Installation de Docker

Introduction

Docker est une plateforme permettant de développer, déployer et exécuter des applications dans des conteneurs. Un conteneur est une unité légère, autonome et exécutable qui contient tout ce qui est nécessaire pour faire fonctionner une application, y compris le code, les bibliothèques, les dépendances et les configurations système. Cela permet d'assurer une portabilité maximale entre différents environnements (développement, production, etc.).

Mise à jour des paquets et installation des dépendances

sudo apt-get update
sudo apt-get install ca-certificates curl

Créer le répertoire pour la clef GPG

sudo install -m 0755 -d /etc/apt/keyrings

Télécharger la clef GPG officielle

sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

Ajouter le dépôt Docker

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Installation de Docker

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildxplugin docker-compose plugin

Vérification de l'installation

docker run hello-world

Réponse si Docker est installé correctement

Hello from Docker! This message shows that your installation appears to be working correctly.

Commandes de base de Docker

Récupérer l'image d'un service

docker pull [nom_du_service]

Afficher la liste des images téléchargées

docker images

Exécuter un conteneur en arrière-plan avec mapping de port

docker run -d -p 8080:80 --name [nom_du_conteneur] [nom_du_service]

Accéder au service depuis un navigateur

http://[IP_VirtualMachine]:8080

Lister les conteneurs actifs

docker ps

Lister tous les conteneurs

docker ps -a

Afficher des informations sur le noyau de l'hôte et le système d'exploitation

docker exec -it [nom_du_conteneur] bash
uname -a

Arrêter le conteneur

docker stop [nom_du_conteneur]

Relancer le conteneur arrêté

docker start [nom_du_conteneur]

Supprimer le conteneur

docker rm [nom_du_conteneur]

Supprimer l'image d'un service

docker rmi [nom_du_service]

Création d’images Docker

Créer un fichier `Dockerfile`

```
FROM debian:latest
CMD echo "Hello World!"
```

La spécification CMD dans un Dockerfile simple crée une valeur par défaut : si nous passons des arguments sans option à docker run , ils écraseront la valeur CMD .

Construire une image

```
docker build -t [my-image] .
```

Créer un fichier HTML pour un service

```
 echo "<h1>Hello from my custom page!</h1>" > index.html
```

Créer un `Dockerfile` basé sur une image d’un service

```
FROM [nom_du_service]:latest
COPY index.html /usr/share/[nom_du_service]/html/index.html
```

Lancer un conteneur basé sur une image personnalisée

```
docker run -d -p 8080:80 --name [my-image] [my-image]
```

Automatisation avec Docker Compose

Créer un fichier `docker-compose.yml`

```
version: '3'
services:
  [nom_du_service]:
    image: [nom_du_service]:latest
    ports:
      - "8080:80"
```

Récupérer les images sans lancer les conteneurs

```
docker-compose pull
```

Lancer les conteneurs au premier plan

```
docker -compose up
```

Lancer les conteneurs en arrière-plan

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

Vérifier les logs du conteneur

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

Créer un fichier Docker Compose pour WordPress et PostgreSQL

```
version: '3'
services:
  wordpress:
    image: wordpress:latest
    ports:
      - "8080:80"
    volumes:
      - wordpress_data:/var/www/html
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: example
      WORDPRESS_DB_NAME: wordpress
  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: wordpress
      POSTGRES_PASSWORD: example
    volumes:
      - db_data:/var/lib/postgresql/data
volumes:
  wordpress_data:
  db_data:
```

Model-View-Controller

Introduction

Le Model-View-Controller (MVC) est un modèle architectural conçu pour organiser des applications en séparant clairement la logique métier, l'affichage des données, et le traitement des interactions utilisateur.
Objectif : Faciliter la modularité et la réutilisabilité du code, simplifier la maintenance en séparant les responsabilités et permettre une meilleure collaboration entre développeurs, designers et spécialistes métier.

Concepts

- Model (Modèle) : Gère la logique métier et l'état des données. Ne dépend pas de l'affichage ou des interactions utilisateur.
- View (Vue) : Présente visuellement les données du modèle à l'utilisateur. Elle est passive et sans logique métier.
- Controller (Contrôleur) : Interprète les actions utilisateur, agit sur le modèle, et choisit la vue appropriée à afficher.

Architecture

- Model : Encapsule les données, assure les traitements métiers.
- View : Doit uniquement rendre des informations ; pas de logique métier.
- Controller : Gère la navigation, les flux et les requêtes utilisateur.

Utilisation

- Séparation claire entre présentation et logique métier.
- Testabilité accrue : modèle et contrôleurs testables indépendamment.
- Travail collaboratif plus efficace entre développeurs backend, frontend et UX designers.
- Complexité supplémentaire pour des projets simples.
- Multitude de fichiers/classes à gérer.
- Rendre la Vue légère : pas de traitements métier.
- Garder le Contrôleur fin : il orchestre sans traiter la donnée en profondeur.
- Centraliser la logique métier dans le Modèle.
- Adopter une architecture cohérente et documentée.
- Utiliser des tests unitaires pour les modèles et les contrôleurs.

Conclusion

Le modèle Model-View-Controller (MVC) reste une architecture robuste et éprouvée pour le développement d'applications où l'interface utilisateur évolue indépendamment de la logique métier.
En suivant des bonnes pratiques claires – telles que maintenir une séparation stricte des responsabilités et favoriser des vues passives – MVC permet de construire des systèmes modulaires, testables et évolutifs.
Il est recommandé d'utiliser MVC dans les situations suivantes : applications complexes avec de multiples vues ou états utilisateur, projets collaboratifs où la séparation frontend/backend est forte, développements long-terme nécessitant une maintenance régulière.