1-

- ✓ **les commandes Docker CLI pour démarrer, arrêter et gérer les conteneurs**

Commande pour construire une image :

docker build . -t nomimage

Exemple : docker build . -t frontend

docker build . -t backend

Commande pour démarrer une image à partir d' un conteneur :

docker run -d --name nomcontainer nomimage

Exemple : docker run -d --name projet-docker-3dct-frontend frontend

- ✓ Configurer et utiliser la liaison de port pour permettre la communication vers/depuis des conteneurs :
  docker run -d -p 3000:3000 --name nomduconteneur nomimage
  docker run -d -p 4000:4000 --name nomduconteneur nomimage

  Ex : docker run -d -p 3000:3000 --name projet-docker-3dct-frontend frontend
  docker run -d -p 4000:4000 --name projet-docker-3dct-backend backend

  Exécuter le conteneur dans le un terminal
  docker exec -it nomcontainer bash
  Ex : docker exec -it projet-docker-3dct-frontend bash
  Dans alpine linux le shell par défaut est sh
  docker exec -it nomcontainer sh
  Ex : docker exec -it projet-docker-3dct-frontend sh
  docker exec -it projet-docker-3dct-backend sh

Arrêt d'un conteneur en cours d'exécution :

docker stop nomducontainer|conteneurId

Ex : docker stop projet-docker-3dct-frontend

Redémarrage d'un conteneur arrêté :

docker start nomducontainer|conteneurId

Suppression d'un conteneur :

docker rm nomconteneur|conteneurId

Affichage des logs d'un conteneur :

docker logs nomduconteneur|conteneur Id

Suppression d'un conteneur sans le stopper :

docker rm -f nomduconteneur|conteneurId

Affichage des images :

docker images

Affichages des conteneurs :

docker ps

2- <u>Comparaison entre virtualisation et conteneurisation</u> :

Performance : La virtualisation implique une surcharge supplémentaire, car chaque machine virtuelle exécute une copie complète d'un système d'exploitation. En revanche, les conteneurs partagent le même noyau d'OS, ce qui rend leur exécution plus légère et plus rapide. Ils consomment moins de ressources système, ce qui se traduit par une densité plus élevée (plus de conteneurs peuvent être exécutés sur une machine par rapport aux VMs).

Sécurité : En termes de sécurité, la virtualisation offre une isolation plus robuste puisque chaque machine virtuelle fonctionne comme un système autonome avec son propre OS. Les conteneurs, bien qu'ils soient isolés les uns des autres, partagent le même noyau d'OS, ce qui peut potentiellement constituer une faille de sécurité si un conteneur parvient à s'échapper et à accéder au noyau.

Isolation : La virtualisation assure une isolation totale, chaque VM fonctionnant comme une machine distincte avec son propre OS, ses propres applications, bibliothèques et binaires. Les conteneurs, en revanche, sont isolés au niveau du processus plutôt qu'au niveau du système. Ils partagent le noyau de l'OS et certains binaires et bibliothèques, selon les besoins de l'application.

Portabilité : En matière de portabilité, la conteneurisation l'emporte sur la virtualisation. Les conteneurs encapsulent les dépendances de l'application, garantissant ainsi que celle-ci fonctionnera de la même manière sur n'importe quel système capable de faire tourner le moteur de conteneurisation (comme Docker). Les machines virtuelles, quant à elles, sont généralement plus lourdes et nécessitent un hyperviseur pour fonctionner, ce qui peut rendre leur transfert d'un système à un autre plus complexe.

Infrastructure : Enfin, la virtualisation et la conteneurisation nécessitent différentes infrastructures de gestion. Les machines virtuelles sont généralement gérées à l'aide de logiciels de gestion de la virtualisation comme VMware vSphere ou Microsoft Hyper-V. Les conteneurs, en revanche, sont souvent orchestrés par des systèmes comme Kubernetes ou Docker Swarm.

<u>Discutez de l'impact de Docker sur le développement, les tests et cycles de déploiement dans un contexte d'entreprise :</u>

Docker apporte une valeur immédiate dans les entreprises, augmentant ainsi rapidement sa productivité. Il vous permet de diffuser vos applications en production plus rapidement tout en réduisant les coûts d'infrastructure et de maintenance, accélérant ainsi la mise sur le marché de nouvelles solutions, de ce fait il fournit de nouvelles expériences client allant des applications monolithiques traditionnelles aux applications cloud natives.

3- Création d'images Docker :

Écrire un Dockerfile qui inclut des instructions telles que FROM, RUN, COPY, EXPOSER et CMD :

Exemple : Dockerfile du frontend :

```
# Définir l'image de base

FROM node:14-alpine


# Création de l'utilisateur non-root

RUN adduser -D mohamedfarouck

# Définition de l'utilisateur non-root

USER mohamedfarouck

# Définition du répertoire de travail

WORKDIR /app

# Copie du fichier package.json

COPY package.json /app/package.json

RUN npm install

# Copie du reste de l'application

COPY .  .

# Exposition du port 3000

EXPOSE 3000

CMD ["npm", "start"]
```

4- Sécurité et qualité des conteneurs :

Utilisez les meilleures pratiques de sécurité Docker (par exemple, utilisateur non root, base minimale image) :

```
# Définir l'image de base minimale

FROM node:14-alpine

# Création de l'utilisateur non-root

RUN adduser -D mohamedfarouck

# Définition de l'utilisateur non-root

USER mohamedfarouck
```

Introduire des outils d'analyse de sécurité pour identifier et corriger les vulnérabilités dans les images :

```
# Définir l'image de base

FROM node:14-alpine as base
```

```
# Installation de Trivy

RUN apk --no-cache add wget

RUN wget -qO /usr/local/bin/trivy
https://github.com/aquasecurity/trivy/releases/download/v0.19.2/trivy_0.19.2_linux_amd64

RUN chmod +x /usr/local/bin/trivy

# Création de l'utilisateur non-root

RUN adduser -D -s /bin/bash mohamedfarouck

# Définition de l'utilisateur non-root

USER mohamedfarouck

# Définition du répertoire de travail

WORKDIR /app

# Copie du fichier package.json

COPY package.json .

# Installation des dépendances

RUN npm install

# Copie du reste de l'application

COPY .  .

# Exécution de l'analyse de sécurité avec Trivy

RUN trivy --light --no-progress --exit-code 1 /app

# Exposition du port 3000

EXPOSE 3000

# Commande de démarrage de l'application

CMD ["npm", "start"]
```

Bonus : Optimisation des images Docker : Implémenter des builds en plusieurs étapes et Sélectionner les images de base appropriées et minimisez le nombre de couches.

```
# Définir l'image de base

ARG NODE_VERSION=14-alpine

FROM node:${NODE_VERSION} as base

# Installation de Bash

RUN apk add --no-cache bash

# Création de l'utilisateur non-root

RUN adduser -D -s /bin/bash mohamedfarouck
```

```
# Définition de l'utilisateur non-root
USER mohamedfarouck
# Définition du répertoire de travail
WORKDIR /app
# Copie du fichier package.json
COPY package.json .
# Installation des dépendances
RUN npm install
# Copie du reste de l'application
COPY .  .
# Exposition du port 3000 (uniquement pour le développement)
EXPOSE 3000
# Commande de démarrage de l'application (uniquement pour le développement)
CMD ["npm", "start"]
# Définition des étapes spécifiques à l'environnement de production
FROM base as prod
RUN npm run build
# Utilisation d'une image Nginx pour l'environnement de production
FROM nginx as final
COPY --from=prod /app/build /usr/share/nginx/html
```

NB : la commande pour le docker compose est : docker-compose up --build