[Documentation Docker]

Prise en main de Docker

Informations concernant Docker :

Docker est une plateforme open-source de virtualisation légère qui permet de distribuer et d'exécuter des applications et leurs dépendances dans des conteneurs isolés. Ces conteneurs sont desenvironnements autonomes et portables, ce qui facilite le déploiement et la gestion d'applications sur différentes infrastructures, qu'il s'agisse de serveurs locaux, de cloud ou autre. Docker offre une grande flexibilité et efficacité dans le déploiement et la gestion des applications, ce qui en fait un outil populaire dans le développement logiciel moderne et la gestion des infrastructures.

Étape 1 - Installation de Docker

Installer Docker sur le serveur Debian ciblé :

1. Mettre à jour la liste des paquets :

```
apt-get update
```

2. Installer les dépendances nécessaires :

```
apt-get install ca-certificates curl gnupg
install -m 0755 -d /etc/apt/keyrings
curl -fsSL
https://download.docker.com/linux/debian/gpg |
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg
```

3. Ajouter le référentiel Docker aux sources Apt :

```
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian $(.
/etc/os-release && echo
"$VERSION_CODENAME") stable" | tee
/etc/apt/sources.list.d/docker.list > /dev/null
apt-get update
```

4. Installer Docker :

```
apt-get install docker-ce docker-ce-cli
containerd.io docker-buildx-plugin docker-
compose-plugin
```

5. Vérifier que l'installation est correctement établie en exécutant la commande :

```
docker run hello-world
```

NOTE: Si l’on obtiens le message "Hello from Docker!", Docker est alors correctement installé.

Étape 2 - Découverte des commandes de base de Docker

Commandes de base de Docker.

1. Récupérez l'image NGINX depuis Docker Hub :

```
docker pull nginx
```

2. Vérifier que l'image NGINX est téléchargée :

```
docker images
```

3. Lancer un conteneur NGINX en arrière-plan, en faisant correspondre le port interne 80 au portexterne 8080 de la VM :

```
docker run -d -p 8080:80 nginx
```

4. Lister les conteneurs en cours d'exécution :

```
docker ps
```

5. Vérifier l'état du conteneur NGINX :

```
docker ps -a
```

6. Exécuter une commande à l'intérieur du conteneur NGINX pour obtenir des informationssystème :

```
docker exec -it <CONTAINER_ID> bash
```

7. Arrêter le conteneur NGINX :

```
docker stop <CONTAINER_ID>
```

8. Vérifier que le conteneur est arrêté :

```
docker ps
```

9. Redémarrer le conteneur NGINX :

```
docker start <CONTAINER_ID>
```

10. Vérifier que le conteneur est à nouveau en cours d'exécution :

```
docker ps
```

11. Supprimer le conteneur NGINX :

```
docker rm <CONTAINER_ID>
```

12. Vérifier que le conteneur NGINX a été supprimé :

```
docker ps -a
```

13. Supprimer l'image NGINX :

```
docker image rm nginx –force
```

14. Vérifier que l'image NGINX a été supprimée :

```
docker images
```

Étape 3 - Création d’images customisés Docker

1. Créer un fichier Dockerfile pour une image personnalisée "my-hello-world" :

```
touch Dockerfile
nano Dockerfile
```

Contenu du Dockerfile :

```
```bash
# Comment
FROM debian:12
CMD echo 'Hello World!'
```
```

2. Construire l'image personnalisée :

```
docker build -t my-hello-world .
```

3. Lancer un conteneur basé sur l'image "my-hello-world" :

```
docker run -d -p 8080:80 my-hello-world
```

4. Créer un fichier index.html dans le dossier actuel.

```
touch index.html
```

5. Ajoutez du code HTML au fichier index.html.

```
```html
<!DOCTYPE  html>
<html  lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible"
content="IE=edge">
   <meta name="viewport"
content="width=device-width,  initial-scale=1.0">
   <title>Document</title>
</head>
<body>
   <h1>Welcome to Docker !
```
```

```
</body>
</html>
```

6. Vérifier la présence du fichier HTML et son contenu dans le dossier courant.

```
ls -li
```

7. Créer une nouvelle image "my-nginx" basée sur l'image NGINX et inclure le fichier HTML crééprécédemment.

Étape 4 - Automatisation du déploiement de conteneurs (Docker compose)

1. Créez le fichier de configuration qui permet de déployer le même conteneur que celui obtenu àl'étape 2 petit 3.

```
nano my-nginx-compose.yml
```

2. Récupérez d'abord les images contenues dans le fichier de configuration, sans en lancer lesconteneurs.

```yaml
version: '3'
services:
  my-nginx:
    image: nginx
    ports:
      - "8080:80"
```

3. Lancer au premier plan le conteneur compris dans le fichier de configuration.Vérifier depuis le navigateur l'on accede bien à la page d'accueil de NGINX

```
docker-compose up
```

4. Stopper le conteneur.

```
docker stop <CONTAINER_ID>
```

5. Lancer en arrière-plan les conteneurs compris dans le fichier de configuration de la question 3.

```
docker-compose up -d
```

6. Récupérer les logs de console du conteneur lancé en arrière-plan.

```
docker-compose logs -f
```

7. Stopper le conteneur.

```
docker-compose down
```

8. Créer un fichier de configuration permettant d'obtenir une installation WordPress + PostgreSQL.Les données de la base PostgreSQL ainsi que celles de WordPress seront dans un volume Docker. L'installation WordPress doit être accessible de l'extérieur depuis le port 8080 de la VM.

Créer le fichier docker-compose.yml

```
touch  docker-compose.yml
```

```yaml
version: '3'
services:
  wordpress:
    image: wordpress
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: exampleuser
      WORDPRESS_DB_PASSWORD:
examplepassword
      WORDPRESS_DB_NAME: exampledb
    volumes:
      - wordpress_data:/var/www/html

  db:
    image: postgres
    environment:
      POSTGRES_USER: exampleuser
      POSTGRES_PASSWORD:
examplepassword
      POSTGRES_DB: exampledb
    volumes:
      -
postgres_data:/var/lib/postgresql/data

volumes:
  wordpress_data:
  postgres_data:
```