

Table of Contents

devon4ng	1
Motivation	1
Gotchas	1

devon4ng

This guide describes an application architecture for web client development with [Angular](#).

Motivation

The main challenge we encounter in our projects is to bring junior developers into client development. There are a lot of different frameworks and architectures in the market. The idea is to define an architecture which is a compromise between, on the one hand, leveraging the best practices and latest trends like reactive style development, on the other hand, providing a short onboarding time while still using an architecture that helps us scale and be productive at the same time. Also, the architecture must be compatible with the market. Guides, practices and naming found in the web should still be valid (e.g. a stackoverflow article for a given problem)

Gotchas

What we decided to go for is an architecture that leverages the benefits of reactive frontend architecture while remaining free of a dependency to a concrete framework like [ngrx](#) or [angular-redux](#). The main driver is, to make it easy to adapt the architecture and get going very fast. So, it is not necessary to learn about reducers, actions and middleware (e.g. action-creators, thunks, effects, etc.) and still have the benefits of reactive style programming.

That being said, we provide a [tutorial for using NgRx as a state management framework](#).

□ | *./images/devonfw.png*

devonfw shop floor dev-SNAPSHOT

The devonfw community

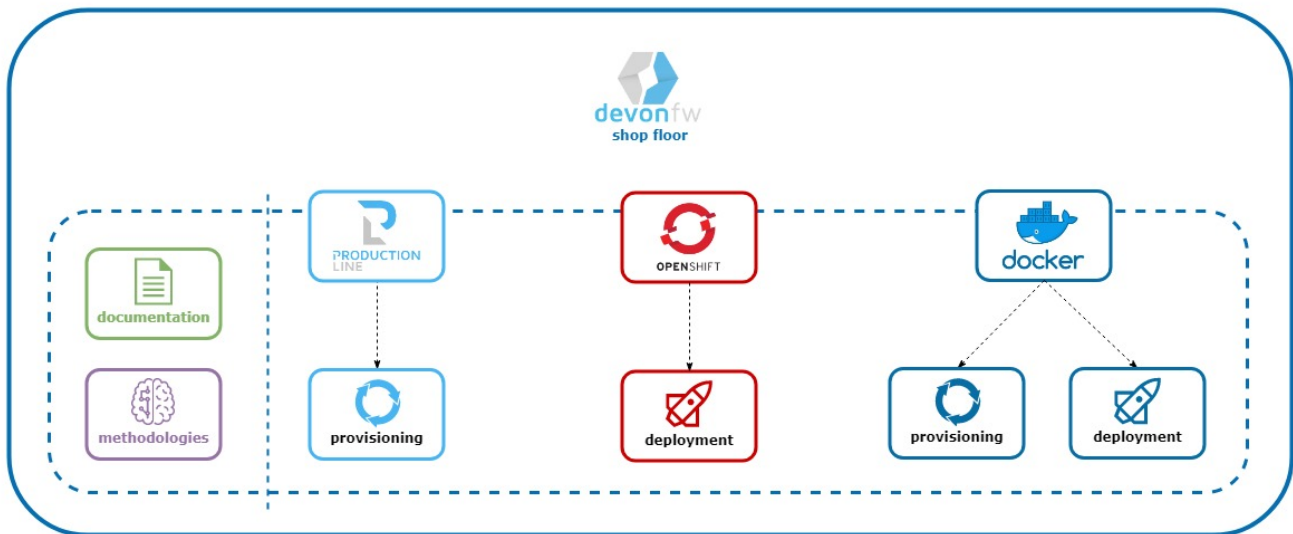
Version , 2019-07-03_11.44.56

Table of Contents

What is devonfw shop floor?	1
1. How to use it	1
1.1. Prerequisites - Provisioning environment	2
1.2. Step 1 - Configuration and services integration	2
1.3. Step 2 - Create the project	2
1.3.1. Create and integrate git repository	2
1.3.2. start new devonfw project	2
1.3.3. cicd configuration	2
1.4. Step 3 - Deployment	3
2. Provisioning environments	3
2.1. Production Line provisioning environment	3
2.1.1. How to obtain your Production Line	3
2.2. dsf4docker provisioning environment	3
2.2.1. Architecture overview	3
2.2.2. Prerequisite	4
2.2.3. How to use it	4
2.2.4. A little history	5
3. Configuration and services integration	5
3.1. Nexus Configuration	5
3.1.1. Prerequisites	5
3.1.2. Jenkins integration	6
3.2. SonarQube Configuration	8
3.2.1. Generate user token	8
3.2.2. Webhook	8
3.2.3. Jenkins integration	8
4. Create project	9
4.1. Create and integrate git repository	9
4.2. start new devonfw project	9
4.3. cicd configuration	9
4.3.1. Manual configuration	9
5. Deployment	9
6. Annexes	9
6.1. Custom Services	9
6.1.1. BitBucket	10
6.2. Mirabaud CICD Environment Setup	21
6.2.1. 1. Install Docker and Docker Compose in RHEL 6.5	21
6.2.2. 2. Directories structure	22
6.2.3. 3. CICD Services with Docker	23

6.2.4. 4. CICD Services with Docker Compose	25
6.2.5. 5. Service Integration	28
6.2.6. Jenkins - GitLab integration	29
6.2.7. Jenkins - Nexus integration	36
6.2.8. Jenkins - SonarQube integration	41
6.3. OKD (<i>OpenShift Origin</i>)	49
6.3.1. What is OKD	49
6.3.2. Install OKD (<i>Openshift Origin</i>)	50
6.3.3. How to use Oc Cluster Wrapper	51
6.3.4. devonfw Openshift Origin Initial Setup	51
6.3.5. s2i devonfw	52
6.3.6. devonfw templates	53
6.3.7. Customize Openshift Origin for devonfw	55

What is devonfw shop floor?



devonfw shop floor is a set of documentation, tools and methodologies used to configure the provisioning, development and uat environments used in your projects. devonfw shop floor allows the administrators of those environments to apply CI/CD operations and enables automated application deployment.

devonfw shop floor is mainly oriented to configure the provisioning environment provided by Production Line and deploy applications on an OpenShift cluster. In the cases where Production Line or OpenShift cluster are not available, there will be alternatives to achieve similar goals.

The **devonfw shop floor 4 OpenShift** is a solution based on the experience of priming devonfw for OpenShift by RedHat.



RED HAT® OPENS SHIFT PRIMED

Let's start.

1. How to use it

This is the documentation about shop floor and its different tools. Here you are going to learn how to create new projects, so that they can include continuous integration and continuous delivery processes, and be deployed automatically in different environments.

1.1. Prerequisites - Provisioning environment

To start working you need to have some services running in your provisioning environment, such as Jenkins (automation server), GitLab (git repository), SonarQube (program analysis), Nexus (software repository) or similar.

To host those services we recommend to have a Production Line instance but you can use other platforms. Here is the list for the different options:

- [Production Line](#).
- [dsf4docker](#).

1.2. Step 1 - Configuration and services integration

The first step is configure your services and integrate them with jenkins. Here you have an example about how to manually configure the next services:

- [Nexus](#).
- [SonarQube](#).

1.3. Step 2 - Create the project

1.3.1. Create and integrate git repository

The second is create or git repository and integrate it with Jenkins.

Here you can find a manual guide about how it:

- [GitLab](#) new project.

1.3.2. start new devonfw project

It is time to create your devonfw project:

- visit our [devon4ng](#) guide.
- visit our [devon4j](#) guide.

1.3.3. cicd configuration

Now you need to add cicd files in your project.

Manual configuration

Jenkinsfile

Here you can find all that you need to know to do your [Jenkinsfile](#).

1.4. Step 3 - Deployment

- [dsf4openshift](#).

2. Provisioning environments

2.1. Production Line provisioning environment



The Production Line Project is a set of server-side collaboration tools for Capgemini engagements. It has been developed for supporting project engagements with individual tools like issue tracking, continuous integration, continuous deployment, documentation, binary storage and much more!

For additional information use the [official documentation](#).

2.1.1. How to obtain your Production Line

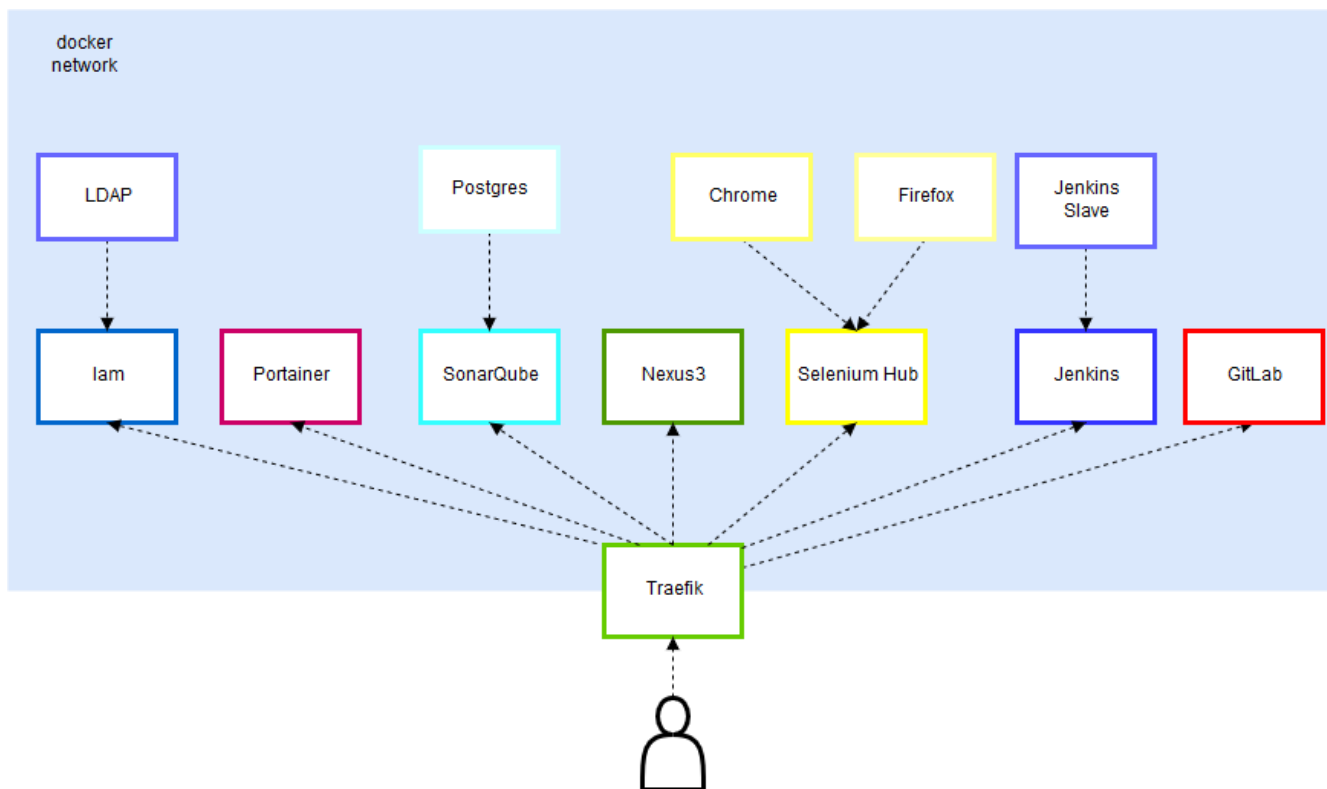
You can order your Production Line environment instance following the [official guide](#). Remember that you need to order at least the next tools: * Jenkins * GitLab * SonarQube * Nexus

[Back](#).

2.2. dsf4docker provisioning environment



2.2.1. Architecture overview



2.2.2. Prerequisite

To use dsf4docker provisioning environment you need a remote server and you must clone or download devonfw shop floor.

2.2.3. How to use it

Navigate to `./devonfw-shop-floor/dsf4docker/environment` and here you can find one scripts to install it, and another one to unistall it.

Install devonfw shop floor 4 Docker

There is an installation script to do so, so the complete installation should be completed by running it. Make sure this script has execution permissions in the Docker Host:

```
# chmod +x dsf4docker-install.sh
# sudo ./dsf4docker-install.sh
```

This script, besides the container "installation" itself, will also adapt the `docker-compose.yml` file to your host (using `sed` to replace the `IP_ADDRESS` word of the file for your real Docker Host's IP address).

Uninstall devonfw shop floor 4 Docker

As well as for the installation, if we want to remove everything concerning **devonfw shop floor 4 Docker** from our Docker Host, we'll run this script:

```
# chmod +x dsf4docker-uninstall.sh
# sudo ./dsf4docker-uninstall.sh
```

2.2.4. A little history

The **Docker** part of the shop floor is created based on the experience of the environment setup of the project **Mirabaud Advisory**, and intended to be updated to latest versions. Mirabaud Advisory is a web service developed with devonfw (Java) that, alongside its own implementation, it needed an environment both for the team to follow CICD rules through their 1-week-long sprints and for the client (Mirabaud) to check the already done work.

There is a practical experience about the [Mirabaud Case](#).

[Back](#).

3. Configuration and services integration

3.1. Nexus Configuration

In this document you will see how you can configure Nexus repository and how to integrate it with jenkins.

3.1.1. Prerequisites

Repositories

You need to have one repository for snapshots, another for releases and another one for release-candidates. Normally you use maven2 (hosted) repositories and if you are going to use a docker registry, you need docker (hosted) too.

To create a repository in Nexus go to the administration clicking on the gear icon at top menu bar. Then on the left menu click on Repositories and press the **Create repository** button.

[nexus create repository] | [./images/configuration/nexus-create-repository.png](#)

Now you must choose the type of the repository and configure it. This is an example for Snapshot:

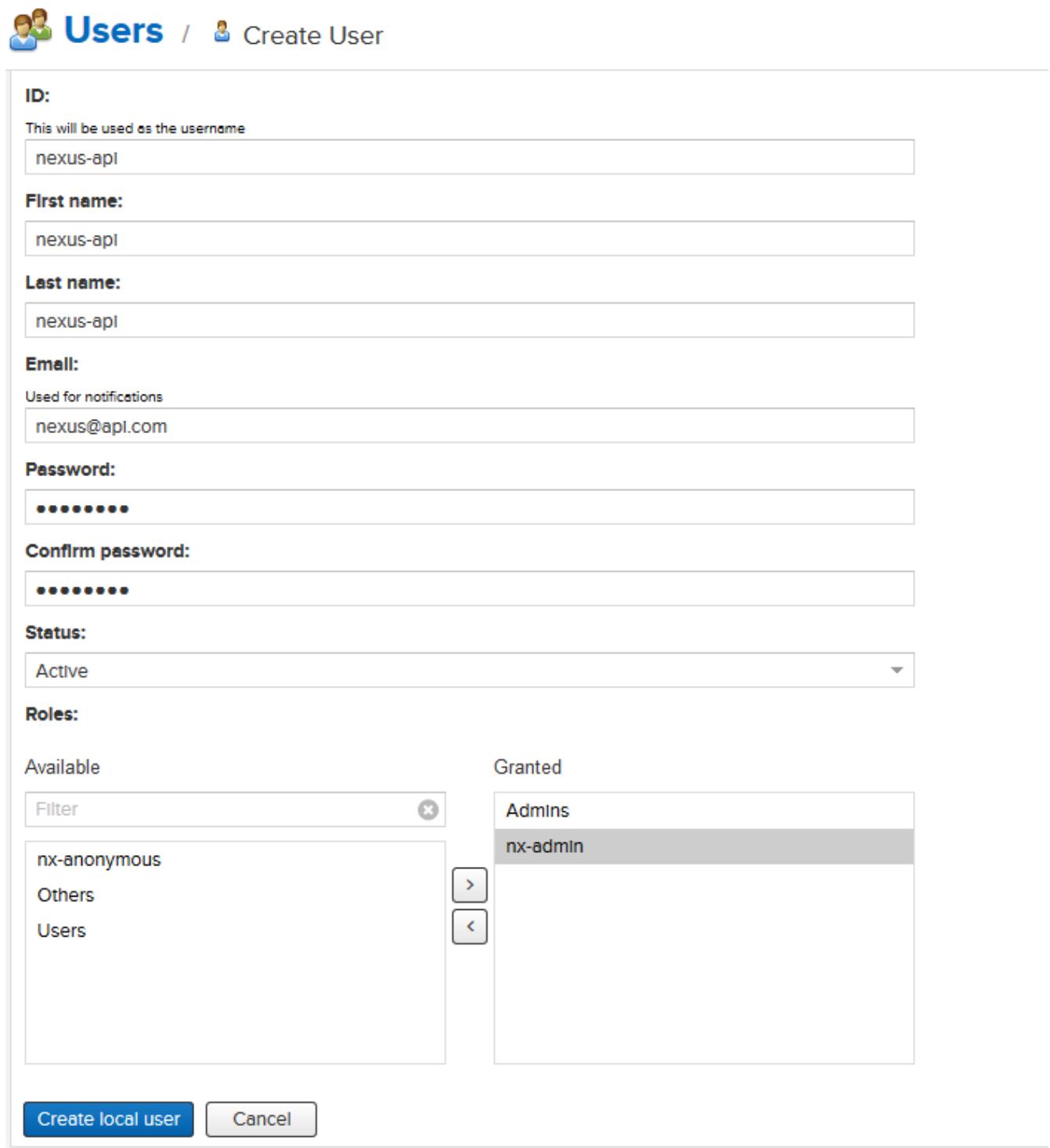
[nexus create repository form] | [./images/configuration/nexus-create-repository-form.png](#)

[[dsf-configure-nexus.asciidoc_create-user-to-upload/download-content]] == Create user to upload/download content

Once you have the repositories, you need a user to upload/download content. To do it go to the administration clicking on the gear icon at top menu bar. Then on the left menu click on Users and press the **Create local** user button.

[nexus create user] | [./images/configuration/nexus-create-user.png](#)

Now you need to fill a form like this:



Users / **Create User**

ID:
This will be used as the username
nexus-api

First name:
nexus-api

Last name:
nexus-api

Email:
Used for notifications
nexus@api.com

Password:
••••••••

Confirm password:
••••••••

Status:
Active

Roles:

Available

Filter ✕

nx-anonymous
Others
Users

Granted

Admins
nx-admin

>
<

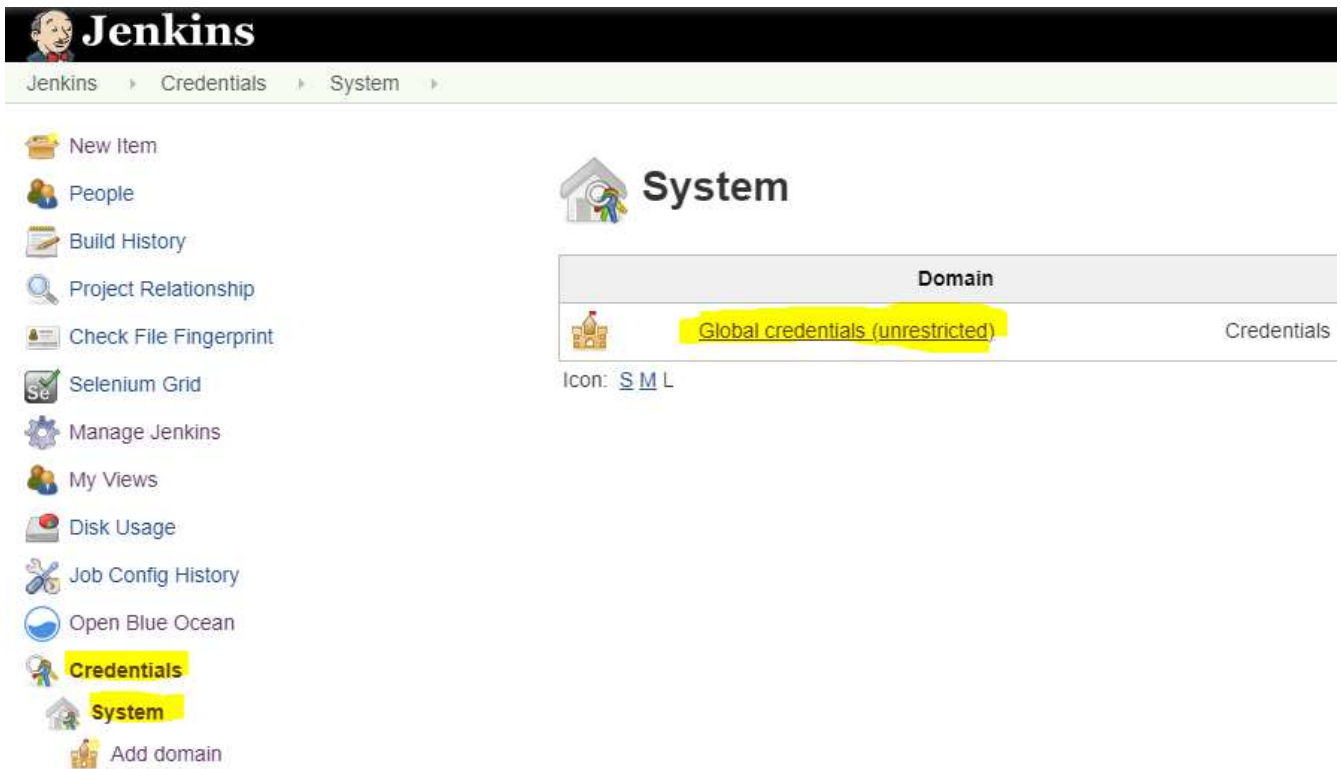
Create local user Cancel

3.1.2. Jenkins integration

To use Nexus in our pipelines you need to configure Jenkins.

Add nexus user credentials

First of all you need to add the user created in the step before to Jenkins. To do it (on the left menu) click on Credentials, then on System. Now you could access to **Global credentials (unrestricted)**.



Enter on it and you could see a button on the left to **Add credentials**. Click on it and fill a form like this:

Add the nexus user to maven global settings

Now you need to go to Manage Jenkins clicking on left menu and enter in **Managed files**.

You can edit or remove your configuration files



Edit the Global Maven settings.xml to add your nexus repositories credentials as you could see in the next image:

The configuration

ID	MavenSettings		
Name	MyGlobalSettings		
Comment	global settings		
Replace All	<input checked="" type="checkbox"/>		
Server Credentials	ServerId	pl-nexus	
	Credentials	nexus-api/***** (nexus-api)	
		Add	
			Delete

And you are done.

3.2. SonarQube Configuration

To use SonarQube you need to use a token to connect, and to know the results of the analysis you need a webhook. Also, you need to install and configure SonarQube in Jenkins.

3.2.1. Generate user token

To generate the user token, go to your account clicking in the left icon on the top menu bar.

[sonarqube administration] | *./images/configuration/sonarqube-administration.png*

Go to security tab and generate the token.

[sonarqube token] | *./images/configuration/sonarqube-token.png*

3.2.2. Webhook

When you execute our SonarQube scanner in our pipeline job, you need to ask SonarQube if the quality gate has been passed. To do it you need to create a webhook.

Go to administration clicking the option on the top bar menu and select the tab for Configuration.

Then search in the left menu to go to webhook section and create your webhook.

An example for Production Line:

[sonarqube webhook] | *./images/configuration/sonarqube-webhook.png*

3.2.3. Jenkins integration

To use SonarQube in our pipelines you need to configure Jenkins to integrate SonarQube.

SonarQube Scanner

First, you need to configure the scanner. Go to Manage Jenkins clicking on left menu and enter in **Global Tool Configuration**.

Go to SonarQube Scanner section and add a new SonarQube scanner like this.

[sonarqube jenkins scanner] | [./images/configuration/sonarqube-jenkins-scanner.png](#)

SonarQube Server

Now you need to configure where is our SonarQube server using the user token that you create before. Go to Manage Jenkins clicking on left menu and enter in **Configure System**.

For example, in ProductionLine the server is the next:

[sonarqube jenkins server] | [./images/configuration/sonarqube-jenkins-server.png](#)



Remember, the token was created at the beginin of this SonarQube configuration.

4. Create project

4.1. Create and integrate git repository

include::dsf-configure-gitlab.asciidoc[leveloffset=2].

4.2. start new devonfw project

It is time to create your devonfw project:

- visit our [devon4ng](#) guide.
- visitr our [devon4j](#) guide.

4.3. cimd configuration

4.3.1. Manual configuration

Jenkinsfile

include::dsf-configure-jenkins.asciidoc[leveloffset=2].

5. Deployment

include::dsf-deployment-dsf4openshift.asciidoc[leveloffset=2].

6. Annexes

6.1. Custom Services

6.1.1. BitBucket

[Under construction]

The purpose of the present document is to provide the basic steps carried out to setup a BitBucket server in OpenShift.

Introduction

BitBucket is the Atlassian tool that extends the Git functionality, by adding integration with JIRA, Confluence, or Trello, as well as incorporates extra features for security or management of user accounts (See [BitBucket](#)).

BitBucket server is the Atlassian tool that runs the BitBucket services (See [BitBucket server](#)).

The followed approach has been not using command line, but OpenShift Web Console, by deploying the Docker image **atlassian/bitbucket-server** (available in [Docker Hub](#)) in the existing project **Deployment**.

The procedure below exposed consists basically in three main steps:

1. Deploy the BitBucket server image (from OpenShift web console)
2. Add a route for the external traffic (from OpenShift web console)
3. Configure the BitBucket server (from BitBucket server web console)

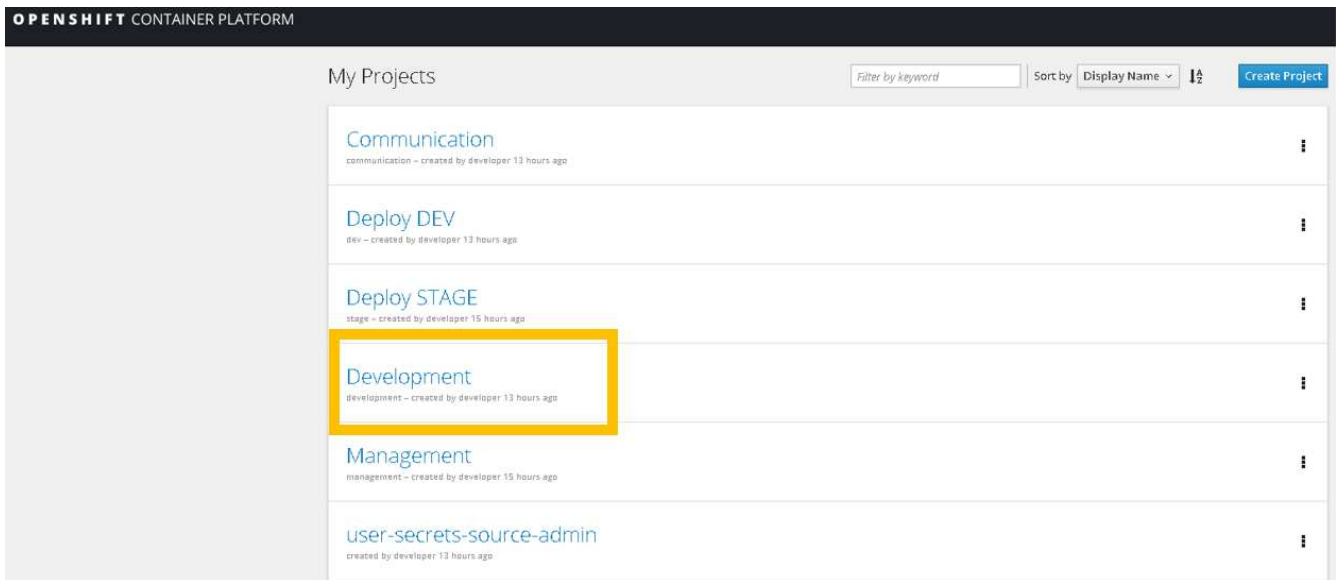
Prerequisites

- OpenShift up & running
- Atlassian account (with personal account key). Not required for OpenShift, but for the initial BitBucket server configuration.

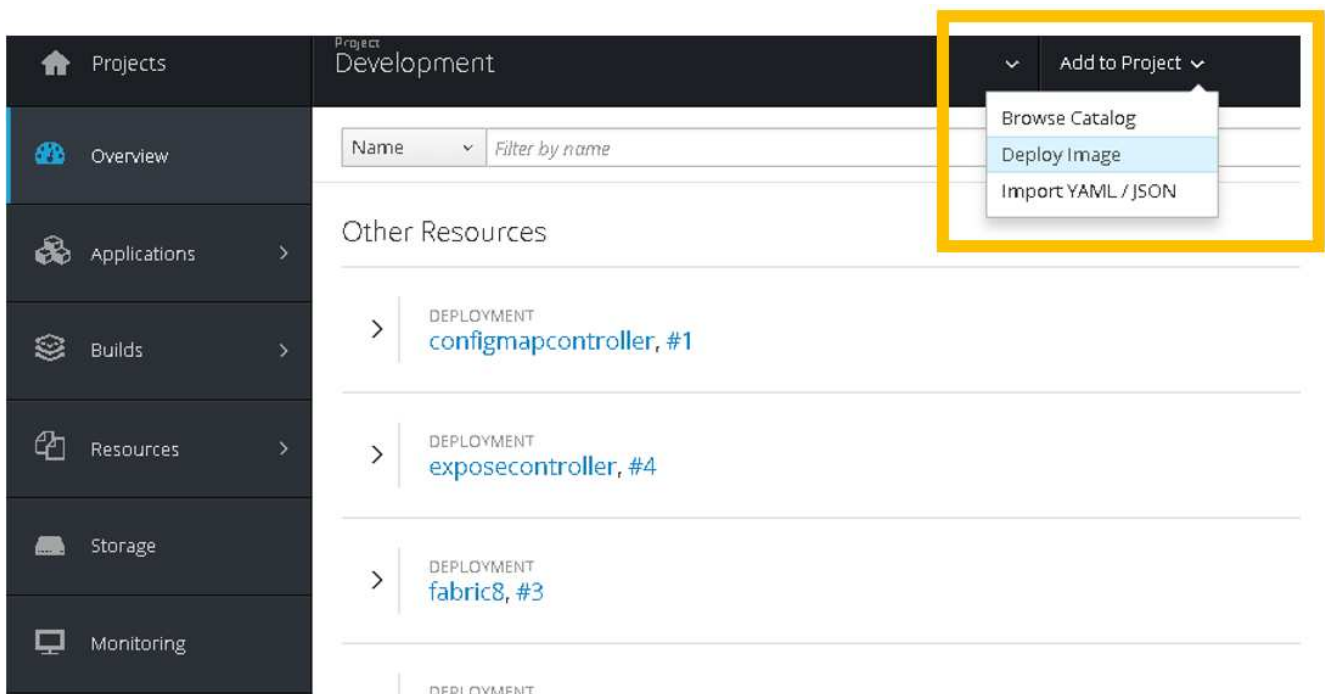
Procedure

[[dsf-openshift-services-bitbucket-basic-server-setup.asciidoc_step-0-log-into-our-linkhttps//10.68.26.1638443/console/logout[openshift-web-console]]] === Step 0: Log into our [OpenShift Web console](#) image::./images/others/bitbucket/step0.png[]

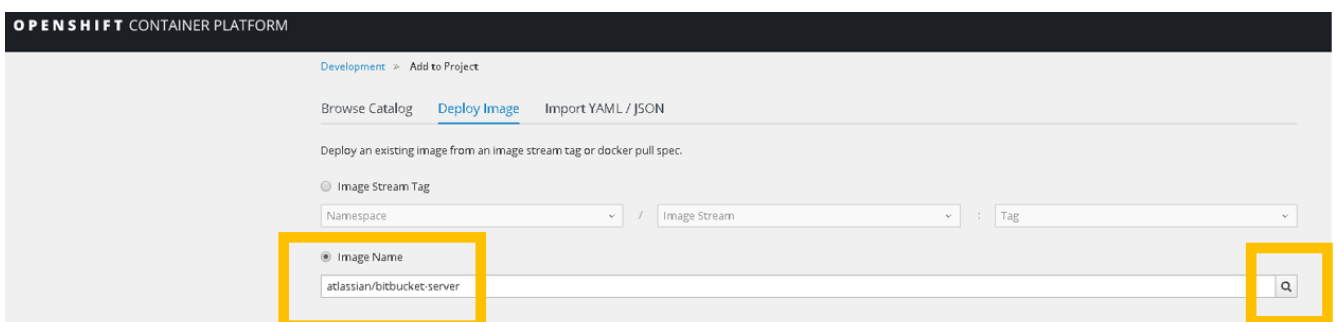
Step 1: Get into Development project



Step 2.1: Deploy a new image to the project



[[dsf-openshift-services-bitbucket-basic-server-setup.asciidoc_step-2.2-introduce-the-image-name-available-in-link<https://hub.docker.com/r/atlassian/bitbucket-server/>[docker-hub]-and-search]] ===
 Step 2.2: Introduce the image name (available in [Docker Hub](https://hub.docker.com/r/atlassian/bitbucket-server/)) and search Image name: **atlassian/bitbucket-server**



Step 2.3: Leave by the moment the default config. since it is enough for the basic setup. Press Create

*** Name**

bitbucket-server

Identifies the resources created for this image.

Pull Secret

Secret name: ✕

Secret for authentication when pulling images from a secured registry. [Learn More](#)

[Create New Secret](#)

Environment Variables 🔗 About Environment Variables

Name: Value: ✕

[Add Environment Variable](#) | [Add Environment Variable Using a Config Map or Secret](#)

Labels 🔗 About Labels

The following labels are being added automatically. If you want to override them, you can do so below.

app: bitbucket-server

Each label is applied to each created resource.

Name: Value: ✕

[Add Label](#)

Create **Cancel**

Step 2.4: Copy the oc commands in case it is required to work via command line, and Go to overview

Completed [Go to overview.](#)

Manage your app

The web console is convenient, but if you need deeper control you may want to try our command line tools.

Command line tools

[Download and install the oc](#) command line tool. After that, you can start by logging in, switching to this particular project, and displaying an overview of it, by doing:

```
oc login https://10.68.26.163:8443
oc project development
oc status
```

For more information about the command line tools, check the [CLI Reference](#) and [Basic CLI Operations](#).

Step 2.5: Wait until OpenShift deploys and starts up the image. All the info will be available.

Please notice that there are no pre-configured routes, hence the application is not accessible from outside the cluster.

APPLICATION
bitbucket-server

DEPLOYMENT
bitbucket-server, #1

CONTAINER: BITBUCKET-SERVER

Image: atlassian/bitbucket-server 7f1a28b 310.0 MiB

Ports: 7990/TCP and 1 other

1 pod

Networking

SERVICE Internal Traffic
bitbucket-server
7990/TCP (7990-tcp) → 7990 and 1 other

ROUTES External Traffic
[Create Route](#)

Step 3: Create a route in order for the application to be accessible from outside the cluster (external traffic). Press Create

Please notice that there are different fields that can be specified (hostname, port). If required, the value of those fields can be modified later.

Hostname

Public hostname for the route. If not specified, a hostname is generated.

The hostname can't be changed after the route is created.

Path

Path that the router watches to route traffic to the service.

* Service

bitbucket-server

Service to route to.

Target Port

7990 → 7990 (TCP)

Target port for traffic.

Alternate Services

☐ Split traffic across multiple services

Routes can direct traffic to multiple services for A/B testing. Each service has a weight controlling how much traffic it gets.

Security

☐ Secure route

Routes can be secured using several TLS termination types for serving certificates.

Labels

Labels for this route.

Name

Value

×

[Add Label](#)

[Copy Service Labels](#)

Create

Cancel

[About Labels](#)

Leave by the moment the default config. as it is enough for the basic setup.

The route for external traffic is now available.

APPLICATION

bitbucket-server

<http://bitbucket-server-development.apps.10.68.26.163.nip.io/>

DEPLOYMENT

bitbucket-server, #1

CONTAINER: BITBUCKET-SERVER

Imager: anlassian/bitbucket-server 7F1a9Bb.318.0 MiB

Ports: 7990/TCP and 1 other

Networking

SERVICE: Internal Traffic

bitbucket-server

7990/TCP (7990-tcp) → 7990 and 1 other

ROUTES: External Traffic

<http://bitbucket-server-development.apps.10.68.26.163.nip.io/>

Route bitbucket-server, target port 7990-tcp

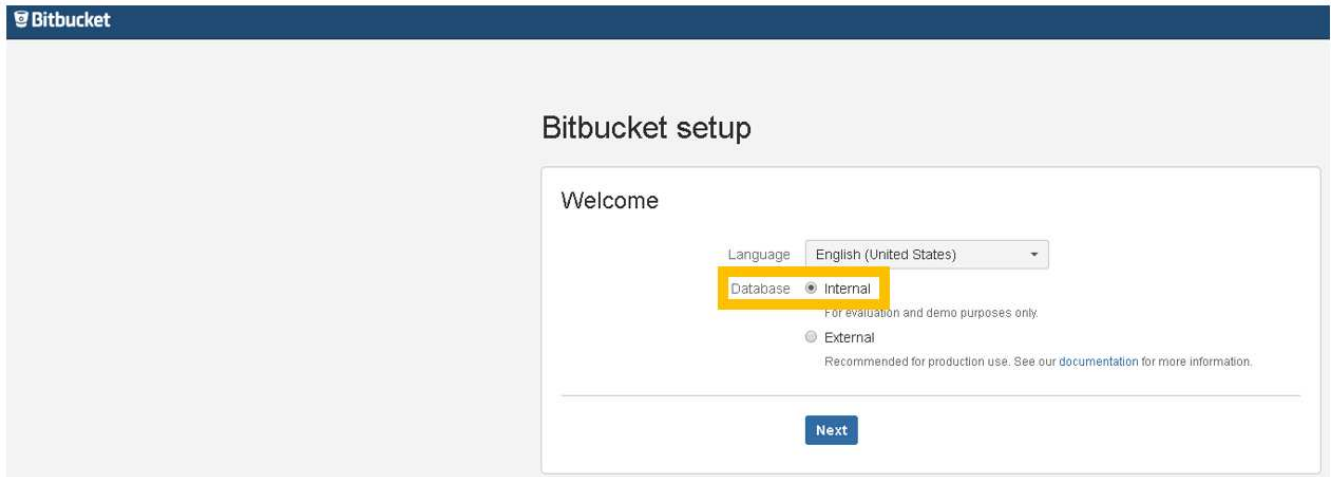
1 pod

Now the BitBucker server container is up & running in our cluster.

The below steps correspond to the basic configuration of our BitBucket server.

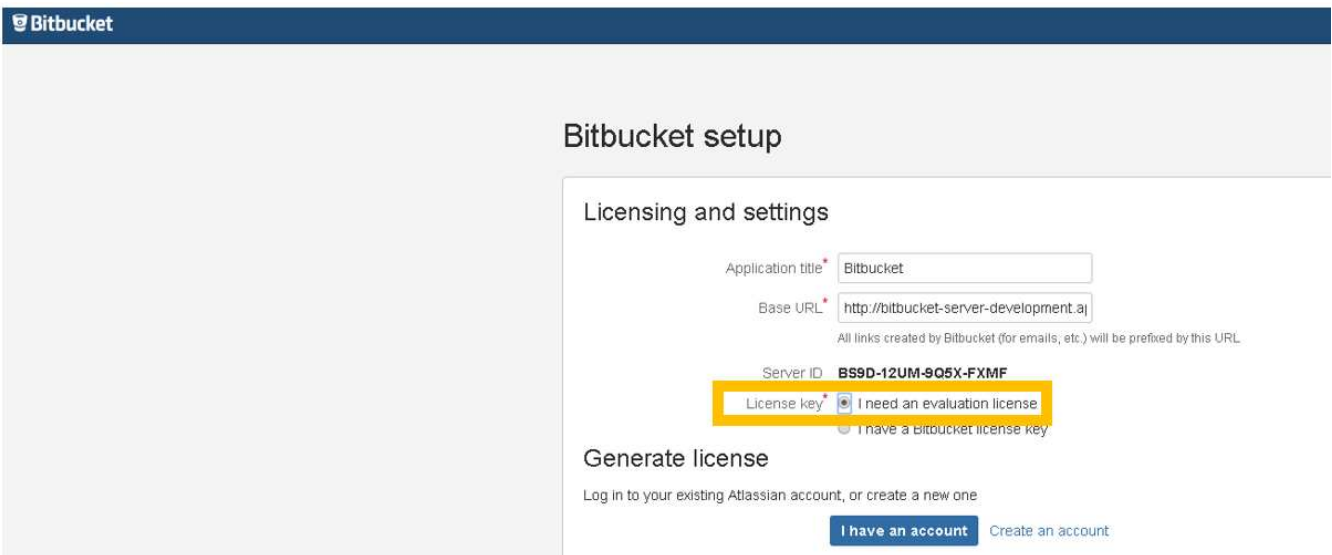
Step 4.1: Click on the link of the external traffic route. This will open our BitBucket server setup wizard

Step 4.2: Leave by the moment the Internal database since it is enough for the basic setup (and it can be modified later), and click Next



The screenshot shows the Bitbucket setup wizard's 'Welcome' screen. At the top, there's a 'Bitbucket' header. The main heading is 'Bitbucket setup'. Below it, the word 'Welcome' is displayed. There are two configuration options: 'Language' set to 'English (United States)' and 'Database' with 'Internal' selected (highlighted by a yellow box) and 'External' as an alternative. A note states 'For evaluation and demo purposes only.' and 'Recommended for production use. See our [documentation](#) for more information.' A 'Next' button is at the bottom right.

Step 4.3: Select the evaluation license, and click I have an account



The screenshot shows the 'Licensing and settings' screen of the Bitbucket setup wizard. It includes fields for 'Application title' (Bitbucket), 'Base URL' (http://bitbucket-server-development.ai), and 'Server ID' (BS9D-12UM-9Q5X-FXMF). The 'License key' section has two radio buttons: 'I need an evaluation license' (selected and highlighted by a yellow box) and 'I have a bitbucket license key'. Below this is the 'Generate license' section with the text 'Log in to your existing Atlassian account, or create a new one' and two buttons: 'I have an account' and 'Create an account'.

Step 4.4: Select the option Bitbucker (Server)

My Atlassian

New Evaluation License

Product: Bitbucket

License type:

Bitbucket (Server)	Bitbucket (Data Center)
<ul style="list-style-type: none">• Manage the entire application on your own servers or virtual machines.• Deployable to a single server.	<p>Everything with server plus:</p> <ul style="list-style-type: none">• Active-active clustering for true high availability and uninterrupted access.• High performance under high load and at peak times• Disaster recovery.
<div>✓</div>	<div>Select</div>

Organization:

Your instance is: ☒ up and running ☐ not installed yet

Server ID:

Please note we only provide evaluation support for 90 days per product.

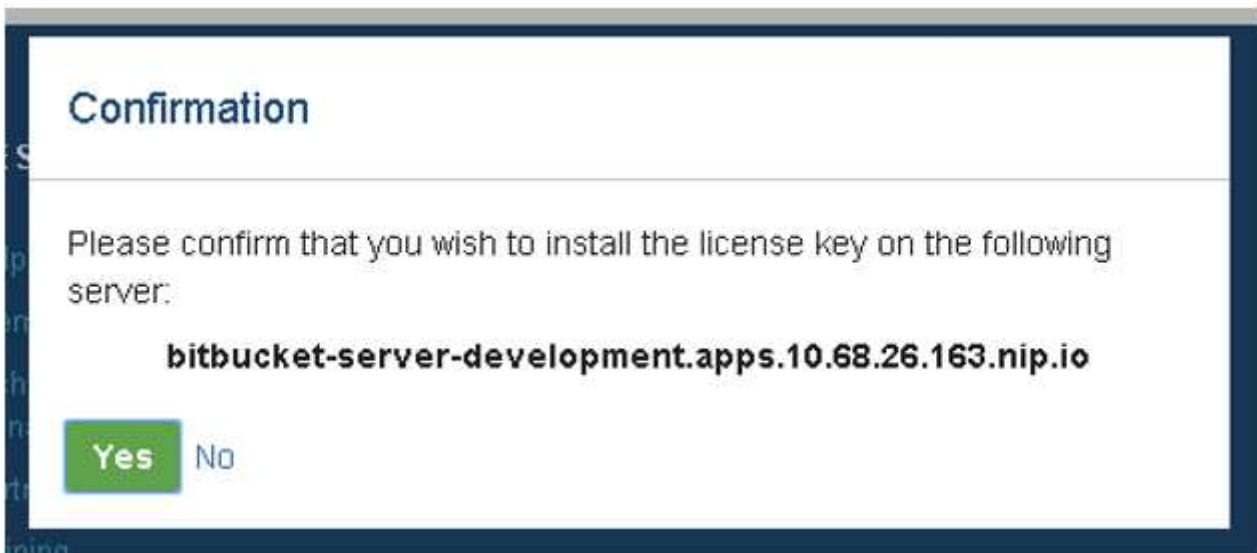
By clicking here you accept the [Atlassian Customer Agreement](#).

Generate License

[Cancel](#)

Step 4.5: Introduce your organization (Capgemini), and click Generate License

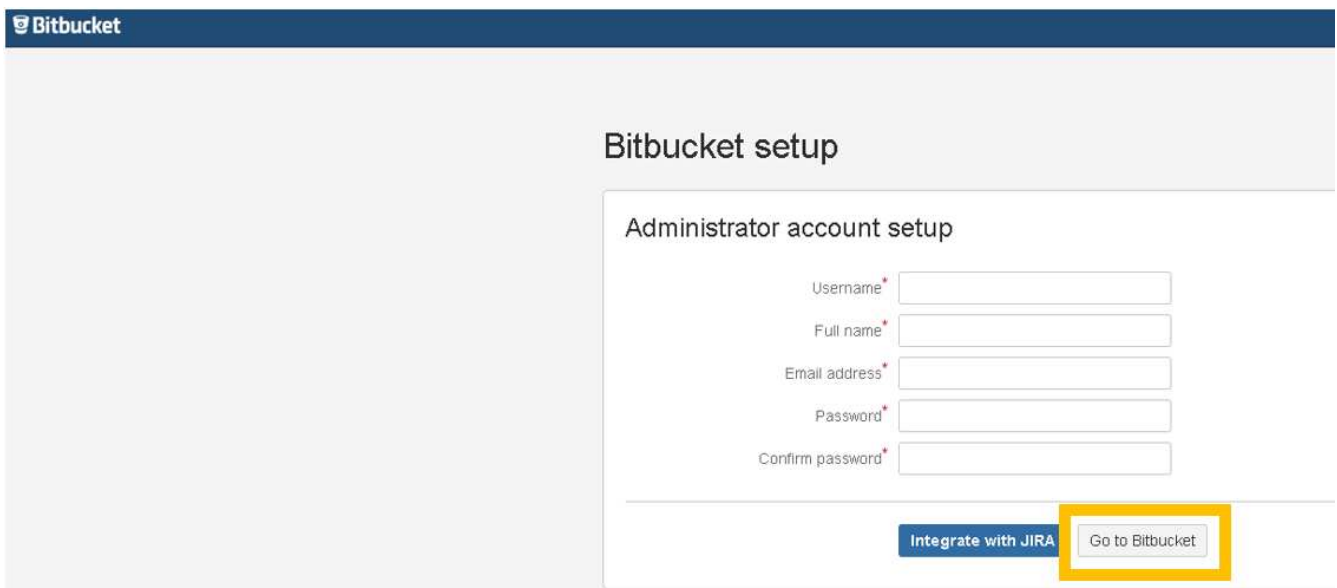
Step 4.6: Confirm that you want to install the license on the BitBucket server



The license key will be automatically generated. Click **Next**

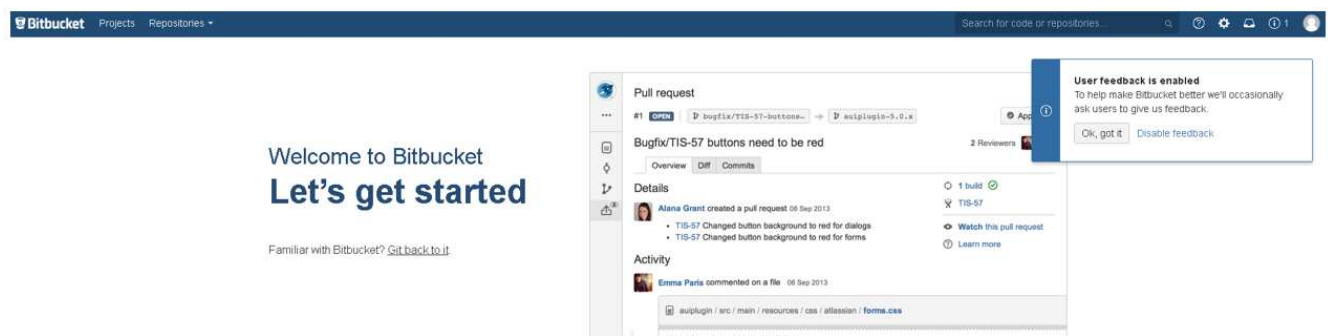
Step 4.7: Introduce the details of the Administration account.

Since our BitBucket server is not going to be integrated with JIRA, click on Go to Bitbucket. The integration with JIRA can be configured later.



Step 4.8: Log in with the admin account that has been just created

DONE !!



The purpose of the present document is to provide the basic steps carried out to improve the configuration of BitBucket server in OpenShift.

The improved configuration consists on:

- Persisten Volume Claims
- Health Checks (*pending to be completed*)

Persisten Volume Claims.

Please notice that the BitBucket server container does not use persistent volume claims by default, which means that the data (e.g.: BitBucket server config.) will be lost from one deployment to another.

The screenshot shows the OpenShift console interface for a deployment named 'bitbucket-server'. The 'Configuration' tab is selected. The 'Details' section shows the deployment configuration, including selectors, replicas, strategy, and timeouts. The 'Template' section shows a warning that the container does not have health checks. The 'Containers' section shows the container configuration, including the image, ports, and mount. The 'Volumes' section is highlighted with a yellow box, showing a volume named 'bitbucket-server-1' of type 'empty dir'.

bitbucket-server created a day ago

app bitbucket-server

History Configuration Environment Events

Details

Selectors: app=bitbucket-server deploymentconfig=bitbucket-server

Replicas: 1 replica

Strategy: Rolling

Timeout: 600 sec

Update Period: 1 sec

Interval: 1 sec

Max Unavailable: 25%

Max Surge: 25%

Template

Container bitbucket-server does not have health checks to ensure your application is running correctly. [Add Health Checks](#)

Containers

CONTAINER: BITBUCKET-SERVER

Image: atlassian/bitbucket-server 7f1a98b 318.0 MiB

Ports: 7990/TCP, 7999/TCP

Mount: bitbucket-server-1 → /var/atlassian/application-data/bitbucket: read-write

Volumes

bitbucket-server-1 [Remove](#)

Type: empty dir (temporary directory destroyed with the pod)

Medium: node's default

Autoscaling

[Add Autoscaler](#)

Hooks [Learn More](#)

none

Triggers

Manual (CLI): [Learn More](#) or rollout latest dc/bitbucket-server -n mypr


Change Of: Config

New Image For: myproject/bitbucket-server:latest

It is very important to create a persisten volume claim in order to prevent the mentioned loss of data.

Step 1: Add storage

Details

Selectors:	app=bitbucket-server deployment:config=bitbucket-server
Replicas:	1 replica 
Strategy:	Rolling
Timeout:	10800 sec
Update Period:	1 sec
Interval:	1 sec
Max Unavailable:	25%
Max Surge:	25%

Template

 Container bitbucket-server does not have health checks to ensure your application is running correctly.
[Add Health Checks](#)

Containers

CONTAINER: BITBUCKET-SERVER

-  **Image:** atlassian/bitbucket-server
- Ports:** 7990/TCP, 7999/TCP
- Mount:** bitbucket-server-storage → /var/atlassian/application-data/bitbucket read-write

Volumes

bitbucket-server-storage [Remove](#)

Type:	persistent volume claim (reference to a persistent volume claim)
Claim name:	bitbucket-server-storage
Mode:	read-write

[Add Storage](#) | [Add Config Files](#)

Autoscaling

[Add Autoscaler](#)Hooks [Learn More](#)

none

Triggers

Manual (CLI):[Learn More](#)**Change Of:**oc rollout latest dc/bitbucket-server -n deve 

Config

Step 2: Select the appropriate storage, or create it from scratch if necessary

Add Storage

Add an existing persistent volume claim to the template of deployment config bitbucket-server.

* Storage

<input checked="" type="radio"/>	bitbucket-server-storage	100 GiB	(Read-Write-Once)	Bound to volume pv0067
<input type="radio"/>	jenkins	100 GiB	(Read-Write-Once)	Bound to volume pv0075

Select storage to use or [create storage](#).

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path

example: /data

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Subpath

example: application/resources

Optional path within the volume from which it will be mounted into the container. Defaults to the volume's root.

Volume Name

(generated if empty)

Unique name used to identify this volume. If not specified, a volume name is generated.

☐ Read only

Mount the volume as read-only.

☐ Pause rollouts for this deployment config

Pausing lets you make changes without triggering a rollout. You can resume rollouts at any time. If unchecked, a new rollout will start on save.

Add

Cancel

Step 3: Introduce the required information

- **Path** as it is specified in the [BitBucket server Docker image](#) (/var/atlassian/application-data/bitbucket)
- **Volume name** with a unique name to clearly identify the volume

Add Storage

Add an existing persistent volume claim to the template of deployment config bitbucket-server.

* Storage

<input checked="" type="radio"/>	bitbucket-server-storage	100 GiB	(Read-Write-Once)	Bound to volume pv0067
<input type="radio"/>	jenkins	100 GiB	(Read-Write-Once)	Bound to volume pv0075

Select storage to use or [create storage](#).

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Subpath

Optional path within the volume from which it will be mounted into the container. Defaults to the volume's root.

Volume Name

Unique name used to identify this volume. If not specified, a volume name is generated.

☐ Read only

Mount the volume as read-only.

☐ Pause rollouts for this deployment config

Pausing lets you make changes without triggering a rollout. You can resume rollouts at any time. If unchecked, a new rollout will start on save.

Add

Cancel

The change will be immediately applied

bitbucket-server created a day ago

app bitbucket-server

History Configuration Environment Events

Details

Selectors:	app=bitbucket-server deploymentconfig=bitbucket-server
Replicas:	1 replica
Strategy:	Rolling
Timeout:	600 sec
Update Period:	1 sec
Interval:	1 sec
Max Unavailable:	25%
Max Surge:	25%

Template

Container bitbucket-server does not have health checks to ensure your application is running correctly.
[Add Health Checks](#)

Containers

CONTAINER: BITBUCKET-SERVER

- Image: atlassian/bitbucket-server 7f1a98b 318.0 MiB
- Ports: 7990/TCP, 7999/TCP
- Mount: bitbucket-server-volume → /var/atlassian/application-data/bitbucket read-write

Volumes

bitbucket-server-volume [Remove](#)

Type:	persistent volume claim (reference to a persistent volume claim)
Claim name:	bitbucket-server-storage
Mode:	read-write

Autoscaling

[Add Autoscaler](#)

Hooks [Learn More](#)

none

Triggers

Manual (CLI):

[Learn More](#)

Change Of:

New Image For:

oc rollout latest dc/bitbucket-server -n mypi

Config

myproject/bitbucket-server:latest

6.2. Mirabaud CICD Environment Setup

Initial requirements:

- **OS:** RHEL 6.5

Remote setup in CI machine (located in the Netherlands)

- Jenkins
- Nexus
- GitLab
- Mattermost
- Atlassian Crucible
- SonarQube

6.2.1. 1. Install Docker and Docker Compose in RHEL 6.5

Docker

Due to that OS version, the only way to have Docker running in the CI machine is by installing it from the **EPEL** repository (Extra Packages for Enterprise Linux).

1. Add EPEL

```
# rpm -iUvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

2. Install **docker.io** from that repository

```
# yum -y install docker-io
```

3. Start Docker daemon

```
# service docker start
```

4. Check the installation

```
# docker -v
Docker version 1.7.1, build 786b29d/1.7.1
```

Docker Compose

Download and install it via **curl**. It will use [this site](#).

```
# curl -L https://github.com/docker/compose/releases/download/1.5.0/docker-compose-
`uname -s`-`uname -m` > /usr/local/bin/docker-compose

# chmod +x /usr/local/bin/docker-compose
```

Add it to your **sudo** path:

1. Find out where it is:

```
# echo $PATH
```

2. Copy the **docker-compose** file from **/usr/local/bin/** to your **sudo** PATH.

```
# docker-compose -v
docker-compose version 1.5.2, build 7240ff3
```

6.2.2. 2. Directories structure

Several directories had been added to organize some files related to docker (like **docker-compose.yml**) and docker volumes for each service. Here's how it looks:

```
/home
  /([username])
    /jenkins
      /volumes
        /jenkins_home
    /sonarqube
      /volumes
        /conf
        /data
        /extensions
        /lib
        /bundled-plugins
    /nexus
      /volumes
        /nexus-data
    /crucible
      /volumes
        /
    /gitlab
      docker-compose.yml
      /volumes
        /etc
          /gitlab
        /var
          /log
          /opt
    /mattermost
      docker-compose.yml
      /volumes
        /db
          /var
            /lib
              /postgresql
                /data
        /app
          /mattermost
            /config
            /data
            /logs
        /web
          /cert
```

6.2.3. 3. CI/CD Services with Docker

Some naming conventions had been followed as naming containers as `mirabaud_[service]`.

Several folders have been created to store each service's volumes, `docker-compose.yml(s)`, extra configuration settings and so on:

Jenkins

Command

```
# docker run -d -p 8080:8080 -p 50000:50000 --name=mirabaud_jenkins \  
-v /home/[username]/jenkins/volumes/jenkins_home:/var/jenkins_home \  
jenkins
```

Generate keystore

```
keytool -importkeystore -srckeystore server.p12 -srcstoretype pkcs12 -srcalias 1 \  
-destkeystore newserver.jks -deststoretype jks -destalias server
```

Start jenkins with SSL (TODO: make a docker-compose.yml for this):

```
sudo docker run -d --name mirabaud_jenkins -v /jenkins:/var/jenkins_home -p 8080:8443 \  
jenkins --httpPort=-1 --httpsPort=8443 \  
--httpsKeyStore=/var/jenkins_home/certs/keystore.jks \  
--httpsKeyStorePassword=Mirabaud2017
```

Volumes

```
volumes/jenkins_home:/var/jenkins_home
```

SonarQube

Command

```
# docker run -d -p 9000:9000 -p 9092:9092 --name=mirabaud_sonarqube \  
-v /home/[username]/sonarqube/volumes/conf:/opt/sonarqube/conf \  
-v /home/[username]/sonarqube/volumes/data:/opt/sonarqube/data \  
-v /home/[username]/sonarqube/volumes/extensions:/opt/sonarqube/extensions \  
-v /home/[username]/sonarqube/volumes/lib/bundled- \  
plugins:/opt/sonarqube/lib/bundled-plugins \  
sonarqube
```

Volumes

```
volumes/conf:/opt/sonarqube/conf \  
volumes/data:/opt/sonarqube/data \  
volumes/extensions:/opt/sonarqube/extensions \  
volumes/lib/bundled-plugins:/opt/sonarqube/lib/bundled-plugins
```

Nexus

Command

```
# docker run -d -p 8081:8081 --name=mirabaud_nexus\  
-v /home/[username]/nexus/nexus-data:/sonatype-work  
sonatype/nexus
```

Volumes

```
volumes/nexus-data:/sonatype-work
```

Atlassian Crucible

Command

```
# docker run -d -p 8084:8080 --name=mirabaud_crucible \  
-v /home/[username]/crucible/volumes/data:/atlassian/data/crucible  
mswinarski/atlassian-crucible:latest
```

Volumes

```
volumes/data:/atlassian/data/crucible
```

6.2.4. 4. CICD Services with Docker Compose

Both Services had been deploying by using the `# docker-compose up -d` command from their root directories (`/gitlab` and `/mattermost`). The syntax of the two `docker-compose.yml` files is the one corresponding with the 1st version (due to the `docker-compose v1.5`).

GitLab

[[dsf-mirabaud-cicd-environment-setup.asciidoc_`docker-compose.yml`]]==== `docker-compose.yml`

```
mirabaud:  
  image: 'gitlab/gitlab-ce:latest'  
  restart: always  
  ports:  
    - '8888:80'  
  volumes:  
    - '/home/[username]/gitlab/volumes/etc/gitlab:/etc/gitlab'  
    - '/home/[username]/gitlab/volumes/var/log:/var/log/gitlab'  
    - '/home/[username]/gitlab/volumes/var/opt:/var/opt/gitlab'
```

Command (docker)

```
docker run -d -p 8888:80 --name=mirabaud_gitlab \  
  -v /home/[username]/gitlab/volumes/etc/gitlab:/etc/gitlab \  
  -v /home/[username]/gitlab/volumes/var/log:/var/log/gitlab \  
  -v /home/[username]/gitlab/volumes/var/opt:/var/opt/gitlab \  
  gitlab/gitlab-ce
```

Volumes

```
volumes/etc/gitlab:/etc/gitlab  
volumes/var/opt:/var/log/gitlab  
volumes/var/log:/var/log/gitlab
```

Mattermost

[[dsf-mirabaud-cicd-environment-setup.asciidoc_`docker-compose.yml`]] == **docker-compose.yml**:

```

db:
  image: mattermost/mattermost-prod-db
  restart: unless-stopped
  volumes:
    - ./volumes/db/var/lib/postgresql/data:/var/lib/postgresql/data
    - /etc/localtime:/etc/localtime:ro
  environment:
    - POSTGRES_USER=mmuser
    - POSTGRES_PASSWORD=mmuser_password
    - POSTGRES_DB=mattermost

app:
  image: mattermost/mattermost-prod-app
  links:
    - db:db
  restart: unless-stopped
  volumes:
    - ./volumes/app/mattermost/config:/mattermost/config:rw
    - ./volumes/app/mattermost/data:/mattermost/data:rw
    - ./volumes/app/mattermost/logs:/mattermost/logs:rw
    - /etc/localtime:/etc/localtime:ro
  environment:
    - MM_USERNAME=mmuser
    - MM_PASSWORD=mmuser_password
    - MM_DBNAME=mattermost

web:
  image: mattermost/mattermost-prod-web
  ports:
    - "8088:80"
    - "8089:443"
  links:
    - app:app
  restart: unless-stopped
  volumes:
    - ./volumes/web/cert:/cert:ro
    - /etc/localtime:/etc/localtime:ro

```

SSL Certificate

How to generate the certificates:

Get the **crt** and **key** from CA or **generate a new one self-signed**. Then:


```
// 1. create the p12 keystore
# openssl pkcs12 -export -in cert.crt -inkey mycert.key -out certkeystore.p12

// 2. export the pem certificate with password
# openssl pkcs12 -in certkeystore.p12 -out cert.pem

// 3. export the pem certificate without password
# openssl rsa -in cert.pem -out key-no-password.pem
```

SSL:

Copy the cert and the key without password at:

`./volumes/web/cert/cert.pem`

and

`./volumes/web/cert/key-no-password.pem`

Restart the server and the SSL should be enabled at port **8089** using **HTTPS**.

Volumes

```
-- db --
volumes/db/var/lib/postgresql/data:/var/lib/postgresql/data
/etc/localtime:/etc/localtime:ro                                # absolute path

-- app --
volumes/app/mattermost/config:/mattermost/config:rw
volumes/app/mattermost/data:/mattermost/data:rw
volumes/app/mattermost/logs:/mattermost/logs:rw
/etc/localtime:/etc/localtime:ro                                # absolute path

-- web --
volumes/web/cert:/cert:ro
/etc/localtime:/etc/localtime:ro                                # absolute path
```

6.2.5. 5. Service Integration

All integrations had been done following **CICD Services Integration** guides:

- [Jenkins - Nexus integration](#)
- [Jenkins - GitLab integration](#)
- [Jenkins - SonarQube integration](#)



These guides may be obsolete. You can find here the [official configuration guides](#),

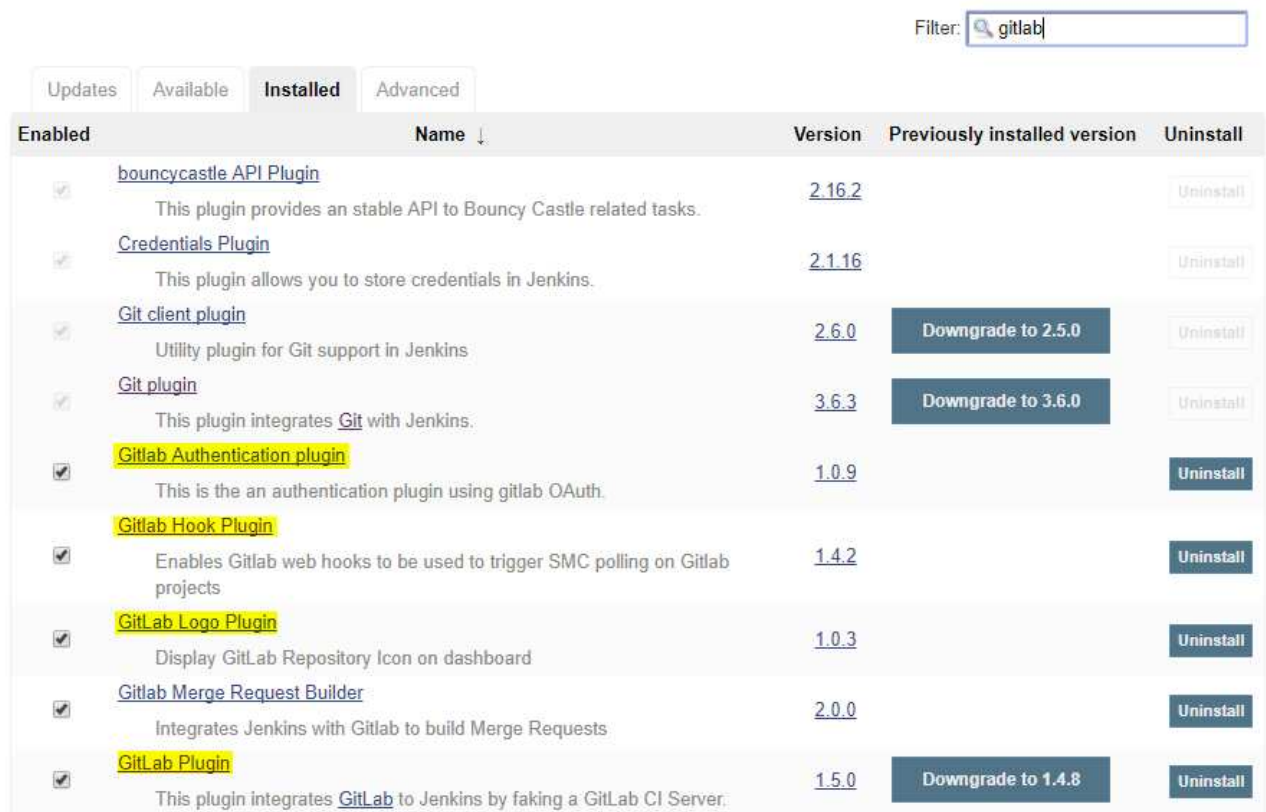
6.2.6. Jenkins - GitLab integration

The first step to have a Continuous Integration system for your development is to make sure that all your changes to your team's remote repository are evaluated by the time they are pushed. That usually implies the usage of so-called *webhooks*. You'll find a fancy explanation about what Webhooks are in [here](#).

To resume what we're doing here, we are going to prepare our Jenkins and our GitLab so when a developer pushes some changes to the GitLab repository, a pipeline in Jenkins gets triggered. Just like that, in an automatic way.

1. Jenkins GitLab plugin

As it usually happens, some Jenkins plug-in(s) must be installed. In this case, let's install those related with GitLab:



Filter: <input type="text" value="gitlab"/>				
Updates	Available	Installed	Advanced	
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.16.2		Uninstall
<input checked="" type="checkbox"/>	Credentials Plugin This plugin allows you to store credentials in Jenkins.	2.1.16		Uninstall
<input checked="" type="checkbox"/>	Git client plugin Utility plugin for Git support in Jenkins	2.6.0	Downgrade to 2.5.0	Uninstall
<input checked="" type="checkbox"/>	Git plugin This plugin integrates Git with Jenkins.	3.6.3	Downgrade to 3.6.0	Uninstall
<input checked="" type="checkbox"/>	Gitlab Authentication plugin This is the an authentication plugin using gitlab OAuth.	1.0.9		Uninstall
<input checked="" type="checkbox"/>	Gitlab Hook Plugin Enables Gitlab web hooks to be used to trigger SMC polling on Gitlab projects	1.4.2		Uninstall
<input checked="" type="checkbox"/>	GitLab Logo Plugin Display GitLab Repository Icon on dashboard	1.0.3		Uninstall
<input checked="" type="checkbox"/>	Gitlab Merge Request Builder Integrates Jenkins with Gitlab to build Merge Requests	2.0.0		Uninstall
<input checked="" type="checkbox"/>	GitLab Plugin This plugin integrates GitLab to Jenkins by faking a GitLab CI Server.	1.5.0	Downgrade to 1.4.8	Uninstall

2. GitLab API Token

To communicate with GitLab from Jenkins, we will need to create an authentication token from your GitLab user settings. A good practice for this would be to create it from a *machine user*. Something like (i.e.) `devonfw-ci/*****`.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access Token

Pick a name for the application, and we'll give you a unique personal access Token.

Name

Expires at

Scopes

- ☒ **api** Access your API
- ☐ **read_user** Read user information
- ☐ **read_registry** Read Registry

Create personal access token

Active Personal Access Tokens (2)

Name	Created	Expires	Scopes
Jenkins CI server	Nov 2, 2017	In 6 months	api, read_user, read_registry
devonfw-ci-token	Oct 30, 2017	In over 2 years	api, read_user, read_registry

Simply by adding a name to it and a date for it expire is enough:

Add a personal access Token

Pick a name for the application, and we'll give you a unique personal access Token.

Name

devonfw-ci

Expires at

2023-03-13

Scopes

- ☒ **api** Access your API
- ☒ **read_user** Read user information
- ☒ **read_registry** Read Registry

Create personal access token

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

Your New Personal Access Token

zFcZsZ4TC2ZM_Txu6rzo

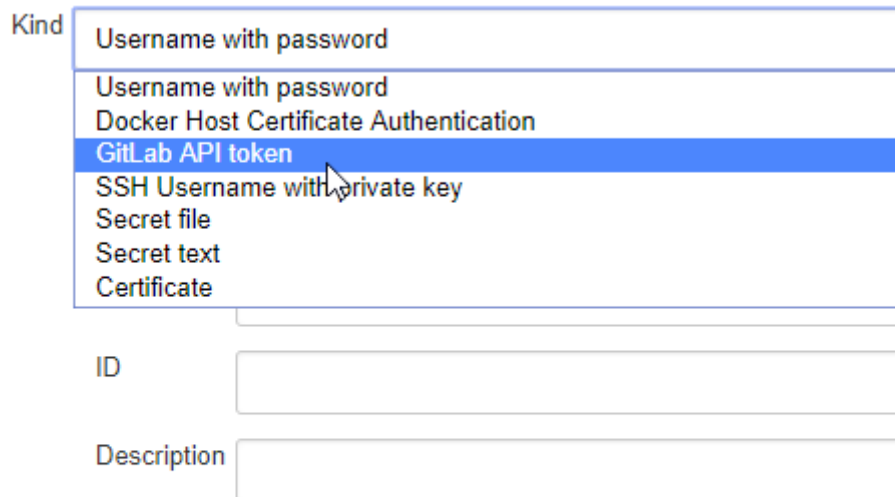
Make sure you save it - you won't be able to access it again.

As GitLab said, you should make sure you don't lose your token. Otherwise you would need to create a new one.

This will allow Jenkins to connect with right permissions to our GitLab server.

[[dsf-mirabaud-jenkins-gitlab-integration.asciidoc_3.-create-"gitlab-api"-token-credentials]] == 3.
Create "GitLab API" Token credentials

Thos credentials will use that token already generated in GitLab to connect once we declare the GitLab server in the Global Jenkins configuration. Obviously, those credentials must be **GitLab API token**-like.



Kind

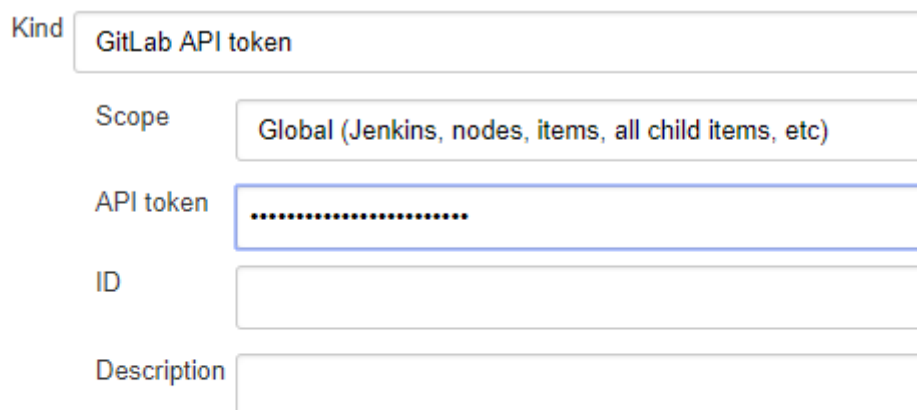
- Username with password
- Username with password
- Docker Host Certificate Authentication
- GitLab API token**
- SSH Username with private key
- Secret file
- Secret text
- Certificate

ID

Description

OK

Then, we add the generated token in the **API token** field:



Kind

GitLab API token

Scope

Global (Jenkins, nodes, items, all child items, etc)

API token

.....

ID







































































Description

OK

Look in your Global credentials if they had been correctly created:

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

	Name	Kind	Description	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	
	asciidoc[asciidoc] Jenkins user	Username with password	Username with password	

Icon: S M L

4. Create GitLab connection in Jenkins

Specify a GitLab connection in your Jenkins's **Manage Jenkins > Configure System** configuration. This will tell Jenkins where is our GitLab server, a user to access it from and so on.

You'll need to give it a name, for example, related with what this GitLab is dedicated for (specific clients, internal projects...). Then, the **Gitlab host URL** is just where your GitLab server is. If you have it locally, that field should look similar to:

- Connection name: **my-local-gitlab**
- Gitlab host URL: **http://localhost:[dsf-mirabaud-jenkins-gitLab-integration.asciidoc_PORT_NUMBER]**

Finally, we select our recently GitLab API token as credentials.

Gitlab

Enable authentication for '/project' end-point ☒

GitLab connections

Connection name

A name for the connection

Gitlab host URL

The complete URL to the Gitlab server (i.e. http://gitlab.org)

Credentials GitLab API token Add

API Token for accessing Gitlab

Success

Advanced...

Test Connection

Delete

Add

5. Jenkins Pipeline changes

5.1 Choose GitLab connection in Pipeline's General configuration

First, our pipeline should allow us to add a GitLab connection to connect to (the already created one).

☐ GitHub project

GitLab connection

GitLab Repository Name

In the case of the local example, could be like this:

- GitLab connection: `my-local-gitlab`
- GitLab Repository Name: `myusername/webhook-test` (for example)

5.2 Create a Build Trigger

1. You should already see your GitLab project's URL (as you stated in the General settings of the Pipeline).
2. Write `.*build.*` in the comment for triggering a build
3. Specify or filter the branch of your repo you want use as target. That means, whenever a git action is done to that branch (for exapmle, `master`), this Pipeline is going to be built.
4. Generate a Secret token (to be added in the yet-to-be-created GitLab webhook).

Build Triggers

☐ Build after other projects are built
☐ Build periodically
☒ Build when a change is pushed to GitLab. GitLab CI Service URL: 1

Enabled GitLab triggers

Push Events	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input type="checkbox"/>
Accepted Merge Request Events	<input type="checkbox"/>
Closed Merge Request Events	<input type="checkbox"/>
Rebuild open Merge Requests	Never
Comments	<input type="checkbox"/>
Comment (regex) for triggering a build	<input type="text"/> .*build.* 2

☒ Enable [ci-skip]
☒ Ignore WIP Merge Requests
☒ Set build description to build cause (eg. Merge request or Git Push)
☐ Build on successful pipeline events

Allowed branches

☐ Allow all branches to trigger this job
☒ Filter branches by name

Include 3
 ⚠ Cannot connect to GitLab to check whether selected branches exist. (show details)

Exclude

☐ Filter branches by regex
☐ Filter merge request by label

Secret token 4

6. GitLab Webhook

1. Go to your GitLab project's **Settings > Integration** section.
2. Add the path to your Jenkins Pipeline. Make sure you add **project** instead of **job** in the path.
3. Paste the generated Secret token of your Jenkins pipeline
4. Select your git action that will trigger the build.

The screenshot shows the GitLab interface for configuring webhooks. On the left sidebar, the 'Integrations' menu item is highlighted with a red circle labeled '1'. The main content area is titled 'Integrations' and includes a description of webhooks. The configuration fields are as follows:

- URL:** A text input field containing 'http://path/to/your/jenkins:[PORT_NUMBER]/project/name-of-the-pipeline' with a red circle labeled '2' next to it.
- Secret Token:** A text input field containing a masked token with a red circle labeled '3' next to it.
- Trigger:** A section with several checkboxes:
 - ☒ **Push events** (labeled with a red circle '4'): This URL will be triggered by a push to the repository.
 - ☐ **Tag push events**: This URL will be triggered when a new tag is pushed to the repository.
 - ☐ **Comments**: This URL will be triggered when someone adds a comment.
 - ☐ **Issues events**: This URL will be triggered when an issue is created/updated/merged.
 - ☐ **Confidential Issues events**: This URL will be triggered when a confidential issue is created/updated/merged.
 - ☐ **Merge Request events**: This URL will be triggered when a merge request is created/updated/merged.
 - ☐ **Job events**: This URL will be triggered when the job status changes.
 - ☐ **Pipeline events**: This URL will be triggered when the pipeline status changes.
 - ☐ **Wiki Page events**: This URL will be triggered when a wiki page is created/updated.
- SSL verification:** A checkbox labeled 'Enable SSL verification' which is checked.
- Add webhook:** A green button at the bottom of the configuration section.

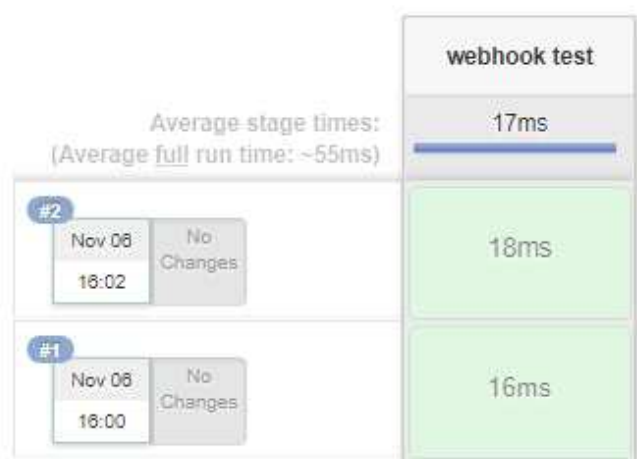
7. Results

After all those steps you should have a result similar to this in your Pipeline:

- Open Blue Ocean
- Full Stage View
- Pipeline Syntax

The 'Build History' section shows a list of builds for a pipeline. The first build, #2, is highlighted with a yellow background and labeled 'Started by GitLab push by devonfw'. The second build, #1, is labeled 'Started by GitLab push by devonfw'. At the bottom, there are links for 'RSS for all' and 'RSS for failures'.

Stage View



Permalinks

- [Last build \(#2\), 42 min ago](#)
- [Last stable build \(#2\), 42 min ago](#)
- [Last successful build \(#2\), 42 min ago](#)
- [Last completed build \(#2\), 42 min ago](#)

Enjoy the Continuous Integration! :)

6.2.7. Jenkins - Nexus integration

Nexus is used to both host dependencies for devonfw projects to download (common Maven ones, custom ones such as `ojdb` and even devonfw so-far-IP modules). Moreover, it will host our projects' build artifacts (`.jar`, `.war`, ...) and expose them for us to download, wget and so on. A team should have a bidirectional relation with its Nexus repository.

1. Jenkins credentials to access Nexus

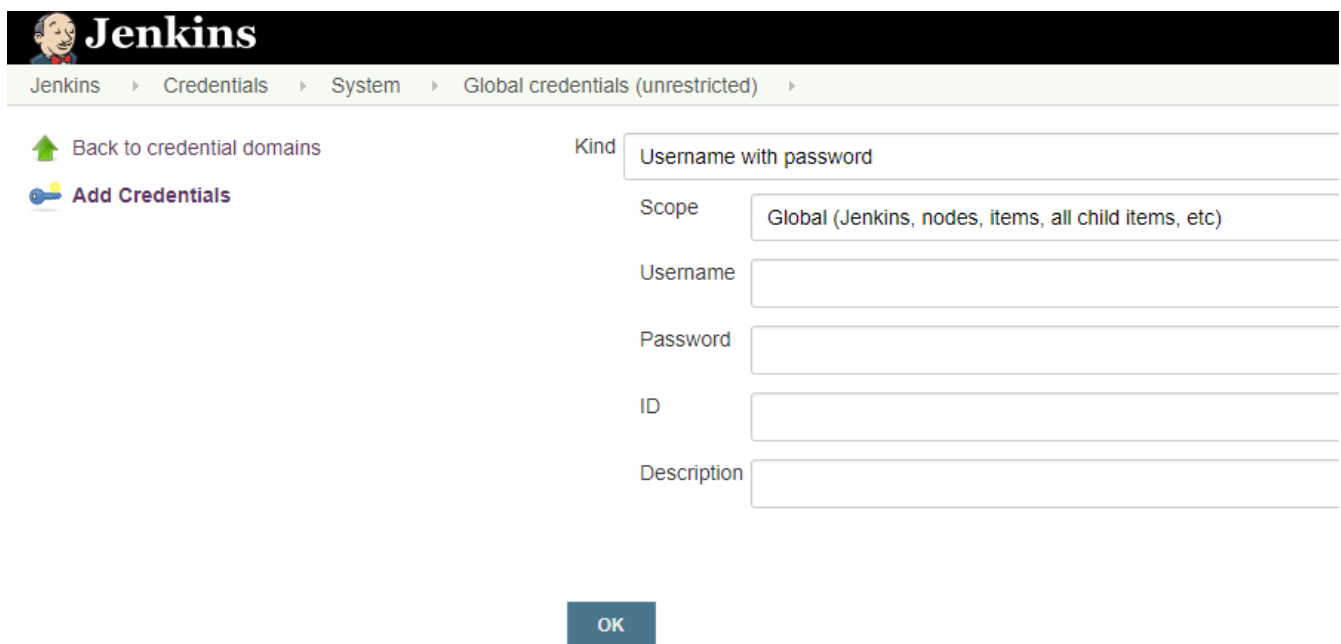
By default, when Nexus is installed, it contains 3 user credentials for different purposes. The admin ones look like this: `admin/admin123`. There are also other 2: `deployment/deployment123` and `T000`.

```
// ADD USER TABLE IMAGE FROM NEXUS
```

In this case, let's use the ones with the greater permissions: `admin/admin123`.

Go to **Credentials** > **System** (left sidebar of Jenkins) then to **Global credentials (unrestricted)** on the page table and on the left sidebar again click on **Add Credentials**.

This should be shown in your Jenkins:



The screenshot shows the Jenkins web interface. At the top is the Jenkins logo and name. Below it is a breadcrumb trail: Jenkins > Credentials > System > Global credentials (unrestricted). On the left sidebar, there is a link 'Back to credential domains' with a green arrow icon and a link 'Add Credentials' with a blue key icon. The main content area is a form for adding a new credential. It has a 'Kind' dropdown menu set to 'Username with password'. Below this are several input fields: 'Scope' (a dropdown menu set to 'Global (Jenkins, nodes, items, all child items, etc)'), 'Username', 'Password', 'ID', and 'Description'. At the bottom of the form is a blue 'OK' button.

Fill the form like this:

2. Jenkins Maven Settings

```
/${devonfw-dist-path}
  /software
    /maven
      /conf
        settings.xml
```

37

Type

Select the file type you want to create

☒ **Global Maven settings.xml**

A global maven settings.xml which can be referenced within Apache Maven jobs.

Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)

☐ **Maven settings.xml**

A settings.xml which can be referenced within Apache Maven jobs.

Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)

☐ **Json file**

a Json file

☐ **Maven toolchains.xml**

a toolchains.xml which can be referenced within Apache Maven jobs

☐ **Simple XML file**

a general xml file

☐ **Groovy file**

a reusable groovy script

☐ **Custom file**

a custom file (e.g. text or any other not yet available format)

☐ **Extended Email Publisher Groovy Template**

A Groovy template used by the Extended Email Publisher plugin to generate emails.

☐ **Extended Email Publisher Jelly Template**

A Jelly template used by the Extended Email Publisher plugin to generate emails.

☐ **Npm config file**

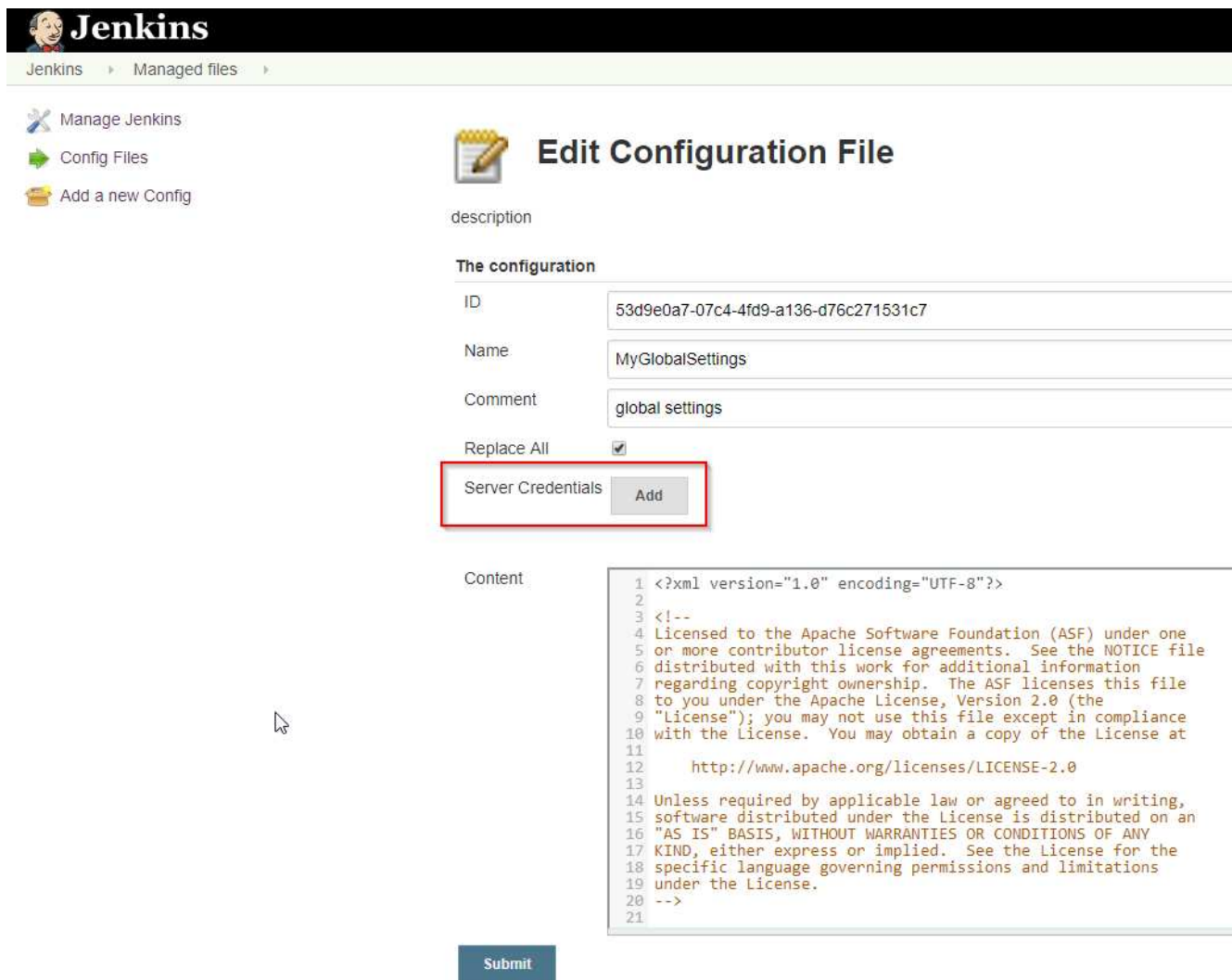
a npmrc config file (an ini-formatted list of *key = value* parameters)

ID 053e193c-e53e-4108-b390-933880a2c251

ID of the config file

Submit

The ID field will get automatically filled with a unique value if you don't set it up. No problems about that. Click on **Submit** and let's create some Servers Credentials:



Jenkins

Jenkins > Managed files >

- Manage Jenkins
- Config Files
- Add a new Config

Edit Configuration File

description

The configuration

ID	53d9e0a7-07c4-4fd9-a136-d76c271531c7
Name	MyGlobalSettings
Comment	global settings
Replace All	<input checked="" type="checkbox"/>

Server Credentials Add

Content

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4 Licensed to the Apache Software Foundation (ASF) under one
5 or more contributor license agreements. See the NOTICE file
6 distributed with this work for additional information
7 regarding copyright ownership. The ASF licenses this file
8 to you under the Apache License, Version 2.0 (the
9 "License"); you may not use this file except in compliance
10 with the License. You may obtain a copy of the License at
11
12     http://www.apache.org/licenses/LICENSE-2.0
13
14 Unless required by applicable law or agreed to in writing,
15 software distributed under the License is distributed on an
16 "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
17 KIND, either express or implied. See the License for the
18 specific language governing permissions and limitations
19 under the License.
20 -->
21
  
```

Submit

Those **Server Credentials** will allow Jenkins to access to the different repositories/servers that are going to be declared afterwards.

Let's create 4 server credentials.

- **my.nexus**: Will serve as general profile for **Maven**.
- **my nexus.releases**: When a **mvn deploy** process is executed, this will tell **Maven** where to push **releases** to.
- **my nexus.snapshots**: The same as before, but with **snapshots** instead.
- **my nexus.central**: Just in case we want to install an specific dependency that is not by default in the Maven Central repository (such as **ojdbc**), Maven will point to it instead.

Server Credentials	
Serverid	my.nexus
Credentials	admin/***** (Admin credentials to access Nexus) Add
Serverid	my.nexus.releases
Credentials	admin/***** (Admin credentials to access Nexus) Add
Serverid	my.nexus.snapshots
Credentials	admin/***** (Admin credentials to access Nexus) Add
Serverid	my.nexus.central
Credentials	- current - Add

A more or less complete Jenkins Maven settings would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <mirrors>
    <mirror>
      <id>my.nexus.central</id>
      <mirrorOf>central</mirrorOf>
      <name>central</name>
      <url>http://${URL-TO-YOUR-NEXUS-REPOS}/central</url>
    </mirror>
  </mirrors>

  <profiles>
    <profile>
      <id>my.nexus</id>
      <!-- 3 REPOS ARE DECLARED -->
      <repositories>
        <repository>
          <id>my.nexus.releases</id>
          <name>my.nexus Releases</name>
          <url>http://${URL-TO-YOUR-NEXUS-REPOS}/releases</url>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
          </releases>
          <snapshots>
            <enabled>false</enabled>
            <updatePolicy>always</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
    </profile>
  </profiles>
</settings>
```

```

        </snapshots>
    </repository>
    <repository>
        <id>my nexus.snapshots</id>
        <name>my nexus Snapshots</name>
        <url>http://${URL-TO-YOUR-NEXUS-REPOS}/snapshots</url>
        <releases>
            <enabled>false</enabled>
            <updatePolicy>always</updatePolicy>
        </releases>
        <snapshots>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
        </snapshots>
    </repository>
</repositories>
<pluginRepositories>
    <pluginRepository>
        <id>public</id>
        <name>Public Repositories</name>
        <url>http://${URL-TO-YOUR-
NEXUS}/nexus/content/groups/public/</url>
        <releases>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
        </releases>
        <snapshots>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
</profile>
</profiles>
<!-- HERE IS WHERE WE TELL MAVEN TO CHOOSE THE my.nexus PROFILE -->
<activeProfiles>
    <activeProfile>my.nexus</activeProfile>
</activeProfiles>
</settings>

```

3. Use it in Jenkins Pipelines

6.2.8. Jenkins - SonarQube integration

First thing is installing both tools by, for example, Docker or Docker Compose. Then, we have to think about how they should collaborate to create a more efficient Continuous Integration process.

Once our project's pipeline is triggered (it could also be triggered in a fancy way, such as when a merge to the `develop` branch is done).

1. Jenkins SonarQube plugin

Typically in those integration cases, Jenkins plug-in installations become a **must**. Let's look for some available SonarQube plug-in(s) for Jenkins:

Filter:

Updates	Available	Installed	Advanced	
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	jQuery plugin This plugin provides an stable version of jQuery Javascript Library	1.12.4-0		<button>Uninstall</button>
<input checked="" type="checkbox"/>	Pipeline: Groovy Pipeline execution engine based on continuation passing style transformation of Groovy scripts.	2.41		<button>Uninstall</button>
<input checked="" type="checkbox"/>	SonarQube Scanner for Jenkins This plugin allows an easy integration of SonarQube , the open source platform for Continuous Inspection of code quality.	2.6.1		<button>Uninstall</button>

2. SonarQube token

Once installed let's create a **token** in SonarQube so that Jenkins can communicate with it to trigger their Jobs. Once we install SonarQube in our CI/CD machine (ideally a remote machine) let's login with **admin/admin** credentials:

Log In to SonarQube

Log in [Cancel](#)

Afterwards, SonarQube itself asks you to create this token we talked about (the name is up to you):

Welcome to SonarQube!

Want to quickly analyze a first project? Follow these 2 easy steps.

1

Provide a token

☒ Generate a token

devonfw-ci-token

Generate

☐ Use existing token

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

Then a token is generated:

Welcome to SonarQube!

Want to quickly analyze a first project? Follow these 2 easy steps.

1

Provide a token

devonfw-ci-token:  ✖

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

Continue

You click in "continue" and the token's generation is completed:

✓ devonfw-ci-token: 

3. Jenkins SonarQube Server setup

Now we need to tell Jenkins where is SonarQube and how to communicate with it. In [Manage Jenkins > Configure Settings](#). We add a name for the server (up to you), where it is located (URL), version and the Server authentication token created in point 2.

SonarQube servers

Environment variables

☐ Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

SonarQube

Server URL

http://localhost:9000

Default is http://localhost:9000

Server version

5.3 or higher

Configuration fields depend on the SonarQube server version.

Server authentication token

.....

4. Jenkins SonarQube Scanner

Install a SonarQube Scanner as a Global tool in Jenkins to be used in the project's pipeline.

SonarQube Scanner

SonarQube Scanner installations



SonarQube Scanner

Name

SonarQube scanner



Install automatically



Install from Maven Central

Version

SonarQube Scanner 3.0.3.778

Delete Installer

5. Pipeline code

Last step is to add the SonarQube process in our project's Jenkins pipeline. The following code will trigger a SonarQube process that will evaluate our code's quality looking for bugs, duplications, and so on.

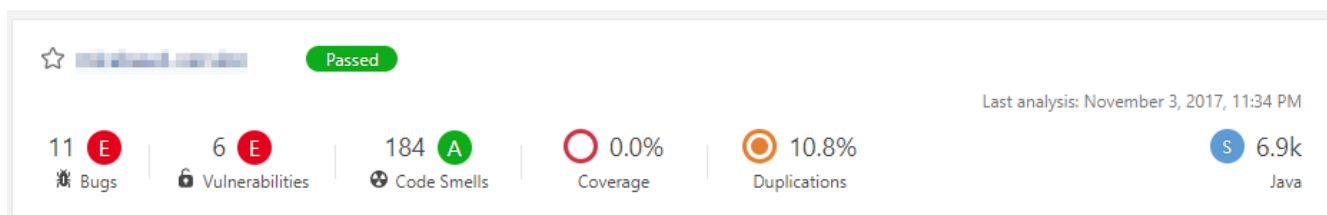
```
stage 'SonarQube Analysis'
def scannerHome = tool 'SonarQube scanner';
sh "${scannerHome}/bin/sonar-scanner \
  -Dsonar.host.url=http://url-to-your-sq-server:9000/ \
  -Dsonar.login=[SONAR_USER] -Dsonar.password=[SONAR_PASS] \
  -Dsonar.projectKey=[PROJECT_KEY] \
  -Dsonar.projectName=[PROJECT_NAME] \
  -Dsonar.projectVersion=[PROJECT_VERSION] \
  -Dsonar.sources=. -Dsonar.java.binaries=. \
  -Dsonar.java.source=1.8 -Dsonar.language=java"
```

6. Results

After all this, you should end up having something like this in Jenkins:



And in SonarQube:



7. Changes in a devonfw project to execute SonarQube tests with Coverage

The plugin used to have Coverage reports in the SonarQube for devonfw projects is **Jacoco**. There are some changes in the project's parent `pom.xml` that are mandatory to use it.

Inside of the `<properties>` tag:

```

<properties>

    (...)

    <sonar.jacoco.version>3.8</sonar.jacoco.version>
    <sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
    <sonar.core.codeCoveragePlugin>jacoco</sonar.core.codeCoveragePlugin>
    <sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>
    <sonar.language>java</sonar.language>
    <sonar.java.source>1.7</sonar.java.source>
    <sonar.junit.reportPaths>target/surefire-reports</sonar.junit.reportPaths>
    <sonar.jacoco.reportPaths>target/jacoco.exec</sonar.jacoco.reportPaths>
    <sonar.sourceEncoding>UTF-8</sonar.sourceEncoding>
    <sonar.exclusions>
        **/generated-sources/**/*,
        **io/oasp/mirabaud/general/**/*,
        **/*Dao.java,
        **/*Entity.java,
        **/*Cto.java,
        **/*Eto.java,
        **/*SearchCriteriaTo.java,
        **/*management.java,
        **/*SpringBootApplication.java,
        **/*SpringBootBatchApp.java,
        **/*.xml,
        **/*.jsp
    </sonar.exclusions>
    <sonar.coverage.exclusions>
        **io/oasp/mirabaud/general/**/*,
        **/*Dao.java,
        **/*Entity.java,
        **/*Cto.java,
        **/*Eto.java,
        **/*SearchCriteriaTo.java,
        **/*management.java,
        **/*SpringBootApplication.java,
        **/*SpringBootBatchApp.java,
        **/*.xml,
        **/*.jsp
    </sonar.coverage.exclusions>
    <sonar.host.url>http://${YOUR_SONAR_SERVER_URL}</sonar.host.url>
    <jacoco.version>0.7.9</jacoco.version>

    <war.plugin.version>3.2.0</war.plugin.version>
    <assembly.plugin.version>3.1.0</assembly.plugin.version>
</properties>

```

Of course, those **sonar** and **sonar.coverage** can/must be changed to fit with other projects.

Now add the **Jacoco Listener** as a dependency:

```

<dependencies>
  <dependency>
    <groupId>org.sonarsource.java</groupId>
    <artifactId>sonar-jacoco-listeners</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

```

Plugin Management declarations:

```

<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.2</version>
    </plugin>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>${jacoco.version}</version>
    </plugin>
  </plugins>
</pluginManagement>

```

Plugins:

```

<plugins>

  (...)

  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.20.1</version>
    <configuration>
      <argLine>-XX:-UseSplitVerifier -Xmx2048m ${surefireArgLine}</argLine>
      <testFailureIgnore>>false</testFailureIgnore>
      <useFile>>false</useFile>
      <reportsDirectory>
        ${project.basedir}/${sonar.junit.reportPaths}</reportsDirectory>
      <argLine>${jacoco.agent.argLine}</argLine>
      <excludedGroups>${oasp.test.excluded.groups}</excludedGroups>
      <alwaysGenerateSurefireReport>true</alwaysGenerateSurefireReport>
      <aggregate>true</aggregate>
      <properties>
        <property>
          <name>listener</name>

```

```

        <value>org.sonar.java.jacoco.JUnitListener</value>
      </property>
    </properties>
  </configuration>
</plugin>
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <configuration>
    <argLine>-Xmx128m</argLine>
    <append>true</append>
    <propertyName>jacoco.agent.argLine</propertyName>
    <destFile>${sonar.jacoco.reportPath}</destFile>
    <excludes>
      <exclude>**/generated-sources/**/*,</exclude>
      <exclude>**io/oasp/${PROJECT_NAME}/general/**/*</exclude>
      <exclude>**/*Dao.java</exclude>
      <exclude>**/*Entity.java</exclude>
      <exclude>**/*Cto.java</exclude>
      <exclude>**/*Eto.java</exclude>
      <exclude>**/*SearchCriteriaTo.java</exclude>
      <exclude>**/*management.java</exclude>
      <exclude>**/*SpringBootApplication.java</exclude>
      <exclude>**/*SpringBootBatchApp.java</exclude>
      <exclude>**/*.class</exclude>
    </excludes>
  </configuration>
  <executions>
    <execution>
      <id>prepare-agent</id>
      <phase>initialize</phase>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
      <configuration>
        <destFile>${sonar.jacoco.reportPath}</destFile>
        <append>true</append>
      </configuration>
    </execution>
    <execution>
      <id>report-aggregate</id>
      <phase>verify</phase>
      <goals>
        <goal>report-aggregate</goal>
      </goals>
    </execution>
    <execution>
      <id>jacoco-site</id>
      <phase>verify</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

```
        </goals>
      </execution>
    </executions>
  </plugin>
</plugins>
```

Jenkins SonarQube execution

If the previous configuration is already setup, once Jenkins execute the sonar maven plugin, it will automatically execute coverage as well.

This is an example of a block of code from a devonfw project's [Jenkinsfile](#):

```
withMaven(globalMavenSettingsConfig: 'YOUR_GLOBAL_MAVEN_SETTINGS', jdk: 'OpenJDK
1.8', maven: 'Maven_3.3.9') {
    sh "mvn sonar:sonar -Dsonar.login=[USERNAME] -Dsonar.password=[PASSWORD]"
}
```

6.3. OKD (*OpenShift Origin*)

6.3.1. What is OKD

OKD is a distribution of Kubernetes optimized for continuous application development and multi-tenant deployment. OKD is the upstream Kubernetes distribution embedded in Red Hat OpenShift.

OKD embeds Kubernetes and extends it with security and other integrated concepts. OKD is also referred to as Origin in github and in the documentation.

OKD provides a complete open source container application platform. If you are looking for enterprise-level support, or information on partner certification, Red Hat also offers [Red Hat OpenShift Container Platform](#).

Continue reading...

- [How to install Openshift Origin](#)
- [Initial setup](#)
 - [s2i](#)
 - [templates](#)
 - [Customize Openshift](#)
 - [Customize icons](#)
 - [Customize catalog](#)

6.3.2. Install OKD (*Openshift Origin*)

Pre-requisites

Install docker

<https://docs.docker.com/engine/installation/linux/docker-ce/debian/#set-up-the-repository>

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

Download Openshift Origin Client

Download Openshift Origin Client from [here](#)

When the download it's complete, only extract it on the directory that you want, for example `/home/administrador/oc`

Add oc to path

```
$ export PATH=$PATH:/home/administrador/oc
```

Install Openshift Cluster

Add the insecure registry

Create file `/etc/docker/daemon.json` with the next content:

```
{
  "insecure-registries" : [ "172.30.0.0/16" ]
}
```

Download docker images for openshift

```
$ oc cluster up
```

Install Oc Cluster Wrapper

To manage easier the cluster persistent, we are going to use oc cluster wrapper.

```
cd /home/administrador/oc
wget https://raw.githubusercontent.com/openshift-evangelists/oc-cluster-
wrapper/master/oc-cluster
```

`oc-cluster up devonfw-shop-floor --public-hostname X.X.X.X`

Configure iptables

We must create iptables rules to allow traffic from other machines.

```
- The next commands it's to let all traffic, don't do it on a real server.

- $ iptables -F
- $ iptables -X
- $ iptables -t nat -F
- $ iptables -t nat -X
- $ iptables -t mangle -F
- $ iptables -t mangle -X
- $ iptables -P INPUT ACCEPT
- $ iptables -P OUTPUT ACCEPT
- $ iptables -P FORWARD ACCEPT
```

6.3.3. How to use Oc Cluster Wrapper

With oc cluster wrapper we could have a different clusters with different context.

Cluster up

```
$ oc-cluster up devonfw-shop-floor --public-hostname X.X.X.X
```

Cluster down

```
$ oc-cluster down
```

Use non-persistent cluster

```
oc cluster up --image openshift/origin --public-hostname X.X.X.X --routing-suffix
apps.X.X.X.X.nip.io
```

6.3.4. devonfw Openshift Origin Initial Setup

This is a scripts to customize an Openshift cluster to be a devonfw Openshift.

How to use

Prerequisite: Customize Openshift

devonfw Openshift Origin use custom icons, and we need to add it to openshift. More information:

- [Customize Openshift](#)

Script initial-setup

Download [this](#) script and execute it.

More information about what this script does [here](#).

Known issues

Failed to push image

If you receive an error like this:

```
error: build error: Failed to push image: After retrying 6 times, Push image still
failed due to error: Get http://172.30.1.1:5000/v2/: dial tcp 172.30.1.1:5000:
getsockopt: connection refused
```

It's because the registry isn't working, go to openshift console and enter into the **default** project <https://x.x.x.x:8443/console/project/default/overview> and you must see two resources, **docker-registry** and **router** they must be running. If they don't work, try to deploy them and look at the logs what is happen.

6.3.5. s2i devonfw

This are the s2i source and templates to build s2i images. It provides OpenShift builder images for components of the devonfw (at this moment only for angular and java).

This work is totally based on the implementation of [Michael Kuehl](#) from RedHat for Oasp s2i.

All this information is used as a part of the [initial setup](#) for openshift.

Previous setup

In order to build all of this, it will be necessary, first, to have a running OpenShift cluster. How to install it [here](#).

Usage

Before using the builder images, add them to the OpenShift cluster.

Deploy the Source-2-Image builder images

First, create a dedicated **devonfw** project as admin.

```
$ oc new-project devonfw --display-name='devonfw' --description='devonfw Application
Standar Platform'
```

Now add the builder image configuration and start their build.

```
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-  
floor/master/dsf4openshift/openshift-devonfw-deployment/s2i/java/s2i-devonfw-java-  
imagestream.json --namespace=devonfw  
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-  
floor/master/dsf4openshift/openshift-devonfw-deployment/s2i/angular/s2i-devonfw-  
angular-imagestream.json --namespace=devonfw  
oc start-build s2i-devonfw-java --namespace=devonfw  
oc start-build s2i-devonfw-angular --namespace=devonfw
```

Make sure other projects can access the builder images:

```
oc policy add-role-to-group system:image-puller system:authenticated  
--namespace=devonfw
```

That's all !

Deploy devonfw templates

Now, it's time to create devonfw templates to use this s2i and add it to the browse catalog. More information [here](#).

Build All

Use [this](#) script to automatically install and build all image streams. The script also creates templates devonfw-angular and devonfw-java inside the project 'openshift' to be used by everyone.

1. Open a bash shell as Administrator
2. Execute shell file:

```
$ /PATH/TO/BUILD/FILE/initial-setup.sh
```

More information about what this script does [here](#).

Links & References

This is a list of useful articles etc I found while creating the templates.

- [Template Icons](#)
- [Red Hat Cool Store Microservice Demo](#)
- [Openshift Web Console Customization](#)

6.3.6. devonfw templates

This are the devonfw templates to build devonfw apps for Openshift using the s2i images. They are based on the work of Mickuehl in Oasp templates/mythaistar for deploy My Thai Star.

- Inside the `example-mythaistar` we have an example to deploy My Thai Star application using devonfw templates.

All this information is used as a part of the [initial setup](#) for openshift.

How to use

Previous requirements

Deploy the Source-2-Image builder images

Remember that this templates need a build image from `s2i-devonfw-angular` and `s2i-devonfw-java`. More information:

- [Deploy the Source-2-Image builder images](#).

Customize Openshift

Remember that this templates also have a custom icons, and to use it, we must modify the `master-config.yml` inside openshift. More information:

- [Customize Openshift](#).

Deploy devonfw templates

Now, it's time to create devonfw templates to use this s2i and add it to the browse catalog.

To let all user to use this templates in all openshift projects, we should create it in an openshift namespace. To do that, we must log in as an admin.

```
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-devonfw-deployment/templates/devonfw-java-template.json --namespace=openshift
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-devonfw-deployment/templates/devonfw-angular-template.json --namespace=openshift
```

When it finish, remember to logout as an admin and enter with our normal user.

```
$ oc login
```

How to use devonfw templates in openshift

To use this templates with openshift, we can override any parameter values defined in the file by adding the `--param-file=paramfile` option.

This file must be a list of `<name>=<value>` pairs. A parameter reference may appear in any text field inside the template items.

The parameters that we must override are the following

```
$ cat paramfile
APPLICATION_NAME=app-Name
APPLICATION_GROUP_NAME=group-Name
GIT_URI=Git uri
GIT_REF=master
CONTEXT_DIR=/context
```

The following parameters are optional

```
$ cat paramfile
APPLICATION_HOSTNAME=Custom hostname for service routes. Leave blank for default
hostname, e.g.: <application-name>.<project>.<default-domain-suffix>,
# Only for angular
REST_ENDPOINT_URL=The URL of the backend's REST API endpoint. This can be declared
after,
REST_ENDPOINT_PATTERN=The pattern URL of the backend's REST API endpoint that must
be modify by the REST_ENDPOINT_URL variable,
```

For example, to deploy My Thai Star Java

```
$ cat paramfile
APPLICATION_NAME="mythaistar-java"
APPLICATION_GROUP_NAME="My-Thai-Star"
GIT_URI="https://github.com/oasp/my-thai-star.git"
GIT_REF="develop"
CONTEXT_DIR="/java/mtsj"

$ oc new-app --template=devonfw-java --namespace=mythaistar --param-file=paramfile
```

6.3.7. Customize Openshift Origin for devonfw

This is a guide to customize Openshift cluster.

Images Styles

The icons for templates must measure the same as below or the images don't show right:

- **Openshift logo**: 230px x 40px.
- **Template logo**: 50px x 50px.
- **Category logo**: 110px x 36px.

How to use

To use it, we need to enter in openshift as an admin and use the next command:

```
$ oc login

$ oc edit configmap/webconsole-config -n openshift-web-console
```

After this, we can see in our shell the `webconsole-config.yaml`, we only need to navigate until **extensions** and add the url for our own `css` in the **stylesheetURLs** and `javascript` in the **scriptURLs** section.

IMPORTANT: Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. Scripts must be served with Content-Type: application/javascript and stylesheets with Content-Type: text/css.

In git repositories, the content type of raw is text/plain. You can use [rawgit](#) to convert a raw from a git repository to the correct content type.

Example:

```
webconsole-config.yaml: |
  [...]
  extensions:
    scriptURLs:
      - https://cdn.rawgit.com/devonfw/devonfw-shop-
        floor/master/dsf4openshift/openshift-cluster-setup/initial-
        setup/customize0openshift/scripts/catalog-categories.js
    stylesheetURLs:
      - https://cdn.rawgit.com/devonfw/devonfw-shop-
        floor/master/dsf4openshift/openshift-cluster-setup/initial-
        setup/customize0openshift/stylesheet/icons.css
  [...]
```

More information

- [Customize icons](#) for Openshift.
- [Customize catalog](#) for Openshift.
- [Openshift docs](#) about customization.

Old versions

- Customize Openshift for [version 3.7](#).

How to add Custom Icons inside openshift

This is a guide to add custom icons into an Openshift cluster.

[Here](#) we can find an `icons.css` example to use the devonfw icons.

Images Styles

The icons for templates must measure the same as below or the images don't show right:

- **Openshift logo:** 230px x 40px.
- **Template logo:** 50px x 50px.
- **Category logo:** 110px x 36px.

Create a css

Custom logo for openshift cluster

For this example, we are going to call the css icons.css but you can call as you wish. Openshift cluster draw their icon by the id header-logo, then we only need to add to our icons.css the next Style Attribute ID

```
#header-logo {  
  background-image: url("https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-cluster-setup/initial-setup/customizeOpenshift/images/devonfw-openshift.png");  
  width: 230px;  
  height: 40px;  
}
```

Custom icons for templates

To use a custom icon to a template openshift use a class name. Then, we need to insert inside our icons.css the next Style Class

```
.devonfw-logo {  
  background-image: url("https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-cluster-setup/initial-setup/customizeOpenshift/images/devonfw.png");  
  width: 50px;  
  height: 50px;  
}
```

To show that custom icon on a template, we only need to write the name of our class in the tag "iconClass" of our template.

```
{
  ...
  "items": [
    {
      ...
      "metadata": {
        ...
        "annotations": {
          ...
          "iconClass": "devonfw-logo",
          ...
        }
      },
      ...
    }
  ]
}
```

Use our own css inside openshift

To do that, we need to enter in openshift as an admin and use the next command:

```
$ oc login
$ oc edit configmap/webconsole-config -n openshift-web-console
```

After this, we can see in our shell the webconsole-config.yaml, we only need to navigate until **extensions** and add the url for our own **css** in the **stylesheetURLs** section.

IMPORTANT: Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. stylesheets must be served with Content-Type: text/css.

In git repositories, the content type of raw is text/plain. You can use [rawgit](#) to convert a raw from a git repository to the correct content type.

Example:

```
webconsole-config.yaml: |
  [...]
  extensions:
    stylesheetURLs:
      - https://cdn.rawgit.com/devonfw/devonfw-shop-
        floor/master/dsf4openshift/openshift-cluster-setup/initial-
        setup/customize0penshift/stylesheet/icons.css
  [...]
```

How to add custom catalog categories inside openshift

This is a guide to add custom **Catalog Categories** into an Openshift cluster.

[Here](#) we can find a catalog-categories.js example to use the devonfw catalog categories.

Create a scrip to add custom lengauges and custom catalog categories

Custom language

For this example, we are going add a new language into the languages category. To do that we must created a script and we named as catalog-categories.js

```
// Find the Languages category.
var category = _.find(window.OPENSIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES,
                      { id: 'languages' });
// Add Go as a new subcategory under Languages.
category.subCategories.splice(2,0,{ // Insert at the third spot.
  // Required. Must be unique.
  id: "devonfw-languages",
  // Required.
  label: "devonfw",
  // Optional. If specified, defines a unique icon for this item.
  icon: "devonfw-logo-language",
  // Required. Items matching any tag will appear in this subcategory.
  tags: [
    "devonfw",
    "devonfw-angular",
    "devonfw-java"
  ]
});
```

Custom category

For this example, we are going add a new category into the category tab. To do that we must created a script and we named as catalog-categories.js


```
// Add a Featured category as the first category tab.
window.OPENSIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES.unshift({
  // Required. Must be unique.
  id: "devonfw-featured",
  // Required
  label: "devonfw",
  subCategories: [
    {
      // Required. Must be unique.
      id: "devonfw-languages",
      // Required.
      label: "devonfw",
      // Optional. If specified, defines a unique icon for this item.
      icon: "devonfw-logo-language",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "devonfw",
        "devonfw-angular",
        "devonfw-java"
      ]
    }
  ]
});
```

Use our own javascript inside openshift

To do that, we need to enter in openshift as an admin and use the next command:

```
$ oc login
$ oc edit configmap/webconsole-config -n openshift-web-console
```

After this, we can see in our shell the webconsole-config.yaml, we only need to navigate untill **extensions** and add the url for our own **javascript** in the **scriptURLs** section.

IMPORTANT: Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. Scripts must be served with Content-Type: application/javascript.

In git repositories, the content type of raw is text/plain. You can use [rawgit](#) to convert a raw from a git repository to the correct content type.

Example:

```
webconsole-config.yaml: |
  [...]
  extensions:
    scriptURLs:
      - https://cdn.rawgit.com/devonfw/devonfw-shop-
        floor/master/dsf4openshift/openshift-cluster-setup/initial-
        setup/customizeOpenshift/scripts/catalog-categories.js
  [...]
```

Customize Openshift Origin v3.7 for devonfw

This is a guide to customize Openshift cluster. For more information read the next:

- [Openshift docs customization](#) for the version 3.7.

Images Styles

The icons for templates must measure the same as below or the images don't show right:

- **Openshift logo:** 230px x 40px.
- **Template logo:** 50px x 50px.
- **Category logo:** 110px x 36px.

Quick Use

This is a quick example to add custom icons and categories inside openshift.

To modify the icons inside openshift, we must to modify our master-config.yaml of our openshift cluster. This file is inside the openshift container and to obtain a copy of it, we must to know what's our openshift container name.

Obtain the master-config.yaml of our openshift cluster

Obtain the name of our openshift container

To obtain it, we can know it executing the next:

```
$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND
83a4e3acda5b	openshift/origin:v3.7.0	"/usr/bin/openshift ..." 6 days ago Up 6 days origin

Here we can see that the name of the container is origin. Normally the container it's called as origin.

Copy the master-config.yaml of our openshift container to our directory

This file is inside the openshift container in the next directory: `/var/lib/origin/openshift.local.config/master/master-config.yaml` and we can copy it with the next command:

```
$ docker cp origin:/var/lib/origin/openshift.local.config/master/master-config.yaml ./
```

Now we have a file with the configuration of our openshift cluster.

Copy all customize files inside the openshift container

To use our customization of devonfw Openshift, we need to copy our files inside the openshift container.

To do this we need to copy the images, scripts and stylesheets from [here](#) inside openshift container, for example, we could put it all inside a folder called `openshift.local.devonfw`. On the step one we obtain the name of this container, for this example we assume that it's called `origin`. Then our images are located inside openshift container and we can see an access it in `/var/lib/origin/openshift.local.devonfw/images`.

```
$ docker cp ./openshift.local.devonfw origin:/var/lib/origin/
```

Edit and copy the master-config.yaml to use our customize files

The master-config.yaml have a sections to charge our custom files. All this sections are inside the `assetConfig` and their names are the next:

- The custom stylesheets are into `extensionStylesheets`.
- The custom scripts are into `extensionScripts`.
- The custom images are into `extensions`.

To use all our custom elements only need to add the directory routes of each element in their appropriate section of the master-config.yaml

```
...
assetConfig:
  ...
  extensionScripts:
    - /var/lib/origin/openshift.local.devonfw/scripts/catalog-categories.js
  extensionStylesheets:
    - /var/lib/origin/openshift.local.devonfw/stylesheet/icons.css
  extensions:
    - name: images
      sourceDirectory: /var/lib/origin/openshift.local.devonfw/images
  ...
...
```

Now we only need to copy that master-config.yaml inside openshift, and restart it to load the new configuration. To do that execute the next:

```
$ docker cp ./master-config.yaml
origin:/var/lib/origin/openshift.local.config/master/master-config.yaml
```

To re-start openshift do `oc cluster down` and start again your persistent openshift cluster.

More information

- [Customize icons](#) for Openshift.
- [Customize catalog](#) for Openshift.
- [Openshift docs](#) about customization.

1.My Thai Star – Agile Framework

Table of Contents

1.1 Team Setup	1
1.2 Scrum events	1
Sprint Planning	1
Sprint Review	2
Sprint Retrospective	2
Daily Standups	2
Backlog refinement	2
1.3 Establish Product Backlog	2
2. My Thai Star – Agile Diary	2
24.03.2017 Sprint 1 Planning	3
27.04.2017 Sprint 1 Review	3
03.05.2017 Sprint 2 Planning	3
01.06.2017 Sprint 2 Review	4

Sprint Review

The Sprint Review meetings are time boxed to one hour for the four week Sprint. Within the Sprint Review meeting the team plans to do a retrospective of the finished Sprint. As well as it is done during the Sprint Planning the team receives support from Devon colleagues.

Sprint Retrospective

For this project the team aligned on not having a specific Sprint Retrospective meeting. The team is going to have a retrospective of a finished Sprint during the Sprint Review.

Daily Standups

The team aligned on having two weekly Standup meetings instead of a Daily Standup meeting. In comparison with the time boxed length of 15mins described in the CAF for this project the team extended the Standup meeting to 30mins. The content of the meetings remains the same.

Backlog refinement

The team decided that the backlog refinement meeting is part of the Sprint Planning meeting.

1.3 Establish Product Backlog

For the My Thai Stair project the team decided on using the Jira agile documentation which is one of the widely used agile tools. Jira is equipped with several of useful tools regarding the agile software development (e.g. Scrum-Board). One of the big advantages of Jira are the extensive configuration and personalization possibilities. With having a list of the Epics and User Stories for the My Thai Star development in GitHub, the team transferred the User Stories into the Jira backlog as it is shown in the screenshot below. All User Stories are labeled colorfully with the related Epic which shapes the backlog in clearly manner.

[\[Screenshot of planningpoker.com\]](#) | *images/methodology_2.png*

Figure 3. Screenshot of the Jira backlog during Sprint 2

We decided on working with Subtask as a single user story comprised a number of single and separated tasks. Another benefit of working with subtask is that every single subtask can be assigned to a single team member whereas a user story can only be assigned to one team member. By picking single subtask the whole process of a user story is better organized.

[\[Screenshot of Subtasks\]](#) | *images/methodology_3.png*

Figure 4. Screenshots of Subtasks during Sprint 2

2.My Thai Star – Agile Diary

In parallel to the Diary Ideation we use this Agile Diary to document our Scrum events. The target

of this diary is to describe the differences to the Scrum methodology as well as specific characteristics of the project. We also document the process on how we approach the Scrum methodology over the length of the project.

24.03.2017 Sprint 1 Planning

Within the Sprint 1 Planning we used planning poker.com for the estimation of the user stories. The estimation process usually is part of the backlog refinement meeting. Regarding the project circumstances we decided to estimate the user stories during the Sprint Planning. Starting the estimation process we noticed that we had to align our interpretation of the estimation effort as these story points are not equivalent to a certain time interval. The story points are relative values to compare the effort of the user stories. With this in mind we proceeded with the estimation of the user stories. We decided to start Sprint 1 with the following user stories and the total amount of 37 story points: • ICSDSHOW-2 Create invite for friends (8 Story Points) • ICSDSHOW-4 Create reservation (3) • ICSDSHOW-5 Handle invite (3) • ICSDSHOW-6 Revoke accepted invite (5) • ICSDSHOW-9 Cancel invite (3) • ICSDSHOW-11 Filter menu (5) • ICSDSHOW-12 Define order (5) • ICSDSHOW-13 Order the order (5) As the Sprint Planning is time boxed to one hour we managed to hold this meeting within this time window.

27.04.2017 Sprint 1 Review

During the Sprint 1 Review we had a discussion about the data model proposal. For the discussion we extended this particular Review meeting to 90min. As this discussion took almost 2/3 of the Review meeting we only had a short time left for our review of Sprint 1. For the following scrum events we decided to focus on the primary target of these events and have discussions needed for alignments in separate meetings. Regarding the topic of splitting user stories we had the example of a certain user story which included a functionality of a twitter integration (ICSDSHOW-17 User Profile and Twitter integration). As the twitter functionality could not have been implemented at this early point of time we thought about cutting the user story into two user stories. We aligned on mocking the twitter functionality until the dependencies are developed in order to test the components. As this user story is estimated with 13 story points it is a good example for the question whether to cut a user story into multiple user stories or not. Unfortunately not all user stories of Sprint 1 could have been completed. Due this situation we discussed on whether pushing all unfinished user stories into the status done or moving them to Sprint 2. We aligned on transferring the unfinished user stories into the next Sprint. During the Sprint 1 the team underestimated that a lot of holidays crossed the Sprint 1 goals. As taking holidays and absences of team members into consideration is part of a Sprint Planning we have a learning effect on setting a Sprint Scope.

03.05.2017 Sprint 2 Planning

As we aligned during the Sprint 1 Review on transferring unfinished user stories into Sprint 2 the focus for Sprint 2 was on finishing these transferred user stories. During our discussion on how many user stories we could work on in Sprint 2 we needed to remind ourselves that the overall target is to develop an example application for the DevonFW. Considering this we aligned on a clear target for Sprint 2: To focus on finishing User Stories as we need to aim for a practicable and

realizable solution. Everybody aligned on the aim of having a working application at the end of Sprint 2. For the estimation process of user stories we make again usage of planningpoker.com as the team prefers this “easy-to-use” tool. During our second estimation process we had the situation in which the estimated story points differs strongly from one team member to another. In this case the team members shortly explains how they understood and interpreted the user story. It turned out that team members misinterpreted the user stories. With having this discussion all team members got the same understanding of the specific functionality and scope of a user story. After the alignment the team members adjusted their estimations. Beside this need for discussion the team estimated most of the user stories with very similar story points. This fact shows the increase within the effort estimation for each team member in comparison to Sprint 1 planning. Over the short time of two Sprint planning the team received a better understanding and feeling for the estimation with story points.

01.06.2017 Sprint 2 Review

As our Sprint 1 Review four weeks ago was not completely structured like a Sprint Review meeting we focused on the actual intention of a Sprint Review meeting during Sprint 2 Review. This means we demonstrated the completed and implemented functionalities with screen sharing and the product owner accepted the completed tasks. Within the User Story ICSDSHOW-22 “See all orders/reservations” the functionality “filtering the list by date” could have not been implemented during Sprint 2. The team was unsure on how to proceed with this task. One team member added that especially in regards of having a coherent release, implementing less but working functionalities is much better than implementing more but not working functionalities. For this the team reminded itself focusing on completing functionalities and not working straight to a working application.