
PROTIC.fr
IT / CONSEIL / FORMATION

Projet de Déploiement WordPress sur AWS avec Kubernetes

Phase 1

Déploiement d'un Environnement de Test sur Docker

PROTIC
IT / CONSEIL / FORMATION

🎯 Objectifs

- Objectif: Configurer un environnement de test local pour WordPress en utilisant Docker et Vagrant.

- Tâches:

L'équipe IT installe Docker et Vagrant sur leurs machines locales.

Un environnement Docker est créé pour exécuter WordPress, permettant aux développeurs de tester les fonctionnalités et les mises à jour avant le déploiement.

Livrable: Un rapport détaillant la mise en place de l'environnement de test et les configurations Docker/Vagrant utilisées.

Phase 1 : Préparation et Formation

Dans ce contexte, je me suis servi de mes acquis mais j'ai également dû me former sur des technologies que je n'avais jamais utilisées.

- **Git** : j'ai passé une soirée (3h de pratique) à visionner les vidéo TeamMeMore et à appliquer les exercices en parallèle.

- **Vagrant** : ne connaissant pas cet outil, j'ai dû passé quelques heures pour comprendre la philosophie de l'outils. Je me suis vite aperçu qu'elle était très proche de celle de Git ou encore de Docker.

Vagrant : choix du fournisseur

- ❌ Dans mon quotidien, je travaille depuis plusieurs années en local avec l'hyperviseur de type 2 **VMWare Workstation** et j'ai voulu essayer celui-ci avec Vagrant. Mais je me suis rapidement rendu que c'était non natif avec le produit et qu'il fallait utiliser un plugin qui semble instable. Ne voulant pas me rajouter cette difficulté dans l'immédiat, j'ai laissé de côté cette solution.

- ❌ Dans mon LAN, j'ai également à disposition une machine embarquant l'hyperviseur de type 1 **ProxmoxVE**. J'ai donc effectué quelques recherches sur Internet pour vérifier l'éventuel compatibilité de ce fournisseur. Là encore, bonne surprise, il semble y avoir une communauté qui travaille à sa mise en service. Mais trop peu de box semblent disponibles dans la librairie de Vragrant. Ce choix de fournisseur ne me parait donc pas pertinent pour ce projet. Je dois choisir un fournisseur de confiance et stable pour mon entreprise.

- ✅ Dans ce contexte, je fais le choix du fournisseur nativement compatible avec Vagrant. Ce dernier étant l'hyperviseur de type 2 **VirtualBox de Oracle**.

Mon environnement de test

- Pour mon projet, je choisi de la structurer sur la base d'un répertoire de travail local propre à cette phase, stocké sur ma machine et synchroniser sur un Cloud.

- Par exemple : D:\TMM\SysOps

- Ce répertoire est initialisé localement avec Git et je travaillerai sur la branche « test » pour ensuite merger sur la « master » après validation.

Mon environnement de test (suite)

- Mon répertoire Git contient également un fichier « .gitignore » contenant les extensions ou expressions régulières des fichiers à ignorer lors dans le répertoire.

- Je constate qu'un premier fichier journal est générer lors du l'initialisation d'un projet Vagrant en « .log ».

  - Extension ignorer dans le fichier :
    - *.log

Git : mode de travail

- Je décide de travailler en deux branches depuis le local :
  - Branche « **main** » initialisée, je merge uniquement en version fonctionnel sur des étapes de conformité du projet.
  - Branche « **test** » créée pour le développement continu du projet. Je ne travaillerai principalement que sur cette branche.
- Je pousserai vers GitHub lorsque ma phase 1 sera aboutie.

Création du Vagrantfile

- Je commence par initialiser mon repo avec la commande « vagrant init –m ».
  - -m pour minimiser le contenu du fichier Vagrantfile, sachant que je vais le modifier dans les grandes lignes.
  - Voici mon fichier :

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.network "public_network", bridge: "Intel(R) Wi-Fi 6E AX211 160MHz", ip: "192.168.8.10"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
    vb.cpus = "2"
    #vb.customize ["modifyvm", :id, "--nic1", "none"]
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y 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 $(lsb_release -cs) stable"
    sudo apt-get update
    sudo apt-get install -y docker-ce
    sudo apt-get install -y docker-compose
    cd /vagrant
    sudo docker-compose up -d
  SHELL
end
```

Commentaire du fichier Vagrantfile

- **Directive ruby pour le fichier**

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
```

- **Ouverture de la balise de configuration de la box à créer**

```
Vagrant.configure("2") do |config|
```

  - Version 2 de Vagrant
  - « do » permet de déclarer la variable « config » de ce qui le précède

- **Choix de la box de référence à utiliser**

```
config.vm.box = "ubuntu/bionic64"
```

  - Utilisation d'une image Ubuntu dans sa version Bionic et en architecture 64bits dans le référentiel « https://app.vagrantup.com/boxes »

Commentaire du fichier Vagrantfile (suite)

- **Configuration réseau de la box**

```
config.vm.network "public_network", bridge: "Intel(R) Wi-Fi 6E AX211 160MHz",
ip: "192.168.8.10«
```

  - Ajout d'une carte réseau pontée à ma carte physique nommée. Il est important de préciser son nom, dans la mesure où Vagrant pourrait avoir à faire un choix automatique pour une carte autre que celle connectée.
  - Je précise également une adresse IPv4 statique de manière à maitriser mon infrastructure. Si je ne précise rien, alors l'attribution sera faite par le serveur DHCP disponible sur mon réseau.

- **Choix du fournisseur**

```
config.vm.provider "virtualbox" do |vb|
```

  - Comme évoqué au début de ma présentation, je choisi de commencer mon projet en travaillant avec VirtualBox. Ce qui signifie que le contexte d'exécution de ma box sera l'hyperviseur d'Oracle.

- **Affectation des ressources matérielles pour la box**

```
vb.memory = "4096"
vb.cpus = "2"
```

Commentaire du fichier Vagrantfile (suite)

- **Désactivation de la NIC par défaut fournit par Vagrant**

```
#vb.customize ["modifyvm", :id, "--nic1", "none"]
```

  - Ce paramètre a été mis en commentaire par le caractère « # ».
  - Il s'agit de la désactivation de la carte NAT attachée par défaut à toute box démarrée par Vagrant. Seulement celle-ci sert à faire la liaison SSH entre l'hôte et la VM en utilisant les paires de clés générées par Vagrant. Nous pourrions tout à fait fournir nos propres clés mais cela n'apporte pas de valeur ajoutée à ce stade du projet. Je préfère donc monter ma box avec 2 NIC ; la NAT pour la communication SSH et la bridged pour la communication dans mon LAN.

- **Fermeture de balise**

```
end
```

  - Fin de la configuration matérielle de la box.

Commentaire du fichier Vagrantfile (suite)

- **Choix du provisionneur**

```
config.vm.provision "shell", inline: <<-SHELL
```

  - J'ai choisi d'utiliser un Shell pour exécuter mes prochaines commandes au démarrage de la VM Ubuntu.
  - Vagrant supporte plusieurs types de provisionneurs, y compris Shell, Ansible, Chef, Docker, et Puppet.
  - Il s'agit de la balise ouvrante.

Commentaire du fichier Vagrantfile (suite)

- **Installation du moteur Docker CE + Docker Compose**

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg


# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install -y docker-compose
```

  - J'ai utilisé la méthodologie de la documentation Docker pour effectuer cette installation dans les règles.

Commentaire du fichier Vagrantfile (suite)

- **Exécution du Docker-Compose**

```
# Run Docker Compose
    cd /vagrant
    sudo docker-compose up -d
  SHELL
```

  - Le premier paramètre déplace le prompt dans le dossier /vagrant de la VM. Important : ce dossier est synchronisé localement dans le dossier d'exécution de ma box. J'ai donc placé mon fichier docker-compose.yml dans ce même répertoire.
  - Je peux alors exécuter ma commande d'exécution des conteneurs liés en mode détaché et fermer mon shell.

- **Fermeture de balise**

```
end
```

  - Fin de la configuration du provisionnement de la box.

Commentaire du fichier docker-compose.yml

- **Définition de la version**

```
version: '3.1’
```

  - Il s’agit de la version de la syntaxe du fichier

- **Déclaration des services**

```
services:
```

  - Balise ouvrante (pas de balise fermante, seule l’indentation est importante) pour déclarer les services dont nous aurons besoin. Ces services représentent chacun un conteneur qui sera créé. Dans notre cas, nous aurons besoin de l’application Web ainsi que du moteur de base de données.

Commentaire du fichier docker-compose.yml (suite)

- **Déclaration et configuration de l'application Web**

```
wordpress:
  image: wordpress:latest
  restart: always
  ports:
    - 8080:80
```

  - Le service ou conteneur est nommé « **wordpress** ».
  - Ce conteneur va utiliser l'image nommée « **wordpress:latest** ». Cette image ne sera pas présente localement à la création de ma box Vagrant, elle sera donc téléchargée à chaque fois.
  - La balise « restart » indique le comportement du conteneur en cas d'arrêt de ce dernier. Dans le cas présent, je demande un niveau « **always** » pour que, quel que soit le type d'arrêt, ce conteneur redémarre.
  - Enfin je crée le mappage du port d'écoute du service Web afin que l'application soit accessible sur le port 8080 et que ce dernier soit redirigé sur le port d'écoute HTTP par défaut via l'IP de contact locale.

Commentaire du fichier docker-compose.yml (suite)

- **Déclaration et configuration de l'application Web**

```
environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: miguel
      WORDPRESS_DB_PASSWORD: miguel
      WORDPRESS_DB_NAME: miguel_db
    volumes:
      - wordpress_data:/var/www/html
```

- La balise **environnement** déclare les variables d'environnement dans mon conteneur et leur valeur associée.
- La balise **volumes** présente le point de montage entre le dossier **wordpress_data** dans ma VM (dossier par défaut de docker : `/var/lib/docker/volumes/vagrant_wordpress_data/_data`) vers le dossier de l'application dans le conteneur : `/var/www/html`.

Commentaire du fichier docker-compose.yml (suite)

- **Déclaration et configuration du moteur de base de données**

```
db:
  image: mysql:5.7
  restart: always
```

  - Le service ou conteneur est nommé « **db** ».
  - Ce conteneur va utiliser l'image nommée « **mysql:5.7** ». Cette image ne sera pas présente localement à la création de ma box Vagrant, elle sera donc téléchargée à chaque fois.
  - La balise « restart » indique le comportement du conteneur en cas d'arrêt de ce dernier. Dans le cas présent, je demande un niveau « **always** » pour que, quel que soit le type d'arrêt, ce conteneur redémarre.

Commentaire du fichier docker-compose.yml (suite)

- **Déclaration et configuration de l'application Web**

```
environment:
      MYSQL_DATABASE: miguel_db
      MYSQL_USER: miguel
      MYSQL_PASSWORD: miguel
      MYSQL_RANDOM_ROOT_PASSWORD: '1'
    volumes:
      - db_data:/var/lib/mysql
```

- La balise **environnement** déclare les variables d'environnement dans mon conteneur et leur valeur associée.
- La balise **volumes** présente le point de montage entre le dossier **wordpress_data** dans ma VM (dossier par défaut de docker : `/var/lib/docker/volumes/vagrant_db_data/_data`) vers le dossier de l'application dans le conteneur : `/var/lib/mysql`.

Etat de l'art

- A ce stade, VirtualBox exécute une VM Ubuntu (Bionic64) sur laquelle le moteur Docker ainsi que le composant Docker Compose sont installés.

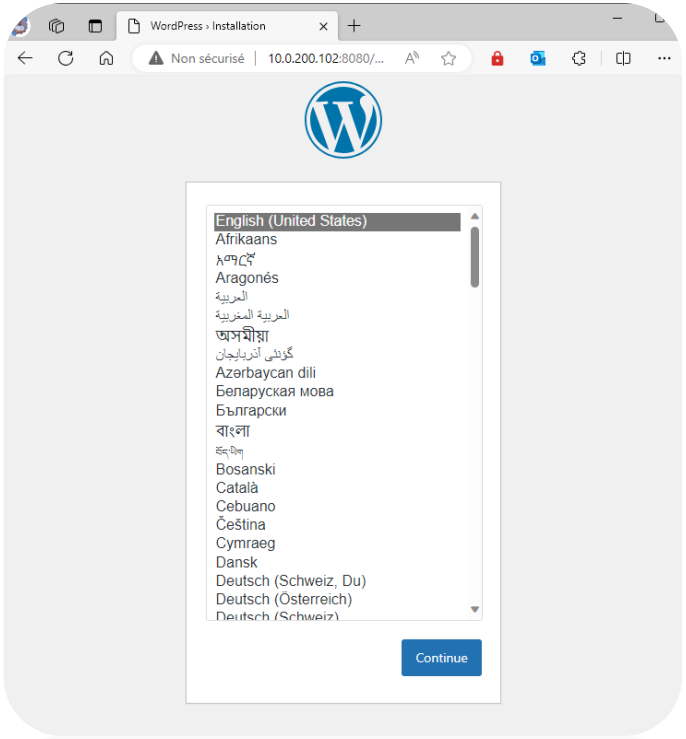
- Les conteneurs WordPress (frontend) et MySQL (backend) sont exécutés à l'intérieur de la VM.

- Le conteneur de l'application WordPress mappe le 8080 externe sur le port 80 localement.

- L'application WordPress est connectée à sa base de données créée et exécutées depuis le moteur MySQL dans son conteneur respectif.

Résultat

- La seule commande **vagrant up** me permet de démarrer cette infrastructure complète.




PROTIC .fr
IT / CONSEIL / FORMATION