TP - Introduction à Docker

Prérequis

- Utilisation du terminal
- Notions de base en infrastructure et réseaux
- Language YAML

Préambule

Docker

Docker est une plateforme open-source lancée en 2013 qui facilite la création, le déploiement et l'exécution d'applications au sein de conteneurs. Un conteneur est une unité logicielle légère et portable qui regroupe tout le nécessaire pour faire fonctionner une application, y compris le code, les bibliothèques et les dépendances.

Avantages

Portabilité

Les conteneurs peuvent être exécutés de manière cohérente sur diverses plateformes, qu'il s'agisse d'un environnement local, de serveurs sur site ou de solutions cloud.

Légèreté

Contrairement aux machines virtuelles traditionnelles, les conteneurs partagent le noyau du système d'exploitation hôte, ce qui réduit leur empreinte mémoire et améliore les performances.

Isolation

Chaque conteneur fonctionne de manière isolée, garantissant que les processus et les dépendances d'une application n'interfèrent pas avec ceux d'une autre.

Scalabilité

Docker facilite le déploiement et la gestion de multiples instances d'une application, répondant ainsi efficacement aux variations de la demande.

Quelques Dates

- 2013 : Création de Docker par Solomon Hykes au sein de la société dotCloud et présentation au public.
- 2014 : Lancement de la première version stable et montée en popularité.
- 2015 : Docker Inc. crée l'Open Container Initiative (OCI) pour standardiser les conteneurs.
- 2017 : Lancement de Docker Enterprise Edition (EE) pour les entreprises.
- Aujourd'hui : Docker est un standard incontournable dans le déploiement d'applications modernes et le DevOps.

Image Docker

Une image Docker est un modèle immuable qui sert de base à la création de conteneurs. Elle encapsule tout le nécessaire pour exécuter une application : système d'exploitation minimal, bibliothèques, dépendances et code de l'application. Les images sont construites à partir de fichiers appelés `Dockerfile`, qui contiennent une série d'instructions pour assembler l'image.

Les images Docker sont stockées dans des registres, tels que Docker Hub, permettant leur partage et leur réutilisation. Lorsqu'un conteneur est lancé, il est instancié à partir d'une image spécifique, garantissant ainsi un environnement cohérent à chaque exécution. En d'autres termes, un conteneur est une instance active d'une image.

Les tags (étiquettes) dans Docker sont des identifiants alphanumériques qui permettent de distinguer et de gérer différentes versions ou variantes d'une image Docker. Ils sont ajoutés au nom de l'image, séparés par un deux-points (:), pour spécifier une version particulière. Par exemple, `mon_application:1.0.0` fait référence à la version `1.0.0` de l'image `mon_application`.

L'utilisation d'images Docker standardise le déploiement des applications, simplifie la gestion des versions et facilite la collaboration entre les équipes de développement et d'exploitation.

1 - Création d'une image Docker

Dans cette partie, vous allez être amenés à empaqueter une application PHP. Pour cela, vous allez devoir créer un `Dockerfile` puis construire votre image pour enfin lancer une instance de celle-ci.

Le `Dockerfile`

Un `Dockerfile` est un fichier permettant de créer une image docker. Ce fichier n'a **pas d'extension**, c'est son nom qui le différencie des autres fichiers. Il contient toutes les étapes de création de l'image

(installation des dépendances, copie des fichiers, lancement du programme, etc.).

Exemple de `Dockerfile` pour une application NodeJs:

```
FROM node:lts # Définit l'image sur laquelle se baser
WORKDIR /app # Définit le répertoire de travail
COPY app.js . # Copie des fichier dans l'image
RUN npm install # Exécution de l'installation des dépendances
EXPOSE 3000 # Ouverture du port 3000
CMD ["node", "app.js"] # Lancement de l'application au démarrage du conte
```

Toutes les informations sur la documentation des Dockerfile.

Informations Importantes

- Vous baserez votre image sur l'image nommée `php:apache`
- Tous les fichiers **.php** de l'application doivent être placés dans `/var/www/html` .
- Vous devez exécuter cette commande `docker-php-ext-install pdo pdo_mysql` pour installer le driver php pour la base de donnée
- Le port servi par l'application est le port **80**

Une fois que votre `Dockerfile` est écrit, vous devez construire *(build)* l'image. Pour cela, allez dans votre terminal puis dans le répertoire où se situe votre `Dockerfile` puis exécutez:

```
docker build -t [nom_image]:[tag] .
```

Note: Le point correspond au répertoire courant (où vous vous situez). Docker va rechercher un fichier nommé `Dockerfile` dans ce répertoire

Félicitation ! Vous venez de créer votre première image Docker.

Lançons maintenant un conteneur avec cette image. Dans votre terminal, écrivez:

```
docker run [paramètres] [nom_image]:[tag]
```

Liste des paramètres possibles

Paramètre	Utilité	Exemple d'utilisation
-p	Lie un port du conteneur à celui de l'hôte	-p [port_conteneur]:[port_hôte]

Paramètre	Utilité	Exemple d'utilisation
-d	Détache les logs du conteneur	-d
-e	Définit les variables d'environnements	-e API_KEY=123456789
-v	Monte un volume entre l'hôte et le conteneur	-v [chemin_hôte/nom_volume]: [chemin_conteneur]

Une fois votre conteneur lancé, vous devriez avoir un résultat de la sorte. Si cela est le cas, votre conteneur est bien lancé.

```
$ docker run -d [image]
40053fda4055f30b04ec49e5e603db327aea2e14237626ee498c8490e9672b7c
```

Voici quelques commandes qui vont vous aider à administrer vos conteneurs.

Commande	Utilité	Utilisation
logs	Permet de voir les logs d'un conteneur	docker logs [nom_conteneur]
ps	Permet de voir les conteneurs lancés	docker ps (-a, pour tous les avoir)
stop	Permet de stopper le conteneur	docker stop [nom_conteneur]
exec	Permet d'exécuter une commande dans le conteneur	docker exec [nom_conteneur] [commande]

Lancement de la base de donnée

L'application repose sur une base de données pour stocker ses informations. Pour garantir son bon fonctionnement, il est nécessaire de lancer un conteneur dédié à cette base de données.

```
# Déplacez vous dans le répertoire du projet

docker run --name database -p 3306:3306 -v ./app/table.sql:/docker-entryp
```

Si tout s'est bien passé, ouvrez votre navigateur et accédez à http://localhost. Vous devriez voir l'application PHP en cours d'exécution.

2 - Déploiement d'une infrastructure avec Docker

Dans la section précédente, nous avons démarré les différents composants de l'infrastructure de l'application séparément. Cependant, à mesure que l'infrastructure se complexifie, cette approche

devient rapidement difficile à gérer.

C'est là qu'intervient **Docker Compose**, un outil permettant de définir et de gérer des applications multi-conteneurs à l'aide d'un simple fichier de configuration *(compose.yml)*. Grâce à Docker Compose, il est possible de lancer l'ensemble des services de l'application en une seule commande, simplifiant ainsi le déploiement et la gestion de l'infrastructure.

Dans cette partie, votre but est de créer un fichier `compose.yml` afin de lancer l'application que vous venez de créer ainsi que sa base de donnée correspondante.

Structure d'un fichier Compose

Un fichier `compose.yml` est structuré autour de plusieurs sections principales :

- **services** : Définit les différents conteneurs (services) qui composent l'application.
- **networks** *(optionnel)* : Permet de configurer des réseaux personnalisés pour la communication entre les conteneurs.
- **volumes** *(optionnel)* : Définit les volumes pour la persistance des données en dehors des conteneurs.

Exemple simple de fichier Compose

Un exemple de fichier `compose.yml` :

```
services:
  web:
    image: nginx:latest # Définit l'image du conteneur
    ports: # Forward de port conteneur:hôte
      - "80:80"
  app:
    container_name: app # Définit le nom et hostname du conteneur
    build: app/ # Définit le répertoire où trouver le Dockerfile pour bui
    depends_on: # Définit les dépendances entre conteneurs
      - db
  db:
    image: postgres:latest
    environment: # Définition des variables d'environnements
      POSTGRES_PASSWORD: 1234
    volumes: # Définition des volumes du conteneur
      - ./data:/data
```

Précisions

- Dans un fichier `compose.yml`, on peut définir une image a utiliser pour notre conteneur ou bien en build une en spécifiant le répertoire où se trouve le Dockerfile.
- Dans un fichier `compose.yml`, on peut définir des volumes qui sont des espaces de stockages que le conteneur va utiliser. Vous verrez en détails comment ils fonctionnent dans le premier bonus.

Une fois que votre fichier Compose est créé, ouvrez un termnial et mettez vous dans le répertoire contenant ce fichier. Pour lancer l'infrastructure, exécutez la commande:

```
docker compose up (-d)
```

Voici quelques commandes qui vont vous aider à administrer vos conteneurs.

Commande	Utilité	Utilisation
down	Détruit l'architecture	docker compose down
stop	Stoppe tout les conteneurs	docker compose stop
start	Démarre tout les conteneurs si ils étaient éteints	docker compose start

Si tout s'est bien passé, ouvrez votre navigateur et accédez à http://localhost. Vous devriez voir l'application PHP en cours d'exécution.

3 - Améliorations bonus

Ajout d'un volume pour la persistence des données de la base

Notre application de liste de courses se connecte à une base de données pour enregistrer et supprimer les éléments de la liste. Cependant, lorsqu'on détruit puis redémarre l'infrastructure, les éléments de la liste disparaissent.
L'objectif de cette section est donc de configurer un volume pour la base de données, afin de rendre les données de la liste persistantes malgré les redémarrages.

Comprendre les volumes

Un volume est un espace de stockage dédié que l'on peut attacher à un conteneur. Cela permet au conteneur de disposer de données qui persistent en dehors de son cycle de vie normal. Les volumes sont particulièrement utiles dans les deux cas suivants :

- **Partage de données entre plusieurs conteneurs** : plusieurs conteneurs peuvent accéder et partager les mêmes informations.
- **Persistance des données** : les données ne sont pas perdues lorsque le conteneur est supprimé ou redémarré.

Les volumes peuvent être créés via la ligne de commande ou définis dans un fichier `compose.yml`. La configuration d'un volume se fait en indiquant le nom du volume *(ou le chemin d'un dossier sur l'hôte)*, suivi du chemin dans le conteneur où la donnée doit être stockée. Ces deux éléments sont séparés par deux-points (`:`).

Exemples :

```
# Ici, un volume nommé "data" est monté dans le répertoire /var/www/html
data:/var/www/html

# Ici, le répertoire "data" sur l'hôte est monté dans le répertoire /var/
./data:/var/www/html
```

Types de volumes

1. Volume Docker

Le volume Docker est un espace de stockage géré par Docker lui-même. Il agit comme une boîte noire : bien que les données y soient stockées en toute sécurité, leur contenu n'est pas directement visible ou modifiable par l'utilisateur.

Avantages :

- Sécurité des données
- Gestion simplifiée (aucune manipulation manuelle nécessaire)

Quelques commandes pour gérer les volumes:

Commande	Utilité	Utilisation
create	Créer un volume	docker volume create [nom]
ls	Liste les volumes	docker volume ls
inspect	Permet d'avoir des informations supplémentaires	docker volume inspect [nom]
rm	Supprime un volume	docker volume rm [nom]

Ces volumes peuvent également être déclarés dans un fichier `compose.yml` :

```
services:
  app:
    image: foo
    volumes:
      - data:/var/www/html  # "data" correspond au volume défini ci-dessc
```

```
volumes:
  data:
```

2. Volume du système de fichiers hôte

Ce type de volume permet de lier un répertoire ou un fichier du système de l'hôte à un conteneur. Cela signifie que le conteneur peut lire, enregistrer ou supprimer des données directement sur le système de l'hôte.

Avantages :

- Transparence et facilité d'accès aux données (les données sont visibles directement sur l'hôte)

Inconvénients :

- Risque de perte ou de déplacement accidentel des données, ce qui pourrait empêcher le conteneur d'y accéder.

L'association d'un volume du système de fichiers hôte se fait via la ligne de commande avec l'argument -v ou directement dans un fichier `compose.yml` :

```
services:
  app:
    image: foo
    volumes:
      - ./data:/var/www/html  # "./data" fait référence au dossier sur l'
```

Ajout de plusieurs réseaux pour segmenter l'infrastructure

Une bonne pratique en matière d'infrastructure réseau consiste à la segmenter, de manière à ce que seules les machines devant être accessibles au public le soient, tandis que les autres restent protégées. Pour cela, il est possible d'utiliser Docker pour créer des réseaux virtuels, permettant ainsi de contrôler l'accès aux différentes parties de l'architecture.

L'objectif est donc de configurer plusieurs réseaux afin de rendre l'application web accessible au public tout en garantissant que la base de données reste sécurisée et inatteignable depuis l'extérieur.

Les réseaux avec Docker

Lorsqu'on travaille avec des conteneurs, la gestion du réseau est essentielle pour permettre aux services de communiquer entre eux tout en contrôlant leur accessibilité depuis l'extérieur. Docker propose un système de gestion des réseaux qui simplifie cette tâche.

À quoi servent les réseaux ?

Les réseaux permettent aux conteneurs de communiquer entre eux et avec l'extérieur. Sans configuration spécifique, un conteneur isolé ne peut ni envoyer ni recevoir de données.

L'utilisation de réseaux permet de :

- Connecter plusieurs conteneurs entre eux, par exemple pour relier une application web à une base de données.
- Contrôler l'accès aux services en limitant la visibilité de certains conteneurs.
- Séparer les environnements pour améliorer la sécurité et la scalabilité.

Les différents types de réseaux Docker

Docker propose plusieurs types de réseaux, adaptés à différents usages :

- **Bridge (par défaut)** : Permet aux conteneurs d'un même réseau de communiquer entre eux tout en restant isolés du reste du système.
- **Host** : Le conteneur partage directement le réseau de l'hôte, supprimant ainsi l'isolation réseau.
- **Overlay** : Utilisé dans un cluster Docker Swarm pour connecter des conteneurs sur plusieurs hôtes.
- **Macvlan** : Permet d'assigner une adresse IP spécifique aux conteneurs, comme s'ils étaient des machines physiques sur le réseau.
- **None** : Désactive complètement la connectivité réseau du conteneur.

Comment Docker gère les réseaux ?

Docker intègre un système de gestion des réseaux qui permet de créer, configurer et gérer des connexions entre conteneurs. Chaque conteneur peut être attaché à un ou plusieurs réseaux, selon les besoins.

Docker fournit des commandes pour manipuler ces réseaux :

Commande	Description
`docker network ls`	Lister les réseaux existants
`docker network create [nom]`	Créer un nouveau réseau
`docker network inspect [nom]`	Obtenir des informations sur un réseau
`docker network connect [réseau] [conteneur]`	Connecter un conteneur à un réseau
`docker network disconnect [réseau] [conteneur]`	Déconnecter un conteneur d'un réseau

Il est aussi possible de créer et gérer des réseaux directement dans un fichier `compose.yml`. Cela permet d'organiser les services et de contrôler leur connectivité de manière déclarative. Par exemple:

```
services:
  web:
    image: nginx
    networks:
      - frontend

  app:
    image: myapp
    networks:
      - frontend
      - backend

  db:
    image: mysql
    networks:
      - backend

networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
```