Partie 2:

1. Récupère l'image avec docker pull nginx
2. Voir les images avec docker images
3. docker run -d -p 8080:80 --name "mon_nginx" nginx #pour créer le docker
4. docker ps -a pour voir
   ID: 434f3518c4c9
   Nom: elastic_easley
5. Le conteneur est UP
6. docker exec -it "ID du container" bash
   uname -a #on obtient :
   Linux 434f3518c4c9 5.10.0-25-amd64 #1 SMP Debian 5.10.191-1 (2023-08-16) x86_64 GNU/Linux
7. docker stop <ID>
8. C'est Exited
9. Docker start <Nom ou ID>
10. On voit avec docker start <Nom>
    C'est en Up
11. Docker rm <ID> #pour supprimer container
12. Docker rmi <Nom> #pour supprimer les images
13. Docker images #pour voir les images

Partie 3:

1. <u>Créer un dossier :</u>
   mkdir my-docker-project
   cd my-docker-project

   <u>Créer le Dockerfile :</u>
   nano Dockerfile

   <u>Ajouter le contenu au Dockerfile:</u>
   # Utiliser l'image Debian comme base
   FROM debian:latest

   # Installer bash (si nécessaire)
   RUN apt-get update && apt-get install -y --no-install-recommends \ bash

   # Commande à exécuter au démarrage du conteneur
   CMD ["echo", "Hello World!"]

   <u>Construire l'image :</u>
   docker build -t my-hello-world .

   <u>Lancer le conteneur :</u>
   docker run --rm my-hello-world

2. cd my-docker-project
   nano index.thml

3. J’ai ajouté <!DOCTYPE html>
   <html lang="fr">
   <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Hello World Page</title>
   </head>
   <body>
   <h1>Hello World!</h1>
   Ceci est une page HTML de test.
   </body>
   </html>

4. faire un ls pour vérifier si il est là et cat index.html pour voir le contenue

5. Créer le fichier index.html et ajouter :
   nano index.html
   <!DOCTYPE html>
   <html lang="fr">
   <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Ma Page HTML</title>
   </head>
   <body>
   <h1>Bienvenue sur ma page personnalisée !</h1>
   </body>
   </html>

   Créer le Dockerfile et mettre:
   nano Dockerfile

   # Utiliser l'image de base NGINX
   FROM nginx:latest

   # Copier le fichier HTML dans le répertoire par défaut de NGINX
   COPY index.html /usr/share/nginx/html/index.html

   Construire l'image donc aller dans le dossier my-nginx puis faite cette cmd:
   docker build -t my-nginx .

6. Pour lancer un conteneur basé sur l'image my-nginx :
   docker run -d -p 8080:80 --name my_nginx_container my-nginx

7. Pour vérifier cherché :
   http://localhost:8080

Partie 4 :

Étape 1 : Créer le fichier de configuration Docker Compose

Pour commencer, on va créer un fichier `docker-compose.yml` qui nous permettra de déployer facilement le conteneur NGINX. Mettez le contenu suivant dans le fichier :

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

Ici, on définit un service appelé `nginx`, basé sur l'image `nginx:latest`. Le port 80 du conteneur est exposé sur le port 8080 de votre machine hôte. Simple et efficace !

Étape 2 : Récupérer les images sans lancer les conteneurs

Avant de lancer quoi que ce soit, on va juste récupérer l'image pour être prêts. Pour ça, tapez :

```
docker-compose pull
```

C'est tout ! Cela va télécharger l'image, sans démarrer de conteneur. C'est utile si vous voulez être sûr que tout est prêt à l'emploi sans encore démarrer le service.

Étape 3 : Lancer le conteneur au premier plan

Prêt à voir NGINX en action ? Lançons le conteneur au premier plan :

```
docker-compose up
```

Maintenant, ouvrez votre navigateur et tapez l'adresse suivante : `http://<IP_VM>:8080`, où `<IP_VM>` est l'adresse IP de votre machine virtuelle Debian. Vous devriez voir apparaître la fameuse page "Welcome to nginx!".

Étape 4 : Stopper le conteneur

Pour arrêter le conteneur qui tourne au premier plan, vous pouvez simplement utiliser `Ctrl+C` dans la console. Si vous préférez ouvrir un autre terminal, vous pouvez taper :

```
docker-compose down
```

Étape 5 : Lancer les conteneurs en arrière-plan

Maintenant, on va lancer tout ça en arrière-plan pour libérer le terminal :

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

Étape 6 : Récupérer les logs en temps réel du conteneur

Vous voulez voir ce qui se passe sous le capot en temps réel ? Voici la commande magique :

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

Le `-f` vous permet de suivre les logs en temps réel. Pratique pour vérifier si tout tourne bien ou pour déboguer si quelque chose ne fonctionne pas comme prévu.

Étape 7 : Stopper le conteneur en arrière-plan

Pour arrêter le conteneur qui tourne en arrière-plan, utilisez :

```
docker-compose down
```

Notez que cette commande n'est pas exactement la même que `Ctrl+C` : ici, le conteneur est arrêté *et* supprimé proprement, alors que `Ctrl+C` se contente d'arrêter le processus.

Étape 8 : Fichier de configuration pour WordPress et PostgreSQL

Passons à quelque chose d'un peu plus complexe : une installation WordPress avec une base de données PostgreSQL. Voici le fichier `docker-compose.yml` pour cela :

```
version: '3'
services:
  wordpress:
    image: wordpress:latest
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: example_user
      WORDPRESS_DB_PASSWORD: example_password
      WORDPRESS_DB_NAME: example_db
    volumes:
      - wordpress_data:/var/www/html

t  db:
```

```
    image: postgres:latest
    environment:
      POSTGRES_USER: example_user
      POSTGRES_PASSWORD: example_password
      POSTGRES_DB: example_db
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  wordpress_data:
  db_data:
```

Ce fichier définit deux services : `wordpress` et `db` (pour PostgreSQL). Les volumes `wordpress_data` et `db_data` vont nous permettre de persister les données entre les redémarrages, ce qui est crucial pour ne pas perdre toutes les configurations et contenus.

Pour lancer cette configuration :

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

Puis, ouvrez votre navigateur et allez à l'adresse `http://<IP_VM>:8080` pour accéder à l'installation de WordPress.