

# PROJET FINAL DEVOPS

GROUPE 4 CAPGEMINI

# Table des matières

<b>Introduction</b>	<b>1</b>
Présentation Devops	1
Présentation Projet	1
Les logiciels utilisés	2
<b>Infrastructure</b>	<b>5</b>
<b>Conteneurisation de l'application web</b>	<b>5</b>
Docker	5
Kubernetes	10
<b>Mise en place d'un pipeline CI/CD</b>	<b>20</b>
Jenkins	20
Ansible	25
Création des Rôles Ansible	26
Rôle Docker	26
tasks/main.yml	26
Rôle Odoo	27
defaults/main.yml	28
tasks/main.yml	28
templates/docker-compose.yml.j2	28
Rôle PgAdmin	29
defaults/main.yml	30
tasks/main.yml	30
templates/docker-compose.yml.j2	30
templates/servers.json.j2	30
Rôle ic-webapp	31
defaults/main.yml	31
tasks/main.yml	31
templates/docker-compose.yml.j2	32

Troubleshooting	32
Les Playbooks Ansible	32
roles/requirements.yml	33
playbook_odoo.yml	33
playbook_pgadmin.yml	33
playbook_ic-webapp.yml	33
hosts_.yml	34
Commandes ansible à exécuter (sur une machine disposant ansible)	35
Console	35
Site odoo	36
Console	37
Site PgAdmin	37
Console	38
Site ic-webapp	39
Terraform	40
Déploiement de l'environnement dev avec TERRAFORM	40
Réflexion	40
Les modules	40
module "sg" security group	40
module "ec2_worker"	42
module "ec2_master"	42
Les credentials	44
Le remote backend S3	45
Le provider	45
Le manifest principal (main)	45
Les variables à surcharger	46
Exemple	47
<b>Test</b>	47
<b>Conclusion</b>	<b>48</b>

# 1 Introduction

## 1.1Présentation Devops

Le fonctionnement devops permet aux équipes de développement et d'opérations de ne plus être isolées. Les équipes devops travaillent alors sur tout le cycle de vie d'une application, depuis sa création jusqu'à l'exploitation.

Les membres des équipes devops peuvent donc mieux communiquer, et la productivité et l'agilité sont meilleures. De manière générale, adoptant la méthodologie devops nous allons gagner en productivité.

## 1.2 Les logiciels utilisés

### AWS

Nous avons choisi Amazon Web Services pour la partie Cloud de notre infrastructure. Cette plateforme permet de nombreux services et c'est de loin la plateforme cloud la plus complète et répandue dans le monde.

La partie Amazon Elastic Compute Cloud nous à servi pour la création de nos Instances EC2. L'intégralité de notre infrastructure est donc basée sur AWS.

### Docker

Nous avons utilisé docker pour la partie Conteneurisation de l'application web. En effet pour le déploiement de notre application des conteneurs docker ont été utilisés.

Les conteneurs vont nous permettre d'exécuter nos environnement de manière simple et légère tout en gardant la performance. Par la suite la création d'un fichier Dockerfile permet d'exécuter les instructions de création des images docker.

```
ubuntu@minikube-master:~$ docker ps -a
CONTAINER ID   IMAGE                                PORTS          COMMAND
CREATED        STATUS          NAMES
6eb3dba55852   dpape/pgadmin4                        "/entrypoint.sh"
15 hours ago   Up 15 hours
k8s_pgadmin_pgadmin-deploy-598c54c4-
bvqsd_icgroup_13254cdb-f5df-4da9-9e11-104e46f5828c_0
58f4547781a6   lianhuahayu/ic-webapp                "python app.py"
15 hours ago   Up 15 hours
k8s_ic-webapp_deploy-ic-webapp-c4fb7
4dc-jmpsh_icgroup_75b3f53b-4992-4c4c-91b1-9a9dc151f6f3_0
89bab054c30d   lianhuahayu/ic-webapp                "python app.py"
15 hours ago   Up 15 hours
k8s_ic-webapp_deploy-ic-webapp-c4fb7
4dc-sqs4k_icgroup_6d563e47-1508-4cd3-841a-09b2875a4eca_0
d8fc76bff91a   k8s.gcr.io/pause:3.5                 "/pause"
15 hours ago   Up 15 hours
k8s_POD_deploy-ic-webapp-c4fb74dc-jm
psh_icgroup_75b3f53b-4992-4c4c-91b1-9a9dc151f6f3_0
bae34431e355   k8s.gcr.io/pause:3.5                 "/pause"
15 hours ago   Up 15 hours
k8s_POD_pgadmin-deploy-598c54c4-bvqs
d_icgroup_13254cdb-f5df-4da9-9e11-104e46f5828c_0
a58c56096a14   6ffaab8969de                         "/entrypoint.sh o
doo" 15 hours ago   Up 15 hours
k8s_odoo_odoo-deploy-57f6d77c9-5j5ss
_icgroup_ed113e13-7718-472e-9b77-f90b901685b2_0
04959a7f4277   0896a8e0282d                         "docker-entrypoin
t.s..." 15 hours ago   Up 15 hours
k8s_postgres-db-pod_icgroup_4203d736
-ce12-4f2a-b4ea-947d91934dd0_0
1ed857b3ea9b   k8s.gcr.io/pause:3.5                 "/pause"
15 hours ago   Up 15 hours
k8s_POD_deploy-ic-webapp-c4fb74dc-sq
```

## Kubernetes:

L'outil Kubernetes nous permet de déployer les applications dans un même cluster. Nous avons besoin du déploiement de l'application odoo pour la partie erp, mais aussi de pgadmin pour la partie base de données. Grâce à la création de différents fichiers « les manifests » qui permettent le bon fonctionnement de l'application pour son déploiement.

Voici la liste des services disponibles dans notre cluster Kubernetes

```
ubuntu@minikube-master:~$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
clusterip-ic-webapp	ClusterIP	10.110.223.10	<none>	80/TCP	14h
db-service	ClusterIP	10.103.85.38	<none>	5432/TCP	14h
odoo-service	NodePort	10.110.147.244	<none>	8069:32020/TCP	14h
pgadmin-service	NodePort	10.100.232.92	<none>	80:32125/TCP	14h

```
ubuntu@minikube-master:~$
```

## Ansible:

Ansible est un outil open-source de provisionnement de logiciels, de gestion des configurations et de déploiement d'applications qui permet de faire de l'infrastructure un code. Il fournit un environnement stable pour l'équipe de développement et d'opérations conduisant ainsi à une orchestration fluide. L'automatisation Ansible aide à la représentation de l'infrastructure en tant que code

Afin de faciliter le déploiement de nos applications à l'aide d'un pipeline CI/CD, nous avons créé des rôles ansible.

**Jenkins:** il rassemble les lignes de code produites par une équipe de développeurs, exécute des tests unitaires et fonctionnels, et pousse le code et les configurations testés vers les systèmes cibles. L'objectif de Jenkins n'est pas seulement d'être centré sur l'infrastructure

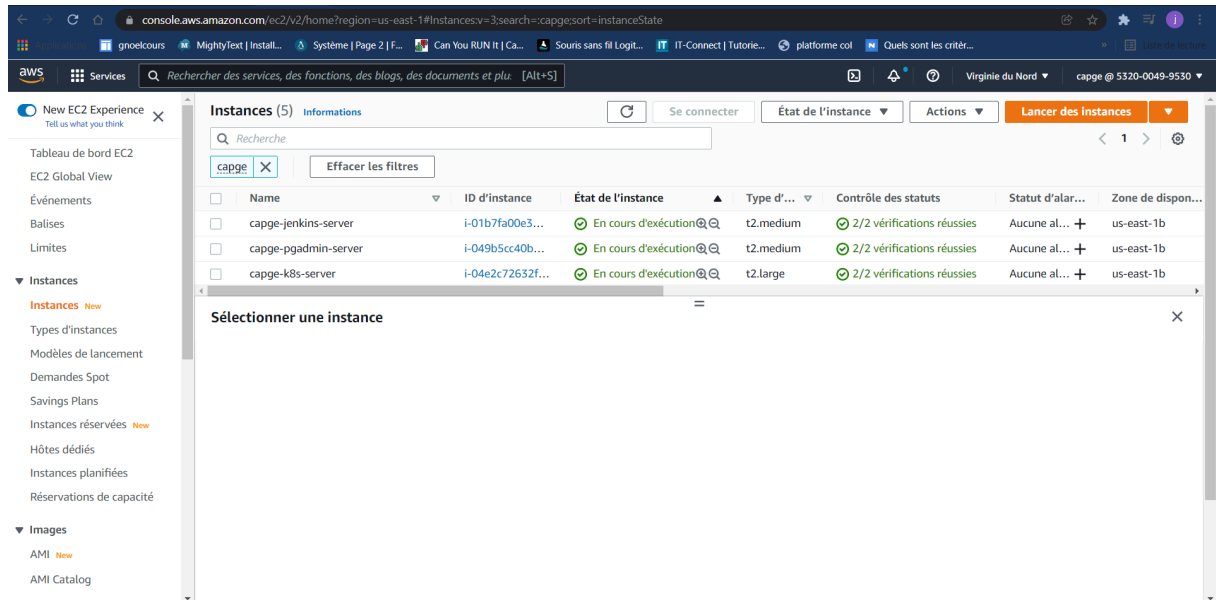
**Terraform:** il a pour finalité d'automatiser le provisioning et le management de n'importe quelle infrastructure cloud ou ressources IT. Terraform permet aux développeurs d'utiliser un langage de configuration appelé HCL (HashiCorp Configuration Language), qui décrit l'infrastructure cloud ou sur site souhaitée pour l'exécution d'une application.



Terraform génère ensuite un plan permettant d'atteindre cet état final et exécute le plan pour mettre à disposition l'infrastructure.

## 2 Infrastructure

Notre infrastructure est basé uniquement sur le cloud AWS :



## 3 Conteneurisation de l'application web

Il s'agit en effet d'une application web python utilisant le module Flask. L'image de base que nous vous recommandons afin de conteneurisé cette application est :

python:3.6-alpine

Une fois le Dockerfile créé, Builder le et lancer un container test permettant d'aller sur les sites web officiels de chacune de ces applications (site web officiels fournis ci-dessus).

**Nom de l'image :** ic-webapp ;

**tag :** 1.0

**container test\_name :** test-ic-webapp\*

Une fois le test terminé, supprimez ce container test, testez les vulnérabilités de votre image à l'aide de l'outil **snyk**. Une fois terminé, poussez votre image sur votre registre Docker hub.

### Répertoire de notre application

Récupération du projet puis se rendre dans le répertoire du projet



```
sudo apt update -y
sudo apt install git -y

git clone https://github.com/sadofrazer/devops-project1.git
cd devops-project1
```

### 3.1 Docker

#### Installation de docker

Installation de docker, si l'on veut build et tester notre image il faudra docker.

```
#Installation de docker via le script fourni par docker
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh

#Ajout du user ubuntu dans le groupe docker
sudo usermod -aG docker ubuntu

#Demarrer docker et configurer son demarrage auto à chaque reboot
sudo systemctl start docker
sudo systemctl enable docker
```

#### Rédaction du Dockerfile

Edition du Dockerfile :

```
FROM python:3.6-alpine
```

```

LABEL maintainer=CAPGEMINI

# Configuration du répertoire de travail
WORKDIR /opt/ic-webapp/

# Copier les fichiers de notre projet dans notre répertoire de travail
COPY . /opt/ic-webapp/

# Set variables env ODOO_URL and PGADMIN_URL (Les variables d'environnement
sont spécifiées dans l'app.py)
ENV ODOO_URL='https://www.odoo.com/'
ENV PGADMIN_URL='https://www.pgadmin.org/'

# Test variables env ODOO_URL and PGADMIN_URL
# Pendant le build de l'image :
# Vérifiez que ces variables ont bien pris les valeurs précédentes
RUN echo ${ODOO_URL}
RUN echo ${PGADMIN_URL}

# Installation Flask avec pip car l'application utilise flask
RUN pip install flask

# Exposition du port 8080 pour l'API (Le port est spécifié dans l'app.py Run
Flask Application)
EXPOSE 8080

# Start le serveur et lancer l'application
ENTRYPOINT [ "python", "app.py" ]

```

## Build de notre image et lancement du container test

Commandes de build et de lancement

```

# Commande pour build notre image
docker build -t ic-webapp:1.0 .

# Commande pour lancer notre container
docker run -d --name test-ic-webapp -p 80:8080 ic-webapp:1.0

```

La commande docker build le -t ou --tag permet de nommer et ajouter un tag à notre image que l'on va build, le . à la fin de notre commande est obligatoire pour dire que l'on s'appuie sur un Dockerfile se trouvant dans le répertoire courant.

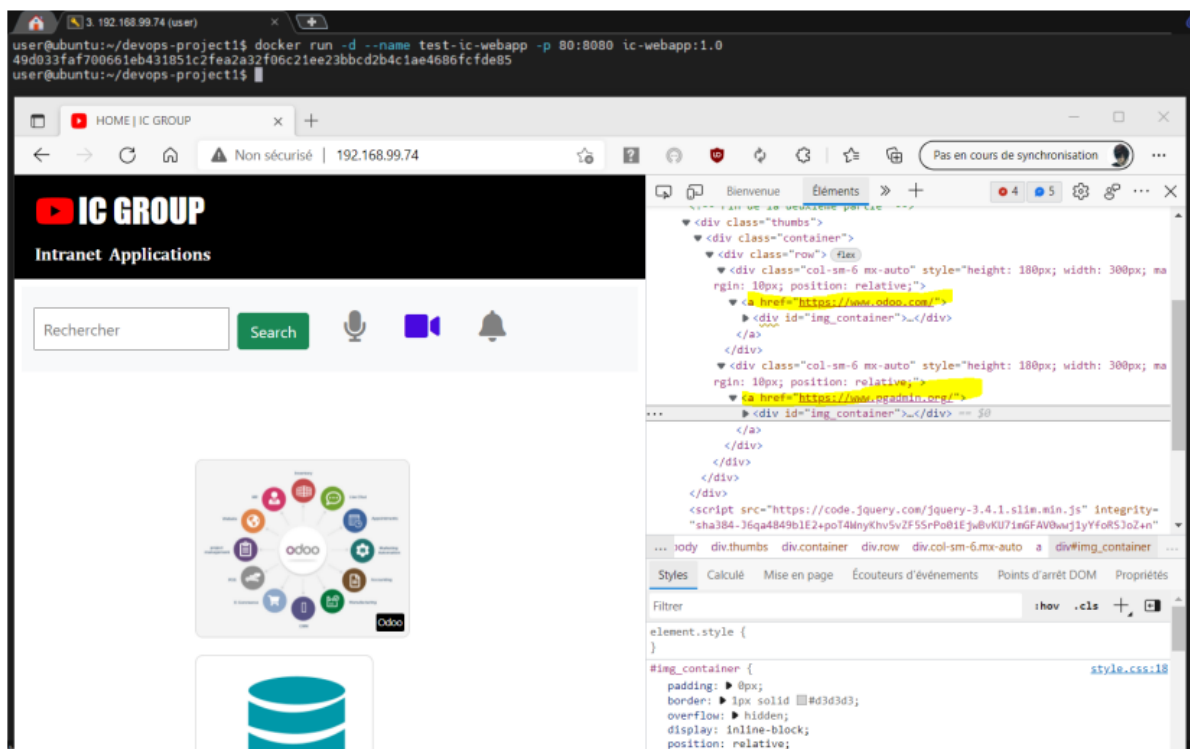
Pour la commande docker run l'option -d ou detach sert à lancer le container en background pour libérer le terminal, le --name sert à donner un nom à notre container, -p

pour exposer un port pour que l'on puisse se connecter à l'API, on fait une redirection du 80 externes (pour de l'hôte) vers le 8080 internes (port du container), étant donné que c'est une application web, c'est mieux d'utiliser le port http tcp/80. (NB : Vérifiez que vous n'avez pas un service web qui tourne sur votre machine et qui soient déjà binder au port 80 de host). Ensuite à la fin de la commande on ajoute le nom et le tag de notre image builder précédemment.

Pour valider que notre image déployée dans le container test-ic-webapp fonctionne bien, on peut tout simplement ouvrir notre navigateur et cliquer sur les différentes applications qui nous redirigeront normalement sur les sites officiels de odoo et de pgadmin.

### Vérification du bon fonctionnement de notre image

Sur la capture ci-dessous on constate bien que ces deux images ont bien pris la référence du site officiel.



On pourra surcharger ces variables d'environnement au lancement du container avec l'option '-e'

## Nettoyage des containers

On peut maintenant après avoir tester notre image dans un container et vérifier que cela fonctionne bien la supprimer comme suit :

```
# Il faut stopper les containers avant de les supprimer
docker container stop test-ic-webapp

# Supprimer notre container qui est à l'arrêt
docker container rm test-ic-webapp

# Une suppression peut s'être mal déroulé et donc quand on voudra rester notre
image on ne pourra pas indiquant que l'image est déjà utilisé dans un
container.
# Je vous suggère de lister tous les containers
docker container ls -aq

# Voir faire un nettoyage forcé de ce qu'il y a via
```

```
docker container stop $(docker container ls -aq)
docker container rm $(docker container ls -aq)
```

Pour nettoyer nos environnements de tests des containers créés plus tard, ce script rapide supprime les containers proprement, puis supprime les volumes et supprime le répertoire où est sauvegarder les configurations persistantes ! Nous avons appelé ce script clean.sh toujours utile.

Pour nettoyer l'environnement de test local :

```
#!/bin/bash
docker container stop $(docker container ls -aq)
docker container rm $(docker container ls -aq)
docker volume prune --force
docker network prune --force
sudo rm -Rf /backup/*
user@worker:~$ sudo sh clean.sh
```

## Test des vulnérabilités de notre images

Ensuite on récupère Snyk pour tester les vulnérabilités de notre image, pour information Docker et Snyk ont un partenariat ce qui fait qu'à l'installation de docker, ce dernier

possède l'outil docker scan qui permet de scanner des containers en s'appuyant sur une api Snyk.

```
docker scan --version

docker scan --login --token <mon token snyk>
docker scan lianhuahayu/ic-webapp:1.0 --file Dockerfile
```

On peut ajouter beaucoup plus de détails de paramètres pour que nos scans soient efficaces. Nous verrons par la suite que dans le pipeline nous pouvons passer par un docker scan ou directement avec une cli Snyk que l'on va récupérer avec nodejs sur notre node Master JENKINS en /bin/bash via la commande d'installation npm install snyk@latest -g

### Push image sur le registry

Il faut d'abord tag notre image avec le nom de notre repository

```
#Listez avec nos images et récupérer l'IMAGE ID de notre image
docker image ls

#Une fois que l'on a notre image ID on peut tag cette dernière avec notre repo
sur dockerhub
docker tag 66682c37eec4 lianhuahayu/ic-webapp:1.0

# Ensuite se connecter à dockerhub avec notre compte, cette étape est
nécessaire pour push une image
```

```
docker login --username lianhuahayu

# Puis push notre image du le registry
docker push lianhuahayu/ic-webapp:1.0

# Une fois que c'est push on peut se logout
docker logout
```

Pour la partie sécurité, on peut pousser notre application sur un registry privé, mais il faudra fournir des accès pour pouvoir push et pull. Alors qu'un registry public ne nécessite pas qu'on soit authentifié pour pull.

Maintenant nous pouvons récupérer notre image pour la déployer où l'on veut :)

```
docker pull lianhuahayu/ic-webapp:1.0
```

## 3.2 Kubernetes

Kubernetes est une plateforme Open Source conçue pour gérer la nature éphémère de milliers de conteneurs. Ces conteneurs démarrent, s'exécutent et s'arrêtent en permanence. Kubernetes gère le versioning des conteneurs, détermine comment les conteneurs peuvent communiquer entre eux sur le réseau, expose les services fonctionnant à l'intérieur des conteneurs et gère les la persistance. Kubernetes permet aussi d'accélérer rapidement le nombre d'instances de conteneurs en fonction des pics de charge.

### Rédaction des manifests:

Nous avons commencé par créer un namespace pour fédérer toutes nos ressources. ensuite nous nous sommes lancé en parallèle sur la création des différentes ressources en respectant le schéma donné dans l'énoncé du projet :

### Base de données:

Pour la création de la base de données, nous avons écrit cinq manifests. Nous allons maintenant vous donner leur ordre de création :

Tout d'abord le pv et le pvc :

- Pour le pv, nous lui donnons une capacité de 1Go, avec un "hostpath" qui a pour valeur "/data-pv", rappelons tout de même que l'objectif de l'attribut "hostpath" est de monter un fichier ou un dossier depuis le système de fichiers du nœud hôte à l'intérieur d'un Pod.
- Pour le pvc, qui sont des demandes pour des ressources contenues dans le pv, nous allons lui fixer une capacité de 100 Mi.
- Pour le pv et le pvc, en "access mode" nous avons opté pour le ReadWriteOnce (le volume peut être monté en lecture-écriture par un seul nœud). Il faut aussi spécifier le

même "storageClassName" pour le pv et le pvc pour que ce dernier puisse se rattacher à lui.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
  namespace: icgroup
  labels:
    app: db_odoo
    env: prod
spec:
  storageClassName: manual
  resources:
    requests:
      storage: 200Mi
  accessModes:
    - ReadWriteOnce
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: db-pv
  namespace: icgroup
  labels:
    app: db_odoo
    env: prod
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data-pv
```

Nous avons aussi un manifest pour le secret, qui a pour attribut "type" la valeur Opaque (le type Opaque est utilisé pour des données arbitraires, en effet nous pouvons spécifier d'autres type pour des token d'authentification kubernetes ou bien docker).

Et le but de ce manifest de contenir le mot de passe de notre base de données (il est transformé en base 64).

```
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: icgroup
  labels:
    app: db_odoo
    env: prod
type: Opaque
data:
  db-passwrd: b2Rvbwo=
```

En respectant le schéma donné en début de projet, nous avons créé un service qui va être relié à notre pod qui contient notre conteneur de base de données.

Le manifest du pod nous crée le conteneur de la base de données à l'aide d'une image "postgres:13". Pour créer notre conteneur, nous devons lui renseigner : le nom de la bdd, un nom d'utilisateur et un mot de passe (ici nous ferons appel à la bonne ressource secrète qui a son nom renseigné dans l'attribut "secretKeyRef").

Nous monterons (ou relieront) aussi le chemin où sont stockées les données à l'intérieur de notre conteneur avec le pvc créé précédemment à l'aide de l'attribut "volumeMounts".

Petite précision avant de passer au service de la bdd, le chemin

"/var/lib/postgresql/data" représente le répertoire à l'intérieur du pod qui va être relié au "hostpath" du pv.

```
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
  namespace: icgroup
  labels:
    app: db_odoo
    env: prod
spec:
  containers:
    - name: postgres
      image: postgres:13
      env:
        - name: POSTGRES_DB
          value: postgres
        - name: POSTGRES_USER
          value: odoo
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: db-passwd
      volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: db-volume
```

Pour le service de la bdd, nous avons opté pour un service de type "clusterIP" pour qu'il ne soit pas consommable de l'extérieur. L'attribut "targetPort" représente le port exposé au niveau du pod et l'attribut "port" représente le celui du service.



Pour sélectionner nos ressources précédemment créées, nous utilisons l'attribut "selector" qui a une valeur qui se retrouvera dans tous les attributs "label" de nos ressources de la bdd (l'attribut "selector" est utilisé dans tous nos services).

```
apiVersion: v1
kind: Service
metadata:
  name: db-service
  namespace: icgroup
  labels:
    app: db_odoo
    env: prod
spec:
  type: ClusterIP
  ports:
    - targetPort: 5432
      port: 5432
  selector:
    app: db_odoo
```

## Odoo:

Pour lancer l'application odoo, nous avons deux manifests: Le premier manifest, créer un déploiement avec une réplica, et le deuxième, un service pour gérer le déploiement.

Pour le premier manifest, Le conteneur est créé à partir d'une image "odoo:13". Les variables d'environnement "PASSWORD" et "USER" ont pour valeur "odoo" et quant à la variable "HOST" elle a comme valeur le nom du service de la bdd ("db-service" dans notre cas de figure), et enfin il faudra exposer le port du conteneur ("8069" pour odoo).

Le deuxième manifest, nous crée un service de type "NodePort", en effet nous avons choisi ce type de service pour qu'il soit consommable depuis l'extérieur. Bien sûr, nous avons mis la valeur '8069' pour l'attribut "targetPort" (pour 'targetter' notre conteneur), et nous avons aussi donné une valeur pour l'attribut "nodePort" pour pouvoir accéder à notre service.

```

apiVersion: v1
kind: Service
metadata:
  name: odoo-service
  namespace: icgroup
  labels:
    app: odoo-server
    env: prod
spec:
  type: NodePort
  ports:
    - port: 8069
      protocol: TCP
      targetPort: 8069
      nodePort: 32020
  selector:
    app: odoo

```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: odoo-deploy
  namespace: icgroup
  labels:
    app: odoo
    env: prod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: odoo
  template:
    metadata:
      labels:
        app: odoo
    spec:
      containers:
        - name: odoo
          image: odoo:13
          env:
            - name: HOST
              value: db-service
            - name: PASSWORD
              value: odoo
            - name: USER
              value: odoo
          ports:
            - name: http
              containerPort: 8069

```

## PgAdmin:

Comme pour les précédents, nous avons créé un fichier secret, contenant le mdp pour se connecter au service. Ici nous l'avons converti en base64.

```

apiVersion: v1
kind: Secret
metadata:
  name: pgadmin-secret
  namespace: icgroup
  labels:
    app: pgadmin
    env: prod
type: Opaque
data:
  pgadmin-password: c6fzc3dvcnQtc6dhZ6lpbg== #password in base 64 (password-pgadmin)

```

Nous avons aussi créé un pvc pour déploiement, le but de ce dernier est de stocker les données de configuration du serveur de pgadmin. Et comme pour celui de la bdd il est de type "ReadWriteOnce".

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pgadmin-pvc
  namespace: icgroup
  labels:
    app: pgadmin
    env: prod
spec:
  #storageClassName: aa
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "3Gi"
```

Passons maintenant aux trois manifests principaux : le **configMap**, le **déploiement**, et enfin le **service**.

Pour se connecter partiellement (cf partie Troubleshooting) à la bdd, nous avons utilisé un fichier configuration qui est déclaré dans un configMap au niveau de la variable "data". Le rôle de ce fichier est de spécifier des éléments reliés à la bdd (nom de la bdd, utilisateur ou bien encore le service de la bdd par exemple).

Pour faire cela, le fichier doit être placé dans le dossier pgadmin4 (dossier de configuration de connexion ) du conteneur de pod de pgadmin pour qu'il soit pris en compte.

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: pgadmin-config
  namespace: icgroup
  labels:
    app: pgadmin
    env: prod
data:
  pgpass: db-service:5432:postgres:odoo:odoo
  servers.json: |
    {
      "Servers": {
        "1": {
          "Name": "pg",
          "Group": "Servers",
          "Port": 5432,
          "Username": "odoo",
          "Host": "db-service",
          "PassFile": "/pgpass",
          "SSLMode": "prefer",
          "MaintenanceDB": "postgres"
        }
      }
    }

```

Pour la partie déploiement de pgadmin, les variable d'environnement qui sont déclaré c'est "PGADMIN\_DEFAULT\_EMAIL" pour l'adresse email de pgadmin et "PGADMIN\_DEFAULT\_PASSWORD" pour le mot de passe (récupéré à partir de manifest secret).

Au niveau des volumes, nous reions le dossier où sont stockées nos données de pgadmin avec le pvc, et nous mettons aussi le fichier server.json déclaré dans le configMap dans le dossier pgadmin4.

Voici des captures d'écran du manifest de déploiement (coupé en deux pour économiser de la place) :

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: pgadmin-deploy
  namespace: icgroup
  labels:
    app: pgadmin
    env: prod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pgadmin
  template:
    metadata:
      labels:
        app: pgadmin
    spec:
      containers:
        - name: pgadmin
          image: "dpape/pgadmin4"
          env:
            - name: PGADMIN_DEFAULT_EMAIL
              value: user@domain.com #je laisse la valeur par défaut
            - name: PGADMIN_DEFAULT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: pgadmin-secret
                  key: pgadmin-password

```

```

      key: pgadmin-password

    ports:
      - name: http
        containerPort: 80
        protocol: TCP

    volumeMounts:
      - name: pgadmin-volume
        mountPath: /var/lib/pgadmin # the working directory in which pgAdmin stores session data
      - name: pgadmin-config
        mountPath: /pgadmin4/servers.json
        subPath: servers.json
      - name: pgadmin-pass
        mountPath: /pgpass # /pgpass
        subPath: pgpass

    volumes:
      - name: pgadmin-config
        configMap:
          name: pgadmin-config
      - name: pgadmin-pass
        configMap:
          name: pgadmin-config
          defaultMode: 0600
      - name: pgadmin-volume
        persistentVolumeClaim:
          claimName: pgadmin-pvc

```

Et tout comme pour odoo, nous avons créé un service de type "NodePort", qui sera appelé par le site vitrine.

```

apiVersion: v1
kind: Service
metadata:
  name: pgadmin-service
  namespace: icgroup
  labels:
    app: pgadmin
    env: prod
spec:
  type: NodePort
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
      nodePort: 32125
  selector:
    app: pgadmin

```

## Troubleshooting

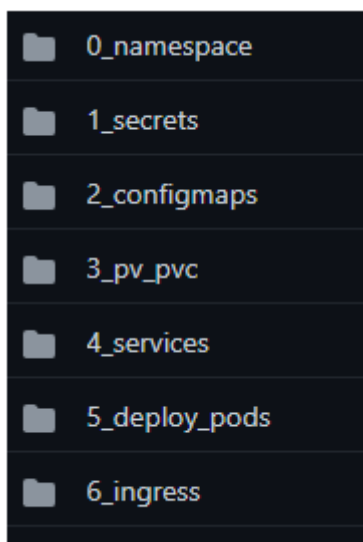
Nous n'avons pas pu faire une liaison complète avec notre bdd, la connexion avec le PassFile (fichier censé contenir le mot de passe de notre base de donnée ) ne passait pas, lors de la connection au serveur pgadmin une fenêtre apparaissait demandant un

mdp pour finaliser la connexion à notre base de donnée. Nous avons essayé de placer ce fichier dans différent emplacement à l'intérieur du conteneur, que ce soit à la racine ou bien dans le dossier de configuration pgadmin4, mais ça ne passait pas. Même en essayant de renseigner le fichier directement à partir de l'interface rien ny fait le fichier n'est pas pris en compte.

Il existe sans doute un moyen de se connecter avec le mdp dans le fichier de configuration, mais pour ne pas nous ralentir dans notre lancée, nous avons laissé ce détail-ci irrésolu.

### **Remarque :**

Cette partie représente la partie de mise en production, l'ordre d'énumération des étapes, n'est pas celui qui sera utilisé dans le fichier final jenkins étant donné que ce dernier ne représente pas une réalité applicative (il faut par exemple créer tout les pv ou bien les secrets en avance). Donc pour le pipeline de jenkins les manifests se lanceront de la façon suivante :



## 4 Mise en place d'un pipeline CI/CD

### 4.1 Jenkins

#### L'installation de Jenkins

L'installation de Jenkins s'est fait sur AWS - une instance t2.medium avec 20Go de stockage.

Pour l'installation nous nous sommes basé sur l'image de notre formateur Frazer : <https://github.com/sadofrazer/jenkins-frazer.git>

Cette image est néanmoins obsolète et n'embarque pas tout ce que l'on souhaite.

Pour l'installation de cette image, il a fallu faire l'installation de docker sur la machine Jenkins, en voici la procédure rapide :

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker centos
sudo systemctl start docker
sudo systemctl enable docker
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Ce script d'installation peut être installé dans les user data aux lancement de l'instance AWS.

#### Paquets utiles à installer sur le node

Plusieurs paquet ont été installés pour une meilleur stabilité de Jenkins et pour des utilisations plus poussés avec le gestionnaire de paquet yum :

java-11-openjdk.x86\_64 : optionnel mais il est toujours bien d'avoir une application à jour, il est recommandé par jenkins d'utiliser Java 11.

awscli : ce paquet nous servira principalement à gérer nos ressources de l'environnement test que l'on va créer au niveau du service ec2 d'aws

nodejs : ce paquet nodejs nous servira pour si l'on souhaite utiliser un snyk-cli au lieu de docker scan, et pouvoir aussi convertir nos rapports snyk.json en rapport.html plus lisible pour les différentes équipes opérationnels dont la branche sécurité évidemment qui auront besoin du rapport du vulnérabilité lisible de notre nouvelle image.

On pourra installer via npm `npm install snyk-to-html -g` (commande à exécuter sur le conteneur) puis exécuter la commande `snyk-to-html -i results.json -o results.html` pour convertir notre rapport `resultats.json` en `results.html` depuis notre pipeline en prenant comme agent notre node Jenkins via sh.

Installation de ansible sur la machine Jenkins, en temps normal, une partie de l'équipe était partie pour utiliser Jenkins comme le serveur ansible pour le déploiement via des rôles, mais une très bonne idée d'un des membres de l'équipe à susciter un changement, voici tout de même le processus pour l'installation d'ansible sur le node Jenkins.

```
sudo apt update -y
curl -sS https://bootstrap.pypa.io/get-pip.py | sudo python3
pip3 install ansible
ansible --version
```

### Connexion à l'interface web d'admin

La première fois que l'interface d'administration est lancée sur le port par défaut 8080, il vous faudra entrer le mot de passe d'initialisation si c'est votre première install, sur docker c'est simple il suffit de passer cette commande ci `docker exec jenkins-frazer_jenkins_1 cat /var/lib/jenkins/secrets/initialAdminPassword`.

Elle permet d'exécuter la commande car sur le conteneur en question.

Je vous suggère également d'installer les plugins par défaut, nous y reviendrons plus tard pour l'installation des plugins.

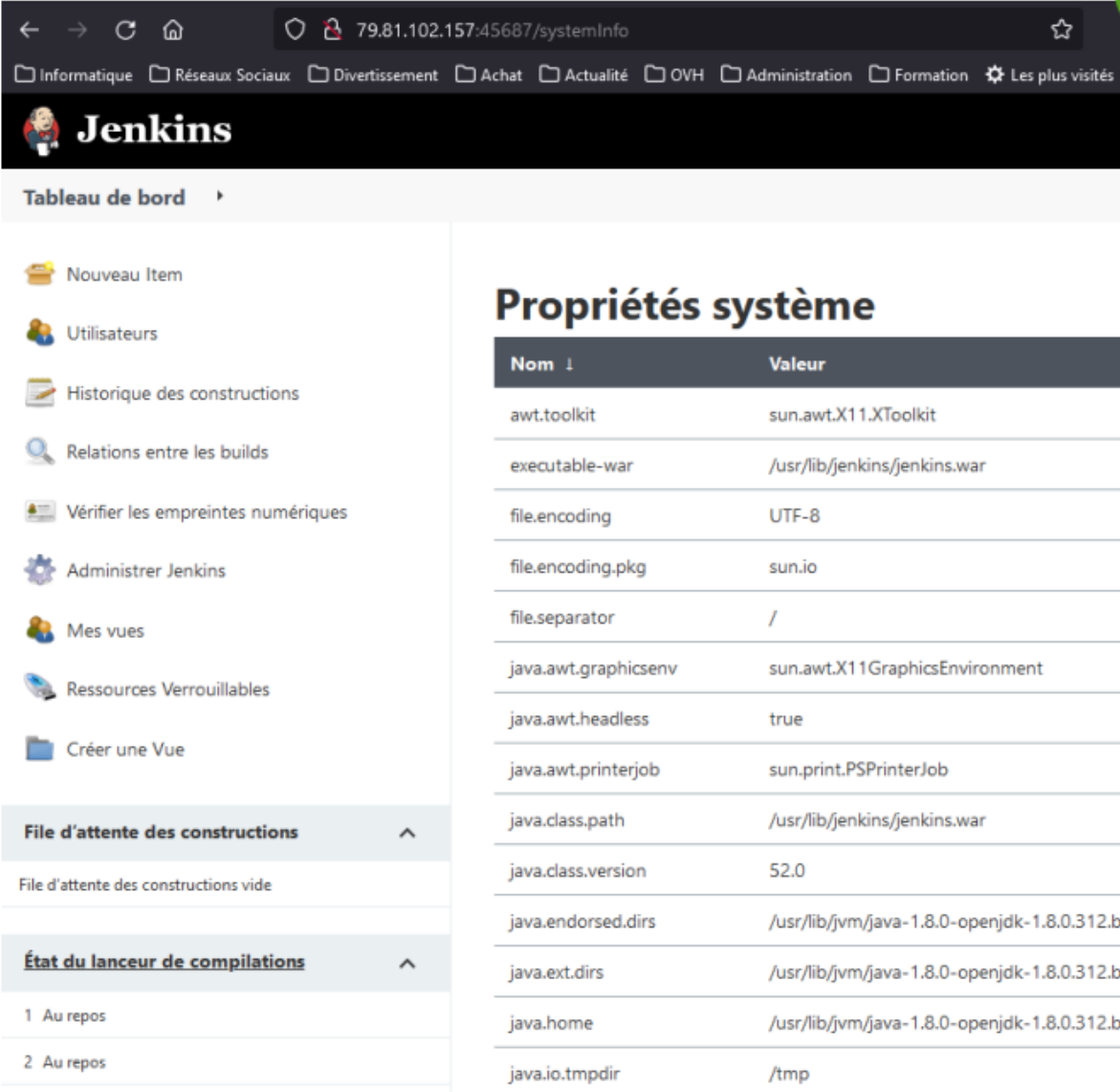
### Mise à jour de Jenkins

Mise à jour de Jenkins en version 2.319.2, pour la mise à jour de la version de notre Jenkins, nous nous sommes connectés directement aux conteneurs via la commande `docker exec -it jenkins-frazer_jenkins_1 /bin/bash`

Pour la mise à jour, il suffit de récupérer <https://get.jenkins.io/war-stable/2.319.2/jenkins.war> via un wget, puis déplacer le `jenkins.war` dans le répertoire `/usr/lib/jenkins/jenkins.war`

Le chemin de l'exécutable-war se trouve dans les informations de votre système de votre Jenkins, que vous pouvez retrouver ici :





The screenshot shows the Jenkins web interface in a browser. The address bar displays '79.81.102.157:45687/systemInfo'. The top navigation bar includes links for 'Informatique', 'Réseaux Sociaux', 'Divertissement', 'Achat', 'Actualité', 'OVH', 'Administration', 'Formation', and 'Les plus visités'. The main header features the Jenkins logo and the title 'Tableau de bord'. On the left sidebar, there are several menu items: 'Nouveau Item', 'Utilisateurs', 'Historique des constructions', 'Relations entre les builds', 'Vérifier les empreintes numériques', 'Administrer Jenkins', 'Mes vues', 'Ressources Verrouillables', and 'Créer une Vue'. The main content area is titled 'Propriétés système' and contains a table of system properties.

Nom ↓	Valeur
awt.toolkit	sun.awt.X11.XToolkit
executable-war	/usr/lib/jenkins/jenkins.war
file.encoding	UTF-8
file.encoding.pkg	sun.io
file.separator	/
java.awt.graphicsenv	sun.awt.X11GraphicsEnvironment
java.awt.headless	true
java.awt.printerjob	sun.print.PSPrinterJob
java.class.path	/usr/lib/jenkins/jenkins.war
java.class.version	52.0
java.endorsed.dirs	/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b
java.ext.dirs	/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b
java.home	/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.312.b
java.io.tmpdir	/tmp

Pour appliquer la mise à jour, il suffit de relancer le container avec les commandes suivantes

```
docker container stop jenkins-frazer_jenkins_1
docker container start jenkins-frazer_jenkins_1
```

### Listes des plugins :

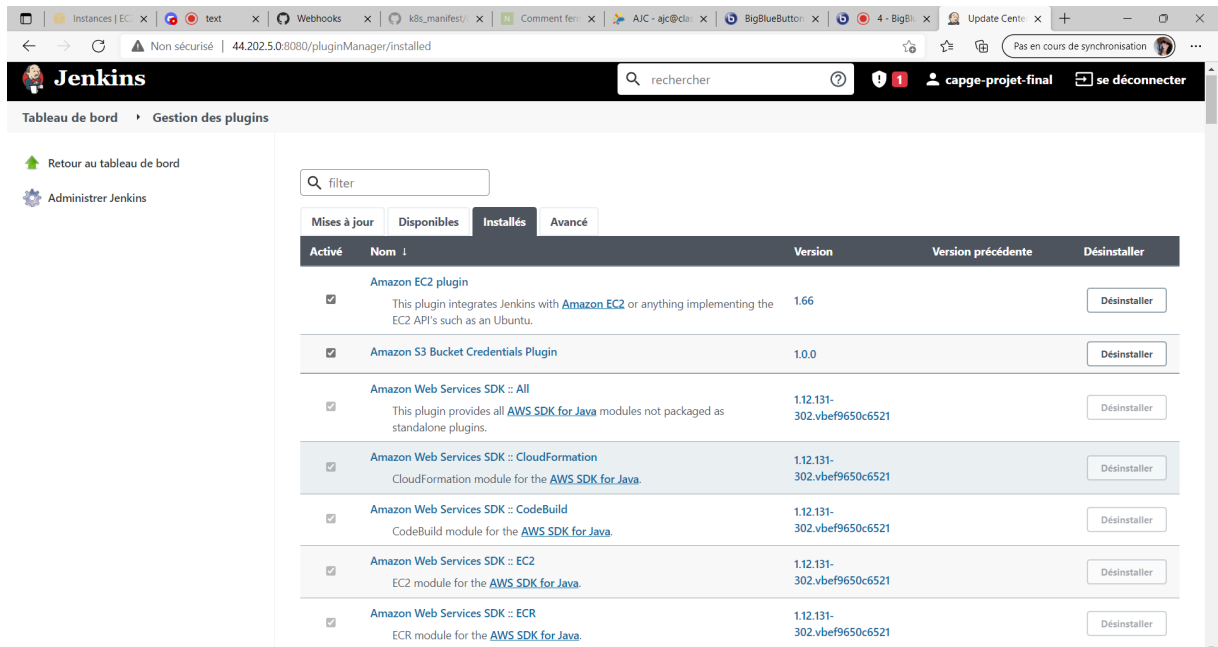
Il n'y a pas beaucoup de plugins utiles pour ce projet, mais l'installation de l'outil Terraform pour la partie plus tard.

L'extensions pour les webhooks afin que le webhooks avec git fonctionne sans souci.

Il y aura aussi l'embedded status pour afficher le status de notre build, de notre pipeline directement sur github par exemple. Pas nécessaire étant donné que l'image

que l'on utilise est lié avec le socket docker de l'hôte, prévoir tout de même l'installation de plugin Docker également.

On aurait pu utiliser Snyk comme plugin, mais ce Snyk n'est pas le snyk container, mais le snyk test, et ce qui nous intéresse c'est le Snyk container pour analyser notre image.



## Listes des credentials :

Ici je vous liste les credentials nécessaires pour que notre pipeline tourne sans souci : - token\_dockerhub : ce token servira à push notre build d'image sur le registry public, dockerhub

- capge\_key\_pair : cette paire d'utilisateur et clé privée contient la clé qui nous permettra de nous connecter sur nos instances aws. DE BONNES PRATIQUES AURAIENT VOULU QUE NOUS AYONS DEUX PAIRES DE CLES. Une paire de clé pour les connexions externes, puis une autre paire de clé pour les serveurs internes.

- AWS\_ACCESS\_KEY\_ID et AWS\_SECRET\_ACCESS\_KEY sont les clés permettant de s'authentifier sur notre compte aws pour faire des actions comme la création d'environnement via Terraform, ou le management de nos instances via AWS CLI.

- Et finalement snyk-api-token, n'est pas nécessaire car docker scan inclus déjà une api de Snyk, mais si l'on veut que nos rapports soit linker à notre compte il faudra utiliser ce token pour se logger.

Les credentials de github ici ne sont pas nécessaires, car le repos est public. Mais en temps normal les repos de futures releases sont privés ! Pour plus de flexibilité nous avons décidé d'utiliser des repos publics. Mais nous avons bien noté qu' en entreprise ça ne sera pas le cas, sauf si on fait de l'open source.

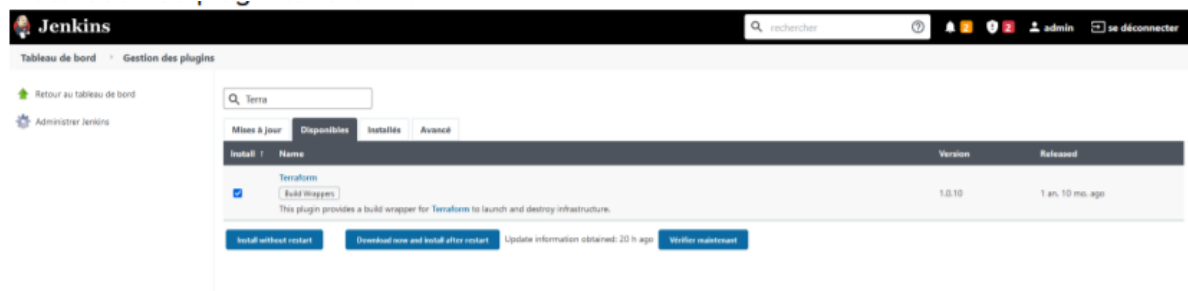
## Credentials

T	P	Store	Domain	ID	Name
		Jenkins	(global)	token_dockerhub	token_dockerhub
		Jenkins	(global)	capge_key_pair	ubuntu
		Jenkins	(global)	AWS_ACCESS_KEY_ID	AWS_ACCESS_KEY_ID
		Jenkins	(global)	AWS_SECRET_ACCESS_KEY	AWS_SECRET_ACCESS_KEY
		Jenkins	(global)	snyk-api-token	snyk-api-token

### Configuration des différents paramètres :

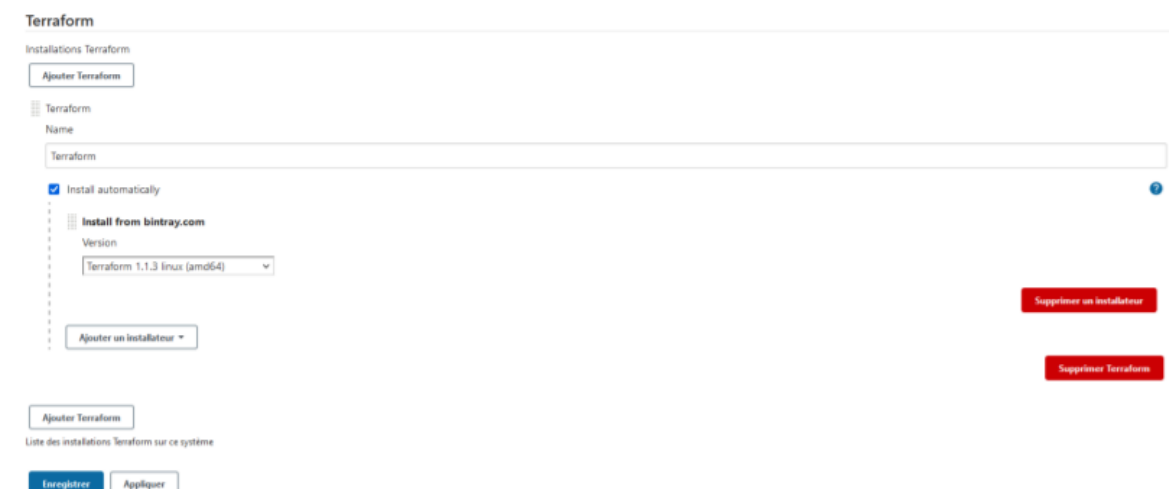
L'installation de Terraform se fait dans l'onglet qui gère les plugins : Allez dans Manage Jenkins > Manage Plugins > Available > Puis chercher Terraform.

Installation du plugin terraform :



Une fois installé nous allons configurer l'outil pour qu'il installe une version pour notre système de Terraform comme suit :

Allez dans Manage Jenkins > Global Tool Configuration > puis vous aurez une liste de Terraform. Il est important de donner un "Name" et choisir une version compatible avec votre système haha.



Dans notre pipeline nous allons l'invoquer avec l'instruction go suivante :

```
tools{  
  terraform Terraform  
}
```

## Mise en place du webhook à partir du repo Github :

Dans Settings ->Webhook, on ajoute l'URL du serveur Jenkins

The screenshot shows the 'Add webhook' form in the GitHub settings for the repository 'Thuy9906/ajc-projet-capge'. The left sidebar lists various settings categories, with 'Webhooks' selected. The main form area contains the following fields and options:

- Payload URL**: A text input field containing 'https://example.com/postreceive'.
- Content type**: A dropdown menu set to 'application/x-www-form-urlencoded'.
- Secret**: A text input field for a secret key.
- Which events would you like to trigger this webhook?**: Radio button options for 'Just the push event.', 'Send me everything.', and 'Let me select individual events.'.
- Active**: A checked checkbox with the text 'We will deliver event details when this hook is triggered.'
- Add webhook**: A green button at the bottom.

Et s'assurer que la requête POST de test retourne bien la réponse 200

The screenshot shows the 'Webhooks' section in the GitHub settings for the repository 'Thuy9906/ajc-projet-capge'. The left sidebar is the same as in the previous screenshot. The main area displays a list of configured webhooks:

- A single webhook is listed with a green checkmark icon, the URL 'http://44.202.5.0:8080/github-we...', and the event type '(push)'.
- Buttons for 'Edit' and 'Delete' are visible next to the webhook entry.
- An 'Add webhook' button is located at the top right of the list.

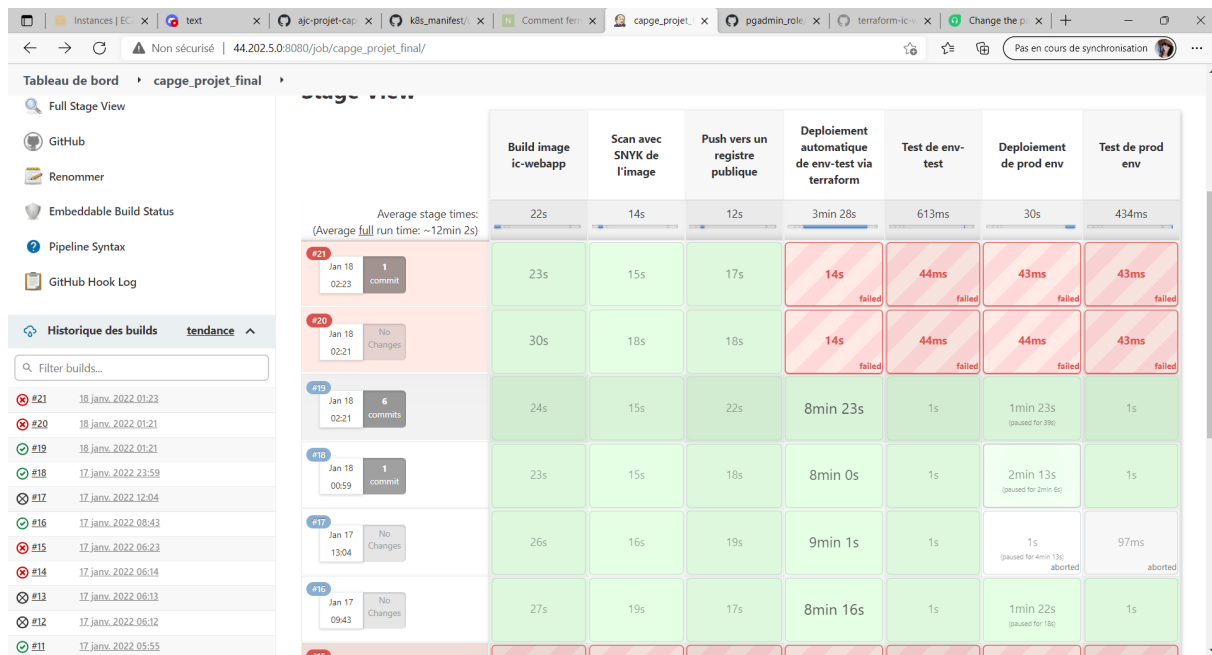
## La liste des requêtes envoyées (à chaque push du repo)

The screenshot shows the GitHub 'Webhooks / Manage webhook' page. On the left is a sidebar with navigation options: Options, Collaborators, Security & analysis, Branches, Webhooks (selected), Notifications, Integrations, Deploy keys, Actions, Environments, Secrets, Pages, and Moderation settings. The main area has two tabs: 'Settings' and 'Recent Deliveries'. The 'Recent Deliveries' tab shows a list of 14 webhook deliveries, each with a green checkmark icon, a unique ID, and a timestamp. The IDs are: a3b037c2-7837-11ec-9502-e9694f440a6d, 3eab3666-77ff-11ec-8493-d93343b92555, 35e6fbb6-77fd-11ec-9949-857a57f0d1e8, f9d1c0fc-77fc-11ec-9610-ff2fc08cd2b4, e15b9066-77fc-11ec-84d0-741d2bb888db, 1852dfe0-77f6-11ec-9315-a937c0534f7d, 6a7c6656-77f1-11ec-9e58-af27563174a7, 0c7fc384-77e9-11ec-8056-0539fb529da0, 78d94126-77e5-11ec-922d-e78c8b39c5e4, e91b4376-77e0-11ec-8a8f-eb1064a80563, dad2e8b4-77e0-11ec-94e1-dc437486222, d0995cc0-77e0-11ec-91b6-28c553a2197a, and da348f2a-77d1-11ec-9c29-edf95b44ffef. The timestamps range from 2022-01-18 09:21:31 to 2022-01-17 21:12:53.

## L'historique de build, avec confirmation manuelle pour lancer en prod

The screenshot shows a CI/CD dashboard for a project named 'capge\_projet\_final'. On the left is a sidebar with navigation options: Changes, Lancer un build, Configurer, Supprimer Pipeline, Full Stage View, GitHub, Renommer, Embeddable Build Status, Pipeline Syntax, and GitHub Hook Log. The main area has a 'Recent Changes' section and a 'Stage View' table. The 'Stage View' table shows the progress of builds across various stages. A confirmation dialog is open, asking 'Confirmer le déploiement sur la production de l'image ? [Cette action supprimera l'environnement de test]' with 'Yes' and 'Abort' buttons. The table has columns for 'Build image ic-webapp', 'Scan avec SNYK de l'image', 'Push vers un registre publique', 'Déploiement automatique de env-test via terraform', 'Test de env-test', 'Déploiement de prod env', and 'Test de prod env'. The rows represent different builds, with the most recent build (#22) highlighted in blue. The average stage times are: Build image ic-webapp (22s), Scan avec SNYK de l'image (13s), Push vers un registre publique (13s), Déploiement automatique de env-test via terraform (4min 18s), Test de env-test (722ms), Déploiement de prod env (30s), and Test de prod env (477ms).

	Build image ic-webapp	Scan avec SNYK de l'image	Push vers un registre publique	Déploiement automatique de env-test via terraform	Test de env-test	Déploiement de prod env	Test de prod env
Average stage times: (Average full run time: ~12min 2s)	22s	13s	13s	4min 18s	722ms	30s	477ms
#22 Jan 18 02:38 1 commit	23s						
#21 Jan 18 02:23 1 commit	23s	15s	17s	14s failed	44ms failed	43ms failed	43ms failed
#20 Jan 18 02:21 No Changes	30s	18s	18s	14s failed	44ms failed	44ms failed	43ms failed
#19 Jan 18 02:21 6 commits	24s	15s	22s	8min 23s	1s	1min 23s	1s



## 4.2 Ansible

### Création des Rôles Ansible

Le but étant de déployer des conteneurs docker, nous avons déterminé 4 rôles :

- `docker_role` : qui permet d'installer tous les outils nécessaire pour la création de conteneur docker et de docker-compose
- `odoo_role` : qui lancera 2 conteneurs "connectés" celui de la base de donnée postgres et celui de odoo
- `pgadmin_role` : qui lancera un conteneur pgadmin (permettant de visualiser la base de donnée postgres de odoo)
- `ic-webapp_role` : lance notre site vitrine ic-webapp.

Pour des raisons d'unicité, et à cause d'un problème rencontré avec pgadmin (voir partie Troubleshooting), nous avons choisi de déployer tous nos conteneur via `docker_compose` de ansible.

Pour des raisons économique et fonctionnelles nous avons décidé que :

- Les machines cibles seront des serveur ubuntu 20.04 LTS (sur aws)
- La région sera la Virginie du Nord
- L'utilisateur par défaut du système sera "ubuntu"

## Rôle Docker

source : [https://github.com/lianhuhayu/docker\\_role.git](https://github.com/lianhuhayu/docker_role.git)

- Installe Docker via le script officiel "get-docker.sh"
- Ajoute et lance le service docker au démarrage du serveur
- Ajoute l'utilisateur "ubuntu" au groupe docker
- Installer python3-pip
- Installer docker-compose (via le script officiel)
- Installe docker-compose via pip (indispensable pour les commandes ansible docker\_compose)

### *tasks/main.yml*

```
- name: "pre-requis pour installer docker"
  package:
    name: curl
    state: present

- name: "Recuperation du script d installation de Docker"
  command: 'curl -fsSL https://get.docker.com -o get-docker.sh'

- name: "Executer le script d'installation de Docker"
  command: "sh get-docker.sh"
  when: ansible_docker0 is undefined

- name: "Démarrage et ajout au redémarrage du service Docker"
  service:
    name: docker
    state: started
    enabled: yes

- name: "Ajout de notre utilisateur au groupe docker"
  user:
    name: "{{ ansible_user }}"
    append: yes
    groups:
      - docker

- name: "Installation de python pip pour ubuntu"
  apt:
    name: python3-pip
    state: present
  when: ansible_distribution == "Ubuntu"

- name: "prerequis dockercompose"
  command: 'curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local.'

- name: "install docker cmpose"
  command: 'chmod +x /usr/local/bin/docker-compose'

- name: "Installation du module docker-compose"
  pip:
    name: docker-compose
    state: present
```

## Rôle Odoo

source : [https://github.com/lianhuahayu/odoo\\_role.git](https://github.com/lianhuahayu/odoo_role.git)

- Va déployer 2 conteneurs avec le template docker-compose:
  - conteneur backend\_odoo
    - basé sur une image postgres:13
    - exposant le port 5432
    - initialisant les variables d'environnement avec
      - postgres\_user (utilisateur de la database)
      - postgres\_password (mot de passe de la database)
      - postgres\_database (nom de la database)
    - persistant les données avec un volume
  - conteneur frontend\_odoo
    - basé sur une image odoo:13
    - exposant le port 8069
    - initialisant les variables d'environnement avec celles de postgres
    - spécifiant aussi l'adresse ip du host de postgres
    - lançant une commande odoo permettant d'initialiser la database pour ne pas le faire manuellement
    - persistant les données avec un volume
- Les deux conteneurs seront sur un network commun
- Toutes les données sont variabilisées donc pourront être surchargée par ansible (-e)

### *defaults/main.yml*

```
---
# Utilisateur par défaut
ansible_user: user
ansible_sudo_pass: user

# Postgres variables
postgres_container: backend_odoo
postgres_image: postgres:13
postgres_port: 5432
postgres_database: odoo
postgres_user: odoo
postgres_password: odoo
postgres_data: backup-postgres-db-data

# Odoo variables
odoo_container: frontend_odoo
odoo_image: odoo:13.0
odoo_port: 8069
odoo_data: backup-odoo-data

# Nom du réseau
network_name: odoo
```



## tasks/main.yml

```
---
- name: "Ajout du template docker-compose.yml"
  template:
    src: "docker-compose.yml.j2"
    dest: /home/{{ ansible_user }}/docker-compose.yml

- name: "Lancer le fichier docker-compose.yml"
  docker_compose:
    project_src: /home/{{ ansible_user }}
    files:
      - docker-compose.yml

- name: "Relancer le container odoo apres l'init de la db"
  shell: |
    sleep 15
    docker container restart {{ odoo_container }}
  exit 0
```

## templates/docker-compose.yml.j2

```
version: '2'
services:
  {{ postgres_container }}:
    container_name: {{ postgres_container }}
    image: {{ postgres_image }}
    restart: always
    ports:
      - "{{ postgres_port }}:5432"
    environment:
      - POSTGRES_DB={{ postgres_database }}
      - POSTGRES_PASSWORD={{ postgres_password }}
      - POSTGRES_USER={{ postgres_user }}
      - PGDATA=/var/lib/postgresql/data/pgdata
    volumes:
      - {{ postgres_data }}:/var/lib/postgresql/data/pgdata
    networks:
      - {{ network_name }}

  {{ odoo_container }}:
    container_name: {{ odoo_container }}
    image: {{ odoo_image }}
    restart: always
    depends_on:
      - {{ postgres_container }}
    ports:
      - "{{ odoo_port }}:8069"
    environment:
      - HOST={{ postgres_container }}
      - PORT={{ postgres_port }}
      - USER={{ postgres_user }}
      - PASSWORD={{ postgres_password }}
    command: odoo -i base -d {{ postgres_database }} --db_host={{ postgres_container }} -r {{ postgres_user }} -w {{ postgres_password }}
    volumes:
      - {{ odoo_data }}:/var/lib/odoo
    networks:
      - {{ network_name }}

networks:
  {{ network_name }}:
    name: {{ network_name }}
    driver: bridge

volumes:
  {{ odoo_data }}:
  {{ postgres_data }}:
```

## Rôle PgAdmin

source : [https://github.com/Yellow-carpet/pgadmin\\_role.git](https://github.com/Yellow-carpet/pgadmin_role.git)

- Va déployer un conteneur pgAdmin via le template docker-compose
  - basé sur une image de dpage/pgadmin4
  - Initialiser les variables d'environnement avec :
    - pgadmin\_email (email et login de l'utilisateur)
    - pgadmin\_pass (mot de passe)
    - pgadmin\_port (port de l'application par défaut 80)
  - Persistance du fichier /pgadmin4/servers.json permettant à l'initialisation, d'avoir déjà du contenu : celui de la base de donnée odoo
- Va copier le fichier "templatisé" servers.json sur la machine distante contenant toute les informations de la base de donnée odoo
- Toutes les données sont variabilisées donc pourront être surchargée par ansible

### *defaults/main.yml*

```
---
# servers.json variables
host_db: 172.31.82.22
port_db: 5432
maintenanceDB: postgres
username_db: odoo

# pgadmin docker-compose variables
pgadmin_email: pgadmin@pgadmin.com
pgadmin_pass: pgadmin
pgadmin_port: 5050
```

### *tasks/main.yml*

```
---
- name: "Ajout du template pgadmin docker-compose.yml"
  template:
    src: "docker-compose.yml.j2"
    dest: /home/{{ ansible_user }}/docker-compose.yml

- name: "Ajout du template servers.json"
  template:
    src: "servers.json.j2"
    dest: /home/{{ ansible_user }}/servers.json

- name: "Lancer le fichier docker-compose.yml pour pgadmin"
  docker_compose:
    project_src: /home/{{ ansible_user }}
    files:
      - docker-compose.yml
```

## *templates/docker-compose.yml.j2*

```
version: "3.3"
services:
  pgadmin:
    image: dpape/pgadmin4
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: {{ pgadmin_email }} #the username to login to pgadmin
      PGADMIN_DEFAULT_PASSWORD: {{ pgadmin_pass }} # the password to login to pgadmin
    ports:
      - "{{ pgadmin_port }}:80"
    volumes:
      - ./servers.json:/pgadmin4/servers.json # preconfigured servers/connections
```

## *templates/servers.json.j2*

```
{
  "Servers": {
    "1": {
      "Name": "docker_postgres",
      "Group": "docker_postgres_group",
      "Host": "{{ host_db }}",
      "Port": {{ port_db }},
      "MaintenanceDB": "{{ maintenanceDB }}",
      "Username": "{{ username_db }}",
      "SSLMode": "prefer"
    }
  }
}
```

## Rôle ic-webapp

source: [https://github.com/omarpiotr/ic-webapp\\_role](https://github.com/omarpiotr/ic-webapp_role)

- Va déployer un conteneur ic-webapp via le template docker-compose
  - basé par défaut sur l'image "lianhuahayu/ic-webapp:1.0" variabilisée que l'on devra surcharger par la suite.
  - Initialiser les variables d'environnement avec :
    - odoo\_url : l'url de notre site odoo sous la forme <http://url-odoo:8069>
    - pgadmin\_url : l'url de notre site pgAdmin sous la forme <http://url-pgadmin:5050>
    - ic\_webapp\_port : port exposé par notre application vers l'extérieur
- Toutes les données sont variabilisées donc pourront être surchargée par ansible

## *defaults/main.yml*

```
# defaults file for ic-webapp_role
ic_webapp_image: "lianhuahayu/ic-webapp:1.0"
odoo_url: "127.0.0.1:8069"
pgadmin_url: "127.0.0.1:5050"
ic_webapp_port: 80
```

## *tasks/main.yml*

```
---
# tasks file for ic-webapp_role

- name: "Ajout du template ic-webapp_role docker-compose.yml"
  template:
    src: "docker-compose.yml.j2"
    dest: /home/{{ ansible_user }}/docker-compose-ic.yml

- name: "Lancer le fichier docker-compose.yml"
  docker_compose:
    project_src: /home/{{ ansible_user }}
    files:
      - docker-compose-ic.yml
```

## *templates/docker-compose.yml.j2*

```
version: "3.3"
services:
  ic-webapp:
    image: {{ ic_webapp_image }}
    restart: always
    environment:
      ODOO_URL: {{ odoo_url }}
      PGADMIN_URL: {{ pgadmin_url }}
    ports:
      - "{{ ic_webapp_port }}:8080"
```

## Troubleshooting

Lorsque nous avons déployé pgAdmin à l'aide de docker\_container, nous nous sommes rendu compte que le fichier "servers.json", bien que présent, n'est pas pris en charge par pgAdmin.

Selon la documentation officielle ci-dessous, le fichier servers.json n'est lu qu'une seule et unique fois, c'est au moment du premier lancement. Etant donné que ce fichier était partagé à l'aide de la propriété volume, et suite à de nombreux divers tests manuels que nous avons effectué nous en avons conclu que docker\_container:

- 1- lance d'abord le conteneur

- 2- montait les volumes ensuite de ce fait le fichier n'était pas pris en compte. Pour résoudre ce problème nous avons utilisé docker\_compose.

```
/pgadmin4/servers.json
```



If this file is mapped, server definitions found in it will be loaded at launch time.  
This allows connection information to be pre-loaded into the instance of pgAdmin in the container.  
Note that server definitions are only loaded on first launch, i.e. when the configuration database is created, and not on subsequent

## Les Playbooks Ansible

Afin de consommer et tester nos rôles, nous avons créé 3 playbooks

- `playbook_odoo` :
  - `docker_role`
  - `odoo_role`
- `playbook_pgadmin`
  - `docker_role`
  - `pgadmin_role`
- `playbook_ic-webapp`
  - `docker_role`
  - `ic-webapp_role`

### *roles/requirements.yml*

```
- src: https://github.com/lianhuhayu/docker_role.git
- src: https://github.com/lianhuhayu/odoo_role.git
- src: https://github.com/Yellow-carpet/pgadmin_role.git
- src: https://github.com/omarpiotr/ic-webapp_role.git
```

### *playbook\_odoo.yml*

```
---
- name: "deploy Odoo with a role"
  hosts: ansible
  become: true
  roles:
    - docker_role
    - odoo_role
```

### *playbook\_pgadmin.yml*

```

---
- name: "deploy pgadmin with a role"
  hosts: ansible
  become: true
  roles:
    - docker_role
    - pgadmin_role

```

### ***playbook\_ic-webapp.yml***

```

---
- name: "deploy pgadmin with a role"
  hosts: ansible
  become: true
  roles:
    - docker_role
    - ic-webapp_role

```

### ***hosts\_.yml***

```

all:
  children:
    ansible:
      hosts:
        localhost:
          ansible_connection: local
          ansible_user: "ubuntu"
          hostname: AnsibleMaster
          ansible_ssh_common_args: "-o StrictHostKeyChecking=no"

```

Afin de tester nos role nous avons créer les deux instances suivantes :

- instance ec2 odoo, sur laquelle on souhaite avoir odoo (frontend + backend)
  - t2.micro
  - ip : 44.201.245.76
  - dns : ec2-44-201-245-76.compute-1.amazonaws.com
  - sécurité Group : 22 / 5432 / 8069
- instance ec2 server, sur laquelle on souhaite avoir pgAdmin et ic-webapp
  - t2.micro
  - ip : 52.87.202.243
  - dns : ec2-52-87-202-243.compute-1.amazonaws.com
  - sécurité Group : 22 / 80 / 5050
- Nous disposons aussi d'un clé privé permettant de se connecter aux deux instances ec2 ci-dessus:
  - /home/ubuntu/.ssh/capge\_projet\_kp.pem
- Pour lancer les playbooks, nous devons surcharger les variables d'environnement suivantes
  - ansible\_connection : spécifier qu'il s'agit d'une connexion ssh
  - ansible\_host : spécifier l'adresse ip de l'instance ec2 hôte sur laquelle on exécute le playbook

- host\_db : l'adresse ip de la machine sur laquelle se trouve la base de donnée postgres
- odoo\_url : l'adresse dns publique ou ip publique de l'instance ec2 odoo
- pgadmin\_url : l'adresse dns publique ou ip publique de l'instance ec2 server
- ic\_webapp\_image : image docker de ic-webapp sur docker-hub
- ic\_webapp\_port (*optionnel*) : port exposé (externe) de l'application ic-webapp
- postgres\_image : image docker de postgres sur docker-hub
- odoo\_image : image docker de odoo sur docker-hub

## Commandes ansible à exécuter (sur une machine disposant ansible)

### install requirements

ansible-galaxy install -r roles/requirements.yml

```

TERMINAL  PORTS  DEBUG CONSOLE  PROBLEMS  OUTPUT
ubuntu@ip-172-31-94-128:~/ansible_deploy_ic-webapp$ ansible-galaxy install -r roles/requirements.yml
- extracting docker_role to /home/ubuntu/.ansible/roles/docker_role
- docker_role was installed successfully
- extracting odoo_role to /home/ubuntu/.ansible/roles/odoo_role
- odoo_role was installed successfully
- extracting pgadmin_role to /home/ubuntu/.ansible/roles/pgadmin_role
- pgadmin_role was installed successfully
- extracting ic-webapp_role to /home/ubuntu/.ansible/roles/ic-webapp_role
- ic-webapp_role was installed successfully
ubuntu@ip-172-31-94-128:~/ansible_deploy_ic-webapp$

```

### deploy odoo container on odoo ec2

```

# deploy odoo container on odoo ec2
ansible-playbook -i hosts.yml playbook_odoo.yml \
  -e ansible_connection='ssh' \
  -e ansible_host='44.201.245.76' \
  -e postgres_image='postgres:10' \
  -e odoo_image='odoo:13.0' \
  --private-key '/home/ubuntu/.ssh/capge_projet_kp.pem'

```

## Console

```
TERMINAL  PORTS  DEBUG CONSOLE  PROBLEMS  OUTPUT
TASK [gathering Facts] *****
ok: [localhost]

TASK [docker_role : pre-requis pour installer docker] *****
ok: [localhost]

TASK [docker_role : Recuperation du script d installation de Docker] *****
[WARNING]: Consider using the get_url or uri module rather than running 'curl'. If you need to use command because get_url or uri is insufficient you can add 'warn: false' to this command task or set
'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [localhost]

TASK [docker_role : Executer le script d'installation de Docker] *****
changed: [localhost]

TASK [docker_role : Démarrage et ajout au redémarrage du service Docker] *****
ok: [localhost]

TASK [docker_role : Ajout de notre utilisateur au groupe docker] *****
changed: [localhost]

TASK [docker_role : Installation de python pip pour ubuntu] *****
changed: [localhost]

TASK [docker_role : prerequis dockercompose] *****
changed: [localhost]

TASK [docker_role : install docker cipose] *****
[WARNING]: Consider using the file module with mode rather than running 'chmod'. If you need to use command because file is insufficient you can add 'warn: false' to this command task or set 'command_warnings=False' in
ansible.cfg to get rid of this message.
changed: [localhost]

TASK [docker_role : Installation du module docker-compose] *****
changed: [localhost]

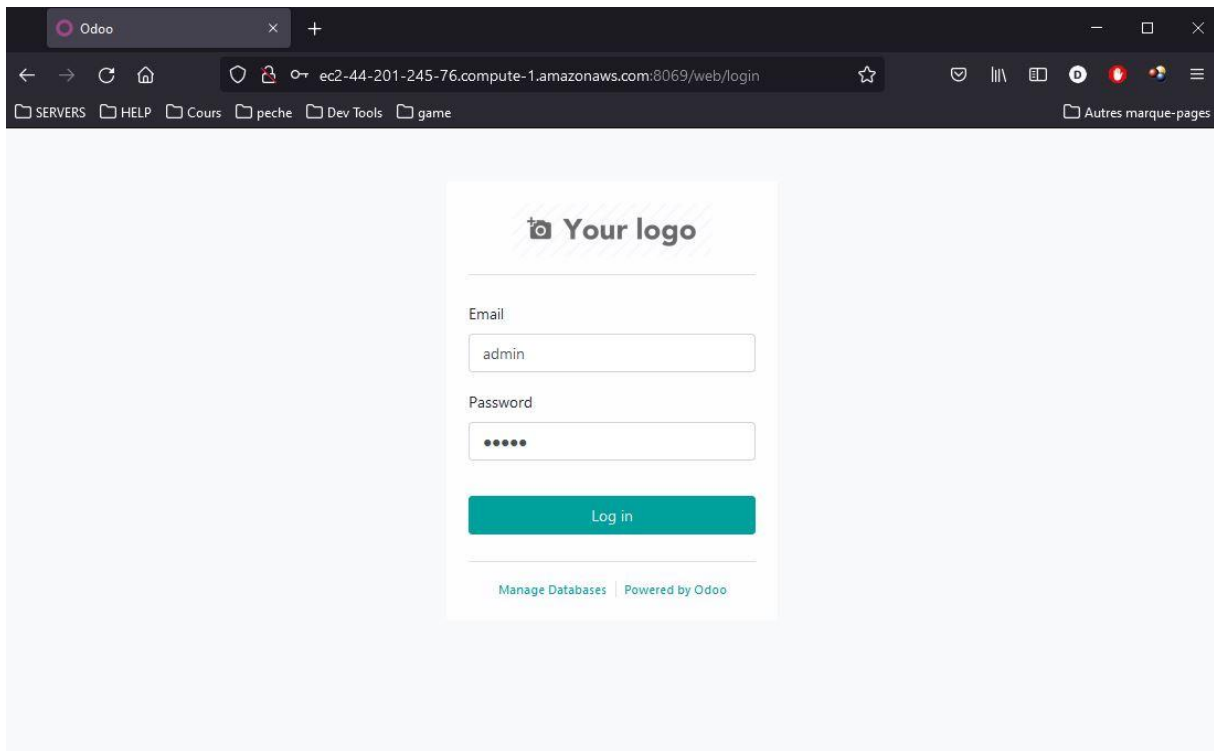
TASK [odoo_role : Ajout du template docker-compose.yml] *****
changed: [localhost]

TASK [odoo_role : Lancer le fichier docker-compose.yml] *****
changed: [localhost]

TASK [odoo_role : Relancer le container odoo apres l'init de la db] *****
changed: [localhost]

PLAY RECAP *****
localhost      : ok=13  changed=10  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

Site odoo





## deploy pgadmin on server ec2

```
# deploy pgadmin on server ec2
ansible-playbook -i hosts.yml playbook_pgadmin.yml \
  -e ansible_connection='ssh' \
  -e ansible_host='52.87.202.243' \
  -e host_db='44.201.245.76' \
  --private-key '/home/ubuntu/.ssh/capge_projet_kp.pem'
```



## Console

```
TERMINAL  PORTS  DEBUG CONSOLE  PROBLEMS  OUTPUT
TASK [gathering Facts] *****
ok: [localhost]

TASK [docker_role : pre-requis pour installer docker] *****
ok: [localhost]

TASK [docker_role : Recuperation du script d installation de Docker] *****
[WARNING]: Consider using the get_url or uri module rather than running 'curl'. If you need to use command because get_url or uri is insufficient you can add 'warn: false' to this command task or set
'command_warnings=false' in ansible.cfg to get rid of this message.
changed: [localhost]

TASK [docker_role : Executer le script d'installation de Docker] *****
changed: [localhost]

TASK [docker_role : Démarrage et ajout au redémarrage du service Docker] *****
ok: [localhost]

TASK [docker_role : Ajout de notre utilisateur au groupe docker] *****
changed: [localhost]

TASK [docker_role : Installation de python pip pour ubuntu] *****
changed: [localhost]

TASK [docker_role : prerequis dockercompose] *****
changed: [localhost]

TASK [docker_role : Install docker compose] *****
[WARNING]: Consider using the file module with mode rather than running 'chmod'. If you need to use command because file is insufficient you can add 'warn: false' to this command task or set 'command_warnings=false' in
ansible.cfg to get rid of this message.
changed: [localhost]

TASK [docker_role : Installation du module docker-compose] *****
changed: [localhost]

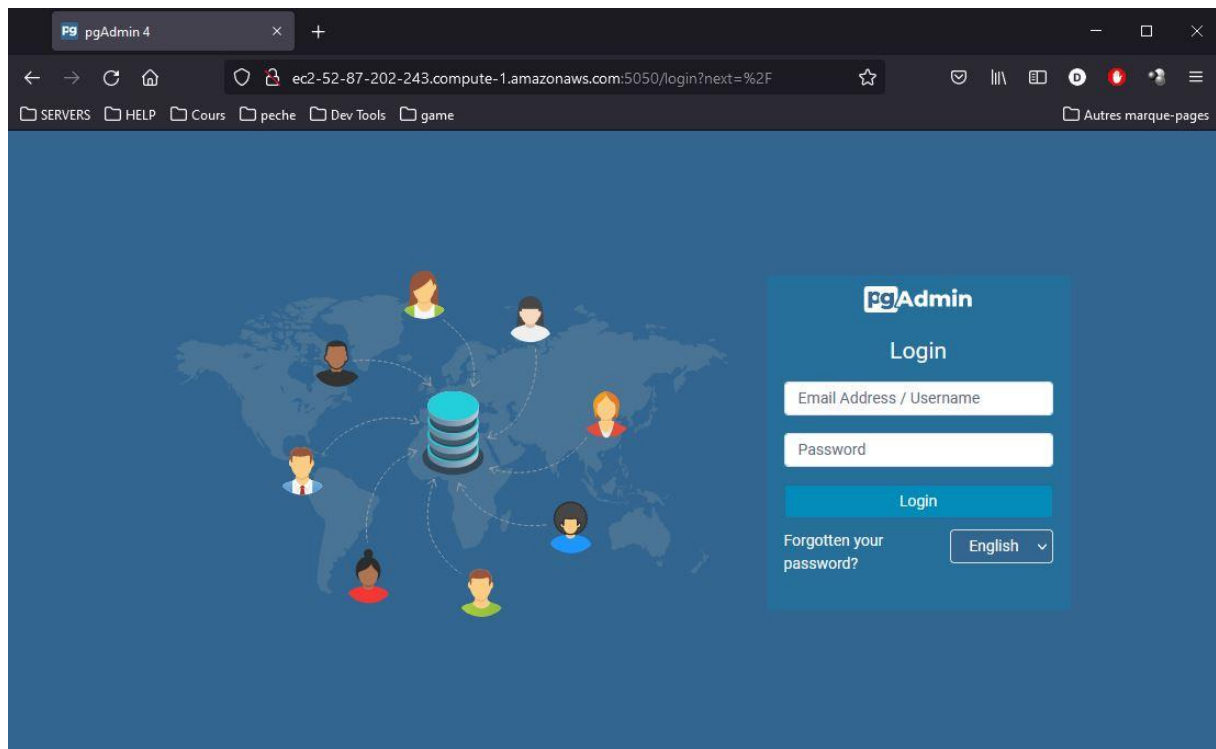
TASK [pgadmin_role : Ajout du template pgadmin docker-compose.yml] *****
changed: [localhost]

TASK [pgadmin_role : Ajout du template servers.json] *****
changed: [localhost]

TASK [pgadmin_role : Lancer le fichier docker-compose.yml pour pgadmin] *****
changed: [localhost]

PLAY RECAP *****
localhost : ok=13  changed=10  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

## Site PgAdmin



## deploy ic-webapp on server ec2

```
# deploy ic-webapp on server ec2
ansible-playbook -i hosts.yml playbook_ic-webapp.yml \
  -e ansible_connection='ssh' \
  -e ansible_host='52.87.202.243' \
  -e odoo_url='http://ec2-44-201-245-76.compute-1.amazonaws.com:8069' \
  -e pgadmin_url='http://ec2-52-87-202-243.compute-1.amazonaws.com:5050' \
  -e ic_webapp_image='lianhuahayu/ic-webapp:1.0' \
  -e ic_webapp_port='80' \
  --private-key '/home/ubuntu/.ssh/capge_projet_kp.pem'
```

## Console

```

PLAY [deploy pgadmin with a role] *****
TASK [Gathering Facts] *****
ok: [localhost]

TASK [docker_role : pre-requis pour installer docker] *****
ok: [localhost]

TASK [docker_role : Recuperation du script d'installation de Docker] *****
[WARNING]: Consider using the get_url or uri module rather than running 'curl'. If you need to use command because get_url or uri is insufficient you can add 'warn: false' to this command task or set 'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [localhost]

TASK [docker_role : Executer le script d'installation de Docker] *****
skipping: [localhost]

TASK [docker_role : Démarrage et ajout au redémarrage du service Docker] *****
ok: [localhost]

TASK [docker_role : Ajout de notre utilisateur au groupe docker] *****
ok: [localhost]

TASK [docker_role : Installation de python pip pour ubuntu] *****
ok: [localhost]

TASK [docker_role : prerequis dockercompose] *****
changed: [localhost]

TASK [docker_role : install docker cpose] *****
[WARNING]: Consider using the file module with mode rather than running 'chmod'. If you need to use command because file is insufficient you can add 'warn: false' to this command task or set 'command_warnings=False' in ansible.cfg to get rid of this message.
changed: [localhost]

TASK [docker_role : Installation du module docker-compose] *****
ok: [localhost]

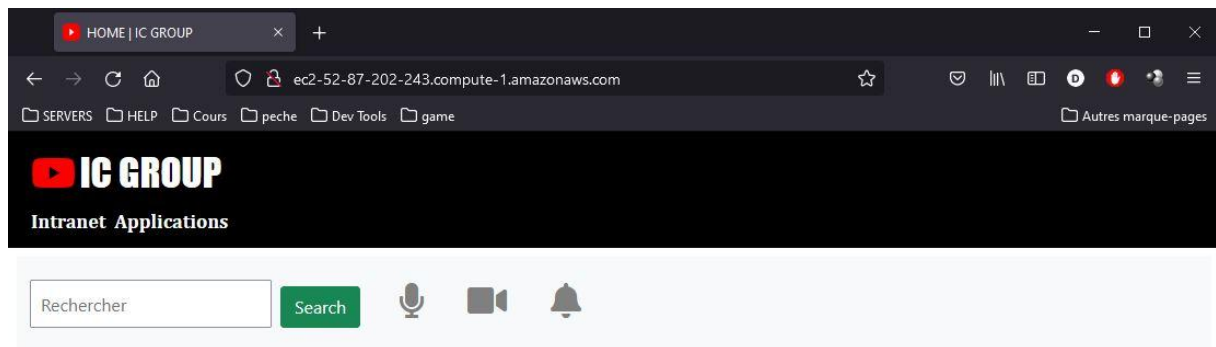
TASK [ic-webapp_role : Ajout du template ic-webapp_role docker-compose.yml] *****
changed: [localhost]

TASK [ic-webapp_role : Lancer le fichier docker-compose.yml] *****
changed: [localhost]

PLAY RECAP *****
localhost : ok=11  changed=5  unreachable=0  failed=0  skipped=1  rescued=0  ignored=0

```

## Site ic-webapp



## 4.3 Terraform

### Déploiement de l'environnement dev avec TERRAFORM

#### Réflexion

- Les instances ec2 qui sur lesquelles seront déployés les conteneurs sont de type t2.micro (imposé)
- Ansible requiert au minimum une machine de type t2.medium
- Nous avons choisi de ne pas surcharger le serveur Jenkins avec Ansible afin qu'il reste uniquement dédié à son rôle de CI/CD.
- Nous aurons donc besoin de créer 3 instances :
  - x1 instance ec2 Master pour Ansible de type t3.medium qui lancera les playbooks
  - x2 instances ec2 Worker de type t2.micro
    - Serveur Admin (ic-webapp et pgadmin)
    - Serveur Odoo (odoo frontend et odoo backend)

#### Les modules

- sg : module permettant de mettre un place un group de sécurité aws
- ec2\_master : module permettant de créer une instance t3.medium et qui lancera les commandes ansible
- ec2\_worker : module permettant des instance t2.micro

**module "sg" security group**

```

resource "aws_security_group" "web-sg" {
  name = var.sg_name
  description = "Allow inbound traffic with port 22 & 80 & 443"

  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [ "::/0" ]
  }

  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [ "::/0" ]
  }

  ingress {
    from_port = 5050
    to_port = 5050
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [ "::/0" ]
  }

  ingress {
    from_port = 8080
    to_port = 8080
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [ "::/0" ]
  }

  ingress {
    from_port = 8069
    to_port = 8069
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [ "::/0" ]
  }

  ingress {
    from_port = 5432
    to_port = 5432
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = [ "::/0" ]
  }
}

```



## module "ec2\_worker"

```
resource "aws_instance" "myec2_worker" {
  ami           = var.ami
  instance_type = var.instance_type
  key_name      = var.key_name
  security_groups = var.sg_group

  root_block_device {
    delete_on_termination = true
  }

  tags = {
    Name        = "capge-${var.env}-${var.serveur}"
    formation   = "Frazer"
    iac         = "terraform"
  }
}
```

```
variable "ami" {
  default = "ami-04505e74c0741db8d"
  type    = string
}

variable "instance_type" {
  default = "t2.micro"
}

variable "key_name" {
  default = "capge_projet_kp"
}

variable "sg_group" {
  type = list(string)
  default = ["security_group_capge"]
}

variable "username" {
  default = "ubuntu"
}

variable "env" {
  default = "dev"
}

variable "serveur" {
  default = "admin"
}

variable "private_key_path" {
  default = "D:/Formation/AJC/05.DevOps/PROJET/capge_projet_kp.pem"
}

output "ec2_ip" {
  value = aws_instance.myec2_worker.public_ip
}

output "ec2_dns" {
  value = aws_instance.myec2_worker.public_dns
}
```

## module "ec2\_master"

```
resource "aws_instance" "myec2_master" {
  ami           = var.ami
  instance_type = var.instance_type
  key_name      = var.key_name
  security_groups = var.sg_group

  root_block_device {
    delete_on_termination = true
  }

  tags = {
    Name        = "capge-${var.env}-${var.serveur}"
    formation   = "Frazer"
    iac         = "terraform"
  }

  provisioner "file" {
    source      = var.private_key_path
    destination = "/home/ubuntu/.ssh/capge_projet_kp.pem"

    connection {
      type      = "ssh"
      user      = "ubuntu"
      private_key = file(var.private_key_path)
      host      = self.public_ip
    }
  }

  provisioner "remote-exec" {
    inline = [
      "sleep 15",
      "sudo apt-get update -y",
      "sleep 5",
      "chmod 400 /home/ubuntu/.ssh/capge_projet_kp.pem",
      "sudo apt-get install ansible -y",
      "sudo apt-get install sshpass -y",
      "mkdir ansible-deploy",
      "git clone https://github.com/omarpiotr/ansible_deploy_ic-webapp.git ./ansible-deploy",
      "cd ./ansible-deploy",
      "ansible-galaxy install -r roles/requirements.yml",
      "ansible-playbook -i hosts.yml playbook_odoo.yml -e ansible_connection='ssh' \\",
      "  -e ansible_host='${var.ec2_odoo_ip}' \\",
      "  --private-key '/home/ubuntu/.ssh/capge_projet_kp.pem'",
      "ansible-playbook -i hosts.yml playbook_pgadmin.yml -e ansible_connection='ssh' \\",
      "  -e ansible_host='${var.ec2_server_ip}' \\",
      "  -e host_db='${var.ec2_odoo_ip}' \\",
      "  --private-key '/home/ubuntu/.ssh/capge_projet_kp.pem'",
      "ansible-playbook -i hosts.yml playbook_ic-webapp.yml -e ansible_connection='ssh' \\",
      "  -e ansible_host='${var.ec2_server_ip}' \\",
      "  -e odoo_url='http://${var.odoo_dns}:${var.odoo_port}' \\",
      "  -e pgadmin_url='http://${var.pgadmin_dns}:${var.pgadmin_port}' \\",
      "  -e ic_webapp_image='${var.ic-webapp_image}' \\",
      "  --private-key '/home/ubuntu/.ssh/capge_projet_kp.pem'"
    ]
  }
}
```

```
connection {
  type      = "ssh"
  user      = "ubuntu"
  private_key = file(var.private_key_path)
  host      = self.public_ip
}

provisioner "local-exec" {
  command = "echo vitrine : ${var.pgadmin_dns} - pgadmin:${var.ec2_server_ip}:5050 - odoo:${var.ec2_odoo_ip}:8069 > ip_ec2.txt"
}
}
```

```

variable "ami" {
    default = "ami-04505e74c0741db8d"
    type    = string
}

variable "instance_type" {
    default = "t3.medium"
}

variable "key_name" {
    default = "capge_projet_kp"
}

variable "sg_group" {
    type = list(string)
    default = ["security_group_capge"]
}

variable "username" {
    default = "ubuntu"
}

variable "env" {
    default = "dev"
}

variable "serveur" {
    default = "AnsibleMaster"
}

variable "private_key_path" {
    default = "D:/Formation/AJC/05.DevOps/PROJET/capge_projet_kp.pem"
}

variable "ec2_server_ip" {
    default = "0.0.0.0"
}

variable "ec2_odoo_ip" {
    default = "0.0.0.0"
}

variable "pgadmin_dns" {
    default = "pgadmin-dns.com"
}

variable "odoo_dns" {
    default = "odoo-dns.com"
}

variable "pgadmin_port" {
    default = 5050
}

variable "odoo_port" {
    default = 8069
}

variable "ic-webapp_image" {
    default = "lianhuahayu/ic-webapp:1.0"
}

variable "odoo_image" {
    default = "odoo:13.0"
}

variable "postgres_image" {
    default = "postgres:10"
}

```





## Les credentials

Pour des raisons de sécurité, le fichier `.aws/credential` est simplement un template.

Dans notre JenkinsFile, nous allons remplacer donc remplacer chaîne de caractères suivantes par les valeurs de nos credentials, stockés de façon sécurisé dans Jenkins :

- YOUR\_KEY\_ID
- YOUR\_ACCESS\_KEY

```
[default]
aws_access_key_id = "YOUR_KEY_ID"
aws_secret_access_key = "YOUR_ACCESS_KEY"
```

## Le remote backend S3

```
terraform {
  backend "s3" {
    bucket      = "capge-bucket-projet"
    key         = "ic-webapp_project.tfstate"
    region      = "us-east-1"
    shared_credentials_file = "../.aws/credentials"
  }
}
```

## Le provider

```
provider "aws" {
  region      = "us-east-1"
  shared_credentials_file = "../.aws/credentials"
}
```

## Le manifest principal (main)

```

module "deploy_sg" {
  source = "../modules/sg"
  sg_name = "capge-sg-dev"
}

module "deploy_ec2_server"{
  source = "../modules/ec2_worker"
  serveur = "admin"
  key_name = var.key_name
  sg_group = [ module.deploy_sg.sg_name]
}

module "deploy_ec2_odoo"{
  source = "../modules/ec2_worker"
  serveur = "odoo"
  key_name = var.key_name
  sg_group = [ module.deploy_sg.sg_name]
}

module "deploy_ec2_master" {
  depends_on = [
    module.deploy_ec2_server,
    module.deploy_ec2_odoo
  ]
  source = "../modules/ec2_master"
  serveur = "AnsibleMaster"
  key_name = var.key_name
  private_key_path = var.key_path
  sg_group = [ module.deploy_sg.sg_name]
  ec2_server_ip = module.deploy_ec2_server.ec2_ip
  pgadmin_dns = module.deploy_ec2_server.ec2_dns
  pgadmin_port = var.pgadmin_port
  ec2_odoo_ip = module.deploy_ec2_odoo.ec2_ip
  odoo_dns = module.deploy_ec2_odoo.ec2_dns
  odoo_port = var.odoo_port
  ic-webapp_image = var.ic-webapp_image
  odoo_image = var.odoo_image
  postgres_image = var.postgres_image
}

```

## Les variables à surcharger

- Obligatoires
  - key\_path : chemin ou se trouve la clé privé que va utiliser ec2 MASTER pour exectuer les script en ssh
  - key\_name : le nom de la clé privé du coté de AWS
  - ic-webapp\_image : le nom de l'image ic-webapp sur dockerhub
  - odoo\_image : nom de l'image odoo sur dockerhub
  - postgres\_image : nom de l'image postgres sur dockerhub
- Optionnelles
  - odoo\_port : port de l'interface web de odoo
  - pgadmin\_port : port de l'interface web de pgAdmin

```
variable "key_path" {
  default = "D:/Formation/AJC/05.DevOps/PROJET/capge_projet_kp.pem"
}

variable "key_name" {
  default = "capge_projet_kp"
}

variable "odoo_port" {
  default = 8069
}

variable "pgadmin_port" {
  default = 5050
}

variable "ic-webapp_image" {
  default = "lianhuahayu/ic-webapp:1.0"
}

variable "odoo_image" {
  default = "odoo:13.0"
}

variable "postgres_image" {
  default = "postgres:10"
}
```



## Exemple

```
terraform init
terraform plan
terraform apply -var='key_path=~/.ssh/key.pem' \
  -var='ic-webapp_image=lianhuahayu/ic-webapp:1.0' \
  -var='odoo_image=odoo:13.0' \
  -var='postgres_image=postgres:10'
```

## 4.4 Test

### Jenkinsfile

Vous pourrez retrouver le Jenkinsfile sur

<https://github.com/Thuy9906/aic-projet-capge.git>

Le Jenkinsfile sera la source de notre pipeline, Jenkins Pipeline est une combinaison de tâches permettant de fournir des logiciels en continu en utilisant Jenkins.

Un pipeline Jenkins se compose de plusieurs états ou étapes, et ils sont exécutés dans une séquence l'un après l'autre. JenkinsFile est un simple fichier texte en Go utilisé pour créer un pipeline sous forme de code dans Jenkins. Il contient du code en langage spécifique au domaine Groovy, qui est simple à écrire et lisible par l'homme plus ou moins.

Soit vous pouvez exécuter JenkinsFile séparément, soit vous pouvez également exécuter le code de pipeline à partir de l'interface utilisateur Web Jenkins. Il existe deux façons de créer un pipeline à l'aide de Jenkins. Nous on a configuré un webhook avec notre repo github pour qu'à chaque push sur le master du Jenkinsfile, le pipeline puisse se lancer.

Voici un aperçu de notre pipeline :



Au début de notre pipeline nous retrouverons les variables globales, ces variables vont être utilisées tout le long du fichier à plusieurs reprises. Voici la liste des variables globales :

### Définition des variables d'environnement

- Le de notre image : `IMAGE_NAME = "ic-webapp"`
- Le tag de notre image : `IMAGE_TAG = "1.0"` (cette variable prendra une autre dimension plus tard dans le projet, car elle sera dynamique)
- Le username pour la connexion à dockerhub : `USERNAME = "lianhuahayu"`
- Le nom de notre container : `CONTAINER_NAME = "test-ic-webapp"`
- Notre clé d'accès à AWS : `AWS_ACCESS_KEY_ID = credentials('AWS_ACCESS_KEY_ID')`
- Notre secret d'accès à AWS : `AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCESS_KEY')`
- L'adresse IP de notre instance de production : `EC2_PROD = "ec2-54-235-230-173.compute-1.amazonaws.com"`

### Build image ic-webapp

Dans la partie build de l'image, nous allons nous assurer qu'aucun container avec notre nom de container test existe, sinon on le supprime et on s'assure de supprimer l'image existante si jamais elle existe. En effet il faut fermer un container pour le supprimer, et supprimer un container pour supprimer l'image, on parle ici d'une dépendance entre les différentes ressources de docker. Nous avons prévu le cas où ces étapes ne seront pas nécessaire car d'une part l'image n'existe pas ou le container n'existe pas, pour éviter les erreurs on ajoute `|| true` à la fin.

Ensuite nous allons build notre image

### Scan avec SNYK de l'image

On retrouve ensuite la partie Snyk, qui va scanner notre image via la commande "docker scan" et ensuite générer un rapport en json qui sera converti en html.

Nous n'avons pas décidé de desservir ce rapport vers un stockage distant, mais ça pourrait être fait par la suite. Il se trouvera dans le workspace Jenkins du build en cours. Pour avoir plus de visibilité sur la sécurité de notre image,

Nous affichons tout de même le message du scan qui indique si notre image docker est sécurisée ou pas.

### Test du container test-ic-webapp

Dans cette partie nous allons s'assurer comme dans le build que notre container n'existe pas et ensuite créer notre container avec une commande **docker run**. Puis simplement tester avec un curl que notre vitrine répond bien avec une réponse ayant un code **200**. C'est ce que nous faisons dans cette ligne :

```
if [[ "`head -n1 <(curl -iq http://localhost:8090)`" == "**200**" ]];then echo "PASS"; else false;
fi
```

Si jamais le test est faux alors le pipeline s'arrête, car le script retourne false.

## Push vers un registre publique

Une fois que le test de notre image est OK, on peut la push sur notre dockerhub pour l'image soit accessible par nos différentes ressources plus tard (notre serveur de production, et nos instances de test etc..) Pour pouvoir push, il faut d'abord se logger

Ici je ne supprime pas encore mon image, car je n'ai push que la version avec le tag en cours, nous reverrons plus tard une autre étape dans la partie production.

## Déploiement automatique de l'env-test via terraform

Maintenant parlons de choses sérieuses, le déploiement de l'environnement test avec Terraform.

Dans cette partie nous allons créer un répertoire terraform\_env\_test dans le workspace de notre projet sur Jenkins puis cloner les fichiers nécessaire pour de déploiement se trouvant à l'adresse <https://github.com/omarpiotr/terraform-ic-webapp.git>.

Ensuite on se déplace dans ce répertoire pour :

- copier la clé privée de notre pair de clé ssh que nous avons sur Jenkins dans ce répertoire car nous aurons besoin de cette clé privée pour déployer les différents rôles ansible.
- Changer les clés d'accès AWS se trouvant dans le fichier anonymiser .aws/credentials par nos clés à nous (il ne faut pas laisser ces clés trainés sur github)
- Ensuite on se déplace dans le répertoire principal pour exécuter nos commandes terraform init, plan et apply. Dans ce répertoire se trouve le manifeste qui appellera nos différents modules. Dans la commande apply nous utilisons notre clé et avons également variables l'image que l'on souhaite déployer (une petite gymnastique avec la création d'une variable IMAGE a été faite pour qu'elle soit prise en compte dans la commande de terraform)
- Le déploiement de l'environnement test peut prendre plus de temps que le reste car il se fait d'une certaine façon abordé dans la partie des roles ansibles et terraform.

## Test de env-test

La suite du pipeline débouche sur le test de l'environnement, dans cette partie nous allons à partir du fichier ip\_ec2.txt crée précédemment par terraform, récupérer les IP de

nos instances Admin et Odoo puis exécuter un curl sur dessus pour savoir si elle répondent correctement.

## Déploiement de l'environnement de prod

Si l'environnement de test a été validé, nous pouvons passer au déploiement de notre image dans l'environnement de production. Elle se scinde en deux parties.

La première partie est la destruction de l'environnement de test car il n'est plus nécessaire, il suffit de retourner dans le répertoire créer pour les manifestes terraform vu précédemment et faire une commande **terraform destroy** pour supprimer nos instances créer.

On push également notre image avec un tag latest sur notre dockerhub pour faciliter la récupération sur k8s de cette dernière.

Puis on nettoie notre environnement en supprimant les images avec le tag en cours et le latest.

La deuxième étape consiste à se connecter sur notre serveur de production et à cloner le repo github contenant tous les manifestes k8s nécessaire pour déployer nos ressources k8s. Une fois que c'est OK, il suffit d'appliquer tous les manifestes via un script qui a été créé à cet effet.

## Test de l'environnement prod

Nous allons dans cette partie effectuer un curl sur les différents liens qui sont censé être accessible.

Une fois tous les stages effectués sans erreur, notre pipeline est fini. Et notre github afficher un "passed" grâce au plugin embedded status.

## TEST

### Test du container

```
#!/bin/bash
docker stop $CONTAINER_NAME || true
```

```

docker rm $CONTAINER_NAME || true
docker run -d --name $CONTAINER_NAME -p8099:8080
$USERNAME/$IMAGE_NAME:$IMAGE_TAG
if [[ "`head -n1 <(curl -iq http://localhost)`" == *"200"* ]];then echo "PASS"; else false; fi
docker stop $CONTAINER_NAME
docker rm $CONTAINER_NAME

```

## Test sur la prod :

```

#!/bin/bash

#Test de la vitrine prod
#Page accessible directement 200
if [[ "`head -n1 <(curl -iq ec2-54-235-sdfsdf.compute-1.amazonaws.com)`" == *"200"*
]];then echo "PASS"; else false; fi

#Test de l'accès à Odoo
#Redirection de la page vers la bonne code 303
#Le code de statut de réponse de redirection 303 See Other, généralement renvoyé
comme résultat d'une opération PUT ou POST, indique que la redirection ne fait pas le
lien vers la ressource nouvellement téléversé mais vers une autre page (par exemple
une page de confirmation ou qui affiche l'avancement du téléversement). La méthode
utilisée pour afficher la page redirigée est toujours GET.
if [[ "`head -n1 <(curl -iq ec2-54-235-sdfsdf.compute-1.amazonaws.com:32020)`" ==
*"302"* ]];then echo "PASS"; else false; fi

#Verification de la présence du lien odoo sur la vitrine ic-webapp
if [[ "`curl -iq http://ec2-54-235-230-173.compute-1.amazonaws.com`" ==
*"http://ec2-54-235-230-173.compute-1.amazonaws.com:32020"* ]];then echo "PASS";
else false"; fi

#Verification de la présence du lien pgadmin sur la vitrine ic-webapp
if [[ "`curl -iq http://ec2-54-235-230-173.compute-1.amazonaws.com`" == *"32020"* ]];then
echo "PASS"; else false; fi

#Test de l'accès à pgAdmin
#Redirection de la page vers la bonne code 302
#Le code de statut de réponse de redirection 302 Found indique que la ressource est
temporairement déplacée vers l'URL contenue dans l'en-tête Location .

if [[ "`head -n1 <(curl -iq ec2-54-235-sdfsdf.compute-1.amazonaws.com:32125)`" == *"303"*
]];then echo "PASS"; else false"; fi

```





## 5 Conclusion

La philosophie DevOps et les nouvelles connaissances nous ont permis de parvenir à réaliser ce projet. Globalement nous avons utilisé les outils et logiciels qui permettent de réaliser l'ensemble des étapes du projet.

La communication et l'organisation nous ont permis de réaliser dans un enchaînement optimal les étapes de ce projet.

