INGENIEUR DEVOPS

08/2021 - 11/2021

Projet Final

CICD - Application en Node.js

Groupe 1 - Auteurs :

Marceline **AUGER**
Pierre-Gilles **LEMASLE**
Amandine **SIMO**

18 Novembre 2021

Table des matières

1 Introduction

Ce document est le résultat de notre formation Devops sur trois mois.

Son objectif est de démontrer, au travers de notre formation Devops, de nos expériences professionnelles et personnelles, que nous avons acquis, développé et confirmé les compétences indispensables pour exercer le métier de consultant/ ingénieur Devops.

Pour cela, nous avons réalisé un projet de fin de formation. Le but de ce projet est de vérifier notre capacité à automatiser la construction et le déploiement d'applications conteneurisées. Il nous a été donné comme consigne de créer notre pipeline CI/CD en utilisant tous les outils recommandés, à savoir : jenkins, kubernetes (ou minikube), terraform (ou ansible). Notre pipeline doit également mettre en œuvre le modèle suivant :

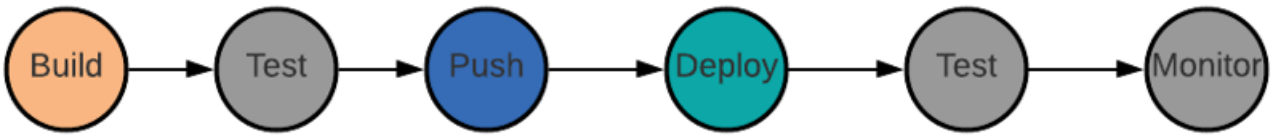

FIGURE 1 – Etapes de la pipeline CI/CD

Dans un premier temps , nous vous présenterons notre infrastructure. Ensuite, nous présenterons notre architecture. Dans une troisième partie, nous exposerons notre pipeline CI/CD.

2 Infrastructure

La réalisation de ce projet nécessite la mise en place d'une infrastructure. Nous avons opté pour une infrastructure virtuelle. Celle-ci nous permet d'avoir une version virtuelle de ressources matérielles qui existent pour les ordinateurs tels ques : des serveur ou des espaces de stockage.

Dans cette section, nous décrivons le role des environnements qui regroupent les différentes ressources que nous avons utilisé et les étapes de création de ces dernières.

2.1 Le role des environnements

Un environnement en informatique, correspond à l'ensemble des matériels et logiciels système, dont le système d'exploitation, sur lesquels les programmes d'une application sont exécutés.

Notre infrastructure virtuelle est constituée de deux environnements de travail : l'environnement de production et l'environnement de test. Ils contiennent chacun les ressources et configurations nécessaires pour la réalisation de nos différentes taches.

2.1.1 Environnement de production

L'environnement de production nous permet d'effectuer les étapes pour la réalisation du projet et de rendre notre application disponible pour les clients. Il est constitué de trois machines virtuelles approvisionnées et de systèmes d'exploitations centos7 :

- la machine **server** est la machine sur laquelle nous réalisons toutes les taches du projet ; nous y avons installé python3, python3-pip, git, docker, docker-compose, jenkins, ansible, des dépendances ainsi que des librairies nécessaires pour le fonctionnement de nos outils.
- les machines clientes **client1** et **client2**, interviennent à la fin de la chaine de notre pipeline CI/CD. En effet, c'est sur ces machines que notre application sera deployée et pourra être utilissée par les clients. Sur ces machines, nous avons installés python3, python3-pip, git, docker, docker-compose et des dépendances nécessaires pour le fonctionnement de nos outils.

La description de l'approvisionnement de ces machines virtuelles est disponible dans le fichier d'approvisionement **"install_ansible.sh"**.

2.1.2 Environnement de test

Cet environnement est constitué d'une machine virtuelle appelée **"testing"**, de système d'exploitation centos 7 et est non approvisionnée. Il nous permet de tester le bon fonctionnement des taches pour pouvoir les mettre en environnement de production. Ainsi, à chaque étape, nous procédons comme suit :

- nous réalisons une tache sur la machine **"testing"**,
- nous vérifions son bon fonctionnement, toujours sur la machine "testing".

Entre autres, l'environnement de test nous permet :

- l'installation des outils comme docker, jenkins ou ansible,
- l'installation des packages/librairies à utiliser et leurs dépendances,
- tester le fonctionnement de ces outils et packages,
- tester le fonctionnement du code utilisé pour la réalisation de le pipeline CI/CD,
- etc.

Ayant à présent une idée du role des environnements de production et de test, nous pouvons passer à notre partie sur la création des ressources.

2.2 La création des ressources

Pour créer et configurer nos ressources, nous utilisons l'outil de virtualisation **Vagrant** (version 2.2.18) par lignes de commandes et le logiciel de virtualisation **VirtualBox** (version 6.1.26). Après avoir installé ces outils(**Vagrant** et **VirtualBox**) sur nos ordinateurs, nous créons deux **Vagrantfile** (pour les 2 environnements respectifs) et un fichier d'approvisionement en **Shell** ("install_ansible.sh") décrivant les configurations et installations à effectuer sur nos machines virtuelles. Nous décrirons ensuite, la création des ressources par environnement.

Une fois les ressources crées nous avons une image globale de ces resources sur **VirtualBox** comme le montre l'image ci-dessous.

FIGURE 2 – infrastructure sur **VirtualBox**

2.2.1 Environnement de production

Nous créons un dossier Projetfinal. Dans celui-ci nous créons un dossier nommé **"Prod"**. Nous y plaçons deux fichiers : le **Vagrantfile** décrivant les trois machines virtuelles et le fichier **"install_ansible.sh"** contenant les informations de configuration de ces machines.

Pour créer les machines virtuelles, nous nous positionnons dans le dossier **"Prod"** à partir de notre terminal et nous lançons la commande : *"vagrant up"*

2.2.2 Environnement de test

Dans le dossier Projetfinal, nous créons un dossier nommé **"Testing"**. Nous y plaçons le fichier **Vagrantfile** qui décrit la machine virtuelle de tests qui est non approvisionnée.

Pour créer cette machines virtuelle, nous nous positionnons dans le dossier **"Testing"** à partir de notre terminal et nous lancons la commande : *vagrant up*

3 Architecture DevOps

Dans cette partie, nous allons décrire les différents outils que nous avons utilisé dans notre architecture.

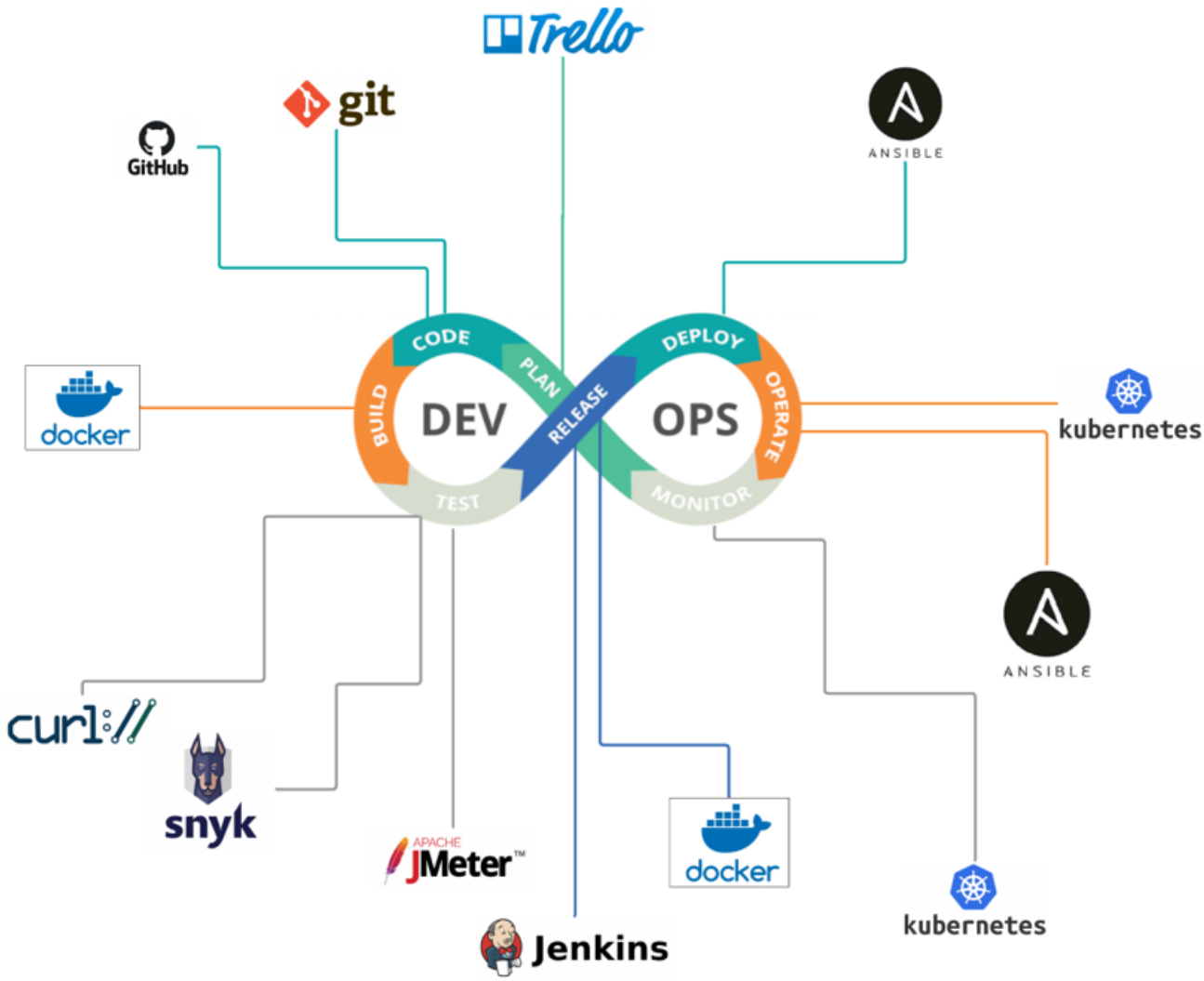

FIGURE 3 – Architecture devops

3.1 Trello

Trello est un outil en ligne de gestion de projet. Il repose sur une organisation de projet en planche listant des cartes dont chacunes représente une tache. Il nous permet de suivre les tâches à effectuer pour notre projet.

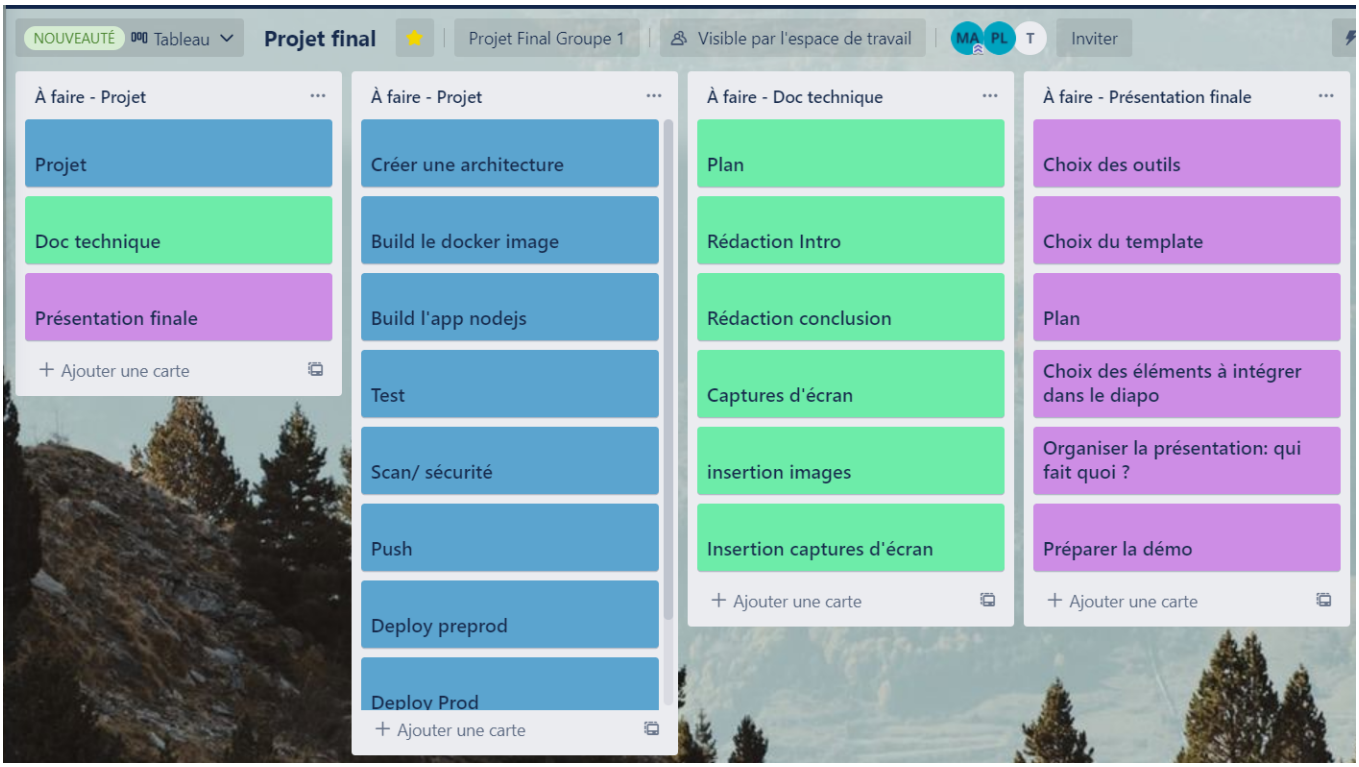

FIGURE 4 – Description de planification sur trello

3.2 git

Git est un logiciel de gestion des versions décentralisé. Il nous permet de pousser notre code sur notre dossier Github.

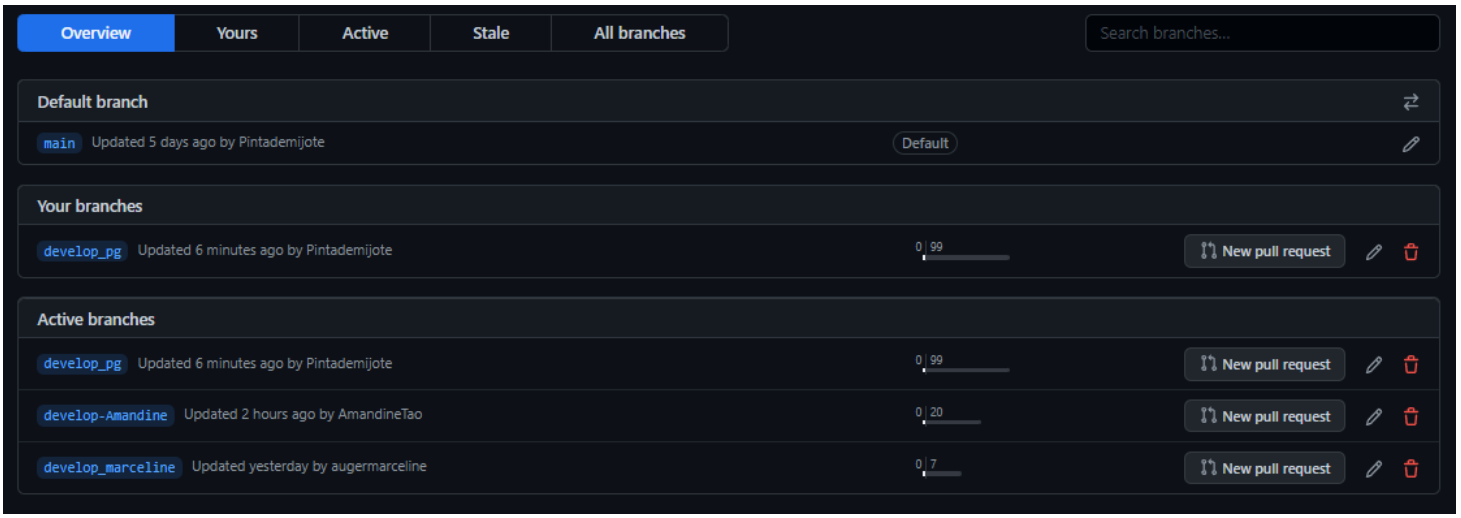

FIGURE 5 – Description branches sur git

3.3 github

GitHub est un service web d'hébergement et de gestion des développements. Il utilise le logiciel de gestion des versions Git. Après avoir forké le projet initial sur l'un de nos comptes privés, nous avons utilisé ce dépôt comme base à notre travail de groupe. Nous avons créé chacun notre branche de developpement qui est basée sur la version initiale de la branche main. Le suivi des mises à jour dans nos codes s'est fait à l'aide des fonctions "merge" et "rebase". Un merge sur la branche master a finallement été effectué par une séance de code reviewing. Ce dépôt se situe à cette adresse : https ://github.com/Pintademijote/projet-groupe-1.

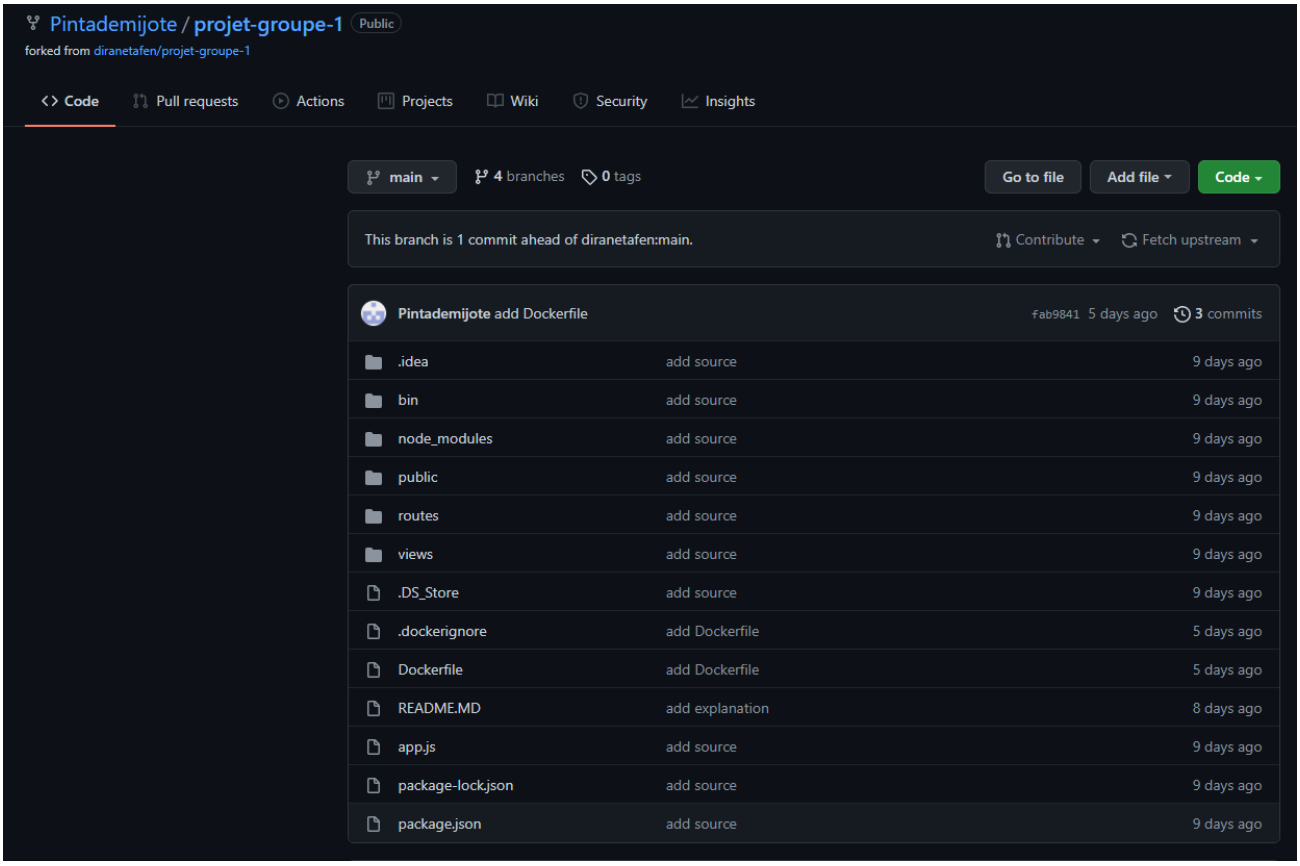

FIGURE 6 – Description du remote distant sur github

3.4 docker

Docker est une plateforme permettant de lancer des applications dans des conteneurs logiciels. Il peut contenir une application et ses dépendances et pourra être exécuté sur n'importe quel serveur. Nous utilisons Docker pour contenir nos applications jenkins, kubernetes et node.js du projet.

3.5 jenkins

Jenkins est un outil type serveur permettant la gestion de l'automatisation de pipeline CI/CD en se basant sur différentes tâches définies. Notre installation se base sur une image customisé de Jenkins (*i.e.* dirane/jenkins) lancé depuis un *docker compose* lors du provisionning de la machine "server". Nous avons ensuite installé les plugins Ansible, snyk, ansible ainsi que le plugins performance sur le serveur Jenkins pour accomplir les différentes tâches de notre pipeline. L'accès à notre serveur

Jenkins a été configuré derrière un reverse proxy. Ce qui nous permet de déclarer un webhook dans notre dépôt Github vers le serveur pour pouvoir lancer notre pipeline automatiquement. Enfin notre projet Jenkins est configuré comme une pipeline basée sur notre dépôt Github, avec à sa base une Jenkinsfile où est indiqué les différentes tâches.

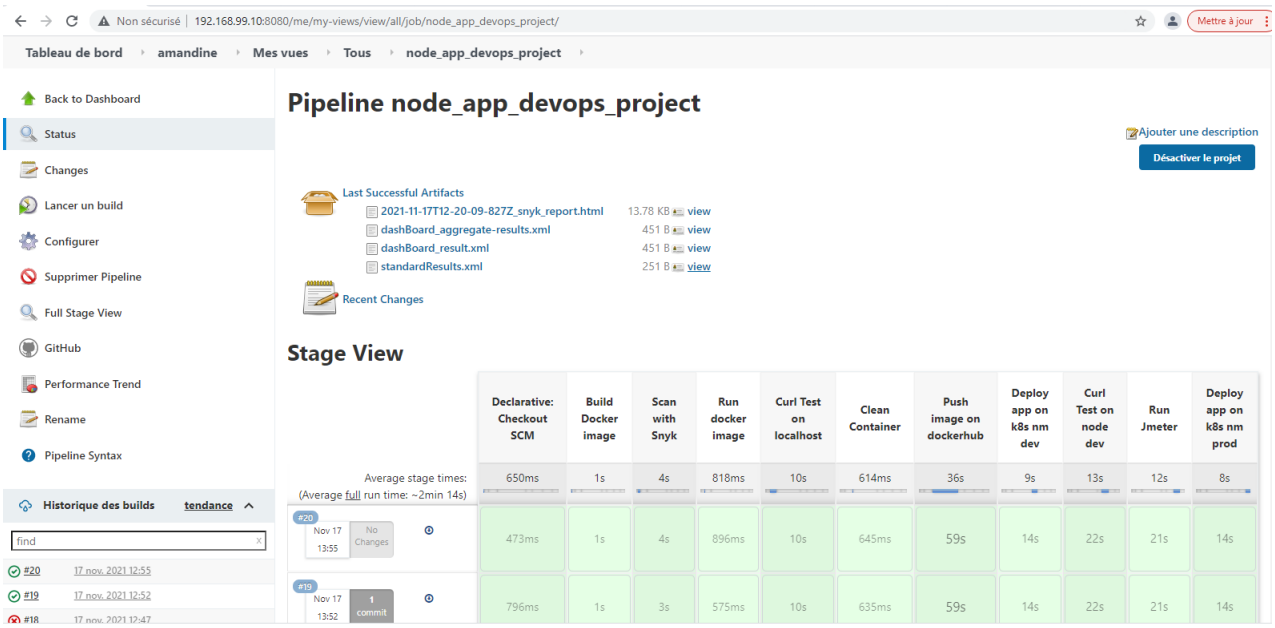

FIGURE 7 – Page Jenkins du projet

3.6 ansible

Ansible est un logiciel développé par Red Hat permettant l'automatisation de la configuration dans un état de machine distante et le déploiement. Son installation a été réalisé sur notre machine "serveur" lors du provisionning et est dépendant de python pour executer les instructions sur les machines clientes. Nous avons également installé python3 et pip3 sur les machines "client" lors du provisionning.

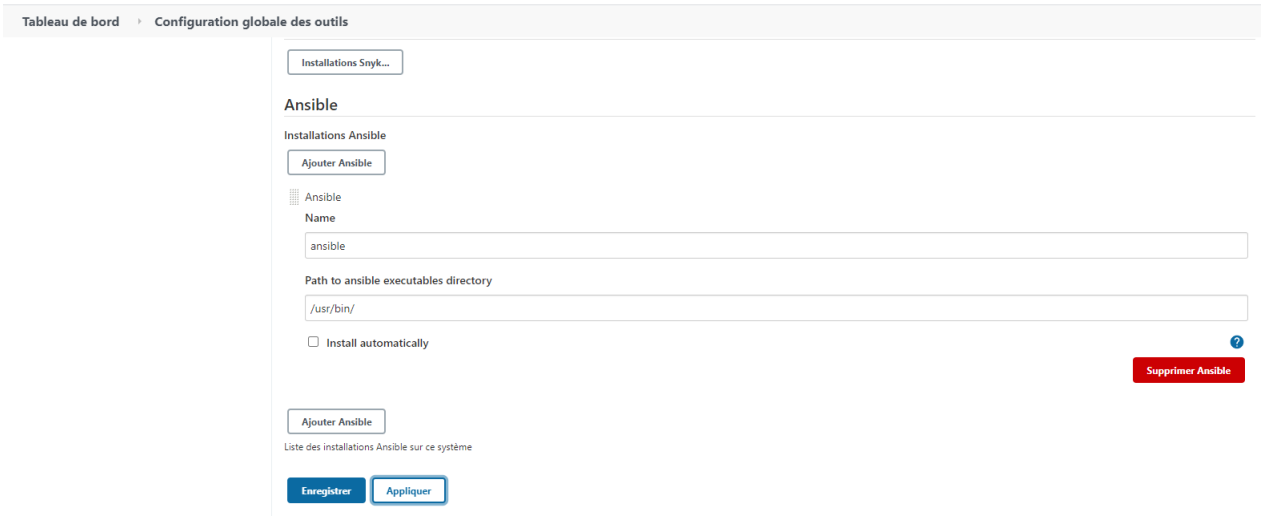

FIGURE 8 – Installation du plugin ansible sur jenkins

3.7 kubernetes

Kubernetes est un orchestrateur de container docker à l'origine créé par Google. Il nous permet une gestion simplifiée des déploiements d'application conteneurisées, leurs mises en réseaux, le scaling suivant la demande, ou du processus de mise à jour. Pour l'installation de notre clusters nous avons choisi d'utiliser l'outil de déploiement kubespray publié par Kubernetes, qui nous permet via Ansible un déploiement facilité. Notre cluster est configuré en sorte que la machine "serveur" : node1 a pour role "master" et les deux machines clientes : node2 et node 3 ayant pour role "worker".

```
NAME    STATUS   ROLES                  AGE    VERSION
node1   Ready    control-plane,master   2d4h   v1.22.3
node2   Ready    worker                 2d4h   v1.22.3
node3   Ready    worker                 2d4h   v1.22.3
```

FIGURE 9 – Liste des noeuds du cluster kubernetes

4 Pipeline CI/CD

Un pipeline CI/CD est une suite d'instructions permettant d'effectuer le processus d'automatisation d'un projet. Dans ce projet, nous réalisons un pipeline Jenkins qui est constitué de plusieurs

étapes ; Ces étapes sont executées de façon séquentielle l'une après l'autre. Un JenkinsFile est un fichier texte utilisé pour créer un pipeline sous forme de code dans Jenkins.

4.1 CI

L'integration continue (CI) est un processus d'automatisation qui permet d'apporter régulièrement des modifications au code d'applications et à les tester.

4.1.1 Build

Notre build est réalisé avec l'outil **docker**. Cette étape consiste à construire une image de notre application à partir d'une image initialement réalisée via le **Dockerfile** et poussée sur le dépot **docker hub** qui est sur le cloud. Pour créer l'image obtenue à partir du **Dockerfile**, nous procédons comme suit :

- se connecter à la machine virtuelle **testing**,
- cloner le dossier distant du projet **projet-groupe-1** qui contient les dossiers et fichiers utiles pour la création de l'application
- se deplacer dans ce dossier et y créer un fichier nommé **Dockerfile**. Voir ci-dessous.

Dockerfile

```
# define from what image we want to build from
FROM node:14.16

# specifies the port in which an application is listening
ENV PORT=8000

# specifies the environment in which an application is running, here "development" environment
ENV NODE_ENV=development

# we create a directory to hold the application code inside the image
WORKDIR /app

## install app dependencies using npm
# copy package.json AND package-lock.json in the created  /app directory
COPY package*.json ./

# install npm
RUN npm install

# bundle app source: copy the directory /projet-groupe-1(except the files/folder of .dockerignore) in  /app directory;
COPY . .

# informs Docker that the container listens on this port at runtime
EXPOSE 8000

#define the command to run the app: here we use node server.js to start our server
CMD [ "npm", "start" ]
```

FIGURE 10 – Description du Dockerfile

— Toujours, dans le dossier du projet, nous exécutons la commande suivante pour créer l'image : *docker build -t docker$_i$d/node$_a$pp$_d$evops*.

Une fois l'image construite, nous créons et exécutons un conteneur à partir cette image.

4.1.2 Test

Cette étape nous permet de tester si notre application fonctionne correctement. Nous avons réalisé deux tests :

- **curl test** : il permet de tester si le conteneur docker lancé à partir de l'image docker de notre application est en vérifiant si le status http associé à l'adresse ip et au port donné au nodeport renvoie bien le code 200.
- **scan test** : il permet de lancer des scans de vulnérabilté. En effet, c'est un test de sécurité utilisé pour assurer la qualité de ce qui sera déployer.

Une fois le fonctionnement du conteneur testé, nous passons à une phase de nettoyage qui consiste à arreter et supprimer le conteneur créé.

4.2 CD

4.2.1 Push

Cette étape nous permet de pousser sur le dépot **docker hub**.

4.2.2 Deploy

Notre déploiement est effectué avec Ansible et l'utilisation du module kubernetes.core. Nous avons créé un rôle pour Ansible qui nous permet de :

- créer un dossier
- copier dans ce dossier les templates du déploiement de l'application nodeJs et d'un nodeport
- créer un namespace kubernetes dans lequel nous pouvons lancer les déploiements
- lancer les déploiements kubernetes

Ce rôle inclus des variables et nous permet la réutilisation du rôle pour le déploiement de l'application et ses réplicas dans un environnement de développement et dans un environnement de production. Le déploiement dans l'environnement de développpement se fait dans un premier temps pour y effectuer des tests et ensuite le déploiement de l'environnement de production.

FIGURE 11 – Image de l'application deployée en dev

FIGURE 12 – Image de l'application deployée en Prod

4.2.3 Test

Comme lors de la phase de test dans la partie CI, le premier test consiste à vérifier si le status http associé à l'ip et au port donné au nodeport renvoie bien le code 200. Ensuite, nous lançons toujours sur cette adressse un test de charge avec l'aide de plugin performance de Jenkins et un script Jmeter pour vérifier l'accessibilité de nos machines pour 100 requêtes http sur une période de 10sec.

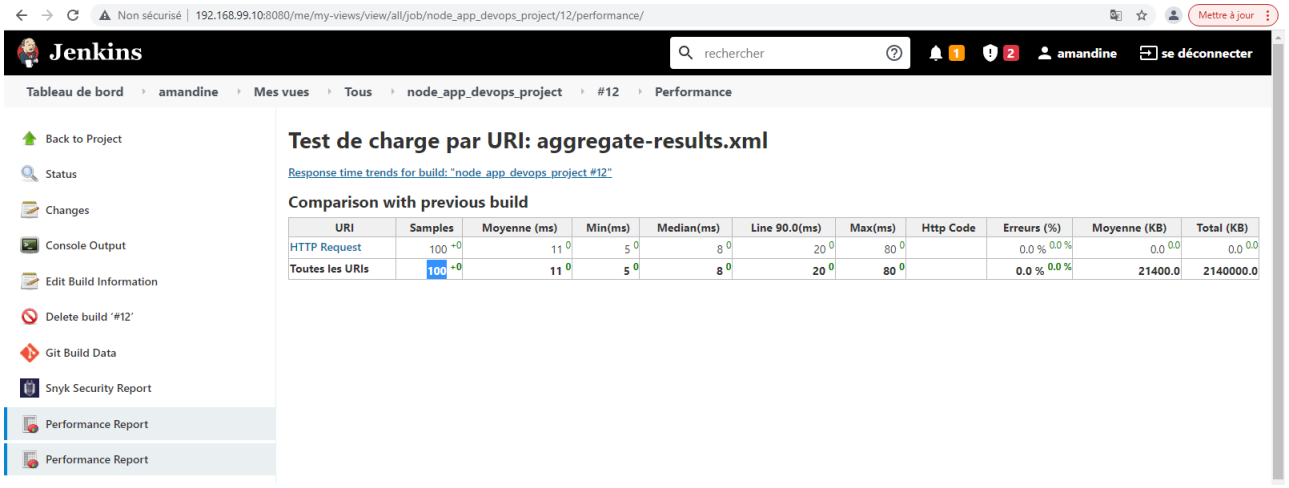

FIGURE 13 – Tableau de performances - jmeter

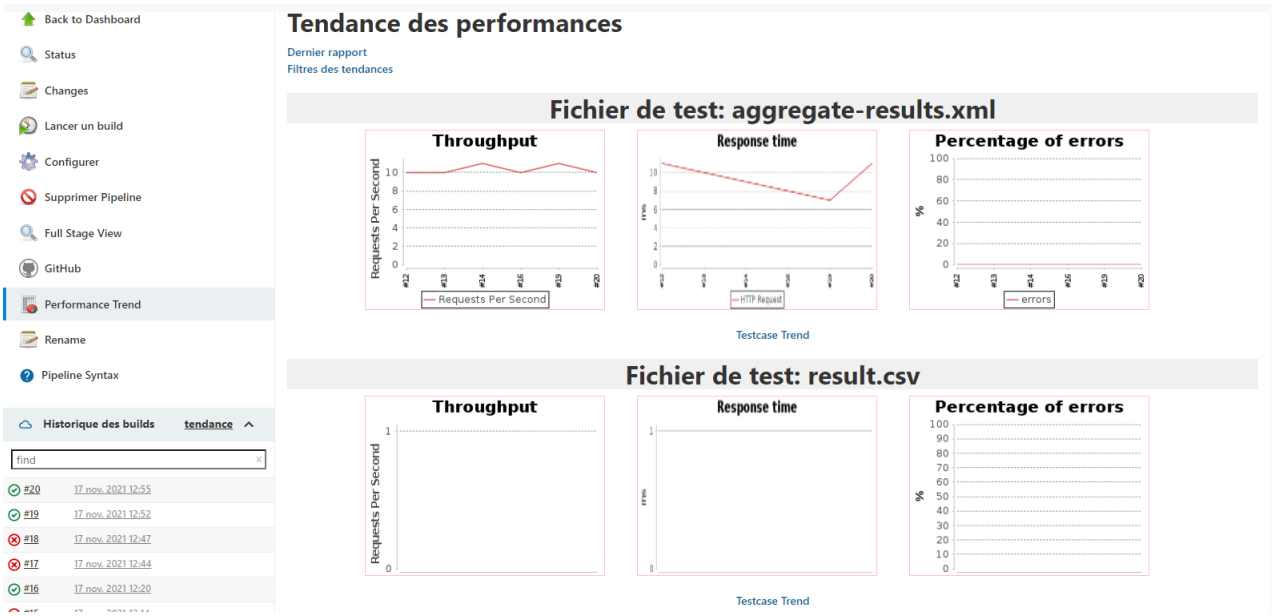

FIGURE 14 – Tendances de performances - jmeter

4.2.4 Monitor

Pour suivre l'utilisation de nos ressources et du déploiement nous avons créé un déploiement de kubernetes dashboard.

4.2.5 Publication du site web

La publication du site web s'est basée sur une infrastructure déjà existante et donc n'a pas été intégrée à la chaîne CD. L'infrastructure se base sur un container hébergeant "let's encrypt" pour signer le certificat tls et enregistrer l'ip dynamique sur le domaine pintade.duckdns.org. Ensuite un deuxième containeur basé sur image caddy server pour être utilisé en tant que reverse proxy vers l'ip de la machine windows où sont contenus les vm hébergeant l'application nodeJs. La redirection de port windows étant configurée pour pointer vers l'ip de l'une des vm client avec le nodeport associé à la production.

L'adresse du site est : https ://devops.pintade.duckdns.org/

5 Conclusion

L'objectif du projet est atteint, l'application est en ligne et suit à la fois une chaîne CI et CD. L'application est buildée, testée et push, l'ajout de tests est facilité notamment grâce à Jenkins et le workflow en pipeline. Grâce à Kubernetes, l'application peut facilement être déployée, être scalée et mise à jour, en suivant au mieux les principes de hautes disponibilités. De même l'orchestration du déployement et de la configuration des machines avec Ansible permet une flexibilité de déploiement sur d'autres machines.

5.1 Axes d'améliorations

Github

Nous avons travaillé chacuns sur nos branches développement sur le dêpot github, nous avons du lors de la merge modifier les credentials en fonction de nos environnements. Une solution serait par exemple de passer par l'export de variables environnementales pour simplifier la merge.

Jenkinsfile

Notre Jenkinsfile est pour le moment assez simple dans son execution et consiste à une succession d'étapes. Idealement, il faudrait que nous ajoutions par exemple des tâches en "post" et mieux ajouter un "failure" ou des "try" pour gérer les problématiques d'erreurs potentielles.

Jmeter template

Notre Jenkinsfile est suffisament variabilisé pour lancer facilement un déploiement sur un nouveau namespace mais le test Jmeter lui se lancera toujours sur une ip et un port prédéfinis. Pour y remedier il faudrait créer un template de ce script et utiliser des fonctions propres à groovy pour appliquer ce template avec des variables d'environnement.

Kubernetes Dashboard

La solution implémentée à l'heure actuelle pour l'authentification lors de la connection à Kubernetes Dashboard est encore provisoire, s'apppuyant sur une solution par token. Une étude plus approfondie de la documentation Kubernetes en rapport à l'authentification sera un ajout à apporter.

Accès Ansible

Nous profitons actuellement de la facilité de déploiement de nos environnements avec Vagrant pour se connecter en ssh avec un couple classique utilisateur-mot de passe. La solution pourrait être plus sécurisée et s'appuyer sur un couple clé privé-publique serait de loin une solution beaucoup plus élégante.