

Wordpress EKS AWS : Automation Deployment

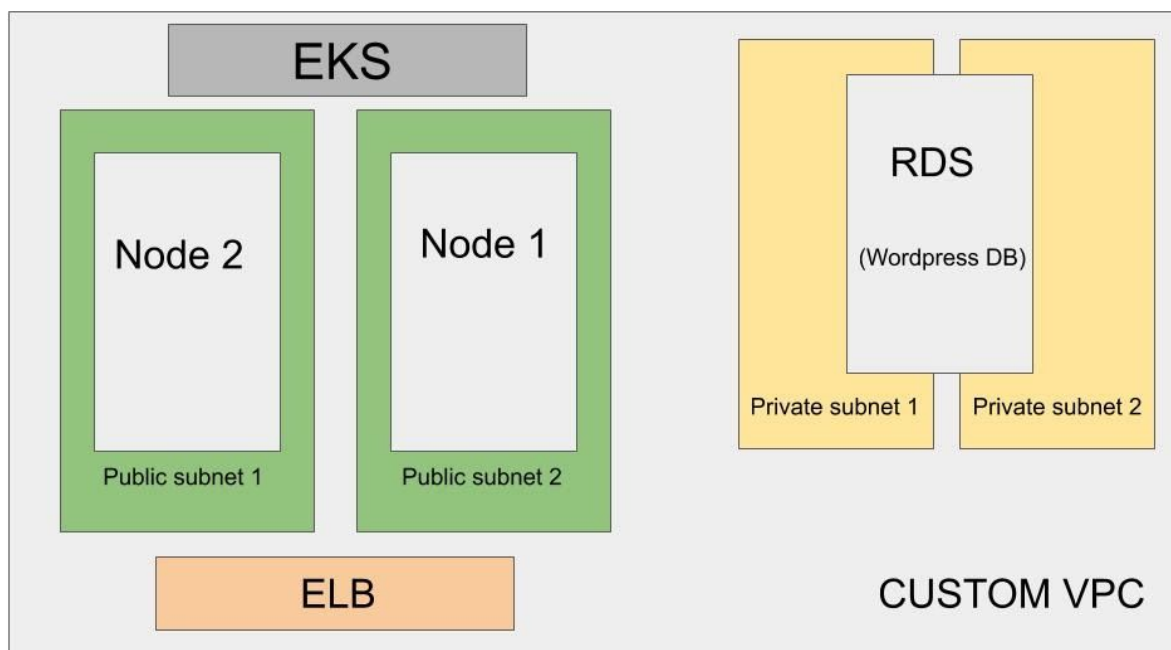
1. INTRODUCTION:

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, ..)



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**.

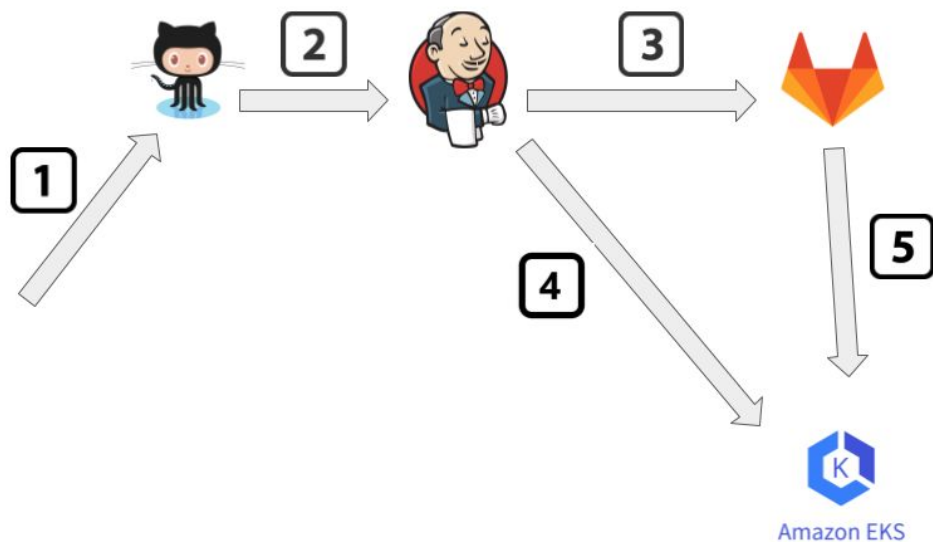
[wordpress_k8s](#)

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)*

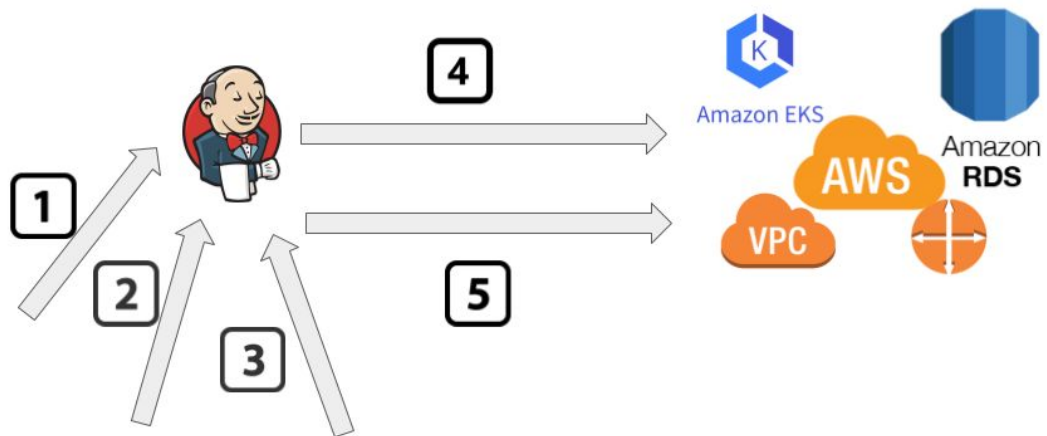
CONTINUOUS DEPLOYMENT



1. Committing 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

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 wordress 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

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"
```

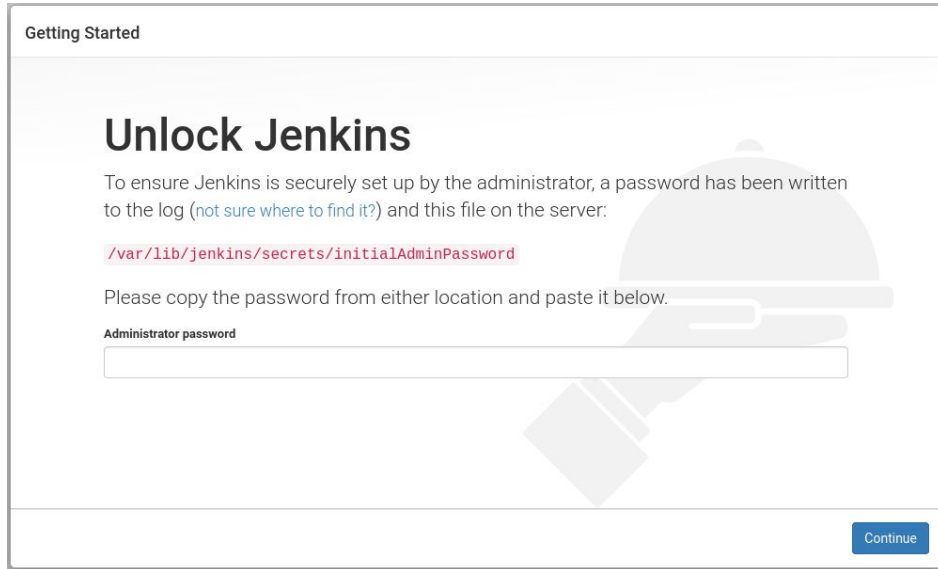
Please note that you can keep all default value except `aws_access_key_id` and `aws_secret_access_key`, it's required to put your AWS user account credentials (aws user to deploy the Jenkins instance and all other resources on AWS)

- 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 : 5be01a84faf640a0bca25edc7cccd350 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
```

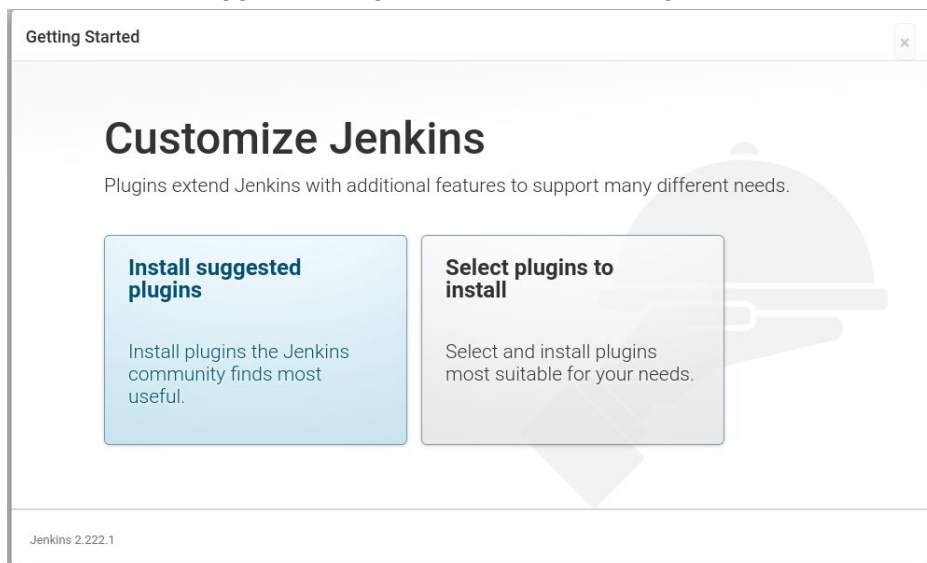
This Manual configuration are described in the output of the script and also is shown here in details:

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



The screenshot shows the 'Getting Started' window of Jenkins. The title is 'Unlock Jenkins'. Below the title, it says: '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:'. A code snippet is shown: `/var/lib/jenkins/secrets/initialAdminPassword`. Below this, it says: 'Please copy the password from either location and paste it below.'. There is a text input field labeled 'Administrator password'. At the bottom right, there is a blue button labeled 'Continue'.

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



The screenshot shows the 'Getting Started' window of Jenkins. The title is 'Customize Jenkins'. Below the title, it says: 'Plugins extend Jenkins with additional features to support many different needs.'. There are two main options: 'Install suggested plugins' (highlighted in blue) and 'Select plugins to install' (highlighted in grey). The 'Install suggested plugins' option says: 'Install plugins the Jenkins community finds most useful.'. The 'Select plugins to install' option says: 'Select and install plugins most suitable for your needs.'. At the bottom left, it says: 'Jenkins 2.222.1'.

Getting Started

Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding	Ant ** JavaScript GUI Lib: ACE Editor bundle ** JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) ** Pipeline: SCM Step ** Pipeline: Groovy ** Pipeline: Job ** Apache HttpComponents Client 4.x API ** Display URL API Mailer ** Pipeline: Basic Steps Gradle ** Pipeline: Milestone Step ** Jackson 2 API ** Pipeline: Input Step ** Pipeline: Stage Step ** Pipeline Graph Analysis ** - required dependency
✓ Timestampers	✓ Workspace Cleanup	✓ Ant	✓ Gradle	
🔗 Pipeline	🔗 GitHub Branch Source	🔗 Pipeline: GitHub Groovy Libraries	🔗 Pipeline: Stage View	
🔗 Git	🔗 Subversion	🔗 SSH Build Agents	🔗 Matrix Authorization Strategy	
🔗 PAM Authentication	🔗 LDAP	🔗 Email Extension	✓ Mailer	

Jenkins 2.222.1

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

Getting Started

Create First Admin User

Username:
 Password:
 Confirm password:
 Full name:
 E-mail address:

Jenkins 2.222.1

Continue as admin

Save and Continue

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

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.222.1

[Not now](#) [Save and Finish](#)

- Click on “Start using Jenkins”

Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

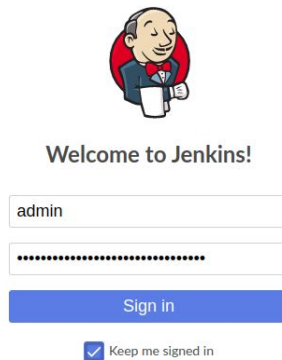
Jenkins 2.222.1

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

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

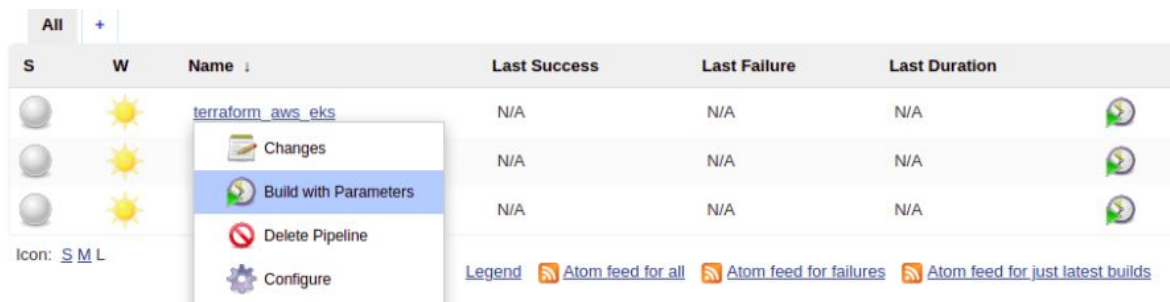
2- Running the Pipelines

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



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

[Back to Dashboard](#)
[Status](#)
[Changes](#)
[Build with Parameters](#)
[Delete Pipeline](#)
[Configure](#)
[Full Stage View](#)
[GitHub](#)
[Rename](#)
[Pipeline Syntax](#)

Build History

[trend](#)

[Atom feed for all](#)
[Atom feed for failures](#)

Pipeline terraform_aws_eks

This build requires parameters:

Action

Deploy

Deploy / Destroy the Infrastructure

aws_region

eu-west-3

the AWS region to deploy the Infrastructure

vpc_cidr

10.0.0.0/16

The CIDR of the new Custom VPC

public_cidr_subnet_1

10.0.1.0/24

The first public subnet, please respect the VPC CIDR. Used to deploy the EKS Cluster Nodes

public_cidr_subnet_2

10.0.2.0/24

The second public subnet, please respect the VPC CIDR. Used to deploy the EKS Cluster Nodes

private_cidr_subnet_1

10.0.3.0/24

The first private subnet, please respect the VPC CIDR. Used to deploy the RDS (the wordpress database)

private_cidr_subnet_2

10.0.4.0/24

The second private subnet, please respect the VPC CIDR. Used to deploy the RDS (the wordpress database)

identifier

wordpressdb

The RDS instance name (hosting the wordpress database)

dbname

wordpressdb

The "wordpress" database name

dbuser

wordpressuser

The "wordpress" database username

dbpassword

mypassword

The "wordpress" database password

Build

Confirm the deployment by clicking on “Deploy”

[r un build avec des paramètres](#)
[imer Pipeline](#)
[jurer](#)
[age View](#)

[ne](#)
[ie Syntax](#)

- Public / Private subnets and security groups
- RDS
- EKS

And many other resources needed to deploy wordpress

[Recent Changes](#)

Stage View

	init	plan
Average stage times:	868ms	3s

Apr 13 23:34

No Change

Deploy environment?

Deploy

Abort

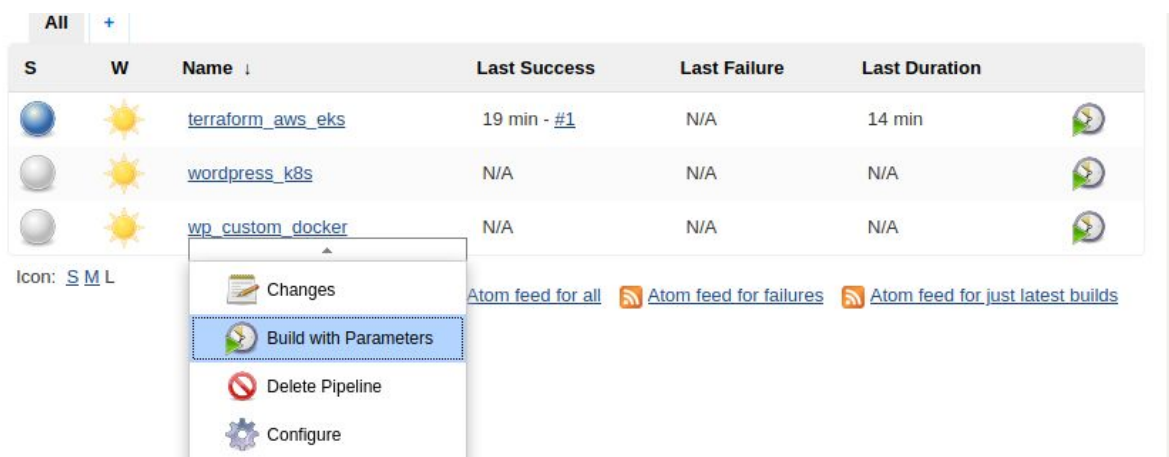
Liens permanents

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"



S	W	Name ↓	Last Success	Last Failure	Last Duration
		terraform_aws_eks	19 min - #1	N/A	14 min
		wordpress_k8s	N/A	N/A	N/A
		wp_custom_docker	N/A	N/A	N/A

Icon: [S](#) [M](#) [L](#)

- Changes
- Build with Parameters**
- Delete Pipeline
- Configure

[Atom feed for all](#) [Atom feed for failures](#) [Atom feed for just latest builds](#)

Pipeline wp_custom_docker

This build requires parameters:

wp_version

The wordpress version to build with our new image

[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)

The screenshot shows a CI/CD pipeline interface. A modal window titled "Stage Logs (Get the LB wordpress URL)" is open, displaying the output of a "Shell Script" stage. The logs show the export of AWS credentials, a sleep command, and the execution of a kubectl command to get the service URL. Below the logs, a progress bar shows the duration of various stages: "Getting 'config' and 'rds_conn_configmap.yaml'" (393ms), "Create Persistent Volume" (1s), "Create the ConfigMap" (851ms), "Create the Deployment" (549ms), "Create the LB Service" (22s), and "Get the LB wordpress URL" (22s). The background interface includes a user profile "admin", a "se déconnecter" button, and a "Désactiver le projet" button.

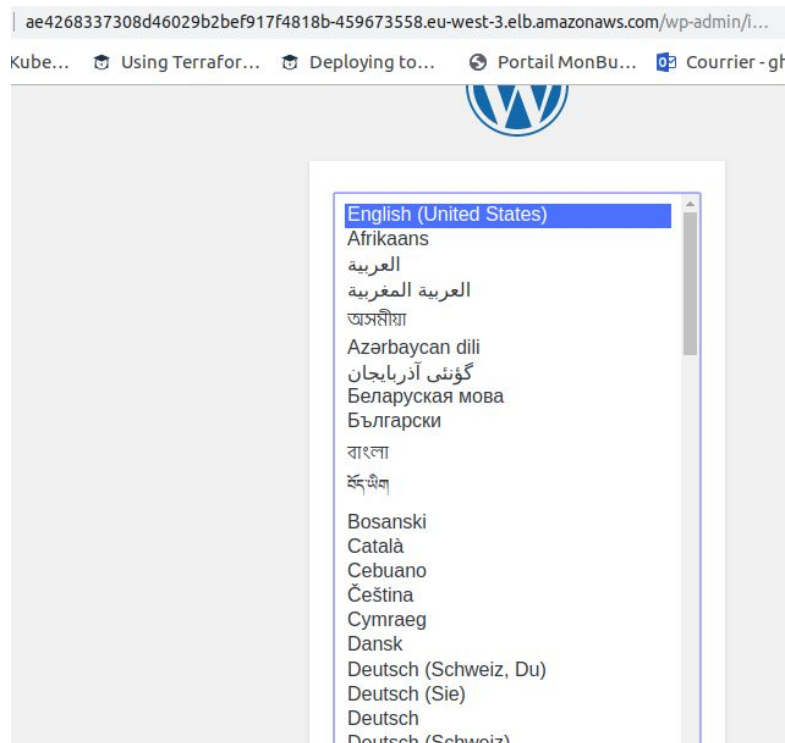
```

+ export AWS_ACCESS_KEY_ID=****
+ AWS_ACCESS_KEY_ID=****
+ export AWS_SECRET_ACCESS_KEY=****
+ AWS_SECRET_ACCESS_KEY=****
+ sleep 20
+ export KUBECONFIG=config
+ KUBECONFIG=config
+ kubectl get service/wordpress-service
NAME                TYPE          CLUSTER-IP   EXTERNAL-IP
wordpress-service   LoadBalancer 172.20.54.28  ae4268337308d46020b2bef017f4818b-450673558.eu-w
est-3.elb.amazonaws.com 80:32446/TCP 22s
  
```

Stage	Duration
Getting "config" and "rds_conn_configmap.yaml"	393ms
Create Persistent Volume	1s
Create the ConfigMap	851ms
Create the Deployment	549ms
Create the LB Service	22s
Get the LB wordpress URL	22s

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

And finally check the URL on the browser



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)

Pipeline wp_custom_docker

This build requires parameters:

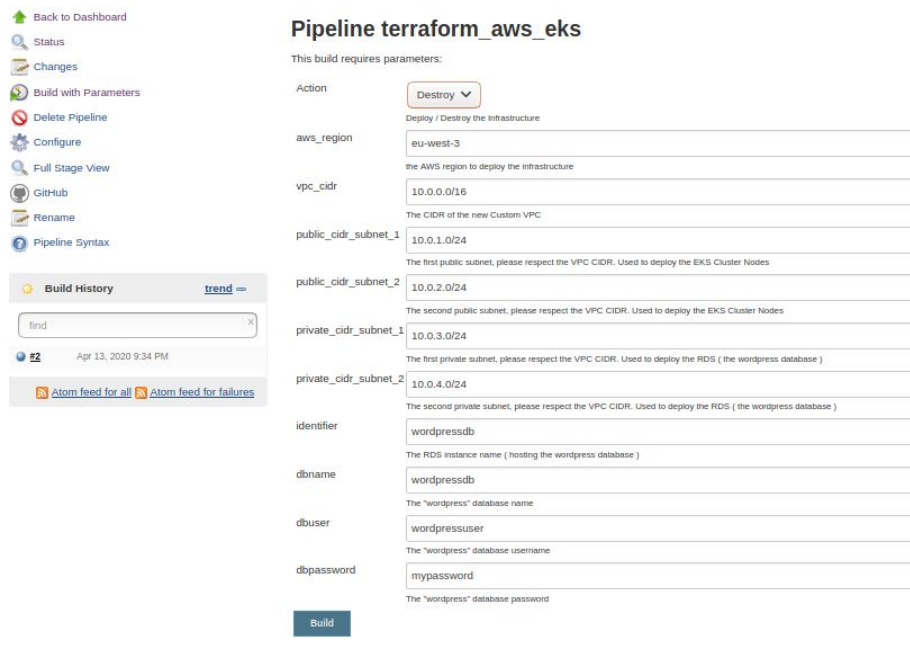
wp_version	<input type="text" value="5.0.1"/>
The wordpress version to build with our new image	
<input type="button" value="Build"/>	

- 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.



Pipeline terraform_aws_eks

This build requires parameters:

Action: **Destroy**

Deploy / Destroy the Infrastructure

aws_region: eu-west-3
the AWS region to deploy the infrastructure

vpc_cidr: 10.0.0.0/16
The CIDR of the new Custom VPC

public_cidr_subnet_1: 10.0.1.0/24
The first public subnet, please respect the VPC CIDR. Used to deploy the EKS Cluster Nodes

public_cidr_subnet_2: 10.0.2.0/24
The second public subnet, please respect the VPC CIDR. Used to deploy the EKS Cluster Nodes

private_cidr_subnet_1: 10.0.3.0/24
The first private subnet, please respect the VPC CIDR. Used to deploy the RDS (the wordpress database)

private_cidr_subnet_2: 10.0.4.0/24
The second private subnet, please respect the VPC CIDR. Used to deploy the RDS (the wordpress database)

identifier: wordpressdb
The RDS instance name (hosting the wordpress database)

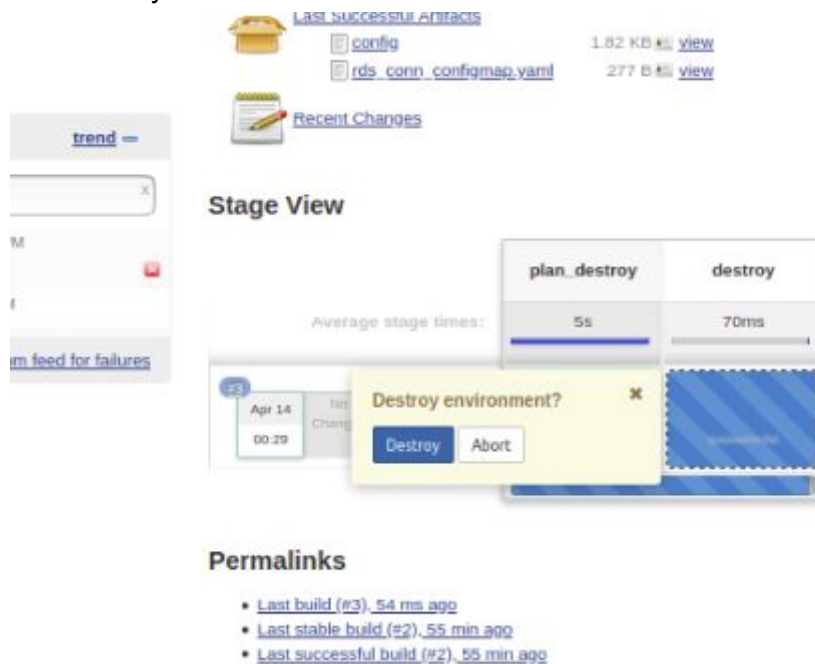
dbname: wordpressdb
The "wordpress" database name

dbuser: wordpressuser
The "wordpress" database username

dbpassword: mypassword
The "wordpress" database password

Build

Confirm the “Distroy”



Last Successful Artifacts

Artifact	Size	View
config	1.62 KB	view
rds_conn_configmap.yaml	277 B	view

Recent Changes

Stage View

Average stage times:

Stage	Time
plan_destroy	5s
destroy	70ms

Destroy environment?

Destroy **Abort**

Permalinks

- [Last build \(#3\), 54 ms ago](#)
- [Last stable build \(#2\), 55 min ago](#)
- [Last successful build \(#2\), 55 min ago](#)

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