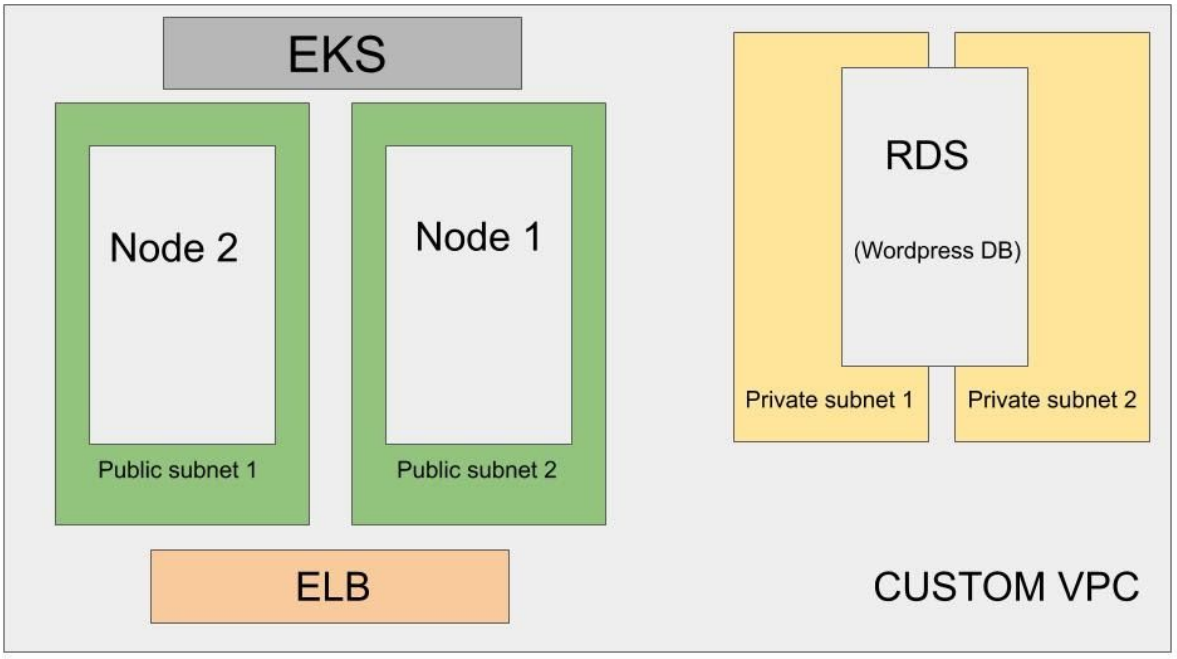
Wordpress EKS AWS : Automation Deployment

1. INTRODUCTION:

The aims of this document is to help end-users to facilitate the automation deployment of wordpress on EKS (AWS) environment.

TARGET ARCHITECTURE:

Below the conception of the Wordpress cluster on AWS EKS, components and choices are described later in this documents.
Please note that several resources are not shown in this figure (security groups, internet gateway, IAM roles, ..)



Ghassen CHARNI
ghassen.cherni@gmail.com

2. WORKFLOW:

The workflow of our project is composed by 3 pipelines (jobs):

terraform_aws_eks:

This is the main job of our workflow, it builds all the infrastructure:

Networking:

- Create the Custom VPC: *It's highly recommended to use a custom VPC instead of the default one in a production environment (network segmentation, security rules, ..)*
- The internet gateway
- Route tables and route tables associations:
- Public subnets, Private subnets and the RDS subnet group: *AWS recommends to always run databases in a private network*

Security:

- Create the public and the private security groups :
- Security group rules (ingress and egress)
- Create a private key and a self signed certificate and upload them to the AWS.

EKS and RDS:

- Create the EKS:
- Create the EKS node group
- Create and attache needed roles for the EKS cluster
- Create the RDS instance: *It's recommended to use RDS instead of running a Database on PODS with persistent volume as the RDS offers several services and ensure more HA then POD (ex Multi AZ RDS, ReadReplica, …). We are able to create a Cluster of Msql RDS but in this project we will keep only one instance for all pods,*

wp_custom_docker:

Allows to build a docker image from a Dockerfile (wordpress 4.8 apache scratch) and push the new build on Gitlab registry:

When running the job manually or by committing change (by webhook jenkins if configured) it creates new Docker images (ghassen-devopstt:last and ghassen-devops:v01.build_number) then pushed to Gitlab registry and triggers the pipeline **wordpress_k8s**.

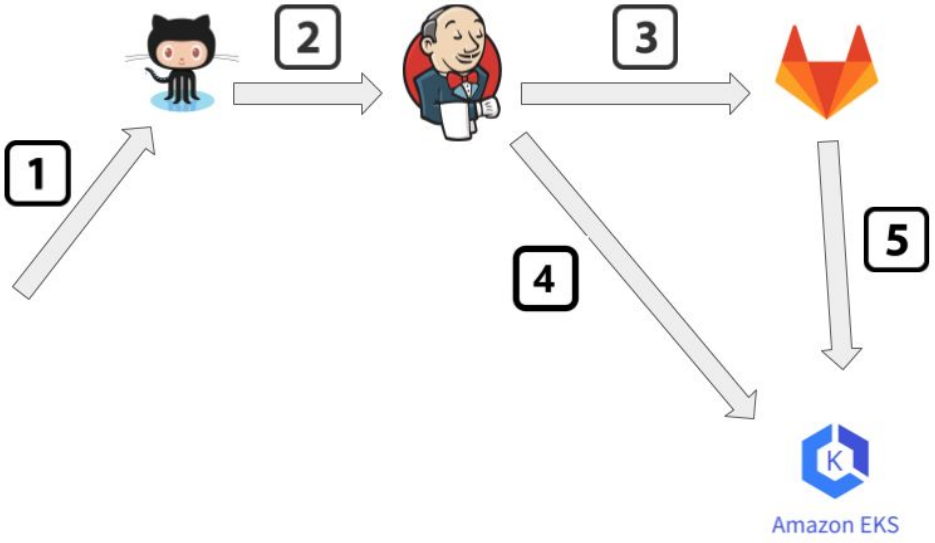
wordpress_k8s

This job allows to build the wordpress on EKS:

- Create the persistent storage (EKS automatically build an EBS volume): *for /var/www/html directory*
- Create wordpress deployment (based on the pushed "ghassen-devopstt" image)

- Create and attach the configMap: *Contains RDS connexion paramètres (Wordpress DB)*
- Create the Loadbalance service: *EKS automatically creates an ALB in front of the two nodes (of the two public subnets)*

CONTINUOUS DEPLOYMENT

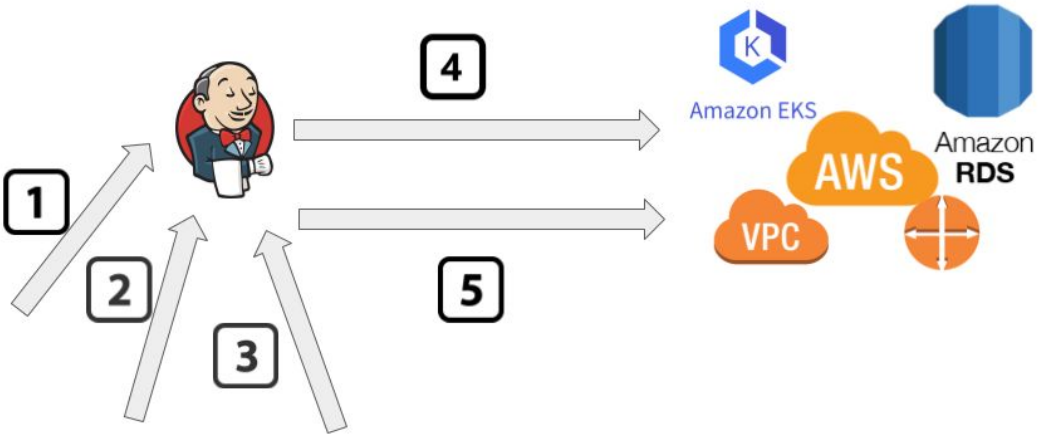


1. Commiting changes by a developer (in our case changing Dockerfile)
2. Github (webhook) will trigger the **wp_custom_docker** job
3. **wp_custom_docker** builds the new image from the committed Dockerfile
4. After pushing the image, **wp_custom_docker** trigger the **wordpress_k8s job**
5. **wordpress_k8s** job pull the new image from gitlab and deploy wordpress on EKS

Ghassen CHARNI
ghassen.cherni@gmail.com

3. INSTALLATION:

HOW-IT-WORKS:



1- Building the Jenkins server:

The first step of the installation is to run the script “init.sh”, this script uses terraform and ansible in order to:

- Deploy a micro EC2 instance : *reachable only by the admin IP address (SSH and HTTP 8080)*
- Install Jenkins
- Install Docker, terraform, kubectl: *Allows jenkins to build the infrastructure and to deploy the wordpress cluster*

2- Creating pipelines on Jenkins:

The second step of “init.sh” is to create the 3 pipelines described above based on Jenkinsfile:

- **terraform_aws_eks**
- **wp_custom_docker**
- **wordpress_k8s**

3 and 4 - Running terraform_aws_eks job:

After adding all pipelines, we launch the **terraform_aws_eks** manually in order to deploy all needed infrastructure to deploy EKS in AWS.

5- Deploy the wordess on EKS:

The last step consists to push a new docker image on Gitlab image using **wp_custom_docker** job which also triggers **wordpress_k8s** and deploy wordpress.

INSTALLATION STEPS:

Requirements:

- We need a **Linux machine** in order to run the first “init.sh” script to build our JENKINS instance.
- **terraform** (**version 11**) and **aws-iam-authenticator** binaries must be installed
- AWS account (ex Free tier) to create a user with access key (AWS access key ID and secret) with “AdministratorAccess” permission.
- **AWS access key ID and secret** key are configured on the Linux machine (for example by adding “.aws/credentials” file or exporting AWS_ACCESS_KEY_ID/AWS_SECRET_ACCESS_KEY)
- A Gitlab account with public docker image registry (Not needed, please use the default one, account name is hardcoded at this version)
- **Ansible** installed and configured (tested with 2.9.1 version)
- **Java** must be installed

Ghassen CHARNI
ghassen.cherni@gmail.com

1- Install and configure Jenkins:

- git clone https://github.com/ghassencherni/deploy_jenkins.git
- cd deploy_jenkins
- Available variables are listed in **terraform.tfvars,** please change it according to your needs or keep the default values:

```
# AWS region where to run our Jenkins instance
aws_region = "eu-west-3"

# The name of the public ssh key stored in AWS
key_name = "artifakt_key"

# The public key for ssh connection
public_key_path = "~/.ssh/id_rsa.pub"

# The private SSH key, used by ansible to configure the Jenkins instance
private_key_path= "~/.ssh/id_rsa"

# The size of the Jenkins instance, micro is sufficient for our deployment
instance_type = "t2.micro"

# Please use the default one, account name is hardcoded at this version
gitlab_account = "myartifakt@outlook.fr"

# Please use the default one, account name is hardcoded at this version
gitlab_password = "artifakt"

aws_access_key_id = "PUT-YOR-ACCESS-ID"

aws_secret_access_key = "PUT-YOUR-SECRET-ACCESS-KEY"
```

Please note that you can keep all default value except aws_access_key_id and aws_secret_access_key, it's rquired to put your AWS user account credentials (aws user to deploy the Jenkins instance and all other resources on AWS)

- Give permissions to scripts init.sh and destroy.sh
  chmod +x ./binaries/init.sh ./binaries/destroy.sh

- Run the init.sh script to install Jenkins
  ./binaries/init.sh

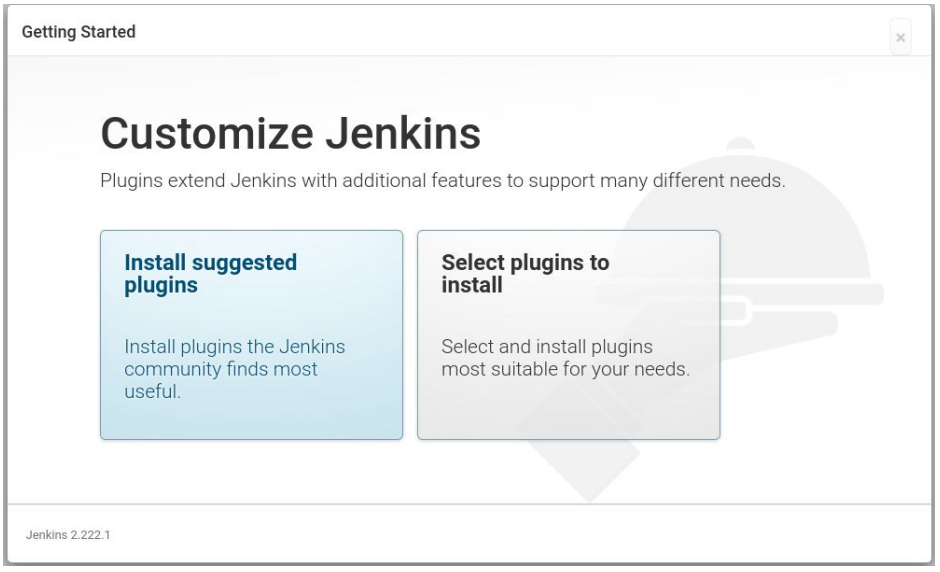
- During the installation you will be asked to make some manual configurations

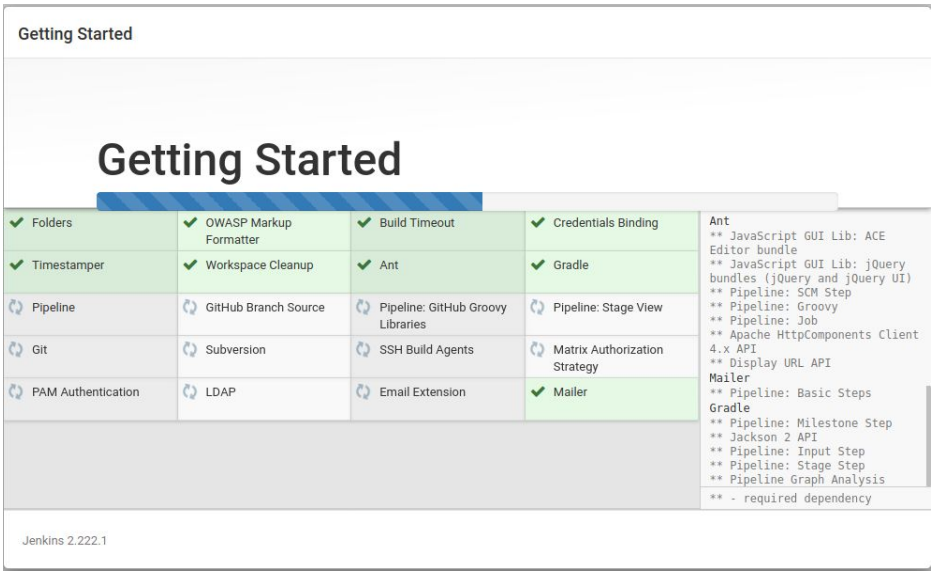
```
##########################################################################################################
#
# Please follow these steps in order to finish the Jenkins deployment:
#
# 1- Open the web browser and connect to : http://ec2-15-188-207-5.eu-west-3.compute.amazonaws.com:8080
#
# 2- Put the Initial Admin Password : 5be01a84faf640a0bca25edc7cced350 then click on continue
#
# 3- Select "Install suggested plugins" and wait until plugins installation finished
#
# 4- Continue with the "admin" without adding "First Admin User"
#
# 5- Keep the Jenkins URL as the default one ( http://ec2-15-188-207-5.eu-west-3.compute.amazonaws.com:8080 ) then click on save and finish.

#
# 6- Click on "Start using Jenkins"
#
# 7- Once Jenkins is ready, press "Yes" to continue
##########################################################################################################
If you have activated the admin / the default plugins, tape 'Yes' to continue configuring Jenkins
```

This Manual configuration are described in the output of the script and also is shown here in details:

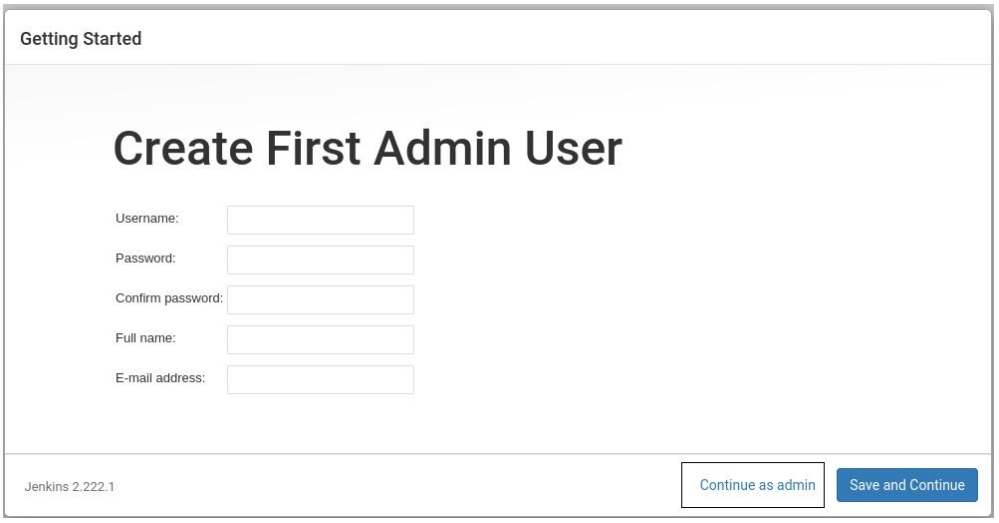
- Open the web browser, connect to the mentioned URL and put the provided “initial admin password” then click continue

- Select “Install suggested plugins” and wait until plugins installation finished



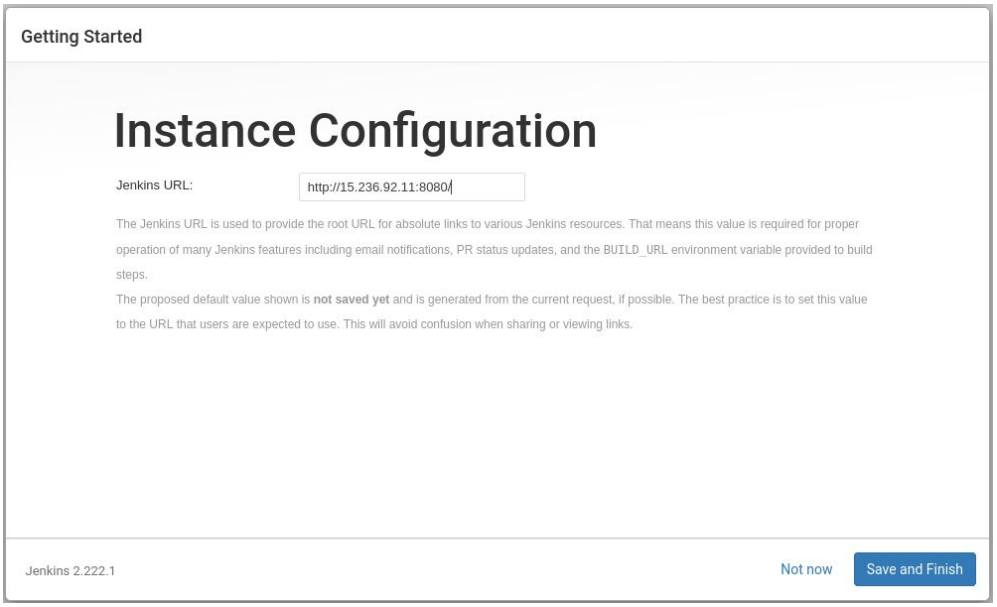


- Continue with the “admin” without adding “First Admin User”

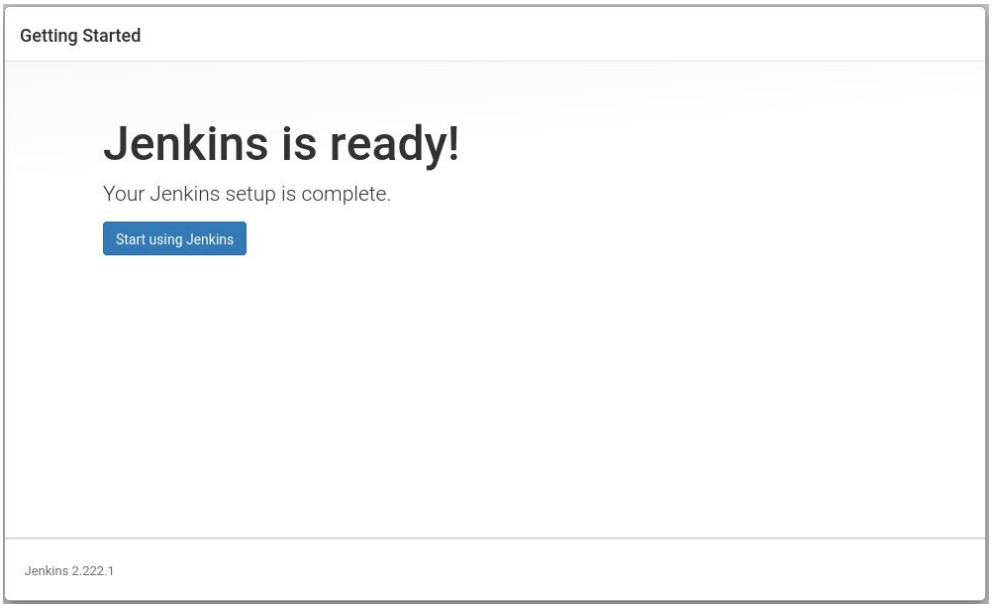


Ghassen CHARNI
ghassen.cherni@gmail.com

- Keep the Jenkins URL as the default one (in my cas http://15.236.92.11:8080) then click on save and finish.



- Click on “Start using Jenkins”



Once Jenkins is ready, switch to the previous terminal and press “Yes” to continue

➔ This step will add credentials and create the 3 pipelines.

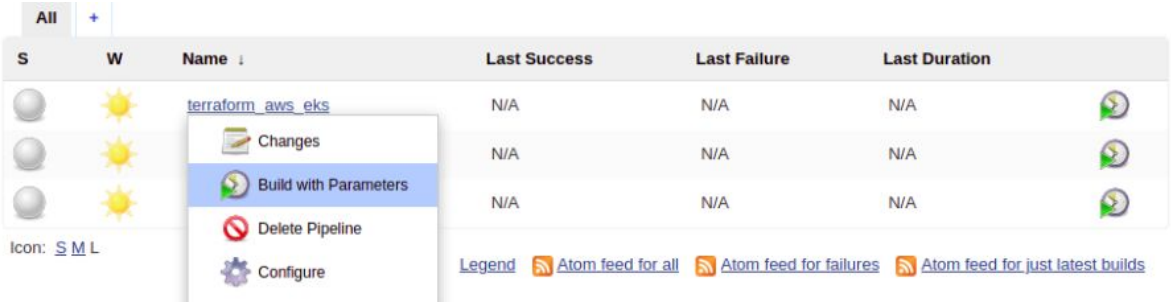
Ghassen CHARNI
ghassen.cherni@gmail.com

2- Running the Pipelines

Once the init.sh script is finished, connect to the Jenkins URL mentioned above and connect as “admin” user with the same password (Initial Admin Password)

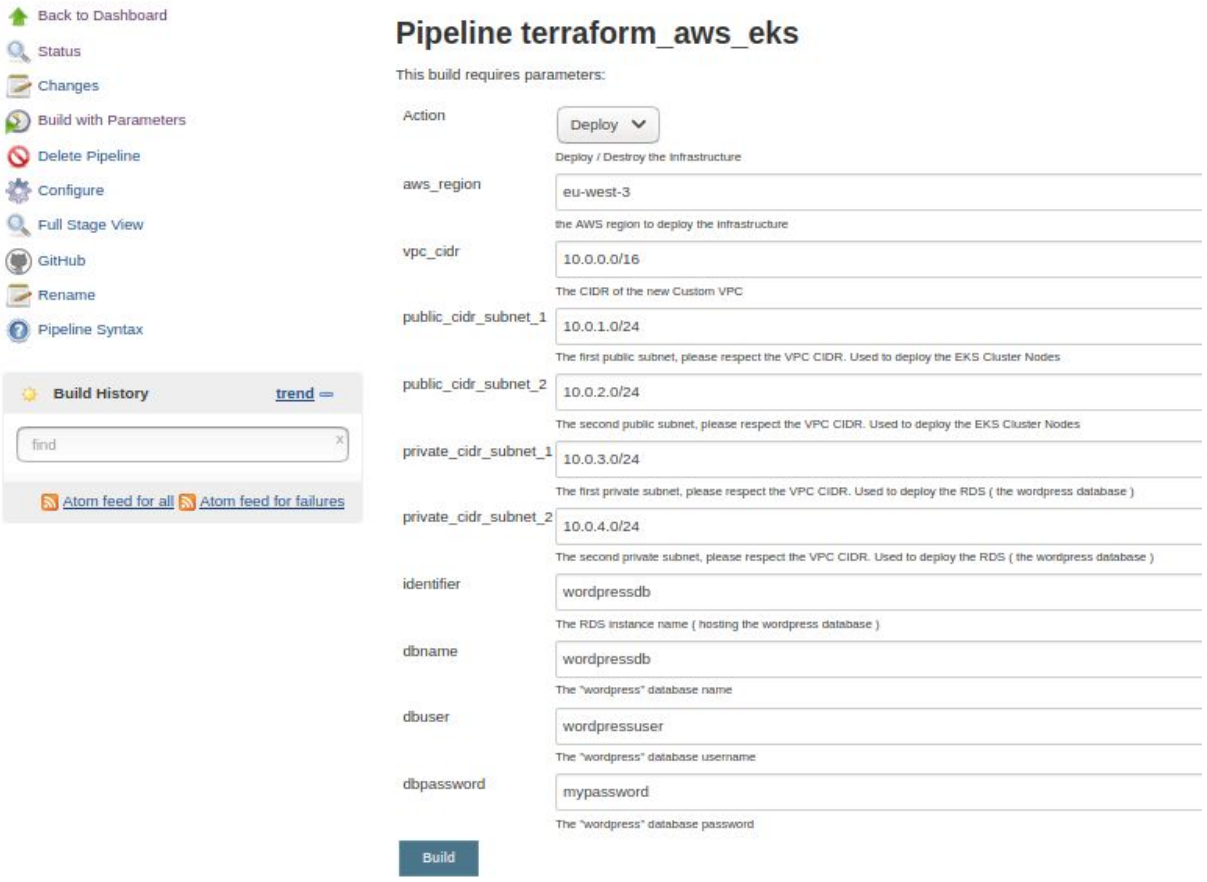
2-1- Running terraform_aws_eks:

The first pipeline to build is the terraform_aws_eks, select the job and run build with parameters

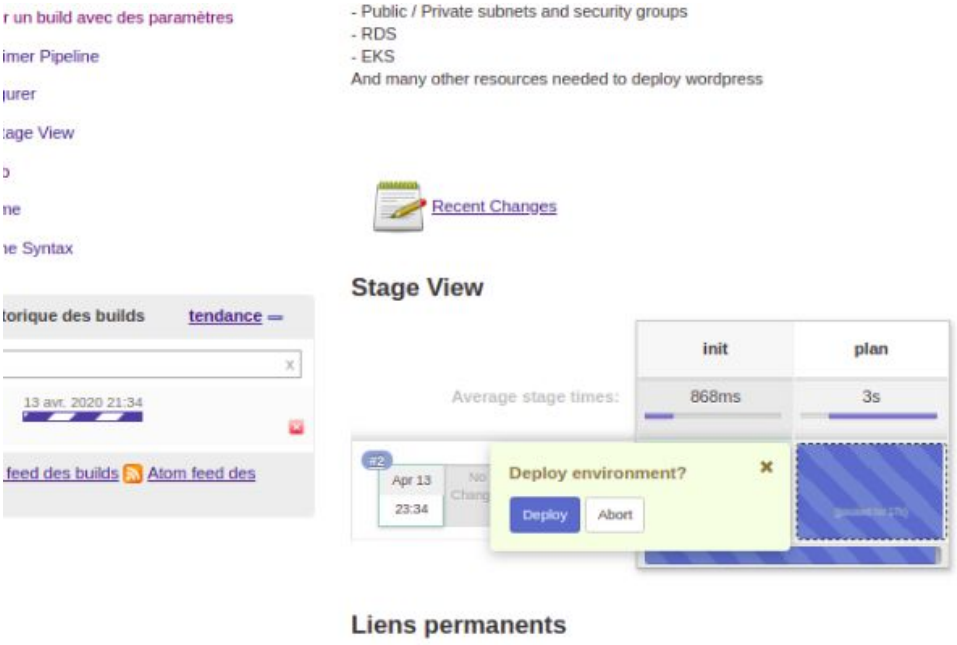


Ghassen CHARNI
ghassen.cherni@gmail.com

Set Action to “Deploy” and change default parameters if needed then click on “Build”



Confirm the deployment by clickIng on “Deploy”



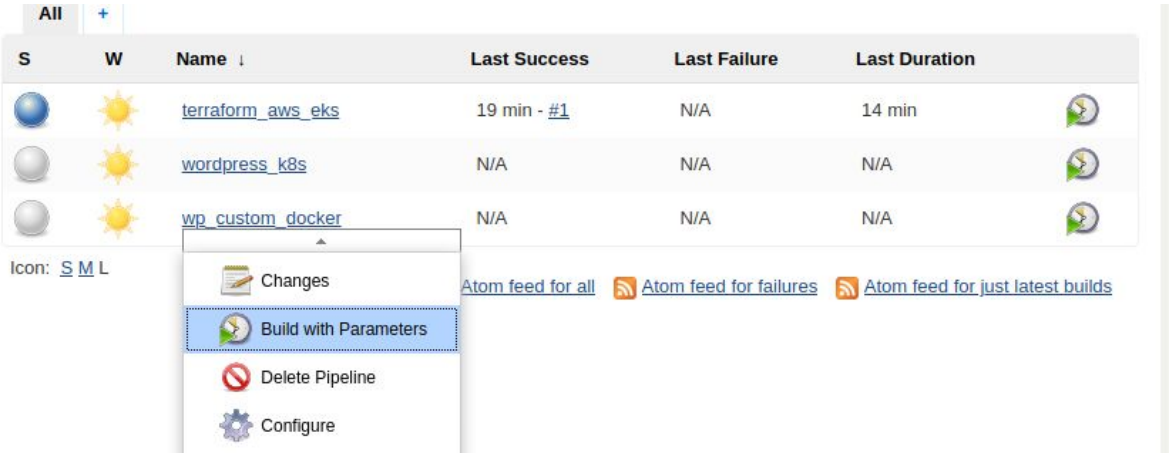
Ghassen CHARNI
ghassen.cherni@gmail.com

This will deploy all the infrastructure on AWS : VPC, EKS, RDS, Subnets, Internet gateway, security tables, … Once terraform_aws_eks, the next step is to build the Wordpress cluster on EKS by running wp_custom_docker job.

2-2- Running wp_custom_docker and wordpress_k8s (build the first Wordpress version on the EKS cluster) :

Now the infrastructure is ready and we can start pushing our first image and building wordpress on EKS.

Select wp_custom_docker job, click build with parameters now , set a wordpress version or keep the default one (4.8.2) then click on “Build”



➔ This job will push the new image on Gitlab then trigger wordpress_k8s to deploy it on EKS

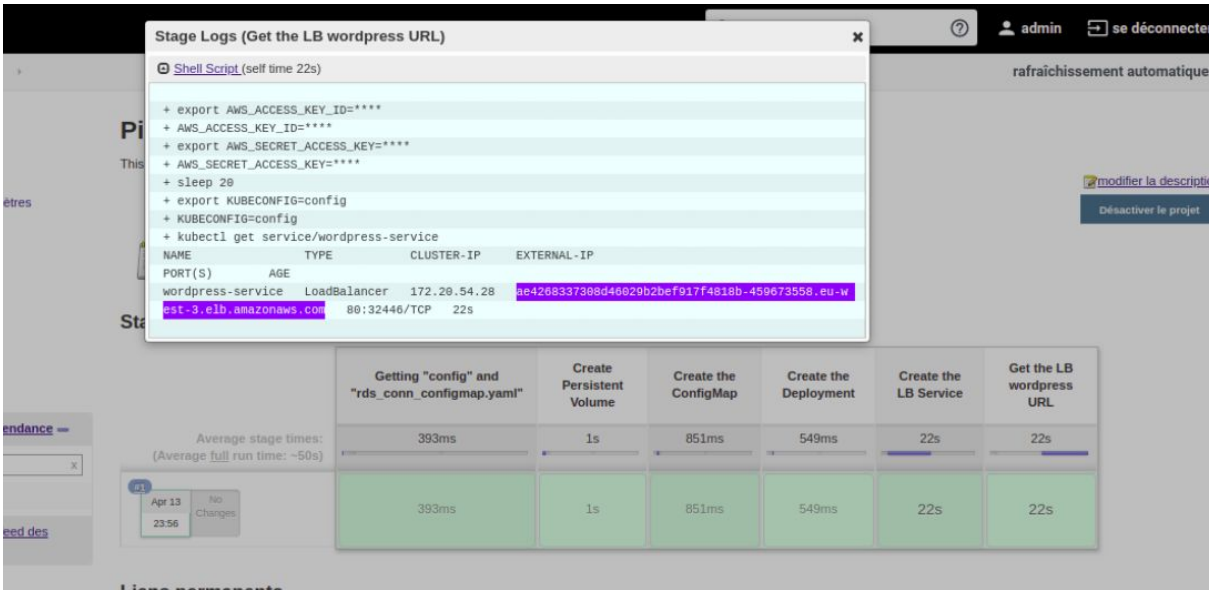
CHECK THE DEPLOYMENT:

Checking that the new Docker image was pushed on the [Gitlab registry]

myartifakt/ghassen-devopstt tags

Tag	Image ID	Compressed Size	Last Updated
latest	58b21ac27	132.60 MiB	19 minutes ago
v1.0.1	58b21ac27	132.60 MiB	19 minutes ago

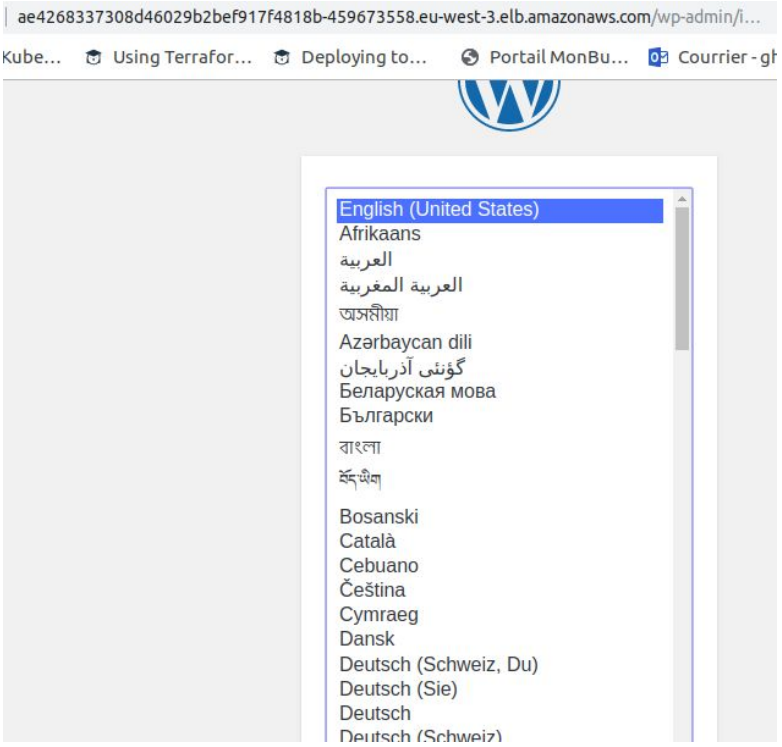
Go to the wordpress_k8s job, check the last build deployment log and you will find the URL of our wordpress (Get the LB wordpress URL)



=> Please note that you can use HTTP or HTTPS to connect to your Wordpress

And finally check the URL on the browser

Ghassen CHARNI
ghassen.cherni@gmail.com



UPDATE WORDPRESS WITH ZERO DOWN TIME:

Let's assume that your first wordpress deploy was made with the default version 4.8.2, now we will try to upgrade to version 5.0.1 and will check that our website still running without a downtime.

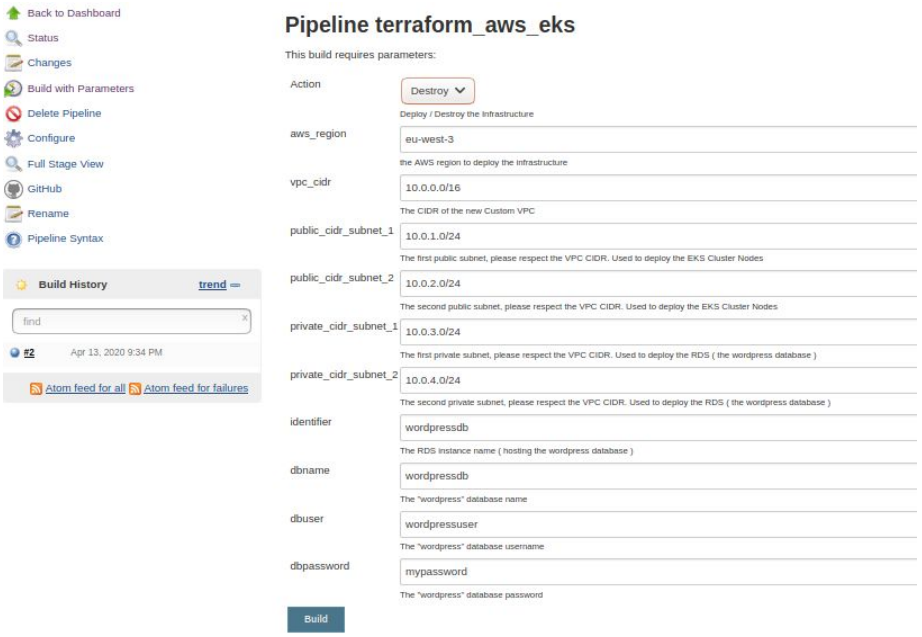
1- Select the wordpress_k8s job
2- Click on build with parameters
3- Put the new version (5.0.1 for example)

4- Click on build and turn back to your Wordpress URL and check that the webiste is running without a down time.

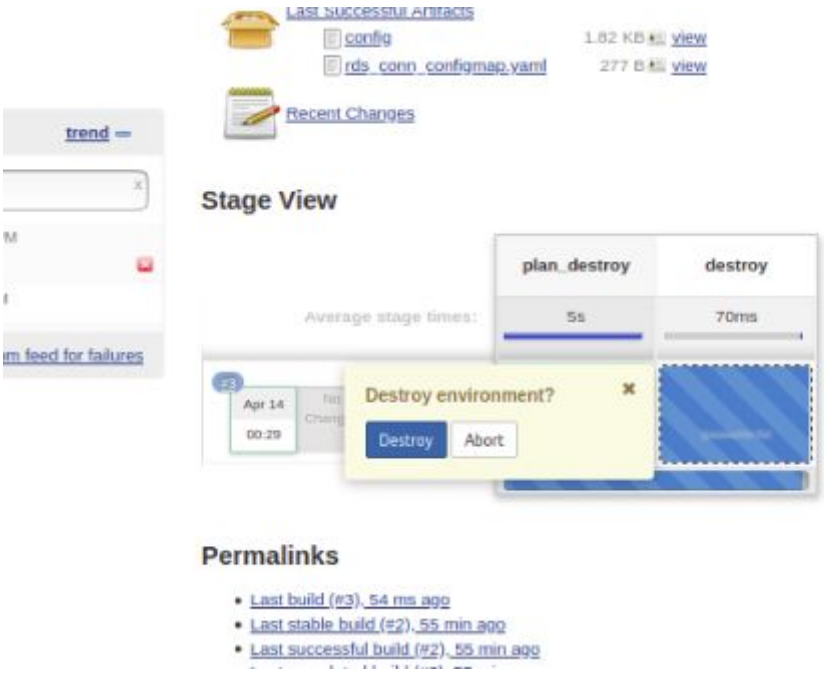
REMOVING AND DESTROYING THE INFRASTRUCTURE:

1- Destroying the AWS infrastructure :

Go to terraform_aws_eks job, run build with parameters then select Destroy instead of Deploy.



Confirm the “Distroy”



Ghassen CHARNI
ghassen.cherni@gmail.com

2- Removing The jenkins instance:

Turn back to the deploy_jenkins directory and run :

```
./binaries/destroy.sh
```

- ➔ This script will remove the Jenkins instance from AWS

Ghassen CHARNI
ghassen.cherni@gmail.com