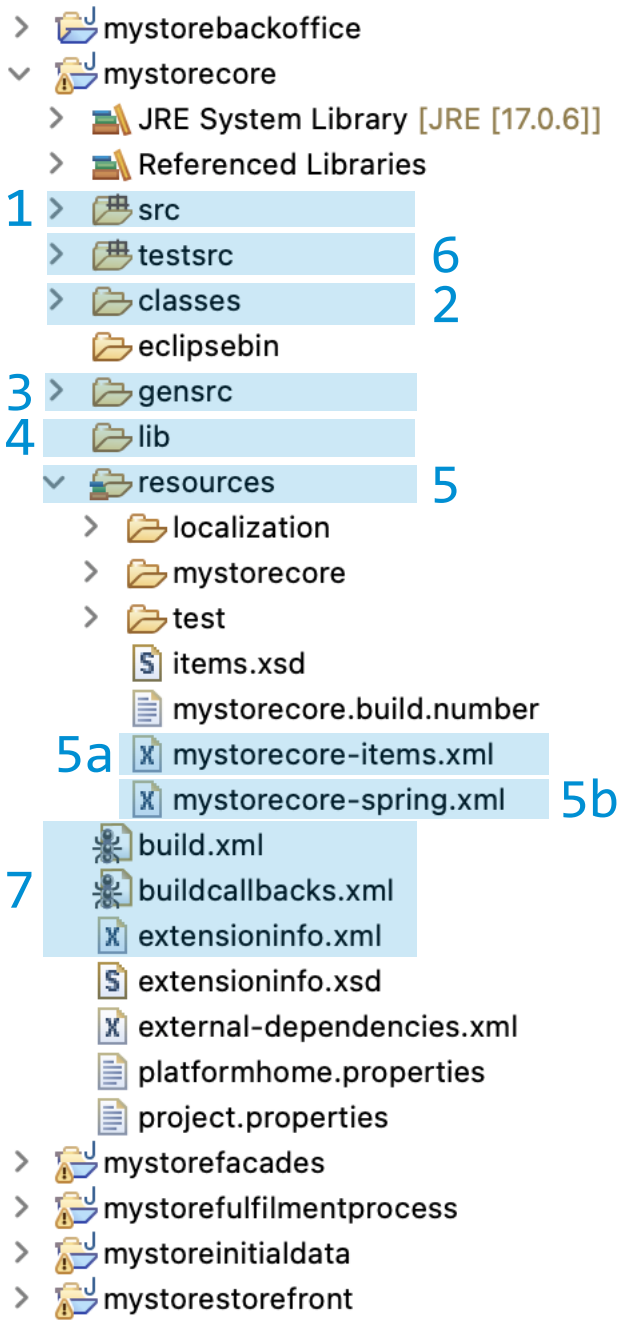
- To create a single extension, invoke **`ant extgen`**
- Reference any required extensions in **extensioninfo.xml**
  - See the next section for details
- Add your extension to **config/localextensions.xml**
- Invoke **`ant [clean] all`**

Structure of a custom Extension

1. Business Logic
2. Compiled sources
3. Generated Java files
4. External library files
5. Resources folder for external data, type definitions & localization

   5a. Model definition

   5b. Spring configuration
6. JUnit test classes
7. Files for Eclipse, Apache Ant, and extension-specific configuration



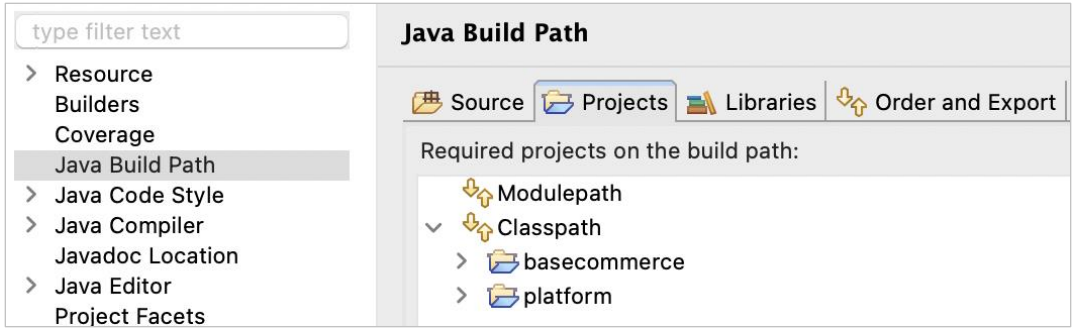
Basic Configuration

Configuration Files

- The list of extensions included in the ant build is defined in
  - `config/localextensions.xml`
- Each extension configures its dependencies in
  - *`extensionName`*`/extensioninfo.xml`

```
<!-- add all dependent extensions -->
<requires-extension name="basecommerce"/>
```

- Must configure all the same dependencies in Eclipse as well



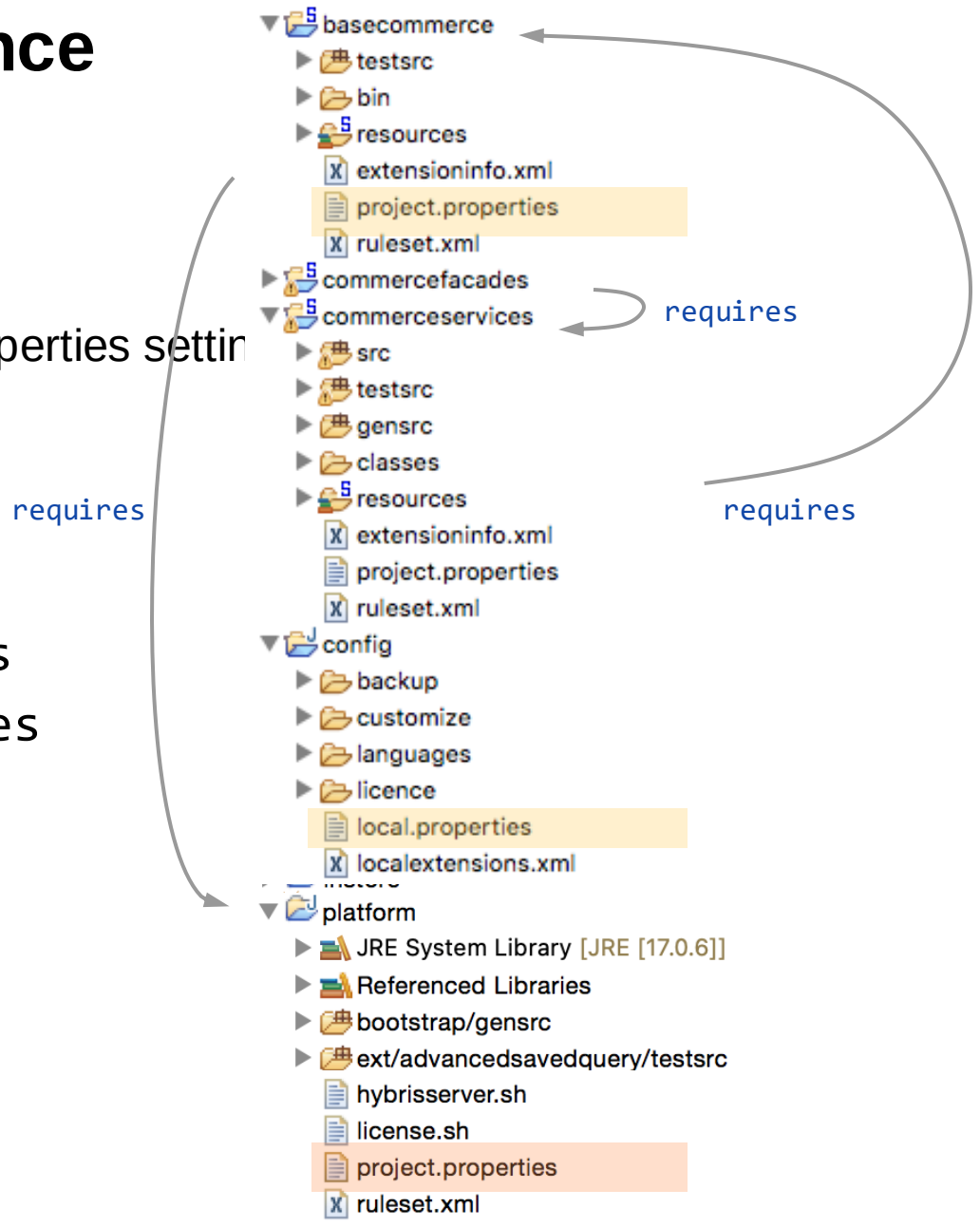
- Put extension-relevant, default configuration in
  - *`extensionName`*`/project.properties`
- Override default configuration in
  - `config/local.properties`
  - Server configuration such as database URL and credentials
- To use a different configuration, i.e., for a test server,
  - Run `ant -Duseconfig=testserver` will use `localtestserver.properties` as override

project.properties Precedence

- All extensions implicitly depend on platform
- Dependency chain determines precedence
- local.properties settings override all project.properties settin
- cyclic dependencies are rejected by ant

```
config/local.properties
...
commercefacades/project.properties
commerceservices/project.properties
basecommerce/project.properties
platform/project.properties
```

Each project.properties overrides preceding ones in build/load order to determine system configuration. Lastly, local.properties overrides.



Configuration changes

- SUMMARY… location of config files:
  - Files `project.properties` and `extensioninfo.xml` are in each extension's base folder
  - Files `local.properties` and `localextensions.xml` are in the `config` directory

- If you modify the `local.properties` file:
  - Restart `hybrisserver` (no build required)
  - SPECIAL CASE: if you change any Tomcat configuration settings (such as tomcat.http.port), invoke the `server` ant target (copies settings to Tomcat installation)
  - If seriously in doubt, call `ant all` (includes ant targets: `build` and `server`)

- It's possible to check and/or change the properties at runtime in the HAC
  - However, after a server restart, values are reset

localextensions.xml

- List of extensions used by build framework

- Extensions can be listed by name instead of location path

- Dependencies are resolved automatically using each `extensioninfo.xml` and the `path` parameter. Build will find dependent extensions which are not explicitly listed
  - A complete `localextensions-generated.xml` can be generated using the **extensionsxml** ant target

Load Required Extensions

- Define extensions explicitly

```
<extensions>
   <extension dir="${HYBRIS_BIN_DIR}/modules/base-commerce/basecommerce"/>
   ...
```

- Or by using a lookup folder (and its nested subfolders), defined with the **<path>** tag
  - Allows extensions to be loaded by name rather than path
  - Allows lazy loading — SAP Commerce searches path directories for any extension referenced by another extension, and pulls it into the current configuration

```
<extensions>
   <path dir="${HYBRIS_BIN_DIR}"/>
   <extension name="commerceservices"/>
   ...
```

```
requires  basecommerce
          cms2
          customerreview
          payment
          ...
```

Loading All Extensions in a Folder

- You may autoload entire extension directories with the `path` tag, and limit lookup to specific directories.
  - OOTB, the **localextensions.xml** file specifies the path `hybris/bin`
  - Be careful not to load more extensions than needed

```
<extensions>
   <path autoload="true"
     dir="${HYBRIS_BIN_DIR}/modules/backoffice-framework" depth="3"/>
    <path dir="${HYBRIS_BIN_DIR}"/>
   <extension name="basecommerce"/>
   <extension name="yoyodynecore"/>
   <extension dir="${HYBRIS_BIN_DIR}/ext-content/cms2"/>
</extensions>
```

Summary of Loaded Extensions in Console Log

- The console log will list the extensions loaded

```
INFO  [localhost-startStop-3] [hybrisserver] ******************************************************************
INFO  [localhost-startStop-3] [hybrisserver]
INFO  [localhost-startStop-3] [hybrisserver] ------------------------------------------------------------------
INFO  [localhost-startStop-3] [hybrisserver] --- Extensions in dependency order ( options:
INFO  [localhost-startStop-3] [hybrisserver] ---  @deprecated: is deprecated, p: platform extension,*: auto-required
INFO  [localhost-startStop-3] [hybrisserver] ---  ?: lazy-loaded, i: got items.xml, b: got beans.xml, c: got core module
INFO  [localhost-startStop-3] [hybrisserver] ---  w: got web module )
INFO  [localhost-startStop-3] [hybrisserver] ------------------------------------------------------------------
INFO  [localhost-startStop-3] [hybrisserver] core 2211.0 [p*cib]
INFO  [localhost-startStop-3] [hybrisserver] commons 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] deliveryzone 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] maintenanceweb 2211.0 [p*w]
INFO  [localhost-startStop-3] [hybrisserver] mediaweb 2211.0 [p*cw]
INFO  [localhost-startStop-3] [hybrisserver] paymentstandard 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] scripting 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] processing->(scripting,commons) 2211.0 [p*cibw]
INFO  [localhost-startStop-3] [hybrisserver] impex->processing 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] testweb 2211.0 [p*w]
INFO  [localhost-startStop-3] [hybrisserver] validation->impex 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] catalog->validation 2211.0 [p*cib]
INFO  [localhost-startStop-3] [hybrisserver] advancedsavedquery->catalog 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] europe1->catalog 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] platformservices->(paymentstandard,deliveryzone,europe1) 2211.0 [p*cb]
INFO  [localhost-startStop-3] [hybrisserver] hac->platformservices 2211.0 [p*cw]
INFO  [localhost-startStop-3] [hybrisserver] oauth2->platformservices 2211.0 [p*cibw]
INFO  [localhost-startStop-3] [hybrisserver] workflow->(catalog,platformservices) 2211.0 [p*ci]
INFO  [localhost-startStop-3] [hybrisserver] comments->workflow 2211.0 [p*ci]
```

SAP Commerce Server

What is the SAP Commerce Server?

- Optimized and pre-configured Apache Tomcat server
- Production-ready quality and best suited to run all applications of SAP Commerce Cloud
- Independent of the operating system
- Easy installation
- Contains a wrapper that automatically restarts the Apache Tomcat Java Virtual Machine if the Apache Tomcat hangs or stops.

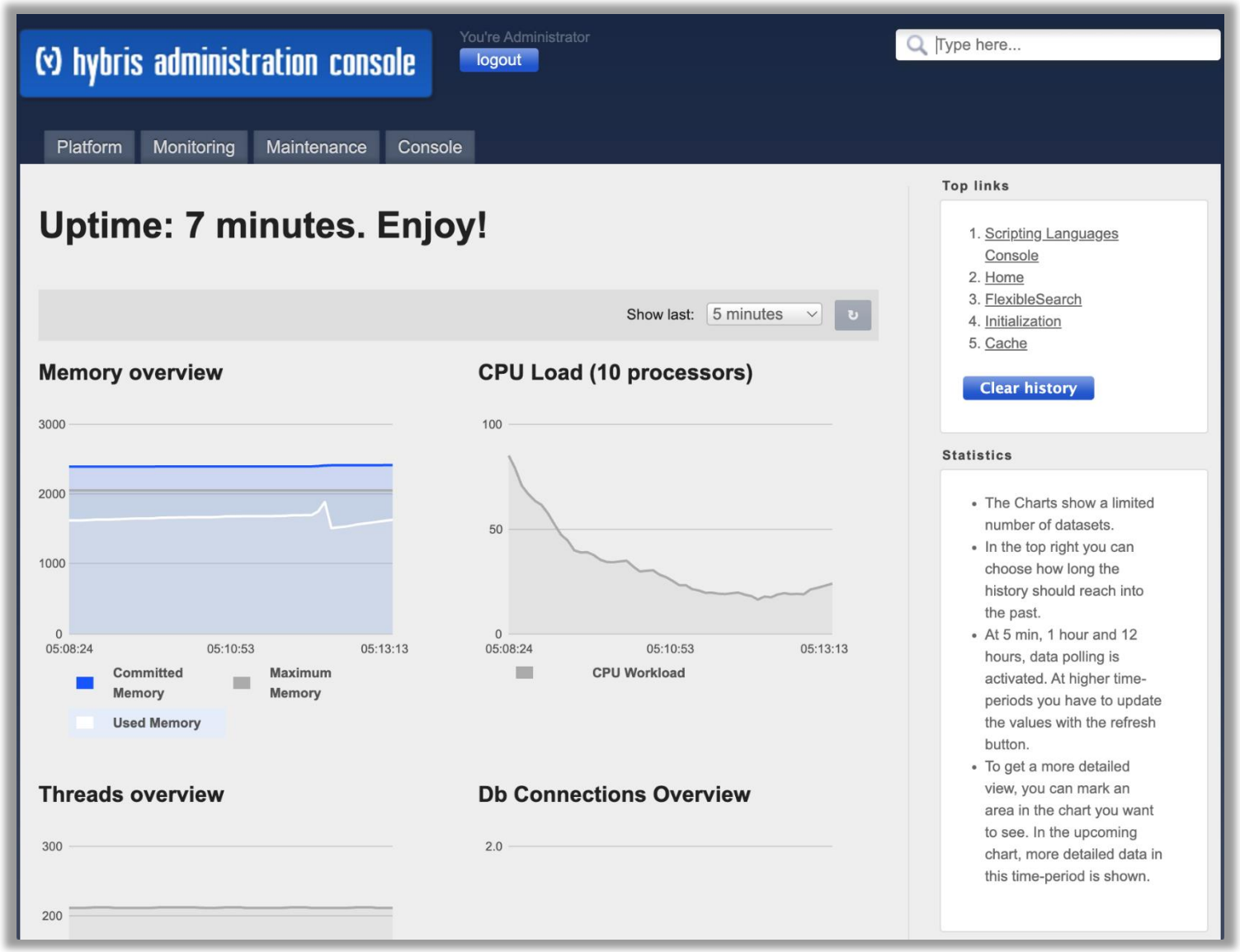
Apache Tomcat

HAC, Initialization, and Update

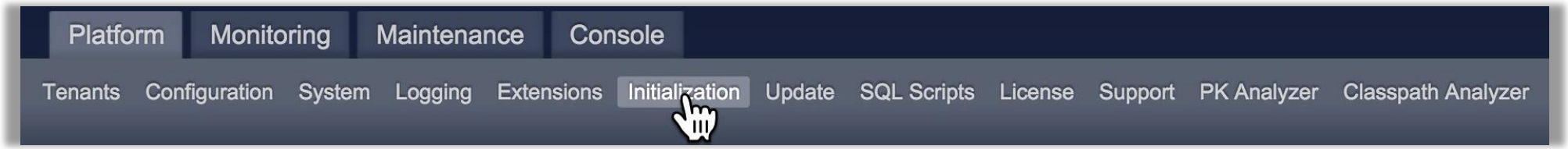
SAP Commerce Cloud Administration Console (aka. HAC)

- Administration
- Monitoring
- Configuration
- Default URL for HAC (can be overridden):
  - http://localhost:9001/
  - https://localhost:9002/
- For SAP Commerce Cloud:
  - Directly configurable,

  e.g. to <Endpoint URL>/hac



Initialize or Update the System



System Initialization:

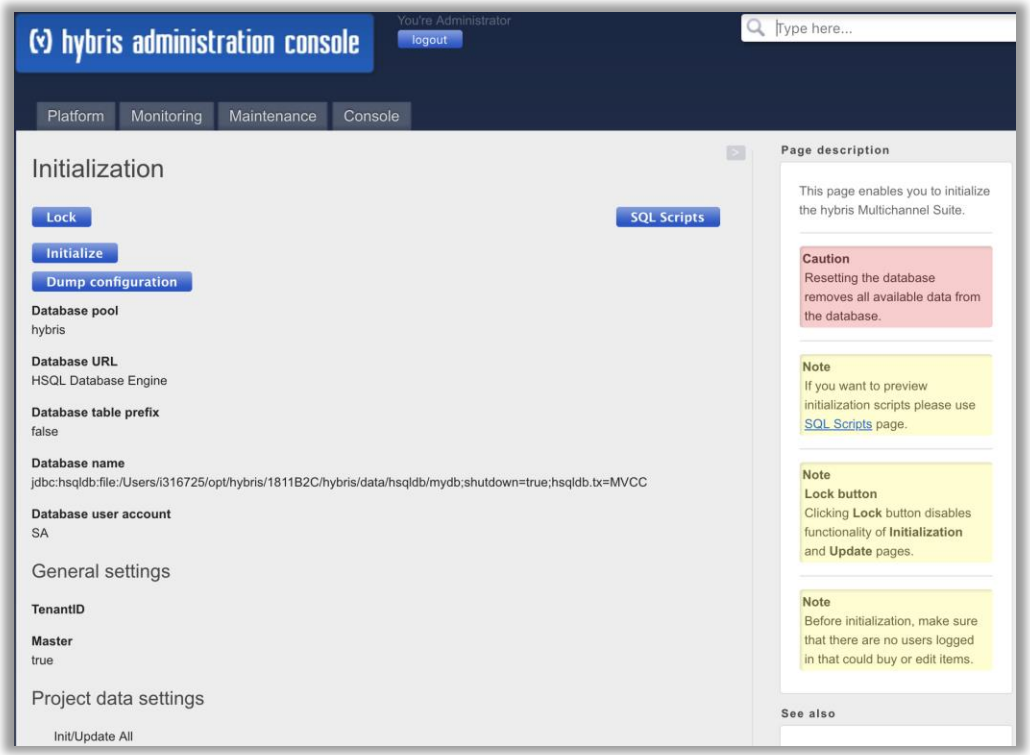
- Entire type system is created from scratch.
- ALL database tables defined in *-items.xml are dropped.
- Data model is created from scratch as defined in the `items.xml` files.
- New tables with initial dataset are created
- Existing data model definitions will be lost!
- Make sure to lock initialization after the first production deployment!

System Update:

- Existing tables are updated to match changes in the domain model.
- No loss of data!
- Two major aspects:
  - Adding newly defined types to the type system definition in the database
  - Modifying type system definition in the database to match the definition in the domain model

Initialization and Update

1. HAC ➡ Platform ➡ Initialize | Update

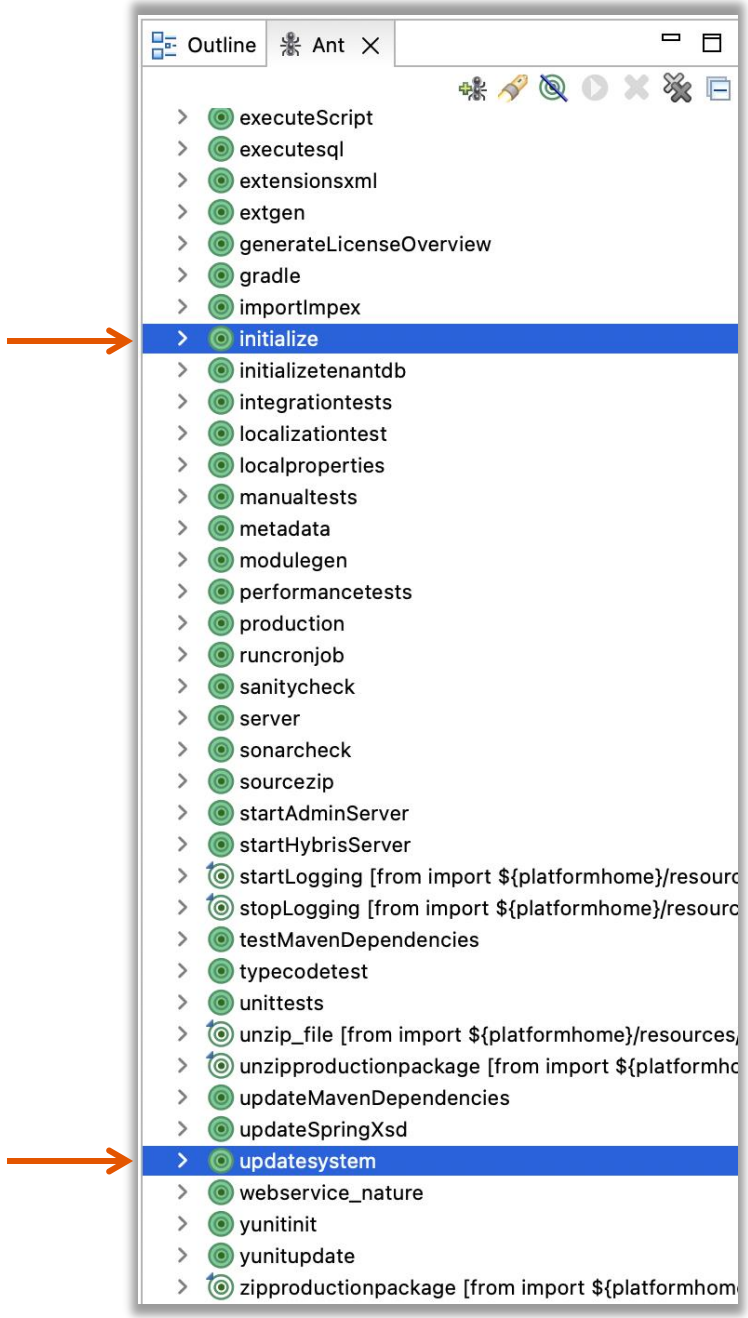


3. Command line

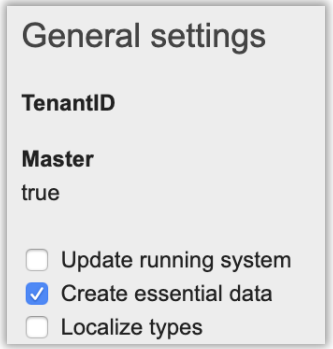
```
/hybris/bin/platform $ ant initialize -Dtenant=master
```

```
/hybris/bin/platform $ ant updatesystem -Dtenant=master
```

2. Ant view in eclipse + tenant



Essential Data vs. Project Data

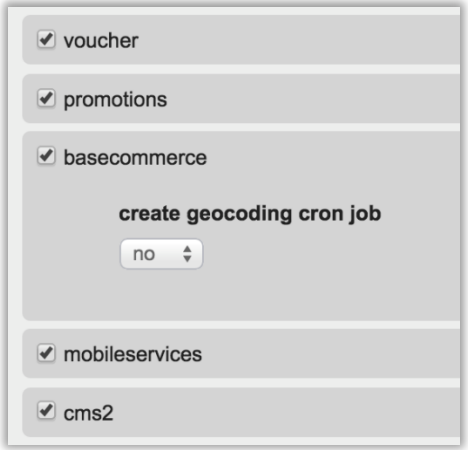


Essential Data

- Necessary during initialization:
  Creates the Default catalog, restrictions, and basic CronJobs, for example.

Project Data:

- Extension-specific project data



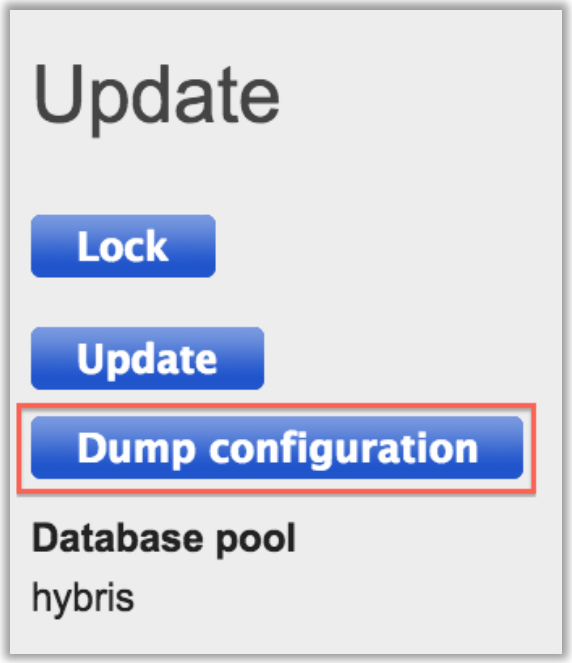
How to include:

- Convention over Configuration `essentialdata*.impex, projectdata*.impex`
- Hook Service Layer code into Commerce initialization, update and patching life-cycle events using the **@SystemSetup** annotation

Configurable update from console

Optionally, the HAC is able to provide the configuration for a system update from the console:

- HAC ➡ Platform ➡ Update
- Choose your update settings
- Click on Dump configuration
- Copy configuration settings from the screen
- Put it in a JSON file
- To execute the update, run the following command:




```
ant updatesystem -Dtenant=<my tenant> -DconfigFile=<path>/<filename>.json
```

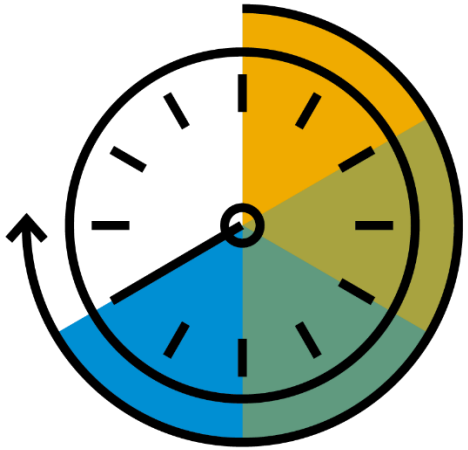
Demo


R100 BIKE
849.40
R300 Bike
373.69
R200 Bike
318.97
R200 BIKe 4s
596.34
M500 BIKE
209.16
eBike E180
75.44
M550 BIKE
74.40
Y200 Bike
62.31
eBike E181
54.06
C900 BIKE
42.47
C950 BIKE
29.90
M525 BIKE
135.61
Y120 Bike
28.38
Horns
19.07
Y240 Bike
43.84
in Thousand USD, %
2,987.26 (+30.59%)
Product Revenue Won Current
in Thousand USD
444.48
Revenue New Products
Lost due to price
348
Won due to product
Won due to price
Won due to service
1,657
Top Customers
Dimension
Industry ID
Territory
Sales Unit
Country
Competitor
Measure
in Thousand USD
NA
EMEA
APJ

More information

Initializing and Updating SAP Commerce Cloud:

- Initializing and Updating SAP Commerce Cloud on //help.sap.com



Recipes

SAP Commerce Installer

- Gradle-based project written in Groovy
- Simplifies installing SAP Commerce Cloud
- Automated script that takes care of
  - Directory creation
  - Moving files
  - Updating configuration and properties files
  - System initialization

Recipes

- Recipes are used to install a flavor of SAP Commerce Cloud (with selected features/properties/extensions)
- A recipe contains the required installation information, such as:
  - Three **mandatory** tasks (setup, initialize, start)
  - Calls to required plugins (e.g. installer-platform-plugin.jar)
  - Local properties
  - Extensions
  - Database configuration
  - Web archives
  - Server information (e.g. Apache Tomcat)

Recipes creation

- In order to create a recipe, you need to:
  - Create a folder for the recipe under `installer/recipes/`
  - The name of your recipe must be the name of your folder.
  - Using Groovy, create the `build.gradle` file containing your Installer recipe
  - Create a `README.txt` describing what your recipe does and providing the commands required to setup, initialize and start your system

More information about how to create custom installer recipes:

- Creating Installer Recipes on //help.sap.com

Install, Initialize, Start using Recipes

- To install SAP Commerce Cloud using recipes (setup, the default action, is optional):

  `install.bat -r <recipe_name> [setup]` (Windows)

  `./install.sh -r <recipe_name> [setup]` (OSX/Linux)

- To initialize SAP Commerce Cloud using recipes:

  `install.bat -r <recipe_name> initialize` (Windows)

  `./install.sh -r <recipe_name> initialize` (OSX/Linux)

- To start SAP Commerce Cloud using recipes:

  `install.bat -r <recipe_name> start` (Windows)

  `./install.sh -r <recipe_name> start` (OSX/Linux)

Example Recipes

Some predefined recipes:

`cx` (main installer recipe for SAP Commerce Cloud including apparel, electronics and powertools)

`cx_old_occ` (Includes all the same modules as the cx recipe, but uses the AddOn version of OCC instead of the OCC Extensions.)

Installer Recipes on //help.sap.com

or just have a look at `installer/recipes/*/readme.txt`

Spring in SAP Commerce

What is Spring?

The Spring Framework is a lightweight open source application framework for the Java platform provided and maintained by SpringSource.

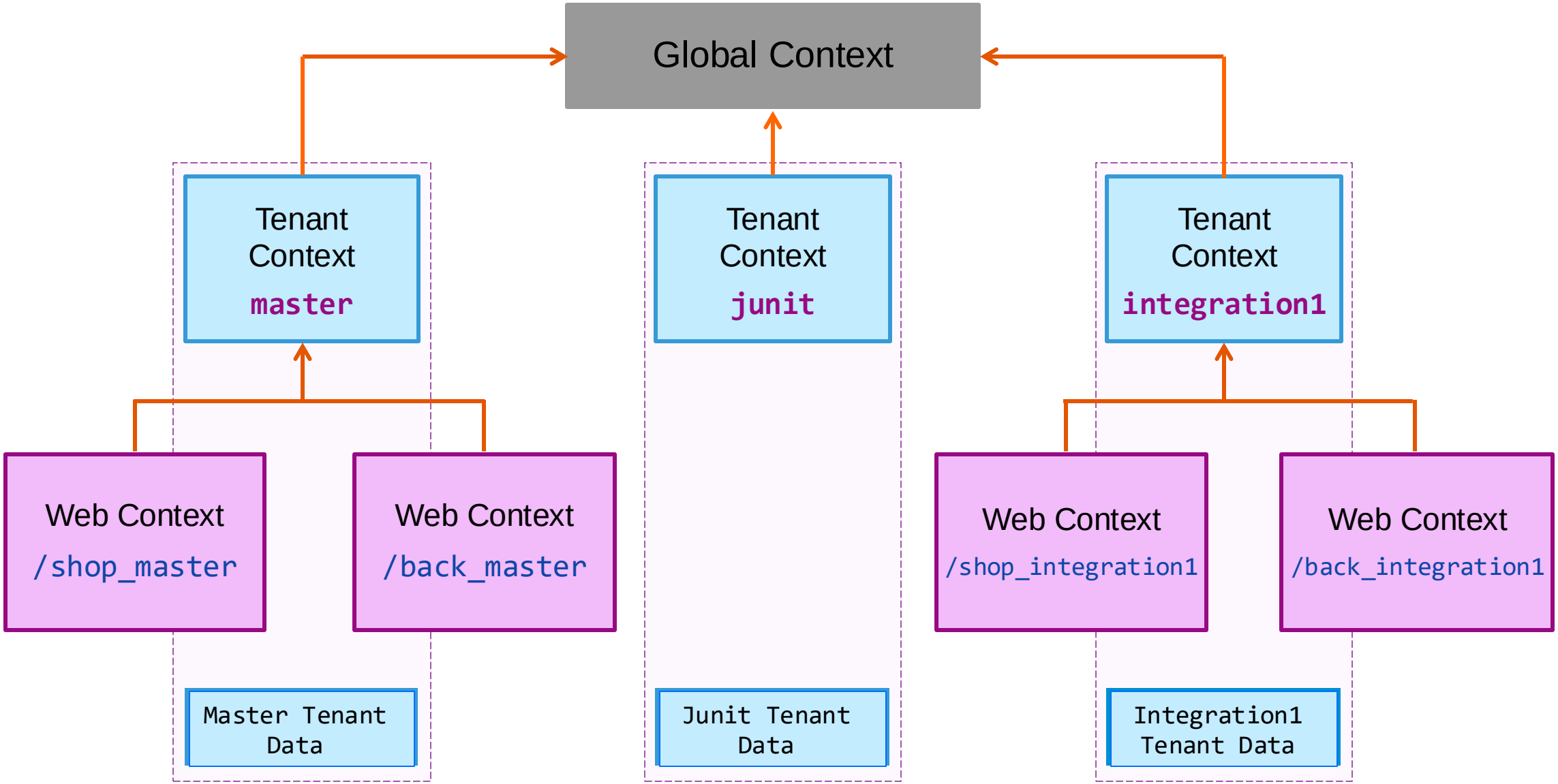
- Provides many components, not all of them used in SAP Commerce Cloud.
- The most important ones that are used:
  - **Dependency Injection** (also known as “Inversion of control”), used heavily, provides better decoupling and improves testability
  - **Aspect-Oriented Programming**, not used by default, but usable for extending stuff which isn’t customizable by default or implementing cross-cutting-concerns
  - **Spring MVC**, request based framework used in Sap Commerce Cloud web layer
  - **Spring Security**, used for authentication and basic authorization

Spring Configuration Review

- Configuring a bean in Spring
  - specify parent bean to inherit its configuration
  - property value can be literal or reference to another bean
  - lists and maps may be merged with definition in other extensions

- Quick Syntax review in our Spring Essentials for SAP Commerce
  - See **Dependency Injection**
  - See **Bean creation and configuration**
  - See **Accessing spring beans from another ones**
  - See **Scopes for beans**

Spring contexts

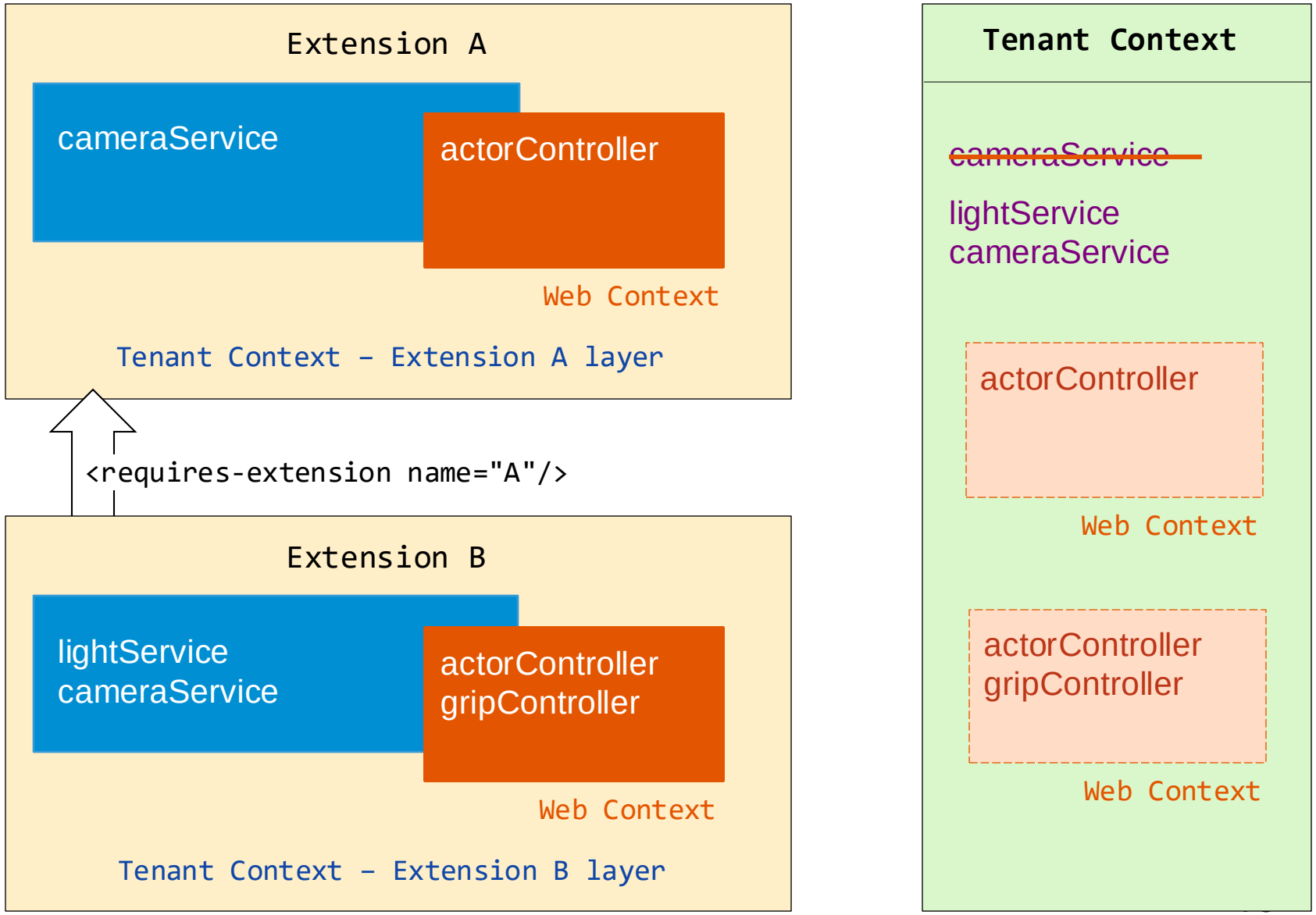


Spring configuration of extension

There are 3 types of xml files for your bean definitions in your extension

- `resources/global-{ext-name}-spring.xml`
  - Beans are shared among all extensions
- `resources/{ext-name}-spring.xml`
  - Beans are shared among all extensions
  - Beans will have as many instances as there are tenants.
- `Web context resources, e.g. web/webroot/WEB-INF/*-web-spring.xml`
  - Beans are available only inside the web context of the extension which defines them

Spring Configuration In Extensions



Using Alias to avoid Overwriting Services

Extension A

```
<alias alias="cameraService"
       name="defaultCameraService"/>
<bean name="defaultCameraService"
      class="35mmFilmCamera"/>
```

Tenant Context – Extension A layer

```
<requires-extension name="A"/>
```

Extension B

```
<alias alias="cameraService"
       name="newCameraService"/>
<bean name="newCameraService"
      class="IMAXCamera"/>
```

Tenant Context – Extension B layer

Tenant Context

~~cameraService~~

defaultCameraService

cameraService

newCameraService

More Information About Spring

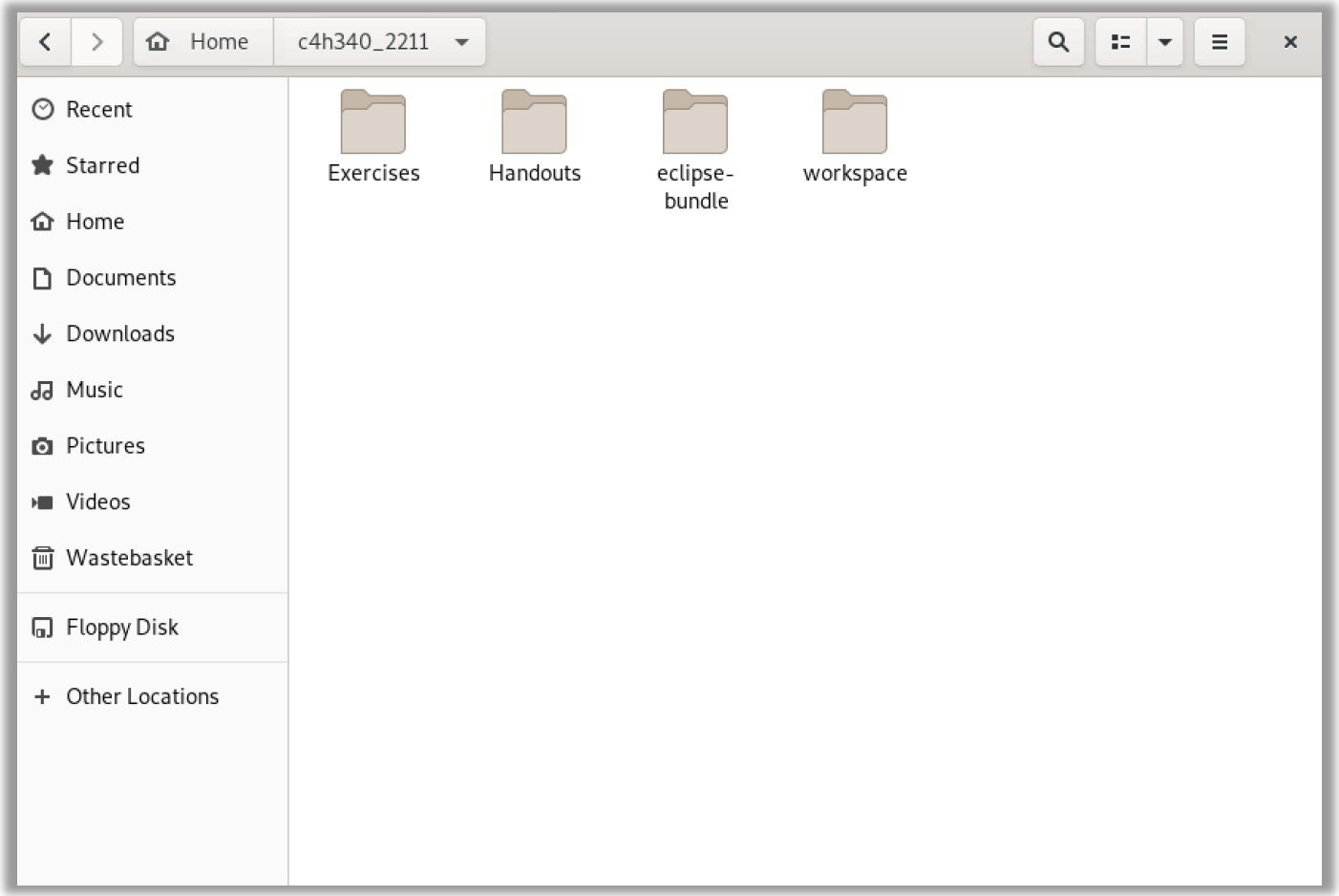
Take a look at the **Spring Essentials** handout we have included with your class handouts in the following directory:

```
/Optional-Reading
```

Exercise How-to

General Information

- Training material: Eclipse + workspace + Handouts + Exercises
- Workspace: SAP Commerce Cloud original + TrainingLabTools
- On SAP VMs, your system is configured to use Java 17 JDK (64 bits)

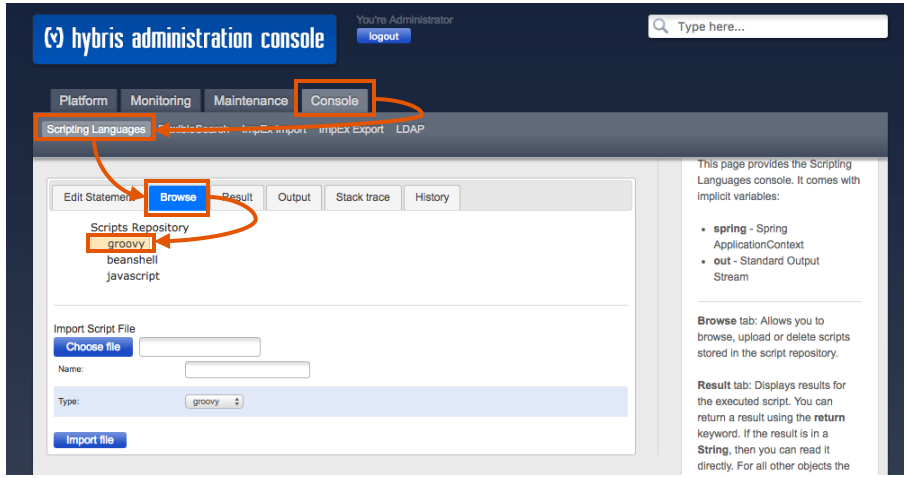


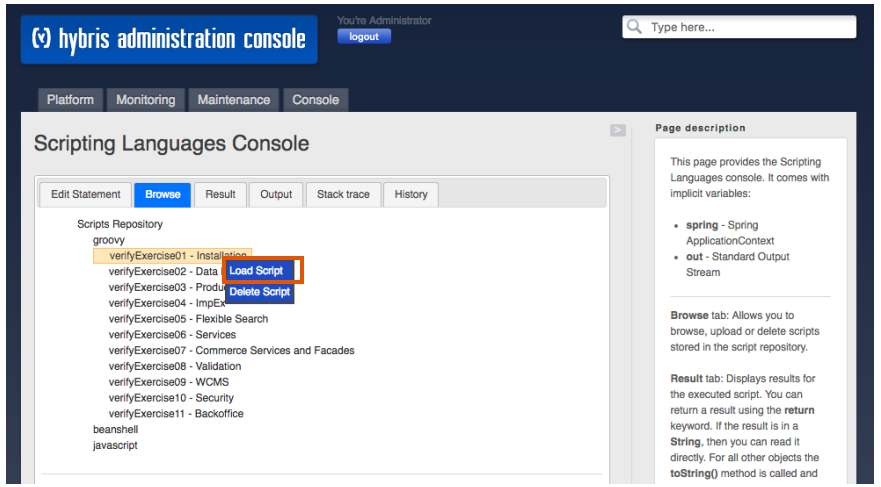
Training Lab Tools

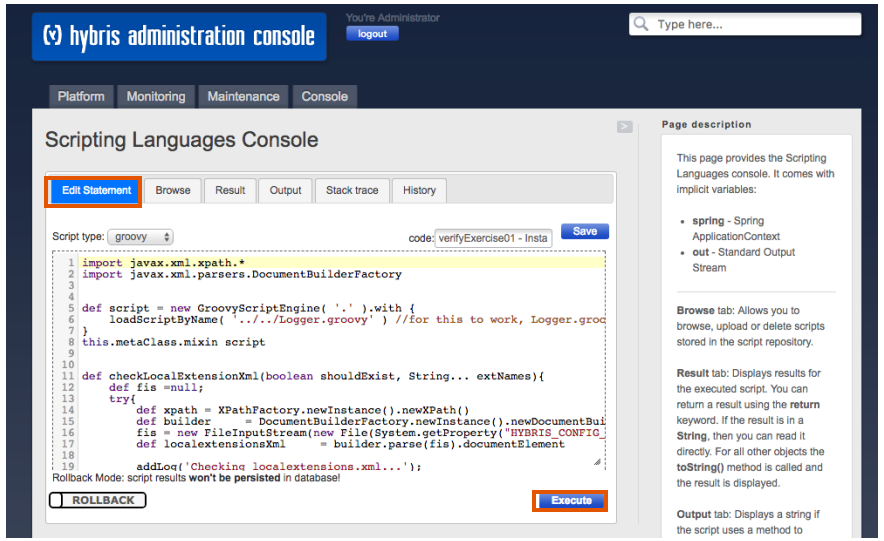
- A special folder provided by SAP Commerce Education
- Provides automated assistance to set up exercises, and a way to install a working solution
- Many exercises have an ant file with a **setup** or **prepare** target and/or a **solution** target
  - The exercise instructions will tell you when to invoke one of these targets
  - Remember to stop the server before running an exercise ant target
- Please DO NOT modify any code/files in the TrainingLabTools folder
  - Modify the code in your custom extension or config folder when doing exercises

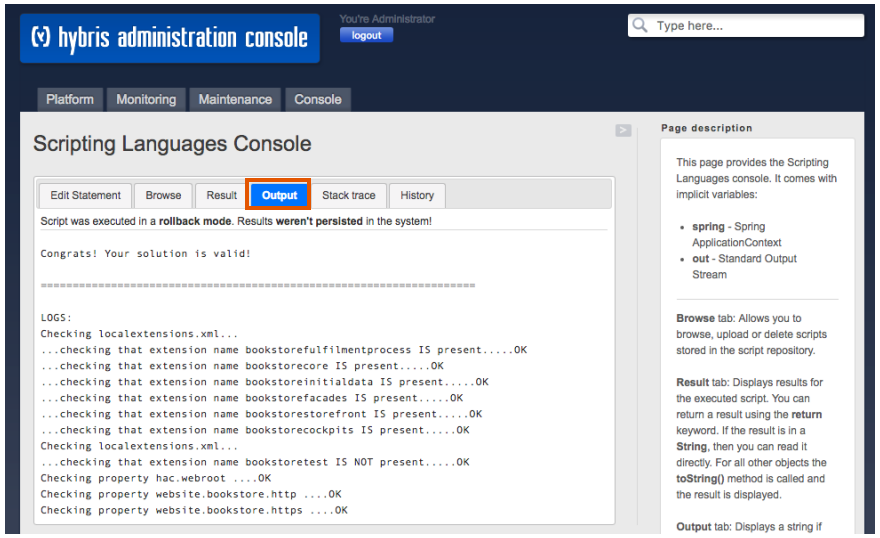
Verification of Your Solution

- Verification scripts are put into SAP Commerce Cloud after the Class Setup Exercise.









Key Points

1. SAP Commerce Cloud has a build framework based on **Apache Ant** - it builds every extension listed in `config/localextensions.xml`
   - use **`ant extgen`** to create a custom extension based on an extension Template
2. Configure dependencies in `extensionName/extensioninfo.xml`
3. The **SAP Commerce Server**, an optimized and pre-configured Apache Tomcat server, is OS-independent, production-ready and easy to install
4. Use the **SAP Commerce Cloud Administration Console** for administration purposes like initializing or updating tenants – this functionality is also (partially) available as ant targets from the console
5. The **SAP Commerce Installer** simplifies the installation of SAP Commerce Cloud with recipes (flavors): just run 3 mandatory tasks (setup, initialize, start)
6. The **Spring** features used most frequently: Dependency Injection, MVC, and Security. We also use three Spring contexts: Global, Tenant and Web

Class Setup Exercise

Thank you.