**Titre professionnel**

**Administrateur DevOps**

**Dossier Projet**

1. **Contexte**

Gozaimass est une entreprise des services qui fournit entre autres des solutions de type DevOps à destination d’autres entreprises.

Lorsqu’une entreprise utilise des services DevOps, c’est à dire qu’elle déploie automatiquement des services sur des serveurs de nature différentes (cloud, VM, local, …etc.). Elle est généralement confrontée aux problèmes de **sécurité des données**, de **disponibilité** et **d’intégrité**, ce qui l’expose à des pénalités et des risques importants. Ce qui impacte l’activité de l’entreprise et aussi la satisfaction de leurs clients. Ainsi, Gozaimass développe une application qui a pour objectif de faciliter la vie de ses clients (particuliers, entreprises, …etc.) en mettant à leur disposition une application DevOps de déploiement rapide et automatique des services sur un serveur cible et aussi qui garantit **l’intégrité**, **la sauvegarde** et **la restauration automatique des données**. Cette application est destinée à toute demande dans le domaine de l’IT souhaitant utiliser des services DevOps (développement et Opérations) dans son exercice.

**L’impact de cette application serait un gain de temps et d’argent pour les clients.**

Ce projet a pour objectif de déployer rapidement et efficacement vos services sur n’importe quelle plateforme (local, développement ou production) et sur n’importe quelle plateforme (cloud, VM en local, …etc.) n’importe où et n’importe quand dans un environnement sécurisé (DevSecOps).

Les services seront également surveillés et monitorés pour garantir la disponibilité et la qualité de nos services afin d’obtenir la meilleure performance de nos solutions.

Le projet Proteus est en cours de développement. **Etant stagiaire pendant 2 mois avec l’entreprise Gozaimass sur ce projet**, j’ai eu l’occasion de participer à la mise en place des services DevOps, à la sauvegarde et restauration automatique, monitoring et supervision de ces services.

Etant donné que le projet est en cours de développement et pas d’environnement de production disponible, j’ai décidé de préparer une plateforme DevOps sur des **VM Cloud AWS** en y installant les outils nécessaires pour mettre en place des services et y appliquer les compétences que j’ai acquis durant mon stage (déploiement, backup, restore, monitoring, …etc.)

Comme service j’ai choisi de déployer un site Prestashop avec docker en réponse au besoin suivant :

La mise en place rapide d’une plateforme sécurisée, facile à déployer et à configurer pour un site eCommerce.

Une plateforme qu’on peut déployer rapidement sur un cloud, VM ou en local.

Possibilité de faire un backup (sauvegarde) des données et de restaurer les données dans le site à n’importe quel moment.

Superviser et monitorer la plateforme (serveurs et site) en temps réel afin de détecter les anomalies à corriger et garantir la performance du site eCommerce.

Personnellement j’ai eu l’occasion de travailler sur un site PrestaShop déployée classiquement et je trouve l’énorme intérêt de faciliter la mise en place d’un site eCommerce en toute rapidité avec des services DevOps. Le gain de temps et de l’argent que cela offre toute en garantissant l’intégrité de données, la sécurité et la supervision ouvre la porte et motive les clients à monter leurs sites en toute confiance.

1. **Architecture**

Dans ce projet les outils DevOps suivants sont utilisés pour installer les services suivants :

1. **Terraform**

Terraform est un outil d’infrastructure as Code qui permet de créer, de modifier et de versionner l’infrastructure de manière sûre et efficace. Cela inclut des composants de bas niveau tels que les instances, le stockage et la mise en réseau, ainsi que des composants de haut niveau tels que les entrées DNS entre autres.

Les serveurs de notre architecture seront créés par terraform. J’utilise mon abonnement Cloud AWS

Pour ce projet, je reste dans l’offre gratuite dans la plupart de temps pour réduire le coût de ma facturation.

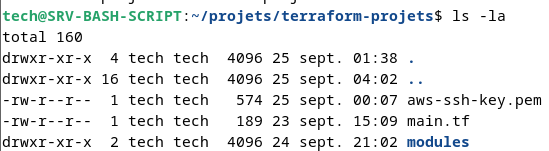
En tout, 6 serveurs (virtuelle machine) Ubuntu seront créés sur le cloud AWS avec les caractéristiques suivantes :

* Type de la machine virtuelle : instance EC2 (Elastic Compute Cloud), AMI (Amazon Machine Image Ubuntu) de type t2.micro
* VM Ubuntu de 30Go de stockage, 1Go de RAM et 1CPU
* SSH installé pour garantir la sécurité d’accès au serveur avec une clé privée
* Les outils docker, prometheuse node exporter, ansible, terraform sont installés
* L’instance de la machine virtuelle porte le nom du service qui y sera déployé
* Un sous réseau (subnet) est choisi parmi les disponibles et une adresse IP publique est attribuée automatiquement par AWS à chaque serveur après la création

Configuration de terraform

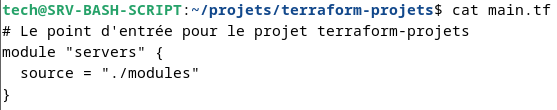
Pour cette partie, j’ai créé le projet ’**’terraform-projets**’’

J’ai choisi d’utiliser une configuration sécurisée et facile en se basant sur l’utilisation des modules constitués d’un fichier main.tf, d’une clé SSH **aws-ssh-key.pem** que j’ai créée localement, et des modules.

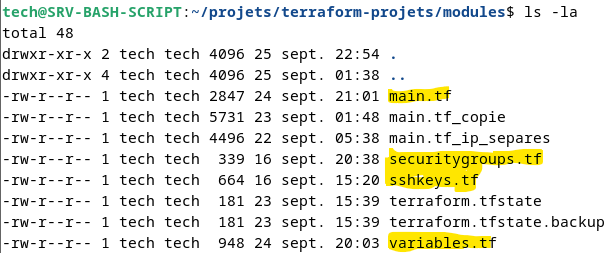
****

La clé **aws-ssh-key.pem** ne sera pas affichée pour raison de sécurité.

main.tf à la racine du projet

****

Le dossier modules, contient le fichier **main.tf** qui contient les commandes principales de création des VM, le fichier **securitygroups.tf** déclare le groupe sécurité à utiliser, **sshkeys.tf** indique la clé SSH à utiliser par les machines VM, le fichier **variables.tf** contient les variables du projet.



le fichier ’’**variables.tf**’’ regroupe toutes les variables utilisées pour ce projet y compris les crédentials AWS (access\_key et secret\_key).

**Le fichier variables.tf**

Dans ce fichier y trouve les variables utilisées par la suite dans le fichier main.tf

# Ficher des variables

# Récupérer les credentials sur l’interface AWS

# AWS access\_key

variable "instance\_access\_key" {

description = "AWS Access\_key"

default = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

}

# AWS secret\_key

variable "instance\_secret\_key" {

description = "AWS Secret\_key"

default = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

}

# subnet (sous réseau disponible)

variable "subnet" {

description = "subnet, le sous réseau"

default = "subnet-041a9f9bfa92522a4"

}

# AWS ami (choisir Amazon Machine Image)

variable "instance\_ami" {

description = "AWS ami"

default = "ami-0e86e20dae9224db8"

}

# Tye d'instance

variable "instance\_type\_micro" {

description = "Instance type"

default = "t2.micro"

}

# Les noms des instances à créer

variable "instance\_vm\_names" {

type = list(string)

#default = ["Prestashop", "Jenkins", "Sonarqube", "Prometheus\_Grafana\_Alertmanager", "Portainer", "Openproject", "Kubernetes"]

default = ["Prestashop"]

}

main.tf de création des instances

provider "aws" {

region = "us-east-1"

access\_key = **var**.instance\_access\_key

secret\_key = **var**.instance\_secret\_key

}

# Création des VMs automatiquement

resource "aws\_instance" "ubuntu-instances" {

**count** = length(**var.instance\_vm\_names**) # Créer autant d'instances qu'il y a dans la liste

ami = var.instance\_ami

instance\_type = var.instance\_type\_micro

key\_name = "aws-ssh-key" # dans sshkeys.tf

vpc\_security\_group\_ids = [ aws\_security\_group.websg.id ]

subnet\_id = var.subnet # le sous réseau

associate\_public\_ip\_address = true # Associe une IP publique

user\_data = "${file("**script-install.sh**")}" # ce script sera exécuté sur chaque instances créé

# Spécification du stockage

root\_block\_device {

volume\_size = 30 # Taille du disque en GiB

volume\_type = "gp2" # Type de volume (gp2 est pour le stockage SSD général)

}

tags = {

Name = **var.instance\_vm\_names[count.index]** # Utiliser l'index pour récupérer le nom de l'instance

}

}

# Groupe sécurité et routage (règles entrantes / sortantes)

resource "aws\_security\_group" "websg" {

name = "web-sg01" # nom du groupe sécurité à créer

# règles entrantes, ici c’est un test ports 3000 au 9999, sinon précisé uniquement les ports utilisés et filtrer l’IP de l’appelant

ingress {

description = "port-8080"

protocol = "tcp"

from\_port = 3000

to\_port = 9999

cidr\_blocks = [ "0.0.0.0/0" ]

}

# règles entrantes, port 22

ingress {

description = "port-ssh"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

# règles sortantes

egress {

description = "port-autre"

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# affichage des @IP des instances créées

output "instance\_public\_ip" {

value = aws\_instance.ubuntu-instances.\*.public\_ip

}

**Créer des VMs avec Terraform**

Azure

[Premiers pas avec Terraform: créer une machine virtuelle Azure. (linkedin.com)](https://fr.linkedin.com/pulse/premiers-pas-avec-terraform-cr%C3%A9er-une-machine-azure-barhoumi-haythem)

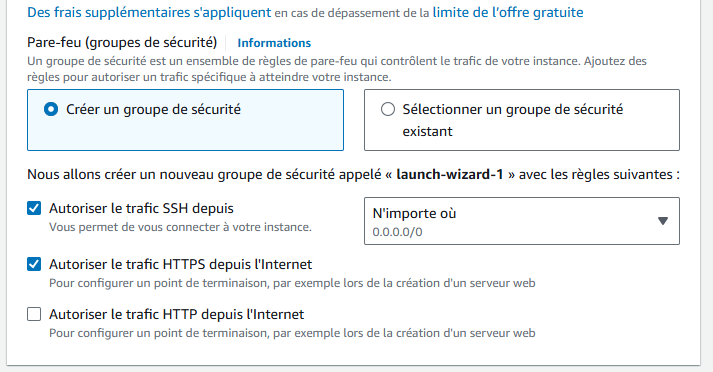
<https://fr.linkedin.com/pulse/premiers-pas-avec-terraform-cr%C3%A9er-une-machine-azure-barhoumi-haythem>

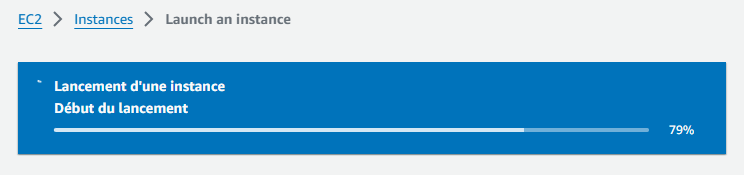
AWS

[AWS Virtual Machine | Terraform AWS Deployment | Terraform Create VM (k21academy.com)](https://k21academy.com/terraform-iac/terraform-automate-aws-vm/)

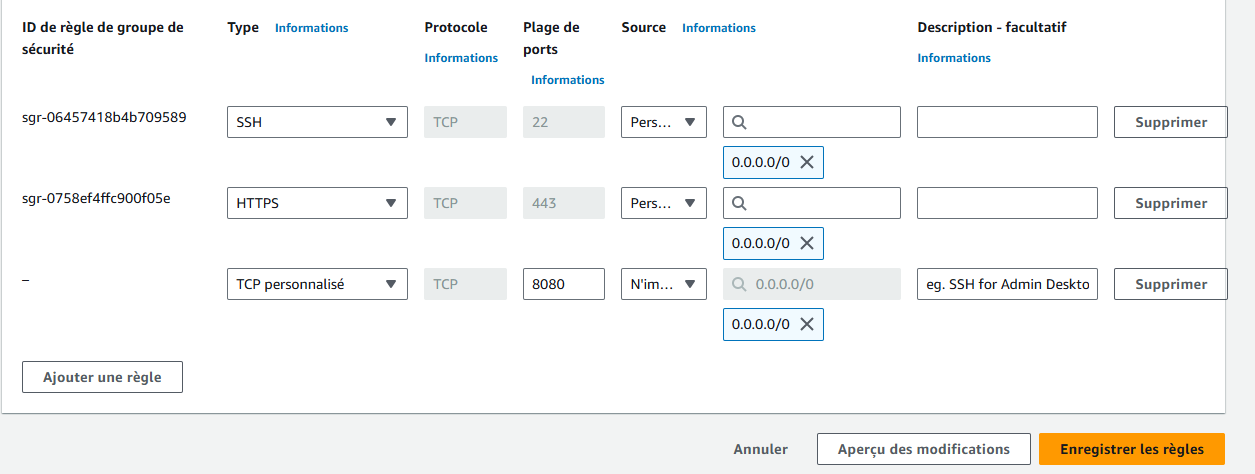
Création de l’instance à partir de la console

Le groupe de sécurité, penser à ajouter une règle

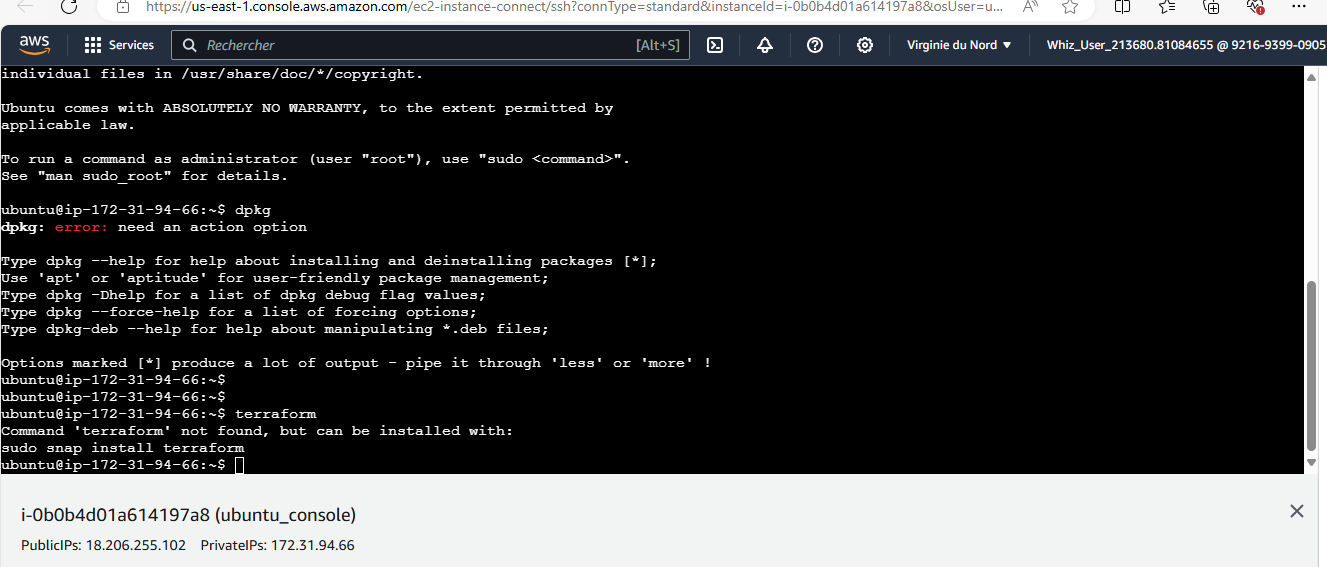




La règle à ajouter après la création



**Revenir aux instances, cliquer sur se connecter**

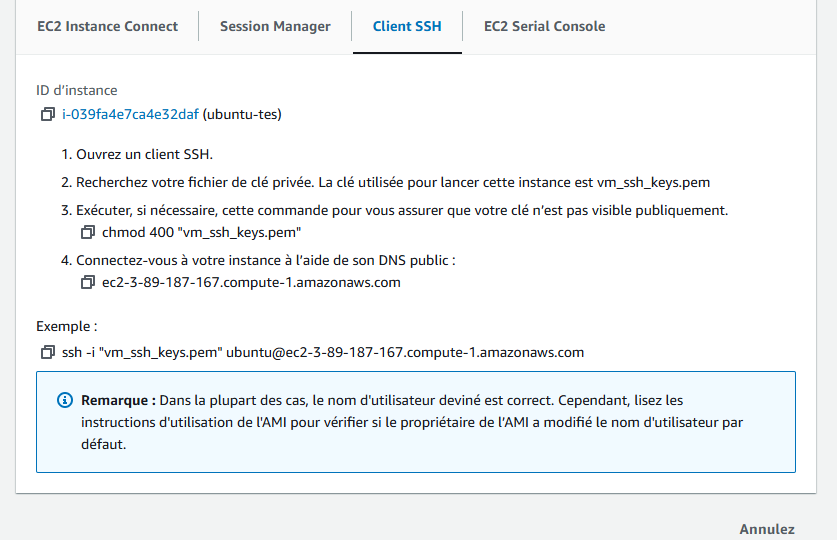


Activer l’accès SSH, créer la clé .pem

Sur la VM AWS, aller sur Se connecter > Client SSH>

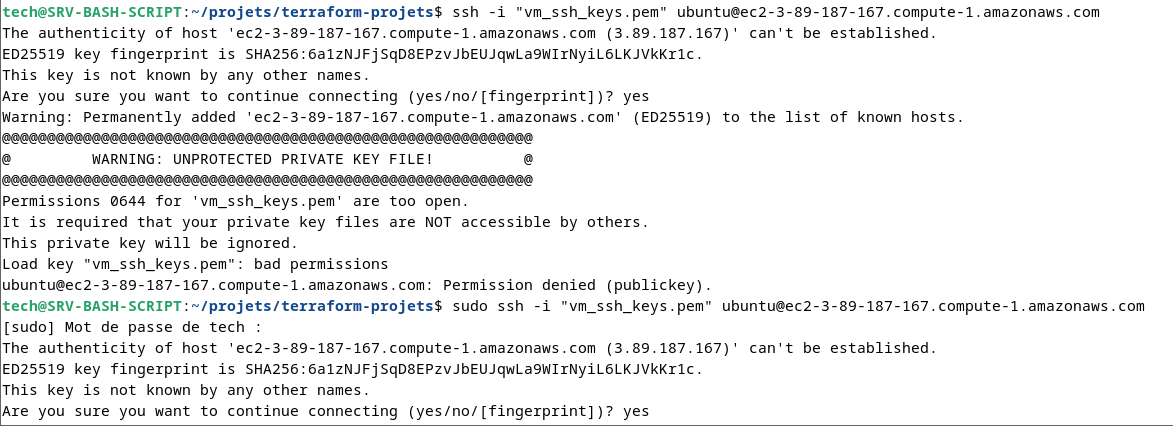
Récupérer cette commande en bas de la page qui s’affiche :

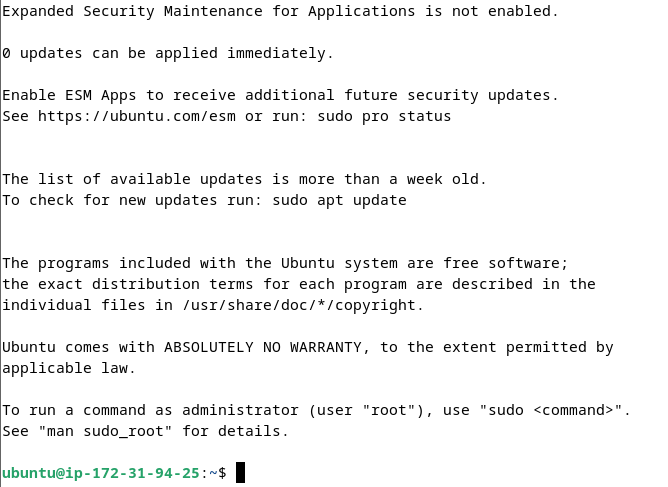
sudo ssh -i "vm\_ssh\_keys.pem" [ubuntu@ec2-3-89-187-167.compute-1.amazonaws.com](mailto:ubuntu@ec2-3-89-187-167.compute-1.amazonaws.com)



A partir de la machine locale, accéder à la VM AWS en exécutant la commande ci-dessus

Cette commande utilise la clé .pem et l’@DNS de la VM,





La connection est établie vers la VM AWS

**Installer terraform si ce n’est pas le cas :**

[**https://medium.com/@sriniv.v29/terraform-introduction-and-installation-on-linux-ubuntu-debian-31486f51fb04**](https://medium.com/@sriniv.v29/terraform-introduction-and-installation-on-linux-ubuntu-debian-31486f51fb04)

**Le projet terraform-projets**

Ce projet est accessible sur github :

Ce projet a pour but de lancer une machine virtuelle sur le cloud AWS avec les caractéristiques suivantes :

System Ubuntu Server 24.04 LTS (HVM), EBS General Purpose (SSD) Volume Type

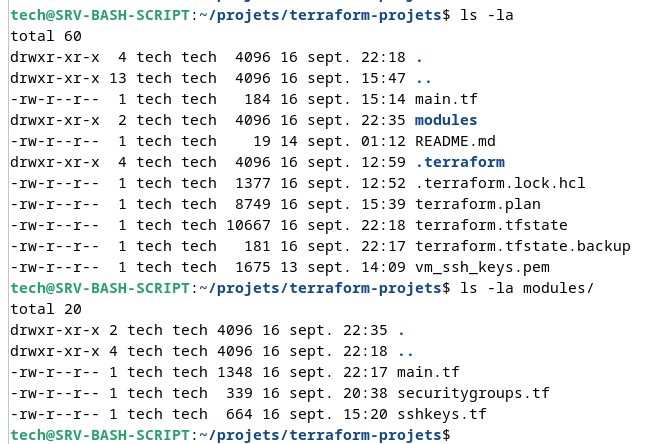
Type t2.micro

Taille de stockage 30 Go

Mémoire 1 Go

CPU 1 Go

Voici les contenus de ce projet

****

**Le fichier main.tf**

provider "aws" {

region = "us-east-1"

access\_key = "AKIAUTJ5VQLMJGXGXJHB"

secret\_key = "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"

}

resource "aws\_instance" "ubuntu-ssh" {

ami = "ami-0885b1f6bd170450c"

instance\_type = "t2.micro"

key\_name = "aws-ssh-key"

vpc\_security\_group\_ids = [ aws\_security\_group.websg.id ]

subnet\_id = "subnet-0cc72ad8e2b985d5b" #le sous réseau

associate\_public\_ip\_address = true # Associe une IP publique

user\_data = <<-EOF

#!/bin/bash

echo "I LOVE TERRAFORM" > index.html

nohup busybox httpd -f -p 8080 &

EOF

tags = {

Name = "ubuntu-server-install"

}

}

resource "aws\_security\_group" "websg" {

name = "web-sg01" # nom du groupe de sécurité, ingress ne peut pas avoir un name juste description

ingress {

description = "port-8080"

protocol = "tcp"

from\_port = 8080

to\_port = 8080

cidr\_blocks = [ "0.0.0.0/0" ]

}

ingress {

description = "port-ssh"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

description = "port-sortant"

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

output "instance\_ips" {

value = aws\_instance.ubuntu-ssh.public\_ip

}

A la racine du projet terrafom-projets où se trouve le fichier main.tf, lancer les commandes suivantes

Initialiser le projet terraform

terraform init

Affcher les détails (plan) de la création dans le fichier main.tf

terraform plan

Deployer le fichier main.tf et afficher l’@IP de l’instance créé

Pour obtenir les logs de la création des VM, affecter la valeur TRACE à la variable d’environnement TF\_LOG comme suit avant d’exécuter apply

export TF\_LOG=TRACE

terraform apply -auto-approve

Si besoin de supprimer la VM, lancer cette commande

terraform destroy

**Se connecter à l’instance créée**

ssh -i "your-key.pem" ubuntu@<instance\_public\_ip>

**Exemple**:

sudo ssh -i "ssh\_test.pem" [ubuntu@ec2-44-204-16-67.compute-1.amazonaws.com](mailto:ubuntu@ec2-44-204-16-67.compute-1.amazonaws.com)

**Ouvrez un client SSH.**

Recherchez votre fichier de clé privée. La clé utilisée pour lancer cette instance est ssh\_test.pem

Exécuter, si nécessaire, cette commande pour vous assurer que votre clé n’est pas visible publiquement. Plus des droits pour groupe et autre peut empêcher l’exécution d’une commande.

chmod 400 "ssh\_test.pem"

**Connectez-vous à votre instance à l’aide de son DNS public :**

ec2-44-204-16-67.compute-1.amazonaws.com

**Renommer la VM**

sudo hostnamectl set-hostname **Jenkins-Sonarqube**

**Exécuter des playbooks sur un VM distante**

**Récupérer l’@IP de la VM**

**Assurer que la clé publique est bien présente dans le répertoire d’exécution d’inventory.yml**

ssh\_test.pem

**Configurer l’inventory.yml**

Jenkins-Sonarqube ansible\_host=ubuntu@ec2-44-204-16-67.compute-1.amazonaws.com ansible\_user=ubuntu ansible\_ssh\_private\_key\_file=./ssh\_test.pem

**Vérifier les logs d’exécution d’un script dans la machine**

sudo cat /var/log/cloud-init-output.log

**Une fois sur la VM, cette commande affiche les commandes exécutées par le script sh**

sudo /var/lib/cloud/instance/scripts/part-001

**Tester en local**

<https://developer.hashicorp.com/terraform/tutorials/provision/cloud-init>

git clone -b cloudinit <https://github.com/hashicorp/learn-terraform-provisioning>

cd learn-terraform-provisioning

ssh-keygen -t rsa -C "your\_email@example.com" -f ./tf-cloud-init

**Add your public SSH key to your cloud-init script**

Copier le contenu du fichier **tf-cloud-init.pub** dans la presse papier.

Ouvriz le fichier **scripts/add-ssh-web-app.yaml** et collez le contenu copié dans la section des données utilisateur. Vous allez passer ce script **cloud-init** à l’attribut de votre ressource d’instance.

Repérer les zones en jaunes et coller la clé sous la zone ssh\_authorized\_keys

users:

- default

- name: terraform

gecos: terraform

shell: /bin/bash

primary\_group: hashicorp

sudo: ALL=(ALL) NOPASSWD:ALL

groups: users, admin

lock\_passwd: false

ssh\_authorized\_keys:

# - ssh-rsa AAAAHHHHHH

- ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB

**Passez en revue le script cloud-init dans la configuration Terraform**

Ouvrez le fichier main.tf situé dans le répertoire instance de ce projet. Notez comment la fonction charge le contenu du fichier **scripts/add-ssh-web-app.yaml**. Ensuite, la configuration transmet le contenu du fichier en tant que valeur à initialiser lorsque l’instance est créée.

**main.tf**

resource "aws\_instance" "web" {

ami = data.aws\_ami.ubuntu.id

instance\_type = "t2.micro"

subnet\_id = aws\_subnet.subnet\_public.id

vpc\_security\_group\_ids = [aws\_security\_group.sg\_22\_80.id]

associate\_public\_ip\_address = true

user\_data = file("../scripts/add-ssh-web-app.yaml")

tags = {

Name = "Learn-CloudInit"

}

}

Ensuite, allez dans le répertoire contenant votre configuration.

$ cd instances

Initialisez votre configuration.

$ terraform init

Appliquez votre configuration. Entrez lorsque vous êtes invité à créer votre instance.yes

$ terraform apply

Installer cloud-init sur votre machine locale (Linux) : Si vous utilisez une distribution basée sur Debian/Ubuntu, vous pouvez installer cloud-init :

sudo apt-get install cloud-init

Créer un fichier user\_data : Sauvegardez votre script user\_data dans un fichier texte.

Exemple :

cat <<EOF > test-user-data.sh

#cloud-config

runcmd:

- echo "Le nom de la VM = VM-1" >> /tmp/vm-name.txt

EOF

Tester avec cloud-init : Vous pouvez simuler l'exécution du script en utilisant cloud-init pour vérifier si votre script fonctionne correctement.

sudo cloud-init init --local

sudo cloud-init --file test-user-data.sh

**Installer terraform**

Préparer l’installation de 2 VM Ubuntu

Installer SSH sur les 2 machines et configurer l’accès à ces VM avec l’extérieur

**GitHub en local**

**Sur la VM1**

Installer Docker

Installer ansible

Préparer les playbook et les inventory pour installer les outils sur la VM2

Les scripts de backup et restauration des conteneurs

**Sur la VM2**

Installer Docker

Installer portainer

Installer prestashop + base MySQL

Installer Prometheus et Grafara (monitoring)

Installer Zabbix et zabbix agent

Installer OpenProject

Installer Jenkins

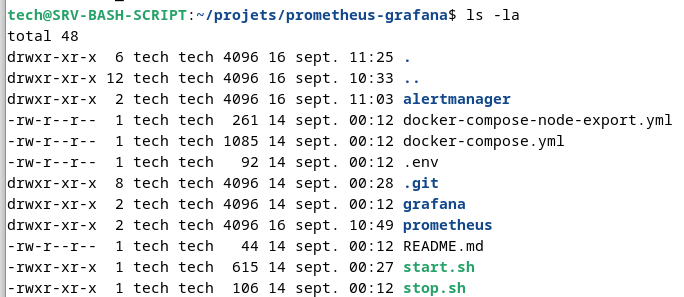
Installer Postgress pour Sonarqube

Installer Sonarqube

**Installer le projet prometheus-grafana**

Le projet sur le github : [git@github.com:lauboudou/prometheus-grafana.git](mailto:git@github.com:lauboudou/prometheus-grafana.git)

**La structure du projet sur le serveur**

****

**Configurer prometheus**

Dans le dossier prometheus se trouve le prometheus.yml et le fichier alert.yml

prometheus/prometheus.yml

global:

scrape\_interval: 15s

evaluation\_interval: 15s

scrape\_timeout: 10s

alerting:

alertmanagers:

- static\_configs:

- targets:

- alertmanager:9093

rule\_files:

- "alert.yml"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: "prometheus"

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ["localhost:9090"]prometheus/prometheus.yml

prometheus/alert.yml

groups:

- name: GC cyles more than 10

rules:

- alert: GC cycle

expr: go\_gc\_cycles\_automatic\_gc\_cycles\_total > 10

for: 1m

labels:

severity: Critical

annotations:

summary: "Your summary"

description: "Your description"

Ce fichier est à modifier à chaque fois qu’on veut ajouter une nouvelle alerte.

Pour éviter l’arrêt/relance de prometheus exécuter cette commande pour charger le fichier alert.yml

curl -X POST http://localhost:9090/-/reload

Dans le dossier alertmanager se trouve le fichier alertmanager.yml

Configurer ce fichier avec les crédentials d’un mail qui va recevoir l’alerte

global:

smtp\_smarthost: 'sandbox.smtp.mailtrap.io:587'

smtp\_from: 'alertmanager@example.com'

smtp\_auth\_username: 'your\_username'

smtp\_auth\_password: 'your\_password'

smtp\_require\_tls: true

route:

group\_by: ['alertname']

receiver: 'email'

receivers:

- name: 'email'

email\_configs:

- to: 'it-team@example.com'

send\_resolved: true

Lancer le projet prometheus-grafana : ./start.sh

**Si un de ces fichiers est modifié, penser à copier le fichier modifié et relancer le conteneur concerné**

Copier le fichier prometheus.yml dans le conteneur prometheus

docker cp prometheus/alert.yml prometheus:/etc/prometheus/

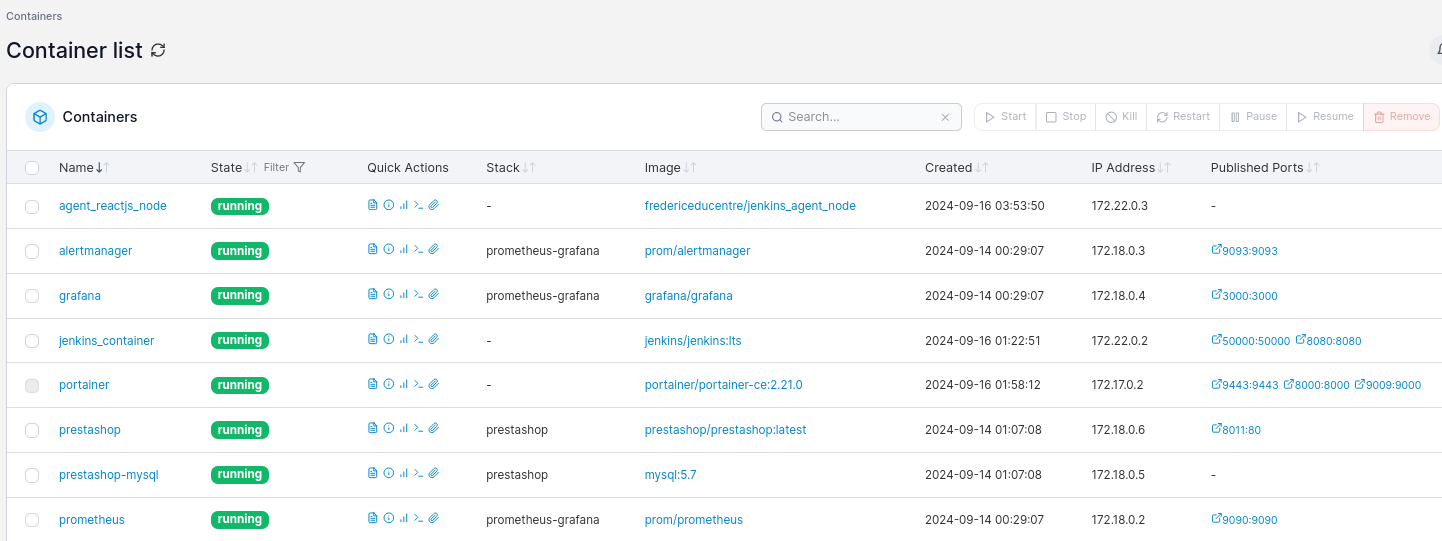
Copier le fichier alert.yml dans le conteneur prometheus

docker cp alertmanager/alertmanager.yml /etc/alertmanager/

Copier le fichier alertmanager.yml dans le conteneur alertmanager

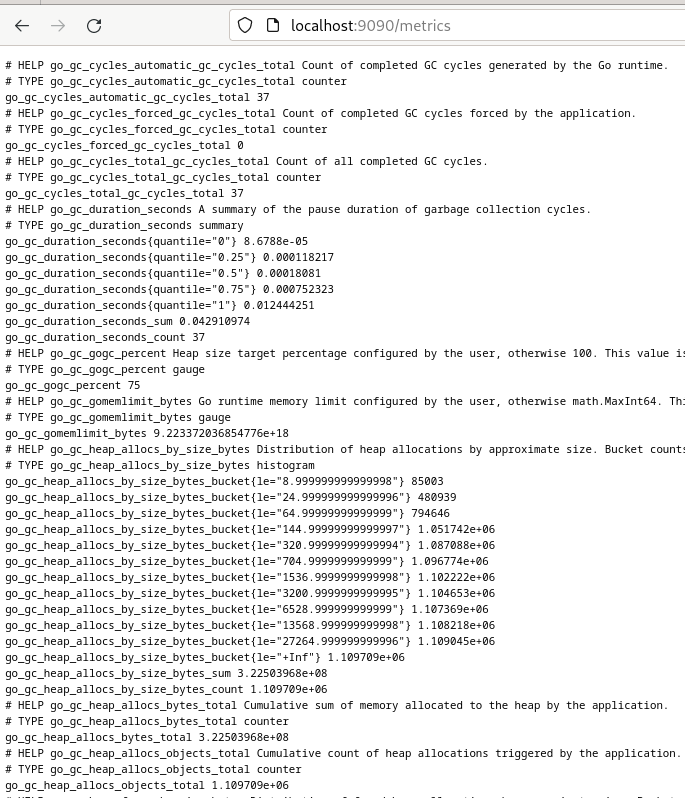
docker cp alertmanager/alertmanager.yml alertmanager:/etc/alertmanager/

Vérifier sur l’interface **portainer** que les conteneurs grafana, prometheus et alertmanager sont démarrés correctement



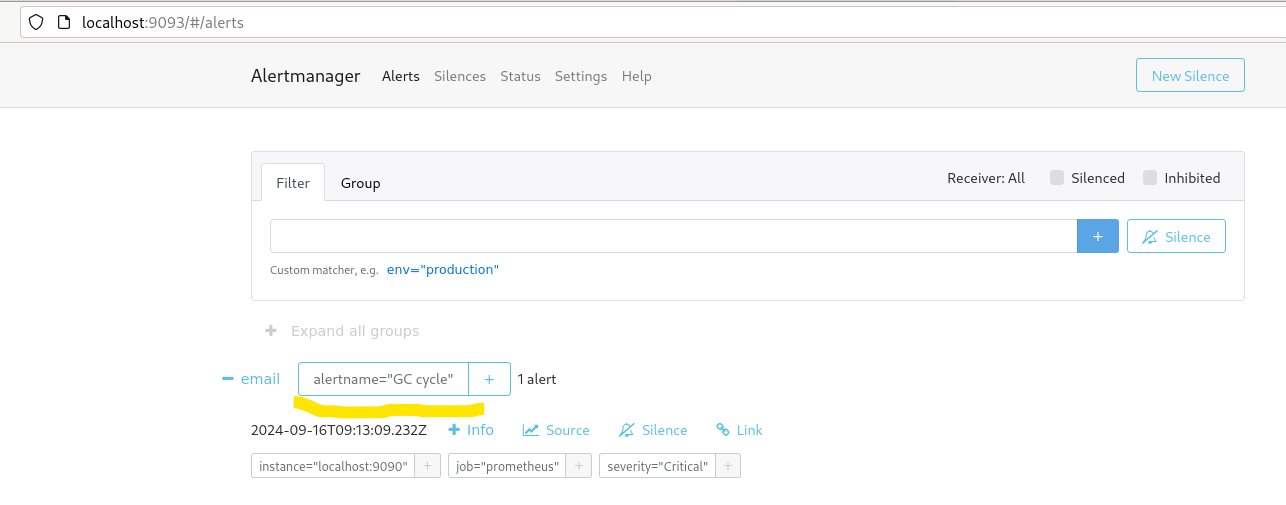
L’url suivante affiche les métriques collectées par **prometheus** sur lui même

http://localhost:9090/metrics

****

L’url suivante affiche les alertes remontées par le service alertmanager, nous y retrouvons nos alertes configurées. Exemple : le nom souligné en jaune.

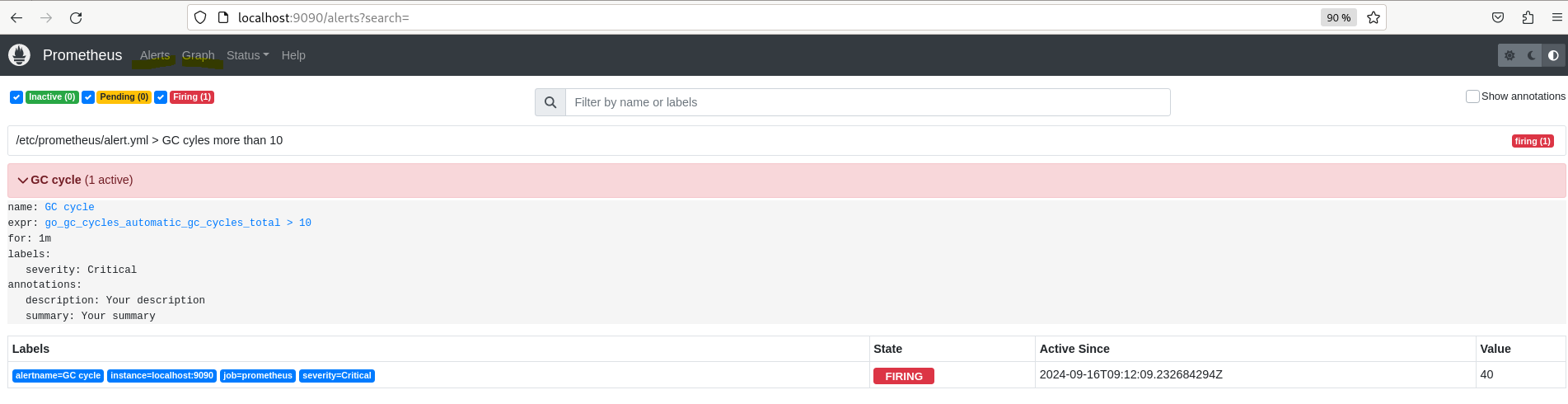
[**http://localhost:9093/#/alerts**](http://localhost:9093/#/alerts)

****

Vérifier les alertes envoyées par prometheus s’il y’en a via ce lien et sur l’onglet Alertes.

L’alerte est envoyée à l’adresse mail configuré si la condition dans prometheus/alert.yml est remplie

http://localhost:9090/alerts?search=

****

**Pour configurer plusieurs alertes, aller dans le fichier prometheus/alert.yml et ajouter l’alerte comme suit :**

prometheus/alert.yml

groups:

- name: GC cyles more than 10

rules:

- alert: GC cycle

expr: go\_gc\_cycles\_automatic\_gc\_cycles\_total > 10

for: 1m

labels:

severity: Critical

annotations:

summary: "Your summary"

description: "Your description"

- name: Copie GC cyles more than 10

rules:

- alert: Copie GC cycle

expr: go\_gc\_cycles\_automatic\_gc\_cycles\_total > 10

for: 1m

labels:

severity: Critical

annotations:

summary: "Your summary"

description: "Your description"

recharger le fichier dans le conteneur et redémarrer le.

**Installer le node exporter sur le serveur pour collecter des métriques et pouvoir les afficher dans grafana**

Monitorer l’application prestashop par Grafana

Configurer le monitoring de l’application prestashop et sa base MySQL

Configurer le monitoring de service & matériels :

**Sur sonarqube**

Préparer un projet de tests avec prestashop ?

Test sur VM-DevOps

sonar-scanner \

-Dsonar.projectKey=Test-horoscope-zodiac-js \

-Dsonar.sources=. \

-Dsonar.host.url=http://172.22.0.5:9000 \

-Dsonar.token=sqp\_4302017c95f9f0810103a9d26090f23be16e311e

**Sur jenkins**

Préparer un projet de clone, scan et delivery du projet prestashop, automatisation de pipeline si github est mis à jour ?

CI/ git clone

CD/ Déploiement dans ce cas est l’équivalent de docker build de l’image de notre conteneur

Préparer un Dockerfile avec les dépendances (php composer, …)

CD/ délivery dans un environnement (prod, test, ….)

Développer une page landing ave une invitation au site calanly : application de prendre de visioconférence

Déployer l’index.html sur nginx

Créer un projet node js

**Installer nodes**

curl -fsSL https://deb.nodesource.com/setup\_14.x | sudo -E bash -

sudo apt-get install -y nodejs

Vérifier l’installation

node -v

npm -v

npx -v

si npm pas installé

sudo apt install -y npm

Créer le projet

npx create-react-app prestashop-landing-page

cd prestashop-landing-page

cd src

**Création d’un fichier Dockerfile avec ce contenu**

FROM nginx:latest

RUN apt update -y && apt upgrade -y

COPY index.html /usr/share/nginx/html

Lancer le script **build.sh** pour la création de l’image

docker build . -t prestashop-lading-page:latest

Lancer le script **start.sh** pour la création du conteneur prestashop-lading-page

Utiliser Dependencies check ?

Trivicy ?

Automatisation de pipeline avec github (obligatoirement projet avec tag) ?

Credential repo privé ZodiacJS.git

github\_pat\_11BHX2UZA0C3dFWPepPxcb\_FW9zPJZrSpVuCJclkucFBpdzPbTkjTl0va3r4y6S08iOEW2HCFToks8orzb

Credential dockerhub

dckr\_pat\_sns0tmEeD1wqLNeGxwfVb5ee7mw

Pod https://kubernetes.io/fr/docs/concepts/workloads/pods/pod/#:~:text=GestionLes%20pods%20fournissent%20une%20unit%C3%A9%20des kubernetes

**Créer un repo git à partir de la ligne de commandes**

**echo "# prometheus-grafana2" >> README.md**

**git init**

**git add README.md**

**git commit -m "first commit"**

**git branch -M main**

**git remote add origin https://github.com/lauboudou/prometheus-grafana2.git**

**git push -u origin main**

**Installer Docker sur Ubuntu**

# Update package list

sudo apt-get update

# Install prerequisites

sudo apt-get install -y ca-certificates curl gnupg lsb-release

# Add Docker’s official GPG key

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Set up the stable repository

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update package list again

sudo apt-get update

# Install Docker Engine

sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# Verify installation

sudo docker run hello-world

# Optional: Manage Docker as a non-root user

sudo usermod -aG docker $USER

newgrp docker

Installer node\_exporter sur ubuntu

Suivre le tuto : <https://doc.ataxya.net/books/grafana/page/installer-node-exporter-et-ajouter-%C3%A0-prometheus>

Créer une clé SSH en local sur la VM pour se connecter sur github

ssh-keygen

dans /home/tech/.ssh/

clés : id\_rsa et id\_ras.pub

Copier la clé privée dans Github> Settings > SSH & keys

Ajouter la clé

**Créer un network dans le docker**

docker network create diamon-network

Installer prestashop

Installer portainer

Installer prometheus-grafana

Installer node\_exporter

Sécurisation de l’accès prometeus à node\_exporter

Si htpasswd n’est pas installé, exécuter la commande : **sudo apt install apache2-utils**