

Course-end Project

Deployment of WordPress Application on Kubernetes

Antonio Gandia

antonio.gandia@vodafone.com

Table of Contents

1. Development Environment.....	3
2. Preparing the Lab Machine	4
2.1. Jenkins.....	4
2.2. Docker	4
2.3. Kubernetes	5
3. Configuring Jenkins Project.....	7
4. Configuration Dockerfile for Building WP Docker Image.....	8
5. Configuration to deploy app in Kubernetes.....	9
6. Running the Pipeline in Jenkins	11
7. Testing the App	14

1. Development Environment

We're going to develop the solution to the project using the following resources:

- SimpliLearn Practice Labs **"DevOps in AWS V2"** provided for the DevOps Certification Training:
 - We're going to use the local (already installed) Jenkins in this machine.
We couldn't manage to deploy and make work the K8s plugins, probably due to different version of Jenkins and the plugins.

Kubernetes





agent Cloud Providers Cluster Management **kubernetes**

This plugin integrates Jenkins with **Kubernetes**

Warning: This plugin is built for Jenkins 2.332.1 or newer. Jenkins will refuse to load this plugin if installed.

Warning: This plugin version may not be safe to use. Please review the following security notices:

- [Improper masking of credentials](#)

Kubernetes	 Failure - Details
Kubernetes Client API	 Success
Kubernetes CLI	 Success
Loading plugin extensions	 Success

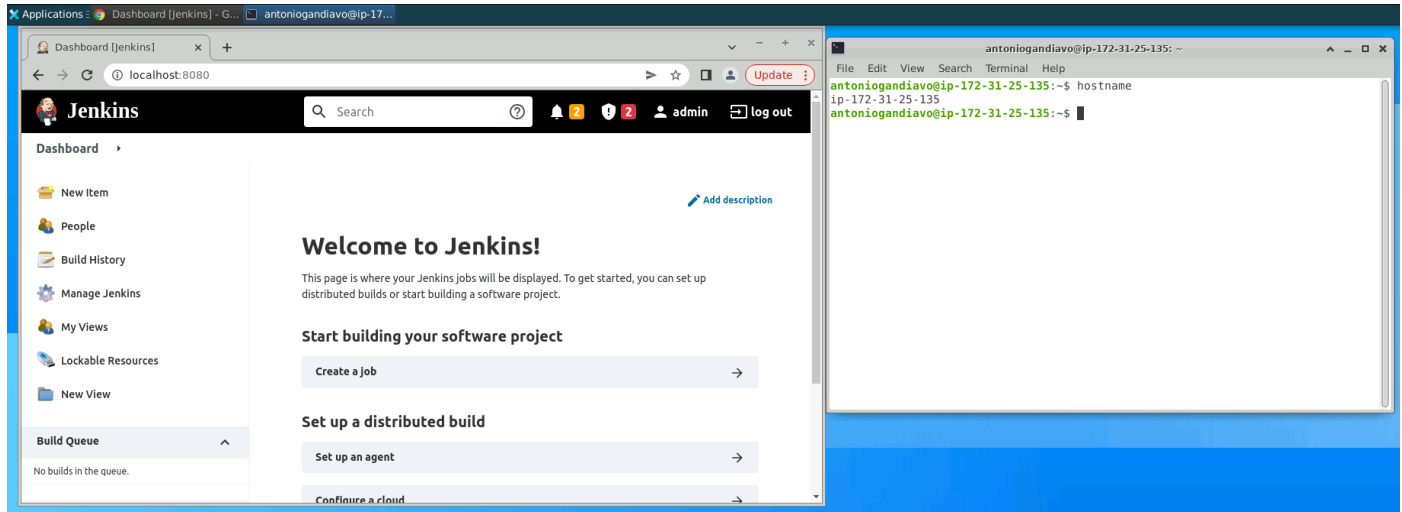
So **we're going to use simply shell commands to connect and interact with K8s**, as it was described during the mentoring session.

- As we couldn't connect this Lab with the "Kubernetes New Version Lab" (in the Container Orchestration with Kubernetes Course), we're going to install and run Kubernetes in the same machine as Jenkins, with only one master and remove taints to allow schedule pods on the master.
- My **personal account of GitHub** to store all the needed configuration files (Jenkins pipeline, Dockerfile, and the Pod's YAML)
- My **personal account of DockerHub** to store (push) the Docker Images created.

2. Preparing the Lab Machine

2.1. Jenkins

Jenkins was already installed and running in the machine.



I added the user “Jenkins” to the docker group to give permissions to build images, and after that, restart Jenkins:

```
sudo usermod -a -G docker jenkins
sudo systemctl restart Jenkins
```

2.2. Docker

Docker was already running... But, as it's recommended, we're going to upgrade to the last version directly from docker:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
```

I also edited the /etc/docker/daemon.json, to add "exec-opts": ["native.cgroupdriver=systemd"]

And the result after restarting the service:

```
antoniogandiavo@ip-172-31-25-135:~$ docker --version
Docker version 20.10.12, build e91ed57
```

2.3. Kubernetes

Kubernetes was also installed but I upgraded to the last version from Kubernetes:

```
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
sudo apt-get update
sudo apt-get install kubelet kubeadm kubectl
```

To allow Kubernetes to pull docker containers, I had to comment out in the file `/etc/containerd/config.toml` the following default line:

```
disabled_plugins = ["cri"]
```

And then restart containerd with: `service containerd restart`

Then, I initialized the kube cluster

```
sudo kubeadm init --ignore-preflight-errors=all
```

Setup the environment variables for the users that are going to be able to manage kubectl:

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

And set up the calico network:

```
sudo kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/calico.yaml
```

I checked, the node was ready. I saw that in the actual version of K8s, there's no "master role":

```
root@ip-172-31-25-135:~# kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-25-135    Ready    control-plane  13m   v1.27.2
```

And then I **removed the taints to allow this single control-plane/master node to be also scheduled** with:

```
kubectl taint node ip-172-31-25-135 node-role.kubernetes.io/control-plane:NoSchedule-
```

Finally, I checked I was able to create and run a pod:

```
root@ip-172-31-25-135:~# kubectl run pod2 --image nginx
pod/pod2 created
root@ip-172-31-25-135:~# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
pod2    1/1     Running   0           5s
root@ip-172-31-25-135:~# kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP          NODE                NOMINATED NODE   READINESS GATES
pod2    1/1     Running   0           16s   192.168.144.4   ip-172-31-25-135    <none>           <none>
root@ip-172-31-25-135:~# kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-25-135    Ready    control-plane  12m   v1.27.2
```

And after that, clean all the testing pods and resources.

We also had to give a valid configuration for user “jenkins” to execute kubectl commands, but as Jenkins doesn’t have sudo privileges I did:

```
sudo cp -i /etc/kubernetes/admin.conf /var/lib/jenkins/.kube/config
id jenkins
## Response uid=126(jenkins) gid=135(jenkins) groups=135(jenkins),998(docker)
sudo chown 126:135 /var/lib/jenkins/.kube/config
```

And check it works:

```
antoniogandiavo@ip-172-31-25-135:~$ sudo su jenkins
jenkins@ip-172-31-25-135:/home/antoniogandiavo$ kubectl get nodes
NAME                STATUS    ROLES    AGE    VERSION
ip-172-31-25-135    Ready     control-plane  23h    v1.27.2
jenkins@ip-172-31-25-135:/home/antoniogandiavo$
```

3. Configuring Jenkins Project

We're going to define Build Project as a Pipeline in Jenkins to do this 3 tasks:

- Connect to my personal Github and fetch the repository
https://github.com/toni31416/Course_DevOps_Deployment_WordPress_Kubernetes.git
This repo contains needed files to complete the rest of the tasks
- Build the customized image of Wordpress app and push it to my personal Dockerhub
- Execute in the Kubernetes cluster the creation of necessary pods and services to run the app (from the customized image of Wordpress and using a MySQL DB)

This is the configuration of the Jenkins Pipeline:

The screenshot shows the Jenkins web interface at `localhost:8080/job/Project_Wordpress_Kubernetes/configure`. The 'Pipeline' tab is selected, showing the 'Definition' section with a 'Pipeline script' editor. The script is a Jenkins Pipeline script with three stages: 'Clone Repo', 'Build WP Docker Image', and 'Deploy in Kubernetes'. The 'Clone Repo' stage uses the 'main' branch of the repository. The 'Build WP Docker Image' stage uses a Docker build command and pushes the image to Docker Hub using a credential named 'dockerhub_password'. The 'Deploy in Kubernetes' stage uses 'kubectl' to delete and create resources from a 'wordpress.yml' file.

```
1 pipeline{
2   agent any
3   stages{
4     stage('Clone Repo'){
5       steps{
6         git branch: 'main', url: 'https://github.com/toni31416/Course_DevOps_Deployment_WordPress_Kubernetes.git'
7       }
8     }
9     stage('Build WP Docker Image'){
10      steps{
11        sh 'docker build -t wordpress .'
12        withCredentials([string(credentialsId: 'dockerhub_password', variable: 'password')]) {
13          sh 'docker login -u toni31416 -p ${password}'
14        }
15        sh 'docker tag wordpress toni31416/wordpress'
16        sh 'docker push toni31416/wordpress'
17      }
18    }
19    stage('Deploy in Kubernetes'){
20      steps{
21        sh 'kubectl delete -f wordpress.yml --ignore-not-found=true'
22        sh 'kubectl create -f wordpress.yml'
23      }
24    }
25  }
26 }
```

Notes:

- In the Clone Repo, I had to specify the branch “main” that now is the old master in GitHub
- For connecting to my personal DockerHub, I had to define the Credentials as a Secret Text with the ID “dockerhub_password”:

The screenshot shows the Jenkins 'Global credentials (unrestricted)' page. It displays a table of credentials. One credential is listed: 'dockerhub_password' of type 'Secret text'.

ID	Name	Kind	Description
dockerhub_password	dockerhub_password	Secret text	

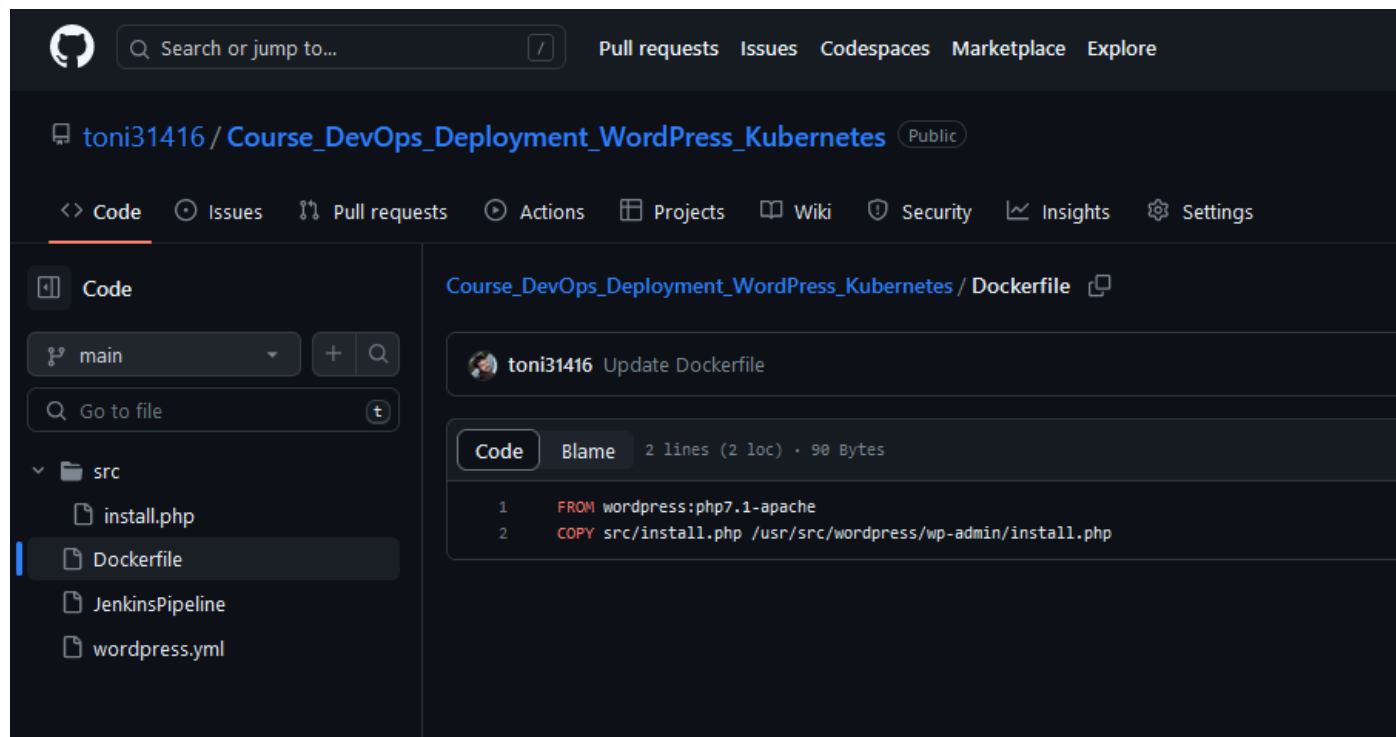
- In the Deploy Kubernetes I chose to delete previous configuration (ignoring if it's not deployed) and then create. Probably in an important production environment I would have chosen another strategy (kube apply & rollout restart)

4. Configuration Dockerfile for Building WP Docker Image

To simulate the build of the application, we're going to make an image with the base of the official Wordpress image in DockerHub.

We're going to overwrite a file (wp-admin/install.php) adding a custom greeting that indicates to the user that it's a customized version.

Here you can see the Dockerfile that is used:

A screenshot of a GitHub repository interface. The repository is named 'toni31416 / Course_DevOps_Deployment_WordPress_Kubernetes' and is public. The 'Code' tab is selected, showing the file structure on the left with 'Dockerfile' highlighted. The main area displays the content of the Dockerfile, which consists of two lines: 'FROM wordpress:php7.1-apache' and 'COPY src/install.php /usr/src/wordpress/wp-admin/install.php'. The commit history shows a recent update by 'toni31416' titled 'Update Dockerfile'.

The new file is also included in src/install.php in the repo.

5. Configuration to deploy app in Kubernetes

The configuration is written in the file wordpress.yml

toni31416 Update wordpress.yml

Code

Blame

88 lines (86 loc) · 1.49 KB

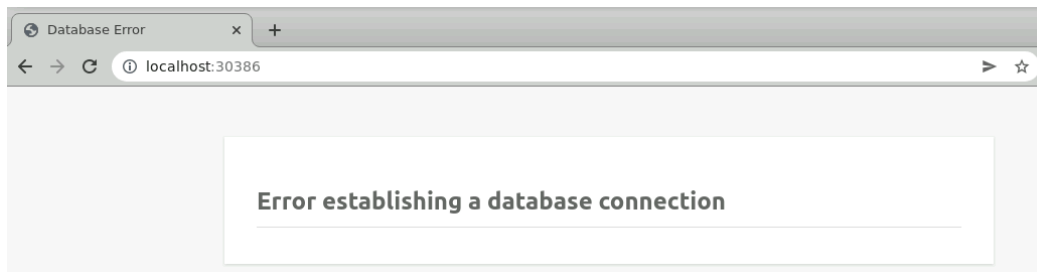
```
1  ---
2  apiVersion: v1
3  kind: Service
4  metadata:
5    name: mysql
6    labels:
7      app: mysql-wordpress
8  spec:
9    ports:
10     - port: 3306
11       targetPort: 3306
12     selector:
13       app: mysql-wordpress
14       product: mysql
15  ---
16  apiVersion: apps/v1
17  kind: Deployment
18  metadata:
19    name: mysql
20    labels:
21      app: mysql-wordpress
22  spec:
23    selector:
24      matchLabels:
25        app: mysql-wordpress
26        product: mysql
27    template:
28      metadata:
29        labels:
30          app: mysql-wordpress
31          product: mysql
32      spec:
33        containers:
34          - image: mysql:5.7
35            name: mysql-container
36            env:
37              - name: MYSQL_DATABASE
38                value: wordpress
39              - name: MYSQL_ROOT_PASSWORD
40                value: password
41
```

```
43  ---
44  apiVersion: v1
45  kind: Service
46  metadata:
47    name: wordpress
48    labels:
49      app: mysql-wordpress
50  spec:
51    ports:
52     - port: 80
53       targetPort: 80
54       nodePort: 30386
55     selector:
56       app: mysql-wordpress
57       tier: frontend
58     type: NodePort
59  ---
60  apiVersion: apps/v1
61  kind: Deployment
62  metadata:
63    name: wordpress
64    labels:
65      app: mysql-wordpress
66  spec:
67    selector:
68      matchLabels:
69        app: mysql-wordpress
70        tier: frontend
71    strategy:
72      type: Recreate
73    template:
74      metadata:
75        labels:
76          app: mysql-wordpress
77          tier: frontend
78      spec:
79        containers:
80          - image: toni31416/wordpress
81            name: wordpress
82            env:
83              - name: WORDPRESS_DB_HOST
84                value: mysql
85              - name: WORDPRESS_DB_USER
86                value: root
87              - name: WORDPRESS_DB_PASSWORD
88                value: password
```

As we can see, there is a couple of resources (Service + Deployment) for Wordpress and for MySQL (it's going to be the storage for WP).

Notes:

- We had to use a specific version of mysql (version 5.7). Otherwise, I got a problem with the encryption of the communication between WP and MySQL:



- As a demonstration project, with the configuration of MySQL IMHO is enough. But in a production environment I would add:
 - A Persistent Volume for storage in MySQL
 - A Secret object to store a real password for the MySQL database
- Regarding the Service for wordpress, I created a nodePort type, and I statically assigned the external port 30386 to make testing easier.

Here you can see several printouts of the Kubernetes cluster running our app:

```
antonioyadiavo@ip-172-31-25-135:~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/mysql-7c4f989788-btjtx	1/1	Running	0	9m27s
pod/wordpress-76db7885b7-nsdm2	1/1	Running	0	9m27s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	23h
service/mysql	ClusterIP	10.105.206.205	<none>	3306/TCP	9m27s
service/wordpress	NodePort	10.104.2.159	<none>	80:30386/TCP	9m27s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/mysql	1/1	1	1	9m27s
deployment.apps/wordpress	1/1	1	1	9m27s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/mysql-7c4f989788	1	1	1	9m27s
replicaset.apps/wordpress-76db7885b7	1	1	1	9m27s

6. Running the Pipeline in Jenkins

To build the complete application configured as described before, in Jenkins we go to our Pipeline and select “Build Now”. We got a clean execution without any errors:

The screenshot shows the Jenkins web interface for the pipeline 'Project_Wordpress_Kubernetes'. The left sidebar contains a 'Build Now' button. The main area displays the 'Stage View' for the pipeline, showing a table of stage execution times for builds #31 to #37. The stages are 'Clone Repo', 'Build WP Docker Image', and 'Deploy in Kubernetes'.

Build	Clone Repo	Build WP Docker Image	Deploy in Kubernetes
#37	252ms	1s	844ms
#36	275ms	1s	845ms
#35	316ms	1s	844ms
#34	370ms	1s	1s

And here we can see the Console Output

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Project_Wordpress_Kubernetes
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Clone Repo)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Project_Wordpress_Kubernetes/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/toni31416/Course_DevOps_Deployment_WordPress_Kubernetes.git #
timeout=10
Fetching upstream changes from https://github.com/toni31416/Course_DevOps_Deployment_WordPress_Kubernetes.git
> git --version # timeout=10
> git --version # 'git version 2.34.1'
> git fetch --tags --force --progress --
https://github.com/toni31416/Course_DevOps_Deployment_WordPress_Kubernetes.git +refs/heads/*:refs/remotes/origin/* #
timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 89193c548052d8ebe50bc4178736c62534e0e624 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 89193c548052d8ebe50bc4178736c62534e0e624 # timeout=10
> git branch -a -v --no-abbrev # timeout=10
```

```

> git branch -D main # timeout=10
> git checkout -b main 89193c548052d8e50bc4178736c62534e0e624 # timeout=10
Commit message: "Update wordpress.yml"
> git rev-list --no-walk 89193c548052d8e50bc4178736c62534e0e624 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build WP Docker Image)
[Pipeline] sh
+ docker build -t wordpress .
#2 [internal] load .dockerignore
#2 sha256:92486c9cc98c4d7b601ddafb792f1327458ce6b0d0b05589e05a8cf6e5f3e425
#2 DONE 0.0s

#1 [internal] load build definition from Dockerfile
#1 sha256:e6f5db1d598f32c9636f4b5870100871794527a501557e365477c566bb048e3b
#1 DONE 0.0s

#2 [internal] load .dockerignore
#2 sha256:92486c9cc98c4d7b601ddafb792f1327458ce6b0d0b05589e05a8cf6e5f3e425
#2 transferring context: 2B done
#2 DONE 0.0s

#1 [internal] load build definition from Dockerfile
#1 sha256:e6f5db1d598f32c9636f4b5870100871794527a501557e365477c566bb048e3b
#1 transferring dockerfile: 132B done
#1 DONE 0.0s

#4 [auth] library/wordpress:pull token for registry-1.docker.io
#4 sha256:2a9d74f15a6fce9552e493f81d6b05036b8996380c309bf0ceab6a0be94f6245
#4 DONE 0.0s

#3 [internal] load metadata for docker.io/library/wordpress:php7.1-apache
#3 sha256:6b1f5f25d72eab13dec51e03d6282453dc8e4b6bdba5d08908978c5a4318da74
#3 DONE 0.2s

#7 [1/2] FROM docker.io/library/wordpress:php7.1-
apache@sha256:a83b97d11bf719a1e3a441ede3e3efc2396cf419a8581c24b37d61d5bf7c9ee
#7 sha256:9c06c956473a6ac21298af82ccae044bfbdcc3e6da457186cdc0b7cab754d1a
#7 DONE 0.0s

#6 [internal] load build context
#6 sha256:06bd1d5e98db681f4ff5e4284e97b50dc7540f6f6ed87952989f8e85adbb81a5
#6 transferring context: 61B done
#6 DONE 0.0s

#5 [2/2] COPY src/install.php /usr/src/wordpress/wp-admin/install.php
#5 sha256:dfbd7b47375d877a45d845cc318c81d7428e938bff38634520866c307d12fdf6
#5 CACHED

#8 exporting to image
#8 sha256:6b4af803dfbdf89d9e8a1a0e11e1c13529185d76f463f5a32ab1bf36ea2a5368
#8 exporting layers done
#8 writing image sha256:11f55a435dbddf2758f36396d2b81ba0a38f793a13dc31afcf9e94010bb23d92 done
#8 naming to docker.io/library/wordpress:latest done
#8 DONE 0.0s
[Pipeline] withCredentials
Masking supported pattern matches of $password
[Pipeline] {
[Pipeline] sh
+ docker login -u toni31416 -p ****
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /var/lib/jenkins/.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] // withCredentials
[Pipeline] sh
+ docker tag wordpress toni31416/wordpress
[Pipeline] sh
+ docker push toni31416/wordpress
Using default tag: latest
The push refers to repository [docker.io/toni31416/wordpress]
a6efbb3f28e6: Preparing
87cda92b5749: Preparing
6a764028a11a: Preparing
d08582be360a: Preparing

```

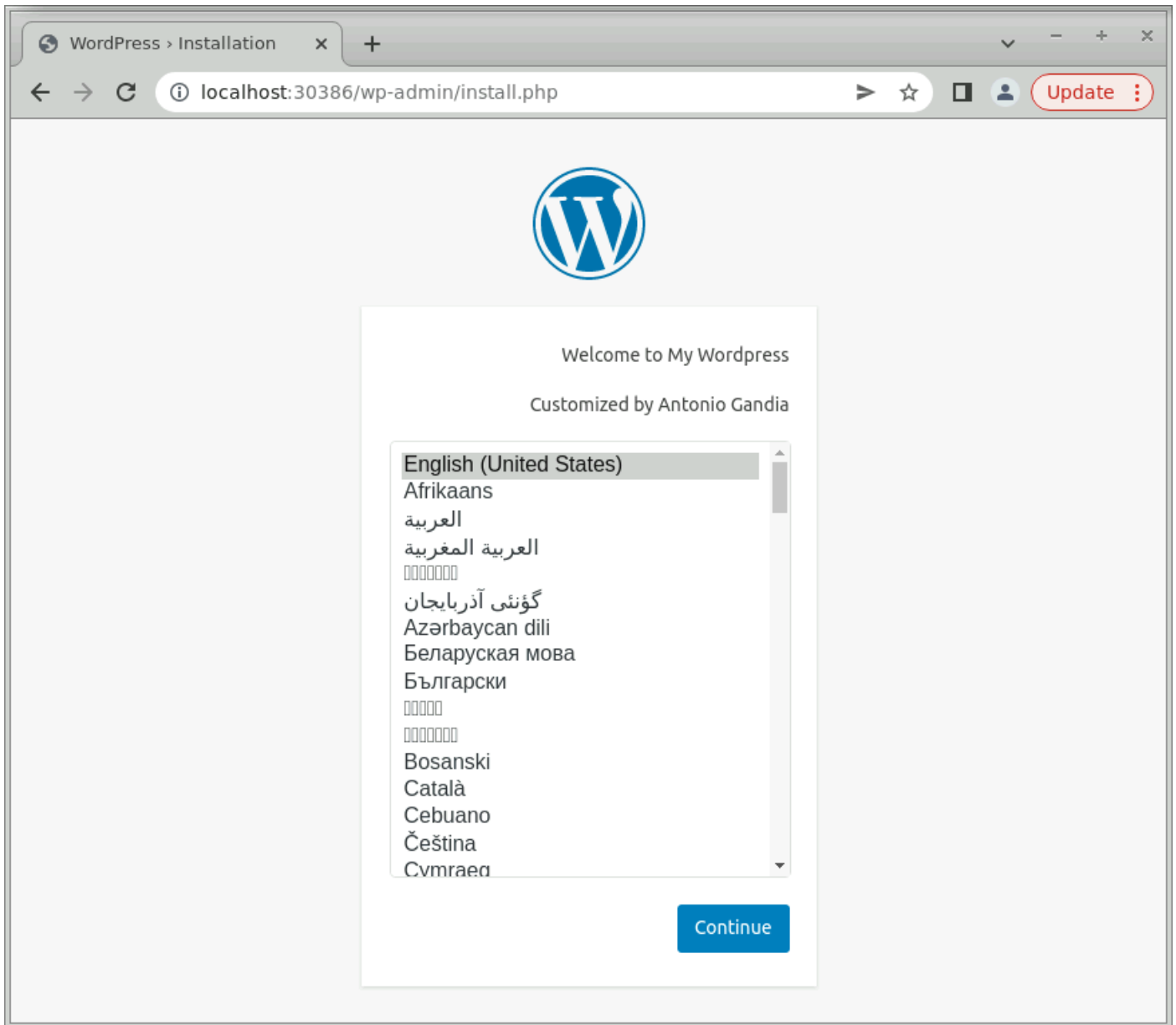
```

686d006c14a7: Preparing
60bc7d9b5271: Preparing
489d523737b2: Preparing
a3ca3612aa01: Preparing
0817436a8f49: Preparing
3385a426f542: Preparing
35c986c7de74: Preparing
53bab0663330: Preparing
606c36b65880: Preparing
ab99fcc1a184: Preparing
a3ca3612aa01: Waiting
9691e5d7a4c7: Preparing
0817436a8f49: Waiting
6a4d393f0795: Preparing
e38834ac7561: Preparing
ec64f555d498: Preparing
840f3f414cf6: Preparing
17fce12edef0: Preparing
831c5620387f: Preparing
3385a426f542: Waiting
35c986c7de74: Waiting
53bab0663330: Waiting
606c36b65880: Waiting
ab99fcc1a184: Waiting
9691e5d7a4c7: Waiting
6a4d393f0795: Waiting
e38834ac7561: Waiting
ec64f555d498: Waiting
840f3f414cf6: Waiting
17fce12edef0: Waiting
831c5620387f: Waiting
60bc7d9b5271: Waiting
489d523737b2: Waiting
a6efbb3f28e6: Layer already exists
6a764028a11a: Layer already exists
686d006c14a7: Layer already exists
d08582be360a: Layer already exists
87cda92b5749: Layer already exists
489d523737b2: Layer already exists
60bc7d9b5271: Layer already exists
a3ca3612aa01: Layer already exists
0817436a8f49: Layer already exists
3385a426f542: Layer already exists
35c986c7de74: Layer already exists
606c36b65880: Layer already exists
53bab0663330: Layer already exists
ab99fcc1a184: Layer already exists
9691e5d7a4c7: Layer already exists
6a4d393f0795: Layer already exists
840f3f414cf6: Layer already exists
ec64f555d498: Layer already exists
e38834ac7561: Layer already exists
17fce12edef0: Layer already exists
831c5620387f: Layer already exists
latest: digest: sha256:efd66b859fd98bb21c74f79db818afaed0977f47590fc2f6eb2fe319fa79d23 size: 4709
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy in Kubernetes)
[Pipeline] sh
+ kubectl delete -f wordpress.yml --ignore-not-found=true
service "mysql" deleted
deployment.apps "mysql" deleted
service "wordpress" deleted
deployment.apps "wordpress" deleted
[Pipeline] sh
+ kubectl create -f wordpress.yml
service/mysql created
deployment.apps/mysql created
service/wordpress created
deployment.apps/wordpress created
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

7. Testing the App


Finally if we open a web browser on the localhost:30386 we can reach the WP App and test that everything works properly. We can also see that it was correctly customized as we desired:



WordPress › Installation

localhost:30386/wp-admin/install.php?step=1

Update



Welcome to My Wordpress

Customized by Antonio Gandia

Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Testing Jenkins Docker & K8s

Username

toni31416

Username can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password

Strong

Show

Important: You will need this password to log in. Please store it in a secure location.

Your Email

antonio.gandia@vodafone.com


Double-check your email address before continuing.

Search Engine Visibility

☐ Discourage search engines from indexing this site

It is up to search engines to honor this request.

Install WordPress



Welcome to My Wordpress

Customized by Antonio Gandia

Success!

WordPress has been installed. Thank you, and enjoy!


Username

toni31416

Password

Your chosen password.

Log In



Username or Email Address

toni31416

Password

Show

Remember Me

Log In

Lost your password?

[← Back to Testing Jenkins Docker & K8s](#)

