Installation de l'environnement de travail du développeur

1) Installer un éditeur de code

- Installer l'éditeur de code **Visual Studio Code** sur votre ordinateur.
- Ouvrir VS Code et créer un nouveau fichier dans lequel vous écrirez le texte suivant : « Ma page web »
- Enregistrer le fichier sous le nom index.html
- Fermer VS Code, puis double-cliquer sur votre fichier pour l'ouvrir.

Modifiez l'extension du fichier (par exemple, .txt au lieu de .html), quelle différence notez-vous ?

_____

Que pouvez-vous en déduire sur les fichiers HTML ?

_____

2) Docker

Précédemment, nous avons ouvert un fichier HTML dans notre navigateur. Nous allons maintenant faire la même chose en passant par un protocole HTTP via une communication client-serveur pour simuler le web réel.

Pour cela, nous allons installer sur notre machine un serveur web grâce à Docker.

Docker est un outil qui permet d'exécuter des applications dans des **conteneurs**.
Ces applications n'ont pas besoin d'être installées directement sur votre ordinateur : elles fonctionnent dans un environnement isolé, le conteneur.

Un **conteneur**, c'est comme un **mini-ordinateur** très léger qui contient **tout ce qu'il faut** pour faire tourner des applications.
Pour créer et gérer des conteneurs, **installer Docker** sur votre ordinateur.

3) Installation d'un serveur en local avec Docker

Pour développer une application web, les développeurs codent et testent leur application sur un serveur local. Ce n'est qu'une fois leur code validé qu'ils le déploient sur le serveur de production.

Nous allons maintenant installer un serveur local. Plutôt que de l'installer directement sur notre ordinateur, nous allons le faire tourner dans un conteneur Docker, qui est propre, isolé et facile à gérer.

- Créer un dossier nommé **mon-site** et y placer le fichier **index.html**.
- Créer un fichier **Dockerfile** et écrire les instructions ci-dessous :

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
EXPOSE 80
```

→ *Ligne 1* : utilise l'image officielle nginx (serveur web) disponible sur le **registry** officiel Docker Hub https://hub.docker.com

→ *Ligne 2* : copie le site (. qui est dans le répertoire courant) dans le répertoire par défaut du serveur nginx qui est dans le conteneur
→ *Ligne 3* : expose le port 80 et indique que le conteneur écoutera le port 80 (port standard pour HTTP)

- doc
- Construire l'image : docker build -t image-mon-site .
  docker build : construit l'image de nginx à partir du Dockerfile (télécharge Nginx)
  -t : donne un nom à l'image (ici image-mon-site)
  . : indique l'endroit où se trouve le Dockerfile (. : dans le répertoire courant)
- Lancer le conteneur : docker run -p 8080:80 image-mon-site
  docker run : crée et lance le conteneur à partir d'une image docker
  -p 8080:80 : fait correspondre le port 8080 de l'hôte au port 80 du conteneur
  mon-site : nom de l'image docker utilisé pour construire le conteneur
  On peut ajouter des options à notre commande docker :
  --name : pour donner un nom au conteneur.
  -d option pour faire tourner le conteneur en arrière-plan (le terminal est disponible)
  docker run -d --name mon-site -p 8080:80 image-mon-site
- Ouvrir un navigateur et saisir l'adresse : http://localhost:8080/

4) Commandes docker utiles

docker images : Liste les images buildées

```
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
image-mon-site latest    d1a310a59db8   2 hours ago      52.9MB
```

docker ps : liste tous les conteneurs en cours d'exécution

```
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                   NAMES
d81c0ea63610   image-mon-site "/docker-entrypoint.…"   2 minutes ago   Up About a minute   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   mon-site
```

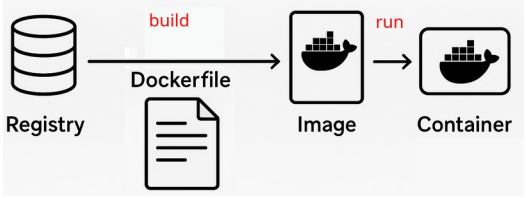
docker ps -a : liste tous les conteneurs (même ceux arrêtés)

docker stop <id> ou <name> : arrête un conteneur en cours
docker start <id> ou <name> : démarre un conteneur stoppé
docker rm <id> ou <name> : supprime un conteneur arrêté (il faut l'arrêter avant de le supprimer)
docker rm -f <id> ou <name> : arrête et supprime un conteneur
docker rmi <id> ou <name> : supprime une image

docker exec -ti mon-site sh : permet de naviguer dans un conteneur en cours d'exécution et d'exécuter des actions à l'intérieur.
docker exec : pour exécuter des commandes à l'intérieur du conteneur.
-ti : pour utiliser un terminal interactif comme sur un ordinateur normal.
sh : shell pour naviguer dans le système de fichiers et exécuter des commandes.
exit : ferme le shell pour revenir dans le terminal local.

5) Résumer

- Un **registry** est un service qui stocke et distribue des images.
- **Dockerfile** est un fichier texte qui décrit les instructions nécessaires pour construire une image Docker.



- Une **image** est un modèle contenant tout le nécessaire pour créer et exécuter un conteneur.
- Un **conteneur** est une instance d'une image.

6) Exercice pour comprendre docker

- Éditer le fichier index.html : « Ma page web avec docker » et enregistrer.
- Réactualiser le navigateur. Quel problème observez-vous ?

- Utiliser la commande docker exec -ti mon-site sh pour expliquer votre observation.
- Réfléchir au fonctionnement de docker vu précédemment et utiliser les commandes docker pour résoudre ce problème. Expliquer votre démarche.

---

7) Volume docker

Le problème vu précédemment est très contraignant, car il nous fait perdre beaucoup de temps pour tester chaque modification de notre application.
Pour le résoudre nous allons utiliser un **volume docker**.

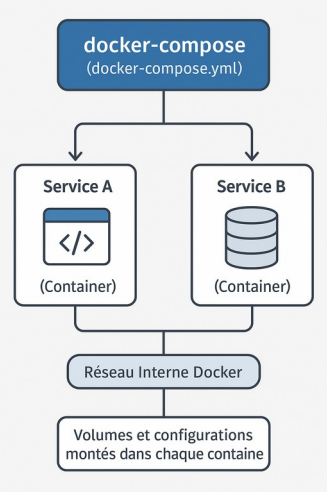
Un **volume Docker** est un mécanisme de stockage persistant qui conserve les données même après l'arrêt ou la suppression d'un conteneur.

Pour que nos modifications soient visibles immédiatement dans le navigateur sans « rebuilder » l'image, nous allons monter le dossier HTML local dans le conteneur avec un volume Docker (**Bind mount**).
docker run -d --name mon-site -p 8080:80 -v .:/usr/share/nginx/html image-mon-site
-v .:/usr/share/nginx/html : bind mount du dossier courant (.) vers le dossier Nginx

Si votre terminal n'est pas ouvert dans votre projet :
-v ~/…/mon-site:/usr/share/nginx/html image-mon-site
~ représente le répertoire personnel de l'utilisateur courant

Vérifier que les modifications réalisées localement sont immédiatement visibles dans le navigateur sans avoir à « rebuilder » l'image.

8) Docker compose



Docker Compose est un outil permettant de définir et gérer plusieurs conteneurs Docker à la fois, qui peuvent communiquer entre eux.

Ceci se fait grâce au fichier **docker-compose.yml** qui permet de démarrer l'environnement en une seule commande au lieu de lancer chaque conteneur avec une longue commande docker run.

Docker-compose devient indispensable dès qu'un projet utilise plusieurs services.

Même si dans notre exemple, nous n'utilisons qu'un seul conteneur, nous allons créer un fichier docker-compose.yml pour :

- Définir le build de l'image
- Configurer le conteneur
- Exposer le port
- Monter le volume

Créer un fichier docker-compose.yml dans votre projet en respectant l'indentation ci-dessous.

```
services:
  mon-site:
    build: .
    image: image-mon-site
    container_name: conteneur-mon-site
    ports:
      - "8080:80"
    volumes:
      - .:/usr/share/nginx/html
```

→ *Ligne 1* : Indique la version du format Docker Compose (la version 3.9 est récente et compatible avec la plupart des fonctionnalités modernes).
→ *Ligne 2* : Définit la liste des **services** (conteneurs) que Docker Compose va lancer.
→ *Ligne 3* : Définit le nom du service.
→ *Ligne 4* : Indique de construire l'image depuis le Dockerfile présent dans le dossier courant (**.**).
→ *Ligne 5* : Définit le nom de l'image buildée.
→ *Ligne 6* : Définit le nom du conteneur.
→ *Ligne 7* : Définit la liste des ports exposés.
→ *Ligne 8* : Redirige le port 80 du conteneur (où Nginx écoute) vers le port 8080 de ta machine hôte.
→ *Ligne 9* : Définit la liste des volumes.
→ Ligne 10 : Monte le dossier local actuel (.) dans le conteneur à l'emplacement où Nginx sert les fichiers (/usr/share/nginx/html) pour voir les modifications locales immédiatement.

Supprimer tous vos conteneurs et toutes vos images.
Lancer la commande docker compose up -d et tester le bon fonctionnement de votre application.

9) Commandes docker-compose utiles

docker compose up : crée et démarre tous les services définis dans le docker-compose.yml
docker compose up -d : même chose en mode détaché (le terminal n'est pas bloqué).
docker compose up -d --build: même chose en forçant un rebuild (si le Dockerfile a été modifié).
docker compose stop : Arrête les conteneurs sans les supprimer.
docker compose down : Arrête et supprime tous les conteneurs.
docker compose start : Démarre les conteneurs arrêtés.
docker compose restart : Redémarre les conteneurs (stop + start).
docker compose ps : Liste les conteneurs gérés par Compose.
docker compose exec <nom du service> sh : Ouvre un shell interactif dans le conteneur.