Deployment of WordPress Application on Kubernetes

By Jesús Manuel Mariño Valcarce

(jesus.manuel.marino at vodafone dot com)

2023/05/19

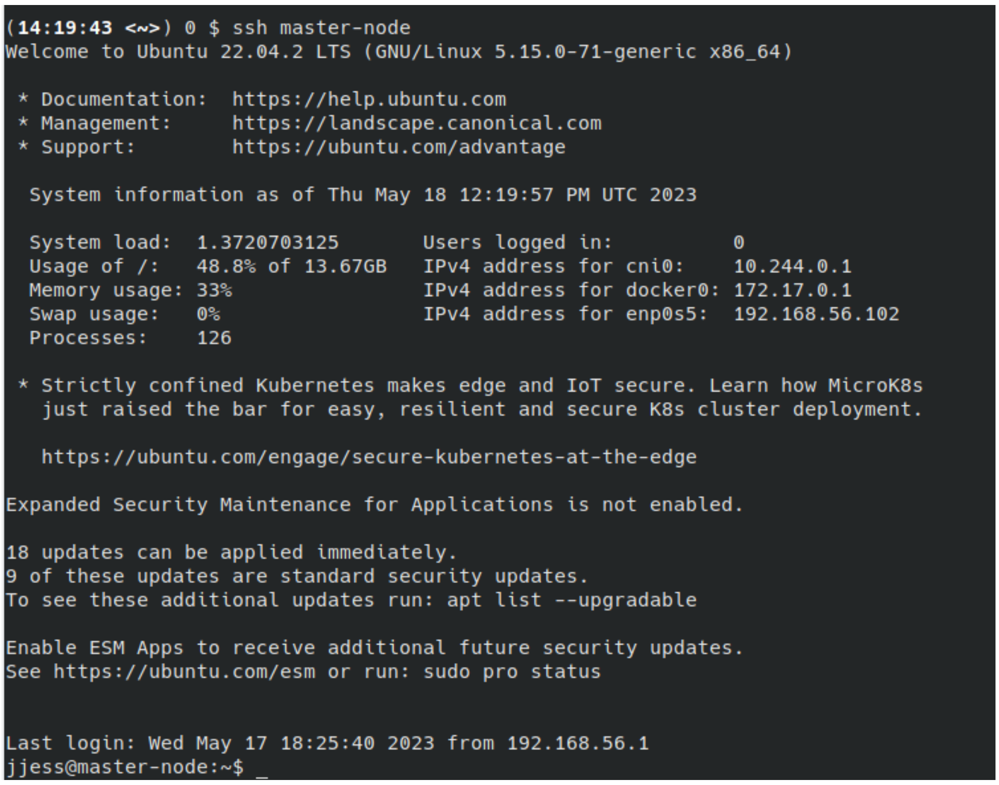
Considerations

I didn't use the SimpliLearn's Lab because I started to play with components discused in the DevOps module before they would be available. I've used my own computer, deploying all necesary components in it by my own means. My lab setup will be detailed below.

Lab Setup

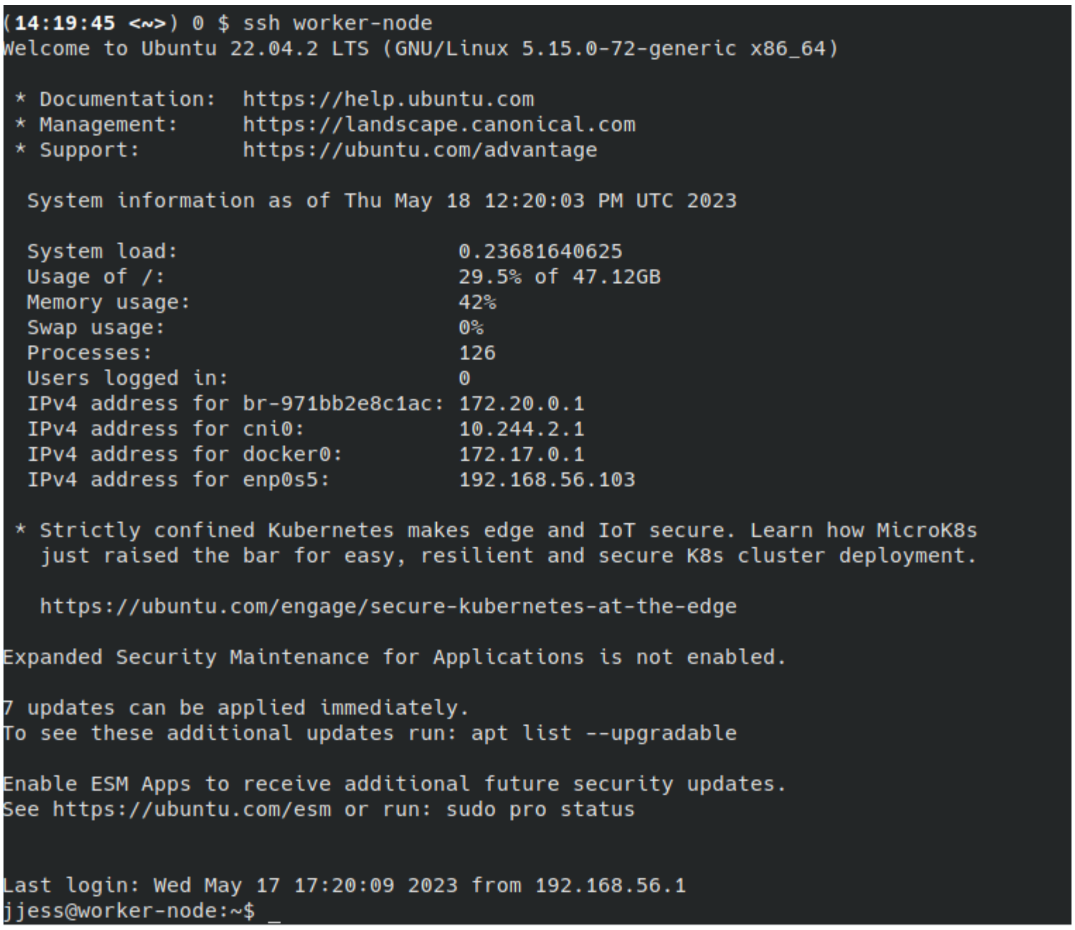
My computer runs FreeBSD 13.2. With the native FreeBSD hypervisor, **bhyve**, I've created two vms running Ubuntu 22.04LTS:

master-node / 192.168.56.102



```
(14:19:43 <~>) 0 $ ssh master-node
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-71-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu May 18 12:19:57 PM UTC 2023

  System load:  1.3720703125       Users logged in:          0
  Usage of /:   48.8% of 13.67GB   IPv4 address for cni0:    10.244.0.1
  Memory usage: 33%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                 IPv4 address for enp0s5:  192.168.56.102
  Processes:    126

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

18 updates can be applied immediately.
9 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Wed May 17 18:25:40 2023 from 192.168.56.1
jjess@master-node:~$ _
```

worker-node / 192.168.56.103



```
(14:19:45 <~>) 0 $ ssh worker-node
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-72-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu May 18 12:20:03 PM UTC 2023

  System load:                      0.23681640625
  Usage of /:                       29.5% of 47.12GB
  Memory usage:                     42%
  Swap usage:                       0%
  Processes:                        126
  Users logged in:                  0
  IPv4 address for br-971bb2e8c1ac: 172.20.0.1
  IPv4 address for cni0:            10.244.2.1
  IPv4 address for docker0:         172.17.0.1
  IPv4 address for enp0s5:          192.168.56.103

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

7 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Wed May 17 17:20:09 2023 from 192.168.56.1
jjess@worker-node:~$ _
```

This two nodes will have:

- master and worker nodes will conform a Kubernetes cluster
- worker-node will have a docker container running Jenkins
- worker-node will run the pods for MariaDB (as a requirement for Wordpress) and Wordpress
- Kubernetes' Storage Class will be **local-storage** type, and it will be located in worker-node's filesystem.

Kubernetes installation

Both vms acting as master-node and worker-node require some preparation in order to build the kubernetes cluster.

First, install docker and the kubernets packages. In both nodes:

```
sudo apt-get install -y docker.io
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

sudo systemctl enable docker

sudo curl -fsSLo /etc/apt/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/
kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubectl
sudo apt install kubeadm kubelet
```

Additional configurations in both nodes:

```
# swapping must be deactivated
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab

# kubernetes networking in /etc/sysctl.d/kubernetes.conf :


       net.bridge.bridge-nf-call-ip6tables = 1
       net.bridge.bridge-nf-call-iptables = 1
       net.ipv4.ip_forward = 1
```

Kubernetes install:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=192.168.56.102
```

pod-network-cidr is the network address for pods, and the API server is the master-node vm’s IP

The previous command answers with another command to be able to join nodes to the cluster:

```
To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.56.102:6443 --token 91a6n1.lfhvws3lnjhnjqsl \
        --discovery-token-ca-cert-hash
sha256:998f02b695463b9eecb3543bbc51653946eeab762e814ae943a2e674dc3b8649
```

Now, in order to avoid execute Kubernetes commands as root, create custom configuration for a regular user:

```
rm -fR $HOME/.kube
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Quick check for kubernetes internal pods (kube-system):

```
kubectl get pods -n kube-system

NAME                                   READY   STATUS    RESTARTS   AGE
coredns-5d78c9869d-6gcv4               0/1     Pending   0          2m53s
coredns-5d78c9869d-dhmhs               0/1     Pending   0          2m53s
etcd-master-node                       1/1     Running   12         3m7s
kube-apiserver-master-node             1/1     Running   13         3m3s
kube-controller-manager-master-node    1/1     Running   0          3m3s
kube-proxy-2sgln                       1/1     Running   0          2m53s
kube-scheduler-master-node             1/1     Running   12         3m5s
```

In order to get network connectivity between pods, a network layer for this purpose has to be deployed. In my case I’ve opted for Flannel, a network layer designed for Kubernetes:

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml

kubectl get pods -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
coredns-5d78c9869d-6gcv4               1/1     Running   0          4m10s
coredns-5d78c9869d-dhmhs               1/1     Running   0          4m10s
etcd-master-node                       1/1     Running   12         4m24s
kube-apiserver-master-node             1/1     Running   13         4m20s
kube-controller-manager-master-node    1/1     Running   0          4m20s
kube-proxy-2sgln                       1/1     Running   0          4m10s
kube-scheduler-master-node             1/1     Running   12         4m22s
```

Now it’s time to add the worker-node to the cluster. If we missed the command to add new nodes we can retrieve it doing:

```
kubeadm token create --print-join-command

kubeadm join 192.168.56.102:6443 --token 91a6n1.lfhvws3lnjhnjqsl \
        --discovery-token-ca-cert-hash
sha256:998f02b695463b9eecb3543bbc51653946eeab762e814ae943a2e674dc3b8649
```

So, in the worker-node we do:

```
kubeadm join 192.168.56.102:6443 --token 91a6n1.lfhvws3lnjhnjqsl \
        --discovery-token-ca-cert-hash
sha256:998f02b695463b9eecb3543bbc51653946eeab762e814ae943a2e674dc3b8649
```

After that, in the master-node:

```
kubectl  get nodes
NAME          STATUS   ROLES           AGE     VERSION
master-node   Ready    control-plane   31m     v1.27.1
worker-node   Ready    <none>          2m22s   v1.27.1
```

In order to ckeck if the cluster is able to create new pods, we can create quickly a new one:

```
kubectl run curl --image=radial/busyboxplus:curl -i --tty

kubectl get pods
NAME   READY   STATUS    RESTARTS      AGE
curl   1/1     Running   1 (15s ago)   56s

kubectl delete pods curl
```

Finally, to be able to shutdown the cluster, we must shutdown before all worker nodes, and the master-node at the end. It’s enogh to send an ACPI shutdown to the worker-node (simulated ACPI shutdown to the vm), and another to the master-node when the worker is down.

Jenkins installation

I’ve followed the instructions available in:

https://www.jenkins.io/doc/book/installing/docker/

I’ve chosen the image **docker:dind** (Docker in Docker) in order to be able to create Docker images inside Jenkins. To create the Jenkins container I’ve used the example available in the previous URL. The script contains:

```
#!/bin/bash

docker run \
  --name jenkins-docker \
  --rm \
  --detach \
  --privileged \
  --network jenkins \
  --network-alias docker \
  --env DOCKER_TLS_CERTDIR=/certs \
  --volume jenkins-docker-certs:/certs/client \
  --volume jenkins-data:/var/jenkins_home \
  --publish 2376:2376 \
  docker:dind \
  --storage-driver overlay2
```

Important here is the persistent volumes used. I put the script in this path:

```
/home/jjess/jenkins-docker
```

And the persistent volumes for data and certs were created with:

```
mkdir -p /home/jjess/jenkins-docker/jenkins-data
mkdir -p /home/jjess/jenkins-docker/jenkins-docker-certs
```

Executing the script the container is started automatically (in the worker-node):

```
sudo docker ps

CONTAINER ID   IMAGE                      COMMAND                  CREATED       STATUS       PORTS
                                                                                       NAMES
48c76d757ee5   jenkins_compose_jenkins   "/usr/bin/tini -- /u…"   3 days ago    Up 6 hours
0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 0.0.0.0:50003->50000/tcp, :::50003->50000/tcp
jenkins
```

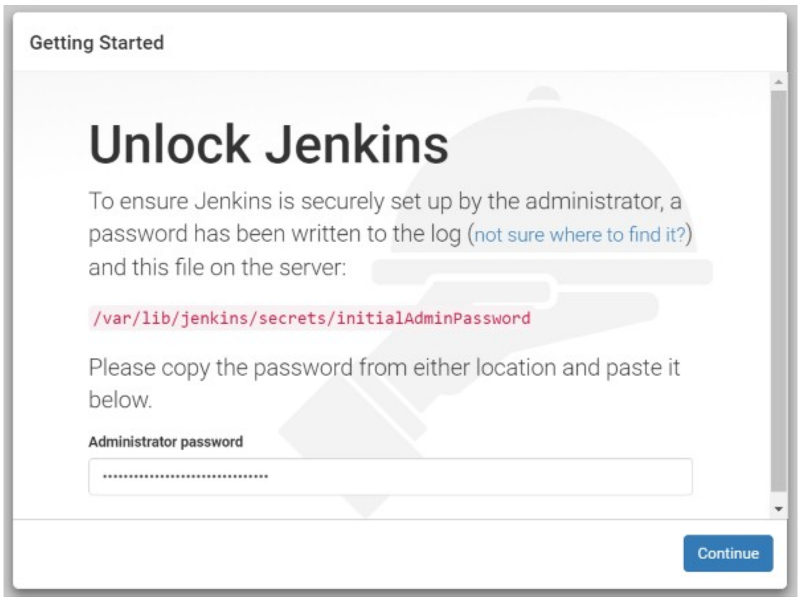
We can access to Jenkins navigating to:

```
http://192.168.56.103:8080/
```

That’s the IP of the worker-node and the default TCP port.

The first task after deploying Jenkins is to change the administrator password. When the container starts a provisional password should be shown in the console (for example with docker-compose, or checking the logs with *docker logs jenkins*, or in the path ‘*/var/jenkins_home/secrets/initialAdminPassword*’ accesible from the persitent volume).

The initial Jenkins web page is similar to this one:



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

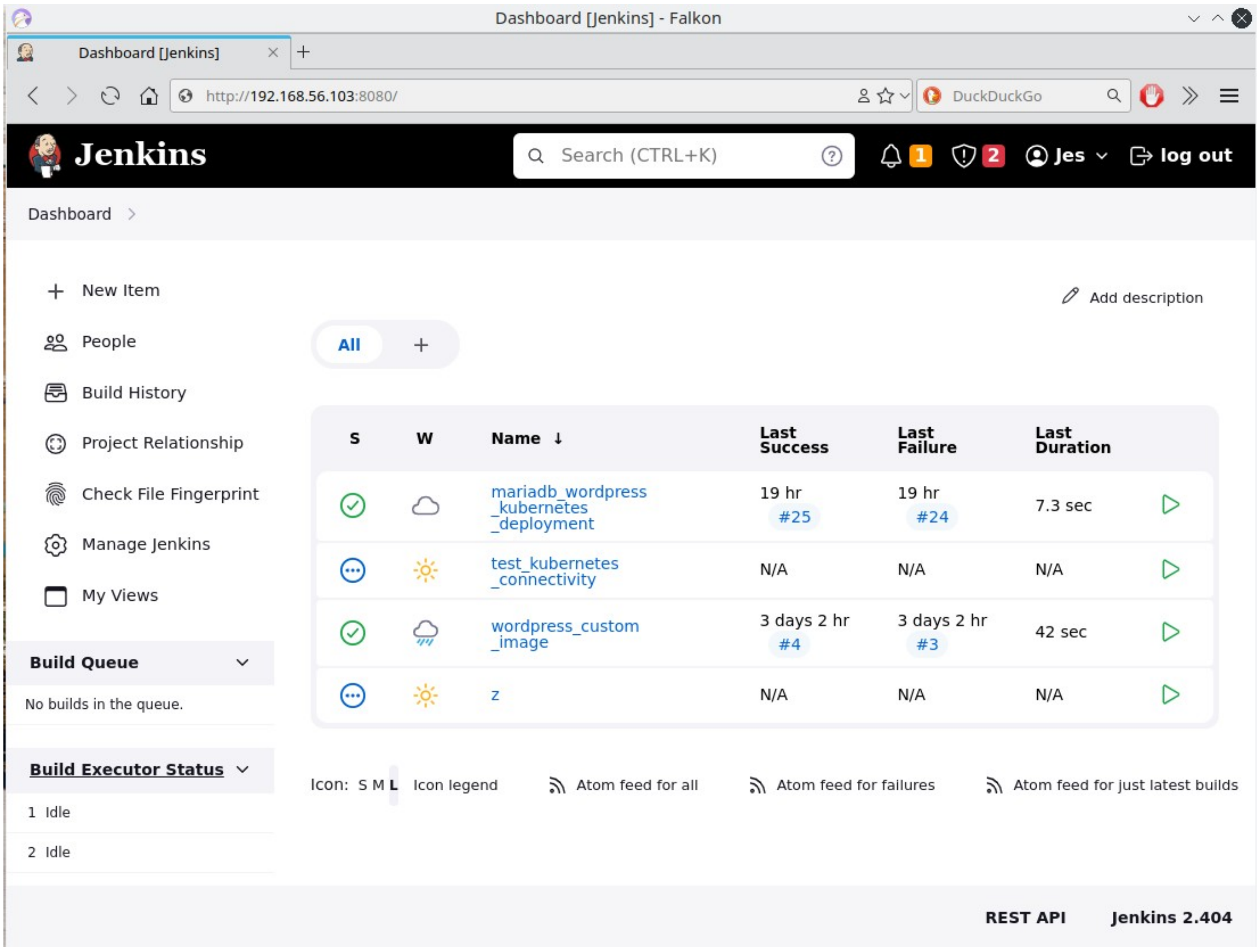
Please copy the password from either location and paste it below.

Administrator password

••••••••••••••••••••••••••••••••

Continue

An screenshot of my Jenkins instance after creating and running some jobs:



Dashboard [Jenkins] - Falkon

http://192.168.56.103:8080/

Jenkins — Search (CTRL+K) — 1 — 2 — Jes — log out

Dashboard

New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

Add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☁	mariadb_wordpress_kubernetes_deployment	19 hr #25	19 hr #24	7.3 sec
⋯	☀	test_kubernetes_connectivity	N/A	N/A	N/A
✓	🌧	wordpress_custom_image	3 days 2 hr #4	3 days 2 hr #3	42 sec
⋯	☀	z	N/A	N/A	N/A

Icon: S M L   Icon legend   Atom feed for all   Atom feed for failures   Atom feed for just latest builds

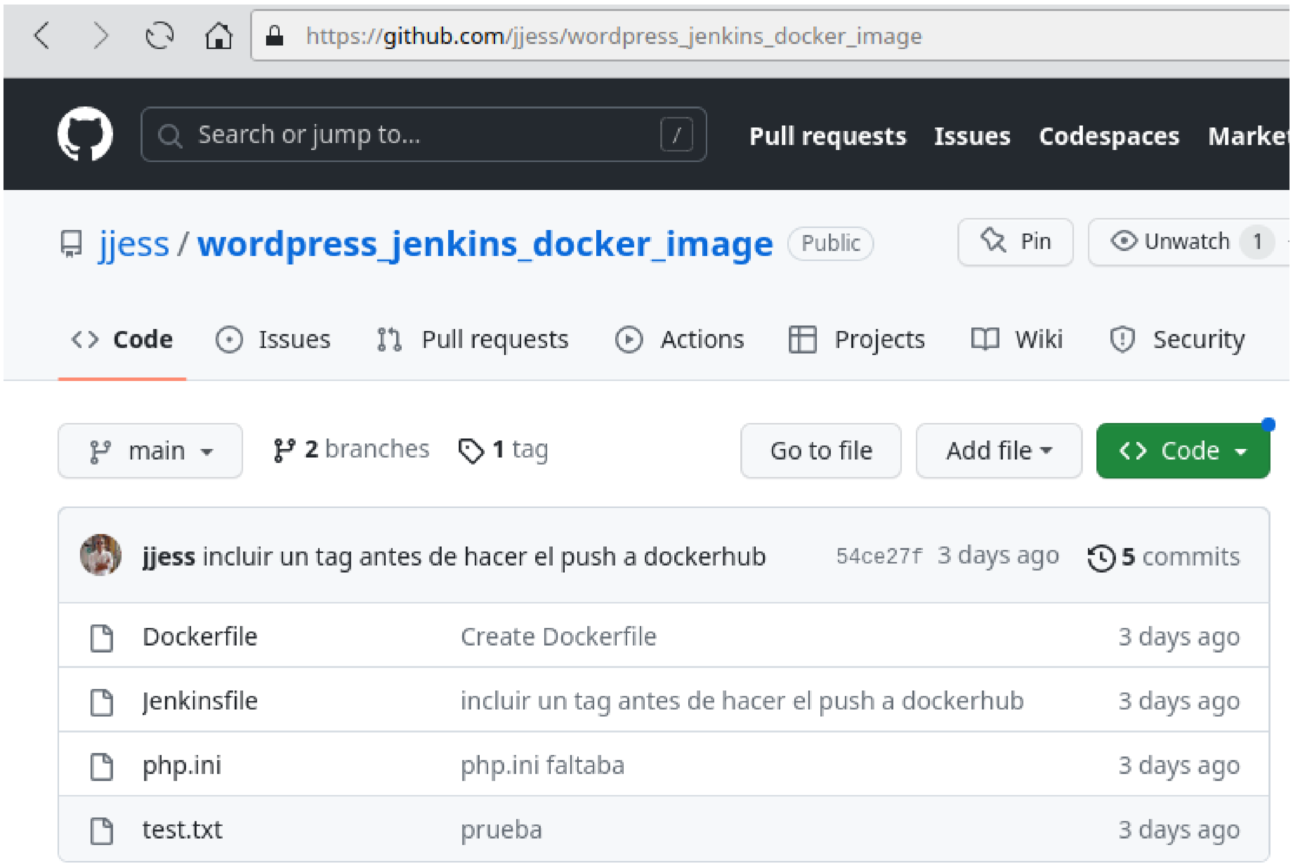
REST API   Jenkins 2.404

Custom Wordpress Docker image

I’ve considered that the wordpress to be deployed in kubernetes could be a custom one, not the included with official Docker image. For the sake of practicing I’ve prepared with Jenkins a custom wordpress image.

My GitHub repository for that custom wordpress image is available at:

https://github.com/jjess/wordpress_jenkins_docker_image



jjess incluir un tag antes de hacer el push a dockerhub 54ce27f 3 days ago 5 commits

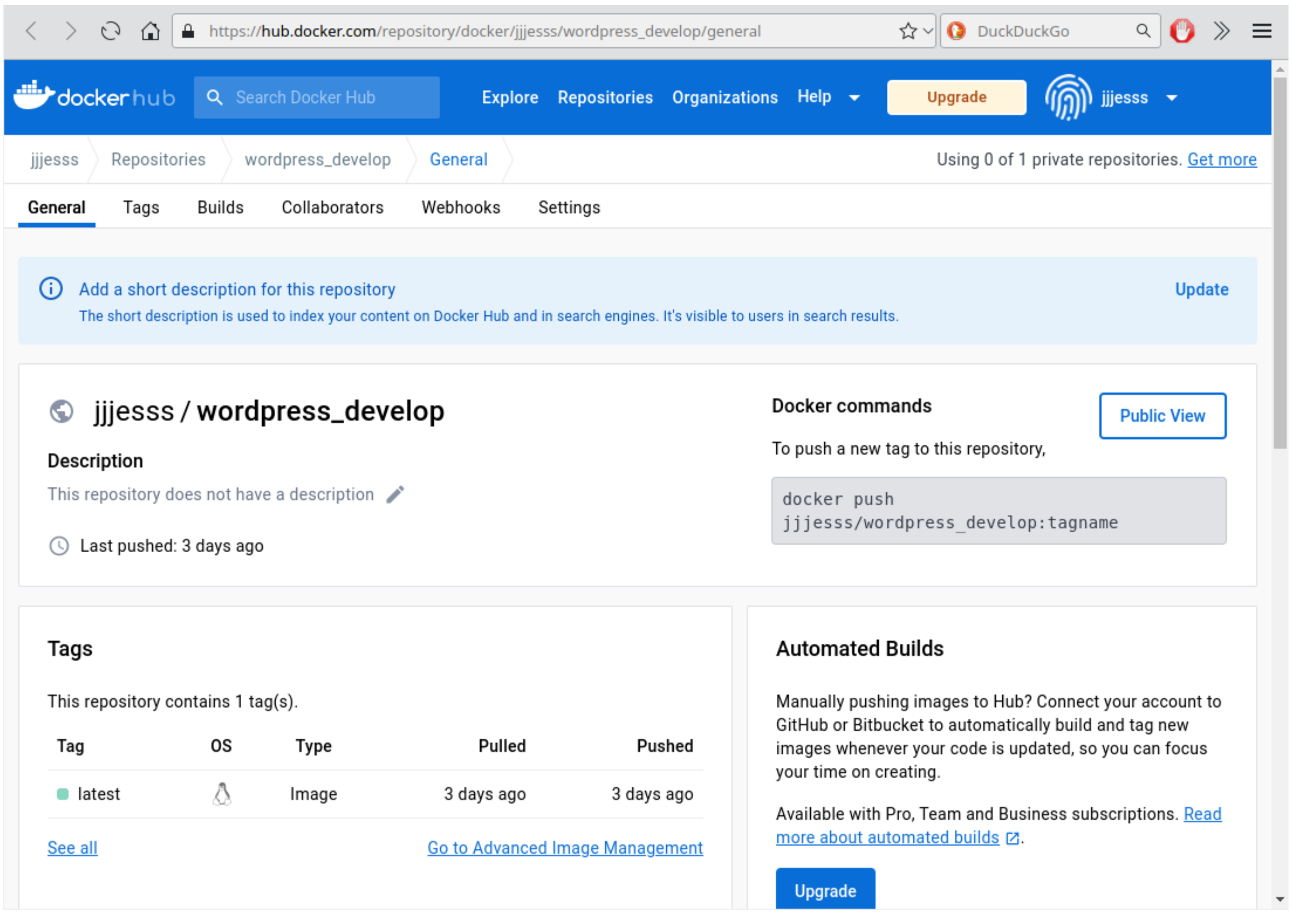
Dockerfile	Create Dockerfile	3 days ago
Jenkinsfile	incluir un tag antes de hacer el push a dockerhub	3 days ago
php.ini	php.ini faltaba	3 days ago
test.txt	prueba	3 days ago

What it’s included is:

- Dockerfile to build the image
- Jenkinsfile with the pipeline to build the image and send it to my Docker Hub repository
- One custom php.ini, to be included in the image
- test.txt is unused

My Docker Hub repository is available at:

https://hub.docker.com/repository/docker/jjjesss/wordpress_develop/general


https://hub.docker.com/repository/docker/jjjesss/wordpress_develop/general

jjjesss / Repositories / wordpress_develop / General — Using 0 of 1 private repositories. Get more

General | Tags | Builds | Collaborators | Webhooks | Settings

Add a short description for this repository
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

jjjesss / wordpress_develop

Description
This repository does not have a description

Last pushed: 3 days ago

Docker commands
To push a new tag to this repository,

```
docker push
jjjesss/wordpress_develop:tagname
```

Tags
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	3 days ago	3 days ago

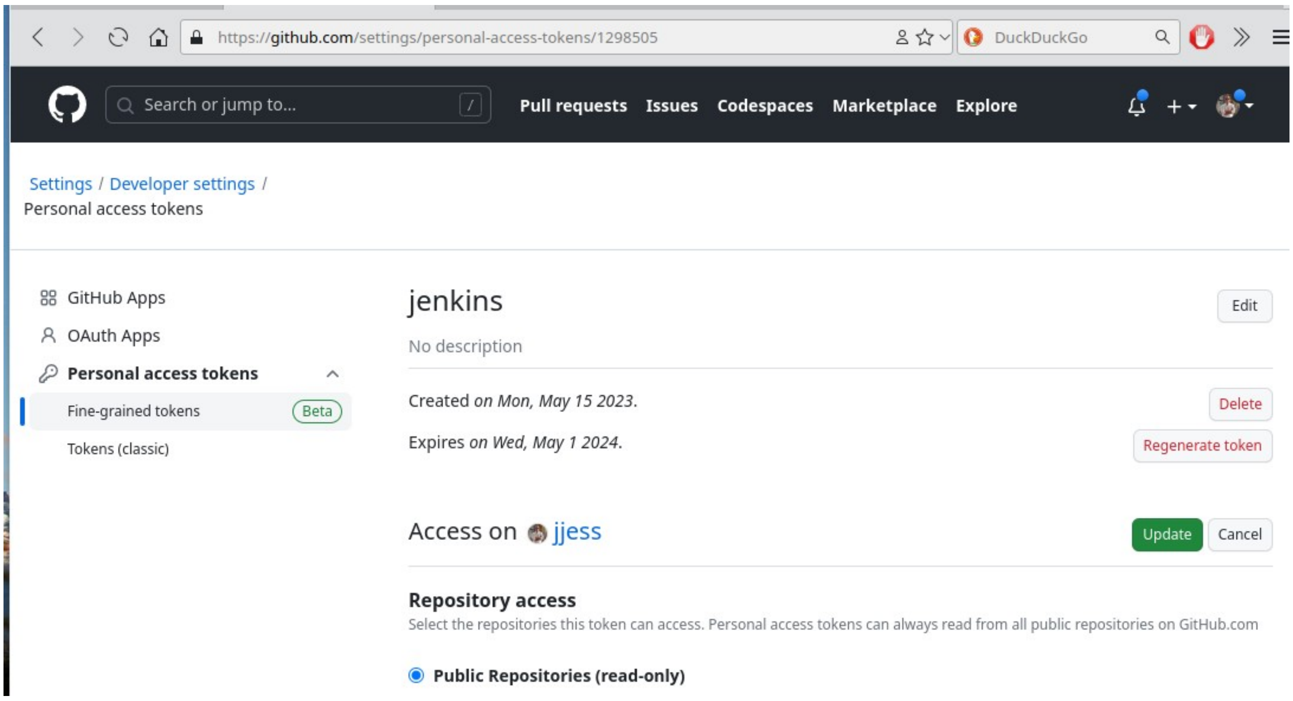
See all — Go to Advanced Image Management

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Before creating a new job in Jenkins to create the Wordpress Docker Image we must provide credentials to connect to Github (where all relevant files for building are stored) and DockerHub (where it will be pushed and be retrieved later when deploying to kubernetes).

In Github I've created a token to be able to connect from Jenkins to all my public repositories:


https://github.com/settings/personal-access-tokens/1298505

Settings / Developer settings / Personal access tokens

GitHub Apps · OAuth Apps · Personal access tokens (Fine-grained tokens Beta, Tokens (classic))

jenkins
No description

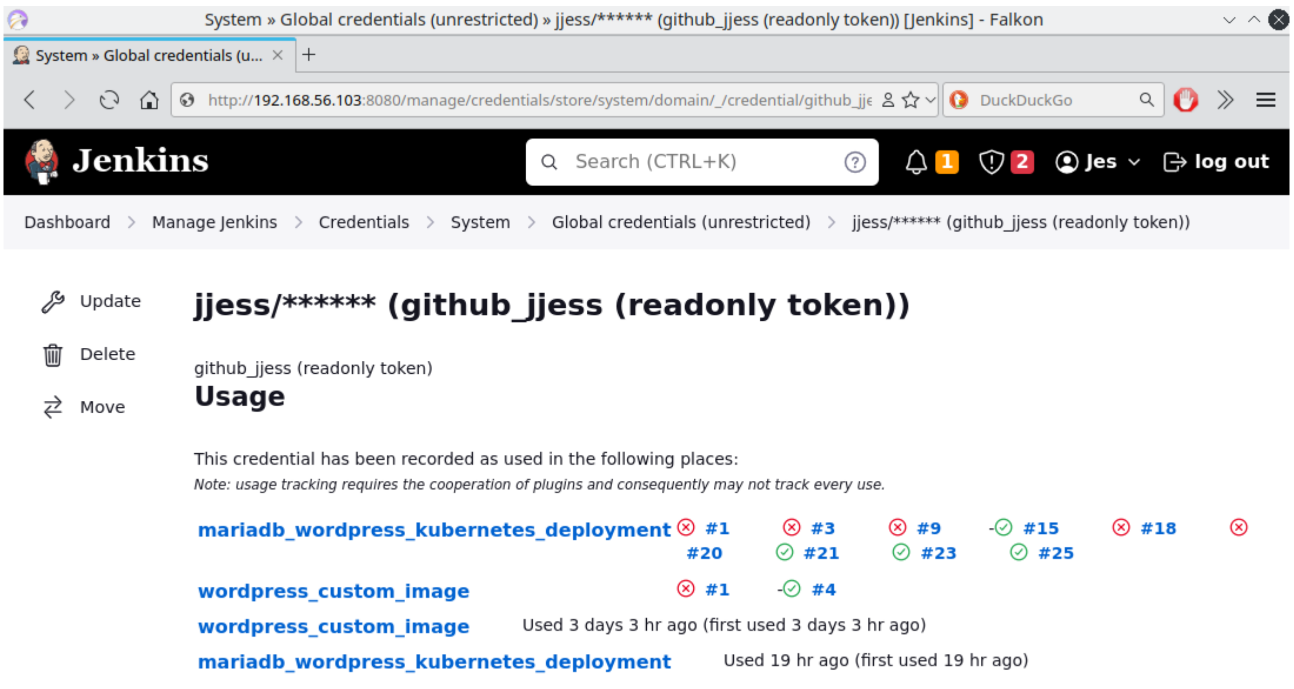
Created on Mon, May 15 2023.
Expires on Wed, May 1 2024.

Access on jjess

Repository access
Select the repositories this token can access. Personal access tokens can always read from all public repositories on GitHub.com

● Public Repositories (read-only)

After that, I’ve created a new credential in Jenkins using the token provided by Github:


System » Global credentials (unrestricted) » jjess/****** (github_jjess (readonly token)) [Jenkins] - Falkon

System » Global credentials (u...

http://192.168.56.103:8080/manage/credentials/store/system/domain/_/credential/github_jje

DuckDuckGo

Jenkins   Search (CTRL+K)   1   2   Jes   log out

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > jjess/****** (github_jjess (readonly token))

Update
Delete
Move

jjess/**** (github_jjess (readonly token))**

github_jjess (readonly token)

Usage

This credential has been recorded as used in the following places:

Note: usage tracking requires the cooperation of plugins and consequently may not track every use.

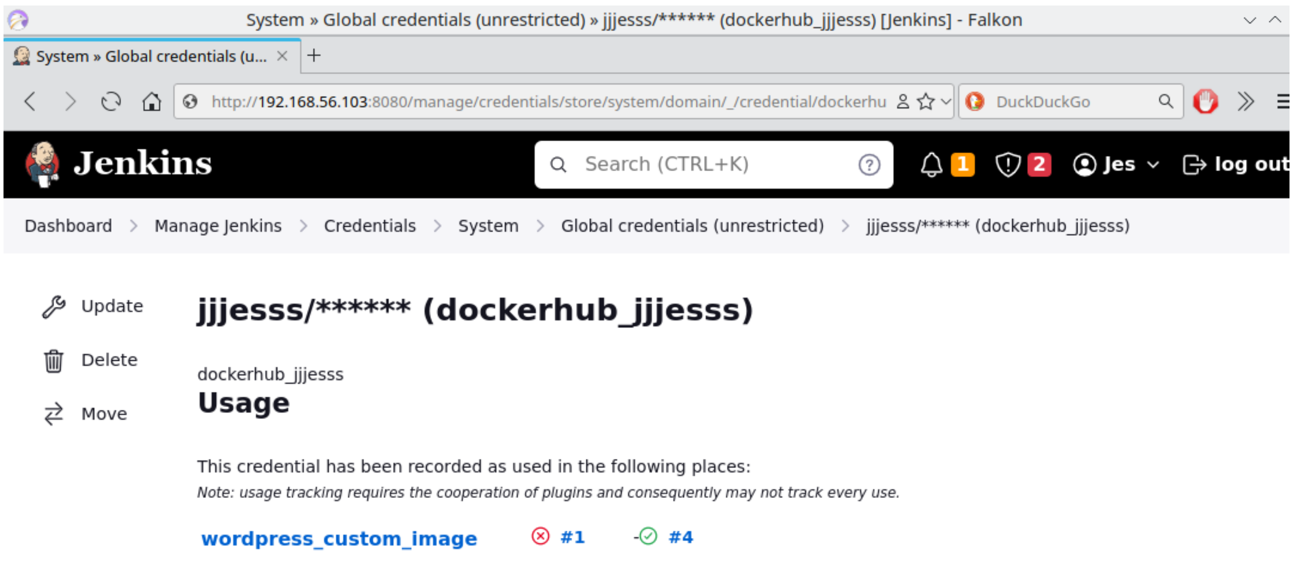
mariadb_wordpress_kubernetes_deployment ⊗ #1 ⊗ #3 ⊗ #9 -✓ #15 ⊗ #18 ⊗ #20 ✓ #21 ✓ #23 ✓ #25

wordpress_custom_image ⊗ #1 -✓ #4

wordpress_custom_image Used 3 days 3 hr ago (first used 3 days 3 hr ago)

mariadb_wordpress_kubernetes_deployment Used 19 hr ago (first used 19 hr ago)

For DockerHub access I used my login:password:


System » Global credentials (unrestricted) » jjjesss/****** (dockerhub_jjjesss) [Jenkins] - Falkon

System » Global credentials (u...

http://192.168.56.103:8080/manage/credentials/store/system/domain/_/credential/dockerhu

DuckDuckGo

Jenkins   Search (CTRL+K)   1   2   Jes   log out

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > jjjesss/****** (dockerhub_jjjesss)

Update
Delete
Move

jjjesss/**** (dockerhub_jjjesss)**

dockerhub_jjjesss

Usage

This credential has been recorded as used in the following places:

Note: usage tracking requires the cooperation of plugins and consequently may not track every use.

wordpress_custom_image ⊗ #1 -✓ #4

The Dockerfile contains:

```
# WordPress Dockerfile: Create container from official WordPress image, basic customizations.
# docker build -t jjess:wordpress_jjess:latest .

FROM wordpress:latest

# APT Update/Upgrade, then install packages we need
RUN apt update && \
    apt upgrade -y && \
    apt autoremove && \
    apt install -y \
    vim \
    wget \
    mariadb-client

# Replace php.ini
COPY php.ini /usr/local/etc/php

# Install WP-CLI
RUN wget https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar && \
    php wp-cli.phar --info&& \
    chmod +x wp-cli.phar && \
    mv wp-cli.phar /usr/local/bin/wp && \
    # Remove old php.ini files (wihtout creating new image)
    rm /usr/local/etc/php/php.ini-development && \
    rm /usr/local/etc/php/php.ini-production
```

The Jenkinsfile with the pipeline:

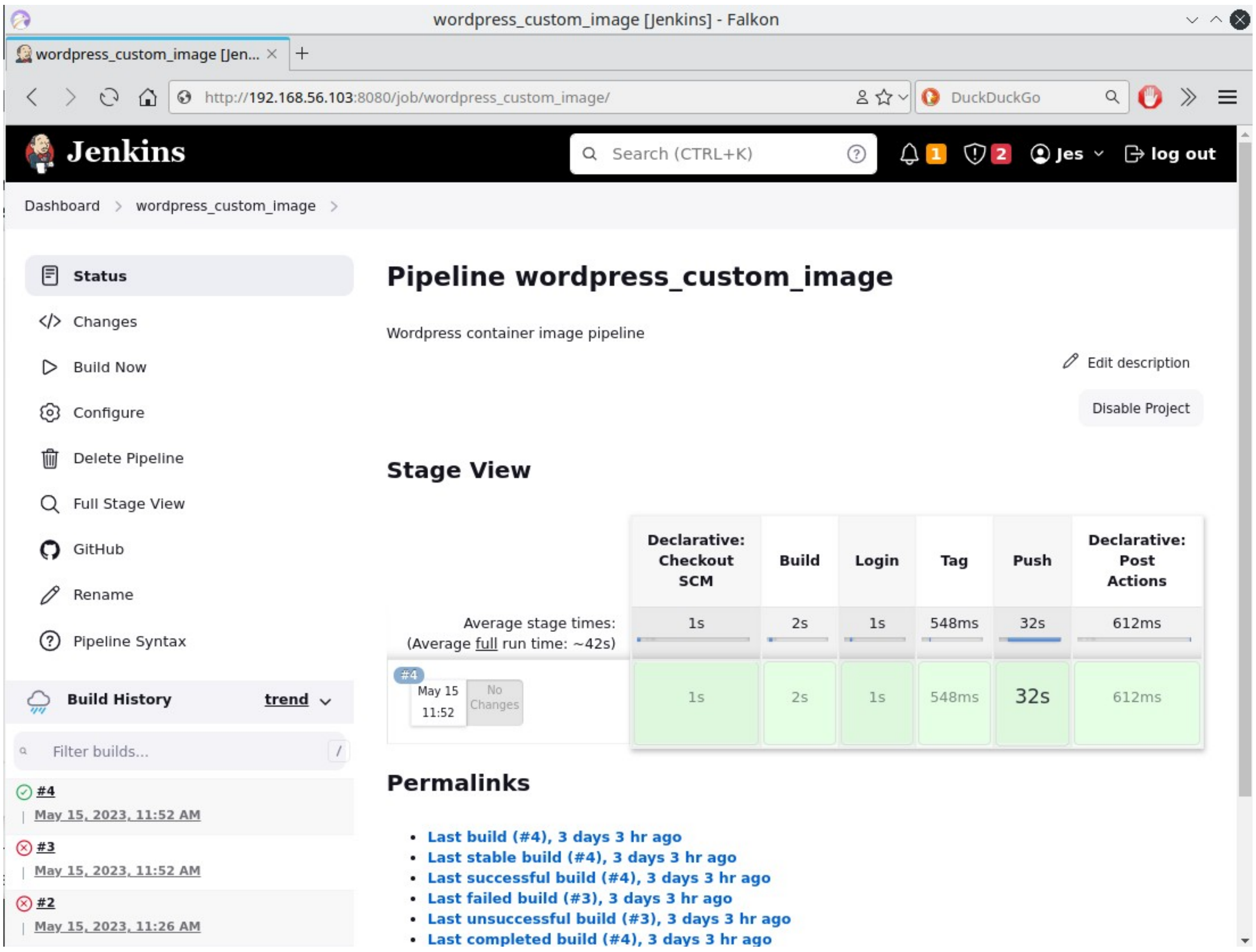
```
pipeline{

        agent any

        environment {
                DOCKERHUB_CREDENTIALS=credentials('dockerhub_jjjesss')
        }

        stages {

                stage('Build') {

                        steps {
                                sh 'docker build -t jjess/wordpress_develop:latest .'
                        }
                }

                stage('Login') {

                        steps {
                                sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u
$DOCKERHUB_CREDENTIALS_USR --password-stdin'
                        }
                }

                 stage('Tag') {

                         steps {
                                 sh 'docker tag jjess/wordpress_develop:latest
jjjesss/wordpress_develop:latest'
                         }
                 }

                stage('Push') {

                        steps {
                                sh 'docker push jjjesss/wordpress_develop:latest'
                        }
                }
        }

        post {
                always {
                        sh 'docker logout'
                }
        }

}
```

The stages in the pipeline are:

The Jenkins job I’ve created is **Pipeline** type, and it has the following features:

- Discard old builds
- Github project (https://github.com/jjess/wordpress_jenkins_docker_image/)
- Pipeline script from SCM
  - SCM: **git**
    - repository URL: https://github.com/jjess/wordpress_jenkins_docker_image/
    - credentials: **github_jjess** (created previously)
    - branches to build: ***/main**
  - Script Path: **Jenkinsfile** (name of the pipeline file in github)

And it looks like:



Pipeline wordpress_custom_image

Wordpress container image pipeline

Stage View

	Declarative: Checkout SCM	Build	Login	Tag	Push	Declarative: Post Actions
Average stage times: (Average full run time: ~42s)	1s	2s	1s	548ms	32s	612ms
#4 May 15 11:52 No Changes	1s	2s	1s	548ms	32s	612ms

Permalinks

- Last build (#4), 3 days 3 hr ago
- Last stable build (#4), 3 days 3 hr ago
- Last successful build (#4), 3 days 3 hr ago
- Last failed build (#3), 3 days 3 hr ago
- Last unsuccessful build (#3), 3 days 3 hr ago
- Last completed build (#4), 3 days 3 hr ago

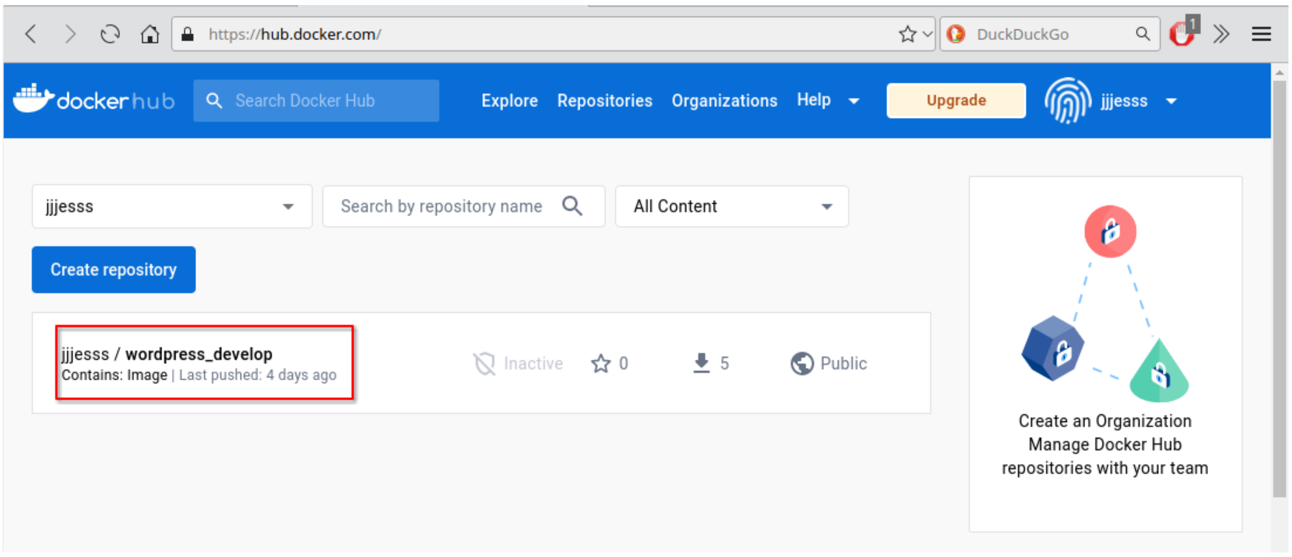
Build History

- #4 May 15, 2023, 11:52 AM
- #3 May 15, 2023, 11:52 AM
- #2 May 15, 2023, 11:26 AM

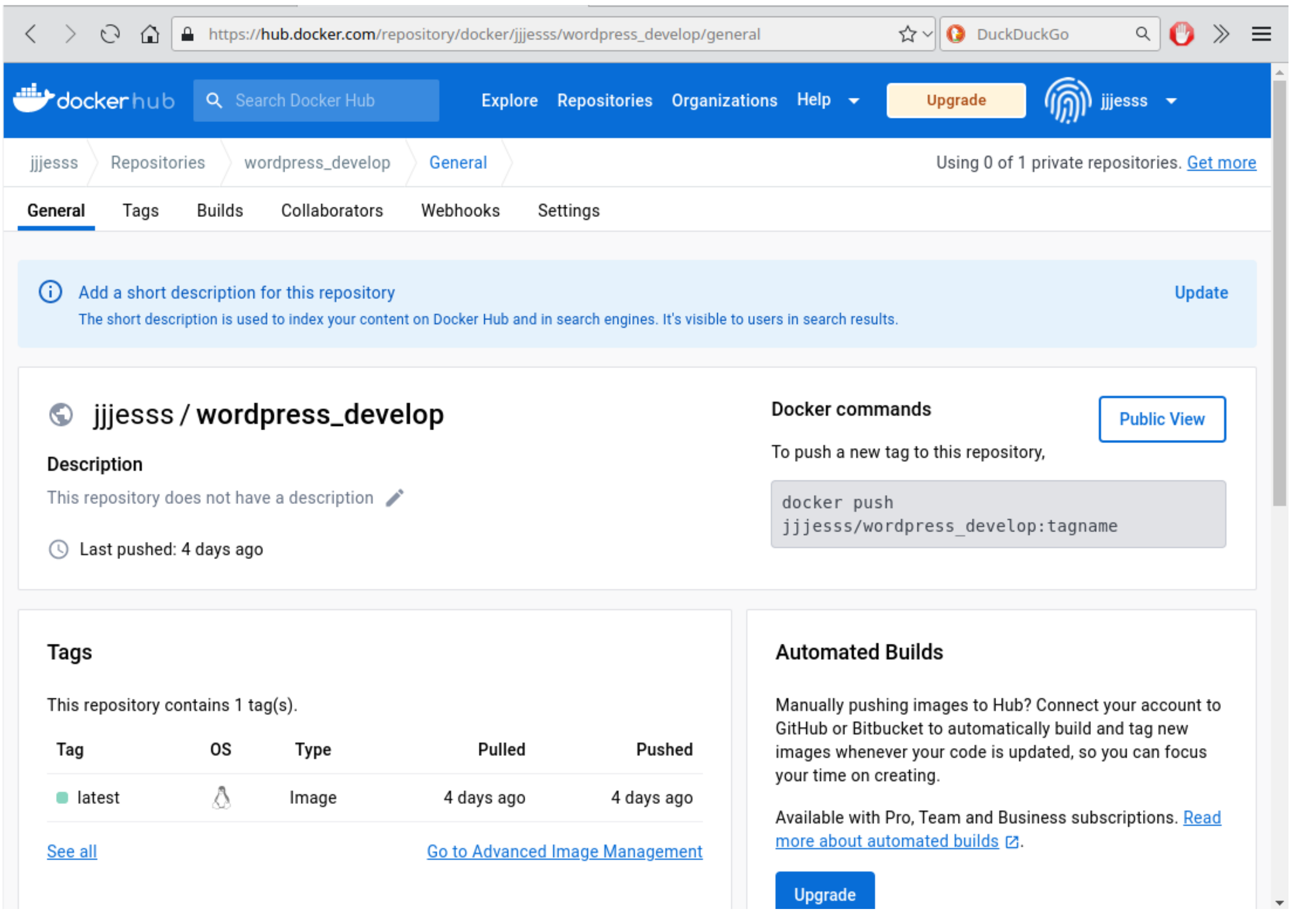
The last succesful build (console, reduced content because it’s very extense):

```
Console Output
Started by user Jes
Obtained Jenkinsfile from git https://github.com/jjess/wordpress_jenkins_docker_image/
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/wordpress_custom_image
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: NONE
using credential github_jjess
 > git rev-parse --resolve-git-dir /var/jenkins_home/workspace/wordpress_custom_image/.git #
timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/jjess/wordpress_jenkins_docker_image/ #
timeout=10
Fetching upstream changes from https://github.com/jjess/wordpress_jenkins_docker_image/
 > git --version # timeout=10
 > git --version # 'git version 2.30.2'
using GIT_ASKPASS to set credentials github_jjess (readonly token)
 > git fetch --tags --force --progress --
https://github.com/jjess/wordpress_jenkins_docker_image/ +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 54ce27ffae22e713ba5cc5a9c10dc8bee7ecab7d (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 54ce27ffae22e713ba5cc5a9c10dc8bee7ecab7d # timeout=10
Commit message: "incluir un tag antes de hacer el push a dockerhub"
 > git rev-list --no-walk 54ce27ffae22e713ba5cc5a9c10dc8bee7ecab7d # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withCredentials
Masking supported pattern matches of $DOCKERHUB_CREDENTIALS or $DOCKERHUB_CREDENTIALS_PSW
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] sh
+ docker build -t jjess/wordpress_develop:latest .
#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 32B 0.0s done
#1 DONE 0.1s

#2 [internal] load .dockerignore
#2 transferring context:
#2 transferring context: 2B done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/wordpress:latest
#3 DONE 0.6s

#4 [1/4] FROM
docker.io/library/wordpress:latest@sha256:06b3c3b2fdc126d5e28b1f1c78a99009fe186d7354c907074095d56
61bd18570
#4 DONE 0.0s

#5 [internal] load build context
#5 transferring context: 30B done
#5 DONE 0.0s
#6 [2/4] RUN apt update &&     apt upgrade -y &&     apt autoremove &&     apt install -y     vim
    wget     mariadb-client
#6 CACHED

#7 [3/4] COPY php.ini /usr/local/etc/php
#7 CACHED

#8 [4/4] RUN wget https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar &&
 php wp-cli.phar --info&&     chmod +x wp-cli.phar &&     mv wp-cli.phar /usr/local/bin/wp &&
rm /usr/local/etc/php/php.ini-development &&     rm /usr/local/etc/php/php.ini-production
#8 CACHED

#9 exporting to image
#9 exporting layers done
#9 writing image sha256:b7c0998618c879f1dd4dcd971e958a6d00de1d951ecc9977707f668b34fb91e3 done
#9 naming to docker.io/jjess/wordpress_develop:latest done
#9 DONE 0.2s
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Login)
[Pipeline] sh
+ echo ****
+ docker login -u jjjesss --password-stdin
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Tag)
[Pipeline] sh
+ docker tag jjess/wordpress_develop:latest jjjesss/wordpress_develop:latest
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Push)
[Pipeline] sh
+ docker push jjjesss/wordpress_develop:latest
The push refers to repository [docker.io/jjjesss/wordpress_develop]
c41696ad2087: Preparing
5ffb78c01086: Waiting
d9cd28510979: Mounted from library/wordpress
71a7c564911a: Pushed
66a345e4caf6: Mounted from library/wordpress
latest: digest: sha256:e8c3087897cf29ee951624fea2de169b3b8fa8b824c3a63d2084daa187d1ea4a size:
5342
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] sh
+ docker logout
Removing login credentials for https://index.docker.io/v1/
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Finally we chan check in DockerHub the recentely created image:



jjjesss / wordpress_develop
Contains: Image | Last pushed: 4 days ago



Add a short description for this repository
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

jjjesss / wordpress_develop

Description
This repository does not have a description

Last pushed: 4 days ago

Docker commands
To push a new tag to this repository,

```
docker push
jjjesss/wordpress_develop:tagname
```

Tags
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	4 days ago	4 days ago

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

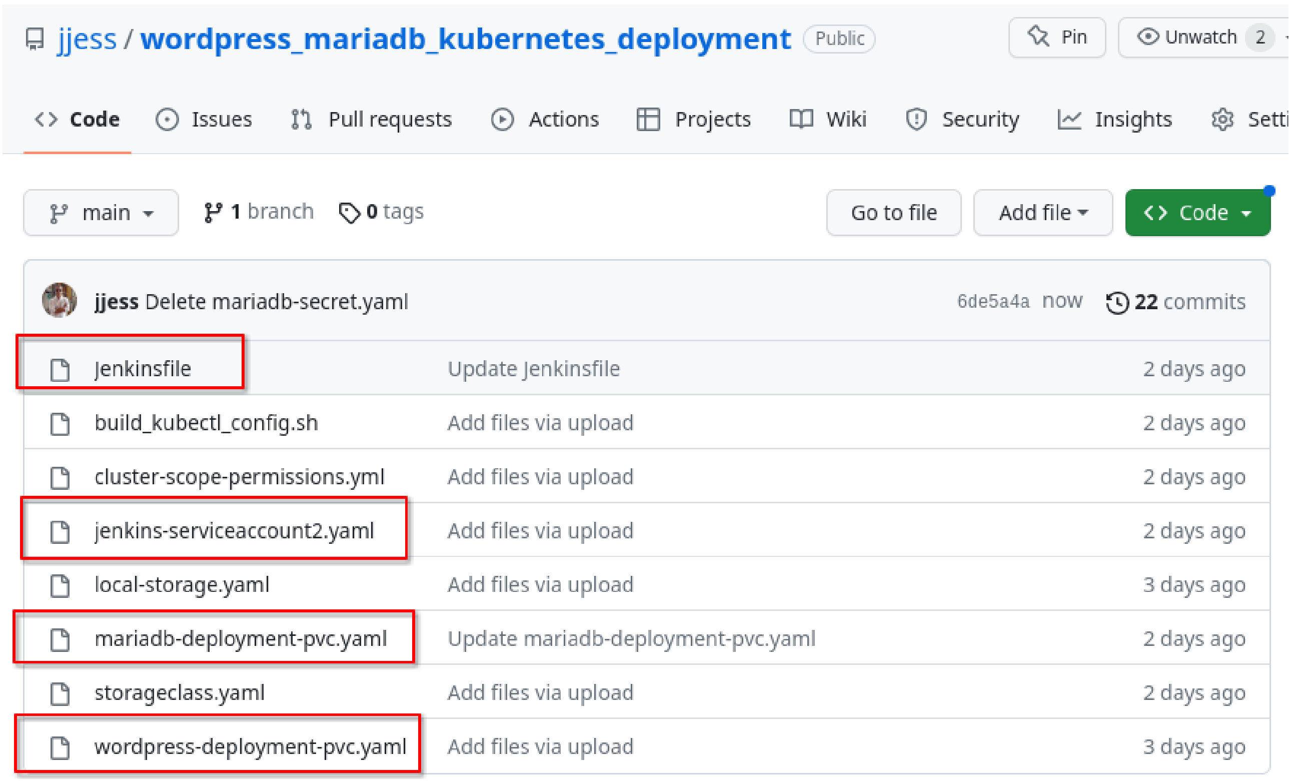
Deploying Wordpress to Kubernetes with Jenkins

I’ve chosen to deploy Wordpress in Kubernetes along with MariaDB for storage, using Persistent Volumes.

The Github repository with all the necesary files is available at:

https://github.com/jjess/wordpress_mariadb_kubernetes_deployment

The repository contains more files than the required ones, because I was testing different configurations, mainly related to the permissions of Jenkins for the deployment in Kubernetes. The main files are:



jjess Delete mariadb-secret.yaml 6de5a4a now 22 commits

Jenkinsfile	Update Jenkinsfile	2 days ago
build_kubectl_config.sh	Add files via upload	2 days ago
cluster-scope-permissions.yml	Add files via upload	2 days ago
jenkins-serviceaccount2.yaml	Add files via upload	2 days ago
local-storage.yaml	Add files via upload	3 days ago
mariadb-deployment-pvc.yaml	Update mariadb-deployment-pvc.yaml	2 days ago
storageclass.yaml	Add files via upload	2 days ago
wordpress-deployment-pvc.yaml	Add files via upload	3 days ago

Briefly, Jenkinsfile for the pipeline, other one related to the permissions for Jenkins in the kubernetes cluster (service account, clusterroles, etc.) and two yaml files for the mariadb and wordpress deployments. Every yaml file contains all the necessary stuff to do what is intended, I mean, there’s no separation, for instance, for service accounts, secrets, persistent volumes, etc.

The file **jenkins-serviceaccount2.yaml** contains:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: default
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods","services"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["list", "get", "create", "delete", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: jenkins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: jenkins
subjects:
  - kind: ServiceAccount
    name: jenkins
    namespace: default
  - kind: User
    name: system:anonymous
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Secret
metadata:
  name: jenkins
  namespace: default
  annotations:
    kubernetes.io/service-account.name: jenkins
type: kubernetes.io/service-account-token
```

Basically, I’ve preferred to specify every single permission in the kubernetes cluster instead of giving full acces (as cluster-admin).

The **mariadb-deployment-pvc.yaml** contains:

```yaml
apiVersion: v1
kind: Secret
metadata:
    name: mariadb-secret
type: Opaque
data:
  mariadb-root-password: c2VjcmV0 #echo -n 'secret'|base64
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: mariadb-configmap
data:
  database_url: mariadb-internal-service #name of service
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb-pv-claim
  labels:
    app: mariadb
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 300M
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mariadb-pv
  labels:
    type: local
spec:
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  capacity:
    storage: 500M
  accessModes:
    - ReadWriteOnce
  local:
    path: /storage/pv/mariadb-pv
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - worker-node
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-internal-service
spec:
  selector:
    app: mariadb
  ports:
    # - protocol: TCP
    - port: 3306
      targetPort: 3306
      protocol: TCP
      nodePort: 31234
  type: LoadBalancer
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deployment
spec: # specification for deployment resource
  replicas: 1
  selector:
    matchLabels:
      app: mariadb
  template: # blueprint for pods
    metadata:
      labels:
        app: mariadb # service will look for this label
    spec: # specification for pods
      containers:
      - name: mariadb
        image: mariadb
        ports:
        - containerPort: 3306 #default one
        env:
        - name: MARIADB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mariadb-secret
              key: mariadb-root-password
        - name: MARIADB_DATABASE
          value: wordpress
        volumeMounts:
        - name: mariadb-pv
          mountPath: /var/lib/mysql
      volumes:
      - name: mariadb-pv
        persistentVolumeClaim:
          claimName: mariadb-pv-claim
```

Important things in this file:

- A secret for the mariadb root's password has been defined in base64 format.
- A configmap has been created to store environment variables, mainly the URL for the mariadb instance (database_url), used later in the Wordpress deployment.
- storage is local-storage, an special kind of StorageClass automatically created when the cluster was created.
- There's one PV (persistent volume) with size 300M. But the PVC (persistent volume claim) has been specified with 500M. The path for the PV (the local path) is **/storage/pv/mariadb**. This path is local to the worker-node, and it must exists (it was created in the worker-node with command **`sudo mkdir -p /storage/pv/mariadb`**).
- A service is created specifiying the ports internal and external. The external port TCP 31234 it not necesary (the wordpress deployment will use the internal ones but it's intented in order to test the database creation from outside, for simplicity lets say, instead of accessing the pod itself). I've chosen LoadBalancer type only because I was playing with different types; it would work with NodePort as well.
- Finally we have the deployment itself specifying the containter port, the root password, the volumes, the docker image, etc. The image is **mariadb**, as no other indication has been stablished, kubernets will search for that image in the DockerHub registry. So, it's the official Docker image for MariaDB.

The **wordpress-deployment-pvc.yaml** file looks like:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
spec:
  selector:
    app: wordpress
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP #default
      nodePort: 31000
  type: LoadBalancer
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-deployment
spec: # specification for deployment resource
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template: # blueprint for Pod
    metadata:
      labels:
        app: wordpress
    spec: # specification for Pod
      containers:
      - name: wordpress
        image: jjjesss/wordpress_develop:latest
        ports:
        - containerPort: 80
        env:
        - name: WORDPRESS_DB_HOST
          valueFrom:
            configMapKeyRef:
              name: mariadb-configmap
              key: database_url
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mariadb-secret
              key: mariadb-root-password
        - name: WORDPRESS_DB_USER
          value: root
        - name: WORDPRESS_DEBUG
          value: "1"
```

What it’s important here is:

- Service with external TCP 31000 port to acces to the Wordpress instance.
- The image for the container, instead of using the official Docker image for Wordpress, we will use our own imagen, namely **`jjjesss/wordpress_develop:latest`**`. Specifying the full repository name, Kubernetes will use that one instead of the default.`
- The host IP for the DB extracted fromt he **mariadb-configmap**.
- The DB root’s password extracted from the secret created in the Mariadb deployment.

The Jenkins pipeline file contains:

```
pipeline{

        agent any


        stages {
                stage('Deploy mariadb to Kubernetes') {
                        steps {
                                withKubeConfig([credentialsId: 'kubernetes_token',
                                                serverUrl: 'https://192.168.56.102:6443',
                                                namespace: 'default'
                                                ]) {
                                                        sh 'kubectl apply -f mariadb-deployment-
pvc.yaml'
                                                        sh 'kubectl apply -f wordpress-deployment-
pvc.yaml'
                                                        sh 'kubectl get deployments'
                                                        sh 'kubectl get pods'
                                                        // kubernetesDeploy(configs: "mariadb-
deployment-pvc.yaml")
                                } // withKubeConfig
                        } // steps
                } // stage
        } // stages

} // pipeline
```

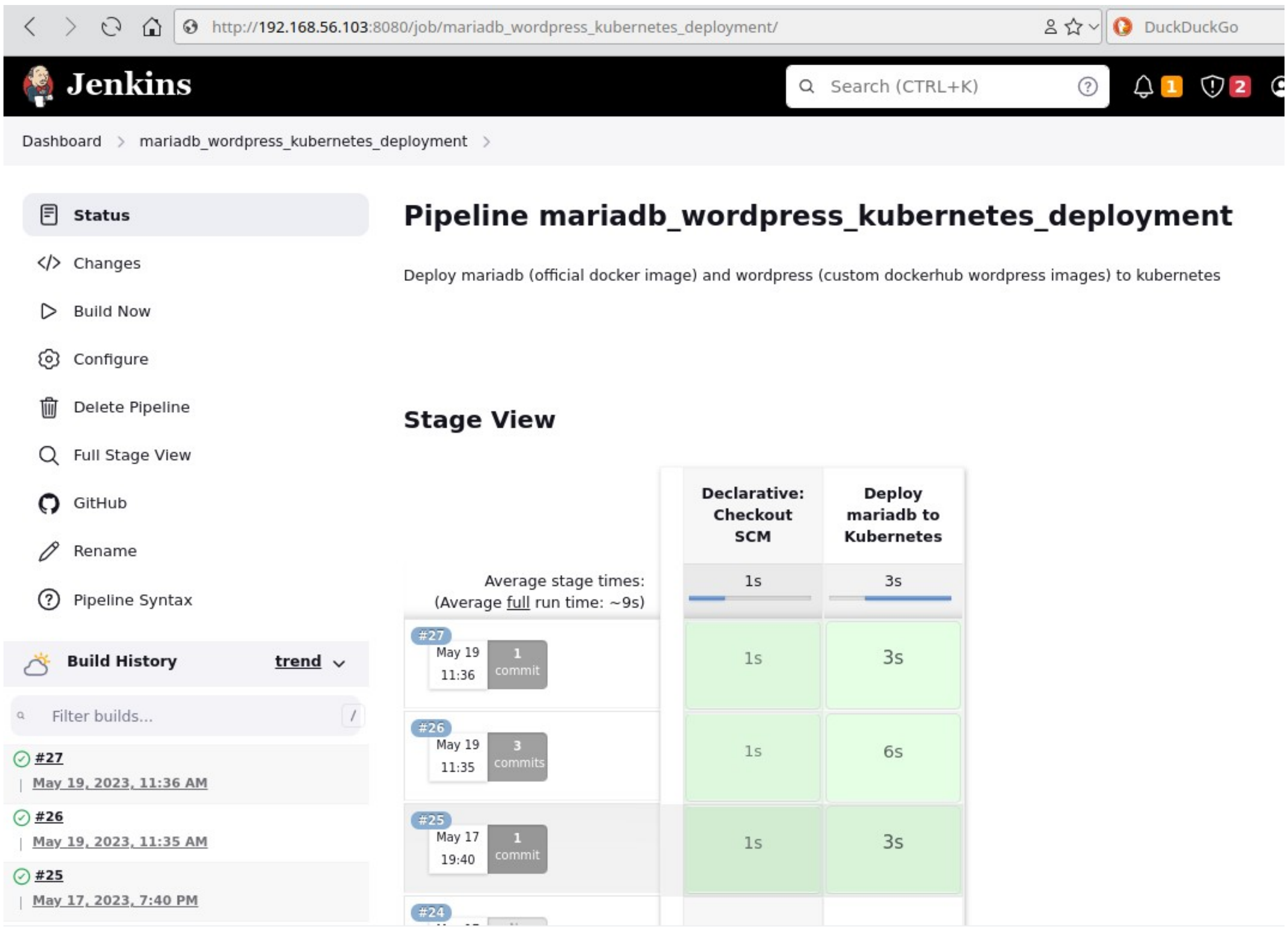
Important in this file:

- Only one stage for everything
- Jenkins must know the credentials for the kubernetes cluster. So a new credential must be configured in Jenkins. I'll explain this point later.
- Jenkins must know the IP of the Kubernetes API and the namespace. Having defined previously the credential, we can configure the access with the **withKubeConfig** directive. To do this, the **Kubernetes CLI** plugin must be installed in Jenkis. More on this in:

  https://plugins.jenkins.io/kubernetes-cli/
- The pipeline in Jenkins is SCM type, as well, pointing to the proper github repository with all the required files, in my case is this one:

  https://github.com/jjess/wordpress_mariadb_kubernetes_deployment

  A couple of screenshots to illustrate this job in Jenkins:



http://192.168.56.103:8080/job/mariadb_wordpress_kubernetes_deployment/

Jenkins

Dashboard > mariadb_wordpress_kubernetes_deployment >

Pipeline mariadb_wordpress_kubernetes_deployment

Deploy mariadb (official docker image) and wordpress (custom dockerhub wordpress images) to kubernetes

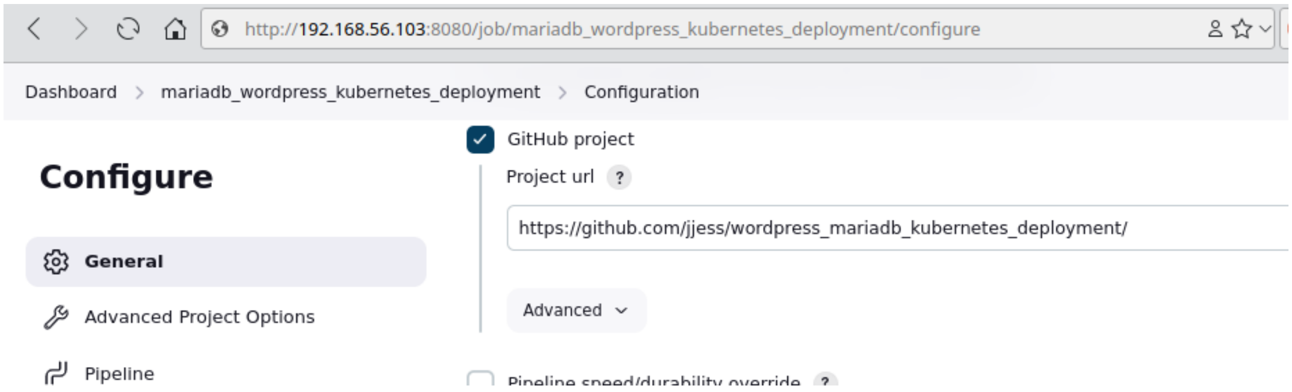
Status · Changes · Build Now · Configure · Delete Pipeline · Full Stage View · GitHub · Rename · Pipeline Syntax

Stage View

	Declarative: Checkout SCM	Deploy mariadb to Kubernetes
Average stage times: (Average full run time: ~9s)	1s	3s
#27 May 19 11:36, 1 commit	1s	3s
#26 May 19 11:35, 3 commits	1s	6s
#25 May 17 19:40, 1 commit	1s	3s

Build History

- #27 May 19, 2023, 11:36 AM
- #26 May 19, 2023, 11:35 AM
- #25 May 17, 2023, 7:40 PM



http://192.168.56.103:8080/job/mariadb_wordpress_kubernetes_deployment/configure

Dashboard > mariadb_wordpress_kubernetes_deployment > Configuration

Configure

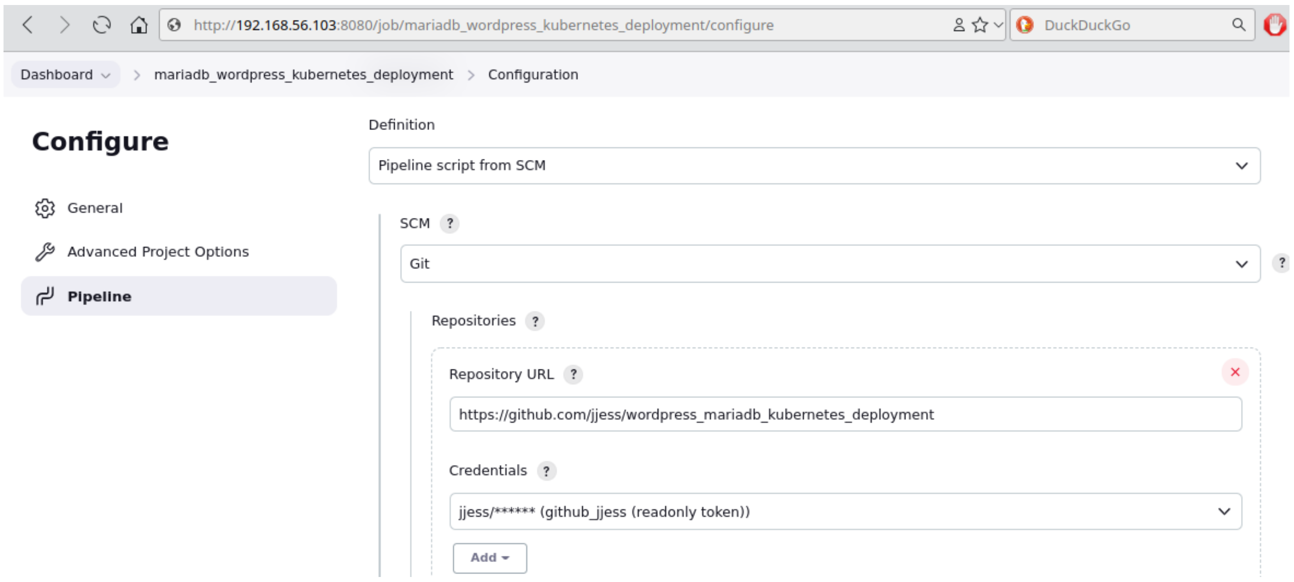
General · Advanced Project Options · Pipeline

[✓] GitHub project

Project url: https://github.com/jjess/wordpress_mariadb_kubernetes_deployment/

Advanced

Pipeline speed/durability override


http://192.168.56.103:8080/job/mariadb_wordpress_kubernetes_deployment/configure

Dashboard > mariadb_wordpress_kubernetes_deployment > Configuration

Configure

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/jjess/wordpress_mariadb_kubernetes_deployment

Credentials: jjess/****** (github_jjess (readonly token))

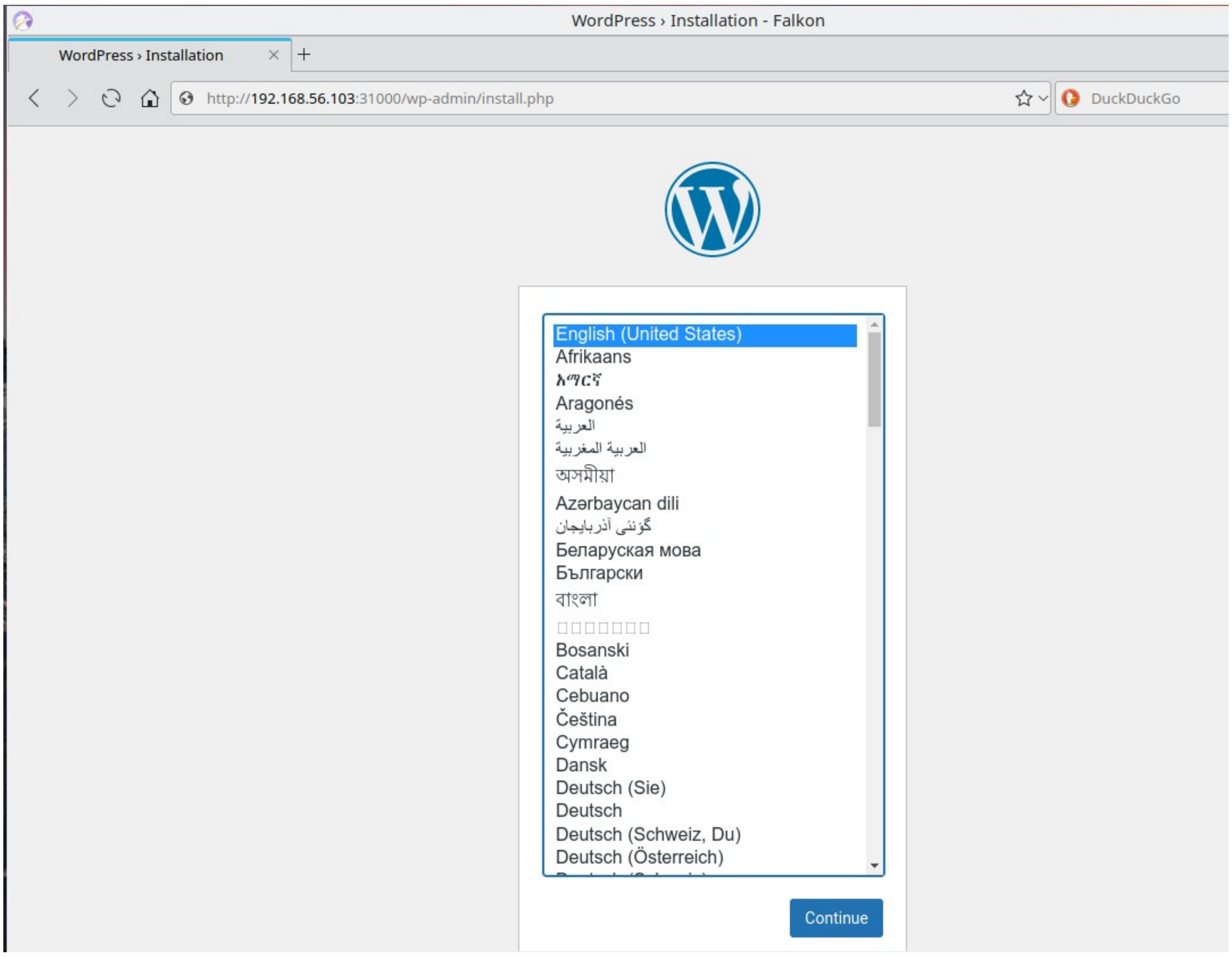
- Finally, with the properly defined kubernetes configuration access method we can apply the deployment yaml files stored in Github. Firstly the DB instance, and after that the Wordpress one. At the end we launch two kubectl commands only to check that the pods are starting up (to review in the Jenkins Job Console).

So, the wordpress instance will be available in TCP port 31000 and whichever cluster node IPs we choose. In my case in any of this:

http://192.168.56.102:31000

http://192.168.56.103:31000

A screenshot of the Wordpress instance asking for initial configuracion:



As and apend to this part, I must explain how to give permissions to jenkins in the kubernetes cluster, to success in the deployment. This issue has been accomplished with the **jenkins-serviceaccount2.yaml** file that contains:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: default
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods","services"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["list", "get", "create", "delete", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: jenkins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: jenkins
subjects:
  - kind: ServiceAccount
    name: jenkins
    namespace: default
  - kind: User
    name: system:anonymous
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Secret
metadata:
  name: jenkins
  namespace: default
  annotations:
    kubernetes.io/service-account.name: jenkins
type: kubernetes.io/service-account-token
```

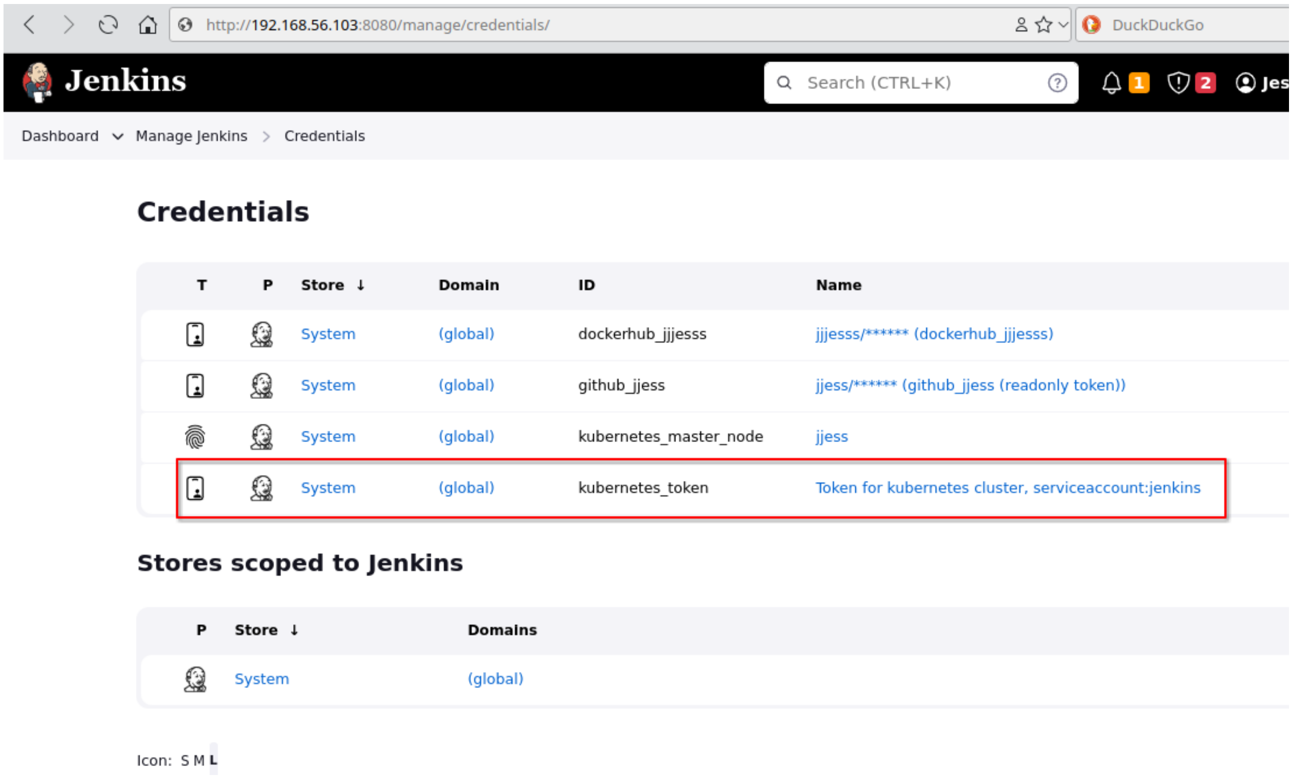
Important in this file:

- name of the serviceaccount: **jenkins**
- ClusterRole **jenkins** with a bunch of specific permissions, only in the **default** namespace.
- ClusterRoleBinding of the ClusterRole **jenkins**, to the serviceaccount **jenkins**.

- Secret created for the service account **jenkins**, with type **service-account-token**. The token generated when applying this yaml file in the cluster can be obtained with the command:

```
kubectl get secret jenkins --output=jsonpath='{.data.token}' | base64 -d

eyJhbGciOiJSUzI1NiIsImtpZCI6Im1qWmdYY3J5YWFmdU9oa2MtSjRMSi1uMWVHMGVHWWdpV0FWZnN1MHVQY3MifQ.eyJpc3
MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJ
kZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImplbmtpbnMiLCJrdWJlcm5ldGVz
LmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiamVua2lucyIsImt1YmVybmV0ZXMuaW8vc2VydmljZ
WFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImFlODZiYjUwLTE0MmMtNDg3My1hYjljLTQwYzEzNDMwZTI4ZiIsInN1Yi
I6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmplbmtpbnMifQ.DVBDMf03rXfyvJBOglJ5ZJVikAOlpOEkwvJsgxSB
4p4639d4wHpvissWu6ZxykbHcZJCGfb-
Kco1HuMiThqKzzurFBJOVrdXJe4AYC3qDepFmrLXaEDyqelSCwW4PcXOELnMAn6Dy-xpM-
Gis2M3017comI4y1SIeanrfCEsmLGgzTTKFwSHMDabMsirThG1DOPJBJ5o9UGLkp2bQAHrceptoElr0ObFOxvmUmu0gHLGQWh
A-21PjP4m1p8Zyoj-PORM25ZcRnEyvte-
MmRZtBGhLIjdzFBC4RXZ_21baQllu6FEiZnUU9FPJKVEMjlYle10eo5mSFIc5lsu72mCZw
```

This token must be configured as a credential in Jenkins:



Credentials

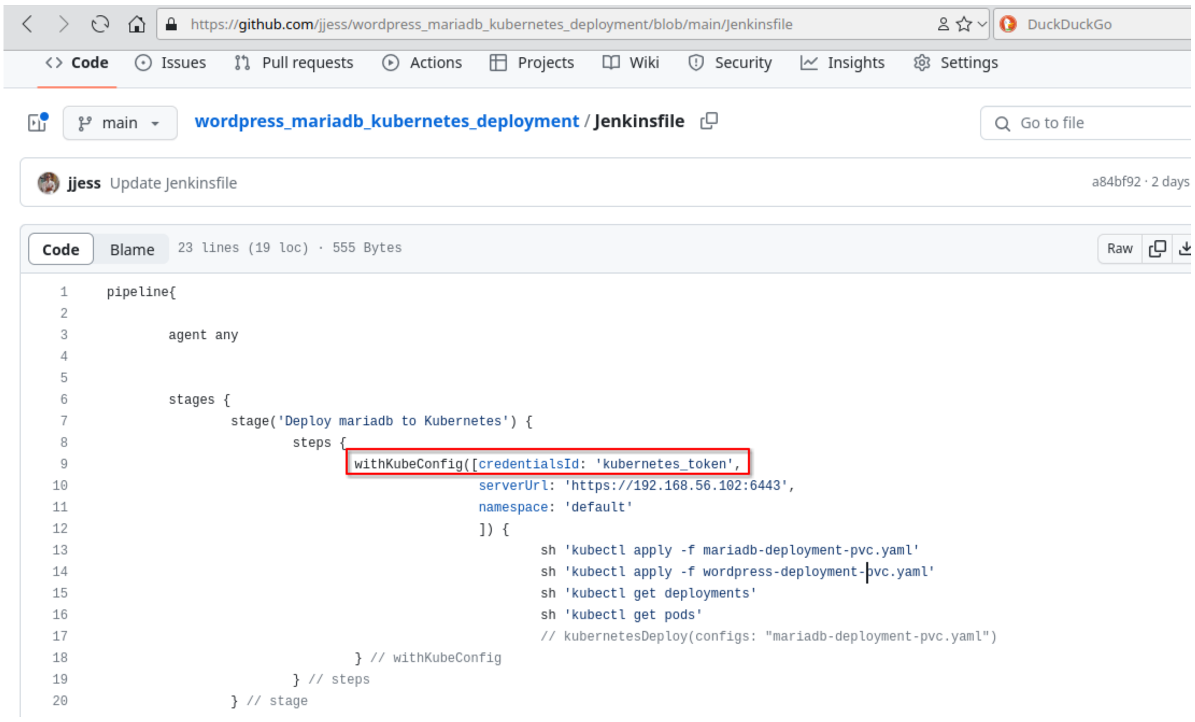
T	P	Store ↓	Domain	ID	Name
		System	(global)	dockerhub_jjjesss	jjjesss/****** (dockerhub_jjjesss)
		System	(global)	github_jjess	jjess/****** (github_jjess (readonly token))
		System	(global)	kubernetes_master_node	jjess
		System	(global)	kubernetes_token	Token for kubernetes cluster, serviceaccount:jenkins

Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

Icon: S M L

In the pipeline, when defining the **withKubeConfig** setup, we will use this token:



https://github.com/jjess/wordpress_mariadb_kubernetes_deployment/blob/main/Jenkinsfile

wordpress_mariadb_kubernetes_deployment / Jenkinsfile

jjess Update Jenkinsfile   a84bf92 · 2 days

23 lines (19 loc) · 555 Bytes

```
pipeline{

        agent any


        stages {
                stage('Deploy mariadb to Kubernetes') {
                        steps {
                                withKubeConfig([credentialsId: 'kubernetes_token',
                                                serverUrl: 'https://192.168.56.102:6443',
                                                namespace: 'default'
                                                ]) {
                                                        sh 'kubectl apply -f mariadb-deployment-pvc.yaml'
                                                        sh 'kubectl apply -f wordpress-deployment-pvc.yaml'
                                                        sh 'kubectl get deployments'
                                                        sh 'kubectl get pods'
                                                        // kubernetesDeploy(configs: "mariadb-deployment-pvc.yaml")
                                } // withKubeConfig
                        } // steps
                } // stage
```