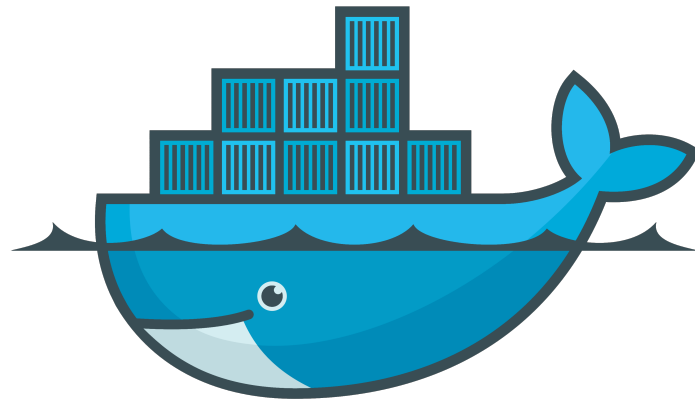
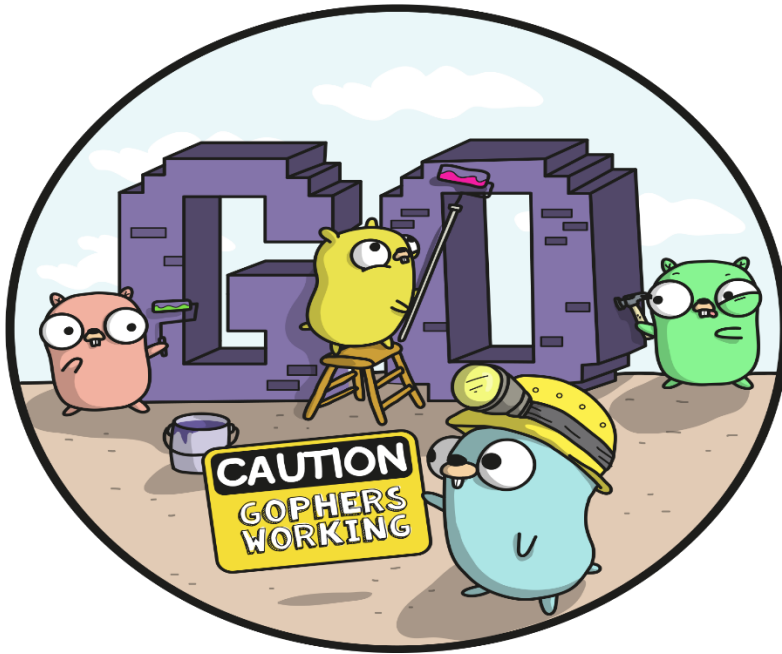


Projet BLOG (partie GitLab) :



docker

Tables des matières

Mise en place d'une VM ubuntu qui vas héberger notre usine logicielle	3
Mise en place de Docker	3
Mise en place de portainer IO	4
Mise ne place de GitLab CI	5
Mise en place d'un gitlab-runner	6
Création du pipeline	8
Création des Schedules	9
Exposition des difficultés rencontrées durant le projet :	10

Mise en place d'une VM ubuntu qui vas héberger notre usine logicielle

Nous avons choisi Ubuntu pour une question de simplicité d'environnement

Cela nous permet d'avoir un contrôle total sur notre environnement de déploiement.

Mise en place de Docker

Nous avons choisi Docker pour héberger le déploiement de notre usine logiciel car fournissant une solution pratique de virtualisation qui exécute les processus de façon isolée.

La technologie de conteneur de Docker nous permet d'exécutent de manière autonome depuis Gitlab. Cela permet aux nœuds d'être déployés au fur et à mesure que les ressources sont disponibles, offrant un déploiement transparent.

Pour l'installation de docker on entre les commandes suivantes

```
$> sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

```
$> curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
$> sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

```
$> sudo apt update
```

```
$> sudo apt install docker-ce
```

Notre docker est prêt a recevoir nos premier container

Mise en place de portainer IO

Portainer est un outil qui nous permet la gestion de nos applications conteneurisées.

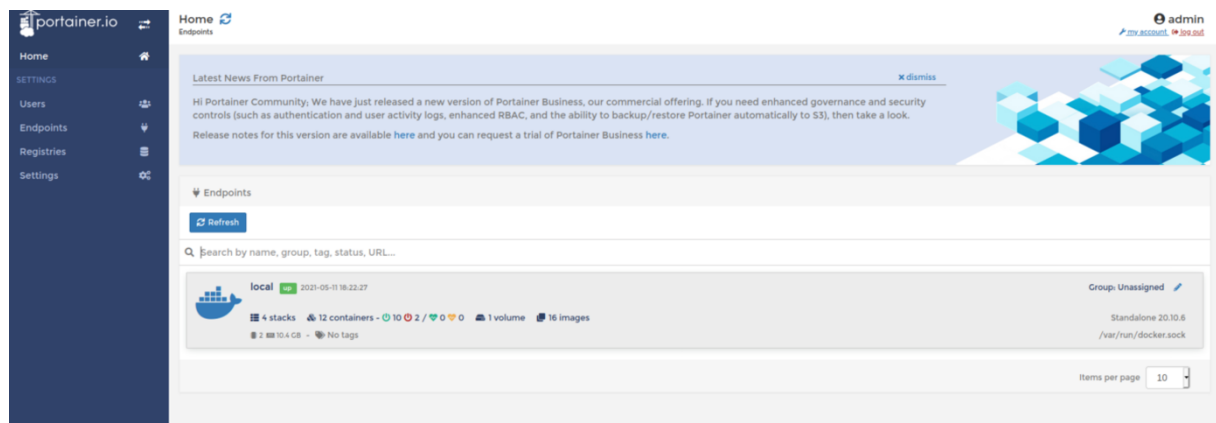
Portainer supprime la complexité associée aux orchestrateurs afin que tout le monde puisse gérer les conteneurs.

Pour installer portainer io on entre les commandes suivantes

```
$> docker volume create portainer_data
```

```
$> docker run -d -p 9001:9001 --name portainer_agent --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/docker/volumes:/var/lib/docker/volumes portainer/agent
```

Une fois le container démarré il suffit d'aller sur la page web et de créer notre user



Mise ne place de GitLab CI

Pour le déploiement de notre usine logiciel nous avons choisi Gitlab-ci cela nous permet de déployer une chaine qui déploie notre application et nous permet de la faire évoluer

En continue

Pour le déploiement de Gitlab nous avons choisis de le déployer sur notre docker



Pour déployer pour la première fois notre gitlab nous rentrons cette commande avec la configuration su dessous




```
$> docker run --detach \
  --ppublish 80:80 --publish 22:22 \
  --name gitlab \
  --restart always \
  --volume $GITLAB_HOME/gitlab/config:/etc/gitlab \
  --volume $GITLAB_HOME/gitlab/logs:/var/log/gitlab \
  --volume $GITLAB_HOME/gitlab/data:/var/opt/gitlab \
  gitlab/gitlab-ce:latest
```

Une fois le gitlab installer il faut créer générer une clé ssh et la copier dans le gitlab pour que notre ubuntu puisse push dans le gitlab






Une fois notre les configuré il nous reste plus qu'à instancier notre répertoire et push le code fourni




Administrator > GlobalNet






 **GlobalNet** 
Project ID: 2


  Star 0  Fork 0










25 Commits 1 Branch 0 Tags 543 KB Files 1 MB Storage

master globalnet / +  History  Find file  Web IDE   Clone

 **Deploy-globalnet-v5**
Administrator authored 2 hours ago  f95adc79 

 README  CI/CD configuration  Add LICENSE  Add CHANGELOG  Add CONTRIBUTING Auto DevOps enabled

 Add Kubernetes cluster

Name	Last commit	Last update
 apache	Deploy-globalnet-v5	2 hours ago
 build_repo	Deploy-globalnet-v5	2 hours ago
 ci-playbook	Deploy-globalnet-v5	2 hours ago
 golang/influx	Deploy-globalnet-v5	2 hours ago
 mysql	Deploy-globalnet-v5	2 hours ago
 registry	Deploy-globalnet-v5	2 hours ago
 traefik	Deploy-globalnet-v5	2 hours ago
 vault_conf	Deploy-globalnet-v5	2 hours ago
 .gitlab-ci.yml	Deploy globalnet-v5	2 hours ago

Mise en place d'un gitlab-runner

GitLab Runner est une application qui fonctionne avec GitLab CI / CD pour exécuter des tâches dans un pipeline.

Nous avons choisi d'installer l'application GitLab Runner sur notre VM Ubuntu des raisons de sécurité et de performances.

Pour installer gitlab-runner on exécute les commandes suivantes

```
$> sudo curl -L --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64"
$> sudo chmod +x /usr/local/bin/gitlab-runner
$> sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
$> sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner
$> sudo gitlab-runner start
```

Une fois gitlab-runner installer on va configurer notre runner avec gitlab

Pour cela on tape la commande suivante

```
$> gitlab-runner register
```

Puis on va dans les paramètres des runner dans gitlab et on rentre le lien de notre gitlab ainsi que sont Token

Ensuite on configure le runner en lui donnant un nom et on lui dit que les commandes seront exécutées en shell

Pour que notre runner fonctionne correctement nous avons dû paramétrer différents droits pour qu'il puisse exécuter toutes ses tâches

1- Être super utilisateur pour exécuter des commandes docker

On crée un groupe docker avec la commande suivante

```
$> sudo groupadd -f docker
```

On ajoute ensuite notre utilisateur au groupe

```
$> sudo usermod -a -G docker "$(whoami)"
```

On ajoute notre compte runner au groupe docker

```
$> sudo usermod -a -G docker gitlab-runner
```

On applique les modifications

```
$> newgrp docker
```

```
$> sudo systemctl restart docker
```

2- On donne à notre runner la possibilité de taper des commandes root sans mot de passe

On entre dans le fichier de gestion des droits avec la commande

```
$> visudo
```

Et on modifie la ligne du mot de passe sudo et on remplace par

```
%sudo  ALL=NOPASSWD: ALL
```

Voilà notre runner peut effectuer toutes les tâches

Création du pipeline

Pour le déploiement de notre usine logiciel nous avons décidé de créer des tâches qui seront appelées par des variables dans des schedules pour avoir une grande

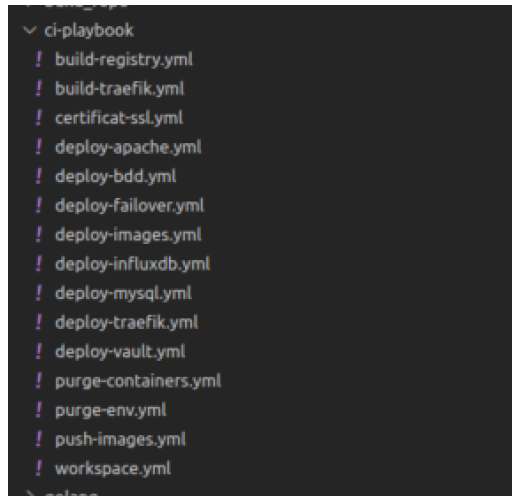
Liberté dans l'utilisation des différents jobs que nous placerons dans un dossier appelé ci-playbook

```
1 stages:
2   - workspace
3   - vault
4   - traefik
5   - ssl
6   - registry
7   - deploy
8   - purge
9   - push
10  - influxdb
11  - bdd
12  - apache
13  - mysql
14  - failover
15  - purge-env
16
17
18
19 include:
20   - local: 'ci-playbook/workspace.yml'
21   - local: 'ci-playbook/deploy-vault.yml'
22   - local: 'ci-playbook/deploy-traefik.yml'
23   - local: 'ci-playbook/certificat-ssl.yml'
24   - local: 'ci-playbook/build-registry.yml'
25   - local: 'ci-playbook/deploy-images.yml'
26   - local: 'ci-playbook/purge-containers.yml'
27   - local: 'ci-playbook/push-images.yml'
28   - local: 'ci-playbook/deploy-bdd.yml'
29   - local: 'ci-playbook/deploy-apache.yml'
30   - local: 'ci-playbook/deploy-mysql.yml'
31   - local: 'ci-playbook/deploy-failover.yml'
32   - local: 'ci-playbook/deploy-influxdb.yml'
33   - local: 'ci-playbook/purge-env.yml'
```

Nos différents jobs sont :

- Workspace : ce job prépare le répertoire de travail, tous les différents scripts sont exécutés depuis ce répertoire.
- Deploy vault : Déploie et configure notre serveur Vault.
- Deploy traefik : Déploie et configure le serveur de load balancing.
- Certificat ssl : génère et copie les certificats ssl.
- Build registry: Build Notre registry docker.
- Deploy images : déploie nos images dans docker
- Purge containers : supprime tous les containers en état « stop » ou « create ».
- Push images : Push nos images sur le registry.
- Deploy influxdb : déploie et configure le container influxdb.
- Gather Power: récupère l'information de consommation des containers
- Deploy bdd : déploie et configure le container de base de données
- Deploy apache : déploie et configure les 3 containers apache
- Deploy mysql: déploie et configure les 3 containers mysql
- Deploy failover : déploie et configure le container du failover
- Purge environnement : Supprime toutes les images et les containers de notre infra

Notre fichier gitlab-ci.yml est juste renseigner de l'ordre d'exécution des différents jobs ainsi que du chemin pour aller chercher les jobs



Création des Schedules

Afin de faciliter le déploiement et avoir une flexibilité de déploiement de jobs nous avons décidé de mettre en place des schedules avec des variables qui appel ou nous des jobs

Variable	WORKSPACE	true	X
Variable	DEPLOY_VAULT	true	X
Variable	TRAEFIK	true	X
Variable	CERTIFICAT	true	X
Variable	BUILD_REGISTRY	true	X
Variable	DEPLOY_IMAGES	true	X
Variable	PURGE	true	X
Variable	PUSH	true	X
Variable	INFLUXDB	true	X
Variable	BDD	true	X
Variable	APACHE	true	X
Variable	MYSQL	true	X
Variable	FAILOVER	true	X
Variable	PURGE_ENV	false	X

Exposition des difficultés rencontrées durant le projet :

Les principales difficultés rencontrées durant ce projet furent :

- Un manque de connaissances sur les différentes technologies
- Des compétences sur Linux
- La gestion des différents droits Linux
- La gestion des différents espaces de travail
- Rendre chaque technologie compatible entre elles
- Des délais pouvant être assez justes